NILO NEY COUTINHO MENEZES

PYTHON

FROM SCRATCH

PROGRAMMING FOR ABSOLUTE BEGINNERS WITH PYTHON

SOLVED EXERCISES

1st edition - updated on Aug 2, 2025

LEARN STEP-BY-STEP HOW TO PROGRAM

FREE DISTRIBUTION ONLY
NOT FOR SALE

novatec

LogiKraft

PYTHON
FROM SCRATCH

SOLVED EXERCISES

1st edition - updated on Aug 2, 2025

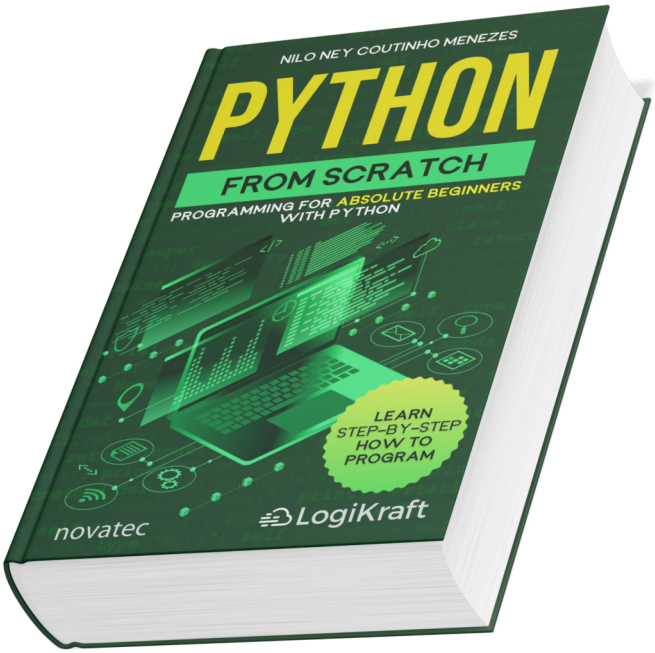
Nilo Ney Coutinho Menezes

questions@pythonfromscratch.com
Telegram: https://t.me/pythonfromscratchbook
Website: https://pythonfromscratch.com

Contents

About the book

This book is aimed at beginners in programming. The basic concepts of programming, such as expressions, variables, loops, decisions, lists, dictionaries, sets, functions, files, classes, objects, databases with SQLite 3, regular expressions, and graphical interfaces with tkinter are presented one by one with examples and exercises. The work aims to explore computer programming as a day-to-day tool. It can be read during an introductory computer programming course and used as a study guide for self-learners. To fully benefit from the content, only basic computer knowledge, such as typing texts, opening and saving files, is sufficient. All software used in the book can be downloaded for free and runs on Windows, Linux, and macOS.



https://pythonfromscratch.com/wheretobuy.html

Introduction

This document was created to provide all the solved exercises from the book in a single file. The book's website can be accessed at https://pythonfromscratch.com or via the QR code below:

Book website: https://www.pythonfromscratch.com

Chapter 02

Exercise 02-01

Convert the following mathematical expressions so that they can be calculated using the Python interpreter.

10 + 20 × 30

42 ÷ 30

(94 + 2) × 6 - 1

```
# To execute the calculation and view the answer,
# copy and paste the lines below into the interpreter window,
# one at a time.
# The exercise answers are the lines below:
10 + 20 * 30
4**2 / 30
(9**4 + 2) * 6 - 1
```

Exercise 02-02

Type the following expression in the interpreter:

10 % 3 * 10 ** 2 + 1 - 10 * 4 / 2

Try to solve the same calculation using only pencil and paper. Notice how important the priority of operations is.

```
# The result of the expression:
# 10 % 3 * 10 ** 2 + 1 - 10 * 4 / 2
# is 81.0
#
# Performing the calculation with the priorities from page 52,
# executing only one operation per line,
# we have the following calculation order:
# 0 --> 10 % 3 * 10 ** 2 + 1 - 10 * 4 / 2
# 1 --> 10 % 3 * 100     + 1 - 10 * 4 / 2
# 2 --> 1      * 100     + 1 - 10 * 4 / 2
# 3 -->          100     + 1 - 10 * 4 / 2
# 4 -->          100     + 1 - 40     / 2
# 5 -->          100     + 1 - 20
# 6 -->          101         - 20
# 7 -->                        81
#
# If you're curious to know why the result
# is 81.0 and not 81, read section 3.2, page 67.
# The division operation always results in a floating point number.
```

Exercise 02-03

Make a program that displays your name on the screen.

```
print("Write your name between the quotes")
```

Exercise 02-04

Write a program that displays the result of 2a × 3b, where a is 3 and b is 5.

```
a = 3
b = 5
print(2 * a * 3 * b)
```

Exercise 02-05

Write a program that calculates the sum of three variables and prints the result on the screen

```
a = 2
b = 3
c = 4
print(a + b + c)
```

Exercise 02-06

Modify Program 2.2 so that it calculates a 15% increase for a salary of $750.

```
salary = 750
raise_percentage = 15
print(salary + (salary * raise_percentage / 100))
```

Exercise 02-07

Using the properties of division and multiplication, try to understand how these results are the same:

0.2 * 6 + 8 * 0.3 + 7 * 0.5 = (20 * 6 + 8 * 30 + 7 * 50) / 100

```
# 0.2 * 6 + 8 * 0.3 + 7 * 0.5 = (20 * 6 + 8 * 30 + 7 * 50) / 100
# We can distribute the division on the left side:
# 20 / 100 * 6 + 8 * 30 / 100 + 7 * 50 / 100
# regrouping the common element (100):
# ((20 * 6) + (8 * 30) + 7 * 50) / 100
# All these expressions result in 7.1
```

Chapter 03

Exercise 03-01

Complete the following table, marking integer or floating-point depending on the number presented.

```
# integer
# floating point
# floating point
# integer
# integer
# floating point
```

Exercise 03-02

Complete the following table, answering True or False. Consider a = 4, b = 10, c = 5.0, d = 1, and f = 5.

```
# False (a==c)
# True  (a<b)
# False (d>b)
# False (c!=f)
# False (a==b)
# False (c<d)
# True (b>a)
# True (c>=f)
# True (f>=c)
# True (c<=c)
# True (c<=f)
```

Exercise 03-03

Complete the following table using a = True, b = False, and c = True.

```
# True (a and a)
# False (b and b)
# False (not c)
# True (not b)
# False (not a)
# False (a and b)
# False (b and c)
# True (a or c)
# True (b or c)
# True (a or c)
# True (b or c)
# True (c or a)
# True (c or b)
# True (c or c)
# False (b or b)
```

Exercise 03-04

Write an expression to determine whether a person should pay tax. Consider that people whose salary is greater than $1,200.00 pay taxes.

```
salary > 1200
```

Exercise 03-05

Calculate the result of the expression A > B and C or D, using the values in the following table.

```
A  B C     D      Result
1  2 True  False
10 3 False False
5  1 True  True


# False
# False
# True
```

Exercise 03-06

Write an expression that will be used to decide whether a student is approved. To be approved, all student averages (arithmetic mean) must be greater than or equal to 7 (consider 10 the maximum grade). Consider that the student only takes three subjects and that the grade for each one is stored in the following variables: grade1, grade2, and grade3.

```
# According to the statement:
grade1 > 7 and grade2 > 7 and grade3 > 7
# In practice, the student is approved if they get a grade greater than or
equal to the average, therefore:
grade1 >= 7 and grade2 >= 7 and grade3 >= 7
```

Exercise 03-07

Make a program that asks for two integer numbers. Print the sum of these two numbers on the screen.

```
first_number = int(input("Enter the first number:"))
second_number = int(input("Enter the second number:"))
print(first_number + second_number)
```

Exercise 03-08

Write a program that reads a value in meters and displays it converted to millimeters.

```
meters = float(input("Enter the value in meters: "))
millimeters = meters * 1000
print("%10.3f meters equals %10.3f millimeters." % (meters, millimeters))
```

Exercise 03-09

Write a program that reads the user's number of days, hours, minutes, and seconds. Calculate the total in seconds.

```
days = int(input("Days:"))
hours = int(input("Hours:"))
minutes = int(input("Minutes:"))
seconds = int(input("Seconds:"))
# One minute has 60 seconds
# One hour has 3600 (60 * 60) seconds
# One day has 24 hours, so 24 * 3600 seconds
total_in_seconds = days * 24 * 3600 + hours * 3600 + minutes * 60 +
seconds
print("Converted to seconds equals %10d seconds." % total_in_seconds)
```

Exercise 03-10

Make a program that calculates a pay raise. It must request the amount of the salary and the percentage of the raise. Display the amount of the raise and the new salary.

```
salary = float(input("Enter current salary:"))
raise_percentage = float(input("Enter raise percentage:"))
raise_amount = salary * raise_percentage / 100
new_salary = salary + raise_amount
print("A raise of %5.2f %% on a salary of $ %7.2f" % (raise_percentage,
salary))
print("equals a raise of $ %7.2f" % raise_amount)
print("Resulting in a new salary of $ %7.2f" % new_salary)
```

Exercise 03-11

Make a program that asks for the price of a commodity and the discount percentage. Display the discount amount and the price to pay.

```
price = float(input("Enter the price of the item:"))
discount = float(input("Enter the discount percentage:"))
discount_amount = price * discount / 100
to_pay = price - discount_amount
print("A discount of %5.2f %% on an item of $ %7.2f" % (discount, price))
print("equals $ %7.2f." % discount_amount)
print("The amount to pay is $ %7.2f" % to_pay)
```

Exercise 03-12

Write a program that calculates the time of a car trip. Ask the distance to cover and the average speed expected for the trip.

```
distance = float(input("Enter the distance in km:"))
average_speed = float(input("Enter the average speed in km/h:"))
time = distance / average_speed
print("The estimated time is %5.2f hours" % time)
# Optional: print the time in hours, minutes and seconds
time_s = int(time * 3600)  # convert from hours to seconds
hours = int(time_s / 3600)  # integer part
time_s = int(time_s % 3600)  # remainder
minutes = int(time_s / 60)
seconds = int(time_s % 60)
print("%05d:%02d:%02d" % (hours, minutes, seconds))
```

Exercise 03-13

Write a program that converts a temperature entered in °C to °F. The formula for this conversion is:

```
C = float(input("Enter the temperature in °C:"))
F = (9 * C / 5) + 32
print("%5.2f°C equals %5.2f°F" % (C, F))
```

Exercise 03-14

Write a program that asks the number of kilometers traveled by a rental car and the number of days the vehicle was rented. Calculate the price to pay, knowing that the car costs $60 a day and $0.15 per km driven.

```
km = int(input("Enter the number of kilometers driven:"))
days = int(input("Enter how many days you kept the car:"))
price_per_day = 60
price_per_km = 0.15
total_to_pay = km * price_per_km + days * price_per_day
print("Total to pay: $ %7.2f" % total_to_pay)
```

Exercise 03-15

Write a program to calculate a smoker's lifespan reduction. Ask how many cigarettes a day they smoke and how many years they have smoked. Consider that a smoker loses 10 minutes of life with each cigarette and calculate how many days of life the smoker will lose. Display the total in days.

```
cigarettes_per_day = int(input("Number of cigarettes per day:"))
years_smoking = float(input("Number of years smoking:"))
reduction_in_minutes = years_smoking * 365 * cigarettes_per_day * 10
# One day has 24 x 60 minutes
reduction_in_days = reduction_in_minutes / (24 * 60)
print("Life time reduction %8.2f days." % reduction_in_days)
```

Chapter 04

Exercise 04-01

Analyze Program 4.1. What happens if the first and second values are the same? Explain.

```
# If the values are equal, nothing will be printed.
# This happens because a > b and b > a are False when a == b.
# Thus, neither print 2 nor print 3 will be executed, so nothing will be
printed.
```

Exercise 04-02

Write a program that asks the speed of a user's car. If it exceeds 80 km/h, display a message stating that the user has been fined. In this case, show the amount of the fine, charging $5 per km above 80 km/h.

```
speed = float(input("Enter your car speed:"))
if speed > 80:
    fine = (speed - 80) * 5
    print(f"You were fined ${fine:7.2f}!")
if speed <= 80:
    print("Your speed is ok, have a good trip!")
```

Exercise 04-03

Write a program that reads three numbers and prints the largest and the smallest.

```
a = int(input("Enter the first value:"))
b = int(input("Enter the second value:"))
c = int(input("Enter the third value:"))
largest = a
if b > a and b > c:
    largest = b
if c > a and c >= b:
    largest = c
smallest = a
if b < c and b < a:
    smallest = b
if c <= b and c < a:
    smallest = c
print(f"The smallest number entered was {smallest}")
print(f"The largest number entered was {largest}")
```

Exercise 04-04

Write a program that asks for the employee's salary and calculates the amount of the rise. For salaries above $1,250, calculate a raise of 10%. For those equal or lower, 15%.

```
salary = float(input("Enter your salary: "))
raise_percentage = 0.15
if salary > 1250:
    raise_percentage = 0.10
raise_amount = salary * raise_percentage
print(f"Your raise will be: ${raise_amount:7.2f}")
```

Exercise 04-05

Run Program 4.5 and try some values. Check that the results are the same as in Program 4.2.

```
# Yes, the results are the same.
```

Exercise 04-06

Write a program that asks the distance a passenger wishes to cover in kilometers. Calculate the ticket price, charging $0.50 per km for trips up to 200 km and $0.45 for longer trips.

```
distance = float(input("Enter the distance to travel: "))
if distance <= 200:
    fare = 0.5 * distance
else:
    fare = 0.45 * distance
print(f"Ticket price: ${fare:7.2f}")
```

Exercise 04-07

Analyze Program 4.3. Does using else in that program make sense? Explain your answer.

```
# Analyzing program 4.3, where we have two if statements,
# it's clear that using else doesn't make sense in this program.
# Since the program calculates tax in brackets and modifies
# the base in the first if, the second if needs to be evaluated
# in any case. Therefore, using else
# doesn't make sense in this program.
```

Exercise 04-08

Rewrite Program 4.4 and calculate the Bye operator account using else.

```
plan = input("What is your cell phone plan? ")
if plan != "littletalk" and plan != "talkmore":
    print("I don't know this plan")
else:
    if plan == "littletalk":
        minutes_in_plan = 100
        extra = 0.20
        price = 50
    else:
        minutes_in_plan = 500
        extra = 0.15
        price = 99

    minutes_used = int(input("How many minutes did you use? "))
    print("You will pay:")
    print(f"Plan price      ${price:10.2f}")
    supplement = 0
    if minutes_used > minutes_in_plan:
        supplement = extra * (minutes_used - minutes_in_plan)
    print(f"Supplement      ${supplement:10.2f}")
    print(f"Total           ${price + supplement:10.2f}")
```

Exercise 04-09

Trace Program 4.8. Compare your result to that shown in Table 4.2.

```
# The exercise consists of tracing the program from listing 4.8.
# The result should be the same as shown in table 4.2.
# The tracing technique is presented on page 83,
# section 3.6 Tracing.
```

Exercise 04-10

Write a program that reads two numbers and asks what operation you want to perform. You must be able to calculate sum (+), subtraction (-), multiplication (*), and division (/). Display the result of the requested operation.

```
a = float(input("First number:"))
b = float(input("Second number:"))
operation = input("Enter the operation to perform (+, -, * or /):")
if operation == "+":
    result = a + b
elif operation == "-":
    result = a - b
elif operation == "*":
    result = a * b
elif operation == "/":
    result = a / b
else:
    print("Invalid operation!")
    result = 0
print("Result: ", result)
```

Exercise 04-11

Write a program to approve a bank loan for the purchase of a home. The program must ask the price of the house to buy, the salary, and the number of years to pay. The amount of the monthly installment cannot exceed 30% of the salary. Calculate the installment as the amount of the house to be purchased divided by the number of months to pay.

```
value = float(input("Enter the house value: "))
salary = float(input("Enter the salary: "))
years = int(input("How many years to pay: "))
months = years * 12
installment = value / months
if installment > salary * 0.3:
    print("Unfortunately you cannot get the loan")
else:
    print(f"Installment value: $ {installment:7.2f} Loan OK")
```

Exercise 04-12

Write a program that calculates the price to pay for electricity. Ask the amount of kWh consumed and the type of installation: R for residential, I for industrial, and C for commercial. Calculate the price to pay according to the following table.

```
consumption = int(input("Consumption in kWh: "))
type = input("Installation type (R, C or I): ")
if type == "R":
    if consumption <= 500:
        price = 0.40
    else:
        price = 0.65
elif type == "I":
    if consumption <= 5000:
        price = 0.55
    else:
        price = 0.60
elif type == "C":
    if consumption <= 1000:
        price = 0.55
    else:
        price = 0.60
else:
    price = 0
    print("Error! Unknown installation type!")
cost = consumption * price
print(f"Amount to pay: $ {cost:7.2f}")
```

Exercise 04-13

In the following program, invert the if and else lines, negating the condition. Add the necessary lines to make it work in Python.

```
 if a > b:
    print("a is greater than b")
else:
    print("b is greater than a")


a = int(input("a: "))
b = int(input("b: "))
if a <= b:
    print("b is greater than a")
else:
    print("a is greater than b")
```

Exercise 04-14

Rewrite the following program with if-elif-else. Add the necessary lines to make it work in Python.

```
if a < 10:
    print("a is less than 10")
if a >= 10 and a < 20:
    print("a is greater than 10 and less than 20")
if a >= 20:
    print("a is greater than 20")
```

```
a = int(input("a: "))
if a < 10:
    print("a is less than 10")
elif a < 20:
    print("a is greater than or equal to 10 and less than 20")
else:
    print("a is greater than or equal to 20")
```

Exercise 04-15

Rewrite the following program with if-elif-else.

```
time = int(input("Enter the current time:"))
if time < 12:
    print("Good morning!")
if time >= 12 and time <= 18:
   print("Good afternoon!")
if time >= 18:
    print("Good evening!")
```

```
hour = int(input("Enter the current hour:"))
if hour < 12:
    print("Good morning!")
elif hour < 18:
    print("Good afternoon!")
else:
    print("Good evening!")
```

Exercise 04-16

Correct the following program:

```
score = input("Enter your exam scores:")
if score < 4:
    print("Unfortunately you didn't pass")
if score < 7:
    print("You need to resit the exam")
if score > 7:
    print("You passed the year")
```

```
average = float(input("Enter your average: "))
if average < 4:
    print("Unfortunately you failed")
elif average < 7:
    print("You need to take a make-up exam")
else:
    print("You passed the year")
```

Chapter 05

Exercise 05-01

Modify the program to display numbers from 1 to 100.

```
x = 1
while x <= 100:
    print(x)
    x = x + 1
```

Exercise 05-02

Modify the program to display numbers from 50 to 100.

```
x = 50
while x <= 100:
    print(x)
    x = x + 1
```

Exercise 05-03

Make a program to write the countdown of a rocket launch. The program must print 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, and Fire! on the screen.

```
x = 10
while x >= 0:
    print(x)
    x = x - 1
print("Fire!")
```

Exercise 05-04

Modify the previous program to print from 1 to the number entered by the user, this time producing only odd numbers.

```
end = int(input("Enter the last number to print:"))
x = 1
while x <= end:
    print(x)
    x = x + 2
```

Exercise 05-05

Rewrite the previous program to write the first 10 multiples of 3.

```
end = 30
x = 3
while x <= end:
    print(x)
    x = x + 3
```

Exercise 05-06

Change the previous program to display the results in the same format as a multiplication table: 2 x 1 = 2, 2 x 2 = 4,...

```
number = int(input("Multiplication table of:"))
x = 1
while x <= 10:
    print(f"{number} x {x} = {number * x}")
    x = x + 1
```

Exercise 05-07

Modify the previous program so that the user must input the beginning and end of the multiplication table, instead of starting and ending with 1 and 10.

```
n = int(input("Multiplication table of: "))
start = int(input("From: "))
end = int(input("To: "))
x = start
while x <= end:
    print(f"{n} x {x} = {n * x}")
    x = x + 1
```

Exercise 05-08

Write a program that reads two numbers. Print the result of multiplying the first by the second. Use only the addition and subtraction operators to calculate the result. Remember that we can understand the multiplication of two numbers as successive sums of one of them (e.g., $4 \times 5 = 5 + 5 + 5 + 5 = 4 + 4 + 4 + 4 + 4$).

```
first = int(input("First number: "))
second = int(input("Second number: "))
x = 1
result = 0
while x <= second:
    result = result + first
    x = x + 1
print(f"{first} x {second} = {result}")
```

Exercise 05-09

Write a program that reads two numbers. Print the result of dividing the first by the second. Use only the addition and subtraction operators to calculate the result. Remember that the quotient of dividing two numbers is the number of times we can subtract the divisor from the dividend. For example, 20 ÷ 4 = 5 since we can subtract 4 five times from 20.

```
dividend = int(input("Dividend: "))
divisor = int(input("Divisor: "))
quotient = 0
x = dividend
while x >= divisor:
    x = x - divisor
    quotient = quotient + 1
remainder = x
print(f"{dividend} / {divisor} = {quotient} (quotient) {remainder}
(remainder)")
```

Exercise 05-10

Modify the previous program so it accepts answers with uppercase and lowercase letters for all questions.

```
points = 0
question = 1
while question <= 3:
    answer = input(f"Answer to question {question}: ")
    if question == 1 and (answer == "b" or answer == "B"):
        points = points + 1
    if question == 2 and (answer == "a" or answer == "A"):
        points = points + 1
    if question == 3 and (answer == "d" or answer == "D"):
        points = points + 1
    question += 1
print(f"The student scored {points} point(s)")
```

Exercise 05-11

Write a program that asks for a savings account's initial deposit and interest rate. Display month-by-month values for the first 24 months. Write the total interest earned for the period.

```
deposit = float(input("Initial deposit: "))
rate = float(input("Interest rate (Ex.: 3 for 3%): "))
month = 1
balance = deposit
while month <= 24:
    balance = balance + (balance * (rate / 100))
    print(f"Balance for month {month} is ${balance:5.2f}.")
    month = month + 1
print(f"The gain obtained with interest was ${balance-deposit:8.2f}.")
```

Exercise 05-12

Amend the previous program to ask for the monthly amount deposited. This amount will be deposited at the beginning of each month. Include the deposit amount when calculating interest for the following month.

```
deposit = float(input("Initial deposit: "))
rate = float(input("Interest rate (Ex.: 3 for 3%): "))
monthly_investment = float(input("Monthly deposit: "))
month = 1
balance = deposit
while month <= 24:
    balance = balance + (balance * (rate / 100)) + monthly_investment
    print(f"Balance for month {month} is ${balance:5.2f}.")
    month = month + 1
print(f"The gain obtained with interest was ${balance-deposit:8.2f}.")
```

Exercise 05-13

Write a program that asks for the initial amount of debt and the monthly interest. Also, ask for the monthly amount that will be paid. Print the number of months it will take for the debt to be repaid, the full amount that will be paid, and the total interest that will be paid.

```
debt = float(input("Debt: "))
rate = float(input("Interest (Ex.: 3 for 3%): "))
payment = float(input("Monthly payment: "))
month = 1
if debt * (rate / 100) > payment:
    print(
        "Your debt will never be paid off, as the interest is higher than
the monthly payment."
    )
else:
    balance = debt
    interest_paid = 0
    while balance > payment:
        interest = balance * rate / 100
        balance = balance + interest - payment
        interest_paid = interest_paid + interest
        print(f"Debt balance in month {month} is ${balance:6.2f}.")
        month = month + 1
    print(f"To pay off a debt of ${debt:8.2f}, at {rate:5.2f}% interest,")
    print(
        f"you will need {month - 1} months, paying a total of ${interest_
paid:8.2f} in interest."
    )
    print(
        f"In the last month, you would have a remaining balance of
${balance:8.2f} to pay."
    )
```

Exercise 05-14

Write a program that reads integers input by the user. The program must read the numbers until the user enters 0 (zero). At the end of the execution, display how many numbers were entered, their sum, and their arithmetic mean.

```
nsum = 0
quantity = 0
while True:
    n = int(input("Enter an integer: "))
    if n == 0:
        break
    nsum = nsum + n
    quantity = quantity + 1
print("Number of integers entered:", quantity)
print("Sum: ", nsum)
print(f"Average: {nsum/quantity:10.2f}")
```

Exercise 05-15

Write a program to control a small cash register. You must ask the user to enter the product code and the purchase quantity. Use the following code table to obtain the price of each product:

```
Code Price
1    0.50
2    1.00
3    4.00
5    7.00
9    8.00
```

Your program should display the total of the purchases after the user enters 0. Any other code should generate the "Invalid Code" error message.

```
to_pay = 0
while True:
    code = int(input("Product code (0 to exit): "))
    price = 0
    if code == 0:
        break
    elif code == 1:
        price = 0.50
    elif code == 2:
        price = 1.00
    elif code == 3:
        price = 4.00
    elif code == 5:
        price = 7.00
    elif code == 9:
        price = 8.00
    else:
        print("Invalid code!")
    if price != 0:
        quantity = int(input("Quantity: "))
        to_pay = to_pay + (price * quantity)
print(f"Total to pay ${to_pay:8.2f}")
```

Exercise 05-16

Run Program 5.1 for the following values: 501, 745, 384, 2, 7, and 1.

```
# The program should work normally with the values requested by the
exercise.
```

Exercise 05-17

What happens if we enter 0 (zero) in the amount to be paid?

```
# The program stops right after printing the quantity of $50.00 bills
```

Exercise 05-18

Modify the program so it also works with $100 bills.

```
value = int(input("Enter the amount to pay:"))
bills = 0
current = 100
to_pay = value
while True:
    if current <= to_pay:
        to_pay -= current
        bills += 1
    else:
        print(f"{bills} bill(s) of ${current}")
        if to_pay == 0:
            break
        elif current == 100:
            current = 50
        elif current == 50:
            current = 20
        elif current == 20:
            current = 10
        elif current == 10:
            current = 5
        elif current == 5:
            current = 1
        bills = 0
```

Exercise 05-19

Modify the program to accept decimal values and count coins of $0.01, $0.05, $0.10, and $0.50.

```
# Note: some values will not be calculated correctly
# due to rounding issues and the representation of 0.01
# in floating point. An alternative is to multiply all values
# by 100 and perform all calculations with integers.

value = float(input("Enter the amount to pay:"))
bills = 0
current = 100
to_pay = value
while True:
    if current <= to_pay:
        to_pay -= current
        bills += 1
    else:
        if current >= 1:
            print(f"{bills} bill(s) of ${current}")
        else:
            print(f"{bills} coin(s) of ${current:5.2f}")
        if to_pay < 0.01:
            break
        elif current == 100:
            current = 50
        elif current == 50:
            current = 20
        elif current == 20:
            current = 10
        elif current == 10:
            current = 5
        elif current == 5:
            current = 1
        elif current == 1:
            current = 0.50
        elif current == 0.50:
            current = 0.10
        elif current == 0.10:
            current = 0.05
        elif current == 0.05:
```

```
            current = 0.02
        elif current == 0.02:
            current = 0.01
        bills = 0
```

Exercise 05-20

What happens if we type 0.001 in the previous program? If it doesn't work, change the program to fix the issue.

```
# As we prepared the program for values less than 0.01,
# it stops executing after printing 0 bill(s) of $100.
# See also the note in exercise 05.19 to better understand
# this problem.
```

Exercise 05-21

Rewrite Program 5.1 to continue running until the value entered is 0. Use nested loops.

```
while True:
    value = int(input("Enter the amount to pay:"))
    if value == 0:
        break
    bills = 0
    current = 50
    to_pay = value
    while True:
        if current <= to_pay:
            to_pay -= current
            bills += 1
        else:
            print(f"{bills} bill(s) of ${current}")
            if to_pay == 0:
                break
            if current == 50:
                current = 20
            elif current == 20:
                current = 10
            elif current == 10:
                current = 5
            elif current == 5:
                current = 1
            bills = 0
```

Exercise 05-22

Write a program that displays a list of options (menu): addition, subtraction, division, multiplication, and exit. Print the multiplication table for the chosen operation. Repeat until the exit option is chosen.

```
while True:
    print(
        """

Menu
\----
1 - Addition
2 - Subtraction
3 - Division
4 - Multiplication
5 - Exit

"""
    )
    option = int(input("Choose an option:"))
    if option == 5:
        break
    elif option >= 1 and option < 5:
        n = int(input("Multiplication table of:"))
        x = 1
        while x <= 10:
            if option == 1:
                print(f"{n} + {x} = {n + x}")
            elif option == 2:
                print(f"{n} - {x} = {n - x}")
            elif option == 3:
                print(f"{n} / {x} = {n / x:5.4f}")
            elif option == 4:
                print(f"{n} x {x} = {n * x}")
            x = x + 1
    else:
        print("Invalid option!")
```

Exercise 05-23

Write a program that reads a number and checks whether it is a prime number. To check, calculate the rest of the division of the number by 2 and then by all the odd numbers up to the number read. If the remainder of one of these divisions equals zero, the number is not prime. Note that 0 and 1 are not prime and that 2 is the only even prime number.

```
n = int(input("Enter a number:"))
if n < 0:
    print("Invalid number. Please enter only positive values")
if n == 0 or n == 1:
    print(f"{n} is a special case.")
else:
    if n == 2:
        print("2 is prime")
    elif n % 2 == 0:
        print(f"{n} is not prime, as 2 is the only even prime number.")
    else:
        x = 3
        while x < n:
            if n % x == 0:
                break
            x = x + 2
        if x == n:
            print(f"{n} is prime")
        else:
            print(f"{n} is not prime, as it is divisible by {x}")
```

