



FORMATION

Python par la pratique

10/05/2023



m2iinformation.fr

A. Plan de cours



Jour 1

- Premier pas
- Les imports et modules
- Les variables
- Les blocs, conditions et boucles
- Les fonctions

Jour 2

- Les exceptions
- Le débogage
- Les fichiers
- Les regex
- Les classes

Jour 3

- Modules utiles
- Base de données
- Gestionnaires de paquets
- Les environnements virtuels
- CGI



Plan de cours

Jour 4

- Tkinter
- Django

B. Introduction



Introduction

Présentation des participants

- Votre nom
- Votre entreprise et le poste occupé
- Votre expérience dans la programmation
- Vos attentes sur cette formation
- Votre système d'exploitation et accès administrateur à votre machine

Introduction

Horaire

- Début tous les jours à 9h
- Pause de 15 min le matin et l'après-midi
- Pause déjeuner d'une heure à 12h30
- Fin de la journée à 17h30

Formation

- Cours théoriques et documentations
- Cours pratique
- Travaux pratiques
- Evaluation finale

Présence

- Emargé à chaque demi journée sur le site sign.m2information.fr

Introduction

Pourquoi Python ?

- Python est un langage interprété avec un typage dynamique fort.
- Il est lisible, concis, gratuit, portable, extensible, modulaire et permet la programmation orientée objet (POO).

En PHP :

```
<?php
function valeurs_paires($liste_valeurs) {
    $classement = array();
    for($i=0; $i<count($liste_valeurs); $i++) {
        if($liste_valeurs[$i] % 2 == 0) {
            array_push($classement, True);
        } else {
            array_push($classement, False);
        }
    }
    return $classement;
}

echo var_dump(valeurs_paires([51, 8, 85, 9]));
?>
```

En Python :

```
def valeurs_paires(liste_valeurs):
    return [(False if v % 2 else True) for v in liste_valeurs]

print valeurs_paires([51, 8, 85, 9])
```

```
[False, True, False, False]
```

Introduction

Champs d'application de Python :

Nous trouvons Python dans le Web, les multimédias, la bureautique, les utilitaires, l'intelligence artificielle, ...

Nous le retrouvons dans tous les domaines professionnels tel que :
Le domaine scientifique, les finances, la programmation système, les base de données, ...

Python à cette force de pouvoir réunir des profils d'informaticiens assez différents (administrateur système, développeur généraliste, développeur web, etc...).

Positionnement de Python



Introduction

Evolution



Introduction

Objectif d'apprentissage



**INSTALLATION DE PYTHON
ET CHOIX D'UN EDITEUR
OU IDE**



**LECTURE ET ECRITURE
DE FICHIERS**



**TRAVAILLER AVEC LES
VARIABLES, EXPRESSIONS
ET BOUCLES**



**MANIPULATION
DES FONCTIONS**



**UTILISATION DES
MODULES ET CLASSES**



**DECOUVRIR QUELQUES
LIBRAIRIES**

Introduction

Prérequis

BASE DE LA PROGRAMMATION

Les variables, les fonctions, les expressions.



OUTILS COURANT DE PROGRAMMATION

Les éditeurs et les IDE



LES CONCEPTS POO

Classe, Encapsulation, Héritage, Polymorphisme.



LANGAGE INTERPRETE

Mode d'exécution.



C. Installation



Installation



Installer Python

Python sera installé grâce à un fichier exécutable que vous pouvez télécharger à ce lien :

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



Installer un éditeur et ses extensions

Nous allons utiliser VSCode pour écrire nos codes.

Utilisez ce lien pour installer l'application :

<https://code.visualstudio.com/download>



Se connecter à Slack

Slack est un outil de collaboration professionnel où nous pourrions échanger et partager efficacement.

https://join.slack.com/t/slack-b9n6781/shared_invite/zt-154p48d91-5nbBJweh_U7wl2L8vuXyRA



Lancer son premier programme

Grâce à la console Python, nous pouvons exécuter notre première application.

Avant de se rendre pour le téléchargement, vous pouvez vérifier si Python est installé dans votre ordinateur en exécutant la commande : `python --version`

Python fait partie des principales distributions Linux et vient avec tout Mac équipé de Mac OS.

Grace au lien ci-dessous, vous êtes en mesure de sélectionner l'exécutable qui correspond à votre système.

<https://www.python.org/downloads/release/python-3113/>

Files						
Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore
Gzipped source tarball	Source release		1aea68575c0e97bc83ff8225977b0d46	26006589	SIG	CRT SIG
XZ compressed source tarball	Source release		b8094f007b3a835ca3be6bdf8116cccc	19618696	SIG	CRT SIG
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	4c89649f6ca799ff29f1d1dffcb9393	40865361	SIG	CRT SIG
Windows embeddable package (32-bit)	Windows		7e4de22bfe1e6d333b2c691ec2c1fcee	7615330	SIG	CRT SIG
Windows embeddable package (64-bit)	Windows		7f90f8642c1b19cf02bce91a5f4f9263	8591256	SIG	CRT SIG
Windows help file	Windows		643179390f5fd9d6b1ad66355c795bb	9355326	SIG	CRT SIG
Windows installer (32-bit)	Windows		58755d6906f825168999c83ce82315d7	27779240	SIG	CRT SIG
Windows installer (64-bit)	Windows	Recommended	bfb8467c7e3504f3800b0fe94d9a3e6	28953568	SIG	CRT SIG

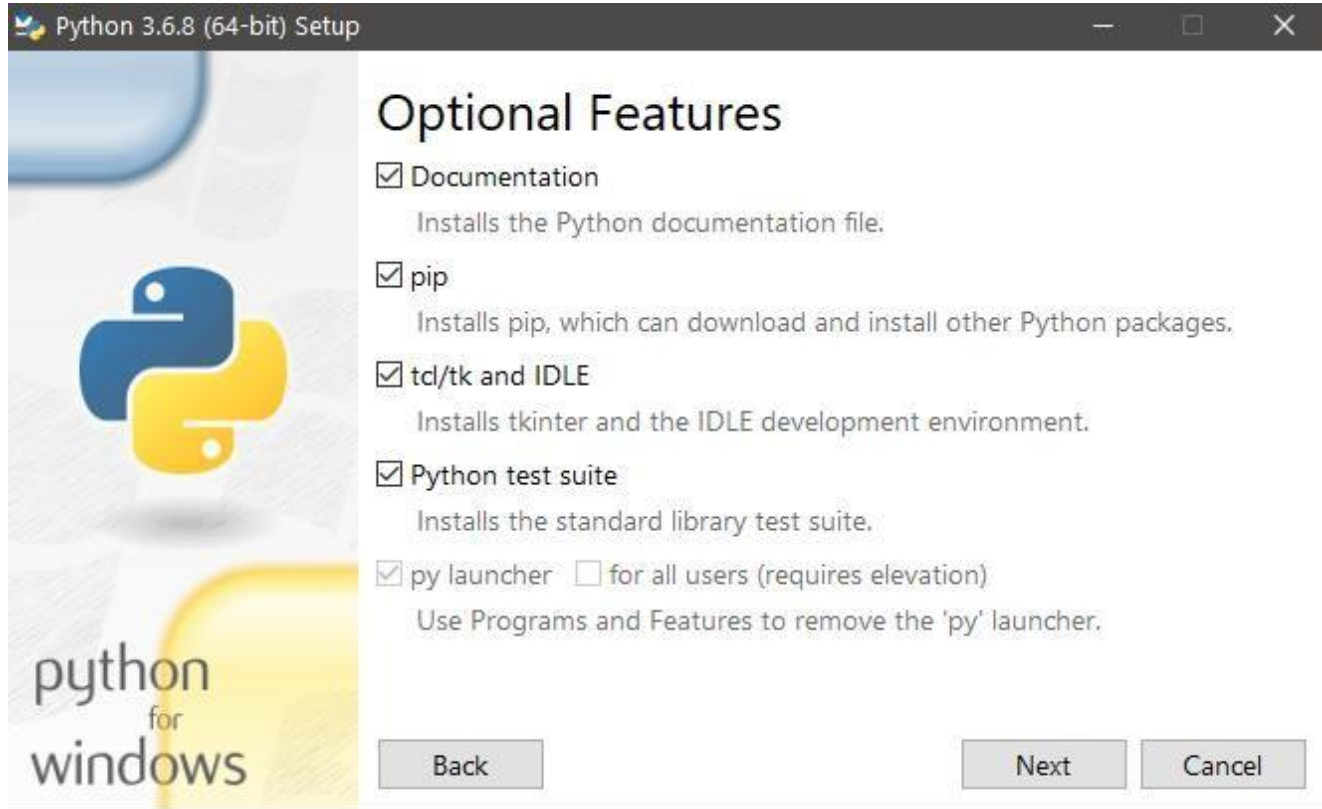
Installation

A la première fenêtre choisir Customize Installation et cliquez sur suivant.



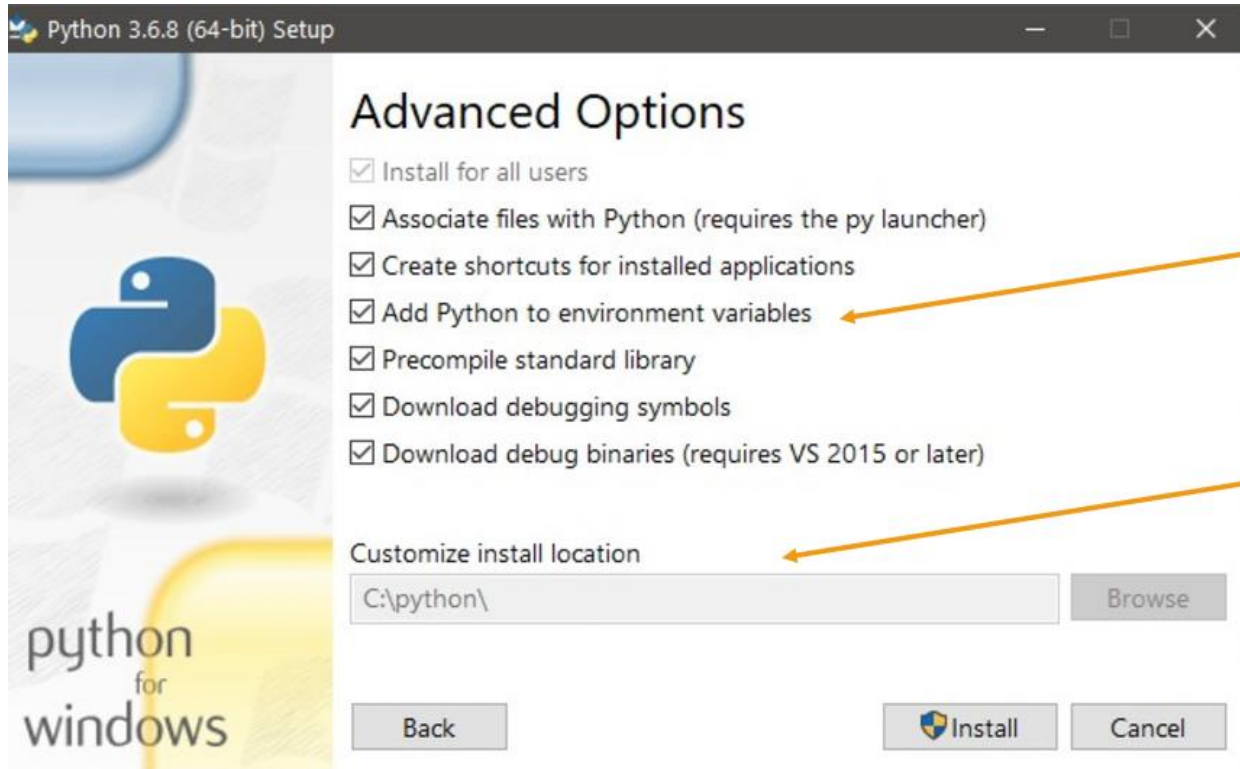
Installation

A la fenêtre des options cocher toutes les cases puis suivant.



