



**RÉPUBLIQUE
FRANÇAISE**

*Liberté
Égalité
Fraternité*



8487: PYTHON INITIATION



**RÉPUBLIQUE
FRANÇAISE**

*Liberté
Égalité
Fraternité*



Objectifs de la formation

- Découvrir le langage Python
- Prise en main du langage Python

Objectifs pédagogiques :



- ☐ Connaitre les types de variable de base
- ☐ Savoir structurer son programme (fonction , boucle)
- ☐ Utiliser un module
- ☐ Appréhender le modèle objet en Python
- ☐ Lire et écrire un fichier

Le programme



Jour 1

- Présentation
- Jupyter
- Les principes de base
- Manipulations sur des modules



Jour 2

- Les instructions structurantes
- Le modèle objet
- Opération sur les fichiers
- Aller plus loin

Présentation du formateur, des stagiaires et du stage

Presentation de l'outillage

Séquence 1

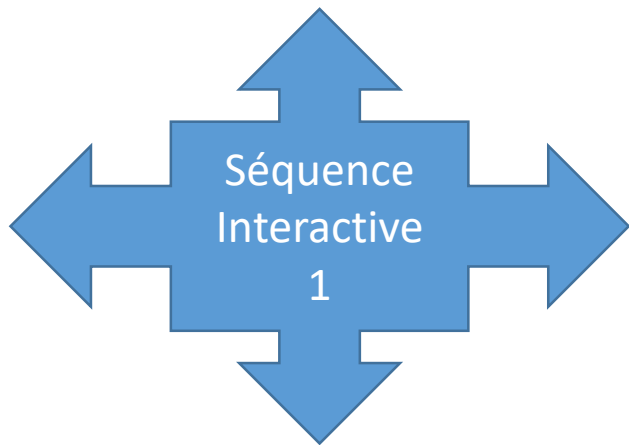
Sommaire

- ❑ Python et PIP
- ❑ Anaconda:Jupyter et Conda
- ❑ Les notebooks
- ❑ Console
- ❑ Spyder

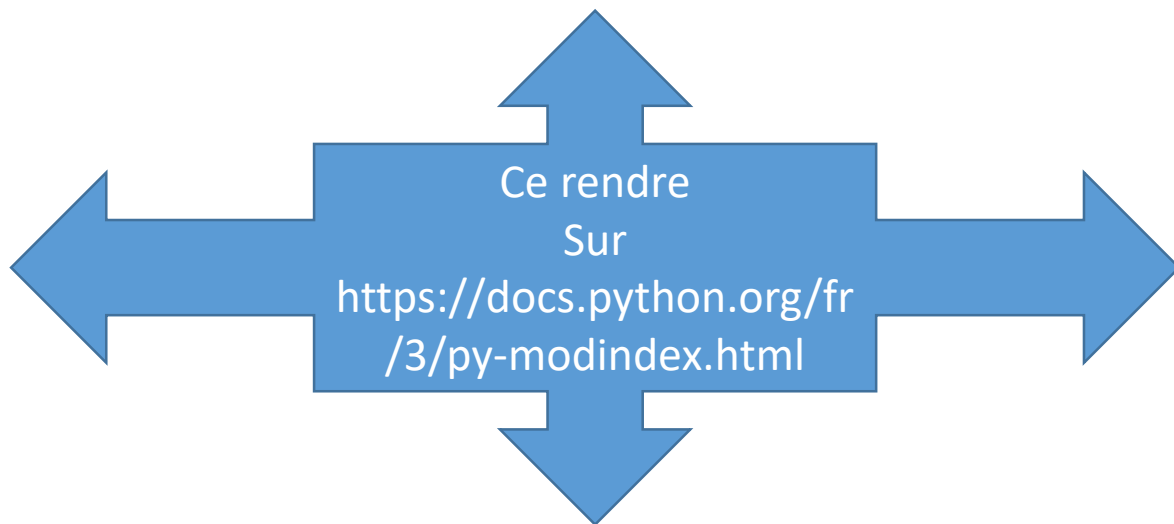
Python : qu'est ce que c'est ?

- C'est un langage de programmation orienté objet et interprété
 - Le typage des variables est dynamique
 - Il est portable sur différents systèmes (Linux, Unix, Windows, MacOSX)
 - Il est extensible par l'utilisation principalement de bibliothèques python mais aussi en C ou Java.
-
- Il se présente sous la forme d'une exécutable: `python` et d'un gestionnaire de paquet: `pip`

Pourquoi utiliser Python ?



