Introduction au langage Python

Table des matières

Introduction....3
Distribution python utilisée dans ce document....3
Documentation python / liens utiles....3
Structure du langage python....4
Les noms de variable....4
Mots clés réservés du langage:....4
Les commentaires....4
Indentation....5
Instructions multi-ligne....5
Plusieurs instructions par ligne....5
Les types simples....6
Les types numériques....6
Les entiers (int)....6
Les nombres à virgule flottante (float)....6
Les nombres complexes (complex)....6
Les fractions....7
Conversion de type....7
Chaînes de caractères (string)....7
Extraire une partie d'un texte....7
Mise en forme de chaînes de caractères....9
Les opérateurs....10
Les opérateurs arithmétiques....10
Les opérateurs sur les chaines de caractère....10
Opérateurs logiques....10
Opérateurs de comparaison....11
Précédence des opérateurs....11
La bibliothèque standard et ses modules....14
Utilisation des modules....14
Connaitre le contenu d'un module et consulter l'aide....14
exemple d'utilisation du module math....16
Quelques modules....16
Conteneurs: listes, tuples et dictionnaires....18
Listes....18
Tuples....20
Dictionnaires....21
Structures de contrôle: conditions et boucles....22

Branchements: if, elif, else....22
Boucles while....22
Boucles for....22
EXERCICE : Compter le nombre d'occurences de chaque charactère dans la chaîne de caractères "HelLo WorLd!!" On renverra un dictionaire qui à la lettre associe son nombre d'occurences....24
EXERCICE : Message codé par inversion de lettres....24
création de liste avec for :....24
Les fonctions....26
Arguments par défault....27
Classes....27
Exemple....28
Remarques....28
Exceptions....28

Introduction

Distribution python utilisée dans ce document

La distribution python utilisée dans ce document: https://www.anaconda.com/products/individual

Il existe de nombreuses autres distributions python libres:
https://wiki.python.org/moin/PythonDistributions

Documentation python / liens utiles

Page Python officielle: http://www.python.org

The Python Language Reference: https://docs.python.org/3.8/reference/index.html

The Python Standard Library: https://docs.python.org/3.8/library/index.html

Recommandations de style d'écriture: http://www.python.org/dev/peps/pep-0008

Un livre gratuit sur Python (en anglais): http://www.greenteapress.com/thinkpython/

Structure du langage python

Les noms de variable

- commencent toujours par une lettre

- peuvent contenir a-z, A-Z, 0-9 et quelques caractères spéciaux tels que _

- Par convention les noms de variables sont en minuscule.

- les mots clés du langage ne sont pas autorisés

```
ma_variable = 10
print( ma_variable )
del ma_variable # supprime ma_variable
print(ma_variable) # erreur car ma_variable n'existe plus
```

```
10
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-111-561080332388> in <module>
      2 print( ma_variable )
      3 del ma_variable # supprime ma_variable
----> 4 print(ma_variable) # erreur car ma_variable n'existe plus

NameError: name 'ma_variable' is not defined
```

Mots clés réservés du langage:

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

Les commentaires

```
'''
Parametres par defaut de l'imprimante 3D
configuration specifique dans config.json
'''
# Decalage de la table
offset_x = 12 # decalage x standard
offset_y =  8 # decalage y standard
```

Indentation

Un bloc de code est indenté du même nombre d'espaces (ou tabulation). En Python **l'indentation est obligatoire** car elle influence l'exécution du code.

```
if True:
   print("Vrai")
else:
   print("Faux")
```

Instructions multi-ligne

Une instruction peut se continuer sur plusieurs lignes en utilisant le '\'

```
calcul = 1 * 2 + \
         3 * 4 + \
         5 * 6
print( calcul )
```

```
44
```

Les blocs [], {}, et () peuvent continuer sur plusieurs lignes:

```
jours = [
    'lundi',
    'mardi',
    'mercredi',
    'jeudi',
    'vendredi'
    ]
print(jours)
```

```
['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']
```

Plusieurs instructions par ligne

```
 a = b = 1; print(a, b) # A éviter
```

Les types simples

Les types numériques

Les entiers (int)

```
# nombre entier
a = 4
print(a)
print(type(a))
```

```
4
<class 'int'>
```

```
a = 8      # decimal
b = 0b1100 # binaire
c = 0o777  # octal
d = 0xCAFE # hexadecimal
print(a)
print(b)
print(c)
print(d)
```

```
 8
 12
 511
 51966
```

Les nombres à virgule flottante (float)

```
c = 2.1 # nombre à virgule flottante
print(type(c))
```

```
 <class 'float'>
```

```
vitesse_lumiere = 3e8
print(vitesse_lumiere," m/s")
```

```
 300000000.0  m/s
```

