



Toute reproduction de ce document doit faire l'objet d'une autorisation.

Introduction au langage Python

par :

Céline OULMI

11/12/2024

→ Caractéristiques générales

2

- Créé par Guido Van Rossum, fin 1980, aux pays bas.
- Open source (compatible avec la licence GPL)
- Portable (Unix, Linux, Windows, MacOS...)
- Plusieurs implémentations écrites en langage C.
- Autres implémentations écrites en Java et .Net.
- Orienté objet et de haut niveau.
- Evolutif.

→ Domaines d'application

3

- ▶ Enseignement de la programmation (projet initié par le DARPA).
- ▶ Prototypage rapide des applications.
- ▶ Le calcul scientifique et l'imagerie.
- ▶ Scripts d'administration système et réseaux
- ▶ Automatisation de déploiement des services.
- ▶ Développement Web (Scrapy, Django).

→ Suite :

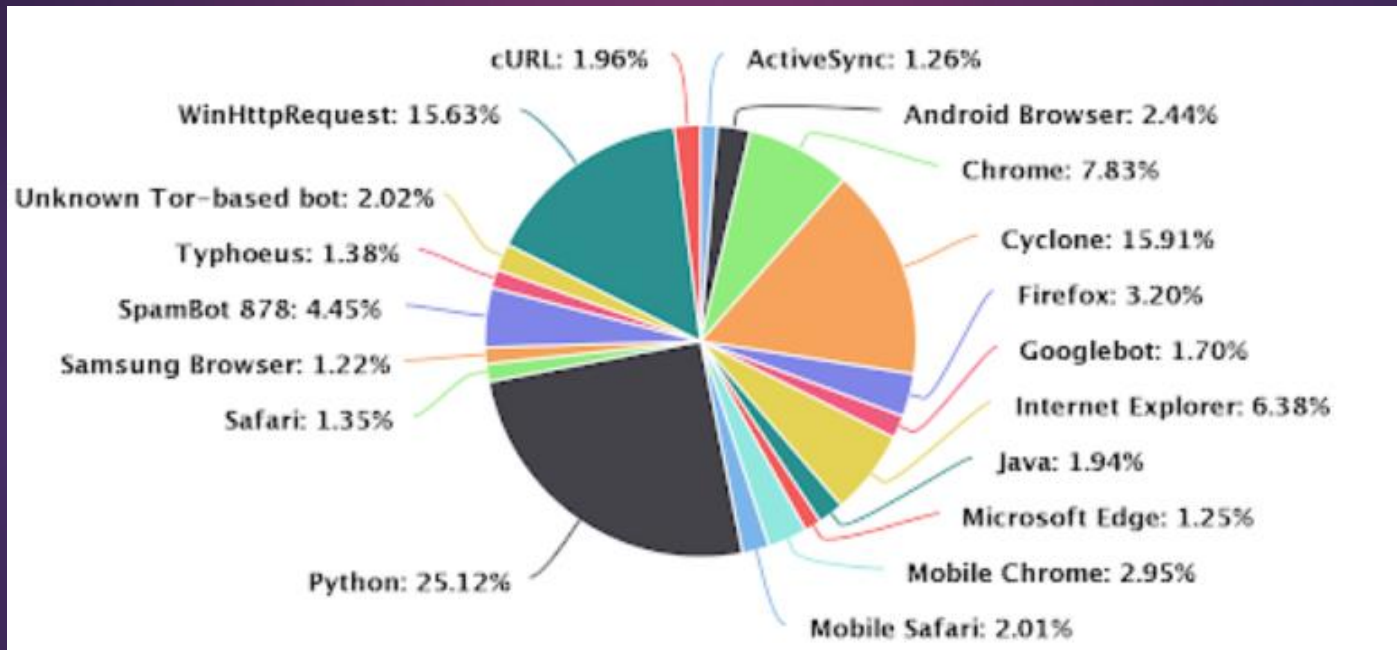
- ▶ Gestion de bases de données (mongoDB, SQLite, MySql...)
- ▶ Ecosystème pour Data Scientists (NumPy, Pandas, extraction et visualisation des données)
- ▶ Génération et création de graphique 2D et 3D (Matplotlib)
- ▶ Machine learning (Scikit learn) et big data (PySpark)....

Exemples de projets référents :

5

- ▶ Mozilla, Google, Yahoo
- ▶ Nasa (Swim, XPC....)
- ▶ Youtube
- ▶ Cisco
- ▶ Stockage en ligne (Dropbox, Ubuntu)
- ▶ Cloud-computing (Dotcloud, Nebula...)

Attention : une montée en puissance des attaques utilisant des outils codés avec Python!!!