Les modules Python



Quelle distribution Python ?

- Distribution de base : elle amène les commandes python et pip (installeur de dépendance). Il faut compléter l'installation par le choix d'un éditeur de code (de notepad++ à eclipse).
- Distribution Jupyter : il apporte un environnement python interactif : édition, exécution, restitution, le tout dans un navigateur
- Distribution anaconda : cette distribution embarque la plupart des librairies et outillages qui sont nécessaires à la mise en œuvre de programme d'IA
- *Bon à savoir: la distribution anaconda fournit aussi l'environnement Jupyter qui est parfaitement adapté au mode de travail d'un data-scientiste.*
- *Bon à savoir: Il existe des images Python utilisables pour créer des containers sous Docker.*

Exemple d'installation:

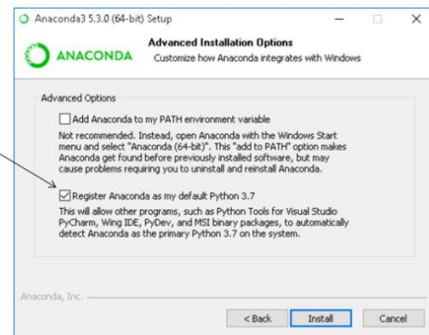
- Installation d'Anaconda.

- A charger sur :

<https://www.anaconda.com/distribution/#window>

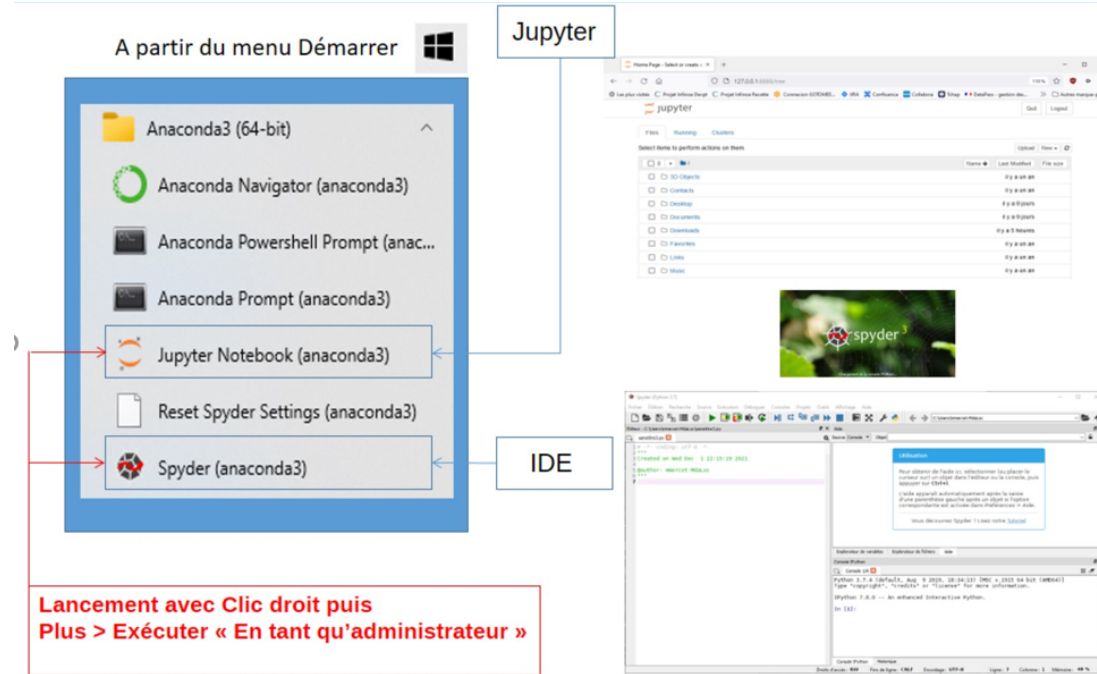


Cochez la case



(Sauter l'installation de Vs code Visual studio si proposée)

