PYTHON

Introduction à la programmation pour les biologistes

ENSEIGNANT

- Dr. Djoudi Brahim
- Faculté des science de la nature et de la vie
- Département de Biochimie et Biologie cellulaire et moléculaire

- Email (meilleure façon de me contacter): DjoudiBrahim@hotmail.fr

OBJECTIFS

- Dans ce cours, vous apprendrez tous les aspects fondamentaux de la programmation informatique nécessaires à la conduite de la recherche biologique. À la fin du cours, vous pourrez utiliser ces outils pour importer des données en Python, effectuer une analyse sur ces données et exporter les résultats sous forme de graphiques, de fichiers texte ou de tout autre élément dont vous pourriez avoir besoin. En apprenant comment obtenir l'ordinateur pour faire votre travail pour vous, vous serez en mesure de faire plus de science plus rapidement.

OBJECTIFS

- Écrire des programmes informatiques simples en Python
- Automatiser l'analyse des données
- Appliquer ces outils pour répondre à des questions biologiques
- Apprendre et comprendre les concepts de programmation qui aideront à utiliser d'autres langues

RESPONSABILITÉS DE L'ÉTUDIANT

- Les élèves doivent lire / visionner le matériel assigné avant la classe pour laquelle ils sont programmés, assister aux cours, participer en classe, remplir les devoirs et demander de l'aide tôt s'ils ont des problèmes.

PYTHON

Python est un langage de programmation facile à apprendre. Vous pouvez l'utiliser pour créer des applications web, des jeux, même un moteur de recherche!


```
1. print "Welcome to Python!"
```

Exécuter

```
Welcome to Python!
```

VARIABLES

La création d'applications Web, de jeux et de moteurs de recherche implique de stocker et de travailler avec différents types de données. Ils le font en utilisant **des variables.**

- Une variable stocke une donnée et lui donne un nom spécifique. Par exemple:

```
spam = 5
```

  - La variable spam stocke maintenant le nombre 5.

Exercice :
Définissez la variable **my_variable** égale à la valeur 10. puis imprimer ce variable.

```
1. # Écrivez votre code ci-dessous!
2. my_var = 10
3. print " la valeur de mon variable =", my_var
```

Exécuter

```
la valeur de mon variable =  10
```

VARIABLES

Les variables peuvent être de l'un des types suivants:

- Numérique
- Chaine de caractère (Alphanumérique)
- Booléen

  - Les variables numériques sont susceptibles de recevoir des nombres. Il existe également plusieurs types numériques tels que:

Type Numérique	Plage
Byte (octet)	0 à 255
Entier simple	-32 768 à 32 767
Entier long	-2 147 483 648 à 2 147 483 647
Réel simple	$-3,40x10^{38}$ à $-1,40x10^{45}$ pour les valeurs négatives $1,40x10^{-45}$ à $3,40x10^{38}$ pour les valeurs positives
Réel double	$1,79x10^{308}$ à $-4,94x10^{-324}$ pour les valeurs négatives $4,94x10^{-324}$ à $1,79x10^{308}$ pour les valeurs positives

Exécuter

```
# Définissez les variables aux valeurs listées dans les instructions!
my_int = 7
my_float = 1.23
print " la valeur de mon variable", my_float
```

VOUS AVEZ ÉTÉ AFFECTÉ !

Maintenant, vous savez comment utiliser les variables pour stocker des valeurs. Dites my_int = 7.Vous pouvez changer la valeur d'une variable en la "réaffectant", comme ceci:

```
my_int = 3
```

```
1. my_int = 7
2. my_int = 3
3. print " la valeur de my_int", my_int
```

Exécute

```
la valeur de my_int =  3
```

L'INSTRUCTION D'AFFECTATION

L'instruction d'affectation permet d'attribuer une valeur (non définitive) a une variable déclarée. L'affectation s'effectue en utilisant le symbole (=).

```
spam = 5
ping = 9
```

- On peur affecter a une variable le contenu d'une autre variable

```
spam = ping
```

- On peur incrémenter la valeur d'une même variable sans utiliser une deuxième variable

```
spam = spam + 1
```

```
1. # Écrivez votre code ci-dessous!
2. A = 10
3. B = A + 3
4. A = 3
5. Print  " A = ", A
6. print " B = ", B
```

Exécuter

```
A = 3
B = 13
```

L'INSTRUCTION D'AFFECTATION

L'instruction d'affectation permet d'attribuer une valeur (non définitive) a une variable déclarée. L'affectation s'effectue en utilisant le symbole (=).

```
spam = 5
ping = 9
```

- On peur affecter a une variable le contenu d'une autre variable

```
spam = ping
```

- On peur incrémenter la valeur d'une même variable sans utiliser une deuxième variable

```
spam = spam + 1
```

```
1. # Écrivez votre code ci-dessous!
2. A = 5
3. B = 3
4. C = A + B
5. A = 3
6. B = C – 2
7. print  " A = ", A
8. print " B = ", B
9. print " C = ", C
```

Exécuter

```
A = 3
B = 6
C = 8
```

L'INSTRUCTION D'AFFECTATION

L'instruction d'affectation permet d'attribuer une valeur (non définitive) a une variable déclarée. L'affectation s'effectue en utilisant le symbole (=).

```
spam = 5
ping = 9
```

- On peur affecter a une variable le contenu d'une autre variable

```
spam = ping
```

- On peur incrémenter la valeur d'une même variable sans utiliser une deuxième variable

```
spam = spam + 1
```

```
1. # Écrivez votre code ci-dessous!
2. A = 5
3. B = A + 4
4. A = A + 1
5. B = A - 4
6. C = A – B
7. Print  " A = ", A
8. print " B = ", B
9. print " C = ", C
```

Exécuter

```
A = 6
B = 2
C = 4
```

L'INSTRUCTION D'AFFECTATION

Exercice :

écrire un algorithme permettant d'echanger les valeurs de deux variables A et B, et ce quel que soit leur contenu préalable.

```
1. # Écrivez votre code ci-dessous!
2. A = 5
3. B = 6
4. C = A
5. A = B
6. B = C
7. Print  " A = ", A
8. print " B = ", B
9. print " C = ", C
```

Exécuter

```
A = 6
B = 5
C = 5
```

BOOLÉENS

Les nombres sont un type de données que nous utilisons dans la programmation. Un deuxième type de données s'appelle un **booléen.**

- Un booléen est comme un interrupteur d'éclairage. Il ne peut avoir que deux valeurs. Tout comme un interrupteur ne peut être activé ou désactivé, un booléen ne peut être que Vrai (True) ou Faux (False.).

- Vous pouvez utiliser des variables pour stocker des booléens comme ceci:

```
a = True
b = False
```

Exercice : Définissez les variables suivantes sur les valeurs correspondantes:

- my_int à la valeur 7
- my_float à la valeur 1.23
- my_bool à la valeur True

```
1. # Définissez les variables aux valeurs listées dans les instructions!
2. my_int = 7
3. my_float = 1.23
4. my_bool = True
5. print " la valeur de mon variable", my_bool
```

Exécuter

```
la valeur de mon variable = True
```

COMMENTAIRES

Les commentaires rendent votre programme plus facile à comprendre. Lorsque vous regardez votre code ou que d'autres veulent collaborer avec vous, ils peuvent lire vos commentaires et facilement comprendre ce que fait votre code.

- Le signe # est pour les commentaires. Un commentaire est une ligne de texte que Python n'essaiera pas d'exécuter en tant que code. C'est juste pour les humains à lire.
- Vous pouvez écrire un commentaire multi-ligne, en commençant chaque ligne avec #, cela peut être une douleur.
- Au lieu de cela, pour les commentaires sur plusieurs lignes, vous pouvez inclure le bloc entier dans un ensemble de guillemets (" " ")

```
# Écrivez votre commentaire ici
# Écrivez un autre


" " " Siégeant de ta tasse jusqu'à ce que ça coule,
Saint Graal.
" " "
```

Exécute

LES MATHS

Vous pouvez ajouter, soustraire, multiplier, diviser des nombres comme celui-ci

```
addition = 72 + 23
subtraction = 108 - 204
multiplication = 108 * 0.5
division = 108 / 9
```

Exercice : Définissez la variable **count** égale à la somme de deux grands nombres.

```
1. count = 10000 + 50000
2. print count
```

Exécute

```
60000
```

LES MATHS: EXPONENTIATION

Tout ce calcul peut être fait sur une calculatrice, alors pourquoi utiliser Python?

- ✓ Parce que vous pouvez combiner les mathématiques avec d'autres types de données (par exemple, des booléens) et des commandes pour créer des programmes utiles. Les calculatrices se contentent de chiffres.

Maintenant, travaillons avec les exposants.

```
eight = 2** 3
```

Exercice :
Créez une nouvelle variable appelée **eggs** et utilisez des exposants pour définir **eggs** égal à 100.

```
1. eggs = 10**2
2. print eggs
```

Exécute

```
100
```

LES MATHS: MODULO

Notre opérateur final est modulo. Modulo renvoie le reste d'une division. Donc, si vous tapez 3 % 2, cela retournera 1, car 2 va en 3 une fois, avec 1 restant.

```
un = 3 % 2
```

Exercice :

Utilisez modulo pour définir un variable **spam** égal à 3.

```
1. spam= 7 % 4
2. print spam
```

Exécute

```
3
```

A LA PROCHAINE

```
1. # Écrivez votre code ci-dessous!
2. A = 5  # Variable A
3. B = A + 4 # Affectation ‘=‘
4. A = A + 1
5. my_int = 7 # Variable entier
6. my_float = 1.23 # Variable réel
7. my_bool = True # Variable
   Boolean (or False)
8. count = 10000 + 50000 # addition
9. eggs = 10**2 # exponentiel
10.spam= 7 % 4 # Modulo
11.Print  " A = ", A # Impression
```

SYNTHÈSE DE PYTHON

Jusqu'à présent, vous avez appris sur:

- Variables, qui stockent des valeurs pour une utilisation ultérieure
- Types de données, tels que les nombres et les booléens
- Commentaires, qui facilitent la lecture de votre code
- Opérations arithmétiques, y compris +, -, *, /, ** et %

STRINGS

Un autre type de données utile est la chaîne de caractère. Une chaîne peut contenir des lettres, des chiffres et des symboles.

```
name = "Ryan"
age = "19"
food = "cheese"
```

Les chaînes de caractères doivent être entre guillemets.

