FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

DÉPARTEMENT DE MATHÉMATIQUES ET INFORMATIQUE

PROGRAMMATION PYTHON

(OPTIMISATION NUMERIQUE ET PYTHON)

Master : MNSA

Pr: Youssef Ouassit

Objectifs du module

- Connaître les bases du langage Python
- Savoir utiliser la programmation orientée objet en Python.
- Implémenter quelques exemples des méthodes d'optimisation
- Manipulation des librairies d’optimisation en Python.

Descriptif horaire

- 11 heures de cours ;
- 10 heures de TD/TP;
- Mini-projets :
  - Apprentissage automatique
  - Les heuristiques et méta-heuristiques
    - Algorithmes génétiques
    - Algorithme Glouton
    - Recuit simulé
  - ….

FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

DÉPARTEMENT DE MATHÉMATIQUES ET INFORMATIQUE

Introduction au Python : Présentation de Python, La syntaxe de base

Master : MNSA

Pr: Youssef Ouassit

Plan Séance 1

1. **Introduction à Python**
    a. Historique
    b. Syntaxe et concept de base
    c. Installation
    d. TD/TP N° 1

Un peu d’histoire

Origines (1980s - 1990)

Créateur : Python a été créé par **Guido van Rossum**, un programmeur néerlandais, à la fin des années 1980. Van Rossum travaillait au Centrum Wiskunde & Informatica (CWI) aux Pays-Bas lorsqu'il a commencé à développer Python.

Nom : Le nom "Python" ne vient pas du serpent, mais plutôt du groupe comique britannique Monty Python. Van Rossum voulait un nom court, unique et un peu mystérieux.

Un peu d’histoire

La version 0.9.0 (1991)

Première version publique : Python 0.9.0 a été publié en février 1991. Cette version incluait déjà des fonctionnalités clés telles que les classes avec héritage, les exceptions, et les fonctions avec arguments nommés. Les structures de données intégrées comme les listes, les dictionnaires et les chaînes de caractères étaient également présentes.

Un peu d’histoire

La version 1.0 (1994)

Python 1.0 a été publié en janvier 1994. Cette version a introduit des fonctionnalités importantes comme les modules, les expressions **lambda**, les fonctions **map**, **filter**, et **reduce**.

Adoption : À cette époque, Python a commencé à gagner en popularité grâce à sa syntaxe simple et à sa capacité à s'intégrer facilement à d'autres langages et outils.

Un peu d'histoire

La version 2.x (2000)

Python 2.0 a été publié en octobre 2000. Cette version a introduit de nombreuses améliorations, notamment la collecte des ordures (**garbage collection**) pour gérer la mémoire, la **liste des compréhensions**, et le support complet de **Unicode**.

Python 2.x a continué à évoluer avec de nombreuses versions mineures, ajoutant des fonctionnalités et améliorant la performance. Python 2.7, publié en 2010, est devenu la version finale de la série Python 2.

Un peu d’histoire

La version 3.x (2008 à présent)

Python 3.0 a été publié en décembre 2008. Cette version a introduit des changements majeurs qui ont **rompu** la compatibilité avec Python 2.x, rendant la transition difficile pour certains projets.
L'objectif principal de Python 3 était de corriger les défauts fondamentaux du langage, en rendant la syntaxe plus cohérente et en supprimant les fonctionnalités obsolètes.

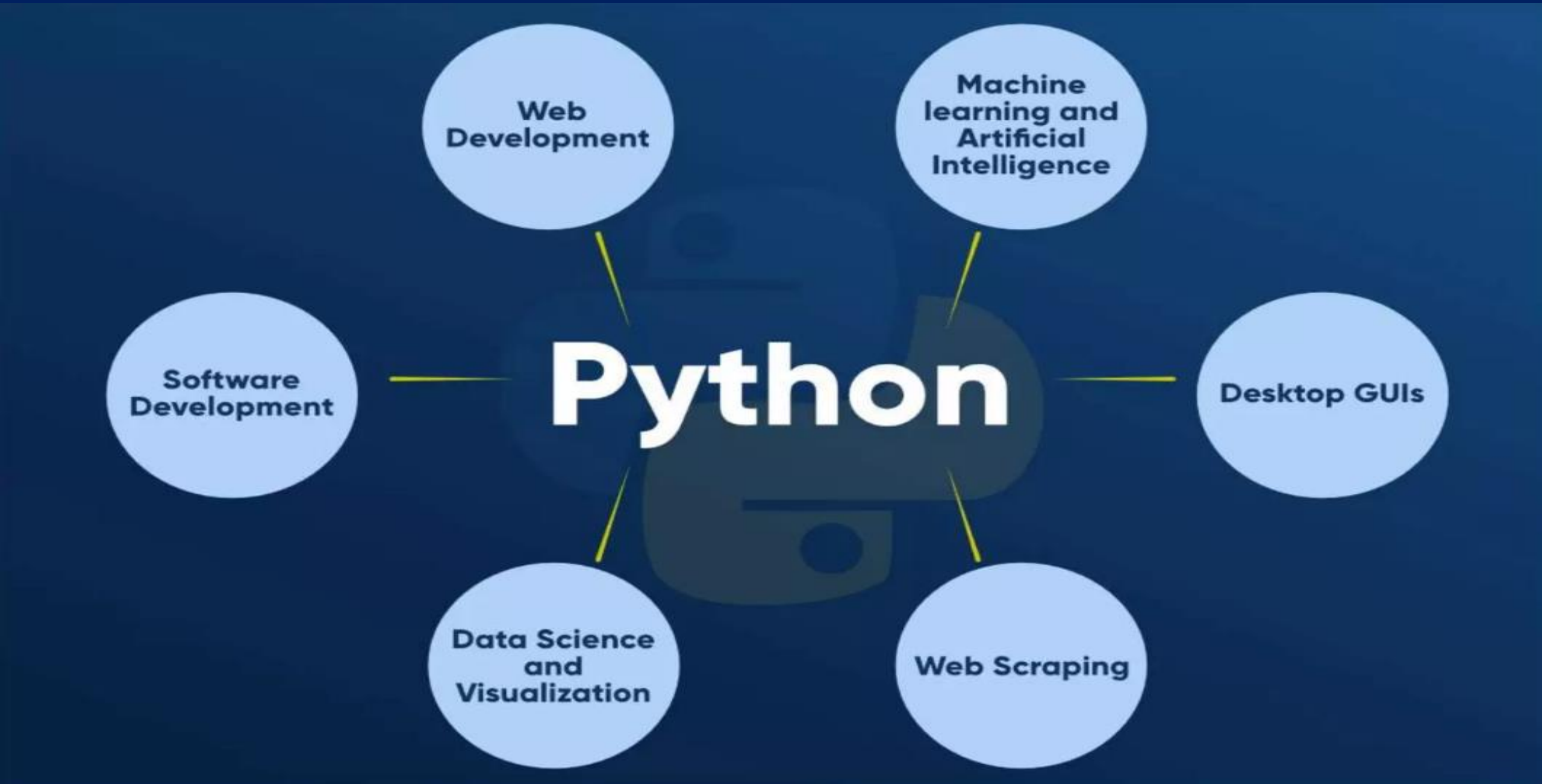
Adoption de Python 3 : Bien que la transition ait été lente au départ, Python 3 est maintenant la version principale utilisée par la majorité de la communauté Python. Le support officiel de Python 2 a pris fin le 1er janvier 2020, marquant la fin d'une ère.