Exercise 05-24

Modify the previous program to read a number n. Print the first n prime numbers.

```
prime_count = int(input("Enter the number of prime numbers to generate:
"))
if prime_count < 0:
    print("Invalid number. Please enter only positive values")
else:
    if prime_count >= 1:
        print("2")  # 2 is the only number that is both prime and even
    primes_generated = 1  # so it's the first prime number generated
    next_prime = 3  # the next prime starts with 3
    while primes_generated < prime_count:
        # Since all following primes are odd
        divisor = 3
        while divisor < next_prime:
            # If remainder is zero, the number is divisible
            if next_prime % divisor == 0:
                break
            # Increment the divisor
            divisor = divisor + 2
        # When the number is prime, it's only divisible by itself
        if divisor == next_prime:
            print(next_prime)
            primes_generated = primes_generated + 1
        # move to the next odd number,
        # since even numbers are not prime, except 2
        next_prime = next_prime + 2
```

Exercise 05-25

Write a program that calculates the square root of a number, using Newton's method to get an approximate result. Since n is the number to obtain the square root, consider the base b = 2. Calculate p using the formula p = (b + (n/b)) / 2. Now, calculate the square of p. At each step, do b = p and recalculate p using the formula presented. Stop when the absolute difference between n and the square of p is less than 0.0001.

```
# The abs function is used to calculate the absolute value of a number,
# that is, its value without sign.
# Examples: abs(1) returns 1 and abs(-1) returns 1

n = float(input("Enter a number to find its square root: "))
b = 2
while abs(n - (b * b)) > 0.00001:
    p = (b + (n / b)) / 2
    b = p
print(f"The square root of {n} is approximately {p:8.4f}")
```

Exercise 05-26

Write a program that calculates the rest of the integer division between two numbers. Use only the addition and subtraction operations to calculate the result.

```
# Note: this exercise is very similar to exercise 5.08
dividend = int(input("Dividend: "))
divisor = int(input("Divisor: "))
quotient = 0
x = dividend
while x >= divisor:
    x = x - divisor
    quotient = quotient + 1
remainder = x
print(f"The remainder of {dividend} / {divisor} is {remainder}")
```

Exercise 05-27-a

Write a program that checks whether a number is palindromic. A number is palindromic if it remains the same if its digits are reversed. Examples include 454 and 10501.

```
# To solve this problem, we can use strings, as presented in section 3.4
of the book
# Note that we are reading the number without converting it to int or
float,
# this way the value of s will be a string
number = input("Enter the number to verify, without spaces:")
from_left = 0
from_right = len(number) - 1  # position of the last character in the
string
while from_right > from_left and number[from_left] == number[from_right]:
    from_right = from_right - 1
    from_left = from_left + 1
if number[from_left] == number[from_right]:
    print(f"{number} is a palindrome")
else:
    print(f"{number} is not a palindrome")
```

Exercise 05-27-b

Write a program that checks whether a number is palindromic. A number is palindromic if it remains the same if its digits are reversed. Examples include 454 and 10501.

```
# Exercise 5.27
# Alternative solution, using only integers
n = int(input("Enter the number to verify:"))
# Since n is an integer, we'll calculate its
# number of digits by finding the first
# power of 10 greater than n.
# Example: 341 - first power of 10 greater: 1000 = 10 ^ 4
# We'll use 4 and not 3 to allow handling numbers
# with a single digit. The adjustment is made in the formulas below
maximum_exponent_of_10 = 0
while 10**maximum_exponent_of_10 < n:
    maximum_exponent_of_10 += 1
# Positions relative to the right and left of n
from_right = maximum_exponent_of_10
from_left = 0
# Here we copy n to number_from_left and number_from_right
number_from_left = number_from_right = n
# and make digit_from_right = digit_from_left (for special cases)
digit_from_right = digit_from_left = 0
while from_right > from_left:
    digit_from_right = int(
        number_from_right / (10 ** (from_right - 1))
    )  # Rightmost digit
    digit_from_left = number_from_left % 10  # Leftmost digit
    if digit_from_right != digit_from_left:  # If they are different, we
exit
        break
    from_left = from_left + 1  # Move to the next digit on the left
    from_right = from_right - 1  # Move to the next digit on the right
    number_from_right = number_from_right - (
        digit_from_right * (10**from_right)
    )  # Adjust ni to remove the previous digit
    number_from_left = int(number_from_left / 10)  # Adjust nf to remove
the last digit

if digit_from_right == digit_from_left:
```

```
    print(f"{n} is a palindrome")
else:
    print(f"{n} is not a palindrome")
```

Chapter 06

Exercise 06-01

Modify Program 6.2 to read seven grades instead of five.

```
grades = [0, 0, 0, 0, 0, 0, 0]  # Or [0] * 7
sum_of_grades = 0
x = 0
while x < 7:
    grades[x] = float(input(f"Grade {x}:"))
    sum_of_grades += grades[x]
    x += 1
x = 0
while x < 7:
    print(f"Grade {x}: {grades[x]:6.2f}")
    x += 1
print(f"Average: {sum_of_grades/x:5.2f}")
```

Exercise 06-02

Make a program that reads two lists and generates a third one with the elements of the first two lists.

```
first = []
second = []
while True:
    element = int(input("Enter a value for the first list (0 to finish): "))
    if element == 0:
        break
    first.append(element)
while True:
    element = int(input("Enter a value for the second list (0 to finish):
"))
    if element == 0:
        break
    second.append(element)
# Copies elements from the first list to the third list
third = first[:]
# Extends the third list with the elements of the second list
third.extend(second)
x = 0
while x < len(third):
    print(f"{x}: {third[x]}")
    x += 1
```

Exercise 06-03

Make a program that goes through two lists and generates a third one without repeated elements.

```
first = []
second = []
while True:
    element = int(input("Enter a value for the first list (0 to finish):"))
    if element == 0:
        break
    first.append(element)
while True:
    element = int(input("Enter a value for the second list (0 to
finish):"))
    if element == 0:
        break
    second.append(element)
third = []
# Here we will create another list with elements from the first
# and second lists. There are several ways to solve this exercise.
# In this solution, we will search for values to insert into the third
# list. If they don't exist, we'll add them to the third. Otherwise,
# we won't copy them, thus avoiding duplicates.
two_lists = first[:]
two_lists.extend(second)
x = 0
while x < len(two_lists):
    y = 0
    while y < len(third):
        if two_lists[x] == third[y]:
            break
        y += 1
    if y == len(third):
        third.append(two_lists[x])
    x += 1
x = 0
while x < len(third):
    print(f"{x}: {third[x]}")
    x += 1
```

Exercise 06-04

Modify the first example (Program 6.7) to perform the same task without using the variable found. Tip: Look at the while exit condition.

```
L = [15, 7, 27, 39]
p = int(input("Enter the value to search for:"))
x = 0
while x < len(L):
    if L[x] == p:
        break
    x += 1
if x < len(L):
    print(f"{p} found at position {x}")
else:
    print(f"{p} not found")
```

Exercise 06-05

Modify the example to search for two values. Instead of just p, read another value v that will also be searched. In the printout, indicate which of the two values was found first.

```
L = [15, 7, 27, 39]
p = int(input("Enter the value to search for (p): "))
v = int(input("Enter the other value to search for (v): "))
x = 0
foundP = False
foundV = False
first = 0
while x < len(L):
    if L[x] == p:
        foundP = True
        if not foundV:
            first = 1
    if L[x] == v:
        foundV = True
        if not foundP:
            first = 2
    x += 1
if foundP:
    print(f"p: {p} found")
else:
    print(f"p: {p} not found")
if foundV:
    print(f"v: {v} found")
else:
    print(f"v: {v} not found")
if first == 1:
    print("p was found before v")
elif first == 2:
    print("v was found before p")
```

Exercise 06-06

Modify the Exercise 6.5 program to search for p and v throughout the list and inform the user of the position in which p and the position in which v were found.

```
L = [15, 7, 27, 39]
p = int(input("Enter the value to search for (p):"))
v = int(input("Enter the other value to search for (v):"))
x = 0
foundP = -1  # Here -1 indicates we haven't found the value yet
foundV = -1
first = 0
while x < len(L):
    if L[x] == p:
        foundP = x
    if L[x] == v:
        foundV = x
    x += 1
if foundP != -1:
    print(f"p: {p} found at position {foundP}")
else:
    print(f"p: {p} not found")
if foundV != -1:
    print(f"v: {v} found at position {foundV}")
else:
    print(f"v: {v} not found")
# Check if both were found
if foundP != -1 and foundV != -1:
    # since foundP and foundV store the positions where they were found
    if foundP <= foundV:
        print("p was found before v")
    else:
        print("v was found before p")
```

Exercise 06-07

Modify Program 6.6 using for. Explain why every while cannot be turned into a for.

```
L = []
while True:
    n = int(input("Enter a number (0 to exit):"))
    if n == 0:
        break
    L.append(n)
for e in L:
    print(e)
# The first while loop couldn't be converted to a for loop because
# the number of repetitions is unknown at the start.
```

Exercise 06-08

Change Program 6.9 to print the smallest element in the list.

```
L = [4, 2, 1, 7]
minimum = L[0]
for e in L:
    if e < minimum:
        minimum = e
print(minimum)
```

Exercise 06-09

The temperature for Mons, Belgium, was stored in the list T = [-10, -8, 0, 1, 2, 5, -2, -4]. Make a program that prints the lowest, highest, and average temperatures.

```
temperature_mons = [-10, -8, 0, 1, 2, 5, -2, -4]
minimum = temperature_mons[
    0
]  # The choice of the first element is arbitrary, could be any valid
element
maximum = temperature_mons[0]
sum = 0
for e in temperature_mons:
    if e < minimum:
        minimum = e
    if e > maximum:
        maximum = e
    sum = sum + e
print(f"Maximum temperature: {maximum} °C")
print(f"Minimum temperature: {minimum} °C")
print(f"Average temperature: {sum / len(temperature_mons)} °C")
```

Exercise 06-10

Modify Program 6.11 to show how many tickets were sold in each room. Use a list that is the same size as the number of rooms and count the number of tickets sold in each room using its elements as counters. Print the total sales at the end of the program on the screen.

```
available_seats = [10, 2, 1, 3, 0]
sold = [0] * len(available_seats)
while True:
    room = int(input("Room (0 to exit): "))
    if room == 0:
        print("End")
        break
    if room > len(available_seats) or room < 1:
        print("Invalid room")
    elif available_seats[room - 1] == 0:
        print("Sorry, room is full!")
    else:
        seats = int(
            input(
                f"How many seats do you want ({available_seats[room - 1]}
available):"
            )
        )
        if seats > available_seats[room - 1]:
            print("That number of seats is not available.")
        elif seats < 0:
            print("Invalid number")
        else:
            available_seats[room - 1] -= seats
            sold[room - 1] += seats
            print(f"{seats} seats sold")
print("\nRoom utilization")
for room, available in enumerate(available_seats):
    print(f"Room {room + 1} - {available} seat(s) available")
print("\nSales by room")
total_sold = 0
for room, sales in enumerate(sold):
    print(f"Room {room + 1} - {sales} ticket(s) sold")
    total_sold += sales
print(f"Total tickets sold: {total_sold}")
```

Exercise 06-11

Modify Program 6.11 to ask for the number of rooms and the number of available seats in each.

```
n_rooms = int(input("Number of rooms: "))
available_seats = []
for room in range(n_rooms):
    available_seats.append(int(input(f"Available seats in room {room + 1}:
")))

sold = [0] * len(available_seats)
while True:
    room = int(input("Room (0 to exit): "))
    if room == 0:
        print("End")
        break
    if room > len(available_seats) or room < 1:
        print("Invalid room")
    elif available_seats[room - 1] == 0:
        print("Sorry, room is full!")
    else:
        seats = int(
            input(
                f"How many seats do you want ({available_seats[room - 1]}
available):"
            )
        )
        if seats > available_seats[room - 1]:
            print("That number of seats is not available.")
        elif seats < 0:
            print("Invalid number")
        else:
            available_seats[room - 1] -= seats
            sold[room - 1] += seats
            print(f"{seats} seats sold")
print("\nRoom utilization")
for room, available in enumerate(available_seats):
    print(f"Room {room + 1} - {available} seat(s) available")
print("\nSales by room")
total_sold = 0
for room, sales in enumerate(sold):
```

```
    print(f"Room {room + 1} - {sales} ticket(s) sold")
    total_sold += sales
print(f"Total tickets sold: {total_sold}")
```

Exercise 06-12

What happens when the list is already ordered? Trace Program 6.18 but with the list L = [1, 2, 3, 4, 5].

```
# If the list is already sorted, no element is greater than the next
element.
# Therefore, after the first verification of all elements,
# the inner loop is interrupted by condition (9).
```

Exercise 06-13

What happens when two values are the same? Trace Program 6.18 but with the list L = [3, 3, 1, 5, 4].

```
# As we use the bubble sort method, in the first verification, 3, 3 are
considered in the correct order.
# When we check the second 3 with 1, a swap occurs.
# The same will happen with the first 3, but only in the next iteration.
Notice that 1 rises to the first position
# like an air bubble in water.
```

Exercise 06-14

Modify Program 6.18 to sort the list in descending order. L = [1, 2, 3, 4, 5] must be ordered as L = [5, 4, 3, 2, 1].

```
L = [1, 2, 3, 4, 5]
end = 5
while end > 1:
    swapped = False
    x = 0
    while x < (end - 1):
        if L[x] < L[x + 1]:  # Only the verification condition was changed
            swapped = True
            temp = L[x]
            L[x] = L[x + 1]
            L[x + 1] = temp
        x += 1
    if not swapped:
        break
    end -= 1
for e in L:
    print(e)
```

Exercise 06-15

What happens when we don't check that the list is empty before calling the pop method?

```
# If we don't check that the list is empty before calling pop(),
# the program stops with an error message, informing that we tried
# to remove an element from an empty list.
# The verification is necessary to control this error and ensure
# the proper functioning of the program.
```

Exercise 06-16

Change Program 6.19 so that you can work with several commands entered at once. Currently, only one command can be entered at a time. Please change it to consider the operation as a string. For example, AAASSSX would mean three new customer arrivals, three services, and finally, the exit from the program.

```
last = 10
queue = list(range(1, last + 1))
while True:
    print(f"\nThere are {len(queue)} customers in the queue")
    print("Current queue:", queue)
    print("Enter F to add a customer to the end of the queue,")
    print("or A to serve a customer. X to exit.")
    operation = input("Operation (F, A or X):")
    x = 0
    exit = False
    while x < len(operation):
        if operation[x] == "A":
            if len(queue) > 0:
                served = queue.pop(0)
                print(f"Customer {served} served")
            else:
                print("Empty queue! No one to serve.")
        elif operation[x] == "F":
            last += 1  # Increments the new customer's ticket
            queue.append(last)
        elif operation[x] == "X":
            exit = True
            break
        else:
            print(
                f"Invalid operation: {operation[x]} at position {x}! Enter
only F, A or X!"
            )
        x += 1
    if exit:
        break
```

Exercise 06-17

Modify the program to work with two lines. To make your job easier, consider command S for serving line 1 and T for servicing line 2. The same for the arrival of customers: A for line 1 and B for line 2.

```
last = 0
queue1 = []
queue2 = []
while True:
    print(
        f"\nThere are {len(queue1)} customers in queue 1 and {len(queue2)}
in queue 2."
    )
    print("Current queue 1:", queue1)
    print("Current queue 2:", queue2)
    print("Enter A to add a customer to the end of queue 1 (or B for queue
2),")
    print("or S to serve queue 1 (or T for queue 2)")
    print("X to exit.")
    operation = input("Operation (A, B, S, T or X):")
    x = 0
    exit = False
    while x < len(operation):
        # Here we'll use queue as a reference to queue 1
        # or queue 2, depending on the operation.
        if operation[x] == "A" or operation[x] == "S":
            queue = queue1
        else:
            queue = queue2
        if operation[x] == "S" or operation[x] == "T":
            if len(queue) > 0:
                served = queue.pop(0)
                print(f"Customer {served} served")
            else:
                print("Empty queue! No one to serve.")
        elif operation[x] == "A" or operation[x] == "B":
            last += 1  # Increments the new customer's ticket
            queue.append(last)
        elif operation[x] == "X":
            exit = True
            break
```

```
        else:
            print(
                f"Invalid operation: {operation[x]} at position {x}! Enter
only A, B, S, T or X!"
            )
        x += 1
    if exit:
        break
```

Exercise 06-18

Make a program that reads an expression with parentheses. Using stacks, verify that the parentheses have been opened and closed in the correct order. Example:

(()) OK

()()(OK

()) Error

You can add elements to the stack whenever you find an open parenthesis and unstack it whenever you find a closed one. When unstacking, make sure that the top of the stack is an open parenthesis. If the expression is correct, your stack will be empty at the end.

```
expression = input("Enter the sequence of parentheses to validate:")
x = 0
stack = []
while x < len(expression):
    if expression[x] == "(":
        stack.append("(")
    if expression[x] == ")":
        if len(stack) > 0:
            top = stack.pop(-1)
        else:
            stack.append(")")  # Forces error message
            break
    x += 1
if len(stack) == 0:
    print("OK")
else:
    print("Error")
```

Exercise 06-19

Change Program 6.22 to request the product and quantity sold from the user. Check if the product name entered exists in the dictionary and only then carry out the stock operation.

```
inventory = {
    "tomato": [1000, 2.30],
    "lettuce": [500, 0.45],
    "potato": [2001, 1.20],
    "beans": [100, 1.50],
}
total = 0
print("Sales:\n")
while True:
    product = input("Product name (end to exit):")
    if product == "end":
        break
    if product in inventory:
        quantity = int(input("Quantity:"))
        if quantity <= inventory[product][0]:
            price = inventory[product][1]
            cost = price * quantity
            print(f"{product:12s}: {quantity:3d} x {price:6.2f} =
{cost:6.2f}")
            inventory[product][0] -= quantity
            total += cost
        else:
            print("Requested quantity not available")
    else:
        print("Invalid product name")
print(f" Total cost: {total:21.2f}\n")
print("Inventory:\n")
for key, data in inventory.items():
    print("Description: ", key)
    print("Quantity: ", data[0])
    print(f"Price: {data[1]:6.2f}\n")
```

Exercise 06-20-a

Write a program that generates a dictionary, where each key is a character and its value is the number of that character found in a sentence input by the user.

Example: “The mouse” → \{“T”: 1, “h”: 1, “e”: 2, ’ “: 1, ’m”: 1, “o”: 1, “u”: 1, “s”: 1}

```
sentence = input("Enter a sentence to count the letters:")
letter_counter = {}
for letter in sentence:
    if letter in letter_counter:
        letter_counter[letter] = letter_counter[letter] + 1
    else:
        letter_counter[letter] = 1
print(letter_counter)
```

Exercise 06-20-b

Write a program that generates a dictionary, where each key is a character and its value is the number of that character found in a sentence input by the user.

Example: “The mouse” → \{“T”: 1, “h”: 1, “e”: 2, ’ “: 1, ’m”: 1, “o”: 1, “u”: 1, “s”: 1}

```
# Alternative solution, using the dictionary's get method

sentence = input("Enter a sentence to count the letters:")
letter_counter = {}
for letter in sentence:
    # If letter doesn't exist in dictionary, returns 0
    # if it exists, returns the previous value
    letter_counter[letter] = letter_counter.get(letter, 0) + 1
print(letter_counter)
```

Exercise 06-21

Write a program that compares two lists. Using operations with sets, print:

```
L1 = [1, 2, 6, 8]
L2 = [3, 6, 8, 9]

print(f"List 1: {L1}")
print(f"List 2: {L2}")

set_1 = set(L1)
set_2 = set(L2)

# Sets support the & operator to perform intersection, that is,
# A & B results in the set of elements present in both A and B
print("Values common to both lists:", set_1 & set_2)
print("Values that only exist in the first:", set_1 - set_2)
print("Values that only exist in the second:", set_2 - set_1)

# Sets support the ^ operator that performs symmetric difference.
# A ^ B results in elements of A not present in B combined
# with elements of B not present in A
# A ^ B = A - B | B - A
print("Elements not repeated in both lists:", set_1 ^ set_2)

# Repeated:
print("First list, without elements repeated in the second:", set_1 - set_
2)
```

Exercise 06-22

Write a program that compares two lists. Consider the first list as the initial version and the second as the version after changes. Using operations with sets, your program should print the list of modifications between these two versions, listing:

- The elements that haven't changed
- The new elements
- The elements that were removed

```
BEFORE = [1, 2, 5, 6, 9]
AFTER = [1, 2, 8, 10]

before_set = set(BEFORE)
after_set = set(AFTER)

# Sets support the & operator to perform intersection, that is,
# A & B results in the set of elements present in both A and B
print("Before:", BEFORE)
print("After:", AFTER)
print("Elements that did not change: ", before_set & after_set)
print("New elements:", after_set - before_set)
print("Elements that were removed:", before_set - after_set)
```

Chapter 07

Exercise 07-01

Write a program that reads two strings. Check that the second occurs inside the first one and print the starting position.

1st string: AABBEFAATT

2nd string: BE

Result: BE found in position 3 of AABBEFAATT

```
first = input("Enter the first string: ")
second = input("Enter the second string: ")

position = first.find(second)

if position == -1:
    print(f"'{second}' not found in '{first}'")
else:
    print(f"{second} found at position {position} in {first}")
```

Exercise 07-02

Write a program that reads two strings and generates a third with the characters common to the two strings read.

1st string: AAACTBF

2nd string: CBT

Result: CBT

The order of the characters in the result string is not important, but it must contain all the letters common to both.

```
first = input("Enter the first string: ")
second = input("Enter the second string: ")

third = ""

# For each letter in the first string
for letter in first:
    # If the letter is in the second string (common to both)
    # To avoid duplicates, it should not be in the third string
    if letter in second and letter not in third:
        third += letter

if third == "":
    print("No common characters found.")
else:
    print(f"Common characters: {third}")
```

Exercise 07-03

Write a program that reads two strings and generates a third one with the characters that appear in only one string.

1st string: CTA

2nd string: ABC

3rd string: BT

The order of the characters in the third string is not important.

```
first = input("Enter the first string: ")
second = input("Enter the second string: ")

third = ""

# For each letter in the first string
for letter in first:
    # Check if the letter does not appear in the second string
    # and also if it's not already listed in the third
    if letter not in second and letter not in third:
        third += letter

# For each letter in the second string
for letter in second:
    # Besides not being in the first string,
    # check if it's not already in the third (avoid repetitions)
    if letter not in first and letter not in third:
        third += letter

if third == "":
    print("No uncommon characters found.")
else:
    print(f"Uncommon characters: {third}")
```

Exercise 07-04

Write a program that reads a string and prints how many times each character appears in that string.

String: TTAAC

Result:

T: 2x

A: 2x

C: 1x

```
sequence = input("Enter the string: ")

counter = {}

for letter in sequence:
    counter[letter] = counter.get(letter, 0) + 1

for key, value in counter.items():
    print(f"{key}: {value}x")
```

Exercise 07-05

Write a program that reads two strings and generates a third one in which the characters of the second are removed from the first.

1st string: AATTGGAA

2nd string: TG

3rd string: AAAA

```
first = input("Enter the first string: ")
second = input("Enter the second string: ")

third = ""

for letter in first:
    if letter not in second:
        third += letter

if third == "":
    print("All characters were removed.")
else:
    print(f"The characters {second} were removed from {first}, resulting
in: {third}")
```

Exercise 07-06

Write a program that reads three strings. The first string is your source string. The second one has the characters that will be replaced by the ones in the third string. Your program should create a fourth string, the resulting string, which is the first string with the characters of the second replaced by the ones in the third.

1st string: AATTCGAA

2nd string: TG

3rd string: AC

Result: AAAACCAA

```
first = input("Enter the first string: ")
second = input("Enter the second string: ")
third = input("Enter the third string: ")

if len(second) == len(third):
    result = ""
    for letter in first:
        position = second.find(letter)
        if position != -1:
            result += third[position]
        else:
            result += letter

    if result == "":
        print("All characters were removed.")
    else:
        print(
            f"The characters {second} were replaced by "
            f"{third} in {first}, resulting in: {result}"
        )
else:
    print("ERROR: The second and third strings must have the same
length.")
```

Exercise 07-07

Write a program that asks the user to type a phrase and print out how many vowels it contains. Don't consider uppercase and lowercase letters to be different. Example: A phrase like "The house" should print three "eou".

```
vowels = "aeiou"
phrase = input("Enter a phrase: ")
lowercase_phrase = phrase.lower()
for vowel in vowels:
    vowel_count = lowercase_phrase.count(vowel)
    if vowel_count > 0:
        print(f"{vowel} appears {vowel_count} time(s)")
```

Exercise 07-08

Write a program to display all the words in a sentence. Consider that a word ends with a blank space or when the string ends. Example: “The mouse gnawed at the clothes” should print 6.

```
phrase = input("Enter a phrase: ")
words = phrase.split()
for word in words:
    print(word)
print("Number of words:", len(words))
```

Exercise 07-09

Modify the hangman game (Program 7.2) to write the secret word in case the player loses.

```
word = input("Enter the secret word:").lower().strip()
for x in range(100):
    print()
typed = []
hits = []
errors = 0
while True:
    password = ""
    for letter in word:
        password += letter if letter in hits else "."
    print(password)
    if password == word:
        print("You got it right!")
        break
    attempt = input("\nEnter a letter:").lower().strip()
    if attempt in typed:
        print("You already tried this letter!")
        continue
    else:
        typed += attempt
        if attempt in word:
            hits += attempt
        else:
            errors += 1
            print("You missed!")
    print("X==:==\nX  :   ")
    print("X  O   " if errors >= 1 else "X")
    line2 = ""
    if errors == 2:
        # The r before the string indicates that its content should not be
processed
        # This way, we can use the characters \ and / without confusing
them
        # with masks like \n and \t
        line2 = r"  |   "
    elif errors == 3:
        line2 = r" \|   "
```

```
    elif errors >= 4:
        line2 = r" \|/ "
    print(f"X{line2}")
    line3 = ""
    if errors == 5:
        line3 += r" /     "
    elif errors >= 6:
        line3 += r" / \ "
    print(f"X{line3}")
    print("X\n===========")
    if errors == 6:
        print("Hanged!")
        print(f"The secret word was: {word}")
        break
```

Exercise 07-10

Modify Program 7.2 to use a list of words. At the beginning, ask for a number and calculate the index of the word to be used using the formula: index = (number * 776) % len(word_list).

```
words = [
    "house",
    "ball",
    "hose",
    "grape",
    "okra",
    "computer",
    "snake",
    "lentil",
    "rice",
]

index = int(input("Enter a number:"))
word = words[(index * 776) % len(words)]
for x in range(100):
    print()
typed = []
hits = []
errors = 0
while True:
    password = ""
    for letter in word:
        password += letter if letter in hits else "."
    print(password)
    if password == word:
        print("You got it right!")
        break
    attempt = input("\nEnter a letter:").lower().strip()
    if attempt in typed:
        print("You already tried this letter!")
        continue
    else:
        typed += attempt
        if attempt in word:
            hits += attempt
        else:
```

```
            errors += 1
            print("You missed!")
    print("X==:==\nX  :   ")
    print("X  O   " if errors >= 1 else "X")
    line2 = ""
    if errors == 2:
        line2 = r"  |   "
    elif errors == 3:
        line2 = r" \|   "
    elif errors >= 4:
        line2 = r" \|/ "
    print(f"X{line2}")
    line3 = ""
    if errors == 5:
        line3 += r" /     "
    elif errors >= 6:
        line3 += r" / \ "
    print(f"X{line3}")
    print("X\n==========")
    if errors == 6:
        print("Hanged!")
        print(f"The secret word was: {word}")
        break
```

Exercise 07-11

Modify Program 7.2 to use lists of strings to draw the hangman doll. You can use a list for each row and organize them into a list of lists. Instead of controlling when to print each part, draw on those lists, replacing the element to be drawn.

Example:

>>> line = list(“X------”) >>> line [“X”, “-”, “-”, “-”, “-”, “-”, “-”]

>>> line[6] = “|” >>> line [“X”, “-”, “-”, “-”, “-”, “-”, “|”]

>>> ““.join(line) “X-----|”

```
words = [
    "house",
    "ball",
    "hose",
    "grape",
    "okra",
    "computer",
    "snake",
    "lentil",
    "rice",
]

index = int(input("Enter a number:"))
word = words[(index * 776) % len(words)]
for x in range(100):
    print()
typed = []
hits = []
errors = 0

lines_txt = """
X==:==
X  :
X
X
X
X
=======

"""
```

```
lines = []

for line in lines_txt.splitlines():
    # Padding spaces to each line
    # in case your text editor removed the
    # spaces at the end of each line
    lines.append(list(line.ljust(8, " ")))

while True:
    password = ""
    for letter in word:
        password += letter if letter in hits else "."
    print(password)
    if password == word:
        print("You got it right!")
        break
    attempt = input("\nEnter a letter:").lower().strip()
    if attempt in typed:
        print("You already tried this letter!")
        continue
    else:
        typed += attempt
        if attempt in word:
            hits += attempt
        else:
            errors += 1
            print("You missed!")
            if errors == 1:
                lines[3][3] = "O"
            elif errors == 2:
                lines[4][3] = "|"
            elif errors == 3:
                lines[4][2] = "\\"
            elif errors == 4:
                lines[4][4] = "/"
            elif errors == 5:
                lines[5][2] = "/"
            elif errors == 6:
                lines[5][4] = "\\"

    for line in lines:
```

```
        print("".join(line))
    if errors == 6:
        print("Hanged!")
        print(f"The secret word was: {word}")
        break
```

Exercise 07-12

Write a tic-tac-toe game for two players. The game should ask you where you want to play and switch between players. With each move, check if the position is free. Also, check when a player has won the match. A tic-tac-toe game can be seen as a list of three elements, each element being another list with three elements.