- ▶ **25 %** des clients sont des outils écrits en Python utilisés par des malveillants.
- ▶ Les plus grands vecteurs d'attaques sont codés en Python → nouvelle arme de prédilection des hackers!!!
- ▶ Plus de **20%** des répertoires GitHub destinés à mettre en œuvre un outil d'attaque ou exploitant une vulnérabilité est codé en Python.
(source : globbsecurity.fr)

Références (minimales)

8

- ▶ www.Python.org
- ▶ <http://www.afpy.org/>

(Association Francophone Python)

- ▶ Ouvrage : « Apprendre à programmer avec Python 3 » de Gérard Swinnen, téléchargeable en pdf.
- ▶ « Le Zen de Python » :
(<http://python.org/dev/peps/pep-0020/>)

➔ Présentation technique

9

- ▶ Langage de Scripting, de prototypage et de développement de projets complexes.
- ▶ **M**ulti-paradigme (procédural/fonctionnel, objet)
- ▶ Python s'interface avec les langages C et Java.
- ▶ Multiplateforme, il produit du **byte-code**.
- ▶ Dynamiquement typé et fortement typé.
- ▶ Pour Python, tout est objet et tous les types sont des classes, il permet l'héritage multiple la surcharge des opérateurs...

- ▶ Il gère par lui-même ses ressources (mémoires, descripteurs de fichiers...).
- ▶ Il intègre un système d'exception pour la gestion des erreurs.
- ▶ Les bibliothèques permettent la manipulation de chaînes de caractères , de services Unix/linux (pipes, signaux, socket, threads)... , protocoles réseaux (HTTP, FTP, TCP/UDP...), la gestion de bases de données, interfaces graphiques (TkInter...).

➔ Plusieurs implémentations :

1. **Cpython** : écrite en C permettant des extensions de python . Elle est disponible sur toutes les plateformes.
2. **Jython** : écrite en Java permet l'utilisation des bibliothèques Java (Swing...), la JVM.
3. **Ironpython** : écrite en C# pour la plateforme .Net
4. **PyPy** : écrite en Python, niveau de services équivalent à Cpython.

➔ Projet de recherche européen d'un interpréteur Python en Python!

Comment exécuter un programme Python? ¹²



Technique mixte

- Interprétation du **byte code** compilé, puis
- L'exécution par la machine virtuelle Python



➔ Outils de développement

13

► Windows :

<http://python.org/download/>

Exemples d'IDE :

- VS Code,
- PyCharm,
- IDLE (éditeur interpréteur développé en Python),
- Spyder....

► **Mac OS :**

1. Python est préinstallé sur Mac, utilisé dans certains de ses composants .

(<https://www.python.org/downloads/mac-osx/>)

...

► **Linux/Unix :**

1. Usage natif de Python.

2. La console, selon le gestionnaire de la distribution :
(\$python -V, \$ sudo apt-get install python ...)

\$ sudo aptitude install python3

\$ sudo yum install python3

Cas d'IDLE (simple) :

1. En mode console : (fig 1)

- ▶ Taper le code au fur et à mesure.
- ▶ Une ligne de l'invite interactif commence par : **>>>** ou bien ... selon l'indentation.
- ▶ Créer un fichier source (avec.py) puis lancer la commande « **import nom_du_fichier** » pour l'exécution.

2. Via une interface graphique (EDI) : (fig 2)

- ▶ Rassembler un ensemble de commandes Python dans un fichier.py, c'est le **script Python**.
- ▶ Exécuter par une touche du menu de l'EDI (Run/F5)

Fig 1

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.1.2 (r312:79149, Mar 21 2010, 00:41:52) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 5+2
7
>>> |
```

Fig 2

```
module1.py - C:\Users\c\Desktop\module1.py
File Edit Format Run Options Windows Help
# Name:      module1
# Purpose:
#
# Author:    c
#
# Created:   15/02/2013
# Copyright: (c) c 2013
# Licence:   <your licence>
#-----
#!/usr/bin/env python

def main():
    pass

if __name__ == '__main__':
    main()

# programme principal -----
print("Donnez deux valeurs entieres :")
x = int(input("n1 = "))
y = int(input("n2 = "))
# ecriture classique :
if x < y:
    pt = x
else:
    pt = y
# ecriture compacte :
```



```
>>> help()
```

Welcome to Python 3.7's help utility!

If this is your first time using Python, you should definitely check out the tutorial on the Internet at <https://docs.python.org/3.7/tutorial/>.

Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", type "modules spam".

```
help> keywords
```

Here is a list of the Python keywords. Enter any keyword to get more help.

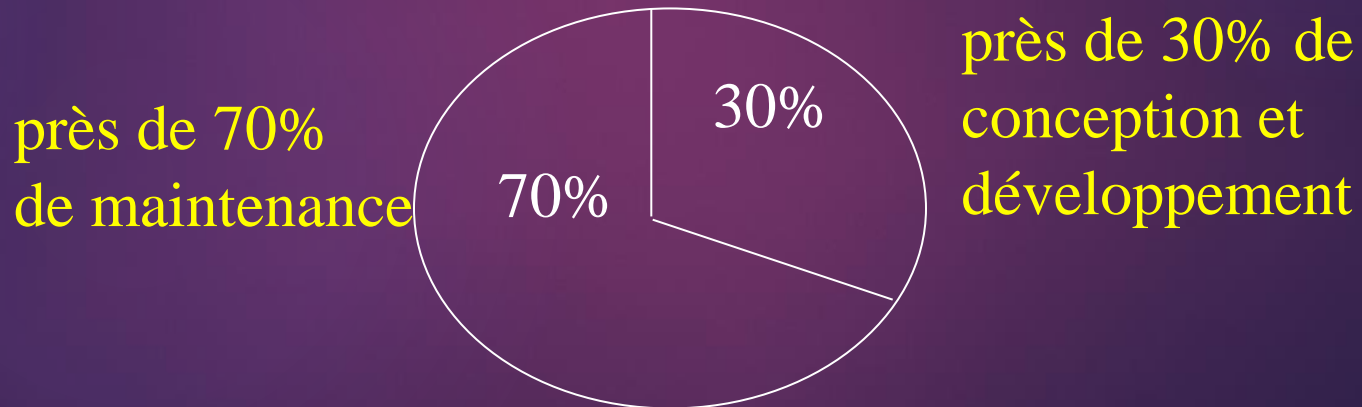
False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