Exercice :

- Créez une nouvelle variable et attribuez-lui la chaîne "Hello life!".
- Définissez et imprimer les variables suivantes
  - Affecter César à "Graham"
  - Affecter praline a "John"
  - Affecter Viking a “Ragnar"

```
1.  brian = "hello life!«
2.  # Affectez vos variables ci-dessous, chacune sur sa propre ligne!
3.  César = "Graham"
4.  praline = "John"
5.  Viking = “Ragnar"
6.  # afficher vos variables
7.  print César
8.  print praline
9.  print Viking
```

Exécute

```
Graham
John
Ragnar
```

STRINGS: ACCÈS PAR INDEX

Chaque caractère d'une chaîne est attribué un numéro. Ce nombre est appelé l'indice. Consultez le diagramme dans l'éditeur.

```
c = "cats"[0]
n = "Ryan"[3]
```

Dans l'exemple ci-dessus, nous créons une nouvelle variable appelée c et le mettre à « c », le caractère à l'indice zéro de la chaîne « chats ».

En Python, nous commençons à compter l'indice de zéro au lieu d'un.

Exercice :
Quelle est le résultats de « print fifth_letter »

```
" " "
La chaîne "PYTHON" a six caractères,
numérotés de 0 à 5, comme indiqué ci-dessous:

+---+---+---+---+---+---+
| P | Y | T | H | O | N |
+---+---+---+---+---+---+
 0   1   2   3   4   5

Donc, si vous vouliez "Y", vous pouviez
simplement taper "PYTHON" [1]
(toujours compter à partir de 0!)
" " "

fifth_letter =  "MONTY"[4]

print fifth_letter
```

Exécute

```
Y
```

FONCTIONS INTÉGRÉES

Une fonction (ou function) est une suite d'instructions regrouper et définie par un nom et que on peut l'appeler avec ce nom.

Les fonctions intégrées au langage sont les fonction prédéfini dans un langage de programmation.

Les fonctions intégrées au langage sont relativement peu nombreuses : ce sont seulement celles qui sont susceptibles d'être utilisées très fréquemment.

Voyons comment nous pouvons changer les chaine de caractères en utilisant ces méthodes.

MÉTHODES DE STRINGS

- Les méthodes de String permettent d'effectuer des tâches spécifiques sur les chaînes.
  - Nous allons nous concentrer sur 4 méthodes:
    - len()
    - lower()
    - upper()
    - str()

Commençons par la plus simple **len (…)**, qui obtient la longueur (le nombre de caractères) d'une chaîne!

Exercice :
créer une variable nommée parrot et la définir sur la chaîne "Norwegian Blue et afficher le nombre de caractères dans parrot.

```
1. # La sortie sera le nombre de lettres en "Norwegian Blue"!
2. parrot = "Norwegian Blue"
3. print len (parrot)
```

Exécute

```
14
```

MÉTHODES DE STRINGS: LOWER()&UPPER()

Vous pouvez utiliser la méthode **lower ()** pour vous débarrasser de toutes les majuscules dans vos chaînes. Vous appelez **lower ()** comme ça:

```
"Ryan".lower()
```

Ce qui retournera « ryan ». Maintenant, votre chaîne est 100% minuscule!

Upper() est une méthode similaire pour faire une chaîne complètement en majuscule.

```
"Ryan".upper()
```

Ce qui retournera « RYAN ».

```
parrot = "Norwegian Blue"
print  parrot.lower()
print  "Norwegian Blue".lower()
print  parrot.upper()
print  "Norwegian Blue".upper()
```

Exécute

```
norwegian blue
norwegian blue
NORWEGIAN BLUE
NORWEGIAN BLUE
```

MÉTHODES DE STRINGS: STR()

Maintenant, regardons str (), ce qui est un peu moins simple. La méthode str () transforme les non-chaînes en chaînes! Par exemple:

```
str(2)
```

Ce qui fait de 2 un " 2 " .

C'est quoi la différence.???????

```
1. pi = str(3.14) + 1
2. print pi
```

Exécute

```
Traceback (most recent call last):
File "python", line 4, in <module>
TypeError: cannot concatenate 'str'
and 'int' objects
```

Imprimer des String

- La zone où nous avons écrit notre code s'appelle l'éditeur.
- La console (la fenêtre à droite de l'éditeur) est l'endroit où les résultats de votre code sont affichés.

  - **print** affiche simplement votre code dans la console.

```
1. # instruction print
2. count = 1000
3. print  " Imprimer dans le console "
4. print count
5. print  " Imprimer un variable " , count
```

Exécuter

```
Imprimer dans le console
1000
Imprimer un variable 1000
```

Concaténation des String

- Vous connaissez les chaînes de caractères et vous connaissez les opérateurs arithmétiques. Maintenant, combinons les deux!

```
print "Life " + "of " + "Brian"
```

- Cela va imprimer la phrase Life of Brian

```
1. # instruction print
2. count = 1000
3. print  " Imprimer dans le console "
4. print count
5. print  " Imprimer un variable " , count
```

Exécuter

```
Imprimer dans le console
1000
Imprimer un variable 1000
```

Concaténation des String

- Vous connaissez les chaînes de caractères et vous connaissez les opérateurs arithmétiques. Maintenant, combinons les deux!

```
print "Life " + "of " + "Brian"
```

- Cela va imprimer la phrase Life of Brian

L'opérateur « + » entre les chaînes les "ajoute" l'une après l'autre. Notez qu'il y a des espaces à l'intérieur des guillemets après la vie et de sorte que nous pouvons faire ressembler la chaîne combinée à 3 mots.

```
1. # Imprimez la concaténation de
   "Spam and eggs" sur la ligne 3!


2. print "Spam" + "  and " + "eggs"
```

Exécuter

```
Spam and eggs
```

LES MATHS

Python peut faire tout ce qui est sur une calculatrice, est plus?

- Avant d'essayer d'aller plus loin, voyons ce que Python sait déjà des racines carrées. Demandant à Python de faire la racine carrée de 25.

```
print sqrt(25)
```

Maintenant, travaillons avec les exposants.

```
Traceback (most recent call last): File
"python", line 2, in <module> NameError:
name 'sqrt' is not defined
```

Avez-vous vu ça? Python a déclaré:
NameError: le nom 'sqrt' n'est pas défini.
Python ne sait pas **encore** ce que sont les racines carrées.

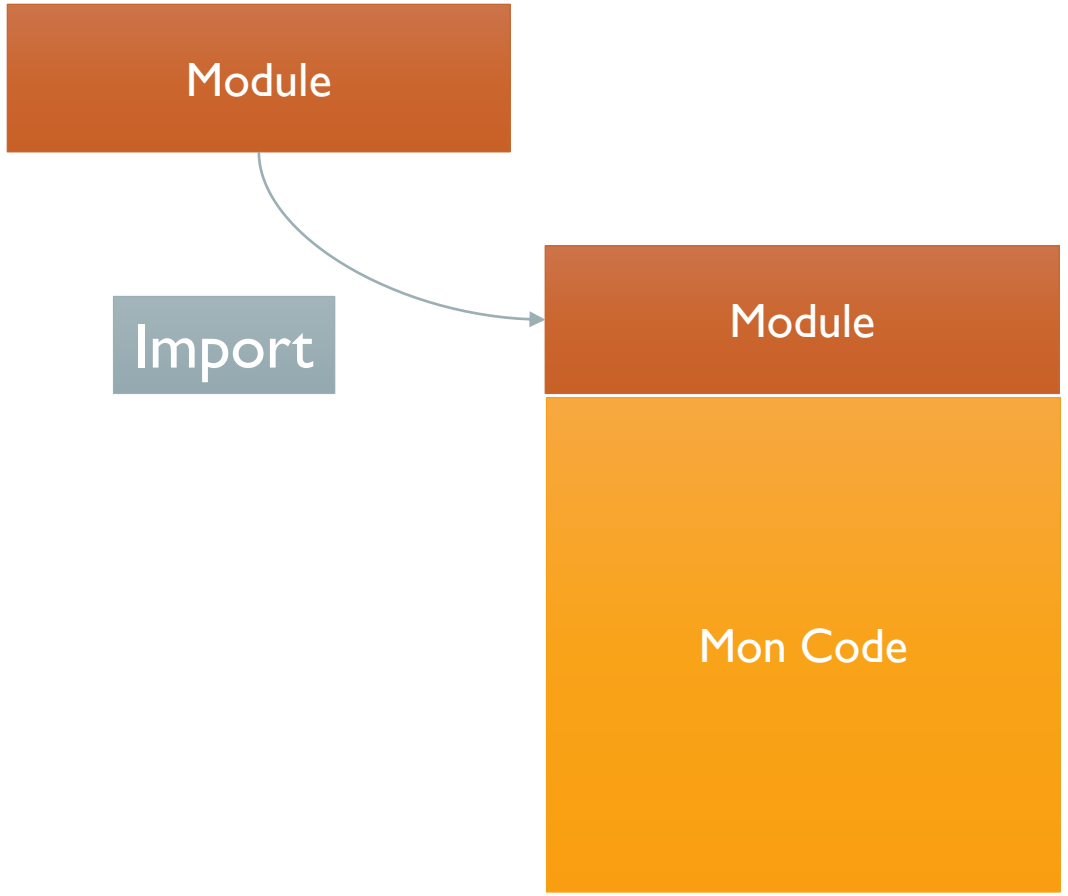
IMPORTATIONS

len("AAACGTR")

Les **fonctions intégrées** au langage sont relativement peu nombreuses : ce sont seulement celles qui sont susceptibles d'être utilisées très fréquemment. Les autres sont regroupées dans des fichiers séparés que l'on appelle des **modules**.

Les modules sont donc des fichiers qui regroupent un ensembles de fonctions. Il existe un grand nombre de modules pré-programmés qui sont fournis d'office avec Python.

Pour utiliser des fonctions de modules dans un programme, il faut au début du fichier **importer** ceux-ci.