Game example:

```
 X | O |
---+---+---
   | X | X
---+---+---
   |   | O
```

Each position can be viewed as a number. Below is an example of the positions mapped to the same position on your numeric keypad.

```
 7 | 8 | 9
---+---+---
 4 | 5 | 6
---+---+---
 1 | 2 | 3


#
# Tic Tac Toe
#

# The board
board = """               Positions
   |   |      7 | 8 | 9
---+---+---  ---+---+---
   |   |      4 | 5 | 6
---+---+---  ---+---+---
   |   |      1 | 2 | 3
"""
# A list of positions (row and column) for each valid game position
# An extra element was added to facilitate index manipulation
# and so that they have the same value as the position
#
#  7 | 8 | 9
# ---+---+---
```

```
#  4 | 5 | 6
# ---+---+---
#  1 | 2 | 3

positions = [
    None,  # Element added to facilitate indices
    (5, 1),  # 1
    (5, 5),  # 2
    (5, 9),  # 3
    (3, 1),  # 4
    (3, 5),  # 5
    (3, 9),  # 6
    (1, 1),  # 7
    (1, 5),  # 8
    (1, 9),  # 9
]

# Positions that lead to winning the game
# Moves making a row, column or diagonals win
# The numbers represent the winning positions
winning = [
    [1, 2, 3],  # Rows
    [4, 5, 6],
    [7, 8, 9],
    [7, 4, 1],  # Columns
    [8, 5, 2],
    [9, 6, 3],
    [7, 5, 3],  # Diagonals
    [1, 5, 9],
]

# Build the board from strings
# generating a list of lists that can be modified
board_grid = []
for line in board.splitlines():
    board_grid.append(list(line))

player = "X"  # Start playing with X
playing = True
moves = 0  # Move counter - used to know if it's a draw
while True:
    for t in board_grid:  # Print the board
```

```
        print("".join(t))
    if not playing:  # End after printing the last board
        break
    if moves == 9:  # If 9 moves were made, all positions have been filled
        print("It's a draw! No one won.")
        break
    move = int(input(f"Enter position to play 1-9 (player {player}):"))
    if move < 1 or move > 9:
        print("Invalid position")
        continue
    # Check if the position is free
    if board_grid[positions[move][0]][positions[move][1]] != " ":
        print("Position occupied.")
        continue
    # Mark the move for the player
    board_grid[positions[move][0]][positions[move][1]] = player
    # Check if won
    for p in winning:
        for x in p:
            if board_grid[positions[x][0]][positions[x][1]] != player:
                break
        else:  # If the for loop ends without break, all positions in p
belong to the same player
            print(f"Player {player} won ({p}): ")
            playing = False
            break
    player = "X" if player == "O" else "O"  # Switch player
    moves += 1  # Move counter

# About coordinate conversion:
# board_grid[positions[x][0]][positions[x][1]]
#
# Since board_grid is a list of lists, we can access each character
# by specifying a row and column. To get the row and column based on
# the played position, we use the positions list which returns a tuple
with 2 elements:
# row and column. Where row is element [0] and column is element [1].
# What these lines do is convert a game position (1-9)
# into board rows and columns. Note that in this example we use the board
as
# move memory, in addition to displaying the current game state.
```

Chapter 08

Exercise 08-01

Write a function that returns the greater of two numbers.

Expected values: maximum(5, 6) == 6

maximum(2, 1) == 2

maximum(7, 7) == 7

```
def maximum(a, b):
    if a > b:
        return a
    else:
        return b


print(f"maximum(5,6) == 6 -> obtained: {maximum(5,6)}")
print(f"maximum(2,1) == 2 -> obtained: {maximum(2,1)}")
print(f"maximum(7,7) == 7 -> obtained: {maximum(7,7)}")
```

Exercise 08-02

Write a function that takes two numbers and returns True if the first number is a multiple of the second.

Expected values: multiple(8, 4) == True

multiple(7, 3) == False

multiple(5, 5) == True

```
def multiple(a, b):
    return a % b == 0


print(f"multiple(8,4) == True  -> obtained: {multiple(8,4)}")
print(f"multiple(7,3) == False -> obtained: {multiple(7,3)}")
print(f"multiple(5,5) == True  -> obtained: {multiple(5,5)}")
```

Exercise 08-03

Write a function that takes the length of the side of a square and returns its area (A = side2).

Expected values:

square_area(4) == 16

square_area(9) == 81

```
def square_area(side):
    return side**2


print(f"square_area(4) == 16 -> obtained: {square_area(4)}")
print(f"square_area(9) == 81 -> obtained: {square_area(9)}")
```

Exercise 08-04

Write a function that takes the base and height of a triangle and returns its area (A = (base x height) / 2).

Expected values: triangle_area(6, 9) == 27 triangle_area(5, 8) == 20

```
def triangle_area(b, h):
    return (b * h) / 2


print(f"triangle_area(6, 9) == 27 -> obtained: {triangle_area(6,9)}")
print(f"triangle_area(5, 8) == 20 -> obtained: {triangle_area(5,8)}")
```

Exercise 08-05

Rewrite the function of Program 8.1 to use the list search methods (seen in Chapter 7).

```
def search(list_to_search, value):
    if value in list_to_search:
        return list_to_search.index(value)
    return None


L = [10, 20, 25, 30]
print(search(L, 25))
print(search(L, 27))
```

Exercise 08-06

Rewrite Program 8.2 to use for instead of while.

```
def sum(values):
    total = 0
    for e in values:
        total += e
    return total


L = [1, 7, 2, 9, 15]
print(sum(L))
print(sum([7, 9, 12, 3, 100, 20, 4]))
```

Exercise 08-07

Define a recursive function that calculates the greatest common divisor (G.C.D.) between two numbers a and b, where a > b.

Where

l c m (a , b) = \{ | a × b | g c d (a , b) [.tml-eqn]##

can be written in Python as: a % b.

```
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)


print(f"GCD of 10 and 5 -->  {gcd(10,5)}")
print(f"GCD of 32 and 24 --> {gcd(32,24)}")
print(f"GCD of 5 and 3 -->   {gcd(5,3)}")
```

Exercise 08-08

Using the gcd function defined in the previous exercise, define a function to calculate the least common multiple (LCM) between two numbers.

Where |a × b| can be written in Python as: abs(a * b).

```
def gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)


def lcm(a, b):
    return abs(a * b) / gcd(a, b)


print(f"LCM of 10 and 5 -->  {lcm(10, 5)}")
print(f"LCM of 32 and 24 --> {lcm(32, 24)}")
print(f"LCM of 5 and 3 -->   {lcm(5, 3)}")
```

Exercise 08-09

Trace Program 8.6 and compare your result with the one presented.

```
# The program calculates the factorial of 4
# From the printed output messages and program tracing,
# we can conclude that the factorial of 4 is calculated with recursive
calls
# in the line: fact = n * factorial(n-1)
#
# Since the factorial call precedes the "Factorial of" line print,
# we can visualize the sequence in stack form, where the calculation is
done from outside
# to inside: Calculation of factorial of 4, 3, 2 and 1
# to then proceed to the next line, which prints the results:
# factorial of 1,2,3,4
```

Exercise 08-10

Rewrite the function for calculating the Fibonacci sequence without recursion.

```
def fibonacci(n):
    prev = 0
    next = 1
    while n > 0:
        prev, next = next, next + prev
        n -= 1
    return prev


for x in range(10):
    print(f"fibonacci({x}) = {fibonacci(x)}")
```

Exercise 08-11

Write a function to validate a string variable. This function takes the string, the minimum and maximum number of characters, as parameters. Return True if the string size is between the maximum and minimum values; otherwise, return False.

```
def validate_string(s, min_length, max_length):
    length = len(s)
    return min_length <= length <= max_length


print(validate_string("", 1, 5))
print(validate_string("ABC", 2, 5))
print(validate_string("ABCEFG", 3, 5))
print(validate_string("ABCEFG", 1, 10))
```

Exercise 08-12

Write a function that takes a string and a list. The function must compare the string passed with the elements of the list, which is also passed as a parameter. Return True if the string is found within the list; otherwise, return False.

```
def search_string(s, list_to_search):
    return s in list_to_search


L = ["AB", "CD", "EF", "FG"]

print(search_string("AB", L))
print(search_string("CD", L))
print(search_string("EF", L))
print(search_string("FG", L))
print(search_string("XYZ", L))
```

Exercise 08-13-a

Write a function that receives a string with the valid options to accept (each option is a letter). Convert valid options to lowercase letters. Use input to read an option, convert the value to lowercase letters, and verify that the option is valid. In the case of an invalid option, the function must ask the user to re-enter another option.

```
def validate_input(message, valid_options):
    options = valid_options.lower()
    while True:
        choice = input(message)
        if choice.lower() in options:
            break
        print("Error: invalid option. Please try again.\n")
    return choice


# Example: print(validate_input("Choose an option:", "abcde"))
#
# Extra question: what happens if the user types more than one option?
# For example, ab.
```

Exercise 08-13-b

Write a function that receives a string with the valid options to accept (each option is a letter). Convert valid options to lowercase letters. Use input to read an option, convert the value to lowercase letters, and verify that the option is valid. In the case of an invalid option, the function must ask the user to re-enter another option.

```
def validate_options(valid_options):
    valid_options = valid_options.lower()
    while True:
        option = input("Enter an option:").lower()
        if option in valid_options:
            return option
        print("Invalid option, please choose again.")
```

Exercise 08-14

Change Program 8.22 so that the user has three chances of getting the number right. The program terminates if the user finds the right number or makes three mistakes.

```
import random

n = random.randint(1, 10)
attempts = 0
while attempts < 3:
    x = int(input("Choose a number between 1 and 10: "))
    if x == n:
        print("You got it right!")
        break
    else:
        print("You got it wrong.")
    attempts += 1
```

Exercise 08-15

Change Program 7.2, the hangman game. Choose the word to guess using random numbers.

```
import random

words = [
    "house",
    "ball",
    "hose",
    "grape",
    "okra",
    "computer",
    "snake",
    "lentil",
    "rice",
]
# Choose a random word
word = words[random.randint(0, len(words) - 1)]

typed = []
hits = []
errors = 0

lines_txt = """
X==:==
X  :
X
X
X
X
=======

"""

lines = []

for line in lines_txt.splitlines():
    lines.append(list(line.ljust(8, " ")))

while True:
```

```
    password = ""
    for letter in word:
        password += letter if letter in hits else "."
    print(password)
    if password == word:
        print("You got it right!")
        break
    attempt = input("\nType a letter:").lower().strip()
    if attempt in typed:
        print("You already tried this letter!")
        continue
    else:
        typed += attempt
        if attempt in word:
            hits += attempt
        else:
            errors += 1
            print("You missed!")
            if errors == 1:
                lines[3][3] = "O"
            elif errors == 2:
                lines[4][3] = "|"
            elif errors == 3:
                lines[4][2] = "\\"
            elif errors == 4:
                lines[4][4] = "/"
            elif errors == 5:
                lines[5][2] = "/"
            elif errors == 6:
                lines[5][4] = "\\"

    for line in lines:
        print("".join(line))
    if errors == 6:
        print("Hanged!")
        print(f"The secret word was: {word}")
        break
```

Exercise 08-16

Modify the alien game. Create a variable that represents the player's life, starting with 100 points. The game ends when you find the alien or you run out of life (<=0). With each mistake, your life is decreased by a random value between 5 and 20 points, representing an attack by the alien. You can remove the part of the game's code that limits the number of attempts and let only the player's or alien's life decide when the match ends. Show how much life the player has left before guessing the next tree number.

```
import random

player_health = 100
tree = random.randint(1, 100)
print("An alien is hiding behind a tree")
print("Each tree has been numbered from 1 to 100.")
print("You have 3 attempts to guess which tree")
print("the alien is hiding behind.")

while player_health > 0:
    print(f"Health points: {player_health}")
    guess = int(input("Choose a tree [1-100]: "))
    if guess == tree:
        print("You got it right! The alien was found!")
        break
    elif guess > tree:
        print("Too high")
    else:
        print("Too low")
    damage = random.randint(5, 20)
    player_health -= damage

if player_health <= 0:
    print("You didn't survive. The alien won.")
    print(f"The alien was behind tree {tree}.")
```

Exercise 08-17

Improve the program from the previous exercise by asking the player for the desired difficulty level. In easy mode, life starts at 100 points, and the alien can do between 5 and 20 points of damage. In normal mode, life begins at 80 points, and the alien can cause damage between 10 and 25 points. In hard mode, on the other hand, life begins at 75, and the alien causes damage between 20 and 30 points. Add messages and special characters to make the game more fun.

```
import random

print("🎮 Welcome to Alien Hunter! 👽")
print("\nChoose difficulty level:")
print("1 - Easy   (❤️     100 HP | 💥 Damage: 5-20)")
print("2 - Medium (❤️      80 HP | 💥 Damage: 10-25)")
print("3 - Hard   (❤️      75 HP | 💥 Damage: 20-30)")

while True:
    level = input("\nEnter level number (1-3): ")
    if level in ["1", "2", "3"]:
        break
    print("❌ Invalid option! Choose 1, 2, or 3.")

if level == "1":
    player_health = 100
    min_damage, max_damage = 5, 20
elif level == "2":
    player_health = 80
    min_damage, max_damage = 10, 25
else:
    player_health = 75
    min_damage, max_damage = 20, 30

tree = random.randint(1, 100)
print("\n🌳 An alien is hiding behind a tree!")
print("🔢 Each tree has been numbered from 1 to 100.")
print("❗ You have to guess which tree the alien is hiding behind.")
print("⚠️     Careful! The alien will attack you with each wrong attempt!
\n")

while player_health > 0:
    print(f"❤️     Health points: {player_health}")
```

```
    guess = int(input("🎯 Choose a tree [1-100]: "))
    if guess == tree:
        print("\n🎉 CONGRATULATIONS! You got it right! The alien was
found! 🎊")
        break
    elif guess > tree:
        print("⬇️     Too high! Try a lower number.")
    else:
        print("⬆️     Too low! Try a higher number.")
    damage = random.randint(min_damage, max_damage)
    player_health -= damage
    print(f"💥 The alien attacked you! Damage: {damage}\n")

if player_health <= 0:
    print("\n💀 Game Over! You didn't survive.")
    print(f"👽 The alien was behind tree {tree}.")
```

Exercise 08-18

Modify Program 8.26 to receive two optional parameters. One is to indicate the character to print before the number, with white space being the default value. The second optional parameter is how many characters to add per level, with 2 as the default value.

```
def print_lists(values, level=0, character=" ", increment=2):
    for x in values:
        if isinstance(x, int):
            print(f"{character * (level * increment)}{x}")
        else:
            print_lists(x, level + 1, character, increment)


# Usage example:
# print_lists([1, 2, 3, [4, 5, 6, [7, 8, 9]], 10], character="*",
increment=4)
```

Exercise 08-19

Write a generator capable of generating the sequence of prime numbers.

```
def primes(n):
    p = 1  # Position in sequence
    yield 2  # 2 is the only even prime number
    d = 3  # divisor starts with 3
    b = 3  # dividend starts with 3, is the number we'll test if it's
prime
    while p < n:
        # print(d, b, d % b, p, n)
        if b % d == 0:  # If b is divisible by d, the remainder will be 0
            if b == d:  # If b equals d, all d values have been tested
                yield b  # b is prime
                p += 1  # increment the sequence
            b += 2  # Move to the next odd number
            d = 3  # Start dividing by 3 again
        elif d < b:  # Continue trying?
            d += 2  # Increment the divisor to the next odd number
        else:
            b += 2  # Try another odd number


for prime in primes(10):
    print(prime)
```

Exercise 08-20

Write a generator capable of generating a sequence with the factorial from 1 to n, where n is passed as a parameter to the generator.

```
def factorial_generator(n):
    value = 1
    for element in range(1, n + 1):
        value *= element
        yield value


# Usage example:
# Generate factorials from 1 to 5
for n, factorial in enumerate(factorial_generator(5), 1):
    print(f"{n}! = {factorial}")
```

Exercise 08-21

Write a function that generates numbers like Python's range function, but the last number is included in the interval. This function takes three parameters, and its behavior changes if we pass one, two, or three parameters. Call it a myrange.

Examples: list(myrange(1)) [0, 1]

list(myrange(1, 10)) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

list(myrange(0, 10, 2)) [0, 2, 4, 6, 8, 10]

You may have noticed that, unlike range, the myrange function considers the end of the interval as closed; the last number is part of the range.

```
def myrange(start, end=None, step=1):
    if end is None:
        start, end = 0, start

    current = start
    while current <= end:  # Note the <= to include the last value
        yield current
        current += step


# Test cases
print(list(myrange(1)))  # [0, 1]
print(list(myrange(1, 10)))  # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(list(myrange(0, 10, 2)))  # [0, 2, 4, 6, 8, 10]
```

Exercise 08-22

Modify the calculator program that uses partial to support two more operations: root for square root and power for exponentiation.

```
import math
import operator
from functools import partial


def execute_unary(operation, symbol, operand1):
    result = operation(float(operand1))
    print(f"{symbol} {operand1} = {result}")


def execute(operation, symbol, operand1, operand2):
    result = operation(float(operand1), float(operand2))
    print(f"{operand1} {symbol} {operand2} = {result}")


operations = {
    "+": partial(execute, operator.add, "+"),
    "-": partial(execute, operator.sub, "-"),
    "*": partial(execute, operator.mul, "×"),
    "/": partial(execute, operator.truediv, "÷"),
    "sqrt": partial(execute_unary, math.sqrt, "square root of "),
    "power": partial(execute, operator.pow, "power"),
}
operand1 = input("Operand 1: ")
operation = input("Operation: ").strip().lower()
if operation in operations:
    if operation == "sqrt":  # Square root has only one operand
        operations[operation](operand1)
    else:
        operand2 = input("Operand 2: ")
        operations[operation](operand1, operand2)
else:
    print("Invalid operation!")
```

Chapter 09

Exercise 09-01

Write a program that takes the name of a file from the command line and prints every line in that file.

```
import sys

# Check if the parameter was passed
if len(sys.argv) != 2:  # Remember that the program name is the first in
the list
    print("\nUsage: e09-01.py filename\n\n")
else:
    name = sys.argv[1]
    file = open(name, "r")
    for line in file.readlines():
        # Since the line ends with ENTER,
        # we remove the last character before printing
        print(line[:-1])
    file.close()

# Don't forget to read about encodings
# Depending on the file type and your operating system,
# it may not print correctly on screen.
```

Exercise 09-02

Modify the program from Exercise 9.1 to receive two more parameters for printing: the start line and the end line. The program must print only the lines between these values (including the start and end lines).

```
import sys

# Check if parameters were passed
if len(sys.argv) != 4:  # Remember that the program name is the first in
the list
    print("\nUsage: e09-02.py filename start end\n\n")
else:
    name = sys.argv[1]
    start = int(sys.argv[2])
    end = int(sys.argv[3])
    file = open(name, "r")
    for line in file.readlines()[start - 1 : end]:
        # Since the line ends with ENTER,
        # we remove the last character before printing
        print(line[:-1])
    file.close()

# Don't forget to read about encodings
# Depending on the file type and your operating system,
# it may not print correctly on screen.
```

Exercise 09-03

Create a program that reads the files evens.txt and odds.txt and creates a single file evensandodds.txt with all the lines from the other two files in numerical order.

```
# Assume that even and odd files contain only integers
# Assume that values in each file are sorted
# Values don't need to be sequential
# Tolerates blank lines
# Even and odd files can have different number of lines


def read_number(file):
    while True:
        number = file.readline()
        # Check if something was read
        if number == "":
            return None
        # Ignore blank lines
        if number.strip() != "":
            return int(number)


def write_number(file, n):
    file.write(f"{n}\n")


even = open("evens.txt", "r")
odd = open("odds.txt", "r")
even_odd = open("even_odd.txt", "w")
even_num = read_number(even)
odd_num = read_number(odd)
while True:
    if even_num is None and odd_num is None:  # End if both are None
        break
    if even_num is not None and (odd_num is None or even_num <= odd_num):
        write_number(even_odd, even_num)
        even_num = read_number(even)
    if odd_num is not None and (even_num is None or odd_num <= even_num):
        write_number(even_odd, odd_num)
        odd_num = read_number(odd)
```

```
even_odd.close()
even.close()
odd.close()
```

Exercise 09-04

Create a program that takes the names of two files as command line parameters and generates an output file with the lines from the first followed by the lines from the second file. The name of the output file can also be passed as a parameter on the command line.

```
import sys

# Check if parameters were passed
if len(sys.argv) != 4:  # Remember that the program name is the first in
the list
    print("\nUsage: e09-04.py first second output\n\n")
else:
    first = open(sys.argv[1], "r")
    second = open(sys.argv[2], "r")
    output = open(sys.argv[3], "w")

    # Works similar to readlines
    for l1 in first:
        output.write(l1)
    for l2 in second:
        output.write(l2)

    first.close()
    second.close()
    output.close()
```

Exercise 09-05

Create a program that reverses the order of the lines in the evens.txt file. The first line must contain the largest number, and the last line must contain the smallest number.

```
even = open("even.txt", "r")
output = open("even_reversed.txt", "w")

L = even.readlines()
L.reverse()
for line in L:
    output.write(line)

even.close()
output.close()

# Note that we read all lines before doing the reversal
# This approach doesn't work with large files
# Alternative using with:
#
##with open("even.txt","r") as even, open("even_reversed.txt","w") as
output:
##    L = even.readlines()
##    L.reverse()
##    for line in L:
##        output.write(line)
```

Exercise 09-06

Modify Program 9.5 to print the = symbol 40 times if = is the first character in the line. Also, add the option to stop printing until you press the Enter key each time a line starts with . (dot).

```
WIDTH = 79
input_file = open("input.txt")
for line in input_file.readlines():
    if line[0] == ";":
        continue
    elif line[0] == ">":
        print(line[1:].rjust(WIDTH))
    elif line[0] == "*":
        print(line[1:].center(WIDTH))
    elif line[0] == "=":
        print("=" * 40)
    elif line[0] == ".":
        input("Press Enter to continue")
        print()
    else:
        print(line)
input_file.close()
```

Exercise 09-07

Create a program that reads a text file and generates a paginated output file. Each line must not contain more than 76 characters. Each page should have a maximum of 60 lines. The last line of each page should include the number of the current page and the name of the original file.

```
# Uma boa fonte de textos para teste é o projeto Gutemberg
# http://www.gutenberg.org/
# Não esqueça de configurar o encoding do arquivo.
#
# Este programa foi testado com Moby Dick
# http://www.gutenberg.org/cache/epub/2701/pg2701.txt
# Gravado com o nome de mobydick.txt
#
WIDTH = 76
LINES = 60
FILENAME = "mobydick.txt"


def check_page(file, line, page):
    if line == LINES:
        footer = f"= {FILENAME} - Page: {page} ="
        file.write(footer.center(WIDTH - 1) + "\n")
        page += 1
        line = 1
    return line, page


def write(file, line, nlines, page):
    file.write(line + "\n")
    return check_page(file, nlines + 1, page)


input_file = open(FILENAME, encoding="utf-8")
output = open("paginated_output.txt", "w", encoding="utf-8")

page = 1
lines = 1

for line in input_file.readlines():
    words = line.rstrip().split(" ")
```

```
    line = ""
    for word in words:
        word = word.strip()
        if len(line) + len(word) + 1 > WIDTH:
            lines, page = write(output, line, lines, page)
            line = ""
        line += word + " "
    if line != "":
        lines, page = write(output, line, lines, page)

# To print the number on the last page
while lines != 1:
    lines, page = write(output, "", lines, page)

input_file.close()
output.close()
```

Exercise 09-08

Modify the program from Exercise 9.7 to receive the number of characters per line and the number of lines per page from the command line.

```
import sys


def check_page(file, line, page):
    if line == LINES:
        footer = f"= {FILENAME} - Page: {page} ="
        file.write(footer.center(WIDTH - 1) + "\n")
        page += 1
        line = 1
    return line, page


def write(file, line, nlines, page):
    file.write(line + "\n")
    return check_page(file, nlines + 1, page)


if len(sys.argv) != 4:
    print("\nUsage: exercise-09-08.py file width lines\n\n")
    sys.exit(1)

FILENAME = sys.argv[1]
WIDTH = int(sys.argv[2])
LINES = int(sys.argv[3])

input_file = open(FILENAME, encoding="utf-8")
output = open("paginated_output.txt", "w", encoding="utf-8")

page = 1
lines = 1

for line in input_file.readlines():
    words = line.rstrip().split(" ")
    line = ""
    for word in words:
        word = word.strip()
        if len(line) + len(word) + 1 > WIDTH:
```

```
            lines, page = write(output, line, lines, page)
            line = ""
        line += word + " "
    if line != "":
        lines, page = write(output, line, lines, page)

# To print the number on the last page
while lines != 1:
    lines, page = write(output, "", lines, page)

input_file.close()
output.close()
```

Exercise 09-09

Create a program that prints a list of files. The file names will be passed in the command line. You should open and print them one by one.

```
import sys

if len(sys.argv) < 2:
    print("\nUsage: exercise-09-09.py file1 [file2 file3 fileN]\n\n\n")
    sys.exit(1)

for name in sys.argv[1:]:
    file = open(name, "r")
    for line in file:
        print(line, end="")
    file.close()
```

Exercise 09-10

Create a program that receives a list of file names and generates one large output file containing all other files.

```
import sys

if len(sys.argv) < 2:
    print("\nUsage: exercise-09-10.py file1 [file2 file3 fileN]\n\n\n")
    sys.exit(1)

output = open("single_output.txt", "w", encoding="utf-8")
for name in sys.argv[1:]:
    file = open(name, "r", encoding="utf-8")
    for line in file:
        output.write(line)
    file.close()
output.close()
```

Exercise 09-11

Create a program that reads a file and creates a dictionary where each key is a word and each value is the number of occurrences in the file.

```
import sys

if len(sys.argv) != 2:
    print("\nUsage: exercise-09-11.py file1\n\n\n")
    sys.exit(1)

name = sys.argv[1]
counter = {}

file = open(name, "r", encoding="utf-8")
for line in file:
    line = line.strip().lower()
    words = line.split()
    for word in words:
        if word in counter:
            counter[word] += 1
        else:
            counter[word] = 1
file.close()

for key in counter:
    print(f"{key} = {counter[key]}")
```

Exercise 09-12

Modify the Exercise 9.11 program to also record the row and column of each occurrence of the word in the file. To do this, use lists with the values of each word, saving the row and column of each occurrence.

```
# Column counting is not very precise

import sys

if len(sys.argv) != 2:
    print("\nUsage: exercise-09-12.py file1\n\n\n")
    sys.exit(1)

name = sys.argv[1]
counter = {}
line_num = 1
column = 1

file = open(name, "r", encoding="utf-8")
for line in file:
    line = line.strip().lower()
    words = line.split(" ")  # With parameter considers repeated spaces
    for word in words:
        if word == "":
            column += 1
            continue
        if word in counter:
            counter[word].append((line_num, column))
        else:
            counter[word] = [(line_num, column)]
        column += len(word) + 1
    line_num += 1
    column = 1
file.close()

for key in counter:
    print(f"{key} = {counter[key]}")
```

Exercise 09-13

Create a program that prints the lines of a file. This program must receive three parameters via the command line: the file's name, the starting line, and the last line to print.

```
# Identical to exercise 9.02
import sys

# Check if parameters were provided
if len(sys.argv) != 4:  # Remember that the program name is the first in
the list
    print("\nUsage: exercise-09-13.py filename start end\n\n")
else:
    name = sys.argv[1]
    start = int(sys.argv[2])
    end = int(sys.argv[3])
    file = open(name, "r")
    for line in file.readlines()[start - 1 : end]:
        # Since the line ends with ENTER,
        # we remove the last character before printing
        print(line[:-1])
    file.close()

# Don't forget to read about encodings
# Depending on the file type and your operating system,
# it may not print correctly on screen.
```

Exercise 09-14

Create a program that reads a text file and eliminates repeated spaces between words and at the end of lines. The output file must also not have more than one repeated blank line.

```
# Pay attention to encoding on Windows

import sys

if len(sys.argv) != 3:
    print("\nUsage: exercise-09-14.py input output\n\n\n")
    sys.exit(1)

input_file = sys.argv[1]
output_file = sys.argv[2]

file = open(input_file, "r", encoding="utf-8")
output = open(output_file, "w", encoding="utf-8")
blank = 0

for line in file:
    # Removes spaces on the right
    # Replace with strip if you also
    # want to remove spaces on the left
    line = line.rstrip()
    line = line.replace("  ", "")  # Removes repeated spaces
    if line == "":
        blank += 1  # Counts blank lines
    else:
        blank = 0  # If the line is not blank, resets the counter
    if blank < 2:  # Doesn't write from the second blank line
        output.write(line + "\n")

file.close()
output.close()
```

