

Scripting Python

Présentation du cours

Plan du cours

- Présentation de python
- Les bases de python
- Programmation avancée
- Scripting Python

Modalités d'évaluation

- QCM lors de la 7^e séance (10^{ème} heure de cours)
- Exam sous forme de projet, 1 mois pour le faire à partir de la fin du cours.

Modalités d'évaluation

Evaluation finale:

Projet sans soutenance

Critère	Points
FLAG 1, 2, 3	1
FLAG 4, 5, 6	2
FLAG 7, 8	2
FLAG 9, 10	2
FLAG 11	3
FLAG 12	4
Qualité du code	2
Qualité des commentaires	2
Complexité technique	2

Présentation de Python



Histoire

Python est un langage de programmation créé par Guido van Rossum en 1991.

Il avait pour objectif de créer un langage de programmation :

- Facile à lire, à écrire et à maintenir
- Puissant et flexible

C'est un langage dit interprété, multiplateformes. Il permet de faire de la programmation impérative, structurée, fonctionnelle et objet.

Il fonctionnera aussi bien, sous Windows, linux, mac qu'Android, iOS, etc



Les types de langages

- Langage interprété
 - Le code source passe par un interpréteur qui va le transformer en instructions machine qui sera ensuite exécutée.
 - L'interpréteur doit déjà être présent sur la machine
 - L'interpréteur va lire ligne par ligne le code
- Langage compilé
 - Le code source passe par un compilateur qui va le transformé en un langage bas niveau (assembleur ou code machine)
 - La compilation doit être réalisée manuellement pour chacune des plateformes sur lesquelles l'exécutable doit fonctionner.

Les types de langages, les + et -

Langage interprété	Langage compile
Portable	Rapide
Développement et débogage rapide	Peut-être exécuter directement
Lent	Sécurité avancée
Lourd	Développement et débogage lent
Peu de sécurité	Peu portable
Nécessite un interpréteur	

Python dans la cybersécurité

Pourquoi utiliser Python, un langage interprété, dans la Cybersécurité ?

- Syntaxe claire et intuitive
- Facilité d'apprentissage avec une grande communauté
- Souvent utilisé pour:
 - Scripts automatisés
 - Analyse de données
 - Pentest


Installation

Penser à sélectionner au minima les options suivantes:

- Add python.exe to PATH
- Pip
- Tcl/tk and IDLE
- Python test suite
- Py launcher
- Associate files with Python
- Create shortcuts for installed applications
- Add python to environment variables

Install Python 3.10.10 (64-bit)

Select Install Now to install Python with default settings, or Customize to enable or disable features.

 **Install Now**
C:\Users\... \AppData\Local\Programs\Python\Python310

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ **Customize installation**
Choose location and features

- ☒ Use admin privileges when installing py.exe
- ☒ Add python.exe to PATH

Optional Features

- ☒ Documentation
Installs the Python documentation file.
- ☒ pip
Installs pip, which can download and install packages.
- ☒ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.
- ☒ Python test suite
Installs the standard library test suite.
- ☒ py launcher ☒ for all users (requires administrator)
Installs the global 'py' launcher to make it easier to run Python scripts.

Advanced Options

- ☐ Install Python 3.10 for all users
- ☒ Associate files with Python (requires administrator)
- ☒ Create shortcuts for installed applications
- ☒ Add Python to environment variables
- ☐ Precompile standard library
- ☐ Download debugging symbols
- ☐ Download debug binaries (requires administrator)

Mise en pratique: Les différents modes

Python possède deux modes:

- Interprété:

Le code est écrit à la volée, le comportement est similaire à Bash.

- Scripté:

Le code est écrit dans un fichier *code.py*, qui sera exécuté



Mise en pratique: IDE

Un IDE (*Integrated Development Environment – Environnement de développement intégré*) est un ensemble d'outils utilisé par les développeurs pour augmenter leur productivité et l'efficacité.

Ils intègrent en général un éditeur de texte, un compilateur, un débogueur et un éditeur de liens.

Certains IDE sont dédiés à un langage de programmation spécifique.

Voici quelques IDE utilisés pour Python:

- PyCharm (Disponible via la suite JetBrains, gratuit via le mail étudiant) – (only Python)
- Wing (only Python)
- Thonny (only Python)
- VsCode (multilangage)



Mise en pratique: extensions

Les IDE permettent en général d'ajouter des extensions spécifiques pour faciliter le développement.

