



BLEKINGE INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

EXERCISES PYTHON

PYTHON PROGRAMMING I
BTH000 ZUT

Authors: Carina Nilsson, Birgitta Hermanson, Siva Krishna Dasari

Contents

Introduction	3
0.1 Purpose	3
0.2 Document structure	3
0.3 To study	4
0.4 Exercises	4
1 Expressions and Interactivity	5
1.1 To Study	5
1.2 Exercises	5
2 Functions	8
2.1 To Study	8
2.2 Exercises	8
3 Conditions	11
3.1 To study	11
3.2 Exercise	11
4 Iteration	19
4.1 To study	19
4.2 Exercises	19
5 Strings and Files	28
5.1 To study	28

5.2	Exercises	28
6	Lists	33
6.1	To study	33
6.2	Exercises	33
7	Tuples	40
7.1	To study	40
7.2	Exercises	40
8	Dictionaries	45
8.1	To study	45
8.2	Exercises	45
9	Classes and Objects	50
9.1	To study	50
9.2	Exercises	50
10	Extra Exercises	53
10.1	To study (<i>Recursion</i>)	53
10.2	Exercises on Recursion	53

Introduction

This course material is produced for the course Python Programming I(BTH000 ZUT).

0.1 Purpose

The purpose of this document is to see to that you, as a student, always will have exercises to work with, in order to improve your skills in problem solving and Python programming. It should be regarded as a complement to the recommended main course literature.

0.2 Document structure

This document is divided into several parts, where each part deals with different elements of the course¹:

- Part 1 – Expressions and Interactivity
- Part 2 – Functions
- Part 3 – Conditions
- Part 4 – Iteration
- Part 5 – Strings and Files
- Part 8 – Dictionaries
- Part 7 – Tuples
- Part 9 – Classes and Objects
- Part 10 – Extra Exercises

If you read this document digitally the orange color shows clickable links within the document and blue color are links to other web sites.

¹The parts depend on each other sequentially


Example	Description
 Canvas	Clickable link to a web site
See figure 3.1	Clickable link taking you to Figure 3.1.

Table 1: Document link examples.

0.3 To study

It is important to read the book before or along with solving the exercise problems.

0.4 Exercises

At first glance it may appear as a huge amount of exercises in here. You do not need to solve all of them, but the more the better. The goal is that there should always be challenging exercises for everyone to do.

The appearance of an exercise looks like this:

1.12 Exercise name

Description, with optional code snippet and/or run example.

</>

Code 1: Code example

</>

1 `def hello():`

2 `print("Hello, World!")`

Run example

`>hello()`
`Hello, World!`

Level of difficulty Exercises with stars ★ can be regarded as more time consuming, the more stars the more time consuming or harder exercise.

Expressions and Interactivity

1.1 To Study

Course Book

Read chapter 1 and 2.

Recommended exercises from the book: 1.1, 2.1, 2.2

1.2 Exercises

Text in ***bold italics*** means user input.

Exercise 1.1 Expressions

Type the following expressions at the prompt in the Python interpreter console. Try to predict what the result is going to be in advance.

a) `4 - 2`

f) `11 / 3`

b) `2 * 5.0`

g) `11 // 3`

c) `(2 - 4) * 5.0`

h) `11 % 3`

d) `4 ** 2`

i) `a = 2 * 3`

e) `10 / 2`

j) `a - 2.0`

Exercise 1.2 More Expressions

Type the following expressions at the prompt in the Python interpreter console. Try to predict what the result is going to be in advance.

- a) `a = 0`
- b) `b = 42`
- c) `a + 1000`
- d) `a - 2`
- e) `b = a`
- f) `b = b + 15`
- g) `a + 17`

Exercise 1.3 Further More Expressions

Write the following expressions at the Python prompt one at a time. Try to predict what the value of the variables `a`, `b` and `c` will be after each row, and what Python replays.

- a) `a = 42`
- b) `a == 37`
- c) `b = (a == a)`
- d) `c = a + 12`
- e) `b == c`

Exercise 1.4 Calculate

Write a program that calculates the following:

- How many seconds are there in 45 minutes and 20 seconds?
- How many miles are 20 kilometers?

Hint: There are 1.61 kilometers in a mile.

Exercise 1.5 Welcome

Write a program that uses `input()` to prompt the user for name and prints a welcome greeting.

```
Welcome
```

```
Enter your name: Siva
Welcome to BTH000 Python Course Siva
```

Exercise 1.6 How old are you?

Write a program that looks like this when running

```
How old are you
```

```
What year were you born in? 1973
```

```
What month? March
```

```
What day? 28
```

```
OK, then you will turn 44 on March 28!
```

★**Challenge:** Use the `time()` function in module `time` to calculate how many years and days a person has lived based on the same input information. A call to `time.time()` returns the GMT time in seconds passed since 1st of January 1970. To use a function in the `time` module you need to import it first. A good approximation is that we have 365.2425 days in a year.

Exercise 1.7 How many hours, minutes and seconds?

Write a program that looks like this when running

```
How many hours, minutes and seconds?
```

```
How many seconds? 4000
```

```
4000 seconds are 1 hours, 6 minutes and 40 seconds
```


Functions

Python provides many built-in modules with many useful functions. One such module is the math module. The math module provides many useful functions such as `sqrt(x)`, `pow(x, y)`, `ceil(x)`, `floor(x)` etc. You will need to do an `"import math"` before you are allowed to use the functions within the math module.

2.1 To Study

Course Book

Read book chapter 3.

Recommended problems from the book: 3.1, 3.2, 4.3

2.2 Exercises

Text in ***bold italics*** means user input.

Exercise 2.1 Plus

Define a function `plus(x, y)` that returns the sum of the two numbers `x` and `y` given as parameters.

Exercise 2.2 Minus

Define a function `minus(x, y)` that returns the difference of the two numbers `x` and `y` given as parameters.

Exercise 2.3 Average

Define a function `average(x, y)` that returns the average of the two numbers `x` and `y` given as parameters.

Exercise 2.4 Fahrenheit and Celcius

Write a function to convert temperature from Celsius to Fahrenheit scale.

°C to °F conversion: Multiply by 9, then divide by 5, then add 32.

Exercise 2.5 Body Mass Index (BMI)

Write a function to compute the BMI of a person.

$BMI = weight(kg) / (height(m) * height(m))$

BMI Calculator Examples

```
>>> BMI(63, 1.70)
'21.8'

>>> BMI(110, 2.00)
'27.5'
```

Exercise 2.6 Percentage

Write a function `percent(value, total)` that takes in two numbers as arguments, and returns the percentage value as an integer.

Percentage Calculation Examples

```
>>> percent(46, 90)
51

>>> percent(51, 51)
100
```

Exercise 2.7 Pythagora's Theoreme

The Pythagoras' Theorem for a right-angle triangle can be written as $a^2 + b^2 = c^2$, where a and b are sides of the right angle and c is the hypotenuse. Write a function to compute the hypotenuse given sides a and b of the triangle.

Hint: You can use `math.sqrt(x)` to compute the square root of x .

Pythagora's Calculation Examples

```
>>> hypotenuse(3, 4)
5

>>> hypotenuse(5, 12)
13
```

Exercise 2.8 Units and tens

Define two functions `unit(int_num)` and `ten(int_num)` that picks up the ones digit and the tens digit in an integer number `int_num`. Running the functions in the Python interpreter console should look like this:

Units and tens

```
>>> unit(123)
3

>>> ten(123)
2
```

Exercise 2.9 ★ Swap units and tens

Define a function `swap_unit_and_ten(int_num)` that swaps the order between the ones digit and the tens digit in an integer number using the two functions from the previous exercise. Running the function in the Python interpreter console should look like this:

Units and tens

```
>>> swap_unit_and_ten(123)
132
```

Conditions

This chapter is about selection. Basically the selection allows you to control the flow of your program, letting it make decisions what code to execute according to current the circumstances. You can make choices based on the value of a variable or user input.

