Python en mode console

Présentation de Python

- Python est un langage **interprété**. C'est-à-dire que les instructions sont exécutées dès qu'elles sont lu.
- Il est **objet** et **impératif**. Il supporte aussi le paradigme **fonctionnel**.
- Il est **multiplateforme**.
- Son typage est **fort** et **dynamique**.
- Python offre des **outils de haut niveau** et une **syntaxe simple** à utiliser.
- Il n'y a pas de **point-virgule** à la fin des instructions en Python.
- Il n'y a pas **acrobate** `{ }` qui délimite un bloc d'instruction.

Comment exécuter Python ?

- Mode **REPL** : Dans ce mode, les instructions sont **lues**, **exécutées** et **affichées** directement après avoir été entrées.
  - Les commandes **ne sont pas** sauvegardées.
  - La mémoire disparaît **dès que l'on quitte** l'application.
  - Pratique pour **essayer rapidement** des commandes.
  - Pour l'utiliser :
    1. Ouvrez votre **terminal** (sous Windows, utiliser **l'invite de commande** ou **PowerShell**)
    2. Demandez d'exécuter le programme **python**.
       - Si vous avez la version 2 de python d'installer, utilisez le programme **python3**.
    3. Écrivez vos commandes. Elles seront exécutées dès que vous appuyez sur **entrée**.
    4. Pour quitter, utiliser la fonction `exit()`.

- Mode **Exécution** : Dans ce mode, les instructions sont lues à partir d'un fichier source. Les instructions sont exécutées après la lecture du fichier.
    - Les commandes **sont** sauvegardées.
    - L'extension d'un fichier source Python est `.py`.
    - Permets de créer un programme **réutilisable**.
    - Pour assurer la **compatibilité** avec la langue de Molière, tous les scripts Python **commencent** par la ligne `# -*- coding: utf8 -*-`.
    - Pour l'utiliser :
        1. Ouvrez votre **terminal** (sous Windows, utiliser **L'invite de commande** ou **PowerShell**)
        2. Demandez d'exécuter le programme **python nom_du_fichier.py**.

Les variables en Python

- Toutes les variables possèdent un **type** (entier, nombre flottant, booléen, objet, etc.) comme en Java.
- La déclaration d'une variable est simple : `nom de la variable = valeur initiale`.
    - Le code suivant déclare une variable **x** et elle lui donne la valeur 65

```
x = 65
```

- Contrairement à Java, le type d'un variable **n'est pas fixe**.
    - Il est donc possible de changer **dynamiquement** le type d'une variable.
    - À la fin de ce code, la variable **x** est de type **string** et sa valeur est le caractère **A**.

```
x = 65
x = "A"
```

- Python supporte les accents (é, è, etc.) dans les noms de variable.
    - Toutefois, ce **n'est pas recommandé** à cause de la compatibilité.

Les types de variables

- Il existe plusieurs types de variables en Python.
- **Nombre entier** : Représenté **par des chiffres** qui commence par un chiffre **autre que 0**.

```
nombre_entier = 42
```

- **Nombre décimal** ou flottant : Représenté **par des chiffres** et un **point**.

```
nombre_decimal = 42.0
nombre_decimal = 0.42
```

- **Chaîne de caractères** : Équivalent des *strings* en Java, l'ensemble des caractères est contenu entre **guillemets anglais** `" "`.
    - Python supporte les chaînes écrites entre **apostrophes** `' '`.
    - Pour avoir une chaîne de caractère multiligne, on utilise **trois guillemets anglais** l'une à suite de l'autre.