Installation

La prochaine étape vous permettra de choisir dans quel dossier sera installé Python et principalement de l'ajouter aux variables d'environnements.



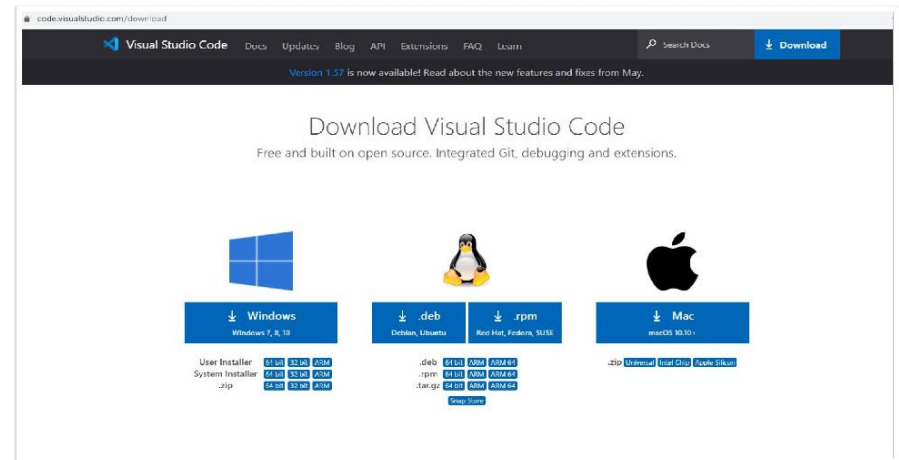
Installation

Un IDE (Integrated Development Environment) est un regroupement d'outils utiles pour le développement d'applications (éditeur de code, débbugger, builder, indexation du code pour recherches « intelligentes » dans les projets...), rassemblés dans un logiciel unique. (Eclipse, Netbeans, Xcode, Pycharm, Wingware).

Python est avant tout un langage de script, et un simple éditeur de code avec quelques fonctions utiles peut suffire.

Cet éditeur, qui est gratuit, nous accompagnera tout au long de notre programmation en python.

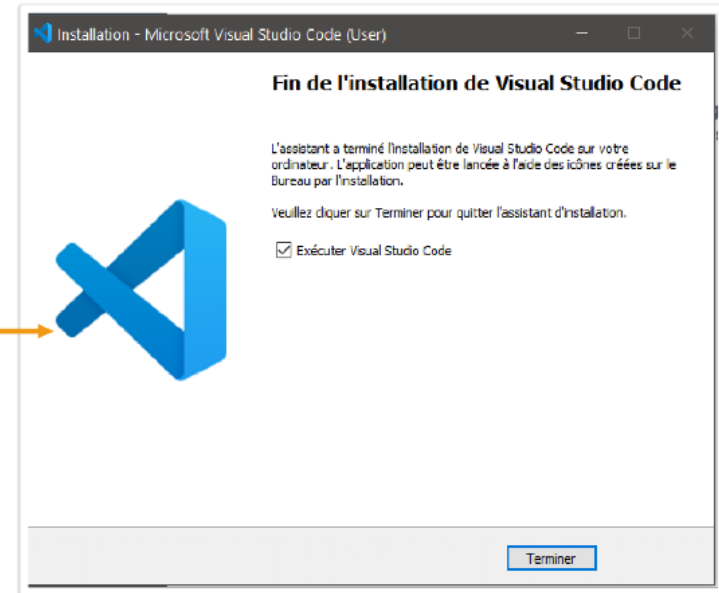
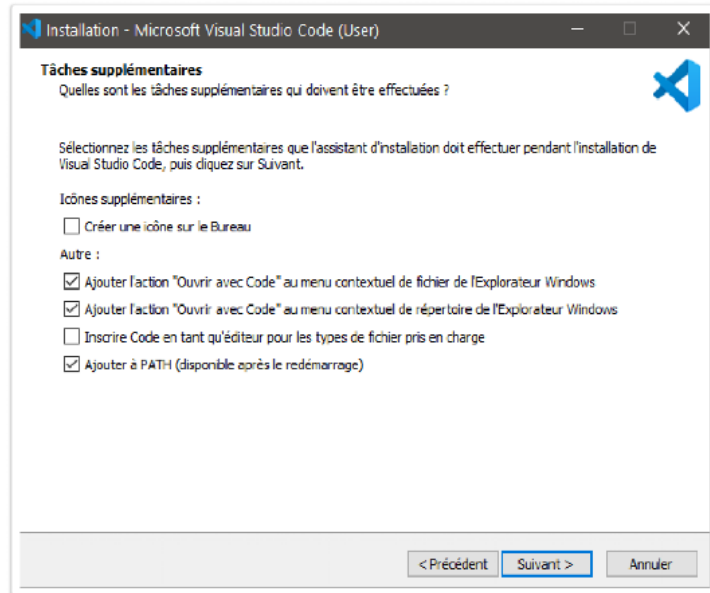
<https://code.visualstudio.com/download>



Installation

Le processus d'installation de VS Code est très simple et n'exigera pas trop de configurations.

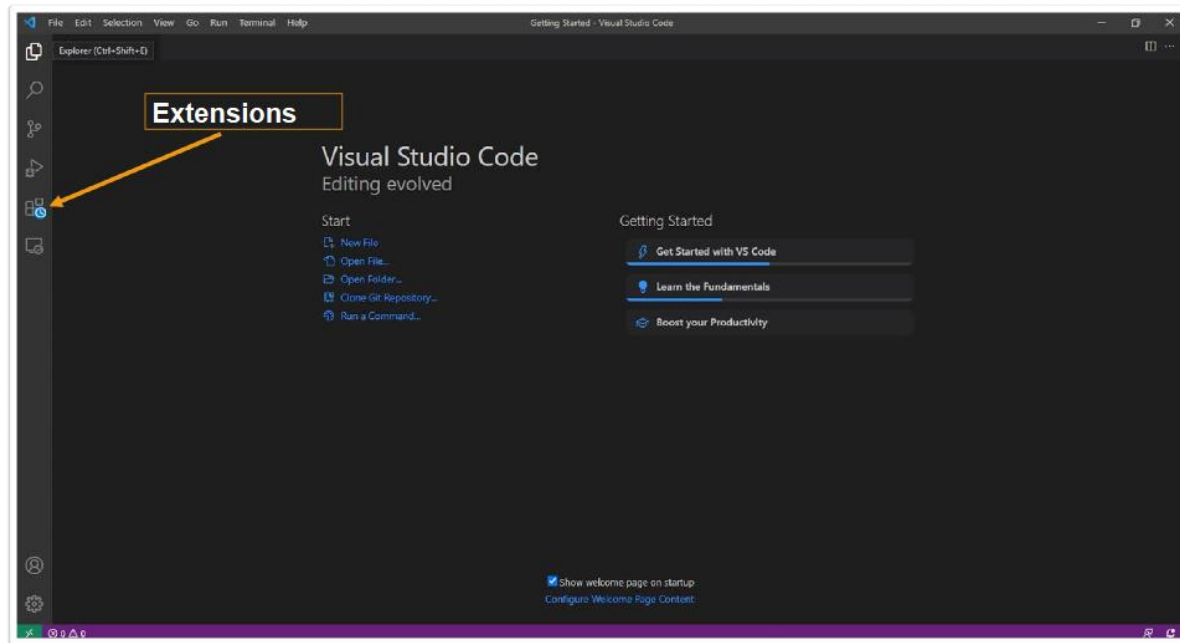
Une fois la tâche finie, vous pouvez déjà lancer l'éditeur.



Installation

Afin de faciliter la saisie de nos codes Python, nous allons installer l'extension « Python ».

1. Placer votre curseur à gauche et cliquer sur l'option « Extensions ».



Installation

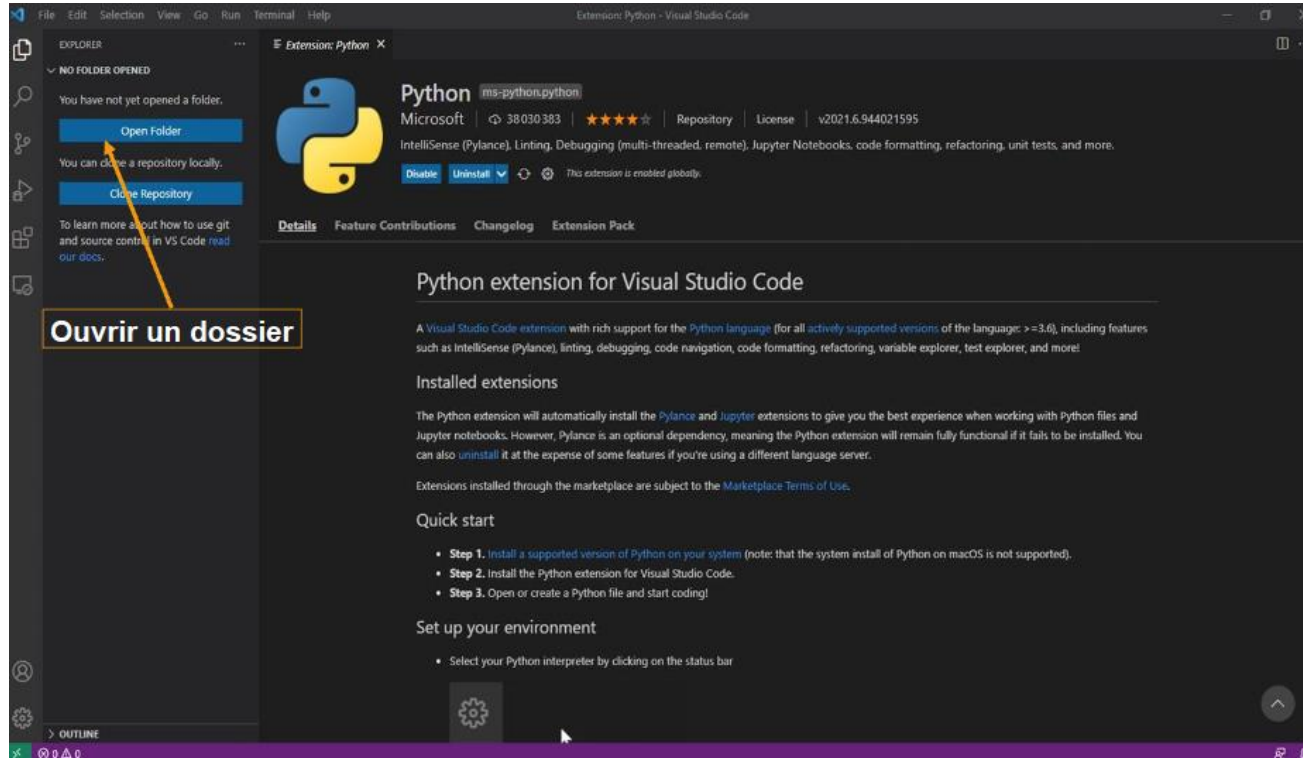
2. Dans la zone de recherche (coin supérieur gauche), vous cherchez et trouvez l'extension « Python » et ensuite vous l'installez.