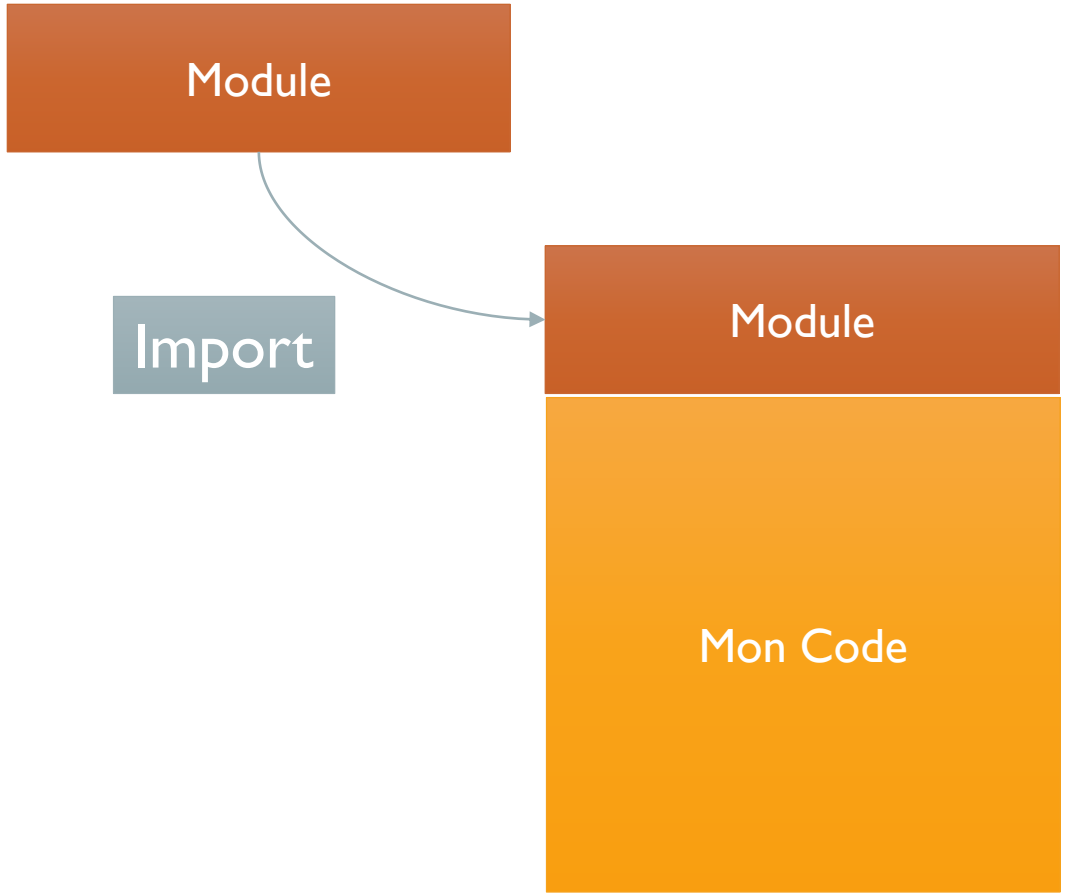


IMPORTATIONS: IMPORTATION GÉNÉRIQUE

Il existe un module Python nommé **math** qui inclut un certain nombre de variables et de fonctions utiles, et **sqrt ()** est l'une de ces fonctions. Pour accéder aux mathématiques, tout ce dont vous avez besoin est le mot-clé **import**. Lorsque vous importez simplement un module de cette façon, cela s'appelle une **importation générique**.

```
import math
print math.sqrt(25)
```

Pour utiliser des fonctions de modules dans un programme, il faut au début du fichier **importer** ceux-ci.



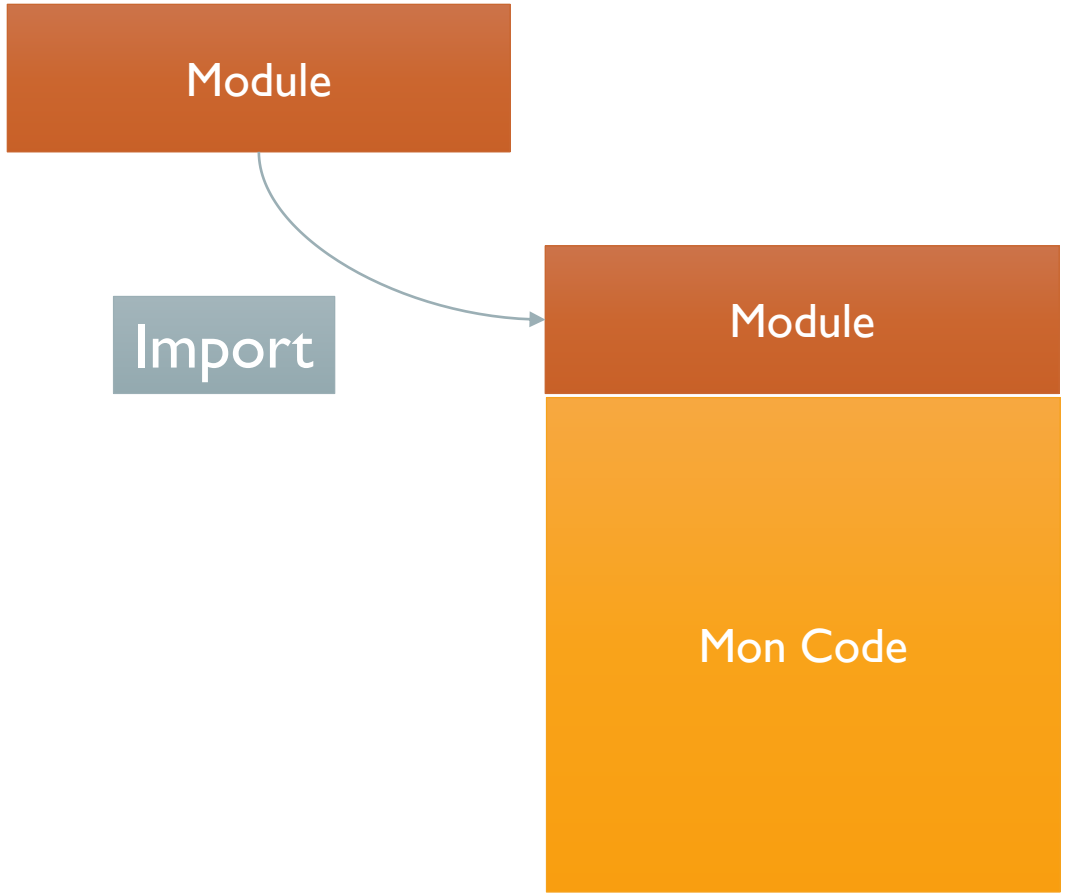
IMPORTATIONS DE FONCTIONS

Il est possible d'importer uniquement certaines variables ou fonctions d'un module donné. Extraire une seule fonction d'un module s'appelle une importation de fonction, et c'est fait avec le mot-clé **from**:

```
from math import sqrt
print sqrt (25)
```

Maintenant, vous pouvez simplement taper sqrt () pour obtenir la racine carrée d'un nombre - plus math.sqrt ()!

Pour utiliser des fonctions de modules dans un programme, il faut au début du fichier **importer** ceux-ci.

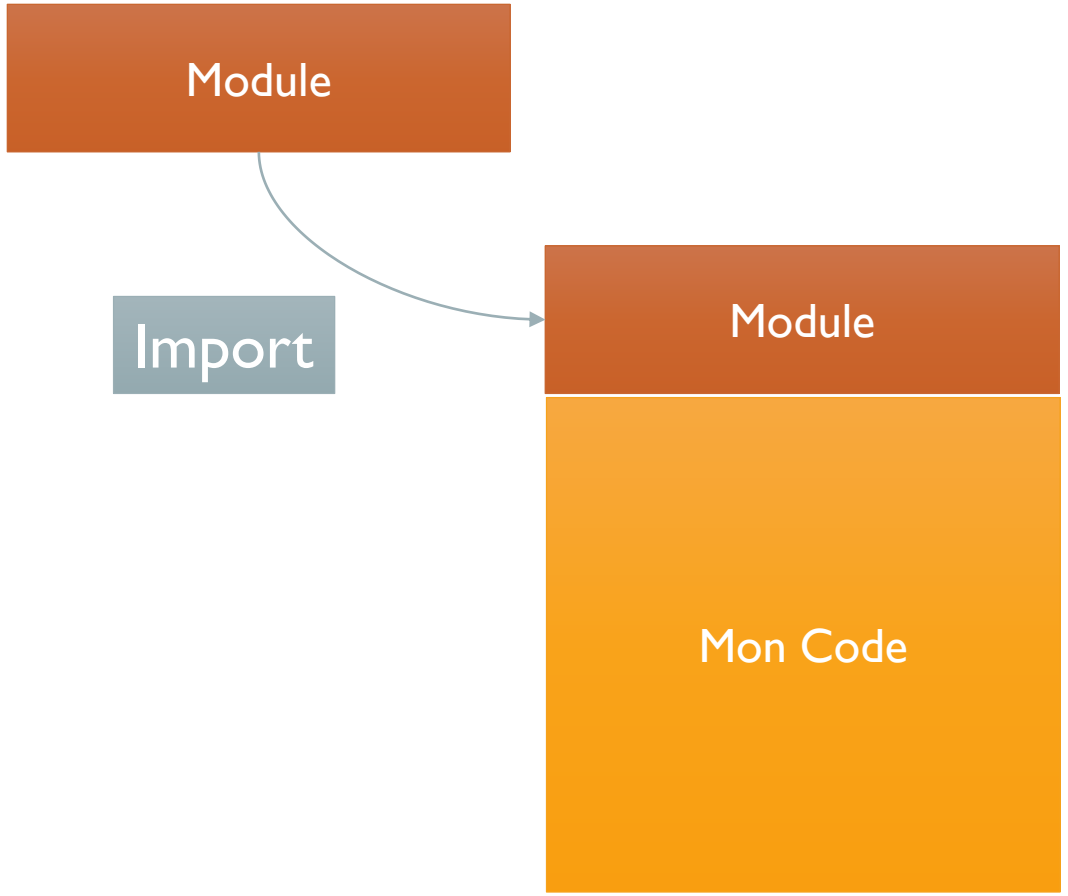


IMPORTATIONS UNIVERSELLES

Que faire si nous voulons toujours toutes les variables et les fonctions dans un module, mais ne veulent pas avoir à taper constamment des maths.?

```
from math import *
print sqrt (25)
```

