# Introduction to Python

**Programming IV Class Exercises**

João Caldeira, ISTEC, Lisbon, Portugal

6/19/23

# Table of contents

**Preface** **1**
Author . . . 1
License . . . 1

**I. Exercises** **3**

**1. Virtual Environments** **5**
1.1. Task : Manage Python Virtual Environments . . . 5
1.1.1. Virtual Environment 1 . . . 5
1.1.2. Virtual Environment 2 . . . 5
1.1.3. Virtual Environment 3 . . . 5
1.1.4. Questions . . . 5

**2. Projects Structure** **7**
2.1. Task : Manage Python Projects Structure . . . 7
2.1.1. Project 1 . . . 7
2.1.2. Project 2 . . . 7
2.1.3. Questions . . . 7

**3. Building Packages** **9**
3.1. Task : Create a Python Package . . . 9
3.1.1. Build Package "**istec-p4**" . . . 9
3.1.2. Install your Package . . . 9

**4. Publishing Packages** **11**
4.1. Task : Publish a Python Package . . . 11
4.1.1. Package "**istec-p4**" . . . 11
4.1.2. Using Package "**istec-p4**" . . . 11

**5. Data Types** **13**
5.1. Task : Return changed size strings . . . 13
5.1.1. Package "**istec-p4**" . . . 13
5.2. Task : Return string parts . . . 13
5.2.1. Package "**istec-p4**" . . . 13

**6. Working with Lists** **15**
6.1. Task : Creating Lists . . . 15
6.1.1. Package "**istec-p4**" . . . 15
6.2. Task : Changing List Items . . . 15
6.2.1. Package "**istec-p4**" . . . 15
6.3. Task : Add and Remove List Items . . . 15
6.3.1. Package "**istec-p4**" . . . 15

**7. Working with Dictionaries** **17**
7.1. Task : Creating a Dictionary . . . 17
7.1.1. Package "**istec-p4**" . . . 17
7.2. Task : Changing Dictionary Items . . . 17
7.2.1. Package "**istec-p4**" . . . 17

**8. Working with Tuples** **19**
8.1. Task : Creating a Tuple . . . 19
8.1.1. Package "**istec-p4**" . . . 19
8.2. Task : Evaluating Tuple Items . . . 19
8.2.1. Package "**istec-p4**" . . . 19
8.3. Task : Counting Tuple Items . . . 19

**9. Working with Sets** **21**
9.1. Task : Creating a Set . . . 21
9.1.1. Package "**istec-p4**" . . . 21
9.2. Task : Managing Set Items . . . 21
9.2.1. Package "**istec-p4**" . . . 21
9.3. Task : Evaluating Multiple Sets . . . 22

**10. Working with Decision Structures** **23**
10.1. Task : Creating a selective List . . . 23
10.1.1. Package "**istec-p4**" . . . 23
10.2. Task : Managing a selective List . . . 23
10.2.1. Package "**istec-p4**" . . . 23

**11. Working with Repetition Structures** **25**
11.1. Task : Creating a selective reversed list . . . 25
11.1.1. Package "**istec-p4**" . . . 25
11.2. Task : Managing a selective List . . . 25
11.2.1. Package "**istec-p4**" . . . 25
11.3. Task : Managing exceptions . . . 25
11.3.1. Package "**istec-p4**" . . . 25

**12. Working with Functions** **27**
12.1. Task : Creating User Defined Functions . . . 27
12.2. Task : Creating Lambda Functions . . . 27
12.3. Task : Creating Recursive Functions . . . 28

**13. Working with Classes** **31**
13.1. Task : Creating a Class . . . 31
13.2. Task : Creating an Object . . . 31
13.3. Task : Creating Methods . . . 31

**II. Solutions** **33**

**14. Virtual Environments** **35**
14.1. Task : Manage Python Virtual Environments . . . 35
14.1.1. Virtual Environment 1 . . . 35
14.1.2. Virtual Environment 2 . . . 36
14.1.3. Virtual Environment 3 . . . 37

**15. Projects Structure** **39**
15.1. Task : Manage Python Projects Structure . . . 39
15.1.1. Project 1 . . . 39
15.1.2. Project 2 . . . 40

**16. Building Packages** **43**
16.1. Task : Create a Python Package . . . 43
16.1.1. Build Package "**istec-p4**" . . . 43
16.1.2. Install your Package . . . 43

**17. Publishing Packages** **45**
17.1. Task : Publish a Python Package . . . 45
17.1.1. Package "**istec-p4**" . . . 45
17.1.2. Using Package "**istec-p4**" . . . 45

**18. Data Types** **47**
18.1. Task : Return changed size strings . . . 47
18.2. Task : Return string parts . . . 47

**19. Working with Lists** **49**
19.1. Task : Creating Lists . . . 49
19.2. Task : Changing List Items . . . 49
19.3. Task : Add and Remove List Items . . . 49

**20. Working with Dictionaries** **51**
20.1. Task : Creating a Dictionary . . . 51
20.2. Task : Changing Dictionary Items . . . 51

**21. Working with Tuples** **53**
21.1. Task : Creating a Tuple . . . 53
21.2. Task : Evaluating Tuple Items . . . 53
21.3. Task : Counting Tuple Items . . . 53

**22. Working with Sets** **55**
22.1. Task : Creating a Set . . . 55
22.2. Task : Managing Set Items . . . 55
22.3. Task : Evaluating Multiple Sets . . . 55

**23. Working with Decision Structures** **57**
23.1. Task : Creating a selective List . . . 57
23.2. Task : Managing a selective List . . . 57

**24. Working with Repetition Structures** **59**
24.1. Task : Creating a selective reversed list . . . 59
24.2. Task : Managing a selective List . . . 59
24.3. Task : Managing exceptions . . . 59

**25. Working with Functions** **61**
25.1. Task : Creating User Defined Functions . . . 61
25.2. Task : Creating Lambda Functions . . . 61
25.3. Task : Creating Recursive Functions . . . 61

**26. Working with Classes** **63**
26.1. Task : Creating a Class . . . 63

26.2. Task : Creating an Object . . . . . 63
26.3. Task : Creating Methods . . . . . 64

**27. Main Program (For Test Purposes) 65**

**28. Test Solutions (19-May-2023) 69**
28.1. Practical Questions Solution . . . . . 69

# List of Figures

# List of Tables

# Preface

This guide is a compilation of a long list of exercises of Python in the context of the **Programming IV** classes within **ISTEC**. It covers many aspects of the Python language, from just introductory exercises to more complex and time consuming tasks, including **if statements, loops, functions, lists, dictionaries, tuples, dataframes, file and exception** exercises and much more. It is mainly focused on students starting to learn the Python language, however, it can also be used by individuals with higher levels of experience. Each exercise will have the correspondent solution and eventually some alternatives to accomplish the same task.