Rechercher et sélectionner



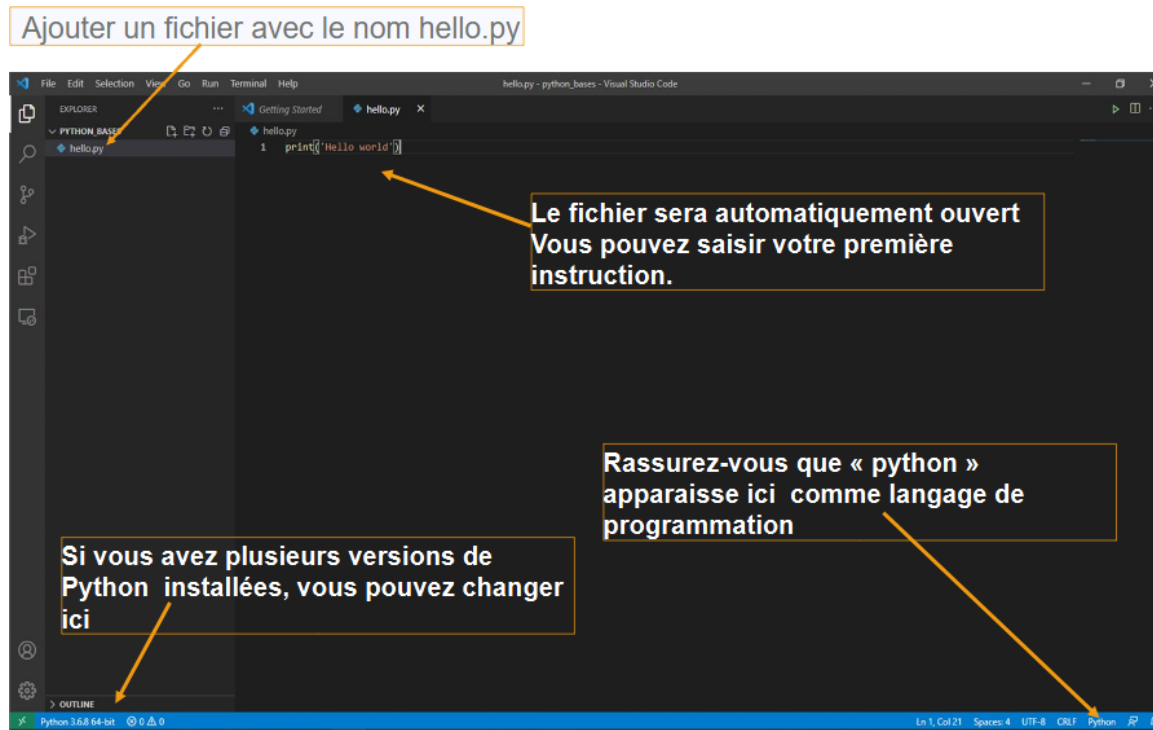
Installation

Nous allons créer notre premier fichier capable d'exécuter du code python. D'abord, il faudra sélectionner ou créer un dossier.



Installation

Une fois le dossier ouvert, nous créons un fichier avec l'extension python « .py »
Ensuite, nous nous rassurons si le fichier est bien prise en charge par l'extension
« python » que nous avons installé préalablement.



Installation

Comment exécuter Python ?



INVITE DE COMMANDE



CONSOLE PYTHON

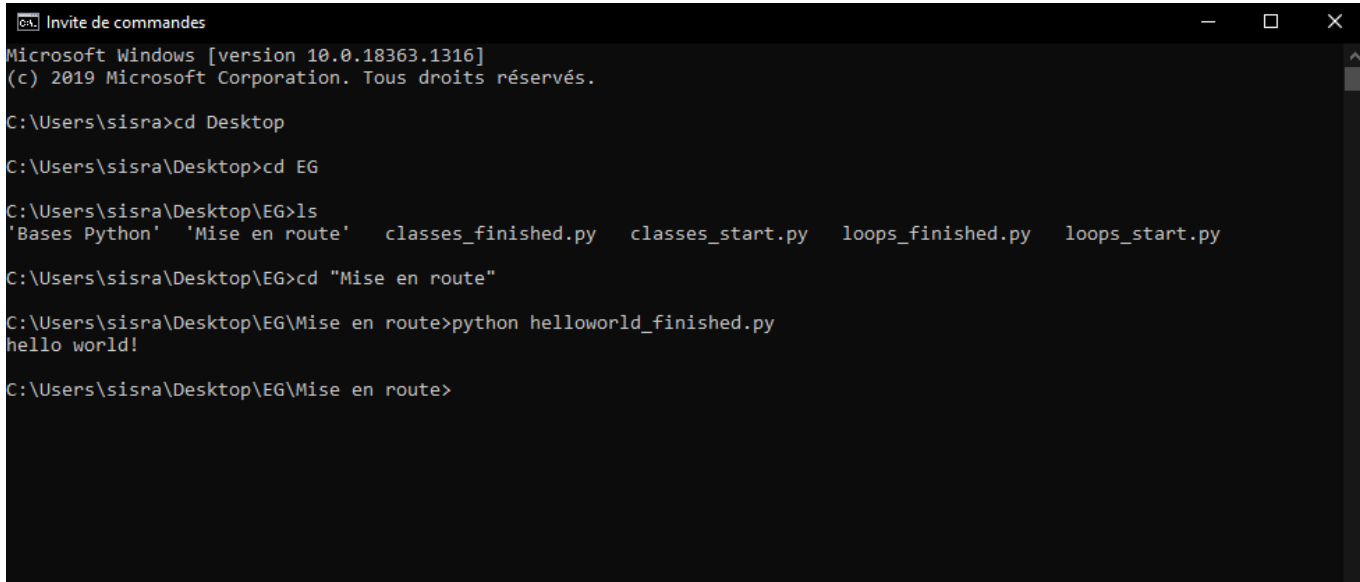


VS CODE

Installation

Nous pouvons tester différentes commandes du langage python grâce à l'invite de commande.

En ouvrant le terminal sur notre machine, il suffit de se rendre dans le répertoire où se trouve notre fichier .py puis saisir la commande python suivi du nom du fichier.

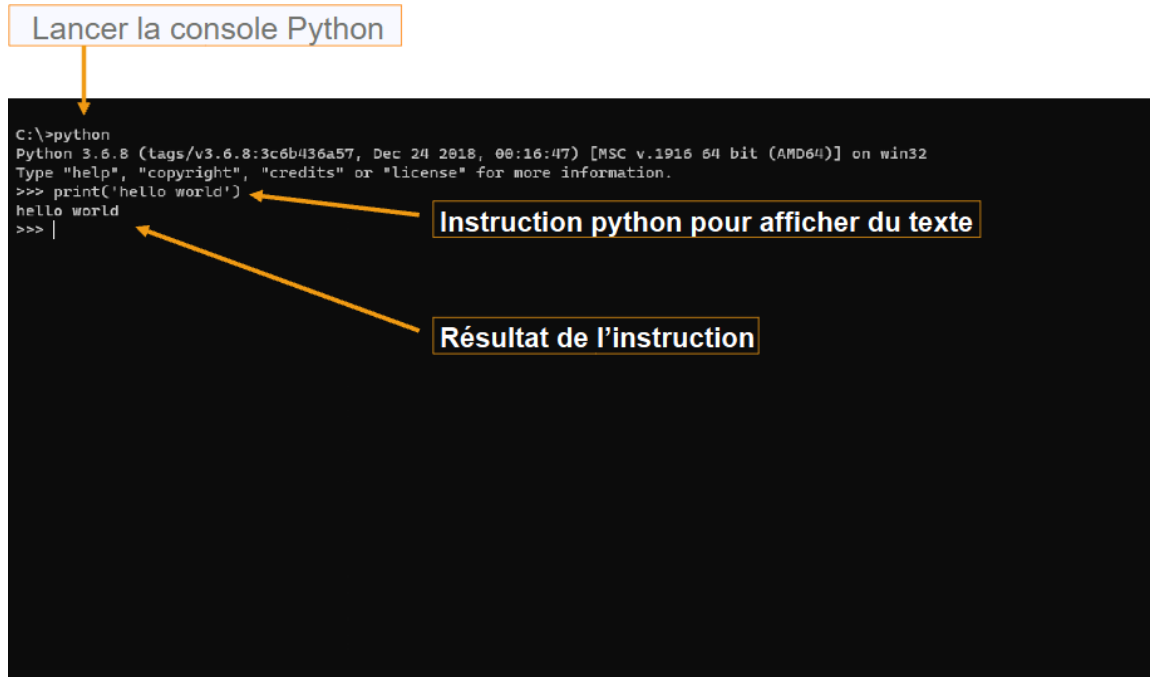


```
Invite de commandes
Microsoft Windows [version 10.0.18363.1316]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\sisra>cd Desktop
C:\Users\sisra\Desktop>cd EG
C:\Users\sisra\Desktop\EG>ls
'Bases Python'  'Mise en route'  classes_finished.py  classes_start.py  loops_finished.py  loops_start.py
C:\Users\sisra\Desktop\EG>cd "Mise en route"
C:\Users\sisra\Desktop\EG\Mise en route>python helloworld_finished.py
hello world!
C:\Users\sisra\Desktop\EG\Mise en route>
```

Installation

Nous pouvons tester différentes commandes du langage python grâce à la console. En ouvrant le terminal sur notre machine, il suffit de saisir la commande « python » pour accéder à la console Python.



```
C:\>python
Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('hello world')
hello world
>>> |
```

Lancer la console Python

Instruction python pour afficher du texte

Résultat de l'instruction

1. Premier pas



Premier pas

- Utilisons la console Python.



