SISTEMAS COMPUTACIONAIS AVANÇADOS (SISTCA)

ADVANCED COMPUTING SYSTEMS

LICENCIATURA EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

POLITÉCNICO DO PORTO

Lab Classes Script:

Introduction to Python

Version Control

Version number	Date issued	Authors	Update information
V1	May 2018	Luís Vilaça (1121405) Supervision: Paula Viana (pmv)	Original version of the script.
V2	April 2019	Ákos Komlósi (1182050) Cédric Spiegel (1181971) Supervision: Paula Viana (pmv)	Revised version of the script. Introduction, setup, screenshots, more functionalities, self-explanatory code snippets for each functionality, two new exercises and provided commented solutions with proper syntax layout, added bibliographic references.
V2.1	March 2020	André Rocha (anr) Mário Alves (mjf)	Revised version of the script. Crosschecked all code/steps. Changed example codes, added information about *Logical Operators*, changed Exercise A and updated the exercises solutions. Text revision, version control table.
V2.1a	March 2021	Mário Alves (mjf) André Rocha (anr) Veríssimo Santos (vms)	Minor text/formatting corrections.

Table of Contents

1. Objectives 2
2. Introduction 2
3. Setup 3
4. Functionalities 4
   - 4.1. Variables and data structures 4
   - 4.2. Input and output [7] 9
   - 4.3. Conditions [8] [9] 9
   - 4.4. Functions [10] 9
   - 4.5. Iterations [9] [11] [12] [13] 10
   - 4.6. Modules [14] 11
   - 4.7. Classes [15] [16] 11
   - 4.8. Methods [10] [17] 12
   - 4.9. Subclasses (Inheritance) [18] [19] [20] 12
5. Exercises 14
   - 5.1. Exercise A: Rock, Paper, Scissors 14
   - 5.2. Exercise B: Shapes and Sizes 14
6. Solutions 15
   - 6.1. Exercise A 15
   - 6.2. Exercise B 16
7. Bibliography 16

1. Objectives

This lab script aims at introducing the student to the *Python* programming language, based on the assumption that the student has basic C programming and algorithmic skills.

After reading, executing and understanding the reasoning of the experiments in this lab script, the student should be able to complete the exercises provided at the end.

For more information, the student should visit the official website [1] and read the existing documentation.

2. Introduction

Python is one of the most interesting and widely adopted programming languages. According to the TIOBE index [2], which ranks the most popular programming languages, Python is the third most popular programming language.

Python has a couple of enhancements when compared to other programming languages [3]:

- It is an interpreted language, which means that the language can execute instructions directly and freely, without previously compiling the code into low-level machine instructions.
- It is a high-level language, which means that it does not depend on details of the computer/microprocessor but rather tries to be readable and understandable by human beings; Python doesn't deal with registers, memory addresses and call stacks, but rather with variables, objects and arrays.
- It is a general-purpose language, which means that it can be used to program software applications in a wide variety of application domains. Python can, for example, be used to develop an application for a mobile platform such as a smartphone or tablet, a web application or a data processing application.

This programming language focuses on making code readable for human beings by emphasizing the importance of whitespace and using many English keywords.

As mentioned before, Python is a multifunctional, general-purpose programming language. This means that it is suited for multiple programming categories like object-oriented, functional, imperative and aspect-oriented.

All of the above-mentioned aspects have led to big companies supporting and embracing Python: Wikipedia, Google, Yahoo!, NASA, Facebook, Amazon,… all use Python in one way or another.

In this lab script, we are going to look at some of Python's major programming functionalities. To assess these functionalities and the students' skills, we propose a couple of challenge exercises at the end.

