Lab 1: Python Basics

The objective of this notebook is to learn how to use Jupyter Notebook and to start writing simple Python code. You can find a Python guide at this link.

Outline

1. Variables and Basic Data Types

Exercise 1.1

Print in output the following text: `Hello World`.

In [1]:
```
#### START CODE HERE (~ 1 line) ####
print("Hello World")
#### END CODE HERE ####
```

```
Hello World
```

Expected output

`Hello World`

Now print in output the number 100.

In [2]:
```
#### START CODE HERE (~ 1 line) ####
print(100)
#### END CODE HERE ####
```

```
100
```

Expected output

`100`

Exercise 1.2

Define a variable `x` with the value 100, then print the value of `x`. Remember that variable names should:

- Start with a letter or an underscore.
- Only contain letters, numbers, and some special characters (e.g., underscores).

In [3]:
```
#### START CODE HERE (~ 2 lines) ####
x=100
print(x)
#### END CODE HERE ####
```

```
100
```

Expected output

`100`

Exercise 1.3

Now, print the **type** of the variable `x`.

▶ **Hints**

In [4]:
```
#### START CODE HERE (~ 1 line) ####
print(type(x))
#### END CODE HERE ####
```

```
<class 'int'>
```

Expected output

`<class 'int'>`

Exercise 1.4

Define a variable `s` containing the string `"hello world"`, and print the **value** of `s` and its **type**.

▶ **Hints**

In [5]:
```
#### START CODE HERE (~ 3 lines) ####
s = "hello world"
print(s)
print(type(s))
#### END CODE HERE ####
```

```
hello world
<class 'str'>
```

Expected output

`hello world`
`<class 'str'>`

Exercise 1.5

Define a variable `pi` containing the number 3.14, then:

- Print the **value** and the **type** of pi.
- Cast the type of pi into **int**.
- Print the **value** and the **type** of pi again.

In [6]:
```
#### START CODE HERE (~ 6 lines) ####
pi = 3.14
print(pi)
print(type(pi))
pi = int(pi)
print(pi)
print(type(pi))
#### END CODE HERE ####
```

```
3.14
<class 'float'>
3
<class 'int'>
```

Expected output

`3.14`
`<class 'float'>`
`3`
`<class 'int'>`

Notice that the **int() cast** function converts the float number 3.14 into 3 when casting the variable to int.

Exercise 1.6

Define a variable `x` containing the number 10, then:

- Create a new variable y and assign to it the value of x.
- Print the values of x and y.
- Assign to x the value 100.
- Print the values of x and y.

In [7]:
```
#### START CODE HERE (~ 7 lines) ####
x = 10
y = x
print(x)
print(y)
x = 100
print(x)
print(y)
#### END CODE HERE ####
```

```
10
10
100
10
```

Expected output

`10`
`10`
`100`
`10`

Notice that the previous code assigns the value of x to y. But they are **two distinct variables** that point to different memory spaces. Therefore, when the value of x is changed, the value of y is not affected.

2. Basic Math operands

Exercise 2.1

Define a variable `x` containing the number 10, then:

- Print the **value** of x
- **Increment** the value of x by 100
- Print the new **value** of x
- **Decrement** the value of x by 1
- Print the new **value** of x

▶ **Hints**

In [8]:
```
#### START CODE HERE (~ 6 lines) ####
x = 10
print(x)
x += 100
print(x)
x -= 1
print(x)
#### END CODE HERE ####
```

```
10
110
109
```

Expected output

`10`
`110`
`109`

Exercise 2.2

Define a variable `x` containing the number 10 and a variable `y` containing the number 200, then:

- Create a new variable z containing the value of $x + y$
- Print the **value** and the **type** of z

In [9]:
```
#### START CODE HERE (~ 5 lines) ####
x = 10
y = 200
z = x + y
print(z)
print(type(z))
#### END CODE HERE ####
```

```
210
<class 'int'>
```

Expected output

`210`
`<class 'int'>`

Exercise 2.3

Define a variable `x` containing the number 10 and a variable `y` containing the number 200, then:

- Create a new variable z containing the **average** of x and y
- Print the **value** and the **type** of z

▶ **Hints**

In [10]:
```
#### START CODE HERE (~ 5 lines) ####
x = 10
y = 200
z = (x + y) / 2
print(z)
print(type(z))
#### END CODE HERE ####
```

```
105.0
<class 'float'>
```

Expected output

`105`
`<class 'float'>`

Notice that the **type** of the variable `z` is changed to **float**.