CALCULER AVEC PYTHON

```
>> 15 + 4    >> 5 ** 2
>> 2 - 9     >> 20 / 3
>> 4 * 3     >> 5 // 3
```



DONNEES, VARIABLES, AFFECTATION

```
>> a = 15
>> nom = "Kaiser"
```



AFFICHAGE DE LA VALEUR D'UNE VARIABLE

```
>> print(a)
>> print(nom)
```



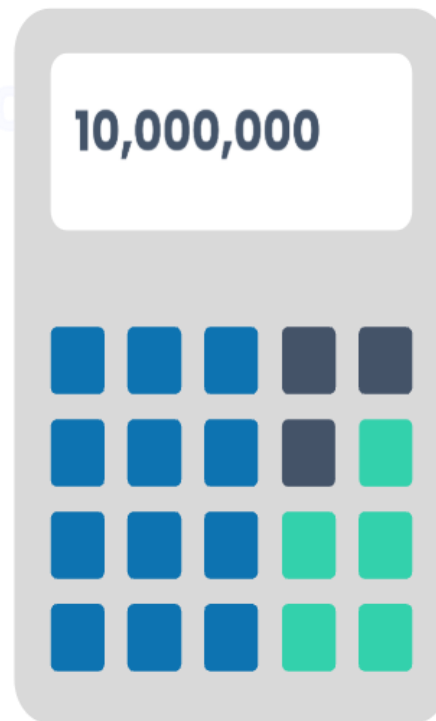
AFFECTATION MULTIPLE

```
>> x = y = 7    # affectation parallèle
>> a, b = 5, 18 # Affectation multiple
```



COMPOSITION

```
>> print(17 + 3) # ceci est un affichage
>> print("somme est de", 15 * 3 + 4)
```



Premier pas

- **Commentaire en python**

Les commentaires en Python commencent avec un caractère dièse, #, et s'étendent jusqu'à la fin de la ligne. Un commentaire peut apparaître au début d'une ligne ou à la suite d'un espace ou de code, mais pas à l'intérieur d'une chaîne de caractères littérale.

Il y a aussi les commentaires multi lignes commençant et finissant par triple quotes simples ou doubles. On les appelle les docstrings.

- **Quelques mots sur l'encodage des sources et UTF-8**

Pour pouvoir utiliser des caractères « spéciaux » (accents notamment) dans vos programmes (même dans les commentaires), vous devez dire à Python, de manière explicite, que vous souhaitez utiliser le codage de caractères UTF-8.

-*- coding: UTF-8 -*-

Premier pas

- `if __name__ == '__main__': # kesako ?`

`main()` n'existe pas en Python, comme on peut le trouver en C ou java par exemple. Il y a néanmoins un cas de figure où le fait de ne pas avoir ce genre de fonction peut être problématique : quand on inclut un module dans un autre, Python réalise un import de tout le contenu du module. Le problème, c'est que si on y place des instructions à l'extérieur de toute fonction ou méthode, elles seront exécutées systématiquement, même lors de l'inclusion du module, ce qui n'est pas terrible : on souhaite généralement importer les fonctions et classes, mais pas lancer les instructions.

C'est ici qu'intervient le test `if __name__ == '__main__': # (Programme Principal)`

- Python Extension Proposal : PEP-8 - style guide

PEP 8 (pour Python Extension Proposal) est un ensemble de règles qui permet d'homogénéiser le code et d'appliquer de bonnes pratiques. L'exemple le plus connu est la guerre entre les développeurs à savoir s'il faut indenter son code avec des espaces ou avec des tabulations. La PEP8 tranche : ce sont les espaces qui gagnent, au nombre de 4.

- Python Extension Proposal : PEP-8 - style guide

Encodage

Import

Indentation Lignes

Les lignes ne doivent pas dépasser 79 caractères. Séparer les fonctions et les classes à la racine d'un module par 2 lignes vides. Les méthodes par 1 ligne vide.

Les espaces

Les opérateurs doivent être entourés d'espaces. On ne met pas d'espace à l'intérieur des parenthèses, crochets ou accolades.

Premier pas

1. Assignez les valeurs respectives 3, 5, 7 à trois variables a, b, c.
2. Effectuez l'opération $a - b // c$. (division entière)
3. Interprétez le résultat obtenu ligne par ligne.

2. Imports et modules





Imports et modules

Un module est un fichier python que l'on veut importer.

Un package est un dossier dans lequel on stock des fichiers pythons (modules).

Ordonner les lignes d'import :

import de module

import du contenu du module

import de librairie standard

import de librairie tierce

import de votre projet

Imports et modules

```
import os                # import module de la librairie standard
import sys               # on groupe car même type
```

```
from itertools import islice    # import le contenu d'une partie d'un module
from collections import namedtuple # on groupe car même type
```

```
import requests           # import librairie tierce partie
import arrow              # on groupe car même type
```

```
from django.conf import settings    # contenu d'une partie d'un module tierce
from django.shortcuts import redirect # on groupe car même type
```

```
# import une fonction du projet
from myPackage.mySubPackage.myModule import myFunc
# import une classe du projet
from myPackage.mySubPackage.myModule import MyClass
```


3. Variables



Variables



**Python ne possède pas de syntaxe
Particulière pour créer ou “declarer” une
variable**

**Il existe quelques règles usuelles pour la
Denomination des variables**



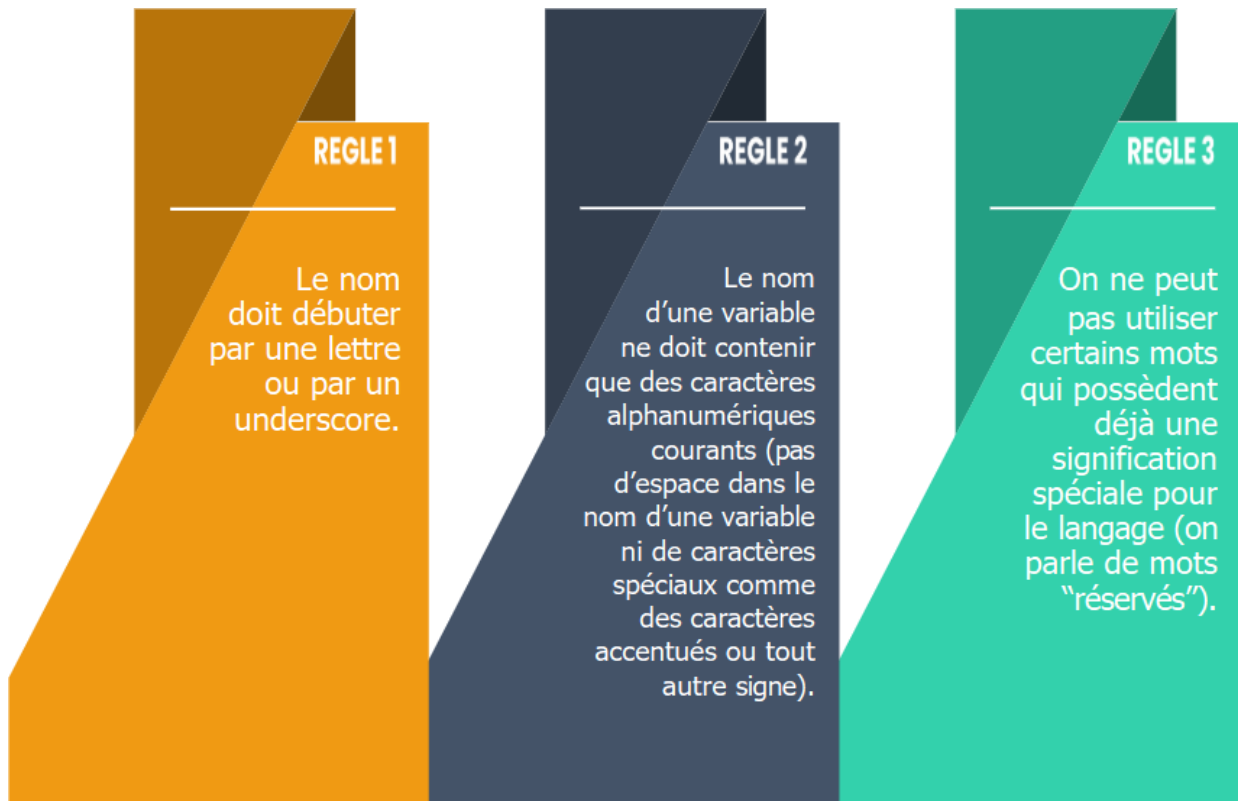
**Les variables vont pouvoir stocker
différents types de valeurs comme des
nombres, des chaines de caractères,
des booléens, et plus encore.**

**la casse est significative dans les noms
de variables**



Variables

- Règle de nommage



Variables

- Les types



Integer

```
>>> a = 15  
>>> type(a)
```



String

```
>>> texte1 = 'Les œufs durs.'  
>>> texte2 = ' "Oui", répondit-il,'  
>>> texte3 = "j'aime bien "  
>>> print(texte1, texte2, texte3)
```



Float

```
>>> b = 16.0  
>>> type(b)
```



Booleen

```
>>> c = True  
>>> type(c)
```

Variables

- Les chaînes de caractères

Une donnée de type string peut se définir en première approximation comme une suite quelconque de caractères. Dans un script python, on peut délimiter une telle suite de caractères, soit par des apostrophes (simple quotes), soit par des guillemets (double quotes).

- Le caractère spécial « \ » (antislash)

En premier lieu, il permet d'écrire sur plusieurs lignes une commande qui serait trop longue pour tenir sur une seule (cela vaut pour n'importe quel type de commande). À l'intérieur d'une chaîne de caractères, l'antislash permet d'insérer un certain nombre de codes spéciaux (sauts à la ligne, apostrophes, guillemets, etc.). Exemples :

```
>>> print("ma\tpetite\nchaine")    # affiche ma      petite
                                     # chaîne
```

Variables

- Notion de formatage

%s : string
%d : decimal integer
%f : float
%g : generic number

```
>>> n = "Celine"  
>>> a = 42  
>>> print("nom : %s - age %d" %(n, a))
```

.format()

```
>>> print("nom : {1} - age : {0}".format(n, a))  
>>> print(f"nom : {n} - age : {a}")
```

- Typage dynamique fort

Python est fortement typé dynamiquement.

