

Programmation Python

(adoptez un serpent !)

Pr ADG

dahirou.gueye@uadb.edu.sn

Université Alioune Diop de Bambey

Master SI/SR

Année académique: 2021-2022

Juin 2022

« Serpent scientifique et littéraire »



Plan

Partie 1- Bases, programmation objet, réseau et Web

1. Présentation du langage python
2. Calculs, variables, types, opérations et chaînes
3. Contrôle du flux d'instructions
4. Listes et boucles
5. Dictionnaires, Tuples et ensembles
6. Fichiers
7. Fonctions
8. Programmation orientée objet
9. Modules/packages
10. Exceptions
11. Interface graphique tKinter
12. Programmation réseau: socket
13. Base de données
14. Programmation web, Framework Django et APIs python

Partie 2 - Python for Data science

15. Numpy & Matplotlib
16. Pandas
17. Traitement d'images avec OpenCV et détection d'objets avec Python

Programmation orientée objet en Python

Partie 1- Bases, programmation objet, réseau et Web

Programmation orientée objet en Python

Chapitre 1 :

Présentation du langage Python

Présentation & Historique

Présentation:

- Python, langage très moderne à la popularité grandissante
- Python est un langage portable, dynamique, extensible, gratuit, qui permet (sans l'imposer) une approche modulaire et orientée objet de la programmation.

Historique:

- Python est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles.
- 1991 : Création de la première version publique de Python par Guido van Rossum au **CWI Centrum Wiskunde & Informatica** d'Amsterdam
- 1996 : sortie de la librairie NumPy
- 2001 : création de la Python Software Foundation (PSF) qui prend en charge le développement du langage
- 2008 : sortie de Python 2.6 et 3.0
- Python est resté peu connu pendant de longues années
- Croissance lente mais constante
- Depuis 2015, python est le langage le plus enseigné aux USA

Caractéristiques

- Python est **portable**, non seulement sur les différentes variantes d'Unix, mais aussi sur les OS propriétaires: MacOS, BeOS, NeXTStep, MS-DOS et les différentes variantes de Windows. Un nouveau compilateur, baptisé **JPython**, est écrit en Java et génère du bytecode Java.
- Python est **gratuit**, mais on peut l'utiliser sans restriction dans des projets commerciaux.
- Python convient aussi bien à des **scripts** d'une dizaine de lignes qu'à des projets complexes de plusieurs dizaines de milliers de lignes.
- Python utilise le **typage dynamique** et sa **syntaxe est très simple** et, combinée à des types de données évolués (listes, dictionnaires,...), conduit à des programmes) la fois très compacts et très lisibles. A fonctionnalités égales, un programme Python (abondamment commenté et présenté selon les canons standards) est souvent de 3 à 5 fois plus court qu'un programme C ou C++ (ou même Java) équivalent, ce qui représente en général un temps de développement de 5 à 10 fois plus court et une facilité de maintenance largement accrue.
- Python gère ses ressources (mémoire, descripteurs de fichiers...) sans intervention du programmeur, par un mécanisme de **comptage de références** (proche, mais différent, d'un garbage collector).
- Il n'y a **pas de pointeurs** explicites en Python.
- Python est (optionnellement) **multi-threadé**.

Caractéristiques

■ Python est un langage de programmation

- Orienté objet
 - ✗ Classes, objets, méthodes, héritage, etc.
 - ✗ Tout est objet en Python (Y compris les chaînes de caractères, les nombres, etc.)
 - ✗ supporte l'héritage multiple et la surcharge des opérateurs
- Multi-paradigme
 - ✗ Programmation objet mais aussi programmation impérative, fonctionnelle, etc.
- Interprété: ce qui signifie qu'il n'y a pas de phase de compilation qui traduit le programme en langage machine mais les instructions sont traitées au fur et à mesure de leur lecture par l'interprète.
- Dynamique: Tout peut être modifié au cours de l'exécution du programme
- Syntaxe souple
 - ✗ Faible typage, simplification commande, affectation multiple
- Multi-plateforme, libre et open-source (MacOS, Windows, Linux, Android)
- Extensible: on peut facilement l'interfacer avec des bibliothèques C existantes. On peut aussi s'en servir comme d'un langage d'extension pour des systèmes logiciels complexes.

Caractéristiques

- La **bibliothèque standard** de Python, et les paquetages contribués, donnent accès à une grande variété de services : chaînes de caractères et expressions régulières, services UNIX standards (fichiers, pipes, signaux, sockets, threads...), protocoles Internet (Web, News, FTP, CGI, HTML...), persistance et bases de données, interfaces graphiques.
- Python optimise le temps du programmeur plutôt que le temps de calcul !
- Python: langage des Data Scientists (Big Data, Machine Learning et Deep Learning)
- Python reste le langage de programmation **le plus populaire** et **le plus puissant** à la **croissance la plus rapide** en 2022
- Python de **haut niveau** qui **continue à évoluer** et qui pourrait devenir **5 fois plus rapide d'ici 5 ans**
- **Python:** l'avantage d'une **grande communauté**: En effet python est la 5ème communauté de développeur sur StackOverflow, la 3ème sur Meetup et le 4ème langage utilisé sur GitHub

Avantages et inconvénients

■ Avantages :

- Gain de temps pour le programmeur
- Syntaxe très claire
- Nombreux modules
 - ✗ À rechercher et télécharger sur <https://pypi.org/>
- Multi-plateforme

■ Inconvénients :

- Langage interprété donc peu rapide et aura besoin d'un interpréteur pour fonctionner

Python 2.x et Python 3.x

■ Deux versions de Python cohabitent encore :

- La version 2.7, stable et assez largement utilisé
- La version 3.x, dont le développement se poursuit

■ Ces deux versions sont très proches mais incompatibles

■ Nous étudierons la version 3.x

- Actuellement (juin 2022) on est à la version 3.10 qui est stable et la toute dernière est 3.10.5

■ Les deux versions sont téléchargeables sur <https://www.python.org/downloads/>

Que fait-on en Python ?

