Functions

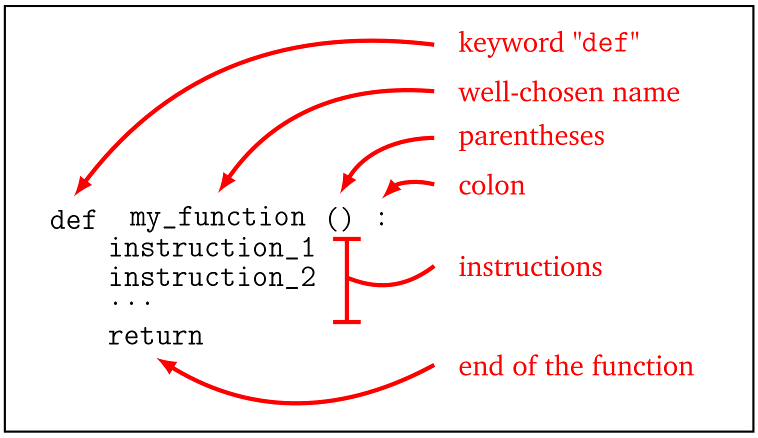
Writing a function is the easiest way to group code for a particular task, in order to execute it once or several times later.

Lesson 1 (Function (start)).
A computer function is a portion of code that performs a specific task and can be used one or more times during the rest of the program. To define a function with Python, it's very simple. Here are two examples:

```
def say_hello():
    print("Hello world!")
    return
```

```
def print_squares():
    for i in range(20):
        print(i**2)
    return
```

The instructions are grouped into an indented block. The word `return` (optional) indicates the end of the function. These instructions are executed only if I call the function. For example, each time I execute the command `say_hello()`, Python displays the sentence "Hello world!". Each time I execute the command `print_squares()`, Python displays $0, 1, 4, 9, 16, \ldots$, i.e. the numbers i^2 for $i = 0, \ldots, 19$.



Lesson 2 (Function (continued)).
Functions achieve their full potential with:

- an ***input***, which groups variables that serve as ***parameters***,
- an ***output***, which is a result returned by the function (and which will often depend on the input parameters).

Here are two examples:

```
def display_month(number):
    if number == 1:
        print("We are in January.")
    if number == 2:
        print("We are in February.")
    if number == 3:
        print("We are in March.")
    # etc.
    return
```

When called this function displays the name of the month based on the number provided as input. For example `display_month(3)` will display `"We are in March."`.

```
def compute_cube(a):
    cube = a * a * a     # or a**3
    return cube
```

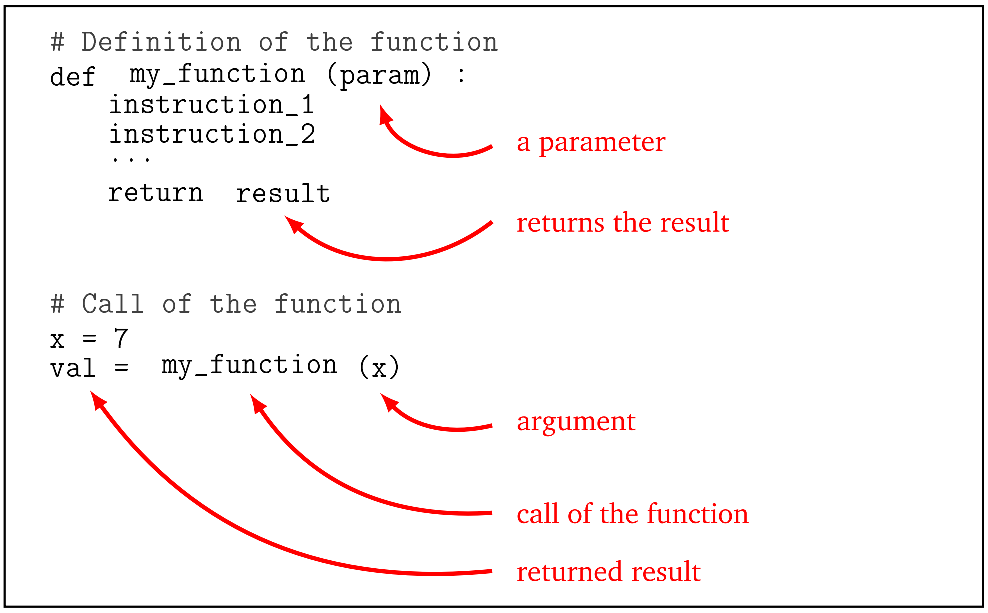
This function calculates the cube of a number, for example `compute_cube(2)` does not display anything but returns the value 8. This value can be used elsewhere in the program. For example, what do the following instructions do?

```
x = 3
y = 4
z = compute_cube(x) + compute_cube(y)
print(z)
```