Présentation de Jupyter



Lancement de Jupyter notebook

Jupyter se lance dans un terminal (partie Serveur)

```
Jupyter Notebook (anaconda3)

[I 16:15:00.946 NotebookApp] JupyterLab extension loaded from C:\Users\mmmercet01\AppData\Local\Continuum\anaconda3\li
b\site-packages\jupyterlab
[I 16:15:00.946 NotebookApp] JupyterLab application directory is C:\Users\mmmercet01\AppData\Local\Continuum\anaconda3
\share\jupyter\lab
[I 16:15:00.967 NotebookApp] Serving notebooks from local directory: C:\Users\mmmercet01
[I 16:15:00.967 NotebookApp] The Jupyter Notebook is running at:
[I 16:15:00.977 NotebookApp] http://localhost:8888/?token=f6b982c571e9a1482b9b8e88a2efbc17caeeaa88e8ac5cde
[I 16:15:00.977 NotebookApp] or http://127.0.0.1:8888/?token=f6b982c571e9a1482b9b8e88a2efbc17caeeaa88e8ac5cde
[I 16:15:00.978 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation)

[C 16:15:01.180 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/mmmercet-MdaLoc/AppData/Roaming/jupyter/runtime/nbserver-5420-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=f6b982c571e9a1482b9b8e88a2efbc17caeeaa88e8ac5cde
or http://127.0.0.1:8888/?token=f6b982c571e9a1482b9b8e88a2efbc17caeeaa88e8ac5cde
```

Avec création de l'url d'accès à la partie cliente

Jupyter

Ouverture automatique dans un navigateur (*préférez Firefox ou Chrome à Internet Explorer*)

Home Page - Select or create +

127.0.0.1:8888/tree

110 %

Les plus visités C Projet Infinoe Devpt C Projet Infinoe Recette * Connexion GOTOMEE... JIRA X Confluence Collabora Tchap DataPass - gestion des... Autres marque-pages

jupyter Quit Logout

Files Running Clusters

Select items to perform actions on them.

Upload New ↻

<input type="checkbox"/>	Name	Last Modified	File size
<input type="checkbox"/>	/		
<input type="checkbox"/>	3D Objects	il y a un an	
<input type="checkbox"/>	Contacts	il y a un an	
<input type="checkbox"/>	Desktop	il y a 9 jours	
<input type="checkbox"/>	Documents	il y a 9 jours	
<input type="checkbox"/>	Downloads	il y a 5 heures	
<input type="checkbox"/>	Favorites	il y a un an	
<input type="checkbox"/>	Links	il y a un an	
<input type="checkbox"/>	Music	il y a un an	

Jupyter: 1^{er} notebook

The image shows the Jupyter Notebook interface. At the top, there are 'Upload' and 'New' buttons. A dropdown menu is open under 'New', showing options: 'Notebook:', 'Python 3', 'Créer un nouveau notebook avec Python 3', 'Text File', 'Folder', and 'Terminal'. A blue box with an arrow points to the 'Python 3' option, containing the text: 'Créer un nouveau notebook avec le moteur Python 3'. Below the menu, a blue box with an arrow points to a code cell, containing the text: 'Cellule de saisie pour :

- du code Python
- du texte brut au format Markdown
- du texte au format HTML

'. The code cell itself contains the text 'Entrée []: 1'.

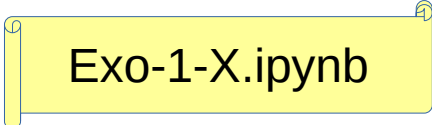
Créer un nouveau notebook avec le moteur Python 3

Cellule de saisie pour :

- du code Python
- du texte brut au format Markdown
- du texte au format HTML

Jupyter: exercice

.Ouvrir le notebook 'exo-1-X' du répertoire 'enonces'



Exo-1-X.ipynb

Les principes de base

Séquence 2

Sommaire

- ❑ L'indentation
- ❑ Les fonctions
- ❑ Les variables
- ❑ Les types de base

Principes

.Les principes directeurs du langage sont au nombre de 3

Principe 1 : Python est fortement objet

Principe 2 : Il n'y a pas de délimiteur de bloc en Python.

Toutes la structuration du code se fait par l'indentation. (marge de gauche)

```
int main() {  
  
    printf ("Hello, World!\n");  
  
    return 0;  
  
}
```

Exemple en C
=

```
int main() {  
    printf ("Hello, World!\n");  
    return 0;  
}
```

Principe 3 : Les instructions structurant le code se terminent par ':'

Python initiation

Anatomie d'une fonction

The diagram shows a code snippet with several annotations:

