

1

# PYTHON

Michel Blache

Version 1.1 du 6 Novembre 2022

2

Ifpa - Poitiers

## Python

### Votre Animateur

- Michel Blache
- Intervenant pour le Groupe AFIB
- Chef de Projet Infrastructure Airbus, AFI, EFS
- Technologies :
  - ITIL,
  - Linux,
  - Merise, UML,
  - DataBase & SQL
  - AMOA
  - Langage C, Cobol, PL/SQL, Python

3

Ifpa - Poitiers

## Python

### Parlez-moi de vous

- Employeur
- Fonction
- Utilisation de l'outil informatique
- Besoins
- Attentes

4

Ifpa - Poitiers

## Python

### Au programme

- Objectifs de la formation
  - Maîtriser dans le langage python sur :
    - Les variables et affectations
    - Les fonctions
    - Les instructions d'écriture et de lecture
    - La structure conditionnelle
    - Les boucles
    - Les listes
    - Triage des listes

5

Ifpa - Poitiers

## Python

### Organisation de la formation

- Horaires
- Pauses
- Fin de la formation
  - Supports de cours
  - Evaluation de la formation



6

Ifpa - Poitiers

## Python

### Pédagogie

- Théorie
- Démonstration du Formateur
- Mise en Application collective puis autonome
- Questions et Réponses

7

Ifpa - Poitiers

## Python

Avez-vous des questions ?

8

Ifpa - Poitiers

## Python

### Introduction

- Le langage Python date de la fin des années 80 et a été écrit par Guido Van Rossum (Hollande)
- Open source et date pour la première version de 1990
- C'est un langage
  - Interprété (avantage et inconvénient)
  - Orienté objet
  - Multiplateforme (Linux, Windows ..)
  - Polyvalent (N'importe quel projet)
  - Facile à apprendre

9

Ifpa - Poitiers

## Python

### Introduction

- Le nom Python fait référence aux Monty Python (humour britannique)
- Langage interprété c'est-à-dire qu'il n'a pas besoin d'être compilé pour être exécuté. Programme interpréteur.
- Multiplateforme : Windows, Ios, Linux ...
- Outils pour faire du développement
  - Web (Django) Instagram a été développé en Python
  - de Jeux (Pygame),
  - d'interface graphique avec Tk& GUI.
- Automatisation de tâches de production (Sauvegarde ....)

10

Ifpa - Poitiers

## Python

### Introduction

- Machine Learning (Intelligence artificielle ...)
- Big Data : Données massives
- Data Science : Collecter et décrypter des données.
- Langage de haut niveau (ne se préoccupe pas du fonctionnement de la machine) à l'opposé de l'assembleur et du langage machine.
- En 2022, langage au top du marché de l'emploi.

11

Ifpa - Poitiers

## Python

### Installation

- Sur Windows 10
  - Python.org Télécharger la dernière version 3.7.3
  - Ajouter Python 3.7 au Path Windows
  - Sous CMD Taper « Python »
- Sous Linux
  - Debian : \$ sudo apt-get install python3.7
  - Sur Red Hat : \$ sudo yum install python37
- Sur Mac
  - <https://programwithus.com/learn/python/install-python3-mac>

12

Ifpa - Poitiers

## Python

### Conventions pour la présentation du code

- Pour l'écriture du code, vous rencontrerez plusieurs présentations :
  - Session interactive : instructions précédées de chevrons « >>> » (sous cmd : taper « python »)

```

C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 3+5
8
>>> from math import sqrt
>>> print(sqrt(5))
2.23606797749979
>>> print(sqrt(25))
5.0
>>>
  
```

13

Ifpa - Poitiers

## Python

### Conventions pour la présentation du code

- Script : les instructions sans chevrons dans une boîte grisée sont des bouts de code à écrire dans un fichier linux;

```
print('Bonjour !')
```

```
fichier.py
```

```
#!/usr/bin/python3
```

```
#!/usr/bin coding: Utf-8 -x*-
```

```
print('Voici le résultat')
```

14

Ifpa - Poitiers

## Python

### Les opérateurs python : Opérateurs arithmétiques

- $x + y$  # Addition
- $x - y$  # Soustraction
- $x * y$  # Multiplication
- $x / y$  # Division
- $x // y$  # Pour obtenir le quotient de la division
- $x \% y$  # Modulo (reste)
- $-x$  # Opposé – opérateur unaire
- $\text{abs}(x)$  # Valeur absolue – opérateur unaire
- $x ** y$  # Puissance

15

Ifpa - Poitiers

## Python

### Les opérateurs python : Opérateurs arithmétiques

- Certains opérateurs peuvent avoir des comportements différents en fonction des types d'opérandes sur lesquels ils agissent : on parle alors de surcharge des opérateurs.
- Exemple :
  - "+" additionne des nombres, mais concatène des chaînes de caractères.
  - "\*" multiplie des nombres entre eux, mais duplique des chaînes de caractères.
- La différence entre  $x/y$  et  $x//y$  est que le premier donne une valeur approchée décimale à 16 chiffres après la virgule alors que la deuxième nous donne l'entier.

16

Ifpa - Poitiers

## Python

### Les opérateurs python : Opérateurs de comparaison

- Tout comme les opérateurs logiques, les opérateurs de comparaison renvoient une valeur booléenne "True" ou "False"
- $x < y$  # Inférieur
- $x \leq y$  # Inférieur ou égal
- $x > y$  # Supérieur
- $x \geq y$  # Supérieur ou égal
- $x == y$  # Égal (attention !)
- $x != y$  # Différent
- $x <> y$  # Différent
- $x \text{ is } y$  # Identité
- $x \text{ is not } y$  # Non identité
- L'égalité de deux valeurs est comparée avec l'opérateur " $==$ " et non " $=$ ". Ce dernier est en effet l'opérateur d'affectation et ne doit pas être utilisé dans une condition.



17

Ifpa - Poitiers

## Python

### Les opérateurs python : Opérateurs d'affectation

- L'affectation est l'instruction qui associe une valeur à une variable.
- Elle est symbolisée par le signe '='.
- Attention à ne pas confondre l'affectation avec un test d'égalité !
- L'affectation 'x = 98' se lit "x reçoit 98".
- La partie droite est d'abord évaluée, son résultat est ensuite stocké dans la partie gauche.
- En Python, l'affectation est une instruction et non pas une opération, elle n'a donc pas de valeur.
- `x = 98`      # Affectation simple
- `y = 98 > 50`   # 'y' sera True ou False
- `x = y`
- `x == y` # Attention, test d'égalité ! Retourne True dans notre cas.