3. String manipulation

In [11]:
```
s1 = "This is a simple string"
print(s1)
s2 = 'This is another string, but using quote instead of double quote'
print(s2)
```

```
This is a simple string
This is another string, but using quote instead of double quote
```

In [12]:
```
s = """This is a
multiline
string"""

print(s)
```

```
This is a
multiline
string
```

In [13]:
```
i=5
s="5"
print(i, type(i))
print(s, type(s))
print(i + 1)
print(s + 1)
```

```
5 <class 'int'>
5 <class 'str'>
6
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[13], line 6
      4 print(s, type(s))
      5 print(i + 1)
----> 6 print(s + 1)

TypeError: can only concatenate str (not "int") to str
```

`5` is the number 5 (int). In contrast, `"5"` is the character 5 (string). If you want to use `"5"` for math operations, you must convert `"5"` to int or float.

In [14]:
```
i=5
s="5"
print(i, type(i))
print(s, type(s))
print(i + 1)
print(int(s) + 1) # cast the character "5" to int
```

```
5 <class 'int'>
5 <class 'str'>
6
6
```

Exercise 3.1

Define a variable `s1` containing the string `"this is a string"` and a variable `s2` containing `", this is another string"`. Then, create a third variable `s3` by **concatenating** `s1` and `s2`. Print the value of `s3`.

▶ **Hints**

In [15]:
```
#### START CODE HERE (~ 4 lines) ####
s1 = "this is a string"
s2 = ", this is another string"
s3 = s1 + s2
print(s3)
#### END CODE HERE ####
```

```
this is a string, this is another string
```

Expected output

`this is a string, this is another string`

Exercise 3.2

Replace each occurrence of the substring `"four"` with the other substring `"three"` from the variable `s`, and assign the resulting string to the same variable. Then, print the new value of `s`.

▶ **Hints**

In [16]:
```
s = "I bought four apples, two bananas, and four oranges"
print(s)
#### START CODE HERE (~ 2 lines) ####
s = s.replace("four", "three")
print(s)
#### END CODE HERE ####
```

```
I bought four apples, two bananas, and four oranges
I bought three apples, two bananas, and three oranges
```

Expected output

`I bought four apples, two bananas, and four oranges`
`I bought three apples, two bananas, and three oranges`

4. List manipulation

Lists are sequences of any type of value. They are defined by **square brackets**, and the elements are separated by **commas**. The elements in a list are **ordered**.

Exercise 4.1

Create a **list** named `my_list` containing the following elements: 1, 2, 3 (as integer). Then, print the values of the list.

In [17]:
```
#### START CODE HERE (~ 2 lines) ####
my_list = [1, 2, 3]
print(my_list)
#### END CODE HERE ####
```

```
[1, 2, 3]
```

Expected output

`[1, 2, 3]`

Exercise 4.2

Change the first element of `my_list` with the value 10. Then, print the new values of the list.

▶ **Hints**

In [18]:
```
my_list = [1, 2, 3]
print(my_list)

#### START CODE HERE (~ 1 line) ####
my_list[0] = 10
#### END CODE HERE ####

print(my_list)
```

```
[1, 2, 3]
[10, 2, 3]
```

Expected output

`[1, 2, 3]`
`[10, 2, 3]`

Exercise 4.3

Print the **first 10 elements** of the following list. Then, print the **last two elements**. Finally, print the elements from **5 to 10** (both included).

