Développeur d'Applications Python
Formation qualifiante

M105 PROGRAMMATION ORIENTÉE OBJET EN PYTHON

Résumé théorique

Abdelkrim Ben yahia
DR Fès Meknès
Formateur en digital & IA

M105 PROGRAMMATION ORIENTÉE OBJET EN PYTHON

OBJECTIF

Ce module permet au stagiaire de maitriser la programmation orientée objet, les fichiers, les scriptes d'automatisation et le déploiement d'applications python dans le git

90 heures

SOMMAIRE

PARTIE 1

Utiliser les packages en Python

Chapitre 1 : Utiliser et créer un module

Chapitre 2 : Gérer les chaines de caractères.

PARTIE 2

Coder une solution orientée objet en Python

Chapitre 1 : Coder d'une classe

Chapitre 2 : Intégrer les concepts POO

Chapitre 3 : Gérer les Exceptions

PARTIE 3

Apprendre à utiliser d'autres fonctionnalités Python

Chapitre 1 : Manipuler les Générateurs

Chapitre 2 : Utiliser les fonctions lambda

Chapitre 3 : Travailler avec des fichiers réels

Chapitre 4 : Ecrire des scripts d'automatisation avec Python

Chapitre 5 : Gérer et déployer un projet Python dans le Git

PARTIE 1

Utiliser les packages en Python

CHAPITRE 1

Utiliser et créer un module

1. **Utiliser les packages en Python**
2. Coder une solution orientée objet en Python
3. Apprendre à utiliser d'autres fonctionnalités Python

Définition de code réel

- Un code réel, largement utilisé se développe en continu, à mesure que les demandes et les attentes des utilisateurs se développent dans leurs propres rythmes.

- Un code qui n'est pas en mesure de répondre aux besoins des utilisateurs sera rapidement oublié et remplacé instantanément par un nouveau code, meilleur et plus flexible.

- La croissance du code est en fait un problème croissant. Un code plus grand signifie toujours un entretien plus difficile.

Définition d’un module

- Si vous souhaitez mener à bien un tel projet logiciel, vous devez disposer des moyens vous permettant de:

  ✓ répartir toutes les tâches entre les développeurs;

  ✓ joindre toutes les parties créées en un seul ensemble fonctionnel.

- Chacune de ces parties peut être (très probablement) divisée en plus petites, et ainsi de suite. Un tel processus est souvent appelé décomposition .

- Par exemple, si on vous demandait d'organiser un mariage, vous ne feriez pas tout vous-même - vous trouveriez un certain nombre de professionnels et répartiriez la tâche entre eux tous.

- Comment divisez-vous un logiciel en parties distinctes mais coopérantes? Ceci est la question. Les modules sont la réponse.

Utilisation d’un module

- La gestion des modules se compose de deux manière différents:

  ✓ le premier (probablement le plus courant) se produit lorsque vous souhaitez utiliser un module déjà existant, écrit par quelqu'un d'autre ou créé par vous-même lors de votre travail sur un projet complexe - dans ce cas, vous êtes l' utilisateur du module

  ✓ la seconde se produit lorsque vous souhaitez créer un tout nouveau module, soit pour votre propre usage, soit pour faciliter la vie d'autres programmeurs - vous êtes le fournisseur du module.

- un module est identifié par son nom . Si vous souhaitez utiliser n'importe quel module, vous devez connaître le nom. Un nombre (assez important) de modules est fourni avec Python lui-même.

Manipulation des module standard python

- Tous ces modules, ainsi que les fonctions intégrées, forment la bibliothèque standard Python,

- Chaque module se compose d'entités (comme un livre se compose de chapitres). Ces entités peuvent être des fonctions, des variables, des constantes, des classes et des objets.

- l'un des modules les plus fréquemment utilisés, nommé math, contient une riche collection d'entités comme sin () ou log ().

- Un exemple de deux entités fournies par le module math:

  ✓ un symbole (constant) représentant une valeur précise de π (pi).

  ✓ une fonction nommée sin()

Importation d’un module

- Pour rendre un module utilisable, vous devez l'importer. L'importation d'un module se fait par une instruction nommée import.

- Ces deux entités sont disponibles via le module math, mais la façon dont vous pouvez les utiliser dépend fortement de la façon dont l'importation a été effectuée.

- La façon la plus simple d'importer un module particulier consiste à utiliser l'instruction d'importation comme suit: import math, sys

- L'instruction import peut se trouver n'importe où dans votre code, mais elle doit être placée avant la première utilisation de l'une des entités du module .

Définition d’un espace de noms

- Un espace de noms est un espace dans lequel certains noms existent et les noms ne sont pas en conflit les uns avec les autres. Nous pouvons dire que chaque groupe social est un espace de noms - le groupe a tendance à nommer chacun de ses membres d'une manière unique

- À l'intérieur d'un certain espace de noms, chaque nom doit rester unique .

- Si le module d'un nom spécifié existe et est accessible (un module est en fait un fichier source Python), Python importe son contenu, c'est-à-dire que tous les noms définis dans le module deviennent connus , mais ils n'entrent pas dans l'espace de noms de votre code.

- Exemple : math.pi et math.sin

Importation d'un module

- la deuxième méthode pour l’importation, syntaxe indique avec précision l'entité (ou les entités) du module qui est acceptable dans le code: from math import sin, pi

- la troisième méthode pour l’importation, syntaxe est une forme plus agressive de celle présentée précédemment (* indique tous les entités) : from module import *

- Pour la troisième méthode essayez de ne pas l'utiliser dans le code normal parce que vous ne pourrez peut-être pas éviter les conflits de noms.

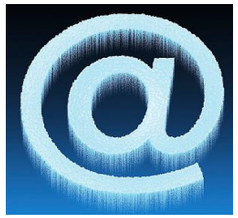
Utilisation d’alias d’un module

- Si vous utilisez la 1ère méthode d'importation du module et que vous n'aimez pas le nom d'un module particulier, vous pouvez lui donner le nom que vous voulez - c'est ce qu'on appelle l' aliasing.

- Un alias fait en sorte que le module soit identifié sous un nom différent de l'original. Cela peut également raccourcir les noms qualifiés. Ex: import module as alias

- après l'exécution réussie d'une importation avec alias, le nom du module d'origine devient inaccessible et ne doit pas être utilisé.

- Vous pouvez créez un alias pour l’entité quand vous utilisez l’instruction (from module import nom_entité). Ex: from module import n as a, m as b, o as c



Travailler avec des modules standard

- la fonction dir() (n'a rien à voir avec la commande dir dans les consoles Windows et Unix), il est capable de révéler tous les noms fourni par un module particulier, si le module à été précédemment importé dans son ensemble(import module).

- La fonction renvoie une liste triée alphabétiquement contenant tous les noms d'entités disponibles dans le module identifiés par un nom passé à la fonction en argument: dir(module)

```
#Exemple  1

import math
print(dir(math))
```

```
#Exemple  2

import math
for name in dir(math):
    print(name, end="\t")
```

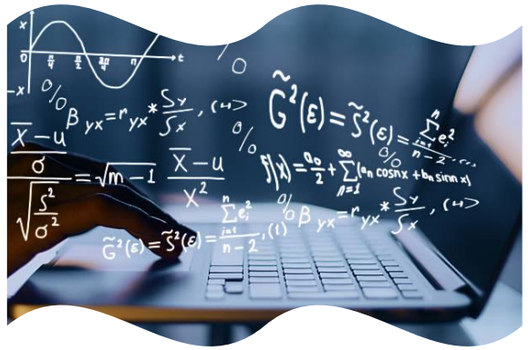
Manipulation des fonctions du module math 1

- Le premier groupe de fonctions math est lié à la trigonométrie: sin(x), cos(x), tan(x)

- Toutes ces fonctions prennent un argument (une mesure d'angle exprimée en radians) et retournent le résultat approprié (attention tan()- tous les arguments ne sont pas acceptés).

- il y a aussi leurs versions inversées: asin(x), acos(x), atan(x). Ces fonctions prennent un argument (attention aux domaines) et renvoient une mesure d'un angle en radians.

- Pour opérer efficacement sur les mesures d'angle, le module math vous fournit les entités suivantes: pi (une constante avec une valeur qui est une approximation de π), radians(x) (une fonction qui convertit x de degrés en radians), degrees(x) (agir dans l'autre sens des radians aux degrés)



Manipulation des fonctions du module math 2

- le module math contient également un ensemble de leurs analogues hyperboliques : sinh(x), cosh(x), tanh(x), asinh(x), acosh(x), atanh(x)

- Un autre groupe de fonctions dans le module math est liées à l'exponentiation: e (constante avec une valeur approximation du nombre d'Euler e), exp(x) (trouver la valeur de ex), log(x) (le logarithme naturel de x), log(x, b) (le logarithme de x à la base b), log10(x) (le logarithme décimal de x), log2(x) (le logarithme binaire de x), pow(x, y) (trouver la valeur de xy)

- Le dernier groupe comprend quelques fonctions à usage général comme: ceil(x) (le plus petit entier supérieur ou égal à x), floor(x) (le plus grand entier inférieur ou égal à x), trunc(x) (la valeur de x tronquée à un entier), factorial(x) (renvoie x! x doit être une intégrale et non un négatif), hypot(x, y) (renvoie la longueur de l'hypoténuse d'un triangle rectangle avec des longueurs de jambe égales à x et y)

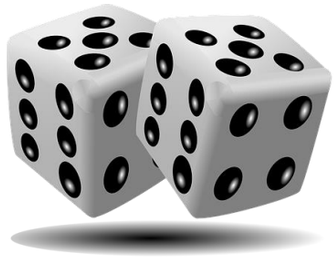
Utilisation du module random

- Le module random, Il fournit certains mécanismes vous permettant de fonctionner avec des nombres pseudo-aléatoires.

- les nombres générés par les modules peuvent sembler aléatoires dans le sens où vous ne pouvez pas prédire leurs valeurs ultérieures, mais n'oubliez pas qu'ils sont tous calculés à l'aide d'algorithmes très raffinés.

- Un générateur de nombres aléatoires prend une valeur appelée graine , la traite comme une valeur d'entrée, calcule un nombre "aléatoire" en fonction de celle-ci et produit une nouvelle valeur de graine.

- Le facteur aléatoire du processus peut être augmenté en définissant la graine avec un nombre pris à partir de l'heure actuelle - cela peut garantir que chaque lancement de programme commencera à partir d'une valeur de graine différente



Manipulation des réels et caractères aléatoire dans les ordinateurs

- La fonction la plus générale nommée random(), produit un nombre flottant x provenant de la plage(0.0, 1.0) - en d'autres termes: (0,0 <= x <1,0).

- La fonction seed() est capable de définir directement la graine du générateur. Nous allons vous montrer deux de ses variantes: seed() (définit la graine avec l'heure actuelle), seed(int_value) (définit la graine avec la valeur entière int_value)

- Si vous voulez des valeurs aléatoires entières, l'une des fonctions suivantes conviendrait mieux: randrange(end), randrange(beg, end), randrange(beg, end, step) les 3 généreront un entier pris (pseudo-aléatoirement) dans la plage, randint(left, right) (elle génère la valeur entière i, qui tombe dans la plage [gauche, droite])

```
#Exemple

from random import random
for i in range(5):
    print(random())ar
```

```
#Exemple

from random import random, seed
seed(0)
for i in range(5):
    print(random())
```

```
#Exemple

from random import randrange, randint
print(randrange(1), end=' ')
print(randrange(0, 1), end=' ')
print(randrange(0, 1, 1), end=' ')
```

Utilisation de l'unicité des nombres

- Les fonctions précédentes ont un inconvénient important - elles peuvent produire des valeurs répétitives même si le nombre d'appels ultérieurs n'est pas supérieur à la largeur de la plage spécifiée.

- il existe une meilleure solution que d'écrire votre propre code pour vérifier l'unicité des nombres "dessinés": choice(sequence) (choisit un élément "aléatoire" dans la séquence d'entrée et le renvoie), sample(sequence, elements_to_choose=1) (choisit certains des éléments d'entrée, renvoyant une liste avec le choix).

```
#Exemple
from random import choice, sample
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(choice(lst))
print(sample(lst, 5))
print(sample(lst, 10))
```

Moyens pour savoir où tu es?

- Imaginez l'environnement de votre programme comme une pyramide composée d'un certain nombre de couches ou de plates-formes.

- certaines de vos actions (ou plutôt celles de votre programme) doivent parcourir un long chemin pour être exécutées avec succès - imaginez que:

  - ✓ votre code veut créer un fichier, donc il invoque une des fonctions de Python;
  - ✓ Python accepte la commande, la réorganise pour répondre aux exigences du système d'exploitation local et l'envoie
  - ✓ le système d'exploitation vérifie si la demande est raisonnable et valide et essaie de créer le fichier;
  - ✓ le matériel , qui est responsable de l'activation des périphériques de stockage (disque dur, périphériques SSD, etc.) pour répondre aux besoins du système d'exploitation.

Moyens d'accéder aux données de la plate-forme (1/2)

- Le module platform vous permet d'accéder aux données de la plate-forme sous-jacente, c'est-à-dire au matériel, au système d'exploitation et aux informations de version de l'interpréteur.

- La fonction platform renvoie simplement une chaîne décrivant l'environnement; ainsi, sa sortie s'adresse plutôt aux humains qu'au traitement automatisé.

- platform(aliased = False, terse = False), aliased (lorsqu'elle est définie sur True ou sur toute valeur non nulle, la fonction peut présenter les noms de couche sous-jacents alternatifs au lieu des noms communs), terse (lorsqu'elle est définie sur True ou toute valeur non nulle, elle peut convaincre la fonction de présenter une forme plus courte du résultat si possible)

```
#Exemple
from platform import platform
print(platform())
print(platform(1))
print(platform(0, 1))
```

Moyens d'accéder aux données de la plate-forme (2/2)

- La fonction machine() renvoie le nom générique du processeur qui exécute votre système d'exploitation avec Python et votre code.
- La fonction processor() renvoie une chaîne remplie du vrai nom du processeur si possible.
- La fonction system() renvoie le nom générique du système d'exploitation sous forme de chaîne.
- La fonction version() renvoie La version du système d'exploitation est fournie sous forme de chaîne.

```
#Exemple
from platform import
machine
print(machine())
```

```
#Exemple
from platform import
processor
print(processor())
```

```
#Exemple
from platform import
system
print(system())
```

```
#Exemple
from platform import
version
print(version())
```

Moyens pour connaitre la version Python

- Si vous avez besoin de savoir quelle version de Python exécute votre code, vous pouvez le vérifier en utilisant un certain nombre de fonctions dédiées - en voici deux:

  ✓ La fonction python_implementation() renvoie une chaîne indiquant l'implémentation Python (attendez-vous Cpython ici, à moins que vous ne décidiez d'utiliser une branche Python non canonique).

  ✓ La fonction python_version_tuple() renvoie un tuple à trois éléments: la majeure partie de la version de Python, la partie mineure et le numéro de niveau du patch

```
#Exemple
from platform import python_implementation,
python_version_tuple
print(python_implementation())
print(python_version_tuple())
```

Indexation du module Python

- Les modules de Python constituent leur propre univers, dans lequel Python lui-même n'est qu'une galaxie, et nous oserions dire que l'exploration des profondeurs de ces modules peut prendre beaucoup plus de temps que de se familiariser avec Python "pur".

- la communauté Python du monde entier crée et maintient des centaines de modules supplémentaires utilisés dans des applications très niches comme la génétique, la psychologie ou même l'astrologie.

- Ces modules ne sont pas (et ne seront pas) distribués avec Python, ou via les canaux officiels, ce qui rend l'univers Python plus large - presque infini.

- Vous pouvez lire sur tous les modules Python standard ici:

Définition d’un package

- L'écriture de vos propres modules ne diffère pas beaucoup de l'écriture de scripts ordinaires. Il y a certains aspects spécifiques que vous devez connaître, mais ce n'est certainement pas sorcier.

  ✓ un module est une sorte de conteneur rempli de fonctions, vous pouvez regrouper autant de fonctions que vous le souhaitez dans un module et le distribuer à travers le monde;

  ✓ Paquet: de la même manière que vous avez précédemment groupé les fonctions, vous voudrez regrouper vos modules dans un paquet qui joue un rôle similaire à un dossier / répertoire dans le monde des fichiers.

Création de votre premier module

- Créer deux fichiers dans le même dossier:

  ✓ Le premier fichier est d'un module C'est vide maintenant nommé le fichier module1.py

  ✓ Le deuxième fichier contient le code utilisant le module1. Son nom est main.py

- Un nouveau sous-dossier nommée (__pycache__) est apparu après l'exécution du fichier main.py qui contient un fichier nommé (plus ou moins) module1.cpython-xy.pyc où x et y sont des chiffres dérivés de votre version de Python (par exemple, ils seront 3 et 8 si vous utilisez Python 3.8).

- Lorsque Python importe un module pour la première fois, il traduit son contenu en une forme quelque peu compilée . Le fichier ne contient pas de code machine - c'est du code interne semi-compilé Python , prêt à être exécuté par l'interpréteur de Python.

#Fichier Module1	#Fichier main Import module1

Utilisation de la variable __name__

- Lorsqu'un module est importé, son contenu est implicitement exécuté par Python. l' initialisation n'a lieu qu'une seule fois, lors de la première importation, de sorte que les affectations effectuées par le module ne sont pas répétées inutilement.

  ✓ lorsque vous exécutez un fichier directement, sa variable __name__ est définie sur __main__

  ✓ lorsqu'un fichier est importé en tant que module, sa variable __name__ est définie sur le nom du fichier (à l'exception de .py)

- Si vous écrivez un module rempli d'un certain nombre de fonctions complexes, vous pouvez utiliser la variable __name__ pour placer une série de tests pour vérifier si les fonctions fonctionnent correctement.

```
#Fichier main
Import module1
```

```
#Fichier Module1
print("J'aime être un module.")
print(__name__)
```

```
#Fichier Module1
if __name__ == "__main__":
    print("Je préfère être un module")
else:
    print("J'aime être un module")
```

Utilisation la variable d’un module

- si vous voulez savoir combien de fois les fonctions ont été appelées, vous avez besoin d'un compteur initialisé à zéro lors de l'importation du module.

- Contrairement à de nombreux autres langages de programmation, Python n'a aucun moyen de vous permettre de cacher ces variables aux yeux des utilisateurs du module. Vous pouvez seulement informer vos utilisateurs qu'il s'agit de votre variable, qu'ils peuvent la lire, mais qu'ils ne doivent en aucun cas la modifier Cela se fait en précédant le nom de la variable par _((un trait de soulignement) ou __(deux traits de soulignement).

```
#Fichier main
Import module1
print(module1.counteur)
```

```
#Fichier Module1
counteur=0
if __name__ == "__main__":
    print("Je préfère être un module")
else:
    print("J'aime être un module")
```

Création de votre premier module avancée

- Maintenant, mettons deux fonctions dans le module - elles évalueront la somme et le produit des nombres collectés dans une liste.

- la ligne commençant par (#!) a plusieurs noms (hashbang, shabang…) ce n'est qu'un commentaire au départ pour python mais pour les systèmes d'exploitation Unix indique au système d'exploitation comment exécuter le contenu du fichier (quel programme doit être lancé pour interpréter le texte)

- une chaîne (peut-être une multiligne) placée avant toute instruction de module doc-string

```
#Fichier main
from module1 import suml,
prodl
l1 = [0 for i in range(5)]
l2 = [1 for i in range(5)]
print(suml(l1))
print(prodl(l2))
```

```
#Fichier Module1
#!/usr/bin/env python3
""" module1.py - un exemple de module Python """
__counteur = 0
#fonction suml
#fonction prodl
if __name__ == "__main__":
    print("Je préfère être un module, mais je peux faire des tests pour
vous")
    l = [i+1 for i in range(5)]
    print(suml(l) == 15)
    print(prodl(l) == 120)
```

```
#fonction suml
def suml(list):
    global __counteur
    __counteur += 1
    sum = 0
    for el in list:
        sum += el
    return sum
```

```
#fonction prodl
def prodl(list):
    global __counteur
    __counteur += 1
    prod = 1
    for el in list:
        prod *= el
    return prod
```

Utilisation de la variable path

- La variable path en fait une liste (accessible via le module sys) stockant tous les emplacements (dossiers / répertoires) qui sont recherchés afin de trouver un module qui a été demandé par l'instruction d'importation. Ex1

- Python est capable de traiter les fichiers zip comme des dossiers ordinaires - cela peut économiser beaucoup de stockage.

- Pour ajouter un dossier contenant le module à la variable(liste) path (il est entièrement modifiable) comme ceci:

path.append('C:\\Users\\user\\py\\modules’)

```
#Exemple 1
from sys import path
for i in path:
    print(i)
```

```
#Fichier main
from sys import path
path.append('.\\modules')
from module1 import suml, prodl
l1 = [0 for i in range(5)]
l2 = [1 for i in range(5)]
print(suml(l1))
print(prodl(l2))
```

Création de votre premier package

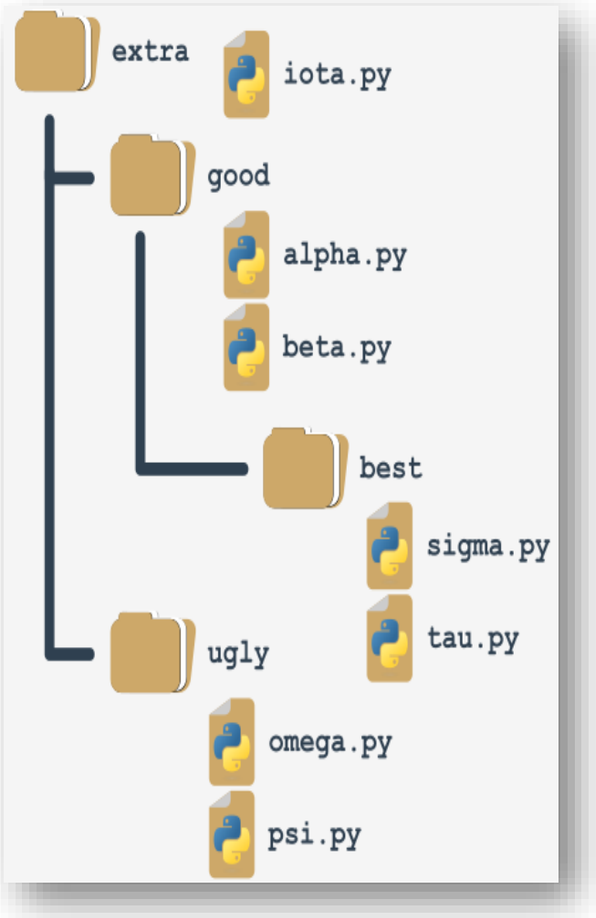
- Imaginez que vous et vos associés écrivez un grand nombre de fonctions Python, Votre équipe décide de regrouper les fonctions dans des modules séparés. (image)

- Une telle structure est presque un package (au sens Python). Il manque le moindre détail pour être à la fois fonctionnel et opérationnel

- Il y a deux questions auxquelles répondre:

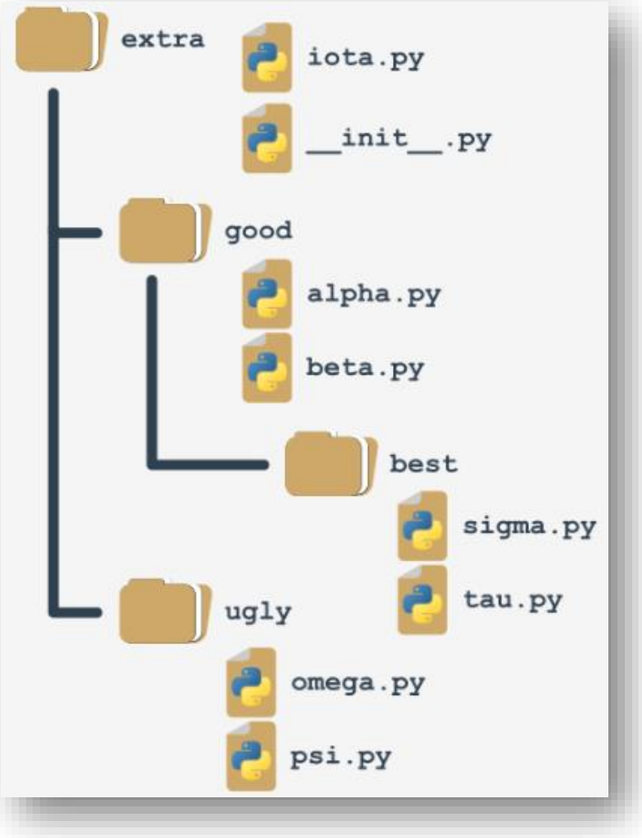
  ✓ comment transformer un tel arbre (en fait, un sous-arbre) en un véritable package Python (en d'autres termes, comment convaincre Python qu'un tel arbre n'est pas seulement un tas de fichiers indésirables, mais un ensemble de modules)?

  ✓ où placez-vous le sous-arbre pour le rendre accessible à Python?



initialisation d'un package

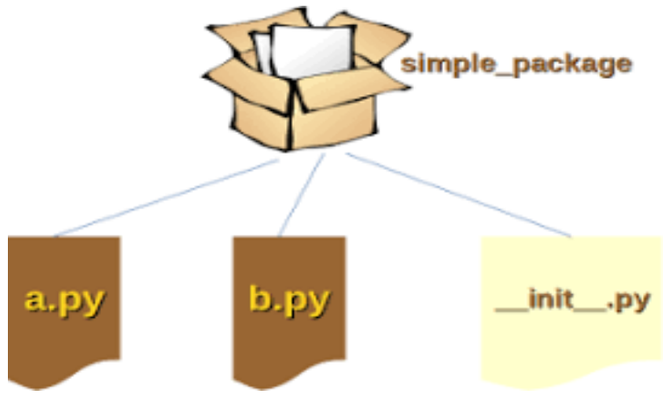
- La première question a une réponse surprenante: les packages, comme les modules, peuvent nécessiter une initialisation .
- L'initialisation d'un module se fait par un code indépendant (ne faisant partie d'aucune fonction) situé à l'intérieur du fichier du module. Comme un package n'est pas un fichier, cette technique est inutile pour initialiser des packages.
- Vous devez utiliser un truc plutôt différent - Python attend qu'il ya un fichier avec un nom unique dans le dossier du package: __init__.py.



Elément important d’un package

- ce n'est pas seulement le dossier racine qui peut contenir le fichier __init__.py - vous pouvez également le placer dans l'un de ses sous-dossiers (sous-packages). Cela peut être utile si certains des sous-ensembles nécessitent un traitement individuel et des types spéciaux d'initialisation.

- Il est maintenant temps de répondre à la deuxième question - la réponse est simple: n'importe où . Vous devez seulement vous assurer que Python connaît l'emplacement du package.



CHAPITRE 2

Manipuler les chaines de caractères

1. Mettre en place un environnement Python
2. Appliquer les bases de la programmation en Python
3. **Appliquer les techniques de programmation avancée en utilisant Python**

Définition de code ASCII (1/2)

- Les ordinateurs stockent les caractères sous forme de nombres . Chaque caractère utilisé par un ordinateur correspond à un numéro unique, et vice versa.

- Certains de ces caractères sont appelés espaces blancs , tandis que d'autres sont appelés caractères de contrôle , car leur objectif est de contrôler les périphériques d'entrée / sortie.

- ASCII (abréviation de American Standard Code for Information Interchange) est le plus largement utilisé, et vous pouvez supposer que presque tous les appareils modernes (comme les ordinateurs, les imprimantes, les téléphones mobiles, les tablettes, etc.) utilisent ce code.

Définition de code ASCII (2/2)

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Définition de point de code / page de codes

- I18N c’est Une forme classique de code ASCII utilise huit bits pour chaque signe . Huit bits signifient 256 caractères différents. Les 128 premiers sont utilisés pour l'alphabet latin standard (à la fois en majuscules et en minuscules).

- Un point de code est un nombre qui fait un caractère . Par exemple, 32 est un point de code qui crée un espace dans le codage ASCII. Nous pouvons dire que le code ASCII standard se compose de 128 points de code.

- Une page de codes est une norme(ISO / IEC 8859-2, etc) d'utilisation des 128 points de code supérieurs pour stocker des caractères nationaux spécifiques . (arabe, grec…)

Définition de Unicode (1/2)

- Il nous arrive souvent d'utiliser les codes de 128 à 255 pour les accents, mais ces codes sont différents d'un pays à l'autre ! Pas pratique pour échanger des documents. Il faut donc trouver un code plus pratique.

- Unicode attribue des caractères uniques (non ambigus) (lettres, traits d'union, idéogrammes, etc.) à plus d'un million de points de code. Les 128 premiers points de code Unicode sont identiques à ASCII et les 256 premiers points de code Unicode sont identiques à la page de codes ISO / IEC 8859-1 (langues d'Europe occidentale).

Définition de Unicode (2/2)

- UCS-4 (Universal Character Set), utilise 32 bits (quatre octets) pour stocker chaque caractère , et le code est juste le numéro unique des points de code Unicode.(taille=4 fois que ASCII).

- UTF-8(format de transformation Unicode) Le concept est très intelligent, utilise autant de bits pour chacun des points de code qu'il en a vraiment besoin pour les représenter.

- Dans UTF-8 tous les caractères ASCII standard occupent 8bits, les caractères non latins occupent 16 bits, Les idéogrammes CJK (Chine-Japon-Corée) occupent 24bits.

- Python 3 prend entièrement en charge Unicode et UTF-8

Nature des chaînes en Python

- Tout d'abord, les chaînes de Python sont des séquences immuables. Ex la fonction len() utilisée pour les chaînes retourne un certain nombre de caractères contenus dans les arguments.