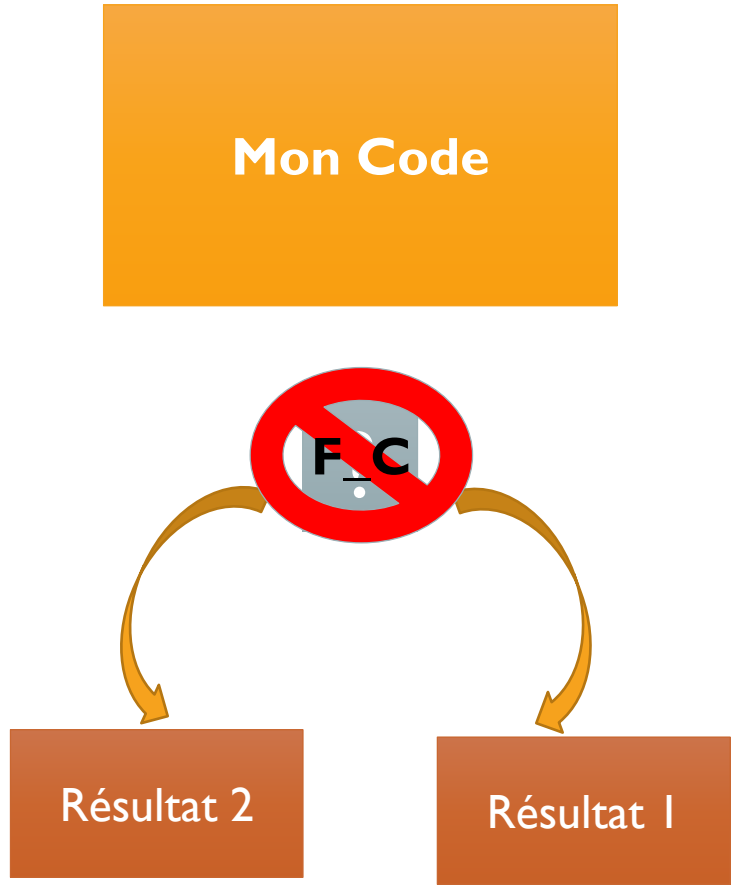
Pour utiliser des fonctions de modules dans un programme, il faut au début du fichier **importer** ceux-ci.



ALLER AVEC LE FLUX

Les programmes Python que nous avons écrits jusqu'ici ont eu un esprit unique: ils peuvent ajouter deux nombres ou imprimer quelque chose, mais ils n'ont pas la possibilité de choisir l'un de ces résultats par rapport à l'autre.

Le flux de contrôle nous donne cette possibilité de choisir parmi les résultats en fonction de ce qui se passe d'autre dans le programme.



Comparer !

- Commençons par l'aspect le plus simple du flux de contrôle: **les comparateurs**. Il y a six:

Égal à (==)

Different de (! =)

Inférieur à (<)

Inférieur ou égal à (<=)

Supérieur à (>)

Supérieur ou égal à (> =)

Notez que == compare si deux choses sont égales, et = assigne une valeur à une variable.

```
1.  3 == 3
True
2.  3 == 6
False
```

```
1. 3 != 6
True
2.  3 != 3
False
```

```
1. 3 < 6
True
2.  6 < 3
False
```

Comparer !

- Commençons par l'aspect le plus simple du flux de contrôle: **les comparateurs**. Il y a six:

Égal à (==)

Different de (! =)

Inférieur à (<)

Inférieur ou égal à (<=)

Supérieur à (>)

Supérieur ou égal à (> =)

Notez que == compare si deux choses sont égales, et = assigne une valeur à une variable.

```
1. 3 <= 3
True
2.  6 <= 3
False
```

```
1. 6 > 3
True
2.  3 > 6
False
```

```
1. 6 >= 6
True
2.  3 >= 6
False
```

Comparer !

- Exercice
- Définissez chaque variable sur Vrai ou Faux en fonction de ce que vous pensez que le résultat sera. Par exemple, 1 <2 sera True, car 1 est inférieur à 2.

```
1. # Réglez ce paramètre sur Vrai si 17 <328 ou False si elle est pas.
2. ResBool =
```

```
1. # Réglez ce paramètre sur Vrai si 100 == (2 * 50) ou False autrement.
2. ResBool =
```

```
1. Réglez ce paramètre sur Vrai si -22> = -18 ou False si elle est pas.
2. ResBool =
```

```
1. # C'est quoi le resultat de print ResBool
2. ResBool = (-22> = -18)
3. print ResBool
```

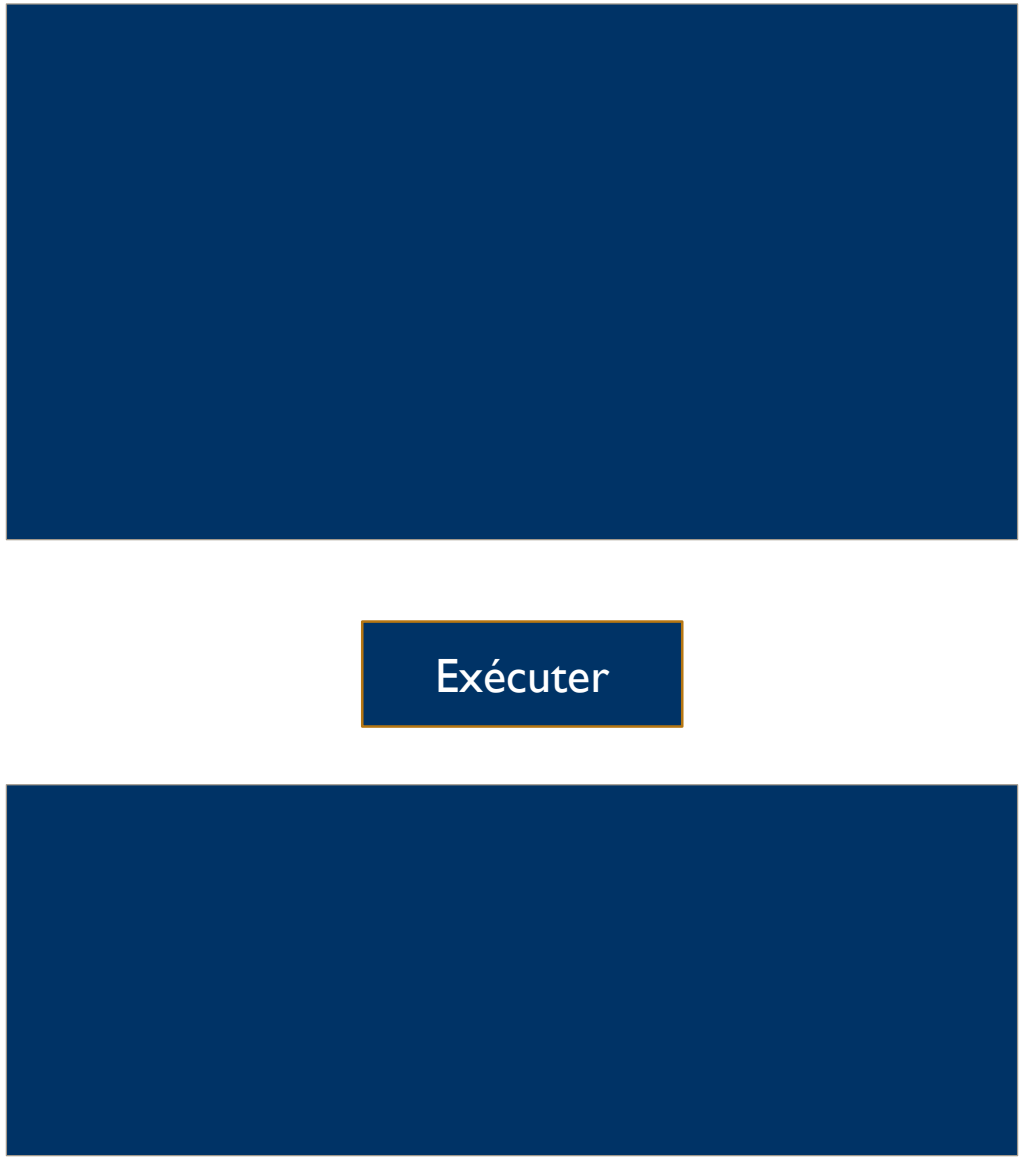
Utiliser les Boolean

- **Les opérateurs booléens** comparent les instructions et produisent des valeurs booléennes. Il y a trois opérateurs booléens:

and qui vérifie si les deux instructions sont vraies;

or qui vérifie si au moins l'une des déclarations est True;

not ce qui donne le contraire de la déclaration.



Les Opérateurs Booléens

and qui vérifie si les deux instructions sont vraies;

- L'opérateur booléen **and** renvoie **True** lorsque les expressions des deux côtés de **and** sont vraies.

`2 < 3 and 3 < 4` **True**

`2 > 3 and 3 < 4` **False**

and

X	Y	X and Y
True	True	True
False	True	False
True	False	False
False	False	False

Les Opérateurs Booléens

and qui vérifie si les deux instructions sont vraies;

- L'opérateur booléen **and** renvoie **True** lorsque les expressions des deux côtés de **and** sont vraies.