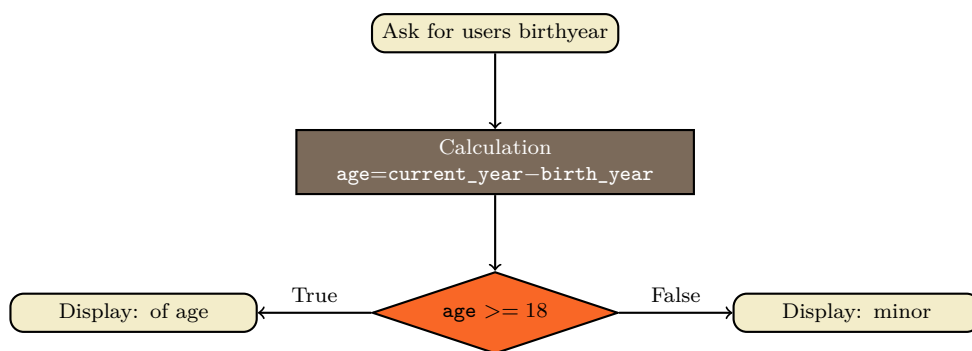



Figure 3.1: Shows a program flowchart.

3.1 To study

 Course literature

Read book chapter 5.1 - 5.7.

3.2 Exercise

Exercise 3.1 Ice Cream Machine

Write a program that lets the user enter an answer to the question whether the user likes ice cream or not. A run of the program could look like this:

Ice Cream Machine

```
Do you like ice cream (yes/no)? yes
Yay! Me too.
```

If the user answers **no** when the program run, it can look like this:

Ice Cream Machine

```
Do you like ice cream (yes/no)? no
Boo! :(
```

Enhance your program with some error-handling, so you get the following output if the user put in another word than **yes** or **no**.

Then the following behavior shall be shown:

Ice Cream Machine

```
Do you like ice cream (yes/no)? maybe
Input error!
```

Exercise 3.2 Odd or Even

Write a program asking the user for a number.

Odd or even - Run 1

```
Write a number? 5
5 is Odd
```

Odd or even - Run 2

```
Write a number? 8
8 is Even
```

Enhance your program:

Step1. Also test if the number is a multiple of 4 and print out a different message.

Step2. Ask the user for two numbers: one number to check (call it **num**) and one number to divide by (**check**). If **check** divides **num** evenly, inform the user by a printed message. If not, print a different message.

Exercise 3.3 Maximum

Define a function `max2(x, y)` that returns the largest of the two numbers `x` and `y` given as parameters. Running the function from the Python interpreter console should look like this:

Maximum

```
>>> max2(8, 4)
8
```

Exercise 3.4 Maximum, version 2

Define a function `max3(x, y, z)` that returns the largest of the three numbers `x`, `y` and `z` given as parameters. You are encouraged to call `max2()` from the previous exercise in your implementation of `max3()`. Running the function from the Python interpreter console should look like this:

Maximum, version 2

```
>>> max3(3, 7, 9)
9

>>> max3(1, 3, 2)
3
```

Exercise 3.5 Checking Isosceles Triangle

Write a function `is_isosceles(x, y, z)` that accepts the 3 sides of a triangle as inputs. The function should return `True` if it is an isosceles triangle. An isosceles triangle has 2 equal sides. An equilateral triangle is a special case of isosceles triangle.

is_isosceles examples

```
>>> is_isosceles(2, 4, 3)
False

>>> is_isosceles(3, 3, 3)
True

>>> is_isosceles(2, 4, 2)
True

>>> is_isosceles(-2, 4, -2)
False

>>> is_isosceles(0, 0, 2)
False
```

Exercise 3.6 Checking Scalene Triangle

Write a function `is_scalene(x, y, z)` that accepts the 3 sides of a triangle as inputs. A scalene triangle is a triangle that has three unequal sides

is_scalene examples

```
>>> is_scalene(2, 4, 3)
True

>>> is_scalene(3, 3, 3)
False

>>> is_scalene(2, 4, 2)
False
```

Exercise 3.7 Fitness Test

Define a function to determine the standard achieved by a participant taking a physical fitness test.

The standard is determined based on the individual and total scores for 3 stations.

Gold	Silver	Pass	Fail
Min. of 4 points for each station, and min. total of 13	Min. of 3 points for each station, and min. total of 10	Min. of 2 points for each station, and min. total of 7	Less than 2 points for any station or total < 7

Fitness Test examples

```
>>> fitness(4,5,4)
'Gold'

>>> fitness(4,4,4)
'Silver'

>>> fitness(3,2,4)
'Pass'

>>> fitness(3,2,1)
'Fail'
```

Exercise 3.8 Who is Older?

Write a program that asks the user for name and birth year. The birth year must not be larger than 2019 and not less than 1900. Then the program shall present both name and age on the terminal screen.

Extend your program further:

1. The program shall ask two users for name and birth year. Then the program shall present their name and age, and also who is older.
2. Let the program also output who is younger.
3. Finally handle the case when the two users are the same age.

Exercise 3.9 Guess the Number I

Write a program that makes the computer generate a random number (below is an example with random number shown), then the user can guess what random number the computer generated.

```
</> Code 2: Random Numbers </>  
1 import random  
2  
3 #random number from the corresponding range(), here in the interval 0..99  
4 random1 = random.randrange(100)  
5  
6 #random number from the interval 1..100  
7 random2 = random.randint(1, 100)
```

To give the user a fair chance to guess the correct number, pick a random number in between 1 and 10. The program shall output information whether the guess is correct or not.

Suggested alternative versions:

- Pick a random number in the interval 1..5 instead.
- Pick a random number in the interval 2..6 instead.
- Let the user input the minimum and maximum limits of the interval to pick a random number from.

Exercise 3.10 Leap Year

According to the Gregorian calendar there are leap years following these rules:

The year is a leap year if it is evenly divisible by 4, but not by 100. If the year is evenly divisible by 400, then it is a leap year. That is, the years 1100, 1300, 1700 are not leap years. On the other hand 400, 800, 1200, 1600, 2000 are leap years.

Hint! Use the modulus operator (%).

The task is to write a program that takes a year (integer number) as a user input and outputs whether the inputted year is a leap year or not.

Exercise 3.11 Dice Roll

Write a simple game allowing two players roll a six-sided dice each. Thereafter the program outputs who won, the name of the player with the higher dice outcome that is.

A run of the program can look like this:

Dice Roll

```
Player 1, enter your name: Christian
Player 2, enter your name: Cuong
Christian, your dice shows 4
Cuong, your dice shows 5
Winner is Cuong!
```

Exercise 3.12 Function with Default Parameters

Write a function that presents a person with name and age and has default parameters.

Running the function from the Python interpreter can look like this:

Default Parameters

```
>>> introduce("Lin", 19)
My name is Lin. I am 19 years old.

>>> introduce("Carina")
My name is Carina. My age is a secret.

>>> introduce()
My name is unknown. My age is a secret.
```

Exercise 3.13 Correctness of if statements

Is there any difference between these two if statements? What makes them different? What values of *n* gives the different prints?

```
</> Code 3: if - version 1 </>
1  if n < 0:
2      print('Negative')
3  elif n < 1000:
4      print('Small')
5  elif n > 1000:
6      print('Large')
7  else:
8      print('Medium')
```

```
</> Code 4: if - version 2 </>
1  if n > 1000:
2      print('Large')
3  elif n < 1000:
4      print('Small')
5  elif n < 0:
6      print('Negative')
7  else:
8      print('Medium')
```

Exercise 3.14 Rewrite if-statement

Rewrite the following if-statement without nesting (if inside if-statements). For what values of *n* are the different messages printed?

```
</> Code 5: if-statement </>
1  if n > 0:
2      if n > 100:
3          print('Big!')
4      else:
5          if n <= 50:
6              print('Medium')
7          else:
8              print('A little larger')
9  else:
10     print('Tiny')
```

Exercise 3.15 ★ Rewrite larger if-statement

Rewrite the following if-statement without nesting. This complicated if-statement can in fact be simplified to only 6 rows of code.



```
</> Code 6: Larger if-statement </>
1  if n > 10:
2      if n > 100:
3          if n < 1000:
4              print(n)
5          else:
6              if n < 1001:
7                  print('Pretty large')
8              else:
9                  print(n)
10     else:
11         if n == 11:
12             print(n)
13         else:
14             if n == 10:
15                 print('10!')
16             else:
17                 print(n)
18 else:
19     if n < 0:
20         print('Below zero')
21     else:
22         print(n)
```

Iteration

4.1 To study

Course Book

Read book chapter 7.