18

Ifpa - Poitiers

## Python

### Les opérateurs python : Opérateurs d'affectation

Opérateur	Exemple	Equivalent à	Description
=	<code>x = 1</code>	<code>x = 1</code>	Affecte 1 à la variable x
+=	<code>x += 1</code>	<code>x = x + 1</code>	Ajoute 1 à la dernière valeur connue de x et affecte la nouvelle valeur (l'ancienne + 1) à x
-=	<code>x -= 1</code>	<code>x = x - 1</code>	Enlève 1 à la dernière valeur connue de x et affecte la nouvelle valeur à x
*=	<code>x *= 2</code>	<code>x = x * 2</code>	Multiplie par 2 la dernière valeur connue de x et affecte la nouvelle valeur à x
/=	<code>x /= 2</code>	<code>x = x / 2</code>	Divise par 2 la dernière valeur connue de x et affecte la nouvelle valeur à x
%=	<code>x %= 2</code>	<code>x = x % 2</code>	Calcule le reste de la division entière de x par 2 et affecte ce reste à x
//=	<code>x //= 2</code>	<code>x = x // 2</code>	Calcule le résultat entier de la division de x par 2 et affecte ce résultat à x
**=	<code>x **= 4</code>	<code>x = x ** 4</code>	Elève x à la puissance 4 et affecte la nouvelle valeur dans x

19

Ifpa - Poitiers

## Python

### Utiliser Python comme une calculette

- Sous CMD, L'invite de commande se compose de trois chevrons ; il suffit de saisir à la suite une instruction puis d'appuyer sur la touche « Entrée » de votre clavier.

```
>>> print('Bonjour !')
Bonjour !
```

- La console Python fonctionne comme une simple calculatrice : vous pouvez y saisir une expression dont la valeur est renvoyée dès que vous pressez la touche « Entrée ».

20

Ifpa - Poitiers

## Python

### Utiliser Python comme une calculette

```
>>> 2 * 5 + 6 - (100 + 3 )
-87
>>> 7 / 2; 7/3
3-5
2.3333333333333335
>>> 34 // 5; 34 % 5 # quotient & reste de la division euclidienne de 34 par 5
6
4
>>> 2**7 # pour l'exponentiation (et non pas 2*7 !!)
128
```

Au passage, nous avons utilisé le symbole « dièse » # pour placer des commentaires dans les lignes de commande; tout ce qui se situe à droite d'un symbole # (jusqu'au changement de ligne) est purement et simplement ignoré par l'interpréteur.

21

Ifpa - Poitiers

## Python

### Variables et affectations

```

Invite de commandes - python
Microsoft Windows [version 10.0.19044.2130]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 128, id(128), type(128)
(128, 2153894711504, <class 'int'>)
>>>
>>> a=128
>>> a
128
>>> a, id(a), type(a)
(128, 2153894711504, <class 'int'>)
>>> 2*a
256
>>>

```

- Cet objet possède une valeur (ici 128), un identifiant, c'est-à-dire une « carte d'identité » permettant de savoir où il est gardé 137182768 en mémoire (ici 137182768), et enfin un type (ici le type entier dénommé int).

22

Ifpa - Poitiers

## Python

### Variables et affectations

```

Invite de commandes - python
Microsoft Windows [version 10.0.19044.2130]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 128, id(128), type(128)
(128, 2153894711504, <class 'int'>)
>>>
>>> a=128
>>> a
128
>>> a, id(a), type(a)
(128, 2153894711504, <class 'int'>)
>>> 2*a
256
>>>

```

- Au lieu de désigner 128 par son identifiant, on va lui donner un nom commode à manipuler. Au lieu d'appeler l'objet « objet137180768 », on l'appellera « a », après avoir indiqué à l'interpréteur une fois pour toute l'identification opérée par un message du type : « a alias objet 137180768 ». En pratique, une affectation s'effectue à l'aide du symbole « = »

23

lfpa - Poitiers

## Python

### Variables et affectations

- Ainsi chaque fois que nous appellerons le nombre 128, il nous suffira d'invoquer la variable a. Notez que l'objet désigné par a est toujours l'objet 128, rangé encore à la même adresse. Il faut bien prendre garde au fait que l'instruction d'affectation «=» n'a pas la même signification que le symbole d'égalité «=» en mathématiques". Par exemple, le premier n'est pas symétrique, alors que le second l'est : vouloir échanger l'ordre des éléments dans une instruction d'affectation produira inmanquablement une erreur dans l'interpréteur :

```
>>> 128 = a
File "<stdin>", Line 1
SyntaxError: can't assign to literal
```

24

lfpa - Poitiers

## Python

### Variables et affectations

- Effectuons à présent la suite d'affectations suivantes :



```
Invite de commandes - python
C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=128
>>> a, id(a), type(a)
(128, 3137739886800, <class 'int'>)
>>> b=a*2
>>> b, id(b), type(b)
(256, 3137739890896, <class 'int'>)
>>> a=0
>>> a, id(a), type(a)
(0, 3137739882704, <class 'int'>)
>>> b, id(b), type(b)
(256, 3137739890896, <class 'int'>)
>>>
```

- On remarque que l'identifiant de l'objet auquel renvoie la variable b n'a plus rien à voir avec a. Autrement dit, l'objet nommé b n'a plus aucune relation avec l'objet nommé a. Ainsi, une réaffectation ultérieure de la variable a n'entraînera aucun changement pour la variable b.

25

Ifpa - Poitiers

## Python

### Variables et affectations

- Il est fréquent qu'en programmation, on se serve d'une variable comme d'un compteur et que l'on ait donc besoin d'incrémenter (c'est-à-dire augmenter) ou de décrémenter (c'est-à-dire diminuer) la valeur de la variable d'une certaine quantité. On procède alors de la manière suivante:



```

Invite de commandes - python
C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x=0
>>> x=x+1
>>> x, id(x), type(x)
(1, 2666599874800, <class 'int'>)
>>> x=x+1
>>> x, id(x), type(x)
(2, 2666599874832, <class 'int'>)
>>> x=x+1
>>> x, id(x), type(x)
(3, 2666599874864, <class 'int'>)
>>>
  
```

- Notez bien la différence avec le calcul algébrique : alors que l'équation  $x = x + 1$  n'a pas de solution, l'instruction `x = x + 1` est parfaitement licite.

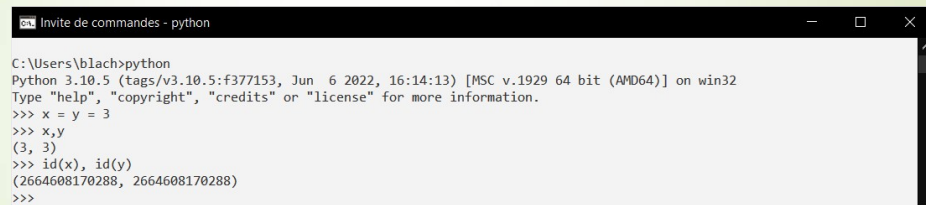
26

Ifpa - Poitiers

## Python

### Variables et affectations

- Autre raccourci intéressant, on peut assigner un même objet à plusieurs variables simultanément. Ces deux variables renvoient alors au même objet (on dit parfois que ce sont deux alias du même objet):



```

Invite de commandes - python
C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = y = 3
>>> x,y
(3, 3)
>>> id(x), id(y)
(2664608170288, 2664608170288)
>>>
  
```

- On peut aussi effectuer des affectations parallèles à l'aide d'un seul opérateur «`=`». Toutes les expressions sont alors évaluées avant la première affectation.  
`>>> X, Y = 128, 256`
- Encore une fois, il faut bien comprendre qu'une affectation se déroule en deux temps : évaluation de l'expression de droite puis création de l'alias avec le nom du terme de gauche.

