Introduction to Python

Block 1. Part 2.

Introduction to Objects
Examples: Strings, Files

Variables, expressions and statements

- **Formal Language**
    - Strict rules of Syntax:
        - Tokens: Basic elements
            - Variables
        - Structure: How elements are arranged
            - Expressions, statements, control structures, conditionals…

Variables, expressions and statements

- **Formal Language**
  - Strict rules of Syntax:
    - Tokens: Basic elements
      - Variables

[a **built-in type** is a data type for which the programming language provides **built-in** support]

Built-in types:

- Integer
- Float
- Boolean (True/False)
- None
- String

- Function
- List
- Dictionary
- Set
- ...

Variables, expressions and statements

- **Formal Language**
  - Strict rules of Syntax:
    - Tokens: Basic elements
      - Variables

[a **built-in type** is a data type for which the programming language provides **built-in** support]

Built-in types:

- Integer
- Float
- Boolean (True/False)
- None
- String

- Function
- List
- Dictionary
- Set
- ...

Strings

- A **String** is a sequence of characters.

- Creation with:
  - Double quotes: “
  - Single quotes: ‘
  - Triple quotes: “”” (allows span in multiple lines)

```
>>> word = "bioinformatics"
>>> paragraph = """This is the first block
... of the subject "Introduction to Python" """
```

Strings

- The backslash (\) character is used to escape characters that otherwise have a special meaning, such as newline, backslash itself, or the quote character.

```
>>> a="hello\nworld"
>>> print(a)
hello
world
>>> a = "asdjah\"adsas"
>>> print(a)
asdjah"adsas
```

Strings

- String operators:
  - **Concatenate: +**

```
>>> word1 = "biomedical"
>>> word2 = "informatics"
>>> word1 + word2
'biomedicalinformatics'
```

  - **Replicate: ***

```
>>> word = "spam"
>>> word*4
'spamspamspamspam'
```

  - **Indexing: []**
  - **Slicing: [:]**

Strings

- Index: position in the sequence. Strings are ordered!

```
>>> word = ”BIOINFORMATICS”
```

B	I	O	I	N	F	O	R	M	A	T	I	C	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> word[3]
'I’
>>> word[-2]
'C'
```

Negative indices

Strings

- Index: position in the sequence.

```
>>> word = ”BIOINFORMATICS”
```

B	I	O	I	N	F	O	R	M	A	T	I	C	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Slice: segment of a string. `[Start index:End index+1:step]`

```
>>> word[0:4]
'BIOI’
>>> word[3:]
'INFORMATICS’
>>> word[:-1]
'BIOINFORMATIC’
>>> word[::2]
'BONOMTC'
```

Strings

- Strings are inmutable!

```
>>> word[3]="B"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item
assignment
```

Strings

- len: built-in function to get the length of a string

```
>>> len(word)
14
```

Strings

- Traversal of a String

```
>>> i = 0
>>> while i<len(word):
...     print(i,":",word[i])
...     i+=1
...
0 : b
1 : i
2 : o
3 : i
4 : n
5 : f
6 : o
7 : r
8 : m
9 : a
10 : t
11 : i
12 : c
13 : s
```

Strings

- Traversal of a String: **for … in…**

```
>>> for character in word:
...     print(character)
...
B
I
O
I
N
F
O
R
M
A
T
I
C
S
```

Strings

- **in** operator:

```
>>> word = "bio"info"rmatics"
>>> "info" in word
True
```

- Comparing strings:

```
>>> string1 = "abcd"
>>> string2 = "BCD"
>>> string1 == string2
False
>>> string1 > string2
True
```

Uppercase letters come before lowercase letters!

( )

Introduction to OOP (Object Oriented Programming)

Block 1. Part 2.

Introduction to OOP
(Object Oriented Programming)

Functional/Structured programming

- **Variables**
- **Functions**
  - Group of statements: easier to read programs and debug.
  - Smaller programs by eliminating repetitive code.
  - Divide a long program into functions. Decomposition of a problem into sub-problems and assemble them in a workflow script.
  - Reuse of the same functions in several programs.
- **Modules/Libraries** that are collections of variables and functions.

Object Oriented Programming

Introduction to OOP
(Object Oriented Programming)

Object Oriented Programming

Attributes: Variables that define the state of the object

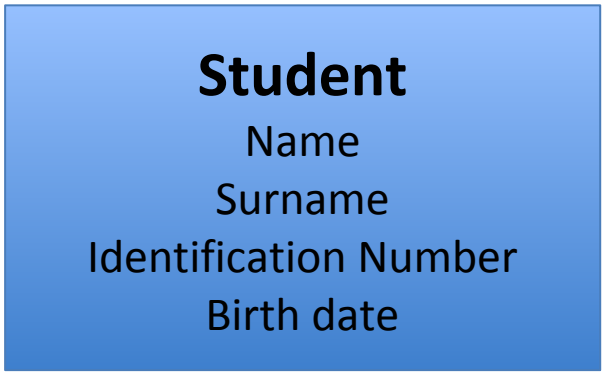
Methods: Define the behavior of the object.
A method is a function associated to an object.

Object: Concept in the real world.

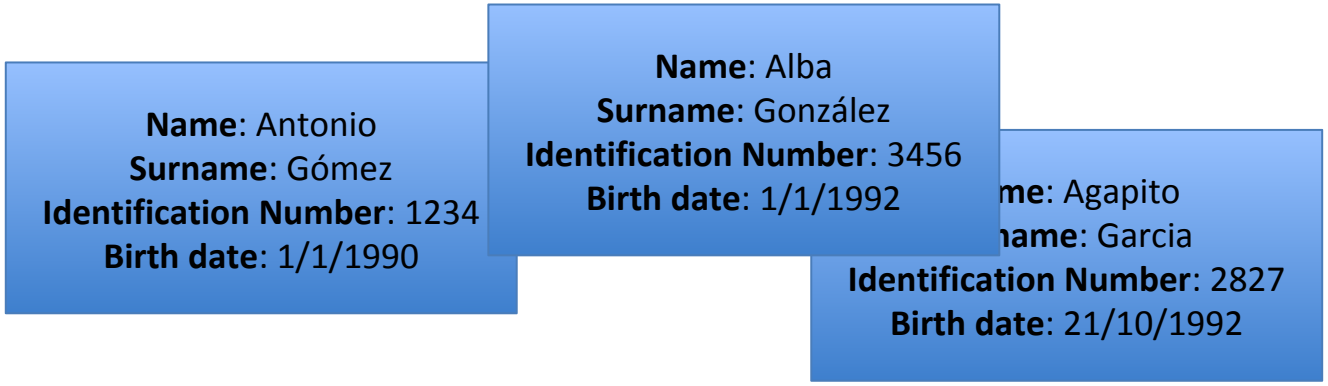
Class vs **Object Instance**

Introduction to OOP
(Object Oriented Programming)

Class: Defines the structure: attributes and methods

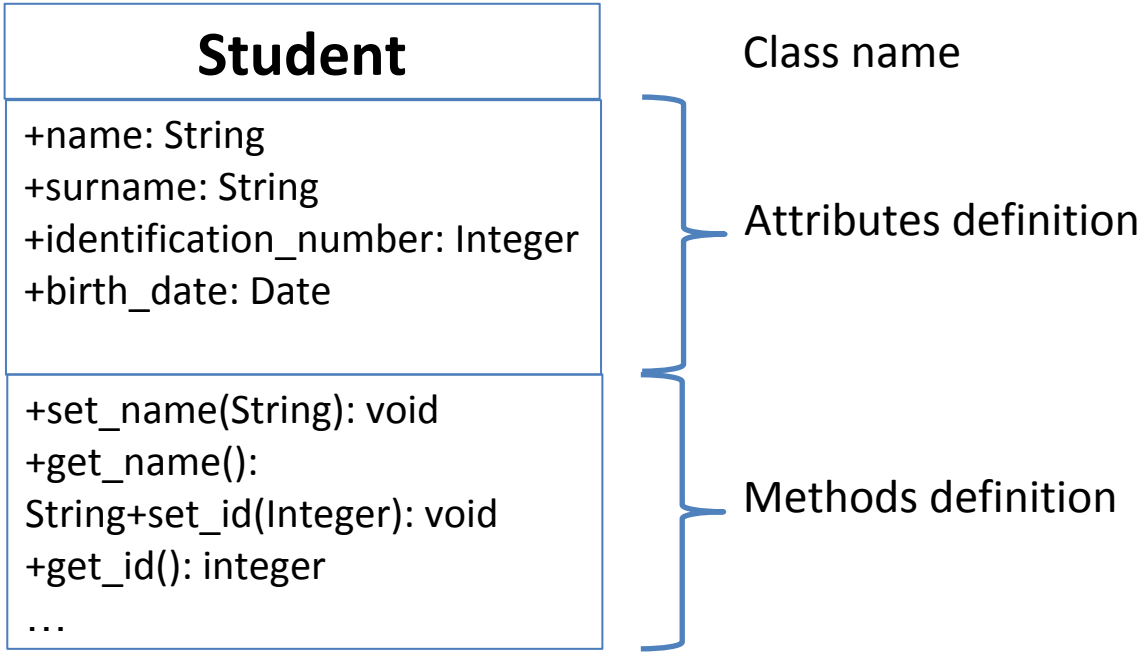


Instances: Specific realization of any object.
"objects" that exist in a given program execution



Introduction to OOP (Object Oriented Programming)

Class representation: UML diagram

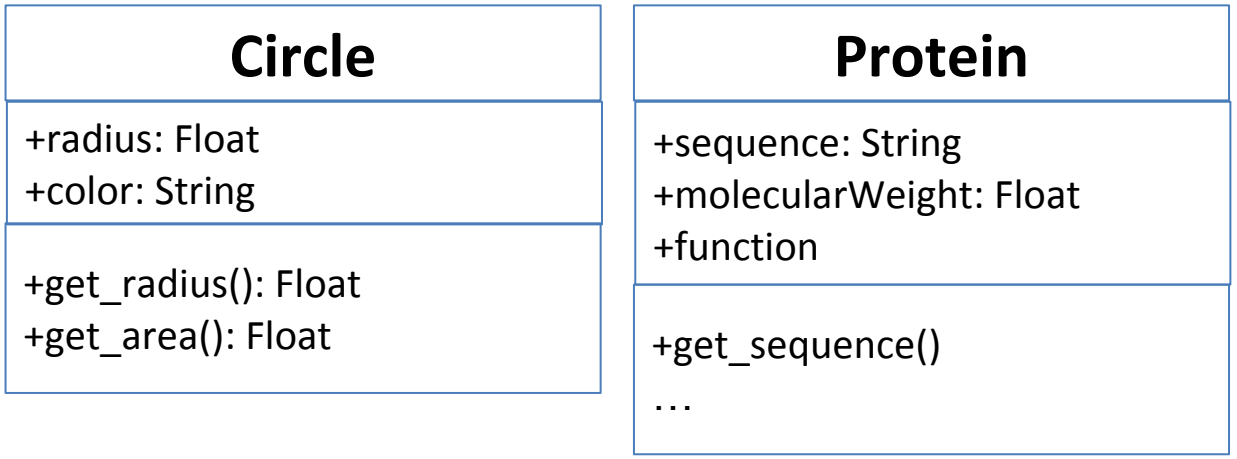


To call the method of an object:

```
object_name.method()
```

Introduction to OOP
(Object Oriented Programming)

Examples



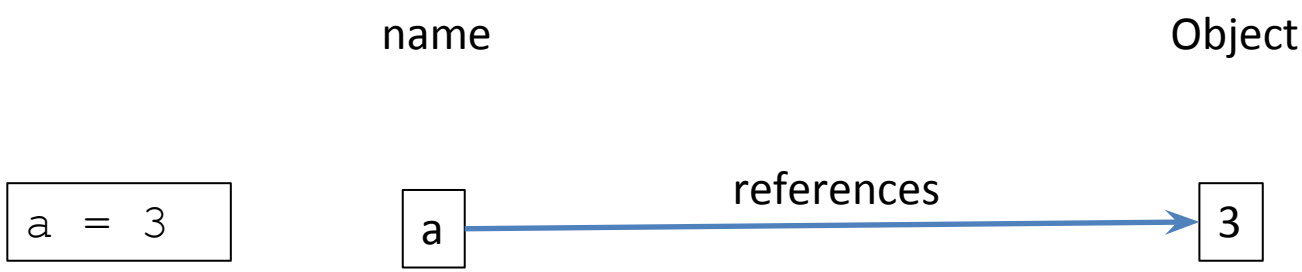
Introduction to OOP
(Object Oriented Programming)

In Python, everything is an object!

strings, files, modules,…

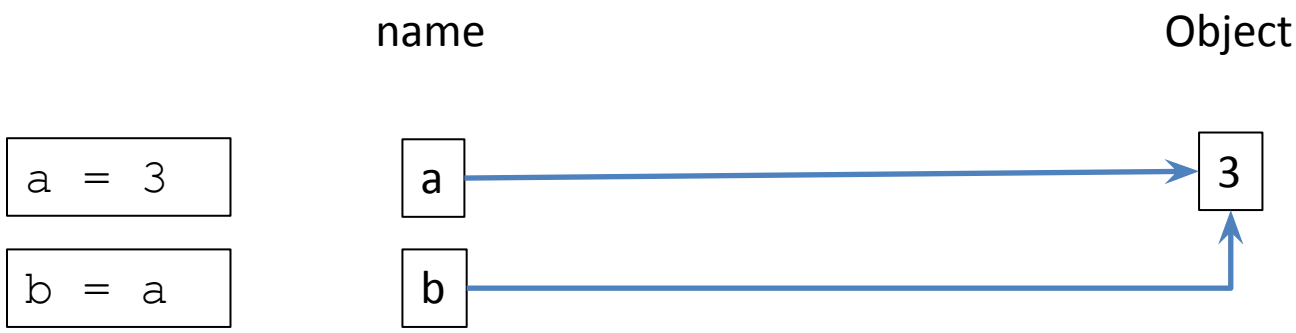
Variable names are references to objects

- **Variable assignment**: Create a new variable and assign a value. In Python, we assign a reference of an object to a name.



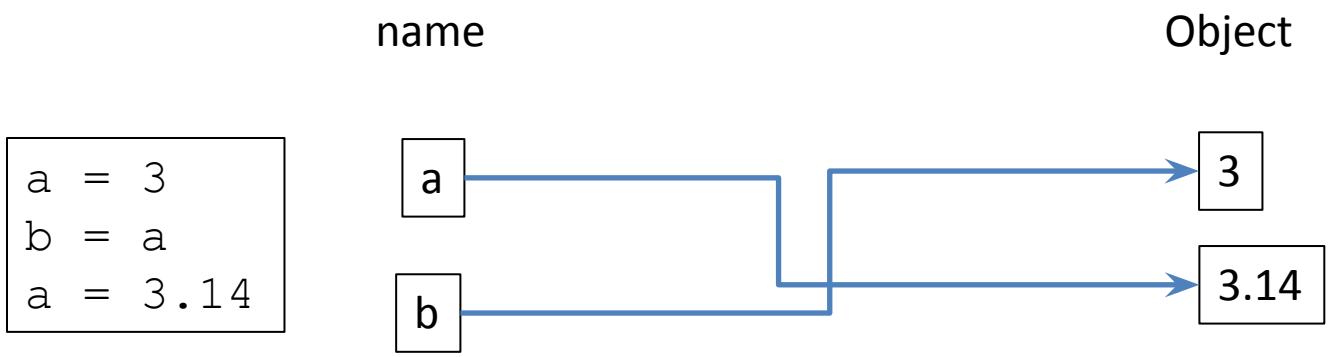
Variable names are references to objects

- **Variable assignment**: Create a new variable and assign a value. In Python, we assign a reference of an object to a name.



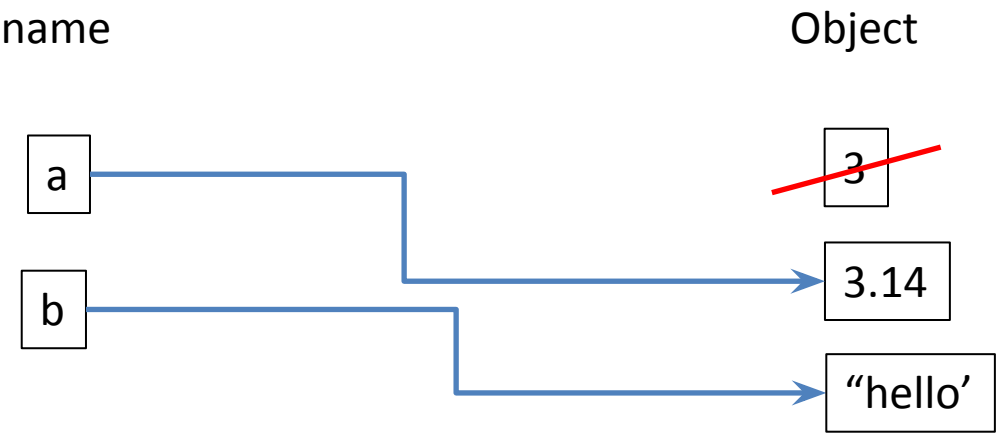
Variable names are references to objects

- **Variable assignment**: Create a new variable and assign a value. In Python, we assign a reference of an object to a name.



Variable names are references to objects