■ Python est utilisé pour :

- **Scripting:** Du côté du scripting, en local ou via http, Python est un très bon outil qui permettra de rechercher, de trier, de récupérer des données mais aussi de les traiter et de les mettre à jour.
- **Cloud et automatisation :** Côté cloud, Python est un incontournable qui pourra automatiser l'infrastructure cloud, la configuration de serveur ainsi que leur administration. Là encore, polyvalence puisque vous trouverez des librairies chez tous les éditeurs tels que AWS, Azure, Google Cloud,...
- **Développement web:** Du côté Back-End Python dispose de frameworks évolués et populaires tels que **Django** et **Flacon** qui offrent des bases solides pour des développements d'envergures.
- **Développement de jeux avec Pygame**
- **Applications mobiles** avec des cadres comme **Kivy**. Android avec **SL4A**

Que fait-on en Python ?

- **Développement d'application classique (desktop)**
 - Python est également souvent utilisé par les admin système pour créer des tâches dites répétitives ou simplement de maintenance. D'ailleurs si vous voulez créer des applications java en codant en python, c'est possible grâce au projet **Jython**.
- **Data science, big data et IA:** Oui le domaine n°1 où Python excelle est la science des données, si **R** est aussi plébiscité, les compétences les plus recherchées le concerne directement que ce soit pour le big data, l'analytique ou l'IA avec le machine learning (ex. bibliothèque TensorFlow). De plus de nombreux packages existent et étendent ses capacités tant pour l'analyse et leur spécialisation que pour la visualisation des données. Voici donc encore une preuve de sa polyvalence et de sa « modernité ».
 - Réseau de neurones avec le module **Keras** de **TensorFlow**
 - Réseau de neurones convolutionnel **YOLOv3**
 - Chatbot avec le module **space**, **nltk**
- **Programmer arduino et Raspberry Pi en python**

Qui utilise python?

- **Google** (Guido van Rossum a travaillé pour Google de 2005 à 2012),
 - Python a été une partie importante de Google depuis le début et le reste à mesure que le système grandit et évolue.
- **Microsoft** avec
 - Microsoft store pour windows 10, visual studio,
 - Microsoft Azure: Générer et déployer vos applications Python dans le cloud, et aller plus loin avec l'IA et la science des données
- **Yahoo Groups** utilise Python "pour maintenir ses discussions de groupe".
- **YouTube** utilise Python pour produire des fonctionnalités maintenables en des temps records, avec un minimum de développeurs,
- la **Nasa** revendique l'utilisation de Python,
- Etc.

Comment utiliser python?

Python présente la particularité de pouvoir être utilisé de plusieurs manières différentes :

- En mode interactif
 - c'est à dire de manière à dialoguer avec lui directement depuis le clavier

The screenshot shows a terminal window with the title bar 'macbookpro — Python — 80x24'. The window displays the following text:
Last login: Sat Jan 11 11:14:47 on console
You have new mail.
[macs-MBP:~ DAHIROU\$ python3
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 05:52:31)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █

- Les trois caractères « supérieur à » constituent le signal d'invite, ou prompt principal, lequel vous indique que Python est prêt à exécuter une commande.
- Sous forme de programmes (scripts) et les sauvegarder sur disque.
 - (fichier complet avec l'extension .py) en utilisant par exemple un éditeur de fichier
- IDE (PyCharm, Visual studio code, TextWrangler, SublimeText, jupyter notebook etc.)

Installation de base

■ Windows

- Se rendre à la page <https://www.python.org/download> où vous trouverez un programme d'installation qui contient tout ce dont vous aurez besoin pour suivre le cours.
- Pour vérifier que vous êtes prêts, il vous faut lancer IDLE (quelque part dans le menu Démarrer) et vérifier le numéro de version.

■ MacOs

- Vous pouvez télécharger la version exécutable de python sur Mac OS à l'adresse suivante : <https://www.python.org/downloads/mac-osx/>
- Sachez aussi, si vous utilisez déjà MacPorts <https://www.macports.org>, que vous pouvez également utiliser cet outil pour installer, par exemple Python 3.6, avec la commande

```
$ sudo port install python36
```

Installation de base

■ Fedora

```
sudo yum install python3-tools
```

■ Debian/Ubuntu

- Ici encore, Python-2.7 est sans doute déjà disponible. Procédez comme ci-dessus, voici un exemple recueilli dans un terminal sur une machine installée en Ubuntu-14.04/trusty :

```
$ python3
```

```
Python 3.6.2 (default, Jul 20 2017, 12:30:02)
```

```
[GCC 6.3.1 20161221 (Red Hat 6.3.1-1)] on linux
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> exit()
```

- Pour installer python

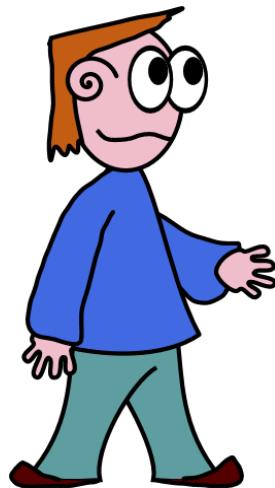
```
$ sudo apt-get install python3
```

- Pour installer idle

```
$ sudo apt-get install idle3
```

Le génie de la programmation

Appelez-moi « ordinateur » !