▶ **Hints**

In [19]:
```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9 , 10, 11, 12, 13, 14, 15]
```

```
#### START CODE HERE (~ 3 lines) ####
print(my_list[:10])
print(my_list[-2:])
print(my_list[4:10])
#### END CODE HERE ####
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[14, 15]
[5, 6, 7, 8, 9, 10]
```

Expected output

`[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
`[14, 15]`
`[5, 6, 7, 8, 9, 10]`

Exercise 4.4

Concatenate the two lists `l1` and `l2` into a new variable `my_list`. Then, print the **length** of the new list (i.e., the number of elements).

▶ **Hints**

In [20]:
```
l1 = ["one", "two"]
l2 = [1, 2]

#### START CODE HERE (~ 1 line) ####
my_list = l1 + l2
print(len(my_list))
#### END CODE HERE ####
print(my_list)
```

```
4
['one', 'two', 1, 2]
```

Expected output

`4`
`['one', 'two', 1, 2]`

Exercise 4.5

Add at the end of `my_list` a new element 5 (int).

▶ **Hints**

In [21]:
```
my_list = [1, 2, 3, 4]
print(my_list)
#### START CODE HERE (~ 1 line) ####
my_list.append(5)
#### END CODE HERE ####
print(my_list)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
```

Expected output

`[1, 2, 3, 4]`
`[1, 2, 3, 4, 5]`

Exercise 4.6

Order the following list `my_list` in **ascending** order.

▶ **Hints**

In [22]:
```
my_list = [100, 12, 72, 33, 99, 24, 49, 1, 15, 50]
print(my_list)

#### START CODE HERE (~ 1 line) ####
my_list = sorted(my_list) # alternative: my_list.sort()
#### END CODE HERE ####

print(my_list)
```

```
[100, 12, 72, 33, 99, 24, 49, 1, 15, 50]
[1, 12, 15, 24, 33, 49, 50, 72, 99, 100]
```

Expected output

`[100, 12, 72, 33, 99, 24, 49, 1, 15, 50]`
`[1, 12, 15, 24, 33, 49, 50, 72, 99, 100]`

Now **order** the list `my_list` in **descending** order.

In [23]:
```
my_list = [100, 12, 72, 33, 99, 24, 49, 1, 15, 50]
print(my_list)
#### START CODE HERE (~ 1 line) ####
my_list = sorted(my_list, reverse=True)
#### END CODE HERE ####
print(my_list)
```

```
[100, 12, 72, 33, 99, 24, 49, 1, 15, 50]
[100, 99, 72, 50, 49, 33, 24, 15, 12, 1]
```

Expected output

```
[100, 12, 72, 33, 99, 24, 49, 1, 15, 50]
[100, 99, 72, 50, 49, 33, 24, 15, 12, 1]
```

Exercise 4.7

Lists in Python are **mutable**. If you define a list variable, it is only a **reference** to its items. If you define a new variable with `l1 = l2` it is only a new reference to the same items. If you change the value of an item from one list, the changed item is also viewed from the other reference.

In [24]:
```
l1 = [1, 2, 3]
l2 = l1
print("l1: ", l1)
print("l2: ", l2)
l1[0] = 4
print("\n")
print("l1: ", l1)
print("l2: ", l2)
```

```
l1:  [1, 2, 3]
l2:  [1, 2, 3]


l1:  [4, 2, 3]
l2:  [4, 2, 3]
```

In this case, changing the first element in l1 also changes the first element in l2.

Define a new list variable by making a **hard copy** (i.e., the two lists have the same values but point at different memory spaces). Copy the list `l1` into a new list `l2`.

▶ **Hints**

In [25]:
```
l1 = [1, 2, 3]
#### START CODE HERE (~ 1 line) ####
l2 = l1.copy()
#### END CODE HERE ####
print("l1: ", l1)
print("l2: ", l2)
l1[0] = 4
print("\n")
print("l1: ", l1)
print("l2: ", l2)
```

```
l1:  [1, 2, 3]
l2:  [1, 2, 3]


l1:  [4, 2, 3]
l2:  [1, 2, 3]
```

Expected output

```
l1:  [1, 2, 3]
l2:  [1, 2, 3]

l1:  [4, 2, 3]
l2:  [1, 2, 3]
```

In this case, changing the first element in l1 do not affect the first element in l2.

