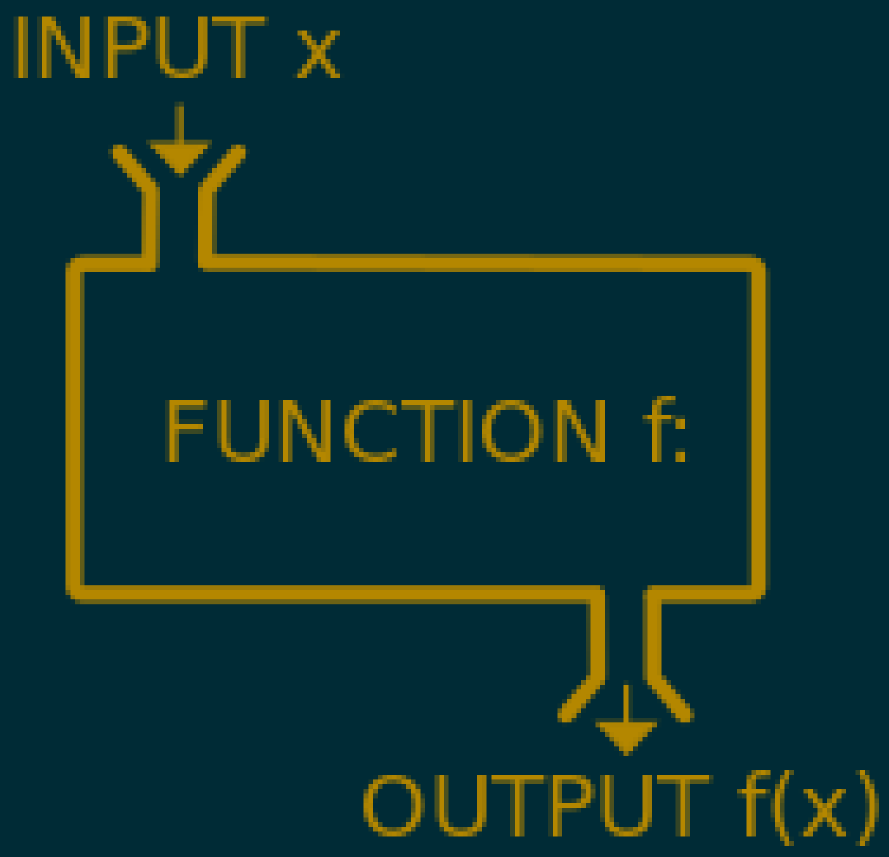
LEARN CODING

ale66

github.com/ale66/learn-coding

FUNCTIONS



Functions are a key abstraction to model nature and processes

a regular input/output or cause/effect behaviour is identified and *given a name*

```
The higher the temperature the quicker the veggies cook.

Cooking time is a function of the temperature in the oven.
```

FUNCTIONS IN CODING

A function is a block of code that

- has a clear input/output definition and
- executes in a separated environment

```
def marks2pc(marks: int):
  ''' Convert Italian exam marks into percentages '''

  pc = int((marks / 30) * 100)

  return pc
```

`marks` is a *parameter* of the f.

`pc` is the *return value* of the f.

OBSERVATIONS

Functions only run when they are called (‘invoked’) within a code in execution

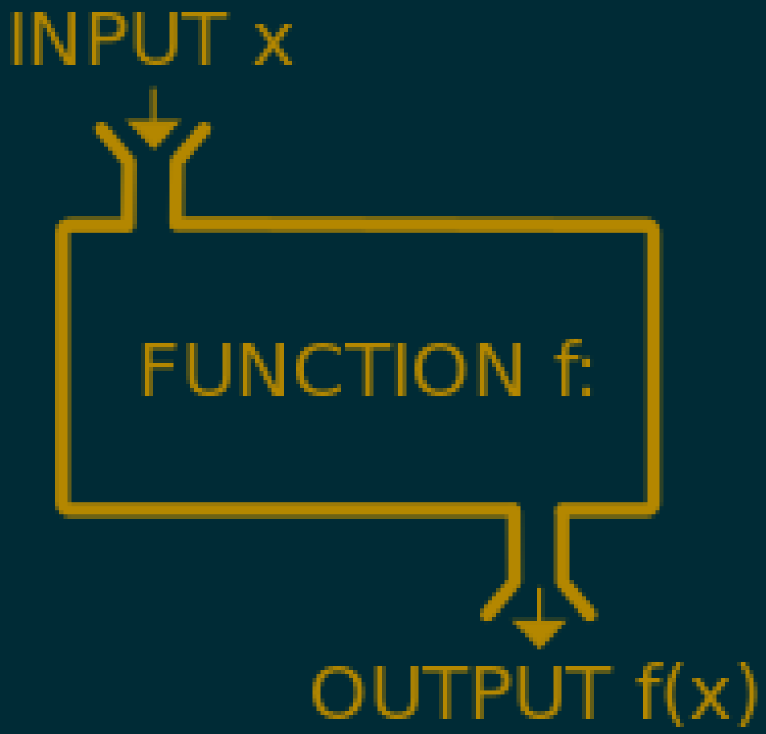
```
for m in my_italian_exam_marks:
  uk_marks = marks2pc(m)
  print(uk_marks)
```

