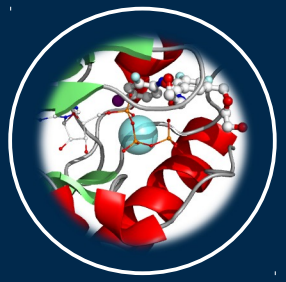
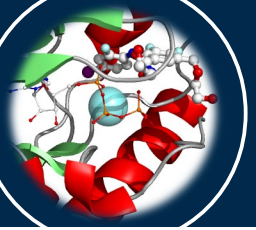


Introduction à Python

Jérôme Pansanel

Toulouse, 18 et 19 janvier 2016



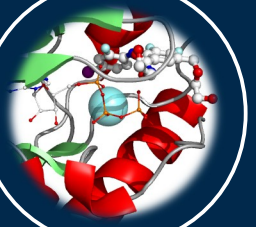
Objectifs

Objectifs de cette introduction :

- Apprendre les bases de Python
- Créer des programmes simples
- Savoir trouver de la documentation
- Connaître les rudiments des bonnes pratiques en programmation
- Utiliser des codes externes pour faire des choses compliquées simplement

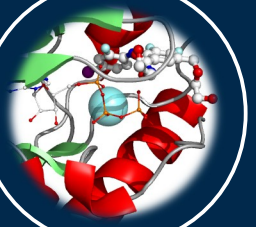
Les points qui ne seront pas traités :

- Écrire du code mal formaté
- Comment écrire un code de 200.000 lignes sans erreur
- Toutes les fonctionnalités de Python



Sommaire

- Python et l'environnement de travail
- La base
- Les types de base
- Les structures de contrôle
- Les fonctions internes et les fichiers
- Les fonctions
- Les modules
- *Des exercices*

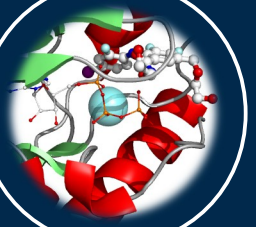


Python

Python en quelques mots :

- Langage de programmation objet créé par Guido van Rossum
- Distribué sous licence libre
- Code interprété
- Multiplate-formes
- Gestion automatique de la mémoire
- Deux versions : 2.7 et 3.5
- Langage extrêmement utilisé dans tous les domaines scientifiques

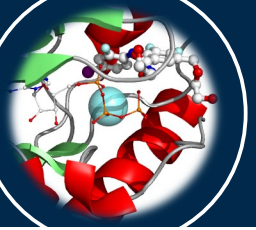
→ http://www.python.org



Documentation

Des sites importants :

- https://docs.python.org/2/tutorial/index.html
- https://docs.python.org/2/reference/index.html
- https://www.pasteur.fr/formation/infobio/python/
- http://biopython.org/wiki/Main_Page



Environnement de travail

Spyder

Fichier Édition Recherche Source Exécution Déboguer Interpréteurs Outils Affichage ?

/home/jerome

Éditeur - /home/jerome/pngpymol.py

function.py | plugin.py | pngpymol.py

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Jan 14 13:42:54 2016
4
5 @author: jerome
6 """
7
8 import __main__
9 __main__.pymol_argv = ['pymol','-q'] # Pymol: quiet and no GUI
10 import pymol
11 pymol.finish_launching()
12 pymol.cmd.fetch('3DV3')
13 pymol.cmd.png('script_py')
14 pymol.cmd.quit()
15
```

Inspecteur d'objets

Source Console Objet os.getcwd

getcwd

Definition : getcwd(...)
Type : Function of posix module

getcwd() -> path

Return a string representing the current working directory.

Inspecteur d'objets | Explorateur de variables | Explorateur de fichiers

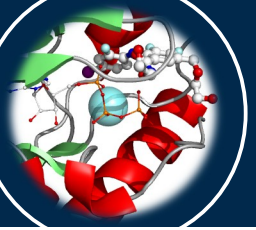
Console

Python 1 00:01:40

```
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.