Recommended exercises from the book: 7.1

4.2 Exercises

Text in *italics* means user input.

Exercise 4.1 Output Loops

Write a program that uses **for**-loops to output the following:

- a) All integer numbers between 0 and 25: 0, 1, 2, . . . , 23, 24, 25
- b) All integer numbers between 0 and 25 backwards: 25, 24, 23, . . . , 2, 1, 0
- c) All even numbers between 0 and 30. (**Hint!** Use the %-operator.)
- d) Sum up all integers between 37 and 149 and present the result.
- e) Sum up all integers in an interval that the user specifies by input, and present the result.

Exercise 4.2 What Numbers are Printed?

For each part, study the code and answer the questions!

a) What is printed when the code is run?

```
</> Code 7: Part a) </>  
1 for i in range(0, 100, 3):  
2 print(i)
```

b) What is printed when the code is run?

```
</> Code 8: Part b) </>  
1 i = 0  
2 while i < 100:  
3     print(i)  
4     i *= 3
```

c) What is printed when the code is run?

```
</> Code 9: Part c) </>  
1 k = 0  
2 while 2*k + 1 < 100:  
3     print(2*k + 1)  
4     k += 1
```

d) How many times is the loop executed?

```
</> Code 10: Part d) </>  
1 number = -1  
2 for i in range(number):  
3     print("Hello!")
```

e) How many times is the loop executed?

```
</> Code 11: Part e) </>  
1 number = 4  
2 i = 0  
3 while i < number:  
4     print("Hello!")  
5     i += 1
```

Exercise 4.3 Nagging Terminal

Write a small program that uses a loop to repeat a prompt until the user inputs the letter **x**.

Running the program should look like this:

```
Nagging Terminal

Press x to exit: y
Press x to exit: a
Press x to exit: b
Press x to exit: x
>
```

Exercise 4.4 Interest on Interest

When you put money in a bank account you get interest on the deposited capital, that is the deposited capital increases by a certain percentage each year.

Example:

If you deposit 10 000 RMB at 4% interest you will have $10000 \cdot 1.04 = 10400$ RMB after the first year. After the second year you will have $10000 \cdot 1.04 \cdot 1.04 = 10816$ RMB, after the third year $10000 \cdot 1.04 \cdot 1.04 \cdot 1.04 = 11248.64$ and so on ...

Write a program that lets the user give a start amount, interest percentage and number of years the deposit is going to sit in the bank. Your program is supposed to use a loop to calculate how much money your bank account will hold after these years. Finally let the program present the final amount on the screen.

Exercise 4.5 Guess the Number II

You are supposed to implement a simple game resembling a former task (see Exercise *Guess the Number I*).

The rules are:

1. The program picks a random number between 1 and 100.
2. The player guesses a number between 1 and 100.
3. Depending on the guess the program responds with one of three possible choices: (i) The player guess is too high, (ii) The player guess is too low. (iii) The player guess is correct. The player will be able to commit new guesses and get feedback until the correct answer is received.
4. The program reports how many guesses the user spent, and the program terminates.

Additional task: Let the user have a question before the game ends whether to play again Y(es)/N(o). If the answer is Y, restart the game. To make this possible you will need another outer loop in your program.

Exercise 4.6 Divisibility I

Write a program summing up all positive integer numbers below 2000 that are evenly divisible by 3.

Exercise 4.7 Divisibility II

Write a program summing up all positive integer numbers below 2000 that are evenly divisible by 3 or 7.

Exercise 4.8 Multiplication table

Write a program that performs calculation of the multiplication table for a number inputted by the user. Running the program should look like this:

Multiplication table

```
This program prints out a multiplication table.
Enter a number: 7
0 * 7 = 0
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
4 * 7 = 28
5 * 7 = 35
6 * 7 = 42
7 * 7 = 49
8 * 7 = 56
9 * 7 = 63
```

Exercise 4.9 Determine if a Number is Prime

Define a function `is_prime(number)` that takes in a number as argument and return True if the number is a prime number. A number is a prime number if it is only divisible by 1 and the number itself. By definition, 1 is not a prime number.

is_prime examples

```
>>> is_prime(97)
True

>>> is_prime(1)
False

>>> is_prime(-2)
False
```

Exercise 4.10 Print Range

Write a function `print_range(m, n)` that prints all numbers between `m` and `n` (it is understood that `m < n`). Running the function in the Python interpreter console should look like this:

Print Range

```
>>> print_range(1, 5)
1
2
3
4
5
```

Exercise 4.11 Fix Error

We have written the following program. The idea is to print the sum of all the numbers from 0 to 5. What will the program print to the console? What went wrong, and how can we fix it?

</>

Code 12: Erroneous code

</>

```
1 n = 0
2 for i in range(6):
3     n + i
4 print(n)
```

Exercise 4.12 Sum Range

Write a function `sum_range(n)` that sums up all numbers from 0 up to `n` and returns the sum. Running the function in the Python console should look like this:

Sum Range

```
>>> sum_range(5)
15
```

Exercise 4.13 Erroneous code

What will the following function return? Try implementing it and run it. Compare your prediction with the results. How do you explain the result?

</>

Code 13: Erroneous code

</>

```
1 def sum_first(n):
2     sum = 0
3     for i in range(n):
4         sum = sum + i
5     return sum
6 return sum
```


Exercise 4.14 Sum Odd

Write a function `oddsum_loop(n)` that asks the user for `n` integer numbers. The function then returns the sum of all the odd inputted numbers. You are given the code for a function `isodd(number)` that returns `True` if the number is odd and `False` otherwise.

```
</> Code 14: isodd() </>  
1 def isodd(number):  
2     return number % 2 == 1
```

Running the function `oddsum_loop()` in the Python console should look like this:

```
Sum Odd  
  
>>> oddsum_loop(4)  
Enter a number: 5  
Enter a number: 24  
Enter a number: 123  
Enter a number: -101  
The sum of the odd numbers is: 27
```

Exercise 4.15 Sum Odd in Range

Write a function `sumodd_range(m, n)` that goes through all the numbers from `m`, `m + 1`, ..., `n-1`, `n` (i.e. from and including `m` to and including `n`) and returns the sum of all odd in that range numbers. When it encounters an even number, print it on the screen. A run should look like this:

```
Sum Odd Range  
  
>>> sumodd_range(5, 10)  
6 is even!  
8 is even!  
10 is even!  
21
```

Exercise 4.16 ★ Sum Below

Write a function `sum_below(m, n, cutoff)` which returns the sum of all numbers below `cutoff`. You can assume that `cutoff` is greater than `m`. For tracing the execution, the function shall also print out all the numbers added, to ensure the correctness of the function.

Sum Below, run examples

```
>>> sum_below(5, 10, 7)
Added 5
Added 6
11

>>> sum_below(1, 3, 17)
Added 1
Added 2
Added 3
6
```

Exercise 4.17 ★ Fibonacci

A Fibonacci number is a number from the sequence that can be calculated from the following:

$$F(n) = \begin{cases} 0 & \text{om } n = 0 \\ 1 & \text{om } n = 1 \\ F(n-1) + F(n-2) & \text{otherwise.} \end{cases}$$

A number in the sequence is the sum of the two proceeding numbers except the first two numbers that are predefined 0 and 1. The sequence then becomes: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

- You shall write a program that takes user input n and presents $F(n)$ (the n :th number from the Fibonacci sequence).
- Rewrite the program to output the full sequence $F(0), F(1), F(2), F(3), \dots, F(n-1), F(n)$.

Exercise 4.18 Turtle Exercise

Use the turtle package. Make a program that draws a square using a for loop.

