FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 5:
Introduction à Python

Licence : Physique Chimie

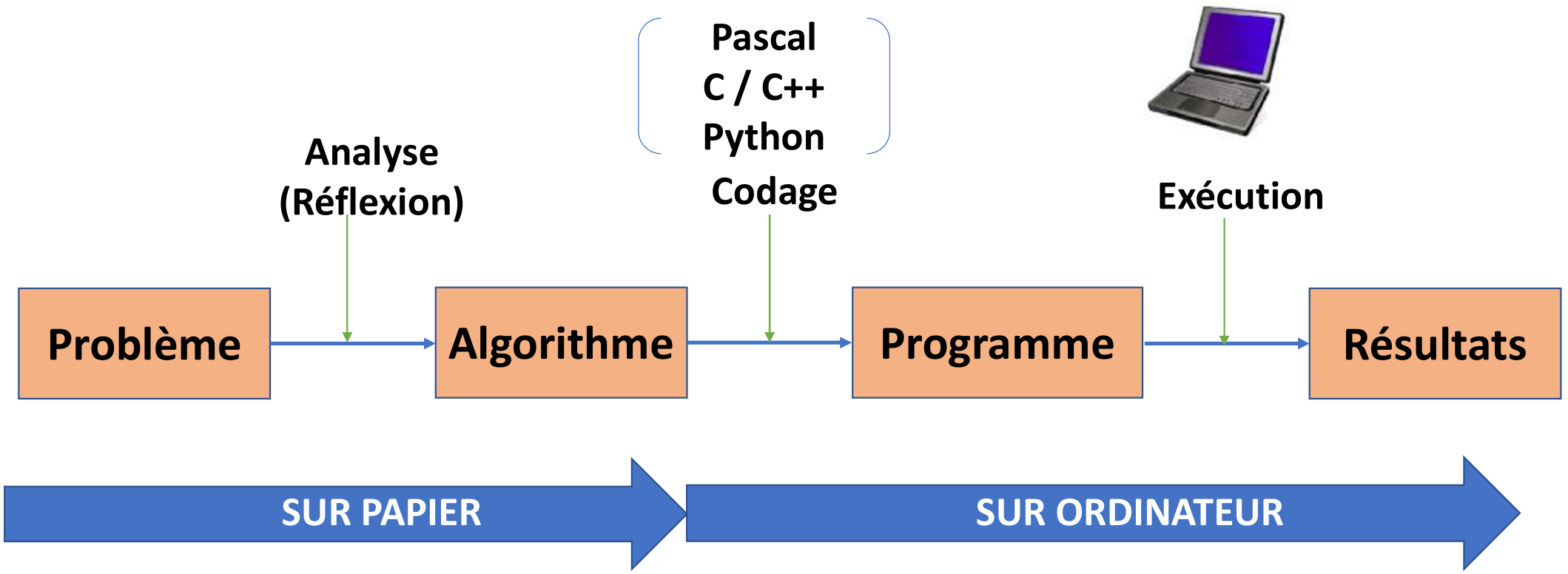
Pr: Youssef Ouassit

Plan Séance 1

1. **Introduction à Python**
    a. Historique
    b. Syntaxe de base
    c. Installation

Introduction à Python

Etapes de la programmation



Un peu d’histoire

Python est un langage de programmation **polyvalent** et **puissant**, largement utilisé dans divers domaines, notamment le développement web, l'analyse de données, l'intelligence artificielle, la science des données, l'automatisation, et bien plus encore.

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.