Exercise 09-15

Refer to Program 7.2, the hangman game. Use a text editor to generate a file with a word written on each line. Modify the program to load (read) the list of words from the text file. Also, try asking for the player's name and generating a file with the number of correct answers for the five best players.

```
# Modified to read the word list from a file
# Reads a score.txt file with the number of hits per player
# Reads a words.txt file with the list of words
#
# Before running:
#
# Create an empty file named score.txt
# Create a words file named words.txt
# containing one word per line.
#
# The game randomly chooses a word from this file
import random

words = []
score = {}


def load_words():
    file = open("words.txt", "r", encoding="utf-8")
    for word in file.readlines():
        word = word.strip().lower()
        if word != "":
            words.append(word)
    file.close()


def load_score():
    file = open("score.txt", "r", encoding="utf-8")
    for line in file.readlines():
        line = line.strip()
        if line != "":
            user, counter = line.split(";")
            score[user] = int(counter)
    file.close()
```

```
def save_score():
    file = open("score.txt", "w", encoding="utf-8")
    for user in score.keys():
        file.write("{user};{score[user]}\n")
    file.close()


def update_score(name):
    if name in score:
        score[name] += 1
    else:
        score[name] = 1
    save_score()


def display_score():
    sorted_score = []
    for user, score_value in score.items():
        sorted_score.append([user, score_value])
    sorted_score.sort(key=lambda score_value: score_value[1])
    print("\n\nBest players by number of hits:")
    sorted_score.reverse()
    for up in sorted_score:
        print(f"{up[0]:30s} {up[1]:10d}")


load_words()
load_score()

word = words[random.randint(0, len(words) - 1)]

typed = []
hits = []
errors = 0
while True:
    password = ""
    for letter in word:
        password += letter if letter in hits else "."
    print(password)
    if password == word:
        print("You got it right!")
```

```
        name = input("Enter your name: ")
        update_score(name)
        break
    attempt = input("\nEnter a letter:").lower().strip()
    if attempt in typed:
        print("You already tried this letter!")
        continue
    else:
        typed += attempt
        if attempt in word:
            hits += attempt
        else:
            errors += 1
            print("You missed!")
    print("X==:==\nX  :   ")
    print("X  O   " if errors >= 1 else "X")
    line2 = ""
    if errors == 2:
        line2 = "  |   "
    elif errors == 3:
        line2 = r" \|   "
    elif errors >= 4:
        line2 = r" \|/ "
    print(f"X{line2}")
    line3 = ""
    if errors == 5:
        line3 += r" /     "
    elif errors >= 6:
        line3 += r" / \ "
    print(f"X{line3}")
    print("X\n==========")
    if errors == 6:
        print("Hanged!")
        break

display_score()
```

Exercise 09-16

Explain how the name and phone fields are stored in the output file.

```
# Each address book record is saved in a line of the file.
# The fields are separated by the # symbol (Hash)
# for example:
# Two entries, Nilo and João are saved in 2 lines of text.
# The entry name is on the left of # and the phone number on the right
#
# Nilo#1234
# João#5678
```

Exercise 09-17

Change Program 9.6 to display the phonebook size in the main menu. Consider the number of names it contains as its size.

```
address_book = []


def ask_name():
    return input("Name: ")


def ask_phone():
    return input("Phone: ")


def show_data(name, phone):
    print(f"Name: {name} Phone: {phone}")


def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(address_book):
        if e[0].lower() == mname:
            return p
    return None


def new():
    global address_book
    name = ask_name()
    phone = ask_phone()
    address_book.append([name, phone])


def delete():
    global address_book
    name = ask_name()
```

```
    p = search(name)
    if p is not None:
        del address_book[p]
    else:
        print("Name not found.")


def modify():
    p = search(ask_name())
    if p is not None:
        name = address_book[p][0]
        phone = address_book[p][1]
        print("Found:")
        show_data(name, phone)
        name = ask_name()
        phone = ask_phone()
        address_book[p] = [name, phone]
    else:
        print("Name not found.")


def list_all():
    print("\nAddress Book\n\n\------")
    for e in address_book:
        show_data(e[0], e[1])
    print("\------\n")


def read():
    global address_book
    filename = ask_filename()
    file = open(filename, "r", encoding="utf-8")
    address_book = []
    for l in file.readlines():
        name, phone = l.strip().split("#")
        address_book.append([name, phone])
    file.close()


def save():
    filename = ask_filename()
    file = open(filename, "w", encoding="utf-8")
```

```
    for e in address_book:
        file.write(f"{e[0]}#{e[1]}\n")
    file.close()


def validate_integer_range(question, start, end):
    while True:
        try:
            value = int(input(question))
            if start <= value <= end:
                return value
        except ValueError:
            print(f"Invalid value, please enter a number between {start}
and {end}")


def menu():
    print(
        """
   1 - New
   2 - Modify
   3 - Delete
   4 - List
   5 - Save
   6 - Read

   0 - Exit
"""
    )
    print(f"\nNames in address book: {len(address_book)}\n")
    return validate_integer_range("Choose an option: ", 0, 6)


while True:
    option = menu()
    if option == 0:
        break
    elif option == 1:
        new()
    elif option == 2:
        modify()
    elif option == 3:
```

```
        delete()
    elif option == 4:
        list_all()
    elif option == 5:
        save()
    elif option == 6:
        read()
```

Exercise 09-18

What happens if your name or phone number contains the character used as a separator in your content? Explain the problem and propose a solution.

```
# If # appears in the name or phone number of an address book entry,
# an error will occur when reading the file.
# This error occurs because the number of expected fields in the line will
be different
# from 2 (name and phone).
# The program has no way of knowing whether the character is part of one
field or another.
# One solution to this problem is to replace the # within a field before
saving it.
# This way, the field separator in the file won't be confused with the
content.
# During reading, the replacement must be reversed to obtain the same
content.
```

Exercise 09-19

Change the list_all function so that it also displays the position of each element.

```
address_book = []


def ask_name():
    return input("Name: ")


def ask_phone():
    return input("Phone: ")


def show_data(name, phone):
    print(f"Name: {name} Phone: {phone}")


def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(address_book):
        if e[0].lower() == mname:
            return p
    return None


def new():
    global address_book
    name = ask_name()
    phone = ask_phone()
    address_book.append([name, phone])


def delete():
    global address_book
    name = ask_name()
    p = search(name)
```

```
    if p is not None:
        del address_book[p]
    else:
        print("Name not found.")


def modify():
    p = search(ask_name())
    if p is not None:
        name = address_book[p][0]
        phone = address_book[p][1]
        print("Found:")
        show_data(name, phone)
        name = ask_name()
        phone = ask_phone()
        address_book[p] = [name, phone]
    else:
        print("Name not found.")


def list_all():
    print("\nAddress Book\n\n\------")
    # We use the enumerate function to get the position in the address
book
    for position, e in enumerate(address_book):
        # Print the position without line break
        print(f"Position: {position}", end="")
        show_data(e[0], e[1])
    print("\------\n")


def read():
    global address_book
    filename = ask_filename()
    file = open(filename, "r", encoding="utf-8")
    address_book = []
    for l in file.readlines():
        name, phone = l.strip().split("#")
        address_book.append([name, phone])
    file.close()
```

```
def save():
    filename = ask_filename()
    file = open(filename, "w", encoding="utf-8")
    for e in address_book:
        file.write(f"{e[0]}#{e[1]}\n")
    file.close()


def validate_integer_range(prompt, start, end):
    while True:
        try:
            value = int(input(prompt))
            if start <= value <= end:
                return value
        except ValueError:
            print(f"Invalid value, please enter a number between {start}
and {end}")


def menu():
    print(
        """
   1 - New
   2 - Modify
   3 - Delete
   4 - List
   5 - Save
   6 - Read

   0 - Exit
"""
    )
    print(f"\nNames in address book: {len(address_book)}\n")
    return validate_integer_range("Choose an option: ", 0, 6)


while True:
    option = menu()
    if option == 0:
        break
    elif option == 1:
        new()
```

```
    elif option == 2:
        modify()
    elif option == 3:
        delete()
    elif option == 4:
        list_all()
    elif option == 5:
        save()
    elif option == 6:
        read()
```

Exercise 09-20

Add the option to sort the list by name in the main menu.

```
address_book = []


def ask_name():
    return input("Name: ")


def ask_phone():
    return input("Phone: ")


def show_data(name, phone):
    print(f"Name: {name} Phone: {phone}")


def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(address_book):
        if e[0].lower() == mname:
            return p
    return None


def new():
    global address_book
    name = ask_name()
    phone = ask_phone()
    address_book.append([name, phone])


def delete():
    global address_book
    name = ask_name()
    p = search(name)
```

```
    if p is not None:
        del address_book[p]
    else:
        print("Name not found.")


def modify():
    p = search(ask_name())
    if p is not None:
        name = address_book[p][0]
        phone = address_book[p][1]
        print("Found:")
        show_data(name, phone)
        name = ask_name()
        phone = ask_phone()
        address_book[p] = [name, phone]
    else:
        print("Name not found.")


def list_all():
    print("\nAddress Book\n\n\------")
    # We use the enumerate function to get the position in the address
book
    for position, e in enumerate(address_book):
        # We print the position without line break
        print(f"Position: {position} ", end="")
        show_data(e[0], e[1])
    print("\------\n")


def read():
    global address_book
    filename = ask_filename()
    file = open(filename, "r", encoding="utf-8")
    address_book = []
    for l in file.readlines():
        name, phone = l.strip().split("#")
        address_book.append([name, phone])
    file.close()
```

```
def sort():
    # You can sort the list as shown in the book
    # with the bubble sort method
    # Or combine Python's sort method with lambdas to
    # define the list key
    # address_book.sort(key=lambda e: return e[0])
    end = len(address_book)
    while end > 1:
        i = 0
        swapped = False
        while i < (end - 1):
            if address_book[i] > address_book[i + 1]:
                # Option: address_book[i], address_book[i+1] = address_
book[i+1], address_book[i]
                temp = address_book[i + 1]
                address_book[i + 1] = address_book[i]
                address_book[i] = temp
                swapped = True
            i += 1
        if not swapped:
            break


def save():
    filename = ask_filename()
    file = open(filename, "w", encoding="utf-8")
    for e in address_book:
        file.write(f"{e[0]}#{e[1]}\n")
    file.close()


def validate_integer_range(question, start, end):
    while True:
        try:
            value = int(input(question))
            if start <= value <= end:
                return value
        except ValueError:
            print(f"Invalid value, please enter between {start} and
{end}")
```

```
def menu():
    print(
        """
   1 - New
   2 - Modify
   3 - Delete
   4 - List
   5 - Save
   6 - Read
   7 - Sort by name

   0 - Exit
"""
    )
    print(f"\nNames in address book: {len(address_book)}\n")
    return validate_integer_range("Choose an option: ", 0, 7)


while True:
    option = menu()
    if option == 0:
        break
    elif option == 1:
        new()
    elif option == 2:
        modify()
    elif option == 3:
        delete()
    elif option == 4:
        list_all()
    elif option == 5:
        save()
    elif option == 6:
        read()
    elif option == 7:
        sort()
```

Exercise 09-21

Using the update and delete functions, ask the user to confirm the change and deletion of a name before performing the operation itself.

```
address_book = []


def ask_name():
    return input("Name: ")


def ask_phone():
    return input("Phone: ")


def show_data(name, phone):
    print(f"Name: {name} Phone: {phone}")


def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(address_book):
        if e[0].lower() == mname:
            return p
    return None


def new():
    global address_book
    name = ask_name()
    phone = ask_phone()
    address_book.append([name, phone])


def confirm(operation):
    while True:
        option = input(f"Confirm {operation} (Y/N)? ").upper()
```

```
        if option in "YN":
            return option
        else:
            print("Invalid response. Choose Y or N.")


def delete():
    global address_book
    name = ask_name()
    p = search(name)
    if p is not None:
        if confirm("deletion") == "Y":
            del address_book[p]
    else:
        print("Name not found.")


def modify():
    p = search(ask_name())
    if p is not None:
        name = address_book[p][0]
        phone = address_book[p][1]
        print("Found:")
        show_data(name, phone)
        name = ask_name()
        phone = ask_phone()
        if confirm("modification") == "Y":
            address_book[p] = [name, phone]
    else:
        print("Name not found.")


def list_all():
    print("\nAddress Book\n\n\------")
    # We use the enumerate function to get the position in the address
book
    for position, e in enumerate(address_book):
        # Print the position without line break
        print(f"Position: {position} ", end="")
        show_data(e[0], e[1])
    print("\------\n")
```

```
def read():
    global address_book
    filename = ask_filename()
    file = open(filename, "r", encoding="utf-8")
    address_book = []
    for l in file.readlines():
        name, phone = l.strip().split("#")
        address_book.append([name, phone])
    file.close()


def sort():
    # You can sort the list as shown in the book
    # using the bubble sort method
    # Or combine Python's sort method with lambdas to
    # define the list key
    # address_book.sort(key=lambda e: return e[0])
    end = len(address_book)
    while end > 1:
        i = 0
        swapped = False
        while i < (end - 1):
            if address_book[i] > address_book[i + 1]:
                # Option: address_book[i], address_book[i+1] = address_
book[i+1], address_book[i]
                temp = address_book[i + 1]
                address_book[i + 1] = address_book[i]
                address_book[i] = temp
                swapped = True
            i += 1
        if not swapped:
            break


def save():
    filename = ask_filename()
    file = open(filename, "w", encoding="utf-8")
    for e in address_book:
        file.write(f"{e[0]}#{e[1]}\n")
    file.close()
```

```
def validate_integer_range(prompt, start, end):
    while True:
        try:
            value = int(input(prompt))
            if start <= value <= end:
                return value
        except ValueError:
            print(f"Invalid value, please enter a number between {start}
and {end}")


def menu():
    print(
        """
   1 - New
   2 - Modify
   3 - Delete
   4 - List
   5 - Save
   6 - Read
   7 - Sort by name

   0 - Exit
"""
    )
    print(f"\nNames in address book: {len(address_book)}\n")
    return validate_integer_range("Choose an option: ", 0, 7)


while True:
    option = menu()
    if option == 0:
        break
    elif option == 1:
        new()
    elif option == 2:
        modify()
    elif option == 3:
        delete()
    elif option == 4:
        list_all()
```

```
    elif option == 5:
        save()
    elif option == 6:
        read()
    elif option == 7:
        sort()
```

Exercise 09-22

When reading or writing a new phonebook, verify that the current phonebook has already been saved. You can use a variable to control when the phonebook was changed (new, updated, deleted) and reset that value when loaded or saved.

```
address_book = []

# Variable to mark a change in the address book
changed = False


def ask_name():
    return input("Name: ")


def ask_phone():
    return input("Phone: ")


def show_data(name, phone):
    print(f"Name: {name} Phone: {phone}")


def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(address_book):
        if e[0].lower() == mname:
            return p
    return None


def new():
    global address_book, changed
    name = ask_name()
    phone = ask_phone()
    address_book.append([name, phone])
    changed = True
```

```
def confirm(operation):
    while True:
        option = input(f"Confirm {operation} (Y/N)? ").upper()
        if option in "YN":
            return option
        else:
            print("Invalid response. Choose Y or N.")


def delete():
    global address_book, changed
    name = ask_name()
    p = search(name)
    if p is not None:
        if confirm("deletion") == "Y":
            del address_book[p]
            changed = True
    else:
        print("Name not found.")


def modify():
    global changed
    p = search(ask_name())
    if p is not None:
        name = address_book[p][0]
        phone = address_book[p][1]
        print("Found:")
        show_data(name, phone)
        name = ask_name()
        phone = ask_phone()
        if confirm("modification") == "Y":
            address_book[p] = [name, phone]
            changed = True
    else:
        print("Name not found.")


def list_all():
    print("\nAddress Book\n\n\------")
```

```
    # We use the enumerate function to get the position in the address
book
    for position, e in enumerate(address_book):
        # Print the position without line break
        print(f"Position: {position} ", end="")
        show_data(e[0], e[1])
    print("\------\n")


def read():
    global address_book, changed
    if changed:
        print(
            "You haven't saved the list since the last change. Do you want
to save it now?"
        )
        if confirm("saving") == "Y":
            save()
    print("Read\n---")
    filename = ask_filename()
    file = open(filename, "r", encoding="utf-8")
    address_book = []
    for l in file.readlines():
        name, phone = l.strip().split("#")
        address_book.append([name, phone])
    file.close()
    changed = False


def sort():
    global changed
    # You can sort the list as shown in the book
    # using the bubble sort method
    # Or combine Python's sort method with lambdas to
    # define the list key
    # address_book.sort(key=lambda e: return e[0])
    end = len(address_book)
    while end > 1:
        i = 0
        swapped = False
        while i < (end - 1):
            if address_book[i] > address_book[i + 1]:
```

```
                # Option: address_book[i], address_book[i+1] = address_
book[i+1], address_book[i]
                temp = address_book[i + 1]
                address_book[i + 1] = address_book[i]
                address_book[i] = temp
                swapped = True
            i += 1
        if not swapped:
            break
    changed = True


def save():
    global changed
    if not changed:
        print("You haven't changed the list. Do you want to save it
anyway?")
        if confirm("saving") == "N":
            return
    print("Save\n\------")
    filename = ask_filename()
    file = open(filename, "w", encoding="utf-8")
    for e in address_book:
        file.write(f"{e[0]}#{e[1]}\n")
    file.close()
    changed = False


def validate_integer_range(prompt, start, end):
    while True:
        try:
            value = int(input(prompt))
            if start <= value <= end:
                return value
        except ValueError:
            print(f"Invalid value, please enter a number between {start}
and {end}")


def menu():
    print(
        """
```

```
    1 - New
    2 - Modify
    3 - Delete
    4 - List
    5 - Save
    6 - Read
    7 - Sort by name

    0 - Exit
"""
    )
    print(f"\nNames in address book: {len(address_book)} Changed:
{changed}\n")
    return validate_integer_range("Choose an option: ", 0, 7)


while True:
    option = menu()
    if option == 0:
        break
    elif option == 1:
        new()
    elif option == 2:
        modify()
    elif option == 3:
        delete()
    elif option == 4:
        list_all()
    elif option == 5:
        save()
    elif option == 6:
        read()
    elif option == 7:
        sort()
```

Exercise 09-23

Change the program to load the last phonebook when initializing. Tip: Use another file to store the filename.

```
address_book = []

# Variable to mark a change in the address book
changed = False


def ask_name():
    return input("Name: ")


def ask_phone():
    return input("Phone: ")


def show_data(name, phone):
    print(f"Name: {name} Phone: {phone}")


def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(address_book):
        if e[0].lower() == mname:
            return p
    return None


def new():
    global address_book, changed
    name = ask_name()
    phone = ask_phone()
    address_book.append([name, phone])
    changed = True
```

```
def confirm(operation):
    while True:
        option = input(f"Confirm {operation} (Y/N)? ").upper()
        if option in "YN":
            return option
        else:
            print("Invalid response. Choose Y or N.")


def delete():
    global address_book, changed
    name = ask_name()
    p = search(name)
    if p is not None:
        if confirm("deletion") == "Y":
            del address_book[p]
            changed = True
    else:
        print("Name not found.")


def modify():
    global changed
    p = search(ask_name())
    if p is not None:
        name = address_book[p][0]
        phone = address_book[p][1]
        print("Found:")
        show_data(name, phone)
        name = ask_name()
        phone = ask_phone()
        if confirm("modification") == "Y":
            address_book[p] = [name, phone]
            changed = True
    else:
        print("Name not found.")


def list_all():
    print("\nAddress Book\n\n\------")
    # We use the enumerate function to get the position in the address
```

```
book
    for position, e in enumerate(address_book):
        # Print the position without line break
        print(f"Position: {position} ", end="")
        show_data(e[0], e[1])
    print("\------\n")


def read_last_saved_book():
    last = last_book()
    if last is not None:
        read_file(last)


def last_book():
    try:
        file = open("last book.dat", "r", encoding="utf-8")
        last = file.readline()[:-1]
        file.close()
    except FileNotFoundError:
        return None
    return last


def update_last(name):
    file = open("last book.dat", "w", encoding="utf-8")
    file.write(f"{name}\n")
    file.close()


def read_file(filename):
    global address_book, changed
    file = open(filename, "r", encoding="utf-8")
    address_book = []
    for l in file.readlines():
        name, phone = l.strip().split("#")
        address_book.append([name, phone])
    file.close()
    changed = False


def read():
```

```
    global changed
    if changed:
        print(
            "You haven't saved the list since the last change. Do you want
to save it now?"
        )
        if confirm("save") == "Y":
            save()
    print("Read\n---")
    filename = ask_filename()
    read_file(filename)
    update_last(filename)


def sort():
    global changed
    # You can sort the list as shown in the book
    # using the bubble sort method
    # Or combine Python's sort method with lambdas to
    # define the list key
    # address_book.sort(key=lambda e: return e[0])
    end = len(address_book)
    while end > 1:
        i = 0
        swapped = False
        while i < (end - 1):
            if address_book[i] > address_book[i + 1]:
                # Option: address_book[i], address_book[i+1] = address_
book[i+1], address_book[i]
                temp = address_book[i + 1]
                address_book[i + 1] = address_book[i]
                address_book[i] = temp
                swapped = True
            i += 1
        if not swapped:
            break
    changed = True


def save():
    global changed
    if not changed:
```

```
        print("You haven't changed the list. Do you want to save it
anyway?")
        if confirm("save") == "N":
            return
    print("Save\n\------")
    filename = ask_filename()
    file = open(filename, "w", encoding="utf-8")
    for e in address_book:
        file.write(f"{e[0]}#{e[1]}\n")
    file.close()
    update_last(filename)
    changed = False


def validate_integer_range(prompt, start, end):
    while True:
        try:
            value = int(input(prompt))
            if start <= value <= end:
                return value
        except ValueError:
            print(f"Invalid value, please enter a number between {start}
and {end}")


def menu():
    print(
        """
   1 - New
   2 - Modify
   3 - Delete
   4 - List
   5 - Save
   6 - Read
   7 - Sort by name

   0 - Exit
"""
    )
    print(f"\nNames in address book: {len(address_book)} Changed:
{changed}\n")
    return validate_integer_range("Choose an option: ", 0, 7)
```

```
read_last_saved_book()

while True:
    option = menu()
    if option == 0:
        break
    elif option == 1:
        new()
    elif option == 2:
        modify()
    elif option == 3:
        delete()
    elif option == 4:
        list_all()
    elif option == 5:
        save()
    elif option == 6:
        read()
    elif option == 7:
        sort()
```

Exercise 09-24

What happens to the phonebook if a loading or saving error occurs? Explain.

```
# In case of a read error, the program stops executing.
# If the error occurs during saving, the unsaved data
# will be lost.
# These problems can be solved by modifying the
# read and save functions, adding try/except blocks.
# Ideally, the program should display the error message and continue
running.
# In the case of saving, the data should not be lost and the user should
be able
# to try again.
```

Exercise 09-25

Change the ask_name and ask_telephone functions to receive an optional parameter. If this parameter is passed, use it as the value returned if the data entry is empty.

```
address_book = []

# Variable to mark a change in the address book
changed = False


def ask_name(default=""):
    name = input("Name: ")
    if name == "":
        name = default
    return name


def ask_phone(default=""):
    phone = input("Phone: ")
    if phone == "":
        phone = default
    return phone


def show_data(name, phone):
    print(f"Name: {name} Phone: {phone}")


def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(address_book):
        if e[0].lower() == mname:
            return p
    return None


def new():
```

```
    global address_book, changed
    name = ask_name()
    phone = ask_phone()
    address_book.append([name, phone])
    changed = True


def confirm(operation):
    while True:
        option = input(f"Confirm {operation} (Y/N)? ").upper()
        if option in "YN":
            return option
        else:
            print("Invalid response. Choose Y or N.")


def delete():
    global address_book, changed
    name = ask_name()
    p = search(name)
    if p is not None:
        if confirm("deletion") == "Y":
            del address_book[p]
            changed = True
    else:
        print("Name not found.")


def modify():
    global changed
    p = search(ask_name())
    if p is not None:
        name = address_book[p][0]
        phone = address_book[p][1]
        print("Found:")
        show_data(name, phone)
        name = ask_name(name)  # If nothing is entered, keep the value
        phone = ask_phone(phone)
        if confirm("modification") == "Y":
            address_book[p] = [name, phone]
            changed = True
    else:
```

```
        print("Name not found.")


def list_all():
    print("\nAddress Book\n\n\------")
    # We use the enumerate function to get the position in the address
book
    for position, e in enumerate(address_book):
        # Print the position without line break
        print(f"Position: {position} ", end="")
        show_data(e[0], e[1])
    print("\------\n")


def read_last_saved_address_book():
    last = last_address_book()
    if last is not None:
        read_file(last)


def last_address_book():
    try:
        file = open("last address book.dat", "r", encoding="utf-8")
        last = file.readline()[:-1]
        file.close()
    except FileNotFoundError:
        return None
    return last


def update_last(name):
    file = open("last address book.dat", "w", encoding="utf-8")
    file.write(f"{name}\n")
    file.close()


def read_file(filename):
    global address_book, changed
    file = open(filename, "r", encoding="utf-8")
    address_book = []
    for l in file.readlines():
        name, phone = l.strip().split("#")
```

```
            address_book.append([name, phone])
        file.close()
        changed = False


def read():
    global changed
    if changed:
        print(
            "You haven't saved the list since the last change. Do you want
to save it now?"
        )
        if confirm("saving") == "Y":
            save()
    print("Read\n---")
    filename = ask_filename()
    read_file(filename)
    update_last(filename)


def sort():
    global changed
    # You can sort the list as shown in the book
    # using the bubble sort method
    # Or combine Python's sort method with lambdas to
    # define the list key
    # address_book.sort(key=lambda e: return e[0])
    end = len(address_book)
    while end > 1:
        i = 0
        swapped = False
        while i < (end - 1):
            if address_book[i] > address_book[i + 1]:
                # Option: address_book[i], address_book[i+1] = address_
book[i+1], address_book[i]
                temp = address_book[i + 1]
                address_book[i + 1] = address_book[i]
                address_book[i] = temp
                swapped = True
            i += 1
        if not swapped:
            break
```

```
    changed = True


def save():
    global changed
    if not changed:
        print("You haven't changed the list. Do you want to save it
anyway?")
        if confirm("saving") == "N":
            return
    print("Save\n\------")
    filename = ask_filename()
    file = open(filename, "w", encoding="utf-8")
    for e in address_book:
        file.write(f"{e[0]}#{e[1]}\n")
    file.close()
    update_last(filename)
    changed = False


def validate_integer_range(prompt, start, end):
    while True:
        try:
            value = int(input(prompt))
            if start <= value <= end:
                return value
        except ValueError:
            print(f"Invalid value, please enter a number between {start}
and {end}")


def menu():
    print(
        """
   1 - New
   2 - Modify
   3 - Delete
   4 - List
   5 - Save
   6 - Read
   7 - Sort by name
```

```
   0 - Exit
"""
    )
    print(f"\nNames in address book: {len(address_book)} Changed:
{changed}\n")
    return validate_integer_range("Choose an option: ", 0, 7)


read_last_saved_address_book()

while True:
    option = menu()
    if option == 0:
        break
    elif option == 1:
        new()
    elif option == 2:
        modify()
    elif option == 3:
        delete()
    elif option == 4:
        list_all()
    elif option == 5:
        save()
    elif option == 6:
        read()
    elif option == 7:
        sort()
```

