

```
1
2
3 Programming 'python' {
4
5     [Scripting Workshop]
6
7
8     < Slides du cours >
9
10
11
12 }
13
14
```

Table Of 'Contents' {

01 Les bases du python

< Comprendre et maîtriser
la syntaxe du langage >

02 L'apprentissage des modules

< >

03 Réalisation de scripts complexe

< >

}

Qu'est-ce que 'python' ? {

[]

Langage cross
plateforme

Fonctionne sur
quasiment tous les
OS

[]

Langage haut
niveau

Demande peu de connaissance
hardware/os pour comprendre
son fonctionnement

[]

Langage
interprété

Pas besoin d'être
compilé

[]

Langage orienté
objet

On peut créer des
entités
représentatives
du monde réel

}

De quoi avez-vous besoin ? {

Les ressources essentielles dont vous aurez besoin

- * Python3

- <https://www.python.org/>

- * Un bon IDE

- exemple VSCode : <https://code.visualstudio.com/>

- * Des librairies et modules

- Pip3
 - Virtualenv
 - Pandas, numpy etc.

}

Let's install! {

Place au terminal

* Sur Linux

\$ python3 -V #si une version s'affiche, python est installé

\$ sudo apt update && sudo apt install

* Sur MacOS

→ brew install python3

* Sur Windows

- Aller sur le site de python.org et télécharger la dernière version

01

Les bases du Python

< Comprendre et maîtriser
la syntaxe du langage >

Premier { Code;

Réalisez un 'Hello World!', dans la console
Python, puis dans un script

```
>>> print("Hello World!")
```

}

Appel de l'interpréteur {

La console représente vite une limite, dès que l'on travail
à partir d'un fichier on parle de script

* Appel direct de l'interpréteur

```
$ python3 test.py
```

```
Hello World !
```

* Appel direct du script

1. Précisez au shell la localisation de l'interpréteur Python en
indiquant dans la première ligne du
script : (exemple pour linux) `#!/usr/bin/env python3`

2. Donner une permission d'exécution avec : `$ chmod +x test.py`

3. Puis exécuter avec : `$./test.py`

}

Commenter son code ! {

* Dans un script, tout ce qui suit le caractère # est ignoré par Python jusqu'à la fin de la ligne et est considéré comme un commentaire. Il est également possible d'écrire un gros block de commentaire avec

```
'''
```

```
Ceci est un commentaire !
```

```
'''
```

* Les commentaires sont indispensables pour que vous puissiez annoter votre code. Il faut absolument les utiliser pour décrire les fonctions ou parties principales. Votre code doit être rapidement compréhensible par d'autre !

```
}
```

L'indentation en Python {

- * En python, l'indentation est essentielle. En effet, une mauvaise indentation créera des bugs ou un mauvais comportement de votre code !
- * Pratiquement, l'indentation en Python doit être homogène (soit des espaces, soit des tabulations, mais pas un mélange des deux). Une indentation avec 4 espaces est le style d'indentation le plus traditionnel et celui qui permet une bonne lisibilité du code.

}

Exemple de code

```
1  import os
2  import sys
3  import subprocess
4
5  # Simple SSL/TLS python3 script vulnerabilities test
6
7  rc = subprocess.call(['which', 'testssl'], stdout=subprocess.PIPE)
8  if rc == 0:
9      print("[+] testssl.sh is installed on this machine")
10 else:
11     print("[-] testssl.sh is missing in path.")
12     sys.exit()
13
14 # sys.argv[1] is the first argument given after the python script
15 # Check if argument was give, if not, print usage
16
17 if len(sys.argv) == 1:
18     print("[*] Simple SSL/TLS vulnerabilities tests Script")
19     print("[*] Usage      : " + sys.argv[0] + " <IP addresses file path> ")
20     print("[*] Note: IP addresses file must use the following format @ip:port ")
21     sys.exit()
22
```

Les variables {

Une `variable` est une zone de la mémoire dans laquelle une `valeur` est stockée. Aux yeux du programmeur, cette variable est définie par un `nom`, alors que pour l'ordinateur, il s'agit en fait d'une adresse

- * En Python, la déclaration d'une variable et son initialisation (c'est-à-dire la première valeur que l'on va stocker dedans) se font en même temps.

```
>>> x = 7
>>> x
7
```

- * Que c'est il passé dans cet exemple ?

}

Les variables {

```
>>> x = 7
>>> x
7
```

- * Python a deviné que la variable était un entier. On dit que Python est un langage au typage dynamique.
- * Python a alloué l'espace en mémoire pour y accueillir un entier et a fait en sorte qu'on puisse retrouver la variable sous le nom x
- * Python a assigné la valeur 2 à la variable x.