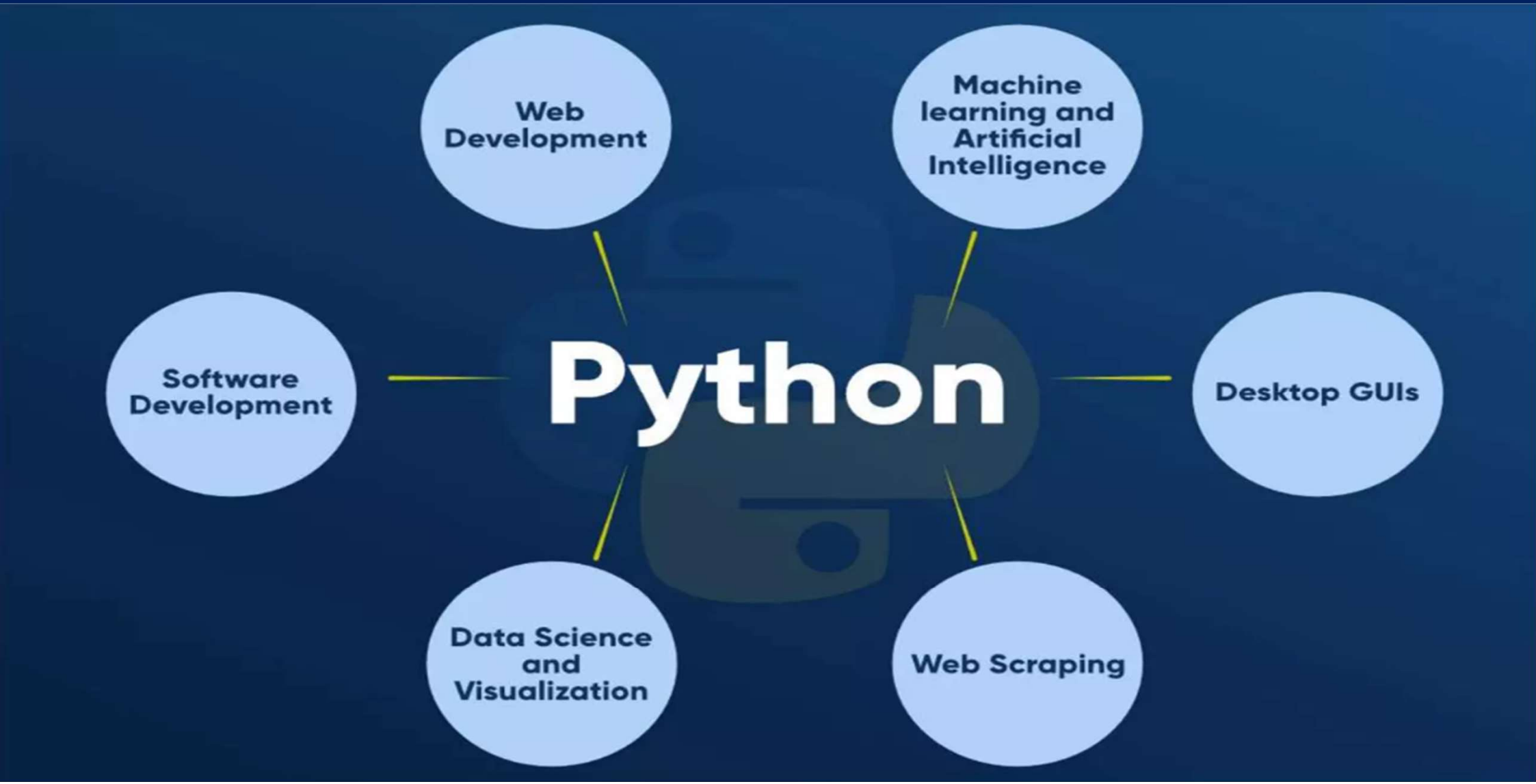
Un peu d’histoire

La version 3.x (2008 à présent)