Imported NumPy 1.8.2, SciPy 0.13.3, Matplotlib 1.3.1

Type "scientific" for more details.
>>>
```

Console | Console IPython | Historique

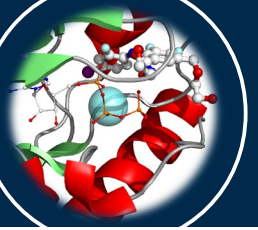
Droits d'accès : RW Fins de ligne : LF Encodage : UTF-8 Ligne : 15 Colonne : 1 Mémoire : 36 %



Points essentiels

Au cours de la formation, de nombreux points vont être abordés. Il n'y en a que quatre à retenir (prioritairement) :

- L'indentation
- La commande `help()`
- La commande `dir()`
- La commande `print()`



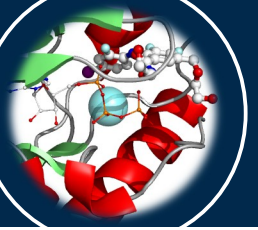
Un code propre

Pour garder un code propre et homogène, des règles communes à de nombreux développeurs ont été adoptées. Elles sont résumées dans le document PEP 8.

Les règles importantes :

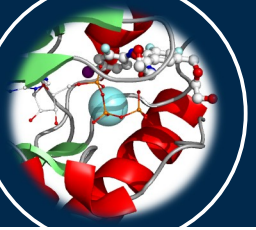
- Utiliser des espaces pour l'indentation (4 par niveau en général)
- Limiter le nombre de caractère par ligne à 79 (72 pour les commentaires)
- Les classes sont séparées par deux lignes vides et les fonctions par une ligne vide. Des lignes vides peuvent également être ajoutées dans des fonctions pour les découper en sous-partie.
- Ecrire les noms de classe sous la forme NomDeClasse.
- Ecrire les noms de fonction et de variable sous la forme nom_de_fonction.
- Utiliser des majuscules pour les noms de variables

→ https://www.python.org/dev/peps/pep-0008/
→ https://google.github.io/styleguide/pyguide.html



Un code propre

```
class SampleClass(object):
    """Summary of class here.

    Longer class information....
    Longer class information....

    Attributes:
        likes_spam: A boolean indicating if we like SPAM or not.
        eggs: An integer count of the eggs we have laid.
    """

    def __init__(self, likes_spam=False):
        """Inits SampleClass with blah."""
        self.likes_spam = likes_spam
        self.eggs = 0

    def public_method(self):
        """Performs operation blah."""
```



La base

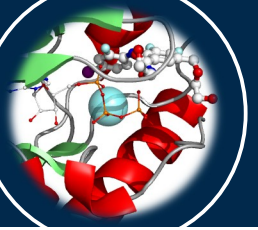
La première fonction :

```
>>> print("Hello World!")
Hello World!
```

Les variables :

- Les noms de variable commence par une lettre ou un '_'
- Les noms de variable peut contenir des lettres, chiffre ou '_'

```
>>> a = 1
>>> _b = 2
>>> var1 = 3
```



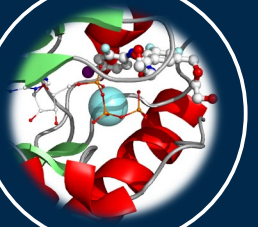
Les types de base

Les valeurs numériques

```
>>> a = 1
>>> b = 3L
>>> c = 2.0
>>> d = 1+2j
>>> c
2.0
```

Les chaînes de caractère

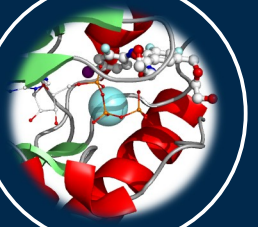
```
>>> a = 'un mot\n'
>>> b = "sans fin"
>>> print(a)
un mot

>>> print(b)
sans fin
>>> bloc = '''Un bloc long
... de plusieurs lignes'''
>>> a + b
```



Les types de base

Les fonctions des chaînes de caractères

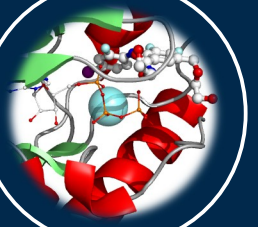
```
>>> sequence = 'MELKDDDFEK'
>>> sequence.find('D')
4
>>> sequence.rfind('D')
6
>>> sequence.count('E')
2
>>> sequence.split('E')
['M', 'LKDDDF', 'K']
>>> 'E'.join(['M', 'LKDDDF', 'K'])
'MELKDDDFEK'
>>> dirty_seq = '  MELKDDDFEK    '
>>> dirty_seq.strip()
'MELKDDDFEK'
```



Les types de base

Le formatage des chaînes de caractère