Un peu d’histoire

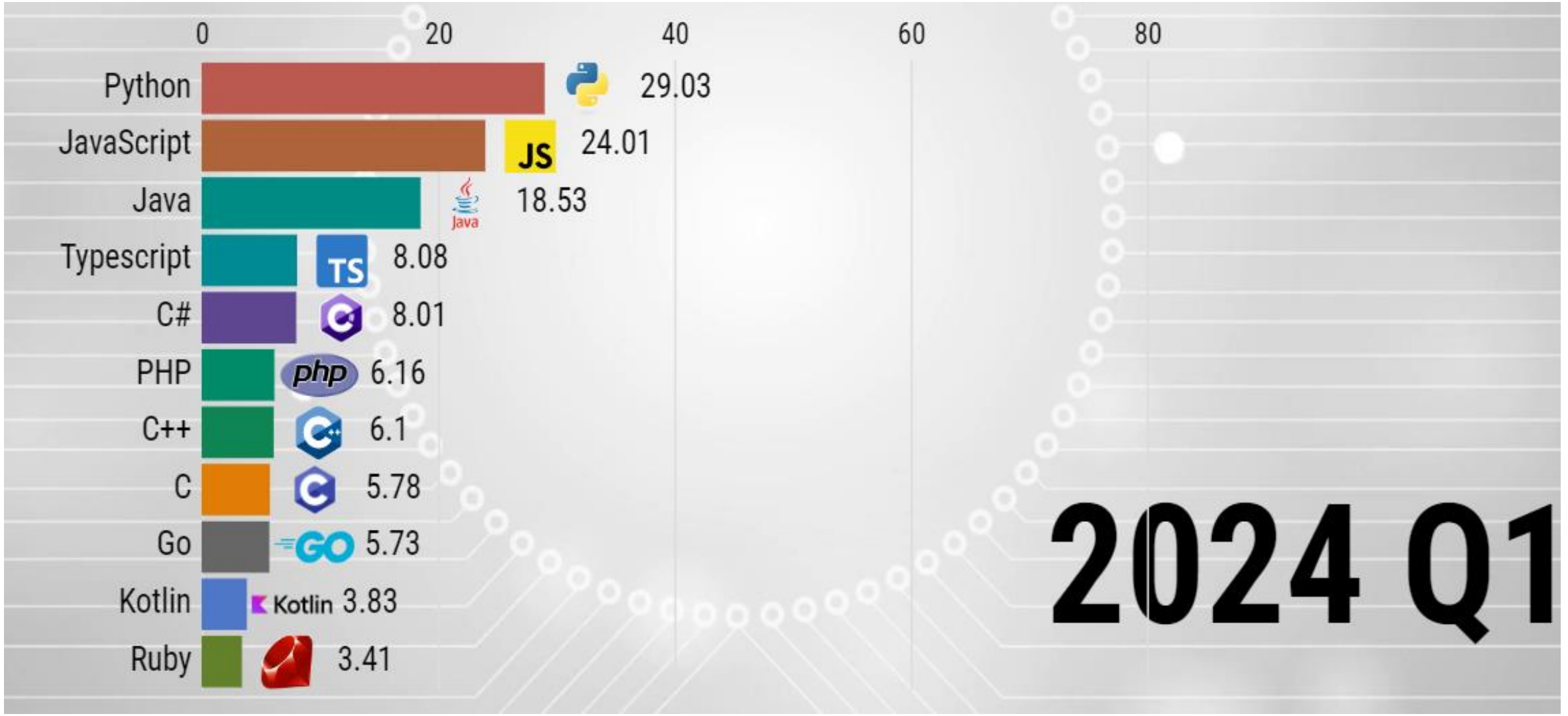
Python 3.x :

- Support de l'Unicode (sans utiliser le préfixe 'u')
- Division flottante par défaut pour les opérateurs de division
- print est devenu une fonction
- Meilleure gestion des exceptions
- Annotation de type (à partir de 3.5)
- range() est devenu un itérable par défaut (au lieu de générer une liste en 2.x)
- …

Domaines d’applications



Un peu d’histoire



statisticsanddata.org

Pourquoi Python ?

Caractéristiques de Python:

Simple: avec une syntaxe qui privilège la lisibilité, libéré de celle de C/C++ :

- Programme compact et lisible
- Pas de notions complexe (pointeurs, …)

Portable : disponible sous toutes les plateformes (Unix, Windows, …)

Open source: Supervisées par la Python Software Foundation (PSF)

Pourquoi Python ?

Riche: il incorpore plusieurs possibilités de langage:

tiré de la **programmation impérative** : séquences des instructions qui change l'état du système, structure de contrôle, manipulation de nombres comme les flottants, doubles, complexe, de structures complexes comme les tableaux, les dictionnaires, etc.

tiré des **langages de script** : accès au système, manipulation de processus, de l'arborescence fichier, d'expressions rationnelles, etc.

tiré de la **programmation fonctionnelle** : les fonctions sont dites «fonction de première classe», car elles peuvent être fournies comme argument d'une autre fonction, il dispose aussi de lambda expression, de générateur etc.

tiré de la **programmation orienté objet** : définition de classe, héritage multiple, introspection (consultation du type, des méthodes proposées), ajout/retrait dynamique de classes, de méthode, compilation dynamique de code, délégation ("duck typing"), passivation/activation, surcharge d'opérateurs, etc.

Pourquoi Python, ses caractéristiques ?

Il est :

- **dynamique** : il n'est pas nécessaire de déclarer le type d'une variable dans le source. Le type est associé lors de l'exécution du programme ;

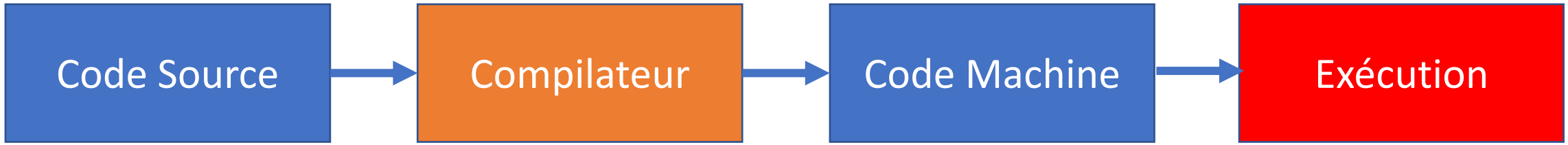
- **fortement typé** : les types sont toujours appliqués (un entier ne peut être considéré comme une chaîne sans conversion explicite, une variable possède un type lors de son affectation).

- **compilé/interprété** à la manière de Java. Le source est compilé en bytecode (pouvant être sauvegardé) puis exécuté sur une machine virtuelle.

Pourquoi Python, ses caractéristiques ?

Compilé vs Pseudo-Compilé vs interprété

Un langage compilé:



- Le code source est directement traduit en code machine natif spécifique à une plateforme.
- Exécution rapide car tout est compilé en amont.
- Nécessite une recompilation pour chaque type de machine.
- Exemple : C, C++

Pourquoi Python, ses caractéristiques ?

Compilé vs Pseudo-Compilé vs interprété

Un langage interprété:

- Le code source est interprété ligne par ligne à l'exécution.
- Facilité de développement et de debugging, mais exécution plus lente.
- Aucune compilation préalable en code machine.

Pourquoi Python, ses caractéristiques ?

Compilé vs Pseudo-Compilé vs interprété

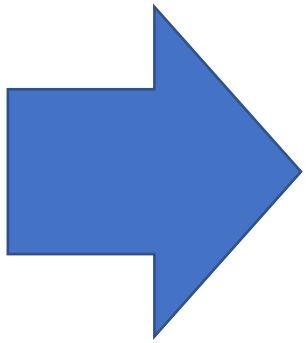
Un langage Pseudo-Compilé:

- Le code source est d'abord compilé en bytecode indépendant de la plateforme.
- Le bytecode est ensuite interprété ou compilé en code machine par la VM lors de l'exécution.
- Portabilité élevée car le bytecode peut être exécuté sur toute machine avec une VM.

Pourquoi Python, ses caractéristiques ?

Compilé vs Pseudo-Compilé vs interprété

Python c'est quoi ?



Python est principalement considéré comme un langage interprété, mais il inclut une étape intermédiaire de compilation en bytecode avant que l'interpréteur ne prenne le relais. C'est pour cela qu'on dit que Python est pseudo-compilé ou qu'il combine des éléments des langages compilés et interprétés.

Pourquoi Python, ses caractéristiques ?

