Formation Python

PAR ALAIN CARIOU, NOVEMBRE 2023

I – Présentation de Python

Qu’est-ce que le Python ?

- Il s’agit d’un langage de programmation de haut niveau **multiparadigme** (orienté objet et impératif) créé en **1991**.

- Il est conçu pour être puissant, simple à apprendre et utilisable sur de très nombreuses plateformes.

- Il s’agit d’un langage **interprété**.

- Le nom du langage est une référence aux Monty Python !

L’utilisation du Python aujourd’hui

- Aujourd’hui le Python reste encore un langage très utilisé. Bien qu’il soit connu par ses fameux scripts, il est notamment utilisé dans les domaines suivants :
  - le **scripting**
  - la **création de sites web**
  - les **applications mobiles** (Android, IOS)
  - la **sécurité informatique**
  - les **bases de données**
  - les **systèmes embarqués** ...
- Parmi les entreprises bien connues utilisant Python on retrouve : Google et la NASA.

Les outils pour développer en Python

- Installer Python :
  - *https://www.python.org/downloads/*

- Utilisez votre IDE favori :
  - **PyCharm**, Emacs, Atom, Sublime Text, Visual Studio, Nano, …
  - Pour téléchargez Pycharm : *https://www.jetbrains.com/pycharm/download/#section=windows*

II – Développer en Python

Quelques bonnes pratiques

- **Indentez** votre code avec des indentations de 4 espaces et évitez les lignes dépassant 80 caractères. Par convention : on utilise le ***lower_snake_case***.

- Utilisez des noms de variables (*my_var*), fonctions (*my_func*), classes (*myClass*) **simples** et **claires**. Et évitez les **mots-clés** du Python !

- Soyez **explicites** dans le code que vous développez.

- Mieux vaut quelque chose de simple plutôt que complexe, de complexe plutôt que compliqué.

Un premier programme : Hello World

- Ouvrez l'interpréteur Python (***py*** ou ***python*** dans un terminal) et écrivez ceci :

- ***print("Hello World")***

- Félicitation vous avez écrit votre premier programme Python !

- On utilise la fonction native **print()** pour afficher un message.

L’interpréteur Python

- L’interpréteur nous permet d’exécuter le code Python.
- Il est particulièrement utile pour effectuer de petits scripts courts mais pour des programmes plus complexes, il sera rapidement limité.
- C’est pourquoi je vous propose de passer sur un IDE supportant le Python.

Les variables et les types de données

- Contrairement à d'autres langages, Python se distingue par un **typage dynamique fort**. C'est-à-dire que vous n'avez presque pas à vous préoccuper des types des variables.

- Néanmoins on va tout de même reconnaître plusieurs types de variables :
  - les **nombres entiers et décimaux** (*int* et *float*),
  - les **chaînes de caractères**, elles sont entre guillemets,
  - les **listes**,
  - les **booléans**.
- Vous devez toujours **initialiser** une variable avant de l'utiliser !

Les variables contenant des nombres

- Pour déclarer une variable contenant un nombre, il suffit d'effectuer l'assignation suivante :
  - ***mon_nombre = 42***
- Ici j'ai créé une variable s'appelant « *mon_nombre* » qui a pour valeur *42*.
- Il est possible d'effectuer des opérations arithmétiques avec les variables :
  - ***a = 5***
  - ***b = 7***
  - ***c = a + b***
  - ***print(c)***

L’affichage et la concaténation

- L’affichage se fait avec la méthode ***print()***.

- On peut ensuite concaténer des **chaînes de caractères** avec le **plus** « **+** » :
    - ***print(« Hello » + « world !»)***

- Ou, **peu importe le type de données**, avec la **virgule** « **,** » :
    - ***print(« Hello », « world ! »)***

Connaître le type d'une variable

- Vous pouvez connaître le type d'une variable avec la fonction ***type()***.

- Elle retourne le type de l'objet passé en paramètre.

- Par exemple : « ***<class 'int'>*** », « ***<class 'float'>*** », « ***<class 'bool'>*** », « ***<class 'str'>*** », etc.