Extra challenge: You can also fill a drawn closed area with a color in turtle. Use the turtle function `fillcolor(color_string)`. The function takes a parameter, the color as a string. Turtle accepts almost every color you can think of. For a full color reference list see [rgb-colors](#). To color fill an area you need to enclose the part of code that generates the pattern to fill by the turtle calls `begin_fill()` and `end_fill()`. The code example shows how to color fill a circle given there is an existing function `circle()` drawing a circle.

</> **Code 15: Color fill circle in turtle** </>

```
1 shellback = turtle.Turtle()    # Create an instance of a turtle
2 shellback.shape('turtle')     # Make it look like one
3 shellback.begin_fill()        # Fill the next shape
4 shellback.fillcolor('aqua')    # Color of filling
5 shellback.circle(100)         # Draw shape
6 shellback.end_fill()          # Finish filling
```

Exercise 4.19 Calculator

This task is to implement a simple calculator dealing with the operations $+$, $-$, \cdot , $/$. That is, it should be possible to add, subtract, multiply or divide two integer numbers inputted by the user. The user interface should be a menu.

If you input an invalid menu choice, for example -1, an error message should appear and the menu should be shown again.

Run example::

Calculator run example

```
1. Enter two integers
2. Add
3. Subtract
4. Multiply
5. Divide
0. Exit
Your choice: -1
Input must be either 0,1,2,3,4 or 5.
```

```
1. Enter two integers
2. Add
3. Subtract
4. Multiply
5. Divide
0. Exit
Your choice: 1
Input first number: 20
Input second number: 34
```

```
1. Enter two integers
2. Add
3. Subtract
4. Multiply
5. Divide
0. Exit
Your choice: 2
The result is 54
```

```
1. Enter two integers
2. Add
3. Subtract
4. Multiply
5. Divide
0. Exit
Your choice: 0
Exiting ... Goodbye!
```

Exercise 4.20 ★★ Estimate π - Leibnitz's Formula

It can be shown that the following infinite summation

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4} \iff \pi = 4 \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

approaches π (slowly though).

The task is to implement a program that calculates the sum using a loop, to calculate estimations of π . As this is an infinite sum with an unlimited number of addends we can not allow the computer to go on forever. Therefor we need to modify the formula (make an approximation):

$$\pi \approx 4 \sum_{n=0}^k \frac{(-1)^n}{2n+1}$$

By increasing the value of k , for example 100, 200, 300, . . . you get more and more accurate approximations. Increase k in your program until you recognize the value of π .

Hint: Note that $(-1)^n = 1$ when n is even and $(-1)^n = -1$ when n is odd, if you are setting a threshold value for the size of the addends to end the loop.

Note: The computer works with floating point numbers that are approximations of decimal numbers. This means that calculations are not exact.

Exercise 4.21 ★★ Estimate Pi - Ramanujan's Formula

Write a function `estimate_pi()` to estimate and return the value of π based on the formula found by an Indian Mathematician, Srinivasa Ramanujan. It should use a while loop to compute the terms of the summation until the last item is smaller than 10^{-15} . The formula is given below:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 \cdot 396^{4k}}$$

Strings and Files

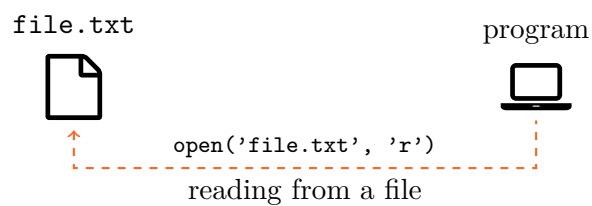


Figure 5.1: Reading from the file `file.txt`

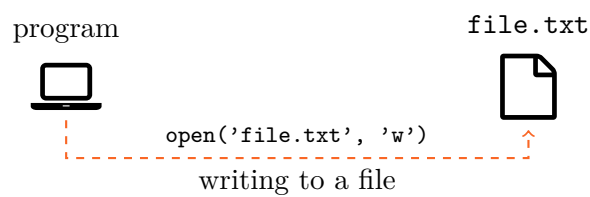


Figure 5.2: Writing to the file `file.txt`

5.1 To study

Course Book

Read book chapter 8 and 14.1 - 14.4.
Recommended exercises from the book: 8.1 - 8.5

5.2 Exercises

Exercise 5.1 Getting used to strings

Try the following lines at the Python console prompt. What do you expect to happen after each row? What values have changed?

```
a) intro = 'Hello'
b) name = 'Charlie'
c) greeting = intro + ' ' + name
d) greeting = greeting + ' at '
e) greeting = greeting + 15
f) print(greeting)
g) new_greeting = print(greeting)
```

Exercise 5.2 Count 'e'

Write the function `count_e(word)` that takes in a word (string) as argument and returns the number of 'e' (lower case letter) in that word.

Count 'e'

```
>>> count_e('elephant')
2

>>> count_e('Egg')
0

>>> count_e('interference')
4
```

Exercise 5.3 Count Letters

Write the function `count_letter(word, letter)` that takes in a word and a letter as arguments and returns the number of occurrences of letter found in that word.

Count Letters

```
>>> count_letter('elephant', 'e')
2

>>> count_letter('lion', 'i')
1

>>> count_letter('Giraffe', 'g')
0
```

Exercise 5.4 ★ Palindrome

Write a program that checks whether an inputted word is a palindrome or not. The program shall ask for a new word repeatedly until the user type `quit`

Palindrome

```
Write a word: Carrot
Carrot is not a palindrome

Write a word: mom
mom is a palindrome

Write a word: racecar
racecar is a palindrome

Write a word: quit
Exiting ...
```

Challenge: See to that uppercase letters does not impact the result in your program. That means `Racecar` as well as `racecar` is a palindrome.

Exercise 5.5 Find Substring

Write the function `search(word, substring)` that takes in a word and a substring as arguments and returns the position (0 indexed) of the substring if it is found in the word. The function returns -1 if the substring is not found.

Find Substring

```
>>>search('elephant', 'ant')
5

>>>search('apple', 'p')
1

>>>search('Giraffe', 'aff')
3
```

Exercise 5.6 Get Character

A string contains a sequence of characters. Elements within a string can be accessed using index that starts from 0. Write the function `get_char(word, pos)` that takes in a word and a number as argument and returns the character at that position.

Get Character

```
>>> word = "elephant"
>>> word[0]
'e'

>>> word[-1]
't'

>>> get_char("apple", 2)
'p'

>>> get_char("giraffe", 0)
'g'

>>> get_char("giraffe", 10)
Invalid Range.
```

Exercise 5.7 Count Vowels

Write a function `count_vowels(word)` that takes in a word as an argument and returns the total number of vowels ('a', 'e', 'i', 'o', 'u') in the word.

Count Vowels

```
>>> count_vowels("elephant")
3

>>> count_vowels("apple")
2
```

Exercise 5.8 Reading a File

Write a program that reads the content from a text file to the computer memory in a string. Then print the string to make sure your program works.

Exercise 5.9 Writing a file

Write a function that writes all multiplication tables up to a number to a text file. The function shall take the file name (a string) and the number as input parameters.

Exercise 5.10 ★ Find and replace

Write a function `find_and_replace(find_str, replace_str, infile, outfile)` that searches for a word or a phrase (string). The phrase is replaced by another phrase given as the second parameter. Parameter 3 and 4 respectively are input and output file names.

Tip: All string methods are described in the documentation of Python Built-in Types available on [Python Standard Library Documentation \(section 4.7.1\)](#).

Lists

6.1 To study

Course Book

Read chapter 10.

Recommended exercises from the book: 10.1 - 10.4

6.2 Exercises

Text in ***bold italics*** means user input.

Exercise 6.1 Add and Remove Elements in a List

Start with the following list declaration: `a_list = ['Hello', 0]`

- a) Now you want to add elements to the list. Use the list `append(x)` function to add some more items to the list to produce the same content as shown in the sample below.

Modify list

```
>>> a_list[0]
'Hello'
>>> a_list[1:3]
[0, 20.0]
>>> a_list[3]
'World'
```

- b) A list can be modified, and elements can be removed from an existing list. Use the list `remove(x)` function to remove some items from a list. Start with the list declaration: `a_list = ['hello', 'I', 'love', 'Python', 'programming']`
After removing elements all that remains in the list should be: `['hello', 'Python', 'programming']`.