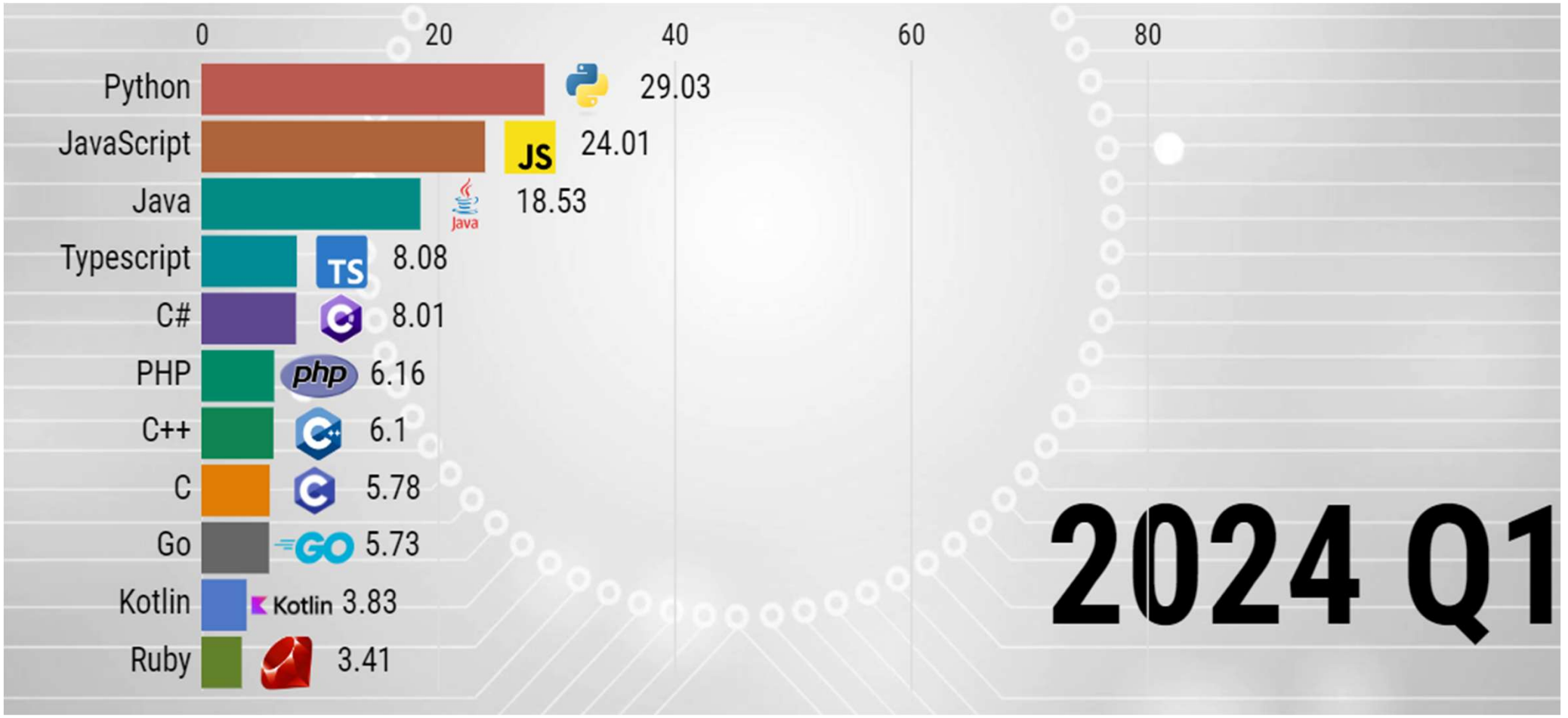
Python 3.0 a été publié en décembre 2008. Cette version a introduit des changements majeurs qui ont **rompu** la compatibilité avec Python 2.x, rendant la transition difficile pour certains projets.
L'objectif principal de Python 3 était de corriger les défauts fondamentaux du langage, en rendant la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

Domaines d’applications



Un peu d’histoire



statisticsanddata.org

Pourquoi Python ?

Python est :

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

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

Riche: il incorpore plusieurs possibilités de langage: Programmation Impérative, POO

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 ;

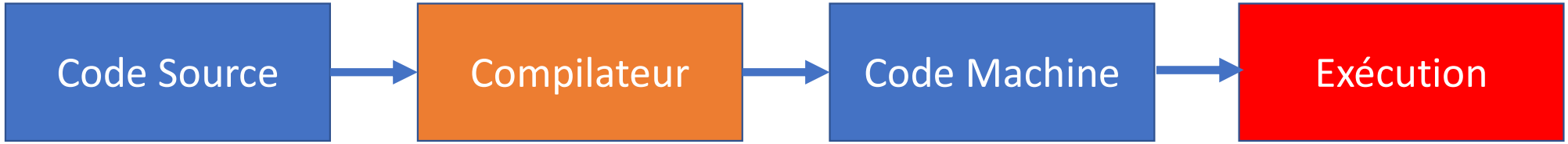
- **fortement typé** : les types sont toujours appliqués (un entier ne peut être considéré comme une chaîne sans conversion explicite, une variable possède un type lors de son affectation).