3. Setup

The installation of Python depends on the Operating System, as follows:

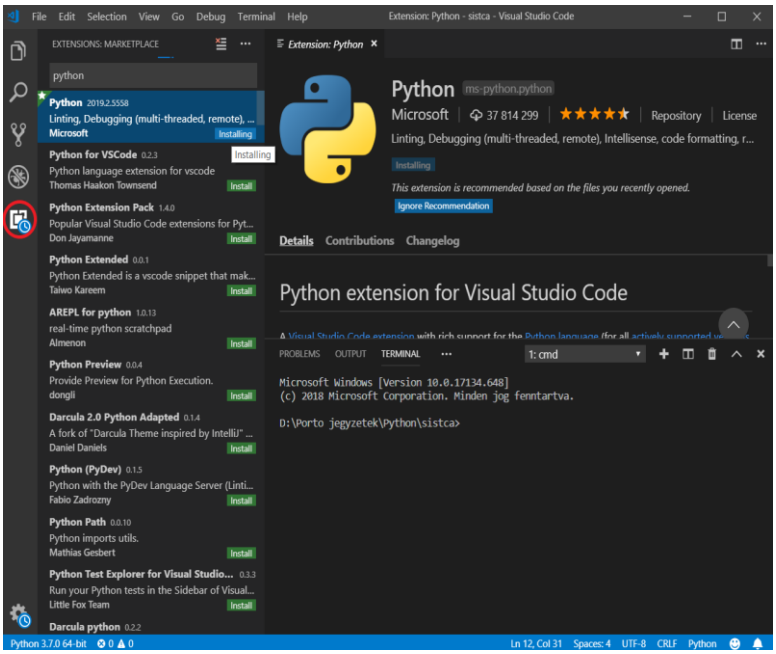
1. Linux: Python is pre-installed in most Linux distributions. We are going to use Python version 3. To confirm this, just write in the console "python3" and check if a prompt gets launched.

2. Windows: to install Python, go to the following website and download the latest version, selecting your system architecture (recommended python3.x). Do not forget to select the "ADD VARIABLE TO PATH" field. https://www.python.org/downloads/windows/

3. macOS: Python is usually pre-installed, but it is advisable to upgrade to the latest version, in https://www.python.org/downloads/mac-osx/

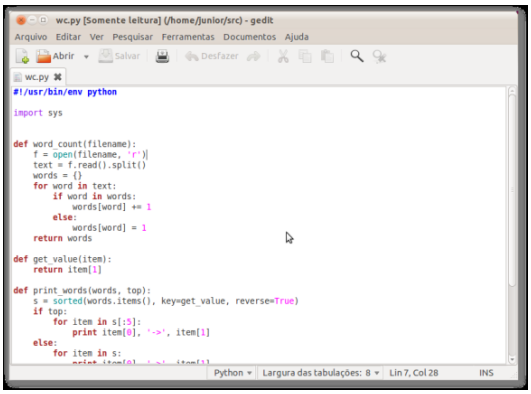
It is recommended that the student uses a text editor such as *Visual Studio Code* because with the correct packages you will be able to run your Python file through the editor as well as to use *Auto-Complete* and debugging functionality that will assist you in programming and learning.

To download *Visual Studio Code*, go to https://code.visualstudio.com/download and download the version selecting your system architecture. After installing it, you need to download the python extension. You can do it by searching for it in the extensions part shown in the figure below.

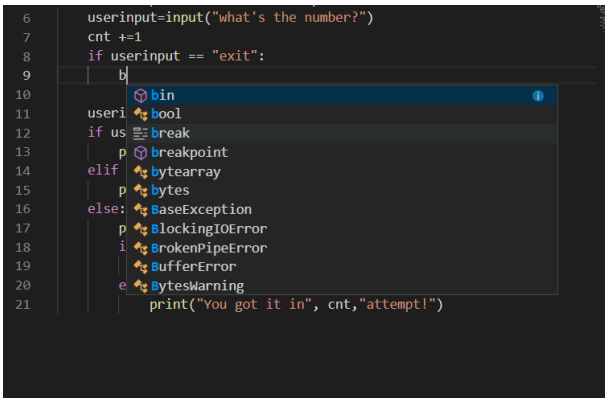


These text editors have the advantage of being able to install packages that allow the user to develop their code through functions such as ***linter*** that helps debug the syntax as we develop it, because it alerts for possible syntax errors, and ***autocomplete***, that shows and inserts suggestions, with the consent of the user, regarding the context of the program (functions, structures, etc.).

The following two images illustrate the differences between using a traditional editor such as ***gedit*** (on the left) or a more elaborated text editor such as *Visual Studio Code* (on the right). The auxiliary screen of the *Auto-Complete*" function is shown in the right-hand image.

