Functions 1/2

What is a function?

- Like in maths, a function takes **arguments as inputs**, and **returns a value**
- It is a **shortcut** toward a block of code
- The code inside a function can be **reused** many times with **different arguments**
- It **avoids copy/pasting** code, which is good:
  - if we make a mistake, we only have to **correct it once**
  - it **organizes** the code by **functionality**, making it **easier** to read & modify
  - we can use functions without having to know their **implementation**
- Functions help keep our **code factored, structured and modular**

Keywords

- To **define** a function, we use the keyword **def**
- A function may return a value using the keyword **return**
- Code inside a function definition is **indented**
- Example:

```
def average(a, b):
    m = (a + b) / 2
    return m
```

- What does **print(average(10, 2))** display?

A word on types

- So far, we have studied **floats, ints, booleans, strings** and **lists**
- Functions are usually written with **a type for each parameter** in mind
- Functions are also written with **a return type** in mind
- What happens if I write:
  - **print(average("foo", 3))**
- Always **define the type of your parameters and result** before writing a function!

print vs. return

- **print** lets us **display** information to the user
- Printed information cannot be reused by the program, it is only for the user

- **return** indicates what has been computed by a function
- returned values can be **reused** by the code that called the function
- For example, the result returned by a function can be **stored** in a variable

Example

- Every time we call “bidon”, we print 1
- The function returns **2, not 1**
- We can store the returned value in a variable and print it

```
def bidon():
    print(1)
    return 2
```

```
a = bidon()     # it will still display a 1
```

```
1
```

```
print(a)        # But the returned value is completely different!
```

```
2
```

Returning multiple values

- In Python, a function can return multiple values
- We write: return a, b
- This is short for return (a, b)
- Our return type is tuple

```
def f(a, b):
    return a + b, a – b
c, d = f(3, 4)
```

- What is the value of c? of d?

Returning void

- When a function does not return a value, we say that its return type is **void**
- A function that only prints thing has return type void
- A function that modifies a list in place (using .append() for example) also has return type void

Example

```
def add_to_list(a, b, l):
    l.append(a)
    l.append(b)
my_list = [2, 3]
print(my_list)
add_to_list(4, 5, my_list)
print(my_list)
```

- What is the type of the 3 parameters? What is the return type?
- What does the 1st print display? What about the 2nd?

Parameters vs. arguments

```
def average(a, b):
    m = (a + b) / 2
    return m
average(5, 10)
```

- a and b are the **parameters of the function**
- 5 and 10 are the **arguments** used to invoke the function

Named arguments

- When we have a lot of arguments, things can get confusing
- We can use the name of the parameters to help with that!

```
def name_complete(firstname, name):
    return firstname + ' ' + name

name_complete(firstname='James', name='Bond')
```

```
James Bond'
```

Using **named arguments** allows you to pass a function's arguments out of order.

```
name_complete(name='Proust', firstname='Marcel')
```

```
Marcel Proust'
```

```python
def name_complete(firstname='John', name='Doe'):    # Le nom par défaut est John Doe
    return firstname + ' ' + name

print(name_complete())    # Aucun paramètre donc nom et prenom par défaut
print(name_complete('Ulysse'))    # Le prenom est donné, seul le nom sera par défaut
print(name_complete(name='Machin'))    # Le nom est spécifié, seul le prénom sera par défaut
```

```
John Doe
Ulysse Doe
John Machin
```

Default parameters

- Sometimes, we want a paremeter to be **optional**
- In order to do so, we can give it a **default value**

Conclusion

- Functions are one of the **cornerstones of computer programming**
- When **professional developpers** write code, they use function all the time:
  - There is usually **no code** at the "root level" (no indentation)
  - **Everything** is a function!
  - We will use functions all the time in 2nd term for **projects**
- **Good practices:**
  - Name your parameters and your functions smartly, it needs to mean something
  - Try to write a function for each functionality your program should have!