Some exercises may start by listing the learning objectives and required reading materials, with additional references if needed. Including notes may introduce the methods and give a few examples but are less detailed than the reading materials. Some notes are important messages to complete the tasks properly.

The audience is expected to be beginners of the Python programming language but with some experience in coding and algorithms knowledge.

> **Note**
>
> **These exercises assume the use of macOS and Visual Studio code IDE.**

## Author

João Caldeira is an Invited Professor at ISTEC and Associate Researcher at ISCTE-IUL in Lisbon, Portugal. https://www.researchgate.net/profile/Joao-Caldeira-2

## License

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

# Part I.

# Exercises

# 1. Virtual Environments

## 1.1. Task : Manage Python Virtual Environments

Based on your Operating System, **create** and **activate** multiple/different Python virtual environments.

### 1.1.1. Virtual Environment 1

- Create a Python virtual environment named **env1**
- Activate the environment named **env1**
- Install package **requests** version **2.18.4** in this environment
- List the existing packages in this environment
- Create a **Notebook (*.ipynb)** called **FirstNotebook.ipynb** in this environment
- Add in the notebook, as the first block, the following **Markdown**
  - "# My First Notebook"
- Add as the next block the following **Code**
  - **print("I now know about Markdown & Code blocks.")**
  - **print("I also created my first Python environment.")**

### 1.1.2. Virtual Environment 2

- Create a Python environment named **env2**
- Install package **requests** version **2.18.4** in this environment
- List the existing packages in this environment

### 1.1.3. Virtual Environment 3

- Create a Python environment named **env3**
- Activate the environment named **env3**
- Install package **requests** with the latest version in this environment
- List the existing packages in this environment

### 1.1.4. Questions

Are the virtual environments the same ?

What is different ?

What are the benefits of having multiple environments ?

# 2. Projects Structure

## 2.1. Task : Manage Python Projects Structure

### 2.1.1. Project 1

- Create a project named **project1** which will use the environment named **env1**
- Use the correct naming for your project, packages and modules
- Create a package named **istec.p4.exercises**
- Within this package create a module named **exercise1.py**
- Inside this module create a basic function named **hello()** to execute **print("Hello World!!")**
- Execute your program

### 2.1.2. Project 2

- Start from **project1** code and duplicate it as **project2**
- Update your code in **project2** to produce an output similar to this:
  - **Hello World!!**
  - **I'm running using virtual environment: env1.** *(or)*
  - **I'm running using virtual environment: env2.** *(for when you are running it using another virtual environment)*
- Execute your program

> ! Important
>
> **Your code should be dynamic, meaning, you should not hard code your virtual environment name.**

### 2.1.3. Questions

Did it run ?

What was the output ?

# 3. Building Packages

## 3.1. Task : Create a Python Package

### 3.1.1. Build Package "istec-p4"

- Create a project/package named **istec-p4** which will use any of your existing virtual environments.
- Use the correct naming for your project, packages and modules.
- Create a package with the following structure **istec/p4/exercises**.
- Within this package, create a module named **greetings** with the following within:
  - A function named **hello** to execute **print("Hello World!!")**
  - A function named **good_morning** to execute **print("Good Morning!!")**
  - A function named **good_afternoon** to execute **print("Good Afternoon!!")**

### 3.1.2. Install your Package

- Make the necessary configurations to make your package named **istec-p4** locally instalable.
- Install your package locally in the virtual environment you are using.
- To be locally instalable means, you will install your package in any of your virtual environments, and then, from any other project using the same environment, you can call your package/modules like this:

```
from istec.p4.exercises import greetings
```

- Once you load your package then you can call the existing functions within the package.

# 4. Publishing Packages

## 4.1. Task : Publish a Python Package

### 4.1.1. Package “istec-p4”

- Use the package you created in the previous task and make it public.
- Publish your package to the **TestPyPi** repository.
  - Navigate to **https://test.pypi.org/** and Register yourself as an user.
- This means your package will be available for everyone (worldwide) to install:

### 4.1.2. Using Package “istec-p4”

- Later, to install from **TestPyPi** you will issue this command:

```
pip install --index-url https://test.pypi.org/simple/ istec-p4
```

- Later, to install from **PyPi** you will issue this command:

```
pip install istec-p4
```

- Then you can load your package and call the existing functions within that package/module.

```
from istec.p4.exercises import greetings

# Calling the hello function
greetings.hello()
```

> **Note**
>
> The official **Python** packages repository is called **PyPi (https://pypi.org/)**.

# 5. Data Types

## 5.1. Task : Return changed size strings

### 5.1.1. Package “istec-p4”

- Use the package you created in the previous session.
- Use the package locally (you don’t need to publish it to everyone)
- Within this package, create a module named **data-types** with the following within:
  - A function named **myupper**.
    - This function will receive as an input a string and return the same string in uppercase.
  - A function named **mylower**.
    - This function will receive as an input a string and return the same string in lowercase.
  - A function named **invertSizing**
    - This function will receive as an input a string and if the string is in uppercase, it will return the same string as lowercase. If the string is in lowercase it will return the same string in uppercase.

## 5.2. Task : Return string parts

### 5.2.1. Package “istec-p4”

- Use the package you created in the previous session.
- Use the package locally (you don’t need to publish it to everyone)
- Within this package, create a module named **data-types** with the following within:
  - A function named **string-part**
    - This function will receive 2 parameters as an input: a string **st** and a number **n**.
    - The function return the first **n** characters of the string **st**.

# 6. Working with Lists

## 6.1. Task : Creating Lists

### 6.1.1. Package “istec-p4”

- Use the package you created in the previous session.
- Use the package locally (you don’t need to publish it to everyone)
- Within this package, create a module named **lists** with the following within:
  - A function named **mylistst**.
    - This function will receive as an input a string and return each character in a list.
  - A function named **mylistnum**.
    - This function will receive as an input a number and return a list of numbers from 0 to that number.

## 6.2. Task : Changing List Items

### 6.2.1. Package “istec-p4”

- Use the package you created in the previous session.
- Use the package locally (you don’t need to publish it to everyone)
- Within this package, create a module named **lists** with the following within:
  - A function named **myfirstnames**
    - This function will receive as a parameter a list of lowercase names **(ex: [“joão”,“pedro”,“luís”,“ana”,“susana”,“teresa”])** as an input and should return a list with the names where the first character is in uppercase **(ex: [“João”,“Pedro”,“Luís”,“Ana”,“Susana”,“Teresa”])**.

## 6.3. Task : Add and Remove List Items

### 6.3.1. Package “istec-p4”

- Use the package you created in the previous session.
- Use the package locally (you don’t need to publish it to everyone)
- Within this package, create a module named **lists** with the following within:
  - A function named **myaddremove**
    - This function will receive as a parameter a list of strings, and 2 other parameters, as individual strings.
    - The first string will be added to the list.
    - the second string will be removed from the list.
    - The changed list is returned.