5. Functions

A **function** is a block of code that only runs when it is called. A function can take some **input parameters** and return some **output** data as a result. You can learn more about python functions here.

Exercise 5.1

Create a function called `my_add_fn`. It takes two values as input `(x, y)`, and returns their **sum**.

In [26]:
```
def my_add_fn(x, y):
#### START CODE HER (~ 1 line) ####
    return x + y
#### END CODE HERE ####
```

In [27]:
```
my_add_fn(10, 20)
```

Out[27]:
```
30
```

Expected output

`30`

Exercise 5.2

Create a function called `my_avg_fn` that takes two values as input `(x, y)`, and returns their **average**.

In [28]:
```
#### START CODE HERE (~ 2 lines) ####
def my_avg_fn(x, y):
    return (x + y)/2
#### END CODE HERE ####
```

In [29]:
```
my_avg_fn(10, 20)
```

Out[29]:
```
15.0
```

Expected output

`15.0`

Exercise 5.3

Create a function called `my_replace_fn` that takes three strings as input `s1`, `s2`, and `s3`, and returns the value of `s1` by replacing each occurrence of the string `s2` from the string `s1` with a new string `s3`.

▶ **Hints**

In [30]:
```
def my_replace_fn(s1, s2, s3):
#### START CODE HERE (~ 1 line) ####
    return s1.replace(s2, s3)
#### END CODE HERE ####
```

In [31]:
```
s1 = "I don't know how to code in python"
s2 = "don't know"
s3 = "am learning"
my_replace_fn(s1, s2, s3)
```

Out[31]:
```
'I am learning how to code in python'
```

Expected output

`'I am learning how to code in python'`

6. Tuples

A tuple is a collection that is ordered and **unchangeable**. They are defined by **round brackets**, and the elements are separated by **commas**. Different from lists that are mutable. You can find more about python tuples here.

Exercise 6.1

Print:

- **all** the elements of the tuple
- the **second** element of the tuple
- the **last** element of the tuple
- the **length** of the tuple

▶ **Hints**

In [32]:
```
my_tuple = ("math", "science", "art", "history")
#### START CODE HERE (~ 4 lines) ####
print(my_tuple)
print(my_tuple[1])
print(my_tuple[-1])
```

```
print(len(my_tuple))
#### END CODE HERE ####
```

```
('math', 'science', 'art', 'history')
science
history
4
```

Expected output

`('math', 'science', 'art', 'history')`

`science`

`history`

`4`

Python tuples are **unchangeable**. What do you think will happen if I try to change the first element of the tuple?

In [33]:
```
my_tuple[0] = "new_subject"
print(my_tuple)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[33], line 1
----> 1 my_tuple[0] = "new_subject"
      2 print(my_tuple)

TypeError: 'tuple' object does not support item assignment
```

It will raise a TypeError

7. Sets

A set is a collection which is **unordered**, **unchangeable**, and **unindexed**. They are defined by **curly brackets**, and the elements are separated by **commas**. You can't change values in a set, but you can remove items or add new items. Sets do not allow duplicate values. You can find more about python sets here.

Exercise 7.1

Create a set named `my_set` with the following items: `'math'`, `'science'`, `'art'`, `'history'`. Then:

- print the **items** of the set
- print the **length** of the set
- **remove** "math" from the set
- **add** "computer science" to the set

In [34]:
```
#### START CODE HERE (~ 5 lines) ####
my_set = {"math", "science", "art", "history"}
print(my_set)
print(len(my_set))
my_set.remove("math")
my_set.add("computer science")
#### END CODE HERE ####
print(my_set)
```

```
{'history', 'science', 'art', 'math'}
4
{'art', 'computer science', 'science', 'history'}
```

Expected output

`{'science', 'history', 'math', 'art'}`

`4`

`{'computer science', 'science', 'art', 'history'}`

Exercise 7.2

Get the list of the **distinct elements** in `my_list`, save the distinct elements into a new list named `my_distinct_list`. The distinct elements are a list without the repetitions of elements.