Compilé vs Pseudo-Compilé vs interprété

Un langage Pseudo-Compilé:

```
import dis

def add(a, b):
    return a + b

# Désassemblage du bytecode
dis.dis(add)
```

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 BINARY_ADD
            6 RETURN_VALUE
```

Plusieurs implémentation de Python

La spécification VS l'implémentation :

Python le langage (la spécification) : Python est un langage de programmation de haut niveau, interprété.

C'est une spécification du langage, ce qui signifie qu'il définit la syntaxe, la sémantique, et le comportement de base que toute implémentation de Python doit suivre.

A travers : Les PEP (Python Enhancement Proposals)

Plusieurs implémentation de Python

La spécification VS l'implémentation :

Les implémentations de Python:

- ***CPython*** : L'implémentation standard de Python, écrit en langage C.
- ***PyPy***: Une implémentation de Python avec un compilateur Just-In-Time (JIT), qui peut rendre l'exécution du code Python plus rapide que CPython.
- ***Jython*** : Une implémentation de Python qui s'exécute sur la machine virtuelle Java (JVM). Elle permet au code Python d'interagir avec les bibliothèques Java.
- ***IronPython*** : Une implémentation de Python pour le framework .NET, permettant d'utiliser Python avec les langages et bibliothèques .NET.
- ***MicroPython*** : Une implémentation légère de Python conçue pour les microcontrôleurs et les environnements contraints.

Plusieurs implémentation de Python

La spécification VS l'implémentation :

Choisir l'implémentation:

```
$ python script.py
```

```
$ pypy script.py
```

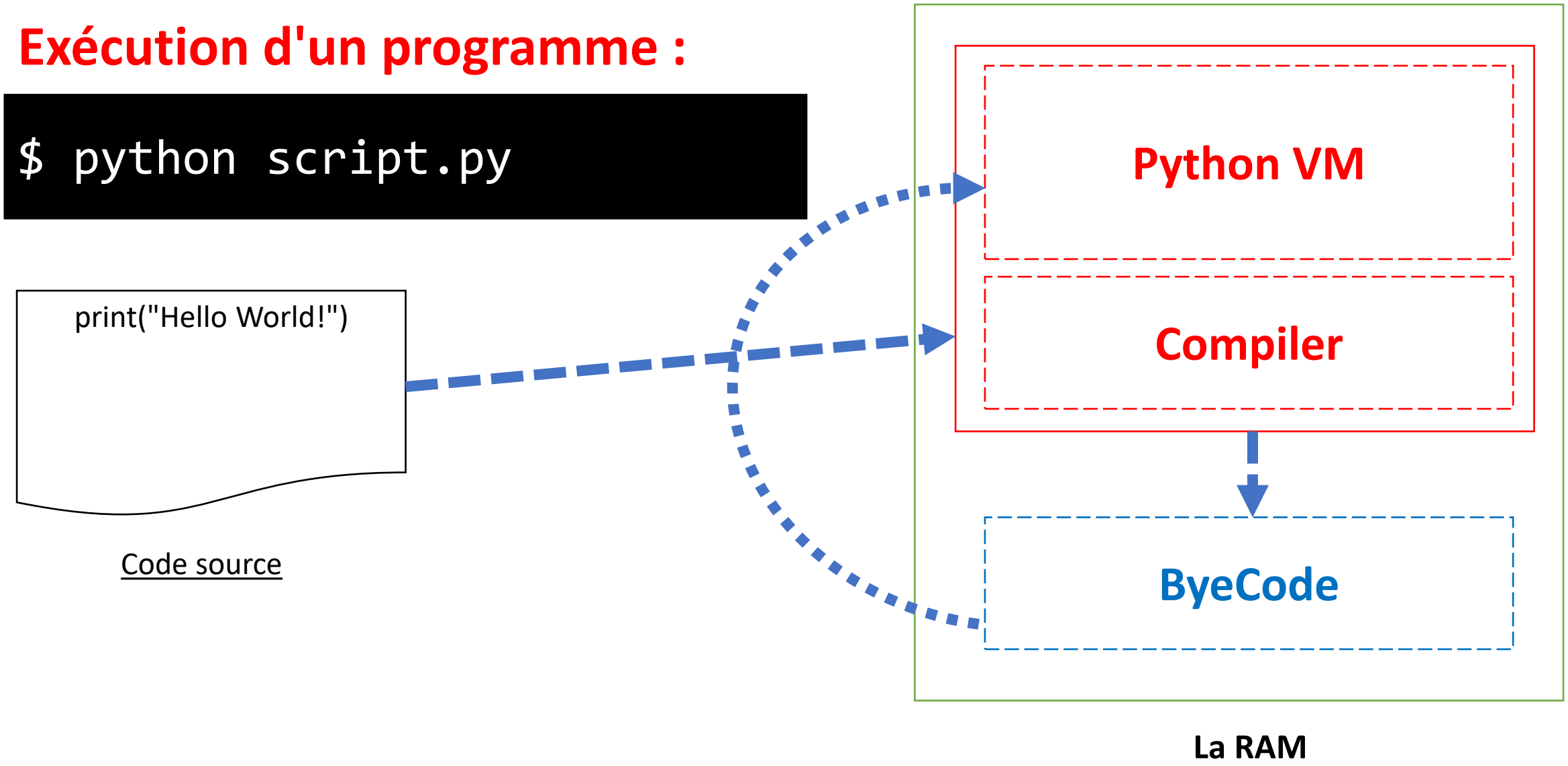
```
$ jpython script.py
```

```
$ ipy script.py
```

Plusieurs implémentation de Python

Exécution d'un programme :

```
$ python script.py
```



Installation de Python

Installation sur Windows:

Téléchargement : Rendez-vous sur le site officiel de Python www.python.org et téléchargez la dernière version stable.

Exécution de l'Installateur : Lancez l'installateur. Assurez-vous de cocher la case "Add Python to PATH" avant de cliquer sur "Install Now".

Vérification de l'Installation : Ouvrez l'invite de commande et tapez python --version ou python -V pour vérifier que Python est installé correctement.

Installation de Python

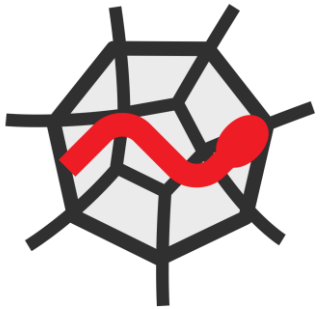
Installation sur Windows: Installation via Anaconda (optionnel)

Téléchargement d'Anaconda : Anaconda est une distribution Python qui inclut de nombreuses bibliothèques scientifiques et outils de gestion d'environnement. Téléchargez-le depuis www.anaconda.com .

Installation : Suivez les instructions spécifiques à votre système d'exploitation.

Vérification de l'Installation : Ouvrez Anaconda Prompt ou le terminal et tapez conda --version pour vérifier que Anaconda est installé.

Installation de Python



Jupyter:

- **Idéal pour la data science** : Jupyter est un environnement interactif parfait pour la visualisation et l'analyse de données, très utilisé en data science, machine learning, et dans le monde académique.

- **Exécution cellulaire** : Il permet d'exécuter des cellules de code indépendamment les unes des autres, ce qui est pratique pour tester des blocs de code sans exécuter l'ensemble du programme.

- **Visualisation directe** : Intègre des bibliothèques comme Matplotlib et Seaborn pour des graphiques interactifs et des analyses en temps réel.

- **Collaboration** : Possible de partager votre document en ligne.

Mode d'Emploi de l'Interpréteur Python :

Exécution de Scripts :

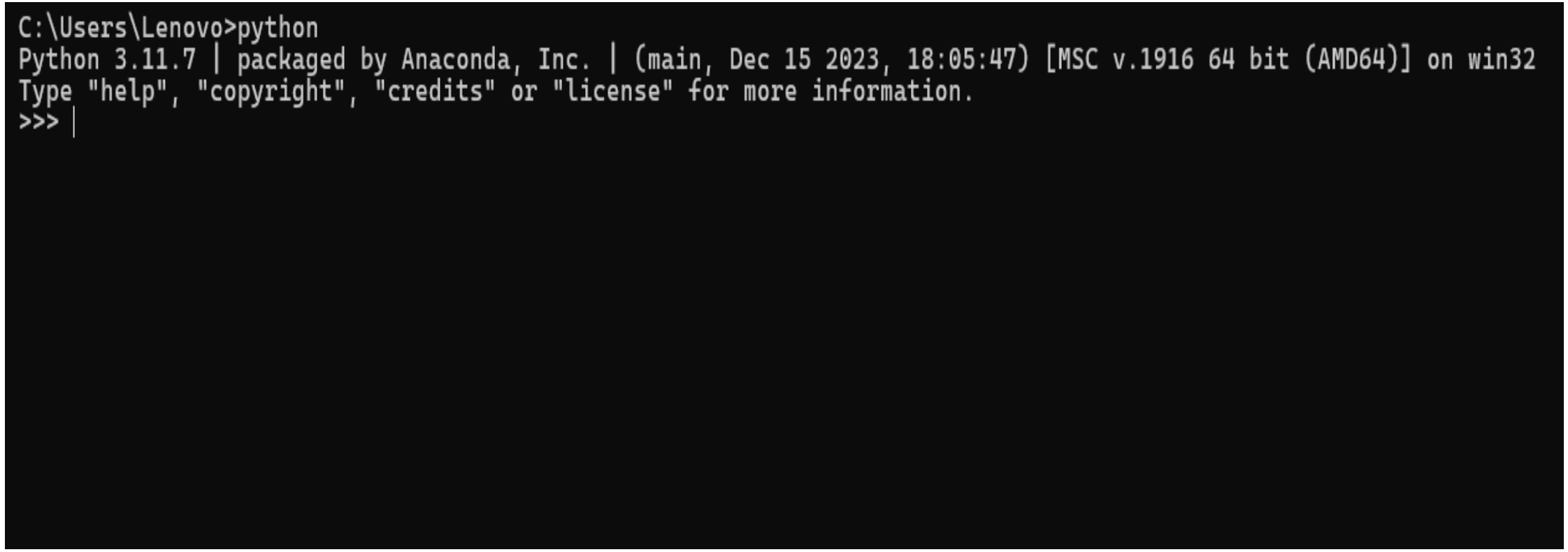
```
python script.py
```

Mode interactif:

En lançant simplement **python** ou **python3**, vous entrez dans le mode interactif. Cela vous permet d'exécuter des commandes Python ligne par ligne.

Mode d'Emploi de l'Interpréteur Python :

Mode interactif:



Syntaxe de base

Python est un langage de programmation interprété. C'est un «vrai» langage c.-à-d. types de données, branchements conditionnels, boucles, organisation du code en procédures et fonctions, objets et classes, découpage en modules.

Très bien structuré, facile à appréhender, c'est un langage privilégié pour l'enseignement.

Syntaxe de base

Les types de base :

int : Représente les nombres entiers, positifs ou négatifs, sans décimale.
Exemples : 5, -10, 0
Taille à partir de : 28 octets

float : Représente les nombres à virgule flottante, c'est-à-dire les nombres décimaux.
Exemples : 3.14, -0.001, 2.0
Taille à partir de : 24 octets

Syntaxe de base

Les types de base :

bool : Représente les valeurs de vérité, soit True soit False.
Exemples : True, False
Taille à partir de : 28 octets

complex : Représente les nombres complexes, qui ont une partie réelle et une partie imaginaire.
Exemples : 1 + 2j, 3 - 4j
Taille à partir de : 32 octets

Syntaxe de base

Les structure de données:

str : Représente une chaîne de caractères, utilisée pour stocker du texte.

list : Représente une collection ordonnée et modifiable d'éléments, qui peut contenir des éléments de différents types.

tuple : Similaire à une liste, mais immuable (les éléments ne peuvent pas être modifiés après la création).

range : Représente une séquence d'entiers, souvent utilisée dans les boucles.

dict : Représente une collection non ordonnée de paires clé-valeur. Les clés doivent être uniques et immuables, mais les valeurs peuvent être de n'importe quel type.

set : Représente une collection non ordonnée d'éléments uniques, sans doublons.

Syntaxe de base – Instructions de base

Affectation : Typage dynamique

Déclaration par affectation :

