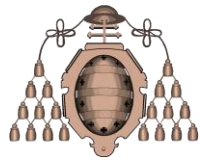


## Table of contents

- 2.1 Problem abstraction for programming. Basic concepts.
- 2.2 Variables, expressions, assignment
- **2.3 Console input/output**
- 2.4 Basic structures for control flow handling: sequential, choice and repetitive.
- 2.5 Definition and use of subprograms and functions. Variable scope.
- 2.6 File input / output
- 2.7 Basic data types and structures: arrays



## Standard input

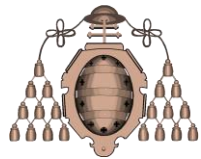
In order to enter data using the standard input (i.e. the keyboard), the function `input` will be used.

- It returns a string with the characters typed by the user on the keyboard.

```
input_data = input()
```

- It allows us to show an informative message to the user.

```
input_data = input("Type your name:")
```

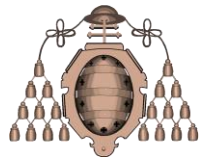


## Standard input

- The read string can be converted into another data type.

```
int_number = int(input("Type an integer:"))  
real_number = float(input("Type a real number:"))
```

**A conversion error may appear**

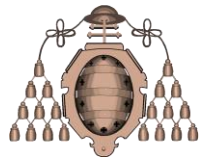


## Standard output

In order to show data on the standard output (i.e. the screen), we will use the `print` function:

```
print(expr1, expr2, ..., sep=" ", end="\n")
```

- `expr`: it can be a string, an integer, a float, a Boolean, a variable or any expression in Python.
- `sep`, `end`: they are optional (they take their default values if we do not specify anything).



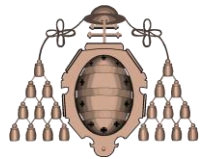
## Standard output

```
print()  
print("Hello world")  
print(25)  
print(3.14)  
print(2+2)  
print(5*0.25)
```



```
Hello world  
25  
3.14  
4  
1.25
```

- Double quotes are not shown in the output
- A line break is introduced by default → `end="\n"`



## Standard output

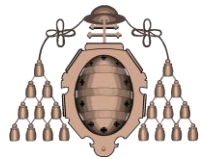
In order to avoid the default line break, a value has to be given to the argument `end`.

- It is very common to use the value `end=" "`; for example, to print in the same line.

```
print("Hello", end=" ")  
print("world")  
print("Bye bye")
```



```
Hello world  
Bye bye
```



## Standard output

- Several values or expressions can be printed with the same function `print` by separating them with commas

```
print("I am", 20, "years old")
```



```
I am 20 years old
```

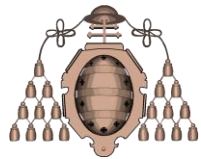
Commas are replaced by the default separator  $\rightarrow$  `sep=" "`

- In order to use a different separator, a value has to be given to the argument `sep`

```
print(1, 2, 3, 5, sep=" < ")  
print(3, 1416, sep=" . ")
```

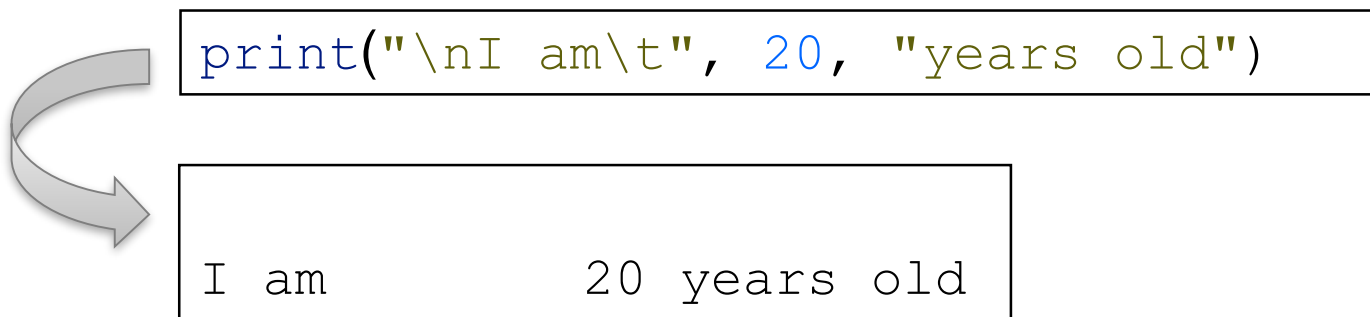


```
1 < 2 < 3 < 5  
3 . 1416
```



### Standard output

- Strings used in a `print` function can contain special characters, which will have to be preceded by a backslash character → \



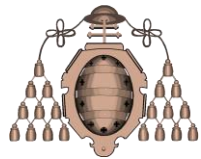
A diagram illustrating the execution of a `print` statement. A grey curved arrow points from the code box to the output box. The code box contains the statement `print("\nI am\t", 20, "years old")`. The output box shows the result: a line break followed by "I am", a tabulation followed by "20", and then "years old".

```
print("\nI am\t", 20, "years old")
```

I am                      20 years old

- The character `\n` introduces a line break
- The character `\t` introduces a tabulation





## Standard output

Strings allow advanced formatting techniques using the `.format()` function and replacement fields:

- `{}`: replaces the field by the parameter in `.format()`, in sequential order.
- `{i}`: replaces the field by the  $i^{\text{th}}$  parameter in `.format()`.
- `{i:.mf}`: replaces the field by the  $i^{\text{th}}$  parameter in `.format()` as a float value using  $m$  decimal digits.

It admits many other possibilities: hexadecimal, binary, exponent notation, tabulation...

```
print("I am {} years old".format(25))
```

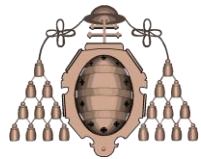


I am 25 years old



```
print("I am {0} years old, I am {1:.2f} meters tall  
and I am from {2}.".format(25, 1.8, "Tehran"))
```

I am 25 years old, I am 1.80 meters tall and I am  
from Tehran.



## Standard output

Alternatively, Python 3 includes f-strings.

- The syntax is similar to `.format()`, but strings are defined with a leading `f`.
- Basically, any `{}` field is interpreted at run time.
- Can convert values to formatted float numbers as in `.format()`.

```
print(f"I am {6 * 5} years old")
```



```
I am 30 years old
```

```
age = 25  
height = 1.8  
birth_place = "Tehran"  
print(f"I am {age} years old, I am {height:.2f}  
meters tall and I am from {birth_place}.")
```



```
I am 25 years old, I am 1.80 meters tall and I am  
from Tehran.
```