```
>>> a = "moléculaire"
>>> print("visualisation %s" % a)
visualisation moléculaire
>>> b = "avec PyMOL"
>>> print("visualisation %s %s" % (a,b))
>>> c = 2016
>>> print(a + " " + c)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> print(a + " " + str(c))
moléculaire 2016
>>> print("visualisation %s en %s" % (a,c))
visualisation moléculaire en 2016
>>> print("visualisation %s en %f" % (a,c))
visualisation moléculaire en 2016.000000
>>> a[1]
'o'
```



Les types de base

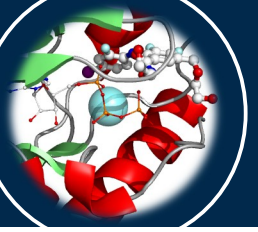
Les listes :

- Tableau modifiable contenant des objets, elles peuvent être parcourues à l'aide d'un index

```
>>> liste1 = []
>>> liste2 = ['1','2']
>>> liste3 = ['3']
>>> liste = liste1 + liste2 + liste3
>>> liste
['1', '2', '3']
>>> liste[1]
'2'
>>> ''.join(liste)
'123'
```

Les fonctions utiles :

```
maliste.append(x)
maliste.sort(fonc)
maliste.insert(i, x)
maliste.count()
maliste.extend(x)
```



Les types de base

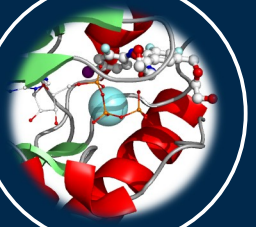
Les dictionnaires :

- Tableau modifiable contenant des objets qui sont référencés à l'aide d'une clé

```
>>> dict = {}
>>> one_letter = {'ALA':'A','CYS':'C','LEU':'L'}
>>> one_letter['ALA']
'A'
```

Les fonctions utiles :

```
one_letter.has_key(k)
one_letter.keys()
one_letter.values()
one_letter.items()
```

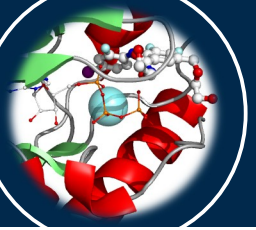


Les types de base

Les tuples :

- Tableau non modifiable d'objets parcouru à l'aide d'un index

```
>>> collection = ()
>>> collection = (0,)
>>> collection = (0, 'a', 3.14)
>>> collection[1]
'a'
>>> collection_bis = (1, 'b')
>>> collection + collection_bis
(0, 'a', 3.14, 1, 'b')
>>> collection = (0, 'a', 3.14, 'a')
>>> collection.count('a')
2
```

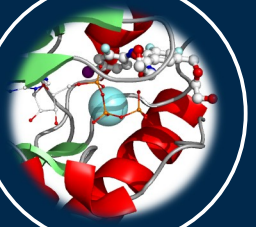


Les plages

Les plages :

- Sélection d'une section continue dans une séquence
- Les limites sont 0, la longueur de la séquence ou un index négatif

```
>>> liste = ['1', '2', '3', '4', '5']
>>> liste[1:3]
['2', '3']
>>> liste[1:]
['2', '3', '4', '5']
>>> liste[:-2]
['1', '2', '3']
>>> liste[:]
['1', '2', '3', '4', '5']
```



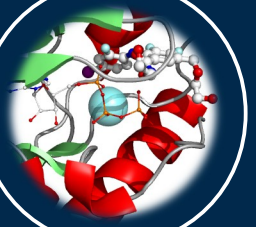
Les autres types

Il existe deux autres types de variable

- La variable vide : `None`
- Le booléen : `True`, `False`

```
>>> a = None
>>> a
>>> b = True
True
```

Ces types sont très utilisés dans les structures de contrôle.

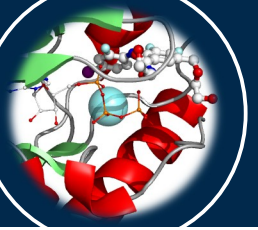


Des commentaires ?

Les commentaires :

- Partie du code utile pour documenter le code

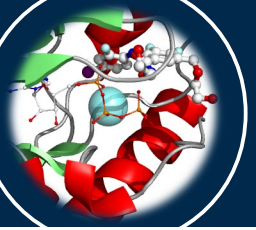
```
# Tableau permettant de passer du code 3 lettres à une lettre
one_letter = {'ALA':'A','CYS':'C','LEU':'L'}

seq = one_letter['ALA']    # Documentation en ligne

'''Une documentation sur plusieurs lignes
Cela permet de détailler le fonctionnement d'une fonction
et de ses variables
'''
```



Immuable ?

Certains types de variable sont immutables (chaîne de caractères, tuple, int)
Les listes et les dictionnaires sont modifiables (possibilité d'ajouter des éléments ou d'en supprimer)