```
>> a = 1.2
```

a est une variable, en interne elle a été automatiquement typée en flottant «float» parce qu'il y a un point décimal. a est l'identifiant de la variable (attention à ne pas utiliser le mots réservés comme identifiant), = est l'opérateur d'affectation

Après un calcul :

```
>> a = 1.2
>> d = a + 3
```

d sera un réel contenant la valeur 4.2

Supprimer un objet de la mémoire

➢ **`del`** `nom_de_variable`

où nom_de_variable représente le nom de l'objet à supprimer.

Concepts de base

Transtypage :

En Python, le **transtypage** (ou "conversion de types") désigne le processus de conversion d'une valeur d'un type de données à un autre. Il existe deux types :

1 - Transtypage implicite :

Python effectue automatiquement certaines conversions de type lorsqu'il est nécessaire de combiner différents types de données.

```
x = 10    # int
y = 3.5   # float
z = x + y # Python convertit 'x' en float avant l'addition
print(z)  # 13.5
```

Concepts de base

Transtypage :

En Python, le **transtypage** (ou "conversion de types") désigne le processus de conversion d'une valeur d'un type de données à un autre. Il exite deux types :

2 - Transtypage explicite :

Dans d'autres situations, il est nécessaire de convertir manuellement un type de données en un autre à l'aide de fonctions de conversion.

Principe Utilisation du mot-clé désignant le type

>> nouveau_objet = **nouveau_type** (objet)

Concepts de base

Transtypage :

Conversion en numérique

```
a = "12 " # a est de type chaîne caractère
b = int(a) #b est de type int
c = float(a) #c est de type float
```

N.B. Si la conversion n'est pas possible ex. float(« toto »), Python renvoie une erreur

Conversion en logique

```
a = bool("TRUE") # a est de type bool est contient la valeur True
a = bool(1) # renvoie True également
```

Conversion en chaîne de caractères

```
a = str(15) # a est de type chaîne et contient « 15 »
```

Concepts de base

Enchaînement des instructions

La plus couramment utilisé

1 instruction = 1 ligne

```
#même valeur pour plusieurs variables
a = 1
b = 5
d = a + b
```

Autres possibilités

```
a = 1;b = 5 ;d = a + b;
```

Peu utilisé

```
a = 1;
b = 5;
d = a + b;
```

Concepts de base

La gestion de la mémoire :

les variables sont considérées comme des abstractions de haut niveau. Cela signifie qu'une variable en Python n'est pas directement liée à un emplacement mémoire fixe (comme dans les langages de bas niveau tels que C ou C++), mais qu'elle sert plutôt de référence à un objet qui est géré par l'interpréteur Python.

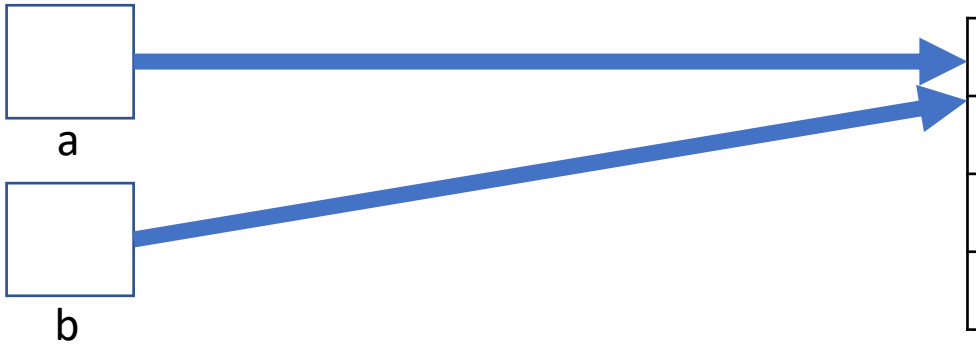
```
>>> a = 500
```

PyObject	
Type	int
Reference count	1
Value	500

Concepts de base

La gestion de la mémoire : Le compteur de références

```
>>> a = 500
>>> b = 500
```



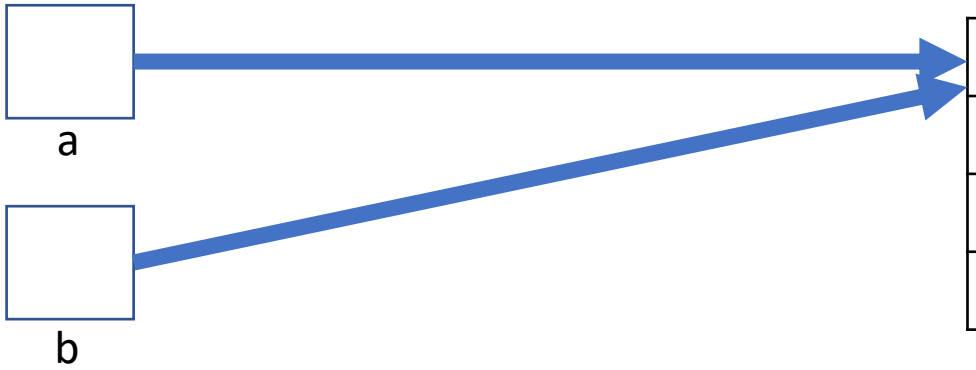
PyObject	
Type	int
Reference count	2
Value	500

```
>>> hex(id(a))
0x012493ff
>>> hex(id(b))
0x012493ff
>>> import sys;  sys.getrefcount(500)
2
```

Concepts de base

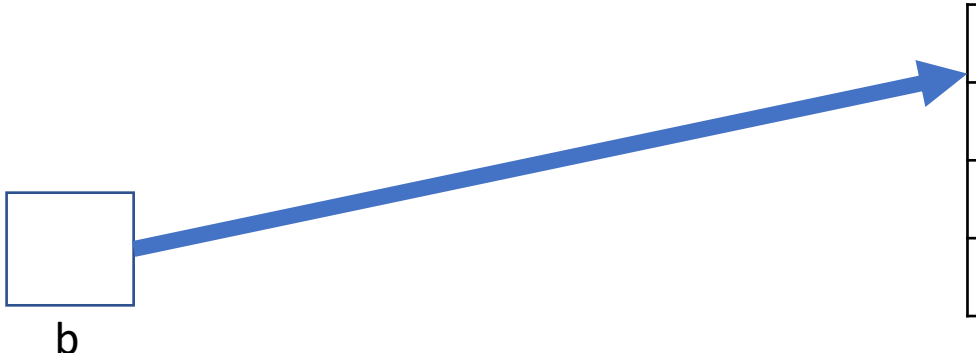
La gestion de la mémoire : Le compteur de références