- Chaînes multilignes commence par trois apostrophes(guillemets), pas une. La même apostrophe (guillemets), triplée est utilisée pour y mettre fin.

```
# Example 1
word = 'by'
print(len(word)) # 2

# Example 2
empty = ''
print(len(empty)) # 0

# Example 3
i_am = 'I\'m'
print(len(i_am)) # 3
```

```
# Exemple 4
multiLine = '''Line #1
Line #2'''

print(len(multiLine))
```

Opérations sur les chaînes

- Comme d'autres types de données, les chaînes ont leur propre ensemble d'opérations autorisées (limitées par rapport aux nombres) concaténé(joint) + et répliqué *.
- L'opérateur + utilisé contre deux ou plusieurs chaînes produit une nouvelle chaîne contenant tous les caractères de ses arguments (l'ordre est important)
- l'opérateur * a besoin d'une chaîne et d'un nombre comme arguments (l'ordre n'a pas d'importance)
- les variantes de raccourci des opérateurs ci-dessus s'appliquent également aux chaînes (+=et *=).
- La fonction ord () pour connaître la valeur du point de code ASCII / UNICODE d'un seul caractère spécifique

```
# Example 1

str1 = 'a'

str2 = 'b'

print(str1 + str2) # ab

print(str2 + str1) # ba

print(5 * 'a') # aaaaa

print('b' * 4) # bbbb
```

```
# Exemple 2


ch1 = 'a'

ch2 = ' ' # space

print(ord(ch1)) # 97

print(ord(ch2)) # 32

Print(chr(97)) # a
```

Chaînes utiliser comme séquences

- Les chaînes ne sont pas des listes, mais vous pouvez les traiter comme des listes dans de nombreux cas particuliers.
- l'indexation / itération : Si vous souhaitez accéder à l'un des caractères d'une chaîne ou itérer la chaîne fonctionne aussi.
- Tout ce que vous savez sur les tranches est toujours utilisable.
- L'opérateur in (not in) il vérifie simplement si son argument de gauche (une chaîne) peut être trouvé n'importe où dans le bon argument (une autre chaîne).

```
# Exemple Indexation/iteration

chaine = 'une ligne'
for i in range(len(chaine)):
    print(chaine[i], end=' ')
print()
for car in chaine:
    print(car, end=' ')
```

```
# tranches

alpha = "abdefg"
print(alpha[1:3]) #bd
print(alpha[3:])
print(alpha[3:-2])
print(alpha[::2])
print(alpha[1::2]) #beg
```

```
# Exemple in / not in

alphabet = "chAine"
print("a" not in alphabet)
print("A" in alphabet)
print("ain" in alphabet)
```

Chaînes immuables

- Les chaînes de Python sont immuables. Cela signifie principalement que la similitude des chaînes et des listes est limitée.

- La première différence importante ne vous permet pas d'utiliser l'instruction del pour supprimer des éléments d'une chaîne mai tu peut supprimer la chaîne dans son ensemble.

- Les chaînes Python n'ont pas la méthode append() et insert(). vous ne pouvez pas les développer en aucune façon.

Autres opérations sur les chaînes

- La fonction min()/max() recherche l'élément minimum/maximal de la séquence passée en argument. Il y a une condition - la séquence (chaîne, liste, peu importe) ne peut pas être vide , sinon vous obtiendrez une exception ValueError.

- La méthode index() recherche la séquence depuis le début, afin de trouver le premier élément de la valeur spécifiée dans son argument. L'absence d'élément recherché entraînera une exception ValueError .

- La fonction list() prend son argument (une chaîne) et crée une nouvelle liste contenant tous les caractères de la chaîne, un par élément de liste.

- La méthode count() compte toutes les occurrences de l'élément à l'intérieur de la séquence . L'absence de tels éléments ne pose aucun problème.

```
# Demonstration min() max()
print(min("aAbByYzZ")) #A
print(max("aAbByYzZ")) #z

# Demonstration index()
print("aAbBy".index("b")) #2
print("aAbBy".index("c"))
#ValueError
```

```
 # Demonstration list()
print(list("abcabc"))

# Demonstration count()
print("abcabc".count("b"))
```

Méthodes pour modifier la case

Méthode	Description	Exemple
lower()	crée une copie d'une chaîne source, remplace toutes les lettres majuscules par leurs homologues minuscules et renvoie la chaîne comme résultat.	print("SiGmA=60".lower()) #sigma=60
upper()	crée une copie de la chaîne source, remplace toutes les lettres minuscules par leurs homologues majuscules et renvoie la chaîne comme résultat.	print("SiGmA=60".upper()) #SIGMA=60
capitalize()	si le premier caractère à l'intérieur de la chaîne est une lettre, il sera converti en majuscule puis toutes les lettres restantes de la chaîne seront converties en minuscules	print('aBcD'.capitalize()) #Abcd
title()	remplit une fonction quelque peu similaire - elle change la première lettre de chaque mot en majuscule, transformant tous les autres en minuscule	Ex: print("ta bY.".title()) #Ta By.
swapcase()	crée une nouvelle chaîne en permutant la casse de toutes les lettres de la chaîne source : les caractères en minuscule deviennent en majuscule et vice versa	print("BoNjoUr.".swapcase()) #bOnJOuR.

Méthodes pour vérifier la case le type de données

Méthode	Description	Exemple
islower()	elle accepte uniquement les lettres minuscules.	print("Moo".islower()) #False print("moo".islower()) #True
isspace()	identifie uniquement les espaces blancs - elle ignore tout autre caractère (le résultat est alors False)	print('\n'.isspace()) #True
isupper()	elle se concentre uniquement sur les lettres majuscules	print("Moo".isupper()) #False print("MOO".isupper()) #True
isalnum()	sans paramètre nommée vérifie si la chaîne ne contient que des chiffres ou des caractères alphabétiques (lettres) et renvoie True ou False en fonction du résultat	print('lambda30'.isalnum()) #True print(''.isalnum()) #False | print('a b'.isalnum()) #False
isalpha()	est plus spécialisée - elle ne s'intéresse qu'aux lettres.	print("Moooo".isalpha()) #True | print('Mu40'.isalpha()) #False
isdigit()	ne regarde que les chiffres - tout le reste produit False comme résultat.	print('2018'.isdigit()) #True print("Year2019".isdigit()) #False

Méthodes de changement

Méthode	Description	Exemple
join()	effectue une jointure -elle attend un argument comme une liste; il faut s'assurer que tous les éléments de la liste sont des chaînes – sinon la méthode déclenchera une exception TypeError.	print("-".join(["mic", "pi", "ro"])) #mic-pi-ro
split()	elle fractionne la chaîne et construit une liste de toutes les sous-chaînes détectées (split('-')).	print("phi chi\npsi".split()) #['phi', 'chi', 'psi']
replace()	à deux paramètres renvoie une copie de la chaîne d'origine dans laquelle toutes les occurrences du premier argument ont été remplacées par le deuxième argument.	print("This is it!".replace("is", "are")) #Thare are it!
replace()	à trois paramètres utilise le troisième argument (un nombre) pour limiter le nombre de remplacements.	print("This is it!".replace("is", "are", 1)) #Thare is it!

Méthodes pour la suppression d'une partie

Méthode	Description	Exemple
lstrip()	sans paramètre renvoie une chaîne nouvellement créée formée à partir de la chaîne d'origine en supprimant tous les espaces blancs de tête	print("[" + " ta u ".lstrip() + "]") #[ta u]
	à un paramètre fait la même chose que sa version sans paramètre, mais supprime tous les caractères inscrits dans son argument (une chaîne), pas seulement les espaces	print("www.ben.com".lstrip("w.")) #ben.com
rstrip()	font presque la même chose que lstrips, mais affectent le côté opposé de la chaîne.	
strip()	combine les effets provoqués par rstrip() et lstrip()- elle fait une nouvelle chaîne sans tous les espaces blancs de début et de fin.	

Méthodes de centrage & vérifier le début et fin

Méthode	Description	Exemple
center()	sans paramètre crée une copie de la chaîne d'origine, en essayant de la centrer dans un champ d'une largeur spécifiée. Le centrage se fait en ajoutant des espaces avant et après la chaîne.	print('['+ 'alpha'.center(10) +']’) #[alpha]
	à deux paramètres utilise le caractère du deuxième argument, au lieu d'un espace.	print('['+ 'gam'.center(10, '*') +']’) #[***gam****]
endswith()	vérifie si la chaîne donnée se termine par l'argument spécifié et retourne True ou False , selon le résultat de la vérification.	print('zeta'.endswith("ta")) #True
startswith()	est une réflexion miroir de endswith()- elle vérifie si une chaîne donnée commence par la sous-chaîne spécifiée.	print("omega".startswith("om")) #True

Méthodes de recherche

Méthode	Description	Exemple
find()	est similaire à celle index(), elle recherche une sous-chaîne et retourne l'indice de première occurrence de cette sous-chaîne.	print("Eta".find("ta")) #1
	il ne génère pas d'erreur pour un argument contenant une sous-chaîne inexistante (retourne -1), il ne fonctionne qu'avec des chaînes.	print("Eta".find("mma")) #-1
	une variante à deux paramètres de la méthode find() effectuer la recherche, non pas depuis le début de la chaîne, mais depuis n'importe quelle position.	print('kappa'.find('a', 2)) #4
	Il y a aussi une mutation à trois paramètres de la méthode find() - le troisième argument pointe vers le premier index qui ne sera pas pris en considération pendant la recherche (c'est en fait la limite supérieure de la recherche).	print('kappa'.find('a', 2, 4)) #-1
rfind()	à un, deux et trois paramètres nommées font presque les mêmes choses que leurs homologues (celles dépourvues du préfixe r), mais commencent leurs recherches à la fin de la chaîne , pas au début (d'où le préfixe r , pour la droite).	

Manipulation des comparaison des chaînes

Description	Exemple
Les chaînes de Python peuvent être comparées en utilisant le même ensemble d'opérateurs qui sont utilisés par rapport aux nombres.	==, !=, >, >=, <, <=
N'oubliez pas que Python, il compare simplement les valeurs des points de code , caractère par caractère.	'alpha' == 'alpha’ #True 'alpha' != 'Alpha’ #True
La relation finale entre les chaînes est déterminée en comparant le premier caractère différent dans les deux chaînes (gardez à l'esprit les points de code ASCII / UNICODE à tout moment.)	
La comparaison de chaînes est toujours sensible à la casse (les lettres majuscules sont considérées comme inférieures aux minuscules).	'beta' > 'Beta’ #True
Même si une chaîne ne contient que des chiffres, ce n'est toujours pas un nombre. Il est interprété tel quel, comme toute autre chaîne régulière, et son aspect numérique (potentiel) n'est en aucune façon pris en considération.	'10' > '010’ #True '20' < '8’ #True '10' > '8’ #False
Comparer des chaînes avec des nombres est généralement une mauvaise idée. Les seules comparaisons que vous pouvez effectuer en toute impunité sont celles symbolisées par les opérateurs == et !=. Le premier donne toujours False, tandis que le second produit toujours True.	'10' == 10 #False '10' != 10 #True '10' > 10 #TypeError

Utilisation de Tri

- La comparaison est étroitement liée au tri (ou plutôt, le tri est en fait un cas très sophistiqué de comparaison). C'est une bonne occasion de vous montrer deux façons possibles de trier les listes contenant des chaînes .

- Le premier est implémenté comme une fonction nommée sorted(). prend un argument (une liste) et retourne une nouvelle liste , remplie des éléments de l'argument trié.

- La deuxième méthode affecte la liste elle-même - aucune nouvelle liste n'est créée . La commande est effectuée par la méthode nommée sort().

```
# Demonstration sorted()

firstGreek = ['omega', 'alpha', 'pi', 'gamma']
firstGreek2 = sorted(firstGreek)
# ['alpha', 'gamma', 'omega', 'pi']
```

```
# Demonstration sort()

secondGreek = ['omega', 'alpha', 'pi', 'gamma']
secondGreek.sort()
# ['alpha', 'gamma', 'omega', 'pi']
```

Conversion des nombres & chaines

- La conversion de nombres en chaînes est simple, car elle est toujours possible. Cela se fait par une fonction nommée str().

- La transformation inverse (nombre-chaîne) est possible lorsque et seulement lorsque la chaîne représente un nombre valide. Si la condition n'est pas remplie, attendez-vous à une exception ValueError.

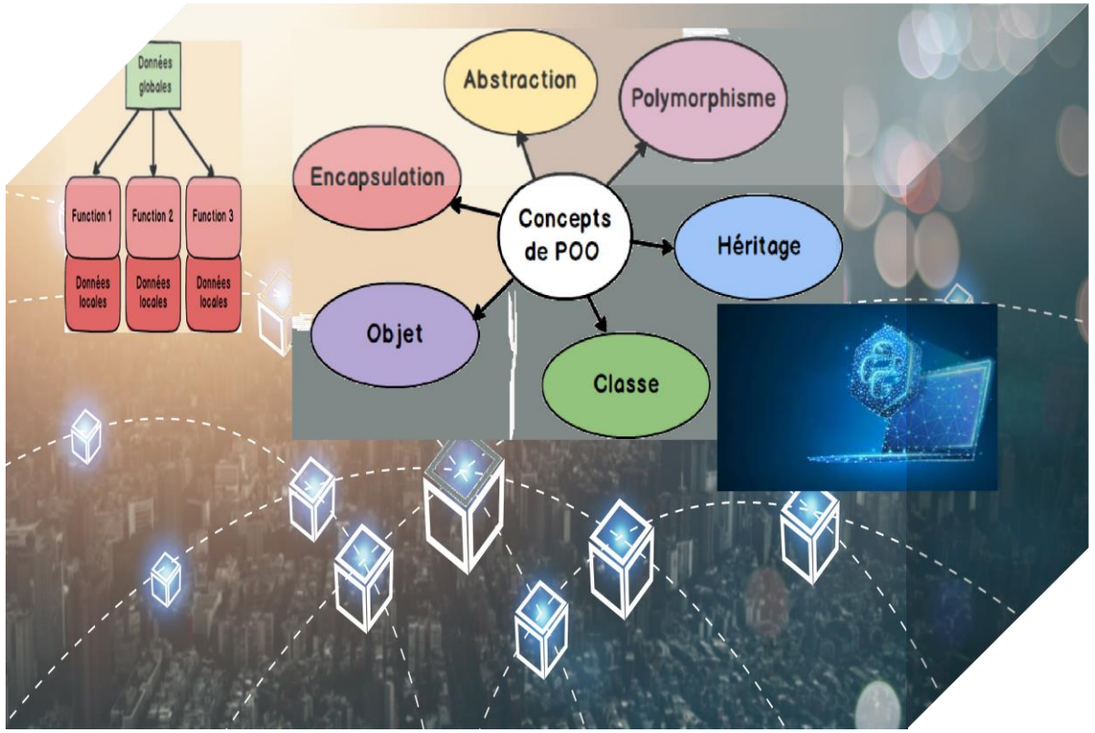
- Utilisez la int()fonction si vous souhaitez obtenir un entier et float()si vous avez besoin d'une valeur à virgule flottante.

```
itg = 13
flt = 1.3
si = str(itg)
sf = str(flt)
print(si + ' ' + sf)  # 13 1.3
```

```
 si = '13'
sf = '1.3'
itg = int(si)
flt = float(sf)
print(itg + flt)  # 14.3
```

PARTIE 2

Coder une solution orientée objet en Python



CHAPITRE 1

Coder une classe

1. Utiliser les packages en Python
2. **Coder une solution orientée objet en Python**
3. Apprendre à utiliser d'autres fonctionnalités Python

Définition programmation procédural et L'approche objet

- Presque tous les programmes et techniques que vous avez utilisés jusqu'à présent relèvent du style procédural de programmation.

- Le style procédural de programmation a été l'approche dominante du développement logiciel pendant des décennies d'informatique, et il est toujours utilisé aujourd'hui.

- L'approche objet est assez jeune (beaucoup plus jeune que l'approche procédurale) et est particulièrement utile lorsqu'elle est appliquée à des projets grands et complexes réalisés par de grandes équipes composées de nombreux développeurs.

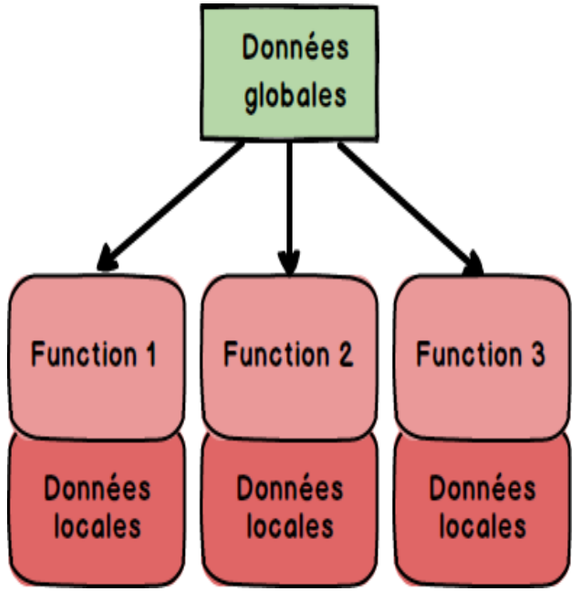
- Python est un outil universel pour la programmation d'objets et de procédures . Il peut être utilisé avec succès dans les deux sphères.

Description de la programmation procédurale

- Dans l'approche procédurale , il est possible de distinguer deux mondes différents et complètement séparés: le monde des données (variables) et le monde du code (modules et fonctions) .

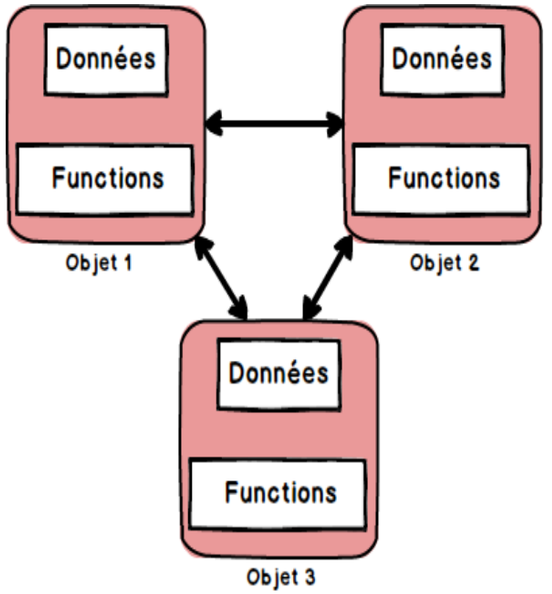
- Les fonctions peuvent utiliser des données, mais pas l'inverse. De plus, les fonctions sont capables d'abuser des données, c'est-à-dire d'utiliser la valeur de manière non autorisée.

- Dans la programmation procédurale, les programmes sont basés sur des fonctions, et les données peuvent être facilement accessibles et modifiables.



Description de l'approche objet 1

- Dans la programmation orientée objet le programme est divisé en parties appelées objets. qui ne sont pas facilement accessibles et modifiables.
- La programmation orientée objet est un concept de programmation qui se concentre sur l'objet plutôt que sur les actions et les données plutôt que sur la logique.
- L' approche objet suggère une manière de penser complètement différente. Les données et le code sont enfermés ensemble dans le même monde, divisés en classes.
- Chaque classe est comme une recette qui peut être utilisée lorsque vous souhaitez créer un objet utile.



Description de l'approche objet 2

- Chaque objet a un ensemble de traits (ils sont appelés propriétés ou attributs) et est capable d'effectuer un ensemble d'activités (appelées méthodes).

- Les failles de la programmation procédurale posent le besoin de la programmation orientée objet.

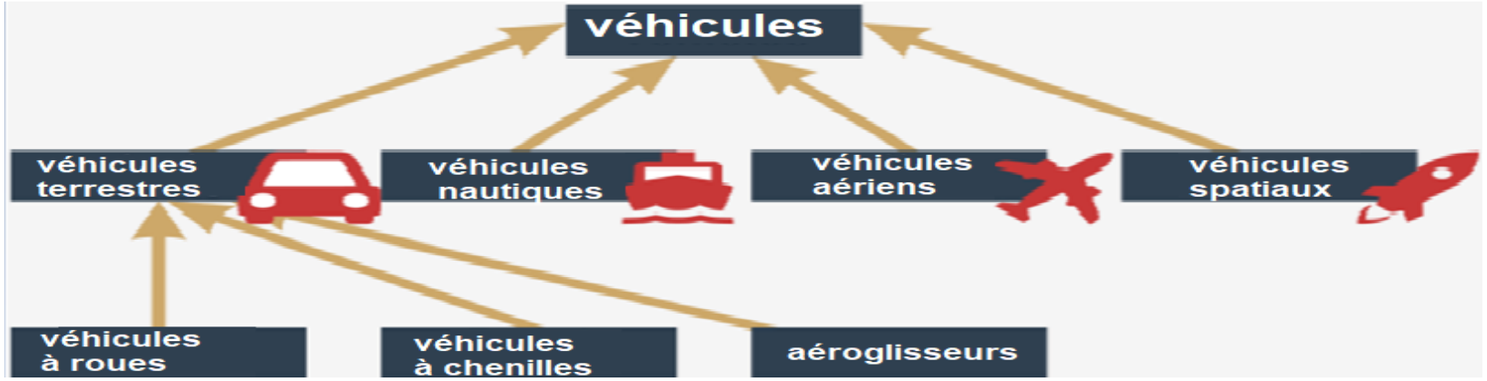
- La programmation orientée objet corrige les défauts du programmation procédurale en introduisant le concept «objet» et «classe». Il améliore la sécurité des données, ainsi que l'initialisation et le nettoyage automatiques des objets. La programmation orientée objet permet de créer plusieurs instances de l'objet sans aucune interférence.

Exemple de l'approche objet =/= procédural

- Prenant l'exemple d'un programme de " gestion d'une école ", dans le procédural on va se poser la question que fait une école ? (inscrire, editer Bulletin, ...) on va diviser notre programme sous forme des sous-programmes ou procédures. Par contre dans l'orientée objet on va se poser la question qu'est ce qu'il y a dans une école (Etudiant, Enseignant, Cours, Exam, ...) donc on va définir des "classes" d'objets, chaque disposera des caractéristiques(attributs) et aura des comportements(méthodes).

Définition de l’hiérarchies des classes

- La classe qui nous intéresse est comme une catégorie , résultant de similitudes précisément définies.
- La classe des véhicules est très large. Nous devons alors définir des classes plus spécialisées . Les classes spécialisées sont les sous - classes . La classe des véhicules sera une superclasse pour tous.
- la hiérarchie croît de haut en bas, comme les racines des arbres, pas les branches . La classe la plus générale et la plus large est toujours en haut (la superclasse) tandis que ses descendants sont situés en dessous (les sous-classes).



Définition d’ un objet 1

- Un « objet » est une représentation d'une chose matérielle ou immatérielle du réel à laquelle on associe des propriétés et des actions. Par exemple : une voiture, une personne, un animal, un nombre ou bien un compte bancaire peuvent être vus comme des objets.

- Une « classe » est un modèle de données définissant la structure commune à tous les objets qui seront créés à partir d'elle. Plus concrètement, nous pouvons percevoir une classe comme un moule grâce auquel nous allons créer autant d'objets de même type et de même structure qu'on le désire.

- Par exemple, pour modéliser n'importe quelle personne, nous pourrions écrire une classe Personne dans laquelle nous définissons les attributs (couleurs des yeux, couleurs des cheveux, taille, sexe...) et méthodes (marcher, courir, manger, boire...) communs à tout être humain.

Définition d’ un objet 2

- Une classe (parmi d'autres définitions) est un ensemble d'objets . Un objet est un être appartenant à une classe.
- Chaque sous-classe est plus spécialisée (ou plus spécifique) que sa superclasse. Inversement, chaque superclasse est plus générale (plus abstraite) que n'importe laquelle de ses sous-classes.
- Tout objet lié à un niveau spécifique d'une hiérarchie de classes hérite de tous les traits (ainsi que les exigences et les qualités) définis dans l'une des superclasses .

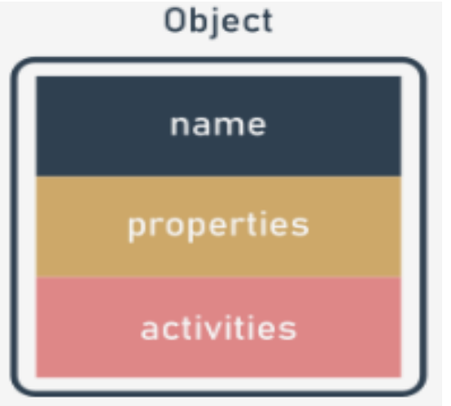
Eléments d’un objet

- La convention de programmation d'objet suppose que chaque objet existant peut être équipé de trois groupes d'attributs :

  ✓ un objet a un nom qui l'identifie de manière unique dans son espace de noms d'origine (bien qu'il puisse également y avoir des objets anonymes)

  ✓ un objet possède un ensemble de propriétés (attributs) individuelles qui le rendent original, unique ou exceptionnel (bien qu'il soit possible que certains objets n'aient aucune propriété)

  ✓ un objet a un ensemble de capacités pour effectuer des activités spécifiques (méthodes) , capable de changer l'objet lui-même, ou certains des autres objets.



Création de votre premier objet

- l'existence d'une classe ne signifie pas que l'un des objets compatibles sera automatiquement créé . La classe elle-même n'est pas en mesure de créer un objet - vous devez le créer vous-même, et Python vous permet de le faire.

- La classe nouvellement définie devient un outil capable de créer de nouveaux objets. L'outil doit être utilisé explicitement, sur demande.

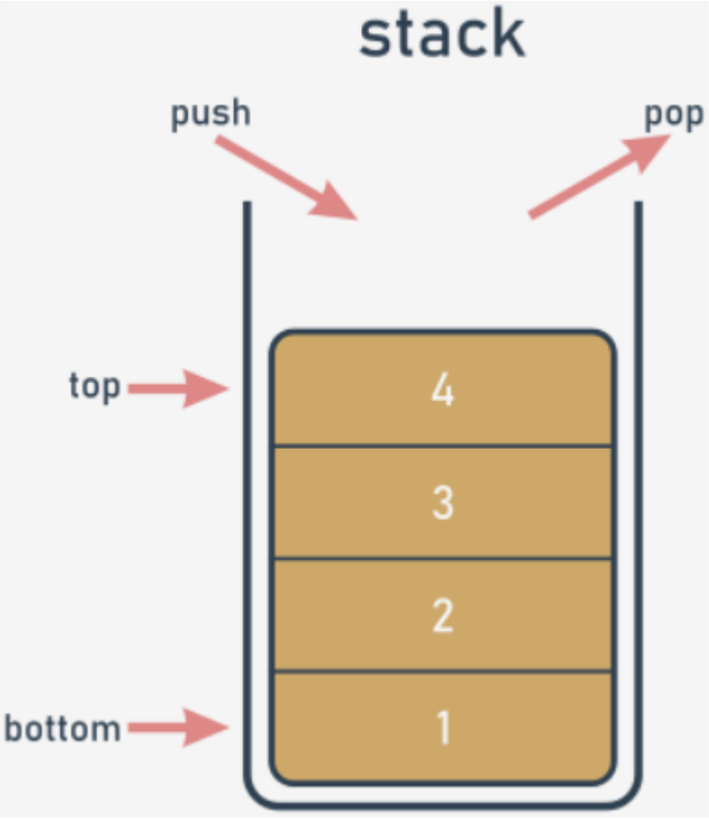
- L'acte de créer un objet de la classe sélectionnée est également appelé une instanciation (lorsque l'objet devient une instance de la classe).

```
# Créer une classe vide

class simpleClasse:
    pass
```

```
#créer un objet de la classe simple classe

monObjet = simpleClasse()
```

Définition d’une pile

- Une pile est une structure développée pour stocker des données d'une manière très spécifique.
- Le nom alternatif pour une pile est LIFO . C'est une abréviation pour une description très claire du comportement de la pile: Last In - First Out . La pièce qui est arrivée en dernier sur la pile sortira en premier.
- Une pile est un objet avec deux opérations élémentaires, appelées conventionnellement push (lorsqu'un nouvel élément est placé en haut) et pop (lorsqu'un élément existant est retiré du haut).



Utilisation de l'approche procédurale pour une pile

- ✓ utiliser une liste pour stocker les valeurs qui arriveront sur la pile. Ex: stack = []
- ✓ définir une fonction qui met une valeur sur la pile .
- ✓ définir une fonction de retirer une valeur de la pile .

- La pile procédurale est prête. Bien sûr, il y a quelques faiblesses, et l'implémentation pourrait être améliorée de plusieurs façons, mais en général la pile est entièrement implémentée, et vous pouvez l'utiliser si vous en avez besoin.

```
# la pile (stack)
stack = []
def push(val):
    stack.append(val)


def pop():
    val = stack[-1]
    del stack[-1]
    return val
```

```
push(3)
push(2)
push(1)


print(pop()) # 1
print(pop()) # 2
print(pop()) # 3
```

Inconvénients de de l'approche procédurale pour une pile

- La pile procédurale est prête. Mais plus vous l'utilisez souvent, plus vous rencontrez d'inconvénients. En voici quelques uns:

  - ✓ la variable essentielle (la liste des piles) est très vulnérable ; n'importe qui peut le modifier de manière incontrôlable, détruisant la pile, etc.

  - ✓ il peut également arriver qu'un jour vous avez besoin de plus d'une pile;

  - ✓ il peut aussi arriver que vous avez besoin non seulement les fonctions push et pop, mais aussi d'autres commodités;

Utilisation de l'approche objet pour une pile 1

- L'approche objective fournit des solutions pour chacun des problèmes ci-dessus.

  ✓ la possibilité de cacher (protéger) les valeurs sélectionnées contre tout accès non autorisé est appelée encapsulation; les valeurs encapsulées ne sont ni accessibles ni modifiables si vous souhaitez les utiliser exclusivement ;

  ✓ lorsque vous avez une classe implémentant tous les comportements de pile nécessaires, vous pouvez produire autant de piles que vous le souhaitez; vous n'avez pas besoin de copier ou de répliquer aucune partie du code;

  ✓ la possibilité d'enrichir la pile de nouvelles fonctions provient de l'héritage; vous pouvez créer une nouvelle classe (une sous-classe) qui hérite de tous les traits existants de la superclasse et en ajoute de nouveaux.