Exercise 09-26

Change the program to verify the repetition of names. Generate an error message to avoid having two phonebook entries with the same name. You should check this before adding or updating names.

```
address_book = []

# Variable to mark a change in the address book
changed = False


def ask_name(default=""):
    name = input("Name: ")
    if name == "":
        name = default
    return name


def ask_phone(default=""):
    phone = input("Phone: ")
    if phone == "":
        phone = default
    return phone


def show_data(name, phone):
    print(f"Name: {name} Phone: {phone}")


def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(address_book):
        if e[0].lower() == mname:
            return p
    return None
```

```
def new():
    global address_book, changed
    name = ask_name()
    if search(name) is not None:
        print("Name already exists!")
        return
    phone = ask_phone()
    address_book.append([name, phone])
    changed = True


def confirm(operation):
    while True:
        option = input(f"Confirm {operation} (Y/N)? ").upper()
        if option in "YN":
            return option
        else:
            print("Invalid response. Choose Y or N.")


def delete():
    global address_book, changed
    name = ask_name()
    p = search(name)
    if p is not None:
        if confirm("deletion") == "Y":
            del address_book[p]
            changed = True
    else:
        print("Name not found.")


def modify():
    global changed
    p = search(ask_name())
    if p is not None:
        name = address_book[p][0]
        phone = address_book[p][1]
        print("Found:")
        show_data(name, phone)
        name = ask_name(name)  # If nothing is entered, keep the value
        phone = ask_phone(phone)
```

```
        if confirm("modification") == "Y":
            address_book[p] = [name, phone]
            changed = True
    else:
        print("Name not found.")


def list_all():
    print("\nAddress Book\n\n\------")
    # We use the enumerate function to get the position in the address
book
    for position, e in enumerate(address_book):
        # Print the position, without line break
        print(f"Position: {position} ", end="")
        show_data(e[0], e[1])
    print("\------\n")


def read_last_saved_address_book():
    last = last_address_book()
    if last is not None:
        read_file(last)


def last_address_book():
    try:
        file = open("last address book.dat", "r", encoding="utf-8")
        last = file.readline()[:-1]
        file.close()
    except FileNotFoundError:
        return None
    return last


def update_last(name):
    file = open("last address book.dat", "w", encoding="utf-8")
    file.write(f"{name}\n")
    file.close()


def read_file(filename):
    global address_book, changed
```

```
    file = open(filename, "r", encoding="utf-8")
    address_book = []
    for l in file.readlines():
        name, phone = l.strip().split("#")
        address_book.append([name, phone])
    file.close()
    changed = False


def read():
    global changed
    if changed:
        print(
            "You haven't saved the list since the last change. Do you want
to save it now?"
        )
        if confirm("saving") == "Y":
            save()
    print("Read\n---")
    filename = ask_filename()
    read_file(filename)
    update_last(filename)


def sort():
    global changed
    # You can sort the list as shown in the book
    # using the bubble sort method
    # Or combine Python's sort method with lambdas to
    # define the list key
    # address_book.sort(key=lambda e: return e[0])
    end = len(address_book)
    while end > 1:
        i = 0
        swapped = False
        while i < (end - 1):
            if address_book[i] > address_book[i + 1]:
                # Option: address_book[i], address_book[i+1] = address_
book[i+1], address_book[i]
                temp = address_book[i + 1]
                address_book[i + 1] = address_book[i]
                address_book[i] = temp
```

```
                swapped = True
            i += 1
        if not swapped:
            break
    changed = True


def save():
    global changed
    if not changed:
        print("You haven't changed the list. Do you want to save it
anyway?")
        if confirm("saving") == "N":
            return
    print("Save\n\------")
    filename = ask_filename()
    file = open(filename, "w", encoding="utf-8")
    for e in address_book:
        file.write(f"{e[0]}#{e[1]}\n")
    file.close()
    update_last(filename)
    changed = False


def validate_integer_range(prompt, start, end):
    while True:
        try:
            value = int(input(prompt))
            if start <= value <= end:
                return value
        except ValueError:
            print(f"Invalid value, please enter a number between {start}
and {end}")


def menu():
    print(
        """
   1 - New
   2 - Modify
   3 - Delete
   4 - List
```

```
   5 - Save
   6 - Read
   7 - Sort by name

   0 - Exit
"""
    )
    print(f"\nNames in address book: {len(address_book)} Changed:
{changed}\n")
    return validate_integer_range("Choose an option: ", 0, 7)


read_last_saved_address_book()


while True:
    option = menu()
    if option == 0:
        break
    elif option == 1:
        new()
    elif option == 2:
        modify()
    elif option == 3:
        delete()
    elif option == 4:
        list_all()
    elif option == 5:
        save()
    elif option == 6:
        read()
    elif option == 7:
        sort()
```

Exercise 09-27

Modify the program to control each person's birthday and email address.

```
address_book = []

# Variable to mark a change in the address book
changed = False


def ask_name(default=""):
    name = input("Name: ")
    if name == "":
        name = default
    return name


def ask_phone(default=""):
    phone = input("Phone: ")
    if phone == "":
        phone = default
    return phone


def ask_email(default=""):
    email = input("Email: ")
    if email == "":
        email = default
    return email


def ask_birthday(default=""):
    birthday = input("Birthday: ")
    if birthday == "":
        birthday = default
    return birthday


def show_data(name, phone, email, birthday):
    print(f"Name: {name}\nPhone: {phone}\n" f"Email: {email}\nBirthday:
{birthday}\n")
```

```
def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(address_book):
        if e[0].lower() == mname:
            return p
    return None


def new():
    global address_book, changed
    name = ask_name()
    if search(name) is not None:
        print("Name already exists!")
        return
    phone = ask_phone()
    email = ask_email()
    birthday = ask_birthday()
    address_book.append([name, phone, email, birthday])
    changed = True


def confirm(operation):
    while True:
        option = input(f"Confirm {operation} (Y/N)? ").upper()
        if option in "YN":
            return option
        else:
            print("Invalid response. Choose Y or N.")


def delete():
    global address_book, changed
    name = ask_name()
    p = search(name)
    if p is not None:
        if confirm("deletion") == "Y":
            del address_book[p]
```

```
            changed = True
    else:
        print("Name not found.")


def edit():
    global changed
    p = search(ask_name())
    if p is not None:
        name = address_book[p][0]
        phone = address_book[p][1]
        email = address_book[p][2]
        birthday = address_book[p][3]
        print("Found:")
        show_data(name, phone, email, birthday)
        name = ask_name(name)  # If nothing is entered, keep the value
        phone = ask_phone(phone)
        email = ask_email(email)
        birthday = ask_birthday(birthday)
        if confirm("edit") == "Y":
            address_book[p] = [name, phone, email, birthday]
            changed = True
    else:
        print("Name not found.")


def list_all():
    print("\nAddress Book\n\n\------")
    # We use the enumerate function to get the position in the address
book
    for position, e in enumerate(address_book):
        # Print the position
        print(f"\nPosition: {position}")
        show_data(e[0], e[1], e[2], e[3])
    print("\------\n")


def read_last_saved_address_book():
    last = last_address_book()
    if last is not None:
        read_file(last)
```

```
def last_address_book():
    try:
        file = open("last address book.dat", "r", encoding="utf-8")
        last = file.readline()[:-1]
        file.close()
    except FileNotFoundError:
        return None
    return last


def update_last(name):
    file = open("last address book.dat", "w", encoding="utf-8")
    file.write(f"{name}\n")
    file.close()


def read_file(filename):
    global address_book, changed
    file = open(filename, "r", encoding="utf-8")
    address_book = []
    for l in file.readlines():
        name, phone, email, birthday = l.strip().split("#")
        address_book.append([name, phone, email, birthday])
    file.close()
    changed = False


def read():
    global changed
    if changed:
        print(
            "You haven't saved the list since the last change. Do you want
to save it now?"
        )
        if confirm("save") == "Y":
            save()
    print("Read\n---")
    filename = ask_filename()
    read_file(filename)
    update_last(filename)
```

```
def sort():
    global changed
    # You can sort the list as shown in the book
    # with the bubble sort method
    # Or combine Python's sort method with lambdas to
    # define the list key
    # address_book.sort(key=lambda e: return e[0])
    end = len(address_book)
    while end > 1:
        i = 0
        swapped = False
        while i < (end - 1):
            if address_book[i] > address_book[i + 1]:
                # Option: address_book[i], address_book[i+1] = address_
book[i+1], address_book[i]
                temp = address_book[i + 1]
                address_book[i + 1] = address_book[i]
                address_book[i] = temp
                swapped = True
            i += 1
        if not swapped:
            break
    changed = True


def save():
    global changed
    if not changed:
        print("You haven't changed the list. Do you want to save it
anyway?")
        if confirm("save") == "N":
            return
    print("Save\n\------")
    filename = ask_filename()
    file = open(filename, "w", encoding="utf-8")
    for e in address_book:
        file.write(f"{e[0]}#{e[1]}#{e[2]}#{e[3]}\n")
    file.close()
    update_last(filename)
    changed = False
```

```
def validate_integer_range(question, start, end):
    while True:
        try:
            value = int(input(question))
            if start <= value <= end:
                return value
        except ValueError:
            print("Invalid value, please enter between {start} and {end}")


def menu():
    print(
        """
   1 - New
   2 - Edit
   3 - Delete
   4 - List
   5 - Save
   6 - Read
   7 - Sort by name

   0 - Exit
"""
    )
    print(f"\nNames in address book: {len(address_book)} Changed:
{changed}\n")
    return validate_integer_range("Choose an option: ", 0, 7)


read_last_saved_address_book()


while True:
    option = menu()
    if option == 0:
        break
    elif option == 1:
        new()
    elif option == 2:
        edit()
    elif option == 3:
```

```
            delete()
        elif option == 4:
            list_all()
        elif option == 5:
            save()
        elif option == 6:
            read()
        elif option == 7:
            sort()
```

Exercise 09-28

Modify the program to register multiple phones for the same person. This also allows you to register the type of telephone number: cell phone, landline, home, or work.

```
# As the file format becomes increasingly complex,
# we'll use Python's pickle module to save and read the address book.
#
# Extra challenge:
# Modify the program to display a phone management submenu.
# This submenu would be displayed when adding and changing phones.
# Operations: add new phone, delete phone, change phone
import pickle

address_book = []

# Variable to mark a change in the address book
changed = False

phone_types = ["mobile", "landline", "home", "work", "fax"]


def ask_name(default=""):
    name = input("Name: ")
    if name == "":
        name = default
    return name


def ask_phone(default=""):
    phone = input("Phone: ")
    if phone == "":
        phone = default
    return phone


def ask_phone_type(default=""):
    while True:
        type = input("Phone type [%s]: " % ",".join(phone_types)).lower()
        if type == "":
            type = default
        for t in phone_types:
```

```
            if t.startswith(type):
                return t  # Returns the full name
        else:
            print("Invalid phone type!")


def ask_email(default=""):
    email = input("Email: ")
    if email == "":
        email = default
    return email


def ask_birthday(default=""):
    birthday = input("Birthday: ")
    if birthday == "":
        birthday = default
    return birthday


def show_data(name, phones, email, birthday):
    print(f"Name: {name.capitalize()}")
    print("Phone(s):")
    for phone in phones:
        print(f"\tNumber: {phone[0]:15s} Type: {phone[1].capitalize()}")
    print(f"Email: {email}\nBirthday: {birthday}\n")


def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(address_book):
        if e[0].lower() == mname:
            return p
    return None


def new():
    global address_book, changed
```

```
    name = ask_name()
    if search(name) is not None:
        print("Name already exists!")
        return
    phones = []
    while True:
        number = ask_phone()
        type = ask_phone_type()
        phones.append([number, type])
        if confirm("that you want to register another phone") == "N":
            break
    email = ask_email()
    birthday = ask_birthday()
    address_book.append([name, phones, email, birthday])
    changed = True


def confirm(operation):
    while True:
        option = input(f"Confirm {operation} (Y/N)? ").upper()
        if option in "YN":
            return option
        else:
            print("Invalid response. Choose Y or N.")


def delete():
    global address_book, changed
    name = ask_name()
    p = search(name)
    if p is not None:
        if confirm("deletion") == "Y":
            del address_book[p]
            changed = True
    else:
        print("Name not found.")


def change():
    global changed
    p = search(ask_name())
    if p is not None:
```

```
        name, phones, email, birthday = address_book[p]
        print("Found:")
        show_data(name, phones, email, birthday)
        name = ask_name(name)  # If nothing is entered, keeps the value
        for phone in phones:
            number, type = phone
            phone[0] = ask_phone(number)
            phone[1] = ask_phone_type(type)
        email = ask_email(email)
        birthday = ask_birthday(birthday)
        if confirm("change") == "Y":
            address_book[p] = [name, phones, email, birthday]
            changed = True
    else:
        print("Name not found.")


def list_all():
    print("\nAddress Book\n\n\------")
    # We use the enumerate function to get the position in the address
book
    for position, e in enumerate(address_book):
        # Print the position
        print(f"\nPosition: {position}")
        show_data(e[0], e[1], e[2], e[3])
    print("\------\n")


def read_last_saved_address_book():
    last = last_address_book()
    if last is not None:
        read_file(last)


def last_address_book():
    try:
        file = open("last address book pickle.dat", "r", encoding="utf-8")
        last = file.readline()[:-1]
        file.close()
    except FileNotFoundError:
        return None
    return last
```

```
def update_last(name):
    file = open("last address book pickle.dat", "w", encoding="utf-8")
    file.write(f"{name}\n")
    file.close()


def read_file(filename):
    global address_book, changed
    file = open(filename, "rb")
    address_book = pickle.load(file)
    file.close()
    changed = False


def read():
    global changed
    if changed:
        print(
            "You haven't saved the list since the last change. Do you want
to save it now?"
        )
        if confirm("saving") == "Y":
            save()
    print("Read\n---")
    filename = ask_filename()
    read_file(filename)
    update_last(filename)


def sort():
    global changed
    # You can sort the list as shown in the book
    # with the bubble sort method
    # Or combine Python's sort method with lambdas to
    # define the list key
    # address_book.sort(key=lambda e: return e[0])
    end = len(address_book)
    while end > 1:
        i = 0
        swapped = False
```

```
        while i < (end - 1):
            if address_book[i] > address_book[i + 1]:
                # Option: address_book[i], address_book[i+1] = address_
book[i+1], address_book[i]
                temp = address_book[i + 1]
                address_book[i + 1] = address_book[i]
                address_book[i] = temp
                swapped = True
            i += 1
        if not swapped:
            break
    changed = True


def save():
    global changed
    if not changed:
        print("You haven't changed the list. Do you want to save it
anyway?")
        if confirm("saving") == "N":
            return
    print("Save\n\------")
    filename = ask_filename()

    file = open(filename, "wb")
    pickle.dump(address_book, file)
    file.close()
    update_last(filename)
    changed = False


def validate_integer_range(question, start, end):
    while True:
        try:
            value = int(input(question))
            if start <= value <= end:
                return value
        except ValueError:
            print(f"Invalid value, please enter between {start} and
{end}")
```

```
def menu():
    print(
        """
   1 - New
   2 - Change
   3 - Delete
   4 - List
   5 - Save
   6 - Read
   7 - Sort by name

   0 - Exit
"""
    )
    print(f"\nNames in address book: {len(address_book)} Changed:
{changed}\n")
    return validate_integer_range("Choose an option: ", 0, 7)


read_last_saved_address_book()

while True:
    option = menu()
    if option == 0:
        break
    elif option == 1:
        new()
    elif option == 2:
        change()
    elif option == 3:
        delete()
    elif option == 4:
        list_all()
    elif option == 5:
        save()
    elif option == 6:
        read()
    elif option == 7:
        sort()
```

Exercise 09-29

Modify Program 9.8 to use the p element for movie titles instead of h2.

```
movies = {
    "drama": ["Citizen Kane", "The Godfather"],
    "comedy": ["Modern Times", "American Pie", "Dr. Dolittle"],
    "crime": ["Black Rain", "Death Wish", "Hard to Kill"],
    "war": ["Rambo", "Platoon", "Tora!Tora!Tora!"],
}

page = open("movies.html", "w", encoding="utf-8")
page.write(
    """
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Movies</title>
</head>
<body>
"""
)
for category, values in movies.items():
    page.write(f"<h1>{category}</h1>")
    for entry in values:
        page.write(f"<p>{entry}</p>")
page.write(
    """
</body>
</html>
"""
)
page.close()
```

Exercise 09-30

Modify Program 9.8 to generate an HTML list using the ul and li elements. Every element in the list must be inside the ul element and inside an li element. Here's an example:

```
<ul\><li>Item1</li><li>Item2</li><li>Item3</li></ul>
```

```
movies = {
    "drama": ["Citizen Kane", "The Godfather"],
    "comedy": ["Modern Times", "American Pie", "Dr. Dolittle"],
    "crime": ["Black Rain", "Death Wish", "Hard to Kill"],
    "war": ["Rambo", "Platoon", "Tora!Tora!Tora!"],
}

page = open("movies.html", "w", encoding="utf-8")
page.write(
    """
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Movies</title>
</head>
<body>
"""
)
for c, v in movies.items():
    page.write(f"<h1>{c.capitalize()}</h1>")
    page.write("<ul>")
    for e in v:
        page.write(f"<li>{e}</li>")
    page.write("</ul>")
page.write(
    """
</body>
</html>
"""
)
page.close()
```

Exercise 09-31

Create a program that corrects Program 9.9 to verify that z exists and is a directory.

```
import os.path

if os.path.isdir("z"):
    print("The directory z exists.")
elif os.path.isfile("z"):
    print("z exists, but it is a file and not a directory.")
else:
    print("The directory z does not exist.")
```

Exercise 09-32

Modify Program 9.9 to receive the file name or directory to be verified through the command line. Print if it exists and if it's a file or a directory.

```
import sys
import os.path

if len(sys.argv) < 2:
    print("Enter the name of the file or directory to check as a
parameter!")
    sys.exit(1)

name = sys.argv[1]
if os.path.isdir(name):
    print(f"The directory {name} exists.")
elif os.path.isfile(name):
    print(f"The file {name} exists.")
else:
    print(f"{name} does not exist.")
```

Exercise 09-33

Create a program that generates an HTML page with links to all the jpg and png files found in a directory entered on the command line.

```
# This exercise can also be done using the glob module
# Check Python documentation for more information
import sys
import os
import os.path

# this module helps with converting filenames to valid HTML links
import urllib.request

if len(sys.argv) < 2:
    print("Enter the directory name to collect jpg and png files!")
    sys.exit(1)

directory = sys.argv[1]

page = open("images.html", "w", encoding="utf-8")
page.write(
    """
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>PNG and JPG Images</title>
</head>
<body>
"""
)
page.write(f"Images found in directory: {directory}")
for entry in os.listdir(directory):
    name, extension = os.path.splitext(entry)
    if extension in [".jpg", ".png"]:
        full_path = os.path.join(directory, entry)
        link = urllib.request.pathname2url(full_path)
        page.write(f"<p><a href='{link}'>{entry}</a></p>")

page.write(
    """
```

```
</body>
</html>
"""
)
page.close()
```

Exercise 09-34

Revisit Program 7.2, the hangman game. Modify it to use time functions to record the duration of the matches.

```
import time

word = input("Enter the secret word:").lower().strip()
for x in range(100):
    print()
typed = []
hits = []
errors = 0

start = time.time()  # Records the start of the game
while True:
    password = ""
    for letter in word:
        password += letter if letter in hits else "."
    print(password)
    if password == word:
        print("You got it right!")
        break
    attempt = input("\nEnter a letter:").lower().strip()
    if attempt in typed:
        print("You already tried this letter!")
        continue
    else:
        typed += attempt
        if attempt in word:
            hits += attempt
        else:
            errors += 1
            print("You missed!")
    print("X==:==\nX  :   ")
    print("X  O   " if errors >= 1 else "X")
    line2 = ""
    if errors == 2:
        line2 = "  |   "
    elif errors == 3:
        line2 = r" \|   "
    elif errors >= 4:
```

```
        line2 = r" \|/ "
    print(f"X{line2}")
    line3 = ""
    if errors == 5:
        line3 += " /     "
    elif errors >= 6:
        line3 += r" / \ "
    print(f"X{line3}")
    print("X\n==========")
    if errors == 6:
        print("Hanged!")
        break
end = time.time()  # time at the end of the game
print(f"Game duration {end - start} seconds")
```

Exercise 09-35

Using the os.walk function, create an HTML page with the name and size of each file from a directory passed in the command line, including all its sub-directories.

```
import sys
import os
import os.path

# this module helps with converting filenames to valid HTML links
import urllib.request

style_mask = "'margin: 5px 0px 5px %dpx;'"


def generate_style(level):
    return style_mask % (level * 20)


def generate_listing(page, directory):
    root_count = os.path.abspath(directory).count(os.sep)
    for root, directories, files in os.walk(directory):
        level = root.count(os.sep) - root_count
        page.write(f"<p style={generate_style(level)}>{root}</p>")
        style = generate_style(level + 1)
        for f in files:
            full_path = os.path.join(root, f)
            size = os.path.getsize(full_path)
            link = urllib.request.pathname2url(full_path)
            page.write(f"<p style={style}><a href='{link}'>{f}</
a>  ({size} bytes)</p>")


if len(sys.argv) < 2:
    print("Enter the directory name to collect files!")
    sys.exit(1)

directory = sys.argv[1]

page = open("files.html", "w", encoding="utf-8")
page.write(
    """
```

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Files</title>
</head>
<body>
"""
)
page.write(f"Files found starting from directory: {directory}")
generate_listing(page, directory)
page.write(
    """
</body>
</html>
"""
)
page.close()
```

Exercise 09-36

Using the os.walk function, create a program that calculates the space occupied per directory, generating an HTML page with the results.

```
import sys
import os
import os.path
import math


# This function converts the size
# into more readable units, avoiding
# returning and printing very large values.
def size_to_human_readable(size):
    if size == 0:
        return "0 byte"
    magnitude = math.log(size, 10)
    if magnitude < 3:
        return f"{size} bytes"
    elif magnitude < 6:
        return f"{size / 1024.0:7.3f} KB"
    elif magnitude < 9:
        return f"{size / pow(1024, 2)} MB"
    elif magnitude < 12:
        return f"{size / pow(1024, 3)} GB"


style_mask = "'margin: 5px 0px 5px %dpx;'"


def generate_style(level):
    return style_mask % (level * 30)


# Returns a function where the nroot parameter is used
# to calculate the indentation level
def generate_level_and_style(root):
    def level(path):
        xlevel = path.count(os.sep) - nroot
        return generate_style(xlevel)
```

```
    nroot = root.count(os.sep)
    return level


# Uses os.walk to traverse directories
# And a stack to store the size of each directory
def generate_listing(page, directory):
    directory = os.path.abspath(directory)
    # indenter is a function that calculates how many levels
    # from the directory level a path should have.
    indenter = generate_level_and_style(directory)
    stack = [[directory, 0]]  # Guard element, to avoid empty stack
    for root, directories, files in os.walk(directory):
        # If the current directory: root
        # Is not a subdirectory of the last traversed
        # Pop until finding a common parent
        while not root.startswith(stack[-1][0]):
            last = stack.pop()
            page.write(
                f"<p style={indenter(last[0])}>Size: (size_to_human_
readable(last[1]))</p>"
            )
            stack[-1][1] += last[1]
        page.write(f"<p style={indenter(root)}>{root}</p>")
        d_size = 0
        for f in files:
            full_path = os.path.join(root, f)
            d_size += os.path.getsize(full_path)
        stack.append([root, d_size])
    # If the stack has more than one element
    # pop them
    while len(stack) > 1:
        last = stack.pop()
        page.write(
            f"<p style={indenter(last[0])}>Size: ({size_to_human_
readable(last[1])})</p>"
        )
        stack[-1][1] += last[1]


if len(sys.argv) < 2:
    print("Enter the directory name to collect files!")
```

```
    sys.exit(1)

directory = sys.argv[1]

page = open("files.html", "w", encoding="utf-8")
page.write(
    """
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Files</title>
</head>
<body>
"""
)
page.write(f"Files found starting from directory: {directory}")
generate_listing(page, directory)
page.write(
    """
</body>
</html>
"""
)
page.close()
```

Exercise 09-37

Write a program that reads a student's name and four grades. The program must write the data into a file using the JSON format.

```
import json


# Function to read student grades
def read_grades():
    grades = []
    for i in range(4):
        grade = float(input(f"Enter the {i+1}th grade: "))
        grades.append(grade)
    return grades


# Reading student data
name = input("Enter student name: ")
grades = read_grades()

# Creating dictionary with student data
student = {"name": name, "grades": grades}

# Saving data to a JSON file
with open("student_grades.json", "w") as file:
    json.dump(student, file, indent=4)

print("Student data successfully saved to 'student_grades.json' file.")
```

Exercise 09-38

Modify the previous program to read the same file, allowing you to add more data. If the same name is entered twice, update the data using the new entry.

Tip: You may have to use a list of dictionaries to hold multiple entries.

```
import json
import os


def read_grades():
    grades = []
    for i in range(4):
        grade = float(input(f"Enter the {i+1}th grade: "))
        grades.append(grade)
    return grades


def load_data():
    # If the file doesn't exist, returns an empty list
    if not os.path.exists("student_grades.json"):
        return []

    try:
        with open("student_grades.json", "r") as file:
            return json.load(file)
    except json.JSONDecodeError:
        return []


# Reading existing data
students = load_data()

# Reading new data
name = input("Enter student name: ")
grades = read_grades()

# Creating dictionary with new data
new_student = {"name": name, "grades": grades}

# Checks if student already exists and updates, or adds new student
```

```
student_exists = False
for i, student in enumerate(students):
    if student["name"] == name:
        students[i] = new_student
        student_exists = True
        break

if not student_exists:
    students.append(new_student)

# Saving all data to JSON file
with open("student_grades.json", "w") as file:
    json.dump(students, file, indent=4)

print("Student data successfully saved to 'student_grades.json' file.")
```

Exercise 09-39

Modify Program 9.6, the phonebook. Have it read and write the phonebook in a file in JSON format.

```
import json

contacts = []


def ask_name():
    return input("Name: ")


def ask_phone():
    return input("Phone: ")


def show_data(name, phone):
    print(f"Name: {name} Phone: {phone}")


def ask_filename():
    return input("File name: ")


def search(name):
    mname = name.lower()
    for p, e in enumerate(contacts):
        if e[0].lower() == mname:
            return p
    return None


def new():
    name = ask_name()
    phone = ask_phone()
    contacts.append([name, phone])


def delete():
    name = ask_name()
```

```
    p = search(name)
    if p is not None:
        del contacts[p]
    else:
        print("Name not found.")


def modify():
    p = search(ask_name())
    if p is not None:
        name = contacts[p][0]
        phone = contacts[p][1]
        print("Found:")
        show_data(name, phone)
        name = ask_name()
        phone = ask_phone()
        contacts[p] = [name, phone]
    else:
        print("Name not found.")


def list_all():
    print("\nContacts\n\n\------")
    for e in contacts:
        show_data(e[0], e[1])
    print("\------\n")


def read():
    global contacts
    filename = ask_filename()
    try:
        with open(filename, "r", encoding="utf-8") as file:
            contacts = json.load(file)
    except FileNotFoundError:
        print("File not found")
    except json.JSONDecodeError:
        print("Error reading JSON file")


def save():
    filename = ask_filename()
```

```
    with open(filename, "w", encoding="utf-8") as file:
        json.dump(contacts, file, ensure_ascii=False, indent=2)


def validate_integer_range(prompt, start, end):
    while True:
        try:
            value = int(input(prompt))
            if start <= value <= end:
                return value
        except ValueError:
            print(f"Invalid value, please enter a number between {start}
and {end}")


def menu():
    print(
        """
  1 - New
  2 - Modify
  3 - Delete
  4 - List
  5 - Save
  6 - Read

  0 - Exit
"""
    )
    return validate_integer_range("Choose an option: ", 0, 6)


while option := menu():
    if option == 0:
        break
    elif option == 1:
        new()
    elif option == 2:
        modify()
    elif option == 3:
        delete()
    elif option == 4:
        list_all()
```

```
    elif option == 5:
        save()
    elif option == 6:
        read()
```

Exercise 09-40

Modify the visualize.py program to print only the first 512 bytes of the file.

```
import sys
import itertools


def print_bytes(image, bytes_per_line=16):
    for b in itertools.batched(image, bytes_per_line):
        hex_view = " ".join([f"{v:02x}" for v in b])
        tview = "".join([chr(v) if chr(v).isprintable() else "." for v in
b])
        print(f"{hex_view} {" " * 3 * (bytes_per_line - len(b))}{tview}")


if __name__ == "__main__":
    with open(sys.argv[1], "rb") as f:
        image = f.read(512)  # Doesn't read the entire file, only the first
512 bytes
    print_bytes(image)
```

Exercise 09-41

Change the visualize.py program to receive the maximum number of bytes to print and how many bytes per line as command line parameters.

```
import sys
import itertools


def print_bytes(image, bytes_per_line=16):
    for b in itertools.batched(image, bytes_per_line):
        hex_view = " ".join([f"{v:02x}" for v in b])
        tview = "".join([chr(v) if chr(v).isprintable() else "." for v in
b])
        print(f"{hex_view} {" " * 3 * (bytes_per_line - len(b))}{tview}")


if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage: python program.py file max_bytes bytes_per_line")
        sys.exit(1)

    file = sys.argv[1]
    max_bytes = int(sys.argv[2])
    bytes_per_line = int(sys.argv[3])

    with open(file, "rb") as f:
        image = f.read(max_bytes)

    print_bytes(image, bytes_per_line)
```

Exercise 09-42

Modify Program 9.20 so that it uses the name of the image to be generated from the command line.