Par exemple, sous VsCode:

- GitHub Copilot (disponible gratuitement avec votre mail étudiant) ou BlackBox AI
- MySQL de Weijan Chen
- Prettier – Code formatter de prettier
- GitLens de gitkraken
- Python & Python debugger de Microsoft

Les bases de Python



Les commentaires

Quel que soit le langage que vous allez utiliser, les commentaires sont importants et nécessaires. Que vous soyez en équipe ou en solo.

En python, il existe deux types de commentaires:

- Le ``#``, permet de commenter une ligne uniquement
- Les triples quotes `«"""Ceci est un commentaire"""»`, permettent de commenter plusieurs lignes.

Pour rappel, sur la première ligne du fichier, il est habituel de mettre un shebang (`#!`)

- Indique au système d'exploitation que le fichier est un script
- Sur la même ligne, on précise l'interpréteur utiliser (python/python3/bash/etc)
- Exemple: `#!/usr/bin/python`

Les commentaires

Python > exemples > comment.py

```
1  #!/usr/bin/python
2
3  # Mon programme permet de faire des calculs de base (addition, soustraction, multiplication, division)
4
5  """
6  Mon programme permet de faire des calculs de base:
7  - addition
8  - soustraction
9  - multiplication
10 - division
11 """
```

```
13 def addition(a: float, b: float) -> float:
14     """
15     Enables you to perform addition
16
17     Args:
18         - a: The first number
19         - b: The second number
20
21     Returns:
22         - Error if first or second number is null
23         - Result of addition
24     """
25     if a is None or b is None:
26         return "Error: Both numbers must be provided"
27     else:
28         return a + b
```




Normes de codage

Lorsque plusieurs personnes travaillent sur un projet, il est commun de définir des normes de codage, afin que tout le monde code de la même façon:

- Le style d'indentation (tab/espaces, 2 ou 4 – PEP8 indique 4 espaces)
- Les commentaires (Anglais, clair et concis)
- Le nom et l'organisation des fichiers
- camel case ou snake case
 - En général, l'un est choisi pour les fonctions et l'autre pour les variables

```
31 def xml_to_pdf(optionsInfos, optionsObjects, xmlReport):  
32  
33 def xmlToPdf(options_infos, options_objects, xml_report):
```

Les variables

Comme tous les langages, python dispose de plusieurs types de variables.
Les plus courants sont:

- Int (integer, nombres entiers)
- Float (nombre décimal,)
- Str (string, chaîne de caractères, toujours entre `""` ou `'''`)
- Bool (booléen, True ou False, 0 ou 1)
- Hex (hexadecimal)
- Bin (binaire)
- Etc

La commande `type(varName)` permet de connaître le type d'une variable.

Les variables

Python attribut automatiquement le bon type de variable

Une variable change de type automatiquement si le type de son contenu change.

```
>>> a = 3
>>> type(a)
<class 'int'>
>>> a = 3.15
>>> type(a)
<class 'float'>
```

```
>>> a = b = c = 10
>>> print(a,b,c)
10 10 10
>>> a,b,c = 8,9,10
>>> print(a,b,c)
8 9 10
```

```
>>> def add(a: float, b: float) -> float:
...     return a + b
...
>>> k = add(3, 5)
>>> type(k)
<class 'int'>
>>> k = add(3, 5.15)
>>> type(k)
<class 'float'>
>>> print(k)
8.15
```

Interaction utilisateur

Avoir une interaction avec l'utilisateur peut être utile:

- Nous allons donc utiliser le mot-clé: input
- Tous les inputs sont des string, si vous souhaitez avoir un integer ou autre, il faudra faire la conversion

```
mail = input("Mail ?")  
print(type(mail))  
print(mail)
```

```
Mail ?toto@toto.com  
<class 'str'>  
toto@toto.com
```

```
nb = input("Numéro ?")  
print(type(nb))  
print(nb)  
nb = int(nb)  
print(type(nb))  
print(nb)
```

```
Numéro ?1  
<class 'str'>  
1  
<class 'int'>  
1
```

Les structures de données

Il existe plusieurs structures de données au sein de Python:

- Les **listes**, associées aux crochets []
- Les **dictionnaires**, associés aux accolades {}
- Les **tuples**, 2 éléments ou plus, immuables, entre parenthèses ()

Ces structures de données sont des types de variables à part entière

Les structures de données : Les listes

```
string = "Hello, World!"  
print(string[0])  
  
a = ['blue', 'red', 'green']  
print(a[1])
```

Les listes sont des suites de valeurs (et/ou variables) ordonnées. Les données peuvent être de types différents. On y accède à partir de leur index (position).

Pour rappel, en programmation, les listes commencent à zero !