Utilisation de l'approche objet pour une pile 2

- Bien sûr, l'idée principale reste la même. Nous utiliserons une liste comme stockage de la pile. Il suffit de savoir mettre la liste dans la classe.

  ✓ nous voulons que la classe ait une propriété comme stockage de la pile - nous devons "installer" une liste à l'intérieur de chaque objet de la classe (note: chaque objet doit avoir sa propre liste - la liste ne doit pas être partagée entre différentes piles)

  ✓ ensuite, nous voulons que la liste soit cachée aux utilisateurs de la classe.

Définition de constructeur pour une pile

- Il existe un moyen simple pour garantissez-vous qu'une telle activité se déroule à chaque création de la nouvelle pile - vous devez équiper la classe d'une fonction spécifique : il doit être nommé de manière stricte et il est invoqué implicitement, lorsque le nouvel objet est créé.

- Une telle fonction est appelée constructeur , car son objectif général est de construire un nouvel objet. Le constructeur doit effectuer toutes les initialisations nécessaires.

  - ✓ le nom du constructeur est toujours __init__;
  - ✓ il doit avoir au moins un paramètre; le paramètre
  - ✓ est utilisé pour représenter l'objet nouvellement créé
  - ✓ le paramètre obligatoire est généralement nommé self

```
class Stack:    # definir la classe Stack
    def __init__(self):    # definir le constructeur
        print("objet créer avec succès")

stackObject = Stack()    # instantiation d'objet
```

Encapsulation pour une pile

- Toute modification apportée à l'intérieur du constructeur qui modifie l'état du paramètre self reflétera l'objet nouvellement créé.

- nous avons utilisé la notation pointillée , tout comme lors de l'appel de méthodes; il s'agit de la convention générale pour accéder aux propriétés d'un objet.

- nous avons essayé d'accéder à la propriété stackList depuis l'extérieur de la classe immédiatement après la création de l'objet;

- Lorsqu'un composant de classe a un nom commençant par deux traits de soulignement (__), il devient privé - cela signifie qu'il n'est accessible qu'à partir de la classe. (le concept d'encapsulation)

```
class Stack:

    def __init__(self):

        self.stackList = []

        self.__stackList = []

stackObject = Stack()

print(len(stackObject.stackList)) # 0

print(len(stackObject.__stackList)) # AttributeError
```

Définition des méthodes pour une pile

- Il est maintenant temps pour les deux fonctions (méthodes) d'implémenter les opérations push et pop.
- Nous voulons invoquer ces fonctions push et pop. Cela signifie qu'ils doivent tous deux être accessibles à l'utilisateur de chaque classe
- Un tel composant est appelé public , vous ne pouvez donc pas commencer son nom par deux (ou plusieurs) traits de soulignement . Il y a une autre exigence - le nom ne doit pas avoir plus d'un trait de soulignement de fin.
- Toutes les méthodes doivent obligée d'avoir au moins un paramètre (self), qui est utilisé par Python lui - même - vous n'avez aucune influence sur lui.

```
class Stack:
    def __init__(self):
        self.__stackList = []

    def push(self, val):
        self.__stackList.append(val)

    def pop(self):
        val = self.__stackList[-1]
        del self.__stackList[-1]
        return val

stackObject = Stack()

stackObject.push(3)
stackObject.push(2)
stackObject.push(1)

print(stackObject.pop())
print(stackObject.pop())
print(stackObject.pop())
```

Définition des sous-classe pour une pile 1

- Nous ne voulons pas modifier la pile précédemment définie. Nous voulons une nouvelle pile avec de nouvelles capacités.

  ✓ La première étape est simple: il suffit de définir une nouvelle sous-classe pointant vers la classe qui sera utilisée comme super-classe. Ex: class AddingStack(Stack):

  ✓ La sous-classe Il obtient tous les composants définis par sa superclasse.

  ✓ Contrairement à de nombreux autres langages, Python vous oblige à invoquer explicitement le constructeur d'une superclasse. Ex: Stack.__init__(self)

```
class Stack:
    def __init__(self):
        self.__stackList = []

    def push(self, val):
        self.__stackList.append(val)

    def pop(self):
        val = self.__stackList[-1]
        del self.__stackList[-1]
        return val

class AddingStack(Stack):
    def __init__(self):
        Stack.__init__(self)
        self.__sum = 0
```

Définition des sous-classe pour une pile 2

- l'invocation d'une méthode (y compris les constructeurs) de l'extérieur de la classe ne vous oblige jamais à placer l'argument self dans la liste des arguments.
- l'invocation d'une méthode à l'intérieur de la classe nécessite une utilisation explicite de l'argument self, et il doit être placé en premier sur la liste.
- il est généralement recommandé d'invoquer le constructeur de la superclasse avant toute autre initialisation que vous souhaitez effectuer à l'intérieur de la sous-classe.

```
class Stack:
    def __init__(self):
        self.__stackList = []

    def push(self, val):
        self.__stackList.append(val)

    def pop(self):
        val = self.__stackList[-1]
        del self.__stackList[-1]
        return val

class AddingStack(Stack):
    def __init__(self):
        Stack.__init__(self)
        self.__sum = 0
```

Définition des sous-classe pour une pile 3

- Deuxièmement, ajoutons deux méthodes(déjà ces méthodes dans la superclasse). Cela signifie que nous allons changer la fonctionnalité des méthodes, pas leurs noms.

- Nous disons que la méthode push a été remplacée - le même nom que dans la superclasse représente désormais une fonctionnalité différente.

- Nous devons définir une nouvelle méthode. Nous allons le nommer getSum. Sa seule tâche sera de renvoyer la valeur __sum.

```
class Stack:
    def __init__(self):
        self.__stackList = []

    def push(self, val):
        self.__stackList.append(val)

    def pop(self):
        val = self.__stackList[-1]
        del self.__stackList[-1]
        return val
#  sous classe
stackObject = AddingStack()

for i in range(5):
    stackObject.push(i)
print(stackObject.getSum())

for i in range(5):
    print(stackObject.pop()
```

```
class AddingStack(Stack):
    def __init__(self):
        Stack.__init__(self)
        self.__sum = 0

    def getSum(self):
        return self.__sum

    def push(self, val):
        self.__sum += val
        Stack.push(self, val)

    def pop(self):
        val = Stack.pop(self)
        self.__sum -= val
        return val
```

CHAPITRE 2

Intégrer les concepts POO

1. Utiliser les packages en Python
2. **Coder une solution orientée objet en Python**
3. Apprendre à utiliser d'autres fonctionnalités Python

Définition des propriétés d’une class

- En général, une classe peut être équipée de deux types de données différents pour former les propriétés d'une classe.
- Ce type de propriété de classe existe lorsque et seulement lorsqu'elle est explicitement créée et ajoutée à un objet.
- différents objets d'une même classe peuvent posséder différents ensembles de propriétés ;
- il doit y avoir un moyen de vérifier en toute sécurité si un objet spécifique possède la propriété que vous souhaitez utiliser (sauf si vous voulez provoquer une exception - cela vaut toujours la peine d'être considéré)

Utilisation des variables d'instance

- chaque objet possède son propre ensemble de propriétés - ils n'interfèrent pas les uns avec les autres.
- Les objets Python, lorsqu'ils sont créés, sont dotés d'un petit ensemble de propriétés et de méthodes prédéfinies. Chaque objet les a, L'un d'eux est une variable nommée __dict__(c'est un dictionnaire).
- la modification d'une variable d'instance de n'importe quel objet n'a aucun impact sur tous les objets restants. Les variables d'instance sont parfaitement isolées les unes des autres.

```
class ExampleClass:
    def __init__(self, val = 1):
        self.first = val

    def setSecond(self, val):
        self.second = val


exampleObject1 = ExampleClass()
exampleObject2 = ExampleClass(2)

exampleObject2.setSecond(3)

exampleObject3 = ExampleClass(4)
exampleObject3.third = 5

print(exampleObject1.__dict__)
print(exampleObject2.__dict__)
print(exampleObject3.__dict__)
```

Utilisation des variables d'instance privé

- dans les noms de propriété. Nous avons ajouté deux traits de soulignement (__) devant eux. (rend la variable d'instance privée)

- Dans ce ca python modifie l'opération de la manière suivante: il met un nom de la classe avant votre nom de propriété et il met un trait de soulignement supplémentaire au début. Ex: _ExampleClass__first

- Le nom est désormais entièrement accessible depuis l'extérieur de la classe. Ex: print(exampleObject1._ExampleClass__first)

- cette modification ne fonctionnera pas si vous ajoutez une variable d'instance en dehors du code de classe. Ex: print(exampleObject3.__third)

```
class ExampleClass:
    def __init__(self, val = 1):
        self.__first = val

    def setSecond(self, val = 2):
        self.__second = val

exampleObject1 = ExampleClass()
exampleObject2 = ExampleClass(2)

exampleObject2.setSecond(3)

exampleObject3 = ExampleClass(4)
exampleObject3.__third = 5

print(exampleObject1.__dict__)
print(exampleObject2.__dict__)
print(exampleObject3.__dict__)
```

Utilisation des variables de class

- Une variable de classe est une propriété qui existe en une seule copie et est stockée en dehors de tout objet.
- aucune variable d'instance n'existe s'il n'y a pas d'objet dans la classe; une variable de classe existe dans une copie même s'il n'y a pas d'objets dans la classe.
- les variables de classe ne sont pas affichées dans la variable __dict__ d'un objet (c'est naturel car les variables de classe ne font pas partie d'un objet).
- une variable de classe présente toujours la même valeur dans toutes les instances de classe (objets).

```
class ExampleClass:
    counter = 0
    def __init__(self, val = 1):
        self.__first = val
        ExampleClass.counter += 1

exampleObject1 = ExampleClass()
exampleObject2 = ExampleClass(2)
exampleObject3 = ExampleClass(4)

print(exampleObject1.__dict__)
print(exampleObject1.counter)
print(exampleObject2.__dict__)
print(exampleObject2.counter)
print(exampleObject3.__dict__)
print(exampleObject3.counter)
```

Utilisation de la variable de class __dict__

- Nous avons déjà dit que les variables de classe existent même lorsqu'aucune instance de classe (objet) n'a été créée. Cet exemple montre ca.
- la variable __dict__ d'une classe contient beaucoup plus de données que l'homologue de son objet.
- Il y a une différence entre ces variable __dict__, celle de la classe et celle de l'objet.

```
class ExampleClass:
    varia = 1

    def __init__(self, val):
        ExampleClass.varia = val

print(ExampleClass.__dict__)
exampleObject = ExampleClass(2)

print(ExampleClass.__dict__)
print(exampleObject.__dict__)
```

Vérification de l'existence d'un attribut

- contrairement à d'autres langages de programmation, vous ne pouvez pas vous attendre à ce que tous les objets de la même classe aient les mêmes ensembles de propriétés.
- L'instruction try-except vous donne la possibilité d'éviter les problèmes avec des propriétés inexistantes.
- Python fournit une fonction qui est en mesure de vérifier en toute sécurité si un objet / classe contient une propriété spécifiée. Ex: hasattr(exampleObjet,'b').
- N'oubliez pas que la fonction hasattr() peut également fonctionner sur les classes. Vous pouvez l'utiliser pour savoir si une variable de classe est disponible

```
class ExampleClass:
    def __init__(self, val):
        if val % 2 != 0:
            self.a = 1
        else:
            self.b = 1

exampleObject = ExampleClass(1)

print(exampleObject.a)
#print(exampleObject.b)
#try:
   # print(exampleObject.b)
#except AttributeError:
  #  pass

if hasattr(exampleObject, 'b'):
    print(exampleObject.b)
```

Utilisation du nom self dans une class 1

- une méthode est une fonction intégrée à une classe.
- une méthode est obligée d'avoir au moins un paramètre (il n'existe pas de méthodes sans paramètre - une méthode peut être invoquée sans argument, mais pas déclarée sans paramètres).
- Le nom self suggère le but du paramètre - il identifie l'objet pour lequel la méthode est invoquée .

```
class Classy:

    def method(self):
        print("method")

    def m2(self, par):
        print("method:", par)


obj = Classy()

obj.method()

obj.m2(1)
obj.m2(2)
obj.m2(3)
```

Utilisation du nom self dans une class 2

- Dans l'instanciation d'un objet nous avons traité le nom de classe comme une fonction
- Le paramètre self est utilisé pour obtenir l'accès aux variables d'instance et de classe de l'objet.
- Le paramètre self est également utilisé pour appeler d'autres méthodes objet / classe à l'intérieur de la classe.

```
class Classy:
    varia = 2

    def method(self):
        print(self.varia, self.var)
        self.other()

    def other(self):
        print("autre")

obj = Classy()
obj.var = 3
obj.method()
```

Définition de constructeur d’une class 1

- Si vous nommez une méthode comme celle-ci __init__:, ce ne sera pas une méthode régulière - ce sera un constructeur qu’il est invoqué automatiquement et implicitement lorsque l'objet de la classe est instancié.
- Le constructeur est obligé d'avoir le paramètre self.
- Le constructeur peut (mais n'a pas besoin) avoir plus de paramètres que juste self;
- Le constructeur peut être utilisé pour configurer l'objet, (initialiser son état interne, créer des variables d'instance, instancier tout autre objet si leur existence est nécessaire, etc).

```
class Classy:

    def __init__(self, value):
        self.var = value


obj1 = Classy("object")

print(obj1.var)
```

Définition de constructeur d’une class 2

- le constructeur ne peut pas renvoyer une valeur, car il est conçu pour renvoyer un objet nouvellement créé et rien d'autre;
- le constructeur ne peut pas être invoqué directement depuis l'objet ou depuis l'intérieur de la classe (vous pouvez invoquer à partir de n'importe quelle superclasse de l'objet).
- Dans le constructeur on peut définir une valeur d'argument par défaut.
- une méthode dont le nom commence par __est (partiellement) masquée.

```
class Classy:
    def __init__(self, value = None):
        self.var = value

    def visible(self):
        print("visible")

    def __hidden(self):
        print("hidden")

obj1 = Classy("object")
obj2 = Classy()

print(obj1.var); print(obj2.var) ; obj1.visible()

try:
    obj1.__hidden()
except:
    print("failed")

obj1._Classy__hidden()
```

Définition des attributs utiles pour une class 1

- Chaque classe Python et chaque objet Python est pré-équipé d'un ensemble d'attributs utiles qui peuvent être utilisés pour examiner ses capacités (__dict__).
- La propriété __name__ il n'existe que dans les classes renvoi le nom de la classe.
- la fonction type(), qui est capable de trouver une classe qui a été utilisée pour instancier n'importe quel objet.
- __module__ est aussi une chaîne - elle stocke le nom du module qui contient la définition de la classe. (__main__ le fichier en cours d'exécution)

```
class Classy:
    pass

print(Classy.__name__)
print(Classy.__module__)

obj = Classy()

print(type(obj).__name__)
print(obj.__module__)
```

Définition des attributs utiles pour une class 2

- __bases__ est un tuple. Le tuple contient des classes (pas des noms de classe) qui sont des superclasses directes pour la classe.
- seules les classes ont cet attribut , contrairement aux objets.
- une classe sans superclasses explicites pointe vers l'objet (une classe Python prédéfinie) comme ancêtre direct.

```
class SuperOne:
    pass
class SuperTwo:
    pass
class Sub(SuperOne, SuperTwo):
    pass
print(Sub.__bases__)
print(SuperOne.__bases__)
def printBases(cls):
    print('( ', end='')
    for x in cls.__bases__:
        print(x.__name__, end=' ')
    print(')')
printBases(SuperOne)
printBases(SuperTwo)
printBases(Sub)
```

Définition de la réflexion et l'introspection

- l'introspection , qui est la capacité d'un programme à examiner le type ou les propriétés d'un objet au moment de l'exécution;
- la réflexion , qui va plus loin, et est la capacité d'un programme à manipuler les valeurs, les propriétés et / ou les fonctions d'un objet au moment de l'exécution.
- La réflexion et l'introspection permettent au programmeur de faire n'importe quoi avec chaque objet, peu importe d'où il vient.
- getattr(), isinstance(), setattr()

```
class MyClass:
    pass

obj = MyClass()
obj.a = 1 ; obj.b = 2 ; obj.i = 3
obj.ireal = 3.5 ; obj.integer = 4
obj.z = 5

def incIntsI(obj):
    for name in obj.__dict__.keys():
        if name.startswith('i'):
            val = getattr(obj, name)
            if isinstance(val, int):
                setattr(obj, name, val + 1)


print(obj.__dict__)
incIntsI(obj)
print(obj.__dict__)
```

Utilisation de la méthode __str__

- Lorsque Python a besoin que n'importe quelle classe / objet soit présenté comme une chaîne (Ex print(objet)), il essaie d'invoquer une méthode nommée __str__() à partir de l'objet et d'utiliser la chaîne qu'il renvoie.

- La méthode __str__() par défaut renvoie la chaîne précédente - moche et peu informative. Vous pouvez le changer simplement en définissant votre propre méthode du nom .

```
class Star:

    def __init__(self, name, galaxy):
        self.name = name
        self.galaxy = galaxy

    def __str__(self):
        return self.name + ' in ' + self.galaxy


sun = Star("Sun", "Milky Way")
print(sun)

# <__main__.Star object at 0x02FD1EE0>
# Sun in Milky Way
```

Définition de l’héritage

- Le terme héritage il décrit la pratique courante de transmettre différents biens d'une personne à une autre au décès de cette personne. Le terme, lorsqu'il est lié à la programmation informatique, a une signification entièrement différente.

- L'héritage est une pratique courante (en programmation d'objets) qui consiste à transmettre des attributs et des méthodes de la superclasse (définie et existante) à une classe nouvellement créée, appelée la sous-classe .

- l'héritage est un moyen de construire une nouvelle classe, non pas à partir de zéro, mais en utilisant un répertoire de traits déjà défini .

- Grâce à cela, il est possible de construire des classes plus spécialisées (plus concrètes) en utilisant certains ensembles de règles et comportements généraux prédéfinis.

Utilisation de quelques fonction dans l'héritage

- Python offre une fonction qui est capable d'identifier une relation entre deux classes , il peut vérifier si une classe particulière est une sous-classe d'une autre classe. Ex: issubclass(ClassOne, ClassTwo)

- Python offre une fonction qui est capable de vérifier si un objet est une instance d'une classe ou l'une de ses superclasses. Ex: isinstance(objectName, ClassName)

- Il y a aussi l'opérateur python is qui vérifie si deux variables (objectOne et objectTwo) font référence au même objet . Ex: objectOne is objectTwo

```
class SuperC():
    pass

class SubC1(SuperC):
    pass

class SubC2(SubC1):
    pass

mysub1=SubC1()

print(issubclass(SubC1, SuperC))
#True
print(issubclass(SubC1, SubC2))
#False

print(isinstance(mysub1, SuperC))
#True
print(isinstance(mysub1, SubC2))
#False
```

```
class SampleClass:
    def __init__(self, val):
        self.val = val

ob1 = SampleClass(0)
ob2 = SampleClass(2)
ob3 = ob1
ob3.val += 1

print(ob1 is ob2) #False
print(ob2 is ob3) #False
print(ob3 is ob1) #True
print(ob1.val, ob2.val) #1 2
print(ob3.val) # 1

str1 = "Mary "
str2 = "Mary lamb"
str1 += "lamb"

print(str1 == str2) #True
print(str1 is str2) #False
```

Utilisation de la fonction super()

- la fonction super(), qui accède à la superclasse sans avoir besoin de connaître son nom (vous ne devez pas passer l'argument self à la méthode invoquée). Ex: super().__init__(name)

- vous pouvez utiliser ce mécanisme non seulement pour appeler le constructeur de la superclasse, mais également pour accéder à toutes les ressources disponibles à l'intérieur de la superclasse .

```
class SuperC:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return "Mon nom est " + self.name + "."

class Sub(SuperC):
    def __init__(self, name):
        SuperC.__init__(self, name)
        #super().__init__(name)

obj = Sub("Andy")

print(obj) #Mon nom est Andy.
```

Utilisation des variables dans l’héritage

- Les variables de classe qui se trouve dans la SuperClass et sous-classe sont visibles à l'intérieur des objets de la sous-classe.

- Les variables d’instance qui se trouve dans la SuperClass et sous-classe sont visibles à l'intérieur des objets de la sous-classe.

- Lorsque vous essayez d'accéder à l'entité d'un objet, Python essaiera (dans cet ordre): à l'intérieur de l'objet lui-même; puis dans toutes les classes impliquées dans la ligne d'héritage de l'objet de bas en haut; sinon une exception (AttributeError) est levée .

```
class Super:
    supVarC = 1
    def __init__(self):
        self.supVarI = 11

class Sub(Super):
    subVarC = 2
    def __init__(self):
        super().__init__()
        self.subVarI = 22

obj = Sub()

print(obj.subVarC , obj.supVarC) # 2 1

print(obj.subVarI , obj.supVarI) # 22 11
```

Utilisation de l’héritage multiple

- L'héritage multiple se produit lorsqu'une classe a plus d'une superclasse.
- La Subclasse a deux superclasses: SuperA et SuperB. Cela signifie que la Subclasse hérite de tous les produits offerts par SuperA et SuperB .
- L'entité définie ultérieurement (au sens de l'héritage) remplace la même entité définie précédemment.
- Nous pouvons également dire que Python recherche une entité de bas en haut et est pleinement satisfait de la première entité du nom souhaité.
- s'il y a plus d'une classe sur un chemin d'héritage particulier, Python les analyse de gauche à droite.

```
class SuperA:
    varA = 10
    varAA=100
    def funA(self):
        return 11

class SuperB:
    varB = 20
    def funB(self):
        return 21

class Sub(SuperA, SuperB):
    varAA=200

obj = Sub()

print(obj.varA, obj.funA()) # 10 11
print(obj.varB, obj.funB()) # 20 21
print(obj.varAA) # 200
```

Définition du polymorphisme

- la situation dans laquelle la sous-classe est capable de modifier son comportement de super-classe (comme dans l'exemple) est appelée polymorphisme.
- polymorphisme signifie qu'une même classe peut prendre différentes formes selon les redéfinitions effectuées par l'une de ses sous-classes.
- Le comportement de chaque classe peut être modifié à tout moment par n'importe laquelle de ses sous-classes.

```
class One:
    def doit(self):
        print("doit from One")

    def doanything(self):
        self.doit()

class Two(One):
    def doit(self):
        print("doit from Two")

one = One()
two = Two()

one.doanything()
#doit from One
two.doanything()
#doit from Two
```

Utilisation de la coposition

- L'héritage n'est pas le seul moyen de construire des classes adaptables. Vous pouvez atteindre les mêmes objectifs en utilisant une technique nommée composition.
- l'héritage étend les capacités d'une classe en ajoutant de nouveaux composants et en modifiant ceux existants;
- La composition est le processus de composition d'un objet à l'aide d'autres objets différents.
- Les objets de la classe conteneur possèdent donc un attribut qui est un objet de la classe contenue.

```
# Héritage
class Vehicule:
    pass

class Bicycle(Vehicule):
    pass

##############################
# Composition
class Engine:
    pass

class Car:
    def __init__(self):
        self.engine = Engine()
```

Transformation d’héritage simple vs l’héritage multiple

- une seule classe d'héritage est toujours plus simple, plus sûre et plus facile à comprendre et à maintenir;
- l'héritage multiple est toujours risqué, car vous avez beaucoup plus d'occasions de faire une erreur en identifiant ces parties des superclasses qui influenceront efficacement la nouvelle classe;
- l'héritage multiple peut rendre la substitution extrêmement délicate; de plus, l'utilisation de la fonction super() devient ambiguë;
- nous suggérons fortement l'héritage multiple comme dernière solution possible - si vous avez vraiment besoin des nombreuses fonctionnalités différentes offertes par différentes classes, la composition peut être une meilleure alternative.

CHAPITRE 3

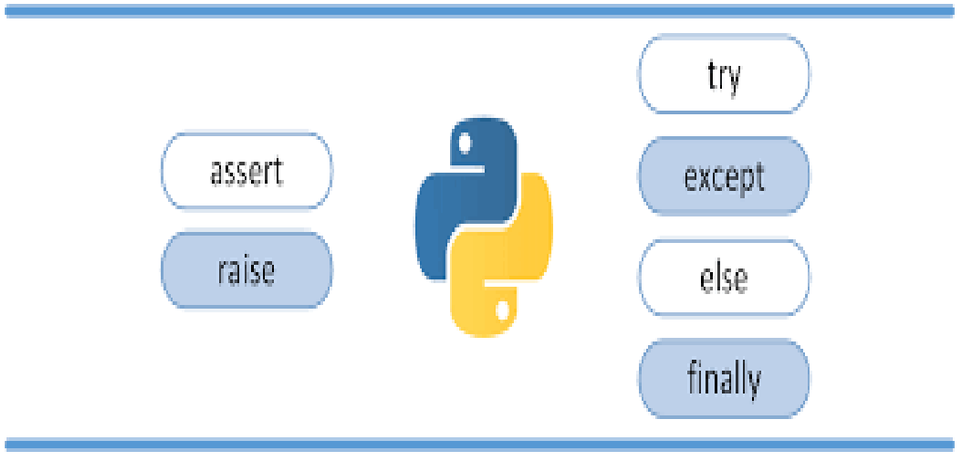
Gérer les Exceptions

1. Utiliser les packages en Python
2. **Coder une solution orientée objet en Python**
3. Apprendre à utiliser d'autres fonctionnalités Python

Définition des exceptions (1/2)

- Chaque fois que votre code essaie de faire quelque chose de mal, inapplicable, Python fait deux choses: il arrête votre programme ou il crée un type spécial de données, appelé exception .

- Nous pouvons dire que Python lève toujours une exception (ou qu'une exception a été déclenchée) quand il n'a aucune idée de quoi faire avec votre code.



Définition des exceptions (2/2)

- si rien ne se passe pour gérer l'exception levée, le programme sera interrompu de force et vous verrez un message d'erreur envoyé à la console par Python

- sinon, si l'exception est prise en charge et gérée correctement, le programme suspendu peut être repris et son exécution peut continuer.

- Python fournit des outils efficaces qui vous permettent d'observer les exceptions, de les identifier et de les gérer efficacement.(Ex : ValueError, ZeroDivisionError, IndexError)

Utilisation du mot – clé try 1

- le mot – clé try commence un bloc du code qui peut ou ne peut pas fonctionner correctement;
- ensuite, Python essaie d'effectuer l'action risquée; s'il échoue, une exception est levée et Python commence à chercher une solution;
- le mot - clé except démarre un morceau de code qui sera exécuté si quelque chose à l'intérieur du bloc try va mal - si une exception est levée à l'intérieur d'un bloc try précédent , il échouera ici , donc le code situé après le mot-clé except devrait fournir une réaction adéquate au levé exception;
- le retour au niveau d'imbrication précédent met fin à la section try-except .

```
#Exemple 1

a = int(input("Entrez le premier numéro: "))

b = int(input("Entrez le deuxième numéro: "))

try:

    print(a / b)

except:

    print("Cette opération ne peut pas être effectuée")

print("Fin.")
```

Utilisation du mot – clé try 2

- il existe une possibilité que plusieurs exceptions sautent dans une branche except:, vous pouvez avoir du mal à comprendre ce qui s'est réellement passé est ca c'est un inconvénient important.

- Techniquement, il existe deux façons de résoudre le problème: construire deux blocs try-except consécutifs , un pour chaque raison d'exception possible (facile, mais provoquera une croissance de code défavorable) ou utiliser une variante plus avancée de l'instruction.

- Le mot-clé else: va permettre d'exécuter une action si aucune erreur ne survient dans le bloc.

- Le mot-clé finally permet d'exécuter du code après un bloc try, quel que soit le résultat de l'exécution dudit bloc.

```
try:
    :
except exc1:
    :
except exc2:
    :
except:
    :
else:
    :
finally:
    :
```

```
#Exemple 1
try:
    x = int(input("Entrer un entier : "))
    y = 1 / x
    print(y)
except ZeroDivisionError:
    print("Vous ne pouvez pas diviser par zéro, désolé.")
except ValueError:
    print("Vous devez entrer une valeur entière.")
except:
    print("quelque chose s'est mal passé ...")
print("FIN.")
```

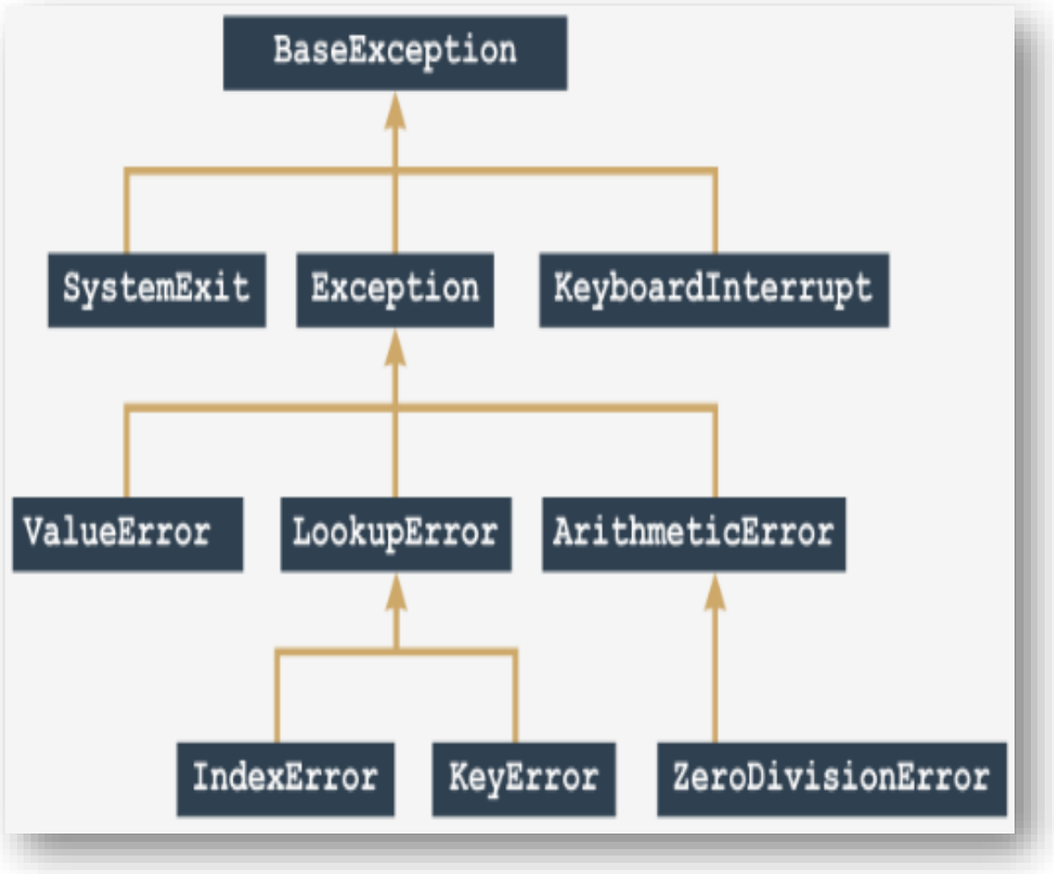
Définition de la hiérarchie d'exception

- Python 3 définit 63 exceptions intégrées , et toutes forment une hiérarchie en forme d'arbre , bien que l'arbre soit un peu bizarre car sa racine est située en haut.