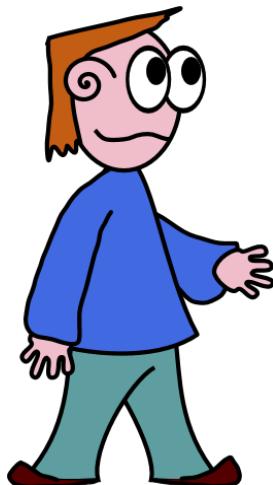
Le génie de la programmation

Ordinateur, calcule $5 + 2$!

Ok, calcul effectué sans erreur.

Tu ne m'as pas donné le résultat ?

Tu ne m'as pas demandé de l'afficher.



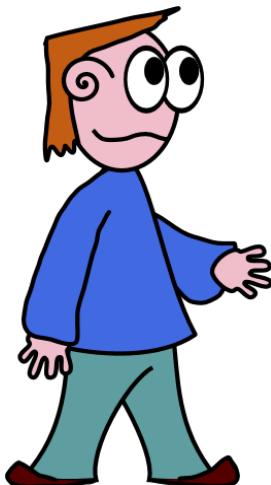
Le génie de la programmation

Ordinateur, calcule $5 + 2$ et affiche le résultat !

7.

Ajoute encore 3 à ce résultat !

Tu ne m'a pas demandé
de le garder en mémoire.
Je ne m'en souviens plus
donc recommence de zéro !



Le génie de la programmation

Ordinateur, calcule $5 + 2$ et appelle ce résultat « r » !

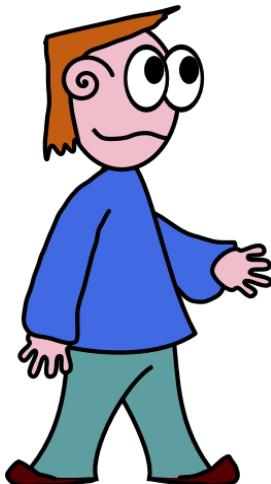
Affiche « r » !

7.

Ajoute 3 à « r » !

Affiche « r » !

10.



Le génie de la programmation

`r = 5 + 2`

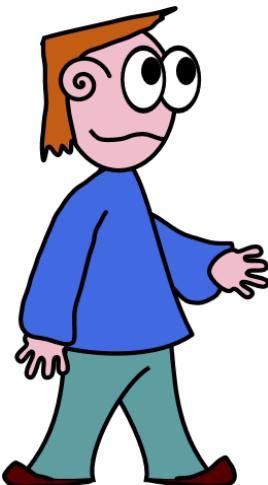
`print(r)`

7.

`r = r + 3`

`print(r)`

10.



*Ne jamais laisser le « génie »
avoir le dernier mot !*



Programmation orientée objet en Python

Chapitre 2:
Syntaxe, Variables et Types de données

Les commentaires

- Les commentaires ne sont pas exécutés par Python, ils sont destinés aux humains et non à l'ordinateur
- Ils facilitent la lecture et la compréhension du programme par d'autres personnes (ou par soi-même quelques années plus tard !)
- Commentaires simples (mono lignes)
 - Les commentaires commencent par un dièse #
 - Ils se prolongent jusqu'à la fin de la ligne
- # Ceci est mon premier commentaire !
- Commentaires longs
 - Il n'y a pas de commentaire long (type /* */) à proprement parler en Python
 - ✗ Commenter chaque ligne
 - ✗ Utiliser une chaîne de caractères longue :

"""Ceci est une chaîne longue non utilisée,
qui fait office de commentaire multi-ligne !
"""

Afficher à l'écran

■ La fonction `print()` permet d'écrire à l'écran (dans le terminal)

```
>>> print(5 + 2)
7
>>> print("Bonjour")
Bonjour
>>> print("Bonjour", "Master SI/SR")
Bonjour Master SI/SR
```

Rq: double quote " et code simple ' sont les mêmes

■ En ligne de commande, le `print()` peut être omis

```
>>> 5 + 2
7
```

Aide

■ Plusieurs commandes permettent d'obtenir de l'aide en Python

- Aide sur une classe ou une méthode :

help(list)

help(list.append)

- Lister les méthodes d'un objet :

dir(objet)

■ Aide en ligne

- <https://docs.python.org/3/>
- Aussi disponible en téléchargement ou en paquet Linux

Les types de données

Votre programme !

