Introduction to Python

Session 2

Introduction to objects.
Strings and files

Introduction to OOP

(Object Oriented Programming)

Introduction to OOP

Functional/Structured programming:

- <u>**Variables**</u>
- <u>**Functions**</u>
  - **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 subproblems and assemble them in a workflow script.
  - **Reuse** of the same functions in several programs.
- <u>**Modules/Libraries**</u> that are collections of variables and functions.

Object Oriented Programming: Objects

Introduction to OOP

Object Oriented Programming:

Object: A combination of variables, functions, and data structures.

They can represent a specific entity of the real world or a more abstract concept.

Objects

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.

Class vs **Object Instance**

Introduction to OOP

This are attributes

Student
Name
Surname
Identification Number
Birth date

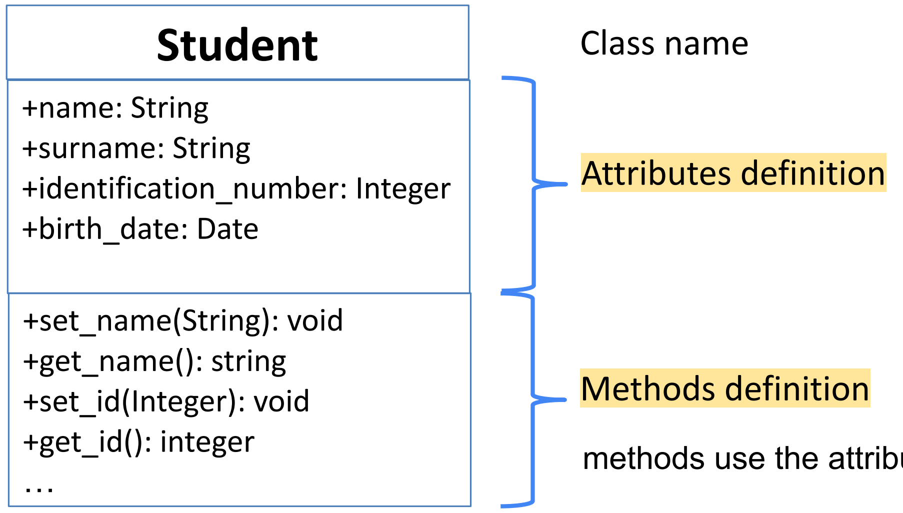
Class: Defines the structure: attributes and methods

Instances: Specific realization of any class. Objects that exist in a given program execution

Name: Antonio	**Name**: Alba	**Name**: Agapito
Surname: Gómez	**Surname**: González	**Surname**: Garcia
Identification Number: 1234	**Identification Number**: 3456	**Identification Number**: 2827
Birth date: 1/1/1990	**Birth date**: 1/1/1992	**Birth date**: 21/10/1992

Introduction to OOP

Class representation: UML diagram



methods use the attributes to generate the functiion

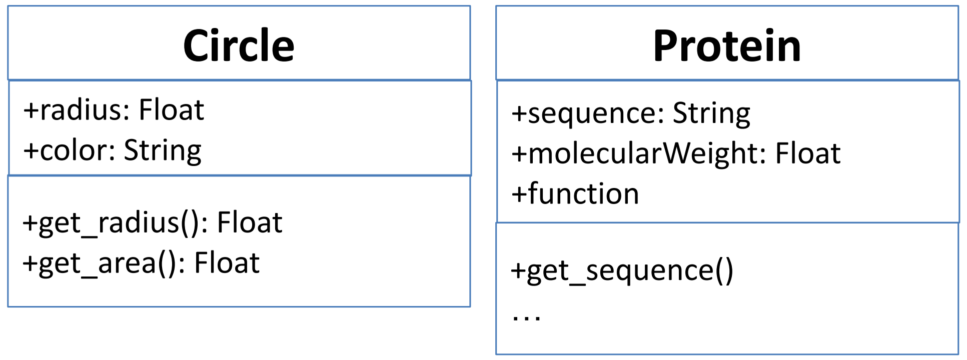
To call the method of an object:

```
object_name.method()
```

the way to access things via instances

Introduction to OOP

Examples of classes



Introduction to OOP

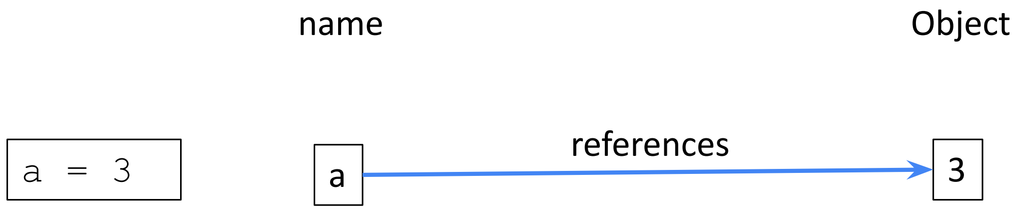
In Python, **everything** is an object !

strings, files, modules, integers, floats, …

when typing integers, floats, a class is an object

Variable names are references to objects.