Les strings sous python sont des listes de caractères



Les structures de données : Les listes

Suivant les situations, vous devrez faire des tableaux à 2 ou 3 dimensions, voire plus, pour ce faire, vous utiliserez les listes:

- 2D
 - Liste de liste

```
twoDimensions = [[1, 2, 3],[4, 5, 6]]  
print(twoDimensions[0])  
print(twoDimensions[0][1])
```

- 3D
 - Liste de liste de liste

```
threeDimensions = [[[1, 2, 3],[4, 5, 6]],[[7, 8, 9],[10, 11, 12]]]  
print(threeDimensions[0])  
print(threeDimensions[0][0])  
print(threeDimensions[0][0][1])
```

Les structures de données : Les listes

En utilisant les listes, vous serez amené à faire différents types d'opérations, dans notre exemple, notre liste est la variable a:

a[-1]	Retourne le dernier élément de la liste
a[3:]	Retourne les éléments à partir de l'index 3
a[:5]	Retourne les 5 premiers éléments de la liste
a[-5:]	Retourne les 5 derniers éléments de la liste
a[-2]	Retourne le second élément en partant de la fin
a[::-1]	Retourne la liste en ordre inverse
a[:]	Retourne toute la liste... on aurait juste pu faire : a
a[2::3]	Retourne les éléments à partir du second par pas de 3
a[2:5]	Retourne les éléments contenus des indexes 2 à 4 (5-1)
a[2:8:2]	Retourne les éléments de l'index 2 à 7 (8-1) par pas de 2

La borne supérieure est toujours exclue

Les structures de données : Les listes

Il est également possible d'utiliser des mots-clés:

a.copy()	Copie la liste dans une autre variable
a.pop()	Retourne la dernière valeur et supprime l'entrée de la liste
a.append(val)	Ajoute une valeur à la liste
a.count(val)	Retourne le nombre de 'val' présent dans la liste
a.index(val)	Retourne l'index (la position) de 'val' dans la liste
a.remove(val)	Supprime 'val' de la liste
a.extend(list2)	Intègre list2 à la liste 'a'
a.insert(i, val)	Insère 'val' à la position numéro 'i'
a.reverse()	Retourne la liste en ordre inverse
a.clear()	Supprime toute la liste

Les structures de données : Les dictionnaires

Les dictionnaires sont des types de données très puissants et pratique, ils permettent de stocker des paires clé/valeur (key/value).

Les Valeurs stockées peuvent aussi être des tuples et/ou listes.

```
identity = {  
    "name": "John",  
    "age": 30,  
    "city": "New York"  
}  
  
print(identity)  
print(identity["name"])
```

Les structures de données : Les dictionnaires

Tout comme les listes, il est possible de faire des dictionnaires à plusieurs dimensions

2D

```
notes = {  
    "John": {"Math": 90, "Science": 80},  
    "Jane": {"Math": 70, "Science": 90}  
}  
  
print(notes)  
print(notes["John"])  
print(notes["Jane"]["Math"])
```

3D

```
notes = {  
    "John": {  
        "Trimestre 1" : {"Math": 90, "Science": 80},  
        "Trimestre 2" : {"Math": 70, "Science": 90}  
    },  
    "Jane": {  
        "Trimestre 1" : {"Math": 70, "Science": 90},  
        "Trimestre 2" : {"Math": 90, "Science": 80}  
    }  
}  
  
print(notes)  
print(notes["John"]["Trimestre 1"]["Math"])
```

Les structures de données : Les dictionnaires

Il est également possible d'utiliser des mots-clés:

Dico.items()	Retourne une liste de tuple clé, valeur
Dico.keys()	Retourne une liste contenant les clés
Dico.values()	Retourne une liste contenant les valeurs
Dico.copy()	Copie tout le dico dans une autre variable
Dico.pop('key')	Retourne la valeur de la dernière clé et supprime l'entrée du dictionnaire
Dico.popitem()	Retourne la dernière paire clé, valeur et la supprime du dictionnaire.
Dico.clear()	Efface toutes les entrées du dictionnaire

Les structures de données : Les tuples

Les tuples sont des types de données immuables (qu'on ne peut modifier), ainsi il ne sera ni possible d'ajouter, supprimer ou modifier des éléments.

Comme les précédentes structures, les tuples peuvent contenir d'autres types de données. En revanche celle-ci étant mutable, elles pourront être modifiées.

Par exemple:

Si j'ai un tuple, qui contient une valeur standard et une liste.

Je ne pourrais pas modifier la valeur standard mais le contenu de la liste pourra être modifier.