```
a = 3
b = a
a = 3.14
b = “hello”
```

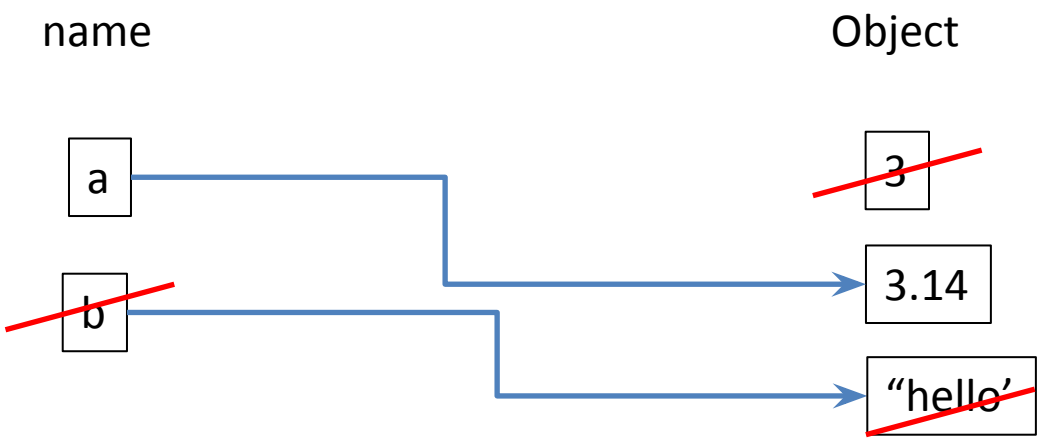


- **Garbage Collection:** When an object is not referenced by any variable, it is automatically destroyed.

Variable names are references to objects

- **del statement:** Removes a variable from the current scope.