Exercise 6.2 Iterate Through List

Previously, we iterated over the ranges. Now we are trying to iterate directly over lists. You are given the following list of years: `disc_years = [1997, 2007, 2009, 2013, 2015, 2016]`

- a) Now you want to print out how many years ago, each of these years occurred. Write a loop that runs through `disc_years`. A run of the loop should look like this:

Iterate through list

```
20
10
8
3
2
1
```

- b) Now you also have a parallel list with film titles from films released the corresponding years.

```
disc_titles = ['Titanic', 'Transformers', 'Avatar', 'The Hobbit', 'Jurassic World', 'Zootopia']
```

Write a program that presents the following output iterating through the given lists.

Iterate through parallel lists

```
Titanic was released in 1997.
Transformers was released in 2007.
Avatar was released in 2009.
The Hobbit was released in 2013.
Jurassic world was released in 2015.
Zootopia was released in 2016.
```

Exercise 6.3 Update Elements in a List

Write a program that performs the following.

Create two lists: `start_list = [5, 3, 1, 2, 4]` and `square_list = []`

- Copy the values of `start_list` into `square_list`
- Replace every element in `square_list` with its squared value (power of two).
- Sort the elements in `square_list`
- Print `square_list` on the screen

Exercise 6.4 Sum Numbers

Write a function `sum_numbers(any_list)` that sums up the elements in a list that are numbers and returns the sum. All list elements that are not numbers should be ignored. This code snippet shows how to check if something is a number:

```
</> Code 16: isnumber() </>
1 import numbers
2
3 def isnumber(x):
4     return isinstance(x, numbers.Number)
```

The return value from `isnumber()` will be `True` if the argument is a number `False` if it is not. Now you can check if something is a number by making a call like `isnumber(5.324)`.

When you make a call to your `sum_numbers()` function in the Python interpreter console it should look like this:

Sum numbers

```
>>> sum_numbers(['a', 1, 'b', 2, [['b', 4], 2], 3])
6
```

Exercise 6.5 Find Letter

Write a function `find_letter(character, letter_list)` that checks if a certain letter is in a word (the word should be represented as a list of letters, and letters are strings in Python). The function should return `True` if the letter is in the list and it should return `False` otherwise. Here follows two run examples from the Python console.

Find Letter

```
>>> find_letter('u', ['h', 'o', 'u', 's', 'e'])
True

>>> find_letter('b', ['c', 'a', 'r'])
False
```

Exercise 6.6 Sum Last Digits

Write a function `sum_last_digits()` that takes in a list of positive numbers and returns the sum of all the last digits from all numbers in the list.

Sum Last Digits

```
>sum_last_digits([1, 2, 10])
3
```

Exercise 6.7 Add First and Last

Define a function called `add_first_and_last(x)` that takes in a list of numbers and returns the sum of the first and last numbers.

Add First and Last

```
>>> add_first_and_last([])
0

>>> add_first_and_last([2, 7, 3])
5

>>> add_first_and_last([10])
10
```

Exercise 6.8 Duplicate List and Distribute Values to Sub Lists

Construct a function `distribute(my_item, my_list)` that takes an element and a list of lists as arguments. The function shall return a new list starting from the list content sent into to the function, where the value `my_item` is inserted as element in each sub list. See example of a run below.

Distribute

```
>>> old_list = [['kangar'], ['z'], ['f']]
>>> distribute('oo', old_list)
[['kangar', 'oo'], ['z', 'oo'], ['f', 'oo']]
>>> old_list
[['kangar'], ['z'], ['f']]
```

Exercise 6.9 Extend Each

Construct a function `extend_each(my_item, my_list)` that takes an element and a list of lists, and extends every sub list with that element. The difference between this function and the previous one is that this time the function returns nothing, but changes the original list. See example of a run below.

Extend Each

```
>>> extend_each('oo', old_list)
>>> old_list
[['kangar', 'oo'], ['z', 'oo'], ['f', 'oo']]
```

Exercise 6.10 Caesar Cipher

The task is to implement a rotation cipher that also is called Caesar Cipher. The Caesar Cipher is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plain text is replaced by a letter some fixed number of positions down the alphabet. This fixed number is called key. For example, with a left shift of 3 (i.e. key=3) all letters are left shifted 3 positions in the alphabet so all occurrences of 'A' becomes 'D' and all occurrences of 'Z' becomes 'C'.

Hint! It can be useful to create a list with all characters in the alphabet in the correct order.

You are supposed to write two functions, one for encryption and one for decryption:

`encrypt(msg, key)`, to return the encrypted text corresponding to the plain text `msg`.

`decrypt(msg, key)`, to return the decrypted plain text corresponding to the encrypted text `msg`.

Your code must fulfill the following demands:

- Only English alphabet characters are encrypted/decrypted.
- Only use upper case letters, that is all alphabet characters are transformed to upper case before encryption/decryption.
- Characters not belonging to the English alphabet (dots, blanks etc.) are not encrypted/decrypted, but preserved unchanged in the text.

Run example:

Caesar Cipher

```
>>> msg = 'We attack at dawn!'
>>> encrypt(msg, 5)
'BJ_FYYFHP_FY_IFBS!'

>>> msg = 'Bruce Wayne is Batman!'
><< code = encrypt(msg, 17)
>>> code
'SILTV_NRPEV_ZJ_SRKDRE!'

>>> decrypt(code, 17)
'BRUCE_WAYNE_IS_BATMAN!'
```

Exercise 6.11 ★ Matrix Dimensions

A 2D matrix with dimensions $m \times n$ can be represented in Python by a nested list.

Example of 3×2 matrix (3 rows with 2 numbers in each row, that is 2 columns):

$$A = \begin{pmatrix} 1 & 3 \\ -5 & 6 \\ 2 & 4 \end{pmatrix}$$

Write a function `matrix_dimensions(matrix)` that returns the dimensions of a 2D matrix sent to the function as a nested list argument. The function shall return an information string (see run example below).

Run example:

Matrix Dimensions

```
>>> A = [ [1, 3], [-5, 6], [2, 4] ]
>>> matrix_dimensions(A)
'This is a 3x2 matrix.'

>>> B = [ [1, 3, 2], [-5, 6, 0] ]
>>> matrix_dimensions(B)
'This is a 2x3 matrix.'

>>> C = [ [1, 3], [-5, 6, 0] ]
>>> matrix_dimensions(C)
'This is not a valid matrix.'
```

Exercise 6.12 ★ Transpose Matrix

The transpose of a matrix A is an operator which flips the matrix over its diagonal, that is it switches the row and column indices of the matrix by producing another matrix denoted as A^T .

Example with 3×2 matrix:

$$A = \begin{pmatrix} 1 & 3 \\ -5 & 6 \\ 2 & 4 \end{pmatrix} \qquad A^T = \begin{pmatrix} 1 & -5 & 2 \\ 3 & 6 & 4 \end{pmatrix}$$

Write a function `transpose(matrix)` that returns the transpose of a 2D matrix as a nested list.

Run example:

Transpose Matrix

```
>>> A = [[1, 2, 3], [4, 5, 6]]
>>> transpose(A)
[[1, 4], [2, 5], [3, 6]]

>>> transpose([[1, 2]])
[[1], [2]]

>>> transpose([[3]])
[[3]]
```

Exercise 6.13 ★ ★ ★ PLAYFAIR-cipher

<https://open.kattis.com/problems/itsasecret>

Exercise 6.14 Turtle Exercise

Use the `turtle` module. Make a list of four strings representing colors. To see what colors you can use, visit [rgb-colors](#). The function for setting the pen color in turtle is `color()`, and takes one argument - the color as a string. Write a program that loops through your color list and draws a square one, side in each color in your list. The turtle line width defaults to 1 pixel. If you think the lines are too thin you can thicken them with the turtle method `pensize(pixels)`

Tuples

7.1 To study

 Course Book

Read book chapter 12

7.2 Exercises

Text in *italics* means user input.