```
>>> a = 1
>>> b = a
>>> b
1
>>> a = 2
>>> b
1
>>> a = [1, 2]
>>> b = a
>>> b
[1, 2]
>>> a.append(3)
>>> b
[1, 2, 3]
```

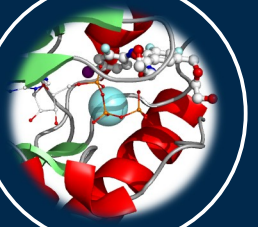


Un peu de maths

Opérations arithmétiques de base :

- *, /, +, -, % et **

```
>>> a = 5
>>> b = 3
>>> a * b
15
>>> a / b
1
>>> a / float(b)
1.6666666666666667
>>> a % b
2
```

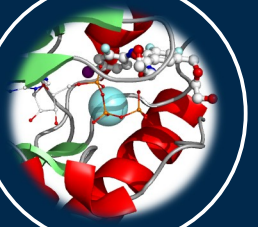


Structure de contrôle

Les tests conditionnels :

- Les instructions `if` et `else` permettent d'effectuer un ensemble d'instructions en fonction de la valeur d'une expression

```
>>> aa = 'ILE'
>>> if aa == 'ILE' :
>>>     print('I')
>>> elif aa == 'LEU' :
>>>     print('L')
>>> else
>>>     print("Acide aminé inconnu")
I
```

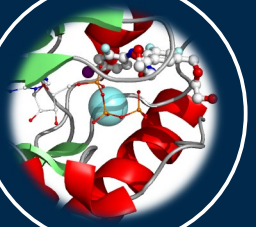


Structure de contrôle

Les opérations de comparaison :

- `A == B` : vrai si `A` est égal à `B`
- `A > B`, `A < B` : vrai si `A` est supérieur à `B` et inversement
- `not X` : vrai si `X` est faux
- `A or B` : renvoie `A` si `A` est vrai, `B` si `A` est faux mais `B` est vrai, faux si `A` et `B` sont faux
- `A and B` : renvoie `A` si `A` est faux, `B` si `A` est vrai
- Les opérateurs `and` et `or` sont court-circuitants

```
>>> a = True
>>> b = False
>>> a or b
True
>>> a and b
False
I
```

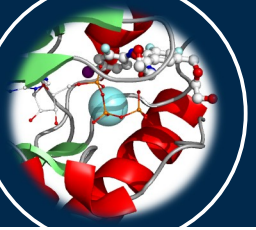


Structure de contrôle

Séquence d'itération :

- L'instruction `for` permet d'itérer dans une séquence

```
>>> sequence = 'MELKDDDFEK'
>>> for aa in sequence:
...     print(aa)
...
M
E
L
K
D
D
D
F
E
K
```



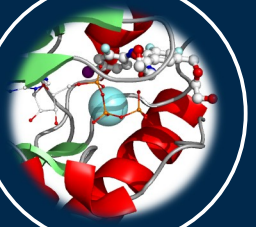
Exercice 1

A l'aide de la variable suivante :

```
amino_acid_weight = { 'D' : 133,'E' : 147, 'F' : 165, 'L' :
131, 'K' : 146, 'M' : 149}
```

Calculez la masse moléculaire de la séquence suivante :

```
sequence = 'MELKDDDFEK'
```



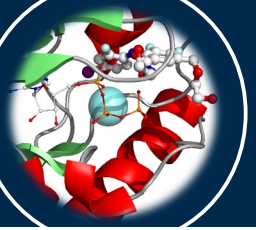
Exercice 2

A l'aide de la variable suivante :

```
hydrophobic_amino_acids = [ 'A', 'I', 'L', 'F', 'V', 'P',
'G' ]
```

Calculez le pourcentage d'acide aminé dans la séquence suivante :

```
sequence = 'MELKDDDFEK'
```

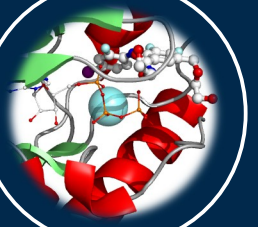


Structure de contrôle

Séquence d'itération :

- L'instruction `for` permet également d'itérer dans un dictionnaire