- On peut dire que plus une exception est proche de la racine, plus elle est générale (abstraite) . À leur tour, les exceptions situées aux extrémités des branches (on peut les appeler feuilles) sont concrètes.

- chaque exception levée tombe dans la première branche correspondante ;

- la branche correspondante ne doit pas spécifier exactement la même exception - il suffit que l'exception soit plus générale (plus abstraite) que celle levée. (ArithmeticError)



Définition de l'ordre des exceptions

- l'ordre des exceptions prendre en compte, ne mettez pas d'exceptions plus générales avant des exceptions plus concrètes;

- Si vous souhaitez gérer deux ou plusieurs exceptions de la même manière, Vous devez simplement mettre tous les noms d'exceptions engagés dans une liste séparée par des virgules et ne pas oublier les parenthèses.

- Si une exception est levée à l'intérieur d'une fonction , elle peut être gérée: à l'intérieur de la fonction ou en dehors de la fonction.

```
#Exemple 1
def badFun(n):
    try:
        return 1 / n
    except ArithmeticError:
        print("Problème arithmétique!")
    return None
badFun(0)
print("FIN.")
```

```
#Exemple 2
def badFun(n):
    return 1 / n
try:
    badFun(0)
except ArithmeticError:
    print("Une exception a été levée!")
print("FIN.")
```

Utiliser raise pour lever une exception

- L'instruction raise déclenche l'exception spécifiée nommée exc comme si elle avait été déclenchée de manière normale (naturelle): raise exc

- L'instruction raise vous permet de simuler la levée d'exceptions réelles (par exemple, pour tester votre stratégie de gestion) et gérer partiellement une exception et faire une autre partie du code responsable de l'achèvement de la gestion (séparation des préoccupations).

- L'instruction raise peut également être utilisée de la manière suivante (notez l'absence du nom de l'exception): raise

```
#Exemple 1
def badFun(n):
    raise ZeroDivisionError
try:
    badFun(0)
except ArithmeticError:
    print("Une erreur?")
print("FIN.")
```

```
def badFun(n):
    try:
        return n / 0
    except:
        print("Je l'ai encore
fait!")
        raise
try:
    badFun(0)
except ArithmeticError:
    print("je voix!")
```

Manipulation des assertions

- Les assertions sont un moyen simple de s'assurer, avant de continuer, qu'une condition est respectée. En général, on les utilise dans des blocs try … except.

- Le mot clé assert vous permet de tester si une condition dans votre code retourne True, sinon, le programme déclenchera une AssertionError.

```
#Exemple 1

import math
x = float(input("Enter a number: "))
try:
    assert x >= 0.0
    x = math.sqrt(x)
    print(x)
except AssertionError:
    print("valeur inf à 0")
```

```
#Exemple 2

def badFun(n):
    try:
        return n / 0
    except:
        print("Je l'ai encore fait!")
        raise
try:
    badFun(0)
except ArithmeticError:
    print("je voix!")
```

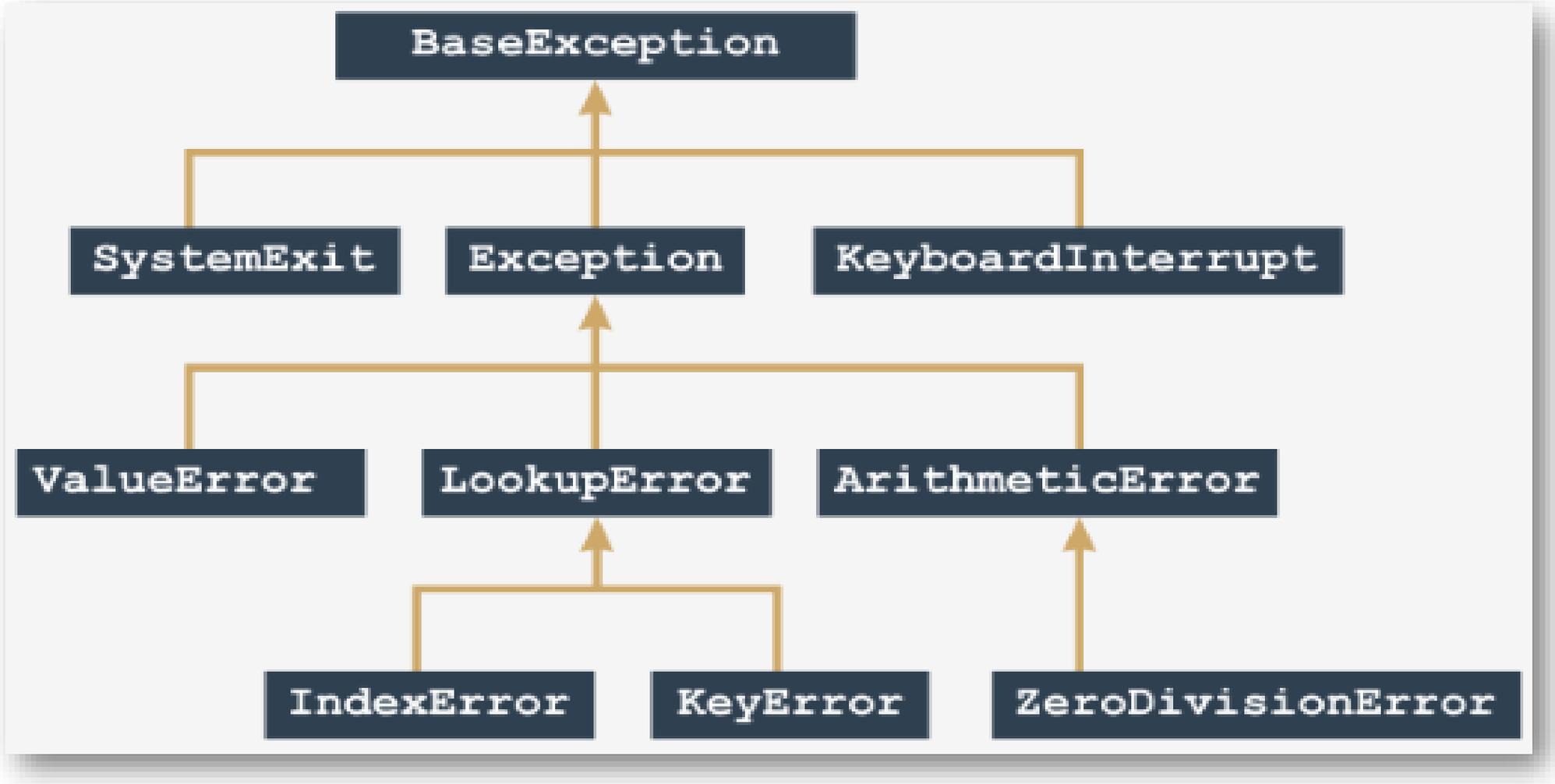
Manipulation de quelques Exceptions intégrées (1/3)

Exceptions	Description	Chemins
ArithmeticError	une exception abstraite comprenant toutes les exceptions causées par des opérations arithmétiques comme la division zéro ou le domaine invalide d'un argument	BaseException ← Exception ← ArithmeticError
AssertionError	une exception de béton soulevée par l'instruction d'assertion lorsque son argument évalue à False, None, 0, ou une chaîne vide	BaseException ← Exception ← AssertionError
BaseException	la plus générale (abstraite) de toutes les exceptions Python - toutes les autres exceptions sont incluses dans celle-ci; on peut dire que les deux branches except suivantes sont équivalentes: except: et BaseException:.	
IndexError	une exception concrète levée lorsque vous essayez d'accéder à un élément de séquence inexistant (par exemple, un élément de liste)	BaseException ← Exception ← LookupError ← IndexError
ImportError	une exception concrète levée lorsqu'une opération d'importation échoue	BaseException ← Exception ← StandardError ← ImportError

Manipulation de quelques Exceptions intégrées (2/3)

Exceptions	Description	Chemins
KeyboardInterrupt	une exception concrète levée lorsque l'utilisateur utilise un raccourci clavier conçu pour mettre fin à l'exécution d'un programme (Ctrl-C dans la plupart des OS); si la gestion de cette exception n'entraîne pas l'arrêt du programme, le programme continue son exécution. Remarque: cette exception n'est pas dérivée de la classe Exception .	BaseException ← KeyboardInterrupt
LookupError	une exception abstraite comprenant toutes les exceptions causées par des erreurs résultant de références invalides à différentes collections (listes, dictionnaires, tuples, etc.)	BaseException ← Exception ← LookupError
MemoryError	une exception concrète levée lorsqu'une opération ne peut pas être terminée en raison d'un manque de mémoire libre	BaseException ← Exception ← MemoryError
OverflowError	une exception concrète levée lorsqu'une opération produit un nombre trop grand pour être stocké avec succès	BaseException ← Exception ← ArithmeticError ← OverflowError
KeyError	une exception concrète levée lorsque vous essayez d'accéder à un élément inexistant d'une collection (par exemple, un élément de dictionnaire)	BaseException ← Exception ← LookupError ← KeyError

Manipulation de quelques Exceptions intégrées (3/3)



Utilisation de la branche else

- La nature objective des exceptions de Python en fait un outil très flexible, capable de s'adapter à des besoins spécifiques, même ceux que vous ne connaissez pas encore.
- La première fonctionnalité que nous voulons discuter ici est une branche supplémentaire possible qui peut être placée à l'intérieur (ou plutôt directement derrière) le bloc try-except - c'est la partie du code commençant par else- tout comme dans l'exemple de l'éditeur.
- la branche else: doit être située après la dernière branche except.
- la branche else: est exécuté quand (et seulement quand) aucune exception n'a été levée à l'intérieur de la pièce try:.

```
def reciprocal(n):
    try:
        n = 1 / n
    except ZeroDivisionError:
        print("Division failed")
        return None
    else:
        print("Tout est bien")
        return n

print(reciprocal(2))
#Tout est bien # 0.5
print(reciprocal(0))
#Division failed  #None
```

Utilisation du bloc finally

- Le bloc try-except peut être étendu d'une autre manière - en ajoutant une partie dirigée par le mot - clé finally (doit être la dernière branche du code conçue pour gérer les exceptions).

- Le bloc finally est toujours exécuté (il finalise l'exécution du bloc try-except, d'où son nom), peu importe, même lors de la levée d'une exception, que cela ait été géré ou non.

```
def reciprocal(n):
    try:
        n = 1 / n
    except ZeroDivisionError:
        print("Division failed")
        n = None
    else:
        print("Tout est bien")
    finally:
        print("goodbye")
        return n

print(reciprocal(2)) #Tout est bien #goodbye #0,5
print(reciprocal(0)) #Division failed #goodbye #None
```

Utilisation des exceptions comme des classes

- les exceptions sont des classes. lorsqu'une exception est levée, un objet de la classe est instancié et passe par tous les niveaux d'exécution du programme, à la recherche de la branche except qui est prête à y faire face.
- le mot-clé as, suivie d'un identifiant est conçu pour intercepter l'objet d'exception afin que vous puissiez analyser sa nature et tirer des conclusions appropriées.
- la portée de l'identifiant couvre sa branche except et ne va pas plus loin.
- Le même message sera imprimé s'il n'y a pas de bloc except approprié dans le code et Python est obligé de le gérer seul.

```
try:
    i = int("Hello!")
except Exception as e:
    print(e)
    print(e.__str__())


#invalid literal for int() with
#base 10: 'Hello!'

#invalid literal for int() with
#base 10: 'Hello!'
```

Arborescence des Exceptions

- Toutes les exceptions Python intégrées forment une hiérarchie de classes.
- Ce programme vide toutes les classes d'exceptions prédéfinies sous la forme d'une impression arborescente.
- Pour chacune des classes rencontrées, effectuez le même ensemble d'opérations:
  - ✓ imprimer son nom, tiré de la propriété __name__;
  - ✓ parcourir la liste des sous-classes fournies par la méthode __subclasses__() et invoquer récursivement la fonction printExcTree(), en incrémentant le niveau d'imbrication respectivement.

```
def printExcTree(thisclass, nest = 0):
    if nest > 1:
        print("    |" * (nest - 1), end="")
    if nest > 0:
        print("    +---", end="")

    print(thisclass.__name__)

    for subclass in
thisclass.__subclasses__():
        printExcTree(subclass, nest + 1)

printExcTree(BaseException)
```

Utilisation de la propriété args

- La classe BaseException introduit une propriété nommée args. Il s'agit d'un tuple conçu pour rassembler tous les arguments passés au constructeur de classe .

- Il est vide si la construction a été invoquée sans aucun argument, ou contient un seul élément lorsque le constructeur obtient un argument (nous ne comptons pas l'argument self ici), et ainsi de suite.

```
def printargs(args):
    lng = len(args)
    if lng == 0: print("")
    elif lng == 1: print(args[0])
    else: print(str(args))
try:
    raise Exception
except Exception as e:
    print(e, e.__str__(), sep=' : ' ,end=' : ')
    printargs(e.args)
try:
    raise Exception("my exception")
except Exception as e:
    print(e, e.__str__(), sep=' : ', end=' : ')
    printargs(e.args)
try:
    raise Exception("my", "exception")
except Exception as e:
    print(e, e.__str__(), sep=' : ', end=' : ')
    printargs(e.args)
```

Création de votre propre structure d'exception 1

- La hiérarchie des exceptions n'est ni fermée ni terminée, et vous pouvez toujours l'étendre si vous voulez ou devez créer votre propre monde rempli de vos propres exceptions.

- Vous pouvez définir vos propres nouvelles exceptions en tant que sous-classes dérivées de celles prédéfinies.

```
class MyZeroDivisionError(ZeroDivisionError):
    pass

def doTheDivision(mine):
    if mine:
        raise MyZeroDivisionError("nouvelles")
    else:
        raise ZeroDivisionError("mauvaise")

for mode in [False, True]:
    try:
        doTheDivision(mode)
    except MyZeroDivisionError:
        print('My division by zero')
    except ZeroDivisionError:
        print('Original division by zero')
```

Création de votre propre structure d'exception 2

- vous voudrez peut-être créer votre propre structure d'exception qui n'ont rien en commun avec toutes les choses familières.
- Vous pouvez commencer à la créer en définissant une exception générale en tant que nouvelle classe de base pour toute autre exception spécialisée.
- Un problème plus spécifique (comme un excès de fromage) peut nécessiter une exception plus spécifique. Il est possible de dériver la nouvelle classe de la classe PizzaError déjà définie.

```
class PizzaError(Exception):
    def __init__(self, pizza, message):
        Exception.__init__(message)
        self.pizza = pizza


class FormageError(PizzaError):
    def __init__(self, pizza, formage, message):
        PizzaError._init__(self, pizza, message)
        self.formage = formage
```

Création de votre propre structure d'exception 3


```
class PizzaError(Exception):
    def __init__(self, pizza, message):
        Exception.__init__(message)
        self.pizza = pizza

class FormageError(PizzaError):
    def __init__(self, pizza, formage, message):
        PizzaError._init__(self, pizza, message)
        self.formage = formage
```

```
def makePizza(pizza, formage):
    if pizza not in ['margherita', 'viande', 'poisson']:
        raise PizzaError(pizza, "pas une telle pizza au menu")
    if formage > 100:
        raise FormageError(pizza, formage, "trop de fromage")
    print("Pizza prête!")

for (pz, ch) in [('poule', 0), ('margherita', 110), ('poisson', 20)]:
    try:
        makePizza(pz, ch)
    except FormageError as tmce:
        print(tmce, ':', tmce.formage)
    except PizzaError as pe:
        print(pe, ':', pe.pizza)
```

PARTIE 3

Apprendre à utiliser d’autres fonctionnalités Python

CHAPITRE 1

Manipuler les Générateurs

1. Utiliser les packages en Python
2. Coder une solution orientée objet en Python
3. **Apprendre à utiliser d'autres fonctionnalités Python**

Définition d’un itérateur 1

- Un itérateur est un objet permettant de parcourir tout autre objet dit « itérable ». Les exemples les plus connus d'objets itérables sont les chaines de caractères, les listes, les fichiers… Bref tout objet que l'on peut parcourir via un index (appelés séquences).

- Un itérateur est une sorte de curseur qui a pour mission de se déplacer dans une séquence d'objets.

- Les itérateurs sont implicitement utilisés chaque fois que nous manipulons des collections de données comme les list, tuple ou string (qui sont des objets dits “itérables”).

Définition d’un itérateur 2

- Le protocole itérateur est un moyen par lequel un objet doit se comporter pour se conformer aux règles imposées par le contexte des instructions for et in.

- Un itérateur doit fournir deux méthodes:

  ✓ __iter__() qui devrait renvoyer l'objet lui - même et qui est invoqué une fois (il est nécessaire que Python démarre avec succès l'itération).

  ✓ __next__() qui est destiné à renvoyer la valeur suivante (première, seconde, etc.) de la série souhaitée - elle sera invoquée par les instructions for / in afin de passer par la prochaine itération; s'il n'y a plus de valeurs à fournir, la méthode doit lever l'exception StopIteration.

Création de l'itérateur Fibonacci 1

- Rappelons-nous que les nombres de Fibonacci (Fib i) sont définis comme suit:
  - ✓ les deux premiers nombres de Fibonacci sont égaux à 1. fib1=fib2=1
  - ✓ tout autre nombre de Fibonacci est la somme des deux précédents (Ex: Fib 3 = 2, Fib 4 = 3, Fib 5 = 5, etc.);
- On va construire une classe capable d'itérer à travers les premières n valeurs des nombres de Fibonacci.

```
class Fib:
    def __init__(self, nn):
        print("__init__")
        self.__n = nn
        self.__i = 0
        self.__p1 = self.__p2 = 1

    def __iter__(self):
        print("__iter__")
        return self
    # def __next__

for i in Fib(5):
        print(i)
```

```
def __next__(self):
        print("__next__")
        self.__i += 1
        if self.__i > self.__n:
            raise StopIteration
        if self.__i in [1, 2]:
            return 1
        ret = self.__p1 + self.__p2
        self.__p1, self.__p2 = self.__p2, ret
        return ret
```

Création de l'itérateur Fibonacci 2

- L'exemple précédent vous montre une solution où l'objet itérateur fait partie d'une classe plus complexe.
- Nous avons intégré l'itérateur Fib dans une autre classe (nous pouvons dire que nous l'avons composé dans la classe Class). Il est instancié avec l'objet de Class.
- L'objet de la classe peut être utilisé comme itérateur quand (et seulement quand) il répond positivement à l'invocation __iter__

```
class Fib:
    def __init__(self, nn):
        self.__n = nn
        self.__i = 0
        self.__p1 = self.__p2 = 1

    def __iter__(self):
        print("Fib iter")
        return self

    def __next__(self):
        self.__i += 1
        if self.__i > self.__n:
            raise StopIteration
        if self.__i in [1, 2]:
            return 1
        ret = self.__p1 + self.__p2
        self.__p1, self.__p2 = self.__p2, ret
        return ret
```

```
class Class:
    def __init__(self, n):
        self.__iter = Fib(n)

    def __iter__(self):
        print("Class iter")
        return self.__iter;


object = Class(8)

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

Création de l’itérateur Fibonacci 3

- Le protocole itérateur n'est pas particulièrement difficile à comprendre et à utiliser, mais il est sûrement que le protocole est inconfortable.
- Le principal inconfort qu'il apporte est la nécessité de sauvegarder l'état de l'itération entre les invocations __iter__ suivantes.
- Par exemple, l'itérateur Fib est obligé de stocker avec précision l'endroit où la dernière invocation a été arrêtée. Cela rend le code plus grand et moins compréhensible.
- C'est pourquoi Python propose une manière beaucoup plus efficace, pratique et élégante d'écrire des itérateurs.

Définition de générateur

- Un générateur Python est un morceau de code spécialisé capable de produire une série de valeurs et de contrôler le processus d'itération. C'est pourquoi les générateurs sont très souvent appelés itérateurs.
- Un générateur renvoie une série de valeurs et, en général, il est (implicitement) invoqué plus d'une fois.
- le générateur range() est invoqué six fois, fournissant cinq valeurs suivantes de zéro à quatre, et signalant enfin que la série est terminée.
- les générateurs ont été créés afin de simplifier la création et l'utilisation d'itérateurs, utilisent un mot magique : yield.

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

Différences entre return & yield

- Le concept est fondamentalement basé sur un mécanisme très spécifique et puissant fourni par le mot – clé yield.
- Vous pouvez considérer le mot - clé yield comme un frère plus intelligent de la déclaration return, avec une différence essentielle.
- Dans l'exemple la boucle for n'a aucune chance de terminer sa première exécution.
- On peut dire qu'une telle fonction n'est pas capable de sauvegarder et de restaurer son état entre les invocations suivantes.
- Cela signifie également qu'une fonction comme celle- ci ne peut pas être utilisée comme générateur.

```
def fun(n):
    for i in range(n):
        return i
print(fun(5)) # 0
```

Utilisation de yield

- Nous avons ajouté yield au lieu de return. Ce petit amendement transforme la fonction en générateur , et l'exécution de l'instruction yield a des effets très intéressants.
- Tout d'abord, il fournit la valeur de l'expression spécifiée après le mot – clé yield, tout comme return, mais ne perd pas l'état de la fonction.
- Il y a une limitation importante: une telle fonction ne devrait pas être invoquée explicitement car - en fait - ce n'est plus une fonction; c'est un objet générateur.
- L'invocation renverra l'identifiant de l'objet , pas la série que nous attendons du générateur.
- Pour les mêmes raisons, la fonction précédente ne peut être invoquée qu'explicitement et ne doit pas être utilisée comme générateur.

```
def fun(n):
    for i in range(n):
        yield I
```

Construction d’un générateur

- si vous avez besoin d'un générateur pour produire les n premières puissances de 2.
- Les générateurs peuvent également être utilisés dans les listes de compréhension.
- La fonction list() peut transformer une série d'appels de générateur ultérieurs en une véritable liste.
- De plus, le contexte créé par l'opérateur in vous permet également d'utiliser un générateur.

```
def powersOf2(n):
    pow = 1
    for i in range(n):
        yield pow
        pow *= 2

for v in powersOf2(6):
    print(v)

t = [x for x in powersOf2(5)]
print(t)

t = list(powersOf2(3))
print(t)

for i in range(20):
    if i in powersOf2(4):
        print(i)
```

Construction de générateur Fibonacci

- Voyons maintenant un générateur de nombres de Fibonacci , et assurons qu'il est bien meilleur que la version objective basée sur l'implémentation directe du protocole de l'itérateur.

```
def Fib(n):
    p = pp = 1
    for i in range(n):
        if i in [0, 1]:
            yield 1
        else:
            n = p + pp
            pp, p = p, n
            yield n

fibs = list(Fib(10))

print(fibs)
```

CHAPITRE 2

Utiliser les fonctions lambda

1. Utiliser les packages en Python
2. Coder une solution orientée objet en Python
3. **Apprendre à utiliser d'autres fonctionnalités Python**

Utilisation de la compréhension des listes

- Le phénomène Python très spécial nommé compréhension de liste - une façon simple et très impressionnante de créer des listes et leur contenu.

- Il y a une syntaxe très intéressante, Il s'agit d'une expression conditionnelle - un moyen de sélectionner l'une des deux valeurs différentes en fonction du résultat d'une expression booléenne.

  Ex: expression1 si condition else expression2

- vous devez garder à l'esprit qu'il ne s'agit pas d'une instruction conditionnelle. De plus, ce n'est pas du tout une instruction. C'est un opérateur.

```
listOne = []
for ex in range(5):
    listOne.append(10 ** ex)

listTwo = [10 ** ex for ex in range(5)]

print(listOne) #[1,10,100,1000,10000]
print(listTwo) #[1,10,100,1000,10000]
##########################################################################
lst1 = []
for x in range(10):
    lst1.append(1 if x % 2 == 0 else 0)

print(lst1) #[1,0,1,0,1,0,1,0,1,0]

lst2 = [1 if x % 2 == 0 else 0 for x in range(10)]
print(lst2) #[1,0,1,0,1,0,1,0,1,0]
```

Transformation de compréhension vs générateur 1

- Un seul changement peut transformer n'importe quelle compréhension en générateur.
- Ce sont les parenthèses . Les crochets font une compréhension, les parenthèses font un générateur.
- Il y a des preuves pour savoir que la deuxième affectation crée un générateur, pas une liste: len(lst) évaluera à 10. Clair et prévisible. len(genr) lèvera une exception

```
lst = [1 if x % 2 == 0 else 0 for x in range(10)]
genr = (1 if x % 2 == 0 else 0 for x in range(10))

for v in lst:
    print(v, end=" ")       # 1 0 1 0 1 0 1 0 1 0
print()

for v in genr:
    print(v, end=" ")       # 1 0 1 0 1 0 1 0 1 0
print()
```

Transformation de compréhension vs générateur 2

- Bien sûr, il n'est pas nécessaire d'enregistrer la liste ou le générateur - vous pouvez les créer exactement à l'endroit où vous avez besoin.
- la même apparence de la sortie ne signifie pas que les deux boucles fonctionnent de la même manière.
- Dans la première boucle, la liste est créée (et itérée) dans son ensemble.
- Dans la deuxième boucle, il n'y a pas de liste du tout - il n'y a que des valeurs ultérieures produites par le générateur, une par une.

```
for v in [1 if x % 2 == 0 else 0 for x in range(10)]:
    print(v, end=" ")
print()

for v in (1 if x % 2 == 0 else 0 for x in range(10)):
    print(v, end=" ")
print()
```

Définition de la fonction lambda

- La fonction lambda est un concept emprunté aux mathématiques, plus précisément à une partie appelée le calcul Lambda , mais ces deux phénomènes ne sont pas les mêmes.
- Les mathématiciens utilisent le calcul Lambda dans de nombreux systèmes formels liés à la logique, à la récursivité ou à la probabilité des théorèmes.
- Les programmeurs utilisent la fonction lambda pour simplifier le code, pour le rendre plus clair et plus facile à comprendre.
- une fonction lambda est une mini-fonctions d'une ligne à la volée qui prend un nombre quelconque d'arguments et retourne la valeur d'une expression unique.

Utilisation de la fonction lambda 1

- Une fonction lambda est une fonction sans nom (vous pouvez également l'appeler une fonction anonyme).
- La déclaration de la fonction lambda ne ressemble en rien à une déclaration de fonction normale.

  Ex: lambda parameters : expression

- Une telle clause renvoie la valeur de l'expression lors de la prise en compte de la valeur courante de l'argument courant lambda.

```
two = lambda : 2
carré = lambda x : x * x
pwr = lambda x, y : x ** y

for a in range(-2, 3):
    print(carré(a), end=" ")
    print(pwr(a, two()))
```

Utilisation de la fonction lambda 2

- La partie la plus intéressante de l'utilisation de lambda apparaît lorsque vous pouvez les utiliser sous leur forme pure - en tant que parties anonymes de code destinées à évaluer un résultat .

- Imaginez que nous ayons besoin d'une fonction qui imprime les valeurs d'une (autre) fonction donnée pour un ensemble d'arguments sélectionnés.

- Ex: f (x) = 2x 2 - 4x + 2

```
def printfunction(args, fun):
    for x in args:
        print('f(', x,')=', fun(x), sep='')

def poly(x):
    return 2 * x**2 - 4 * x + 2

printfunction([x for x in range(-2, 3)], poly)
```

```
def printfunction(args, fun):
        for x in args:
                print('f(', x,')=', fun(x), sep='')
printfunction([x for x in range(-2, 3)], lambda x: 2 * x**2 - 4 * x + 2)
```

Utilisation de la Lambda avec la fonction map()

- Dans le cas le plus simple de tous, la fonction map() prend deux arguments: une fonction et une liste(un tuple ou simplement un générateur). Ex: map(function, list)

- La fonction map() applique la fonction passée par son premier argument à tous les éléments de son second argument et renvoie un itérateur fournissant tous les résultats de fonction suivants.

- Vous pouvez utiliser l'itérateur résultant dans une boucle ou le convertir en liste à l'aide de la fonction list().