Exercise 7.1 Tuple

Tuples work basically in the same way as a lists, but with two notable exceptions. The first difference is that tuples are in parentheses and lists are in brackets. You will discover the other difference in this exercise. Tuples can, just as lists, contain any data type. Let us type the following in the Python interpreter console, representing a point (x, y) in a coordinate system:

Tuple

```
>>> point = (5, 0)
```

Then try to change only the y-coordinate to 5 in the tuple by accessing only that position. Is it possible?

Exercise 7.2 All Pairs

Write a function `all_pairs(my_list)` that on the basis of a list produces a list of pairs (tuples), that is all the possible ways to combine two elements from the original list. The order matters so the pair (x, y) is considered different from the pair (y, x). This means that if you send in a list of length n , the resulting list has the length n^2 . An example from running the function is demonstrated below.

```
all_pairs()
```

```
>>> all_pairs([1, 2, 3])
[(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)]
```

Exercise 7.3 Unordered Pairs

Write a function `unordered_pairs(my_list)` that on the basis of a list produces a list of unordered pairs. The output of the function will be, just like in the previous exercise, tuples in a list. The difference now is that the order does not matter, so if (1,2) is in the list (2,1) can not be part of it since it is considered the same pair. An example from running the function is demonstrated below.

```
unordered_pairs()
```

```
>>> unordered_pairs([1, 2, 3])
[(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)]
```

Exercise 7.4 ★ Determinant of 2x2 Matrix

The determinant of a 2x2 matrix is the product of the elements on the main diagonal minus the product of the elements off the main diagonal. Write a function `det(matrix)` that calculates such a determinant. The behaviour should be as follows:

```
Determinant 2x2
```

```
>>> M = ((3,1), (5,2))
>>> det(M)
1
```

Exercise 7.5 ★ Same Content

Write a function `same_content(tup1, tup2)` that takes in two tuples as arguments and returns `True` if the two tuples contains the same (or at least identical) elements and `False` otherwise.

Run example:

Same Content

```
>>> same_content((1, 2), (1, 2))
True

>>> same_content((1, 2), (1, 2, 1))
False

>>> same_content((1, 2), ())
False
```

Exercise 7.6 ★ Common Content

Write a function `common_content(tup1, tup2)` that takes in two tuples as arguments and returns a tuple containing all elements common for both argument tuples.

Run example:

Common Content

```
>>> common_content((1, 2, 3), (2, 5, 1))
(1, 2)

>>> common_content((1, 2, 3, 'p', 'n'), (2, 5, 1, 'p'))
(1, 2, 'p')

>>> same_content((1, 3, 'p', 'n'), ('a', 2, 5, 1, 'p'))
(1, 'p')
```

Exercise 7.7 ★ Not Common Content

Write a function `not_common(tup1, tup2)` that takes in two tuples as arguments and returns a tuple containing all elements that are not common for both argument tuples. The resulting tuple is supposed to be sorted.

Run example:

Common Content

```
>>> not_common((1, 2, 3, 4), (3, 4, 5, 6))
(1, 2, 5, 6)

>>> not_common(('b', 'a', 'c', 'd'), ('a', 'b', 'c'))
('d',)

>>> not_common(('a', 'b', 'c'), ('a', 'b', 'c'))
()

>>> not_common(('a', 'b'), ('c', 'd'))
('a', 'b', 'c', 'd')

>>> not_common(('b', 'a', 'd', 'c'), ('a', 'b'))
('c', 'd')
```

Exercise 7.8 Sort by Number

Write a function `sort_by_number(tup_list)` that takes in a list of tuples with two elements each, where the first element is a number. Return a tuple that is a concatenation of all second elements from the list tuples ordered after the number in the same original tuple.

Run example:

Sort by Number

```
>>> sort_by_number([(3, 'BTH000'), (4, 'course'), (1, 'Welcome'), (2, 'to')])
('Welcome', 'to', 'BTH000', 'course')

>>> sort_by_number([(2, 'programming'), (3, 'is'), (1, 'Python'), (4, 'fun')])
('Python', 'programming', 'is', 'fun')

>>> sort_by_number([(2, 'are'), (3, 'Immutable'), (1, 'Tuples')])
('Tuples', 'are', 'Immutable')
```

Exercise 7.9 Sort by Length

Write a function `sort_by_len(tup, direction)` that takes in a tuple of strings. Return a new tuple where the strings are sorted with aspect of their lengths. The sort order is defined by a second argument containing `'asc'` for ascending order or `'des'` for descending order.

Run example:

Sort by Number

```
>>> sort_by_len(('Windows', 'Linux', 'OSX'), 'asc')
('OSX', 'Linux', 'Windows')


>>> sort_by_len(('apple', 'orange', 'pear'), 'des')
('orange', 'apple', 'pear')

>>> sort_by_len(('begin', 'python', 'programming'), 'des')
('programming', 'python', 'begin')

>>> sort_by_len(('begin', 'python', 'programming'), 'asc')
('begin', 'python', 'programming')
```

Dictionaries

8.1 To study

 Course Book

Read book chapter 11

8.2 Exercises

Text in *italics* means user input.

Exercise 8.1 Dictionary

In a dictionary you pair up *data* with a *key*. Then you can access data using the key. You could think of a dictionary as an advanced list, where instead of using a number as the index, using virtually any type of data. Try typing the following in the Python interpreter console:

Dictionary

```
>>> numbers = {'one': 1, 'two': 2, 'seven': 6, 'eight': 9}
>>> numbers['one']
1
```

To get used to handling dictionaries now type commands to do the following:

- add the key 'three' linked with an appropriate number
- make a change in the dictionary `numbers` so that `numbers['eight']` will be linked with the number 8 instead

Exercise 8.2 Dictionary properties

Here we see two suggestions how to create a dictionary. One way will work and the other will not. Try to predict which one works. Then try to type them at the Python prompt to see if you were right.

Dictionary properties

```
>>> coordinates_one = {(1, 0): 'x-axis', (0, 1): 'y-axis'}  
>>> coordinates_two = {[1, 0]: 'x-axis', [0, 1]: 'y-axis'}
```

Also try to figure out why one version works and the other does not.

Exercise 8.3 Show Wordlist

Write a function `show_wordlist(wlist)` that goes through a dictionary and writes the values to the corresponding keys. Running the function on the `numbers` dictionary from [Exercise 8.1](#) should look something like this. Remember dictionaries are not ordered in any particular order.

Maximum

```
>>> show_wordlist(numbers)  
--  
one - 1  
seven - 6  
eight - 9  
two - 2  
--
```

Exercise 8.4 Looping through a Dictionary

When we loop through a list we write code in the following manner:

```
</> Code 17: Looping through all elements of a list </>  
1 for item in my_list:  
2     print(item)
```

Rewrite the code so you can loop through all key-value pairs in the dictionary `numbers` from [Exercise 8.1](#) instead.

Exercise 8.5 Genes

In gene expression, mRNA is transcribed from a DNA template.

The 4 nucleotide bases of A, T, C, G corresponds to the U, A, G, C bases of the mRNA.

Write a function `mRNAtranscript(template)` that returns the mRNA transcript given the sequence of a DNA strand.

Genes

```
>>> mRNAtranscription("ATCGATTG")
'UAGCUAAC '

>>> mRNAtranscription("CTATCGGCACCCTTTCAGC")
'GAUAGCCGUGGGAAAGUCG '
```

Hint! Use a dictionary to provide the mapping of DNA to RNA bases.

Exercise 8.6 More Genes

A DNA strand consisting of the 4 nucleotide bases is usually represented with a string of letters: A,T, C, G. Write a function that computes the base composition of a given DNA sequence. The function shall return a dictionary with the four keys 'A', 'T', 'C' and 'G' and the count of each nucleotide base as values.