Exercice - 1

- Créez un script contenant les variables suivante :
  - *num1 qui équivaut à 20*
  - *num2 qui est à égal à 3*
  - *num3 qui est le produit de num1 et num2*
  - *num4 qui est le résultat de la division de num1 par num2*
  - *result qui est la somme de num3 et num4 moins onze virgule cent onze.*
- Puis affichez la valeur de chacune de ces variables.
- Pensez-vous que vous pouvez afficher la valeur de result avec seulement deux chiffres après la virgule ?

Les chaînes de caractères

- On peut échapper des caractères grâce au caractère ****

- Les opérateurs **+** et ***** permettent de concaténer et répéter des chaînes de caractères :
  - ***my_str = « Mou » + 3 * « ha »***

- Les chaînes de caractères sont indexées : ***my_str[0]***
- L'index peut aussi être négatif pour avoir un décompte qui part de la droite : ***my_str[-9]***
- On peut récupérer juste une partie d'une chaîne avec : ***my_str[0:3]***
- Les commentaires se font avec le symbole **#**

Le formatage

- En Python, on peut formater les chaînes de caractères de différentes façons :

- Avec **l'opérateur %** :
  - ***print("Hello %s ! I'm %d years old." % (name, age))***
- Avec la méthode **format** :
  - ***print("Hello {} ! I'm {} years old !".format(name, age))***
  - ***print("Hello {0} ! I'm {1} years old !".format(name, age))***
- Avec les **f-Strings**, aussi appelées **interpolation de chaîne** :
  - ***print(f"Hello {name} ! I'm {age} years old !")***

Les listes - 1

- Les listes sont des types de données pouvant regrouper plusieurs valeurs qui ne sont pas toujours du même type :
  - ***my_list = [1, 2, 3, « ha », « hello »]***
- Elles peuvent être manipulées comme les chaînes de caractères.
- Les fonctions **append()** et **remove()** permettent d'ajouter et supprimer l'élément spécifié :
  - ***my_list.append(« orange »)***

Les listes - 2

- Les méthodes **insert()** et **pop()** permettent d'ajouter et supprimer un élément à l'index spécifié :
  - ***my_list.insert(1, « orange »)***
  - ***my_list.pop(1)***
  - **pop()** peut aussi s'utiliser sans index, dans ce cas il supprime le dernier élément.

- La méthode **clear()** permet de vider la liste.

Exercice - 2

- Créer un script qui contient une liste de noms et va trier cette liste par ordre alphabétique avant d'afficher le résultat.

- Quelle est la différence entre une **liste**, un **tuple**, un **set** et un **dictionary** ?

Les conditions

- Pour effectuer une condition on va utiliser le mot-clé **if** qui peut potentiellement être suivi d'autant de **elif** que besoin ou d'être fini par un **else** :

```
x = int(input("Please enter an integer: "))

if x == 0:
    x = 0
    print('Zéro')
elif x > 0:
    print('Supérieur')
elif x < 0:
    print('Inférieur !')
else:
    print('De zero en hero !')
```

Les opérateurs de comparaison

Symbole	Signification
==	Est égale à
!=	Est différent de
<	Est strictement inférieur à
<=	Est inférieur ou égale à
>	Est strictement supérieur à
>=	Est strictement supérieur ou égale à

Les opérateurs logiques

Symbole	Signification
and	Et
or	Ou
not	Non / contraire à

- On utilise les opérateurs logiques pour complexifier nos conditions afin d'avoir des résultats plus précis.

Les conditions : les ternaires - 1

- Il existe une dernière manière d'effectuer une condition que l'on appelle les **ternaires**. Elles furent rajouter dans la version **2.5** de Python.

- Une ternaire utilise la syntaxe suivante :
  - ***condition_is_true*** **if** ***condition*** **else** ***condition_is_false***

- Elles sont plus rapides mais aussi plus difficilement lisibles que les conditions classiques.

Les conditions : les ternaires - 2

- Voici un exemple de ternaire :

```
is_ok = True
state = "nice" if is_ok else "not nice"
print(state)
```

Les boucles - 1

- Les boucles sont des éléments permettant d'effectuer un certain nombre de fois la même action : c'est à dire d'exécuter un bloc de code en continu tant qu'une instruction est remplie.