```
>>> import sys
>>> dir(sys)
['_breakpointhook_', '_displayhook_', '_doc_', '_excepthook_', '_interactivehook_',
'ader_', '_name_', '_package_', '_spec_', '_stderr_', '_stdin_', '_stdout_', '_
blehook_', '_base_executable', '_clear_type_cache', '_current_frames', '_debugmallocstats',
lelegacywindowsfsencoding', '_framework', '_getframe', '_git', '_home', '_xoptions', 'addaud
, 'api_version', 'argv', 'audit', 'base_exec_prefix', 'base_prefix', 'breakpointhook', 'buil
ule_names', 'byteorder', 'call_tracing', 'copyright', 'displayhook', 'dllhandle', 'dont_writ
ode', 'exc_info', 'excepthook', 'exec_prefix', 'executable', 'exit', 'flags', 'float_info',
repr_style', 'get_asyncgen_hooks', 'get_coroutine_origin_tracking_depth', 'getallocatedblock
tdefaultencoding', 'getfilesystemencodeerrors', 'getfilesystemencoding', 'getprofile', 'getr
nlimit', 'getrefcount', 'getsizeof', 'getswitchinterval', 'gettrace', 'getwindowsversion', '
fo', 'hexversion', 'implementation', 'int info', 'intern', 'is_finalizing', 'last traceback'
```

➔ A voir plu loin pour d'autres modules

→ Objectifs méthodologiques d'un développement :

- ▶ Faire face à la complexité.
- ▶ Maîtriser les délais de développement.
- ▶ Maîtriser les coûts.



Les bases du langage Python

- ▶ Python est un langage orienté objets.
- ▶ Les objets sont regroupés en classes.
- ▶ La classe Python correspond au type Python.
- ▶ Une classe englobe des attributs et des méthodes. (Voir exemple Employés)

→ Structure d'un programme Python

- ▶ Une ligne se termine par un passage à la ligne
- ▶ Une instruction par ligne (ou séparées par “;”)
- ▶ Pas plus de 80 caractères (/ pour la continuité des lignes)
- ▶ **Indentation** pour un bloc : 4 car (ou TAB).

Exemple :

// En langage C

```
int a=0;  
for(int i=0;i<10;i++)  
{   a=a+i;  
    //pas de tabulation  
} // fin de bloc
```

// En Python