2 < 3 and 3 < 4	**True**
2 > 3 and 3 < 4	**False**

```
1. A = False and False
2. B = -(-(-(-2))) == -2 and 4 >= 16**0.5
3. C = 19 % 4 != 300 / 10 / 10 and False
4. D=-(1**2) < 2**0 and 10 % 10 <= 20 - 10*2
5. print A, B, C, D
```

Exécuter

Les Opérateurs Booléens

or qui vérifie si au moins l'une des déclarations est True;

- L'opérateur booléen **or** retourne **True** lorsque au moins une expression de chaque côté de **ou** est vraie. Par exemple:

2 < 3 or 3 < 4	True
2 > 3 or 3 < 4	True
2 > 3 or 3 > 4	False

or

X	Y	X or Y
True	True	True
False	True	True
True	False	True
False	False	False

Les Opérateurs Booléens

or qui vérifie si au moins l'une des déclarations est True;

- L'opérateur booléen **or** retourne **True** lorsque au moins une expression de chaque côté de **ou** est vraie. Par exemple:

2 < 3 or 3 < 4	True
2 > 3 or 3 < 4	True
2 > 3 or 3 > 4	False

or

X	Y	X or Y
True	True	True
False	True	True
True	False	True
False	False	False

Les Opérateurs Booléens

or qui vérifie si au moins l'une des déclarations est True;

- L'opérateur booléen **or** retourne **True** lorsque au moins une expression de chaque côté de **ou** est vraie. Par exemple:

2 < 3 or 3 < 4	True
2 > 3 or 3 < 4	True
2 > 3 or 3 > 4	False

```
1. A = 2**3 == 108 % 100 or 'Rymond' == 'Redington'
2. B = True or False
3. C = 100**0.5 >= 50 or False
4. D = 1**100 == 100**1 or 3 * 2 * 1 != 3 + 2 + 1
5. print A, B, C, D
```

Exécuter

Les Opérateurs Booléens

not ce qui donne le contraire de la déclaration.

- L'opérateur booléen **not** renvoie **True** pour les fausses instructions et **False** pour les vraies instructions. Par exemple:

not False	True
not 2 > 3	True
not 3 < 4	False

not

X	Not X
True	False
False	True

Les Opérateurs Booléens

not ce qui donne le contraire de la déclaration.

- L'opérateur booléen **not** renvoie **True** pour les fausses instructions et **False** pour les vraies instructions. Par exemple:

not False	True
not 2 > 3	True
not 3 < 4	False

```
1. A = not True
2. B = not 3**4 < 4**3
3. C = not 10 % 3 <= 10 % 2
4. D = not 3**2 + 4**2 != 5**2
5. E = not not False
6. print A,B,C,D,E
```

Exécuter

Ceci and cela (or ceci, not ça!)

- Les opérateurs booléens ne sont pas simplement évalués de gauche à droite. Tout comme avec les opérateurs arithmétiques, il existe un ordre des opérations pour les opérateurs booléens:

not évalué en premier;

and est évalué ensuite;

Or est évalué en dernier.

Les parenthèses () s'assurent que vos expressions sont évaluées dans l'ordre que vous voulez. Tout ce qui est entre parenthèses est évalué comme sa propre unité.

```
1. bool = True or not False and False
2. bool_one = False or not True and True
3. bool_two = False and not True or True
4. bool_three = True and not (False or False)
5. bool_four = not not True or False and not True
6. bool_five = False or not (True and True)
```

Exécuter

```
1. True
```

Utiliser les Boolean

```
1. bool= (2 <= 2) and "Alpha" == "Bravo“
2. print bool
```

Exécuter

```
False
```

Syntaxe de l'instruction conditionnelle

The big IF

- **if** est une instruction conditionnelle qui exécute du code spécifié après avoir vérifié si son expression est True.
- Voici un exemple de syntaxe d'instruction if:

```
if 8 < 9:
 print "Eight is less than nine!"
```

Dans cet exemple, 8 <9 est l'expression conditionnelle a contrôlée et l'impression "Huit est inférieur à neuf!" est le code spécifié.

```
1. reponse = "Y"
2. answer = "Left"
3. if answer == "Left“ :
4.     print "This is the Verbal Abuse
   Room, you heap of parrot droppings!"
```

Exécuter

```
1. This is the Verbal Abuse Room,
   you heap of parrot droppings!
```

Syntaxe de l'instruction conditionnelle

The big IF

Exercice 1:

Ecrire un algorithme qui demande un nombre a l'utilisateur, et l'informe ensuite si ce nombre est positif ou negatif (le cas ou le nombre est nul n'est pas traite).

```
Enterer = input ()
```

```
Enterer = input (“message du commande ”)
```

```
1. reponse = input (“Entrer un numero!")
2. if reponse > 0 :
3.  print “ton numero est positif"
```

Exécuter

```
1. Entrer un numero!
2. --
```

Syntaxe de l'instruction conditionnelle

The big IF

- **if** l'expression conditionnelle :
  - Code spécifié.

- l'expression conditionnelle =
  - (2 <= 2) and "Alpha" == "Bravo "
  - Var1 > Var2
  - Bool1 == True
  - Trouvé != False

```
1. reponse = "Y"
2. answer = "Left"
3. if answer == "Left "  and reponse != " y
   " :
4.     print "This is the Verbal Abuse
   Room, you heap of parrot droppings!"
```

Exécuter

```
1. This is the Verbal Abuse Room,
   you heap of parrot droppings!
```

Syntaxe de l'instruction conditionnelle

The big IF

Exerice 2 :

Ecrire un programme demandant à l'utilisateur de donner sa moyenne au bac et affichant s'il est admis,

```
note =  input ("Entrez votre moyenne obtenu au bac ")
if note >= 10 :
   print ("Vous êtes admis")
```

Exécuter

```
1. Entrer qulques choses!
```

Syntaxe de l'instruction conditionnelle

The big IF / ELSE

L'instruction **else** complète l'instruction **if**. Une paire if / else dit:

Si cette expression est vraie,

exécutez ce bloc de code en retrait,

sinon,

exécutez ce code après l'instruction else."

Contrairement à si, sinon ne dépend pas d'une expression.

```
if 8 > 9 :
    print "I don't printed!"
else:
    print "I get printed! "
```

Exécuter

```
1. I get printed!
```

Syntaxe de l'instruction conditionnelle

Exerice 2 :
Ecrire un programme demandant à l'utilisateur de donner sa moyenne au bac et affichant s'il est admis, ainsi que sa mention.

[12-14] mention AB
[14-16] mention B
[16 - >16] mention A

```
note =  input ("Entrez votre moyenne obtenu au bac ")
if note >= 12 and note <14 :
•       print ("Mention AB")
• if note >= 14 and note<16 :
•       print ("Mention B")
• if note >=16 :
•       print ("Mention TB")
```

Exécuter

Syntaxe de l'instruction conditionnelle

Exerice 2 :
Ecrire un programme demandant à l'utilisateur de donner sa moyenne au bac et affichant s'il est admis, ainsi que sa mention.

[12-14] mention AB
[14-16] mention B
[16 - >16] mention A

```
note =  input ("Entrez votre moyenne obtenu au bac ")
if note >= 10 :
print ("Vous êtes admis")
    if note >= 12 and note <14 :
        print ("Mention AB")
    if note >= 14 and note<16 :
        print ("Mention B")
    if note >=16 :
        print ("Mention TB")
else
print ("Vous n'êtes pas admis")
```

Syntaxe de l'instruction conditionnelle

The big elif

elif est l'abréviation de "else if". Cela signifie exactement ce que cela ressemble: "sinon, si l'expression suivante est vraie, faites ceci!"

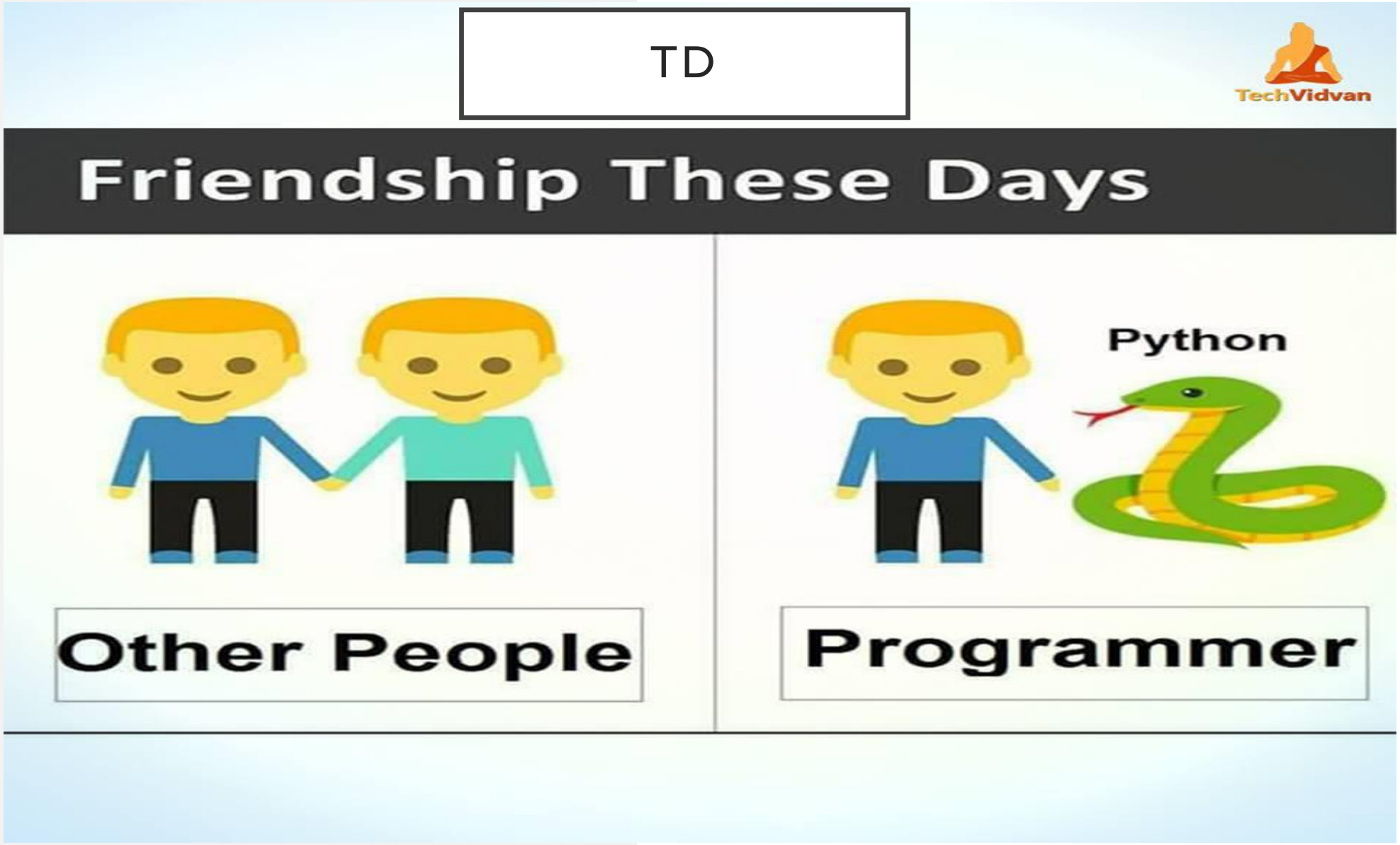
```
if 8 > 9:
    print "I don't get printed!"
elif 8 < 9:
    print "I get printed!"
else:
    print "I also don't get printed!"
```

Dans l'exemple ci-dessus, l'instruction elif n'est vérifiée que si l'instruction if d'origine est False.

```
note =  input ("Entrez votre moyenne obtenu au bac ")
if note > 10 :
    print ("Vous êtes admis")
Elif note < 10 :
    print ("Vous êtes Fichu ")
Elif note = 10 :
    print ("Vous êtes sauver")
Else :
    print (" y a un problème la ")
```

Exécuter

TD


TechVidvan
Friendship These Days
Python
Other People
Programmer

TD

Exercice

– L'indice de masse corporelle d'une personne peut être calculé avec la formule suivante :

$$imc = poids / (taille^2)$$

– Écrire un programme qui demande à l'utilisateur les paramètres nécessaires, calcule l'indice de masse corporelle et affiche le résultat

– Quels sont les paramètres ? Quels sont leur type ?