27

Ifpa - Poitiers

## Python

### Variables et affectations : Types de variable

- Une variable sans valeur est définie par : `myVar = None` (None est l'équivalent de null dans d'autres langages).
- Types primitifs :
  - `bool` : booléen (True ou False)
  - `int` : entier.
  - `float` : nombre flottant qui a la précision d'un double.
  - `str` : chaîne de caractère (string).
- Pour détruire une variable :
 

```
>>> del mavar
```

28

Ifpa - Poitiers

## Python

### Fonctions

- Soit la fonction  $f: x \rightarrow x^7 - 6x^6 + 15x^4 + 23x^3 + x - 9$ , il est rapidement fastidieux de la saisir à chaque fois que l'on souhaite calculer l'image d'un nombre par cette fonction.
- Une première idée est d'utiliser l'historique de la console pour éviter de saisir à chaque fois la fonction :

```

C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x=2
>>> x**7 - 6*x**6 + 15*x**4 + 23*x**3 + x - 9
161
>>> x=3
>>> x**7 - 6*x**6 + 15*x**4 + 23*x**3 + x - 9
-357
>>> x=4
>>> x**7 - 6*x**6 + 15*x**4 + 23*x**3 + x - 9
-2885
>>>
  
```

29

lfpa - Poitiers

## Python

### Fonctions

- Il est tout à fait possible de définir une fonction (au sens du langage Python) qui ressemble à une fonction mathématique. La syntaxe est alors la suivante :

```

Invite de commandes - python
C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> def f(x) :
...     return x**7 - 6*x**6 + 15*x**4 + 23*x**3 + x - 9
...
>>> f(2)
161
>>> f(7)
161551
>>> f(3)
-357
>>> .

```

30

lfpa - Poitiers

## Python

### Fonctions : erreurs standards

```

>>> # ---> Erreur 1 : l'oubli des deux points en fin de ligne
>>> def f(x)
File "<stdin>", line 1
def f(x)
#
SyntaxError: invalid syntax
>>> # ---> Erreur 2 : le non-respect de l'indentation
>>> def f(x):
return xxx7 - GxxxxG + 15xxxx4 + 23xxxx3 + x - 9
File "<stdin>", line 2
return Xxx7 - GxxxxG + 15xxxx4 + 23xxxx3 + X - 9
A
IndentationError: expected an indented block
>>> # ---> Erreur 3 : l'oubli du mot return
>>> def f(x):
X*x*x7 - Gxxxx6 + 15*xxxx4 + 23xxxx3 + x - 9
>>> f(2), f(3), f(4)
(None, None, None)

```



31

Ifpa - Poitiers

## Python

### Fonctions : différence entre return et print

```
>>> def f(x):
    return x*x*x/ - 0*x*x*x0 + 15*x*x*x4 + 23*x*x*x3 + x - 9
>>> f(2) + f(3) + f(4)
187
>>> def f(x):
    print(x*x*x7 - G*x*x*x6 + 15*x*x*x4 + 23*x*x*x3 + X - 9)
>>> f(2) + f(3) + f(4)
19
113
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'NoneType' and 'NoneType'
```

- Il vaut mieux éviter de prendre la mauvaise habitude d'utiliser la fonction print à l'intérieur des fonctions, sauf si c'est explicitement demandé.

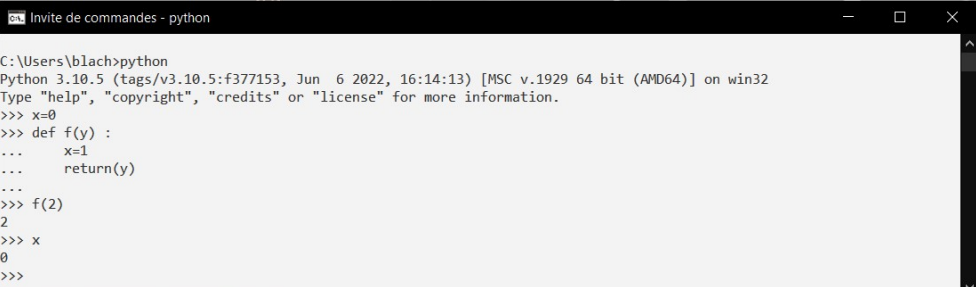
32

Ifpa - Poitiers

## Python

### Fonctions

- Si une variable x existait déjà avant l'exécution de la fonction, tout se passe comme si, durant l'exécution de f, cette variable était masquée momentanément, puis restituée à la fin de l'exécution de la fonction :



```

C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x=0
>>> def f(y) :
...     x=1
...     return(y)
...
>>> f(2)
2
>>> x
0
>>>
```



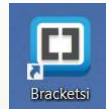
33

Ifpa - Poitiers

## Python

### Travailler avec Bracket et CMD sous Windows

- Afin de pouvoir travailler confortablement pendant la réalisation des scripts et l'exécution des scripts en test, je vous conseille fortement de télécharger (<https://brackets.io/>) et d'installer l'open source « bracket » et d'utiliser simultanément l'invite de commande CMD.

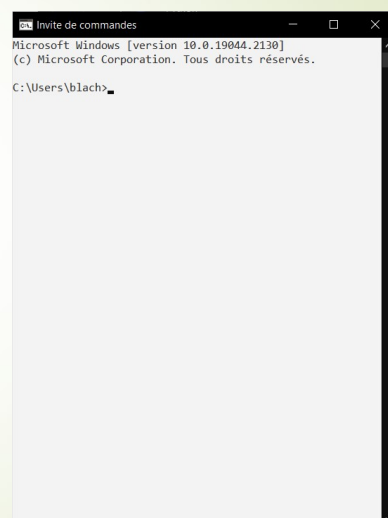
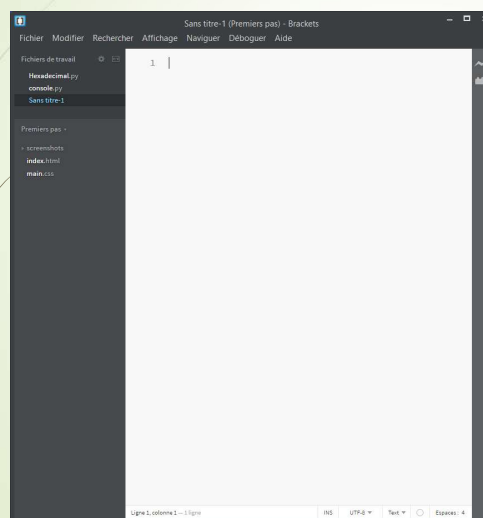