```
import sys

# Check if filename was provided as argument
if len(sys.argv) != 2:
    print("Usage: python exercise-09-42.py filename.bmp")
    sys.exit(1)

filename = sys.argv[1]


def bytes_little_endian(number, nbytes=4, signed=False):
    """Converts an integer to a sequence of bytes using little endian
encoding.
    If signed is passed, reserves a bit to represent the sign."""
    return number.to_bytes(nbytes, "little", signed=signed)


def padding(value, size=4):
    """Calculates the next multiple for size"""
    if remainder := value % size:
        return value + size - remainder
    return value


# Letter to color conversion table
# in RGB format (red, green, blue)
# Each color can range from 0 to 255.
letter_to_color = {
    " ": (0, 0, 0),  # black
    "r": (255, 0, 0),  # red
    "g": (0, 255, 0),  # green
    "b": (0, 0, 255),  # blue
}
# Drawing that we will transform into an image
drawing = [
    "  rrrr r r bbbbb b    b   ggggg   g    g  r",
    "  r   r r r   b   b    b  g     g  gg   g  r",
```

```
    "  r  r r r   b   b    b  g r r g  g g  g  r",
    "  rrr   r    b   bbbbbb  g    g  g  g g  r",
    "  r     r    b   b    b  gr b rg  g  g g   ",
    "  r     r    b   b    b  g rrr g  g   gg  r",
    "  r     r    b   b    b   ggggg   g    g  r",
]
# Point multiplier
# Each point will be copied multiplier times in the image
# If equal to 4, each point generates a 4x4 point block
multiplier = 32

# Check if all lines have the same size
drawing_width = len(drawing[0])

for line, z in enumerate(drawing):
    if len(z) != drawing_width:
        raise ValueError(
            f"Lines must have the same size. Line with different width:
{line} instead of {len(z)}"
        )

# Calculate data based on multiplier
expanded_drawing = []
for line in drawing:
    new_line = []
    for letter in line:
        new_line.append(letter * multiplier)
    for _ in range(multiplier):
        expanded_drawing.append("".join(new_line))


width = len(expanded_drawing[0])  # Number of columns in the image
height = len(expanded_drawing)  # Number of lines in the image

# Check if letters represent colors
binary_data = []
for line in expanded_drawing:
    binary_line = []
    for character in line:
        # Invert byte order for BMP RGB format
        binary_line.append(bytes(letter_to_color[character][::-1]))
    binary_data.append(b"".join(binary_line))
```

```
# Add padding
width_bytes = width * 3
width_with_padding = padding(width_bytes)
if width_bytes != width_with_padding:
    for p, d in enumerate(binary_data):
        binary_data[p] = b"".join(
            [binary_data[p], bytes(width_with_padding - width_bytes)]
        )

# Calculate image size in bytes with padding
size = padding(width * 3) * height

bmp_header = [
    b"BM",  # Identifier
    bytes_little_endian(54 + size),  # Image size in bytes
    bytes(4),  # 4 bytes 0x00
    bytes_little_endian(54),  # Header size
]

dib_header = [
    bytes_little_endian(40),  # DIB header size
    bytes_little_endian(width),
    bytes_little_endian(
        -height, signed=True
    ),  # Negative height to build image from top to bottom
    bytes_little_endian(1, 2),  # Color planes
    bytes_little_endian(24, 2),  # Bits per pixel
    bytes_little_endian(0),  # No compression
    bytes_little_endian(size),
    bytes_little_endian(2835),  # ceil(72 dpi x 39.3701 in/m) horizontal
    bytes_little_endian(2835),  # ceil(72 dpi x 39.3701 in/m) vertical
    bytes_little_endian(0),  # Number of colors in palette
    bytes_little_endian(0),  # Important colors
]

bmp_header_binary = b"".join(bmp_header)
dib_header_binary = b"".join(dib_header)
binary_data = b"".join(binary_data)

# Verify binary header sizes
assert len(bmp_header_binary) == 14
```

```
assert len(dib_header_binary) == 40
assert len(binary_data) == size

# Write the image
with open(filename, "wb") as f:
    f.write(bmp_header_binary)
    f.write(dib_header_binary)
    f.write(binary_data)

print(f"File {filename} generated. {width=} x {height=} {size=} bytes")
```

Exercise 09-43

Modify the program from the previous exercise to receive a second parameter with the file name that contains the drawing. The goal is to read the drawing from that file.

```
import sys

# Check if the number of arguments is valid
if len(sys.argv) != 3:
    print("Usage: python exercise-09-43.py output_file.bmp drawing_file.
txt")
    sys.exit(1)

output_file = sys.argv[1]
drawing_file = sys.argv[2]

# Read the drawing from file
try:
    with open(drawing_file, "r") as f:
        drawing = [line.strip() for line in f.readlines()]
except FileNotFoundError:
    print(f"Error: Drawing file '{drawing_file}' not found.")
    sys.exit(1)
except IOError:
    print(f"Error: Could not read drawing file '{drawing_file}'.")
    sys.exit(1)


def bytes_little_endian(number, nbytes=4, signed=False):
    """Converts an integer to a sequence of bytes using little endian
encoding.
    If signed is passed, reserves a bit to represent the sign."""
    return number.to_bytes(nbytes, "little", signed=signed)


def padding(value, size=4):
    """Calculates the next multiple for size"""
    if remainder := value % size:
        return value + size - remainder
    return value
```

```
# Letter to color conversion table
# in RGB format (red, green, blue)
# Each color can range from 0 to 255.
letter_to_color = {
    " ": (0, 0, 0),  # black
    "r": (255, 0, 0),  # red
    "g": (0, 255, 0),  # green
    "b": (0, 0, 255),  # blue
}

# Point multiplier
# Each point will be copied multiplier times in the image
# If equal to 4, each point generates a 4x4 point block
multiplier = 32

# Check if all lines have the same size
drawing_width = len(drawing[0])

for line, z in enumerate(drawing):
    if len(z) != drawing_width:
        raise ValueError(
            f"Lines must have the same size. Line with different width:
{line} instead of {len(z)}"
        )

# Calculate data based on multiplier
expanded_drawing = []
for line in drawing:
    new_line = []
    for letter in line:
        new_line.append(letter * multiplier)
    for _ in range(multiplier):
        expanded_drawing.append("".join(new_line))


width = len(expanded_drawing[0])  # Number of columns in the image
height = len(expanded_drawing)  # Number of lines in the image

# Check if letters represent colors
binary_data = []
for line in expanded_drawing:
    binary_line = []
```

```
    for character in line:
        # Invert byte order for BMP RGB format
        binary_line.append(bytes(letter_to_color[character][::-1]))
    binary_data.append(b"".join(binary_line))

# Add padding
width_bytes = width * 3
width_with_padding = padding(width_bytes)
if width_bytes != width_with_padding:
    for p, d in enumerate(binary_data):
        binary_data[p] = b"".join(
            [binary_data[p], bytes(width_with_padding - width_bytes)]
        )

# Calculate image size in bytes with padding
size = padding(width * 3) * height

bmp_header = [
    b"BM",  # Identifier
    bytes_little_endian(54 + size),  # Image size in bytes
    bytes(4),  # 4 bytes 0x00
    bytes_little_endian(54),  # Header size
]

dib_header = [
    bytes_little_endian(40),  # DIB header size
    bytes_little_endian(width),
    bytes_little_endian(
        -height, signed=True
    ),  # Negative height to build image top to bottom
    bytes_little_endian(1, 2),  # Color planes
    bytes_little_endian(24, 2),  # Bits per pixel
    bytes_little_endian(0),  # No compression
    bytes_little_endian(size),
    bytes_little_endian(2835),  # ceil(72 dpi x 39.3701 in/m) horizontal
    bytes_little_endian(2835),  # ceil(72 dpi x 39.3701 in/m) vertical
    bytes_little_endian(0),  # Number of colors in palette
    bytes_little_endian(0),  # Important colors
]

bmp_header_binary = b"".join(bmp_header)
dib_header_binary = b"".join(dib_header)
```

```
binary_data = b"".join(binary_data)

# Verify binary header sizes
assert len(bmp_header_binary) == 14
assert len(dib_header_binary) == 40
assert len(binary_data) == size


# Write the image
with open(output_file, "wb") as f:
    f.write(bmp_header_binary)
    f.write(dib_header_binary)
    f.write(binary_data)

print(f"File {output_file} generated. {width=} x {height=} {size=} bytes")
```

Exercise 09-44

Modify the previous program to receive a third parameter with the color conversion table in JSON format.

```
import sys
import json

# Check if both filenames were provided as arguments
if len(sys.argv) != 4:
    print(
        "Usage: python exercise-09-44.py output_file.bmp drawing_file.txt
color_table.json"
    )
    sys.exit(1)

output_file = sys.argv[1]
drawing_file = sys.argv[2]
color_file = sys.argv[3]

# Read the drawing from the file
try:
    with open(drawing_file, "r") as f:
        drawing = [line.strip() for line in f.readlines()]
except FileNotFoundError:
    print(f"Error: Drawing file '{drawing_file}' not found.")
    sys.exit(1)
except IOError:
    print(f"Error: Could not read drawing file '{drawing_file}'.")
    sys.exit(1)

# Load the color table from the JSON file
try:
    with open(color_file, "r") as f:
        letter_to_color = json.load(f)
except FileNotFoundError:
    print(f"Error: Color file '{color_file}' not found.")
    sys.exit(1)
except json.JSONDecodeError:
    print(f"Error: File '{color_file}' does not contain valid JSON.")
    sys.exit(1)
```

```
# Verify and build the color table
for char in letter_to_color:
    color = letter_to_color[char]
    if len(color) != 3 or not all(isinstance(x, int) and 0 <= x <= 255 for
x in color):
        print(f"Error: Invalid color for character '{char}'")
        sys.exit(1)


def bytes_little_endian(number, nbytes=4, signed=False):
    """Converts an integer to a sequence of bytes using little endian
encoding.
    If signed is passed, reserves a bit to represent the sign."""
    return number.to_bytes(nbytes, "little", signed=signed)


def padding(value, size=4):
    """Calculates the next multiple for size"""
    if remainder := value % size:
        return value + size - remainder
    return value


# Point multiplier
# Each point will be copied multiplier times in the image
# If equal to 4, each point generates a 4x4 point block
multiplier = 32

# Check if all lines have the same size
drawing_width = len(drawing[0])

for line, z in enumerate(drawing):
    if len(z) != drawing_width:
        raise ValueError(
            f"Lines must have the same size. Line with different width:
{line} instead of {len(z)}"
        )

# Calculate data based on multiplier
expanded_drawing = []
for line in drawing:
    new_line = []
```

```
    for letter in line:
        new_line.append(letter * multiplier)
    for _ in range(multiplier):
        expanded_drawing.append("".join(new_line))


width = len(expanded_drawing[0])  # Number of columns in the image
height = len(expanded_drawing)  # Number of lines in the image

# Check if letters represent colors
binary_data = []
for line in expanded_drawing:
    binary_line = []
    for char in line:
        # Invert byte order for BMP RGB format
        binary_line.append(bytes(letter_to_color[char][::-1]))
    binary_data.append(b"".join(binary_line))

# Add padding
width_bytes = width * 3
width_with_padding = padding(width_bytes)
if width_bytes != width_with_padding:
    for p, d in enumerate(binary_data):
        binary_data[p] = b"".join(
            [binary_data[p], bytes(width_with_padding - width_bytes)]
        )

# Calculate image size in bytes with padding
size = padding(width * 3) * height

bmp_header = [
    b"BM",  # Identifier
    bytes_little_endian(54 + size),  # Image size in bytes
    bytes(4),  # 4 bytes 0x00
    bytes_little_endian(54),  # Header size
]

dib_header = [
    bytes_little_endian(40),  # DIB header size
    bytes_little_endian(width),
    bytes_little_endian(
        -height, signed=True
```

```
    ),  # Negative height to build image top to bottom
    bytes_little_endian(1, 2),  # Color planes
    bytes_little_endian(24, 2),  # Bits per pixel
    bytes_little_endian(0),  # No compression
    bytes_little_endian(size),
    bytes_little_endian(2835),  # ceil(72 dpi x 39.3701 in/m) horizontal
    bytes_little_endian(2835),  # ceil(72 dpi x 39.3701 in/m) vertical
    bytes_little_endian(0),  # Number of colors in palette
    bytes_little_endian(0),  # Important colors
]

bmp_header_binary = b"".join(bmp_header)
dib_header_binary = b"".join(dib_header)
binary_data = b"".join(binary_data)

# Verify binary header sizes
assert len(bmp_header_binary) == 14
assert len(dib_header_binary) == 40
assert len(binary_data) == size


# Write the image
with open(output_file, "wb") as f:
    f.write(bmp_header_binary)
    f.write(dib_header_binary)
    f.write(binary_data)

print(f"File {output_file} generated. {width=} x {height=} {size=} bytes")
```

Chapter 10

Exercise 10-01

Add size and brand attributes to the Television class. Create two Television objects and assign them different sizes and brands. Then, print the value of those attributes to confirm the independence of the values of each instance (object).

```
class Television:
    def __init__(self):
        self.powered = False
        self.channel = 2
        self.size = 20
        self.brand = "EastTiger"


tv = Television()
tv.size = 27
tv.brand = "DingDang"
living_room_tv = Television()
living_room_tv.size = 52
living_room_tv.brand = "XangLa"

print(f"tv size={tv.size} brand={tv.brand}")
print(f"living_room_tv size={living_room_tv.size} brand={living_room_tv.
brand}")
```

Exercise 10-02

Currently, the Television class initializes the channel with 2. Modify the Television class to receive the initial channel in its constructor as an optional parameter.

```
class Television:
    def __init__(self, initial_channel, min, max):
        self.powered = False
        self.channel = initial_channel
        self.min_channel = min
        self.max_channel = max

    def channel_down(self):
        if self.channel - 1 >= self.min_channel:
            self.channel -= 1

    def channel_up(self):
        if self.channel + 1 <= self.max_channel:
            self.channel += 1


tv = Television(5, 1, 99)

print(tv.channel)
```

Exercise 10-03

Modify the Television class so that, if we ask to change the channel down beyond the minimum, it goes to the maximum channel, and vice versa.

Example: >>> tv = Television(2, 10) >>> tv.change_channel_down() >>> tv.channel 10

>>> tv.change_channel_up() >>> tv.channel 2

```
class Television:
    def __init__(self, min, max):
        self.powered = False
        self.channel = min
        self.min_channel = min
        self.max_channel = max

    def channel_down(self):
        if self.channel - 1 >= self.min_channel:
            self.channel -= 1
        else:
            self.channel = self.max_channel

    def channel_up(self):
        if self.channel + 1 <= self.max_channel:
            self.channel += 1
        else:
            self.channel = self.min_channel


tv = Television(2, 10)
tv.channel_down()
print(tv.channel)
tv.channel_up()
print(tv.channel)
```

Exercise 10-04

Using what we learned with functions, modify the Television class constructor so that channel_min and channel_max are optional parameters, where channel_min defaults to 2 and channel_max defaults to 14.

```
class Television:
    def __init__(self, min=2, max=14):
        self.powered = False
        self.channel = min
        self.min_channel = min
        self.max_channel = max

    def channel_down(self):
        if self.channel - 1 >= self.min_channel:
            self.channel -= 1
        else:
            self.channel = self.max_channel

    def channel_up(self):
        if self.channel + 1 <= self.max_channel:
            self.channel += 1
        else:
            self.channel = self.min_channel


tv = Television()
tv.channel_down()
print(tv.channel)
tv.channel_up()
print(tv.channel)
```

Exercise 10-05

Using the Television class modified in the previous exercise, create two instances (objects), specifying the channel_min and channel_max value by name.

```
class Television:
    def __init__(self, min=2, max=14):
        self.powered = False
        self.channel = min
        self.min_channel = min
        self.max_channel = max

    def channel_down(self):
        if self.channel - 1 >= self.min_channel:
            self.channel -= 1
        else:
            self.channel = self.max_channel

    def channel_up(self):
        if self.channel + 1 <= self.max_channel:
            self.channel += 1
        else:
            self.channel = self.min_channel


tv = Television(min=1, max=22)
tv.channel_down()
print(tv.channel)
tv.channel_up()
print(tv.channel)

tv2 = Television(min=2, max=64)
tv2.channel_down()
print(tv2.channel)
tv2.channel_up()
print(tv2.channel)
```

Exercise 10-06

Modify the Television class so that the mute_channel_up and change_channel_down methods return the channel after the change.

```
class Television:
    def __init__(self, min=2, max=14):
        self.powered = False
        self.channel = min
        self.min_channel = min
        self.max_channel = max

    def channel_down(self):
        if self.channel - 1 >= self.min_channel:
            self.channel -= 1
        else:
            self.channel = self.max_channel
        return self.channel

    def channel_up(self):
        if self.channel + 1 <= self.max_channel:
            self.channel += 1
        else:
            self.channel = self.min_channel
        return self.channel
```

Exercise 10-07

Change the Television class to only accept the commands to change channels if turned on.

```
class Television:
    def __init__(self, min=2, max=14):
        self.powered = False
        self.channel = min
        self.min_channel = min
        self.max_channel = max

    def channel_down(self):
        if not self.powered:
            return
        if self.channel - 1 >= self.min_channel:
            self.channel -= 1
        else:
            self.channel = self.max_channel

    def channel_up(self):
        if not self.powered:
            return
        if self.channel + 1 <= self.max_channel:
            self.channel += 1
        else:
            self.channel = self.min_channel
```

Exercise 10-08

Change the program to produce a message stating the user has an insufficient account balance if the user attempts to withdraw more than the available balance.

```
# Modify the accounts.py file from the listings


class Account:
    def __init__(self, clients, number, balance=0):
        self.balance = 0
        self.clients = clients
        self.number = number
        self.operations = []
        self.deposit(balance)

    def summary(self):
        print(f"AC N°{self.number} Balance: {self.balance:10.2f}")

    def withdraw(self, value):
        if self.balance >= value:
            self.balance -= value
            self.operations.append(["WITHDRAW", value])
        else:
            print("Insufficient balance!")

    def deposit(self, value):
        self.balance += value
        self.operations.append(["DEPOSIT", value])

    def statement(self):
        print(f"Statement AC N° {self.number}\n")
        for o in self.operations:
            print(f"{o[0]:10s} {o[1]:10.2f}")
        print(f"\n    Balance: {self.balance:10.2f}\n")


class SpecialAccount(Account):
    def __init__(self, clients, number, balance=0, limit=0):
        Account.__init__(self, clients, number, balance)
        self.limit = limit
```

```
    def withdraw(self, value):
        if self.balance + self.limit >= value:
            self.balance -= value
            self.operations.append(["WITHDRAW", value])
        else:
            Account.withdraw(self, value)
```

Exercise 10-09

Modify the summary method of the Account class to display the name and phone number of each client.

```
# Here accounts.py and clients.py were copied into a single file.
# This change is only to facilitate the visualization
# of the answer to this exercise.


class Client:
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone


class Account:
    def __init__(self, clients, number, balance=0):
        self.balance = 0
        self.clients = clients
        self.number = number
        self.operations = []
        self.deposit(balance)

    def summary(self):
        print(f"AC N°{self.number} Balance: {self.balance:10.2f}\n")
        for client in self.clients:
            print(f"Name: {client.name}\nPhone: {client.phone}\n")

    def withdraw(self, value):
        if self.balance >= value:
            self.balance -= value
            self.operations.append(["WITHDRAW", value])
        else:
            print("Insufficient balance!")

    def deposit(self, value):
        self.balance += value
        self.operations.append(["DEPOSIT", value])

    def statement(self):
        print(f"Statement AC N° {self.number}\n")
```

```
        for o in self.operations:
            print(f"{o[0]:10s} {o[1]:10.2f}")
        print(f"\n    Balance: {self.balance:10.2f}\n")


mary = Client("Mary", "1243-3321")
john = Client("John", "5554-3322")

account = Account([mary, john], 1234, 5000)
account.summary()
```

Exercise 10-10

Create a new account with Joao and Jose as clients and a balance of $500.

```
# Here accounts.py and clients.py were copied into a single file.
# This change is only to facilitate the visualization
# of the answer to this exercise.


class Client:
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone


class Account:
    def __init__(self, clients, number, balance=0):
        self.balance = 0
        self.clients = clients
        self.number = number
        self.operations = []
        self.deposit(balance)

    def summary(self):
        print(f"AC N°{self.number} Balance: {self.balance:10.2f}\n")
        for client in self.clients:
            print(f"Name: {client.name}\nPhone: {client.phone}\n")

    def withdraw(self, value):
        if self.balance >= value:
            self.balance -= value
            self.operations.append(["WITHDRAW", value])
        else:
            print("Insufficient balance!")

    def deposit(self, value):
        self.balance += value
        self.operations.append(["DEPOSIT", value])

    def statement(self):
        print(f"Statement AC N° {self.number}\n")
        for o in self.operations:
```

```
            print(f"{o[0]:10s} {o[1]:10.2f}")
        print(f"\n    Balance: {self.balance:10.2f}\n")


john = Client("John", "5554-3322")
joe = Client("Joe", "1243-3321")

account = Account([john, joe], 2341, 500)
account.summary()
```

Exercise 10-11

Create classes representing states and cities. Each state has a name, an acronym, and cities. Each city has a name and population. Write a test program to create three states with a few cities. Display the population of each state as the sum of the population of its cities.

```
class State:
    def __init__(self, name, acronym):
        self.name = name
        self.acronym = acronym
        self.cities = []

    def add_city(self, city):
        city.state = self
        self.cities.append(city)

    def population(self):
        return sum([c.population for c in self.cities])


class City:
    def __init__(self, name, population):
        self.name = name
        self.population = population
        self.state = None

    def __str__(self):
        return (
            f"City (name={self.name}, population={self.population},
state={self.state})"
        )


# Populations obtained from Wikipedia
# IBGE 2012 estimate
am = State("Amazonas", "AM")
am.add_city(City("Manaus", 1861838))
am.add_city(City("Parintins", 103828))
am.add_city(City("Itacoatiara", 89064))

sp = State("São Paulo", "SP")
```

```
sp.add_city(City("São Paulo", 11376685))
sp.add_city(City("Guarulhos", 1244518))
sp.add_city(City("Campinas", 1098630))

rj = State("Rio de Janeiro", "RJ")
rj.add_city(City("Rio de Janeiro", 6390290))
rj.add_city(City("São Gonçalo", 1016128))
rj.add_city(City("Duque de Caixias", 867067))


for state in [am, sp, rj]:
    print(f"State: {state.name} Acronym: {state.acronym}")
    for city in state.cities:
        print(f"City: {city.name} Population: {city.population}")
    print(f"State Population: {state.population()}\n")
```

Exercise 10-12

Modify the Account and SpecialAccount classes so that the withdrawal operation returns True if the withdrawal was made and False if the withdrawal failed.

```
# Here accounts.py and clients.py were copied into a single file.
# This change is only to facilitate the visualization
# of the answer to this exercise.


class Client:
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone


# Modify the accounts.py file from the listings


class Account:
    def __init__(self, clients, number, balance=0):
        self.balance = 0
        self.clients = clients
        self.number = number
        self.operations = []
        self.deposit(balance)

    def summary(self):
        print(f"AC N°{self.number} Balance: {self.balance:10.2f}")

    def withdraw(self, value):
        if self.balance >= value:
            self.balance -= value
            self.operations.append(["WITHDRAW", value])
            return True
        else:
            print("Insufficient balance!")
            return False

    def deposit(self, value):
        self.balance += value
        self.operations.append(["DEPOSIT", value])
```

```
    def statement(self):
        print(f"Statement AC N° {self.number}\n")
        for o in self.operations:
            print(f"{o[0]:10s} {o[1]:10.2f}")
        print(f"\n    Balance: {self.balance:10.2f}\n")


class SpecialAccount(Account):
    def __init__(self, clients, number, balance=0, limit=0):
        Account.__init__(self, clients, number, balance)
        self.limit = limit

    def withdraw(self, value):
        if self.balance + self.limit >= value:
            self.balance -= value
            self.operations.append(["WITHDRAW", value])
            return True
        else:
            return Account.withdraw(self, value)


john = Client("John", "5554-3322")
joe = Client("Joe", "1243-3321")

account = Account([john, joe], 2341, 500)
account.summary()
print(account.withdraw(1000))
print(account.withdraw(100))
account.summary()

account2 = SpecialAccount([joe], 3432, 50000, 10000)
account2.summary()
print(account2.withdraw(100000))
print(account2.withdraw(500))
account2.summary()
```

Exercise 10-13

Change the SpecialAccount class so that your statement shows the limit and the total available for withdrawal.

```
# Here accounts.py and clients.py were copied into a single file.
# This change is only to facilitate the visualization
# of the answer to this exercise.


class Client:
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone


# Modify the accounts.py file from the listings


class Account:
    def __init__(self, clients, number, balance=0):
        self.balance = 0
        self.clients = clients
        self.number = number
        self.operations = []
        self.deposit(balance)

    def summary(self):
        print(f"AC N°{self.number} Balance: {self.balance:10.2f}")

    def withdraw(self, value):
        if self.balance >= value:
            self.balance -= value
            self.operations.append(["WITHDRAW", value])
            return True
        else:
            print("Insufficient balance!")
            return False

    def deposit(self, value):
        self.balance += value
        self.operations.append(["DEPOSIT", value])
```

```
    def statement(self):
        print(f"Statement AC N° {self.number}\n")
        for o in self.operations:
            print(f"{o[0]:10s}   {o[1]:10.2f}")
        print(f"\n      Balance: {self.balance:10.2f}\n")


class SpecialAccount(Account):
    def __init__(self, clients, number, balance=0, limit=0):
        Account.__init__(self, clients, number, balance)
        self.limit = limit

    def withdraw(self, value):
        if self.balance + self.limit >= value:
            self.balance -= value
            self.operations.append(["WITHDRAW", value])
            return True
        else:
            return Account.withdraw(self, value)

    def statement(self):
        Account.statement(self)
        print(f"\n     Limit: {self.limit:10.2f}\n")
        print(f"\n Available: {self.limit + self.balance:10.2f}\n")


joe = Client("Joe", "1243-3321")

account = SpecialAccount([joe], 3432, 50000, 10000)
account.statement()
```

Exercise 10-14

Observe the withdrawal method from the Account and SpecialAccount classes. Modify the withdrawal method of the Account class so that the possibility of withdrawal is verified by a new method, replacing the current condition. This new method should return True if the withdrawal can be performed or False if not. Modify the SpecialAccount class to work with this new method. Check whether you need to change the SpecialAccount withdrawal method or just the new method created to verify the possibility of withdrawing.

```
# Here accounts.py and clients.py were copied into a single file.
# This change is only to facilitate the visualization
# of the answer to this exercise.


class Client:
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone


# Modify the accounts.py file from the listings


class Account:
    def __init__(self, clients, number, balance=0):
        self.balance = 0
        self.clients = clients
        self.number = number
        self.operations = []
        self.deposit(balance)

    def summary(self):
        print(f"AC N°{self.number} Balance: {self.balance:10.2f}")

    def can_withdraw(self, value):
        return self.balance >= value

    def withdraw(self, value):
        if self.can_withdraw(value):
            self.balance -= value
            self.operations.append(["WITHDRAW", value])
```

```
            return True
        else:
            print("Insufficient balance!")
            return False

    def deposit(self, value):
        self.balance += value
        self.operations.append(["DEPOSIT", value])

    def statement(self):
        print(f"Statement AC N° {self.number}\n")
        for o in self.operations:
            print(f"{o[0]:10s}   {o[1]:10.2f}")
        print(f"\n      Balance: {self.balance:10.2f}\n")


class SpecialAccount(Account):
    def __init__(self, clients, number, balance=0, limit=0):
        Account.__init__(self, clients, number, balance)
        self.limit = limit

    def can_withdraw(self, value):
        return self.balance + self.limit >= value

    def statement(self):
        Account.statement(self)
        print(f"\n     Limit: {self.limit:10.2f}\n")
        print(f"\n Available: {self.limit + self.balance:%10.2f}\n")


# Note that with the can_withdraw method of SpecialAccount
# we don't even need to write a special withdraw method!

joe = Client("Joe", "1243-3321")

account = SpecialAccount([joe], 3432, 5000, 1000)
account.statement()
account.withdraw(6000)
account.withdraw(3000)
account.withdraw(1000)
account.statement()
```

Exercise 10-15

Modify the UniqueList class to override the UserList.extend method. extend works like append but takes a list as a parameter. Check the type of each element in the list before adding it.

```
from collections import UserList


class UniqueList(UserList):
    def __init__(self, elem_class, enumerable=None):
        super().__init__(enumerable)
        self.elem_class = elem_class

    def append(self, elem):
        self.check_type(elem)
        if elem not in self.data:
            super().append(elem)

    def extend(self, iterable):
        for elem in iterable:
            self.append(elem)

    def __setitem__(self, position, elem):
        self.check_type(elem)
        if elem not in self.data:
            super().__setitem__(position, elem)

    def check_type(self, elem):
        if not isinstance(elem, self.elem_class):
            raise TypeError("Invalid type")
```

Chapter 11

Exercise 11-01

Make a program that creates the prices.db database with the price table to store a list of sales prices for products. The table must contain the name of the product and its respective price. The program must also insert some data for testing.

```
import sqlite3
from contextlib import closing

with sqlite3.connect("prices.db") as connection:
    with closing(connection.cursor()) as cursor:
        cursor.execute(
            """
                create table prices(
                    name text,
                    price numeric)
                """
        )
        cursor.execute(
            """
                insert into prices (name, price)
                    values(?, ?)
                    """,
            ("Potato", "3.20"),
        )
        cursor.execute(
            """
                insert into prices (name, price)
                    values(?, ?)
                    """,
            ("Bread", "1.20"),
        )
        cursor.execute(
            """
                insert into prices (name, price)
                    values(?, ?)
                    """,
            ("Papaya", "2.14"),
        )
```

Exercise 11-02

Make a program to list all the prices in the prices.db database.

```
import sqlite3
from contextlib import closing

with sqlite3.connect("prices.db") as connection:
    with closing(connection.cursor()) as cursor:
        cursor.execute("""select * from prices""")
        for result in cursor.fetchall():
            print("Name: {0:30s} Price: {1:6.2f}".format(*result))
```

Exercise 11-03

Write a program that performs queries in the prices.db database created in Exercise 11.1. The program must ask for the product's name and list its price.

```
import sqlite3
from contextlib import closing

with sqlite3.connect("prices.db") as connection:
    with closing(connection.cursor()) as cursor:
        while True:
            name = input("Product name to search [blank to exit]: ")
            if not name:
                break
            cursor.execute("""select * from prices where name = ?""",
(name,))
            found = 0
            for result in cursor.fetchall():
                print("Name: {0:30s} Price: {1:6.2f}".format(*result))
                found += 1
            if found == 0:
                print("Not found.")
            else:
                print("{} product(s) found.".format(found))
```

Exercise 11-04

Modify the Exercise 11.3 program to ask for two values and list all the products with prices between those two values.