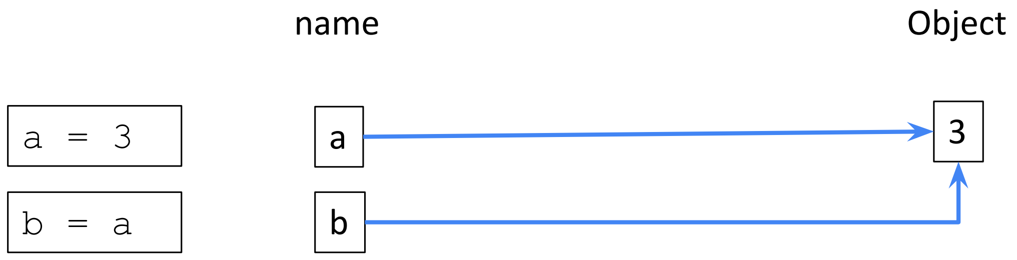
Variable assignment: Create a new variable and assign a value. In Python, we assign a reference of an object to a name.



creates an object of class integer, so it's an instance of an integer and has a value of 3, creating a name differenciating the object.

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.

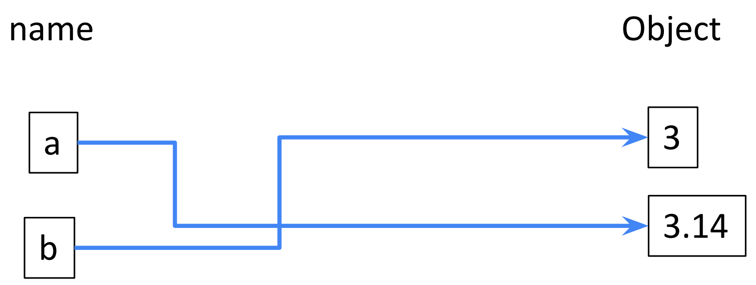


now a and b refer to the same object, so same value of 3

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.

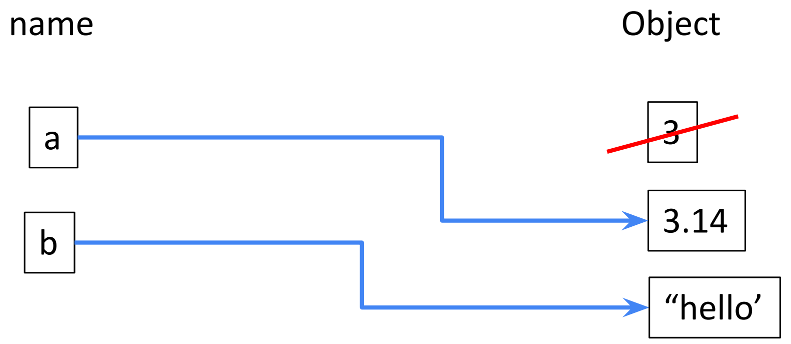
```
a = 3
b = a
a = 3.14
```



Variable names are references to objects.

now python creates a new object of class string,of a vlue = "hello". The object 3 whhc was an instance of value 3, has no variable, so the arbage collector removes the objectes in memory that have lsot their variable

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

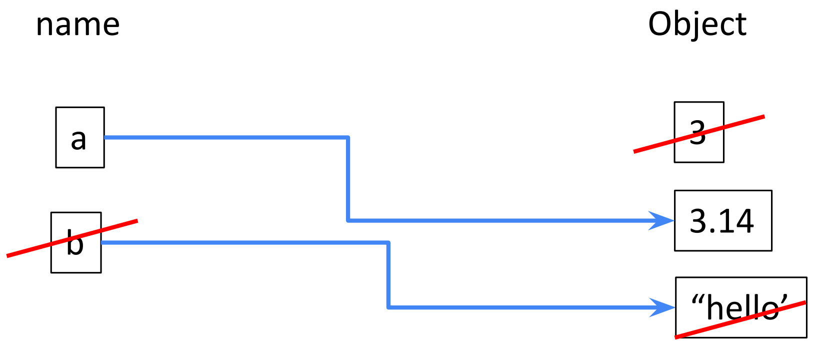


Garbage Collection: When an object is not referenced by any variable, it is **automatically destroyed.**