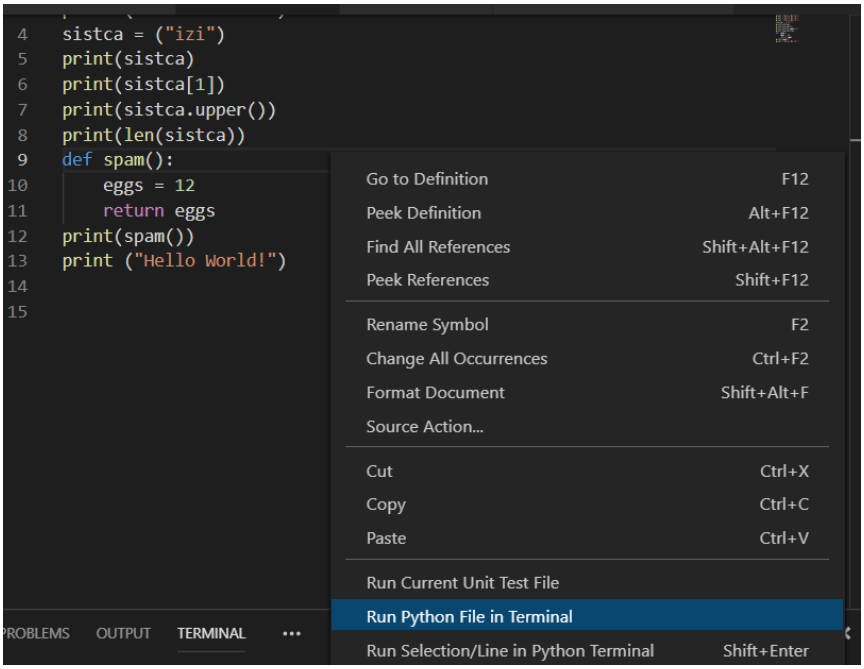


Gedit



Visual Studio Code

To make a Python file, just make a new file with the name extension '.py'. Then you can start programming. To run the code, type in the terminal 'python *nameofthefile*.py' or right-click and select the option '*Run Python File in Terminal*', as illustrated in the following figure:



4. Functionalities

4.1. Variables and data structures

Data structures are a way of organizing and string data in a way that we can access and work with them. There are two types of data structures [4] [5] [6]:

Linear data structure:

In the linear data structure, data items are organized linearly, one after another, and only one data item can be accessed. Example of linear data structures:

- array
- linked list
- queue
- stack

Non-linear data structure:

Opposite to linear data structures, in non-linear data structure data items are not organized in a sequential structure and are connected to several other data items to represent a specific relationship. Example of non-linear data structures:

- tree
- heap
- hash table
- graph

In Python, you do not need to declare variables before using them, or declare their type. Every variable in Python is an object and is created at the moment you first assign a value to it. The equal sign (=) is used to assign values to variables.

There are 4 important rules for creating Python variables:

- A variable name must start with a letter or the underscore character; a variable name cannot start with a number.
- A variable name can only contain alpha-numeric characters and underscores: (A – Z, 0 – 9, _)
- Variable names are case-sensitive, which means that "sistca", "Sistca" and "SISTCA" are three different variables.

Python allows you to simultaneously assign a single value to multiple variables; for instance:

```
a = b = 1
```

Python also allows you to assign multiple objects to multiple variables, e.g.:

```
a, b, c = 1, 2, "sistca"
```

Here, the two *integer* objects – '1' and '2' are assigned to variables *a* and *b*, and one *string* object with the value 'sistca' is assigned to variable *c*.

To output variables, the most often used statement is *print()*. You can combine texts and variables or two variables by using the **+** character.

```
x = "awesome"
print("Python is " + x)
```

```
Python is awesome
```

```
x = "Python is "
y = "awesome"
z = x + y
print(z)
```

```
Python is awesome
```

However, if you use the **+** character for numbers, it works as a mathematical operator.

```
x = 5
y = 10
print(x + y)
```

```
15
```

If you try to combine two variables and one of them is a string and the other is a number, python will give you an error message.

Python has five standard data types that self-define the type of admissible operations and storage methods, as follows:

- numbers
- strings
- lists
- tuples
- dictionaries

Numbers:

It stores numeric values. A *number* object is created when a value is assigned to it, e.g.:

```
num = 10
```

Python enables to define 3 different numerical types: int, float, complex.