Les structures de données : Les tuples

```
tuples = (1, 2, 3, 4, 5)
print(tuples)
print(tuples[0])
```

```
(1, 2, 3, 4, 5)
1
```

```
Dtuples = (1, 2, [3, 4], 5,)
print(Dtuples)
print(Dtuples[2])
Dtuples[2].append(10)
print(Dtuples)
```

```
(1, 2, [3, 4], 5)
[3, 4]
(1, 2, [3, 4, 10], 5)
```

Les structures de données : Les tuples

Il est également possible d'utiliser des mots-clés:

Tuple.count(x)	Retourne le nombre de fois que l'élément "x" apparaît
Tuple.index(x)	Retourne l'index de la première occurrence de l'élément "x"

Les fonctions

En programmation, il est courant d'utiliser des fonctions, elles vont permettre de placer des morceaux de code dans une sorte de "capsule" pouvant être appelé n'importe quand.

Ainsi un code que vous aviez écrit 4 fois, sera écrit une seule fois et appelé 4 fois.

Il est plutôt commun de faire un fichier fonction, dans laquelle vous pourrez mettre toutes vos fonctions pour les appeler plus tard

À savoir:

Que ce soit sur vos fonctions, boucles ou conditions, l'indentation en python est très importante.

```
def addition(a: float, b: float) -> float:
    """
    Enables you to perform addition

    Args:
        - a: The first number
        - b: The second number

    Returns:
        - Error if first or second number is null
        - Result of addition
    """
    if a is None or b is None:
        return "Error: Both numbers must be provided"
    else:
        return a + b

a = 5
b = 3
print(addition(a, b))
```




Création de Pokémon & interaction avec un utilisateur – Etape 1

Aspect du cours à utiliser lors du TP:

- Variables
- Input
- Structures de données
- Fonctions
- Créer des Pokémon en utilisant des variables pour stocker leurs caractéristiques (nom/type/HP/attaque/défense)
- Stocker la liste des Pokémon dans une structure de données
- Créer une interaction avec l'utilisateur pour créer deux Pokémon

Formatage de chaîne

Comme vous avez pu le voir précédemment, nous utilisons “print” pour afficher les données.

Voici différentes façons d’afficher les données et de les formater au besoin.

```
string = "Antoine"
intVar = 10
floatVar = 10.5

print('Bonjour: %s - %i - %f' %(string, intVar, floatVar))
print('Rebonjour: {} - {} - {}'.format(string, intVar, floatVar))
print('Rerebonjour:' + string + ' - ' + str(intVar) + ' - ' + str(floatVar))
print(f"Bonjour: {string} {intVar} {floatVar}")
```

```
Bonjour: Antoine - 10 - 10.500000
Rebonjour: Antoine - 10 - 10.5
Rerebonjour:Antoine - 10 - 10.5
Bonjour: Antoine 10 10.5
```

L'encodage

L'encodage de caractères en python pourra vous poser problème.

Il existe plein de type d'encodage, notamment utf-8, utf-16, hex, unicode, bytes, etc. Dans python, deux mots-clés sont utilisés **encode** et **decode**.

```
string = 'John'
print(string.encode('utf-16')) # encode en utf-16
print(string.encode()) # encode en bytes
print(string.encode().hex()) # encode en bytes puis hexa
print(int(string.encode().hex(), base=16)) # encode en bytes puis hexa puis converti en base16
```

```
b'\xff\xfeJ\x00o\x00h\x00n\x00'
b'John'
4a6f686e
1248815214
```

Les conditions

Lorsque vous développez, vous êtes amenés à utiliser ce qu'on appelle des branchements conditionnels.

Les mots-clés tels que:

- “in” et “not in” peuvent être utilisés couplés avec une liste ou un dictionnaire dans une condition.
- “and” et “or” peuvent être utilisés afin de lier plusieurs conditions. (ex: if $a < b$ or $a == b$)

```
# nb1 = input("nb1") # type str
# nb2 = input("nb2") # type str

nb1 = int(input("nb1"))
nb2 = int(input("nb2"))
print(type(nb1))
print(type(nb2))

if nb1 > nb2:
    print("nb1 est le plus grand")
elif nb1 < nb2:
    print("nb2 est le plus grand")
else:
    print("Les deux nombres sont égaux")
```

Les boucles

Quel que soit le langage utilisé, vous serez amené à utiliser des boucles. Vous pourrez ainsi itérer une liste/tuple/dictionnaire, le contenu d'un fichier ou répéter à plusieurs reprises la même opération.