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
```



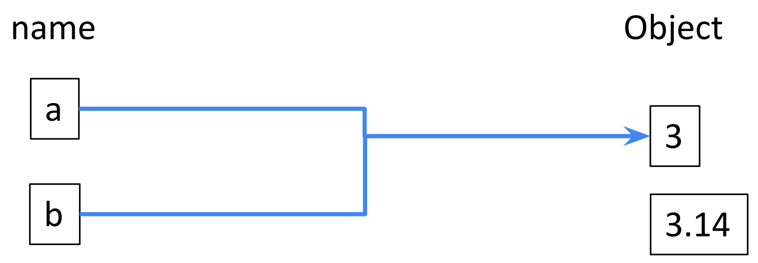
As we remove variable b, the object "hello" is destroyed by the garbage collector.

Introduction to OOP

Some built-in functions

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

```
a = 3
b = a
```



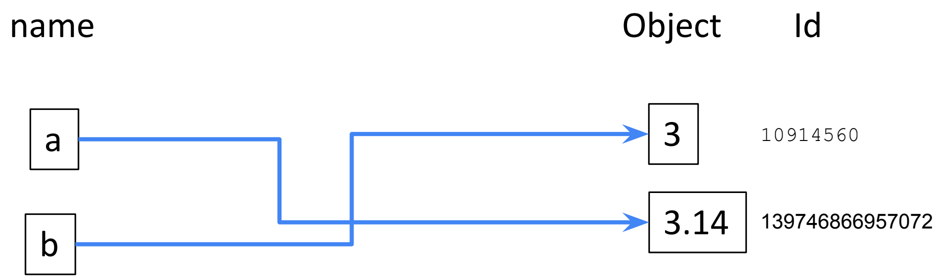
```
>>>  id(a)
10914560
>>> id (b)
10914560
>>> id (3)
10914560
```

Introduction to OOP

Some built-in functions

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

```
a = 3
b = a
a = 3.14
```

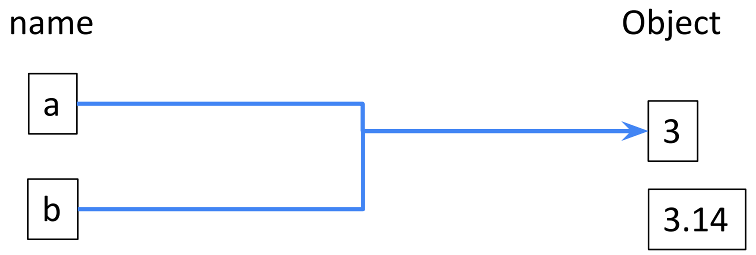


```
>>>  id(a)
139746866957072
>>> id (b)
4461948008
```

Some built-in functions

is operator: Returns **True** when two variables refer to the same object.

```
a = 3
b = a
```



```
>>> a is b
True
>>> c = 5
>>> a is 5
False
```

To check is a variable is true or not instead of an = is better to use is

Introduction to OOP

Singletons: Design pattern that ensure that a class can be only instantiated **once**.

None

True/False

Classes that can only have one instance

```
a = True
>>> a is True
True
>>> a is None
False
```

For efficiency:

Small integers

Short strings

Introduction to OOP

Some built-in functions

dir(): return the available names (variables and methods or functions) of an object instance

For now everythign is an object, we have classes of this object and instances

Variables, expressions and statements

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

Built-in types:

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

this two are singletons (so better use is)

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

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

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

first position is index 0. Also we can have negative index,

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'
```

the 4 is not included

if we do not put any number next to the colon, then goes to the end

the two is the step, starts fomr the begining to the last position, jumping two positions

this might be useful when having odd or even numbers

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

The pythonic way!

- 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 = "bioinformatics"
>>> "info" in word
True
```

- Comparing strings:

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

Uppercase letters come before lowercase letters!

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

help() is a special function that returns a string with the documentation associated to an element (class definition, method documentation, attribute,....)

```
>>> word = "bioinformatics"
>>> help(word.split)
```

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()   removes end of lines at the begginig and at the end
'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’
```

Strings

```
>>> sentence = "This is a test example sentence.\nThis is
the second sentence.\n"

>>> sentence.replace("\n"," ")
'This is a test example sentence. This is the second
sentence. `
```

replacing end of lines by spaces

first letter in upercase

```
>>> word.capitalize().swapcase()
'bIOINFORMATICS'
```

now swap the first letter to lowercase and the rest in upercase!