```
>>> one_letter = {'ALA':'A','CYS':'C','LEU':'L'}
>>> for key in one_letter.keys():
...     print(one_letter[key])
...
C
L
A
>>> for trois,un in one_letter.items():
...     print("%s -> %s" % (trois, un))
...
CYS -> C
LEU -> L
ALA -> A
```

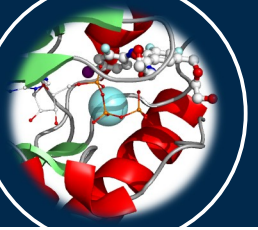


Structure de contrôle

Boucle conditionnelle :

- L'instruction `while` permet d'exécuter un bloc d'instructions tant que la condition testée est vraie

```
>>> a = 0
>>> while a < 5 :
...     a += 1
...     print(a)
...
1
2
3
4
5
>>> # Boucle sans fin
>>> while a > 0 :
...      a+= 1
...      print(a)
...
```

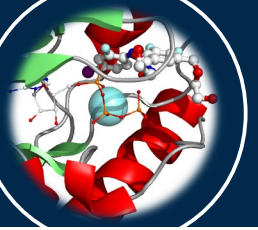


Structure de contrôle

Instructions complémentaires :

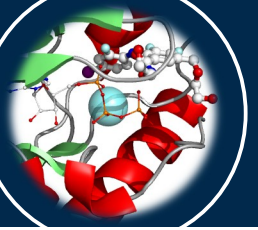
- Les instructions `pass`, `continue` et `break` permettent de contrôler plus finement l'exécution des boucles `while` et `for`
- L'instruction `pass` permet de n'effectuer aucune action
- L'instruction `continue` permet de sauter à la prochaine itération
- L'instruction `break` permet d'arrêter l'itération

```
>>> sequence = 'MELKDDDFEK'
>>> for aa in sequence:
...     if aa == 'D':
...         print(aa)
...         break
...     else:
...         continue
...
D
```



Les fonctions internes

- Python fournit de nombreuses fonctions nativement : `https://docs.python.org/2/library/functions.html`
- `float`, `int`, `long`, `str`, `dict`, `list`, `tuple`
- `min`, `max`
- `pow`
- `range`, `xrange`
- `round`
- `sorted`
- `sum`
- `...`

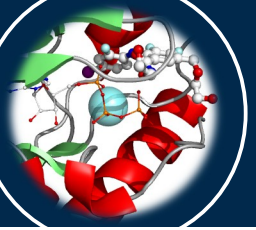


Les fichiers

Lecture d'un fichier :

- Python permet d'accéder aux fichiers simplement à l'aide de la fonction `open`.
- Les options `r` ou `w` permettent d'indiquer si nous souhaitons l'utiliser pour lire et/ou écrire
- Il faut fermer le fichier à la fin de l'écriture pour être sûr qu'il ait bien tout son contenu

```
>>> fichier = open('3DV3.fasta','r')
>>> entete = fichier.readline()
>>> sequence = fichier.readlines()
>>> entete
'>3DV3:A|PDBID|CHAIN|SEQUENCE\n'
>>> sequence
['MELKDDDFEKISELGAGNGGVVFKVSHKPSGLVMARKLIHLEIKPAIRNQIIRELQVL
HECNSPYIVGFYGAFYSDGEIS\n', 'ICMEHMDGGSLDQVLKKAGRIPEQILGKVSIA
VIKGLTYLREKHKIMHRDVKPSNILVNSRGEIKLCDFGVSGQLIDSMA\n','NSFVGTR
SYMSPERLQGTHYSVQSDIWSMGLSLVEMAVGRYPIPPPDAKELELMFGCQVEGDAAETP
PRPRTPGRPLSSY\n','GMDSRPPMAIFELLDYIVNEPPPKLPSGVFSLEFQDFVNKCL
IKNPAERADLKQLMVHAFIKRSDAEEVDFAGWLCSTIG\n', 'LN\n']
```



Les fichiers

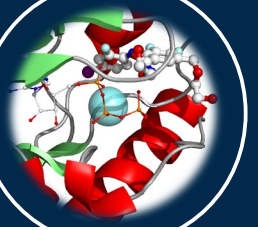
Écriture dans un fichier :

- Python permet d'écrire dans un fichier avec les fonctions `write` et `writelines`
- Le fichier doit être fermé avec la fonction `close`