Un typage fort signifie que le type d'une valeur ne change pas de manière inattendue. Une chaîne contenant uniquement des chiffres ne devient pas par magie un nombre, comme cela peut arriver en Perl. Chaque changement de type nécessite une conversion explicite.

Le typage dynamique signifie que les objets d'exécution (valeurs) ont un type qui peut être changer, par opposition au typage statique où les variables ont un type interchangeable.

Variables

- Typage dynamique fort

```
>>> points = 3.2
>>> print("Tu as " + points + " points !")
>>> points = int(points)
```

```
>>> print("Tu as " + points + " points !")
>>> points = str(points)
>>> print("Tu as " + points + " points !")
```

```
# points est du type float (nombre décimal)
# Génère une erreur de typage
# points est maintenant du type int (entier),
# sa valeur est arrondie à l'unité inférieure
# (ici 3)
# Génère une erreur de typage
# points est maintenant du type str (string)
# Plus d'erreur de typage, affiche
# "Tu as 3 points !"
```


- Les collections

Les chaînes de caractères que nous avons abordées constituaient un premier exemple de données composites. On appelle ainsi les structures de données qui sont utilisées pour regrouper de manière structurée des ensembles de valeurs.

- Les collections



Listes

```
#Déclarer une list avec la possibilité de modifier son contenu  
list_data = [1, 2, 3, 4]
```



Tuples

```
#Déclarer un tuple sans la possibilité de modifier son contenu  
tuple_data = (1, 2, 3, 4)
```



Sets

```
# Un set ne contient pas de doublon.  
# ici, le deuxième élément "pierre" sera ignoré.  
mon_set = {"pascal", "pierre", "paul", "pierre"}
```



Dictionnaires

```
#Construire un dictionnaire  
dico = {}  
dico['computer'] = 'ordinateur'  
dico['mouse'] = 'souris'  
dico['keyboard'] = 'clavier'  
print("dico")  
dico = {'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
```

4. Boucles et Conditions



Boucles et conditions

- Limites des blocs d'instructions

Avec Python, elles sont définies par la mise en page.

Vous devez utiliser les sauts à la ligne et l'indentation, mais en contrepartie vous n'avez pas à vous préoccuper d'autres symboles délimiteurs de blocs.

En définitive, Python vous force donc à écrire du code lisible, et à prendre de bonnes habitudes que vous conserverez lorsque vous utiliserez d'autres langages.

Python ne peut exécuter un programme que si sa syntaxe est parfaitement correcte. Dans le cas contraire, le processus s'arrête et vous obtenez un message d'erreur.

```
def power(num, x=1):  
    result = 1  
    for i in range(x):  
        result = result * num  
    return result
```

Boucles et conditions

- Répétitions en boucle – L’instruction while

```
# boucle while
x = 0
while (x < 5):
    print(x)
    x += 1
```

- Répétitions en boucle – L’instruction For

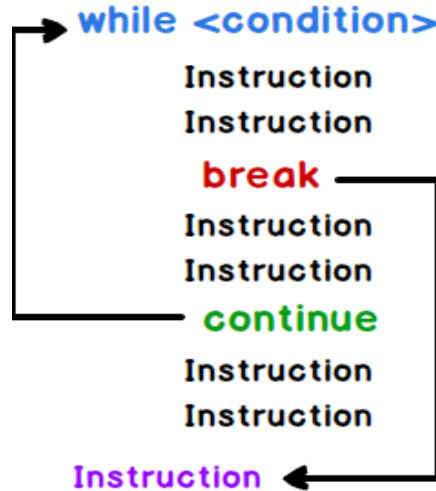
```
# boucle for
for x in range(5, 10):
    print(x)
```

Boucles et conditions

- Instruction continue et break

Break est utilisé pour quitter une boucle for/while en cours d'exécution.

Continue est utilisé pour ignorer la suite du bloc actuel et revenir à l'instruction for/while.



Boucles et conditions

- Instruction conditionnelle If

```
# if, elif, else
if x < y:
    st = "x est plus petit que y"
elif x == y:
    st = "x est égale à y"
else:
    st = "x est plus grand que y"
print(st)
```

Remarques :

Les boucles et conditions sont imbriquables.

Les parenthèses ne sont pas obligatoire mais donne de la visibilité.

Lorsqu'une seule instruction est exécutée, on peut la mettre sur la même ligne que le if, elif ou else.

Boucles et conditions

- Opérateurs de comparaison et booléen

$X == Y$ # égale

$X != Y$ # différent

$X < Y$ # inférieur

$X > Y$ # supérieur

$X >= Y$ # supérieur ou égale

$X <= Y$ # inférieur ou égale

$X \text{ or } Y$ # ou

$X \text{ and } Y$ # et

$\text{not } X$ # négation de X

Les booléens ont la valeur True ou False avec une majuscule

Boucles et conditions

- Condition ternaire

Permet d'affecter une valeur selon une condition de type if else en une seule ligne de code.

- Compréhension de liste

La compréhension de liste permet d'écrire sur une seule ligne une combinaison d'instruction de condition if (ou if else) avec une boucle for.

- Nouvelle structure conditionnelle depuis python v3.10

```
# match case
match (x):
    case 1:
        y = 0
    case 2:
        y -= 1
    case _:
        y += x + 3
```

5. Les fonctions



Les fonctions

- Quelques fonctions prédéfinies / natives (builtin)
 - `abs(x)` # retourne la valeur absolu de x.
 - `all(iterable)` # retourne True si toutes les valeurs de l'itérable sont True.
 - `any(iterable)` # retourne True si au moins une valeur de l'itérable est True.
 - `bin(int)` # convertit nombre entier en chaîne de caractère binaire.
 - `hex(int)` # convertit nombre en valeur hexadécimale.
 - `len(obj)` # retourne la longueur de l'objet.
 - `list(obj)` # cast l'objet au format liste.
 - `map(fct, obj)` # applique une transformation à tout les éléments d'un itérable.
 - `filter(fct, list)` # filtre avec un prédicat les éléments d'un itérable et retourne un # booléen.

Les fonctions

- Fonctions natives de l'objet str

- `str.capitalize()` # retourne la string avec une majuscule en début de phrase
- `str.title()` # retourne la string avec une majuscule en début de chaque
 # mot
- `str.upper()` # retourne la string en majuscule
- `str.lower()` # retourne la string en minuscule
- `str.strip()` # retourne la string str en supprimant les espaces avant et
 # après la phrase
- `str.count(x)` # retourne le nombre d'occurrence de x dans str
- `str.endswith("x")` # retourne True si la str fini par 'x'
- `str.startswith("x")` # retourne True si la str commence par 'x'
- `str.find(x)` # retourne l'index de la première occurrence de x (-1 si
 # n'existe pas)

Les fonctions

- **Fonction print()**

Elle permet d'afficher n'importe quel nombre de valeurs fournies en arguments (c'est-à-dire entre les parenthèses). Par défaut, ces valeurs seront séparées les unes des autres par un espace, et le tout se terminera par un saut à la ligne.

```
>>> print("Bonjour", "à", "tous", sep = "*") Bonjour*à*tous
```

```
>>> print("Bonjour", "à", "tous", sep = "") Bonjouràtous
```

- **Fonction input() : Interaction avec l'utilisateur**

Cette fonction provoque une interruption dans le programme courant. L'utilisateur est invité à entrer des caractères au clavier et à terminer avec <Enter>. Lorsque cette touche est enfoncée, l'exécution du programme se poursuit, et la fonction fournit en retour une chaîne de caractères correspondant à ce que l'utilisateur a saisi.

Les fonctions

```
prenom = input("Entrez votre prénom : ")  
print("Bonjour,", prenom)
```

OU

```
print("Veuillez entrer un nombre positif quelconque : ")  
ch = input()  
nn = int(ch) # conversion de la chaîne en un nombre entier  
print("Le carré de", nn, "vaut", nn ** 2)
```

Les fonctions

- **Fonction**

L'approche efficace d'un problème complexe consiste souvent à le décomposer en plusieurs sous-problèmes plus simples, d'autre part, il arrivera souvent qu'une même séquence d'instructions doive être utilisée à plusieurs reprises dans un programme.

```
def nomDeLaFonction(liste de paramètres):  
    ...  
    bloc d'instructions  
    ...
```

- **Les paramètres des fonctions**

Une fonction peut avoir 0 ou plusieurs paramètres. Ces paramètres peuvent être des variables (int, string, list...) mais aussi d'autres fonctions.

Les fonctions

- Variable locale vs variables globales

Lorsque nous définissons des variables à l'intérieur du corps d'une fonction, ces variables ne sont accessibles qu'à la fonction elle-même. On dit que ces variables sont des variables locales à la fonction.

Les variables définies à l'extérieur d'une fonction sont des variables globales. Leur contenu est « visible » de l'intérieur d'une fonction, mais la fonction ne peut pas le modifier.

- Fonction récursive

Une fonction récursive est une fonction qui s'appelle elle-même. Attention à bien mettre une condition pour sortir de l'appel récursif, sinon on provoque une boucle infinie.

Exemple la fonction factorielle(x)

```
def factorielle(n):  
    if n < 2:  
        return 1  
    else:  
        return n * factorielle(n-1)
```

```
def factorielle(n):  
    return 1 if n < 2 else n * factorielle(n-1)
```

Les fonctions

- Fonction Lambda

Pour Python, c'est la seule façon d'écrire une fonction anonyme. Ceci est particulièrement utile pour la programmation fonctionnelle. En effet, une fonction peut être directement écrite dans un appel de fonction sans avoir à la définir au préalable.

```
>>> list(map(lambda x: x ** 2, range(10)))  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Les fonctions

- Fonction Lambda

Bien que les fonctions lambda soient utilisées dans le but de créer des fonctions anonymes, on peut décider tout de même de leur donner un nom :

```
>>> f = lambda x: x ** 2
>>> f(5)
25
>>> list(map(f, range(10)))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Syntaxe pour passer une variable à l'exécution : `lambda x: monCalculAvecX`

Syntaxe pour passer une variable à la création : `lambda x = x: monCalculAvecX`

6. Les fonctions



Les exceptions

- Les exceptions et la gestion des erreurs

Toutes les erreurs qui se produisent lors de l'exécution d'un programme Python sont représentées par une exception. Une exception est un objet qui contient des informations sur le contexte de l'erreur. Lorsqu'une exception survient et qu'elle n'est pas traitée alors elle produit une interruption du programme et elle affiche sur la sortie standard un message ainsi que la pile des appels (stacktrace). La pile des appels, présente dans l'ordre, la liste des fonctions et des méthodes qui étaient en cours d'appel au moment où l'exception est survenue.