In mathematical terms, we put $x = 3$, $y = 4$, then we calculate the cube of x, the cube of y and add them up:

$$z = x^3 + y^3 = 3^3 + 4^3 = 27 + 64 = 91$$

Thus the program displays `91`.



```
# Definition of the function
def  my_function (param) :
    instruction_1
    instruction_2
    ...
    return  result

# Call of the function
x = 7
val =  my_function  (x)
```

The advantages of programming using functions are as follows:

- you write the code of a function only once, but you can call the function several times;
- by dividing our program into small blocks, each with its own use, the program is easier to write, read, correct and modify;
- you can use a function written by someone else (such as the `sqrt()` function) without knowing all the internal details of its programming.

Activity 1 (First functions).

Goal: write very simple functions.

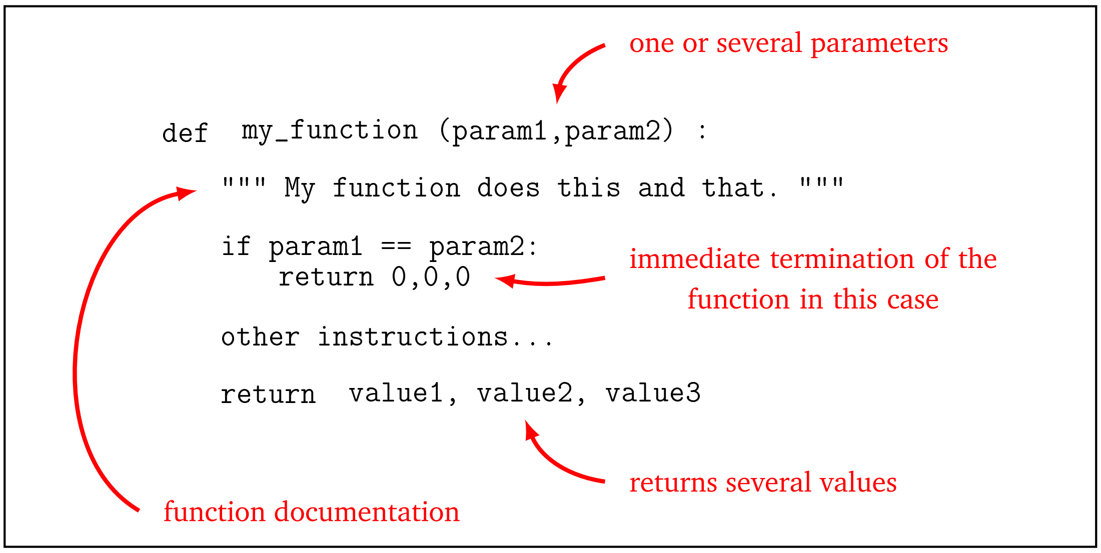
1. **Function without parameter or output.**
   (a) Program a function called `print_table_of_7()` that displays the multiplication table by 7: $1 \times 7 = 7$, $2 \times 7 = 14\ldots$
   (b) Program a function called `print_hello()` that asks the user for his first name and then displays “Hello” followed by his name.
   Hint. Use `input()`.
2. **Function with one parameter and no output.**
   (a) Program a function called `print_a_table(n)` that depends on a parameter `n` and displays the multiplication table by this integer n. For example, the command `print_a_table(5)` must display: $1 \times 5 = 5$, $2 \times 5 = 10\ldots$
   (b) Program a function called `say_greeting(sentence)` that depends on a parameter `sentence`. This function asks for the user’s first name and displays a greeting followed by the first name. For example, `say_greeting("Hi")` would display “Hi” followed by the first name given by the user.
3. **Function without parameter and with output.**
   Program a function called `ask_full_name()` that first asks for the user’s first name, then his last name and returns the complete identity with the last name in upper case as a result. For example, if the user enters “Dark” then “Vador”, the function returns the string `"Dark VADOR"` (the function displays nothing).
   Hints.
   - If `string` is a string, then `string.upper()` is the transformed string with characters in capital letters. Example: if `string = "Vador"` then `string.upper()` returns `"VADOR"`.
   - You can merge two strings by using the operator “+”. Example: `"Dark" + "Vador"` is equal to `"DarkVador"`. Another example: if `string1 = "Dark"` and `string2 = "Vador"` then `string1 + " " + string2` is equal to `"Dark Vador"`.