Les nombres complexes (complex)

```
a = 1.5 + 1j # nombre complexe
print( a. real )
print( a. imag )
print(a)
print(a + 1j)
print(type(a))
```

```
 1.5
 1.0
 (1.5+1j)
 (1.5+2j)
 <class 'complex'>
```

Les fractions

```
import fractions
a = fractions.Fraction(2, 3)
b = fractions.Fraction(1, 2)
print(a + b)
print( isinstance(a, fractions.Fraction))
```

```
7/6
True
```

Conversion de type

```
x = 1.5
print(x, type(x))
x = int(x)
print(x, type(x))
z = complex(x)
print(z, type(z))
```

```
1.5 <class 'float'>
1 <class 'int'>
(1+0j) <class 'complex'>
```

Chaînes de caractères (string)

```
s = 'Hello world!' # ou avec " "
s = "Hello world!"
print(s)
print(type(s))
```

```
Hello world!
<class 'str'>
```

```
chaine1 = "aujourd'hui"
print(chaine1)
chaine2 = 'il a dit "Bonjour"'
print(chaine2)
chaine3 = 'il a dit "aujourd\'hui"'
print(chaine3)
```

Extraire une partie d'un texte

On peut extraire une sous-chaine avec la syntaxe [start:stop:step], qui extrait les caractères entre start et stop (**exclu**) :

Attention: les indices commencent à 0!

```
s[0] # premier élément
```

```
Out[59]: 'H'
```

```
s[-1] # dernier élément
```

```
Out[60]: '!'
```

```
s[1:5]
```

```
Out[61]: 'ello'
```

```
start, stop = 1, 5
print( s[ start:stop] )
```

```
print( len(s[ start:stop]) )
```
```
ello
4
```

On peut omettre start ou stop. Dans ce cas les valeurs par défaut sont respectivement 0 et la fin de la chaine.

```
s[:5] # 5 premières valeurs
```
```
'Hello'
```
```
s[6:] # de l'entrée d'indice 6 à la fin
```
```
'world!'
```

Il est aussi possible de définir le step (pas d'avancement) avec la syntaxe [start:stop:step] (la valeur par défaut de step est 1):

```
s[1 ::2]
```
```
'el ol!'
```
```
s[0 ::2]
```
```
'Hlowrd'
```

Cette technique est appelée *slicing*. Pour en savoir plus:

https://docs.python.org/3/library/functions.html?highlight=slice#slice

exemple : A partir des lettres de l'alphabet, générer par une opération de slicing la chaîne de charactère *cfilorux*

```
import string
alphabet = string.ascii_lowercase
result = alphabet[2::3]
print(result)
```

Mise en forme de chaînes de caractères

```
print("str1", "str2", "str3") # print ajoute des espaces entre les chaînes
print("str1", 1.0, False, - 1j) # print convertit toutes les variables en chaînes
```

```
str1 str2 str3
str1 1.0 False (-0-1j)
```

```
a = 1.0000000002
print("val = %e " % a)     # comme en C (cf. printf)
print("val = %1.15f " % a) # comme en C (cf. printf)
print("val = % 3d " % 10)  # comme en C (cf. printf)
print("val = %s " % a)     # comme en C (cf. printf)
print(str(a))
print("val = " + str(a))
```

```
val = 1.000000e+00
val = 1.000000000200000
val =  10
val = 1.0000000002
1.0000000002
val = 1.0000000002
```

```
# Plus avancé
s = "val1 = %.2f , val2 = %d " % (3.1415, 1.5)
print(s)
s = "Le nombre %s est égal à %s "
print((s % ("pi", math.pi)))
print((s % ("e", math.exp(1.))))

b=12
print("val = %02x en hexadecimal" % b)
```

```
val1 = 3.14 , val2 = 1
Le nombre pi est égal à 3.141592653589793
Le nombre e est égal à 2.718281828459045
val = 0c en hexadecimal
```

Les opérateurs

Les opérateurs arithmétiques

```
1 + 2, 1 - 2, 1 * 2, 1 / 2 # + - / * sur des entiers
1.0 + 2.0, 1.0 - 2.0, 1.0 * 2.0, 1.0 / 2.0 # + - / * sur des flottants
3.0 // 2. # Division entière
2 ** 10 # exposant. attention pas ^
```

```
1024
```

```
5 / 3 # division
```

```
1.6666666666666665
```

```
5 // 3 # division entière
```

```
1
```

```
5 % 3 # reste de la division (modulo)
```

```
2
```

```
a = 2
3 / float(a) # OK
print(7 * 3.0) # int x float -> float
print(type(7 * 3.0))
```

```
1.5
21.0
<class 'float'>
```

exemple:

```
ht = 15.5
tva = 20
ttc = ht * (1 + tva / 100 )
ttc = round( ttc, 2 )
print(ttc)
```

```
18.6
```