Parfois un programme est capable de traiter le problème à l'origine de l'exception. Par exemple si le programme demande à l'utilisateur de saisir un nombre et que celui-ci saisit une valeur erronée, le programme peut simplement demander à l'utilisateur de saisir une autre valeur plutôt que de faire échouer le programme.

7. Le débogage



Le débogage

Pour déboguer lorsque nous ne pouvons pas le faire via un IDE telle de VS Code ou PyCharm, nous pouvons déboguer via la console Python (cf : <https://docs.python.org/fr/3/library/pdb.html>).

Avant la v3.7 nous devons écrire dans notre code les lignes suivantes :

```
import pdb          # import du module pdb
pdb.set_trace()     # instruction du module pdb
```

A partir de la version 3.7 nous devons écrire :

```
breakpoint()        # équivalent au pdb.set_trace() mais intégré à python
```

Le programme s'arrête quand `set_trace()` ou `breakpoint()` est trouvé.

A partir de ce moment on peut entrer dans la console du terminal différentes commandes telle que :

Le débogage

c(ontinue)	: Continuer l'exécution
q(uit)	: Quitter le débogueur / l'exécution
n(ext)	: Passer à la ligne suivante dans la même fonction
s(tep) fonction	: Passer à la ligne suivante dans cette fonction ou une appelée
l(list)	: Liste le code à la position courante
u(p)	: Monte à la pile d'appel
d(own)	: Descend à la pile d'appel
bt	: Affiche la pile d'appel
a(rgs)	: Affiche la liste d'arguments de la fonction courante.
!instruction_python	: Remplacer instruction_python par l'instruction voulue pour l'exécuter.
b(reak)	: Sans argument, liste tous les arrêts, incluant pour chaque point d'arrêt, le nombre de fois qu'un point d'arrêt a été atteint, le nombre d'ignore, et la condition associée le cas échéant.

Le débogage

A partir de la console python on peu exécuter la commande :

```
pdb.run('mymodule.test()') # fichier mymodule.py, fonction test()
```

Il est aussi possible de déboguer a partir de l'invite de commande :

```
python -m pdb myscript.py
```

8. Les fichiers



- Le module OS

Le module OS contient de nombreuses fonctions intéressantes pour l'accès au système d'exploitation.

```
>>> import os
>>> os.getcwd()
>>> os.chdir("C:\\Users\\User\\Documents")
>>> os.path.dirname(__file__)
```

- Les fichiers

Sous Python, l'accès aux fichiers est assuré par l'intermédiaire d'un objet-interface particulier, que l'on appelle objet-fichier. On crée cet objet à l'aide de la fonction intégrée `open()`. Celle-ci renvoie un objet doté de méthodes spécifiques, qui vous permettrons de lire et écrire dans le fichier.

Ecriture

```
>>> objetfichier = open('Monfichier','a')
>>> objetfichier.write('Bonjour, fichier !')
>>> objetfichier.write("Quel beau temps!")
>>> objetfichier.close()
```

Lecture

```
>>> of = open('Monfichier', 'r')
>>> t = of.read()
>>> print(t)
Bonjour, fichier !Quel beau temps !
>>> of.close()
```

- Les fichiers

La fonction `open()` attend deux arguments minimum, qui doivent tous deux être des chaînes de caractères. Le premier argument est le nom du fichier à ouvrir, et le second est le mode d'ouverture.

- 'a' indique qu'il faut ouvrir ce fichier en mode « ajout » (append), ce qui signifie que les données à enregistrer doivent être ajoutées à la fin du fichier, à la suite de celles qui s'y trouvent éventuellement déjà.
- 'w' (pour write), utilisé aussi pour l'écriture mais lorsqu'on utilise ce mode, Python crée toujours un nouveau fichier (vide), et l'écriture des données commence à partir du début de ce nouveau fichier. S'il existe déjà un fichier de même nom, celui-ci est effacé au préalable.

- Les fichiers
- La méthode `write()` réalise l'écriture proprement dite. Les données à écrire doivent être fournies en argument. Chaque nouvel appel de `write()` (en mode `a`) continue l'écriture à la suite de ce qui est déjà enregistré.
- 'r': Ouverture en lecture.
- La méthode `read()` (en mode `r`) permet de lire le contenu d'un fichier dans son ensemble.
- La méthode `readline()` (en mode `r`) lit qu'une ligne par appel de cette instruction.
- La méthode `readlines()` (en mode `r`) lit les lignes individuellement dans une liste.
- La méthode `seek(x)` (en mode `r`) permet de replacer le curseur à la position `x` voulue.
- La méthode `close()` referme le fichier dans n'importe qu'elle mode.

9. Les regex



Les regex

- Expressions régulières

Pour faire des tests, utilisez le site <https://regex101.com>

Les expressions régulières représentent les motifs qu'on recherche dans une chaîne de caractères. Par exemple, on peut chercher dans un fichier :

Les lignes qui contiennent un numéro de téléphone

Les lignes qui débutent par une adresse IP

On peut chercher dans une chaîne de caractère :

- Une séquence qui commence par « ex » et se termine par « le »
- Les mots de quatre caractères de long
- Les mots sans majuscules

On utilise les symboles suivants qui ont une signification spécifique chacune !

. ^ \$ * + ? { } [] \ ()

- Les quantificateurs

- + : Le résultat peut contenir une ou plusieurs occurrences du caractère qui le précède.
- * : Le résultat peut contenir zéro ou plusieurs occurrences du caractère qui le précède.
- ? : Le résultat peut inclure zéro ou une occurrence du caractère qui le précède.
- {x}: Précise le nombre x de répétitions des chaînes de caractères qui le précède.
- [] : Obtient le résultat sur plusieurs occurrences.
- () : Créer des groupes.
- | : Correspond à l'expression OU. On peut s'en servir pour trouver une correspondance entre plusieurs expressions.
- : Sert à regrouper des caractères dans un intervalle donné.

```
>>> re.findall("[0-9]+", "Bonjour 111 Aurevoir 222")  
>>> bool(re.match("^S", "XAVIER"))
```

Les regex

- Les métacaractères

- ^ : Précise que le résultat doit commencer par la chaîne de caractères qui le suit.
- \$: Précise que le résultat doit se terminer par la chaîne de caractère qui le précède.
- . : Correspond à n'importe quel caractère sauf le saut de ligne.

- Tokens

\d : [0-9]

\w : [a-zA-Z0-9_]

\s : correspond à tous les espaces dans la chaîne de caractères

\. : juste les points

\b : fin de mots

Les regex

- Fonctions usuelles

- `re.compile(motif)` : Compile un motif (expression régulière) utilisable par les autres fonctions telle qu `search`, `findall`, etc ...
- `re.search(motif, string)` : Renvoi la première correspondance du motif trouvée dans la string.
- `re.findall(motif, string)` : Renvoie une liste de toutes les correspondances du motif trouvées dans la string.
- `re.match(motif, string)` : Renvoi la correspondance trouvée si zéro ou plus caractères en début de string correspondent au motif.
- `re.sub(motif, chaîne, string)` : Remplace par chaîne les occurrences trouvées dans la string (sans chevauchement) et renvoi le résultat. Si le motif n'est pas trouvé, string est renvoyée inchangée.

10. Les classes



- **Orienté objet : ça veut dire quoi ?**

Globalement, les langages de programmation objet implémentent le paradigme de programmation orientée objet (POO). Ce paradigme consiste en la réunion des données et des traitements associées à ces données au sein d'entités cohérentes appelées objets. Python est un langage objet composé de classes. Une classe représente un « moule » permettant de créer des objets (instances), et regroupe les attributs et méthodes communes à ces objets.

- Les classes

L'orienté objet facilite beaucoup dans la conception, la maintenance, la réutilisabilité des éléments (objets). Le paradigme de POO permet de tirer profit de classes parents et de classes enfants (phénomène d'héritage), etc.

Tout objet donné possède deux caractéristiques :

- Son état courant (attributs)
- Son comportement (méthodes)

En approche orienté objet on utilise le concept de classe, celle-ci permet de regrouper des objets de même nature.

Une classe est un moule (prototype) qui permet de définir les attributs (variables) et les méthodes (fonctions) de tous les objets de cette classe.

Les classes sont les principaux outils de la POO. Ce type de programmation permet de structurer les logiciels complexes en les organisant comme des ensembles d'objets qui interagissent entre eux et avec le monde extérieur.

Les classes

Exemple de classe :

```
#Class avec COstructor
class Velo:
    roues = 2

    def __init__(self, marque, prix, poids):
        self.marque = marque
        self.prix = prix
        self.poids = poids

    def rouler(self):
        print("Wouh, ça roule mieux avec un vélo {} !".format(self.marque))
```

- Quelques remarques importantes
 - Tous les attributs et méthodes des classes Python sont « publics » au sens de C++, parce que « nous sommes tous des adultes ! » (citation de Guido von Rossum, créateur de Python).
 - Le constructeur d'une classe est une méthode spéciale qui s'appelle `__init__()`.
 - En Python, on n'est pas tenu de déclarer tous les attributs de la classe comme d'autres langages : on peut se contenter de les initialiser dans le constructeur !
 - Toutes les méthodes prennent une variable `self` comme premier argument. Cette variable est une référence à l'objet manipulé.
 - Python supporte l'héritage simple et l'héritage multiple. La création d'une classe fille est relativement simple, il suffit de préciser entre parenthèses le nom de la classe mère lors de la déclaration de la classe fille.

11. Modules utiles



Modules

Il existe un grand nombre de modules préprogrammés qui sont fournis d'office avec Python. Vous pouvez en trouver d'autres chez divers fournisseurs. Souvent on essaie de regrouper dans un même module des ensembles de fonctions apparentées, que l'on appelle des bibliothèques.

Module Math

```
from math import *  
sqrt()  
sin()  
cos()  
pi  
...
```

Module DateTime

```
from datetime import date  
from datetime import time  
from datetime import datetime
```

12. Base de données



- SQLite

La bibliothèque standard de Python inclut un moteur de base de données relationnelles SQLite, qui a été développé en C, et implémente le standard SQL. Cela signifie donc que vous pouvez écrire en Python une application contenant son propre SGBDR intégré, sans qu'il soit nécessaire d'installer quoi que ce soit d'autre. Les instructions d'interaction à la BDDR sont standardisées (cf : PEP 249 sur python.org).

- SQLite

```
>>> import sqlite3
>>> fichierDonnees = "E:/python3/essais/bd_test.sq3"
>>> conn = sqlite3.connect(fichierDonnees)
>>> cur = conn.cursor()
>>> cur.execute("CREATE TABLE membres (age INTEGER, nom TEXT, taille REAL)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(21,'Dupont',1.83)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(15,'Blumâr',1.57)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(18,'Özémir',1.69)")
>>> conn.commit()
>>> cur.close()
>>> conn.close()
```

- **Postgres**

Pour Postgres il faut installer le module via pip `install psycopg2`

Puis importer le module `psycopg2`.