- Les paramètres (arguments) à passer à la fonction**: Points to the parameters `a, b` in the function definition `def compare(a, b):`.
- Le ':' qui marque le début De l'indentation**: Points to the colon at the end of the function definition line.
- 4 espaces Ou tabulation**: Points to the indentation of the first line inside the function (`if a > b:`).
- 4 espaces Ou tabulation En plus**: Points to the indentation of the lines inside the `if` block (`print("a")` and `print("b")`).
- Appel des fonctions**: Points to the function call `compare(2, 3)` in the main code block.
- Cette fonction Retourne une valeur**: Points to the `return a*a` statement inside the `carre` function.

```
def compare(a,b):  
    if a > b:  
        print("a")  
    else:  
        print("b")  
  
def carre(a):  
    return a*a  
  
compare(2,3)  
b = carre(6)  
print(b)  
  
b  
36
```

Une variable déclarée en dehors d'une fonction est globale sur tout le programme
Une variable déclarée dans une fonction est locale à la fonction

Exercice

.Ouvrir le notebook 'exo-2-X' du répertoire 'enonces'

Exo-2-X.ipynb

Ajouter une variable

- `nom_de_variable = <valeur>`
- `a = 2`
- `b = 'mistral'`
- `c, d = 3, 'autre'`
- Permutation directe :
- `d, c = c, d`

```
c, d = 3, 'autre'  
print(c, d)  
d, c = c, d  
print(c, d)
```

```
3 autre  
autre 3
```

*Bon à savoir : simple ou double quote ?
Pas de difference.*

Convention : `'_'` ou `'__'` encadrant
un élément le désigne comme
un composant interne

**Une variable ne peut être manipulée
que si elle a été affectée
(partie gauche du '=')**

Typage dynamique !

Type de base: entier

*Bon à savoir :
La fonction type(xx) retourne
le type de xx*

```
x = 3    #type entier  
y = 3.0  #type reel  
print("x=", x, type(x))  
print("y=", y, type(y))
```

1

Affectation

```
x= 3 <class 'int'>  
y= 3.0 <class 'float'>
```

2

Manipulations

```
print(x/2)  
print(y/2)  
print(x//2)  
print(y//2)
```

```
1.5  
1.5  
1  
1.0
```

```
a = 2  
print(a, type(a))  
  
a = a/2  
print(a, type(a))
```

```
2 <class 'int'>  
1.0 <class 'float'>
```

adaptations

3

Type de base: chaîne de caractère (str / string)

```
ex1 = 'un exemple'  
ex2 = "un autre exemple"  
ex3 = '''encore un  
plus compliqué'''  
print(ex1, ex2)  
print(ex3)  
print(ex1[3])  
print(ex1[3:5])  
print(len(ex1))
```

```
un exemple un autre exemple  
encore un  
plus compliqué  
e  
ex  
10
```

*Bon à savoir : len(XX) retourne
la longueur d'une chaîne*

```
a = "objet1"  
print(type(a))  
print(id(a)) # imprime l'adresse de l'objet pointé par la variable a.  
b = a  
print(id(b))
```

```
<class 'str'>  
139744616664232  
139744616664232
```

Les chaînes sont aussi des objets

Les tranches (slices) : [n:m:s]

- Ils permettent de définir des segments à extraire sur des chaînes ... mais pas seulement.
- Le 1^{er} caractère a le rang 0. (le dernier -1 en partant de la fin)

n → début de l'intervalle

m → Borne supérieure NON INCLUE

s → le pas d'incrément (1 par défaut)

L'absence de n ou m remplace les valeurs des extrémités

[:] → tous les caractères

```
chaine = "abcdefghij"
print(chaine[0:2])      #2 caractère
print(chaine[0:5:2])    #5 caractères saut de 2
print(chaine[2:3])
print(chaine[-2:-1])
print(chaine[:2])
```

```
ab
ace
c
i
ab
```

Le formatage des chaines

```
a = 'igpde'  
b = 'à '  
c = f"bienvenue {b}l'{a}"  
print(c)
```

bienvenue à l'igpde



*Bon à savoir : en Python 2
"chaîne".format(...)*

```
c = "bienvenue {}l'{}".format(b,a)  
print(c)
```

Autre forme

```
a = 'igpde'  
b = 'a '  
c = f"bienvenu %s l'%s" % (b,a)  
print(c)
```

bienvenu a l'igpde

%s : string
%d : integer
%f : float

Exercice

Exo-3-X.ipynb

•Ouvrir le notebook ‘exo-3-X’ du répertoire ‘enonces’

Utilisation de la
Méthode lower() de la classe
String (str) pour mettre
en minuscule

Type de base: Les tuples

- C'est une Collection (famille importante de structure : tuples, listes, dictionnaires, set etc.)
- Liste NON MODIFIABLE (immutable) de données hétérogènes.