```
>>> entete = '> ma sequence\n'
>>> sequence = 'MELKDDDFEK\n'
>>> fichier = open('sequence.fasta','w')
>>> fichier.write(entete)
>>> fichier.write(sequence)
>>> fichier.close()
```

Contenu du fichier `sequence.fasta` :

```
> ma sequence
MELKDDDFEK
```

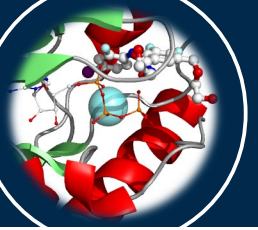


Exercice 3

- Écrire un code qui ouvre le fichier `3dv3.pdb`, lise la séquence et affiche cette séquence à l'aide du code à 1 lettre avec au maximum 72 caractères par ligne :

```
one_letter = {'VAL':'V', 'ILE':'I', 'LEU':'L', 'GLU':'E', 'GLN':'Q', \
              'ASP':'D', 'ASN':'N', 'HIS':'H', 'TRP':'W', 'PHE':'F', \
              'TYR':'Y', 'ARG':'R', 'LYS':'K', 'SER':'S', 'THR':'T', \
              'MET':'M', 'ALA':'A', 'GLY':'G', 'PRO':'P', 'CYS':'C', \
             }
```

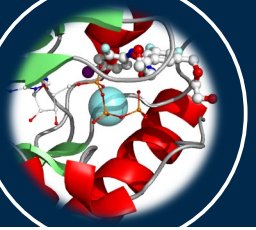
```
MELKDDDFEKISELGAGNGGVVFKVSHKPSGLVMARKLIHLEIKPAIRNQIIRELQVLHECNSPYIVGFYGA
FYSDGEISICMEHMDGGSLDQVLKKAGRIPEQILGKVSIAVIKGLYLREKHKIMHRDKPSNILVNSRGEKLC
DFGVSGQLISMANSFVGTRSYSPERLQGTHYSVSDIWSMGLSLVEAVGRYPIPPPDAELELMFGCQVEGAAE
TPPRPRTPGPLSSYGMDSRPPAIFELLDYIVNEPPKLPSGVFSLEQDFVNKCLIKNPERADLKQLMVHAIKR
SDAEEVDFAWLCSTIGLN
```



Les fonctions

- Une fonction est une portion de code permettant d'effectuer une action précise et pouvant être appelé à différents moments dans le code (réutilisable).
- Pour définir une fonction en Python, le mot-clé `def` est utilisé
- Elle est caractérisée par ses arguments et sa valeur de retour

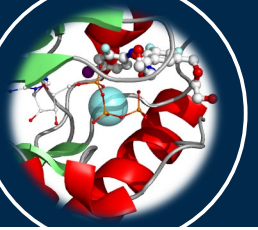
```
>>> def clean_sequence(sequence):
...     '''Cette fonction ne garde que les acides aminés.
...     '''
...     aa_standards = "ACDEFGHIKLMNPQRSTVWY"
...     seq = ""
...     for aa in sequence:
...         if aa in aa_standards:
...             seq += aa
...     return seq
...
>>> clean_sequence('UNE CHAÎNE ALÉATOIRE')
'NECHANEALATIRE'
>>> help(clean_sequence)
```



Exercice 4

- Écrire une fonction qui calcule le nombre d'occurrences de chaque acide aminé dans une séquence (en se basant sur le contenu du fichier `3dv3.pdb`) :

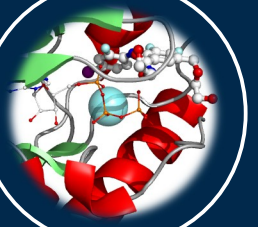
```
>>> amino_acid_count(sequence)
{'A': 15, 'C': 5, 'E': 22, 'D': 15, 'G': 25, 'F': 10, 'I':
22, 'H': 7, 'K': 18, 'M': 9, 'L': 29, 'N': 9, 'Q': 8, 'P':
20, 'S': 25, 'R': 14, 'T': 4, 'W': 1, 'V': 20, 'Y': 8}
```



Exercice 5

- Écrire une fonction `verify_sequence` qui retourne `True` si une chaîne de caractère passé en paramètre est bien une séquence peptidique, et `False` dans les autres cas :

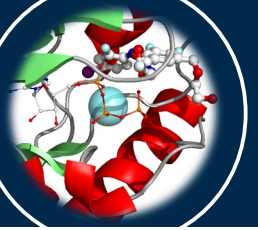
```
>>> verify_sequence('MELKDDDFEK')
True
>>> verify_sequence('Hello World!')
False
```



Les objets

- Les objets en informatique, comme les objets de la vie courante, peuvent être décrit par leurs caractéristiques (couleur, forme) et les méthodes permettant de les utiliser.
- Les objets sont créés avec l'instruction `class`.
- En Python, tous les objets ont un identifiant qui peut être obtenu avec la commande `id`.

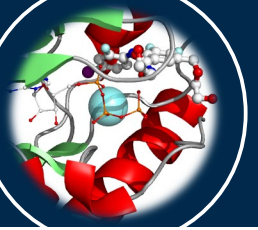
```
>>> class protein:
...     def __init__(self):
...         self.name = ''
...         self.sequence = ''
...         self.molecular_weight = 0
...     def compute_weight(self):
...         pass
...
>>> prot = protein()
>>> prot.name = 'MEK1'
>>> prot.compute_weight()
>>> id(prot)
```



Les modules

- Un module est un ensemble de fonctions regroupées autour d'un thème (par exemple d'accéder à une base de données MySQL, lire un fichier XML, …).
- Il peut être fournit par la distribution Python ou par un logiciel tiers.
- L'instruction `import` permet de charger un module.
- Le module `os` est fournit par Python. Il permet de manipuler les fichiers et les répertoires.

