Lab 2

Objectives:

- Collections: Lists, Sets, Tuples, Dictionaries
- Control flow
- Functions
- Read/Write files

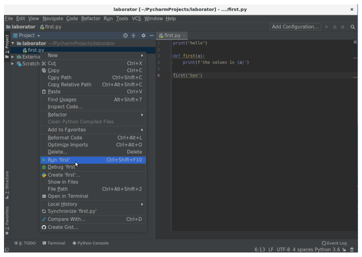
1 Additional resources

- Learn python 3
- Learn python from Scratch - Udemy
- Python in 2 hours
- Datacamp Intro to python for data science
- Python docs

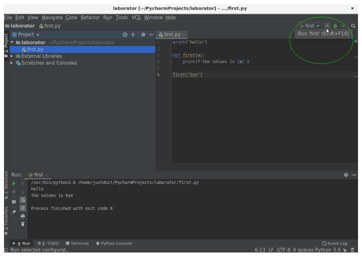
2 Use Integrated Development Environment (IDE)

In case you are new to Pycharm, go to pycharm tutorial. Important things for now:

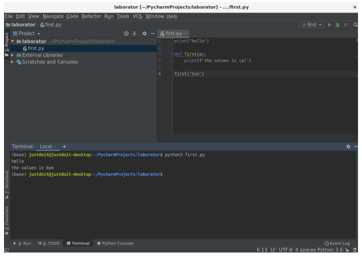
- set the interpreter (from Settings)
- run with Run configurations or,
- run from the terminal



```
print("hello")

def first(a):
    print(f'the values is {a}')

first("bye")
```



```
/usr/bin/python3.6 /home/justdoit/PycharmProjects/laborator/first.py
hello
the values is bye

Process finished with exit code 0
```



```
(base) justdoit@justdoit-desktop:~/PycharmProjects/laborator$ python3 first.py
hello
the values is bye
(base) justdoit@justdoit-desktop:~/PycharmProjects/laborator$
```

3 Collections of data

Tuples

- a tuple can contain different data types
- tuples allow duplicates
- tuples are **ordered** and **unchangeable**

Try some examples with tuples:

```
>>> empty = (); empty2 = tuple() # empty tuples
>>> single = (10,) # single-element tuple, comma matters!
>>> my_tuple = ('Sawyer', 17, 'Kate', True)
>>> my_tuple
('Sawyer', 17, 'Kate', True)
>>> type(my_tuple)
<class 'tuple'>
>>> len(my_tuple)
4
>>> 'Kate' in my_tuple  # test membership
True
>>> my_tuple + (1,2)  # concatenate two tuples
('Sawyer', 17, 'Kate', True, 1, 2)
```

The items of a tuple can be accessed using index, negative index or range of indexing (slicing) just like a list

```
>>> my_tuple[0]
'Sawyer'
>>> my_tuple[-2]
'Kate'
>>> my_tuple[1:3]
(17, 'Kate')
```

Tuples can have items with the same value:

```
another_tuple = ('Sawyer', 17, 'Sawyer', 'Kate', True)
```

Tuples are immutable. The contents of a tuple cannot be modified.

```
>>> my_tuple[0] = 'Sayid'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

As a workaround, to update a tuple it must be converted into a list first.

```
>>> my_list = list(my_tuple)  # convert the tuple into a list,
>>> my_list[0] = 22  # modify the first element
>>> my_list.append('Hugo')  # add an item
>>> my_list.remove(17)  # remove an item
>>> my_tuple = tuple(my_list)
>>> my_tuple
(22, 'Kate', True, 'Hugo')
```

Tuple methods:

count()	Returns the number of elements with the specified value
index()	Returns the index of the first element with the specified value

Sets

- a set can contain different data types
- sets don't allow duplicates
- sets are **unordered**, **unindexed**
- set items are **unchangeable**, but we can remove items and add new items

Try some examples with sets:

```
>>> empty = set()  # empty set
>>> single = {10}; single2 = set([10])  # single-element sets
>>> my_set = {12, "Jack", "Kate", True, "Kate"}
>>> my_set
{'Kate', True, 'Jack', 12}
>>> type(my_set)
<class 'set'>
>>> len(my_set)
4
>>> 12 in my_set  # fast membership testing
True
```

Set operations:

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                              # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                          # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                          # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                          # letters in both a and b
{'a', 'c'}
>>> a ^ b                          # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Set methods:

add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item.
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element.
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	Inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
supdate()	Update the set with the union of this set and others

```
>>> my_set = {'Kate', 'Sawyer', 'Jin'}
>>> my_set.add('Sun')  # add a single item
>>> my_set.update([22, 10])  # add multiple items from iterable object
>>> my_set
{'Sawyer', 'Jin', 'Sun', 'Kate', 10, 22}
>>> my_set.remove('Jin')  # remove an item + rise an error if it doesn't exist
>>> my_set.discard('Jin')  # remove an item
>>> my_set.pop() # remove last item
'Sawyer'
>>> my_set.clear()  # empties the set
```

Join sets:

```
>>> set1 = {"a", 1, "b" , 2, "c"}
>>> set2 = {1, 2, 3}
>>> set1.union(set2)  # combine the sets
{'a', 1, 2, 'c', 3, 'b'}
>>> set1.intersection(set2)  # returns the common values between the two sets
{1, 2}
```

Dictionaries

- a dictionary can store data values in key:value pairs
- the dictionaries don't allow duplicates
- the dictionaries are **ordered**(from Python 3.7 and above) and **changeable**

```
>>> empty = {}; empty2 = dict()  # empty dictionary
>>> my_dict = {"james": 1000, "kate":2005}
>>> type(my_dict)
<class 'dict'>
>>> len(my_dict)
2
>>> "kate" in my_dict  # test membership
True
```

Dictionary methods:

clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Accessing Items:

```
>>> my_dict = {"james": 1000, "kate": 2005}
>>> my_dict["james"]
1000
>>> my_dict.get("james")
1000
```

Get Keys:

```
>>> my_dict.keys()
dict_keys(['james', 'kate'])
```

Get Values:

```
>>> my_dict.values()
dict_values([1000, 2005])
```

Get Items:

```
>>> my_dict.items()
dict_items([('james', 1000), ('kate', 2005)])
```

Add, update, remove items from dictionary:

```
>>> new_dict = {"name":"Jack", "age": 39}
>>> new_dict["job"] = "surgeon"  # add a new item
>>> new_dict["name"] = "James"  # change the value of the key "name"
>>> new_dict
{'name': 'james', 'age': 39, 'job': 'surgeon'}
>>> new_dict.update({"actor":"Matthew"}) # add a new item because
                                           key "actor" doesn't exist
>>> new_dict.update({"name":"Jack"}) # change the value of the key "name"
>>> new_dict
{'name': 'Jack', 'age': 39, 'job': 'surgeon', 'actor': 'Matthew'}
>>> new_dict.pop("actor")  # remove the item with specified key
'Matthew'
>>> new_dict
{'name': 'Jack', 'age': 39, 'job': 'surgeon'}
>>> new_dict.clear() ## empties the dictionary
```

Dictionary comprehension:

```
>>> new_dict_comp = {n:n**2 for n in [0,10,-2,3] if n%2 == 0}
>>> new_dict_comp
{0: 0, 10: 100, -2: 4}
```

4 Control flow

if statement:

```
a = 44
b = 20
if b > a:
  print("b is greater than a")
elif a == b:
{      Kate      , True ,      Jack      , 12}  print("a and b are equal")
else:
  print("a is greater than b")
```

Inline if statement:

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

Inline if else statement:

```
print("A") if a > b else print("B")
```

for statement:

```
word = "Incomprehensibilities"
for char in word:
    print(char)

>>> a=[10,20,3,1]
>>> for e in a:
...     print(e)
...
10
20
3
```

```
1

>>> for i in range(len(a)):  #avoid this, and use iteration on the elements
...     print(a[i])
...
10
20
3
1

>>> for i,e in enumerate(a): # in case you need both the index and the value
...     print(f'at index {i} there is element {e}')
...
at index 0 there is element 10
at index 1 there is element 20
at index 2 there is element 3
at index 3 there is element 1

>>>for k,v in new_dict_comp.items(): #new_dict_comp is a dictionary
...     print(k,v)
...
0 0
10 100
-2 4
```

while statement:

```
n = 10
sum = 0
i = 1

while i <= n:
    sum += i
    i += 1

print("The sum is", sum)
```

break, continue, pass statements:

- **break** - terminates the loop containing it

```
    for val in "string":
        if val == "i":
            break
        print(val)
    print("The end")
```

- **continue** - skip the rest of the code inside a loop for the current iteration only

```
    for val in "string":
        if val == "i":
            continue
        print(val)
    print("The end")
```

- **pass** is a null statement

```
    for val in "string":
        pass
```

5 Functions

In Python a function is defined using the **def** keyword. It is followed by the function's name and its parameters placed inside parenthesis. The first statement of the function body can optionally be a string literal.

```
def print_name(fname, lname):
    "Print the name given as parameters"  # docstring
    print(f"My name is {fname} {lname}")

print_name("James", "Ford")  # calling the function
```

A function can be called either using **positional arguments** or **keyword arguments**. The arguments are passed using **call by value** (where the value is always an object reference, not the value of the object). Immutable arguments like whole numbers, strings or tuples are passed like call-by-value because they can not be changed. Passing mutable objects (e.g. lists) can be considered as call by reference.