– Comment calculer $taille^2$?

- poids = input("votre poids (kg) ? ")
- taille = ("votre taille (m) ? ")
- imc = poids / (taille * taille)
- print "indice de masse corporelle :", imc

TD

Exercice :
– Demander l'âge de la personne
– Si l'âge est supérieur à 60, afficher «c'est une personne âgée»
– Si l'âge est inférieur à 15, afficher «c'est un enfant»
– Sinon, afficher «c'est un adulte»

```
age = input("âge de la personne ?")
if age > 60:
    print "c'est une personne âgée"
elif age < 15:
    print "c'est un enfant"
else:
    print "c'est un adulte"
```

TD

- Exercice :

– Demander l'âge de la personne et afficher «la personne a X an(s)», où X est la valeur entrée, et en ne mettant un «s» à «ans» que si nécessaire

```
age = input("âge de la personne ?")
if age > 1:
    print "la personne a", age, "ans"
else:
    print "la personne a", age, "an"
```

TD

Exercice :

Écrire un programme qui demande à l'utilisateur si c'est un fumeur ou non, puis Demander l'âge de la personne
– Si l'âge est supérieur à 60, afficher «le patient est une personne âgée qui fume»
– Si l'âge est inférieur à 15, afficher «c'est un enfant fumeur»
– Sinon, afficher «c'est un adulte fumeur »
– Sinon « le patient ne fume pas »

- if (fumeur == "oui") and (age > 60):
  - print "le patient est une personne âgée qui fume !"
- if (fumeur == "oui") or (age > 60):
  - print "le patient est une personne âgée ou un fumeur!"
- if not(fumeur == "oui"):
  - print "le patient est non fumeur!"

TD

Exercice :

Écrire un programme qui demande à l'utilisateur d'entrer une séquence d'ADN, et qui affiche «C'est un gène» si celleci correspond à un gène, et «Ce n'est pas un gène» dans le cas contraire.
– On considérera qu'une séquence est un gène si celleci commence par un codon méthionine (ATG) et se termine par un codon STOP (TAA, TAG, TGA)

- Pour vérifier le début et la fin d'une chaine sans avoir à déterminer sur quelle longueur , on peut utiliser "startswith()" et "endswith()", mais c'est moins rapide car faisant appel à une fonction de plus haut niveau. (Référence nécessaire)
- if chain.startswith('1234'):
- print('ok')
- if chain.endswith('89'):
- print('ok')

TD

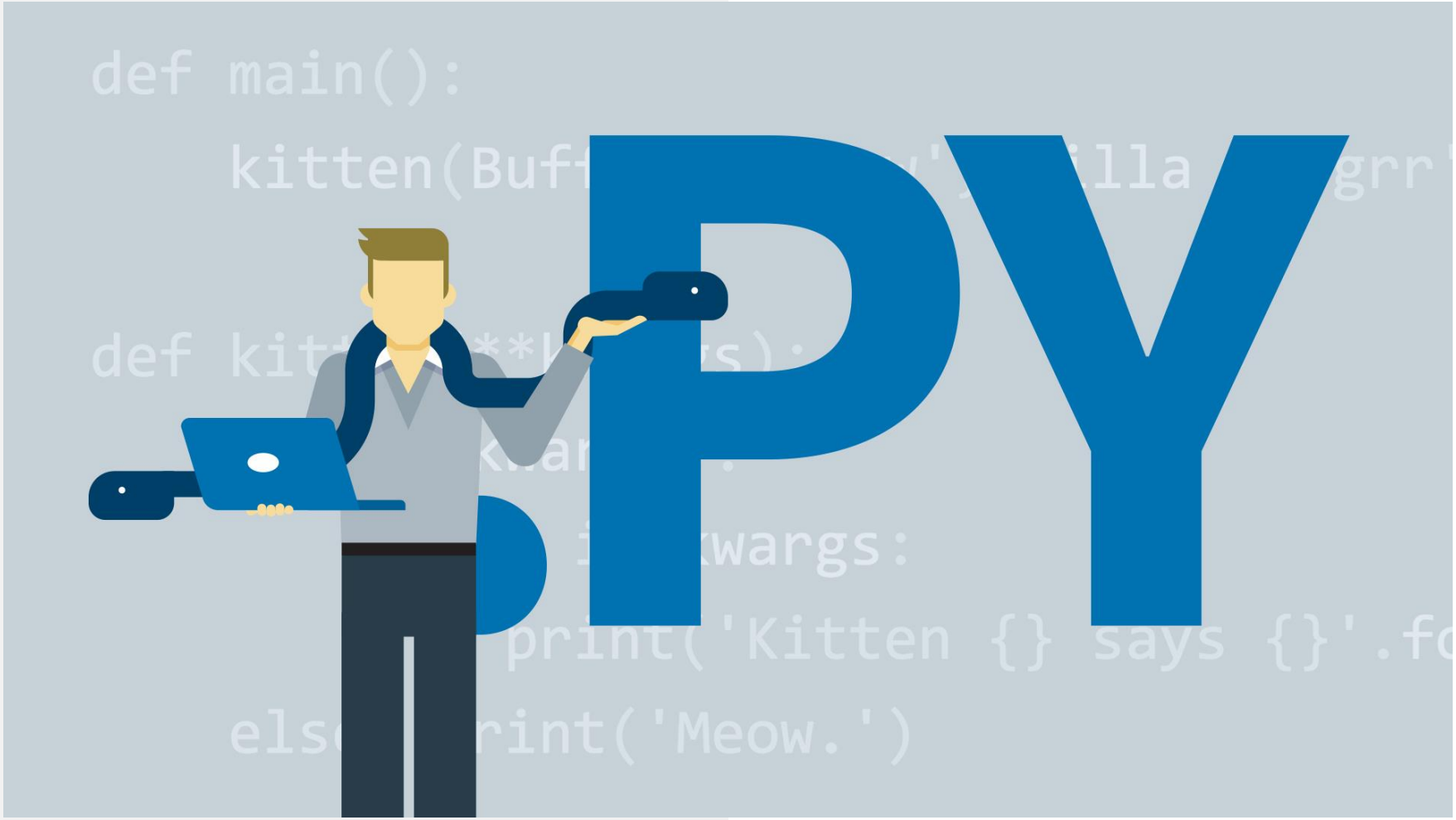
Exercice :

Jeu : l'utilisateur doit penser à un animal, et le programme doit essayer de deviner de quel animal il
s'agit en posant des questions auquelles l'utilisateur
répond par «oui» ou «non»

Pour simplifier, on se limite à 5 animaux : un oiseau, un
poisson, un chien, un tyranosaure (carnivore), un diplodocus

```
existe = input("Est ce que l'animal existe encore aujourd'hui ?")
if existe == "oui":
    vole = input("Est ce que l'animal vole ?")
    if vole == "oui":
        print "l'animal est un oiseau !"
    else:
        nage = input("Est ce que l'animal nage ?")
        if nage == "oui":
            print "l'animal est un poisson !"
        else:
            print "l'animal est un chien !"
else:
    carnivore = input("Est ce que l'animal est carnivore ?")
    if carnivore == "oui":
        print "l'animal est un tyranosore !"
    else:
        print "l'animal est un diplodocus !"
```

PYTHON



Introduction aux listes

- Les listes sont un type de données que vous pouvez utiliser pour stocker une collection de différentes informations en tant que séquence sous un même nom de variable. (Les types de données dont vous avez déjà entendu parler incluent les chaînes, les nombres et les booléens.)
  - Vous pouvez attribuer des éléments à une liste avec une expression de la forme

```
list_name = [item_1, item_2]
```

  - avec les éléments entre parenthèses. Une liste peut également être vide:

```
empty_list = [].
```

```
1. zoo_animals = ["Tigre", "lion", "Singe" ];
2. if len(zoo_animals) > 3:
    1. print " le 1ere animal dans le zoo: " + zoo_animals[0]
    2. print " le 2eme animal dans le zoo: zoo_animals[1]
    3. print " le 3eme animal dans le zoo: " + zoo_animals[2]
    4. print " le 4eme animal dans le zoo: " + zoo_animals[3]
```

Exécuter

- On doit ajouter un élément dans la liste sinon ce code n'affiche rien.