```
>>> import os
>>> help(os)
>>> os.getcwd()
>>> os.path.sep
>>> os.chdir(nouveau_chemin)
>>> os.mkdir(nouveau_repertoire)
>>> os.path.exists(fichier)
```

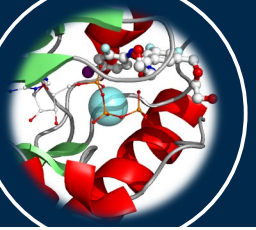


Les modules

- Le module `sys` fournit des fonctions et des variables relatives au système, et en particulier avec l'interpréteur.

```
>>> import sys
>>> sys.platform
'linux2'
>>> sys.argv
['']
>>> sys.version
'2.7.6 (default, Jun 22 2015, 17:58:13) \n[GCC 4.8.2]'
```

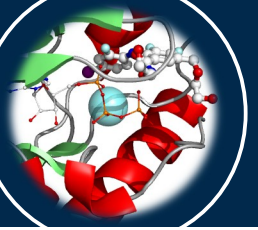
- Le module `math` fournit de nombreuses fonctions et constant pour les calculs.
- Le module `time` fournit des fonctions pour gérer les données de temps.
- Le module `urllib2` permet d'accéder aux données sur le Web.
- Le module `re` implémente des fonctions pour la gestion des expressions régulières.
- Le module `Tkinter` permet de créer des interfaces graphiques simples.
- Le module `random` qui permet de générer des variables aléatoires.



Exercice 6

- Écrire une fonction `generate_sequence`, utilisant le module `random`, qui génère une séquence aléatoire de taille donnée en paramètre de la fonction :

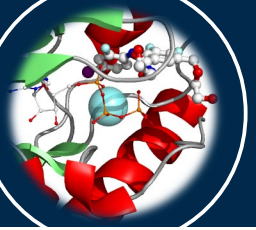
```
>>> generate_sequence(10)
'AIEFDMKLCL'
```



Exercice 7

- Écrire une fonction `mutate_sequence`, utilisant le module `random`, qui effectue une mutation aléatoire sur une séquence passée en paramètre de la fonction :

```
>>> mutate_sequence('MELKDDDFEK')
'MELKDDDIEK'
>>> mutate_sequence('MELKDDDFEK')
'AELKDDDFEK'
```

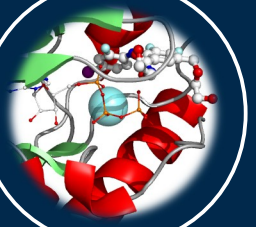


Les modules

- PIP est un outil pour installer des modules externes.

```
localhost:~ user$ pip search nom_du_module
localhost:~ user$ pip install nom_du_module
```

- → https://pip.pypa.io/en/stable/



Exercice 8

- Quels sont les modules pour se connecter aux bases de données MySQL et SQLite ?
- Installer le module pour supporter les bases de données au format SQLite.