```
>>> a = 500
>>> b = 500
```



PyObject	
Type	int
Reference count	2
Value	500

```
>>> del a
```

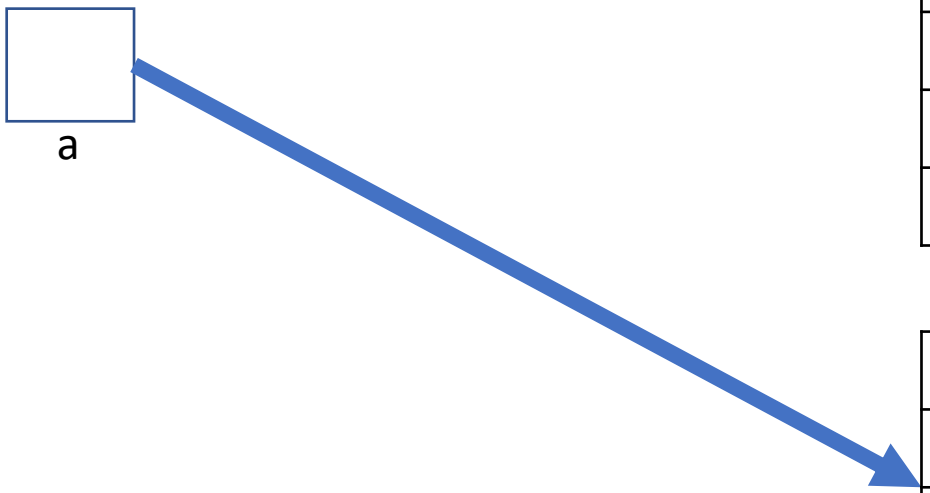


PyObject	
Type	int
Reference count	1
Value	500

Concepts de base

La gestion de la mémoire :

```
>>> a = 500
>>> a = 600
```



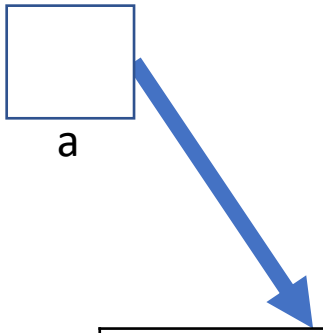
PyObject	
Type	int
Reference count	0
Value	500

PyObject	
Type	int
Reference count	1
Value	600

Concepts de base

La gestion de la mémoire :

```
>>> a = 500
>>> a = 600
>>> a = "Hello"
```



PyObject	
Type	int
Reference count	0
Value	500

PyObject	
Type	str
Reference count	1
Value	Hello

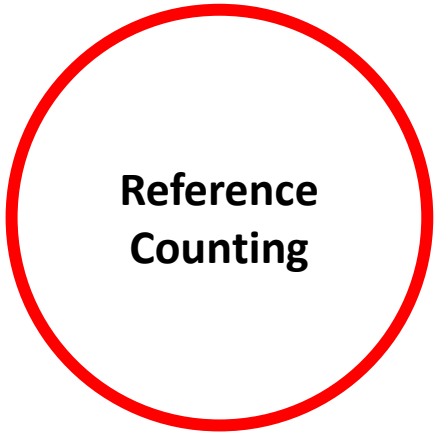
PyObject	
Type	int
Reference count	0
Value	600

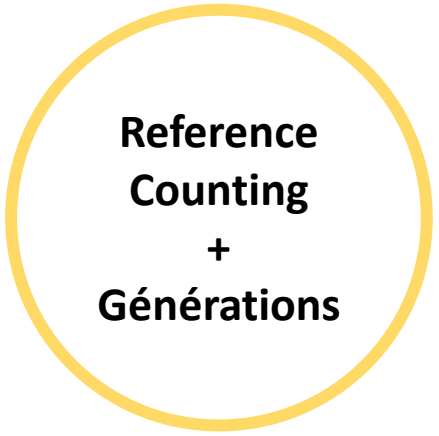
Concepts de base

La gestion de la mémoire : Garbage Collector

En Python, le garbage collection (GC) est le processus de gestion automatique de la mémoire en identifiant et en libérant la mémoire qui n'est plus utilisée, c'est-à-dire les objets qui ne sont plus référencés nulle part dans le programme. Python utilise une combinaison de comptage de références et de garbage collection cyclique pour gérer la mémoire.

Trois stratégies principales :





Concepts de base

La gestion de la mémoire : Interning pour l'optimisations

Pour optimiser l'accès et la création de certains objets, Python interne les valeurs fréquemment utilisés. Pour le cas des nombre entier, par défaut, tous les entiers de -5 à 256 sont internés.
Cela signifie que ces valeurs sont stockées en mémoire à un seul endroit, et lorsque vous créez plusieurs variables avec la même valeur dans cette plage, elles pointeront vers le même objet en mémoire.

```
>>> sys.getrefcount(5)
>>> 1000000059
>>> sys.getrefcount(5000)
>>> 2
```

Concepts de base

La gestion de la mémoire :

```
>>> a = "Hello Ahmed"
>>> b = "Hello Ahmed"
>>> id(a), id(b)
>>> (2498768485296, 2498768486000)
>>> a = "Hello_Ahmed"
>>> b = "Hello_Ahmed"
>>> id(a), id(b)
>>> (2498694187120, 2498694187120)
```

Remarque : Si ces instructions sont exécuté à partir d'un script, les chaînes identiques auront le même id.

Concepts de base

Opérateurs arithmétiques:

Ces opérateurs sont utilisés pour effectuer des opérations mathématiques.

Addition (+) : Additionne deux valeurs.
Soustraction (-) : Soustrait la seconde valeur de la première.
Multiplication (*) : Multiplie deux valeurs.
Division (/) : Divise la première valeur par la seconde, renvoie un flottant.
Division entière (//) : Divise la première valeur par la seconde et renvoie un entier (partie entière).
Modulo (%) : Renvoie le reste de la division de la première valeur par la seconde.
Exponentiation (**) : Élève la première valeur à la puissance de la seconde.

Concepts de base

Opérateurs de comparaison:

Ces opérateurs comparent deux valeurs et renvoient un booléen (True ou False).

Égal à (==) : Vérifie si les deux valeurs sont égales.
Différent de (!=) : Vérifie si les deux valeurs sont différentes.
Supérieur à (>) : Vérifie si la première valeur est supérieure à la seconde.
Inférieur à (<) : Vérifie si la première valeur est inférieure à la seconde.
Supérieur ou égal à (>=) : Vérifie si la première valeur est supérieure ou égale à la seconde.
Inférieur ou égal à (<=) : Vérifie si la première valeur est inférieure ou égale à la seconde.

Concepts de base

Opérateurs logiques:

Ces opérateurs sont utilisés pour combiner des expressions conditionnelles.

Et (and) : Renvoie True si les deux conditions sont vraies.
Ou (or) : Renvoie True si au moins une des conditions est vraie.
Non (not) : Inverse la valeur de vérité.

Concepts de base

Opérateurs d'identité:

Ces opérateurs comparent l'identité des objets, c'est-à-dire s'ils sont le même objet en mémoire.

Est (is) : Renvoie True si les deux variables pointent vers le même objet.
N'est pas (is not) : Renvoie True si les deux variables pointent vers des objets différents.

Concepts de base

Opérateurs d'appartenance:

Ces opérateurs testent l'appartenance d'une valeur à une séquence (comme une liste, un tuple, une chaîne, etc.).

in : Renvoie True si la valeur est trouvée dans la séquence.

not in : Renvoie True si la valeur n'est pas trouvée dans la séquence.

Concepts de base

Opérateurs bit à bit:

Ces opérateurs travaillent directement sur les bits des entiers.

Et bit à bit (&) : Effectue un ET logique sur chaque bit des deux nombres.
Ou bit à bit (|) : Effectue un OU logique sur chaque bit des deux nombres.
Ou exclusif bit à bit (^) : Effectue un OU EXCLUSIF logique sur chaque bit des deux nombres.
Non bit à bit (~) : Inverse tous les bits du nombre.
Décalage à gauche (<<) : Décale les bits du nombre vers la gauche.
Décalage à droite (>>) : Décale les bits du nombre vers la droite.

Entré / Sortie

Instruction de lecture :