}

Les variables {

- * Dans certains autres langages, il faut coder ces différentes étapes une par une (en C par exemple). Python étant un langage dit de haut niveau, la simple instruction `x = 7` a suffi à réaliser les 3 étapes en une fois !
- * Dernière chose, l'opérateur d'affectation `=` s'utilise dans un certain sens :
 - Si on a `x = y - 3`, l'opération `y - 3` est d'abord évaluée et ensuite le résultat de cette opération est affecté à la variable `x`.

}

Les différents types de variables

{

* Le type d'une variable correspond à sa nature. Les trois types principaux sont les entiers (integer ou int), les réels (float) et les chaînes de caractères (string ou str). Il en existe plein d'autres !

* Python reconnaît certains types de variable automatiquement. Par contre, pour une chaîne de caractères, il faut l'entourer de guillemets (simples, doubles etc.) pour indiquer à Python le début et la fin de la chaîne.

}

Les noms de variables {

- * Le nom des variable en Python peut-être constitué de lettres minuscules (a à z), de lettres majuscules (A à Z), de nombres (0 à 9) ou du caractère souligné (_).
- * Un nom de variable ne doit pas débuter ni par un chiffre, ni par _ et ne peut pas contenir de caractère accentué. Il faut absolument éviter d'utiliser un mot "réservé" par Python comme nom de variable (par exemple : print, range, for, from, etc.).
- * Python est sensible à la casse ! Donc les variables Test, test ou TEST sont différentes. Enfin, vous ne pouvez pas utiliser d'espace dans un nom de variable.

}

Les opérateurs {

- * +, -, *, **, /, et % sont des opérateurs. Ils permettent de faire des opérations sur les variables. (**) est l'opérateur de puissance. (%) le modulo renvoi le reste d'une division Euclidienne.
- * Pour les chaînes de caractères, deux opérations sont possibles, l'addition et la multiplication :
 - o L'opérateur d'addition + permet de concaténer (assembler) deux chaînes de caractères et l'opérateur de multiplication * permet de dupliquer plusieurs fois une chaîne.

}

Fonction type {

Pour obtenir le type d'une variable, utilisez la fonction type()

```
>>>x=2
>>> type(x)
<class 'int'>
```

Conversion de type

On est souvent amené à convertir les types avec les fonctions int(), float() et str().

```
>>> i=3
>>> str(i)
'3'
```

}

Fonction d'affichage {

Pour afficher du texte ou le contenu d'une variable utiliser la fonction print()

```
print('Résultat de x :', x)
```

Fonction d'entrée

Grâce à la fonction input(), on peut demander une entrée à l'utilisateur puis affecter la valeur à une variable par exemple :

```
print('Quel âge as-tu ?')  
x = input()  
print('Tu as {} ans !'.format(x))
```

Les listes {

* Une liste est une structure de données qui contient une série de valeurs. Python autorise la construction de liste contenant des valeurs de type différent (par exemple entier et chaîne de caractères), ce qui leur confère une grande flexibilité

```
>>> animaux = ['girafe', 'tigre', 'singe', 'souris']
>>> tailles = [5, 2.5, 1.75, 0.15]
>>> mixte = ['girafe', 5, 'souris', 0.15]
```

* Lorsque l'on affiche une liste, Python la restitue telle qu'elle a été saisie.

```
>>> tailles
[5, 2.5, 1.75, 0.15]
```

}

Les listes {

* Un des gros avantages d'une liste est que vous pouvez appeler ses éléments par leur position. Ce numéro est appelé indice (ou index) de la liste.

liste :	'girafe',	'tigre',	'singe',	'souris']
indice:	0	1	2	3

/!\ Soyez très attentifs au fait que les indices d'une liste de n éléments commence à 0 et se termine à n-1. /!\

}

Opérations sur les listes {

* Tout comme les chaînes de caractères, les listes supportent l'opérateur + de concaténation, ainsi que l'opérateur * pour la duplication.

```
>>> ani1 = ['girafe','tigre']
>>> ani2 = ['singe','souris']
>>> ani1 + ani2
['girafe', 'tigre', 'singe', 'souris']
>>> ani1 * 3
['girafe', 'tigre', 'girafe', 'tigre', 'girafe', 'tigre']
```

}

Opérations sur les listes {

* L'opérateur + est très pratique pour concaténer deux listes. Vous pouvez aussi utiliser la fonction append() lorsque vous souhaitez ajouter un seul élément à la fin d'une liste.

```
>>> a = [] # liste vide
>>> a = a + [15]
>>> a
[15]
```

* Ou avec la fonction append() qui sera préférée pour ce genre d'opération

```
>>> a.append(-5)
>>> a
[15, -5]
```

}

Opérations sur les listes {

- * La liste peut également être indexée avec des nombres négatifs selon le modèle suivant :

liste :	['girafe', 'tigre', 'singe', 'souris']			
indice positif :	0	1	2	3
indice négatif :	-4	-3	-2	-1

- * Les indices négatifs reviennent à compter à partir de la fin. Leur principal avantage est que vous pouvez accéder au dernier élément d'une liste à l'aide de l'indice -1 sans pour autant connaître la longueur de cette liste.