# 7. Working with Dictionaries

## 7.1. Task : Creating a Dictionary

### 7.1.1. Package "istec-p4"

- Use the package you created in the previous session.
- Use the package locally (you don't need to publish it to everyone)
- Within this package, create a module named **dictionaries** with the following within:
  - A function named **mydictionary**.
    - This function will receive as parameters 2 lists: one with keys and another with values.
      - **Ex: keys = ["John","Mary","Martha"] , values = [35,43,28]**
    - The function should return a dictionary of the above data.
      - **Ex: {'John':35, 'Mary':43, 'Martha':28}**

## 7.2. Task : Changing Dictionary Items

### 7.2.1. Package "istec-p4"

- Use the package you created in the previous session.
- Use the package locally (you don't need to publish it to everyone)
- Within this package, create a module named **dictionaries** with the following within:
  - A function named **mydictionary2**.
    - This function will receive as parameters 1 dictionary and 2 lists: one list with keys and another with values.
    - **Ex: {'John':35, 'Mary':43, 'Martha':28}**
    - **Ex: keys = ["Peter","John"] , values = [65,43]**
    - The function should return a dictionary with the addition of the items above.

# 8. Working with Tuples

## 8.1. Task : Creating a Tuple

### 8.1.1. Package "istec-p4"

- Use the package you created in the previous session.
- Use the package locally (you don't need to publish it to everyone)
- Within this package, create a module named **tuples** with the following within:
  - A function named **mytuple**.
    - This function will receive as parameters 2 lists: one with car brands and another with those cars maximum spped.
      - **Ex: cars = ["Bmw","Audi","Volvo"] , speeds = [250,230,220]**
    - The function should return a **Tuple** with the above data as follows.
      - **Ex: {'Bmw',250,'Audi',230,'Volvo',220}**

## 8.2. Task : Evaluating Tuple Items

### 8.2.1. Package "istec-p4"

- Use the package you created in the previous session.
- Use the package locally (you don't need to publish it to everyone)
- Within this package, create a module named **tuples** with the following within:
  - A function named **mytuple2**.
    - This function will receive as parameters 2 lists: one with car brands and another with those cars maximum spped.
      - **Ex: cars = ["Bmw","Audi","Volvo","Audi"] , speeds = [250,230,220,240]**
    - The function should return a **Tuple** with the above data as follows.
      - **Ex: {'Bmw',250,'Audi',230,'Volvo',220,'Audi',240}**

## 8.3. Task : Counting Tuple Items

- Within this package, create a module named **tuples** with the following within:
  - A function named **mytuplecount**.
    - This function will receive a tuple as parameter
    - This function should return as a **dictionary** the number of occurrences of each element in that tuple.
    - **Ex: {'Bmw':1,250:1,'Audi':2,230:1,'Volvo':1,220:1,240:1}**
    - **Elements can be on any order**

# 9. Working with Sets

## 9.1. Task : Creating a Set

### 9.1.1. Package “istec-p4”

- Use the package you created in the previous session.
- Use the package locally (you don’t need to publish it to everyone)
- Within this package, create a module named **sets** with the following within:
  - A function named **myset**.
    - This function will receive as parameters 2 lists: one with comercial car brands and another with those brands present in Formula 1 races.
      - **Ex: commercial = [“Bmw”,“Audi”,“Volvo”,“Mercedes”,“Ferrari”] , fone = [“Bmw”,“McLaren”,“Mercedes”,“Ferrari”,“Williams”]**
    - The function should return 2 **Sets** with the above data as follows.
      - **Ex: commercial = {“Bmw”,“Audi”,“Volvo”,“Mercedes”,“Ferrari”}**
      - **Ex: fone = {“Bmw”,“McLaren”,“Mercedes”,“Ferrari”,“Williams”}**

## 9.2. Task : Managing Set Items

### 9.2.1. Package “istec-p4”

- Use the package you created in the previous session.
- Use the package locally (you don’t need to publish it to everyone)
- Within this package, create a module named **sets** with the following within:
  - A function named **myset2**.
    - This function will receive as parameters the 2 sets obtained in the previous task.
    - The function should return a **Set** with all the brands selling commercial cars plus the ones that are present in the Formula 1 race.
  - A function named **myset3**.
    - This function will receive as parameters the 2 sets obtained in the previous task.
    - The function should return a **Set** with **only** the brands selling commercial cars and that are also present in the Formula 1 race.
  - A function named **myset4**.
    - This function will receive as parameters the 2 sets obtained in the previous task.
    - The function should return a **Set** with the brands that are present in **only** one of the segments.

## 9.3. Task : Evaluating Multiple Sets

- Within this package, create a module named **sets** with the following within:
- Create a third set containing brands running in the Rally championship - **Ex: rally = {“Audi”,“Mercedes”,“Renault”}**
- Create a a function named **myset5**.
  - This function will receive as parameters the 3 sets (**commercial, fone and rally**)
  - This function should return the brands that are in the Commercial, Formula 1 and Rally segments.

# 10. Working with Decision Structures

## 10.1. Task : Creating a selective List

### 10.1.1. Package “istec-p4”

- Use the package you created in the previous session.
- Use the package locally (you don’t need to publish it to everyone)
- Within this package, create a module named **decisions** with the following within:
  - A function named **mydecision**.
    - This function will receive as parameters 2 lists: one with comercial car brands and another with those brands present in Formula 1 races.
      - **Ex: commercial = [“Bmw”,“Audi”,“Volvo”,“Mercedes”,“Ferrari”] , fone = [“Bmw”,“McLaren”,“Mercedes”,“Ferrari”,“Williams”]**
    - The function should return 2 **Lists** with the brand names in only the **odd/(impares)** positions.

## 10.2. Task : Managing a selective List

### 10.2.1. Package “istec-p4”

- Use the package you created in the previous session.
- Use the package locally (you don’t need to publish it to everyone)
- Within this package, create a module named **decisions** with the following within:
  - A function named **mydecision2**.
    - This function will receive as parameters 2 lists as obtained in the previous task.
    - The function should return a **List** with elements from the 2 lists, where, from the first list you should pick only **even/(pares)** positions and from the second list, only **prime/(primos)** positions.

# 11. Working with Repetition Structures

## 11.1. Task : Creating a selective reversed list

### 11.1.1. Package "istec-p4"

- Use the package you created in the previous session.
- Use the package locally (you don't need to publish it to everyone)
- Within this package, create a module named **repetition** with the following within:
  - A function named **myrepetition**.
    - This function will receive as parameters 2 lists: one with comercial car brands and another with those brands present in Formula 1 races.
      - **Ex: commercial = ["Bmw","Audi","Volvo","Mercedes","Ferrari"] , fone = ["Bmw","McLaren","Mercedes","Ferrari","Williams"]** The function should return 2 **Lists** with the brand names in only the **even/(pares)** poslitions in reverse order.