Strings:

Strings are a set of characters enclosed by quotation marks. Python 3 allows either a pair of single or double quotation marks. Parts of strings can be taken using the slice operator ([ ] and [:]) with indexes starting from 0 at the beginning of the string and working their way up to index '*end* – 1'.

There are several useful and interesting *methods* which you can use with *strings*, these are:

- *Strip()*: it removes all whitespaces from the beginning/end of the string.
- *Len()*: returns the string length.
- *Lower()*: transforms the contents of the string to lowercase.
- *Upper()*: transforms the contents of the string to uppercase.
- *Replace()*: replaces the string with another string.
- *Split()*: splits the string into substrings if it finds separators (separator: comma).

Lists:

A *list* contains ordered items, separated by commas and enclosed within square brackets ([]). The items stored in a list can be accessed using a slice operator ([ ] and [:]) with indexes starting from 0 at the beginning of the list and working their way up to index '*end* – 1'.

Creating a list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

You can change the values of the items as shown in the example below.

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"

print(thislist)
```

```
['apple', 'blackcurrant', 'cherry']
```

You can check if a certain item exists in a list by using the ***in*** keyword.

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits
list")
```

```
Yes, 'apple' is in the fruits list
```

In this case, the *len()* method shows the number of items in a list.

You can add items to the list in two different ways. If you use the *append()* method, it will add the new item to the end of the list. However, if you want to add the item to a specified index, you need to use the *insert()* method.

```
thislist = ["apple", "banana", "cherry"]

thislist.append("orange")

print(thislist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

```
['apple', 'orange', 'banana', 'cherry']
```

You can remove items from a list in several ways. If you use the *remove()* method, it removes the specified item. You can also use the *del* keyword which removes an item at a specified index; it can also be used to delete the whole list. You can clear the list by using the *clear()* method.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

```
['apple', 'cherry']
```

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

```
['banana', 'cherry']
```

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

```
[]
```

Tuples:

It is another sequence data type which is similar to the *list*, however, it is written with round brackets (()) and you cannot update them.

The *len()* method and the *in* keyword work the same way as in *lists*.

Dictionaries:

It is a collection which is unordered, updatable and indexed. Dictionaries are Python's built-in mapping type. They map *keys* to *values*, and these pairs provide a useful way to store data. A *dictionary key* can be almost any variable type, but it is usually a *number* or a *string*. *Values*, on the other hand, can be any Python object. Dictionaries are written with curly braces ({ }).

Creating a *tuple*:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Values can be assigned and accessed using square braces ([]).

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

```
Mustang
```

The *len()* method and the *in* keyword work the same way as in the case of lists and tuples.

You can change the value of an item by referring to its key name.

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

thisdict["year"] = 2018

print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

You can add items by using a new index key and assigning a value to it.

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

There are several ways to delete items from a dictionary. The ***pop()*** method removes an item with a specified key name. The ***del*** keyword also removes an item with a specified keyword and also can be used to delete the dictionary. You can clear the dictionary by using the ***clear()*** method.

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

```
{}
```

4.2. Input and output [7]

Taking input from the user is very easy. The ***input()*** function takes input from a user after printing out the statement in the parameter. This input can then be saved in a variable and printed out, like in the following example:

```
x = input("Enter your name:")
print("Hello, " + x)
```

```
Enter your name:Cédric
Hello, Cédric
```

4.3. Conditions [8] [9]

A *condition* can be expressed in Python by using 3 keywords: ***if***, ***elif*** and ***else***, in a similar way to other programming languages. Conditions are compound statements, meaning that the statement consists of a *header* and a *body*. The colon (**:**) is very important because it separates the condition's header from its body. If the header condition is fulfilled, the code in the condition body gets executed. Logical Operators ("and", "or" and "not") can also be used to evaluate two or more expressions in the same header. Example:

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
elif (a > b) and (b > 50):
  print("a is greater than b and b is greater than 50"
else:
  print("a is greater than b")
```

```
a is greater than b
```

