Introduction to Python

Session 06

Modules
Namespaces
sys
os

Organizing the code

Modules
Packages

Modules

- Divide the python code
  - Definition of classes, functions, etc
  - Divide the code in different python files with a logic structure.
  - Each python file ending with .py is a module.

Modules

- **module**: file containing Python definitions and statements.
  - file name: module name
    - The module name can be accessed with the special attribute __name__
  - file suffix: .py
  - Modules can be executed as scripts:
    - **`if __name__ == "__main__":`**

Importing modules

- When importing a module with the **import** statement, the module file is looked for in the directories listed in **sys.path** by order. When the module is found, it stops looking for it.

```
>>> import sys

>>> print(sys.path)

['', '/Users/xavi/miniconda3/lib/python39.zip',
'/Users/xavi/miniconda3/lib/python3.9',
'/Users/xavi/miniconda3/lib/python3.9/lib-dynload',
'/Users/xavi/miniconda3/lib/python3.9/site-packages']
```

It starts looking in the current working directory

Default third--party package directories

Namespace

- **Namespace**: mapping from names to objects (also called context)
  - A "space" that holds a bunch of names
    - variables
    - functions
    - class definitions
    - …
  - Examples:
    - Global names in a module
    - Local names in a function
    - The set of attributes of a Class
    - Set of built-in names

Namespace

- **Namespace**: mapping from names to objects (also called context)
  - There is no relation between names in different namespaces: each namespace is completely **isolated**
  - For this reason, to access names in a namespace, a prefix should be used:
    - module_name.function_name
    - class_instance.attribute_name
  - Two different modules can have the same names within them.

Namespace Scope

- **Scope of a namespace**: Textual region of a Python program where a namespace is directly accessible (i.e. accessed without a prefix)
  - i.e. when using an unqualified reference to a name, Python attempts to find the name in the namespace.
  - Examples:
    - The scope of the current function
    - The scope of the module
    - The scope of Python builtins

Namespace Scope

```
import math
result = math.factorial(10)
```

factorial function is in the **namespace math**. To access it, we need to specify the namespace math as prefix.

```
import math as m
volume = m.get_sphere_volume(10)
```

factorial function is in the **namespace m**. To access it, we need to specify the namespace m.

```
from math import factorial
volume = factorial(10)
```

The factorial function is imported from the math module, but **inserted in the current scope**. Consequently It **is directly accessible**: no need to specify namespace.

```
from math import *
volume = factorial(10)
```

All functions of the math module are **imported to the current scope**. All of them are directly accessible.

Namespace Scope

Returns a list object with all available names in the given namespace:

```
dir(namespace)
```

Returns a list object with all available names in the current scope:

```
dir()
```

Attributes from a namespace

- **Common attributes:**
  - **`__name__`**: string with the name of the namespace.
  - **`__doc__`**: string with the documentation associated to the namespace.
- Attributes when the namespace corresponds to a module:
  - **`__path__`**: path to the file that is being imported
  - **`__package__`**
  - …
- Attributes when the namespace corresponds to a Class:
  - **`__class__`**: reference to the class definition of the object
  - …

upf.

Attributes from a namespace

The attribute **`__name__`** of the global namespace is "**`__main__`**".

`my_module.py`

```
def my_function():
   print("I am executing my_function")

my_function()
```

Executing the script:

```
SYSTEM> python my_module.py
I am executing my_function
```

The function is defined and called.

Attributes from a namespace

The attribute **`__name__`** of the global namespace is "**`__main__`**".

`my_module.py`

```
def my_function():
   print("I am executing my_function")

my_function()
```

Importing the script from another python script:

The module is imported. The function is called

```
import my_module

"I am executing my_function"
```

Attributes from a namespace

The attribute **`__name__`** of the global namespace is "**`__main__`**".

my_module.py

```
def my_function():
   print("I am executing my_function")

if __name__=="__main__":
   my_function()
```

Executing the script:

```
SYSTEM> python my_module.py
I am executing my_function
```

The function is defined and called.

Attributes from a namespace

The attribute **`__name__`** of the global namespace is "**`__main__`**".

my_module.py

```
def my_function():
   print("I am executing my_function")

if __name__=="__main__":
   my_function()
```

Importing the script from another python script:

```
import my_module
```

The module is imported. The function is **NOT** called

Python Standard Library

- Several modules with specific functions and class definitions.
  - Examples:
    - Numeric and Mathematical Modules:
      - **math**: Mathematical functions
      - **random**: Generate pseudo-random numbers
      - …
    - File and directory access
      - **os**
      - **tempfile**
      - …
    - Regular expressions: **re**
    - Data compression…

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

Module sys

- **exit**([arg]): exit from Python.
  - Argument is the exit codes:
    - 0 (default): successful termination.
    - Otherwise, abnormal termination

- **stdin, stderr, stdout**: standard streams. Preconnected input and output channels between a computer program and its environment.