Rôle de la virgule

Tup = (1,)

Tup = ('e',)

<https://pythontutor.com/>

```
# tuples

tup1 = (1, 'eg', 1.3) #c'est un tuple
print(type(tup1))
tup2 = 1,2, 'e'      #c'est aussi un tuple
print(type(tup2))
tup3 = ()           #c 'est un tuple vide : ne sert pas à grand chose
print(type(tup3))
### ce ne sont pas des tuples
tup4 = (1)          # c'est un integer
print(tup4, type(tup4))
tup5 = ('e')        # c'est une chaine
print(tup5, type(tup5))
```

```
<class 'tuple'>
<class 'tuple'>
<class 'tuple'>
1 <class 'int'>
e <class 'str'>
```

Les Tuples

Les constructeurs:

```
] : macollection = ('un' , 'deux' , 3)
```

```
] : macollection = 'un' , 'deux' , 3
```

```
] : macollection = 1,
```

• Et non mutable ; **comme les nombres et les chaînes de caractère**

<https://pythontutor.com/>

```
macollection[1] = 'mod'
```

TypeError

Traceback (most recent call last)

<ipython-input-29-29b67c6f83dd> in <module>

----> 1 macollection[1] = 'mod'

TypeError: 'tuple' object does not support item assignment

Les listes

• Les listes : affectation

```
#ceci est une liste  
malist = [1,3, 'divers'] #utilisation des crochets au lieu des parenthèses  
print(malist, type(malist))
```

```
[1, 3, 'divers'] <class 'list'>
```

• Accéder à un élément (comme pour un tuple)

```
print(malist[2])
```

```
divers
```

Attention à
l'initialisation !!



FAIL

```
a = []  
a[0] = 'premier'
```

```
a = ['premier']  
a[0] = 'zero'
```

```
-----  
IndexError                                Traceback  
<ipython-input-47-10e8d5b6cf1e> in <module>()  
      1 a = []  
>>> 2 a[0] = 'premier'  
IndexError: list assignment index out of range
```

Manipulations des listes

manip_liste.ipynb

Manipulations

manip_dict.ipynb

Type de base: le set

• Ensemble d'éléments unique

– C'est une liste (list) non ordonnée et sans répétition possible d'un élément

– Les crochets des listes sont remplacés par des accolades.

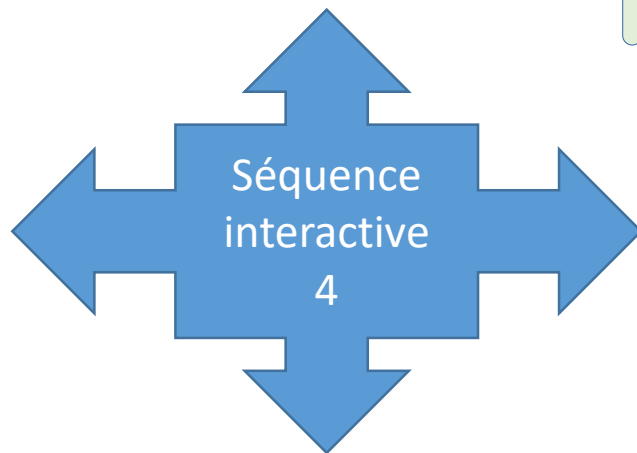
```
mon_set = {1, 2, 'eg', 'echo'}  
print(mon_set, type(mon_set))  
  
{1, 2, 'echo', 'eg'} <class 'set'>
```

Usages du set

manip_set.ipynb

Pourquoi changer le type d'une variable ?

Démonstration
Sys.argv



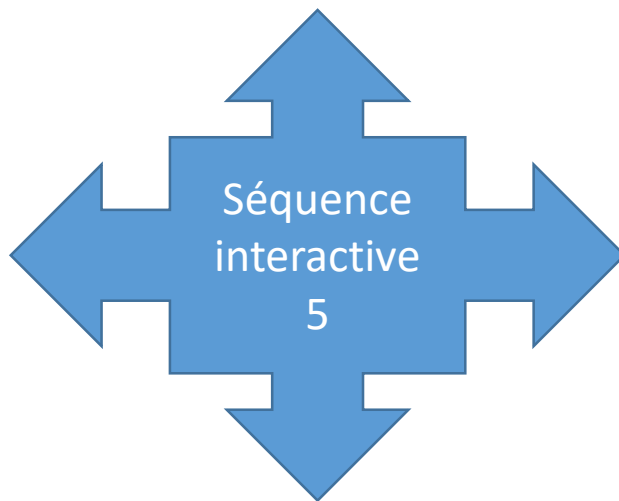
Les opérations de changement de type (caster une..)