4.4. Functions [10]

A *function* is a sequence of code commands that is used to tackle a particular task. To define a function, Python uses the keyword ***def***. Then, to execute this function, just call it by its name:

```
def my_function():
  print("Hello from a function")

my_function()
```

```
Hello from a function
```

4.5. Iterations [9] [11] [12] [13]

An *iteration* is a piece of code that executes instructions again and again, until a condition is met or infinitely. Python provides three types of looping techniques:

- For-loop

This looping technique is used when there is a piece of code and it has to be repeated 'n' number of times.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

```
apple
banana
cherry
```

- While-loop

This looping technique repeats a piece of code until a specific Boolean condition is met.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

- Nested-loop

This represents a looping technique nested in another looping technique. For example, a for-loop nested in a for-loop.

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Why would someone use a while-loop if there are for-loops? A for-loop is only usable in cases of **definite** iteration. This means that the upper boundary is specified, so that the programmer knows exactly how many times the loop has to be executed. However, in the case of an **indefinite** iteration, only a while-loop is adequate, since the programmer does not know how many times the loop will be executed, but he knows when (upon which condition) it should stop.

The body of a loop can be exited at an early stage (before the condition is met) using a ***break*** statement**.** When reaching the break, the program immediately exits the loop and continues with the code after the loop.

```
for i in range(9):
  if i > 3:
    break
  print(i)
print("Next instruction...\nvalue of i = ", i)
```

```
0
1
2
3
Next insctruction
value of i = 4
```

It is also possible to skip the current iteration of the loop, but continue with the next iteration of the loop. This can be done by using a ***continue*** statement**.** In the following example, the number '3' will not be printed because that iteration has been skipped.

```
for i in range(9):
  if i == 3:
    continue
  print(i)
```

```
0
1
2
4
5
6
7
8
9
```

4.6. Modules [14]

The default functionalities of Python can be extended by making use of *modules*. A module is a code library that contains extra functions (e.g. to help the programmer optimizing and organizing the application). A module can be imported into Python by using the ***import*** keyword. The complete list of supported modules can be found in the official docs of Python. Example:

```
import datetime

x = datetime.datetime.now()

print(x)
```

```
2019-03-27 19:11:12.201862
```

4.7. Classes [15] [16]

Python can be used for object-oriented programming, so that a programmer can define *classes*. A class is a blueprint for creating objects and can be created by using the keyword ***class*.** A class can (and most probably will) contain attributes, which are the variable fields that characterize and shape a class. These attributes are automatically public, and can be accessed by using the dot (**.**) operator with an instance of that class.

It is important to note that all classes should have a function called ***__init__()***, which enables to create instances of this class and to assign values to your object attributes. The *__init__()* function is automatically executed when an instance of the class is initiated. Please mind this example, for clarification:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

```
John
36
```

The ***self*** parameter in the example above refers to the object itself. It is used to access and assign values to the variables of this object.

In addition to the *__init__()* function, it is also possible to define other functions inside a Python class. A function that is defined inside a class and is consequently used with objects, is called a ***method***, as outlined in the following section.

4.8. Methods [10] [17]

A *method* is quite similar to a *function*. The main difference between the two is that a method is called on an object, and a function is not. Following this, a method can change the state of an object but a function can only operate on it, and not change it. A method can be added to the previous code snippet to make things clear:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

```
Hello my name is John
```

In the example above, a self-defined method ***myfunc()*** is added, which will print something out when called. The method can then be activated by using the dot (**.**) operator on an object and accessing the method.

4.9. Subclasses (Inheritance) [18] [19] [20]

Inheritance is a feature of object-oriented programming languages that makes it possible for a particular class to take over all functionalities from a base class, and add some things to it. The newly created class is called the **derived** or **child class**, and the one from which it inherits is called the **base** or **parent class.** The example that follows might seem a bit overwhelming at first, but it should get clear at the end:

First, a base class has to be created.