```
# syntaxe de l'instruction de lecture
v = input(message)
```

input() permet d'effectuer une saisie au clavier
Il est possible d'afficher un message d'invite.
La fonction renvoie toujours une chaîne, il faut convertir

input() collecte toujours une chaîne de caractères.

Exemple:

```
v = input("Votre nom:")
```

Entré / Sortie

Instruction de lecture :

Pour récupérer un entier :

```
a = input("Votre age:")
a = int(a)
```

Pour récupérer un flottant :

```
b = input("Le prix de l’article:")
b = float(b)
```

Combiner les deux fonctions :

```
a = int(input("Votre age:"))
b = float(input("Le prix de l’article:"))
```

Entré / Sortie

Instruction de lecture :

La gestion des erreurs de saisie :

```
while True:
    try:
        age = int(input("How old are you? "))
    except ValueError:
        print("Please enter a number for your age.")
    else:
        break

print(f"Next year, you'll be {age + 1} years old")
```

Entré / Sortie

Instruction d'écriture:

```
# syntaxe de l’instruction de sortie
print(expression1, expression2, ...)
```

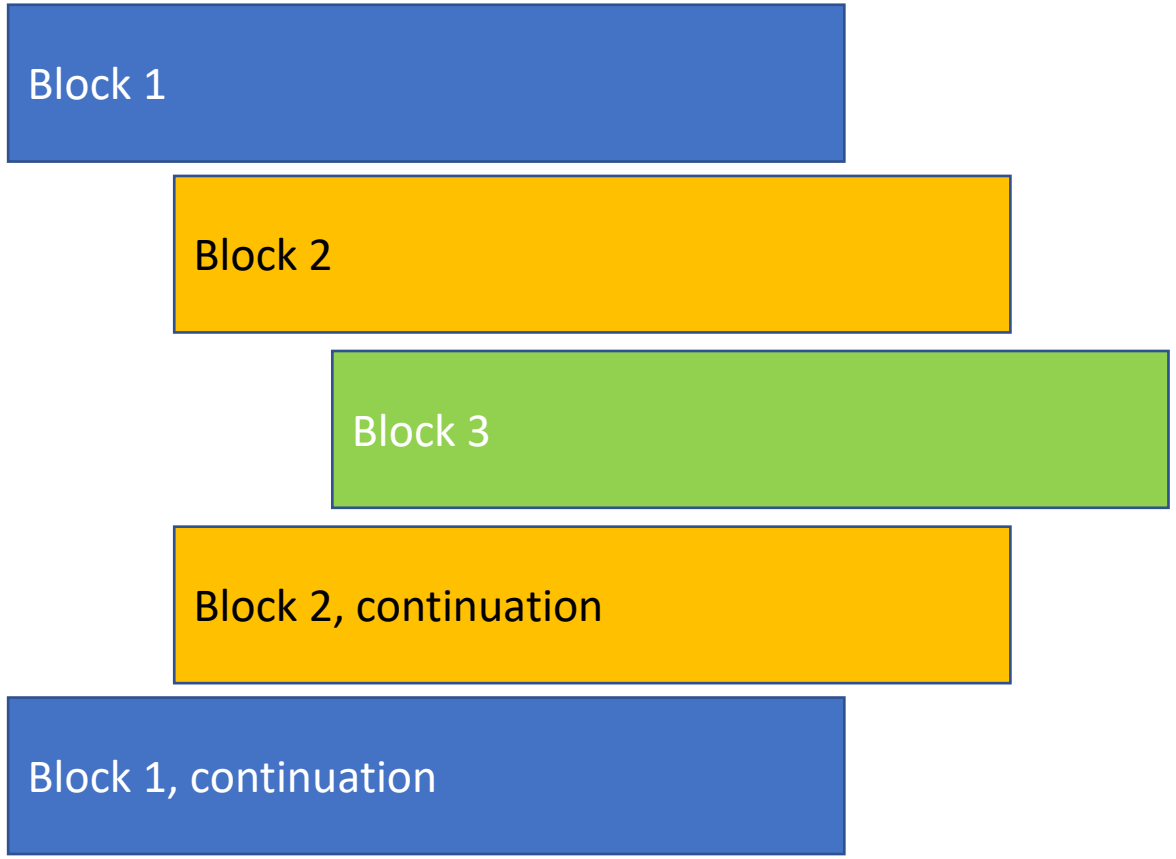
Exemple:

```
>>> a = 5
>>> print(a)
>>> print("La valeur est: ", a)
>>> print(f"La valeur est: {a}")
```

- L’affichage direct du contenu d’un tableau (liste) est possible également.

Concepts de base

La notion d'un bloque :



Indentation :

- Python utilise l'indentation (généralement 4 espaces) pour délimiter les blocs de code.
- Chaque ligne de code appartenant à un même bloc doit être indentée au même niveau.

Concepts de base

Structures conditionnelles

Les structures conditionnelles permettent d'exécuter du code uniquement si certaines conditions sont remplies.

- La structure if
- La structure if … else ….
- La structure if… elif …else
- La structure match

Concepts de base

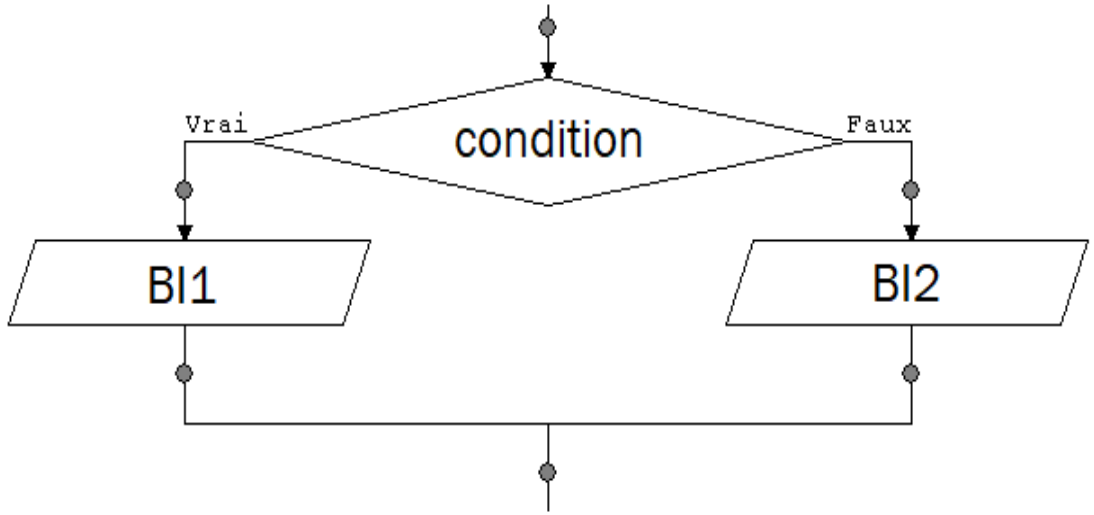
Structures conditionnelles

La structure if … else … :

Syntaxe :

```
if condition :
        BI 1
else :
        BI 2
```

Organigramme:



Concepts de base

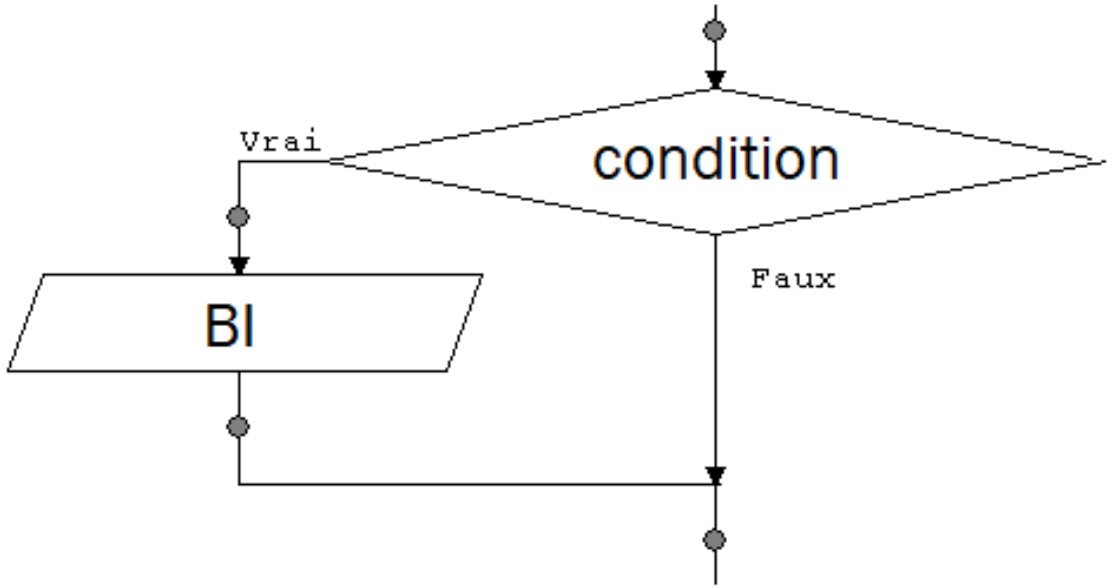
Structures conditionnelles

La structure if … :

Syntaxe :