Les opérateurs sur les chaines de caractère

```
s = "abc" + "def" + "ghi"
print( s )
s = "Ah" * 3
print( s )
```

```
abcdefghi
AhAhAh
```

Opérateurs logiques

Opérations booléennes en anglais and, not, or.

```
True and False
```

False

```
not False
```

True

```
to_be = True
print( to_be or not to_be )
```

True

```
int(True)
```

```
1
```

```
int(False)
```

```
0
```

```
1 == True
```

```
True
```

Opérateurs de comparaison

Comparaisons >, <, >= (plus grand ou égal), <= (inférieur ou égal), == equalité, is identique.

```
3 < 4 # bool
```

```
True
```

```
1 < 3 < 5
```

```
True
```

```
3 < 2
```

```
False
```

```
test = (3 > 4)
print(test)
print(type(test))
```

```
False
<class 'bool'>
```

```
2 > 1, 2 < 1
```

```
(True, False)
```

```
2 > 2, 2 <= 2
```

```
(False, True)
```

```
2 != 3
```

```
True
```

```
not 2 == 3
```

```
True
```

Précédence des opérateurs

The following table summarizes the operator precedence in Python, from lowest precedence (least binding) to highest precedence (most binding). Operators in the same box have the same precedence. Unless the syntax is explicitly given, operators are binary. Operators in the same box group left to right (except for exponentiation, which groups from right to left).

Note that comparisons, membership tests, and identity tests, all have the same precedence and have a left-to-right chaining feature as described in the Comparisons section.

Operator	Description
`:=`	Assignment expression
`lambda`	Lambda expression
`if` – `else`	Conditional expression
`or`	Boolean OR
`and`	Boolean AND
`not x`	Boolean NOT
`in`, `not in`, `is`, `is not`, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
`|`	Bitwise OR
`^`	Bitwise XOR
`&`	Bitwise AND
<<, >>	Shifts
+, -	Addition and subtraction
*, @, /, //, %	Multiplication, matrix multiplication, division, floor division, remainder [5]
+x, -x, ~x	Positive, negative, bitwise NOT
**	Exponentiation [6]
`await x`	Await expression
`x[index]`, `x[index:index]`, `x(arguments...)`, `x.attribute`	Subscription, slicing, call, attribute reference
`(expressions...)`,	Binding or parenthesized expression, list display, dictionary display, set display

Operator	Description
`[expressions...]`, `{key: value...}`, `{expressions...}`	

La bibliothèque standard et ses modules

Les fonctions Python sont organisées par modules.

Bibliothèque standard Python (Python Standard Library) : collection de modules donnant accès à des fonctionnalités de bases : appels au système d'exploitation, gestion des fichiers, gestion des chaînes de caractères, interface réseau, etc.

Utilisation des modules

Un module doit être importé avant de pouvoir être utilisé, exemple :

```
import math
x = math.cos(2 * math.pi)
print(x)
```

```
1.0
```

Ou bien en important uniquement les fonctions dont on a besoin:

```
from math import cos, pi
x = cos(2 * pi)
print( x)
```

```
1.0
```

```
import math as m
print(m. cos(1.))
```

```
0.5403023058681397
```

Connaitre le contenu d'un module et consulter l'aide

Une fois un module importé on peut lister les symboles disponibles avec la fonction dir:

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

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb',
'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma',
'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt',
'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm',
'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'tau', 'trunc']
```

```
help(math.ceil)
```

```
Help on built-in function ceil in module math:

ceil(x, /)
    Return the ceiling of x as an Integral.
    This is the smallest integer >= x.
```

```
help(math)
```

```
Help on built-in module math:
```

```
NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

...

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.


    floor(x, /)
        Return the floor of x as an Integral.

        This is the largest integer <= x.


    log(...)
        log(x, [base=math.e])
        Return the logarithm of x to the given base.