## 11.2. Task : Managing a selective List

### 11.2.1. Package "istec-p4"

- Use the package you created in the previous session.
- Use the package locally (you don't need to publish it to everyone)
- Within this package, create a module named **repetition** with the following within:
  - A function named **myrepetition2**.
    - This function will receive as parameters 1 list and 1 tupple: one with comercial car brands and another with those brands cars horsepower.
      - **Ex: commercial = ["Bmw","Audi","Volvo","Mercedes","Ferrari"] , hpower = (520,622,640, 675, 618)**.
      - The function should return 1 **dictionary** with the brand names in only the **even/(pares)** positions and if the car horsepower is bigger that 600. The dictionary should contain as values the word **"Big Car"** if the horsepower is bigger that 650, **"Medium Car"** if the horsepower is between 620 and 650 and **"Small Car"** if its below 620.

## 11.3. Task : Managing exceptions

### 11.3.1. Package "istec-p4"

- Use the package you created in the previous session.

- Use the package locally (you don't need to publish it to everyone)
- Within this package, create a module named **repetition** with the following within:
  - A function named **myrepetition3**.
    - This function will receive as parameters 3 lists: one with comercial car brands and another with those brands cars horsepower, and the last one with the number of cylinders.
      - **Ex: commercial = ["Bmw","Audi","Volvo","Mercedes","Ferrari"] , hpower = [520,622,640, 675, 618], cyl=[10,40,0,50,0]**.
      - The function should return 1 **dictionary** with the brand names if the car horsepower is bigger that 600. The dictionary should contain as values the word **"Big Car"** if the horsepower divided by the number of cylinders is bigger that 20, **"Medium Car"** if the division result is between 15 and 20 and **"Small Car"** if it's below 15.

# 12. Working with Functions

- **Obs.** If needed, use previously built functions (in other classes) to execute these tasks.

## 12.1. Task : Creating User Defined Functions

- Use the package you created in the previous sessions.
- Use the package locally (you don't need to publish it to everyone)
- Within this package, create a module named **functions** with the following within:
  - A function named **myudffunction1**.
  - This function will receive two parameters, a list of values **X**, and a value **n** to search for in the list, For example,**X= [400,440,500,550,600,750,600,1200,1110,1000,950], n=600**.
  - The function should return a dictionary with the value **n** and it's **index positions** within the list, if it exists.
  - **Obs.** the above example would return **{"600":[4,6]}**.

```
def myudffunction1(list,n):
    dict = {}
    positions = []
    for index in range(0,len(list)):
        if list[index] == n:
            positions.append(index)
    dict[str(n)]=positions
    return dict

BTC=[400,600,440,500,550,600,750,600,1200,1110,1000,950]
print(myudffunction1(BTC,600))
```

```
{'600': [1, 5, 7]}
```

## 12.2. Task : Creating Lambda Functions

- Within the same module create:
  - Another function named **mylambdafunction1**.
    - This function will receive a single list as parameter: For example,**BTC= [400,440,500,550,600,750,1200,1110,1000,950]**. The function should return only a **List** with the values in only the **even/(pares)** positions in the list (not the indexes).

```
mylambdafunction1 = lambda x: [x[idx] for idx in decisions.odds(len(x))]
print(mylambdafunction1(BTC))
```

- Within the same module create:
  - Another function named **mylambdafunction2**.
    - This function will receive a single list as parameter: For example,**BTC=[400,440,500,550,600,750,1200,1110,1000,950]**.
    - The function should return only a **List** with the values in only the **odd/(impares)** positions in the list (not the indexes).

```
mylambdafunction2 = lambda x: [x[idx] for idx in decisions.evens(len(x)-1)]

print(mylambdafunction2(BTC))
```

## 12.3. Task : Creating Recursive Functions

- Within the same module create:
  - Another function named **myrecursivefunction1**.
    - This function will receive a single parameter **n** and will compute and return the factorial of **n**.

```
def myrecursivefunction1(n):
  if n == 1:
    return 1
  else:
    return (n * myrecursivefunction1(n-1) )

for n in range(1,10):
  print("Factorial of",n,"is:",myrecursivefunction1(n))
```

```
Factorial of 1 is: 1
Factorial of 2 is: 2
Factorial of 3 is: 6
Factorial of 4 is: 24
Factorial of 5 is: 120
Factorial of 6 is: 720
Factorial of 7 is: 5040
Factorial of 8 is: 40320
Factorial of 9 is: 362880
```

- Another function named **myrecursivefunction2**.
- This function will receive a single parameter **n** and will countdown from **n** to **0 (inclusive)** jumping over one value at a time.

```
def myrecursivefunction2(n):
  if n < 1:
    return 0
```

```
    else:
      print(n)
      return (myrecursivefunction2(n-2) )

n=12
print("Countdown from",n)
```

```
Countdown from 12
```

```
myrecursivefunction2(n)
```

```
12
10
8
6
4
2
0
```

- Another function named **myrecursivefunction3**.
- This function will receive a single **List** of numbers and will compute and return the multplication of all of them.

```
def myrecursivefunction3(list):
    numelements = len(list)
    if numelements == 1:
      return list[0]
    else:
      last = list[numelements-1]
      list.pop(numelements-1)
      return ( last * myrecursivefunction3(list) )

list=[1,2,3,4,5,6,7]
print("Multiplication of",list)
```

```
Multiplication of [1, 2, 3, 4, 5, 6, 7]
```

```
print(myrecursivefunction3(list))
```

```
5040
```

- Another function named **myrecursivefunction4**.
- This function will receive a single **List** of numbers and will compute and return the multplication of all the **odd/(impares)** positions in the list (not the indexes).

```
def myrecursivefunction4(list):
    return ( myrecursivefunction3(list[0::2]) )

list=[1,2,3,4,5,6,7]
print("Multiplication of",list)
```

```
Multiplication of [1, 2, 3, 4, 5, 6, 7]
```

```
print(myrecursivefunction4(list))
```

```
105
```

- **Obs.** It must be done using recursive functions (functions that call themselves).

# 13. Working with Classes

- **Obs.** If needed, use previously built functions (in other classes) to execute these tasks.

## 13.1. Task : Creating a Class

- Use the package you created in the previous sessions.
- Use the package locally (you don't need to publish it to everyone)
- Within this package, create a module named **classes** with the following within:
  - A class named **Student**.
  - A student might have properties like: **name, age, classes he is attending, exam dates on all classes, grades for those classes and books needed for each class**

## 13.2. Task : Creating an Object

- Crerate a student with the following properties:
  - Name : Your name
  - Age : Your Age
  - Attending 3 classes : Prog. 4, Data Science, Networking
  - Exams Dates: Prog 4 is 21 June, Data Science is 20 June, Networking is 28 June
  - There are no grades yet
  - The blibliography is : For Prog 4 is book A.pdf and B.pdf, for Data Science is DS.pdf and there are no books for Networking

## 13.3. Task : Creating Methods

- The student decided to quit from Networking.
- The student decided to enter the Cloud Computing class.
- The student has now the grade for Prog 4, it was 15.
- The student has now the grade for Data Science, it was 18.
- Print all the classes the student is attending.
- Print all the grades the student has so far.