Module sys

- **argv**: **List** object of command line arguments passed to a Python script.
  - argv[0]: script name
  - argv[1:]: arguments

- …

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

Module os

- **Operation system** functionality
  - Files:
    - **listdir**: List files in a given directory
    - **remove**: remove a file
    - …
  - Commands:
    - **system**: execute a command in a subshell. The return value is the exit status of the process.
    - **popen**: Open a pipe to or from command
    - …

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

Module os

- **Operation system** functionality
- **Functions on pathnames**
  - exists(path)
  - dirname(path)
  - isfile(path)
  - isdir(path)
  - islink(path)
  - relpath
  - split
  - join

https://docs.python.org/3/library/os.path.html

Exercises. Block 1. Part 6.

Create a python script called <**`ID_S06.py`** with:

Create a python function that calculates the **mean** of the **minimum distance between any two residue pairs** found in the same chain of a PDB. The script, when executed by command line, should output in standard output the mean distance for each chain (with 4 decimal positions). The python script should use a single argument corresponding to the PDB file path to use. This command line argument is optional. If the PDB file path is not defined, read the PDB file from standard input.

The function must return a dictionary with **chains** as keys and mean minimum distances as values. It uses a single argument which specifies the path of the PDB file If the argument pdb_file_path is None, read the PDB file from standard input.

```
calculate_pdb_chain_mean_minimum_distances(pdb_file_path)
```

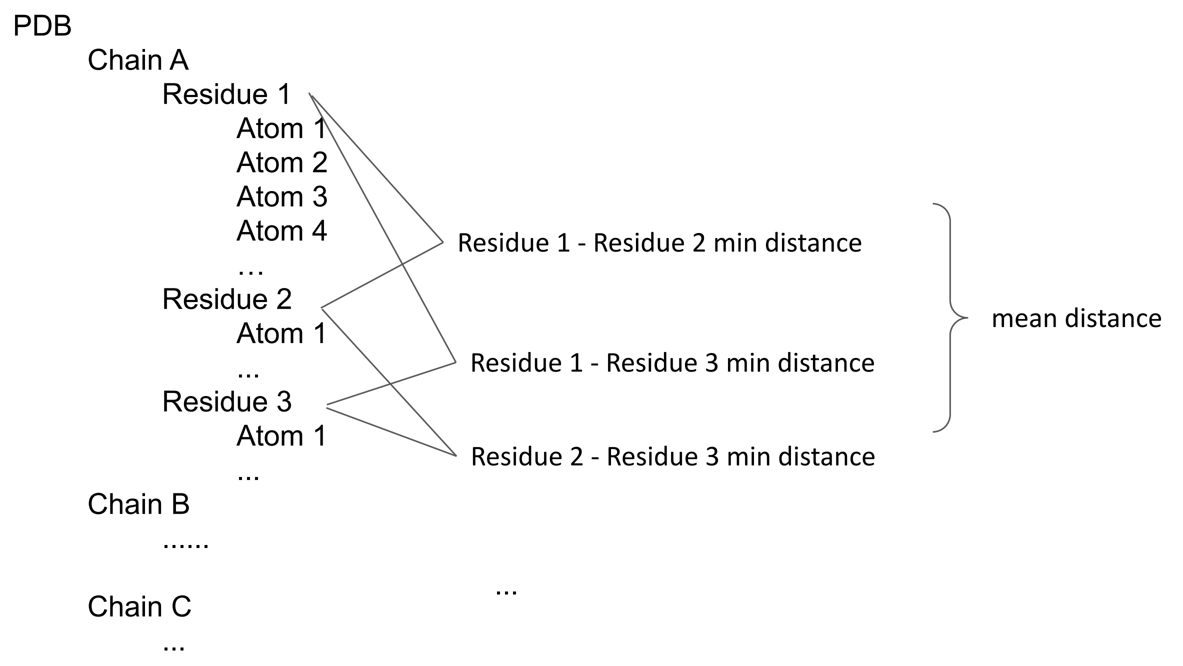
When the file is imported as a module, it should not execute the function. The function should only be called when the script is executed by command line.

Sample output in command line:

```
A: 22.7400
B: 20.4224
N: 23.9393
F: 23.9730
J: 23.4187
```

Exercises. Block 1. Part 6.

Create a python script called <**`ID_S06.py`** with:



Exercises. Block 1. Part 6.

conditions:

- Read the pdb file parsing the pdb file using your code (no third-party libraries)
- PDB File format specification:

https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/tutorials/pdbintro.html

PDB format is positional! (i.e. the information is located in specific positions in each line)