Il existe deux types de boucles : **for** et **while**

Les mots-clés tels que:

- "in" peut être utilisé afin d'itérer sur une liste
- "len" peut être utilisé pour itérer sur une liste en se basant sur sa longueur

Les boucles : For

```
"""
for sur un dico
"""

identity = {
    "name": "John",
    "age": 30,
    "city": "New York"
}

for key in identity:
    print(key)
    # name
    # age
    # city

for value in identity.values():
    print(value)
    # John
    # 30
    # New York
```

```
for identities in identity.items():
    print(identities)
    # ('name', 'John')
    # ('age', 30)
    # ('city', 'New York')

for key, value in identity.items():
    print(key, value)
    # name John
    # age 30
    # city New York
```

Les boucles : Range

Range est une fonction que vous allez utiliser à plusieurs reprises et principalement avec des for/while:

Range(5)	Génère des entiers de 0 à 5(-1)
Range(5, 10)	Génère des entiers de 5 à 10(-1)
Range(0, 10, 2)	Génère des entiers de 0 à 10(-1) avec un step de 2

```
"""  
for sur une liste  
"""  
  
lst = [1, 2, 3, 4, 5]  
for i in lst:  
    print(i)  
  
for i in range(0, len(lst)):  
    print(lst[i])
```

Les boucles : While

Dans certains cas, vous verrez des “break” dans des “while true” ou des “while false”. Cette pratique est fortement déconseillée, faites plutôt comme dans la seconde image

```
a = 10
b = 0
while b <= a:
    print(b)
    b += 1
# 0 1 2 3 4 5 6 7 8 9 10
```

```
userInput = ""
while userInput != "exit":
    userInput = input("Entrez quelque chose (ou 'exit' pour quitter) : ")
    if userInput != "exit":
        print(f"Vous avez entré : {userInput}")
```




Création de Pokémon & interaction avec un utilisateur – Etape 2

En partant du précédent code, et en gardant ses fonctionnalités

Aspect du cours à utiliser lors du TP:

- Conditions
- Boucles
- Formatage de chaînes
- Ajouter des conditions pour vérifier si le Pokémon existe déjà avant sa création
- Utiliser des boucles pour permettre à l'utilisateur de créer autant de Pokémon qu'il le souhaite
- Formater l'affichage des informations des Pokémon

Programmation avancée

Help & Cheat Sheet

Sous linux nous avons le **man**, en python nous avons le **help**.

Le help est une fonction python qui vous permettra d'avoir différentes informations sur une librairie, une fonction.

- Help (<function/lib Name>)

Vous pourrez également trouver des Cheat Sheet sur internet, celles-ci sont des documents concis qui résument les concepts et commandes des librairies/fonctions. Elles sont très puissantes, et peuvent parfois être plus intéressantes que le help et le man.



Les librairies

En programmation, les librairies sont des morceaux de code (fonctions) regroupé ensemble, créées par d'autres développeurs, elles sont mises à votre disposition pour vous simplifier certaines tâches.

En python, vous utiliserez régulièrement des librairies. Certaines sont déjà importées nativement (os/sys/etc).

Plusieurs utilitaires existantes afin d'importer une librairie utilisable par python.

- Pip (pip install <lib>)
- Conda



Les librairies : Requirements.txt

Requirements.txt est un fichier utilisé en python pour lister les dépendances du projet (librairies installées précédemment).

Deux commandes sont liées à ce fichier:

- pip freeze > requirements.txt
 - *pour sauvegarder les dépendances*
- pip install -r requirements.txt
 - *pour installer les dépendances du fichier*

```
Python > ≡ requirements.txt
1  certifi==2024.7.4
2  charset-normalizer==3.3.2
3  idna==3.8
4  requests==2.32.3
5  scapy==2.5.0
6  sockets==1.0.0
7  subprocess.run==0.0.8
8  threaded==4.2.0
9  urllib3==2.2.2
```

Les librairies : Imports

Afin d'utiliser les librairies, vous devez les importer dans le code. Autrement le programme ne pourra effectuer l'action demandée.

En python, nous utilisons le mot-clé **import**.

```
3  import re
4  import os, sys
5  print(os.getcwd())
```

Il est aussi possible de faire appel qu'à une fonction présente dans une librairie.

```
8  from os import getcwd
9  print(getcwd())
```

Les librairies : OS

La librairie OS regroupe plusieurs fonctionnalités utiles pour interagir avec le système d'exploitation.

Voici quelques fonctions, vous pourrez trouver les autres dans le help ou les cheat sheet

Os.getcwd()	Retourne le chemin du répertoire de travail courant
Os.listdir(path)	Retourne une liste des fichiers et sous répertoires
Os.makedirs(path)	Crée un répertoire (y compris les parents)
Os.system	Exécute une commande système

Les librairies : Sys & Argv

La librairie SYS est intéressante pour sa fonction argv.