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname
method:

x = Person("John", "Doe")
x.printname()
```

An object of the class *Person* has a first name, a last name and a method to print both out.

Next, a child class has to be created. This class will take all variables and methods from its parent class. If we just add an *__init__()* function to the child, it will not inherit the parent's variables and methods. The solution for this is calling the *__init__()* twice:

```
class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)
```

Like this, an object from the class *Student* has all the properties of the class *Person*, but additional properties can be added to its own __init__(). When executed successfully, an inheritance should look like this:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    Person.__init__(self, fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname,
self.lastname, "to the class of",
self.graduationyear)

x = Student("Mike", "Olsen", 2019)
x.welcome()
x.printname()
```

```
Welcome Mike Olsen to the class of 2019
Mike Olsen
```

5. Exercises

5.1. Exercise A: Rock, Paper, Scissors

This first program will be a two-player Rock-Paper-Scissors game. Ask for the player's move ("R" for Rock, "P" for Paper or "S" for Scissors), compare it with the randomly chosen move of the program and then print out a message to congratulate the winner. Make the program stop if the user inserts a "Q" (for Quitting). Remember the rules:

- Rock beats scissors
- Scissors beats paper
- Paper beats rock

*Hint: To generate random numbers, the program will have to import the **random** library, which contains several useful functions to play around with random numbers. For this exercise, **choice()** is the function that should be used.*

5.2. Exercise B: Shapes and Sizes

1) Write a Python class named "*Rectangle*", which accepts a *length* and a *width* as parameters.

2) Write a method dubbed "*print_rectangle_surface()*" in this class, which calculates and prints out the area of the rectangle.

3) Write a method called "*print_rectangle_perimeter()*" in this class, which calculates and prints out the perimeter of the created rectangle.

4) Write a method "*rectangle_equalSides()*" in this class, which checks whether or not the sides of the rectangle are equal and prints out the result.

Now, create an equal and an unequal rectangle object and apply all methods on both rectangles.

6. Solutions

6.1. Exercise A

```
import random

listPossibilities = ["rock","paper","scissors"]

print("\n** Rock-Paper-Scissors **")

#infinite while loop
while True:
    print("\nPlease choose \"R\" for rock, \"P\" for paper or \"S\" for scissors. If you
want to quit, type \"Q\".")
    #ask user input and save in variable
    userChoice = input("Please enter your choice: ")
    if userChoice == "Q":
           break

    #choose a random possibility from the list
    pcChoice = random.choice(listPossibilities)
    print("PC chooses: " + pcChoice)

    #condition for every possibility
    if pcChoice == "rock" and (userChoice == "rock" or userChoice == "R"):
        print(pcChoice + " = rock")
        print("Both rock LOL")
        continue
    elif pcChoice == "rock" and (userChoice == "paper" or userChoice == "P"):
        print(pcChoice + " < paper")
        print("User wins")
        continue
    elif pcChoice == "rock" and (userChoice == "scissors" or userChoice == "S"):
        print(pcChoice + " > scissors")
        print("PC wins")
        continue

    if pcChoice == "paper" and (userChoice == "paper" or userChoice == "P"):
        print(pcChoice + " = paper")
        print("Both paper LOL")
        continue
    elif pcChoice == "paper" and (userChoice == "scissors" or userChoice == "S"):
        print(pcChoice + " < scissors")
        print("User wins")
        continue
    elif pcChoice == "paper" and (userChoice == "rock" or userChoice == "R"):
        print(pcChoice + " > rock")
        print("PC wins")
        continue

    if pcChoice == "scissors" and (userChoice == "scissors" or userChoice == "S"):
        print(pcChoice + " = scissors")
        print("Both scissors LOL")
        continue
    elif pcChoice == "scissors" and (userChoice == "rock" or userChoice == "R"):
        print(pcChoice + " < " + userChoice)
        print("User wins")
        continue
    elif pcChoice == "scissors" and (userChoice == "paper" or userChoice == "P"):
        print(pcChoice + " > " + userChoice)
        print("PC wins")
        continue
    else:
        print("false input")
        continue
```

6.2. Exercise B

```
class Rectangle():
    #constructor method
    def __init__(self, length, width):
        self.length = length
        self.width  = width

    #calculate and print the surface with the formula "l*w"
    def print_rectangle_surface(self):
        surface = self.length*self.width
        print(surface)

    #calculate and print the perimeter with the formula "2 * (l+w)"
    def print_rectangle_perimeter(self):
        circum = 2 * (self.length + self.width)
        print(circum)

    #check if sides are equal (by comparing the values) and prints true or false
    def rectangle_equalSides(self):
        if ( self.length == self.width ):
            print(True)
        else:
            print(False)

#create unequal rectangle
unequalRectangle = Rectangle(12, 10)
unequalRectangle.print_rectangle_surface()
unequalRectangle.print_rectangle_perimeter()
unequalRectangle.rectangle_equalSides()
print("------------------------------")
#create equal rectangle
equalRectangle = Rectangle(6,6)
equalRectangle.print_rectangle_surface()
equalRectangle.print_rectangle_perimeter()
equalRectangle.rectangle_equalSides()
```