Strings

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 string integer 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'
```

integer

string

integer

float

Strings

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

```
%[flags][width][.precision]code
```

- − left justify
- + add plus for positive numbers
- 0 pad with zeros

Maximum width

Number of decimals

```
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

- Formatting strings: **.format()**

placeholder

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

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

expectes the same numbers of parameters as placehlders int his form that you have in hthe string

```
my_string = "The name of the student is {}. The surname is {}”
my_string.format(“Roger”, “Puig”)
```

empty place holders

Strings

- Formatting strings: **f-strings**

Using the format function with previously defined variables:

```
name = "Roger"
surname = "Puig"

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

in this case you don't call any function, insie the brakets will contain the variables you previously said.

Strings

- Formatting strings: **f-strings**

Using the format function with previosuly defined variables:

```
name = "Roger"
surname = "Puig"

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

prefixing a string: modifies how this string is created

Strings

- String prefix

  - **r: raw** (do not need to escape)
  - **b**: **bytes**
  - **u**: **unicode** (the default one)
  - **f**: **format**

```
name = "Roger"
surname = "Puig"

my_string = r"The name of the student is \"{name}\". The surname is \"{surname}\""
my_string = rf"The name of the student is \"{name}\". The surname is \"{surname}\""
my_string = b"The name of the student is \"{name}\". The surname is \"{surname}\""
my_string = u"The name of the student is \"{name}\". The surname is \"{surname}\""
```

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()
```

in read mode

to iterate through all the lines in the file

ends when the file ends

Introduction to Files

- **with … as ...**

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

when the with block ends, the file is closed. so no close() operation needed

most common way to use files in python

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 ***<uID>*_S02.py** with the following functions:

Input file: multi-line FASTA file:

Input file example:

```
>PROT1
MGIKGLTGLLSENAPKCMKDHEMKTLFGRKVAIDASMSIYQFLIAVRQQDGQMLMNESGDVTSHLMGFFY
RTIRMVDHGIKPCYIFDGKPPELKGSVLAKRFARREEAKEGEEEAKETGTAEDVDKLARRQVRVTREHNE
ECKKLLSLMGIPVVTAPGEAEAQCAELARAGKVYAAGSEDMDTLTFHSPILLRHLTFSEAKKMPISEIHL
DVALRDLEMSMDQFIELCILLGCDYLEPCKGIGPKTALKLMREHGTLGKVVEHIRGKMAEKAEEIKAAAD
EEAEAEAEAEKYDSDPENEEGGETMINSDGEEVPAPSKPKSPKKKAPAKKKKIASSGMQIPEFWPWEEAK
QLFLKPDVVNGDDLVLEWKQPDTEGLVEFLCRDKGFNEDRVRAGAAKLSKMLAAKQQGRLDGFFTVKPKE
PAAKDAGKGKGKDTKGEKRKAEEKGAAKKKTKK
>PROT2
MGIKGLTQVIGDTAPTAIKENEIKNYFGRKVAIDASMSIYQFLIAVRSEGAMLTSADGETTSHLMGIFYR
TIRMVDNGIKPVYVFDGKPPDMKGGELTKRAEKREEASKQLVLATDAGDAVEMEKMNKRLVKVNKGHTDE
CKQLLTLMGIPYVEAPCEAEAQCAALVKAGKVYATATEDMDSLTFGSNVLLRYLTYSEAKKMPIKEFHLD
KILDGLSYTMDEFIDLCIMLGCDYCDTIKGIGAKRAKELIDKHRCIEKVIENLDTKKYTVPENWPYQEAR
RLFKTPDVADAETLDLKWTQPDEEGLVKFMCGDKNFNEERIRSGAKKLCKAKTGQTQGRLDSFFKVLPSS
KPSTPSTPASKRKVGCIIYLFLYF
...
```

Exercises. Block 1. Part 2.

Create a python script called ***<uID>*_S02.py** with the following functions:

1) Given a multi-line protein FASTA file (stored in a file with path defined *filename*), **returns a float** corresponding to the **ratio** of proteins in the fasta file having a **relative frequency** higher or equal than a given threshold provided as an argument named “relative_threshold” and having an **absolute frequency** of the same residue higher or equal than a given threshold provided as an argument named “absolute_threshold” for a given *residue*. The function should be named as follows, with the same arguments definition:

```
get_proteins_ratio_by_residue_threshold(filename,
     residue,relative_threshold=0.03, absolute_threshold=10)
```

need the file of the path, need a letter

Exercises. Block 1. Part 2.

2) Given a protein FASTA file (*filename*), save on a output file named *output_filename* the protein identifier, the first *N*-aminoacids, the last *M*-aminoacids and the absolute frequency in the protein of all the aminoacids found in the protein (the aminoacids that do not appear in the protein should not be shown). The fields must be separated by a tabulator, and one protein by line.

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

Exercises. Block 1. Part 2.

2) Given a protein FASTA file (*filename*), save on a output file named *output_filename* the protein identifier, the first *N*-aminoacids, the last *M*-aminoacids and the absolute frequency in the protein of all the aminoacids found in the protein (the aminoacids that do not appear in the protein should not be shown). The fields must be separated by a tabulator, and one protein by line.

Input file:

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

Output file:

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