```
a=0  
for i in range(10) :  
    a+=i  
    // tabulation obligatoire  
// fin de bloc implicite
```

► Conventions de nommage/écriture des identifiants :

- > Variables en minuscule, exemple : **mon_id**
- > Constante en majuscule, exemple : **PI=3.14**
- > Classe, mixe des deux, exemple :

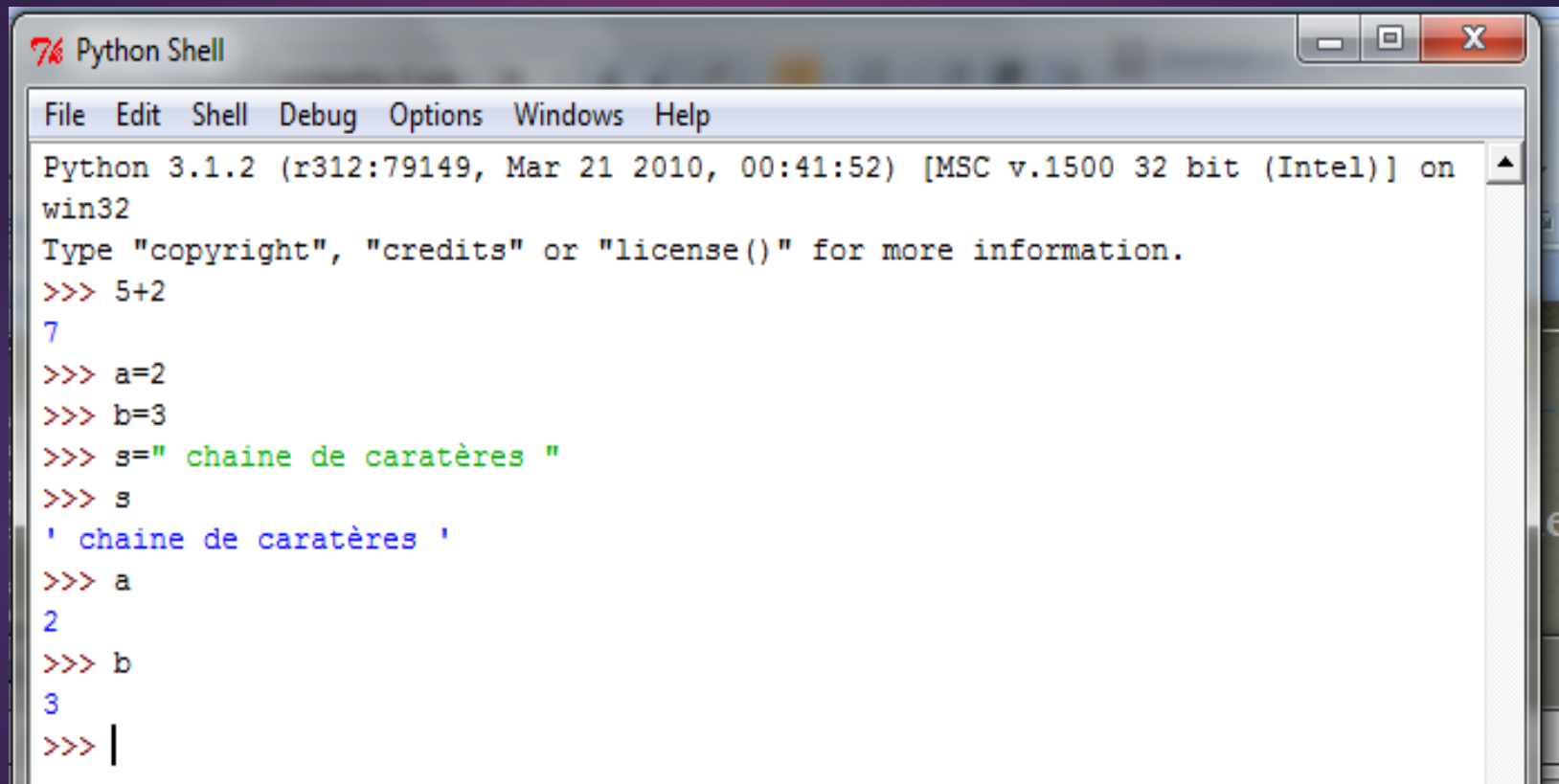
class MaClasse

- > Fonction, mixe des deux, exemple :

def maFonction()

→ Typage **dynamique** : python ne nécessite pas de déclaration de type

23



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.1.2 (r312:79149, Mar 21 2010, 00:41:52) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 5+2
7
>>> a=2
>>> b=3
>>> s=" chaîne de caractères "
>>> s
' chaîne de caractères '
>>> a
2
>>> b
3
>>> |
```

→ Les mots réservés :

and	del	from	None	True
as	elif	global	nonlocal	try
assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	
def	for	lambda	return	

→ Les opérateurs principaux :

Opérateur	Description
x or y	ou logique
x and y	et logique
not x	négation logique
<, <=, >, >=, ==, <>, !=	opérateurs de comparaison
is, is not	Test d'identité
in, not in	Appartenance à une séquence
x y	ou bits-à-bits
x ^ y	ou exclusif bits-à-bits
x & y	et bits-à-bits
x << y, x >> y	Décalage de x par y bits
x + y, x - y	addition ou concaténation / soustraction
x * y	multiplication ou répétition
x / y, x % y	division / reste de la div. (modulo)
-x	négation unaire

→ Les types numériques :

Constantes	Interprétation
314 / -2 / 0	Entiers normaux
314314314L	Entiers longs (taille illimitée)
1.23 / 3.14e-10 / 4E210	Virgules flottantes
0177 / 0x9ff	Constantes Octales et hexadécimales
3+4j / 3.0-4.0j	Constantes complexes

➔ Le type chaîne : (le caractère n'existe pas en Python)

Opération	Intérprétation
<code>s1=""</code>	chaîne vide
<code>s2="l'œuf"</code>	double guillemets
<code>bloc=""...""</code>	bloc à triple guillemet
<code>s1+s2, s2*3</code>	concaténation, répétition
<code>s2[i], s2[i:j], len(s2)</code>	indice, extraction, longueur
<code>"Hello %s" % 'World'</code>	formatage de chaîne
<code>for x in s2, 'o' in s2</code>	itération, appartenance

Le triple double guillemets permet d'entrer une chaîne de caractères sur plusieurs lignes, y compris les caractères de retour de ligne.

Exemple 1 :

```
>>>
>>> s1=""
>>> s2="CCNA2"
>>> s3="3A SRC SI"
>>> s1+s2+s3
'CCNA23A SRC SI'
>>> s2*3
'CCNA2CCNA2CCNA2'
>>> s2[3]
'A'
>>> len(s1)
0
>>> len(s2)
5
>>> len(s3)
9
```

Exemple 2 :

```
>>>
>>> "c'est %s bateau no %d" % (str, 25)
"c'est mon bateau no 25"
>>> 'mon'*5
'monmonmonmonmon'
>>> s="0123456789"
>>> s[0:]
'0123456789'
>>> s[0:4]
'0123'
>>> s[4]
'4'
>>> s[5:]
'56789'
>>> s[-3]
'7'
>>>
```

Ln: 34 Col: 0

Exemple 3 :