```
a = 3
b = a
a = 3.14
b = "hello"
del b
```



As we remove variable b, the object "hello" is destroyed by the garbage collector.

/ ( )

Strings

- String methods:
  - **upper**
  - **lower**
  - **find**
  - **split**
  - **count**
  - **strip, lstrip, rstrip**
  - **ljust, rjust, startswith, endswith,…**

To see all methods of a string:

```
>>> word = "bioinformatics"
>>> dir(word)
```

Strings

```
>>> sentence = "This is a test example sentence.\nThis is
the second sentence.\n"
>>> sentence.split()
['This', 'is', 'a', 'test', 'example', 'sentence', '.',
'This', 'is', 'the', 'second', 'sentence.']
>>> sentence.split("\n")
['This is a test example sentence.', 'This is the second
sentence.']
>>> sentence.strip()
'This is a test example sentence.\nThis is the second
sentence.'
>>> sentence.strip("\n.")
'This is a test example sentence.\nThis is the second
sentence'
>>> sentence.replace("\n"," ")
'This is a test example sentence. This is the second
sentence. '
>>> word.capitalize().swapcase()
'bIOINFORMATICS'
```

String methods

https://docs.python.org/3/library/stdtypes.html

https://docs.python.org/3.1/library/string.html

Strings

- Formatting strings: **Concatenation**
  - With the operator **+**

```
>>> name = "Roger"
>>> "The name of the student is "+ name
'The name of the student is Roger'
```

  - Only two strings can be concatenated:

```
>>> "The result of 2 + 2 is " + 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

```
>>> "The result of 2 + 2 is " + str(4)
'The result of 2 + 2 is 4'
```

Strings

- Formatting strings: operator **%**
  - % followed by a letter

%s string
%d integer
%f float
%e scientific notation
%E scientific notation

Indicates the conversion to be performed. Python automatically performs the necessary conversion

  - Syntax:

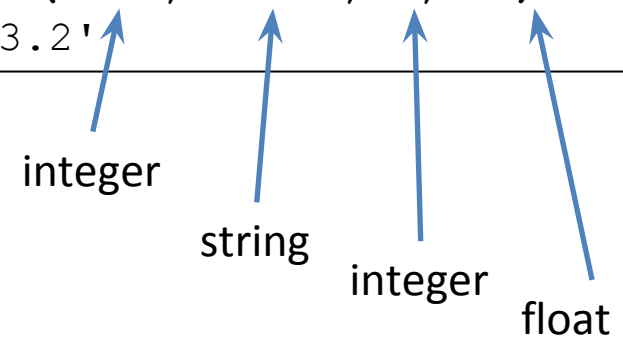
integer integer
string
float

```
"Student ID %d. Name %s. Age %d. Rank %f"  %(1272,"Toni",23,3.2)
'Student ID 1272. Name Toni. Age 23. Rank 3.200000'
```

Strings

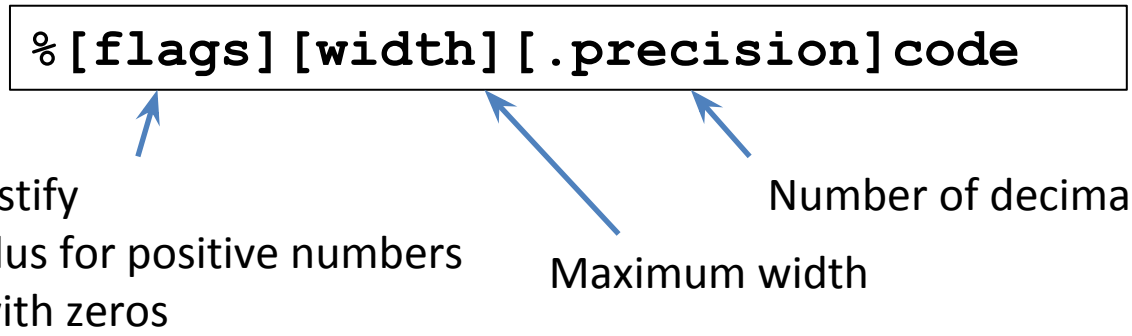
- Formatting strings: operator **%**
  - All basic objects have a string description (they can be converted directly to a String object).
    - **%s** can be used for all basic objects

```
"Student ID %s. Name %s. Age %s. Rank %s"  %(1272,"Toni",23,3.2)
'Student ID 1272. Name Toni. Age 23. Rank 3.2'
```