34

Ifpa - Poitiers

## Python

### Travailler avec Bracket et CMD sous Windows



35

Ifpa - Poitiers

## Python

### Travailler avec Bracket et CMD sous Windows

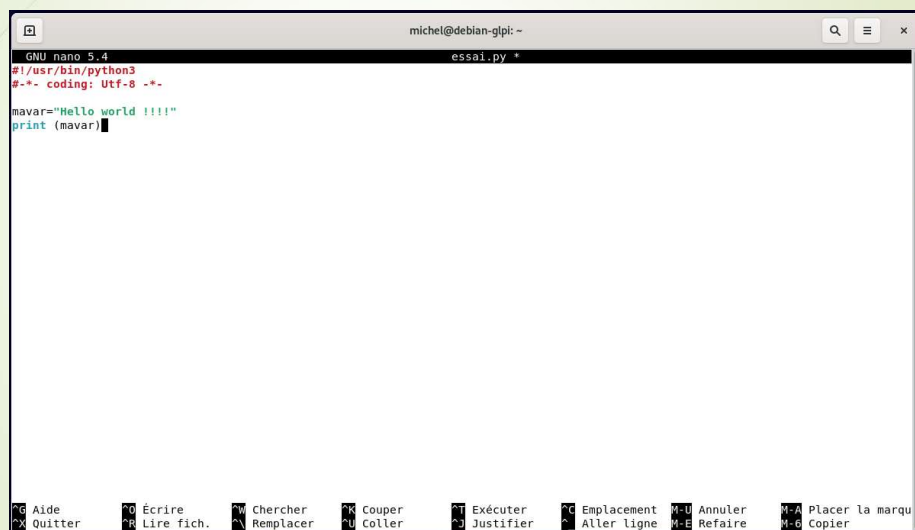
- Sous CMD, créer un répertoire \Python sous C;\
- Déplacez vous sous ce répertoire
- Sous « Bracket » créer un nouveau fichier avec les caractéristique « python » pour le type de fichier et « UTF-8 » pour le codage du fichier.
- Sauvegardez le fichier sous « c:\python »

36

Ifpa - Poitiers

## Python

### Travailler nano sous Linux



```
GNU nano 5.4      essai.py +
#!/usr/bin/python3
#-*- coding: Utf-8 -*-

mavar="Hello world !!!!"
print (mavar)
```

The screenshot shows a terminal window with the nano text editor. The editor is editing a file named 'essai.py'. The content of the file is as follows:

```
#!/usr/bin/python3
#-*- coding: Utf-8 -*-

mavar="Hello world !!!!"
print (mavar)
```

The terminal window title is 'michel@debian-glip: ~'. The nano editor's status bar at the bottom shows various keyboard shortcuts for editing and saving the file.

37

Ifpa - Poitiers

## Python

### Instructions d'écriture et de lecture

- Après écriture du fichier suivant, exécutez-le :

```
# Programme Lambda
x = x ** 8
x
```

- Après exécution il ne se passe rien; En fait, le comportement de l'interpréteur diffère légèrement suivant que l'on travaille dans une console Python ou dans un fichier texte.

```
# Programme Lambda
x = x ** 8
print(x)
```

38

Ifpa - Poitiers

## Python

### Instructions d'écriture et de lecture : print()

#### ➤ Définition :

- La fonction print() affiche l'argument qu'on lui passe entre parenthèses et un retour à ligne.
- La fonction intégrée print() s'écrit toujours en minuscules et est systématiquement suivi de parenthèses qui renferment l'information à afficher.
- print() permet d'afficher des données sur la sortie standard, qui est l'écran
- Cette fonction print() imprime l'objet donné sur le périphérique de sortie standard (écran) ou dans le fichier de flux de texte.
- La fonction input() renvoie toujours une chaîne de caractères

39

Ifpa - Poitiers

## Python

### Instructions d'écriture et de lecture : print()

#### ► Syntaxe :

► `print (valeur (s), sep = '', end = '\ n', file = file, flush = flush)`

#### ► Paramètres :

- `valeur (s)`: N'importe quelle valeur. Sera converti en chaîne avant d'être imprimé
- `sep = 'separator'`: (Facultatif) Comment séparer les objets, s'il y en a plusieurs.
- `end = 'end'`: (Facultatif) Spécifiez quoi imprimer à la fin .Défaut: fichier '\ n'
- `file = file`: (Facultatif) Un objet avec une méthode d'écriture. Par défaut: `sys.stdout`
- `flush`: (Facultatif) Un booléen, spécifiant si la sortie est vidée (True) ou mise en mémoire tampon (False). Par défaut: Faux
- Retours: il renvoie la sortie à l'écran.

40

Ifpa - Poitiers

## Python

### Instructions d'écriture et de lecture : print()

#### ► Paramètres sep :

- Dans python 3.x ou version ultérieure. Il est utilisé pour formater les chaînes de sortie. Le paramètre `sep` signifie séparateur, il est utilisé avec la fonction `print()` pour spécifier le séparateur entre les arguments.

#### ► Exemples :

```
print('a', 'f', 'i', 'b', 'Poitiers', sep='')
#Résultat obtenu:afibPoitiers
print('12', '07', '2020', sep='/')
#Résultat obtenu:12/07/2020
print('michel', 'blache.eu', sep='@')
#Résultat obtenu:michel@blache.eu
```

41

Ifpa - Poitiers

## Python

### Instructions d'écriture et de lecture : print()

#### ► Paramètres end :

- Par défaut , la fonction print() de Python se termine par une nouvelle ligne.
- Un programmeur peut se demander comment imprimer sans retour à la ligne. La fonction print() de Python est fournie avec un paramètre appelé 'end'. Par défaut, la valeur de ce paramètre est '\n', c'est-à-dire le caractère de retour à la ligne.

Si la partie end='final' n'est pas précisée Python utilise end="\n", ce qui signifie que l'on passe à la ligne après l'affichage de text.

42

Ifpa - Poitiers

## Python

### Instructions d'écriture et de lecture : print()

#### ► Utilisation: imprimer des lignes vides

- Pour imprimer des lignes vides vous pouvez utiliser l'une des méthodes suivantes:
- Utiliser un chiffre représentant le nombre de ligne vide suivi de '\*' et un antislash et n comme le montre l'exemple suivant:

```
print ( 5 * " \n " )
```

- Ou remplacer le nombre des lignes vides par des antislash et n comme le montre l'exemple suivant:

```
print ( " \n \n \n \n \n \n " )
```

43

lfpa - Poitiers

## Python

### Instructions d'écriture et de lecture : print()

#### Utilisation: imprimer sur la même ligne

- Par défaut, la fonction print() de Python se termine par une nouvelle ligne. Par défaut, la valeur de ce paramètre est '\n', c'est-à-dire le caractère de retour à la ligne.

```

C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("affiche et passe à la ligne")
affiche et passe à la ligne
>>> print("Toujours "," dans la même ligne")
Toujours  dans la même ligne
>>> print("termine par un + et pas par un passage à la ligne",end="+")
termine par un + et pas par un passage à la ligne>>> print("Je suis sur la même ligne")
Je suis sur la même ligne
>>> print()# simple passage à la ligne
>>> _
  
```

44

lfpa - Poitiers

## Python

### Instructions d'écriture et de lecture : print()

#### Utilisation: imprimer des chaînes formatées

- L'opérateur spécial % permet de créer une sortie formatée. Il faut deux opérandes: une chaîne formatée et une valeur.

```
print ( "pi = % s " % "3.14159" )
```

% s est un spécificateur de conversion qui indique à Python comment convertir la valeur, dans ce cas % s signifie convertir la valeur en chaîne.

```
print ( " % s = % s " % ( "pi" , 3.14159 ) )
```

Les spécificateurs de conversion peuvent également convertir des valeurs en float, en entiers, etc.