7. Bibliography

[1] Python, „Python,” Python, 2019. [Online]. Available: https://www.python.org/. [Geopend 19 March 2019].

[2] TIOBE, „TIOBE Index for April 2019,” TIOBE, 2019. [Online]. Available: https://www.tiobe.com/tiobe-index/. [Geopend 2019 March 19].

[3] Wikipedia, „Python (programming language),” Wikipedia, 9 April 2019. [Online]. Available: https://en.wikipedia.org/wiki/Python_(programming_language). [Geopend 20 March 2019].

[4] Z. Pandovski, „Data Structures in Python 3,” Able, 2 June 2018. [Online]. Available: https://able.bio/ZoranPandovski/data-structures-in-python-3--04jfygn. [Geopend 20 March 2019].

[5] W3Schools, „Python Variables,” W3Schools, [Online]. Available: https://www.w3schools.com/python/python_variables.asp. [Geopend 26 March 2019].

[6] Tutorialspoint, „Python 3 - Variable Types," Tutorialspoint, 2019. [Online]. Available: https://www.tutorialspoint.com/python3/python_variable_types.htm. [Geopend 26 March 2019].

[7] Pythonbasics, „Keyboard input," PythonBasics, 2019. [Online]. Available: https://pythonbasics.org/keyboard-input/. [Geopend 17 March 2019].

[8] W3Schools, „Python If... Else," W3Schools, 2019. [Online]. Available: https://www.w3schools.com/python/python_conditions.asp. [Geopend 26 March 2019].

[9] J. Elkner, „Conditionals and loops," openbookproject, 2017. [Online]. Available: http://www.openbookproject.net/books/bpp4awd/ch04.html. [Geopend 27 March 2019].

[10] W3Schools, „Python Functions," W3Schools, 2019. [Online]. Available: https://www.w3schools.com/python/python_functions.asp. [Geopend 2 April 2019].

[11] H. W. Chaudhary, „Python 3 Basics," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/python-3-basics/. [Geopend 2 April 2019].

[12] PythonLearn, „Chapter 5 Iteration," PythonLearn, [Online]. Available: https://www.pythonlearn.com/html-008/cfbook006.html. [Geopend 2 April 2019].

[13] W3Schools, „Python For Loops," W3Schools, 2019. [Online]. Available: https://www.w3schools.com/python/python_for_loops.asp. [Geopend 2 April 2019].

[14] PythonForBeginners, „How to use the Random Module in Python," PythonForBeginners, 24 December 2012. [Online]. Available: https://www.pythonforbeginners.com/random/how-to-use-the-random-module-in-python. [Geopend 2 April 2019].

[15] W3Schools, „Python Classes and Objects," W3Schools, 2019. [Online]. Available: https://www.w3schools.com/python/python_classes.asp. [Geopend 3 April 2019].

[16] H. W. Chaudhary, „Python 3 Intermediate Level Topics," GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/python3-intermediate-level-topics/. [Geopend 3 April 2019].

[17] K. Boyini, „Difference between Method and Function in Python," Tutorialspoint, 19 February 2019. [Online]. Available: https://www.tutorialspoint.com/difference-between-method-and-function-in-python. [Geopend 3 April 2019].

[18] B. Klein, „Inheritance," Python-course, 2018. [Online]. Available: https://www.python-course.eu/python3_inheritance.php. [Geopend 3 April 2019].

[19] Programiz, „Python Inheritance," Programiz, [Online]. Available: https://www.programiz.com/python-programming/inheritance. [Geopend 3 April 2019].

[20] W3Schools, „Python Inheritance," W3Schools, 2019. [Online]. Available: https://www.w3schools.com/python/python_inheritance.asp. [Geopend 3 April 2019].