```
if condition:
        BI
```

Organigramme:



Exemple :

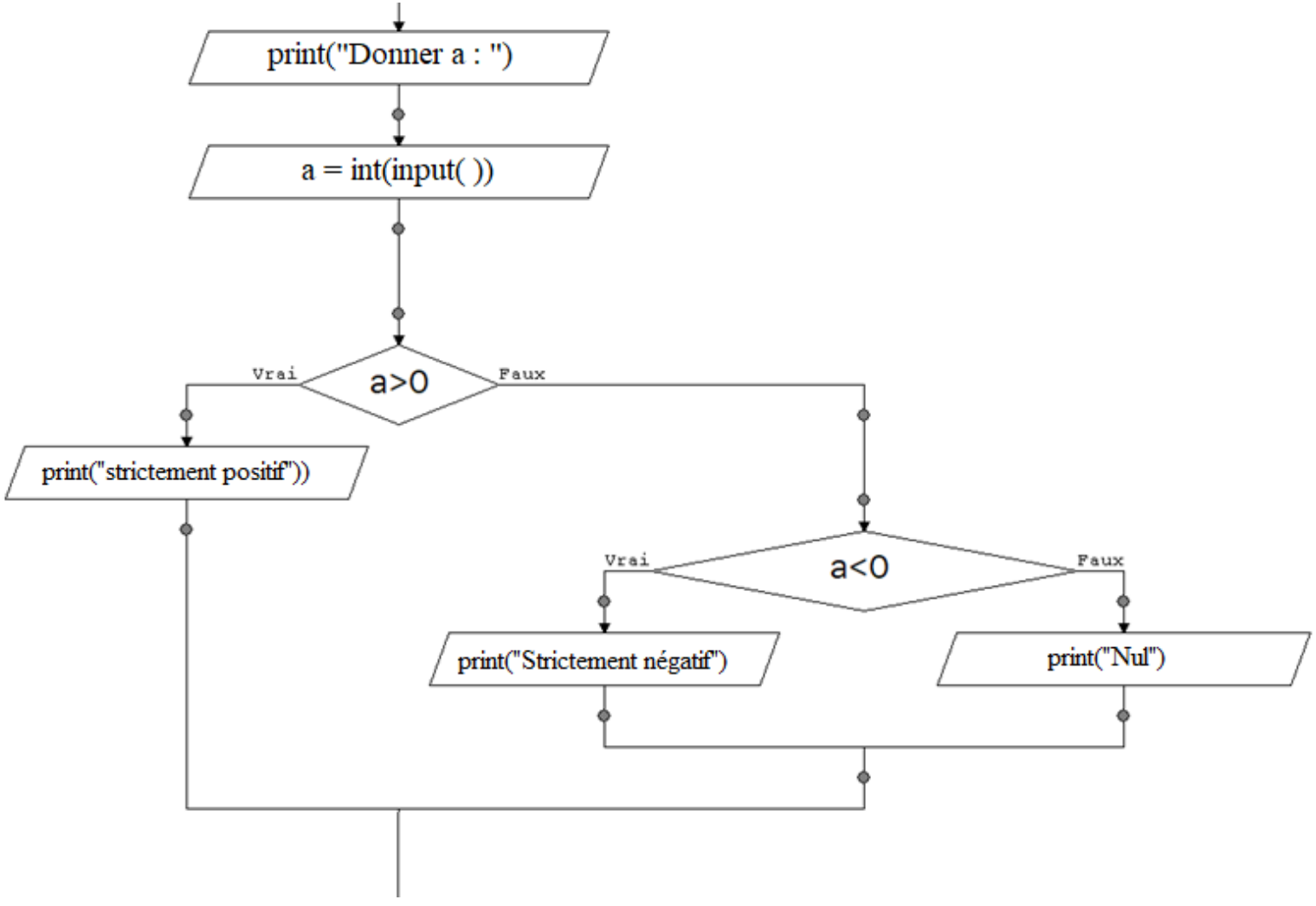
Ecrire un algorithme qui détermine le signe d'un nombre :

- Strictement positif
- Strictement négatif
- nul

Concepts de base

Structures conditionnelles imbriquées

Organigramme :



Structures conditionnelles imbriquées

Solution 1 :

```
print("Donner a : " )
A = int(input())
if a>0 :
      print("Strictement Positif")
else :
      if a<0 :
            print("Strictement Négatif")
      else :
            print("Nul")
```

Structures conditionnelles imbriquées

La structure if … else … :

```
if condition 1 :
        TRAITEMENT 1
elif condition 2 :
        TRAITEMENT 2
elif condition 3:
        TRAITEMENT 3
...
else :
        TRAITEMENT PAR DEFAULT
```

Structures conditionnelles imbriquées

Solution 2 :

```
print("Donner a : " )
A = int(input())


if a>0 :
      print("Strictement Positif")
elif a<0 :
      print("Strictement Négatif")
else:
      print("Nul")
```

Structures conditionnelles imbriquées

La structure match:

```
match variable :
    case valeur1:
        traitement 1
    case valeur1:
        traitement 1
    …
    case _ :
        traitement par défaut
```

Points à retenir :

- match compare des motifs (patterns) et non seulement des valeurs.
- Chaque case peut inclure des motifs spécifiques comme des tuples, des listes, ou des dictionnaires.
- L'underscore (_) est utilisé pour capturer tous les cas non gérés explicitement.

Structures conditionnelles imbriquées

La structure match: Exemple