A screenshot of a Python 3.4.2 Shell window. The window has a yellow title bar with the text "Python 3.4.2 Shell" and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Windows", and "Help". The main text area shows the following text:

```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:15:05) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Chaise Basse".istitle())
True
>>> print("cHaise lonGue".isupper())
False
>>> print("petits enfants".isalpha())
False
>>> print("5 petits enfants".isalpha())
False
>>> print("123456789".isdigit())
True
>>> print("5 petits enfants".isalnum())
False
>>> print("5b".isalnum())
True
>>>
```

Exemple 4 :

```
>>> s="my_string"; type(s)
<class 'str'>
>>> s=[1,2,3,4]; type(s)
<class 'list'>
>>> s=int(2013); type(s)
<class 'int'>
>>> s=3.14; type(s)
<class 'float'>
```

→ Les entrées/sorties

- ▶ La saisie à l'écran se fait avec la fonction standard **input()**.
- ▶ Cette fonction effectue toujours une saisie en mode **texte** avec la possibilité de changer de type ensuite (**transtypage**)
- ▶ La sortie s'effectue par la fonction **print()**

Exemples :


```
File Edit Shell Debug Options Windows Help
Python 3.1.2 (r312:79149, Mar 21 2010, 00:41:52) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import sys
>>> nb_user=input("Entrez le nombre d'utilisateurs :")
Entrez le nombre d'utilisateurs :31
>>> print(type(nb_user))
<class 'str'>
>>> f1=float(nb_user)
>>> print(type(f1))
<class 'float'>
>>> f1
31.0
>>> x,y=10,20
>>> print(x,y)
10 20
>>> print("Somme :",x+y)
Somme : 30
>>> print(y-x, "est la différence")
10 est la différence
>>> print("le produit de",x," par ",y, " vaut :", x*y)
le produit de 10 par 20 vaut : 200
>>> print()

>>> print(x,end=" ")
10
>>> print("On a <",2**16,"> cas possible", sep="###")
On a <###65536###> cas possible
```

➔ Le type booléen

34

```
>>> a,b=10,20
>>> a>b
False
>>> a<b
True
>>> a=True
>>> b=False
>>> a or b
True
>>> a and b
False
>>> not(a>b)
False
>>> a
True
>>> b
False
>>> a,b,c=True,False,True
>>> b=0
>>> a&b
0
>>> print(type(b))
<class 'int'>
>>> print(type(a))
<class 'bool'>
>>>
```

11/12/2024

→ Les instructions conditionnelles

1. L'instruction **not** : elle renverse une condition
2. L'instruction **if**

Exemple :

```
nb=int(input("Saisir un entier :"))
if nb < 0:
    print("Nombre négatif")
else :
    print("Nombre positif ou nul")
```

```
nb=int(input("Saisir un entier :"))
if not (nb < 0):  ### positif ou nul
    print("Nombre positif ou nul")
else :
    print("Nombre négatif")
```

3. Les instructions **if/elif/else**

Exemple :

```
### Test if/elif/else
nb=int(input("Saisir un entier :"))
if nb < 0:
    print("Nombre négatif")
elif nb> 0:
    print("Nombre positif")
else:
    print("Nomvre nul")
```

Remarque: le mot clé **switch-case** n'existait pas en Python. Il existe depuis la version 3.10 avec **match-case**

Exemple :

```
match jour:
  case 1:
    return 'Lundi'
  case 2:
    return 'Mardi'
  case 3:
    return 'Mercredi'
  case 4:
    return 'Jeudi'
  case 5:
    return 'Vendredi'
  case 6:
    return 'Samedi'
  case 7:
    return 'Dimanche'
  case _:
    return 'Pas un jour de la semaine'
```

3. Les instructions **else/ elif** en général :

```
if <test1>:  
    <blocs d'instructions 1>  
elif <test2>:  
    <blocs d'instructions 2>  
else:  
    <blocs d'instructions 3>
```

→ Les itérations

1. L'instruction **for**

Exemple 1 :

```
>>> for a in [5,7,11,13]:  
        print('%d est un nombre premier' % a)  
  
5 est un nombre premier  
7 est un nombre premier  
11 est un nombre premier  
13 est un nombre premier  
>>>
```

→ Les itérations

1. L'instruction **for**

Exemple 2 :

```
Chaine = "Cours en Python"
print (Chaine)
## Afficher tous les caractères de la chaine
for c in Chaine:
    print(c)
#####
#### Incrémentation de valeurs entières
for i in range(10) :    #### 10 valeurs de 0 à 9
    print(i, "initial")
    i=i+2
    print(i, end=", ")
    print()
```


2. L'instruction **while**

Exemple 1 :

```
a=1
## Affichage des 6 valeurs de a: de 1 à 6

while a<7:
    print(a)
    a=a+1
print (" Traitement terminé...voici le a:",a)
```

→ Ruptures de séquences?

1. Instruction **break** : termine l'itération.
2. Instruction **return**: autre moyen de terminer l'itération.

2. L'instruction **while**

Exemple 1 : à compléter!!!!!!

```
a=10  
while a<=10:  
    a-=1
```

➔ Ruptures de séquences?