45

lipa - Poitiers

## Python

### Instructions d'écriture et de lecture : print()

#### ► Utilisation: Plusieurs instructions sur une ligne

- Vous pouvez utiliser plusieurs fonctions print() dans la même ligne on les sépare par des point-virgule(';'), comme le montre l'exemple suivant:

- `print (" Bonjour ") ; print (" Monsieur "); print (" Zola ")`

46

lipa - Poitiers

## Python

### Instructions d'écriture et de lecture : print()

#### ► Ecriture formatée : Exemple

- La méthode .format() permet une meilleure organisation de l'affichage des variables.
- Cette méthode formate là où les valeurs sont spécifiées et les insère dans l'espace réservé de la chaîne, il est défini à l'aide d'accolades : {}.
- Les espaces réservés peuvent être identifiés à l'aide d'index nommés, d'index {number} numérotés {0} ou même d'espaces réservés vides {}.

```
txt1 = "Mon nom est {nom}, j'ai {age} ans".format(nom = " Blache", age = 61)
```

```
txt2 = "Mon nom est {0}, j'ai {1} ans".format(" Blache", 61)
```

```
txt3 = "Mon nom est {}, je ai {} ans".format(" Blache", 61)
```

47

lfpa - Poitiers

## Python

### Instructions d'écriture et de lecture : print()

#### ➤ Ecriture formatée : Écrire un résultat lisible

- Parfois le résultat obtenu présente trop de décimales. Pour écrire le résultat plus lisiblement, vous pouvez spécifier dans les accolades {} le format qui vous intéresse.
- Dans l'exemple suivant, vous voulez formater un float, avec seize décimales pour l'afficher avec deux puis trois décimales :

```
result = (4500 + 2575) / 14800
print(" Le résultat est ", result)
# Le résultat est 0.4780405405405405
print(" Le résultat est {:.2f}".format(result))
# Le résultat est 0.48
print(" Le résultat est {:.3f}".format(result))
# Le résultat est 0.478
```

48

lfpa - Poitiers

## Python

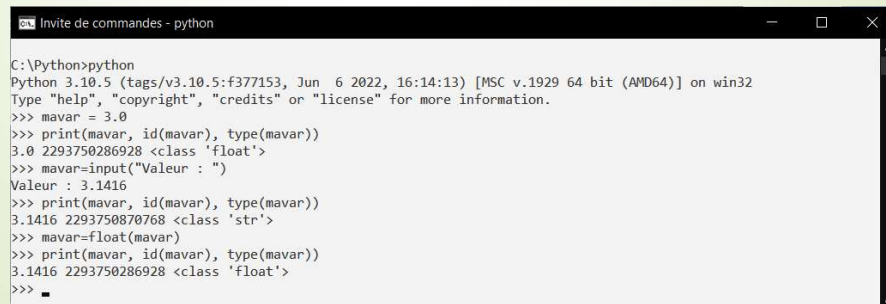
### Instructions d'écriture et de lecture : input()

#### ➤ Syntaxe :

- valeur = input (texte)

#### ➤ Fonctionnement :

- Quoi que vous entriez comme entrée, la fonction input() la convertit en chaîne. Si vous entrez une valeur entière, la fonction input() la convertira en chaîne.



```
Invite de commandes - python
C:\Python>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> mavar = 3.0
>>> print(mavar, id(mavar), type(mavar))
3.0 2293750286928 <class 'float'>
>>> mavar=input("Valeur : ")
Valeur : 3.1416
>>> print(mavar, id(mavar), type(mavar))
3.1416 2293750870768 <class 'str'>
>>> mavar=float(mavar)
>>> print(mavar, id(mavar), type(mavar))
3.1416 2293750286928 <class 'float'>
>>>
```



49

Ifpa - Poitiers

## Python

### La structure de contrôle : structure conditionnelle if

- Supposons que nous souhaitons définir la fonction valeur absolue :  $|x| = \{x \text{ si } x \geq 0 \text{ ou } -x \text{ si } x < 0\}$
- Nous devons alors utiliser une instruction qui opère une disjonction de cas. En Python, il s'agit de l'instruction de choix introduite par le mot-clé **if**. La syntaxe est alors:

```

Invite de commandes - python
C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> def vabs(x) :
...     if x >= 0 :
...         return x
...     else :
...         return -x
...
>>> vabs(2)
2
>>> vabs(-2)
2
>>>

```

50

Ifpa - Poitiers

## Python

### La structure de contrôle : structure conditionnelle if

- Observons la syntaxe de la structure de choix if. Tout d'abord, le mot if est suivi d'une condition de choix : quel est l'objet renvoyé par l'interpréteur lorsqu'il évalue cette condition ?

```

Invite de commandes - python
>>> 2 >= 0
True
>>> b = 2 >= 0
>>> b, id(b), type(b)
(True, 140714947627880, <class 'bool'>)
>>>

```

- La condition « $2 \geq 0$ » est de type booléen. Une variable booléenne ne peut prendre que deux valeurs : True (vrai) ou False (faux).
- La condition de choix est vérifiée quand l'évaluation de cette condition renvoie le booléen True et l'interpréteur exécute alors la suite d'instructions qui se trouve dans le premier bloc d'instructions. Dans le cas contraire l'interpréteur saute au bloc d'instructions situé après le mot-clé else et délimité par l'indentation.

51

Ifpa - Poitiers

## Python

### La structure de contrôle : structure conditionnelle if

- Notons que lorsque les conditions et les instructions d'une structure conditionnelle sont assez brèves, on peut les écrire sur une seule ligne :

```

Invite de commandes - python
Microsoft Windows [version 10.0.19044.2130]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> def valeur_absolue(x):
...     return x if x >= 0 else -x
...
>>> valeur_absolue(6)
6
>>> valeur_absolue(-6)
6
>>>

```

52

Ifpa - Poitiers

## Python

### La structure de contrôle : structure conditionnelle if

```

Invite de commandes - python
C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Soit l'équation ax**2 + bx + c
>>> def nbre_solution(a,b,c):
...     if a == 0 :
...         print("L'équation est du premier degré")
...     else :
...         delta = b**2 - 4*a*c
...         if delta > 0:
...             print("L'équation possède deux solutions réelles")
...         else :
...             if delta == 0:
...                 print("L'équation possède une solution réelle")
...             else :
...                 print("L'équation ne possède pas de solution réelle")
...
>>> nbre_solution(0,4,2)
L'équation est du premier degré
>>> nbre_solution(1,4,2)
L'équation possède deux solutions réelles
>>> nbre_solution(8,4,2)
L'équation ne possède pas de solution réelle
>>> ■

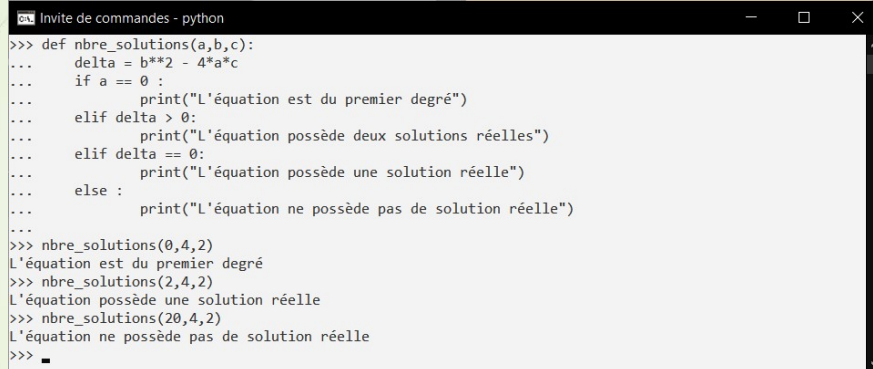
```