Functions should be defined every time a block of code is required to appear more than once:

- improve readability
- improve maintainance

SYNTAX

```
def <name>(<parameter(s)>):
  return <output(s)>
```



KINDS OF FUNCTIONS:

- **built-in:** come with Python, e.g., `input()`, `print()`, `float()`, `int()` etc.
- **methods:** come with types, e.g., `mylist.len()`, `mylist.append('a')`
- **external** are *imported* into the code upon demand (more later)
- **defined** we define then use them

Names of the built-in functions are reserved: avoid them as variable names

`void` FUNCTIONS

Functions may *not* return a value

they need not be to the right of an assignment symbol

```python
def greet(lang):

  if lang == 'es':
    print('Hola!')

  elif lang == 'fr':
    print('Bonjour!')

  else:
    print('Hello!')

# example call
greet('es')
```

```
Hola!
```

CALLING FUNCTIONS

PREPARING FUNCTIONS

Given a text and a character, count the number of occurrences of it

```python
def char_counter(text, c):
  '''returns the number of times c is found in text'''

  # this cannot be empty
  pass
```

A recent development: types

```python
def char_counter(text: str, c: str) --> int:
  '''returns the number of times c is found in text'''

  # this cannot be empty
  pass
```

MULTIPLE ARGUMENTS

f. can take two or more arguments (more later)

mapping calling parameters to input vars. is done by position

it is also possible to return multiple arguments

```python
def func(primero, segundo):

  # ...

  return tercero, quarto

# example call
first_out, second_out =  func(first_in, second_in)
```

what are the bindings?

FUNCTION NESTING

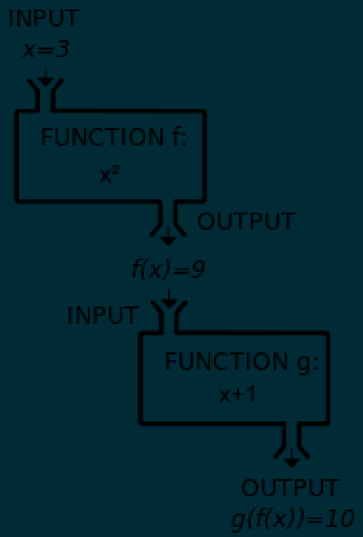
Example: compute some stats at once

```python
def find_tendencies(my_temperatures_list):
    '''Compute some stats on a list of temperatures'''

    l = len(my_temperatures_list)

    if l > 0:
        min_val = min(my_temperatures_list)
        max_val = max(my_temperatures_list)
        avg_val = sum(my_temperatures_list) / l

        # Return multiple values separated by commas (creates a tuple)
        return min_val, max_val, avg_val, l

    else:
      print('Empty input data, defaulting to None')
      return None, None, None, 0
```

Notice the multiple exit points with two or more `returns`

NESTED CALLS

A **return** value is **input parameter** to another f.



```python
def square(x):
  return x**2

def average_of_three(x, y, z):
  return (x+y+z)/3

print(average_of_three(square(10), square(20), square(30)))
```

VISIBILITY OF VARIABLES

argument: what the caller sends to the function, e.g., `'es'`

parameter: the local, received value, e.g., var. `lang`

F. are executed in a *container*, they should

- only *see* their parameters,
- not see the variables of the calling code

Try to make no assumption on who's calling

F. execute in a separated environment with *private* variables

External vars. (from the caller) may be read but not changed

```python
def func(primero, segundo):

    print(f'Inside here, primero is really first_in: {primero} and {first_in}')
    tercero = 3
    quarto = 4

    return tercero, quarto

# example call
first_in = 1
second_in = 2
first_out, second_out =  func(first_in, second_in)
```

```
Inside here, primero is really first_in: 1 and 1
```

```python
def func(primero, segundo):

    first_in = 0
    print(f'Inside here, is primero really first_in? {primero} is {first_in}')
    tercero = 3
    quarto = 4

    return tercero, quarto

first_in = 1
second_in = 2
first_out, second_out =  func(first_in, second_in)
print(f'Outside, first_in is {first_in}')
```

```
Inside here, is primero really first_in? 1 is 0
Outside, first_in is 1
```

A new `first_in` variable is available, but only inside `func` unless we specify `global` or `nonlocal` (not reccommended)

QUIZZES

QUIZ 1/4

What is the default return value for a function that does not return any value explicitly?

- `None`
- `int`
- `null`
- `str`

QUIZ 2/4!

Which of the following items are present in the function header?

- function name
- function name and parameter list
- parameter list
- return value

QUIZ 3/4!

Which of the following keywords marks the beginning of the function block?

- `fun`
- `define`
- `def`
- `function`

QUIZ 4/4!

Which of the following function definition does not return any value?

- print all integers from 1 to 100.
- return a random integer from 1 to 100.
- check whether the current second is an integer from 1 to 100.
- convert an uppercase letter to lowercase.

ANSWERS:

What is the default return value …?

- None

Which … present in the function header?

- function name and parameter list

Which … marks the beginning of the function block?

- `def`

Which … does not return any value?

- print all integers from 1 to 100