```
print_name("James", "Ford")    # calling the function using positional args
print_name(fname="James", lname="Ford")  # calling the function keyword args
print_name(lname="Ford", fname="James")  # calling the function keyword args

>>>a=[1,10,100]
>>>def change(my_list):
...     my_list[0]=1000
...
>>> change(a)
>>> a
[1000, 10, 100]
```

Arguments can have default values. In the example below **age** has a default value of 39 and **job** has a default value of "salesman"

```
def print_info(fname, lname, age=39, job="salesman"):
    "Print the name given as parameters"  # docstring
    print(f"My name is {fname} {lname}. I'm {age} years old and I'm a {job}")
```

The function can be called in several ways:

```
>>> print_info("James", "Ford")  # giving only the mandatory argument
My name is James Ford. I'm 39 years old and I'm a salesman
>>> print_info("James", "Ford", 22)  # giving one of the optional arguments
My name is James Ford. I'm 22 years old and I'm a salesman
>>> print_info("James", "Ford", 22, 'surgeon')  # giving all arguments
My name is James Ford. I'm 22 years old and I'm a surgeon
```

A function can be called with an arbitrary argument list which is passed to the function as a tuple usign the * operator.

```
def show_best_player(team, *players):
    print(f"Team: {team} Best player: {players[1]}")

show_best_player("Man Utd", "Marcus Rashford", "Harry Maguire")
show_best_player("Man City", "Phil Foden", "Erling Haaland", "Sergio Gomez")
```

A function can be called with an arbitrary keyword argument list which is passed to the function as a dictionary using the ** operator.

```
def show_name(**kid):
  print("His last name is " + kid["lname"])

show_name(fname = "Kate", lname = "Austen")
show_name(fname = "James", lname = "Ford", age = "2")
```

To return values from a function it is used the **return** keyword.

```
def function1():
    return 1  # return a single value

def function2():
    a = 10
    return 1, a  # return multiple values
```

6 Read/Write Files

In Python, files are **read** using the built-in **open()** function. The most important arguments for the open() function are the **file path** and a **file opening mode**. The **path** represents the **location of the file on your computer / the file name** and it can be **absolute** or **relative**. The **default** file opening mode is **read-only**.

Python open() mode argument:

'r'	open for reading
'w'	open for writing, truncating the file first
'x'	create a new file and open it for writing
'a'	open for writing / appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open a disk file for updating
'r+'	read and write
'a+'	read and write (if the file exists data will be added at the end otherwise, the file will be created)

Reading files using:

- **read()**: provides the entire contents of the text file in a single string
- **readline()**: provides the contents of the current line in the text file in a string or an empty string when the end of the file is reached
- **readlines()**: provide all lines in the text file in a list of string

```
f = open(r'test_file.txt')
text = f.readline()
count_line = 0
while text != '':
    print(f"{count_line}-->{text}")
    text = f.readline()
    count_line += 1
f.close()
```

```
f = open(r'test_file.txt')
for line in f:
    print(line)
f.close()
```

Without the with statement, the file stream would have to be opened, used and then closed. The with statement automatically closes the file stream at the end of the block.

```
with open('test_file', "r") as f:
    read_data = f.read()
```

Writing files using:

- **write()**: writes a string to the file, without automatically adding a new line
- **writelines()**: writes all the strings in the collection to the file, without automatically adding new lines between them

```
f = open(r'test_file.txt', 'w')
text = ['My', 'name', 'is', 'James']
for word in text:
    f.write(word)
f.close()
```

```
f = open(r'test_file.txt', 'w')
text = ['My', 'name', 'is', 'James']
f.writelines(text)
f.close()
```

7 Exercises

1. You have a dictionary which gives you the price for some products. You also have a shopping list: for each item you know the quantity.

   ```
   price={"apples":10, "milk":12, "bread": 5}
   mylist=[("apples",2), ("milk",2)]
   ```

   Compute the price of your shopping list. If it is possible, avoid using for or while, remember that there are list comprehension, access of values based on the key, and sum of a list.

2. You have a list of friends that gave you money in your last trip. Extract their names (without duplicates) by using sets.

   ```
   friends = [("andrew",10), ("george",20), ("andrew", 5), ("ann", 10)]
   ```

3. Take an integer as standard input. List all numbers up to that number if the number is even and all numbers up to that number, only squared, if the number is odd.

4. Take a string, s_in and a positive integer, num_in as standard input. Create a dictionary that will have all numbers up to num_in as keys and all string characters from the corresponding num_in positions as values.

5. Define a function with a single parameter (a string) that will return a new string: for each letter in the original string, the new string will contain the corresponding next letter in the alphabet.

   HINT:

   ```
   >>>ord("a")
   97
   >>> chr(ord("a")+1)
   'b'
   ```

6. Write a program to open 2 .txt files at the same time. For the first one convert the first letter of every word to upper case and then add the file's content to the other file.