Ensuite, le code ressemble beaucoup à celui de SQLite.

```
>>> import psycopg2
>>> psycopg2.connect(dbname = 'template1', user = 'dbuser', password = 'dbpass', host =
'localhost', port = '5432')
>>> cur = conn.cursor()
>>> cur.execute("CREATE TABLE membres (age INTEGER, nom TEXT, taille REAL)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(21,'Dupont',1.83)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(15,'Blumâr',1.57)")
>>> cur.execute("INSERT INTO membres(age,nom,taille) VALUES(18,'Özémir',1.69)")
>>> conn.commit()
>>> cur.close()
>>> conn.close()
```


13. Gestionnaire de paquets



Gestionnaire de paquets

- PIP, Gestionnaire de paquets

Dans l'invite de commande de Windows :

<code>pip install monPackage</code>	<code># installation d'un paquet dans l'environnement actuel</code>
<code>pip freeze > requirements.txt</code>	<code># liste les packages de l'environnement actuel dans le fichier requirements.txt</code>
<code>pip install -r requirements.txt</code>	<code># installation des paquets listés dans le fichier requirements.txt</code>

Remarque : les dépendances ne sont pas automatiquement mise à jour à ce moment là. pip est le successeur du gestionnaire de paquets easy_install.

Ces gestionnaires de paquets se base sur le dépôt de paquets PyPi (Python Package Index).

Gestionnaire de paquets

- Créer un exécutable avec PyInstaller

`pip install PyInstaller` # installation de PyInstaller

PyInstaller -F chemin_complet\nom_fichier.py # -F pour faire un exécutable indépendant

Le fichier exécutable sera dans un dossier « dist » créer à l'emplacement du script.
--distpath *chemin* permet de changer le dossier où se situe le fichier exécutable.
--noconsole permet de ne pas lancer la console quand on exécute le programme (utile pour les interfaces graphiques)
--onefile permet de créer un exécutable intégrant tout, même python.

La différence entre -F et --onefile c'est que --onefile, décompresse pour installer les dépendances à l'exécution du fichier. -F exécute sans faire d'installation.

14. Environnements virtuels



Environnements virtuels

Installer Python c'est simple.

Mais installer un environnement homogène et performant devient laborieux.
(un environnement scientifique par exemple) :

- Il faut commencer par isoler son environnement de travail du système.
- Il existe une multitude de librairies et gérer leurs dépendances peut s'avérer difficile.
- Il faut les compiler spécifiquement pour votre système si vous souhaitez exploiter toute la puissance de calcul des processeurs modernes.

Les environnements virtuels Python permettent d'avoir des installations de Python isolées du système et séparées les unes des autres. Cela permet de gérer plusieurs projets sur sa machine de développements, certains utilisant des modules de versions différentes, voir même des versions différentes de Python.

Nous utiliserons le module virtualenv pour la suite du cours.

Environnements virtuels

Installation du package virtualenv :

pip install virtualenv # n'est plus nécessaire car le virtualenv est fourni avec Python

Création d'un environnement virtuel :

Python -m venv monvenv

Activation d'un environnement virtuel :

Sous Windows :

cd monvenv

.\Scripts\activate

Sous Linux :

. venv/bin/activate

le « . » sous linux demande à exécuter la commande dans le shell courant au lieu d'un autre shell pour ne pas perdre les variables d'environnements.

Environnements virtuels

Remarque : lorsque l'environnement virtuel est actif, nous avons le nom de l'environnement virtuel entre parenthèses en début de ligne de console.

Désactivation d'un environnement virtuel :

Sous Windows :

`.\Scripts\deactivate`

Sous Linux :

`. venv/bin/deactivate`

Remarque :

PyCharm créer directement un environnement virtuel à la création d'un nouveau projet.

Menu, file, settings, python interpreter : la fenêtre indique les librairies installées et leurs versions. Cela indique aussi si une nouvelle version est disponible.



15. CGI

- Common Gateway Interface

C'est un vieux protocole mais qui fonctionne toujours et qui ne nécessite pas d'installation pour vérifier si le site qu'on veut créer fonctionne bien. Il permet d'appeler un script python côté serveur en passant par une URL dans le navigateur côté client. Les scripts doivent être dans un dossier nommé cgi-bin.

Pour démarrer un serveur web et permettre d'accéder à un site web :

Aller dans le dossier parent de cgi-bin et exécuter :

```
python -m http.server --cgi
```

- Common Gateway Interface

Pour lancer un « monscript.py », se rendre sur la page web :
<http://localhost:8000/cgi-bin/monscript.py>

Pour arrêter le serveur faire Ctrl+C (comme pour stopper n'importe quel script)

Remarque : Sous linux rendre le script exécutable via un chmod par exemple pour accéder au fichier qui contient du html.

16. Tkinter



- Interface graphique avec Python

Le domaine des interfaces graphiques (ou GUI, Graphical User Interfaces) est extrêmement complexe. Chaque système d'exploitation peut en effet proposer plusieurs « bibliothèques » de fonctions graphiques de base, auxquelles viennent fréquemment s'ajouter de nombreux compléments, plus ou moins spécifiques de langages de programmation particuliers. Tous ces composants sont généralement présentés comme des classes d'objets, dont il vous faudra étudier les attributs et les méthodes. Avec Python, la bibliothèque graphique la plus utilisée jusqu'à présent est la bibliothèque Tkinter, qui est une adaptation de la bibliothèque Tk développée à l'origine pour le langage Tcl. Tkinter est un environnement graphique de type événementiel fourni par défaut avec python. Cela permet à l'utilisateur de saisir des informations de façon non séquentielle.

- Interface graphique avec Python

Plusieurs autres bibliothèques graphiques fort intéressantes ont été proposées pour Python : wxPython, pyQT, pyGTK, etc. Il existe également des possibilités d'utiliser les bibliothèques de widgets Java et les MFC de Windows.

Il est aisé de construire différents modules Python, qui contiendront des scripts, des définitions de fonctions, des classes d'objets, etc... On peut alors importer tout ou partie de ces modules dans n'importe quel programme, ou même dans l'interpréteur fonctionnant en mode interactif. Pour utiliser Tkinter, il suffit de l'importer dans votre fichier python.

- Les 15 classes principales (widget) du module Tkinter

Button : Un bouton classique, à utiliser pour provoquer l'exécution d'une commande quelconque.

Canvas : Un espace pour disposer divers éléments graphiques. Ce widget peut être utilisé pour dessiner, créer des éditeurs graphiques, et aussi pour implémenter des widgets personnalisés.

Checkbutton : Une « case à cocher » qui peut prendre deux états distincts (la case est cochée ou non). Un clic sur ce widget provoque le changement d'état.

Entry : Un champ d'entrée, dans lequel l'utilisateur du programme pourra insérer un texte quelconque à partir du clavier.

- Les 15 classes principales (widget) du module Tkinter

Frame : Un cadre rectangulaire dans la fenêtre, où l'on peut disposer d'autres widgets. Cette surface peut être colorée. Elle peut aussi être décorée d'une bordure.

Label : Un texte (ou libellé) quelconque (éventuellement une image).

Listbox : Une liste de choix proposés à l'utilisateur, généralement présentés dans une sorte de boîte. On peut également configurer la Listbox de telle manière qu'elle se comporte comme une série de « boutons radio » ou de cases à cocher.

Menu : Un menu. Ce peut être un menu déroulant attaché à la barre de titre, ou bien un menu « pop up » apparaissant n'importe où à la suite d'un clic.

- Les 15 classes principales (widget) du module Tkinter

Menubutton : Un bouton-menu, à utiliser pour implémenter des menus déroulants.

Message : Permet d'afficher un texte. Ce widget est une variante du widget Label, qui permet d'adapter automatiquement le texte affiché à une certaine taille ou à un certain rapport largeur/hauteur.

Radiobutton : Représente (par un point noir dans un petit cercle) une des valeurs d'une variable qui peut en posséder plusieurs. Cliquer sur un « bouton radio » donne la valeur correspondante à la variable, et « vide » tous les autres boutons radio associés à la même variable.

- Les 15 classes principales (widget) du module Tkinter

Scale : Vous permet de faire varier de manière très visuelle la valeur d'une variable, en déplaçant un curseur le long d'une règle.

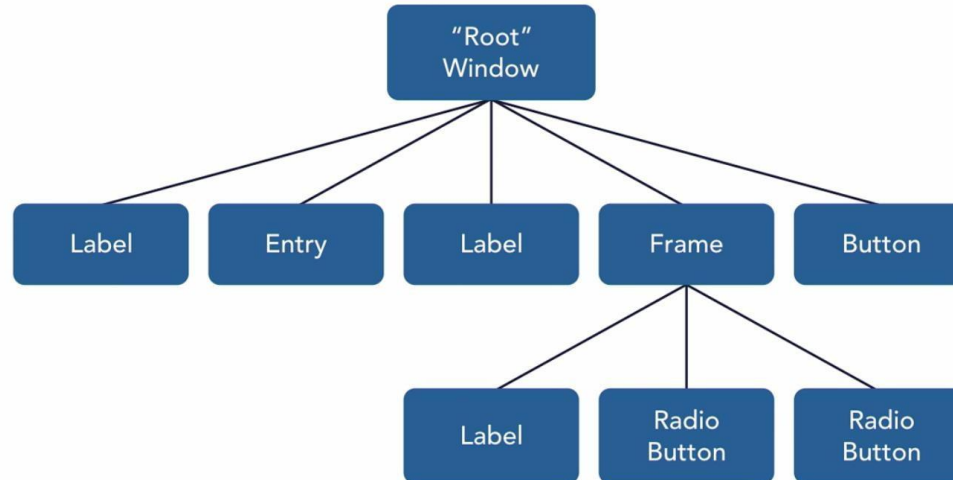
Scrollbar : « ascenseur » ou « barre de défilement » que vous pouvez utiliser en association avec les autres widgets : Canvas, Entry, Listbox, Text.

Text : Affichage de texte formaté. Permet aussi à l'utilisateur d'éditer le texte affiché. Des images peuvent également être insérées. Le texte peut être multiligne.

Toplevel : Une fenêtre affichée séparément, « par-dessus ».

- Chaque type de widget est une classe

Les widgets sont les conteneurs visuels utilisés pour organiser ces contrôles, comme la fenêtre de l'application et les cadres qu'elle contient. Les widgets peuvent également être utilisés pour afficher des informations à l'utilisateur, allant d'une simple étiquette de texte à un graphique complexe sur un canevas.