```
match valeur:
    case (1, x):
        print(f"Tuple avec 1 et {x}")
    case [1, 2, 3]:
        print("Liste exacte [1, 2, 3]")
    case {"clé": y}:
        print(f"Dictionnaire avec 'clé' ayant la valeur {y}")
    case _:
        print("Motif non reconnu")
```

Structures répétitives (Itératives ou Boucles)

Structure indéterministe :

- La boucle : while

Le nombre de répétitions dépend d'une condition

Structure déterministe :

- La boucle: for

Le nombre de répétitions est connu à l'avance

La boucle while

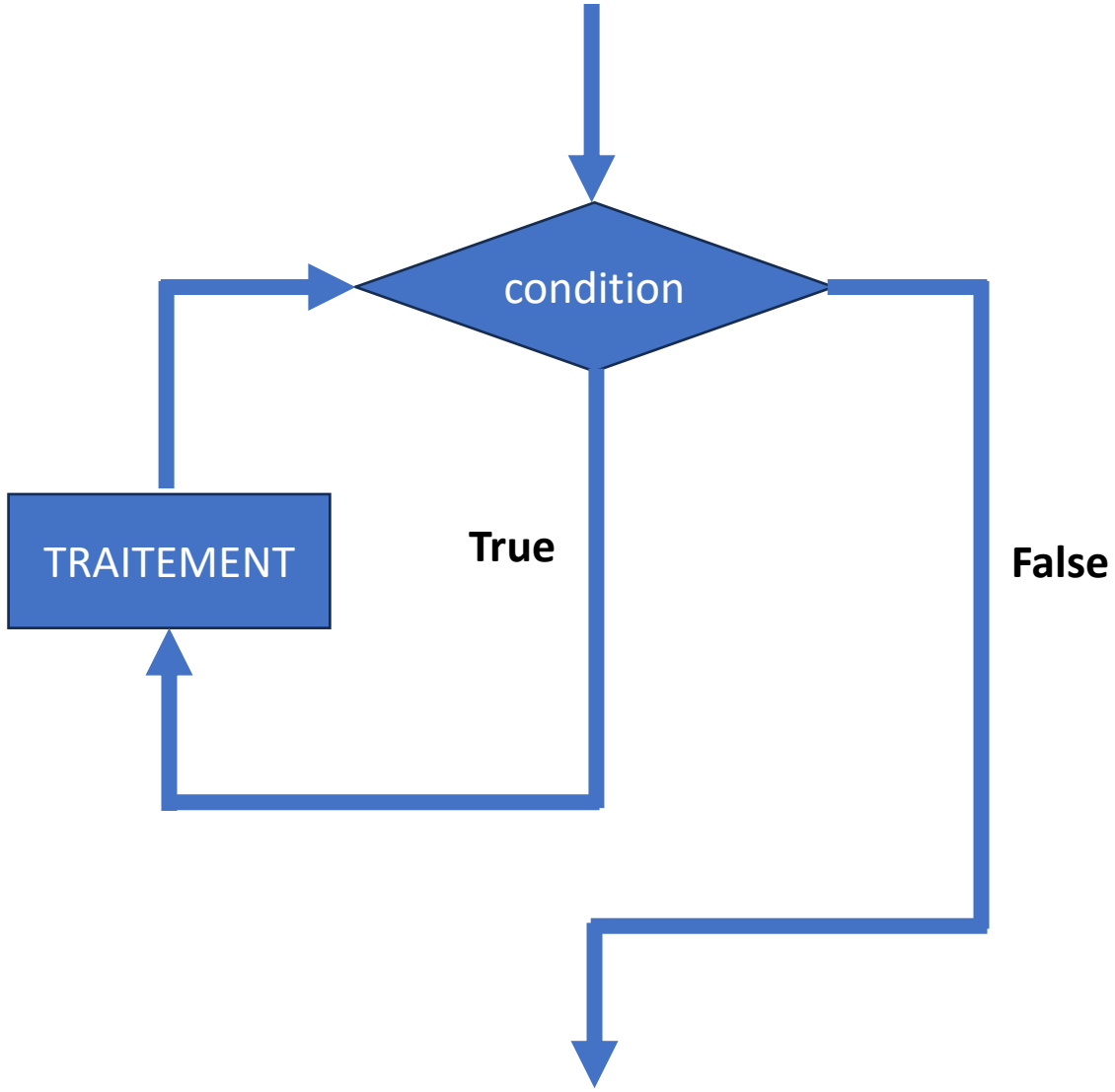
Définition :

La boucle while est une structure de contrôle en programmation qui permet d'exécuter un bloc de code répétitif tant qu'une condition donnée est vraie. Cela signifie que le bloc de code à l'intérieur de la boucle while est exécuté de manière répétée tant que la condition spécifiée reste vraie. Une fois que la condition devient fausse, l'exécution de la boucle s'arrête et le programme passe à l'instruction suivante après la boucle.

La boucle while

Syntaxe :

```
while condition :
    TRAITEMENT
```



La boucle while

Exemple 1 :

```
N=int(input("Donner le nombre de vos enfants : "))
# contrôler la validité des données d'entrés
while N<0:
        N = int(input("Erreur, donner une valeur >= 0 " ))

S = N*300
print("Le montant à verser est :", S)
```

La boucle while

Exemple 2 :

```
R= input("Voulez-vous un café ? O/N : ")
# contrôler la validité des données d'entrés
while R!='O' and R!='N' :
        R = input(« Réessayez .Voulez-vous un café ? O/N : " )

if R=='O' :
        print("Boisson servi !")
else :
        print("Aurevoir!")
```

La boucle while avec else

Syntaxe :

```
while condition :
        TRAITEMENT 1
else:
        TRAITEMENT 2
```

La boucle **while** en Python peut être combinée avec une clause **else** qui s'exécute lorsque la boucle se termine normalement (c'est-à-dire, sans interruption via un break).

La boucle while avec else

Exemple :

```
N = int(input("Un nombre:"))
numbers = [1, 2, 3, 4, 5]
i = 0
while i < len(numbers):
      if numbers[i] == N:
            print("Nombre trouvé.")
            break
      i += 1
else:
      print("Nombre non trouvé.")
```

La boucle for

Définition :

La boucle for est une structure de contrôle en programmation qui permet d'itérer une séquence de valeurs.

Les séquences en Python :

- Les chaînes de caractères
- Les listes
- Les tuples
- Les clés ou items ou valeurs des dictionnaires

La boucle for

Syntaxe :

```
for i  in séquence :
    TRAITEMENT
```

Fonctionnement :
Pour chaque élément i de la séquence exécuter le TRAIEMENT

La boucle for

Itérer une chaîne de caractères :

```
for i in "Maroc" :
    print(i)
```

```
M
a
r
o
c
```

La boucle for

Itérer les valeurs d'un intervalle [a, b]:

La fonction **range** permet de générer une séquence de nombre entiers :

La syntaxe est :

```
range( début,  fin,  pas)     # Le pas est optionnel et a la valeur 1 par défaut
```

La boucle for

Itérer les valeurs d'un intervalle [a, b]:

Exemples :

range(1, 6) va générer la séquence 1, 2, 3, 4, 5
range(1, 11) va générer la séquence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
range(-2, 3) va générer la séquence -2, -1, 0, 1, 2
range(1, 11, 2) va générer la séquence 1, 3, 5, 7, 9
range(1, 11, 3) va générer la séquence 1, 4, 7, 10
range(11) va générer la séquence 0,1, 2, 3, 4, 5, 6, 7, 8, 9, 10
range(11, 1) va générer la séquence VIDE
range(4, 1, -1) va générer la séquence 4, 3, 2
range(-4, -1) va générer -4, -3, -2

La boucle for

Exemple 2:

```
for  i in range(1, 5):
     print("Hello World")
```

```
Hello World
Hello World
Hello World
Hello World
```

La boucle for

Une boucle pour compter :

```
i= 1
while  i<=40 :
        TRAITEMENT
        i = i + 1
```

```
for  i in range(1, 41):
        TRAITEMENT
```

La boucle for avec else

Syntaxe :

```
for  i in sequence:
    TRAITEMENT 1
else:
    TRAITEMENT 2
```

La clause **else** s'exécute après la boucle for si la boucle se termine normalement, sans être interrompue par un break.

Boucles imbriquées

Définition:

```
for  i in séquence:
    Block Instructions
```

```
while  condition:
    Block Instructions
```

Si les instructions d'une boucle contiennent une autre boucle on parle d'une boucle imbriquée.

Boucles imbriquées

Exemples :

```
for  i in range(1, 20):
        for j in range(1, 30):
                Bloc Instructions

while condition:
        for  i in range(1, 20):
                Bloc Instructions
```

```
for  i in range(1, 20):
        while condition:
                Bloc Instructions

while condition1:
        while condition2:
                Bloc Instructions
```

Boucles imbriquées

Fonctionnement :

Pour chaque itération de la première boucle, la boucle imbriquée va s'exécuter jusqu'au bout.

```
for  i in range(1, 4):
        print("i=",i)
        for j in range(1, 3):
                print("   j=",j)
```

```
i=1
   j=1
   j=2
i=2
   j=1
   j=2
i=3
   j=1
   j=2
```

Boucles imbriquées

Exemple:

Afficher les couples de valeurs (x, y) tel que $x \in [1, 5]$ et $y \in [1, 3]$

```
for  x in range(1, 6) :
    for y in range(1, 4):
        print(x, y)
```

Les instructions break et continue :

Les instructions break et continue sont deux outils essentiels en Python pour contrôler le flux d'exécution à l'intérieur des boucles (for ou while). Elles permettent de modifier le comportement d'une boucle en fonction de conditions spécifiques.

Fonctionnement :

- L'instruction break est utilisée pour arrêter immédiatement une boucle.
- L'instruction continue permet de passer directement à l'itération suivante de la boucle, sautant le reste du code pour l'itération en cours.

Les instructions break et continue :

Exemple :

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

```
1
3
5
7
9
```

Boucles imbriquées

Exemple 2:

Comment sortir d'une boucle quand une condition est réalisée dans la boucle imbriquée.

```
for  x in sequence1 :
     for y in sequence2:
          if condition :
               break
```

Ça marche pas, le break utilisé dans la condition ne permet que de sortir de la deuxième boucle et donc continuer vers l'itération suivante de la boucle mère.

Boucles imbriquées

Exemple 2:

Comment sortir d'une boucle quand une condition est réalisée dans la boucle imbriquée.

Solution 1:

```
sortir = False
for  x in sequence1 :
    for y in sequence2:
        if condition :
            sortir = True
            break
    if sortir : break
```

Boucles imbriquées

Exemple 2:

Comment sortir d'une boucle quand une condition est réalisée dans la boucle imbriquée.

Solution 2:

```
for  x in sequence1 :
    for y in sequence2:
        if condition :
            break
    else:
        continue
    break
```