```
import sqlite3
from contextlib import closing

with sqlite3.connect("prices.db") as connection:
    with closing(connection.cursor()) as cursor:
        price1 = input("Enter the minimum price to list: ")
        price2 = input("Enter the maximum price to list: ")

        cursor.execute(
            """select * from prices
                        where price >= ? and price <= ?""",
            (price1, price2),
        )
        found = 0
        for result in cursor.fetchall():
            print("Name: {0:30s} Price: {1:6.2f}".format(*result))
            found += 1
        if found == 0:
            print("Not found.")
        else:
            print("{} product(s) found.".format(found))
```

Exercise 11-05

Write a program that increases the price of all products from the prices.db database by 10%.

```
import sqlite3
from contextlib import closing

with sqlite3.connect("prices.db") as connection:
    with closing(connection.cursor()) as cursor:
        cursor.execute(
            """update prices
                        set price = price * 1.1"""
        )
        # Show the new prices
        cursor.execute("""select * from prices""")
        for result in cursor.fetchall():
            print("Name: {0:30s} Price: {1:6.2f}".format(*result))
```

Exercise 11-06

Write a program that asks for the product's name and a new price. Using the prices.db database, update the product price with the same name in the database.

```
import sqlite3
from contextlib import closing

with sqlite3.connect("prices.db") as connection:
    with closing(connection.cursor()) as cursor:
        name = input("Enter the product name to change price: ")

        cursor.execute(
            """select * from prices
                        where name = ?""",
            (name,),
        )

        result = cursor.fetchone()
        if result:
            print("Name: {0:30s} Price: {1:6.2f}".format(*result))
            new_price = input("Enter the new price: ")
            cursor.execute(
                """update prices
                            set price = ?
                            where name = ?""",
                (new_price, name),
            )
        else:
            print("Not found.")
```

Chapter 12

Exercise 12-01

Modify the previous program to recognize letter sequences. A letter is a character between A and Z or between a and z (you must account for uppercase and lowercase letters). Ignore accented characters. Print a list with the strings of letters found.

```
input_string = "ABC431DEF901c431203FXEW9"
output = []
number = []

for character in input_string:
    if "a" <= character.lower() <= "z":
        if not number:
            output.append(number)
        number += character
    elif number:
        number = []

for found in output:
    print("".join(found))
```

Exercise 12-02

Using the pattern_check function, rewrite the function that parsed the numbers in the entry ABC431DEF901C431203FXEW9 .

```
input_string = "ABC431DEF901c431203FXEW9"


def check_pattern(input_string, patterns):
    position = 0
    for pattern in patterns:
        found, _, end = pattern(input_string[position:])
        if found > 0:
            position += end + 1
        else:
            return -1, -1, -1
    return 1, 0, position - 1


def numbers(input_string):
    found = 0
    end = -1
    for i, character in enumerate(input_string):
        if "0" <= character <= "9":
            found += 1
            end = i
        else:
            break
    return found, 0, end


position = 0
while position < len(input_string):
    found, start, end = check_pattern(input_string[position:], [numbers])
    if found > 0:
        print(input_string[position : position + end + 1])
        position += end + 1
    else:
        position += 1
```

Exercise 12-03

Using the pattern_check function, write a function that detects a date in the format dd/mm/yy where dd is the day, mm the month, and yy the year. The function should only detect the date pattern and it does not need to verify whether the date is valid.

```
# Functions number, sequence and check pattern from listing 12.5
from functools import partial


def number(input_string, min_qty, max_qty):
    num = 0
    for character in input_string:
        if character.isnumeric():
            num += 1
        else:
            break
    if min_qty <= num <= max_qty:
        return num, 0, num - 1
    else:
        return -1, -1, -1


def sequence(input_string, pattern):
    position, max_position = 0, len(pattern)
    for character in input_string:
        if character == pattern[position]:
            position += 1  # Same characters, test next character
        else:
            break  # Out of sequence
        if position == max_position:  # Found entire sequence
            return 1, 0, position - 1
    return -1, -1, -1


def check_pattern(input_string, patterns):
    position = 0
    for pattern in patterns:
        found, _, end = pattern(input_string[position:])
        if found > 0:
            position += end + 1
        else:
```

```
            return -1, -1, -1
    return 1, 0, position - 1


two_numbers = partial(number, min_qty=2, max_qty=2)
slash = partial(sequence, pattern="/")
pattern = [two_numbers, slash, two_numbers, slash, two_numbers]

# Here's a small list of inputs to facilitate visualization
inputs = [
    "12/03/24",  # Pattern found
    "12/3/2024",  # Pattern not found
    "Twelfth of March 12/03",  # Pattern not found
    "12-03-24 12/03/2024 abc 21/30/24",  # Pattern found
    "12/03/24 12/03/624 abc 21/30/24",  # Pattern found twice
]
for input_string in inputs:
    print("Input:", input_string)
    found = False
    position = 0
    while position < len(input_string):
        found, start, end = check_pattern(input_string[position:],
pattern)
        if found > 0:
            print(f"Date positions: {position+start} to {position+end} ",
end="")
            print("Date:", input_string[position + start : position + end
+ 1])
            found = True
            position += end + 1
        else:
            position += 1
    if not found:
        print("No date found in input")
    print()
```

Exercise 12-04

Using the pattern_check function, write a function that detects a value in dollars in the format $999.99, where 9 represents any digit. The first number can have one or more digits, but the second part (cents) must have a maximum of two digits.

```
# Functions number, sequence and check pattern from listing 12.5
from functools import partial


def number(input_string, min_qty, max_qty):
    num = 0
    for character in input_string:
        if character.isnumeric():
            num += 1
        else:
            break
    if min_qty <= num <= max_qty:
        return num, 0, num - 1
    else:
        return -1, -1, -1


def sequence(input_string, pattern):
    position, max_position = 0, len(pattern)
    for character in input_string:
        if character == pattern[position]:
            position += 1  # Same characters, test next character
        else:
            break  # Out of sequence
        if position == max_position:  # Found entire sequence
            return 1, 0, position - 1
    return -1, -1, -1


def check_pattern(input_string, patterns):
    position = 0
    for pattern in patterns:
        found, _, end = pattern(input_string[position:])
        if found > 0:
            position += end + 1
        else:
```

```
            return -1, -1, -1
    return 1, 0, position - 1


# Bonus, how to recognize an optional pattern.
# Not part of the exercise, but interesting.
def optional(input_string, patterns):
    found, start, end = check_pattern(input_string, patterns)
    if found > 0:
        return found, start, end
    else:
        return 1, -1, -1


three_numbers = partial(number, min_qty=1, max_qty=3)
cents = partial(number, min_qty=1, max_qty=2)
dollar_sign = partial(sequence, pattern="$")
comma = partial(sequence, pattern=",")

# Only the values mentioned in the exercise
pattern = [dollar_sign, three_numbers, comma, cents]
# Using the optional function to accept values without cents (bonus)
# pattern = [dollar_sign, three_numbers, partial(optional,
patterns=[comma, cents])]

# Here's a small list of inputs to facilitate visualization
inputs = [
    "123,45",  # Pattern found
    "$123,450",  # Pattern not found
    "$123,45",  # Pattern not found
    "$12,34",  # Pattern found
    "$123,45 $12,34 $1,23 $1,0",  # Pattern found four times
    "$123 $12 $1 $1,0",  # Pattern found four times (if bonus - optional
enabled), once otherwise
]
for input_string in inputs:
    print("Input:", input_string)
    found = False
    position = 0
    while position < len(input_string):
        found, start, end = check_pattern(input_string[position:],
pattern)
```

```
        if found > 0:
            print(f"Dollars at positions: {position+start} to
{position+end} ", end="")
            print("Dollars:", input_string[position + start : position +
end + 1])
            found = True
            position += end + 1
        else:
            position += 1
    if not found:
        print("No dollar values found in input")
    print()
```

Exercise 12-05

Create a sequence function that receives qmax and qmin. It should work similarly to number but call the sequence function. It should also work when qmin is 0 when the sequence is optional.

```
# Functions sequence and check_pattern from listing 12.5
from functools import partial


def sequence(input, pattern):
    position, max_position = 0, len(pattern)
    for character in input:
        if character == pattern[position]:
            position += 1  # Same characters, test next character
        else:
            break  # Out of sequence
        if position == max_position:  # Found entire sequence
            return 1, 0, position - 1
    return -1, -1, -1


def sequences(input, pattern, min_qty=1, max_qty=1):
    position = 0
    end = -1
    found_count = 0
    while position < len(input):
        found, _, iend = sequence(input[position:], pattern)
        if found > 0:
            found_count += 1
            position += iend + 1
            end = position - 1
        else:
            break
    # If pattern is optional, returns found 1, but start and end -1
    # so that check_pattern continues searching
    if min_qty == 0 and found_count == 0:
        return 1, -1, -1
    # Checks if number of findings is between min_qty and max_qty
    elif min_qty <= found_count <= max_qty:
        return found_count, 0, end
    else:
```

```
        return -1, -1, -1


def check_pattern(input, patterns):
    position = 0
    for pattern in patterns:
        found, _, end = pattern(input[position:])
        if found > 0:
            position += end + 1
        else:
            return -1, -1, -1
    return 1, 0, position - 1


inputs = [
    "(((---)))",  # Pattern found
    "(((--)))",  # Pattern not found
    "(\----)",  # Pattern not found
    "\----",  # Pattern not found
    "((--))",  # Pattern not found
    "<(((--)))>",  # Pattern found
    "<<(((--)))>>",  # Pattern found
    "<<(((---)))>>",  # Pattern found
    "<<((--))>> <(((---)))> (((---))) ((((\----))))",  # Pattern found
twice
]
# The pattern is a sequence of characters that can be optional
# < zero or up to two times
# ( three or up to four times
# - two or up to three times
# ) three or up to four times
# > zero or up to two times
# You can create other patterns to test the sequences function
pattern = [
    partial(sequences, pattern="<", min_qty=0, max_qty=2),
    partial(sequences, pattern="(", min_qty=3, max_qty=4),
    partial(sequences, pattern="-", min_qty=2, max_qty=3),
    partial(sequences, pattern=")", min_qty=3, max_qty=4),
    partial(sequences, pattern=">", min_qty=0, max_qty=2),
]

for input in inputs:
```

```
    print("Input:", input)
    found = False
    position = 0
    while position < len(input):
        found, start, end = check_pattern(input[position:], pattern)
        if found > 0:
            print(f"Pattern at positions: {position+start} to
{position+end} ", end="")
            print("Pattern:", input[position + start : position + end +
1])
            found = True
            position += end + 1
        else:
            position += 1
    if not found:
        print("No pattern found in input")
    print()
```

Exercise 12-06

Create a function using pattern_check that validates cell phone numbers. A cell phone has 9 digits after the LDC (for example, (92)99812-1103).

```
# Functions number, sequence and check pattern from listing 12.5
from functools import partial


def number(input, min_qty, max_qty):
    num = 0
    for character in input:
        if character.isnumeric():
            num += 1
        else:
            break
    if min_qty <= num <= max_qty:
        return num, 0, num - 1
    else:
        return -1, -1, -1


def sequence(input, pattern):
    position, max_position = 0, len(pattern)
    for character in input:
        if character == pattern[position]:
            position += 1  # Same characters, test next character
        else:
            break  # Out of sequence
        if position == max_position:  # Found entire sequence
            return 1, 0, position - 1
    return -1, -1, -1


def check_pattern(input, patterns):
    position = 0
    for pattern in patterns:
        found, _, end = pattern(input[position:])
        if found > 0:
            position += end + 1
        else:
            return -1, -1, -1
```

```
    return 1, 0, position - 1


def cell_number(input):
    pattern = [
        partial(sequence, pattern="("),
        partial(number, min_qty=2, max_qty=3),
        partial(sequence, pattern=")"),
        partial(number, min_qty=5, max_qty=5),
        partial(sequence, pattern="-"),
        partial(number, min_qty=4, max_qty=4),
    ]
    found, _, _ = check_pattern(input, pattern)
    return found > 0


inputs = [
    "(92)99999-9999",  # Yes
    "(11)99999-999",  # No
    "(2)99999-9999",  # No
    "(12)9999999999",  # No
    "(312)9999999999",  # No
    "(312)99999-9999",  # Yes
]

for input in inputs:
    print(f"{input}: is it a cell number? {'Yes' if cell_number(input)
else 'No'}")
```

Exercise 12-07

Write a program that validates user data entry. It will validate ISBNs (International Standard Book Numbers), which are used to identify books worldwide. The program must accept ISBNs in the following format: ISBN 999-9-99-999999-9, where each 9 represents a digit. The ISBN letters are optional and be case-insensitive. The space between N and the first number is mandatory if the ISBN is present. Require the dashes as in the example, verifying the correct number of digits.

Valid values: ISBN 123-3-12-123456-1 Isbn 123-3-12-123456-1 123-3-12-123456-1

Invalid values: ISBN123-3-12-123456-1 ISBN 1233-12123456-1

```
import re


def validate_isbn(isbn):
    # Regex pattern to validate the ISBN format
    # ^ - start of string
    # (?:ISBN\s+)? - optional "ISBN" followed by at least one space
    # \d{3}-\d{1}-\d{2}-\d{6}-\d{1} - specific format with hyphens
    # $ - end of string
    # flags=re.IGNORECASE - ignore case for "ISBN"
    pattern = r"^(?:ISBN\s+)?\d{3}-\d{1}-\d{2}-\d{6}-\d{1}$"

    return re.match(pattern, isbn, flags=re.IGNORECASE)


print("Enter ISBNs for validation (or 'exit' to quit)")
print("Expected format: ISBN 999-9-99-999999-9\n")

while True:
    isbn_input = input("Enter an ISBN: ").strip()

    # Check if the user wants to exit
    if isbn_input.lower() in ["sair", "exit", "quit", "q"]:
        print("Program terminated.")
        break

    # Validate the ISBN
    if validate_isbn(isbn_input):
        print("✓ VALID ISBN\n")
    else:
```

```
        print("✗ INVALID ISBN")
        print("   Expected format: ISBN 999-9-99-999999-9")
        print("   Example: ISBN 123-3-12-123456-1\n")
```

Exercise 12-08

Write a program that validates user data entry. It will validate ISSNs (International Standard Serial Numbers), which are used to identify periodical publications, like magazines. The program must accept ISSNs in the following format: ISSN 9999-9999, where each 9 represents a digit. Require the dash at the end, verifying the correct number of digits. The word ISSN is optional and must be accepted regardless of the case. If the word ISSN is specified, the space between it and the number is required.

Valid values : ISSN 1234-4567 1234-4567

Invalid values: ISSN 123-4567 IssN11112-22

```
import re


def validate_issn(issn):
    # Pattern explanation:
    # ^ - start of string
    # (?:ISSN\s+)? - optional "ISSN" followed by one or more spaces
    # \d{4}-\d{4} - exactly 4 digits, dash, exactly 4 digits
    # $ - end of string
    pattern = r"^(?:ISSN\s+)?\d{4}-\d{4}$"

    # Check if the input matches the pattern (case-insensitive)
    return re.match(pattern, issn, re.IGNORECASE)


print("ISSN Validator")
print("=" * 50)
print("Enter ISSN numbers to validate.")
print("Valid formats: 'ISSN 1234-5678' or '1234-5678'")
print("Press Enter without input to exit.")
print()

while True:

    issn_input = input("Enter ISSN: ").strip()

    # Exit condition
    if not issn_input:
        print("Goodbye!")
```

```
        break

    # Validate the ISSN
    if validate_issn(issn_input):
        print("✓ VALID ISSN")
    else:
        print("✗ INVALID ISSN")

    print()
```

Exercise 12-09

Write a program that validates user data entry. Try to find a valid ISBN or ISSN, as defined in previous exercises. Display a message stating whether the number is a valid ISBN or ISSN (identify which one in your response).

```
# Functions number, sequence and check pattern from listing 12.5
# Function cpf from exercise 12.07
# Function cnpj from exercise 12.08
import re

CPF_RE = r"^\d{3}\.\d{3}\.\d{3}-\d{2}$"
CNPJ_RE = r"^\d{2}\.\d{3}\.\d{3}/\d{4}-\d{2}$"


def cpf(input):
    return bool(re.match(CPF_RE, input))


# Checks if input is a valid CNPJ, that is:
# A sequence in the format 99.999.999/9999-99
def cnpj(input):
    return bool(re.match(CNPJ_RE, input))


inputs = [
    "12.345.678/9012-34",  # Valid CNPJ
    "12.345.678-9012-34",  # Invalid
    "99.999.999/9999-99",  # Valid CNPJ
    "123.456.789-01",  # Valid CPF
    "23.456.789-01",  # Invalid
    "999.456.789-01",  # Valid CPF
]

for input in inputs:
    print(
        f"{input}:\nis it a CNPJ? {'Yes' if cnpj(input) else 'No'}\nis it
a CPF? {'Yes' if cpf(input) else 'No'}\n"
    )
```

Exercise 12-10

Write a function that accepts prices in dollars. The program must ignore white spaces and accept values prefixed with $ or not. The user must enter correctly formatted values with a comma separating the thousands and a period separating the cents. If the user types cents, they must have two digits.

Valid values: $500 $500 $500.10 $7,312.10

Invalid values: $500\.1

500\.1$ 500.1 7312.10$

The function must return the entered value converted to float or generate a ValueError exception if the value entered is invalid.

```
import re

DOLLARS_RE = r"^(([rR]\$)?(((\d{1,3}\.)?(\d{3}\.)*?\d{3})|(\d{1,3}))(,
\d{2})?)$"
# ^ - From the start of the string
# ( - Group of the right part of the expression, before the comma
#  ([rR]\$)? - Optionally can have $
#  ( - Group of the integer part
#   ((\d{1,3}\.)?(\d{3}\.)*?\d{3}) - Group of the integer part with dots
separating thousands
#   Required to ensure that parts between dots have 3 digits, except the
first one
#   | - or
#   (\d{1,3}) - Group of the integer part without dots
#  ) - Group of the right part
# (,\d{2})? - Optionally can have the decimal part
# )$ - Until the end of the string


def clean_spaces(input):
    return input.replace(" ", "")


def dollars(input):
    # It's easier to clean spaces before validation,
    # since the regular expression is already quite complex
    # Remember: you don't need to use regex for everything!
    input = clean_spaces(input)
```

```
    return bool(re.match(DOLLARS_RE, input))


inputs = [
    "$ 1.234,56",  # Yes
    "$ 1.234,56",  # Yes
    "$1.234,56",  # Yes
    "$12.123.234,56",  # Yes
    "1.234,56",  # Yes
    "$1.234,56",  # Yes
    "$234,56",  # Yes
    "$234,56",  # Yes
    "234,56",  # Yes
    "$234",  # Yes
    "$234",  # Yes
    "$2",  # Yes
    "$23",  # Yes
    "234",  # Yes
    "34",  # Yes
    "4",  # Yes
    "$234,4",  # No - Cents must have 2 digits
    "$234,4",  # No - Cents must have 2 digits
    "$1234,4",  # No - Missing . to separate thousands
    "$1234,4",  # No - Missing . to separate thousands
    "$1234.12,4",  # No - Incorrect use of thousands separator (.)
    "$1.24,56",  # No - Irregular, only two numbers after the .
]


for input in inputs:
    print(f"{input}: {dollars(input)}")
```

Chapter 13

Exercise 13-01

Modify Program 13.7 to save and load the drawing in JSON format. You can traverse the objects on the canvas and save the chosen shape, coordinates, and colors.

```
import tkinter as tk
import tkinter.ttk as ttk
import json
from tkinter.colorchooser import askcolor
from tkinter.filedialog import asksaveasfilename
from tkinter.filedialog import askopenfilename


class App(tk.Tk):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.background_color = ""
        self.foreground_color = "black"
        self.frame = tk.Frame(self)
        self.create_toolbar()
        self.create_drawing_area()
        self.title("Drawing")
        self.geometry("1200x700")
        self.crosshair = []
        self.crosshair.append(self.canvas.create_line((0, 0, 0, 0),
dash=[2, 4]))
        self.crosshair.append(self.canvas.create_line((0, 0, 0, 0),
dash=[2, 4]))
        self.state = 0
        self.xi = None
        self.yi = None
        self.curr_id = 0
        self.frame.pack(expand=True, fill=tk.BOTH)
        self.tool = self.canvas.create_line

    def create_drawing_area(self):
        self.work_area = tk.Frame(self.frame, height=600)
        self.work_area.grid(column=1, row=0, sticky=tk.NSEW)
        self.frame.grid_columnconfigure(1, weight=1)
        self.frame.grid_rowconfigure(0, weight=1)
```

```
        self.canvas = tk.Canvas(self.work_area, background="white")
        self.canvas.pack(fill=tk.BOTH, expand=True)
        self.canvas.bind("<Motion>", self.mouse_move)
        self.canvas.bind("<Button-1>", self.mouse_click)
        self.canvas.bind("<ButtonRelease-1>", self.mouse_release)
        self.coordinates = tk.Label(self.work_area, text="Move the mouse")
        self.coordinates.pack(ipadx=10, ipady=10)

    def create_toolbar(self):
        self.toolbar = tk.Frame(self.frame, width=100, height=600)
        self.line_button = ttk.Button(
            self.toolbar, text="Line", padding="10", command=self.line_
tool
        )
        self.line_button.pack()
        self.oval_button = ttk.Button(
            self.toolbar, text="Circle", padding="10", command=self.oval_
tool
        )
        self.oval_button.pack()
        self.rectangle_button = ttk.Button(
            self.toolbar,
            text="Rectangle",
            padding="10",
            command=self.rectangle_tool,
        )
        self.rectangle_button.pack()
        undo_button = ttk.Button(
            self.toolbar, text="Undo", padding="10", command=self.undo
        )
        undo_button.pack()
        clear_button = ttk.Button(
            self.toolbar, text="Clear", padding="10", command=self.clear
        )
        clear_button.pack()
        self.foreground_label = ttk.Label(self.toolbar, text="Foreground
Color")
        self.foreground_label.pack()
        self.foreground_button = tk.Button(
            self.toolbar,
            text="Color",
            command=self.foreground_color,
```

```
            bg=self.foreground_color,
        )
        self.foreground_button.pack(fill="x")
        self.background_label = ttk.Label(self.toolbar, text="Background
Color")
        self.background_label.pack()
        self.background_button = tk.Button(
            self.toolbar, text="Transparent", command=self.background_
color, bg=None
        )
        self.background_button.pack(fill="x")
        self.save_button = ttk.Button(
            self.toolbar, text="Save", padding="10", command=self.save
        )
        self.save_button.pack(fill="x")
        self.load_button = ttk.Button(
            self.toolbar, text="Load", padding="10", command=self.load
        )
        self.load_button.pack(fill="x")
        self.toolbar.grid(column=0, row=0, sticky=tk.NS)

    def undo(self):
        if items := self.canvas.find_withtag("drawing"):
            self.canvas.delete(items[-1])

    def clear(self):
        self.canvas.delete("drawing")

    def save(self):
        filename = asksaveasfilename(
            defaultextension=".json", filetypes=[("JSON", ".json")]
        )
        if not filename:
            return  # User cancelled
        drawing = {}  # Create a dictionary with the drawn objects
        for item in self.canvas.find_withtag("drawing"):
            drawing[item] = {
                "type": (type := self.canvas.type(item)),
                "coordinates": self.canvas.coords(item),
                "fill": self.canvas.itemcget(item, "fill"),
            }
            if type in ["rectangle", "oval"]:
```

```
                outline = self.canvas.itemcget(item, "outline")
                drawing[item]["outline"] = outline
        with open(filename, "w") as f:
            json.dump(drawing, f)

    def load(self):
        filename = askopenfilename(filetypes=[("JSON", ".json")])
        if not filename:
            return  # User cancelled
        with open(filename, "r") as f:
            drawing = json.load(f)
        self.clear()
        for data in drawing.values():
            match data["type"]:
                case "line":
                    self.canvas.create_line(
                        data["coordinates"], fill=data["fill"],
tags=["drawing"]
                    )
                case "rectangle":
                    self.canvas.create_rectangle(
                        data["coordinates"],
                        fill=data["fill"],
                        outline=data["outline"],
                        tags=["drawing"],
                    )
                case "oval":
                    self.canvas.create_oval(
                        data["coordinates"],
                        fill=data["fill"],
                        outline=data["outline"],
                        tags=["drawing"],
                    )

    def background_color(self):
        color = askcolor(title="Background color")
        self.background_color = color[1] or ""
        self.background_button.config(
            text="Transparent" if self.background_color == "" else "",
            background=self.background_color or "SystemButtonFace",
        )
```

```
    def foreground_color(self):
        color = askcolor(title="Foreground color")
        if color[1]:
            self.foreground_color = color[1]
            self.foreground_button.config(background=self.foreground_color)

    def line_tool(self):
        self.tool = self.canvas.create_line

    def oval_tool(self):
        self.tool = self.canvas.create_oval

    def rectangle_tool(self):
        self.tool = self.canvas.create_rectangle

    def mouse_move(self, event):
        self.coordinates["text"] = f"Mouse x={event.x} y={event.y}"
        self.canvas.coords(
            self.crosshair[0], event.x, 0, event.x, self.canvas.winfo_
height()
        )
        self.canvas.coords(
            self.crosshair[1], 0, event.y, self.canvas.winfo_width(),
event.y
        )
        if self.state == 1:
            self.canvas.coords(self.curr_id, self.xi, self.yi, event.x,
event.y)

    def mouse_click(self, event):
        if self.state == 0:
            self.xi = event.x
            self.yi = event.y
            self.curr_id = self.tool(
                (self.xi, self.yi, event.x, event.y),
                fill=self.foreground_color,
                tags=["drawing"],
            )
            type = self.canvas.type(self.curr_id)
            if type in ["rectangle", "oval"]:
                self.canvas.itemconfig(
                    self.curr_id,
```

```
                    {
                        "outline": self.foreground_color,
                        "fill": self.background_color,
                    },
                )
            self.state = 1

    def mouse_release(self, event):
        if self.state == 1:
            self.state = 0


App().mainloop()
```

Exercise 13-02

Modify the previous program and save the image in SVG format. The SVG format is a text file that follows a well-defined format.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg width="190mm" height="63mm" viewBox="0 0 190 63" version="1.1"
id="svg1"
    xmlns="http://www.w3.org/2000/svg" xmlns:svg="http://www.w3.org/2000/
svg">
    <g id="layer1">
        <rect style="fill:#cccccc;stroke:#000000;stroke-width:0.264583"
id="rect1" width="42.451897" height="29.063223" x="14.858164"
y="16.001099" />
        <ellipse style="fill:#cccccc;stroke:#000000;stroke-width:0.264583"
id="path1" cx="93.067612" cy="18.940079" rx="15.837823" ry="12.082462" />
        <path style="fill:none;stroke:#000000;stroke-width:0.264583px;
stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1" d="m
172.90984,16.654206 -54.69763,37.5536" id="path2" />
    </g>
</svg>
```

This draws the image shown in Figure 13.14.

Tip: mm — millimeters. We have the rectangle in rect.

The coordinates are inside the style attribute. fill is the internal filling color. strike the outline color. We have our oval in the ellipse element. cx, cy are the coordinates of the center of the ellipse. rx and ry are the horizontal and vertical distance from the center (radius).

The line is in the path element, property d. In d, we have a mini language in which m means to move, followed by the coordinates x1, y1 (without spaces) of the first point and the displacement (x2, y2) after that as the second coordinate. The line is then drawn from (x1, y1) to (x1 + x2, y1 + y2). Try altering the file in a text editor and changing the values before writing the program.