- **Geometry Manager**

Le simple fait de créer un widget Tkinter ne le rend pas visible. Pour qu'un widget soit affiché à l'écran, Tkinter doit savoir exactement où dessiner ce widget. C'est là que la gestion de la géométrie entre en jeu.

Le gestionnaire de géométrie de Tk prend les instructions du programme et fait de son mieux pour placer les widgets à l'endroit prévu. Pour ce faire, il utilise le concept de widgets maître et esclave.

- **Geometry Manager**

Lorsque vous créez un nouvel objet widget et que vous spécifiez son parent, vous identifiez ce widget parent comme le maître du widget enfant. Ce widget maître utilisera un gestionnaire de géométrie pour contrôler le placement de son widget esclave.

Les widgets maîtres sont généralement des conteneurs organisationnels tels que des fenêtres ou des cadres de niveau supérieur. Déterminer l'endroit exact où placer le widget esclave n'est pas toujours un problème simple.

- Geometry Manager

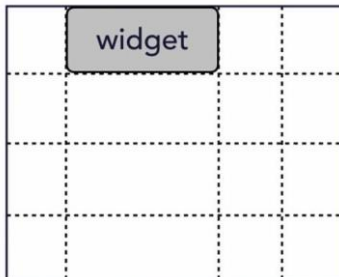
Pack

`Widget.pack(side = RIGHT)`
master



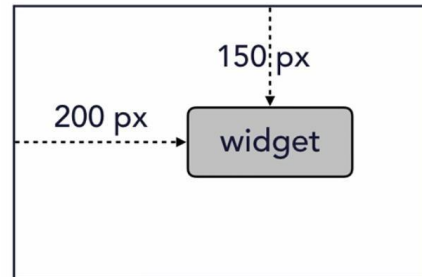
Grid

`Widget.grid(row = 0, column = 1)`
master



Place

`Widget.place(x = 200, y = 150)`
master



Il est possible d'utiliser plusieurs gestionnaires de géométrie au sein d'une même application mais vous devez toujours utiliser le même gestionnaire de géométrie pour organiser les widgets dans le même master.

- **Liaison d'événements**

La liaison d'événement est l'affectation d'une fonction à un événement d'un widget. Lorsque l'événement se produit, la fonction affectée est appelée automatiquement. Cela se fait via la méthode `bind()`. Le nom des événements gérés sont toujours des string commençant par `<` et finissant par `>`. La fonction liée au widget devra obligatoirement avoir un paramètre qui correspondra à l'événement déclenché.

- Liaison d'événements

Liste de quelques événements gérables :

- <Button-1> : clic avec le bouton de gauche.
- <Button-2> : clic avec le bouton du milieu.
- <Button-3> : clic avec le bouton de droite.
- <Double-Button-1> : double clic avec le bouton gauche
- <Button> : un bouton de la souris est enfoncée.
- <ButtonRelease> : un bouton de souris est relâché.
- <Alt> : la touche Alt est enfoncée.
- <Control> : la touche Ctrl est enfoncée.
- <Shift> : la touche Shift est enfoncée.
- <KeyPress> : n'importe quelle touche est enfoncée.
- <KeyRelease> : une touche est relâchée.
- <F1> : la touche F1 est enfoncée.
- <Control-C> : la combinaison de touche Ctrl+C est enfoncée.

17. Django





Django

C'est un framework web python permettant de créer un site web ou une API web.

Django est ordonné de façon MVT (Model View Template) plutôt que MVC (Model View Controller).

Avec Django le view correspond au controller du MVC et les templates correspondent au view du MVC.

Aide sur django : <https://docs.djangoproject.com/>

Etapes pour créer une application Django

Voici les étapes permettant de créer une application Django :

- Création du projet
- Création de l'application principale
- Création du modèle
- Migration du modèle
- Activation de l'admin
- Views
- Urls
- Templates
- Pages dynamiques

- Création d'un projet Django

Créer un environnement virtuel dédié à partir d'une console python dans un dossier :

venv envdjangotest # ou *python -m venv envdjangotest* dans la console
Windows

Activation de l'environnement virtuel :

cd envdjangotest\Scripts
Activate

Remarque : le nom de l'environnement virtuel doit s'afficher avec des () en début de ligne.

- Création d'un projet Django

Installer Django dans l'environnement virtuel et créer un fichier requirements.txt :

```
cd ..
```

```
pip install django
```

```
pip freeze > requirements.txt
```

Création d'un projet wisdompets :

```
django-admin startproject wisdompets
```

```
cd wisdompets
```

Démarrer le serveur local pour vérifier le fonctionnement en allant sur localhost:8000

```
python manage.py runserver
```

Pour arrêter le serveur local, faire *Ctrl+C*

- Création d'un projet Django

Créer une nouvelle application "adoption" du projet wisdompets :
python manage.py startapp adoption

Editer le fichier settings.py situé dans le sous dossier de l'application principale nommée aussi wisdompets.

Ajouter adoption dans la partie INSTALLED_APP pour donner accès à cette application.

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'adoption'  
]
```

- Modèle

Un modèle est une classe qui hérite de `django.db.models.Model`, et qui définit les champs comme des attributs de classe.

Il y a différent type de champ que Django utilise :

Champ	Exemples
CharField	"Parent", "Elève"
TextField	"Ceci est un texte très très long...."
EmailField	mon.email@exemple.fr
URLField	www.example.com
IntegerField	-1, 245, 0
DecimalField	0.5, 3.14
BooleanField	True, False
DateTimeField	Datetime(1960, 1, 1, 8, 0, 0)
ForeignKey	1 (lien vers un identifiant d'une autre table)
ManyToManyField	models.ManyToManyField(classe)

- **Modèle**

Mettre à jour le modèle dans le fichier `models.py` dans le dossier `adoption` :

```
from django.db import models
```

```
# Create your models here.
```

```
class Submitter(models.Model):
```

```
    firstname = models.CharField(max_length = 100)
```

```
    lastname = models.CharField(max_length = 100)
```

```
    is_active = models.BooleanField(default = True)
```

```
    def __str__(self):
```

```
        return f"{self.firstname} {self.lastname}"
```

- Modèle

```
class Pet(models.Model):
    SEX_CHOICES = [('M', 'Male'), ('F', 'Female')]
    name = models.CharField(max_length = 100)
    species = models.CharField(max_length = 30)
    breed = models.CharField(max_length = 30, blank = True)
    description = models.TextField()
    sex = models.CharField(max_length = 1, choices = SEX_CHOICES, blank = True)
    submission_date = models.DateTimeField()
    age = models.IntegerField(null = True)
    submitter = models.ForeignKey(Submitter, on_delete = models.SET_NULL, blank
= True, null = True)
    vaccinations = models.ManyToManyField('Vaccine', blank = True)

    def __str__(self):
        return f"{self.name} {self.species}"
```

- Modèle

```
class Vaccine(models.Model):  
    name = models.CharField(max_length = 50)  
    description = models.CharField(max_length = 500, blank = True)  
  
    def __str__(self):  
        return f"{self.name}"
```

- Migrations

```
python manage.py showmigrations # indique la liste des migrations en attente  
                                # d'exécution (création/modification des tables  
                                # pour les applications)
```

```
python manage.py migrate        # exécute les migrations disponible.
```

```
python manage.py makemigrations # regarde les éléments dans le fichier  
                                # models.py et créer un fichier de migration si  
                                # nécessaire qu'il faudra exécuter
```

Chaque fois qu'on modifie le modèle on doit faire un makemigrations et un migrate pour valider les scripts crée.

Nous pouvons vérifier dans la BDD que cela est effectué.

- Admin

Copier le code suivant dans le fichier `admin.py` situé dans le dossier `adoption` :

```
from django.contrib import admin
from . import models

# Register your models here.
admin.site.register(models.Pet)
admin.site.register(models.Submitter)
admin.site.register(models.Vaccine)
```



Django

- Admin

Puis dans l'invite de commande

python manage.py runserver

Ensuite vérifier en ouvrant la page web localhost:8000/admin qu'une page web s'ouvre, demandant un utilisateur et un mot de passe.

- Création d'un utilisateur admin (super user)

Dans l'invite de commande exécuter :

python manage.py createsuperuser

Permet de créer l'admin et son mot de passe.

Saisir le nom du user, l'email, le mot de passe et le confirmer.

Ensuite tester en ouvrant la page web localhost:8000/admin et se logger.

Maintenant nous pouvons enregistrer, modifier, supprimer des valeurs dans la bdd via la page admin.

Entrer des infos pour tester.

- Shell

Cela permet d'exécuter des requêtes dans une console Django.

- Entrer dans le shell Django
python manage.py shell
- Afficher les animaux

```
from adoption.models import Pet  
results = Pet.objects.all()  
for res in results:  
    print(res)
```

Puis appuyer sur entrée pour valider et afficher le résultat du code.

- Shell
 - Afficher le premier animal

```
pet = Pet.objects.get(id=1)  
print(pet)
```
 - Quitter le shell

```
exit()
```

- View

View sert à définir quelles sont les données qui sont renvoyés.
Editer views.py dans le dossier adoption

```
from django.shortcuts import render  
from django.http import HttpResponse
```

```
# Create your views here.
```

```
def home(request):  
    return HttpResponse('<h1>Menu Accueil</h1>')
```

- Urls

Editer urls.py dans le dossier principal (au même niveau que settings.py).
Ajouter les lignes suivantes :

```
from adoption import views
```

Et

```
path("", views.home, name = "accueil")
```

- **Templates**

Créer le dossier templates dans le dossier adoption, puis un fichier home.html dans ce dossier.

Ecrire dans ce fichier la ligne suivante :

```
<h1>Menu Accueil</h1>
```

Aller dans le fichier views.py et modifier le par :

```
from django.shortcuts import render  
# Create your views here.
```

```
def home(request):  
    context = {}  
    return render(request, 'home.html', context)
```


- Pages dynamiques

- Modifier le fichier views.py par :

```
from django.shortcuts import render
from .models import Pet
# Create your views here.
def home(request):
    pets = Pet.objects.all()
    context = {
        "pets": pets,
        "titre": "accueil"
    }
    return render(request, 'home.html', context)
```

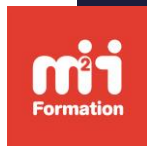
- Pages dynamiques
 - Aller le fichier home.html et modifier le par :

```
<!-- code permettant les pages dynamiques s'appel jinja  
et il est utilisé dans d'autre framework. -->
```

```
<h1>{{ titre }}</h1>
```

```
{% for pet in pets %}  
<h1>{{pet.name}}</h1>  
<h1>{{pet.breed}}</h1>  
<h1>{{pet.description}}</h1>  
{% endfor %}
```

Pour un affichage complet, vous pouvez copier le fichier home.html fournie avec les exercices dans le dossier Django.



FIN



m2information.fr