    - Afin d'avoir des caractères spéciaux (guillemet, saut de ligne, tabulation), nous utilisons les **mêmes symboles qu'en Java** (`\"`, `\n`, `\t`, etc.)

```
chaine_caracteres = "Ceci est une chaîne de caractère."
chaine_caracteres = 'Ceci est une autre chaîne.'
chaine_caracteres_multiligne = """Ceci est une chaîne,
sur plusieurs lignes.
Ha ! Ha ! Ha !"""
```

- **Booléens** : Détermine si une information est vraie ou fausse.
    - Pour une valeur **vraie**, Python utilise le mot clé `True`.
    - Pour une valeur **fausse**, Python utilise le mot clé `False`.
    - **ATTENTION**, les majuscules sont **obligatoires**.

```
valeur_booleen_vrai = True
valeur_booleen_fausse = False
```

- **Liste** : Toutes les listes sont contenues entre **crochets** `[ ]`.
    - Les listes sont l'équivalent des **tableaux dynamique** `ArrayList` en Java.
- Les listes sont toujours **dynamiques**.
    - Les listes peuvent contenir **plus d'un type**.
- Pour **accéder à un élément** de la liste, nous utilisons l'opérateur crochet `liste[i]`.
    - Avec `liste[2]`, le programme récupère la valeur de la liste à la position `2`.
    - Le premier item d'une liste commence à l'index `0`.
    - Pour **ajouter un élément** à la liste, nous appelons sa **méthode** `append()`.
        - Cette méthode prend, en **paramètre**, la **valeur ou la valeur d'une variable** à ajouter.

```
liste_entier = [1, 2, 3, 4]
liste_chaine = ["allo", "bye bye", "hello world"]
liste_plusieurs_type = [1, 'a', True]
liste_plusieurs_type.append(0.42)
```

- **Tuples** : Structure de donnée **immuable** dont le nombre d'éléments est fixé à sa **déclaration**.
    - Les tuples peuvent contenir **plus d'un type**.
    - Les éléments d'un tuple sont contenus entre **parenthèses** `( )`.

```
exemple_tuple = (5, 6, 7, 9)
```

- **Objet** : Comme en Java, un type objet correspond à une **instance d'une classe**.

Les commentaires en Python

- Tous les commentaires commencent par le caractère dièse `#`.

Fonction de base en Python

- `print(x)` affiche la valeur en paramètre dans la console

- `x = input()` récupère la valeur entrée par l'utilisateur et la retourne
    - `input(y)` affiche la valeur en paramètre et récupère la valeur entrée par l'utilisateur et la retourne
- `range(x, y, z)` renvoie un **tableau séquentiel** d'entier de la valeur `x` jusqu'à la valeur `y` **non incluse**. L'intervalle entre deux valeurs est `z`.
    - Par défaut, `x = 0`
    - `y` est requis en tout temps
    - Par défaut, `z = 1`.
    - Toutes les valeurs `x`, `y` et `z` **doivent être des entiers**.
- `len(x)` renvoie la longueur du tableau en paramètre.

Les opérations mathématiques, comparaison et logiques

Les opérations mathématiques courantes

- Python utilise les **mêmes symboles** pour les opérations mathématiques que le **langage Java**.

```
x = 2 + 3 # Addition de 2 et 3, la valeur est affectée à la variable x
y = 2 - 3 # Soustraction de 2 et 3, la valeur est affectée à la variable y
z = 2 * 3 # Multiplication de 2 et 3, la valeur est affectée à la variable z
w = 2 / 3 # Division de 2 et 3, la valeur est affectée à la variable w
alpha = 2 % 3 # Opération modulo de 2 et 3, la valeur est affectée à la variable
alpha
```

- Python supporte les **raccourcis d'opération** comme le langage Java.

```
x += 3 # Équivalent à x = x + 3
x -= 3 # Équivalent à x = x - 3
x *= 3 # Équivalent à x = x * 3
x /= 3 # Équivalent à x = x / 3
x %= 3 # Équivalent à x = x % 3
```

- Python **ne supporte pas** l'incrémentation et la décrémentation.

```
x++ # Génère une erreur car l'opérateur n'existe pas
x-- # Génère une erreur car l'opérateur n'existe pas
```