Argv est une fonction qui permet de récupérer les arguments passé en paramètre d'un programme.

A savoir:

- Argv retournera toujours une liste
- La Valeur de l'index 0 sera toujours le nom du programme exécuté

```
19  from sys import argv
20
21  print(argv)
```

```
python imports.py toto tata
['imports.py', 'toto', 'tata']
```




Création de Pokémon & interaction avec un utilisateur – Etape 3

En partant du précédent code, et en gardant ses fonctionnalités

Aspect du cours à utiliser lors du TP:

- Modules
- Imports
- sys.argv

- Séparer le code dans plusieurs fichiers, pour gérer la création des Pokémon et leur affichage
- Utiliser argv pour passer des arguments directement à l'appel du fichier

Gestion de fichier

En programmation, l'utilisation de fichier est courante. Il est donc utile de savoir comment les utiliser, créer, supprimer, modifier.

Il existe plusieurs modes d'ouverture de fichier.

- r (read) mode lecture simple
- w (write) mode écriture simple, réécrit sur le fichier en effaçant tout ce qui s'y trouve
- a (append) mode ajout dans le fichier, ajoute à la fin sans supprimer le contenu existant
- rb/wb/ab sont des modes d'ouverture de fichiers permettant la lecture et écriture en Byte mode

```
# ouverture du fichier en lecture uniquement
file = open('text.txt', 'r')

# lecture du fichier
print(file.read())

# toujours penser à fermer le fichier
file.close()
```

```
# ouverture du fichier en lecture uniquement
# ajout de l'alias file pour pouvoir le lire
with open('text.txt', 'r') as file:
    print(file.read())

# pas besoin de fermer le fichier en utilisant with, il le gère tout seul
# une fois l'indentation terminée, with ferme le fichier
```

Gestion de fichier

.read()	Lit tout le contenu sous forme de chaîne de caractère Un argument peut être passé pour spécifier le nombre de caractère à lire
.readline()	Lit une seule ligne du fichier, s'il est dans une boucle, il lira la ligne d'après Un argument peut être passé pour spécifier le nombre de caractère à lire
.readlines()	Lit toutes les lignes d'un fichier et les retourne sous forme de liste
.write()	Écrit une chaîne de caractères dans le fichier. Gère les \n
.writelines()	Écrit une séquence de chaînes de caractères dans le fichier. Ne gère pas les \n

Parsing de chaînes de caractères

Pour rappel, sous python, les chaînes sont des listes de caractères.

Split permet de séparer une chaîne à partir d'un caractère donné.

- Les caractères tels que “\t” et “\n” sont pris en compte par split.

```
string = "Hello everyone"

print(string.split(" "))
# Output: ['Hello', 'everyone']
```

```
string = "Hello everyone \n this is a test string \n with multiple lines"

print(string.split(" "))
# Output: ['Hello', 'everyone', '\n', 'this', 'is', 'a', 'test', 'string', '\n', 'with', 'multiple', 'lines']
```

Gestion d'erreur

En programmation, il faut toujours tout prévoir. Vous pouvez avoir fait le meilleur programme du monde que pour x ou y raison, il se peut qu'il ne puisse pas se lancer. (mauvais os, lib manquante, etc).

Vous allez donc devoir apprendre à gérer les erreurs, pour cela, en python, vous pourrez utiliser **try**, **except**, **finally**.

Le code contenu dans le **try** sera celui qui va être tenté de s'exécuter. Si une erreur survient, ce sera celui du **except**. Vous pouvez ajouter un **finally** qui sera exécuté quoi qu'il arrive

Contrairement aux **try/except** le **finally** est optionnel

Gestion d'erreur

```
def division(a: float, b: float) -> float:
    """
    Enables you to perform addition

    Args:
        - a: The first number
        - b: The second number

    Returns:
        - Error if first or second number is null
        - Result of addition
    """
    if a is None or b is None:
        return "Error: Both numbers must be provided"
    else:
        try:
            return a/b
        except ValueError as e:
            print(f"Error : Data invalid. Details : {e}")
        except ZeroDivisionError as e:
            print(f"Error : Zero division. Details : {e}")
        finally:
            print("Division")
```

Print(division(5, 5))

```
Division
1.0
```

Print(division(5, 0))

```
Error : Zero division. Details : division by zero
Division
None
```

Programmation Orientée Objet: POO

La programmation orientée objet est une façon spécifique de programmer qui tourne autour des objets. Ces objets peuvent contenir des données (variables) et des comportements (fonctions).