You can view an SVG file in any web browser, so don't forget to reload the page whenever you save your changes. Once you have a clear understanding of how to manipulate these values, write the Python program. You can search for the specification in SVG format on the internet, starting with the Wikipedia website:

https://en.wikipedia.org/wiki/SVG and the Mozilla Foundation: https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial.

```
import tkinter as tk
import tkinter.ttk as ttk
import json
from tkinter.colorchooser import askcolor
from tkinter.filedialog import asksaveasfilename
from tkinter.filedialog import askopenfilename


class App(tk.Tk):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.background_color = ""
        self.foreground_color = "black"
        self.frame = tk.Frame(self)
        self.create_toolbar()
        self.create_drawing_area()
        self.title("Drawing")
        self.geometry("1200x700")
        self.crosshair = []
        self.crosshair.append(self.canvas.create_line((0, 0, 0, 0),
dash=[2, 4]))
        self.crosshair.append(self.canvas.create_line((0, 0, 0, 0),
dash=[2, 4]))
        self.state = 0
        self.xi = None
        self.yi = None
        self.curr_id = 0
        self.frame.pack(expand=True, fill=tk.BOTH)
        self.tool = self.canvas.create_line

    def create_drawing_area(self):
        self.work_area = tk.Frame(self.frame, height=600)
        self.work_area.grid(column=1, row=0, sticky=tk.NSEW)
        self.frame.grid_columnconfigure(1, weight=1)
        self.frame.grid_rowconfigure(0, weight=1)
        self.canvas = tk.Canvas(self.work_area, background="white")
        self.canvas.pack(fill=tk.BOTH, expand=True)
        self.canvas.bind("<Motion>", self.mouse_move)
        self.canvas.bind("<Button-1>", self.mouse_click)
```

```
        self.canvas.bind("<ButtonRelease-1>", self.mouse_release)
        self.coordinates = tk.Label(self.work_area, text="Move the mouse")
        self.coordinates.pack(ipadx=10, ipady=10)

    def create_toolbar(self):
        self.toolbar = tk.Frame(self.frame, width=100, height=600)
        self.line_button = ttk.Button(
            self.toolbar, text="Line", padding="10", command=self.line_
tool
        )
        self.line_button.pack()
        self.oval_button = ttk.Button(
            self.toolbar, text="Circle", padding="10", command=self.oval_
tool
        )
        self.oval_button.pack()
        self.rectangle_button = ttk.Button(
            self.toolbar,
            text="Rectangle",
            padding="10",
            command=self.rectangle_tool,
        )
        self.rectangle_button.pack()
        undo_button = ttk.Button(
            self.toolbar, text="Undo", padding="10", command=self.undo
        )
        undo_button.pack()
        clear_button = ttk.Button(
            self.toolbar, text="Clear", padding="10", command=self.clear
        )
        clear_button.pack()
        self.foreground_label = ttk.Label(self.toolbar, text="Foreground
Color")
        self.foreground_label.pack()
        self.foreground_button = tk.Button(
            self.toolbar,
            text="Color",
            command=self.foreground_color,
            bg=self.foreground_color,
        )
        self.foreground_button.pack(fill="x")
        self.background_label = ttk.Label(self.toolbar, text="Background
```

```
Color")
        self.background_label.pack()
        self.background_button = tk.Button(
            self.toolbar, text="Transparent", command=self.background_
color, bg=None
        )
        self.background_button.pack(fill="x")
        self.save_button = ttk.Button(
            self.toolbar, text="Save", padding="10", command=self.save
        )
        self.save_button.pack(fill="x")
        self.load_button = ttk.Button(
            self.toolbar, text="Load", padding="10", command=self.load
        )
        self.load_button.pack(fill="x")
        self.save_svg_button = ttk.Button(
            self.toolbar, text="Save SVG", padding="10", command=self.
save_svg
        )
        self.save_svg_button.pack(fill="x")
        self.toolbar.grid(column=0, row=0, sticky=tk.NS)

    def undo(self):
        if items := self.canvas.find_withtag("drawing"):
            self.canvas.delete(items[-1])

    def clear(self):
        self.canvas.delete("drawing")

    def create_drawing_dict(self):
        drawing = {}  # Creates a dictionary with the drawn objects
        for item in self.canvas.find_withtag("drawing"):
            drawing[item] = {
                "type": (type := self.canvas.type(item)),
                "coordinates": self.canvas.coords(item),
                "fill": self.canvas.itemcget(item, "fill"),
            }
            if type in ["rectangle", "oval"]:
                outline = self.canvas.itemcget(item, "outline")
                drawing[item]["outline"] = outline
        return drawing
```

```
    def save(self):
        name = asksaveasfilename(
            defaultextension=".json", filetypes=[("JSON", ".json")]
        )
        if not name:
            return  # User cancelled

        with open(name, "w") as f:
            json.dump(self.create_drawing_dict(), f)

    def save_svg(self):
        name = asksaveasfilename(defaultextension=".svg", filetypes=[("SVG",
".svg")])
        if not name:
            return
        size = self.canvas.winfo_width(), self.canvas.winfo_height()
        base = f"""<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg width="{size[0]}mm" height="{size[1]}mm" viewBox="0 0 {size[0]}
{size[1]}" version="1.1" id="svg1" xmlns="http://www.w3.org/2000/svg"
xmlns:svg="http://www.w3.org/2000/svg">
    <g id="layer1">"""
        for i, item in enumerate(self.create_drawing_dict().values()):
            x, y, xf, yf = item["coordinates"]
            width, height = xf - x, yf - y
            match item["type"]:
                case "line":
                    base += f"""<path style="fill:none;stroke:
{item["fill"]};stroke-width:0.264583px;strokelinecap:butt;stroke-linejoin:
miter;stroke-opacity:1" d="m {x},{y} {width},{height}" id="line{i}" />"""
                case "rectangle":
                    base += f"""<rect style="fill:{item["fill"] or "none"};
stroke:{item["outline"] or "none"};stroke-width:0.264583" id="rect{i}"
width="{width}" height="{height}" x="{item["coordinates"][0]}"
y="{item["coordinates"][1]}" />"""
                case "oval":
                    base += f"""<ellipse style="fill:{item["fill"] or
"none"};stroke:{item["outline"] or "none"};stroke-width:0.264583"
id="oval{i}" cx="{x + width // 2}" cy="{y + height // 2}" rx="{width//2}"
ry="{height//2}" />"""

        base += """</g></svg>"""
        with open(name, "w") as f:
```

```
            f.write(base)

    def load(self):
        name = askopenfilename(filetypes=[("JSON", ".json")])
        if not name:
            return  # User cancelled
        with open(name, "r") as f:
            drawing = json.load(f)
        self.clear()
        for data in drawing.values():
            match data["type"]:
                case "line":
                    self.canvas.create_line(
                        data["coordinates"], fill=data["fill"],
tags=["drawing"]
                    )
                case "rectangle":
                    self.canvas.create_rectangle(
                        data["coordinates"],
                        fill=data["fill"],
                        outline=data["outline"],
                        tags=["drawing"],
                    )
                case "oval":
                    self.canvas.create_oval(
                        data["coordinates"],
                        fill=data["fill"],
                        outline=data["outline"],
                        tags=["drawing"],
                    )

    def background_color(self):
        color = askcolor(title="Background Color")
        self.background_color = color[1] or ""
        self.background_button.config(
            text="Transparent" if self.background_color == "" else "",
            background=self.background_color or "SystemButtonFace",
        )

    def foreground_color(self):
        color = askcolor(title="Foreground Color")
        if color[1]:
```

```
            self.foreground_color = color[1]
            self.foreground_button.config(background=self.foreground_color)

    def line_tool(self):
        self.tool = self.canvas.create_line

    def oval_tool(self):
        self.tool = self.canvas.create_oval

    def rectangle_tool(self):
        self.tool = self.canvas.create_rectangle

    def mouse_move(self, event):
        self.coordenadas["text"] = f"Mouse x={event.x} y ={event.y}"
        self.canvas.coords(
            self.cruz[0], event.x, 0, event.x, self.canvas.winfo_height()
        )
        self.canvas.coords(self.cruz[1], 0, event.y, self.canvas.winfo_
width(), event.y)
        if self.estado == 1:
            self.canvas.coords(self.curr_id, self.xi, self.yi, event.x,
event.y)

    def mouse_click(self, event):
        if self.estado == 0:
            self.xi = event.x
            self.yi = event.y
            self.curr_id = self.ferramenta(
                (self.xi, self.yi, event.x, event.y),
                fill=self.cor_de_frente,
                tags=["desenho"],
            )
            tipo = self.canvas.type(self.curr_id)
            if tipo in ["rectangle", "oval"]:
                self.canvas.itemconfig(
                    self.curr_id,
                    {
                        "outline": self.cor_de_frente,
                        "fill": self.cor_de_fundo,
                    },
                )
            self.estado = 1
```

```
    def mouse_release(self, event):
        if self.estado == 1:
            self.estado = 0


App().mainloop()
```

Exercise 13-03

Modify the Window.ok code (Program 13.12) to validate dates when editing or adding new links. Display an error message if the date is invalid.

```
from datetime import date
import tkinter as tk
import tkinter.ttk as ttk
from tkinter.messagebox import showerror
from site_register import Site
from data import Data


class Window(tk.Toplevel):
    MIN_X = 300
    MIN_Y = 300
    PADXY = 10

    def __init__(self, parent, site, on_change=None):
        super().__init__(parent)
        self.geometry(f"{self.MIN_X}x{self.MIN_Y}")
        self.title("Site")
        self.padding = {"padx": self.PADXY, "pady": self.PADXY}
        self.on_change = on_change
        self.create_controls()
        self.minsize(self.MIN_X, self.MIN_Y)
        self.capture_site(site)

    def capture_site(self, site):
        self.site = site or Site()
        self.url.set(self.site.url or "")
        self.date.set(self.site.data)
        self.category.set(self.site.categoria or "")
        self.t_notes.delete("1.0", tk.END)
        self.t_notes.insert("1.0", self.site.notas or "")

    def create_controls(self):
        self.f_url = ttk.Frame(self)
        self.f_url.grid(row=0, column=0, columnspan=3, sticky=tk.EW,
**self.padding)
        self.l_url = ttk.Label(self.f_url, text="URL")
        self.l_url.pack(anchor=tk.W)
```

```
        self.url = tk.StringVar()
        self.e_url = ttk.Entry(self.f_url, textvariable=self.url)
        self.e_url.pack(fill=tk.X, expand=True)
        self.f_category = ttk.Frame(self)
        self.f_category.grid(row=1, column=0, sticky=tk.W, **self.padding)
        self.l_category = ttk.Label(self.f_category, text="Category")
        self.l_category.pack(anchor=tk.W)
        self.category = tk.StringVar()
        self.e_category = ttk.Entry(self.f_category, textvariable=self.
category)
        self.e_category.pack()
        self.f_date = ttk.Frame(self)
        self.f_date.grid(row=1, column=2, sticky=tk.E, **self.padding)
        self.l_date = ttk.Label(self.f_date, text="Date")
        self.l_date.pack(anchor=tk.W)
        self.date = Data(self.f_date)
        self.date.pack()
        self.f_notes = ttk.Frame(self)
        self.f_notes.grid(row=2, column=0, columnspan=3, sticky=tk.NSEW,
**self.padding)
        self.l_notes = ttk.Label(self.f_notes, text="Notes")
        self.l_notes.pack(anchor=tk.W)
        self.t_notes = tk.Text(self.f_notes, height=3)
        self.t_notes.pack(expand=True, fill=tk.BOTH)
        self.grid_columnconfigure(0, weight=2)
        self.grid_columnconfigure(2, weight=1)
        self.grid_rowconfigure(2, weight=1)
        self.b_frame = ttk.Frame(self)
        self.b_frame.grid(row=3, column=0, columnspan=3, **self.padding)
        self.b_ok = ttk.Button(self.b_frame, text="Ok", command=self.ok)
        self.b_ok.pack(side=tk.LEFT)
        self.b_cancel = ttk.Button(self.b_frame, text="Cancel",
command=self.close)
        self.b_cancel.pack(side=tk.LEFT)

    def close(self):
        self.destroy()

    def ok(self):
        try:
            self.validate_date()
        except ValueError:
```

```
            showerror("Error", "Invalid date")
            return
        self.site.url = self.url.get()
        self.site.categoria = self.category.get()
        self.site.data = self.date.get()
        self.site.notas = self.t_notes.get("1.0", tk.END)
        if self.on_change:
            self.on_change(self.site)
        self.close()

    def validate_date(self):
        day, month, year = self.date.get().split("-")
        valid_date = date(int(year), int(month), int(day))
        return valid_date
```

Exercise 13-04

Modify the Window.ok code (Program 13.12) not to accept blank URLs. Display an error message if the URL is invalid (blank).

```
from datetime import date
import tkinter as tk
import tkinter.ttk as ttk
from tkinter.messagebox import showerror
from site_register import Site
from data import Data


class Window(tk.Toplevel):
    MIN_X = 300
    MIN_Y = 300
    PADXY = 10

    def __init__(self, parent, site, on_change=None):
        super().__init__(parent)
        self.geometry(f"{self.MIN_X}x{self.MIN_Y}")
        self.title("Site")
        self.padding = {"padx": self.PADXY, "pady": self.PADXY}
        self.on_change = on_change
        self.create_controls()
        self.minsize(self.MIN_X, self.MIN_Y)
        self.capture_site(site)

    def capture_site(self, site):
        self.site = site or Site()
        self.url.set(self.site.url or "")
        self.date.set(self.site.data)
        self.category.set(self.site.categoria or "")
        self.t_notes.delete("1.0", tk.END)
        self.t_notes.insert("1.0", self.site.notas or "")

    def create_controls(self):
        self.f_url = ttk.Frame(self)
        self.f_url.grid(row=0, column=0, columnspan=3, sticky=tk.EW,
**self.padding)
        self.l_url = ttk.Label(self.f_url, text="URL")
        self.l_url.pack(anchor=tk.W)
```

```
        self.url = tk.StringVar()
        self.e_url = ttk.Entry(self.f_url, textvariable=self.url)
        self.e_url.pack(fill=tk.X, expand=True)
        self.f_category = ttk.Frame(self)
        self.f_category.grid(row=1, column=0, sticky=tk.W, **self.padding)
        self.l_category = ttk.Label(self.f_category, text="Category")
        self.l_category.pack(anchor=tk.W)
        self.category = tk.StringVar()
        self.e_category = ttk.Entry(self.f_category, textvariable=self.
category)
        self.e_category.pack()
        self.f_date = ttk.Frame(self)
        self.f_date.grid(row=1, column=2, sticky=tk.E, **self.padding)
        self.l_date = ttk.Label(self.f_date, text="Date")
        self.l_date.pack(anchor=tk.W)
        self.date = Data(self.f_date)
        self.date.pack()
        self.f_notes = ttk.Frame(self)
        self.f_notes.grid(row=2, column=0, columnspan=3, sticky=tk.NSEW,
**self.padding)
        self.l_notes = ttk.Label(self.f_notes, text="Notes")
        self.l_notes.pack(anchor=tk.W)
        self.t_notes = tk.Text(self.f_notes, height=3)
        self.t_notes.pack(expand=True, fill=tk.BOTH)
        self.grid_columnconfigure(0, weight=2)
        self.grid_columnconfigure(2, weight=1)
        self.grid_rowconfigure(2, weight=1)
        self.b_frame = ttk.Frame(self)
        self.b_frame.grid(row=3, column=0, columnspan=3, **self.padding)
        self.b_ok = ttk.Button(self.b_frame, text="Ok", command=self.ok)
        self.b_ok.pack(side=tk.LEFT)
        self.b_cancel = ttk.Button(self.b_frame, text="Cancel",
command=self.close)
        self.b_cancel.pack(side=tk.LEFT)

    def close(self):
        self.destroy()

    def ok(self):
        try:
            self.validate_url()
        except ValueError:
```

```
            showerror("Error", "Invalid URL")
            return
        try:
            self.validate_date()
        except ValueError:
            showerror("Error", "Invalid date")
            return
        self.site.url = self.url.get()
        self.site.categoria = self.category.get()
        self.site.data = self.date.get()
        self.site.notas = self.t_notes.get("1.0", tk.END)
        if self.on_change:
            self.on_change(self.site)
        self.close()

    def validate_url(self):
        if not self.url.get():
            raise ValueError("URL cannot be empty")

    def validate_date(self):
        day, month, year = self.date.get().split("-")
        date = date(int(year), int(month), int(day))
        return date
```

Exercise 13-05

Modify the Window.ok code (Program 13.12) to accept only URLs starting with http:// or https://.

```
from datetime import date
import tkinter as tk
import tkinter.ttk as ttk
from tkinter.messagebox import showerror
from site_register import Site
from data import Data


class Window(tk.Toplevel):
    MIN_X = 300
    MIN_Y = 300
    PADXY = 10

    def __init__(self, parent, site, on_change=None):
        super().__init__(parent)
        self.geometry(f"{self.MIN_X}x{self.MIN_Y}")
        self.title("Site")
        self.padding = {"padx": self.PADXY, "pady": self.PADXY}
        self.on_change = on_change
        self.create_controls()
        self.minsize(self.MIN_X, self.MIN_Y)
        self.capture_site(site)

    def capture_site(self, site):
        self.site = site or Site()
        self.url.set(self.site.url or "")
        self.date.set(self.site.data)
        self.category.set(self.site.categoria or "")
        self.t_notes.delete("1.0", tk.END)
        self.t_notes.insert("1.0", self.site.notas or "")

    def create_controls(self):
        self.f_url = ttk.Frame(self)
        self.f_url.grid(row=0, column=0, columnspan=3, sticky=tk.EW,
**self.padding)
        self.l_url = ttk.Label(self.f_url, text="URL")
        self.l_url.pack(anchor=tk.W)
```

```
        self.url = tk.StringVar()
        self.e_url = ttk.Entry(self.f_url, textvariable=self.url)
        self.e_url.pack(fill=tk.X, expand=True)
        self.f_category = ttk.Frame(self)
        self.f_category.grid(row=1, column=0, sticky=tk.W, **self.padding)
        self.l_category = ttk.Label(self.f_category, text="Category")
        self.l_category.pack(anchor=tk.W)
        self.category = tk.StringVar()
        self.e_category = ttk.Entry(self.f_category, textvariable=self.
category)
        self.e_category.pack()
        self.f_date = ttk.Frame(self)
        self.f_date.grid(row=1, column=2, sticky=tk.E, **self.padding)
        self.l_date = ttk.Label(self.f_date, text="Date")
        self.l_date.pack(anchor=tk.W)
        self.date = Data(self.f_date)
        self.date.pack()
        self.f_notes = ttk.Frame(self)
        self.f_notes.grid(row=2, column=0, columnspan=3, sticky=tk.NSEW,
**self.padding)
        self.l_notes = ttk.Label(self.f_notes, text="Notes")
        self.l_notes.pack(anchor=tk.W)
        self.t_notes = tk.Text(self.f_notes, height=3)
        self.t_notes.pack(expand=True, fill=tk.BOTH)
        self.grid_columnconfigure(0, weight=2)
        self.grid_columnconfigure(2, weight=1)
        self.grid_rowconfigure(2, weight=1)
        self.b_frame = ttk.Frame(self)
        self.b_frame.grid(row=3, column=0, columnspan=3, **self.padding)
        self.b_ok = ttk.Button(self.b_frame, text="Ok", command=self.ok)
        self.b_ok.pack(side=tk.LEFT)
        self.b_cancel = ttk.Button(self.b_frame, text="Cancel",
command=self.close)
        self.b_cancel.pack(side=tk.LEFT)

    def close(self):
        self.destroy()

    def ok(self):
        try:
            self.validate_url()
        except ValueError as e:
```

```
            showerror("Error", f"Invalid URL\n{e}")
            return
        try:
            self.validate_date()
        except ValueError:
            showerror("Error", "Invalid date")
            return
        self.site.url = self.url.get()
        self.site.categoria = self.category.get()
        self.site.data = self.date.get()
        self.site.notas = self.t_notes.get("1.0", tk.END)
        if self.on_change:
            self.on_change(self.site)
        self.close()

    def validate_url(self):
        url = self.url.get()
        if not url:
            raise ValueError("URL cannot be empty")
        if not (url.startswith("http://") or url.startswith("https://")):
            raise ValueError("URL must start with http:// or https://")

    def validate_date(self):
        day, month, year = self.date.get().split("-")
        valid_date = date(int(year), int(month), int(day))
        return valid_date
```

Exercise 13-06

Modify Program 13.13 to load and save data from a database. Modify the SiteManager class or create another one that allows you to switch the JSON storage for a database like SQLite.

```
import os.path
import sqlite3
import tkinter as tk
import tkinter.ttk as ttk
from tkinter.messagebox import askquestion, showinfo, showerror

from uuid import uuid4
from datetime import date

# All classes are included in the exercise answer
# Only the App class was modified and
# the SitesDBManager class was created


class Date(ttk.Frame):
    def __init__(self, parent, min_year=00, max_year=40):
        super().__init__(parent)
        self.min_year = min_year
        self.max_year = max_year
        self.day = tk.StringVar()
        self.month = tk.StringVar()
        self.year = tk.StringVar()
        self.create_controls()

    def set(self, date):
        day, month, year = date.split("-")
        self.day.set(day)
        self.month.set(month)
        self.year.set(year)

    def get(self):
        return f"{self.day.get()}-{self.month.get()}-{self.year.get()}"

    def create_controls(self):
        self.c_day = ttk.Combobox(
            self,
```

```
            textvariable=self.day,
            width=3,
            values=[f"{d:02d}" for d in range(1, 32)],
            state="readonly",
        )
        self.c_day.pack(side=tk.LEFT)
        self.c_month = ttk.Combobox(
            self,
            textvariable=self.month,
            values=[f"{m:02d}" for m in range(1, 13)],
            width=3,
            state="readonly",
        )
        self.c_month.pack(side=tk.LEFT)
        self.c_year = ttk.Combobox(
            self,
            textvariable=self.year,
            values=[f"{m:02d}" for m in range(self.min_year, self.max_year
+ 1)],
            width=6,
            state="readonly",
        )
        self.c_year.pack(side=tk.LEFT)


class Site:
    def __init__(self, /, url=None, category=None, date=None, id=None,
notes=None):
        if id is None:
            id = str(uuid4())
        self.id = id
        if date is None:
            date = date.today().strftime("%d-%m-%y")
        self.date = date
        self.url = url
        self.category = category
        self.notes = notes

    def __str__(self):
        return f"Site {self.id} {self.url} {self.category} {self.notes}"
```

```
class SitesDBManager:
    def __init__(self):
        self.name = "agenda.db"
        exists = os.path.exists(self.name)
        self.connect()
        if not exists:
            self.create_table()

    def create_table(self):
        self.connection.execute(
            "CREATE TABLE sites (id TEXT PRIMARY KEY, url TEXT, category
TEXT, date TEXT, notes TEXT)"
        )
        self.connection.commit()

    def connect(self):
        self.connection = sqlite3.connect(self.name)

    def disconnect(self):
        self.connection.close()

    def save(self, site):
        self.connection.execute(
            """INSERT INTO sites (id, url, category, date, notes) VALUES
(?, ?, ?, ?, ?)
              ON CONFLICT(id) DO UPDATE SET url=excluded.url,
category=excluded.category, date=excluded.date, notes=excluded.notes""",
            (site.id, site.url, site.category, site.date, site.notes),
        )
        self.connection.commit()
        self.sites[site.id] = site

    def delete(self, id):
        self.connection.execute("DELETE FROM sites WHERE id=?", (id,))
        self.connection.commit()
        if id in self.sites:
            del self.sites[id]

    def load(self):
        self.sites = {}
        q = self.connection.execute("SELECT * FROM sites")
        for site in q.fetchall():
```

```
            new_site = Site(
                id=site[0],
                url=site[1],
                category=site[2],
                date=site[3],
                notes=site[4],
            )
            self.sites[new_site.id] = new_site
        return self.sites


class Window(tk.Toplevel):
    MIN_X = 300
    MIN_Y = 300
    PADXY = 10

    def __init__(self, parent, site, on_change=None):
        super().__init__(parent)
        self.geometry(f"{self.MIN_X}x{self.MIN_Y}")
        self.title("Site")
        self.padding = {"padx": self.PADXY, "pady": self.PADXY}
        self.on_change = on_change
        self.create_controls()
        self.minsize(self.MIN_X, self.MIN_Y)
        self.capture_site(site)

    def capture_site(self, site):
        self.site = site or Site()
        self.url.set(self.site.url or "")
        self.date.set(self.site.date)
        self.category.set(self.site.category or "")
        self.t_notes.delete("1.0", tk.END)
        self.t_notes.insert("1.0", self.site.notes or "")

    def create_controls(self):
        self.f_url = ttk.Frame(self)
        self.f_url.grid(row=0, column=0, columnspan=3, sticky=tk.EW,
**self.padding)
        self.l_url = ttk.Label(self.f_url, text="URL")
        self.l_url.pack(anchor=tk.W)
        self.url = tk.StringVar()
        self.e_url = ttk.Entry(self.f_url, textvariable=self.url)
```

```
        self.e_url.pack(fill=tk.X, expand=True)
        self.f_category = ttk.Frame(self)
        self.f_category.grid(row=1, column=0, sticky=tk.W, **self.padding)
        self.l_category = ttk.Label(self.f_category, text="Category")
        self.l_category.pack(anchor=tk.W)
        self.category = tk.StringVar()
        self.e_category = ttk.Entry(self.f_category, textvariable=self.
category)
        self.e_category.pack()
        self.f_date = ttk.Frame(self)
        self.f_date.grid(row=1, column=2, sticky=tk.E, **self.padding)
        self.l_date = ttk.Label(self.f_date, text="Date")
        self.l_date.pack(anchor=tk.W)
        self.date = Date(self.f_date)
        self.date.pack()
        self.f_notes = ttk.Frame(self)
        self.f_notes.grid(row=2, column=0, columnspan=3, sticky=tk.NSEW,
**self.padding)
        self.l_notes = ttk.Label(self.f_notes, text="Notes")
        self.l_notes.pack(anchor=tk.W)
        self.t_notes = tk.Text(self.f_notes, height=3)
        self.t_notes.pack(expand=True, fill=tk.BOTH)
        self.grid_columnconfigure(0, weight=2)
        self.grid_columnconfigure(2, weight=1)
        self.grid_rowconfigure(2, weight=1)
        self.b_frame = ttk.Frame(self)
        self.b_frame.grid(row=3, column=0, columnspan=3, **self.padding)
        self.b_ok = ttk.Button(self.b_frame, text="Ok", command=self.ok)
        self.b_ok.pack(side=tk.LEFT)
        self.b_cancel = ttk.Button(self.b_frame, text="Cancel",
command=self.close)
        self.b_cancel.pack(side=tk.LEFT)

    def close(self):
        self.destroy()

    def ok(self):
        try:
            self.validate_url()
        except ValueError as e:
            showerror("Error", f"Invalid URL\n{e}")
            return
```

```
        try:
            self.validate_date()
        except ValueError:
            showerror("Error", "Invalid date")
            return
        self.site.url = self.url.get()
        self.site.category = self.category.get()
        self.site.date = self.date.get()
        self.site.notes = self.t_notes.get("1.0", tk.END)
        if self.on_change:
            self.on_change(self.site)
        self.close()

    def validate_url(self):
        url = self.url.get()
        if not url:
            raise ValueError("URL cannot be empty")
        if not (url.startswith("http://") or url.startswith("https://")):
            raise ValueError("URL must start with http:// or https://")

    def validate_date(self):
        day, month, year = self.date.get().split("-")
        valid_date = date(int(year), int(month), int(day))
        return valid_date


class App(tk.Tk):
    MIN_X = 800
    MIN_Y = 200

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.title("Interesting Sites Manager")
        self.geometry(f"{self.MIN_X}x{self.MIN_Y}")
        self.create_controls()
        self.manager = SitesDBManager()
        self.manager.load()
        self.show_data()
        self.minsize(self.MIN_X, self.MIN_Y)

    def create_controls(self):
        self.frame = ttk.Frame(self)
```

```
        self.frame.grid(row=0, column=0, columnspan=2, padx=10, pady=10,
sticky=tk.NSEW)
        self.grid_rowconfigure(0, weight=1)

        # Create table
        self.table = ttk.Treeview(
            self.frame, columns=["url", "category", "date", "notes"],
show="headings"
        )
        self.table.heading("url", text="URL")
        self.table.heading("category", text="Category")
        self.table.column("category", anchor=tk.CENTER)
        self.table.heading("date", text="Date")
        self.table.column("date", anchor=tk.CENTER)
        self.table.heading("notes", text="Notes")
        self.table.grid(row=0, column=0, sticky=tk.NSEW)
        self.table.config(selectmode="browse")

        # Add scrollbar
        scrollbar = ttk.Scrollbar(
            self.frame, orient=tk.VERTICAL, command=self.table.yview
        )
        self.table.configure(yscroll=scrollbar.set)
        self.table.bind("<Double-Button-1>", self.open_window)
        scrollbar.grid(row=0, column=1, sticky=tk.NS)

        # Configure grid weights
        self.frame.grid_columnconfigure(0, weight=1)
        self.frame.grid_rowconfigure(0, weight=1)

        # Create menu
        self.menu = tk.Menu(self)
        self.m_sites = tk.Menu(self.menu, tearoff=0)
        self.m_sites.add_command(label="Add", command=self.add)
        self.m_sites.add_command(label="Delete", command=self.delete)
        self.m_sites.add_separator()
        self.m_sites.add_command(label="Delete All", command=self.delete_
all)
        self.menu.add_cascade(label="Sites", menu=self.m_sites)
        self.menu.add_command(label="About", command=self.about)
        self.config(menu=self.menu)
```

```
    def add(self):
        self.show_site(None)

    def delete(self):
        if selected_id := self.get_selected():
            self.manager.delete(selected_id)
            self.table.delete(selected_id)

    def delete_all(self):
        if (
            askquestion(
                title="Delete All Sites",
                message="Are you sure you want to delete all sites?",
            )
            == "yes"
        ):
            self.clear()

    def clear(self):
        for id in self.manager.sites.keys():
            self.manager.delete(id)
        self.manager.sites.clear()
        self.table.delete(*self.table.get_children())

    def about(self):
        showinfo(
            title="About",
            message="Introduction to Programming with Python.\nhttps://
python.nilo.pro.br",
        )

    def add_site_to_table(self, site):
        self.manager.save(site)
        self.table.insert(
            "",
            tk.END,
            values=(site.url, site.category, site.date, site.notes),
            iid=site.id,
        )

    def show_data(self):
        for site in self.manager.sites.values():
```

```
            self.add_site_to_table(site)

    def get_selected(self):
        if selected_item := self.table.selection():
            return selected_item[0]
        return None

    def open_window(self, event):
        if selected_id := self.get_selected():
            site = self.manager.sites[selected_id]
        else:
            site = None
        self.show_site(site)

    def show_site(self, site):
        self.window = Window(self, site, on_change=self.update)
        self.window.grab_set()

    def update(self, site):
        if self.table.exists(site.id):
            self.table.item(
                site.id,
                text="",
                values=(site.url, site.category, site.date, site.notes),
            )
        else:
            self.add_site_to_table(site)
        self.manager.save(site)


App().mainloop()
```