`manip_caster.ipynb`

Exercice

Exo-4-X.ipynb

Choix de la structure



Les modules

Séquence 3

Sommaire

- ❑ Utilisation
- ❑ Le module datetime
- ❑ Le module decimal

Utiliser les modules

- Python embarque des modules très puissants qui simplifient les traitements

- L'utilisation de ces modules est activée par :

- `import <module> #formule generale`
- `from <module> import *` #importe tout
- `from <module> import <qqchose>`
- `from <module> import <qqchose> as mon_nom`

```
import datetime
datetime.date.today()

datetime.date(2019, 4, 20)
```

```
: from datetime import date
date.today()

from datetime import date as dt
dt.today()

: datetime.date(2019, 4, 20)
```

Liste des modules

- Dans Python :
 - help('modules')
- Dans le terminal
 - Pydoc modules

```
# python
Python 3.7.2 (default, Dec 29 2018, 06:19:36)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> help('modules')

Please wait a moment while I gather a list of all available modules...
```

Nom du module ou librairie

Module datetime

Structure (classe) à importer

```
from datetime import date
ma_date = date.today()
print(type(ma_date))
print(ma_date.year)
print(dir(ma_date))

<class 'datetime.date'>
2019
['_add_', '_class_', '_delattr_', '_dir_', '_doc_', '_eq_', '_format_', '_ge_', '_getattribute_',
'_gt_', '_hash_', '_init_', '_init_subclass_', '_le_', '_lt_', '_ne_', '_new_', '_radd_', '_reduce_',
'_reduce_ex_', '_repr_', '_rsub_', '_setattr_', '_sizeof_', '_str_', '_sub_', '_subclasshook_',
'ctime', 'day', 'fromisoformat', 'fromordinal', 'fromtimestamp', 'isocalendar', 'isoformat', 'isoweekday', 'max', 'min', 'month', 'replace', 'resolution', 'strftime', 'timetuple', 'today', 'toordinal', 'weekday', 'year']
```

Formatage

```
print(ma_date.strftime("%d/%m/%Y"))
```

09/04/2019

Manipulation de date

manip_date.ipynb

La liste suivante est la liste de tous les codes de formatage requis par le standard C (version 1989), ils fonctionnent sur toutes les plateformes possédant une implémentation de C standard. Notez que la version 1999 du standard C a ajouté des codes de formatage additionnels.

Directive	Signification	Exemple	Notes
%a	Jour de la semaine abrégé dans la langue locale.	Sun, Mon, ..., Sat (<i>en_US</i>); Lu, Ma, ..., Di (<i>fr_FR</i>)	(1)
%A	Jour de la semaine complet dans la langue locale.	<i>Sunday, Monday, ..., Saturday</i> (<i>en_US</i>); Lundi, Mardi, ..., Dimanche (<i>fr_FR</i>)	(1)
%w	Jour de la semaine en chiffre, avec 0 pour le dimanche et 6 pour le samedi.	0, 1, ..., 6	
%d	Jour du mois sur deux chiffres.	01, 02, ..., 31	
%b	Nom du mois abrégé dans la langue locale.	Jan, Feb, ..., Dec (<i>en_US</i>); janv., févr., ..., déc. (<i>fr_FR</i>)	(1)
%B	Nom complet du mois dans la langue locale.	<i>January, February, ..., December</i> (<i>en_US</i>); janvier, février, ..., décembre (<i>fr_FR</i>)	(1)
%m	Numéro du mois sur deux chiffres.	01, 02, ..., 12	
%y	Année sur deux chiffres (sans le siècle).	00, 01, ..., 99	
%Y	Année complète sur quatre chiffres.	0001, 0002, ..., 2013, 2014, ..., 9998, 9999	(2)
%H	Heure à deux chiffres de 00 à 23.	00, 01, ..., 23	

Bon à savoir : aide supplémentaire
 Pages de man (linux) : aide en ligne
man strftime

Exercice

Exo-5-X.ipynb

Module Decimal: manipuler des montants

```
from decimal import Decimal
a = Decimal('0.10')
b = Decimal('0.30')
print(a+a-a-b)

c = 0.10
d = 0.30
print(c+c+c-d)
print(round(c+c+c-d, 2))
```