53

Ifpa - Poitiers

## Python

### La structure de contrôle : structure conditionnelle if



```

Invite de commandes - python
>>> def nbre_solutions(a,b,c):
...     delta = b**2 - 4*a*c
...     if a == 0 :
...         print("L'équation est du premier degré")
...     elif delta > 0:
...         print("L'équation possède deux solutions réelles")
...     elif delta == 0:
...         print("L'équation possède une solution réelle")
...     else :
...         print("L'équation ne possède pas de solution réelle")
...
>>> nbre_solutions(0,4,2)
L'équation est du premier degré
>>> nbre_solutions(2,4,2)
L'équation possède une solution réelle
>>> nbre_solutions(20,4,2)
L'équation ne possède pas de solution réelle
>>>

```

- Dans la succession de choix, le mot-clé elif est une abréviation pour else if. Le principal intérêt de la deuxième syntaxe est surtout que les différents résultats potentiellement renvoyés par la fonction apparaissent avec le même décalage d'indentation, d'où une syntaxe plus lisible.

54

Ifpa - Poitiers

## Python

### La structure de contrôle : conditionnelle match / case

- Avec la version 3.1 de python arrive l'instruction match / case équivalente au switch / case du langage C.
- Exemple d'utilisation :
 

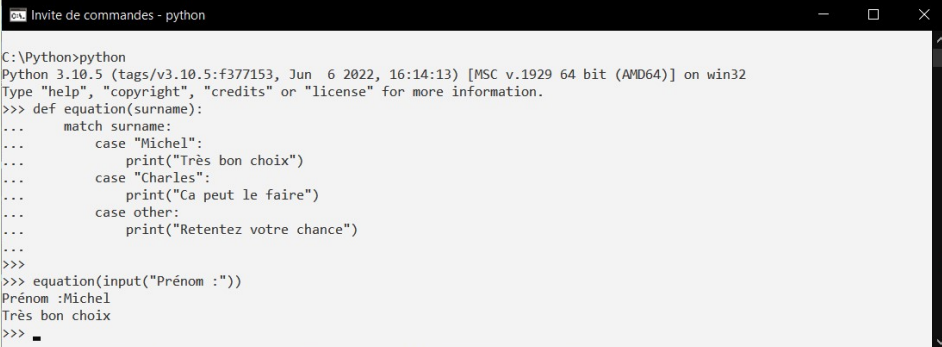
```

choix=input("Saisir un chiffre de 1 à 3 : ")
match choix:
    case "1":
        print("Choix :", choix)
    case "2":
        print("Choix :", choix)
    case "3":
        print("Choix :", choix)
    case other:
        print("Vous êtes en dehors des clous")
      
```

55

## Python

### La structure de contrôle : conditionnelle match / case



```

C:\Python>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> def equation(surname):
...     match surname:
...         case "Michel":
...             print("Très bon choix")
...         case "Charles":
...             print("Ca peut le faire")
...         case other:
...             print("Retentez votre chance")
...
>>>
>>> equation(input("Prénom :"))
Prénom :Michel
Très bon choix
>>>
  
```

56

## Python

### La boucle for

#### ■ Présentation

- Les boucles permettent de répéter une certaine opération autant de fois que nécessaire. En Python, on trouve deux types de boucle :
  - La boucle for dans laquelle une variable parcourt une collection de valeurs.
  - La boucle while qui se répète tant qu'une condition est vraie.
- L'instruction "for" est une structure de contrôle itérative complète qui permet de répéter l'exécution d'une suite d'instructions un nombre fini de fois connu à l'avance. L'instruction "for" travaille sur des séquences. Elle est spécialisée dans le parcours d'une séquence de plusieurs éléments.
- L'indentation, souvent présentée comme un moyen de rendre les programmes plus lisibles, est ici intégrée à la syntaxe du langage. Il n'y a pas non plus de délimiteurs entre deux instructions autre qu'un passage à la ligne.

57

Ifpa - Poitiers

## Python

### La boucle for

#### ■ Syntaxe

```
for var in séquence:  
    instruction 1  
    ...  
    instruction n
```

Où var est un élément de l'ensemble séquence. Les instructions 1 à n sont exécutées pour chaque élément var de l'ensemble séquence.

Cette séquence peut être une chaîne de caractères, un tuple, une liste, un dictionnaire, un set ou tout autre type.

```
t = (1, 2, 3, 4) #tuple  
for x in t:  
    print(x)
```

58

Ifpa - Poitiers

## Python

### La boucle for

```
#itération de 0 à 4  
for i in range(5):  
    print(i)
```

```
#itération de 1 à 4  
for i in range(1,5):  
    print(i)
```

```
#itération de 5 à 1 (décrémentation)  
for i in range(5,0,-1):  
    print(i)
```

59

Ifpa - Poitiers

## Python

### La boucle for

```

Invite de commandes - python

C:\Python>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(1,11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0,30,5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0,10,3))
[0, 3, 6, 9]
>>> list(range(0,-6,-1))
[0, -1, -2, -3, -4, -5]
>>> list(range(0)), list(range(1,0))
([], [])
>>> list(range(9,0,-1))
[9, 8, 7, 6, 5, 4, 3, 2, 1]
>>>

```

60

Ifpa - Poitiers

## Python

### La boucle while

#### ■ Présentation

- En anglais “while” signifie “Tant que”. Pour créer une boucle, il faut donc utiliser ce mot clé suivi d’une indication qui dit quand la boucle s’arrête.
- Lorsque l’on souhaite effectuer une même série d’instructions tant qu’une condition donnée reste vérifiée, on utilise une boucle “while”.
- Cette boucle “while” est utilisée en particulier dans le cas où l’on ne peut déterminer à l’avance le nombre d’itérations nécessaires à la résolution de notre problème.
- Le but de la boucle “while” est de répéter certaines instructions tant qu’une condition est respectée.
- L’avantage de “while” est donc qu’il ne faut pas connaître à l’avance le nombre de fois où on devra répéter notre boucle.