Modules Python

Tableaux :
Module **numpy**
(utilisation rare)

Classes et objets

Type de données de collection :

Séquences :

liste (**list**)

tuple (**tuple**)

ensemble (**set**)

ensemble fixe (**frozenset**)

Tableaux associatifs (**hash**) :

dictionnaire (**dict**)

dictionnaire ordonné

(**OrderedDict**)

Type de données de base :

Nombres :

booléen (**bool**)

entier (**int**)

flottant (**float**)

Textes :

chaîne de

caractères (**str**)

chaîne binaire (**bytes**)

Fonction

Fichier

(**file**)

Nombres entiers

```
>>> print(1)  
1  
>>> print(1 + 2)  
3  
>>> print((3 + 4) * (5 + 8))  
91
```

■ Pas de limite aux nombres entiers en Python !

```
>>> print(123456789123456789123456789)  
123456789123456789123456789
```

Nombres à virgule

```
>>> print(1.3)
```

```
1.3
```

```
>>> print(1.3e6)
```

```
1300000.0
```

- La précision du flottant Python correspond en fait à un double.

Booléens

■ Deux valeurs possibles : vrai ou faux

✗ **True**

✗ **False**

Variables

■ Une variable est un nom auquel on associe une valeur

✗ La valeur est souvent connu seulement à l'exécution du programme (par exemple c'est le résultat d'un calcul)

■ Sous Python, les noms de variables doivent en outre obéir à quelques règles simples :

✗ Un nom de variable est une séquence de lettres (a→z,A→Z) et de chiffres(0→9), qui doit toujours commencer par une lettre.

✗ Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que \$, #, @, etc. sont interdits, à l'exception du caractère _ (souligné).

✗ Attention majuscules et minuscules sont différenciées !

✗ Il est important de trouver des noms parlants

Variables

■ En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme nom de variables les 33 « mots réservés » ci-dessous

and	as	assert	break	class	continue	def
del	elif	else	except	False	finally	for
from	global	if	import	in	is	lambda
None	nonlocal	not	or	pass	raise	return
True	try	while	with	yield		

Affectations simples de variables

■ On crée une variable en lui donnant une valeur, avec =

```
>>> age = 36
```

■ On obtient la valeur d'une variable

En donnant son nom

```
>>> age
```

36

Ou bien avec print

```
>>> print(age)
```

36

■ La valeur de la variable peut être modifiée avec =

```
>>> age = 37
```

```
>>> age = age + 1 # Anniversaire
```

37

■ del permet de supprimer une variable.

del age

Affectations multiples

- Sous Python, on peut assigner une valeur à plusieurs variables simultanément

```
>>> x = y = 2020
[>>> x
2020
[>>> y
2020
>>> █
```

- On peut aussi effectuer des affectations parallèles à l'aide d'un seul opérateur :

```
[>>> note1, note2 = 13, 18.77
[>>> note1
13
[>>> note2
18.77
>>> █
```

Types de données

■ Python peut manipuler différents types de données :

- ✗ des nombres entiers (integer en anglais, abrégé en int),
- ✗ des nombres à virgule (souvent appelé flottants ou float en anglais),
- ✗ des chaînes de caractères (string en anglais, abrégé en str)
- ✗ et des booléens (valeur vraie ou fausse, bool) :

- **age** = 38 # Entier
- **poids** = 76.5 # Flottant
- **nom** = "Dahirou Gueye" # Chaîne de caractères
- **enseignant** = TRUE # Booléen
- **etudiant** = FALSE # Booléen
- **telephone** = "00221 77 501 25 35" # Chaîne de caractères!

Opérations sur les nombres

■ Opérations basique :

✗ Addition : +

✗ Soustraction : -

✗ Multiplication : *

✗ Division : /

✗ Division entière : //

✗ Modulo (reste de la division) : %

✗ Puissance : **

■ D'autres opérations sont définies dans le module **math**

Priorité des opérations

■ Sous Python, les règles de priorité sont les mêmes que celles qui vous ont été enseignées au cours de mathématique. Vous pouvez les mémoriser à l'aide de l'acronyme PEMDAS :

- P pour parenthèses

Ainsi $2*(3-1) = 4$, et $(1+1)**(5-2) = 8$.

- E pour exposants

Ainsi $2**1+1 = 3$ (et non 4), et $3*1**10 = 3$ (et non 59049 !).

- M et D pour multiplication et division, qui ont la même priorité

Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite. Ainsi dans l'expression $59*100/60$, la multiplication est effectuée en premier, et la machine doit donc ensuite effectuer $5900/60$, ce qui donne 98.

- A et S l'addition A et la soustraction S

Ainsi $2*3-1 = 5$ (plutôt que 4), et $2/3-1 = -1$ (Rappelez-vous que par défaut Python effectue une division entière).

Chaînes de caractères

■ Les chaînes de caractère s'écrivent entre guillemets (simples ou doubles)

- > `Structure="Département d'Informatique TIC"`
- > `chaine_vide= " "`

■ Les chaînes de caractère sont en Unicode

- Cela permet de gérer toutes les langues du monde et des symboles

■ Pas de type caractère en Python

- On utilise une chaîne d'un seul caractère

■ Caractères spéciaux

- Retour à la ligne : "`\n`"
- Tabulation : "`\t`"
- Antislash : "`\\"`"

Chaînes de caractères

■ Chaînes avec des guillemets à l'intérieur

- « Antislasher » les guillemets à l'intérieur de la chaîne : "**Il se nomme "Dahirou"**."
- Tripler les guillemets extérieurs : "'''**Il se nomme "Dahirou"**.''''
- Alterner guillemets doubles et simples : '**Il se nomme "Dahirou"**.'