0.00

5.551115123125783e-17

0.0

Synthèse

- Int : Nombre entier 23
- Float : Nombre à virgule flottante 3.1415
- Str : Chaîne de caractère 'Hello'
- Tuple : Liste non modifiable (1,'a',2.5)
- List : Liste de longueur variable
['a','z','e','r']
- Dict : dictionnaire
{'un':1,'deux':2}
- file: Fichier
- Bool : Booléen (1 == 1.0)
- NoneType : Absence de type None
- Function : fonction
- method : méthode
- Module : module
- Byte : octet (immuable)
- Set : ensemble

Quizz

https://www.wooclap.com/nom_du_quizz

Les instructions structurantes

Séquence 4

Sommaire

- ☐ If ... else
- ☐ For ... in
- ☐ While

Les structures de contrôle

.If condition... :

- If
- If else
- If elif

.While condition ... :

.For... :

- in sequence
- in range

La famille des 'IF'

- .Attention à l'indentation.
- .Fonctionne comme dans les autres langages

.If simple

```
def plusgrandquedeux(n):  
    if n > 2:  
        print('plus grand')  
  
plusgrandquedeux(1) # n'affiche rien  
plusgrandquedeux(5) # vrai -> affiche  
  
plus grand
```

Le case ou switch existe depuis peu en Python (3.10)

Les conditions

- < ou >
- <= ou >=
- == ou !=
- **in : appartenance x in ENSEMBLE/LISTE**
- X is Y : X et Y représentent le même objet.
- X is not Y : X et Y ne représentent pas le même objet

- not devant une condition, pour l'inverser

Spécial Python

•if 2<n<4 : #condition valide

•L'opérateur 'in' .. list, tuple, string

•Pas de confusion possible entre = et ==

```
-----  
if a= 1:  
    ^  
SyntaxError: invalid syntax
```

```
print(2 > 3)  
print(not (2 > 3))  
print(2<3<4)
```

```
False  
True  
True
```

En javascript =>

```
> if (a = 1) { console.log('v');}  
v
```


Exercice

Exo-6-X.ipynb

Iteration avec For

- La forme FOR..IN est la clé de voûte du langage.
- Elle permet de traiter chaque élément d'une structures « iterable » (cad : qui sont capables de délivrer un élément après l'autre)

– Identique à un foreach en Perl

- La boucle 'for' sur compteur n'a pas la même forme que dans d'autres langages. (§ Range)
- ex : `for (my $i = 0; $i <= 9; $i++) { ... }` #en Perl

Démonstration de for .. in

- Pour les structures itérables
- Combiné avec l'instruction range

manip_for.ipynb

Exercice

Exo-7-X.ipynb

Pour en finir avec le while

- Très utilisé dans d'autres langages
Et très peu en python...

On le trouve parfois dans la forme :

```
while True :  
    Traitement ....  
    if condition :  
        break
```

```
while True:  
    print('je passe')  
    if 1 == 1 :  
        break  
  
print('je sors')
```

je passe
je sors

exercice

Exo-8-X.ipynb

Documentation des programmes

- Python propose un dispositif de documentation intégré : Pydoc.

```
def mafonction():  
    ''' affiche rien, ne fait rien et ne retourne rien'''  
    pass          # ne fait rien  
    return None   #equivalent du Null en SQL  
print(mafonction())  
help(mafonction)
```

None

Help on function mafonction in module __main__:

```
mafonction()  
    affiche rien, ne fait rien et ne retourne rien
```

Les classes et les objets

Séquence 5

Sommaire

- ☐ Rappels
- ☐ Le constructeur
- ☐ Utilisation des méthodes et des attributs
- ☐ Variable de classe
- ☐ L'héritage

Classes et objets en Python

- Python propose un modèle objet simple situé entre Perl et Java.
- Python est fortement objet mais n'impose rien.
- Le modèle est permissif, c'est au concepteur d'être rigoureux.
- Vocabulaire :

– Classe

– Objet

– Instance / instanciation : objet XXX de la classe YYY

– Variable de classe

– Variable d'instance d'objet

– Méthode de classe

– Méthode d'instance d'objet

Avantage de l'approche objet

- Rendre le code modulaire
- Evite les problèmes de 'portée' (scope) des variables.