▶ **Hints**

In [37]:
```
my_list = ["math", "science", "science", "art", "history", "history", "art", "art"]
my_distinct_list = None
#### START CODE HERE (~ 1 line) ####
my_distinct_list = list(set(my_list))
#### END CODE HERE ####

# Don't change this code!
print("original list:", my_list)
print("distinct list:", my_distinct_list)
```

```
original list: ['math', 'science', 'science', 'art', 'history', 'history', 'art', 'art']
distinct list: ['history', 'science', 'art', 'math']
```

Expected output

`original list: ['math', 'science', 'science', 'art', 'history', 'history', 'art', 'art']`
`distinct list: ['history', 'science', 'art', 'math']`

8. Dictionaries

A dictionary is a collection that is **ordered**, **changeable**, and does **not allow duplicates**. Dictionaries are written with **curly brackets**, and have **keys** and **values**. You can find more about python dictionaries here.

Exercise 8.1

You have a dictionary `my_dict` with the **subjects** as **keys** and **marks** as **values**. Print the mark corresponding to the subject `"art"`.

▶ **Hints**

In [38]:
```
my_dict = {"math":20,
           "science":30,
           "art":18,
           "history":20}

#### START CODE HERE (~ 1 line) ####
print(my_dict["art"])
#### END CODE HERE ####
```

```
18
```

Expected output

`18`

Exercise 8.2

Change the mark corresponding to the subject `math` with a new value equal to 30.

▶ **Hints**

In [39]:
```
my_dict = {"math":20,
           "science":30,
           "art":18,
           "history":20}
print(my_dict)

#### START CODE HERE (~ 1 line) ####
my_dict["math"] = 30
#### END CODE HERE ####

# Don't change this code!
print(my_dict)
```

```
{'math': 20, 'science': 30, 'art': 18, 'history': 20}
{'math': 30, 'science': 30, 'art': 18, 'history': 20}
```

Expected output

`{'math': 20, 'science': 30, 'art': 18, 'history': 20}`
`{'math': 30, 'science': 30, 'art': 18, 'history': 20}`

Exercise 8.3

Define a function that takes three parameters as input `my_dict`, `my_key`, and `my_value`. If `my_key` is already present in `my_dict`, it changes the old value with `my_value`. Otherwise, it adds the new key-value pair. The function returns the new dictionary.

▶ **Hints**

In [40]:
```
my_dict = {"math":20,
           "science":30,
           "art":18,
           "history":20}

def my_add_dict_fn(my_dict, my_key, my_value):
    #### START CODE HERE (~ 3 lines) ####
    my_new_dict = my_dict.copy()
    my_new_dict[my_key] = my_value
    return my_new_dict
    #### END CODE HERE ####
```

In [41]:
```
# Don't change this code!
my_new_dict_1 = my_add_dict_fn(my_dict, "art", 30)
my_new_dict_2 = my_add_dict_fn(my_dict, "english", 22)
print(my_dict)
print(my_new_dict_1)
print(my_new_dict_2)
```

```
{'math': 20, 'science': 30, 'art': 18, 'history': 20}
{'math': 20, 'science': 30, 'art': 30, 'history': 20}
{'math': 20, 'science': 30, 'art': 18, 'history': 20, 'english': 22}
```

Expected output

`{'math': 20, 'science': 30, 'art': 18, 'history': 20}`
`{'math': 20, 'science': 30, 'art': 30, 'history': 20}`
`{'math': 20, 'science': 30, 'art': 18, 'history': 20, 'english': 22}`

9. Conditions and If statements

You can find more about python conditions and if statements here.