        If the base not specified, returns the natural logarithm (base e) of
x.

    log10(x, /)
        Return the base 10 logarithm of x.


    pow(x, y, /)
        Return x**y (x to the power of y).


    sqrt(x, /)
        Return the square root of x.


    trunc(x, /)
        Truncates the Real x to the nearest Integral toward 0.
```

```
        Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    (built-in)
```

Exemple d'utilisation du module math

```
import math

# Calcul de la valeur exacte du nombre d'or (phi)
phi = ( 1 + math.sqrt(5) ) / 2
print(phi)
print(math.pow(phi,2)) # nombre d'or au carré
```

```
1.618033988749895
2.618033988749895
```

Quelques modules

- string — Common string operations
- datetime — Basic date and time types
- array — Efficient arrays of numeric values
- pprint — Data pretty printer
- math — Mathematical functions
- fractions — Rational numbers
- random — Generate pseudo-random numbers
- statistics — Mathematical statistics functions
- pathlib — Object-oriented filesystem paths
- os.path — Common pathname manipulations
- fileinput — Iterate over lines from multiple input streams
- stat — Interpreting stat() results
- filecmp — File and Directory Comparisons
- tempfile — Generate temporary files and directories
- glob — Unix style pathname pattern expansion
- fnmatch — Unix filename pattern matching
- shutil — High-level file operations
- sqlite3 — DB-API 2.0 interface for SQLite databases
- zlib — Compression compatible with **gzip**

- gzip — Support for **gzip** files
- bz2 — Support for **bzip2** compression
- zipfile — Work with ZIP archives
- tarfile — Read and write tar archive files
- csv — CSV File Reading and Writing
- configparser — Configuration file parser
- os — Miscellaneous operating system interfaces
- io — Core tools for working with streams
- time — Time access and conversions
- getopt — C-style parser for command line options
- syslog — Unix syslog library routines
- socket — Low-level networking interface
- json — JSON encoder and decoder
- html.parser — Simple HTML and XHTML parser
- webbrowser — Convenient Web-browser controller
- http — HTTP modules
- tkinter — Python interface to Tcl/Tk

Conteneurs: listes, tuples et dictionnaires

Listes

Les listes sont très similaires aux chaînes de caractères sauf que les éléments peuvent être de n'importe quel type.

La syntaxe pour créer des listes est […]:

```
l = [1, 2, 3, 4]
print(type(l))
print(l)
```

```
<class 'list'>
[1, 2, 3, 4]
```

Exemples de slicing:

```
print(l[1:3])
print(l [::2])
```

```
[2, 3]
[1, 3]
```

Attention: On commence à indexer à 0!

```
print( l[0] )
print( l[-1:0:-1] )
```

```
1
[4, 3, 2]
```

On peut mélanger les types:

```
l = [1, 'a', 1.0, 1-1j]
print(l)
```

```
[1, 'a', 1.0, (1-1j)]
```

On peut faire des listes de listes (par exemple pour décrire un arbre…)

```
list_of_list = [1, [2, [3, [4, [5]]]]]
print(list_of_list)
```

```
[1, [2, [3, [4, [5]]]]]
```

La fonction range pour générer une liste d'entiers:

```
start, stop, step = 10, 30, 2
print(list(range(start, stop, step )))
```

```
[10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

Intération de n-1 à 0

```
n = 10
print(list(range(n-1, -1, -1)))
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

```
# convertir une chaine de caractère en liste
s = "zabcda"
l2 = list(s)
print(l2)
```

```
['z', 'a', 'b', 'c', 'd', 'a']
```

```
# tri
l3 = list(l2)
l2.sort()
print(l2)
print(l3)
print(l2[::-1 ])
```

```
['a', 'a', 'b', 'c', 'd', 'z']
['z', 'a', 'b', 'c', 'd', 'a']
['z', 'd', 'c', 'b', 'a', 'a']
```

Attention l2.sort() ne renvoie rien c'est-à-dire None

```
out = l2.sort()
print (out)
```

```
None
```

Ajout, insertion, modifier, et enlever des éléments d'une liste:

```
# création d'une liste vide
l = [] # ou l = list()
# ajout d'éléments avec `append`
m = l.append("A")
l. append("d")
l. append("d")
print(m)
print(l)
```

```
None
['A', 'd', 'd']
```

Concatenation de listes avec +

```
l. index('A')
```

```
0
```

```
lll = [1, 2, 3]
mmm = [4, 5, 6]
print (lll + mmm) # attention différent de lll.append(mmm)
```

```
[1, 2, 3, 4, 5, 6]
```

```
print (lll * 2)
```

```
[1, 2, 3, 1, 2, 3]
```

On peut modifier une liste par assignation:

```
l[1] = "p"
l[2] = "p"
print (l)
```

```
['A', 'p', 'p']
```

```
l[1:3] = ["d", "d"]
print (l)
```

```
['A', 'd', 'd']
```

Insertion à un index donné avec insert

```
l. insert(0, "i")
l. insert(1, "n")
l. insert(2, "s")
l. insert(3, "e")
l. insert(4, "r")
l. insert(5, "t")
print(l)
```