**Le scope des données en Python n'a pas été abordé
Et dépasse le cadre de cette initiation**

Avant de commencer: quelques notions

Python facilite le passage entre une
approche par fonction et une
approche par objet

`manip_refactorise.ipynb`

`manip_classe.ipynb`

Le constructeur

- Méthode (fonction) spécifique : `__init__`
- C'est le 'constructeur'

méthode

1^{er} paramètre spécial

```
class Contact:
    '''gestion de mes contacts'''
    def __init__(self, libelle, adresse):
        self.libelle = libelle
        self.adresse = adresse

moncontact = Contact('Dupont', '@edupont')
print(moncontact, type(moncontact))
```

Self est une convention
Il désigne la référence de
l'objet

Instanciation

```
<__main__.Contact object at 0x7fad4c4ce128> <class '__main__.Contact'>
```

En coulisse

- A chaque appel de méthode :
 - Python insère automatiquement comme premier paramètre la référence de l'objet (adresse mémoire)

```
def affiche(self):  
    print(self.libelle, self.adresse)
```

A ne pas faire:

Ne pas utiliser le nom self

```
class Prenom2:
    '''prenom unitaire'''
    def __init__(a, l, gen, fre ):
        '''constructeur'''
        a.libelle = l
        a.genre = gen
        a.freq = fre

m = Prenom2('denis', 'M', 12 )
```

Signature de fonction
obscur

Utilisation

- Une méthode s'appelle sous cette forme :
objet.methode(ARG) ou objet.methode()
- Un attribut s'utilise comme :
objet.attribut (getter)
Ou *objet.attribut = xxxx (setter)*

```
print(moncontact.libelle)
moncontact.libelle = 'Martin'
print(moncontact.libelle)
```

```
Dupont
Martin
```


Exemple

```
class Contact:
    '''gestion de mes contacts'''
    def __init__(self, libelle, adresse):
        print(self)
        self.libelle = libelle
        self.adresse = adresse
    def affiche(self):
        print(self.libelle, self.adresse)
    def pprint(self, sep):
        print(f"{self.libelle}{sep}{self.adresse}")
```

Référence de l'objet

Appel de la méthode : `moncontact.pprint(':')`

Dupont:@edupont

Exercice

Exo-9-X.ipynb

Les variables de classe

•Utilisables sans instantiation préalable

•Accès directement sans instantiation

```
class Gaufre:

    compteur = 0
    def __init__(self, modele, garniture, cuisson= 'normale'):
        self.modele = modele
        self.garniture = garniture
        self.cuisson = cuisson
        Gaufre.compteur += 1

    def cuisson(self):
        pass
```

```
commande1 = Gaufre('belge', 'confiture fraise')
commande2 = Gaufre('française', 'sucre')
commande1 = Gaufre('sans gluten', 'confiture fraise')
commande1 = Gaufre('salée', 'fromage')
```

```
print(Gaufre.compteur)
```

L'héritage

- Simple ou multiple

- class(herite_de) :*

- def ...*

- Autorise la surcharge de méthode ou d'attribut

- Possibilité d'appeler directement le constructeur de la classe parent par :

- `super().__init__(..)`

Exercice

Exo-10-X.ipynb

Quizz

Les fichiers

Séquence 6

Sommaire

- ❑ Opérations sur les fichiers
- ❑ Utilisation d'une gestion de contexte (with)

Les fichiers

Les fichiers servent à stocker des informations de manière pérenne

Pour être traité, un fichier doit être :

- 1) Ouvert (en lecture ou en écriture ; en mode texte ou binaire)
- 2) Lu ou écrit pour récupérer ou stocker l'information
- 3) Fermé

Modalités

instruction :

open(nom[, mode] [,encoding='encoding'])

nom : nom du fichier

mode : 'r' (lecture) / 'w' (écriture) / 'a' (ajout)

mode (suite) : 't' (texte) / 'b' (binaire)

exemples :

open('fichier.txt')

open('donnees', 'wb')

Utilisation d'un contexte with...

manip_fichier.ipynb

```
with open('essai.txt') as fic:  
    t = fic.readlines()  
print(t)
```

```
['hello world\n', 'c est un nouveau fichier \n', 'deuxieme ligne\n', 'et troisieme\n']
```

- .Code dans l'esprit de Python
- .Referme le fichier en sortie de bloc
- .Referme le fichier en cas d'exception

Exercice

Exo-11-X.ipynb

Pour aller plus loin

- Stage IGPDE Python perfectionnement et les 3 stages modules
- Tutoriaux

– <https://docs.python.org/fr/3/tutorial/index.html>

– <https://python.sdv.univ-paris-diderot.fr/cours-python.pdf>

– https://inforef.be/swi/download/apprendre_python3_5.pdf

- De très nombreux ouvrages Python sont disponibles sur Internet