# Part II.

# Solutions

# 14. Virtual Environments

## 14.1. Task : Manage Python Virtual Environments

These solutions are assuming the usage of macOS and Visual Studio Code

### 14.1.1. Virtual Environment 1

Create a folder for this work called **istec** and move into it. Make sure you know where you are positioned.

```
mkdir istec
```

- Positioned in folder **istec**
- Create a Python environment named **env1**

```
python -m venv env1
```

- Activate the environment named **env1**

```
source env1/bin/activate
```

- Install package **requests** version **2.18.4** in this environment

```
pip install requests==2.18.4
```

> **Warning**
>
> It's possible that you have a message saying to consider to upgrade **pip**. You may do it or not.

- List the existing packages in this environment

```
pip list
```

You may have a similar output

```
Package    Version
---------- ---------
certifi    2022.12.7
chardet    3.0.4
idna       2.6
pip        23.0.1
requests   2.18.4
```

```
setuptools 58.1.0
urllib3    1.22
```

Confirm the package **requests** version **2.18.4** is installed. Some other packages were installed because **requests** needs them.

- Open you **Visual Studio Code** editor and open folder **istec**. It should have only **env1**.
- **Visual Studio Code** may detect you have a virtual environment already there.
- Select that first environment **env1**.
- Create a new file as a **Notebook (*.ipynb)** called **FirstNotebook.ipynb** in this environment
  - Add in the Notebook, as the first block, the following **Markdown**

```
# My First Notebook
```

- Add as the next block, the following **Code**

```
print("I now know about Markdown & Code blocks.")
print("I also created my first Python environment.")
```

- Execute the Notebook
- If needed, select a **Python Kernel and Environment (env1)** to execute the Notebook
- Install any software that **Visual Studio Code** asks you

### 14.1.2. Virtual Environment 2

- Positioned in folder **istec**
- Create a Python environment named **env2**

```
python -m venv env2
```

- Activate the environment named **env2**

```
source env2/bin/activate
```

- Install package **requests** version **2.18.4** in this environment

```
pip install requests==2.18.4
```

> **Warning**
>
> It's possible that you have a message saying to consider to upgrade **pip**. You may do it or not.

- List the existing packages in this environment

```
pip list
```

You may have a similar output

```
Package    Version
---------- ---------
certifi    2022.12.7
chardet    3.0.4
idna       2.6
pip        23.0.1
requests   2.18.4
setuptools 58.1.0
urllib3    1.22
```

Confirm the package **requests** version **2.18.4** is installed. Some other packages were installed because **requests** needs them.

- In **Visual Studio Code** editor open folder **istec**.
- Now, when selecting a new Kernel, you should have **env1** and **env2** to select.

### 14.1.3. Virtual Environment 3

- Positioned in folder **istec**
- Create a Python environment named **env3**

```
python -m venv env3
```

- Activate the environment named **env3**

```
source env3/bin/activate
```

- Install package **requests** with the latest version in this environment

```
'''
Note the omission of a version.
It's installing the latest available.
'''
pip install requests
```

> **Warning**
>
> It's possible that you have a message saying to consider to upgrade **pip**. You may do it or not.

- List the existing packages in this environment

```
pip list
```

You may have a similar output

```
Package    Version
---------- ---------
certifi            2022.12.7
charset-normalizer 3.1.0
idna               3.4
pip                23.0.1
requests           2.28.2
setuptools         58.1.0
urllib3            1.26.15
```

Confirm the package **requests** version is different from **2.18.4**. Some other packages were installed because **requests** needs them.

- In **Visual Studio Code** editor open folder **istec**.
- Now, when selecting a new Kernel, you should have **env1**, **env2** and **env3** to select.

# 15. Projects Structure

## 15.1. Task : Manage Python Projects Structure

### 15.1.1. Project 1

- Create a folder named **project1** which will use the environment/kernel named **env1**

```
mkdir project1
```

- Use the correct naming for your **project, packages and modules**.
- Create a package named **istec.p4.exercises**

```
mkdir istec
```

```
cd istec
```

```
mkdir p4
```

```
cd p4
```

```
mkdir exercises
```

“‘{python} #| eval: false cd exercises

> **Note**
>
> You may create the above folders using the code editor or the operating system file explorer, but for that, you should not need a guide.

- To tell Python that you are creating a package, you have to put a file named **___init___.py** on every single folder of that package.
- Within this package, in folder **exercises** create a module (which is a Python file) named **exercise1.py**
- Your folder tree should now be similar to this when looked from inside folder **project1**

```
.
   istec
       __init__.py
       p4
           __init__.py
           exercises
```

```
            __init__.py
            exercise1.py
```

- Inside this module, create a basic function named **hello** to execute **print("Hello World!!")**

```
def hello():
    print("Hello World!!")

# This is for demo purposes only.
# There is no need to call the function here.
# Unless you want to see the output of it right away.
hello()
```

- Execute your program

### 15.1.2. Project 2

- Start from **project1** code and duplicate it as **project2**
- Rename file **exercise1.py** to **exercise2.py**
- Your folder tree should now be similar to this when looked from inside folder **project2**

```
.
  istec
      __init__.py
      p4
          __init__.py
          exercises
              __init__.py
              exercise2.py
```

- Update your code in **exercise2.py** to produce an output similar to this:
  - **Hello World!!**
  - **I'm running using virtual environment: env1.** *(or)*
  - **I'm running using virtual environment: env2.** *(for when you are running it using another virtual environment)*
- In file **exercise2.py** update the function **hello**

```
import sys

def hello():
    print("Hello World!!")
    print("I'm running using virtual environment:", sys.prefix )

# This is for demo purposes only.
# There is no need to call the function here.
# Unless you want to see the output of it right away.
```

```
hello()
```

- Execute your program using environment/kernel **env1** and then with environment/kernel **env2**
- Output of using **env1**.

```
Hello World!!
I'm running using virtual environment: /Users/joaocaldeira/istec/env1
```

- Output of using **env2**.

```
Hello World!!
I'm running using virtual environment: /Users/joaocaldeira/istec/env2
```

- Confirm that the outputs should be different based on the selected environment.
- At the end of these exercises, your **istec** folder should have the following structure.

```
env1
env2
env3
project1
    istec
        __init__.py
        p4
            __init__.py
            exercises
              __init__.py
              exercise1.py
project2
    istec
        __init__.py
        p4
            __init__.py
            exercises
                __init__.py
                exercise2.py
```

# 16. Building Packages

## 16.1. Task : Create a Python Package

### 16.1.1. Build Package "istec-p4"

- Create a project/package named **istec-p4** which will use any of your existing virtual environments.
- Use the correct naming for your project, packages and modules.
- Create a package with the following structure **istec-p4/exercises**.
- Within this package, create a module named **greetings** with the following within:
  - A function named **hello** to execute **print("Hello World!!")**
  - A function named **good_morning** to execute **print("Good Morning!!")**
  - A function named **good_afternoon** to execute **print("Good Afternoon!!")**

### 16.1.2. Install your Package

- Make the necessary configurations to make your package named **istec-p4** locally instalable.
- Install your package locally in the virtual environment you are using.
- To be locally instalable means, you will install your package in any of your virtual environments, and then, from any other project using the same environment, you can call your package/modules like this:

```
from istec.p4.exercises import greetings
```

- Once you load your package then you can call the existing functions within the package.