```
list1 = [x for x in range(5)]
list2 = list(map(lambda x: 2 ** x, list1))
print(list2)
for x in map(lambda x: x * x, list2):
        print(x, end=' ')
print()
```

Utilisation de la Lambda avec la fonction filter()

- Une autre fonction Python qui peut être considérablement embellie par l'application d'un lambda est filter().
- Il attend le même type d'arguments que map(), mais fait quelque chose de différent.
- il filtre son deuxième argument tout en étant guidé par des directions provenant de la fonction spécifiée comme premier argument.

```
from random import seed, randint

seed()
data = [ randint(-10,10) for x in range(5) ]
filtered = list(filter(lambda x: x > 0 and x % 2 == 0, data))
print(data)
print(filtered)
```

Utilisation des fermetures 1

- la fermeture est une technique qui permet de stocker des valeurs malgré le fait que le contexte dans lequel elles ont été créées n'existe plus.
- une fermeture ou clôture (en anglais : closure) est une fonction accompagnée de son environnement lexical. L'environnement lexical d'une fonction est l'ensemble des variables non locales qu'elle a capturé, soit par valeur, soit par référence.
- La fonction renvoyée lors de l'invocation outer() est une fermeture.
- Une clôture doit être invoquée exactement de la même manière qu'elle a été déclarée.

```
def outer(par):
    loc = par
    def inner():
        return loc
    return inner

var = 1
fun = outer(var)
print(fun())
```

Utilisation des fermetures 2

- Il est tout à fait possible de déclarer une fermeture équipée d'un nombre arbitraire de paramètres.
- vous pouvez créer autant de fermetures que vous le souhaitez en utilisant une seule et même pièce de code .
- la closure est un espace mémoire spécial est créé automatiquement par Python qui va stocker une référence à cette valeur.
- Pour faire simple, disons que les closures en Python sont en lectures seules, à moins qu'on précise explicitement avec nonlocal qu'on va utiliser une variable qui n'est pas locale et qu'on va la modifier.

```
def makeclosure(par):
    loc = par
    def power(p):
        return p ** loc
    return power

fsqr = makeclosure(2)
fcub = makeclosure(3)

for i in range(5):
    print(i, fsqr(i), fcub(i))
```

CHAPITRE 3

Travailler avec des fichiers réels

1. Utiliser les packages en Python
2. Coder une solution orientée objet en Python
3. **Apprendre à utiliser d'autres fonctionnalités Python**

Introduction aux fichiers

- Lorsqu'un programme termine son exécution, toutes les informations stockées dans des variables sont perdues.
- Un moyen de les conserver est de les enregistrer dans un fichier sur disque dur.
- Un fichier est un ensemble d'informations stockées sur une mémoire de masse (disque dur, disquette, bande magnétique, CD-ROM).
- A l'intérieur de celui-ci, ces informations peuvent apparaître sous un format texte qui est lisible par n'importe quel éditeur de texte, dans un format compressé, ou sous un autre format connu par le concepteur du programme(format binaire).

Définition Chemins relatifs et absolus

- Pour décrire l'arborescence d'un système, on a deux possibilités : les chemins absolus et relatifs; Les chemins absolus et relatifs sont donc deux moyens de décrire le chemin menant à des fichiers ou répertoires.

- Un chemin absolu permet d'accéder à un endroit dans le disque quel que soit le répertoire de travail courant. Ex: C:\test\fic.txt ou /home/user/fic.txt

- Quand on décrit la position d'un fichier grâce à un chemin relatif, cela veut dire que l'on tient compte du dossier dans lequel on se trouve actuellement. Ex: ..\fic.txt ou ../fic.txt

Différenciation des chemins entre Windows et linux 1

- Différents systèmes d'exploitation peuvent traiter les fichiers de différentes manières.
- Par exemple, Windows utilise une convention de dénomination différente de celle adoptée dans les systèmes Unix / Linux.
- les systèmes dérivés d'Unix/Linux n'utilisent pas la lettre du lecteur de disque (Ex, C:) et tous les répertoires se développent à partir d'un répertoire racine appelé root (/), tandis que les systèmes Windows reconnaissent le répertoire racine comme c:\.
- les noms de fichiers système Unix/Linux sont sensibles à la casse. Au contraire que Les systèmes Windows.

Windows

```
C:\directory\file
```

Linux

```
/directory/files
```

Différenciation des chemins entre Windows et linux 2

- La différence principale et la plus frappante est que vous devez utiliser deux séparateurs différents pour les noms de répertoire : \ sous Windows et / sous Unix / Linux.

- Python est suffisamment intelligent pour pouvoir convertir des barres obliques en barres obliques inverses chaque fois qu'il découvre que le système d'exploitation l'exige.

- Je vous conseille, que vous soyez sous Windows ou non, d'utiliser le symbole / pour décrire un chemin.

Définition de descripteur ou flux

- Tout programme écrit en Python (et pas seulement en Python) ne communique pas directement avec les fichiers, mais via certaines entités abstraites qui sont nommées différemment dans différents langages ou environnements - les termes les plus utilisés sont des descripteurs ou des flux.

- Le programmeur, ayant un ensemble plus ou moins riche de fonctions / méthodes, est capable d'effectuer certaines opérations sur les Descripteurs.

- L'opération (la première) de connexion du flux avec un fichier est appelée ouverture du fichier , tandis que la déconnexion(la dernière opération) de ce lien est appelée fermeture du fichier.

Définition des Flux de fichiers

- L'ouverture du flux n'est pas seulement associée au fichier, mais doit également déclarer la manière dont le flux sera traité. Cette déclaration est appelée un mode ouvert.

- Si l'ouverture est réussie, le programme sera autorisé à effectuer uniquement les opérations compatibles avec le mode d'ouverture déclaré .

- Deux opérations de base sont effectuées sur le flux:

  ✓ lire à partir du flux: les parties des données sont récupérées du fichier et placées dans une zone mémoire gérée par le programme (par exemple une variable);

  ✓ écrire dans le flux: les parties des données de la mémoire (par exemple, une variable) sont transférées dans le fichier.

Modes de base utilisés pour ouvrir le flux

- Il existe trois modes de base utilisés pour ouvrir le flux:

  - ✓ mode lecture : un flux ouvert dans ce mode autorise uniquement les opérations de lecture ; essayer d'écrire dans le flux provoquera une exception (l'exception est nommée UnsupportedOperation , qui hérite de OSError et ValueError , et provient du module io);

  - ✓ mode écriture : un flux ouvert dans ce mode autorise uniquement les opérations d'écriture ; toute tentative de lecture du flux entraînera l'exception mentionnée ci-dessus;

  - ✓ mode de mise à jour : un flux ouvert dans ce mode permet à la fois les écritures et les lectures .

Description des Flux de fichiers texte / binaire et brute

- Les flux de texte sont structurés en lignes; c'est-à-dire qu'ils contiennent des caractères typographiques (lettres, chiffres, ponctuation, etc.) disposés en lignes. lues / écrites caractère par caractère. Ex: f = open("myfile.txt", "r", encoding="utf-8")

- Les flux binaires ne contiennent pas de texte mais une séquence d'octets de n'importe quelle valeur (un programme exécutable, une image, un clip audio ou vidéo, un fichier de base de données, etc). lues / écrites octet par octet. Ex: f = open("myfile.jpg", "rb")

Utilisation de Clôture des flux

- La dernière opération effectuée sur un stream devrait être la fermeture.
- Cette action est réalisée par une méthode appelée à l'intérieur objet flux ouvert: stream.close().
- la fonction ne renvoie rien mais lève l'exception IOError en cas d'erreur;

```
stream = open("C:\\Desktop\\file.txt", "rt")

    # processing goes here

    stream.close()
```

Sélection des modes texte et binaire

- S'il y a une lettre b à la fin de la chaîne de mode, cela signifie que le flux doit être ouvert en mode binaire .
- Si la chaîne de mode se termine par une lettre, t le flux est ouvert en mode texte .
- Le mode texte est le comportement par défaut supposé quand aucun spécificateur de mode binaire / texte n'est utilisé.

rt	rb	lis
wt	wb	écrire
at	ab	ajouter

Définition Flux de fichiers

- lorsque le flux est ouvert et qu'il est conseillé que les données du fichier associé soient traitées sous forme de texte, il passe en mode texte.

- un fichier texte n'est qu'un fichier binaire. Les bits stockés dans le fichier sont en fait lus par blocs qui sont ensuite interprétés comme des caractères.

- Cette traduction se fait en suivant l'encodage qui a été utilisé pour sauvegarder le fichier. Python travaille par défaut avec l'encodage UTF-8, où chaque caractère par blocs de 8 bits.

Ouverture du flux texte

- L' ouverture du flux est effectuée par une fonction qui peut être invoquée de la manière suivante: stream = open(file, mode = 'r', encoding = None)

  - ✓ la fonction (open) renvoie un objet stream; sinon, une exception
  - ✓ le 1er paramètre (file) de la fonction spécifie le nom du fichier à associer au flux;
  - ✓ le 2ème paramètre (mode) spécifie le mode ouvert utilisé pour le flux;
  - ✓ le 3ème paramètre (encoding) spécifie le type de codage (Ex, UTF-8 lors de l'utilisation de fichiers texte)
  - ✓ l'ouverture doit être la toute première opération effectuée sur le flux.

Modes d’ouverture du flux

Mode	Description
r (lecture)	le fichier associé au flux doit exister et doit être lisible, sinon la fonction open() lève une exception.
w (écriture)	le fichier s'il n'existe pas, il sera créé; s'il existe, il sera tronqué à la longueur de zéro (effacé);
a (ajouter)	le fichier s'il n'existe pas, il sera créé; s'il existe, la tête d'enregistrement virtuelle sera définie à la fin du fichier.

Traitement des fichiers texte

- Si vos fichiers texte contiennent des caractères nationaux non couverts par le jeu de caractères ASCII standard, vous devrez peut-être une étape supplémentaire.

- L'invocation de votre fonction open() peut nécessiter un argument indiquant un codage de texte spécifique. Ex: mon_fichier = open('file.txt', 'rt', encoding='utf-8’)

- La lecture du contenu d'un fichier texte peut être effectuée en utilisant plusieurs méthodes différentes.

```
# ouvrir test.txt en mode lecture, le renvoyant en tant qu'objet fichier

mon_fichier = open("test.txt", "rt", encoding = "utf-8")

print(mon_fichier.read()) # Afficher le contenu du fichier
```

Utilisation de la méthode read().

- La plus élémentaire méthodes de lecture est la fonction read(). Ex: mon_text=mon_fichier.read()

  - ✓ lire un nombre souhaité de caractères (dont un seul) dans le fichier et les renvoyer sous forme de chaîne;
  - ✓ lire tout le contenu du fichier et le renvoyer sous forme de chaîne;
  - ✓ s'il n'y a plus rien à lire (la tête de lecture virtuelle atteint la fin du fichier), la fonction renvoie une chaîne vide.

```
cnt = 0
 s = open('text.txt', "rt")
 ch = s.read(1)
  while ch != '':
      print(ch, end='')
      cnt += 1
      ch = s.read(1)
 s.close()
  print("\n\nCharacters in file:", cnt)
```

Utilisation de la méthode readline().

- Si vous souhaitez traiter le contenu du fichier comme un ensemble de lignes , pas comme un groupe de caractères, la méthode readline() vous aidera.

- La méthode essaie de lire une ligne complète de texte à partir du fichier et la renvoie sous forme de chaîne en cas de succès. Sinon, il renvoie une chaîne vide.

```
ccnt = lcnt = 0
s = open('text.txt', 'rt')
line = s.readline()
while line != '':
    lcnt += 1
    for ch in line:
        print(ch, end='') ; ccnt += 1
    line = s.readline()
s.close()
print("\n\nCharacters in file:", ccnt)
print("Lines in file:     ", lcnt)
```

Utilisation de la méthode readlines().

- La méthode readlines(), lorsqu'elle est invoquée sans arguments, essaie de lire tout le contenu du fichier et retourne une liste de chaînes, un élément par ligne de fichier.

```
ccnt = lcnt = 0
 s = open('text.txt', 'rt')
 lines = s.readlines(20)
 while len(lines) != 0:
    for line in lines:
        lcnt += 1
        for ch in line:
            print(ch, end='') ; ccnt += 1
    lines = s.readlines(10)
 s.close()
 print("\n\nCharacters in file:", ccnt)
 print("Lines in file:     ", lcnt)
```

Utilisation de la la méthode write()

- L'écriture de fichiers texte semble être plus simple, car en fait, il existe une méthode qui peut être utilisée pour effectuer une telle tâche.
- La méthode est nommée write() et n'attend qu'un seul argument - une chaîne qui sera transférée dans un fichier ouvert (écrire un fichier ouvert en mode lecture ne réussira pas).
- Aucun caractère de nouvelle ligne n'est ajouté à l' argument de write(), vous devez donc l'ajouter vous-même si vous souhaitez que le fichier soit rempli de plusieurs lignes.
- La méthode write() n'accepte en paramètre que des chaînes de caractères. Si vous voulez écrire dans votre fichier des nombres, des scores par exemple, il vous faudra les convertir en chaîne avant de les écrire et les convertir en entier après les avoir lus.

```
fo = open('newtext.txt', 'wt')

# créer un nouveau fichier

for i in range(10):

    s = "line #" + str(i+1) + "\n"

    for ch in s:

        fo.write(ch)

fo.close()
```

Utilisation du mot-clé with

- Il existe en Python le mot-clé with qui permet d'ouvrir et de fermer un fichier de manière efficace.
- Si pour une raison ou une autre l'ouverture ou la lecture du fichier conduit à une erreur, l'utilisation de with garantit la bonne fermeture du fichier.
- Le mot-clé with permet de créer un "contexte manager" (gestionnaire de contexte) qui vérifie que le fichier est ouvert et fermé, même si des erreurs se produisent pendant le bloc.
- Il est inutile, par conséquent, de fermer le fichier à la fin du bloc with. Python va le faire tout seul, qu'une exception soit levée ou non. Je vous encourage à utiliser cette syntaxe, elle est plus sûre et plus facile à comprendre.

```
with open('text.txt', 'rt') as s:
    print(s.readlines())strerr(e.errno))
```

Utilisation d’encodage

- En informatique, tout caractère est en fait associé à un identifiant numérique qui est typiquement un nombre entier. Cette correspondance est établie dans une table de caractères.

- Deux fonctions prédéfinies permettent d'effectuer la conversion entre un caractère et son identifiant numérique correspondant. La fonction ord donne l'identifiant numérique correspondant à un caractère tandis que la fonction chr fait l'opération inverse.

- Lorsqu'on lit ou écrit un fichier texte dont l'encodage n'est pas UTF-8, il faut le préciser lors de l'ouverture à l'aide du paramètre nommé encoding. Évidemment, on sera du coup limité par rapport aux caractères que l'on pourra lire et écrire.

```
with open('data.txt', 'w', encoding='ascii')
as f:
    f.write('ن’)
# UnicodeEncodeError
```

Formats de fichiers

- Il existe différents formats standards de stockage de données. Il est recommandé de favoriser ces formats car il existe déjà des modules Python permettant de simplifier leur utilisation. De plus, ces formats sont adoptés par d'autres programmes avec lesquels vous serez peut-être amené à travailler.

- Le format CSV: Le fichier Comma-separated values (CSV) est un format permettant de stocker des tableaux dans un fichier texte. Chaque ligne est représentée par une ligne de texte et chaque colonne est séparée par un séparateur (virgule, point-virgule …). Les champs texte peuvent également être délimités par des guillemets.

- Le format JSON: Le format JavaScript Object Notation (JSON) est issu de la notation des objets dans le langage JavaScript. Il s'agit aujourd'hui d'un format de données très répandu permettant de stocker des données sous une forme structurée. Il ne comporte que des associations clés → valeurs (à l'instar des dictionnaires).

Définition des fichiers binaires 1

- Avant de commencer à parler de fichiers binaires, nous devons vous parler de l'une des classes spécialisées utilisées par Python pour stocker des données amorphes.

- Les données amorphes sont des données qui n'ont pas de forme ou de forme spécifique - ce ne sont qu'une série d'octets.

- Les données amorphes ne peuvent pas être stockées en utilisant l'un des moyens présentés précédemment - ce ne sont ni des chaînes ni des listes.

- Python a plus d'un conteneur spécial capable de gérer les données amorphes- l'un d'eux est un tableau de bytearray de nom de classe spécialisé - c'est un tableau contenant des octets (amorphes).

- Autres :Bytes, encode, struct, marshal, pickle , Json etc…

Définition des fichiers binaires 2

- Un fichier binaire est constitué d'une séquence de bits, organisés en paquets de huit, appelés octets. Un fichier .jpg avec une image est un exemple d'un tel fichier.

- L'avantage des fichiers binaires, par rapport aux fichiers textes est qu'ils sont plus compacts en termes d'espace occupé et également plus rapide à lire et à écrire. Par contre, la difficulté avec ces fichiers est qu'il faut, pour pouvoir les manipuler, connaitre précisément l'organisation des données en leur sein.

- Contrairement aux fichiers texte, les fichiers ne sont pas lisibles par l'homme. Lorsqu'elles sont ouvertes à l'aide d'un éditeur de texte, les données sont méconnaissables.

- Pour écrire du texte dans un fichier binaire, vous devez préfixer la chaine avec le caractère 'b' pour indiquer à python qu'il s'agit d'une chaine binaire, alors convertissez-là vous-même en une séquence d'octets.

Conteneur spécial capable de gérer les données amorphes

Conteneur	Description
Bytes	Les bytes sont des séquences immuables d'octets. Comme beaucoup de protocoles binaires utilisent l'ASCII.
marshal	sauvegarder de manière très simples dans des fichiers des objets python basiques - nombres (int, float, ...), chaînes de caractères, conteneurs (tuples, listes, dictionnaires)-
pickle ou cPickle	sauvegarder de manière très simples dans des fichiers des objets python basiques ou même des objets que vous auriez vous-mêmes définis.
Struct	Ce module effectue des conversions entre les valeurs Python et les structures C représentées comme des objets bytes Python.
Json	est un format très simple d'échange de données inspiré par la syntaxe des objets littéraux de JavaScript.

Manipulation d’objets bytes

- les objets bytes offrent plusieurs méthodes qui ne sont valables que lors de la manipulation de données ASCII et sont étroitement liés aux objets str dans bien d'autres aspects.

- Seuls les caractères ASCII sont autorisés dans les littéraux de bytes (quel que soit l'encodage du code source déclaré). Toutes les valeurs au delà de 127 doivent être entrés dans littéraux de bytes en utilisant une séquence d'échappement appropriée.

- La représentation des bytes utilise le format littéral (b'...') car il est souvent plus utile que par exemple bytes([46, 46, 46]). Vous pouvez toujours convertir un bytes en liste d'entiers en utilisant list(b).

```
chaine= "première "ف
bit =b'fois'
f = open("test.txt","w", encoding ='utf8')
f.write(chaine)  ;  f.write(bit.decode('utf8'))
f.close
#ouverture en lecture
f = open("test.txt","rb")
#lire 10 octet
a = f.read() ; print(a)
#type de a  array de bytes
print(type(a))
#transformer en chaîne de caractères
s = a.decode("utf8")
print(s)  ;  print(type(s)) ; print(bytes(10))
print('codage avec bytes :',bytes(s,'utf8'))
print('codage avec encode :', s.encode('utf8'))
```

Manipulation de Bytearray

- Les objets bytearray sont l'équivalent muable des objets bytes.
- Comme les bytearray sont muables, ils prennent en charge les opérations de séquence muables
- Les bytearrays ressemblent aux listes à bien des égards.
- vous ne devez définir aucun élément de tableau d'octets avec une valeur qui n'est pas un entier. et vous n'êtes pas autorisé à attribuer une valeur qui ne provient pas du plage de 0 à 255 inclus.
- Vous pouvez traiter tous les éléments d'un tableau d'octets comme des valeurs entières.

```
data = bytearray(10)
data1 = bytearray(10)
j=0
for i in 'python':
    data[j] = ord(i)  ;  j += 1

with  open('file.txt', 'wb') as bf:
    bf.write(data)
    bf.close()
with open('file.txt', 'rb') as bf:
    #data1 = bf.read()
    bf.readinto(data1)
    for i in data1:
        print(chr(i),end='')
    bf.close()
```

Manipulation de Module marshal

Partie 1 | Partie 2 | Partie 3

- Ce module contient des fonctions qui peuvent lire et écrire des valeurs Python dans un format binaire.
- Ce module est utilisé pour sérialiser les données; c'est-à-dire convertir des données vers et à partir de chaînes de caractères, afin qu'elles puissent être stockées dans un fichier ou envoyées sur un réseau.
- Ce module est principalement utilisé par Python lui-même pour prendre en charge la lecture / écrire des opérations sur les versions compilées des modules Python (fichiers .pyc)
- Marshal ne garantit pas la compatibilité d'une version de Python à une autre, de sorte que les données sérialisées avec marshal peuvent ne pas être lisibles si vous mettez à niveau votre version de Python.

```
#Exemple 1
import marshal
data = {12: 'douze', 'feep': list
('ciao'), 1.23: 4 + 5j, (1,2,3): u'wer '}
byt1 = marshal.dumps(data)
print(byt1)
redata = marshal.loads(byt1)
print(redata)

with open ('marshal.txt', 'wb') as
bf:
    marshal.dump(data, bf)
    marshal.dump('bonjour', bf)
    bf.close ()

with open ('marshal.txt', 'rb') as bf:
    print(marshal.load(bf))
    print(marshal.load(bf))
    bf.close()
```

```
#Exemple 2
import marshal
script = """ a = 10 ; b = 20
print ('addition = ',a+b)
input("ok") """
with open('prog.py','w') as f:
    f.write(script)
    f.close()
with open('prog.py','r') as f:
    code_str = f.read()
    f.close()
code = compile(code_str,'prog.py',
"exec")
#code = compile(script,'mod', "exec")
with open("prog.pyc","wb") as f:
    marshal.dump(code, f)
    f.close()
with open("prog.pyc","rb") as f:
    data = marshal.load(f)
    exec (data)
```

Manipulation de Module pickle

- Le module implémente des protocoles binaires pour la sérialisation et la désérialisation d'une structure d'objet Python.
- «Pickling» est le processus par lequel une hiérarchie d'objets Python est convertie en un flux d'octets, et «unpickling» est l'opération inverse
- Le module pickle implémente un algorithme pour transformer un objet Python arbitraire en une série d'octets. Ce processus est également appelé sérialisation de l'objet. Le flux d'octets représentant l'objet peut alors être transmis ou stocké, puis reconstruit pour créer un nouvel objet avec les mêmes caractéristiques.

```
#Exemple 1
import pickle
data = {12: 'douze', 'feep': list
('ciao'), 1.23: 4 + 5j, (1,2,3): u'wer '}
byt1 = pickle.dumps(data)
print(byt1)
redata = pickle.loads(byt1)
print(redata)

with open ('pickle.txt', 'wb') as bf:
    pickle.dump(data, bf)
    pickle.dump('bonjour', bf)
    bf.close ()

with open ('pickle.txt', 'rb') as bf:
    print(pickle.load(bf))
    print(pickle.load(bf))
    bf.close()
```

```
#Exemple 2
import pickle
class Person():
    def __init__(self, name):
        self.name = name
    def __str__(self):
        return "Mon nom est " + self.name + "."
nom1 = Person('Omar')
nom2 = Person('Ahmed')
print(nom1) ; print(nom2)

with open('pickle.txt', 'wb') as f1:
    pickle.dump(nom1, f1)
    pickle.dump(nom2, f1)
    f1.close()

with open('pickle.txt', 'rb') as f1:
    print(pickle.load(f1))
    print(pickle.load(f1))
    f1.close()
```

Manipulation de Module struct

- Le module struct est utilisé pour convertir les types de données natifs de Python en chaîne d'octets et vice versa. Nous n'avons pas à l'installer. C'est un module intégré disponible dans Python3 .

- Le module struct est lié aux langages C. Nous devons connaître les notations utilisées en C pour représenter différents types de données pour travailler avec le module struct. (i:int, c:char, s:string, f:float, d:double, etc..)

- La méthode struct.pack () est utilisée pour convertir les types de données en octets. Il prend plusieurs arguments basés sur la première chaîne. Ex: struct.pack ('iif 3s', 1, 2, 3.5, b'abc ')

```
#Exemple struct
import struct
l1=[2,500,b'test',3.64,bytes('fin','utf8')]
strformat='ii4sf3s'
print(bytes('test','utf8'))
paquet = struct.pack(strformat,*l1)
print(paquet)
with open('struct.txt', 'wb') as f:
    f.write(paquet)  ;    f.close()
with open('struct.txt', 'rb') as f:
    paquet = f.read()
    data =
struct.unpack(strformat,paquet)
    f.close()
data = list(data)  ;  print(data)
```

Manipulation de Module json

- JSON (JavaScript Object Notation), est un format très simple d'échange de données inspiré par la syntaxe des objets littéraux de JavaScript.
- json fournit une API familière aux utilisateurs des modules marshal et pickle de la bibliothèque standard.

```
#Exemple json
import json
prof ={
    "nom": "ben",
    "prenom": "yahia",
    "hobbies": ["lecture", "sport", None],
    "age": 40,
    "enfant": [
        {"nom": "omar",
            "age": 6,
            "etudiant":True},
        {"nom": "ahmed",
            "age": 3,
            "etudiant":False}
        ]
    }
strjson = json.dumps(prof)
print('donnée en json \n',strjson)

data = json.loads(strjson)
print('donnée en dict \n',data)
with open('json.json' , 'w') as f:
    #f.write(strjson)
    json.dump(prof, f)
    f.close()
with open('json.json' , 'r') as f:
    #strjson = f.read()
    #data = json.loads(strjson)
    data = json.load(f)
    f.close()
print("donnée d'un fichier json \n",data)
```

Manipulation de Module CSV

- Le format CSV (Comma Separated Values, valeurs séparées par des virgules) est le format le plus commun dans l'importation et l'exportation de feuilles de calculs et de bases de données.
- Le module csv implémente des classes pour lire et écrire des données tabulaires au format CSV.
- Il vous permet de dire « écris ces données dans le format préféré par Excel » ou « lis les données de ce fichier généré par Excel », sans connaître les détails précis du format CSV utilisé par Excel.

```
#Exemple csv
import csv
l1=[]
strcsv = [
    ['N°','Nom','Prénom','Ville'],
    [1,'daif','fatima','casa'],
    [2,'lahlou','amine','fes'],
    [3,'najem','karim','Ville'],
    ]
```

```
with open('test.csv','w',newline='') as f:
    datacsv = csv.writer(f,delimiter=';')
    datacsv.writerows(strcsv)
    f.close()

with open('test.csv','r') as f:
    data = csv.reader(f,delimiter=';')
    for ligne in data:
        l1.append(ligne)
    f.close()
for ligne in l1:
    for ele in ligne:
        print(ele,end='\t')
    print()
```

Manipulation de XML

- XML, ou Extensible Markup Language, est un langage de balisage couramment utilisé pour structurer, stocker et transférer des données entre des systèmes.

- XML est extrêmement utile pour garder une trace de petites et moyennes quantités de données sans avoir besoin d'un backbone basé sur SQL.