Chaînes de caractères

■ Opérations sur les chaînes

Exemples avec s = "Bonjour"

- ✖ Obtenir la longueur d'une chaîne `len(s)` => 7
 (= le nombre de caractères)
 - ✖ Obtenir un caractère de la chaîne `s[0]` => "B" `s[-1]` => "r"
 - ✖ Obtenir une partie de la chaîne `s[0:3]` => "Bon"
 - ✖ Concaténer deux chaînes `"Hi " + s`
 - ✖ Rechercher si une chaîne est incluse dans une autre `s.find("jour")=> 3 # Trouvé en position 3 (-1 si pas trouvé)`

Chaînes de caractères

■ Opérations sur les chaînes

✗ Passer une chaîne en minuscule `s.lower()`

ou en majuscule `s.upper()`

✗ Remplacer un morceau d'une chaîne `s.replace("Bonjour", "Good morning")`

✗ Demander à l'utilisateur de saisir une chaîne `saisie = input("Entrez un nom : ")`

Chaînes de caractères

■ Formatage de chaînes de caractère : inclure des variables dans une chaîne : %

```
>>> nom = "Gueye"  
>>> prenom = "Dahirou"  
>>> "Salut %s !" % prenom  
      Salut Dahirou !  
>>> "Bonjour %s %s !" % (prenom, nom)  
>>> "Taux de réussite : %s %" % 100  
      Taux de réussite : 100 %
```

■ Attention aux faux nombres !

```
telephone1 = "00221 33 823 37 38"  
telephone2 = 00221338233738
```

- Quel est le type de donnée des deux variables ci-dessus ?
- Pourquoi a-t-on choisi ce type de donnée pour le premier cas ?

Conversions

■ Il est souvent nécessaire de convertir un type de données vers un autre

- Les fonctions `int()`, `float()`, `bool()` et `str()` permettent de convertir une valeur vers un entier, un flottant et une chaîne de caractère
- `int("8") => 8`
`str(8) => "8"`
- `input()` retourne toujours une chaîne de caractères ; il faut penser à la transformer en entier ou en flottant si l'on demande la saisie d'un nombre !

`age = int(input("Entrez votre âge : "))`

`poids = float(input("Entrez votre poids : "))`

Exercices

■ Testez les lignes d'instructions suivantes. Décrivez ce qui se passe :

```
>>> r , pi = 12, 3.14159
```

```
>>> s = pi * r**2
```

```
>>> print s
```

```
>>> print type(r), type(pi), type(s)
```

```
>>>
```

■ Quelle est, à votre avis, l'utilité de la fonction type() ?

■(Note : les fonctions seront décrites en détail, plus loin dans ce cours).

Chapitre 3 :

Contrôle du flux d'instructions

Structures de contrôle

- Pour résoudre un problème informatique, il faut toujours effectuer une série d'actions dans un certain ordre. La description structurée de ces actions et de l'ordre dans lequel il convient de les effectuer s'appelle un algorithme.
- Les structures de contrôle sont les groupes d'instructions qui déterminent l'ordre dans lequel les actions sont effectuées. En programmation moderne, il en existe seulement trois : la séquence et la sélection, que nous allons décrire dans ce chapitre, et la répétition que nous aborderons au chapitre suivant. Les conditions permettent d'exécuter des commandes seulement dans certaines situations

Sélection conditionnelle

if condition :

commande exécutée si la condition est vraie

commande exécutée si la condition est vraie...

suite du programme (exécuté que la condition soit vraie ou fausse)

- **Attention à l'indentation (= les espaces blancs en début de ligne)**
- **La condition est une comparaison utilisant un booléen ou un opérateur :**

Inférieur à : <

Supérieur ou égal à : >=

Supérieur à : >

Égal à : ==

Inférieur ou égal à : <=

Different de : !=

Sélection conditionnelle

- Le résultat doit être ceci:

```
>>> a=120
>>> if(a>100):
...     print("a dépassé la centaine")
...

```

- Frappez encore une fois <Enter>. Le programme s'exécute, et vous obtenez :

```
a dépassé la centaine
>>> █
```

Instructions successives

- Il est possible d'ajouter plusieurs conditions successives, et un bloc par défaut (*else = sinon*)

if *condition1*:

commande exécutée si condition1 est vraie

commande exécutée si condition1 est vraie...

elif *condition2*:

commande exécutée si condition1 est fausse et condition2 est vraie...

else:

commande exécutée si condition1 et condition2 sont fausses...

suite du programme (exécutée que les conditions soient vraies ou fausses)

Instructions successives

■ Exemple1 de condition :

```
[>>> a=0
[>>> if a>0:
[...     print("a est positif")
[... elif a<0:
[...     print("a est négatif")
[... else:
[...     print("a est nul")
[... ]
```

Exemple2 de condition :

```
age = int(input("veuillez saisir votre âge : "))
```

```
if age == 0 :
    print("Vous êtes un nouveau-né.")
elif age < 18 :
    print("Vous êtes un enfant.")
elif age >= 65 :
    print("Vous êtes une personne âgée.")
else :
    print("Vous êtes un adulte.")
```

```
print("L'année prochaine vous aurez ", age + 1, " ans.")
```

Instructions imbriquées

- Il est possible d'imbriquer les unes dans les autres plusieurs instructions composées, de manière à réaliser des structures de décision complexes