Exercise 9.1

Define a function that takes a variable `x` as input and prints `"Greater or equal than zero"` if the variable is greater or equal to zero and `"Less than zero"` if negative. Be careful about python indentation.

In [42]:
```
def my_positive_check_fn(x):
#### START CODE HERE (~ 5 lines) ####
    if x >= 0:
        print("Greater or equal than zero")
    else:
        print("Less than zero")
    return
#### END CODE HERE ####

# Don't change this code!
my_positive_check_fn(100)
my_positive_check_fn(0)
my_positive_check_fn(-1)
```

```
Greater or equal than zero
Greater or equal than zero
Less than zero
```

Expected output

`Greater or equal than zero`
`Greater or equal than zero`
`Less than zero`

Exercise 9.2

Define a function that takes a variable `x` as input and prints `"Odd"` if the the variable is odd and `"Even"` if even. Be careful about python indentation.

▶ **Hints**

In [43]:
```
def my_odd_even_check_fn(x):
#### START CODE HERE (~ 5/6 lines) ####
    if (x % 2) == 0:
        print("Even")
    else:
        print("Odd")
    return
#### END CODE HERE ####

# Don't change this code!
my_odd_even_check_fn(1)
my_odd_even_check_fn(2)
my_odd_even_check_fn(5)
my_odd_even_check_fn(10)
```

```
Odd
Even
Odd
Even
```

Expected output

`Odd`
`Even`

`Odd`
`Even`

Exercise 9.3

Define a function that takes two strings `my_str` and `my_sub_str` as input. Then, it prints `"Found {my_sub_str} into {my_str}"` if `my_str` contains `my_sub_str`, and `"Not found {my_sub_str} into {my_str}"` otherwise. The printed string should substitute `my_sub_str` and `my_str` with their values. It returns the `True` boolean if `my_str` contains `my_sub_str`, `False` otherwise.

▶ **Hints**

In [44]:
```
def my_subtring_check_fn(my_str, my_sub_str):
#### START CODE HERE (~ 5/6 lines) ####
    if my_sub_str in my_str:
        print("Found {} into {}".format(my_sub_str, my_str))
        return True
    else:
        print("Not found {} into {}".format(my_sub_str, my_str))
        return False
#### END CODE HERE ####

# Don't change this code!
str_flag = my_subtring_check_fn("hello world", "hello")
print(str_flag)
str_flag = my_subtring_check_fn("hello world", "bye")
print(str_flag)
```

```
Found hello into hello world
True
Not found bye into hello world
False
```

Expected output

`Found hello into hello world`
`True`
`Not found bye into hello world`
`False`

Exercise 9.4

Define a function that takes a list of numbers named `my_list` and a number `x` input. Then, it returns the `True` boolean if `my_list` contains `x`, `False` otherwise.

▶ **Hints**

In [45]:
```
def my_list_check_fn(my_list, x):
#### START CODE HERE (~ 4 lines) ####
    if x in my_list:
        return True
    else:
        return False
#### END CODE HERE ####

# Don't change this code!
my_flag = my_list_check_fn([0, 4, 2, 5], 1)
print(my_flag)
my_flag = my_list_check_fn([0, 4, 2, 5], 4)
print(my_flag)
```

```
False
True
```

Expected output

`False`
`True`

Exercise 9.5

Create a function called `my_sum_avg_fn`. It takes three numbers as input `x`, `y`, and `z`, and it returns their **sum**, their **average**, and the boolean `True` if **all** the inputs parameters are **postiive** (>= 0). Otherwise, it returns `False`. Please respect the order of the returned parameters.