- On distingue 3 types de boucle : le **while()**, le **for()** et le **range()**

- La boucle **while** permet d'effectuer un certain nombre de fois une instruction tant que la condition est remplie :

```
i = 1
while i < 5:
    print(i)
    i += 1
```

Les boucles - 2

- La boucle **for** permet d'itérer sur les éléments d'une séquence (liste, chaîne de caractères) dans l'ordre dans lesquels ces éléments apparaissent dans la séquence.

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

for i in tab:
    print(i)
```

- Si vous souhaitez modifier des éléments de la séquence sur laquelle s'effectue l'itération, il est conseillé d'en faire une copie. Sans quoi vous risquez d'avoir des erreurs …

Les boucles - 3

- Dans le cas où vous devez itérer sur une suite de nombre, l'instruction **range** permet de boucler sur des suites arithmétiques.

- Par exemple **range(5)** génère une liste de 5 valeurs allant de 0 à 4.

```
for i in range(5, 10):
    print(i)
```

Les boucles - 4

- L'instruction **break** permet de stopper une boucle.

- L'instruction **continue** fait passer la boucle à son itération suivante.

- L'instruction **else** est exécuter lorsque la boucle s'est arrêté normalement : ses itérations sont finies ou la condition est fausse dans le cas d'un **while**. Elle n'est pas exécuté si la boucle est stoppée par un **break**.

- L'instruction **pass** ne fait rien. Elle est utilisée lorsqu'une instruction est nécessaire pour fournir une syntaxe correcte, mais qu'aucune action ne doit être effectuée.

Lire sur l'entrée

- Pour l'instant nos programmes sont autonomes. Mais on souhaiterait parfois interagir avec l'utilisateur et lire les données qu'il nous envoie.

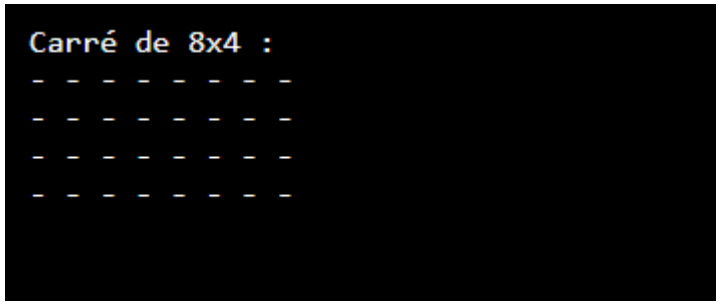
- Pour cela on peut utiliser la fonction **input()** qui permet de récupérer l'entrée d'un utilisateur :
  - **value = input()**
  - **value = input(« Ecrivez quelque chose : »)**

Exercice - 3

- Écrivez un programme en Python qui demande à l'utilisateur la longueur et la largeur d'un côté avant de calculer l'aire et le périmètre d'un quadrilatère.

- Puis la forme sera affichée dans le terminal si la longueur et la largeur sont supérieures à zéro.

- Exemple d'affichage attendu :



Les fonctions

- Pour l'instant nous créons simplement des scripts. Mais si l'on veut créer des programmes plus complexes, il va falloir utiliser des fonctions. Elles peuvent prendre plusieurs paramètres et avoir un valeur de retour :

```
def myconcat(str1, str2):
    print(str1 + str2)
    return str1 + str2

result = myconcat("Hello ", "World !")
print(result)
```

Attention une bonne fonction n'effectue qu'une seule action !

Arguments optionnels et arguments nommés

- Les arguments d'une fonction peuvent avoir des valeurs par défaut. Cela permet d'appeler la fonction avec moins d'arguments que nécessaire. On peut aussi nommer les arguments :

```
def display(str1, str2 = "Hello ", str3 = " there !"):
    print(str2 + str3 + str1)
    return str2 + str3 + str1

display("", "Hi ")
display(" General Kenobi !")
display(str1 = " You're a bold one !")
```

- Ainsi on peut appeler la fonction soit uniquement avec les arguments obligatoires, soit avec une partie des arguments optionnels, soit avec tous les arguments.

Arguments arbitraires

- Enfin lorsque l'on ne sait pas combien d'arguments peut avoir une fonction, on peut utiliser le mot-clé args qui correspond à une liste d'arguments arbitraires :