■ Exemple1:

```
[>>> if embranchement == "vertébrés":  
[...     if classe == "mammifères":  
[...         if ordre == "carnivores":  
[...             if famille == "félins":  
[...                 print("c'est peut-être un chat")  
[...             print("c'est en tous cas un mammifère")  
[...         elif classe == "oiseaux":  
[...             print("c'est peut-être un canari")  
... print("la classification des animaux est complexe")
```

Instructions imbriquées

■ Exemple2 :

```
age = int(input("veuillez saisir votre âge : "))
poids = float(input("veuillez saisir votre poids : "))

if age == 0 :
    print("Vous êtes un nouveau-né.")

    if poids > 10.0 :
        print("Je pense qu'il y a une erreur sur le poids !")
```

■ Les opérateurs logiques **not**, **and** et **or** permettent de combiner plusieurs conditions entre elles

✗ Respectivement NON, ET et OU logique

```
if (age < 18) or (age > 65) :
    print("Vous ne travaillez probablement pas.")
```

Exercice 1 : les champignons

■ On souhaite réaliser un programme Python permettant d'identifier des champignons. Afin de rester simple, nous nous limiterons aux 4 d'espèces de champignon suivant :

- Les Ascomycètes (sans chapeau), comprenant :
 - ✗ Les morilles (avec pied)
 - ✗ Les truffes (sans pied)
- Les Basidiomycètes (avec chapeau), comprenant :
 - ✗ Les bolets (sans lamelle)
 - ✗ Les autres champignons (avec lamelles)

■ Le programme posera à l'utilisateur des questions de la forme
« Le champignon a-t-il un chapeau (o/n) ? », via la fonction input()

Exercice 1 : les champignons

■ On souhaite réaliser un programme Python permettant d'identifier des champignons. Afin de rester simple, nous nous limiterons aux 4 espèces de champignon suivant :

- Demander si le champignon a un chapeau :
 - ✗ S'il n'a pas de chapeau :
 - ⌚ C'est un Ascomycète
 - ⌚ Demander si le champignon a un pied
 - S'il a un pied, c'est une morille
 - S'il n'a pas de pied, c'est une truffe
 - ✗ S'il a un chapeau :
 - ⌚ C'est un Basidiomycète
 - ⌚ Demander si le champignon a des lamelles
 - S'il n'a pas de lamelles, c'est un bolet
 - S'il a des lamelles, c'est un autre champignon

Exercice 1 : les champignons



Exercice 1 : les champignons



Programmation orientée objet en Python

Chapitre 4:
Listes et Boucles

Listes

- Une liste comme une collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets

```
>>> jour = ['lundi', 'mardi', 'mercredi', 2020, 18.5, 'jeudi', 'vendredi']
>>> jour
['lundi', 'mardi', 'mercredi', 2020, 18.5, 'jeudi', 'vendredi']
>>> print (jour)
['lundi', 'mardi', 'mercredi', 2020, 18.5, 'jeudi', 'vendredi']
>>> █
```

- il est possible de changer les éléments individuels d'une liste :

```
>>> print (jour)
['lundi', 'mardi', 'mercredi', 2020, 18.5, 'jeudi', 'vendredi']
>>> jour[3] = jour[3] +37
>>> jour
['lundi', 'mardi', 'mercredi', 2057, 18.5, 'jeudi', 'vendredi']
>>> █
```

- Fonction len()

```
>>> print(len(jour))
```

Listes

■ Fonction del

```
[>>> del(jour[4])
[>>> jour
['lundi', 'mardi', 'mercredi', 2057, 'jeudi', 'vendredi']
>>> ]
```

■ Fonction append()

```
[>>> jour.append('samedi')
[>>> jour
['lundi', 'mardi', 'mercredi', 2057, 'jeudi', 'vendredi', 'samedi']
>>> ]
```

Accès aux éléments

■ Soit les listes

```
>>> nombres = [15, 48, 20, 35]
```

```
>>> animaux = ["éléphant", "girafe", "rhinocéros", "gazelle"]
```

```
>>> liste3 = [2000, "Bambey", 3.14, ["Ablaye", "Daouda", 1980]]
```

>>> print(nombres[2])	>>> print(nombres[1:3])	>>> print(nombres[2:3])	>>> print(nombres[2:])	>>> print(nombres[:2])	>>> print(nombres[-1])	>>> print(nombres[-2])
20	[48, 20]	[20]	[20, 35]	[15, 48]	35	20

■ Notation

- la notation liste[i:j] désigne les éléments i à j-1 de la liste
- la notation [i:] désigne les éléments i et suivants
- la notation [:i] désigne les éléments 0 à i-1

Les listes sont modifiables

■ Modification par affectation directe

■ Exemple1:

```
>>> nombres[0] = 12
```

```
>>> nombres
```

```
[12, 48, 20, 35]
```

■ Exemple2:

```
>>> liste3[3][1] = "Dahirou"
```

```
>>> liste3
```

```
[2000, "Bambey", 3.14, ["Ablaye", "Dahirou", 1980]]
```

Les listes sont modifiables

■ Les listes peuvent être modifiées

- Ajouter un élément à la fin
- Ajouter un élément à une position donnée
- Enlever un élément
- Enlever un élément à une position donnée

Exemples

```
animaux = ["éléphant", "girafe",  
          "rhinocéros", "gazelle"]
```

```
animaux.append("lion")
```

```
animaux.insert(0, "hippopotame")
```

```
animaux.remove("gazelle")
```

```
del animaux[-2]
```

Que contient alors la liste animaux ?