Les concepts clé de la POO en python sont:

Classe	C'est un modèle qui définit la structure et le comportement d'un ensemble d'objet On y spécifie les variables et fonctions que les objets de la classe vont posséder
Objet	C'est une instance d'une classe. Une fois que la classe est définie, on peut créer plusieurs objets basés sur cette classe. Chaque objet peut avoir ses propres valeurs d'variables mais il partage la même structure et fonctions définies dans la classe
Héritage	C'est un mécanisme qui permet de créer une nouvelle classe à partir d'une existante. La classe enfant héritera des variables et fonctions de la classe parent, elle pourra ajouter ou redéfinir des fonctions déjà existantes
Encapsulation	Cela permet de restreindre l'accès aux variables et aux fonctions d'un objet.



Programmation Orientée Objet: Classes

La classe est "Car"

Ses variables sont "brand, model & year"

Sa fonction est "start"

```
class Car:
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    def start(self):
        print(f"The {self.brand} {self.model} is starting...")
```





Programmation Orientée Objet: Objets

L'objet est "myCar", je peux le créer dans le même fichier ou dans un fichier externe, du moment que je fais appel à la classe.

Ici, j'ai défini les valeurs "brand, model & year" demandé par la classe, puis j'ai fait appel à sa fonction

```
myCar = Car("Tesla", "X", 2024)
myCar.start()
# The Tesla X is starting...
```



Programmation Orientée Objet: Héritage

Nous avons deux classes
“Vehicle” & “Car”

La classe “Car” vient hériter de
“Vehicle”, elle a donc accès à
ses variables et fonctions,
nous pouvons aussi lui créer
ses fonctions.

```
class Vehicle:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def start(self):
        print(f"The {self.brand} {self.model} is starting...")

class Car(Vehicle):
    def __init__(self, brand, model, year):
        super().__init__(brand, model)
        self.year = year

    def honk(self):
        print("Honk honk!")

myCar = Car("Tesla", "Y", 2024)
myCar.start()
myCar.honk()
```

Programmation Orientée Objet: Encapsulation

“name” & “balance” sont définis comme des variables privés, par conséquent on ne peut y accéder qu’en utilisant les fonctions de la classe “CB”

```
class CB:
    def __init__(self, name, balance):
        self.__name = name
        self.__balance = balance

    def put(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"{amount} $ were deposited :")
        else:
            print("The amount to be deposited is not positive")

    def show(self):
        print(f"Name: {self.__name}, Balance: {self.__balance}")

myCB = CB("John", 10)
print(myCB)
# Error
myCB.show()
# Print: Name: John, Balance: 10
myCB.put(50)
# Print: 50 $ were deposited in the account.
myCB.show()
# Print: Name: John, Balance: 60
```



Création de Pokémon & interaction avec un utilisateur – Etape 4

En partant du précédent code, et en gardant ses fonctionnalités

Aspect du cours à utiliser lors du TP:

- Gestion de fichiers
- Gestion d'erreurs
- POO
- Sauvegarder les Pokémon dans un fichier txt/json et les charger au démarrage
- Gérer les erreurs liées aux fichiers et à l'utilisateur
- Représenter les Pokémon et caractéristiques avec des classes



Threading et parallélisme

Le **threading** et le **parallélisme** sont deux concepts différents, même si proches sur certains aspects.

Le **threading** permet de créer des sous-processus qui vont se lancer afin d'effectuer deux tâches en "parallèle"

Le **parallélisme** est similaire au threading pour la "parallélisation" des processus, or ils sont bien différents.

Les **threads** ne vont utiliser qu'un processeur de la machine et les tâches parallélisées ne le seront pas réellement puisque chaque processus va effectuer un calcul puis rendre la main au second.

Le **parallélisme**, va vous permettre d'utiliser plusieurs processeurs. Les processus lancés seront alors, là, bien parallélisés.

Les deux sont très utiles, et vont vous permettre dans certains cas de gagner du temps de calcul et accélérer le programme.

Threading

Lorsque vous lancez des threads, vous pouvez les laisser faire leur vie et continuer l'exécution du programme ou les attendre.