- **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

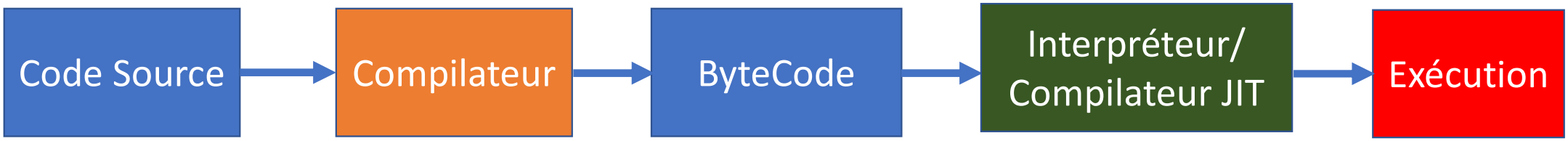
Un langage interprété:

- 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:

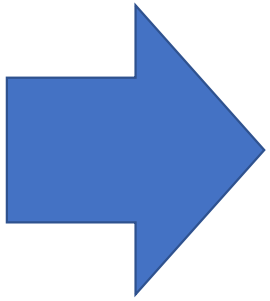


- Le code source est d'abord compilé en bytecode indépendant de la plateforme.
- Le bytecode est ensuite interprété ou compilé en code machine par la VM lors de l'exécution.
- Portabilité élevée car le bytecode peut être exécuté sur toute machine avec une VM.

Pourquoi Python, ses caractéristiques ?

Compilé vs Pseudo-Compilé vs interprété

Un langage Pseudo-Compilé:



Python est pseudo-compilé, il est irrémédiablement lent, mais… on peut lui associer des librairies intégrant des fonctions compilées qui, elles, sont très rapides.

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

Pseudo code

Affectation simple:

A ← 5

Affectation multiple :

A ← 5

B ← 5

Affectation parallèle :

A ← 4

B ← 5

Python

Affectation simple:

En Python : A = 5

Affectation multiple :

A = B = 5

Affectation parallèle :

A, B = 4, 5

Syntaxe de base – Instructions de base

Affectation – Typage automatique

```
➢ 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

Calcul

```
➢ a = 1.2
➢ d = a + 3
```

d sera un réel contenant la valeur 4.2

Syntaxe de base – Instructions de base

Enchainement des instructions

La plus couramment utilisé

1 instruction = 1 ligne

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

Autres possibilités

Peu utilisé

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

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

Syntaxe de base – Instructions de base

Pseudo code

Syntaxe:

Ecrire(expressions)

Exemples :

```
Ecrire("Bonjour")
A ← 5
Ecrire("La valeur est : ", A)
```

Python

Syntaxe:

print(expressions)

Exemples :

```
print("Bonjour")
A = 5
print("La valeur est : ", A)
```

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 chaîne de caractères

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

Syntaxe de base – Instructions de base

Pseudo code

Syntaxe:

Lire(variable)

Exemples (chaîne de caractères):

```
Ecrire("Votre nom : ")
Lire(nom)
```

Python

Syntaxe:

```
variable = input(message)
```

Exemples :

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

Syntaxe de base – Instructions de base

Pseudo code

Syntaxe:

Lire(variable)

Exemples (Entier):

```
Ecrire("Votre âge : ")
Lire(a)
```

Python

Syntaxe:

variable = input(message)

Exemples :

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

Ou :

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

Syntaxe de base – Instructions de base

Pseudo code

Syntaxe:

Lire(variable)

Exemples (Réer):

```
Ecrire("Votre salaire : ")
Lire(s)
```

Python

Syntaxe:

```
variable = input(message)
```

Exemples :

```
s = input("Votre salaire : ")
s = float(s)
```

Ou :

```
s = float(input("Votre salaire : "))
```

Syntaxe de base – Instructions de base

Pseudo code **Python**

En Algorithmique	En Python
Ecrire("Donner un entier : ") Lire(a)	a = int(input("Donner un entier : "))
Ecrire("Donner un réel : ") Lire(b)	b = float(input("Donner un réel : "))
Ecrire("Donner une chaîne : ") Lire(c)	c = input("Donner une chaîne : ")

Concepts de base

Opérateurs Arithmétiques:

Opération	Algorithmique	Python
Addition	+	+
Soustraction	-	-
Multiplication	*	*
Division réel	/	/
Division entière	div	//
Reste de la division	mod	%
Exposant	^	**

Concepts de base

Opérateurs de comparaison:

Opération	Algorithmique	Python
Supérieur	>	>
Inférieur	<	<
Supérieur ou égal	>=	>=
Inférieur ou égal	<=	<=
Différent	!=	!=
Egal	=	==

Concepts de base

Opérateurs logiques:

Opération	Algorithmique	Python
Et	Et	And
Ou	Ou	Or
Non	Non	Not

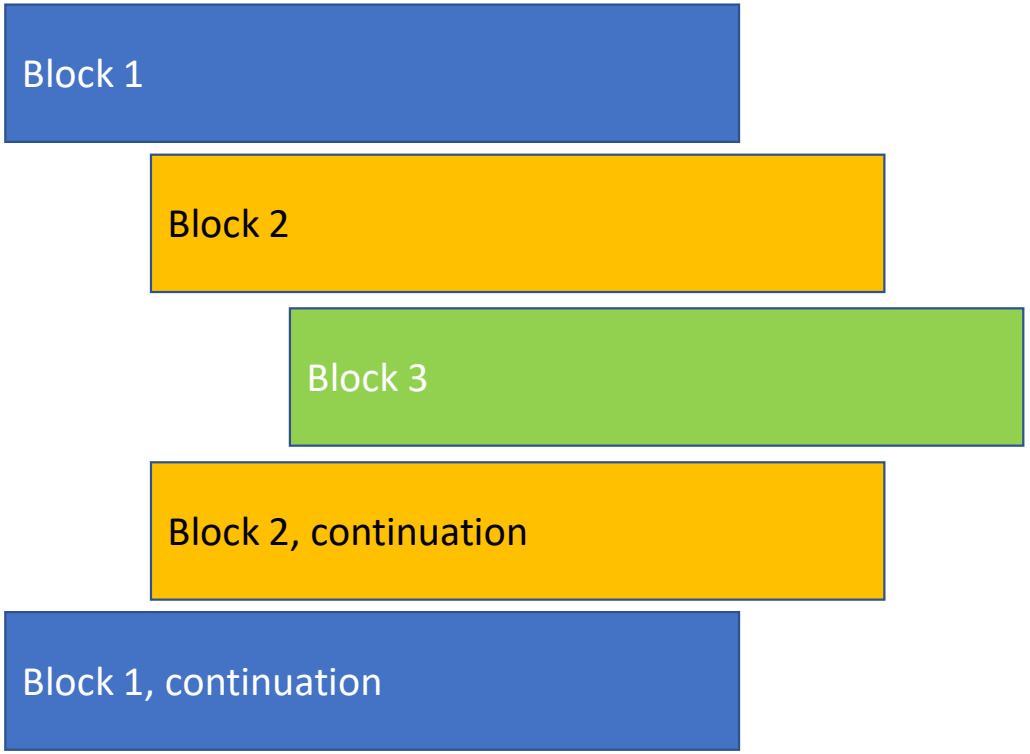
Concepts de base

La notion d’un block:

En Python, un bloc de code est un groupe de lignes de code qui sont exécutées ensemble en fonction d'une condition, d'une boucle ou d'une fonction. La notion de blocs en Python est étroitement liée à l'indentation, car c'est l'indentation qui délimite les blocs de code, contrairement à d'autres langages où des accolades {} ou des mots-clés spécifiques sont utilisés.

Concepts de base

La notion d'un block:



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

Concepts de base

Structures conditionnelles

Pseudo code