Les listes sont modifiables

Exemples

- Ajouter tous les éléments d'une autre liste `animaux.extend(["lion", "buffle"])`
- Trier une liste (ordre numérique / alphabétique) `animaux.sort()`
- Concaténer deux listes `animaux + ["lion", "buffle"]`
- Inverser l'ordre d'une liste `animaux.reverse()`
- Cloner une liste `Lcopie=list(animaux)`

Listes (Exemples (1))

■ Supprimer une entrée avec son index avec la fonction **del**

```
>>> liste = ["a", "b", "c"]
```

```
>>> del liste[1]
```

```
>>> liste
```

```
['a', 'c']
```

□ Supprimer une entrée avec sa valeur avec la méthode **remove**

```
>>> liste = ["a", "b", "c"]
```

```
>>> liste.remove("a")
```

```
>>> liste
```

```
['b', 'c']
```

Listes (Exemples (2))

- Inverser les valeurs d'une liste avec la méthode **reverse**

```
>>> liste = ["a", "b", "c"]
```

```
>>> liste.reverse()
```

```
>>> liste
```

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

- Compter le nombre d'items d'une liste

```
>>> liste = [1,2,3,5,10]
```

```
>>> len(liste)
```

5

Listes (Exemples (3))

□ Compter le nombre d'occurrences d'une valeur

```
>>> liste = ["a","a","a","b","c","c"]
```

```
>>> liste.count("a")
```

3

```
>>> liste.count("c")
```

2

□ Trouver l'index d'une valeur

```
>>> liste = ["a","a","a","b","c","c"]
```

```
>>> liste.index("b")
```

3

Listes (Exemples (4))

- Trouver un item dans une liste avec le mot clé **in**

```
>>> liste = [1,2,3,5,10]
```

```
>>> 3 in liste
```

True

```
>>> 11 in liste
```

False

- La fonction **range**

```
>>> range(10)
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Listes (Exemples (5))

❑ Agrandir une liste

```
>>> x = [1, 2, 3, 4]
```

```
>>> y = [4, 5, 1, 0]
```

```
>>> x.extend(y)
```

```
>>> print x
```

```
[1, 2, 3, 4, 4, 5, 1, 0]
```

❑ Additionner deux listes

```
>>> x = [1, 2, 3]
```

```
>>> y = [4, 5, 6]
```

```
>>> x + y
```

```
[1, 2, 3, 4, 5, 6]
```

Listes (Exemples (6))

❑ Multiplier une liste:

```
>>> x = [1, 2]
```

```
>>> x*5
```

```
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

❑ Initialiser une liste:

```
>>> [0] * 5
```

```
[0, 0, 0, 0, 0]
```

Listes (Exemples (7))

- ❑ Obtenir le plus grand élément d'une liste, ou le plus petit :

```
>>> max([2, 1, 4, 3])
```

4

```
>>> min([2, 1, 4, 3])
```

1

- ❑ Faire la somme d'une liste:

```
>>> sum([2, 1, 4, 3])
```

10

Listes et chaînes de caractères

■ Couper une chaîne de caractères en une liste de chaînes : **split**

✗ **split sans argument** : couper sur les espaces, sauts de ligne, tabulations

```
>>> "Une phrase avec des mots".split()
```

```
["Une", "phrase", "avec", "des", "mots"]
```

✗ **Split avec argument** : couper sur un séparateur donné

```
>>> "Plan-Jaxaay".split("-")
```

```
["Plan", "Jaxaay"]
```

■ Assembler une liste de chaînes en une seule chaîne : **join**

```
>>> "/".join(["répertoire", "sous_répertoire", "fichier"])
```

```
"répertoire/sous_répertoire/fichier"
```

Boucles

■ Une boucle permet d'exécuter plusieurs fois les mêmes commandes

- La boucle parcourt une séquence

for variable in sequence :

commande répétée plusieurs fois

commande répétée plusieurs fois

suite du programme (exécuté une seule fois)

Boucles

- La séquence peut être :

✗ une variable

- Chaîne de caractères
- Liste, ensemble, tuple
- ...

✗ une liste d'indice générée avec la fonction range()

`range(5) => [0, 1, 2, 3, 4]`

Boucles

- Exemple de séquence de type chaîne de caractères:

```
[>>> prenom="Dahirou"
[>>> for car in prenom:
[...     print(car+' *',end=' ')
[...
D * a * h * i * r * o * u * >>> ]
```

- L'instruction `for` permet donc d'écrire des boucles, dans lesquelles l'itération traite successivement tous les éléments d'une séquence donnée.
- L'argument `end = " "` signifie que vous souhaitez remplacer le saut à la ligne par un simple espace. Si vous supprimez cet argument, les nombres seront affichés les uns en-dessous des autres (sur la verticale).