61

Ifpa - Poitiers

## Python

### La boucle while

#### ■ Syntaxe

```
while condition booléenne:
    Instruction 1 # Tant que la condition booléenne est vérifiée
    Instruction 2 # Faire les différentes instructions qui suivent
    .....
    Instruction n
```

- Le corps de la boucle (c'est-à-dire le bloc d'instructions indentées) sera répété tant que la condition est vraie.

```
x = 1
while x < 10:
    print("x a pour valeur", x)
    x += 1
print("Fin")
```

62

Ifpa - Poitiers

## Python

### Break, Continue et Pass

- L'utilisation des boucles for et while vous permet d'automatiser et de répéter des tâches de manière efficace, mais vous pouvez être confronté à une situation dans laquelle vous devez quitter complètement une boucle lorsqu'une condition externe est déclenchée ou il peut également y avoir une situation où vous souhaitez ignorer une partie de la boucle et démarrer la prochaine exécution.
- Pour surmonter une telle situation, Python fournit des instructions break, continue et pass. Une boucle for ou while est destinée à itérer jusqu'à ce que la condition donnée échoue. Lorsque vous utilisez une instruction break, continue ou pass, le flux de la boucle est modifié par rapport à sa manière normale.
- Ces trois instructions permettent de modifier le comportement d'une boucle (for ou while) avec un test.



63

Ifpa - Poitiers

## Python

### Break, Continue et Pass

- L'instruction « break » est utilisée pour quitter une boucle for ou while. Le but de cette instruction est de terminer immédiatement l'exécution de la boucle (for ou while) et le contrôle du programme passe à l'instruction après la dernière instruction de la boucle.

- Syntaxe

```
while (expression1) :  
    statement_1  
    statement_2  
    .....  
    if expression2 :  
        break
```

64

Ifpa - Poitiers

## Python

### Break, Continue et Pass

- Exemple « break »

```
nombres = (1, 2, 3, 4, 5, 6, 7, 8, 9)  
num_sum = 0  
count = 0  
for x in nombres:  
    num_sum = num_sum + x  
    count = count + 1  
    if count == 5:  
        break  
print("Somme du premier ",count,"entiers est: ", num_sum)
```



65

Ifpa - Poitiers

## Python

### Break, Continue et Pass

- L'instruction « continue » est utilisée dans une boucle while ou for pour amener le contrôle en haut de la boucle sans exécuter les instructions qui restent à l'intérieur de la boucle.
- L'instruction continue sera dans le bloc de code sous l'instruction de boucle, généralement après une instruction + if + conditionnelle.

#### ■ Syntaxe

```
while (expression1) :  
    statement_1  
    statement_2  
    .....  
    if expression2 :  
        continue
```

66

Ifpa - Poitiers

## Python

### Break, Continue et Pass

#### ■ Exemple « continue »

```
nombres = (1, 2, 3, 4, 5, 6, 7, 8, 9)  
num_sum = 0  
count = 0  
for x in nombres:  
    count = count + 1  
    if count == 5:  
        continue  
    num_sum = num_sum + x  
print("Somme du premier ",count,"entiers est: ", num_sum)
```

67

Ifpa - Poitiers

## Python

### Les listes

- Une liste est une structure de données qui contient une série de valeurs.
- Les éléments d'une liste sont séparés par des virgules et définis entre crochets []. Le type des listes en Python est list.

#### ■ Exemple

```
jour=["lundi", "mardi", "mercredi", "jeudi", "vendredi"]
loto=[2,14,26,34,45]
```

68

Ifpa - Poitiers

## Python

### Les listes

```
jour=["lundi", "mardi", "mercredi", "jeudi", "vendredi"]
for x in jour:
    print(id(x),type(x),x)
long=len(jour)
print("Longueur du tableau : ",long)
for x in range(long):
    print("indice ",x," Jour : ",jour[x])

loto=[2,14,26,34,45]
for x in loto:
    print(id(x),type(x),x)
long=len(loto)
print("Longueur du tableau : ",long)
for x in range(long):
    print("indice ",x," Numéro : ",loto[x])
```

69

Ifpa - Poitiers

## Python

### Les listes : Création

- méthode `list ()` .
- Accepte une séquence ou un tuple comme argument et se convertit en une liste Python.

#### ➤ Exemple

```
myname='michel'
lsname=list(myname)
print(lsname)
###Production###
['m', 'i', 'c', 'h', 'e', 'l']
```

70

Ifpa - Poitiers

## Python

### Les listes : Création

- méthode `list ()` .
- Accepte une séquence ou un tuple comme argument et se convertit en une liste Python.

#### ➤ Exemple

```
myname='michel'
lsname=list(myname)
print(lsname)
###Production###
['m', 'i', 'c', 'h', 'e', 'l']
```

71

Ifpa - Poitiers

## Python

### Les listes : Création

- Fonction `range()` .
- La fonction `range ()` permet de créer une liste. Elle permet d'utiliser trois paramètres. Le premier contient le numéro de départ de la liste, le second le dernier nombre de la liste et finalement l'incrément entre chaque nombre généré.

#### ■ Exemple

```
print(list(range(0,10)))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

72

Ifpa - Poitiers

## Python

### Les listes : Création

#### ■ Liste vide

- `Tab=[]` : initialisation d'une liste vide.
- `Tab=[0]*10` : permet d'initialiser une liste ou un tableau de 10 cases avec la valeur 0.

#### ■ Exemple

```
print(list(range(0,10)))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

73

Ifpa - Poitiers

## Python

### Les listes : Création

```
table=[]
print(table, id(table),type(table))

table=[0]*10
print(table, id(table),type(table))

max=len(table)
print(max)
for x in range(0,max):
    table[x]=x+1

print(table, id(table),type(table))
```

74

Ifpa - Poitiers

## Python

### Tri d'une liste

- Les listes Python ont une méthode native `list.sort()` qui modifie les listes elles-mêmes. Il y a également une fonction native `sorted()` qui construit une nouvelle liste triée depuis un itérable.

■ Exemple :

- Un tri ascendant simple est très facile : il suffit d'appeler la fonction `sorted()`. Elle renvoie une nouvelle liste triée :

```
>>> sorted([5, 2, 3, 1, 4])
[1, 2, 3, 4, 5]
```

75

Ifpa - Poitiers

## Python

### Tri d'une liste

```

Invite de commandes - python
C:\Users\blach>python
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> sorted([5, 2, 3, 1, 4])
[1, 2, 3, 4, 5]
>>>
>>> a = [5, 2, 3, 1, 4]
>>> a.sort()
>>> a
[1, 2, 3, 4, 5]
>>>
>>> a = [5, 2, 3, 1, 4]
>>> sorted(a)
[1, 2, 3, 4, 5]
>>> sorted(a,reverse=True)
[5, 4, 3, 2, 1]
>>>

```

76

Ifpa - Poitiers

## Python

### Tri d'une liste

- Le module operator contient les fonctions itemgetter(), attrgetter(), et methodcaller().

```

Invite de commandes - python
>>> from operator import itemgetter
>>> eleve=[('Valérie','A',24),('Georges','D',32),('Pascale','C',28)]
>>> sorted(eleve,key=itemgetter(1))
[('Valérie', 'A', 24), ('Pascale', 'C', 28), ('Georges', 'D', 32)]
>>> sorted(eleve,key=itemgetter(0))
[('Georges', 'D', 32), ('Pascale', 'C', 28), ('Valérie', 'A', 24)]
>>>

```

77

Ifpa - Poitiers

## Python

### Les exceptions

- Même si une instruction ou une expression est syntaxiquement correcte, elle peut générer une erreur lors de son exécution. Les erreurs détectées durant l'exécution sont appelées des exceptions et ne sont pas toujours fatales : nous apprendrons bientôt comment les traiter dans vos programmes. La plupart des exceptions toutefois ne sont pas prises en charge par les programmes, ce qui génère des messages d'erreurs comme celui-ci.