1. Instruction **break** : termine l'itération.

2. Instruction **return**: autre moyen de terminer l'itération.

2. L'instruction **while**, **continue**, **break**

Exemple 2 :

```
## Utilisation du While/Break/Continue
x=0
while x>=0:
    x=x+1
    if x==5:continue
    elif x==10: break    ### quitter la boucle
    else: print("Itération numéro :", x)

print( "Itértation numéro",x, "n'est pas réalisable")
print(" A bientôt....")
```

3. L'instruction **continue** : permet de court-circuiter des itérations.

44

Exemple :

```
>>> for x in range(1,11):  
    if x==5:  
        continue  
    print(x, end=" ")  
    print("la boucle a sauté la valeur 5")
```

```
1 la boucle a sauté la valeur 5  
2 la boucle a sauté la valeur 5  
3 la boucle a sauté la valeur 5  
4 la boucle a sauté la valeur 5  
6 la boucle a sauté la valeur 5  
7 la boucle a sauté la valeur 5  
8 la boucle a sauté la valeur 5  
9 la boucle a sauté la valeur 5  
10 la boucle a sauté la valeur 5  
>>> |
```

11/12/2024

Combinaison boucle while et conditionnelle :

```
while <test1>:  
    <blocs d'instructions 1>  
    if <test2>: break  
    if <test3>: continue  
else:  
    <blocs d'instructions 2>
```

Combinaison boucle for et la conditionnelle :

46

```
for <cible> in <objet>:  
    <blocs d'instructions>  
    if <test1>: break  
    if <test2>: continue  
else:  
    <blocs d'instructions>
```

```
#### while avec if/else
i=1
while True:
    if(i<=5):
        print("Hello", i)
        i=i+1
    else :
        print("nombre de fois dépassé....")
        break
print("fin...")
```

```
### boucle for avec else
```

```
for mot in ["une", "liste", "de", "mots"]:  
    if mot.startswith("mo"):  
        print("mot trouvé")  
        break  
else :  
    print("mot non trouvé....")
```


➔ Notion de fonctions

➤ Une fonction est un sous-programme défini par le mot réservé **def** :

Exemple :

```
def Carré(N) :  
    return N*N  
  
N=8 ### ou autre valeur  
print("Le carré de ",N, "est :", Carré(N))
```

→ Autres illustrations :

Exemple 1 :

```
>>>  
>>> def say_hello(to) :  
    print("Hello %s"%to)  
  
>>> to="sam"  
>>> say_hello(to)  
Hello sam  
>>>
```

Exemple 2 :

```
>>> def print_add(a,b):  
    def add(a,b):  
        return a+b  
    print(add(a,b))  
  
>>> print_add(4,6)  
10  
>>> |
```

Une fonction renvoie toujours une valeur unique.
Par défaut il s'agit de **None**.

```
>>> def f():  
    pass  
  
>>> print(f())  
None
```

➔ Remarque :

Une fonction ou une classe peut être définie sans instructions (vide) avec le mot-clé **pass**

Exemples :

```
>>> def ma_fonction():  
    pass
```

```
>>> class ma_classe(object):  
    pass
```

En général :

```
def <nom_fonction>(arg1, arg2,... argN):  
    ...  
    bloc d'instructions  
    ...  
    return <valeur(s)>
```

Exemple :

```
### Table de multiplication  
def table(base, debut, fin):  
    print("une partie de la table de multiplication: ")  
    n=debut  
    tableau=[] ### liste vide  
    while n!=fin:  
        print (n*base)  
        tableau.append(n*base)  
        n=n+1  
    return tableau  
  
L=table(2,0,11)
```

➔ Notion de modules

54

- Un module est un programme Python qui contient des fonctions (méthodes) variées et prêtes à l'emploi.
- Un module est importé par le mot réservé `import`

Exemple : `import sys`

➔ les modules les plus courants

os : pour communiquer avec le système
sys : pour la gestion des entrées/sorties
random : pour la génération des
nombres aléatoires
time : pour la gestion du temps
calendaire : fonctions calendriers
math : fonctions et constantes math...
Tkinter : interfaces graphiques
urllib : pour récupérer des données
d'internet avec Python.....

Exemple: module random

```
>>> import random
>>> random.randint(0,10)
9
>>> del random
>>> random.randint(0,10)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    random.randint(0,10)
NameError: name 'random' is not defined
>>> import random as rand
>>> rand.randint(0,10)
10
>>> a=rand.randint(0,10)
>>> a
10
```


Les méthodes et les variables associées :

57

```
>>> import random
>>> dir(random)
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'System
Random', 'TWOPI', '_BuiltinMethodType', '_MethodType', '__all__', '__builtins__
', '__doc__', '__file__', '__name__', '__package__', '_acos', '_ceil', '_collecti
ons', '_cos', '_e', '_exp', '_hexlify', '_inst', '_log', '_pi', '_random', '_sin
', '_sqrt', '_test', '_test_generator', '_urandom', '_warn', 'betavariate', 'cho
ice', 'division', 'expovariate', 'gammavariate', 'gauss', 'getrandbits', 'getsta
te', 'lognormvariate', 'normalvariate', 'paretovariate', 'randint', 'random', 'r
andrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangular', 'uniform', 'vo
nmisesvariate', 'weibullvariate']
>>> |
```

→ Les conteneurs

Un conteneur est type d'objets contenant d'autres objets :

- conteneurs de séquences
- conteneurs associatifs

- **Les conteneurs de séquence :**
offrent un accès séquentiel ou aléatoire à leurs éléments.
- **Les conteneurs associatifs :**
offrent un accès optimisé à leurs éléments via une valeur de **clé**.