- Les fonctions mathématiques complexes ($sin(x), e^x, log_2(x), ...$) sont définis dans le **module** `math`.
    - Pour utiliser un module, il faut **l'importer** avec l'instruction `import nom_du_module`.
    - Cette instruction est placée en **début de fichier**.
    - Pour utiliser une méthode du module mathématique, on **appelle la fonction** de cette manière `math.fonction(...)`.

```
import math
x = 3
y = math.exp(x) # Fonction exponentielle de x avec x = 3
z = math.sin(x) # Fonction sinus de x avec x = 3
w = math.floor(x * 0.25) # Fonction plancher (entier inférieur) de x * 0.25
```

Les opérateurs de comparaison

- Une opération de comparaison **renvoie toujours** un booléen.
- L'opérateur compare deux valeurs (ou variables) ensemble.
- Les opérateurs de comparaison sont identiques qu'à ceux de Java.

```
>>> 1 == 1  # Comparaison d'égalité
True
>>> 1 != 1  # Comparaison de différence
False
>>> 1 > 2  # Comparaison d'ordre (plus grand que)
False
>>> 1 < 2 # Comparaison d'ordre (plus petit que)
True
>>> 1 >= 2 # Comparaison d'ordre (plus grand ou égale que)
False
>>> 1 <= 2 # Comparaison d'ordre (plus petit ou égale que)
True
```

- Si les deux valeurs **ne sont pas** du même type, la valeur sera **toujours** `False`.

Les opérateurs logiques

- Contrairement à Java, Python **utilise des mots clés** comme opérateur de comparaison.

```
x and y # Opération ET logique
x or y # Opération OU logique
not x # Opération NON logique
```

- Ces opérateurs fonctionnent uniquement sur des types **booléens**.

Les conditions et boucles

Les conditions

- Python supporte l'instruction `if..else` pour effectuer des opérations conditionnelles.
- La syntaxe du `if` est simple : `if condition :`
- La syntaxe du `else` est simple : `else :`
- Contrairement à Java, le bloc d'instruction du `if` (ou du `else`) n'est pas délimité par des acrobates `{ }`.
    - Il **commence** par le symbole double point `:` **suivi** d'un saut de ligne.
    - **Toutes les lignes** du bloc ont une indentation.

```
if x > 4 :
    # Suite d'opération si la condition x > 4 est vrai
else :
    # Suite d'opération si la condition x > 4 est fausse
```

- Python supporte l'instruction si sinon avec le mot clé `elif`
- La syntaxe du `elif` est simple : `elif condition :`
- Comme pour `if` et `else`, le bloc d'instruction commence par le symbole double-point et il est suivi par des lignes avec une indentation.

- Voici un exemple complet :

```
x = 2
if x == 1 :
    x += 5
    x -= 5
elif x >= 3 :
    x /= 5
    x *= 3
else :
    x = 42
```

- Contrairement à Java, l'instruction `switch-case` **n'existe pas** en Python.

Les boucles

- La boucle **tant que** utilise la syntaxe suivante :

```
while condition :
    # Suites d'instruction effectué jusqu'à ce que la condition soit fausse
```

- Comme pour `if` et `else`, le bloc d'instruction commence par le symbole double-point et il est suivi par des lignes avec une indentation.
    - La boucle exécute ce qui est dans son bloc.
- La condition doit être vrai pour que la boucle soit **exécutée au moins une fois**.
- La boucle **pour** sur une liste utilise la syntaxe suivante :

```
for element in liste :
    # Suites d'instruction effectué jusqu'à tous les éléments de la liste soit épuisée.
```

- Pour simuler une boucle `for` sur un intervalle d'entier, nous utilisons la méthode `range`.

```
for i in range(0, 6) :
    print(i)
    # La boucle sera exécuter 5 fois
```

Création et utilisation de fonction en Python

- Toutes les fonctions en Python sont définies avec le mot clé `def`.
- La syntaxe d'une fonction est simple :

```
def nom_de_la_fonction(param1, param2, param3, etc.) :
    # Suite d'instruction de la fonction
```