```
#Exemple xml
import xml.etree.ElementTree as xml
root = xml.Element('Clients')

clt1 = xml.Element('client')
clt1.set('Code_Client ','124')
root.append(clt1)
nom1 = xml.SubElement(clt1,'nom')
nom1.text = "benyahia"
prenom1 = xml.SubElement(clt1,'prenom')
prenom1.text = "omar"

clt2 = xml.Element('client')
clt2.set('Code_Client ','342')
root.append(clt2)
nom2 = xml.SubElement(clt2,'nom')
nom2.text = "daif"
prenom2 = xml.SubElement(clt2,'prenom')
prenom2.text = "karima"
```

```
comment = xml.Comment('commentaire')
#ce commentaire sera ajouté à l'élément parent
root.append(comment)
arbre = xml.ElementTree(root) ; print(xml.dump(root))

with open('test.xml' , 'wb') as f:
    arbre.write(f)
with open('test.xml' , 'rb') as f:
    dataxml = xml.parse(f)
root = dataxml.getroot()

print('les clients sont :',end='\t')
for clt in root.findall('client'):
    print(clt.get('Code_Client'),end='\t')
    print(clt.find('nom').text)

for child in root:
    print(child.tag,child.attrib)
    for ele in child:
        print(ele.text)
```

CHAPITRE 4

Ecrire des scripts d'automatisation avec Python

1. Utiliser les packages en Python
2. Coder une solution orientée objet en Python
3. **Apprendre à utiliser d'autres fonctionnalités Python**

Introduction (1/2)

- L’ordinateur est un outil qui nous facilite la vie. Le système d’opération effectue une grosse partie du travail: il nous permet de contrôler la souris et le clavier, affiche des informations sur l’écran, nous donne la possibilité de créer des fichiers et de les lires, etc.

- Les logiciels qu’on installe en plus, comme la suite Microsoft Office nous facilitent encore plus la vie. On peut aisément formater un rapport, créer des tableaux, envoyer des emails, etc. Malgré tous ces outils, on se retrouve souvent à entreprendre des tâches répétitives.

- On va par exemple copier le contenu de dizaines de fichiers un à un dans un tableau Excel, télécharger quotidiennement sur un site internet les mêmes informations, etc.

Introduction (2/2)

- Lorsqu'on travaille beaucoup sur ordinateur, on finit souvent par vouloir automatiser certaines tâches : par exemple, effectuer une recherche et un remplacement sur un grand nombre de fichiers de texte ; ou renommer et réorganiser des photos d'une manière un peu compliquée. Pour vous, ce peut être créer une petite base de données, une application graphique ou un simple jeu.

- Python est un langage de programmation avec lequel il est facile d'écrire des scripts. Ces scripts peuvent permettre d'automatiser des tâches répétées fréquemment à la main (envoyer des e-mails, analyser des fichiers, etc.) ou de développer des analyses complexes (remplacer une macro Excel).

Définition des scripts en python

- Un script est un programme relativement simple contenu dans un fichier texte. On peut écrire un script qui va:

✓analyser les fichiers d'un dossier et de ses sous-dossiers, ✓zipper des fichiers, ✓supprimer des fichiers, ✓analyser le contenu d'un ou plusieurs fichier(s), ✓modifier le contenu d'un fichier texte, ✓convertir le format d'un fichier (image, vidéo, etc.),	✓gérer l'exécution d'autres programmes, ✓générer des fichiers PDF, ✓récupérer des données sur internet, ✓envoyer des emails, ✓etc.

- Comme on peut le voir, toutes ces tâches peuvent être réalisées manuellement. Mais on peut aussi toutes les programmer en Python!

Eléments du contenu de chapitre

- Dans ce chapitre en va écrire quelques scripts python simples qu'on peut utiliser dans notre vie quotidienne pour automatiser certaines tâches:

  - ✓ organiser les fichiers du répertoire : ce code organisera votre dossier de téléchargement en dossiers catégorisés.
  - ✓ Convertir les images au format PNG : ce script permet converti les images JPG (ou tout autre format valide) en images PNG
  - ✓ Utiliser des paquets Python pour le web scraping: script permet de scraping une site web pour récupérer les données
  - ✓ Utiliser des paquets Python pour l' Automation Excel : script permet de créer un rapport après avoir configuré notre Python Excel Automation.

Script permet d’organiser les fichiers d’un répertoire

- on remarque que le dossier « téléchargements » était vraiment mal organisé. Des fichiers txt, pdf, films, photos, etc.. Et vu le nombre de fichiers qu’on a, un rangement manuel n’est pas envisageable surtout quand nous savons qu’un petit script python peut faire ça en en quelque secondes.

- Notre premier objectif est d’écrire un script pour automatiser le déplacement des fichiers (écrire un script pour déplacer automatiquement nos fichiers téléchargés du dossier Téléchargements vers différents dossiers en fonction de leur type de fichier).

- Par exemple, si nous téléchargeons une image, elle sera automatiquement déplacée vers le dossier images, si nous téléchargeons un document pdf, il sera déplacé vers le dossier documents, et ainsi de suite.

Description du script d’organisation

- Accédez maintenant à votre dossier Téléchargements et créez un fichier appelé organiser_dossiers.py. Dans ce script :

  - ✓ on va utiliser deux librairies ou modules si vous voulez, le premier c’est le module os et le deuxième c’est Le module shutil .

  - ✓ Le module os fournit des fonctions pour interagir avec le système d'exploitation et fournit un moyen portable d'utiliser les fonctionnalités dépendant du système d'exploitation. L'interaction avec le système de fichiers en est un exemple.

  - ✓ Le module shutil est à nouveau un module python qui offre un certain nombre d'opérations de haut niveau sur les fichiers et la collection de fichiers. Il nous fournit des fonctions qui prennent en charge la copie, le déplacement et la suppression de fichiers et de répertoires.

Guide de script d’organisastion

- Dans ce script On va créer une classe Organise qui contient :
- L’importations des modules os et shutil
- Création du constructeur qui permet de Créer une liste de toutes les chemins des dossiers à créer
- Création de la méthode create_dossier qui permet de créer des dossiers pour chaque type de fichier
- Création de la méthode deplcer_fichiers qui permet de parcourir les fichiers qui sont dans le dossier du répertoire actuel et dispatché sur les dossiers déjà créer
- Création du programme principale

```
import os
import shutil

class Organise():

    # constructeur __init__
    # méthode create_dossier
    # méthode deplcer_fichiers

if __name__ == "__main__":
    print("Organisation de vos fichiers intro [images -
medias - exécutable - archive - documents -]")
    objet1 = Organise()
    objet1.create_dossier()
    objet1.deplcer_fichiers()
```

Création du constructeur pour le script d'organisation


```
def __init__(self):
    self.get_dir = os.getcwd() #Avoir le chemin du "dossier actuel"
    # Déterminer les chemins de dossier de chaque type de fichier
    self.documents = self.get_dir + '\\textes' # dossier pour les fichier de type document
    self.images = self.get_dir + '\\images' # dossier pour les fichier de type image
    self.medias = self.get_dir + '\\medias' # dossier pour les fichier de type medias
    self.logiciels = self.get_dir + '\\logiciels' # dossier pour les fichier de type logiciels
    self.archives = self.get_dir + '\\archives' # dossier pour les fichier de type archives
    #Créer une liste de toutes les chemins à créer
    self.list_chemin = [self.documents, self.images, self.medias, self.logiciels, self.archives]
```

Création de la méthode create_dossier pour script d'organisation

-
```
def create_dossier(self):
        """ Création des dossiers pour chaque type de fichier """
        for dossier in self.list_chemin:
            if not os.path.exists(dossier):
                os.mkdir(dossier)
```

Création de la méthode deplcer_fichiers pour le script d'organisation


```
def deplcer_fichiers(self):
        """ parcourir les fichiers qui sont dans le dossier du répertoire actuel """
        for filename in os.listdir(self.get_dir):
            # Vérifiez si les fichiers sont des documents
            # et vous pouvez ajouter plus d'extensions
            if filename.lower().endswith((".txt", ".doc", ".docx", ".pdf", ".xlsx")):
                shutil.move(filename,self.documents)
            # Vérifiez si les fichiers sont des images
            # et vous pouvez ajouter plus d'extensions
            if filename.lower().endswith((".png", ".jpg", ".jpeg", ".gif", ".bmp", ".pbm", ".pnm")):
                shutil.move(filename, self.images)
```

```
 # Vérifiez si les fichiers sont des medias
            # et vous pouvez ajouter plus d'extensions
            if filename.lower().endswith((".mp4", ".3gp", ".webm", ".mp3")):
                shutil.move(filename,self.medias)
            # Vérifiez si les fichiers sont des logiciels
            # et vous pouvez ajouter plus d'extensions
            if filename.lower().endswith((".exe", ".msi", ".deb")):
                shutil.move(filename,self.logiciels)
            # Vérifiez si les fichiers sont des archives
            # et vous pouvez ajouter plus d'extensions
            if filename.lower().endswith((".rar", ".zip", "..gz", ".tar")):
                shutil.move(filename,self.archives)
```

Script rapide pour convertir images au format PNG

- Un script Python qui convertit les images JPG (ou tout autre format valide) en images PNG
- L'idée est de donner un répertoire cible où vont vivre les images au format JPG. Le programme convertira toutes les images de ce répertoire au format PNG.
- Ensuite, le programme enregistrera les images converties dans un répertoire de destination. Nous donnerons les deux noms de répertoire comme arguments lors de l'exécution du programme. Le programme récupérera les noms de répertoire et effectuera la conversion.
- Supposons que nous ayons des images JPG ou autre format dans un répertoire nommé images. Nous voulons convertir ces images en PNG puis les enregistrer dans un répertoire nommé imagesPNG.

Description du script de conversion

- Accédez maintenant à votre dossier et créez un fichier appelé convert_images.py. Dans ce script :

  ✓ nous avons besoin de certains modules à importer (os et Pillow)

  ✓ Le module os fournit des fonctions pour interagir avec le système d'exploitation et fournit un moyen portable d'utiliser les fonctionnalités dépendant du système d'exploitation. L'interaction avec le système de fichiers en est un exemple.

  ✓ Le module Pillow pour traiter les images. Pillowest un module Python tiers pour traiter les images. Nous devons donc l'installer. Nous pouvons utiliser pip. (pip install Pillow)

Guide de script de conversion

Partie 1 | Partie 2 | Partie 3

- Dans ce script On va créer une classe Convert qui contient :
- L'importations des modules os et PIL(pillow)
- Création du constructeur qui permet de récuperer les dossiers source et destinataire
- Création de la méthode creat_des qui permet de créer le dossier destinataire des images png
- Création de la méthode convert_to_png qui permet de parcourir les fichiers qui sont dans le dossier du répertoire actuel et converti en images png puis les mettre dans le dossier PNG déjà créer.
- Création du programme principale

```
import os
import shutil

class Organise():

    # constructeur __init__
    # méthode creat_des
    # méthode convert_to_png

if __name__ == "__main__":
    print("Conversation des images non png vers image PNG")
    objet1 = Convert()
    objet1.creat_des()
    objet1.convert_to_png()
```

Les Méthodes pour le script de conversion


```
# Constructeur
   def __init__(self):
       self.source = os.getcwd()
       self.destinataire = os.getcwd() +'\\PNG'

   # méthode pour créer le dossier destinataire
   def creat_des(self):
       if not os.path.exists(self.destinataire):
           os.mkdir(self.destinataire)
```

```
# méthode pour converti les images en images png
   def convert_to_png(self):
       """ parcourir les fichiers qui sont
       dans le dossier du répertoire actuel pour le convertie vers png"""
       images = os.listdir(self.source)
       for image in images:
           # Vérifiez si les fichiers sont des images
           # et vous pouvez ajouter plus d'extensions
           if image.lower().endswith((".png",".jpg", ".jpeg", ".bmp")):
               # ouvrir l'image pour la traiter
               img = Image.open(self.source + "\\" + image)
               # Récupérer le nom d'image pour l'utiliser après
               nom_fichier = os.path.splitext(image)[0]
               # sauvegarder l'image avec la nouvelle extension
               img.save(self.destinataire + '\\' + nom_fichier +' .png', 'png')
```

Définition de web scraping 1

- Le web scraping est une technique d'extraction du contenu de sites Web, via un script ou un programme, dans le but de le transformer pour permettre son utilisation dans un autre contexte

- Les termes web scraping sont utilisés pour différentes méthodes de collecte d'informations et de données essentielles sur Internet. Il est également appelé extraction de données Web, grattage d'écran ou récolte Web.

- Il y a plusieurs façons de le faire: Manuellement - vous accédez au site Web et vérifiez ce dont vous avez besoin et Automatique - utilisez les outils nécessaires pour configurer ce dont vous avez besoin et laissez les outils travailler pour vous.

Définition de web scraping 2

- Le Web scraping est une méthode automatisée utilisée pour extraire de grandes quantités de données de sites Web. Les données sur les sites Web ne sont pas structurées.

- Le Web scraping permet de collecter ces données non structurées et de les stocker sous une forme structurée.

- Il existe différentes manières de scraper des sites Web tels que des services en ligne, des API ou l'écriture de votre propre code.

- Pour le web scraping, il y a plusieurs bibliothèques qui peuvent être utilisées, notamment : Beautiful Soup, Requests, Scrapy, Selenium et etc.

Domaines d’application du web scraping

- Le web scraping peut être utilisé à des fins diverses. En dehors de l’indexation par les moteurs de recherche, il est notamment utilisé pour :

  - ✓ créer des bases de données de contact,
  - ✓ surveiller et comparer les prix des offres en ligne,
  - ✓ rassembler des données de différentes sources en ligne,
  - ✓ assurer un suivi de la présence et de la réputation en ligne,
  - ✓ collecter des données financières, météorologiques et autres,
  - ✓ surveiller les modifications apportées aux contenus web,
  - ✓ collecter des données pour la recherche,
  - ✓ miner des données.

Aperçu du web scraping

- Risques juridiques du web scraping : Si les informations scrapées sont des données permettant une identification personnelle, l'enregistrement et l'analyse sans autorisation de la personne concernée représentent une violation des dispositions applicables en matière de protection des données. Le fait de scraper des profils Facebook pour collecter des données à caractère personnel est par exemple interdit.

- Limites techniques du web scraping : Le standard robots.txt s'est établi afin de limiter les accès des scrapers : dans ce cadre, l'exploitant du site Internet place un fichier texte intitulé robots.txt dans le répertoire principal du site Internet. Ce fichier définit à l'aide d'entrées spécifiques quels scrapers ou bots peuvent accéder à quels domaines du site Internet. Les entrées du fichier robots.txt s'appliquent toujours à un domaine entier.

- Les API une alternative au web scraping : de nombreux exploitants de sites Internet mettent à disposition les données dans un format structuré, lisible par machine. Pour accéder aux données, on utilise alors des interfaces de programmation spéciales, appelées Application Programming Interfaces (API).

Comment extraire les données d'un site Web ?

- Pour extraire des données à l'aide du web scraping avec python, vous devez suivre ces étapes de base :

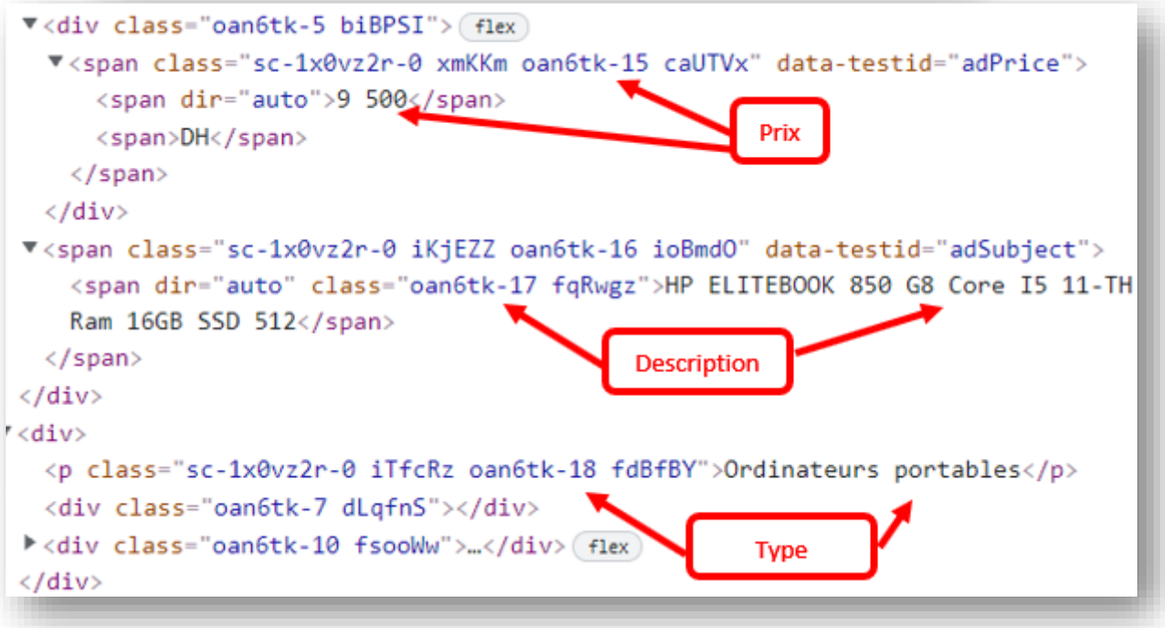
1. Trouvez l'URL que vous souhaitez récupérer
2. Inspection de la page
3. Trouvez les données que vous souhaitez extraire
4. Ecrire le code
5. Exécutez le code et extrayez les données
6. Stocker les données dans le format requis

grattage du site Web avito.ma 1

Partie 1 | Partie 2 | Partie 3

- Étape 1 : Trouvez l'URL que vous souhaitez récupérer Pour cet exemple, nous allons gratter le site Web de avito pour extraire le prix, le nom et l'évaluation des ordinateurs portables. L'URL de cette page est https://www.avito.ma/fr/maroc/ordinateurs_portables-%C3%A0_vendre

- Étape 2 : Inspection de la page Les données sont généralement imbriquées dans des balises. Nous inspectons donc la page pour voir sous quelle balise les données que nous voulons récupérer sont imbriquées. Pour inspecter la page, faites un clic droit sur l'élément et cliquez sur « Inspecter ».



- Étape 3 : Trouvez les données que vous souhaitez extraire Extrayons respectivement le prix, le nom et l'évaluation qui se trouvent dans la balise « div ».

grattage du site Web avito.ma 2

- Étape 4 : écrivez le code Tout d'abord, créons un fichier Python. Ex: pc_avito.py

  - ✓ Installer les bibliothèques nécessaires(requests et bs4) : pip install requests et bs4(beautifulsoup4)
  - ✓ Tout d'abord, importons toutes les bibliothèques nécessaires
  - ✓ Déclarer les listes a remplir (les produits, les prix, les descriptions)
  - ✓ Récupérer la page de résultat d'une requête de recherche sur avito.ma
  - ✓ Récupérer les produits avec leurs détails
  - ✓ Parcourir la liste et récupérer le type de produit, le prix et description puis ajouter les dans les listes

```
import requests as req
from bs4 import BeautifulSoup
produits =[] #Liste pour stocker le nom du produit
prix = [] #Liste pour stocker le prix du produit
descriptions = [] #Liste pour stocker les description du produit
url = "https://www.avito.ma/fr/maroc/ordinateurs_portables-%C3%A0_vendre"
page = req.get(url).text # Récupérer la page html
soup = BeautifulSoup(page, 'html.parser') # Récupérer la page html
# Récupérer la liste des produits dans le résultat de recherche
list_result = soup.findAll('div',attrs={'class':'oan6tk-4'})
for elem in list_result:
    # récupérer la balise prix de produit
    pri = elem.find('span',attrs={'class':'oan6tk-15'})
    # récupérer la balise description de produit
    desc = elem.find('span',attrs={'class':'oan6tk-17'})
    # récupérer la balise type de produit
    pro = elem.find('p',attrs={'class':'oan6tk-18'})
    produits.append(pro.text) # ajouter le type de produit
    prix.append(pri.text) # ajouter le prix de produit
    descriptions.append(desc.text) # ajouter la description de produit
```

grattage du site Web avito.ma 3

- Étape 5 : Exécutez le code et extrayez les données

- Étape 6 : Stockez les données dans un format requis

  ✓ Après avoir extrait les données, : vous souhaiterez peut-être les stocker dans un format. Ce format varie en fonction de vos besoins. Pour cet exemple, nous allons stocker les données extraites au format CSV (Comma Separated Value). Pour ce faire, je vais ajouter les lignes suivantes à mon code

  ✓ Un nom de fichier « produits.csv » est créé et ce fichier contient les données extraites.

```
import csv
list_produits =[["type","prix","description"]]
for i in range(len(list_result)):
    list_produits.append([produits[i],prix[i],descriptions[i]])

with open('produits.csv','w',newline='') as f:
    datacsv = csv.writer(f,delimiter=';')
    datacsv.writerows(list_produits)
    f.close()
```

	A	B	C	D	E	F
1	type	prix	description			
2	Ordinateurs p	Prix non spec	DELL i7 M4800 RAM 1000GB SSD 16GB RAM			
3	Ordinateurs p	Prix non spec	DELL i7 M4800 RAM 1000GB SSD 16GB RAM			
4	Ordinateurs p	Prix non spec	DELL i7 M4800 1TB SSD 24GB RAM Workstation			
5	Ordinateurs p	Prix non spec	DELL i7 M4800 1TB SSD 24GB RAM Workstation			
6	Ordinateurs p	Prix non spec	HP ProBook 4535s I5 3eme 8GB RAM SSD 256GB			
7	Ordinateurs p	3 000 DH	Dell i5 5th			
8	Ordinateurs p	Prix non spec	HP ProBook 4535s I5 3eme 8GB RAM SSD 256GB			
9	Ordinateurs p	Prix non spec	DELL i7 PRECISION 8GB RAM 500GB HDD NVIDIA			
10	Ordinateurs p	Prix non spec	DELL i7 PRECISION 8GB RAM 500GB HDD NVIDIA			
11	Ordinateurs p	5 000 DH	HP 250 G8 i3 4go 500go neuf			

Automatisation Python Excel

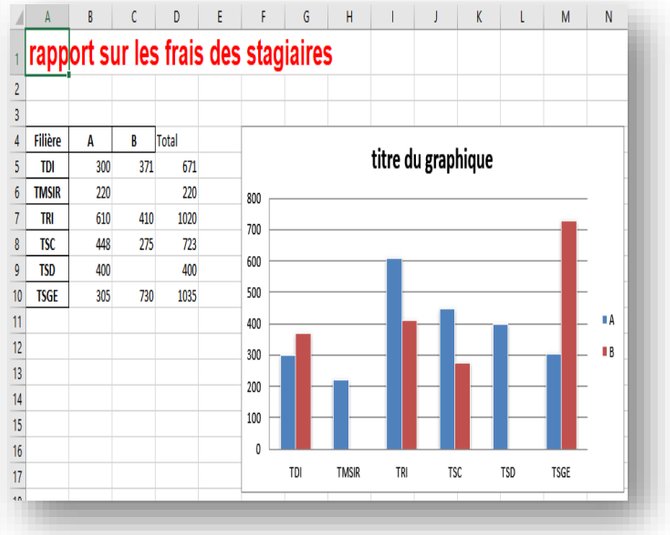
- Maintenant que vous avez une bonne compréhension de Python et de MS Excel, il est maintenant temps de comprendre les étapes pour configurer Python Excel Automation. Vous pouvez suivre les étapes ci-dessous pour configurer Python Excel Automation :

  - ✓ Étape 1 : Analyser l'ensemble de données Excel
  - ✓ Étape 2 : Créer des tableaux croisés dynamiques à l'aide de Pandas
  - ✓ Étape 3 : Concevoir les rapports à l'aide d'Openpyxl
  - ✓ Étape 4 : Automatiser le rapport avec Python
  - ✓ Étape 5 : Planification du script Python

Analyser l'ensemble de données Excel

- La première étape de Python Excel Automation consiste à analyser l'ensemble de données. L'ensemble de données utilisé dans cette exemple est un ensemble de données d'inscription des stagiaires. Les données seront utilisées pour créer le rapport ci-dessous après avoir configuré notre Python Excel Automation.

	A	B	C	D	E	F	G
1	Mle	Nom	Prénom	Filière	langue	Groupe	Frais
2	5647	ABIDINE	HAMZA	TRI	Anglais	A	200,00 DH
3	5661	ACHKY	SIHAM	TSGE	Espagnol	B	250,00 DH
4	5648	ACHOUBIE	MOHAMED	TDI	Français	A	300,00 DH
5	5649	ADLLAL	MOUNA	TRI	Espagnol	B	150,00 DH
6	5674	BARGACH	ASMAA	TSC	Français	B	275,00 DH
7	5650	AOULOUZI	YOUNESS	TSGE	Anglais	A	305,00 DH
8	5845	AALEM	MOHAMED	TDI	Espagnol	B	126,00 DH
9	5846	ABBAD	LAILA	TSD	Anglais	A	400,00 DH
10	5457	AIT BBIH	RABIAA	TSGE	Français	B	159,00 DH
11	5847	AKOUFA	MARIAM	TSC	Espagnol	A	258,00 DH
12	5848	AMMOURY	IBTISSAM	TRI	Français	B	260,00 DH
13	5849	ASBAI	YAHYA	TMSIR	Français	A	220,00 DH
14	5850	AZHAR	ALI	TDI	Anglais	B	245,00 DH
15	5851	BEIDAR	YOUNESS	TRI	Français	A	410,00 DH
16	5852	CHAKHS	FADWA	TSGE	Français	B	321,00 DH
17	5853	CHIHEB	ASMAA	TSC	Anglais	A	190,00 DH



Création des tableaux croisés dynamiques à l'aide de Pandas (1/2)

- La prochaine étape de Python Excel Automation consiste à concevoir des tableaux croisés dynamiques. Avant de faire cela, vous devez installer et importer les bibliothèques suivantes : pandas et openpyxl

- Pandas est utilisé pour lire le fichier Excel, créer le tableau croisé dynamique et l'exporter vers Excel. Vous pouvez ensuite utiliser la bibliothèque Openpyxl en Python pour écrire des formules Excel, créer des graphiques et des feuilles de calcul en Python.

- Pour lire votre fichier Excel, assurez-vous que le fichier est au même endroit où se trouve votre script Python

- Pour créer le tableau croisé dynamique, vous devez accéder au bloc de données file_excel que vous avez créé précédemment. Vous pouvez utiliser le " .pivot_table() " pour créer le tableau. Si vous souhaitez créer un tableau croisé dynamique pour afficher l'argent total réparti entre les hommes et les femmes, vous pouvez exécuter la méthode suivante : .pivot_table()

Création des tableaux croisés dynamiques à l'aide de Pandas (2/2)

- Enfin, pour exporter le tableau croisé dynamique , nous utiliserons la méthode .to_excel()

```
import pandas as pd


# Lire la base de données Excel
file_xl = pd.read_excel('ofppt.xlsx')
# Lire les champs à utiliser dans le tableau croisé
file_xl[['Groupe', 'Filière', 'Frais']]
# Créer le tableau croisé
tableau_croisé = file_xl.pivot_table(index='Filière',columns='Groupe',values='Frais',aggfunc='sum').round(0)
# Expoerter les résultats dans un autre fichier excel sur la feuille report
tableau_croisé.to_excel('export.xlsx',sheet_name='Report',startrow=3)
```

Concevoir les rapports à l'aide d'Openpyxl (1/2)

- La prochaine étape dans Python Excel Automation consiste à concevoir les rapports. Pour faire le rapport, vous devez utiliser la méthode « load_workbook » , qui est importée d'Openpyxl et l'enregistrer à l'aide de la méthode « .save() » .

- Python Excel Automation vous permet de créer des graphiques Excel à l'aide de tableaux croisés dynamiques. Pour créer un graphique Excel à l'aide d'un tableau croisé dynamique, vous devez utiliser le module Barchart et pour identifier la position des données et des valeurs de catégorie, vous pouvez utiliser le module Reference.

- Il faut importer les 2 module. Vous pouvez écrire des formules basées sur Excel en Python, de la même manière que vous les écrivez dans Excel.

Concevoir les rapports à l'aide d'Openpyxl (2/2)


```
import openpyxl as xl
from openpyxl.styles import * # Mise enforme(Font, Border...)
from openpyxl.chart import BarChart, Reference
# suite
wb = xl.load_workbook('export.xlsx')  ;    sheet = wb['Report']
# Graphique
graphe = BarChart() ;  graphe.type = "col" #{'col','bar'}
graphe.style = 10 # style de graphique de 1 à 45
graphe.title = "titre du graphique"; graphe.y_axis.title = 'titre axe vertical'
graphe.x_axis.title = 'titre axe horizontal'
```

```
# Les valeurs à représenter B4:C10
data = Reference(sheet, min_col=2, min_row=4, max_row=10, max_col=3)
# les catégories a utiliser A5:A10
cats = Reference(sheet, min_col=1, min_row=5, max_row=10)
# ajouter les données a représenter au graphique
graphe.add_data(data, titles_from_data=True)
graphe.set_categories(cats) # ajouter les catégories au graphique
graphe.shape = 4 # forme du graphique
sheet.add_chart(graphe, "F4") # afficher le graphique dans F4
sheet['D5'] = '=SUM(B5:C5)' # calculer la somme
sheet['D5'].style ='Currency' # Monétaire
wb.save('export.xlsx')
```

Automatiser le rapport avec Python

Partie 1 | Partie 2 | Partie 3

- La prochaine étape dans Python Excel Automation consiste à automatiser votre rapport. Vous pouvez écrire tout le code dans une seule fonction afin qu'il soit facile d'automatiser le rapport.