In [46]:
```
def my_sum_avg_fn(x, y, z):
#### START CODE HERE (~ 6/7 lines) ####
    my_sum = x + y + z
    my_avg = (x + y + z) / 3
    my_flag = False
    if x >= 0 and y >= 0 and z>= 0:
        my_flag = True
```

```
    return my_sum, my_avg, my_flag
#### END CODE HERE ####
```

In [47]:
```
my_sum, my_avg, my_flag = my_sum_avg_fn(0, 0, 0)
print("Sum: {}. Avg: {}. All Positive: {}".format(my_sum, my_avg, my_flag))
my_sum, my_avg, my_flag = my_sum_avg_fn(10, -1, 3)
print("Sum: {}. Avg: {}. All Positive: {}".format(my_sum, my_avg, my_flag))
my_sum, my_avg, my_flag = my_sum_avg_fn(10, 20, 30)
print("Sum: {}. Avg: {}. All Positive: {}".format(my_sum, my_avg, my_flag))
my_sum, my_avg, my_flag = my_sum_avg_fn(10, 0, 20)
print("Sum: {}. Avg: {}. All Positive: {}".format(my_sum, my_avg, my_flag))
```

```
Sum: 0. Avg: 0.0. All Positive: True
Sum: 12. Avg: 4.0. All Positive: False
Sum: 60. Avg: 20.0. All Positive: True
Sum: 30. Avg: 10.0. All Positive: True
```

Expected output

`Sum: 0. Avg: 0.0. All Positive: True`
`Sum: 12. Avg: 4.0. All Positive: False`
`Sum: 60. Avg: 20.0. All Positive: True`
`Sum: 30. Avg: 10.0. All Positive: True`

Exercise 9.6

Define a function that takes three parameters as input `my_dict`, `my_key`, and `my_value`. It changes the value of `my_key` with `my_value` **only if it is already present** in `my_dict`. Otherwise, it will print: `"The key {my_key} is not in the dictionary"`. You should not make a hard-copy of the dictionary.

▶ **Hints**

In [48]:
```
my_dict = {"math":20,
           "science":30,
           "art":18,
           "history":20}

def my_update_dict_fn(my_dict, my_key, my_value):
    #### START CODE HERE (~ 5 lines) ####
    if my_key in my_dict:
        my_dict[my_key] = my_value
    else:
        print("The key {} is not in the dictionary".format(my_key))
    return my_dict
    #### END CODE HERE ####
```

In [49]:
```
# Don't change this code!
print("original dict:", my_dict)
my_dict = my_update_dict_fn(my_dict, "art", 30)
print("update 1:", my_dict)
my_dict = my_update_dict_fn(my_dict, "english", 22)
print("update 2:", my_dict)
```

```
original dict: {'math': 20, 'science': 30, 'art': 18, 'history': 20}
update 1: {'math': 20, 'science': 30, 'art': 30, 'history': 20}
The key english is not in the dictionary
update 2: {'math': 20, 'science': 30, 'art': 30, 'history': 20}
```

Expected output

`original dict: {'math': 20, 'science': 30, 'art': 30, 'history': 20}`
`update 1: {'math': 20, 'science': 30, 'art': 30, 'history': 20}`
`The key english is not in the dictionary`
`update 2: {'math': 20, 'science': 30, 'art': 30, 'history': 20}`

10. For Loops

A **for loop** is used for **iterating over a sequence** (i.e., either a list, a tuple, a dictionary, a set, or a string). You can find more about python for loops here.

Exercise 10.1

Define a loop from `0` to `x` (both included) that prints `"{i}: greater or equal than 2"` if the index of the loop is greater or equal than 2. `"{i}: less than 2"` if the index of the loop is less than 2. The value of the index of the loop `i` should substitute `{i}` in the printed outputs.

▶ **Hints**

In [50]:
```
x = 5
#### START CODE HERE (~ 5/6 lines) ####
```

```
for i in range(5+1):
    if i >= 2:
        print("{}: greater or equal than 2".format(i))
    else:
        print("{}: less than 2".format(i))
#### END CODE HERE ####
```

```
0: less than 2
1: less than 2
2: greater or equal than 2
3: greater or equal than 2
4: greater or equal than 2
5: greater or equal than 2
```

Expected output

`0: less than 2`
`1: less than 2`
`2: greater or equal than 2`
`3: greater or equal than 2`
`4: greater or equal than 2`
`5: greater or equal than 2`

Exercise 10.2

Define a loop that iterates `my_list` and multiplies each element of the list by `2`. Save the elements multiplied by 2 in the same list `my_list`.