Listes: Accès par index

- Vous pouvez accéder à un élément individuel de la liste par son index.
  - Rappelle : Un index est comme une adresse qui identifie la place de l'élément dans la liste.
- The index appears directly after the list name, in between brackets, like this:

```
list_name[index]
```

Exemple : Ecrire une déclaration qui imprime le résultat de l'ajout des deuxième et quatrième éléments de la liste. Assurez-vous d'accéder à la liste par index!

```
1. numbers = [5, 6, 7, 8]
2. Print numbers [0] + numbers [2]
```

Exécuter

Listes: Accès par index

- Vous pouvez accéder à un élément individuel de la liste par son index.
  - Rappelle : Un index est comme une adresse qui identifie la place de l'élément dans la liste.
- The index appears directly after the list name, in between brackets, like this:

```
list_name[index]
```

Exemple : Ecrire une déclaration qui imprime le résultat de l'ajout des deuxième et quatrième éléments de la liste. Assurez-vous d'accéder à la liste par index!

```
1. numbers = [5, 6, 7, 8]
2. print numbers [0] + numbers [2]
3. numbers [0] = 9
4. print numbers [0]
```

Exécuter

```
12
9
```

Listes: Longueur de la liste

- Une liste n’a pas une longueur fixe. Vous pouvez ajouter des éléments à la fin d'une liste quand vous le souhaitez!

```
1. letters = ['a', 'b', 'c’]
2. letters.append('d’)
3. print len(letters)
4. print letters
```

Exécuter

```
4
['a', 'b', 'c', 'd']
```

Listes: Découpage des liste

L'opérateur d'indiçage:

- L'opérateur d'indiçage ([]) permet aussi de sélectionner des sous-chaines selon leurs indices.
- On appelle cette technique le slicing (« découpage en tranches »).

```
chain = "123456789"
print( chain[1:3] )
>>> 23

print( chain[1:len(chain)] )
>>> 23456789
```

Listes: Découpage des liste

Parfois, vous voulez seulement accéder à une partie d'une liste.

- D'abord, nous créons une liste appelée lettres.
- Ensuite, nous prenons une sous-section de la liste et la stockons dans slice . Nous faisons cela en définissant les indices que nous voulons inclure après le nom de la liste: letters [1: 3].

- En Python, lorsque nous spécifions une partie d'une liste de cette manière, nous incluons l'élément avec le premier index, 1, mais nous excluons l'élément avec le second index, 3.

- Ensuite, nous imprimons la découpe, qui imprimera ['b', 'c']. Rappelez-vous, dans Python, les indices commencent toujours à 0, donc l'élément 1 est en fait b.

```
1. letters = ['a', 'b', 'c', 'd', 'e']
2. slice = letters[1:3]
3. print slice
4. print letters
```

Exécuter

```
['b', 'c']
['a', 'b', 'c', 'd', 'e']
```

Listes: Découpage des liste

- **Exercice:**

- A partir de la liste suivant créez :
  - une liste appelée first contenant des deux premier éléments de valise.
  - une liste appelée middle contenant uniquement les deux éléments du milieu de valise.
  - une liste appelée last composée des deux derniers éléments de valise.

```
1. suitcase = ["sunglasses", "hat",
   "passport", "laptop", "suit", "shoes"]

2. first =

3. middle =

4. last =
```

Exécuter

Listes: Découpage des liste et de Chaine de caractères

- En fait, vous pouvez considérer les chaînes comme des listes de caractères: chaque caractère est un élément séquentiel dans la liste, à partir de l'index 0.
- Vous pouvez trancher une chaîne exactement comme une liste!

- Chaine = “Jullette”
- print ch[:3] # les 3 premiers caractères
- >>>Jul
- ----------
- print ch[3:] # tout sauf les 3 premiers caractères
- >>>lette
- Chaine = “Jullette”
- ----------
- print ch[:-3] # tout sauf les 3 derniers caractères
- >>>Julle
- ----------
- print ch[-3:] # les 3 dernies caractères
- >>>>>Tte

Listes: Découpage des liste et de Chaine de caractères

- En fait, vous pouvez considérer les chaînes comme des listes de caractères: chaque caractère est un élément séquentiel dans la liste, à partir de l'index 0.
- Vous pouvez trancher une chaîne exactement comme une liste!

- animals = "catdogfrog"
- # The first three characters of animals
- cat = animals[:3]
- # The fourth through sixth characters
- dog =
- # From the seventh character to the end
- frog =

TD

Exercice :

Écrire un programme qui demande à l'utilisateur d'entrer une séquence d'ADN, et qui affiche «C'est un gène» si celleci correspond à un gène, et «Ce n'est pas un gène» dans le cas contraire.
– On considérera qu'une séquence est un gène si celleci commence par un codon méthionine (ATG) et se termine par un codon STOP (TAA, TAG, TGA)

```
adn = input("entrez une séquence d'ADN: ")
if (adn[:3] == "ATG") and ((adn[3:]== "TAA")
or (adn[3:] == "TAG") or (adn[3:]== "TGA")):
    print "C'est un gène !"
else:
    print "Ce n'est pas un gène !"
```

Listes: quelque opérations

Parfois, vous voulez seulement accéder à une partie d'une liste.

- Nous pouvons insérer des éléments dans une liste.

```
animals.insert(1, "dog")
```

- Parfois, vous devez rechercher un élément dans une liste.

```
animals.index("bat")
```

```
1. Int = animals.index("bat")
2. Print int
3. Print animals.index("bat")
```

Exécuter

Listes: quelque opérations

- Si votre liste est un désordre brouillé, vous aurez peut être besoin de la trier.

```
if "Y" in chromosomes:
        print "C'est un garçon !"
```

- Attention, «=» ne copie pas les listes !

```
List1 = List2
```

- Pour copier une liste :

```
List3 =
List1[:]
```

```
1.  L1 = [5, 3, 1, 2, 4]
2.  L2 = []

1.  L2=L1
2.  L3=L1[:]
3.  L1.remove(5)

4.  print L1
5.  print L2
6.  print L3
```

```
[3, 1, 2, 4]
[3, 1, 2, 4]
[5, 3, 1, 2, 4]
```

Listes: quelque opérations

- Si votre liste est un désordre brouillé, vous aurez peut être besoin de la trier.

```
animals = ["cat", "ant", "bat"] animals.sort()
```

- D'abord, nous créons une liste appelée animaux avec trois chaînes. Les chaînes ne sont pas dans l'ordre alphabétique.
- Ensuite, nous trions les animaux dans l'ordre alphabétique. Notez que .sort () modifie la liste plutôt que de retourner une nouvelle liste.
- Pour supprimer un élément de la liste utiliser :

```
animals.remove("cat")
```

Exécuter

Listes: Pour un et tous

Si vous voulez faire quelque chose avec chaque élément de la liste, vous pouvez utiliser une boucle **'for'** .

- Nous pouvons insérer des éléments dans une liste.

```
for variable in list_name:
    # Faire des choses!
```

- Pour le nom de variable qui suit le mot-clé **'for'**; il sera attribué la valeur de chaque élément de la liste à son tour.
- Ensuite, dans list_name, le nom de la liste sur laquelle la boucle fonctionnera. La ligne se termine par un deux-points (:) et le code en retrait qui le suit sera exécuté une fois par élément dans la liste.

```
1. for number in my_list:
```

Solution

```
1. my_list = [1,9,3,8,5,7]
2. for number in my_list :
    1. print 2 * number
```

Boucle: Plus avec "For"

- A Faire :
  - Ecrivez une boucle for qui se répète sur une liste de nombre nommé start_list puis
  - Ajouter chaque nombre au carré (x ** 2) à une liste nommé square_list.
  - Puis trier square_list!
  - Imprimer les contenu de square_list.

```
1. start_list = [5, 3, 1, 2, 4]
2. square_list = []

3. for number in start_list:
4.     square_list.append(number**2)

5. square_list.sort()
6. print square_list
```

Exécuter

```
[1, 4, 9, 16, 25]
```

Listes: Pour un et tous

- Si vous voulez obtenir une liste de nombre utiliser la boucle range

```
range([debut], fin, [pas])
```

- Remarque bien :

```
print range(10)
> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print range(4, 10)
> [4, 5, 6, 7, 8, 9]
print range(4, 10, 2)
> [4, 6, 8]
```

```
1. for i in range(10) :
    1. print i
```

Solution

```
0
1
2
...
9
```

Listes: Pour un et tous

- Range peut être utilisé pour boucler sur les indices, et non pas sur les éléments d'une liste

```
1. adn = "atcacgtta«
2. for i in range(len(adn)) :
    1. print "la base n", i, "est", adn[i]
```

Solution

```
la base n° 0 est a
la base n° 1 est t
la base n° 2 est c
...
la base n° 8 est a
```

Listes: Pour un et tous

- Exercice :
- Dans la séquence protéique suivante (utilisant le code international des acides aminés), compter le nombre de Cystéines (code C)

- proteine = "CVAPGPMCAWCDSTAC"

```
1. nb_cysteine = 0
2. for aa in proteine:
    1. if aa == "C":
        1. nb_cysteine = nb_cysteine + 1
3. print "il y a", nb_cysteine, "Cystéine"
```

Solution

Listes: Pour un et tous

- Exercice :
- Dans la séquence protéique suivante (utilisant le code international des acides aminés), compter le nombre d'atomes de soufre
- rappel : les acides aminés soufrés sont la Cystéine (C) et la Méthionine (M)

- proteine = "CVAPGPMCAWCDSTAC"

```
1. nb_soufre = 0
2. for aa in proteine:
    1. if (aa == "C") or (aa == "M"):
        1. nb_soufre = nb_soufre + 1
3. print "il y a", nb_soufre, "atomes de soufre"
```

Solution

Listes: Pour un et tous

- Un des points faibles de l'utilisation de ce style d'itération est que vous ne connaissez pas l'index de la chose que vous regardez. En général, ce n'est pas un problème, mais il est parfois utile de savoir dans quelle mesure vous vous trouvez.

enumerate fonctionne en fournissant un index correspondant à chaque élément de la liste que vous lui transmettez. Chaque fois que vous parcourez la boucle, l'index sera supérieur, et l'élément sera l'élément suivant de la séquence. C'est très similaire à l'utilisation d'une boucle normale avec une liste, sauf que cela nous permet de compter facilement le nombre d'éléments observés jusqu'à présent.

```
1. choices = ['pizza', 'pasta', 'salad', 'nachos']

2. print 'Your choices are:'
3. index = 1
4. for index, item in enumerate(choices):
5.     print index+1, item
```