Lesson 3 (Function (continuation and end for now)).
A function can have several parameters and can return several results. For example, here is a function that calculates and returns the sum and product of two given input numbers.

```
def sum_product(a,b):
    """ Computes the sum and product of two numbers. """
    s = a + b
    p = a * b
    return s, p
```

```
mysum, myprod = sum_product(6,7)
```

The last line calls the function with arguments 6 (for parameter `a`) and 7 (for parameter `b`). This function returns two values, the first one is assigned to `mysum` (which is therefore equal to 13) and the second one to `myprod` (which is equal to 42).



```
def  my_function (param1,param2) :
    """ My function does this and that. """
    if param1 == param2:
        return 0,0,0
    other instructions...
    return  value1, value2, value3
```

one or several parameters
immediate termination of the function in this case
returns several values
function documentation

So let's remember:

- There can be several input parameters.
- There can be several results at the output.
- Very important! Do not confuse displaying and returning a value. The display (by the command `print()`) just displays something on the screen. Most functions do not display anything, but return one (or more) value. This is much more useful because this value can be used elsewhere in the program.
- As soon as the program encounters the instruction `return`, the function stops and returns the result. There may be several times the instruction `return` in a function but only one will be executed. It is also possible not to put an instruction `return` if the function returns nothing.
- In the instructions of a function, you can of course use other functions!
- It is important to comment well on your programs. To document a function, you can describe what it does starting with a *docstring*, i.e. a description (in English) surrounded by three quotation marks:
  ```
  """ My function does this and that. """
  ```
  to be placed just after the header.
- When defining a function, the variables that appear between the parentheses are called the ***parameters***; when calling the function, however, the values between the parentheses are called the ***arguments***. There is of course a correspondence between the two.

Activity 2 (More features).

Goal: build functions with different types of input and output.

1. **Trinomials.**
   (a) Write a function `trinomial_1(x)` that depends on a parameter `x` and returns the value of the trinomial $3x^2 - 7x + 4$. For example, `trinomial_1(7)` returns 102.

(b) Write a function `trinomial_2(a,b,c,x)` that depends on four parameters `a`, `b`, `c` and `x` and returns the value of the trinomial $ax^2 + bx + c$. For example, `trinomial_2(2,-1,0,6)` returns `66`.

2. **Currencies.**
   (a) Write a function `conversion_dollars_to_euros(amount)` which depends on a parameter and which for a sum of money `amount`, expressed in dollars, returns its value in euros (you will take for example 1 dollar = 0.89 euro).
   (b) Write a function `conversion_dollars(amount,currency)` which depends on two parameters `amount` and `currency` and converts the amount given in dollars, to the desired currency. Examples of currencies: 1 dollar = 0.89 euros; 1 dollar = 0.77 pounds; 1 dollar = 110 yen. For example, `conversion_dollars(100,"pound")` returns 77.
   Take care to give an intelligible name to your functions and variables. Don't forget to document each function by adding a small explanatory text between triple quotation marks at the very beginning of your function.
3. **Volumes.**
   Build functions that calculate and return volumes:
   - the volume of a cube according to the length of one side,
   - the volume of a ball according to its radius,
   - the volume of a cylinder according to the radius of its base and its height,
   - the volume of a rectangular parallelepiped box according to its three dimensions.

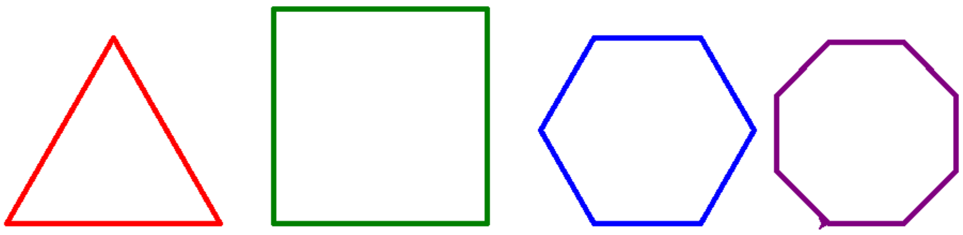
   For the value of π, you will take either the approximate value 3.14, or the approximate value provided by the constant `pi` of the module `math`.
4. **Perimeters and areas.**
   (a) Write a function whose use is `perimeter_area_rectangle(a,b)` and which returns the perimeter and area of a rectangle with dimensions of a and b.
   (b) Same question with `perimeter_area_disc(r)` for the perimeter and area of a disc of radius r.
   (c) Use your previous function to guess from which radius, the area of a disc is larger than the perimeter of that disc.
   Hint. If you want to scan the radius by incrementing the value of 0.1 each time, you can build a loop as follows:

```
for R in range(0,30):
```

   then make a call to the function by `perimeter_area_disc(R/10)`.