Run example:

More Genes

```
>>> base_composition("CTATCGGCACCCTTTCAGCA")
{'A': 4, 'C': 8, 'T': 5, 'G': 3}

>>> base_composition("AGT")
{'A': 1, 'C': 0, 'T': 1, 'G': 1}

>>> base_composition("TGCCTCCGCACAGCAGCCCCGGCCATAGCCACCTCCTAGCCCCTGCCACCA")
{'A': 10, 'C': 27, 'T': 6, 'G': 9}

>>> base_composition("")
{'A': 0, 'C': 0, 'T': 0, 'G': 0}
```


Exercise 8.7 Reverse Lookup

Write a function `reverse_lookup(dictionary, value)` that takes a dictionary and a value as arguments, and returns a sorted list with all keys associated with the given value. The function shall return an empty list if the value is not found in the dictionary

Run example:

Reverse Lookup

```
>>> reverse_lookup('a': 1, 'b': 2, 'c': 2, 1)
['a']

>>> reverse_lookup('a': 1, 'b': 2, 'c': 2, 2)
['b', 'c']

>>> reverse_lookup('a': 1, 'b': 2, 'c': 2, 3)
[]
```

Exercise 8.8 Inverted Dictionary

Write a function `invert_dictionary(dictionary)` that takes a dictionary as an argument, and returns a new dictionary that inverts the keys and the values of the original dictionary.

Run example:

Inverted Dictionary

```
>>> invert_dictionary('a': 1, 'b': 2, 'c': 3, 'd': 2)
{1: ['a'], 2: ['b', 'd'], 3: ['c']}

>>> invert_dictionary('a': 3, 'b': 3, 'c': 3)
{3: ['a', 'c', 'b']}

>>> invert_dictionary('a': 2, 'b': 1, 'c': 2, 'd': 1)
{1: ['b', 'd'], 2: ['a', 'c']}
```

Exercise 8.9 Counting Word Frequency in a File

Write a function that counts how many times each word is found in a file. A word, this is a sequence of characters that is not space, tab, or newline characters. At the end of the execution the function shall print all the words in the file and how many times they were appearing.

The function shall take a file name as a parameter.

Challenge: Present the output printing sorted with the most frequent word at the top.

Exercise 8.10 Sparse Vector

A sparse vector is a vector whose entries are almost all zero, like `[1, 0, 0, 0, 0, 0, 0, 2, 0]`. Storing all those zeros wastes memory and dictionaries are commonly used to keep track of just the nonzero entries. For example, the vector shown earlier can be represented as `{0:1, 7:2}`, since the vector it is meant to represent has the value 1 at index 0 and the value 2 at index 7. Write a function that converts a sparse vector into a dictionary as described above.

Run example:

Sparse Vector

```
>>> list_to_dict([1, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0, 0, 4])
{0: 1, 3: 2, 7: 3, 12: 4}

>>> list_to_dict([1, 0, 1, 0, 2, 0, 1, 0, 0, 1, 0])
{0: 1, 2: 1, 4: 2, 6: 1, 9: 1}

>>> list_to_dict([0, 0, 0, 0, 0])
{}
```

Exercise 8.11 Back To Sparse Vector

A sparse vector is a vector whose entries are almost all zero, like `[1, 0, 0, 0, 0, 0, 0, 2, 0]`. Storing all those zeros wastes memory and dictionaries are commonly used to keep track of just the nonzero entries. For example, the vector shown earlier can be represented as `{0:1, 7:2}`, since the vector it is meant to represent has the value 1 at index 0 and the value 2 at index 7. Write a function that converts a dictionary back to its sparse vector representation.

Run example:

Sparse Vector


```
>>> dict_to_list({0: 1, 3: 2, 7: 3, 12: 4})
[1, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0, 0, 4]

>>> dict_to_list({0: 1, 2: 1, 4: 2, 6: 1, 9: 1})
[1, 0, 1, 0, 2, 0, 1, 0, 0, 1, 0]

>>> dict_to_list({})
[]
```

Classes and Objects

9.1 To study

 Course Book

Read book chapter 15

9.2 Exercises

Exercise 9.1 class Point

Construct a class named `Point`, which takes in the `x` and `y` coordinates as parameters. It should include the documentation string and the methods `__init__` and `__str__`.

Example using class `Point`

```
>>>p1 = Point(2, 4)

>>>p1.__doc__
'A class implementation of a 2-dimensional point.'

>>>str(p1)
'(2, 4) '

>>>print(p1)
(2, 4)
```

Exercise 9.2 class Box

Write a Python class, `Box`. `Box` is meant to represent a box with the attributes `width`, `length` and `height` as dimensions of the box. The attributes should be private. The following methods shall be implemented in the class:

- the constructor `__init__(self, width, length, height)`

- `__str__(self)` - returning a string that can be used for a nice print of an object of the type `class Box`
- `dimensions(self)` - a getter function returning a tuple containing length, width and height of the box
- `volume(self)` - calculates and returns the volume of the box
- `surface(self)` - calculates and returns the entire surface of all six sides of the box

Also write a test program where you create two `Box` objects with different dimensions. Make your program compare the volume of the two boxes and print which one has the largest volume. Also compare the surface of the boxes and let the program print which one has the smallest surface area.

Exercise 9.3 class Dice

Write a Python class, `Dice`. `Dice` is meant to simulate a dice, with the attributes `faces` and `result`, where `faces` is the dice number of sides and `result` is the side facing up right now. All attributes should be private. The following methods shall be implemented in the class:

- the constructor `__init__(self, faces=6)`
- `roll(self)` - updates `result` with a new random number in the range from 1 up to and including `faces`
- `shows(self)` - a getter function returning the current value of `result`

Also write a test program where you create a `Dice` object, roll it and present the outcome a couple of times.

Tip: Import the `random` library with `import random` to be able to call one of the functions `random.randint()` or `random.randrange()`.

Exercise 9.4 class Car

Write a Python class, `Car`. The class should fulfill the following:

- In the constructor, define several attributes of a car. Some good attributes to consider are make (Subaru, Audi, Volvo...), model (Outback, Allroad, C30), year, number of doors, owner, or any other aspect of a car you care to include in your class.
- Write one method `describe_car()`. This method could print a series of statements that describes the car, using the information that is stored in the attributes.
- Write at least one method that adjusts the state of the car, for example changing the mileage, speed or position of the car.

Also write a test program where you create a few `Car` objects with different values of their attributes, also use your method/s on your objects to make sure everything works as intended.

Exercise 9.5 Bank Account

Create a Python class for an account. You should be able to withdraw and deposit money into the account. Think about safety! At least the following methods shall be implemented in the class:

- the constructor
- a method to deposit an amount
- a method to withdraw an amount - be sure to deny if there is not enough money
- a method that returns the account balance as a nice string

Also write a test program where you create an account object and make a sequence of altering deposit and withdrawal calls, and occasionally prints the account balance.

Exercise 9.6 ★ Deck of Cards

Create a Python class for a deck of cards. Internally the deck of cards should use another class, a card class. The requirements are as follows:

- The **Deck** class should have a **deal()** method to deal a single card from the deck. After a card is dealt, it is removed from the deck.
- There should be a shuffle method which makes sure the deck of cards has all 52 cards and then rearranges them randomly.
- The **Card** class should have a suit (Hearts, Diamonds, Clubs, Spades) and a value (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K).

Also write a test program that confirms your deck of cards performs as expected.

Extra Exercises

A function that calls itself is said to be recursive. The creation of recursive functions is a powerful technique to solve problems that can be broken down into smaller or simpler form. Here are some extra exercises on recursion, if you like some extra challenge.

If you still have spare time to work with programming there are some interesting sites on the internet providing programming challenges, for example:

🔗 [Project Euler](#), provides mathematical problems suitable for solving by programming.

🔗 [Open Kattis](#), provides a great variety of problems with automatic assessment.

🔗 [CodeinGame](#), provides automatically assessed problems mostly related to game programming.

10.1 To study (*Recursion*)

📖 Course Book

Read book chapter 5.8 - 5.10

10.2 Exercises on Recursion

Exercise 10.1 Factorial

One common use is to find the factorial of a number. The factorial of a number N is simply the number multiplied by the factorial of $(N - 1)$. The factorial of 0 is 1 by definition.

Write a recursive function that calculates and returns the factorial of a number.

Factorial

```
>>> factorial(5)
120
>>> factorial(1)
1
>>> factorial(0)
1
```

Exercise 10.2 Power

Write a recursive function `power(x, y)` to calculate the value of `x` raised to the power of `y`.