# 17. Publishing Packages

## 17.1. Task : Publish a Python Package

### 17.1.1. Package “istec-p4”

- Use the package you created in the previous task and make it public.
- Publish your package to the **TestPyPi** repository.
  - Navigate to **https://test.pypi.org/** and Register yourself as an user.
- This means your package will be available for everyone (worldwide) to install:

### 17.1.2. Using Package “istec-p4”

- Later, to install from **TestPyPi** you will issue this command:

```
pip install --index-url https://test.pypi.org/simple/ istec-p4
```

- Later, to install from **PyPi** you will issue this command:

```
pip install istec-p4
```

- Then you can load your package and call the existing functions within that package/module.

```
from istec.p4.exercises import greetings

# Calling the hello function
greetings.hello()
```

> **Note**
>
> The official **Python** packages repository is called **PyPi (https://pypi.org/)**.

# 18. Data Types

## 18.1. Task : Return changed size strings

## 18.2. Task : Return string parts

- File **datatypes.py**

```
#This function will receive as an input a string and return the same string in uppercase.
def myupper(str):
    return str.upper()

#This function will receive as an input a string and return the same string in lowercase.
def mylower(str):
    return str.lower()

#This function will receive as an input a string and if the string is in uppercase, it wi
def invertsizing(str):
    if(str.lower() == str):
        return str.upper()
    else:
        return str.lower()

def stringpart(st,n):
    return st[0 : n]
```

# 19. Working with Lists

## 19.1. Task : Creating Lists

## 19.2. Task : Changing List Items

## 19.3. Task : Add and Remove List Items

- File **lists.py**

```
#This function will receive as an input a string and return each character in a list.
def mylist(str):
    list = []
    for char in str:
        list.append(char)
    return list

#This function will receive as an input a number and return a list of numbers from 0 to t
def mylistnum(n):
    numbers= []
    for x in range(0, n+1):
        numbers.append(x)
    return numbers

def myfirstnames(list):
    for name in range(len(list)):
        list[name] = list[name][0].upper() + list[name][1:]
    return list

def myaddremove(list, str, str2):
    list.append(str)
    list.remove(str2)
    return list
```

# 20. Working with Dictionaries

## 20.1. Task : Creating a Dictionary

## 20.2. Task : Changing Dictionary Items

- File **dictionaries.py**

```
def mydictionary(list, list2):
    dictionary = {}
    for x in range(len(list)):
        dictionary[list[x]] = list2[x]
    return dictionary

def mydictionary2(dictionary, list, list2):
    for x in range(len(list)):
        dictionary[list[x]] = list2[x]
    return dictionary
```

# 21. Working with Tuples

## 21.1. Task : Creating a Tuple

## 21.2. Task : Evaluating Tuple Items

## 21.3. Task : Counting Tuple Items

- File **tuples.py**

```
def mytuple(list, list2):
    elements = []
    for x in range(max(len(list),len(list2))):
        if x < len(list):
            elements.append(list[x])
        if x < len(list2):
            elements.append(list2[x])
    return tuple(elements)


def mytuple2(list, list2):
    return mytuple(list, list2)

def mytuplecount(elements):
    dictionary = {}
    for x in range(len(elements)):
        dictionary[elements[x]] = elements.count(elements[x])
    return dictionary
```

# 22. Working with Sets

## 22.1. Task : Creating a Set

## 22.2. Task : Managing Set Items

## 22.3. Task : Evaluating Multiple Sets

- File **sets.py**

```
def myset(list, list2):
    return set(list),set(list2)


def myset2(sets):
    return  set.union(sets[0],sets[1])


def myset3(sets):
    return  set.intersection(sets[0],sets[1])


def myset4(sets):
    return  myset2((set.difference(sets[0],sets[1]),set.difference(sets[1],sets[0])))


def myset5(sets):
    return myset3((myset3((sets[0][0],sets[0][1])), sets[1]))
```

# 23. Working with Decision Structures

## 23.1. Task : Creating a selective List

## 23.2. Task : Managing a selective List

- File **decisions.py**

```
def mydecision(list1,list2):
    return [list1[index] for index in evens(len(list1))], [list2[index] for index in even

def mydecision2(list1,list2):
    return [list1[index] for index in odds(len(list1))], [list2[index-1] for index in pri


def odds(end):
    odd = []
    for x in range(0,end):
        if x % 2 != 0:
            odd.append(x)
    return odd

def evens(end):
    even = []
    for x in range(0,end):
        if x % 2 == 0:
            even.append(x)
    return even


def isprime(num):
    if num < 2:
        return False

    for i in range(2, num):
        if (num % i) == 0:
            #print(i)
            break
    else:
        return True
    return False


def primes(end):
```

```
    prime = []
    for x in range(0,end):
        if x % 2 != 0:
            if isprime(x):
                prime.append(x)
    return prime


def indexes(end):
    odd = odds(end)
    even = evens(end)
    prime = primes(end)

    return odd, even, prime
```

# 24. Working with Repetition Structures

## 24.1. Task : Creating a selective reversed list

## 24.2. Task : Managing a selective List

## 24.3. Task : Managing exceptions

- File **repetitions.py**

```
from istec.p4.exercises import decisions

bigcar = "Big Car"
mediumcar= "Medium Car"
smallcar = "Small Car"

def myrepetition(list1,list2):

    rlist1 = [list1[index] for index in decisions.odds(len(list1))]
    rlist2 = [list2[index] for index in decisions.odds(len(list2))]
    rlist1.reverse()
    rlist2.reverse()

    return rlist1,rlist2

def myrepetition2(list,tuple):
    dictionary = {}
    for x in decisions.odds(len(list)):
        if tuple[x] <= 600:
            continue
        if tuple[x] > 650:
            dictionary[list[x]] = bigcar
        elif tuple[x] > 620:
             dictionary[list[x]] = mediumcar
        else:
            dictionary[list[x]] = smallcar

    return dictionary


def myrepetition3(list1,list2,list3):
    dictionary = {}
    for x in range(len(list1)):
```

```
        if list2[x] <= 600:
            continue
        try:
            size = list2[x]/list3[x]
        except:
            size = 0
        finally:
            if size > 20:
                dictionary[list1[x]] = bigcar
            elif size > 15:
                dictionary[list1[x]] = mediumcar
            elif size > 0:
                dictionary[list1[x]] = smallcar
            else:
                dictionary[list1[x]] = "Unknown"

    return dictionary
```