Les types de séquences :

- les listes
- les n-uplets (tuples)
- les chaînes de caractères.

Les types associatifs :

- les dictionnaires
- Les ensembles

1. Les listes :

- Une liste est une structure de données très flexible, ordonnée et modifiable.
- Elle peut contenir des valeurs de types différents.

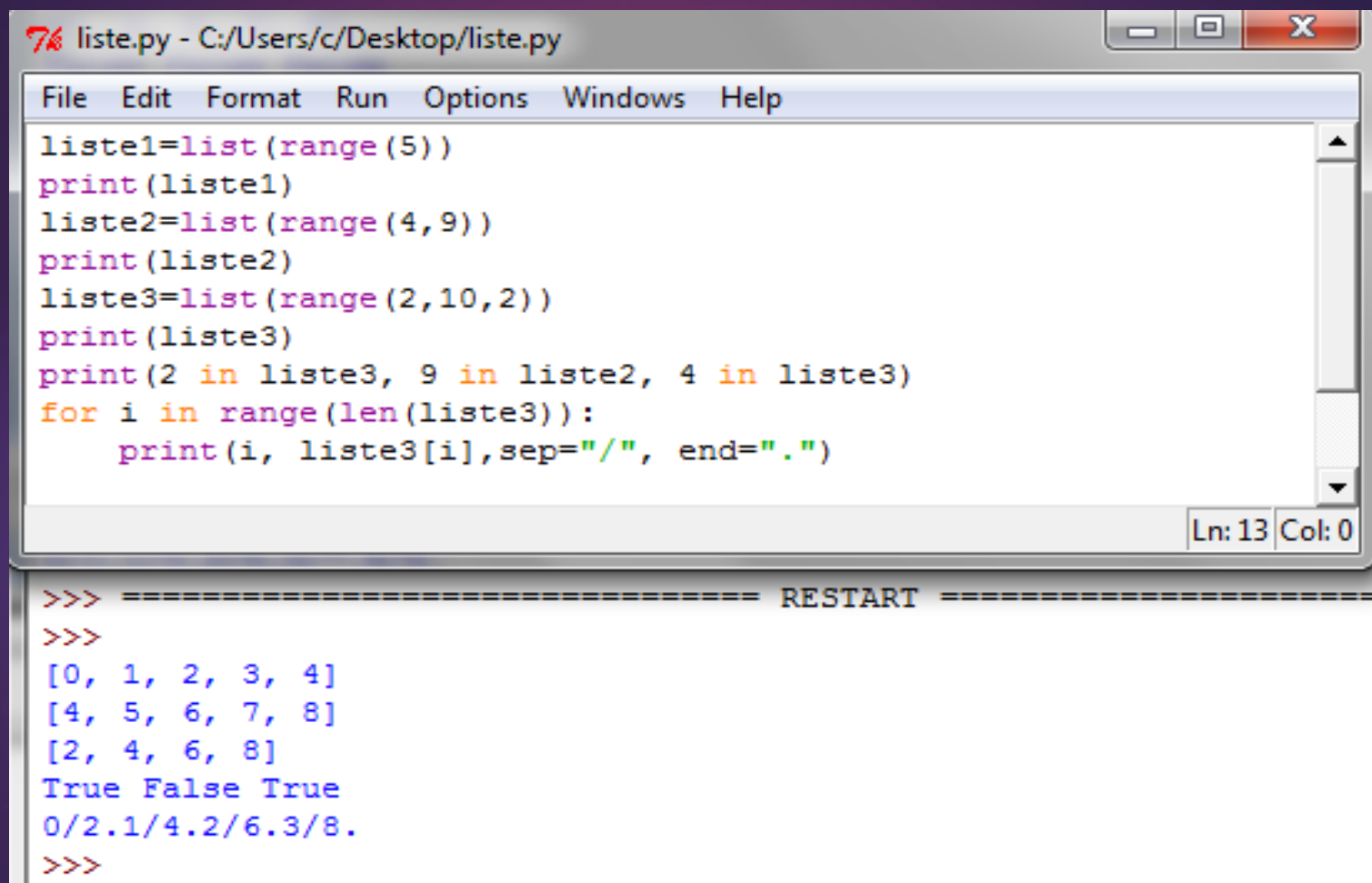
Exemple :

```
Python 3.1.2 (r312:79149, Mar 21 2010, 00:41:52) [MSC v.1500 32 bit  
win32  
Type "copyright", "credits" or "license()" for more information.  
>>> transports=['voiture','bus','train','avion', 'bateau']  
>>> fréquences=[10, 1000,2000,300, 200]
```

Exemples d'initialisation :

```
>>> couleurs=['rouge','vert','noir','blanc','jaune']
>>> colors=['red','green','black','white','yellow']
>>> couleurs[1]
'vert'
>>> colors[1:3]
['green', 'black']
>>> colors[2:]
['black', 'white', 'yellow']
>>> Mixte=[couleurs, colors]
>>> print(Mixte)
[['rouge', 'vert', 'noir', 'blanc', 'jaune'], ['red', 'green', 'black', 'white', 'yellow']]
>>> liste_vide, liste_repet=[], [1.5]*4
>>> liste_vide
[]
>>> liste_repet
[1.5, 1.5, 1.5, 1.5]
>>> |
```