Pour utiliser les threads, il existe une librairie nommée **threading**.

```
import threading

def thd(name):
    print(f"{name} - started")
    print(f"{name} - finished")

if __name__ == "__main__":
    for x in range(4):
        # Création du thread en lui passant une variable en paramètre
        myThread = threading.Thread(target=thd, args=("Thread- {}".format(x + 1),))

        # Lancement du thread
        myThread.start()
```

```
Thread- 1 - started
Thread- 1 - finished
Thread- 2 - started
Thread- 3 - started
Thread- 3 - finished
Thread- 4 - started
Thread- 2 - finished
Thread- 4 - finished
```

Parallélisme

Pour le parallélisme, il existe deux librairies qui peuvent vous être utiles.

Multiprocessing:

```
import multiprocessing
import time

def task(x):
    print(f"Task {x} started")
    time.sleep(2)
    print(f"Task {x} finished")
    return x

if __name__ == "__main__":
    # Créer un pool avec 4 processus
    with multiprocessing.Pool(4) as pool:
        results = pool.map(task, range(4))

    print("Results :", results)
```

```
Task 0 started
Task 1 started
Task 2 started
Task 3 started
Task 0 finished
Task 1 finished
Task 2 finished
Task 3 finished
Results : [0, 1, 2, 3]
```

Joblib:

```
from joblib import Parallel, delayed
import time

def task(x):
    print(f"Task {x} started")
    time.sleep(2)
    print(f"Task {x} finished")
    return x

if __name__ == "__main__":
    # Exécuter la fonction task en parallèle avec 4 processus
    results = Parallel(n_jobs=4)(delayed(task)(i) for i in range(4))

    print("Results :", results)
```

Scripting Python



Un peu plus loin dans python

Python est bien plus puissant et détaillé que ce que nous venons de voir. On va voir ensemble quelques librairies que vous serez amenées à utiliser tel que:

- Subprocess
- Request
- Socket

Ces librairies (comme tout en Informatique) peuvent être “détournées” de leur utilisation initiale afin de créer des scripts d’attaque/défense

Tips

- Faites en sorte que votre script soit le plus modulable possible, afin d'implémenter de Nouvelles fonctionnalités facilement et rapidement
- Réfléchissez à comment construire votre programme avant de vous lancer: 1 seul fichier, plusieurs fichiers, je mets quoi où, etc
- Mettez toujours des commentaires, ils vous seront utiles à vous et à vos collègues.

Subprocess

- Subprocess, vous permet d'utiliser des commandes systèmes plus efficacement que via la librairie OS.
- Ses principales fonctions, sont "call", "Popen", "check_output", "run":

```
from subprocess import call, Popen, PIPE, check_output, run

# Lance la commande ipconfig et affiche directement son résultat
ipconfig = call('ipconfig')

# Lance la commande et récupère le résultat
ping = Popen('ping 8.8.8.8', stdout=PIPE)
print(ping.stdout.read())

# Lance la commande et récupère le résultat dans out
out = check_output('arp -a')
print(out)

# Run la commande whoami et ajoute le résultat dans output
output = run('whoami', stdout=PIPE)
print(output.stdout)
```

Requests

```
from requests import get, post

# Envoie une requete get à un site avec des valeurs d'authentification
r = get('https://api.github.com/user', auth=('user', 'pass'))

# Envoie une requete poste à un site avec des data sous formes de dico
r = post('https://httpbin.org/post', data={'key': 'value'})

# Récupère le code de retour
print(r.status_code)
# Affiche le content-type du header
print(r.headers['content-type'])
# Récupère l'encodage de la requete
print(r.encoding)
# Affiche la réponse au format text
print(r.text)
# Affiche la réponse au format json
print(r.json())
```

- Requests va vous permettre de requêter facilement des sites internet et d'en récupérer la réponse.
- Voici quelques utilisations de base

Socket

```
from socket import socket, AF_INET, SOCK_STREAM

target_host = "www.google.com"
taget_port = 80
# Create a socket object
client = socket(AF_INET, SOCK_STREAM)

# Connect the client
client.connect((target_host, taget_port))

# Send some data
request = "GET / HTTP/1.1\r\nHost:%s\r\n\r\n" % target_host
client.send(request.encode())

# Receive some data
response = client.recv(4096)
http_response = repr(response)
http_response_len = len(http_response)

# Display the response
print(http_response)
```

- Socket est utilisé pour tout ce qui est communication réseau
- On peut l'utiliser pour mettre en écoute un port en tant que serveur ou se connecter à un serveur et envoyer des requêtes
- Voici un exemple de trame http

Pour aller plus loin...

Python à une communauté très active, beaucoup de choses et librairies existe.

Afin de vous améliorer en python, vous pouvez par exemple:

- Parser des pages web avec scrapy
- Développer des sites web avec Django
- Faire du machine learning
- Automatiser des connexion ssh/sftp avec paramiko
- etc