# 25. Working with Functions

## 25.1. Task : Creating User Defined Functions

## 25.2. Task : Creating Lambda Functions

## 25.3. Task : Creating Recursive Functions

- File **functions.py**

```
from istec.p4.exercises import decisions

def hello():
    return "Hello from Functions!!"


def myudffunction1(list,n):
    dict = {}
    positions = []
    for index in range(0,len(list)):
        if list[index] == n:
            positions.append(index)
    dict[str(n)]=positions
    return dict



mylambdafunction1 = lambda x: [x[idx] for idx in decisions.odds(len(x))]

mylambdafunction2 = lambda x: [x[idx] for idx in decisions.evens(len(x)-1)]


def myrecursivefunction1(n):
  if n == 1:
    return 1
  else:
    return (n * myrecursivefunction1(n-1) )



def myrecursivefunction2(n):
  if n < 1:
    return 0
```

```
  else:
    print(n)
    return (myrecursivefunction2(n-2) )


def myrecursivefunction3(list):
    numelements = len(list)
    if numelements == 1:
      return list[0]
    else:
      last = list[numelements-1]
      list.pop(numelements-1)
      return ( last * myrecursivefunction3(list) )


def myrecursivefunction4(list):
    return ( myrecursivefunction3(list[0::2]) )

# [0::2] Only work in generating indexes on lists, nothing else.
```

# 26. Working with Classes

- **Obs.** If needed, use previously built functions (in other classes) to execute these tasks.

## 26.1. Task : Creating a Class

- File **oop.py**

```
class Student:

  def __init__(self,name,age,classes):
    self.name = name
    self.age = age
    self.classes = classes

  def __str__(self):
    return f"{self.name} is {self.age} years old"
```

## 26.2. Task : Creating an Object

- File **main.py**

```
from istec.p4.exercises.oop import Student

disciplinas = {"Prog4":["26/07",0,["Book1,pdf","Book2.pdf"]],
        "DS":["25/08",0,["Book3,pdf","Book4.pdf"]],
        "Networking":["26/09",0,["Book5,pdf","Book6.pdf"]]}

student1 =Student("Joao", 35, disciplinas)

print(student1.name)

print(student1.age)

print(student1.classes)

print(student1)
```

## 26.3. Task : Creating Methods

- File **oop.py**

```
class Student:

  def __init__(self,name,age,classes):
    self.name = name
    self.age = age
    self.classes = classes

  def __str__(self):
    return f"{self.name} is {self.age} years old"

  def adddisciplines(self,disciplines):
    for x in disciplines.keys():
        self.classes[x] = disciplines[x]

  def removedisciplines(self,disciplines):
    for x in disciplines:
        self.classes.pop(x)

  def alldisciplines(self):
    disciplines = []
    for x in self.classes.keys():
        disciplines.append(x)
    return disciplines

  def addgrades(self,disciplines):
    for x in disciplines:
        self.classes[x][1]  = disciplines[x]

  def allgrades(self):
    grades = {}
    for x in self.classes.keys():
       if self.classes[x][1] > 0:
        grades[x] = self.classes[x][1]
    return grades
```

# 27. Main Program (For Test Purposes)

- File **main.py**
- This file is used to test the packages and modules we build.

```
from istec.p4.exercises import exercise1,greetings
from istec.p4.exercises import lists,datatypes,dictionaries
from istec.p4.exercises import tuples,sets
from istec.p4.exercises import decisions,repetitions,functions

# These are just the first exercises
print("Working with Python Packages")
exercise1.hello()
greetings.good_morning()
greetings.good_afternoon()

# Exercises with Strings
print("Working with Strings")
mystring = "Sport Lisboa e Benfica"
print(datatypes.stringpart(mystring,12))
print(datatypes.myupper(mystring))
print(datatypes.mylower(mystring))
print(datatypes.invertsizing(datatypes.myupper(mystring)))
print(datatypes.invertsizing(datatypes.mylower(mystring)))

# Exercises with Lists
print("Working with Lists")
mystring = "a caminho do Marquês"
print(lists.myfirstnames(mystring.split()))
print(lists.mylist(mystring))
print("Dá-me o ",lists.mylistnum(38))

last2champs = "Porto Porto" # This is a string later converted to a list of words
#last2champs = ["Porto",Porto"] # This is the same data as above but already as a list

print(lists.myaddremove(last2champs.split(), "Benfica", "Porto")) # Function split() conv

# Exercises with Dictionaries
print("Working with Dictionaries")
teams = ["Benfica","Porto","Braga","Sporting","Arouca"]
points = [80,73,71,64,48]
print(dictionaries.mydictionary(teams,points))

championscup = {'Benfica': 80, 'Porto': 73, 'Braga': 71}
europecup = ["Sporting","Arouca"]
```

```
points = [64,48]
print(dictionaries.mydictionary2(championscup,europecup,points))


# Exercises with Tuples
print("Working with Tuples")
cars = ["Bmw","Audi","Volvo"]
speeds = [250,230,220]
print(tuples.mytuple(cars,speeds))

cars.append("Audi")
speeds.append(240)
print(tuples.mytuple2(cars,speeds))

print(tuples.mytuplecount(tuples.mytuple2(cars,speeds)))

# Exercises with Sets
print("Working with Sets")
commercial = ["Bmw","Audi","Volvo","Mercedes","Ferrari"]
fone = ["Bmw","McLaren","Mercedes","Ferrari","Williams"]

print(sets.myset(commercial,fone))
print(sets.myset2(sets.myset(commercial,fone)))
print(sets.myset3(sets.myset(commercial,fone)))
print(sets.myset4(sets.myset(commercial,fone)))


rally = {"Audi","Mercedes","Renault"}
print(sets.myset5((sets.myset(commercial,fone),rally)))


# Working with Decisions
print("Working with Decisions")
print(decisions.mydecision(commercial,fone))
print(decisions.mydecision2(commercial,fone))


# Working with Repetitions
print("Working with Repetitions")
print(repetitions.myrepetition(commercial,fone))

hpower = (520,622,640, 675, 618)
print(repetitions.myrepetition2(commercial,hpower))

cyl=[10,40,0,50,0]
print(repetitions.myrepetition3(commercial,hpower,cyl))

a = ["A","A",23,24,25]
b = ["A","B","A","C","D","A"]
print(tuples.mytuplecount(tuples.mytuple2(a , b))["A"])
```

```
# Working with Functions
# Working with User-Defined Functions
print("Working with User-Defined Functions")
BTC=[400,600,440,500,550,600,750,600,1200,1110,1000,950]
print(functions.myudffunction1(BTC,600))

# Working with Lambda Functions
print("Working with Lambda Functions")
print(functions.mylambdafunction1(BTC))
print(functions.mylambdafunction2(BTC))

# Working with Recursive Functions
print("Working with Recursive Functions")
for n in range(1,10):
  print("Factorial of",n,"is:",myrecursivefunction1(n))

n=12
print("Countdown from",n)
myrecursivefunction2(n)

list=[1,2,3,4,5,6,7]
print("Multiplication of",list)
print(myrecursivefunction3(list))

print("Multiplication of odd positions in ",list)
print(myrecursivefunction4(list))

# Working with Object Oriented Programming

# Working with Classes
print("Working with Classes")

from istec.p4.exercises.oop import Student

disciplinas = {"Prog4":["26/07",0,["Book1,pdf","Book2.pdf"]],
        "DS":["25/08",0,["Book3,pdf","Book4.pdf"]],
        "Networking":["26/09",0,["Book5,pdf","Book6.pdf"]]}
student1 = Student("Joao", 35, disciplinas)
print(student1.name)
print(student1.age)
print(student1.classes)
print(student1)


# Working with Methods
print("Working with Methods")

student1.removedisciplines(["Networking"])
print(student1.classes)
```

```
newdisciplines = {"Engenharia de Software":["2/07",0,["Book52,pdf","Book62.pdf"]],
                  "Multimedia":["2/07",0,["Book50,pdf","Book60.pdf"]]}

student1.adddisciplines(newdisciplines)
print(student1.classes)

newgrades = {"Engenharia de Software":15, "Multimedia":16}
student1.addgrades(newgrades)

print(student1.classes)

print(student1.alldisciplines())

print(student1.allgrades())


disciplinas = {"Cloud Computing":["26/07",0,["Book1,pdf","Book2.pdf"]]}
student2 = Student("Luis",25, disciplinas)
print(student2.name)
print(student2.age)
print(student2.classes)

print(student1)
print(student1.classes)
print(student2)
print(student2.classes)
```