}

Fonction len() {

* L'instruction len() vous permet de connaître la longueur d'une liste, c'est-à-dire le nombre d'éléments que contient la liste. Voici un exemple d'utilisation :

```
>>> animaux = ['girafe', 'tigre', 'singe', 'souris']  
>>> len(animaux)  
4
```

}

Time to practice {

Lancez votre IDE et réalisez les petits programmes suivants :

#1

* Créez un script permettant de créer une fiche client (Nom, prénom, téléphone, âge, etc.). Les variables sont remplies grâce aux input utilisateur.

#2

* Créez un script permettant de réaliser une commande au bar. Quel boissons veut l'utilisateur parmi une liste de choix possible (café, bière, etc.).

}

Boucles et comparaisons

Retour sur l'algorithmie

Boucles for {

* En programmation, on est souvent amené à répéter plusieurs fois une instruction. Incontournables à tout langage de programmation, les boucles vont nous aider à réaliser cette tâche de manière compacte. Imaginez par exemple que vous souhaitiez afficher les éléments d'une liste les uns après les autres.

```
animaux = ['girafe', 'tigre', 'singe', 'souris']  
print(animaux[0])  
print(animaux[1])  
print(animaux[2])  
print(animaux[3])
```

```
}
```

Boucles for {

* Si votre liste ne contient que quatre éléments, c'est encore faisable mais imaginez qu'elle en contienne 100 ou plus ! Bien développer c'est avant tout être feignant :

```
>>> animaux = ['girafe', 'tigre', 'singé', 'souris']
>>> for animal in animaux:
...     print(animal)
```

* La variable `animal` est appelée variable **d'itération**, elle prend successivement les différentes valeurs de la liste `animaux` à chaque itération de la boucle.

}

Boucles for {

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

* Le signe deux-points : à la fin de la ligne for. Cela signifie que la boucle for attend un bloc d'instructions.

* On appelle ce bloc d'instructions le corps de la boucle. Comment indique-t-on à Python où ce bloc commence et se termine ? Cela est signalé uniquement par l'indentation,

```
}
```

Fonction range() {

```
for i in range(3) :  
    print(i)
```

```
...
```

```
0
```

```
1
```

```
2
```

* Python possède la fonction range(). Elle est utile pour générer des nombres entiers compris dans un intervalle.

* Pour Python, il s'agit d'un nouveau type, par exemple dans `x = range(3)` la variable `x` est de type *range* (tout comme on avait les types *int*, *float*, *str* ou *list*)

```
}
```

Boucles for - itération sur les indices {

* On peut utiliser un range et itérer sur les éléments d'une liste par ses indices

```
>>> animaux = ['girafe', 'tigre', 'singe', 'souris']
```

```
>>> for animal in animaux:
```

```
...     print(animal)
```

```
...
```

```
girafe
```

```
tigre
```

```
singe
```

```
souris
```

```
}
```


Boucles for - itération sur les indices {

* Python possède également la fonction `enumerate()` qui vous permet d'itérer sur les indices et les éléments eux-mêmes.

```
>>> animaux = ['girafe', 'tigre', 'singe', 'souris']
>>> for i, animal in enumerate(animaux):
...     print("L'animal {} est un(e) {}".format(i, animal))
...
L'animal 0 est un(e) girafe
L'animal 1 est un(e) tigre
L'animal 2 est un(e) singe
L'animal 3 est un(e) souris
```

}

Comparaisons {

Python est capable d'effectuer toute une série de comparaisons entre le contenu de deux variables, telles que :

Syntaxe Python

Signification

==

égal à

!=

différent de

>

supérieur à

>=

supérieur ou égal à

<

inférieur à

<=

inférieur ou égal à

}

Comparaisons {

* Faites bien attention à ne pas confondre l'opérateur d'affectation = qui affecte une valeur à une variable et l'opérateur de comparaison == qui compare les valeurs de deux variables.

```
>>> animal = "tigre"  
>>> animal == "tig"  
False
```

}

While {

* Une autre alternative à l'instruction for couramment utilisée en informatique est la boucle `while`. Le principe est simple. Une série d'instructions est exécutée tant qu'une condition est vraie. Par exemple :

```
>>> i=1
>>> while i ≤ 3:
...     print(i)
...     i=i+1
...
1
2
3
```

While {

- * Une boucle `while` nécessite généralement trois éléments pour fonctionner correctement :
 - l'initialisation de la variable de test avant la boucle ;
 - le test de la variable associé à l'instruction `while` ;
 - la mise à jour de la variable de test dans le corps de la boucle.