Strings

- Formatting strings: operator **%**
  - Extended formatting syntax for numbers




```
print("Example of formatting: %.3f" %(3.141592653589793))
Example of formatting: 3.142
```

```
print("Example of formatting: %+010.2f" %(3.141592653589793))
Example of formatting: +000003.14
```

```
>>> print("Example of formatting: %+.2e" %(3.141592653589793))
Example of formatting: +3.14e+00
```

Strings

- Formatting strings: operator **%**

```
my_string = "The name of the student is %s”

print(my_string %”Roger”)
print(my_string %”Nuria”)
print(my_string %”Alfons”)
```

Strings

- .format

```
my_string = "The name of the student is {name}. The surname is {surname}”

my_string.format(name=”Roger”, surname=”Puig”)
```

Some built-in functions

- **dir():** return the available names (variables and methods or functions)

- **id()**: Return the identity of an object. This is guaranteed to be unique among simultaneously existing objects.

Introduction to Files

- File object
  - Get a file object with the function **`open`**

```
open(name[, mode[, buffering]]) -> file object
```

  - Different modes:
    - ‘r’: read
    - ‘w’: write
    - ‘a’: append

Introduction to Files

- File object
  - Methods:
    - readlines
    - readline
    - writelines
    - write
    - Operator **in**
    - **. . .**

```
fd = open(“my_file.txt”,”r”)

for line in fd:
    print(line)

fd.close()
```

Introduction to Files

- **with … as …**

```
with open(“my_file.txt”,”r”) as fd:
   for line in fd:
      print(line)
```

Introduction to Files

https://docs.python.org/3.6/tutorial/inputoutput.html#reading-and-writing-files

Exercises. Block 1. Part 2.

Create a python script called **`NIE_exercise_block1_part2.py`** with the following functions:

1) Given a multi-line protein FASTA file (*filename*), returns an integer corresponding to the total number of protein sequences having a relative frequency higher or equal than a given threshold for a given *residue*.

```
count_sequences_by_residue_threshold(filename,
                                     residue,
                                     threshold=0.03)
```

2) Given a protein FASTA file (*filename*), save on a file (*output_filename*) the following susmmary, one protein by line: protein identifier, the first *N*-aminoacids, the last *M*-aminoacids and the absolute frequency in the protein of each of the first N-aminoacids and the last M-aminoacids. The first three fields must be separated by a tabulator, and the absolute frequency of the residues must have the format RESIDUE:frequency and must be separated by comma. The first line must be a line with a summary formatted as follows (replace the values FILENAME, N and M with the corresponding values) (see example on next slide)

The file FILENAME contains X proteins. Here we show the code of the protein, the first N aminoacids of each protein and the last M aminoacids

```
print_sequence_tails(filename,
                     output_filename,
                     first_n=10,
                     last_m=10)
```

Exercises. Block 1. Part 2.

2) Given a protein FASTA file (*filename*), save on a file (*output_filename*) the protein identifier, the first *N*-aminoacids, the last *M*-aminoacids and the absolute frequency in the protein of each of the first N-aminoacids and the last M-aminoacids. The fields must be separated by a tabulator, and one protein by line. The first line must be a line with a summary formatted as follows (replace the values FILENAME, N and M with the corresponding values).

Input file:

```
>PROT1
EFTRPTSTWSAAALMTRSSSTRWSPD
>PROT2
SSTPLRRSTPAWEEFGLMCCDPRS
>PROT3
ATRSLEWKSTPW
```

Output file:

```
# The file FILENAME contains 3 proteins. Here we show the code of the protein, the first 3 aminoacids of each protein and the last 5 aminoacids.

PROT1 EFT     RWSPD       E:1,F:1,T:5,R:3,W:2,S:6,P:2,D:1
PROT2 SST     CDPRS       S:4,T:2,C:1,D:1,P:3,R:3
PROT3 ATR     KSTPW       A:1,T:2,R:1,K:1,S:2,P:1,W:2
```