#### ■ Exemple

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

78

Ifpa - Poitiers

## Python

### Les exceptions

- Il est possible d'écrire des programmes qui prennent en charge certaines exceptions.

#### ■ Exemple

```
a=int(input('Valeur de a :'))
try:
    x = a + 1/a
except ZeroDivisionError:
    print("Division par zéro !")
else:
    print(x)
```

79

Ifpa - Poitiers

## Python

### Les exceptions

- L'instruction try fonctionne comme ceci :
- premièrement, la clause try (instruction(s) placée(s) entre les mots-clés try et except) est exécutée ;
- si aucune exception n'intervient, la clause except est sautée et l'exécution de l'instruction try est terminée ;
- si une exception intervient pendant l'exécution de la clause try, le reste de cette clause est sauté. Si le type d'exception levée correspond à un nom indiqué après le mot-clé except, la clause except correspondante est exécutée, puis l'exécution continue après l'instruction try ;
- si une exception intervient et ne correspond à aucune exception mentionnée dans la clause except, elle est transmise à l'instruction try de niveau supérieur ; si aucun gestionnaire d'exception n'est trouvé, il s'agit d'une exception non gérée et l'exécution s'arrête avec un message comme indiqué ci-dessus.

80

Ifpa - Poitiers

## Python

### Les exceptions

- L'instruction try fonctionne comme ceci :
- Une instruction try peut comporter plusieurs clauses except pour permettre la prise en charge de différentes exceptions. Mais un seul gestionnaire, au plus, sera exécuté.
- Une même clause except peut citer plusieurs exceptions sous la forme d'un tuple entre parenthèses, comme dans cet exemple :
 

```
except (RuntimeError, TypeError, NameError):
    pass
```
- L'instruction try ... except accepte également une clause else optionnelle qui, lorsqu'elle est présente, doit se placer après toutes les clauses except.



81

Ifpa - Poitiers

## Python

### Les exceptions

#### ► Liste des exeptions:

► Les plus courantes sont OverflowError, ZeroDivisionError.

► Liste exhaustive sur internet :

► <https://docs.python.org/fr/3/library/exceptions.html>

82

Ifpa - Poitiers

## Python

### Quelques primitives d'usage courant

<code>divmod(47, 5)</code>	# quotient, reste	# rép.: (9, 2)
<code>abs(-1.3)</code>	# valeur absolue d'un nombre	# rép.: 1.3
<code>round(-1.3)</code>	# arrondi d'un flottant	# rép.: -1
<code>pow(10, -5)</code>	# exponentiation	# rép.: 1e-05
<code>maxX([3, 8, 9, -1])</code>	# maximum d'une séquence	# rép.: 9
<code>min("alphabet")</code>	# minimum d'une séquence	# rép.: a
<code>sum(range(100))</code>	# somme d'une séquence	# rép.: 4950

83

Ifpa - Poitiers

## Python

### Quelques primitives d'usage courant

<code>divmod(47, 5)</code>	# quotient, reste	# rép.: (9, 2)
<code>abs(-1.3)</code>	# valeur absolue d'un nombre	# rép.: 1.3
<code>round(-1.3)</code>	# arrondi d'un flottant	# rép.: -1
<code>pow(10, -5)</code>	# exponentiation	# rép.: 1e-05
<code>maxX([3, 8, 9, -1])</code>	# maximum d'une séquence	# rép.: 9
<code>min("alphabet")</code>	# minimum d'une séquence	# rép.: a
<code>sum(range(100))</code>	# somme d'une séquence	# rép.: 4950

84

Ifpa - Poitiers

## Python

### Liste des primitives

Built-in Functions			
<b>A</b> <code>abs()</code> <code>aiter()</code> <code>all()</code> <code>any()</code> <code>anext()</code> <code>ascii()</code>	<b>E</b> <code>enumerate()</code> <code>eval()</code> <code>exec()</code>	<b>L</b> <code>len()</code> <code>list()</code> <code>locals()</code>	<b>R</b> <code>range()</code> <code>repr()</code> <code>reversed()</code> <code>round()</code>
<b>B</b> <code>bin()</code> <code>bool()</code> <code>breakpoint()</code> <code>bytearray()</code> <code>bytes()</code>	<b>F</b> <code>filter()</code> <code>float()</code> <code>format()</code> <code>frozenset()</code>	<b>M</b> <code>map()</code> <code>max()</code> <code>memoryview()</code> <code>min()</code>	<b>S</b> <code>set()</code> <code>setattr()</code> <code>slice()</code> <code>sorted()</code> <code>staticmethod()</code> <code>str()</code> <code>sum()</code> <code>super()</code>
<b>C</b> <code>callable()</code> <code>chr()</code> <code>classmethod()</code> <code>compile()</code> <code>complex()</code>	<b>G</b> <code>getattr()</code> <code>globals()</code>	<b>N</b> <code>next()</code>	<b>T</b> <code>tuple()</code> <code>type()</code>
<b>D</b> <code>delattr()</code> <code>dict()</code> <code>dir()</code> <code>divmod()</code>	<b>H</b> <code>hasattr()</code> <code>hash()</code> <code>help()</code> <code>hex()</code>	<b>O</b> <code>object()</code> <code>oct()</code> <code>open()</code> <code>ord()</code>	<b>V</b> <code>vars()</code>
	<b>I</b> <code>id()</code> <code>input()</code> <code>int()</code> <code>isinstance()</code> <code>issubclass()</code> <code>iter()</code>	<b>P</b> <code>pow()</code> <code>print()</code> <code>property()</code>	<b>Z</b> <code>zip()</code>
			<b>_</b> <code>__import__()</code>

85

lipa - Poitiers

## Python

### Primitive incluse dans des modules

```
Invite de commandes - python
>>> help(divmod)
Help on built-in function divmod in module builtins:

divmod(x, y, /)
    Return the tuple (x//y, x%y). Invariant: div*y + mod == x.

>>> divmod(37,5)
(7, 2)
>>> help(hypot)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'hypot' is not defined
>>> import math
>>> help(math.hypot)
Help on built-in function hypot in module math:

hypot(...)
    hypot(*coordinates) -> value

    Multidimensional Euclidean distance from the origin to a point.

    Roughly equivalent to:
        sqrt(sum(x**2 for x in coordinates))

    For a two dimensional point (x, y), gives the hypotenuse
    using the Pythagorean theorem: sqrt(x**2 + y**2).

    For example, the hypotenuse of a 3/4/5 right triangle is:

    >>> hypot(3.0, 4.0)
    5.0
>>>
```