Activity 3 (Turtle)**.**

Goal: define some functions that draw geometric shapes. Creating a function is similar to creating a block with Scratch.



1. Program a function `triangle()` that draws a triangle (in red, each side measuring 200).

2. Program a function `square()` that draws a square (in green, each side measuring 200). Use a loop "for" so you don't have to rewrite the same instructions several times.
3. Program a `hexagon(length)` function that draws a hexagon (in blue) of a given side length (the angle to turn is 60 degrees).
4. Program a function `polygon(n,length)` that draws a regular polygon of n sides and a given side length (the angle to rotate is $360/n$ degrees).

Activity 4 (Functions again).

Goal: create new functions.

1. (a) Here is the discount for the price of a train ticket based on the age of the passenger:
   - reduction of 50% for those under 10 years old;
   - reduction of 30% for 10 to 18 years old;
   - reduction of 20% for 60 years old and over.

   Write a function `reduction()` that returns the reduction according to age and whose properties are recalled in the box below:

   `reduction()`

   Use: `reduction(age)`
   Input: an integer corresponding to age
   Output: an integer corresponding to the reduction

   Examples:
   - `reduction(17)` returns 30.
   - `reduction(23)` returns 0.

   Deduce a `amount()` function that calculates the amount to be paid based on the normal fare and the traveler's age.

   `amount()`

   Use: `amount(normal_rate,age)`
   Input: a number `normal_rate` corresponding to the price without discount and `age` (an integer)
   Output: a number corresponding to the amount to be paid after reduction
   Note: uses the function `reduction()`

   Example: `amount(100,17)` returns 70.

   A family buys tickets for different trips, here is the normal fare for each trip and the ages of the passengers:
   - normal price 30 dollars, child of 9 years old;
   - normal price 20 dollars, for each of the twins of 16 years old;
   - normal price 35 dollars, for each parent of 40 years old.

   What is the total amount paid by the family?
2. We want to program a small quiz on the multiplication tables.
   (a) Program a function `is_calculation_correct()` that decides if the answer given to a multiplication is right or not.

`is_calculation_correct()`

Use: `is_calculation_correct(a,b,answer)`
Input: three integers, `answer` being the proposed answer to the calculation of $a \times b$.
Output: "True" or "False", depending on whether the answer is correct or not

Examples:

- `is_calculation_correct(6,7,35)` returns `False`.
- `is_calculation_correct(6,7,42)` returns `True`.

(b) Program a function that displays a multiplication, asks for an answer and displays a short concluding sentence. All this in English or a foreign language!

`test_multiplication()`

Use: `test_multiplication(a,b,lang)`
Input: two integers, the chosen language (for example `"english"` or `"french"`)
Output: nothing (but display a sentence)
Note: uses the function `is_calculation_correct()`

Example: `test_multiplication(6,7,"french")` asks, in French, for the answer to the calculation 6×7 and answers if it is correct or not.

Bonus. Improve your program so that the computer offers random operations to the player. (Use the `randint()` function of the module `random`.)

Activity 5 (Experimental equality).

Goal: use the computer to experiment with equality of functions.

1. (a) Build a function `absolute_value(x)` that returns the absolute value of a number (without using the function `abs()` of `Python`).
   (b) Build a function `root_of_square(x)` which corresponds to the calculation of $\sqrt{x^2}$.
   (c) Two functions (of one variable) f and g are said to be *experimentally equal* if $f(i) = g(i)$ for $i = -100, -99, \ldots, 0, 1, 2, \ldots, 100$. Check by computer that the two functions defined by
   $$|x| \qquad \text{and} \qquad \sqrt{x^2}$$
   are experimentally equal.
2. (a) Build a two-parameter function `F1(a,b)` that returns $(a+b)^2$. Same thing with `F2(a,b)` that returns $a^2 + 2ab + b^2$.
   (b) Two functions of two variables F and G are said to be *experimentally equal* if $F(i,j) = G(i,j)$ for all $i = -100, -99, \ldots, 100$ and for all $j = -100, -99, \ldots, 100$. Check by computer that the functions defined by $(a+b)^2$ and $a^2 + 2ab + b^2$ are experimentally equal.
   (c) I know that one of the following two identities is true:
   $$(a-b)^3 = a^3 - 3a^2b - 3ab^2 + b^3 \qquad \text{or} \qquad (a-b)^3 = a^3 - 3a^2b + 3ab^2 - b^3.$$
   Help yourself from the computer to decide which one it is!
3. (a) Build a function `sincos(x)` that returns $(\sin(x))^2 + (\cos(x))^2$ and another `one(x)` that always returns 1. Are these two functions experimentally equal (in the sense of the first question)? Find out what could be the cause of this answer.