Solution

```
Your choices are:
0 pizza
1 pasta
2 salad
3 nachos
```

Boucle: Pendant que tu es là

While

- La boucle **while** est similaire à une instruction if: elle exécute le code à l'intérieur si une condition est vraie.
- **La différence** est que la boucle **while** continuera à exécuter tant que la condition est vraie.
- En d'autres termes, au lieu d'exécuter si quelque chose est vrai, il s'exécute pendant que cette chose est vraie.

```
1. count = 0
2. if count < 5:
     1. print "Hello, je suis un if et count = ",
        count

3. while count < 5:
     1. print "Hello,  count = ", count
     2. count = count +1
```

Solution

Boucle: Pendant que tu es là

While

- **La condition** est l'expression qui décide si la boucle va continuer à être exécutée ou non.
- Le variable condition contient la valeur True
- La boucle while vérifie si la condition a la valeur True. C'est le cas, donc la boucle est entrée.
- L'instruction d'impression est exécutée.
- Le variable condition est définie sur False.
- La boucle while vérifie à nouveau si loop_condition a la valeur True. Ce n'est pas le cas, donc la boucle n'est pas exécutée une seconde fois.
- À l'intérieur d'une boucle while, vous pouvez faire tout ce que vous pourriez faire ailleurs, y compris les opérations arithmétiques

```
1. loop_condition = True

2. while loop_condition:
     1. print "Je suis une boucle"
     2. loop_condition = False
```

Solution

```
Je suis une boucle
```

Boucle: Pendant que tu es là

While

- A faire;

Créer une boucle **while** qui imprime tous les nombres de 1 à 10 au carré (1, 4, 9, 16, ..., 100), chacun sur leur propre ligne.

```
1. num = 1

2. while num < 11 :
      1. print num**2
      2. num=num+1
```

Solution

```
1
4
9
16
25
..
81
100
```

Boucle: Pendant que tu es là

While

- Une application courante d'une boucle **while** consiste à vérifier l'entrée de l'utilisateur pour voir si elle est valide.
- Par exemple, si vous demandez à l'utilisateur d'entrer o ou n et qu'ils saisissent à la place 7, vous devez les inviter à entrer une nouvelle fois.

```
1. choix = input('vous aimer le cours? (o/n)')

2. while  choix != 'o' and choix != 'n' :
    1. choix = input("pardon j'ai pas compris
       essayer encore: ")
```

Solution

Boucle: Pendant que tu es là

- Exercice

Demander à l'utilisateur d'entrer une liste de numéros de chromosome, puis indiquer si la personne ayant ces chromosomes est un homme ou une femme.

- Comment faire pour demander à l'utilisateur d'entrer une liste ?
  - Créer une liste vide
  - L'utilisateur entre un premier chromosome
  - Tant que la valeur entrée est valide, on l'ajoute dans la liste et on demande
  - d'entrer un nouveau chromosome
  - Si la valeur entrée est vide, la liste est terminée et on passe à la suite

```
1. chromosomes = []
2. while 1:
    1. n_chromosome = input("Entrez un n° de
       chromosome : ")
    2. if no_chromosome == "  ":
        1.  break
    3. chromosomes.append(n_chromosome)

3. if "Y" in chromosomes:
        print "C'est un homme.«
   else:
        print  "C'est une femme."
```

Solution

Hobbies

Boucles Infinies

While

- Une boucle infinie est une boucle qui ne sort jamais. Cela peut arriver pour plusieurs raisons:
  - La condition de boucle ne peut pas être fausse (par exemple pendant que 1! = 2)
  - La logique de la boucle empêche la condition de boucle de devenir fausse.

```
while count > 0:
  count += 1    # a la place de count -= 1
```

Solution

BREAK=SE BRISER.

While

- **break** est une instruction d'une ligne qui signifie "quitter la boucle actuelle".
- Une autre façon de faire sortir notre boucle de comptage et arrêter l'exécution est avec l'instruction **break**.

```
while count > 0:
  count += 1    # a la place de count -= 1
```

```
1. from random import randint


2. random_number = randint(1, 10)
3. guesses_left = 3

4. while guesses_left > 0:
5.    guess = int(raw_input("Your guess: "))
6.    if num == random_number:
7.       print "You win!"
8.       break
9.    guesses_left -= 1
10. else:
11.   print "you lose!"
```

INDEX OU CLÉ

- **Un dictionnaire** est similaire à une liste, mais vous accédez aux valeurs en recherchant **une clé au lieu d'un index**.
- Une clé peut être n'importe quelle **chaîne ou numéro.** Les dictionnaires sont placés entre accolades, comme ceci:

```
d = {'key1' : 1, 'key2' : 2, 'key3' : 3}
```

On peut faire quoi avec les dictionnaire ?

```
residents = {'Puffin' : 104, 'Sloth' : 105, 'Burmese Python' : 106}
```

L'accès aux valeurs de dictionnaire par clé est similaire à l'accès aux valeurs de liste par index:

```
#  Imprimez les valeurs stockées sous les clés 'Sloth' et 'Burmese Python'.

print residents['Sloth']
print residents['Burmese Python']
```

INDEX OU CLÉ

- Comme les listes, les dictionnaires sont modifiables. Un avantage de ceci est que nous pouvons ajouter de nouvelles paires clé / valeur au dictionnaire après sa création comme suit:

```
dict_name [new_key] = new_value
```

- Une paire vide d'accolades {} est un dictionnaire vide, tout comme une paire vide de [] est une liste vide.

```
menu = {}
```

```
1. menu = {} # Dictionnaire vide
```

1. Ajoutez au moins trois autres paires clé-valeur à la variable de menu :
2. avec le nom de la plat (sous forme de "chaîne") pour la clé et le prix (un flottant ou un entier) comme valeur.

```
1. menu['Chicken Alfredo'] = 14.50
   # Ajout d'une nouvelle paire clé-valeur
1. print menu['Chicken Alfredo']

2. print "There are " + str(len(menu)) + "
   items on the menu."
```

INDEX OU CLÉ

- Parce que les dictionnaires sont mutable, ils peuvent être modifiés de plusieurs façons.

1. Les éléments peuvent être supprimés d'un dictionnaire avec la commande del:

```
del dict_name [key_name]
```

- Une nouvelle valeur peut être associée à une clé en attribuant une valeur à la clé, comme ceci:

```
dict_name [key] = new_value
```

1. zoo_animals = { }

1. Ajoutez au moins trois autres paires clé-valeur à notre zoo
2. Supprimez deux éléments en utilisant del.
3. Réglez la valeur associée à la 3eme valeur a autre chose !.

INDEX OU CLÉ

- Dans l'exemple ci-dessus, nous avons créé un dictionnaire contenant de nombreux types de valeurs.

```
my_dict = {
    "fish": ["c", "a", "r", "p"],
    "cash": -4483,
    "luck": "good" }
print my_dict["fish"][0]
```

INDEX OU CLÉ

- Dans l'exemple ci-dessus, nous avons créé un dictionnaire contenant de nombreux types de valeurs.

```
inventaire = {
 'or': 500,
 'pochette': ['silex', 'ficelle', 'pierre '],
  'sac': ['xylo', 'dagger', 'bedroll', 'pain']
}
```

1. Ajouter une clé à l'inventaire appelée «poche»
2. Définir la valeur de «poche» comme une liste composée des chaînes «coquillage», «étrange baie» et «peluches»
3. .sort () les éléments de la liste stockés sous la clé 'backpack'
4. Puis, retirez ('dagger') de la liste des éléments stockés sous la clé ‘sac'
5. Ajouter 50 au nombre stocké sous la clé 'or'

BOUCLEZ UN DICTIONNAIRE

- Vous vous demandez peut-être comment une boucle sur un dictionnaire pourrait fonctionner. Voulez-vous obtenir la clé ou la valeur?

- La réponse courte est la suivante: vous obtenez la clé que vous pouvez utiliser pour obtenir la valeur.

```
d = {'x': 9, 'y': 10, 'z': 20}
for key in d:
if d[key] == 10:
   print «  on a trouvé un10!"
```

1. d = {'a': 'apple', 'b': 'berry', 'c': 'cherry'}

1. Imprimez la clé, suivie d'un espace suivi de la valeur associée à cette clé.

1. d = {'a': 'apple', 'b': 'berry', 'c': 'cherry'}
2. for key in d:
    1. print key+" "+ d[key]

BOUCLEZ UN DICTIONNAIRE

- Une faiblesse de l'utilisation des itération (for, while) est que vous ne connaissez pas l'indice de ce que vous regardez. Généralement, ce n'est pas un problème, mais il est parfois utile de savoir à quel point de la liste vous êtes.
- Heureusement, la fonction enumerate intégrée facilite cette tâche.

```
1. choices = ['pizza', 'pasta', 'salad',
   'nachos']

2. print 'Your choices are:'
3. for index, item in enumerate(choices):
       print index, item
```

```
1. Your choices are:
2. 0 pizza
3. 1 pasta
4. 2 salad
5. 3 nachos
```

LISTES MULTIPLE

- Il est également courant de devoir parcourir deux listes à la fois. C'est là que la fonction zip intégrée est très utile.
- zip créera des paires d'éléments lorsque deux listes seront passées et s'arrêtera à la fin de la liste la plus courte.
- zip peut gérer trois listes ou plus!

```
1. list_a = ['V', 9, 17, 15, 19]
2. list_b = ['A', 4, 8, 10, 30, 40, 50, 80, 90]

3. for a, b in zip(list_a, list_b):
4.   # Add your code here!
5.   print a,b
```

```
1. V A
2. 9 4
3. 17 8
4. 15 10
5. 19 30
```

LISTES MULTIPLE

- Il est également courant de devoir parcourir deux listes à la fois. C'est là que la fonction zip intégrée est très utile.

- zip créera des paires d'éléments lorsque deux listes seront passées et s'arrêtera à la fin de la liste la plus courte.

- zip peut gérer trois listes ou plus!

```
1. list_a = ['V', 9, 17, 15, 19]
2. list_b = ['A', 4, 8, 10, 30, 40, 50, 80, 90]
3. list_c = ['C', 9, 17, 15, 19]

4. for a, b, c in zip(list_a, list_b, list_c):
5.   # Add your code here!
6.   print a,b,c
```

```
1. V A C
2. 9 4 9
3. 17 8 17
4. 15 10 15
5. 19 30 19
```

RÉFÉRENCES

- https://www.anaconda.com/download/
- http://www.programmingforbiologists.org/programming/
- https://www.codecademy.com/