}

Time to practice {

Lancez votre IDE et réalisez les petits programmes suivants :

#3

* Soit impairs la liste de nombres [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]. Écrivez un programme qui, à partir de la liste impairs, construit une liste pairs dans laquelle tous les éléments de impairs sont incrémentés de 1.

#4

* Écrivez un script qui dessine un triangle rectangle puis isocèle de 10 '*' de haut

}

Les tests {

* Les tests sont un élément essentiel à tout langage informatique si on veut lui donner un peu de complexité car ils permettent à l'ordinateur de prendre des décisions si telle ou telle condition est vraie ou fausse. Pour cela, Python utilise l'instruction **if**

```
>>> x = 2
>>> if x == 2 :
...     print("Le test est vrai !")
...
Le test est vrai !
```

}

Les tests à plusieurs cas {

* Parfois, il est pratique de tester si la condition est **vraie ou si elle est fausse** dans une même instruction if. Plutôt que d'utiliser deux instructions if, on peut se servir des instructions if et else

```
>>> x = 5
>>> if x == 2 :
...     print("Le test est vrai !")
... else :
...     print("Le test est faux !")
Le test est faux !
```


Les tests à plusieurs cas {

* On peut utiliser une série de tests dans la même instruction if, notamment pour tester plusieurs valeurs d'une même variable.

```
>>> import random
>>> boisson = random.choice(['café', 'bière', 'soda'])
>>> if boisson == 'café':
...     print('Voici un café')
... elif boisson == 'bière':
...     print('Voici une bière')
... elif boisson == 'soda':
...     print('Voici un soda')
} Voici une bière
```

Tests multiples {

* En Python, on utilise le mot réservé **and** pour l'opérateur **ET** et le mot réservé **or** pour l'opérateur **OU**.

```
>>> x = 2
>>> y = 2
>>> if x == 2 and y == 2 :
...     print("le test est vrai")
...
le test est vrai
```

}

Tests multiples {

* Vous pouvez aussi tester directement l'effet de ces opérateurs à l'aide de True et False :

```
>>> True or False  
True
```

* Enfin, on peut utiliser l'opérateur logique de négation not qui inverse le résultat d'une condition :

```
>>> not True  
False
```

```
>>> not False  
True
```

Break & continue {

Ces deux instructions permet de modifier le comportement d'une boucle (for ou while) avec un test.

* L'instruction `break` stoppe la boucle.

```
>>> for i in range(5):  
...     if i > 2 :  
...         break  
...     print(i)
```

* L'instruction `continue` saute à l'itération suivante.

```
>>> for i in range(5):  
...     if i == 2:  
...         continue  
...     print(i)
```

}

Time to practice {

#5

A Fort Boyard, le père Fouras nous pose l'énigme suivante :
Pour ouvrir le coffre où se trouve la clé, trouve la
combinaison à trois chiffres sachant que :

- Le nombre est inférieur à 200.
- Deux de ses chiffres sont identiques.
- La somme de ses chiffres est égale à 5.
- C'est un nombre pair.

* On se propose d'utiliser une méthode dite **brute force**,
c'est à dire d'utiliser une boucle et à chaque itération
on teste les 4 conditions. Avez-vous réussi à trouver la
même solution en raisonnant ?

}

Time to practice { La conjecture de Syracuse

#6

Soit un entier positif n . Si n est pair, alors le diviser par 2. Si il est impair, alors le multiplier par 3 et lui ajouter 1. En répétant cette procédure, la suite de nombres atteint la valeur 1 puis se prolonge indéfiniment par une suite de trois valeurs triviales appelée cycle trivial.

Par exemple, les premiers éléments de la suite de Syracuse si on prend comme point de départ 10 sont : 10, 5, 16, 8, 4, 2, 1. . .

Écrivez un script qui, partant d'un entier positif n (par exemple 10 ou 20), crée une liste des nombres de la suite de Syracuse. Avec différents points de départ (c'est-à-dire avec différentes valeurs de n), la conjecture de Syracuse est-elle toujours vérifiée ? Quels sont les nombres qui constituent le cycle trivial ?
Remarques

1. Pour cet exercice, vous avez besoin de faire un nombre d'itérations inconnu pour que la suite de Syracuse atteigne le chiffre 1 puis entame son cycle trivial. Vous pourrez tester votre algorithme avec un nombre arbitraire d'itérations, typiquement 20 ou 100, suivant votre nombre n de départ.

2. Un nombre est pair lorsque le reste de sa division entière (opérateur modulo %) par 2 est nul.