- Your folder tree should now be similar to this when looked from inside folder **project1**

```
main.py
istec
    __init__.py
    p4
        __init__.py
        exercises
            __init__.py
            datatypes.py
            decisions.py
            dictionaries.py
            exercise1.py
            functions.py
            greetings.py
            lists.py
            oop.py
            repetitions.py
            sets.py
            tuples.py
setup.py
```

# 28. Test Solutions (19-May-2023)

## 28.1. Practical Questions Solution

1) As criptomoedas são um dos ativos digitais mais falados no momento. O valor das criptomoedas ao longo do tempo são fundamentalmente uma série temporal que pode ser representada por uma lista de valores. Os valores da Bitcoin por exemplo podem ser representados de forma ordenada da seguinte maneira : BTC = [232, 601, 675, 798, 799, 849, 929, 948, 953, 959, 1051, 1072, 1099, 1151, 1169, 1230, 1355, 1376, 1479, 1877].

```
BTC = [232, 601, 675, 798, 799, 849, 929, 948, 953, 959, 1051, 1072, 1099, 1151, 1169, 12
```

a) Que código Python utilizaria para obter o valor 929 desta lista ?

```
BTC[6]
```

```
929
```

b) Que código Python utilizaria para adicionar o valor 950 nesta lista mantendo a série ordenada ?

```
BTC.insert(8,950)
BTC # Only needed to print the result
```

```
[232, 601, 675, 798, 799, 849, 929, 948, 950, 953, 959, 1051, 1072, 1099, 1151, 1169, 1230,
```

c) Que código Python utilizaria para apagar o primeiro valor da lista ?

```
BTC.pop(0)
```

```
232
```

```
BTC # Only needed to print the result
```

```
[601, 675, 798, 799, 849, 929, 948, 950, 953, 959, 1051, 1072, 1099, 1151, 1169, 1230, 1355,
```

d) Que código Python utilizaria para adicionar um novo valor no final da lista ?

```
BTC.append(1900)
BTC # Only needed to print the result
```

```
[601, 675, 798, 799, 849, 929, 948, 950, 953, 959, 1051, 1072, 1099, 1151, 1169, 1230, 1355,
```

2) Se tivermos várias listas de valores de criptomoedas, pode ser extremamente importante guardar esses valores, para, caso seja necessário, representá-los num gráfico. Como poderá ter várias criptomoedas, um dicionário pode ser uma estrutura de dados adequada para esse efeito.

a) Escreva o código Python que cria um dicionário que contém as seguintes criptomoedas e os seus respectivos valores: a lista de valores atrás referida para a BTC, mais os valores [0.3,0.31,0.4,0.37,0.34,0.39,0.42] para o XRP, e ainda para o ETH os valores [1200,1220,1190,1050].

```
cryptocoins = {"BTC": BTC,"XRP": [0.3,0.31,0.4,0.37,0.34,0.39,0.42],"ETH": [1200,1220,119
type(cryptocoins) # Just to confirm that the type is a dictionary
```

```
<class 'dict'>
```

```
cryptocoins # Only needed to print the result
```

```
{'BTC': [601, 675, 798, 799, 849, 929, 948, 950, 953, 959, 1051, 1072, 1099, 1151, 1169, 123
```

b) Que código Python utilizaria para saber apenas quais os nomes das criptomoedas que estão guardadas neste dicionário ?

```
cryptocoins.keys()
```

```
dict_keys(['BTC', 'XRP', 'ETH'])
```

c) Que código Python utilizaria para remover os valores do XRP desse diccionário ?

```
cryptocoins.pop("XRP")
```

```
[0.3, 0.31, 0.4, 0.37, 0.34, 0.39, 0.42]
```

```
cryptocoins # Only needed to print the result
```

```
{'BTC': [601, 675, 798, 799, 849, 929, 948, 950, 953, 959, 1051, 1072, 1099, 1151, 1169, 123
```

a) Crie um tuplo apenas com os nomes das criptomoedas atrás referidas.

```
cryptonames = ("BTC","XRP","ETH")
cryptonames # Only needed to print the result
```

```
('BTC', 'XRP', 'ETH')
```

b) De acordo com esse tuplo, escreva o código que traria o elemento representando a BTC.

```
cryptonames.index("BTC")
```

```
0
```

a) Tendo os seguintes valores da Bitcoin e Ethereum, respectivamente BTC - 650, 700, 800, 1000 e ETH - 350, 400, 700, 720, 750, que código Python faria para identificar os valores comuns às duas criptomoedas ?

```
bitcoin = { 650, 700, 800, 1000 }
ethereum = { 350, 400, 700, 720, 750 }
bitcoin.intersection(ethereum)
```

```
{700}
```

```
ethereum.intersection(bitcoin) # You can also do it this way
```

```
{700}
```

b) Que código faria para inserir mais valores de BTC ?

```
bitcoin.add(1100)
bitcoin # Only needed to print the result
```

```
{800, 1000, 650, 1100, 700}
```