Digital Technology

Python – Statements

POLITECNICO
MILANO 1863

Edoardo Ramalli, Barbara Pernici

Material

Colab for this part: NB4

https://colab.research.google.com/drive/1DfqE2GJqtKUCJEWun_LKjuIj1M5jb-Te?usp=sharing

Call Statement: Functions

Functions are reusable pieces of code.
Functions are not run in a program until they are “called” or “invoked».

Functions are characterized by:

- A name
- Parameters (0 or more)
- A docstring (optional but recommended)
- A body
- A return statement

A step back: Mathematical functions

$Y = 2x + 1$

$Y = f(x) = 2x + 1$

f → Function name
x → Function parameter

$f(3) = 2*3 + 1 = 7$
$f(0) = 2*0 + 1 = 1$

$f(x, y) = x + y$

f → Function name
x, y → Function parameters

$f(0, 0) = 0 + 0 = 0$
$f(0, 5) = 0 + 5 = 5$

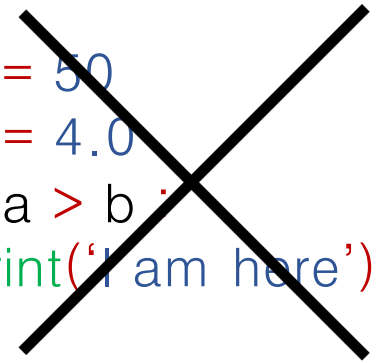
```
def my_function2(x):
    '''the function returns 2x+1'''
    print('x is ',x)
    return 2*x + 1
```

IF Statement

Python relies on the indentation to define the scope!

```
a = 50
b = 4.0
if a > b :
    print('Greater')
elif a == b :
    print('Equal')
else:
    print('I'm here!')
```

```
a = 50
b = 4.0
if a > b :
print('I am here')
```



The if body is not indented correctly

- elif stands for else if
- Remember the colon : at the end of the if, elif and else condition !
- else and elif are optional
- The body of if, else and elif cannot be empty. If you don't have a content use pass

IF Statement: In-line

Let's re-write them in a more pythonic way.

```
a = 50
b = 4.0
if a > b :
    print('Greater')
else:
    print('I'm here!')
```



```
a = 50
b = 4.0
print('Greater') if a > b else print('I'm here!')
```

```
a = 50
b = 4.0
if a > b :
    print('Greater')
```



```
a = 50
b = 4.0
if a > b : print('Greater')
```

For Statement

It is used to iterate over a sequence.
A sequence could be a string, list, set, tuple or a dictionary.

```
my_list = ['blue', 'red', 'green']

for color in my_list:
    print(color)
```

At each iteration of the for loop the temporary variable 'color' takes a different value of the list 'my_list' starting from the item at index 0 to the last item.

```
my_list = ['blue', 'red', 'green']

for my_var in my_list:
    print(my_var)
```

The name of the temporary variable is up to you. You can choose the name that you prefer.

For Statement

It is the same as before but in a more traditional way.

```
my_list = ['blue', 'red', 'green']
for index in range(0, len(my_list)):
    print(my_list[index])
```

In this case the range function creates a sequence of number from 0 to len(my_list) – 1. In this case, from 0 to 2.

At each iteration, the variable 'index' takes a value from the sequence [0, 1, 2]. We use this variable as index to access the content of the list 'my_list' in position 0,1 and 2 using the operator square brackets [ ].

- The pass keywork has the same behaviour in the case of the if statement.
- The break keywork allows to immediately exit from the for loop.
- The continue keywork allows to immediately jump to the next cycle

Homework

Input: a positive Integer from the user
Output: print the first 8 numbers of the Fibonacci sequence

Define a function for computing the sequence

It is not mandatory. It is not graded.

Optional topics

While Statement

The behaviour is quite similar to the for loop.
Use the while loop when you don't know a priori the number of iterations.

```
my_list = ['blue', 'red', 'green']
index = 0
while index < len(my_list):
        print(my_list[index])
        index += 1                          # index = index + 1
```

As in the for loop we can use the pass, break and continue keyword
with the same behaviour.

List comprehension

Creating lists can be tiresome

My_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Compact form to create lists in python

list_variable = [x for x in iterable]
list_variable = [x for x in iterable if condition]

Mathematical forms

R = {x : x in {0 ... 9}}
S = {x^2 : x in {0 ... 9}}
M = {x | x in S and x even}

R = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
S = {0, 1, 4, 9, 16, 25, 36, 49, 64, 81}
M = {0, 4, 16, 36, 64}

solutions in colab