Autres exemples :



```
liste.py - C:/Users/c/Desktop/liste.py
File Edit Format Run Options Windows Help
liste1=list(range(5))
print(liste1)
liste2=list(range(4,9))
print(liste2)
liste3=list(range(2,10,2))
print(liste3)
print(2 in liste3, 9 in liste2, 4 in liste3)
for i in range(len(liste3)):
    print(i, liste3[i],sep="/", end=".")
Ln: 13 Col: 0

>>> ===== RESTART =====
>>>
[0, 1, 2, 3, 4]
[4, 5, 6, 7, 8]
[2, 4, 6, 8]
True False True
0/2.1/4.2/6.3/8.
>>>
```

Initialisation compactée d'une liste

→ en intension :

```
L1=[i+1 for i in range(1,10)]
print("L1=", L1)
L2=[]
for i in range(1,10) :
    L2.append(i+1)
print("L2=", L2)
L3=[i+1 for i in range(1,20) if i>8]
print("L3=",L3)

liste1=list(range(5))
print(liste1)
liste2=list(range(4,9))
print(liste2)
liste3=list(range(2,10,2))
print(liste3)
print(2 in liste3, 9 in liste2, 4 in liste3)
for i in range(len(liste3)):
    print(i, liste3[i],sep="/", end=".")
```


Initialisation compactée d'une liste

→ en intension, résultat :

```
>>>
L1= [2, 3, 4, 5, 6, 7, 8, 9, 10]
L2= [2, 3, 4, 5, 6, 7, 8, 9, 10]
L3= [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[0, 1, 2, 3, 4]
[4, 5, 6, 7, 8]
[2, 4, 6, 8]
True False True
0/2.1/4.2/6.3/8.
>>> |
```

➤ Quelques méthodes :

L.sort(): tri de la liste L (croissant)

L.append(val): ajout de val dans L

L.reverse(): inverse L

L.remove(val) : supprime val de L

L.pop() : supprime le dernier élément de L

L.count(val) : nombre d'occurrences de val dans L.

L.index(val): renvoie l'indice de val dans L

➤ Autre bout de code (TP) :

```
print(mes_nombres.index(30))
mes_nombres[0]=77
print("ajout de 77 :", mes_nombres)
mes_nombres[1:2]=[11,22]
print("ajout de 2 valeurs", mes_nombres)
print("taille liste", len(mes_nombres))
print(mes_nombres.pop())
print(mes_nombres)
print(mes_nombres.count(80))
mes_nombres.extend([100,200])
print(mes_nombres)
print(len(mes_nombres))
```

➤ Résultat global :

68

```
>>>
liste triée [10, 20, 30, 40, 50, 60, 70, 80, 90]
liste inversée [80, 90, 80, 70, 60, 50, 40, 30, 20, 10]
[80, 90, 80, 70, 60, 50, 40, 30, 10]
7
ajout de 77 : [77, 90, 80, 70, 60, 50, 40, 30, 10]
ajout de 2 valeurs [77, 11, 22, 80, 70, 60, 50, 40, 30, 10]
taille liste 10
10
[77, 11, 22, 80, 70, 60, 50, 40, 30]
1
[77, 11, 22, 80, 70, 60, 50, 40, 30, 100, 200]
11
>>>
```

➔ Les types tableaux :

- Le tableau associatif est un type de données permettant de stocker des couples (**clé : valeur**) dont l'accès se fait par la clé.
- La clé n'apparaît qu'une fois dans le tableau.
- La clé peut être définie par l'opérateur **in**
- Le nombre de couples est calculé par la fonction **len()** et visualisés par **item()**.
- Les clés sont visualisées par **keys()** et les valeurs par **values()**.

2 . Les dictionnaires (dict) :

- Un dictionnaire est une collection de couples (**clé : valeur**) qui n'occupent pas un ordre immuable.
- Comme les listes, les dictionnaires sont modifiables.
 - Une clé peut être alphabétique, numérique ou tout type hashable (condensat).

Exemples :

```
>>> dico1={}
>>> dico1["nom"]="sam"
>>> dico1["taille"]=120
>>> print(dico1)
{'nom': 'sam', 'taille': 120}
```

```
>>> dico2={"nom":"charles","taille":170}
>>> print(dico2)
{'nom': 'charles', 'taille': 170}
```

```
>>> dico3={a:a**2 for a in (2,4,6,8)}
>>> print(dico3)
{8: 64, 2: 4, 4: 16, 6: 36}
>>> dico4=dict([("nom","pierre"),("taille",180)])
>>> print(dico4)
{'nom': 'pierre', 'taille': 180}
>>> |
```

```
>>> dico4["diego"]=150
>>> print(dico4)
{'nom': 'pierre', 'diego': 150, 'taille': 180}
>>> print(list(dico4.keys()))
['nom', 'diego', 'taille']
>>> print(sorted(dico4.keys()))
['diego', 'nom', 'taille']
```

Partie 2 :

2.1 Les fichiers

2.2 programmation orientée
objets

2.3 Intégration d'une base
de données