Power

```
>>> power(5, 2)
25

>>> power(5, 1)
5

>>> power(5, 0)
1
```

Exercise 10.3 Fibonacci

The Fibonacci numbers are the integer sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, ..., in which each item is formed by adding the previous two. The sequence can be defined recursively by the following definition:

$$F(n) = \begin{cases} 0 & , n = 0 \\ 1 & , n = 1 \\ F(n-1) + F(n-2) & , n > 1 \end{cases}$$

Write a function `fibonacci(number)` that takes in a number as argument. The function should calculate and return the fibonacci number for the number passed in.

Fibonacci

```
>>> fibonacci(1)
1

>>> fibonacci(2)
1

>>> fibonacci(3)
2

>>> fibonacci(4)
3

>>> fibonacci(5)
5

>>> fibonacci(6)
8

>>> fibonacci(7)
13
```

Exercise 10.4 Greatest Common Divisor

The greatest common divisor (gcd) of 2 or more non-zero integers, is the largest positive integer that divides the numbers without a remainder. Write a function to compute the gcd of 2 integers using Euclid's algorithm:

$$\text{gcd}(a, 0) = a$$

$$\text{gcd}(a, b) = \text{gcd}(b, a \% b)$$

gcd

```
>>>gcd(84, 18)
```

```
6
```

```
>>>gcd(112, 42)
```

```
14
```

```
>>>gcd(5, 4)
```

```
1
```

Exercise 10.5 Palindrome

A palindrome is a word, phrase, number or other sequence of units that can be read the same way in either direction. Write a recursive function that determines whether the given word is a palindrome.

Palindrome

```
>>> is_palindrome("Racecar")
```

```
True
```

```
>>> is_palindrome("Never")
```

```
False
```

```
>>> is_palindrome("level")
```

```
True
```

Hints!

1. Check if the first and last characters are equal.
2. Repeat step 1 recursively on its substrings (with first and last characters removed).

Exercise 10.6 Sum of Digits

Write a recursive function `sum_digits()` that takes in an integer and returns the sum of the digits in the integer.

Sum Digits

```
>>> sum_digits(1234)
10

>>> sum_digits(4321)
10

>>> sum_digits(111)
3

>>> sum_digits(0)
0
```

Exercise 10.7 Count Upper Case X

Write a recursive function `countX()` that takes in a string and returns the number of uppercase character 'X' in the string.

Example countX()

```
>>> countX('aXxXa')
2

>>> countX('aaXaXaax')
2

>>> countX('xx')
0
```

Exercise 10.8 Add Dashes

Write a recursive function `add_dashes()` that takes in a string and returns a new string with all the characters separated by a "-".

Add Dashes

```
>>> add_dashes('google')
'g-o-o-g-l-e'

>>> add_dashes("")
None

>>> add_dashes('g')
'g'
```

Exercise 10.9 Sum Numbers From One

Write a recursive function `sum_all(number)` that takes in a number and returns the sum of all the integer numbers from one to the number passed in as argument.

Run example:

Sum Numbers From One

```
>>> sum_all(-10)
'Invalid'

>>> sum_all(10)
55

>>> sum_all(5)
15
```

Exercise 10.10 Numbers in Between

Write a recursive function `numbers_between(start, end)` that takes in two numbers and returns a comma-separated string with all the numbers in between the start and end number inclusive of both numbers `start` and `end`.

Run example:

Numbers Between

```
>>> numbers_between(5, 10)
'5,6,7,8,9,10'

>>> numbers_between(5, 5)
'5'

>>> numbers_between(5, 8)
'5,6,7,8'
```

Exercise 10.11 Star Power

Write a function `create_stars(num)` that takes in a number as argument and returns a string of stars 2^{num} long.

Run example:

Stars

```
>>> create_stars(0)
'* '

>>> create_stars(1)
'** '

>>> create_stars(2)
'**** '

>>> create_stars(3)
'***** '
```

Exercise 10.12 ★ Create Pattern

Write a function `create_pattern(n)` that takes in a number as argument and returns a string based on the following rules. The middle character of the string should always be an asterisk (*). There will be 2 asterisks if there is an even number of characters. The string will contain the less-than character ('<') before the asterisk and the greater-than character ('>') after the asterisk.

Run example:

Create Pattern

```
>>> create_pattern(1)
'* '

>>> create_pattern(2)
'** '

>>> create_pattern(3)
'<*> '

>>> create_pattern(4)
'<***> '

>>> create_pattern(4)
'<<*>> '
```

Exercise 10.13 ★ Print Twos

Write a function `print_twos(n)` that takes in a number as argument and returns a string composed of an odd number multiplied by 2s such that the final value is equal to `n`. There should be equal number of 2s on both sides. If there is an extra 2 it should appear at the front of the string.

Note: The value of the odd number can be 1.

Run example:

Print Twos

```
>>> print_twos(1)
'1'

>>> print_twos(2)
'2 * 1'

>>> print_twos(10)
'2 * 5'

>>> print_twos(20)
'2 * 5 * 2'

>>> print_twos(30)
'2 * 15'

>>> print_twos(32)
'2 * 2 * 2 * 1 * 2 * 2'

>>> print_twos(80)
'2 * 2 * 5 * 2 * 2'
```

Exercise 10.14 ★★ Pascal's triangle

The Pascal's Triangle is formed by filling the top 2 rows with '1's. For subsequent rows, the numbers at the edge are all '1's. Each element inside the triangle is the sum of the 2 elements above it. Write a recursive function to compute the value of each element given the row and column.

```
row = 0          1
row = 1         1  1
row = 2        1  2  1
row = 3       1  3  3  1
row = 4      1  4  6  4  1
row = 5     1  5 10 10  5  1
row = 6    1  6 15 20 15  6  1
```

Run example:

Pascal's Triangle

```
>>> pascal(0, 0)    # top element
'1'

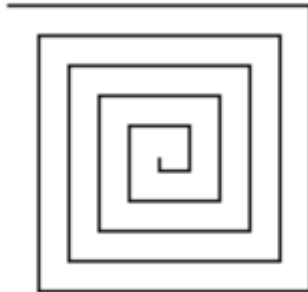
>>> pascal(2, 0)    # third row, first column
'1'

>>> pascal(4, 2)     # 5th row, 3rd col, (4, 2) = (3, 1)+(3, 2)
'6'

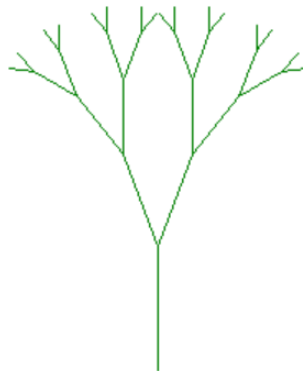
>>> pascal(5, 3)     # 6th row, 4th col, (5, 3) = (4, 2)+(4, 3)
'10'
```

Exercise 10.15 Turtle exercise - Spiral

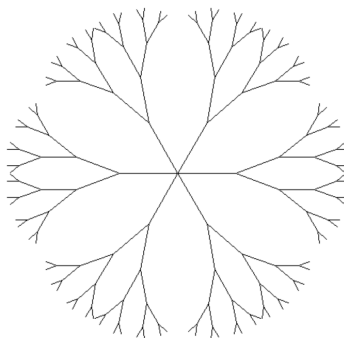
Write a recursive function that creates the spiral pattern shown below. Use the `turtle` graphics module.

**Exercise 10.16** ★ Turtle exercise - Fractal Tree

Write a recursive function that draws a fractal tree as the picture shown below. The branches bends off 20 degrees when they split.



Additional task 1 - Snowflake: Write another recursive function that calls the tree function from above and combines several trees to a "snow flake".



★ ★ **Additional task 2 - Natural Tree:** You can modify the tree function to draw a more natural tree. The trees in nature are not perfect symmetric as the one we have drawn above. Try to modify your function as follows to make your tree look more "real":

- Modify the thickness of the branches to go thinner as they go shorter.
- Modify the color of the branches to change as they go shorter (like leaves).
- Modify the angle where the branches split to be a random angle in a certain interval (15-45 degrees can be feasible).
- Modify the length of the branches, to get shorter by a random value in a fixed interval