In [51]:
```
my_list = [1, 2, 3, 4, 5]
print("original list: {}".format(my_list))
#### START CODE HERE (~ 2 lines) ####
for i in range(len(my_list)):
    my_list[i] = my_list[i]*2
#### END CODE HERE ####

# Don't change this code!
print("modified list: {}".format(my_list))
```

```
original list: [1, 2, 3, 4, 5]
modified list: [2, 4, 6, 8, 10]
```

Expected output

`original list: [1, 2, 3, 4, 5]`
`modified list: [2, 4, 6, 8, 10]`

Exercise 10.3

Define a function that prints numbers from `0` to `n` (both included). It takes as input `n` and prints all the numbers from `0` to `n` (both included).

▶ **Hints**

In [52]:
```
def my_print_n_fn(n):
#### START CODE HERE (~ 2 lines) ####
    for i in range(n+1):
        print(i)
#### END CODE HERE ####
    return

# Don't change this code!
my_print_n_fn(0)
print("\n")
my_print_n_fn(3)
```

```
0


0
1
2
3
```

Expected output

`0`

`0`
`1`
`2`
`3`

Exercise 10.4

The dictionary `my_dict` contains **subjects** as **keys** and **marks** as **values**. Define a loop that iterates the dictionary `my_dict`. For each key-value pair, print `"key: {key}, value: {value}"`. Replace `{key}` and `{value}` with the current key and value at each iteration.

▶ **Hints**

In [53]:
```
my_dict = {"math": 30, "history": 27, "art":24, "computer science": 28}
#### START CODE HERE (~ 2 lines) ####
for key, value in my_dict.items():
    print("key: {}, value: {}".format(key, value))
#### END CODE HERE ####
```

```
key: math, value: 30
key: history, value: 27
key: art, value: 24
key: computer science, value: 28
```

Expected output

`key: math, value: 30`
`key: history, value: 27`
`key: art, value: 24`
`key: computer science, value: 28`

Exercise 10.5

The dictionary `my_dict` contains **subjects** as **keys** and **marks** as **values**. Define a loop that iterates the dictionary `my_dict` and saves in the list `tirty_marks` all the **subjects** with a mark equal to 30 (i.e., all the keys whose value is equal to 30). Count the number of elements with a mark equal to 30 (i.e., values equal to 30) in a variable `count_30`.

▶ **Hints**

In [54]:
```
count_30 = 0
my_dict = {"math": 30, "history": 27, "art":24, "computer science": 28, "science":30}
tirty_marks = []
#### START CODE HERE (~ 4 lines) ####
for key, value in my_dict.items():
    if value == 30:
        tirty_marks.append(key)
        count_30 += 1
#### END CODE HERE ####

# Don't change this code!
print("Number of exams with a mark of 30:", count_30)
print("Exams subjects with a mark of 30:", tirty_marks)
```

```
Number of exams with a mark of 30: 2
Exams subjects with a mark of 30: ['math', 'science']
```

Expected output

`Number of exams with a mark of 30: 2`
`Exams subjects with a mark of 30: ['math', 'science']`

Exercise 10.6

Define a loop that iterates the dictionary `my_dict` and **sums** the values **only if they are positive**. Put the **sum** in a variable called `marks_sum`.

In [55]:
```
my_dict = {"math": 30, "history": -1, "art":24, "computer science": -1, "science":30}
marks_sum = 0
#### START CODE HERE (~ 4 lines) ####
for key, value in my_dict.items():
    if value > 0:
        marks_sum += value
#### END CODE HERE ####

# Don't change this code!
print("Sum of positive marks", marks_sum)
```

```
Sum of positive marks 84
```

Expected output

`Sum of positive marks 84`