```
import pandas as pd
import openpyxl as xl
from openpyxl.styles import * # Mise enforme(Font, Border...)
from openpyxl.chart import BarChart, Reference
def automate_excel(file_name):
    """Le nom du fichier doit avoir la structure suivante : classeur.xlsx"""
    # lire le fichier excel
    file_xl = pd.read_excel(file_name)
    # faire un tableau croisé dynamique
    tableau_croisé =
file_xl.pivot_table(index='Filière',columns='Groupe',values='Frais',aggfunc='sum').r
ound(0)
    # envoyer le tableau du rapport dans un fichier excel
    tableau_croisé.to_excel('rapport.xlsx' ,sheet_name='Report',startrow=3)
    # chargement du classeur et sélection de la feuille
    wb = xl.load_workbook('rapport.xlsx')
    sheet = wb['Report']
    # références de cellules (feuille de calcul d'origine)
    min_col = wb.active.min_column
    max_col = wb.active.max_column
    min_row = wb.active.min_row
    max_row = wb.active.max_row
```

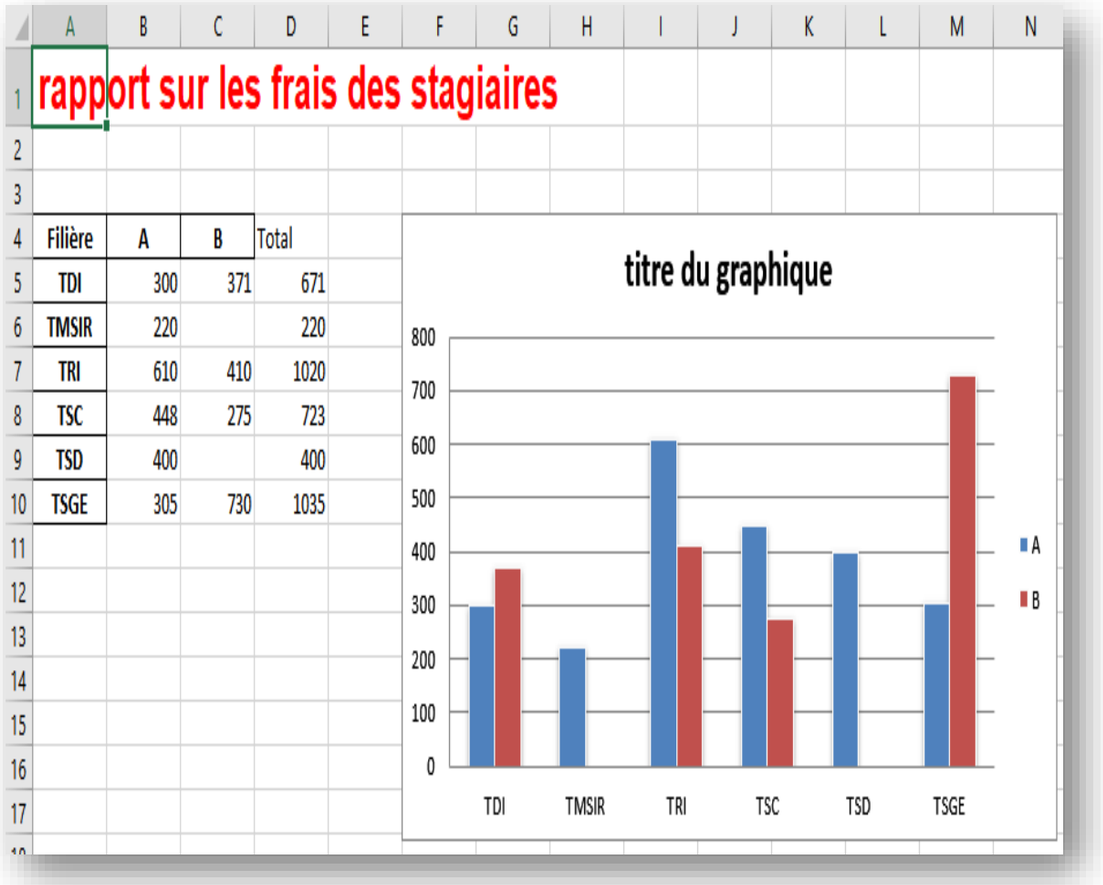
```
# ajout d'un graphique
    graphe = BarChart()
    data = Reference(sheet, min_col=min_col+1, max_col=max_col, min_row=min_row, max_row=max_row) #y
compris les en-têtes
    categories = Reference(sheet, min_col=min_col, max_col=min_col, min_row=min_row+1, max_row=max_row)
#sans compter les en-têtes
    graphe.add_data(data, titles_from_data=True)
    graphe.set_categories(categories)
    sheet.add_chart(graphe, "F4") # afficher le graphique dans F4
    graphe.title = "titre du graphique"     ;    graphe.style = 10
    # appliquer des formules    # somme dans la colonne D
    for i in range(max_row-4):
        sheet[f'D{i+5}'] = f'=SUM(B{i+5}:C{i+5})'
    # mise en forme du rapport
    sheet['D4'] = 'Total'
    sheet['A1'] = 'rapport sur les frais des stagiaires'
    sheet['A1'].font = Font('Arial', bold=True, size=20, color='FFFF0000') #aRGB(red)
    wb.save('rapport.xlsx')
    return
if __name__ == "__main__":
    print("Python Excel Automation consiste à analyser l'ensemble de données")
    automate_excel('ofppt.xlsx')
```

Planification du script Python

- La dernière étape de Python Excel Automation consiste à exécuter le script Python à différents horaires selon les exigences en matière de données. Il vous suffit d'utiliser le planificateur de tâches ou cron sur Windows et Mac respectivement.

- Cet explication vous a donné un guide étape par étape sur la configuration de Python Excel Automation. Il vous a également donné un aperçu de Python et de MS Excel ainsi que de leurs fonctionnalités.

- Dans l'ensemble, Python Excel Automation est un processus innovant que vous pouvez utiliser pour créer des rapports visuels sur Python de la même manière que vous le feriez sur Excel de manière transparente. Les organisations peuvent tirer parti de Python Excel Automation pour exploiter leurs tâches commerciales en fonction de leurs besoins.



Filière	A	B	Total
TDI	300	371	671
TMSIR	220		220
TRI	610	410	1020
TSC	448	275	723
TSD	400		400
TSGE	305	730	1035

CHAPITRE 5

Gérer et déployer un projet Python dans le Git

1. Utiliser les packages en Python
2. Coder une solution orientée objet en Python
3. **Apprendre à utiliser d'autres fonctionnalités Python**

Définition de Git

- Git est un logiciel de versioning créé en 2005 par Linus Torvalds, le créateur de Linux.
- Un logiciel de versioning, ou logiciel de gestion de version est un logiciel qui permet de conserver un historique des modifications effectuées sur un projet afin de pouvoir rapidement identifier les changements effectuées et de revenir à une ancienne version en cas de problème.
- Les logiciels de gestion de versions sont quasiment incontournables aujourd'hui car ils facilitent grandement la gestion de projets et car ils permettent de travailler en équipe de manière beaucoup plus efficace.
- Parmi les logiciels de gestion de versions, Git est le leader incontesté et il est donc indispensable pour tout développeur de savoir utiliser Git.

Utilisation des systèmes de gestion de version

- Imaginez que vous possédiez un site web. A chaque fois que vous voulez modifier quelque chose sur le site ou tester une fonctionnalité, vous êtes obligé d'effectuer une sauvegarde du site avant l'implémentation afin de pouvoir le restaurer si quelque chose se passe mal.

- Imaginez maintenant que vous soyez 10 à travailler sur le même site web, en vous occupant chacun de développer des fonctionnalités différentes mais qui peuvent être liées entre elles.

- Ici, l'idée la plus logique serait de mettre en place un serveur distant qui contiendrait l'historique des modifications faites par chaque développeur afin que chacun ait accès aux avancées des autres.

- les logiciels de versioning et Git en particulier vont nous permettre d'effectuer ces opérations mais de manière beaucoup plus robuste et avec de nombreuses fonctionnalités supplémentaires très utiles.

Modèles des logiciels de gestion de version

- Les logiciels de gestion de version sont aujourd'hui tous construits sur l'un des deux modèles suivants : le modèle centralisé ou le modèle décentralisé encore appelé modèle distribué.

- Le principe de base d'un modèle centralisé est la centralisation du code source lié au projet : la source du code du projet est hébergé sur un serveur distant central et les différents utilisateurs doivent se connecter à ce serveur pour travailler sur ce code.

- Dans un modèle distribué, le principe de base est opposé : le code source du projet est toujours hébergé sur un serveur distant mais chaque utilisateur est invité à télécharger et à héberger l'intégralité du code source du projet sur sa propre machine.

- Le modèle distribué a été popularisé par Git et présente différents avantages notables par rapport au modèle centralisé : (la Simplicité / flexibilité du travail et la Sécurité)

Définition de GitHub

- GitHub est un service en ligne qui permet d'héberger des dépôts ou repo Git. C'est le plus grand hébergeur de dépôts Git du monde.

- Une grande partie des dépôts hébergés sur GitHub sont publics, ce qui signifie que n'importe qui peut télécharger le code de ces dépôts et contribuer à leur développement en proposant de nouvelles fonctionnalités.

- Pour récapituler, et afin d'être bien clair sur ce point : Git est un logiciel de gestion de version tandis que GitHub est un service en ligne d'hébergement de dépôts Git qui fait office de serveur central pour ces dépôts.

Utiliser Git

- On peut utiliser différents types d'interfaces pour installer et pour utiliser Git.

- On va utiliser la console et donc un langage en lignes de commande plutôt qu'une interface graphique. pour différentes raisons suivantes:

  ✓ Utiliser la ligne de commande est le seul moyen d'avoir accès à toutes les commandes Git ;

  ✓ Si vous savez comment utiliser la version en ligne de commande, vous saurez utiliser n'importe quelle autre interface graphique.

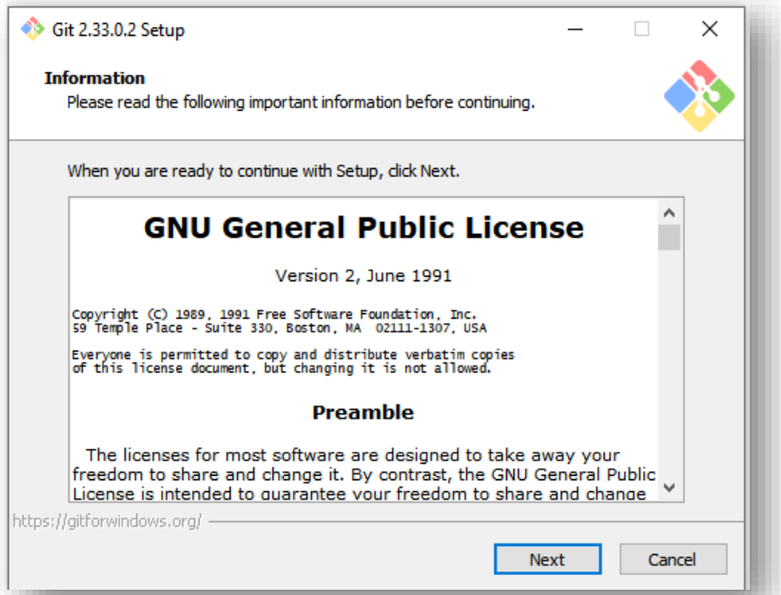
- Les commandes de Git vont une nouvelle fois être les mêmes pour tous les systèmes (Windows, Mac OS, etc.). Les commandes liées aux opérations sur le système comme la création d'un dossier vont elles pouvoir être différentes d'un système à l'autre.

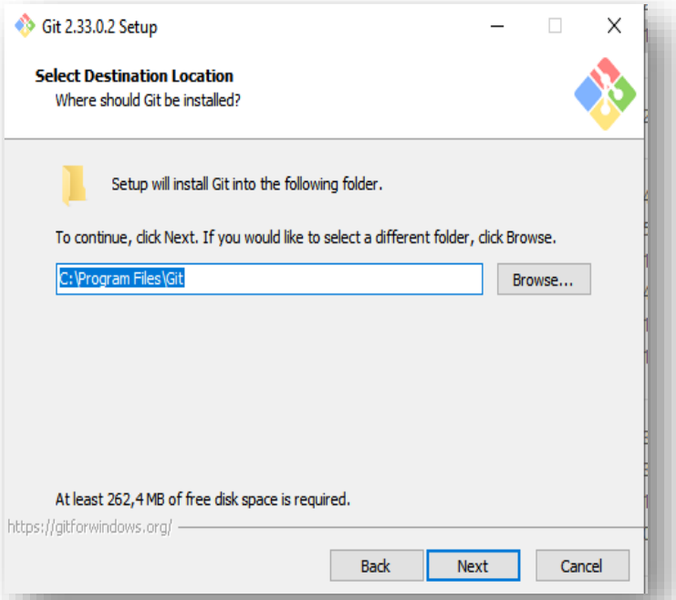
- A titre d'information, les commandes (UNIX) de base à connaitre sont les suivantes : (pwd, ls, cd, mkdir, touch, mv etc)

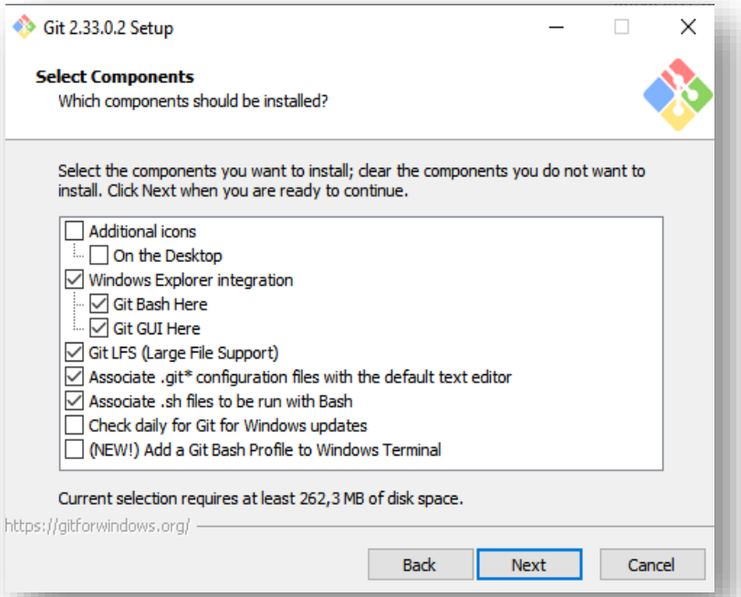
Installation de Git

- La façon la plus simple d'installer Git est de télécharger la dernière version sur le site officiel https://git-scm.com/downloads

- Ouvrir le fichier téléchargé et suivre les instructions à l'écran en laissant toutes les valeurs par défaut.

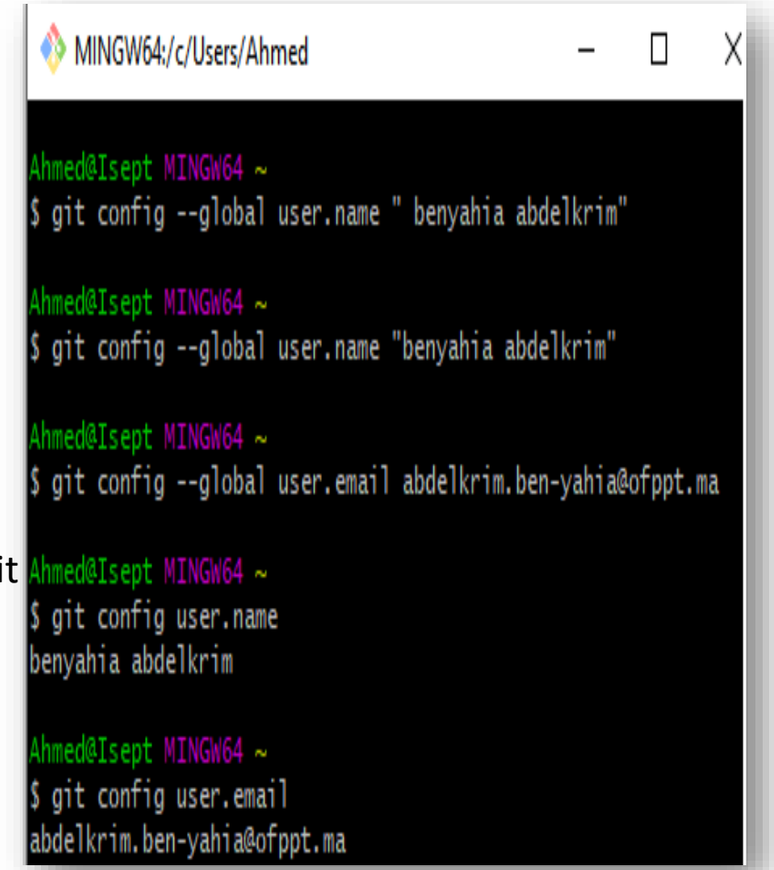






Paramétrage de Git

- Une fois Git installé, nous allons paramétrer le logiciel afin d'enregistrer certaines données pour ne pas avoir à les fournir à nouveau plus tard.
- Nous allons notamment ici renseigner un nom d'utilisateur et une adresse mail que Git devra utiliser ensuite.
- utiliser notre première commande Git qui est la commande git config. Cette commande permet de de voir et modifier les variables de configuration qui contrôlent tous les aspects de l'apparence et du comportement de Git.
- Les options permettent de personnaliser le comportement par défaut de certaines commandes.
- taper les commandes suivantes : git config --global user.name " benyahia abdelkrim" et git config --global user.email abdelkrim.ben-yahia@ofppt.ma à la suite pour renseigner un nom et une adresse email.
- Pour vous assurer que vos informations ont bien été enregistrées, vous pouvez taper git config user.name et git config user.email.



Gestion des informations selon Git

- Un “dépôt” correspond à la copie et à l’importation de l’ensemble des fichiers d’un projet dans Git. Il existe deux façons de créer un dépôt Git :

  ✓ On peut importer un répertoire déjà existant dans Git ;

  ✓ On peut cloner un dépôt Git déjà existant.

- Git pense les données à la manière d’un flux d’instantanés ou “snapshots”. Grosso modo, à chaque fois qu’on va valider ou enregistrer l’état d’un projet dans Git, il va prendre un instantané du contenu de l’espace de travail à ce moment et va enregistrer une référence à cet instantané pour qu’on puisse y accéder par la suite.

- Le fait que l’on dispose de l’historique complet d’un projet localement fait que la grande majorité des opérations de Git peuvent être réalisées localement, c’est-à-dire sans avoir à être connecté à un serveur central distant.

Types des états des fichiers

- il faut savoir qu'un fichier peut avoir deux grands états dans Git : il peut être sous suivi de version ou non suivi.
- Un fichier possède l'état "suivi" si il appartenait au dernier instantané capturé par Git, c'est-à-dire si il est enregistré en base. Tout fichier qui n'appartenait pas au dernier instantané et qui n'a pas été indexé est "non suivi".
- Lorsqu'on démarre un dépôt Git en important un répertoire déjà existant depuis notre machine, les fichiers sont au départ tous non suivis. On va donc déjà devoir demander à Git de les indexer et de les valider(les enregistrer en base).
- Ensuite, chaque fichier suivi peut avoir l'un de ces trois états : Modifié ("modified") ; Indexé ("staged") ; Validé ("committed").
- Lors du démarrage d'un dépôt Git à partir d'un dépôt local, on demande à Git de valider l'ensemble des fichiers du projet. Un fichier est "validé" lorsqu'il est stocké dans la base de donnée locale.

Types des zones de travail 1

- Les états de fichiers sont liés à des zones de travail dans Git. En fonction de son état, un fichier va pouvoir apparaitre dans telle ou telle zone de travail.

- Tout projet Git est composé de trois sections : le répertoire de travail (working tree), la zone d'index (staging area) et le répertoire Git (repository).

- Le répertoire de travail ou "working tree" correspond à une extraction unique ("checkout") d'une version du projet. Les fichiers sont extraits de la base de données compressée située dans le répertoire Git et sont placés sur le disque afin qu'on puisse les utiliser ou les modifier.

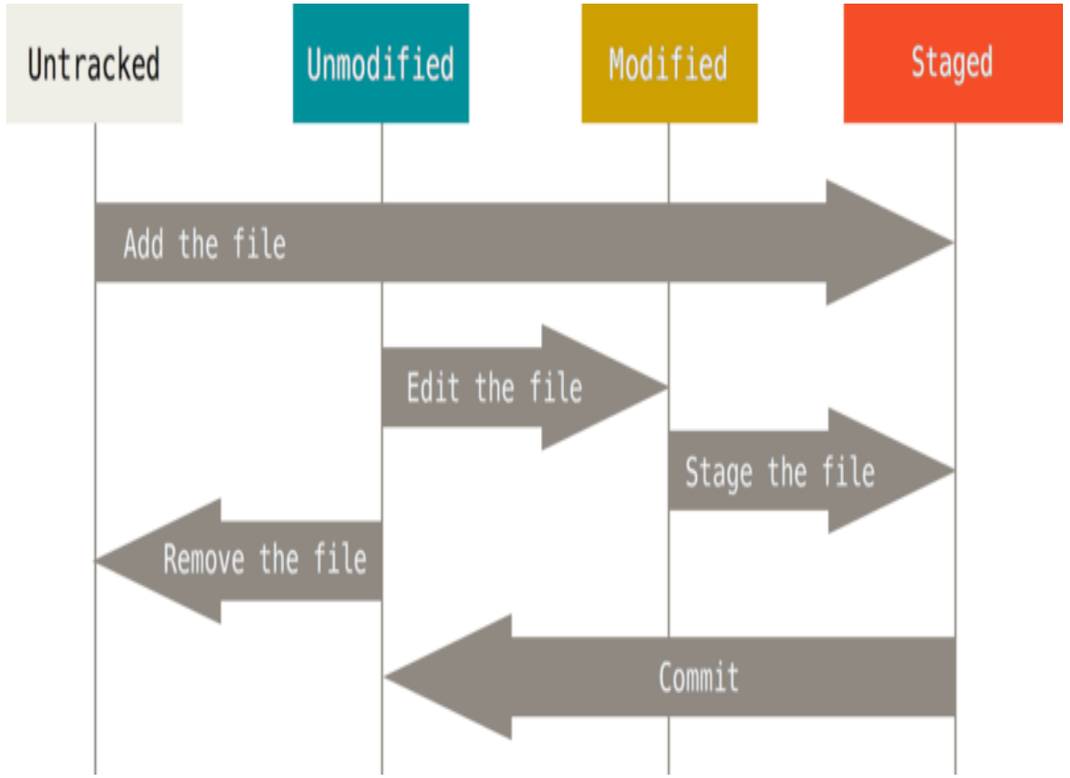
- La zone d'index ou "staging area" correspond à un simple fichier, généralement situé dans le répertoire Git, qui stocke les informations concernant ce qui fera partie du prochain instantané ou du prochain "commit".

- Le répertoire Git est l'endroit où Git stocke les méta-données et la base de données des objets de votre projet. C'est la partie principale ou le cœur de Git.

Types des zones de travail 2

- Le processus de travail va ainsi être le suivant :

  - ✓ nous allons travailler sur nos fichiers dans le répertoire de travail.
  - ✓ Lorsqu'on modifie ou crée un fichier, on peut ensuite choisir de l'indexer. Tant qu'un fichier n'est pas indexé, il possède l'état modifié ou est non suivi si c'est un nouveau fichier.
  - ✓ Dès qu'il est indexé que son nom est ajouté à la zone d'index, il possède l'état indexé. Finalement, on va valider ("commit") la version indexée de nos fichiers pour les ajouter au répertoire Git.



Création d'un un dépôt Git à partir d'un répertoire existant 1

- Lorsqu'on démarre avec Git, on a souvent déjà des projets en cours stockés localement sur notre machine ou sur serveur distant et pour lesquels on aimerait implémenter un système de gestion de version.

- On va créer un répertoire "projet-git" qui se trouve sur le bureau et qui contient deux fichiers texte vides "fichier1.txt" et "README.txt".

- Pour initialiser un dépôt Git, on utilise ensuite la commande git init. Cela crée un sous répertoire .git qui contient un ensemble de fichiers qui vont permettre à un dépôt Git de fonctionner.

- On peut utiliser ici la commande git status pour déterminer l'état des fichiers de notre répertoire.

- En cas de création d'un nouveau répertoire ou de correspondance absolue du répertoire principale et de la copie de travail, vous êtes généralement informé du fait que le projet ne comporte aucune modification (« No commits yet »). Git vous indique par ailleurs que vous n'avez actuellement partagé aucune modification pour le prochain commit (« nothing to commit »).

```
$ cd "C:\Users\vm\Desktop" #On se place sur le bureau
$ mkdir projet-git #Créer un dossier "projet-git"
$ cd projet-git #Se place dans le dossier
$ touch fichier1.txt #Créer un fichier
$ touch README.txt #Créer 2ème fichier
$ ls #Afficher le contenu du dossier
```

```
vm@Isept MINGW64 ~/Desktop/projet-git
$ git init
Initialized empty Git repository in C:/Users/vm/Desktop/projet-git/.git/

vm@Isept MINGW64 ~/Desktop/projet-git (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.txt
        fichier1.txt

nothing added to commit but untracked files present (use "git add" to track)

vm@Isept MINGW64 ~/Desktop/projet-git (master)
$
```

Création d'un un dépôt Git à partir d'un répertoire existant 2

- L'étape suivante va donc ici être d'indexer nos fichiers afin qu'ils puissent ensuite être validés, c'est-à-dire ajouter un nouveau fichier à la gestion des versions ou de signaler un fichier édité pour le prochain commit. on utilise la commande git add.

- À présent, si l'on vérifie à nouveau le statut du répertoire, le document servant d'exemple est présenté comme un candidat potentiel pour la prochaine étape de modification officielle du projet (« Changes to be commited »)

- L'ensemble des modifications que nous avons enregistrées pour la gestion des versions doivent toujours être validées par un commit pour être reprises dans le HEAD.

- Le HEAD est un type d'index qui renvoie au dernier commit ayant pris effet dans l'environnement de travail Git actuel (également appelé « branche »).

```
vm@Isept MINGW64 ~/Desktop/projet-git (master)
$ git add fichier1.txt

vm@Isept MINGW64 ~/Desktop/projet-git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   fichier1.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.txt


vm@Isept MINGW64 ~/Desktop/projet-git (master)
$ git commit -m "modification1"
[master e391a11] modification1
 1 file changed, 1 insertion(+)
```

Création d’un un dépôt Git à partir de cloner un dépôt Git

- La deuxième façon de démarrer un dépôt Git est de cloner localement un dépôt Git déjà existant. Pour cela, on va utiliser la commande Git clone.

- En pratique, dans la grande majorité des cas, nous clonerons des dépôts Git distants, c’est-à-dire des dépôts hébergés sur serveur distants pour pouvoir contribuer à ces projets.

- Si le répertoire Git existe déjà pour votre projet, vous aurez simplement besoin de l’adresse Web ou réseau de ce répertoire pour créer une copie de travail sur votre ordinateur avec la commande « git clone » : git clone https://.....

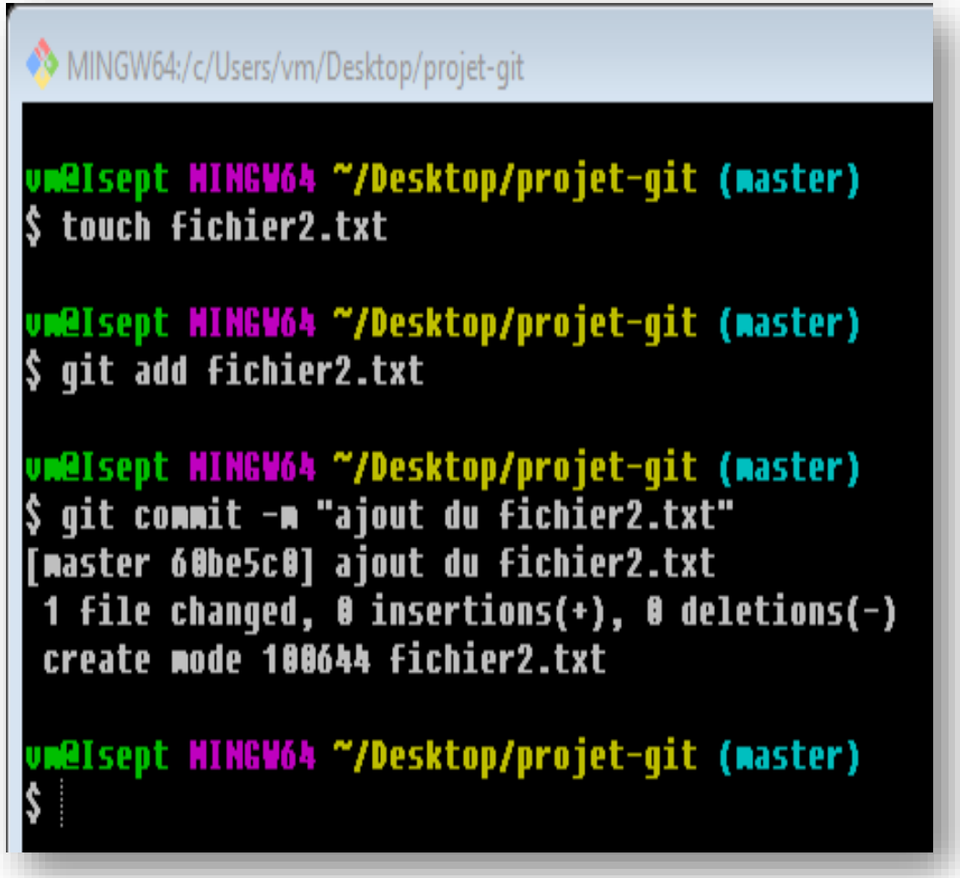
```
vm@Isept MINGW64 ~/Desktop/projet-cloner
$ git clone "C:\Users\vm\Desktop\projet-git"
Cloning into 'projet-git'...
done.
```

Enregistrer des modifications et modifier un dépôt Git 1

- nous allons être amenés à ajouter, modifier, voire supprimer des fichiers. On va indiquer tous ces changements à Git pour qu'il conserve un historique des versions auquel on pourra ensuite accéder pour revenir à un état précédent du projet (dans le cas où une modification entraine un bogue par exemple ou n'amène pas le résultat souhaité).

- A chaque fois qu'on souhaite enregistrer une modification de fichier ou un ajout de fichier dans le dépôt Git, on va devoir utiliser les commandes git add et git commit comme on a pu le faire dans le slide précédente.

- Si vous souhaitez enregistrer toujours la dernière version d'un fichier, pensez donc bien toujours à effectuer un git add juste avant un git commit.



Enregistrer des modifications et modifier un dépôt Git 2

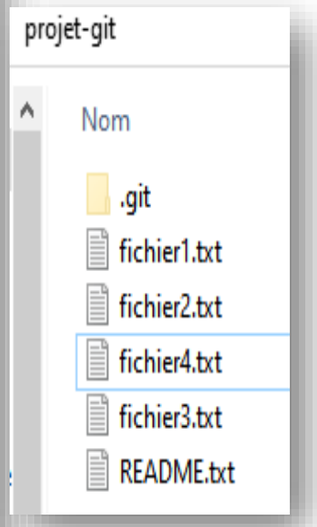
Partie 1 | Partie 2 | Partie 3

- Avant tout, notez que simplement supprimer un fichier de votre dossier avec une commande Bash rm par exemple ne suffira pas pour que Git oublie ce fichier : il apparaitra dans chaque git status dans une section “changes not stages for commit” (modifications qui ne seront pas validées).

- Pour supprimer un fichier et l’exclure du suivi de version, nous allons utiliser la commande git rm (et non pas simplement une commande Bash rm).

- Pour simplement exclure un fichier du suivi Git mais le conserver dans le projet, on va utiliser la même commande git rm mais avec cette fois-ci une option --cached.