- Comme pour `if` et `else`, le bloc d'instruction commence par le symbole double-point et il est suivi par des lignes avec une indentation.
    - La fonction exécute ce qui est dans son bloc.
- Le mot clé `pass` permet d'avoir une fonction vide qui n'existe rien.

```python
def fait_rien() :
    pass # Python n'aime pas le vide
```

- Comme en Java, le nombre de paramètres varie entre 0 et beaucoup.
- Pour retourner une valeur, nous utilisons l'instruction `return`.

```python
def retourne_3() :
    return 3
```

- En python, il est possible de renvoyer plusieurs valeurs :

```python
def retourne_multi() :
    return 3, 5, 6

x, y, z = retourne_multi() # Il s'agit d'affectation multiple
print(x) # Affiche 3
print(y) # Affiche 5
print(z) # Affiche 6

w = retourne_multi() # Sera affecté comme un tuple
print(w) # Affiche (3, 5, 6)
```

Inclure des modules Python

- Les modules Python correspondent à d'**autres fichiers Python** que celui exécuté.
  - Le **nom du module** correspond au **nom du fichier**.
- Pour inclure un module Python, nous utilisons l'instruction `import nom_du_module`
  - Pour **appeler une fonction, ou une classe, du module**, nous devrons utiliser l'**instruction suivante** : `nom_du_module.fonction_appeler()`
  - Le nom du module devra toujours être présent.
- Une autre syntaxe possible est d'importer directement une fonction du module :

```python
from nom_du_module import fonction_appeler

fonction_appeler() # La fonction peut être utiliser directement sans avoir le nom du module
```

- Il est possible d'inclure **toutes** les fonctions d'un module avec l'instruction :

```python
from nom_du_module import *
```

- Le symbole `*` représente tout ce qui est contenu dans le module.

La génération de nombre aléatoire

- Les fonctions de génération aléatoire sont incluses dans le module `random`.
- La fonction `random()` renvoie un nombre décimal entre 0 et 1 non inclus.
- La fonction `randint(x, y)` envoie un nombre entier entre la valeur de x et la valeur de y.

Vérifier l'exécution d'un script par invocation direct

- En Java et en C, tous les programmes commencent par fonction `main()`.
    - En Python, le programme exécute toutes les instructions, même s'il n'y a pas de `main`.
- Afin d'avoir l'équivalent, il faut vérifier la valeur de la variable globale `__name__`.
    - Si le script est directement appelé, il est le script principal.
    - Si le script est le script principal, la valeur de `__name__` sera `__main__` à l'intérieur de celui-ci.
    - Dans le cas contraire, `__name__` prendra le nom du module.
- Essayez le code suivant pour vérifier :

```
# Python program to execute
# main directly
print ("Always executed")

if __name__ == "__main__":
    print ("Executed when invoked directly")
else:
    print ("Executed when imported with name : " + __name__)
```

Documenter son code

- La documentation Python utilise les **chaînes de caractère multiligne** `""" """`.
- La documentation est placée **immédiatement après** la déclaration de la fonction.
- Pour ce cours, nous utiliserons la syntaxe Sphinx.
    - La syntaxe de la documentation suit la forme suivante :

```
"""Description de la fonction

:param nom_du_parametre: Description du paramètre
:type nom_du_parametre: Type attendu du paramètre
:param autre_parametre: On ajoute tous les paramètres en suivant la même forme
:type autre_parametre: Idem
:return: Définit, s'il y a lieu, ce que la fonction envoie
:rtype: Type de la valeur retourné
"""
```

- Exemple de documentation pour la méthode `addition(x, y)` :

```
def addition(a, b = 3) :
    """Effectue l'addition de a et b

    :param a: Première à additionner
    :type a: Int
    :param b: (Optionnel) Deuxième valeur à addition, par défaut sa valeur est 3
    :type b: Int
    :return: a + b
    :rtype: Int
    """
    return a + b
```