Boucles

■ Exemple de séquence de type Liste

```
[>>> liste=['SATIC','SDD','ECOMIJ']
[>>> for ufr in liste:
[...     print('longueur de la chaine',ufr,'=',len(ufr))
[...
 longueur de la chaine SATIC = 5
 longueur de la chaine SDD = 3
 longueur de la chaine ECOMIJ = 6
>>> █
animaux = ["éléphant", "biche", "rhinocéros", "brochet"]
environnements = ["savane", "forêt", "savane", "rivière"]
```

✗ Afficher un animal par ligne

```
for animal in animaux : print(animal)
print("C'est fini !")
```

✗ Afficher un animal par ligne avec son numéro

```
for i in range(len(animaux)) :
    print("Numéro " + str(i) + " : " + animaux[i])
print("C'est fini !")
```

✗ Afficher un animal par ligne avec son environnement

```
for i in range(len(animaux)) :
    print(animaux[i] + " dans la " + environnements[i])
```

Boucles (boucler sur plusieurs listes avec zip)

■ Exemples

```
animaux = ["éléphant", "biche", "rhinocéros", "brochet"]
```

```
environnements = ["savane", "forêt", "rivière", "rivière"]
```

```
for animal, envir in zip(animaux, environnements):
```

```
    print(animal + " vit dans " + envir)
```

■ Ce qui donne :

éléphant vit dans savane

biche vit dans forêt

Rhinocéros vit dans savane

brochet vit dans rivière

Boucles (boucler sur plusieurs listes avec des boucles imbriquées)

■ Exemples

```
animaux = ["éléphant", "biche", "rhinocéros", "brochet"]
```

```
environnements = ["savane", "forêt", "savane", "rivière"]
```

```
for animal in animaux:
```

```
    for envir in environnements:
```

```
        print (animal+ " dans " + envir )
```

■ Ce qui donne :

éléphant dans savane

éléphant dans forêt

éléphant dans rivière

biche dans savane

biche dans forêt

biche dans rivière

rhinocéros dans savane

rhinocéros dans forêt

rhinocéros dans rivière

brochet dans savane

brochet dans forêt

brochet dans rivière

Boucles

- **break** permet d'interrompre la boucle
- **continue** permet de passer immédiatement à l'élément suivant
- La boucle **for** peut aussi avoir un bloc **else** :

for variable in liste :

if condition d'exclusion :

continue

commande répétée plusieurs fois

if condition :

commande exécutée si condition satisfaite

break

else :

commande exécutée si condition jamais satisfaite

Boucles

■ **continue** : permet de passer immédiatement à l'élément suivant

for variable in liste :

if condition d'exclusion : continue

commande répétée plusieurs fois

for i in range(10):

if i == 2:

continue

print(i)

Boucles

■ break et else :

✗ **break** permet d'interrompre la boucle

```
for i in range(10):
    if i == 2:
        break
```

```
    print(i)
```

✗ le bloc **else** est exécuté si la boucle est allée jusqu'au bout (sans rencontrer **break**)

for variable in liste :

commande répétée plusieurs fois

if condition :

commande exécutée si condition satisfaite

break

else :

commande exécutée si condition jamais satisfaite

Boucles

- La boucle **while** dure tant qu'une condition est satisfaite :

while condition :

commande répétée plusieurs fois

suite du programme (la boucle est finie)

- Elle peut aussi avoir un **break** et un **else**

- Exemple: la liste des carrés et des cubes des nombres de 1 à 12

```
[>>> a=0
[>>> while a<12:
[...     a=a+1
[...     print(a , a**2 , a**3)
[...
```

- Afficher les dix premiers termes d'une suite appelée « Suite de Fibonacci »

Boucles

- Afficher les dix premiers termes d'une suite appelée « Suite de Fibonacci ». Il s'agit d'une suite de nombres dont chaque terme est égal à la somme des deux termes qui le précédent

```
[>>> a, b, c = 1, 1, 1
[>>> while c < 11 :
[...     print(b, end = " ")
[...     a, b, c = b, a+b, c+1
```

■ Résultat

```
[...
1 2 3 5 8 13 21 34 55 89 >>> ]
```

Exercice 2 : le dot-plot

■ Le dot-plot est une technique très simple pour analyser visuellement deux séquences d'ADN

■ Principe :

- ✗ Mettre une séquence à l'horizontal et l'autre en vertical, sur une matrice
- ✗ Mettre un X dans les cases où les bases sont les mêmes dans les deux séquences

■ Faire un programme Python pour générer un dot-plot

- ✗ Les séquences d'ADN sont des chaînes de caractères
- ✗ On générera le dot-plot ligne par ligne et on écrira chaque ligne avec print()

A	T	G	G	T	C	A	A	T
A	X						X	X
T		X			X			X
G			X	X				
G			X	X				
T				X	X			
C						X		
A	X						X	X
A	X						X	X
T	X			X				X

Exercice 2: Nombres paires d'une liste

■ Exemple2 : Garder seulement les nombres pairs d'une liste :

nombres = [1, 4, 7, 8, 12, 15]

nombres_pairs = [] ...????

Listes procédurales (*comprehension list*)

- Permet de créer une liste sans donner les éléments un à un mais en donnant une boucle pour la remplir

```
>>> [len(mot) for mot in "Un texte avec des mots".split()]
```

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

sum() retourne la somme d'une liste

```
>>> sum(len(mot) for mot in "Un texte avec des mots".split())
```

```
18
```

- Plusieurs boucles peuvent être combinées

```
print(" " + seq1)
```

```
print("\n".join(base2 + "".join([" ", "X"])[base1 == base2] for base1 in
seq1) for base2 in seq2 ))
```

- => Plusieurs approches différentes pour faire la même chose

✗ Mais en général, une approche est plus simple et claire !