```
def display(my_str, *args):
    print(my_str)
    for i in args:
        print(i)
    return my_str

display(1, 2, 3, 4)
```

Les fonctions récursives

- Il est possible pour une fonction de s'appeler elle-même, c'est ce que l'on appelle une fonction récursive.

- La fonction continuera de s'appeler tant qu'elle n'a pas rencontré de condition d'arrêt.

Attention c'est très souvent une source d'erreur dans les programmes. Utilisez la récursion avec précaution !

Les modules

- A partir de là, vous allez peut-être aussi avoir plusieurs fichiers. En Python chaque fichier s'appelle un module. On peut importer un module en écrivant :
  - ***import my_module***
  - On appellerait les éléments du module de cette manière : **my_module.my_func**
- On peut aussi importer des éléments spécifiques du module ou bien le module entier :
  - ***from my_module import my_func***
  - ***from my_module import ****

Exercice

- **Exercice 4 :**
  - Reprenez l'exercice précédent et divisez-le en fonction :
    - une fonction pour calculer l'air,
    - une fonction pour calculer le périmètre,
    - et une fonction pour afficher la forme.

- **Exercice 5 :**
  - Créez une fonction qui compte le nombre de caractères d'une phrase de manière récursive.

La gestion des fichiers - 1

- Il est possible de manipuler les fichiers en Python.

- Pour ouvrir un fichier on utilisera la fonction **open**. Voici sa déclaration :
- **f = open(filename, mode)**

- **mode** correspond aux modes d'ouverture du fichier : **r, a, w, x**. Les modes a, w et x créent un fichier si celui-ci n'existe pas.
- On peut aussi l'ouvrir en mode texte : « **t** » ou en mode binaire « **b** ».

La gestion des fichiers - 2

- On peut lire le contenu d'un fichier de plusieurs manières :
  - via la méthode **read()** qui permet de lire l'entièreté d'un fichier ou un nombre n de caractères : **read(n)**
  - via la méthode **readline()** qui permet de lire le fichier ligne par ligne
  - via une boucle **for** :

```
f = open("file.txt", "r")
for x in f:
    print(x)
```

La gestion des fichiers - 3

- Pour écrire dans un fichier, il faut que celui-ci ait été écrit en mode **append « a »** ou **write « w »**
- Ensuite on utilise la fonction **write()** pour écrire à l'intérieur.

- Pour supprimer un fichier on importera le module **os** puis on utilisera sa méthode **remove()** :
  - **os.remove(« filename »)**
- Dans le cas d'un répertoire on utilisera la méthode **os.rmdir(« filename »)**

- Pour vérifier l'existence d'un fichier, on utilisera la méthode **os.path.exists(« filename »)**

La gestion des fichiers - 4

- Exemple d'utilisation d'un fichier :

```
import os

if os.path.exists("test.txt"):
    f = open("test.txt")
    result = f.read()
    f.close()
    print(result)
    os.remove("test.txt")
    print("File deleted")
else:
    f = open("test.txt", "w")
    f.write("On va manger des chips !")
    f.close()
    print("file closed")
```

La gestion des fichiers - 5

- On peut aussi gérer la manipulation des fichiers avec la syntaxe suivante :

```
with open('document.txt', 'w') as f:
    f.write(data)
```

- Cela permet à Python de gérer automatiquement la fermeture du fichier lorsque celui-ci n'est plus utilisée.

- **Privilégiez cette syntaxe qui vous évitera des erreurs ..**

La gestion d'erreur simplifiée

- On utilise le mot-clé **try** pour tester un bloc de code
- On utilise le mot-clé **except** pour gérer l'erreur, de manière spécifique ou non
- Le mot-clé **finally** exécute l'erreur, qu'importe le résultat du **try** et de **l'except**

```
try:
    print(x)
except NameError:
    print("Variable x is not defined")
except:
    print("Something else went wrong")
else:
    print("Everything is ok !")
finally:
    print("Continue ...")
```

Exercices

- **Exercice 6 :**
  - Réalisez un programme qui lira sur l'entrée standard ce que l'utilisateur écriera et stockera cela dans un fichier.

- **Exercice 7 :**
  - Réalisez un programme en Python qui récupérera le mot de passe présent dans le fichier « *index.html* » ci-joint et l'affichera.