```
MINGW64:/c/Users/vm/Desktop/projet-git
vm@Isept MINGW64 ~/Desktop/projet-git (master)
$ rm fichier4.txt

vm@Isept MINGW64 ~/Desktop/projet-git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    fichier4.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.txt

no changes added to commit (use "git add" and/or "git commit -a")

vm@Isept MINGW64 ~/Desktop/projet-git (master)
$ git rm fichier3.txt
rm 'fichier3.txt'

vm@Isept MINGW64 ~/Desktop/projet-git (master)
$ git rm --cached fichier2.txt
rm 'fichier2.txt'

vm@Isept MINGW64 ~/Desktop/projet-git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    fichier2.txt
        deleted:    fichier3.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    fichier4.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.txt
        fichier2.txt


vm@Isept MINGW64 ~/Desktop/projet-git (master)
$
```



Enregistrer des modifications et modifier un dépôt Git 3

- Le contenu de la zone d'index est ce qui sera proposé lors du prochain commit. Imaginons qu'on ait ajouté un fichier à la zone d'index par erreur. Pour retirer un fichier de l'index, on peut utiliser git reset HEAD nom-du-fichier.

- A la différence de git rm, le fichier continuera d'être suivi par Git. Seulement, le fichier dans sa version actuelle va être retiré de l'index et ne fera donc pas partie du prochain commit.

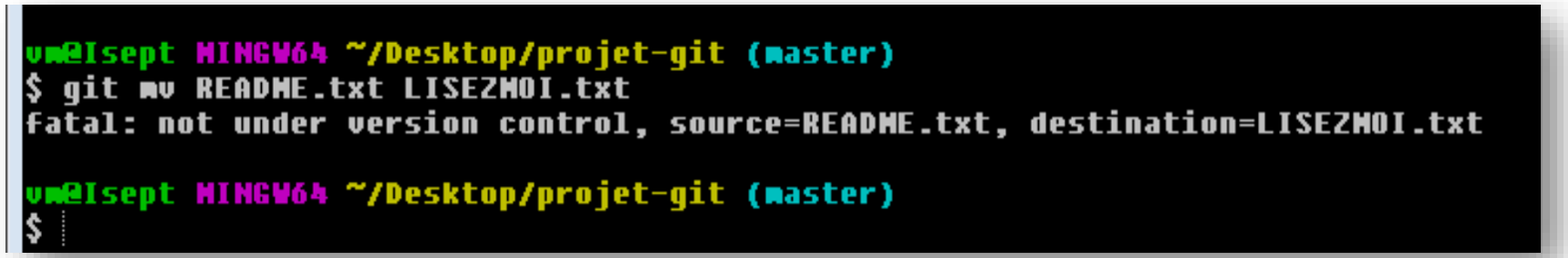
```
vm@Isept MINGW64 ~/Desktop/projet-git (master)
$ git reset fichier2.txt
Unstaged changes after reset:
D       fichier4.txt

vm@Isept MINGW64 ~/Desktop/projet-git (master)
```

Enregistrer des modifications et modifier un dépôt Git 4

- Lorsqu'on dispose d'un projet et qu'on souhaite utiliser Git pour effectuer un suivi de version, il est courant qu'on souhaite exclure certains fichiers du suivi de version comme certains fichiers générés automatiquement, des fichiers de configuration, des fichiers sensibles, etc.

- On peut informer Git des fichiers qu'on ne souhaite pas indexer en créant un fichier .gitignore et en ajoutant les différents fichiers qu'on souhaite ignorer. Notez qu'on peut également mentionner des schémas de noms pour exclure tous les fichiers correspondant à ce schéma et qu'on peut même exclure le contenu entier d'un répertoire en écrivant le chemin du répertoire suivi d'un slash.

- On peut également renommer un fichier de notre projet depuis Git en utilisant cette fois-ci une commande git mv ancien-nom-fichier nouveau-nom-fichier.

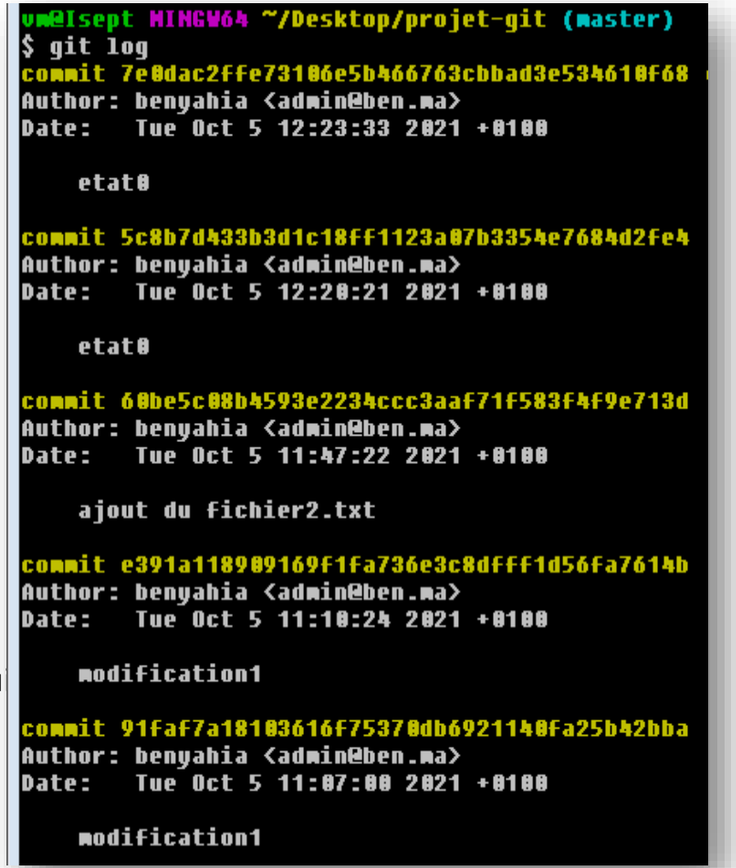


Consulter l'historique des modifications Git

- La manière la plus simple de consulter l'historique des modifications Git est d'utiliser la commande git log. Cette commande affiche la liste des commits réalisés du plus récent au plus ancien. Par défaut, chaque commit est affiché avec sa somme de contrôle SHA-1, le nom et l'e-mail de l'auteur, la date et le message du commit.

- La commande git log supporte également de nombreuses options. Certaines vont pouvoir être très utiles comme par exemple les options -p, --pretty, --since ou --author.

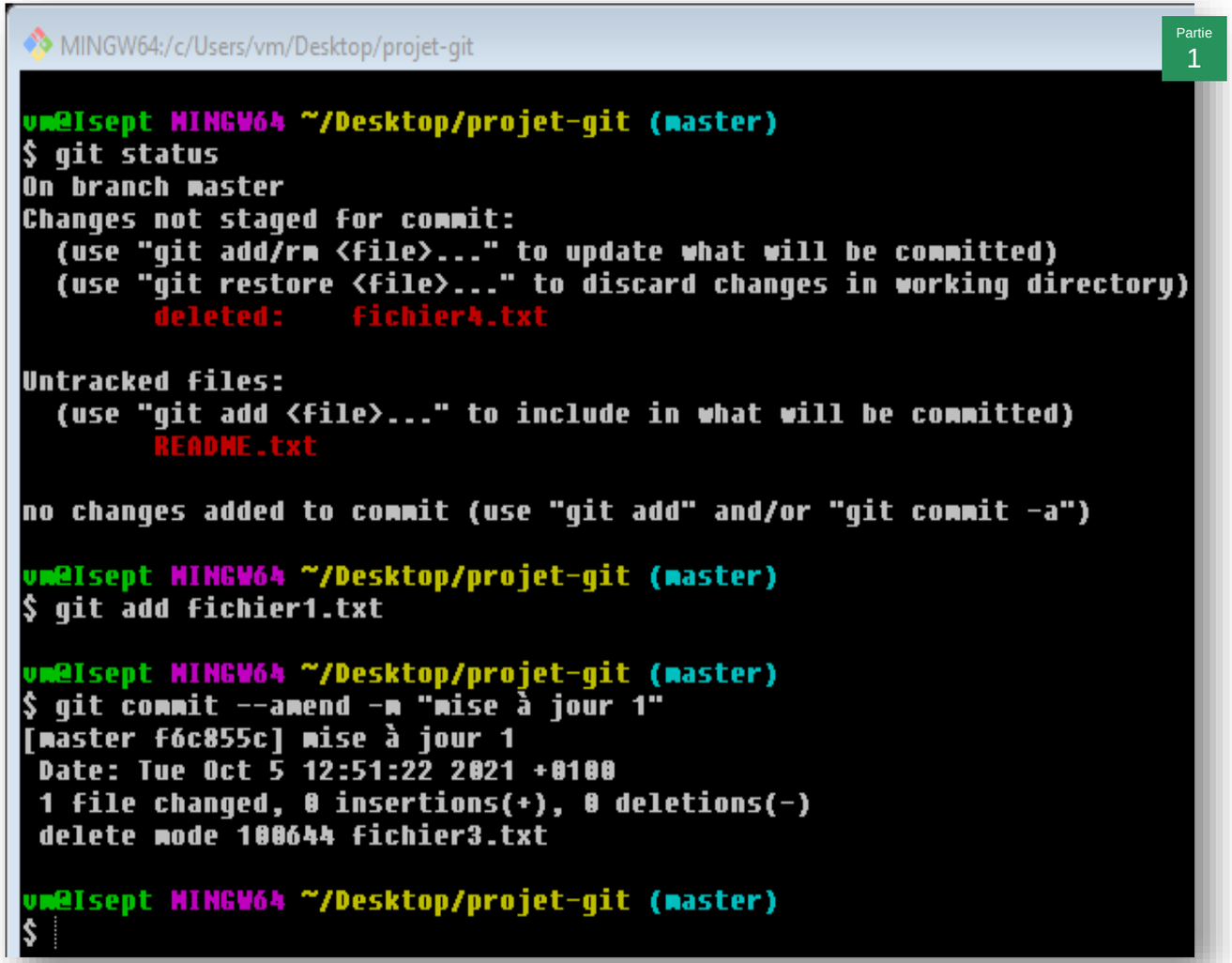
  - ✓ Utiliser git log -p permet d'afficher explicitement les différences introduites entre chaque validation.
  - ✓ L'option --pretty permet, avec sa valeur oneline, d'afficher chaque commit sur une seule ligne ce qui peut faciliter la lecture dans le cas où de nombreux commits ont été réalisés.
  - ✓ L'option --since permet de n'afficher que les modifications depuis une certaine date (on peut lui fournir différents formats de date comme 2.weeks ou 2021-10-10 par exemple).
  - ✓ L'option --author permet de n'afficher que les commits d'un auteur en particulier.



Ecraser et remplacer un commit

- Parfois, on voudra annuler une validation (un commit), notamment lorsque la validation a été faite en oubliant des fichiers ou sur les mauvaises versions de fichiers.

- La façon la plus simple d'écraser un commit est d'utiliser la commande git commit avec l'option --amend. Cela va pousser un nouveau commit qui va remplacer le précédent en l'écrasant.



Annuler des modifications apportées à un fichier

- L'un des principaux intérêts d'utiliser un logiciel de gestion de vision est de pouvoir "roll back", c'est-à-dire de pouvoir revenir à un état antérieur enregistré d'un projet.

- Après un commit, on va continuer à travailler sur nos fichiers et à les modifier. Parfois, certaines modifications ne vont pas apporter les comportements espérés et on voudra revenir à l'état du fichier du dernier instantané Git (c'est-à-dire au dernier état enregistré). On va pouvoir faire cela avec la commande générale git checkout -- nom-du-fichier ou la nouvelle commande spécialisée git restore.

Définition d’une branche

- Créer une branche, c’est en quelque sorte comme créer une “copie” de votre projet pour développer et tester de nouvelles fonctionnalités sans impacter le projet de base.

- Git a une approche totalement différente des branches qui rend la création de nouvelles branches et la fusion de branche très facile à réaliser. Une branche, dans Git, est simplement un pointeur vers un commit (une branche n’est qu’un simple fichier contenant les 40 caractères de l’empreinte SHA-1 du commit sur lequel elle pointe).

- La branche par défaut dans Git s’appelle master. Cette branche master va se déplacer automatiquement à chaque nouveau commit pour pointer sur le dernier commit effectué tant qu’on reste sur cette branche.

- Notez que la branche master n’est pas une branche spéciale pour Git : elle est traitée de la même façon que les autres branches. L’idée est que lorsqu’on tape une commande git init, une branche est automatiquement créée et que le nom donné à cette branche par Git par défaut est “master”. On pourrait très bien la renommer ensuite mais ça ne présente généralement aucun intérêt.

Créer, gérer et supprimer des branches (1/3)

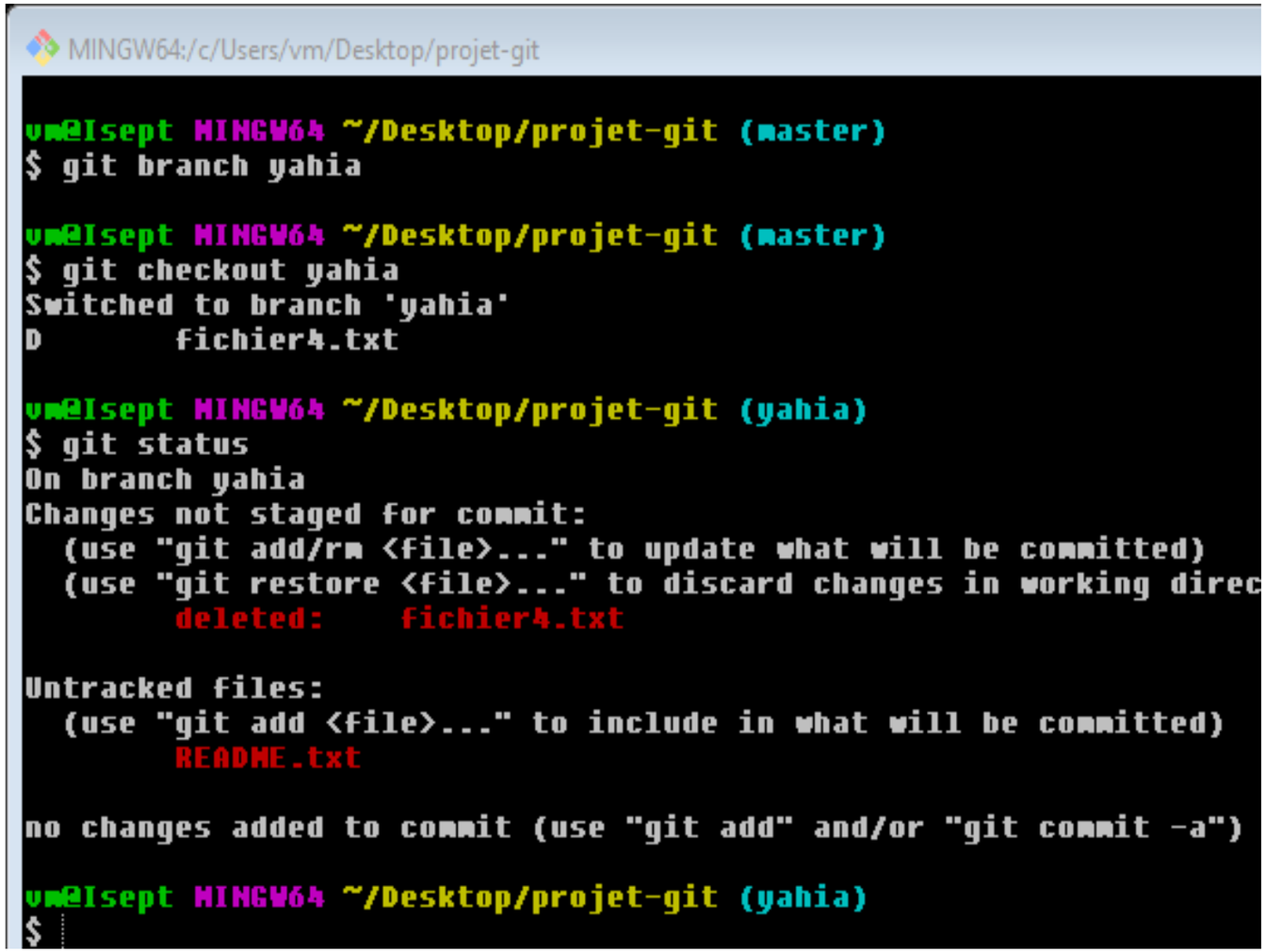
- Pour créer une nouvelle branche, on utilise la commande git branch nom-de-la-branche. A ce stade, vous allez donc avoir deux branches (deux pointeurs) vers le dernier commit : la branche master et la branche tout juste créée.

- Pour déterminer sur quelle branche vous vous trouvez, Git utilise un autre pointeur spécial appelé HEAD. HEAD pointe sur la branche master par défaut. Notez que la commande git branch permet de créer une nouvelle branche mais ne déplace pas HEAD.

- Nous allons donc devoir déplacer explicitement HEAD pour indiquer à Git qu'on souhaite basculer sur une autre branche. On utilise pour cela la commande git checkout suivi du nom de la branche sur laquelle on souhaite basculer.

Créer, gérer et supprimer des branches (2/3)

- On peut également utiliser git checkout -b pour créer une branche et basculer immédiatement dessus. Cela est l'équivalent d'utiliser git branch puis git checkout.

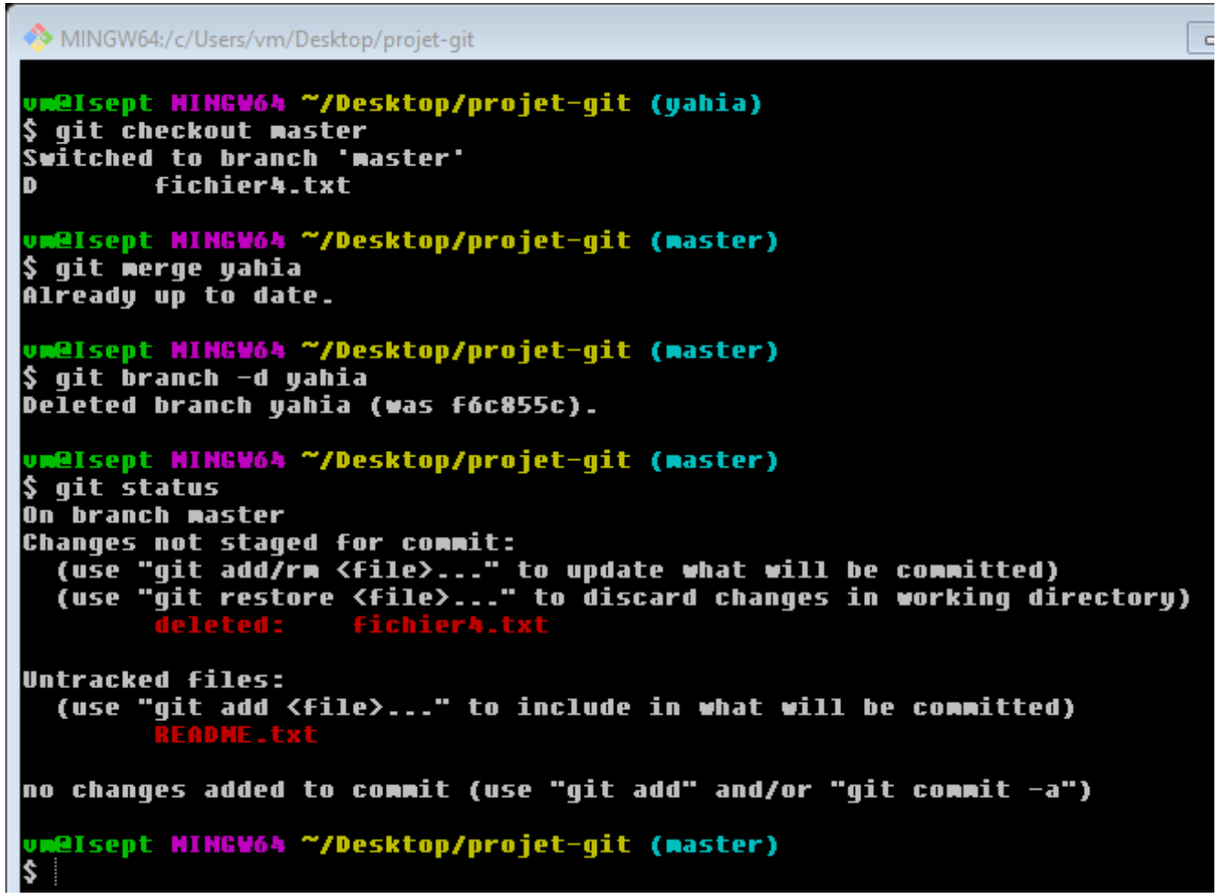


Créer, gérer et supprimer des branches (3/3)

- Si vous souhaitez fusionner des branches, vous devrez utiliser la commande « git merge ». À l'aide de « checkout », rendez-vous dans le répertoire devant accueillir une nouvelle branche et exécutez à cet endroit la commande précitée en indiquant le nom de la branche à enregistrer.

- Si vous avez fusionné des branches de travail et que vous n'avez plus besoin d'une branche particulière, vous pouvez la supprimer en toute simplicité. Pour ce faire, utilisez la suite de commandes « git branch -d » sur la branche de la version qui n'est plus nécessaire.

- Le seul prérequis pour procéder à la suppression est de se trouver dans une autre branche.



Gérer des dépôts Git distants 1

- La gestion de dépôts distants est plus complexe que la gestion d'un dépôt local puisqu'il va falloir gérer les permissions des différents utilisateurs, définir quelles modifications doivent être adoptées ou pas, etc.

- Dans le cas où on travaille à plusieurs sur un dépôt distant, nous n'allons jamais directement modifier le projet distant. Nous allons plutôt cloner le dépôt distant localement (sur notre ordinateur donc), effectuer nos modifications puis les pousser vers le dépôt distant. Ces modifications pourront ensuite être acceptées ou pas.

- Pour cloner un dépôt distant, nous allons pouvoir utiliser la commande git clone. Une fois que vous disposez d'un dépôt distant cloné localement, vous pouvez utiliser la commande git remote pour afficher la liste des serveurs distants des dépôts.

- Le nom donné par défaut par Git à un serveur à partir duquel on a effectué un clonage est origin. On peut également utiliser git remote -v pour afficher les URLs stockées.

- Pour définir un nom personnalisé pour un dépôt distant, on peut utiliser la commande git remote add suivi du nom choisi suivi de l'URL du dépôt. Pour renommer ensuite notre référence, on pourra utiliser git remote rename [ancien-nom] [nouveau-nom].

Gérer des dépôts Git distants 2

- La commande git fetch [nom-distant] va nous permettre de récupérer toutes les données d'un dépôt distant qu'on ne possède pas déjà. Cela permet de récupérer les ajouts du projet distant et d'avoir une version à jour du projet.

- Notez que fetch tire les données dans votre dépôt local mais sous sa propre branche ce qui signifie que ces données ne sont pas fusionnées avec vos autres travaux et que votre copie de travail n'est pas modifiée. Il faudra donc fusionner explicitement les données tirées par fetch par la suite.

- Dans le cas où on a créé une branche sur notre projet pour suivre les avancées d'une branche distante, on peut également utiliser git pull qui va récupérer les données d'une branche distante et les fusionner automatiquement dans notre branche locale.

- Une fois qu'on a terminé nos modifications localement, on va les pousser vers le dépôt distant. On utilise pour cela la commande git push [nom-distant] [nom-de-branche]. Si on souhaite par exemple pousser les modifications faites sur notre branche master vers le serveur origin, on écrira git push origin master.

Gérer des dépôts Git distants 3

- Pour obtenir des informations sur un dépôt distant, on peut utiliser la commande git remote show [nom-du-depot].

- Cette commande peut notamment fournir la liste des URL pour le dépôt distant ainsi que la liste des branches distantes suivies, les branches poussées automatiquement avec une commande git push, les branches qui seront fusionnées avec un git pull, etc.

- Pour retirer un dépôt distant, on utilisera la commande git remote rm [nom-du-depot].

Description de GitHub

- GitHub est la plus grande plateforme d'hébergement de projets Git au monde. Vous serez probablement amené à travailler avec GitHub et il est donc important de comprendre comment ce service fonctionne.

- GitHub est un outil gratuit pour héberger du code open source, et propose également des plans payants pour les projets de code privés.

- Pour utiliser GitHub, il suffit de créer un compte gratuitement sur le site https://github.com.

- Le grand intérêt de GitHub est de faciliter la collaboration à une échelle planétaire sur les projets : n'importe qui va pouvoir récupérer des projets et y contribuer (sauf si le propriétaire du projet ne le permet pas bien entendu).

- Sur GitHub, nous allons en effet notamment pouvoir cloner des projets (dépôts) publics, dupliquer ("fork") des projets ou encore contribuer à des projets en proposant des modifications ("pull request").

Contribuer à un projet avec GitHub ou le copier

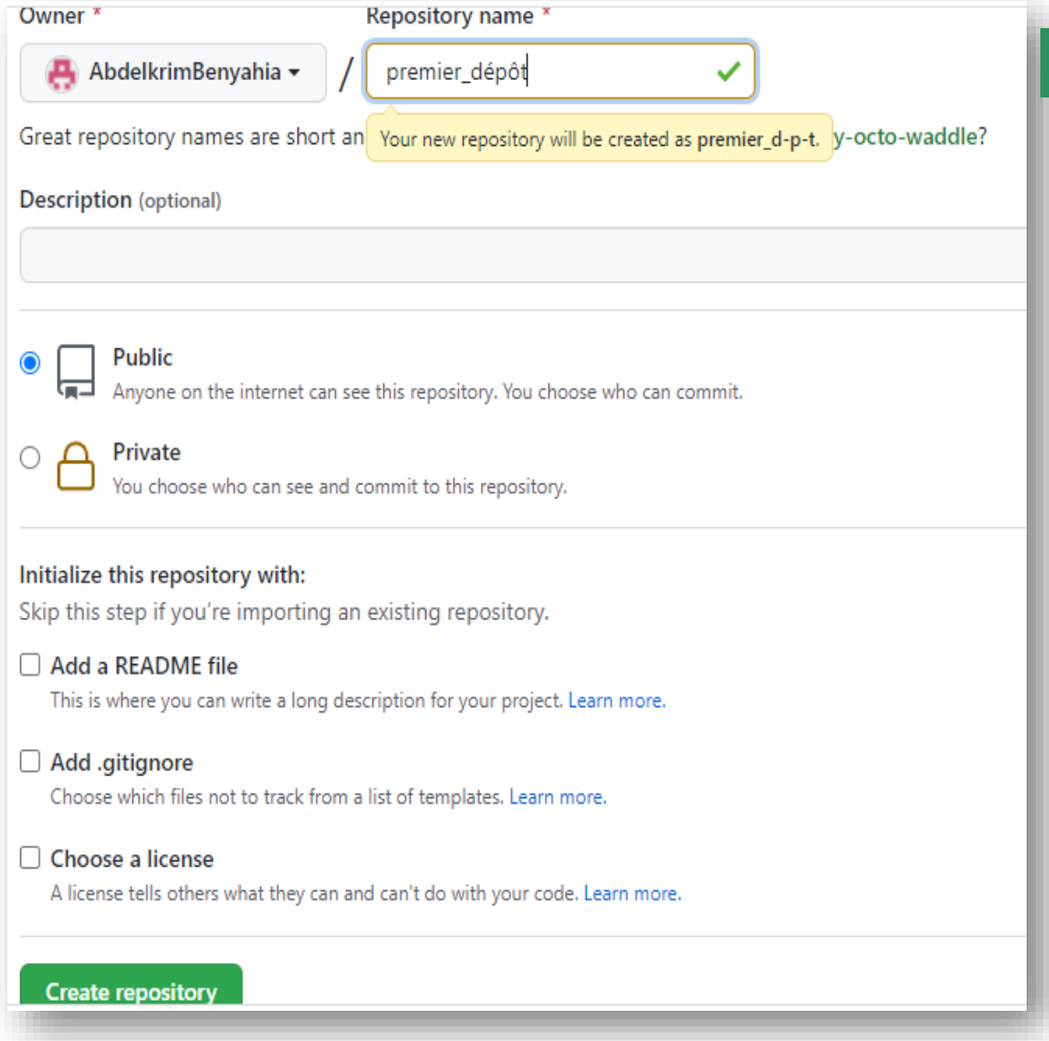
- Les étapes pour contribuer à un projet (le cycle de travail) vont toujours être les mêmes :

1. On copie un projet sur notre espace GitHub ;
2. On créer une branche thématique à partir de master ;
3. On effectue nos modifications / améliorations au projet ;
4. On pousse ensuite la branche locale vers le projet qu'on a copié sur GitHub et on ouvre une requête de tirage depuis GitHub ;
5. Le propriétaire du projet choisit alors de refuser nos modifications ou de les fusionner dans le projet d'origine ;
6. On met à jour notre version du projet en récupérant les dernière modifications à partir du projet d'origine.

Créer un dépôt GitHub

- Pour créer un nouveau dépôt sur GitHub, il suffit de cliquer sur l'icône "+" située en haut à droite puis sur "new repository". Une page s'ouvre vous permettant de créer un nouveau dépôt.

- Sur la page de création de dépôt, vous devez renseigner un nom et une description pour le dépôt. Vous avez également une option « Initialise with a README », qui vous permet de cloner votre dépôt sur votre machine. Cette option est à cocher uniquement dans le cas où vous n'avez pas encore créé le dépôt en question sur votre machine.



De GitHub à Git et inversement

- La plupart des gens qui utilisent GitHub vont cependant souvent préférer travailler hors ligne (en local; sur leur machine) plutôt que de devoir être constamment connecté à GitHub et de devoir passer par cette plateforme pour effectuer toutes les opérations.

- Pour travailler localement, il suffit de cloner un projet après l'avoir forké. On va ensuite pouvoir effectuer nos différentes modifications sur notre machine.

- Pour synchroniser les modifications faites depuis notre machine avec notre dépôt distant (dépôt GitHub), il suffit de faire un git commit depuis le dépôt local et de push (envoyer) les modifications sur le dépôt GitHub à l'aide de la commande git push [nom-distant] [nom-local].

- Taper git push origin master par exemple revient à envoyer les modifications situées dans ma branche master vers origin.

- Pour récupérer en local les dernières modifications du dépôt GitHub, on va utiliser la commande git pull [nom-distant] [nom-local].

```
# …or create a new repository on the command line
$ echo "# demo1" >> README.md
$ git init
$ git add README.md
$ git commit -m "commit 0"
$ git branch -M main
$ git remote add origin https://github.com/AbdelkrimBenyahia/demo1.git
$ git push -u origin main
##############################################
# …or push an existing repository from the command line
$ git remote add origin
$ https://github.com/AbdelkrimBenyahia/demo1.git
$ git branch -M main
$ git push -u origin main
```

•