(b) Fix $\epsilon = 0.00001$. It is said that two functions (of one variable) f and g are *experimentally approximately equal* if $|f(i) - g(i)| \leqslant \epsilon$ for $i = -100, -99, \ldots, 100$. Do the two functions defined by `sincos(x)` and `one(x)` now check this criterion?

(c) Experimentally check and experimentally approximately check the identities:

$$\sin(2x) = 2\sin(x)\cos(x), \qquad \cos\left(\frac{\pi}{2} - x\right) = \sin(x).$$

(d) **Bonus. A counter-example.** Show that the functions defined by $g_1(x) = \sin(\pi x)$ and $g_2(x) = 0$ are experimentally equal (with our definition given above). But also show that you don't get $g_1(x) = g_2(x)$ for every $x \in \mathbb{R}$.

Lesson 4 (Local variable)**.**
Here is a very simple function that takes a number as an input and returns the number increased by one.

```
def my_function(x):
    x = x + 1
    return x
```

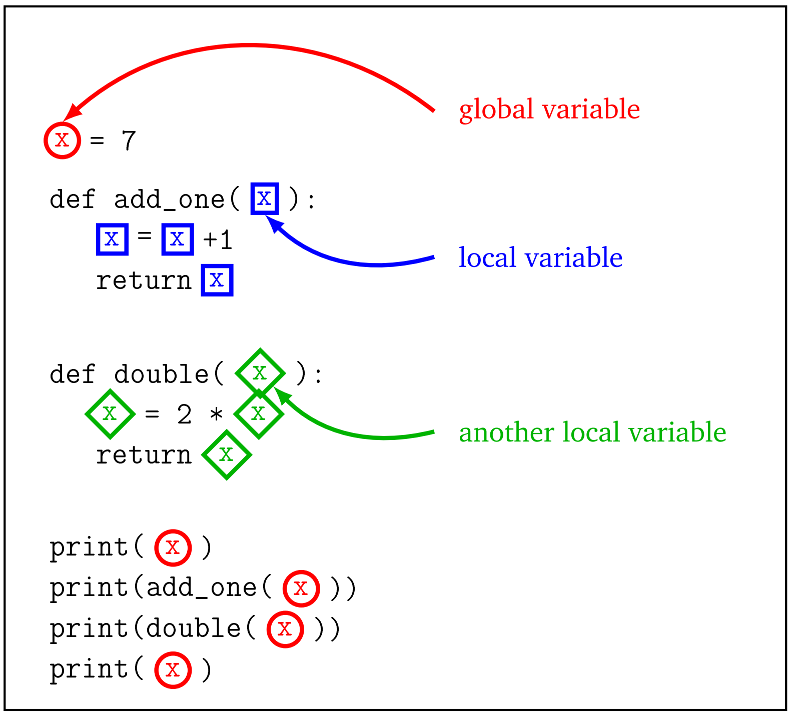
- Of course `my_function(3)` returns 4.
- If I define a variable by `y = 5` then `my_function(y)` returns 6. And the value of `y` has not changed, it is still equal to 5.
- Here is the delicate situation that you must understand:

  ```
  x = 7
  print(my_function(x))
  print(x)
  ```

  - The variable `x` is initialized to 7.
  - The call of the function `my_function(x)` is therefore the same as `my_function(7)` and logically returns 8.
  - What is the value of `x` at the end? The variable `x` is unchanged and is still equal to 7! Even if in the meantime there has been an instruction `x = x + 1`. This instruction changed the `x` inside the function, but not the `x` outside the function.

- Variables defined within a function are called *local variables*. They do not exist outside the function.
- If there is a variable in a function that has the same name as a variable in the program (like the `x` in the example above), it is as if there were two distinct variables; the local variable only exists inside the function.

To understand the scope of the variables, you can color the global variables of a function in red, and the local variables with one color per function. The following small program defines a function that adds one and another that calculates the double.



```
x = 7

def add_one( x ):
    x = x +1
    return x

def double( x ):
    x = 2 * x
    return x

print( x )
print(add_one( x ))
print(double( x ))
print( x )
```

The program first displays the value of x, so 7, then it increases it by 1, so it displays 8, then it displays twice as much as x, so 14. The global variable x has never changed, so the last display of x is still 7.

It is still possible to force the hand to Python and modify a global variable in a function using the keyword global. See the chapter “Polish calculator – Stacks”.