```
Si condition alors
        BI1
SiNon
        BI2
FinSi
```

Python

```
if condition :
        BI1
else :
        BI2
```

Concepts de base

Structures conditionnelles

Pseudo code

```
Si condition alors
        BI1
FinSi
```

Python

```
if condition :
        BI1
```

Concepts de base

Structures conditionnelles

Pseudo code

```
Si condition1 alors
        BI1
SiNon Si condition2 Alors
        BI2
SiNon Si condition3 Alors
        BI3
SiNon
        BI4
FinSi
```

Python

```
if condition1 :
        BI1
elif condition2:
        BI2
elif condition2:
        BI3
else:
        B4
```

Concepts de base

Structures conditionnelles

Exemple :

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

- Strictement positif
- Strictement négatif
- nul

Concepts de base

Structures conditionnelles

```
Algorithme Signe
Variables : N : Réel
Début
        Ecrire("Une nombre : ")
        Lire(N)
        Si N > 0 Alors
                Ecrire("Nombre strictement positif")
        SiNon  Si N < 0 Alors
                Ecrire("Nombre négatif")
        SiNon
                Ecrire("Nul")
        FinSi
        Ecrire("Aurevoir")
Fin
```

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.

Structures répétitives

La boucle while

Pseudo code

```
TantQue condition Faire
        TRAITMENT
FinTantQue
```

Python

```
while condition :
        TRAITEMENT
```

La boucle while

Exemple 1 :

```
N=int(input("Donner le nombre de vos enfants : "))

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 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 while

Pseudo code

```
Pour i←1 à 10 Faire
        TRAITMENT
FinPour
```

Python

```
for i in range(1,11) :
    TRAITEMENT
```

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

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

Syntaxe générale :

```
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
```

L'instruction break

L'instruction break permet d'interrompre l'exécution d'une boucle et de passer à la partie suivante du script.

Exemple : La boucle suivante s'arrête lorsque i atteint 5.

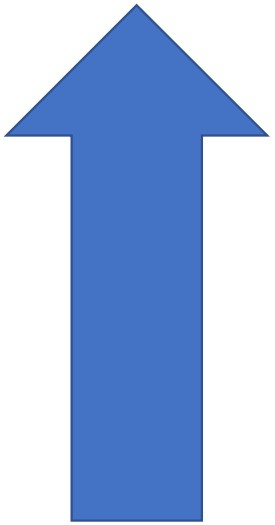
```
for i in range(10):
    if i == 5:
        print("Sortie de la boucle à i =", i)
        break
```

Priorités des opérateurs

La priorité des opérateurs en Python détermine l'ordre dans lequel les opérations sont effectuées dans une expression.

Les opérateurs
Les parenthèses : ( )
La puissance : **
Opérateurs unaire : -5 ou +6
Multiplication, division, modulo, division entière : * , / , % , //
Addition, soustraction : + , -
Opérateurs de comparaison : ==, !=, <, <=, >, >=
Opérateurs logique : and, or, not

Plus prioritaire



Priorités des opérateurs

Exemples :

(2 + 3) * 4	Donne : 20
2 + 3 ** 2	Donne : 11
10 + 2 * 3	Donne : 16
10 - 5 + 2	Donne : 7
5*3//3	Donne : 5
2 + 3 ** 2 * 4	Donne : 38
1 + 2 * (3 - 1) ** 2 // 5	Donne : 2
10 - 3 < 5	Donne : False
10 – (3 < 5)	Donne : 9
2 + 3 * 4 - 8 / 2**2	Donne : 12.0

Installation de Python – Méthode 1

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".

Installation de pyzo:

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

Exécution de l'Installateur : Lancez l'installateur.

Installation de Python – Méthode 2

Accéder à :

https://colab.research.google.com/

Cliquez sur :

Nouveau Notebook