```
['i', 'n', 's', 'e', 'r', 't', 'A', 'd', 'd']
```

Suppression d'un élément avec 'remove'

```
l. remove("A")
print (l)
```

```
['i', 'n', 's', 'e', 'r', 't', 'd', 'd']
```

```
ll = [1, 2, 3, 2]
print (ll)
ll.remove(2)
```

```
print (ll)
```
```
[1, 2, 3, 2]
[1, 3, 2]
```
```
print(ll)
print (2 in ll)
print (ll.index(2))
```
```
[1, 3, 2]
True
2
```

Suppression d'un élément à une position donnée avec del:

```
print(l)
del l[7]
del l[0]
print(l)
```
```
['i', 'n', 's', 'e', 'r', 't']
```

help(list) pour en savoir plus.

Tuples

Les tuples (n-uplets) ressemblent aux listes mais ils sont immuables : ils ne peuvent pas être modifiés une fois créés. On les crée avec la syntaxe (..., ..., ...) ou simplement ..., ...:

```
# un tuple est immutable
point = (10, 20)
print (point, type(point))
print(point[0],point[1])

# une liste est mutable
p2 = list(point)
p2[1] = 5
print (p2, type(p2))
```
```
(10, 20) <class 'tuple'>
10 20
[10, 5] <class 'list'>
```

Un *tuple* peut être dépilé par assignation à une liste de variables séparées par des virgules :

```
x, y = point
print ("x =", x)
print ("y =", y)
```
```
x = 10
y = 20
```

On ne peut pas faire :

```
point[0] = 20
```
```
---------------------------------------------------------------------
-------
TypeError Traceback (most recent cal
l last)
<ipython-input-112-ac1c641a5dca> in <module>()
----> 1 point[0] = 20
TypeError: 'tuple' object does not support item assignment
```

Dictionnaires

Ils servent à stocker des données de la forme *clé-valeur.*

La syntaxe pour les dictionnaires est {key1 : value1, …}:

```
dimensions1 = {
    "longueur" : 0.5,
    "largeur" : 0.4,
    "epaisseur" : 0.2
    }
print( type(dimensions1) )
print( dimensions1 )
print ("longueur =", dimensions1["longueur"])
print ("largeur =", dimensions1["largeur"])
print ("epaisseur =", dimensions1["epaisseur"])
volume = dimensions1["longueur"] * dimensions1["largeur"] *
dimensions1["epaisseur"]
print( "Volume = %5.3f" % volume )
```

```
<class 'dict'>
{'longueur': 0.5, 'largeur': 0.4, 'epaisseur': 0.2}
longueur = 0.5
largeur = 0.4
epaisseur = 0.2
Volume = 0.040
```

```
# Autre syntaxe
dimensions2 = dict(longueur=0.7, largeur=0.5, epaisseur=0.3)
dimensions2["masse_volumique"] = 7860 # masse volumique du fer
print( dimensions2 )
masse = \
    dimensions2["longueur"] * \
    dimensions2["largeur"] * \
    dimensions2["epaisseur"] * \
    dimensions2["masse_volumique"]
print( "masse = {} kg".format(masse) )

# teste si la clé “masse_volumique” existe dans dimensions2
print("masse_volumique" in dimensions2)
del dimensions2["masse_volumique"]
print("masse_volumique" in dimensions2)
print(dimensions2)
```

```
{'longueur': 0.7, 'largeur': 0.5, 'epaisseur': 0.3, 'masse_volumique': 7860}
masse = 825.3 kg
True
False
{'longueur': 0.7, 'largeur': 0.5, 'epaisseur': 0.3}
```

```
notes = {"pierre": 15, "paul":8, "jacques":12}
print (notes)
total = notes['pierre'] + notes['paul'] + notes['jacques']
moyenne = total / len(notes)
print("moyenne = %5.2f" % moyenne)
```

```
{'pierre': 15, 'paul': 8, 'jacques': 12}
moyenne = 11.67
```

```
traduction = {
    "fr": "Bonjour",
    "gb": "Hello",
    "it": "Ciao",
}
langue = input( "langue : ")
if langue in traduction:
    print( traduction[langue] )
else :
    print( "erreur, langue inconnue")
```

```
langue : fr
Bonjour
```

Structures de contrôle: conditions et boucles

Branchements: if, elif, else

if : si / elif : sinon si, contraction de “else if” / else : sinon

```
a = 3
b = 2
if a < b:
    print("a est strictement inférieur à b")
elif a > b:
    print("b est strictement inférieur à a")
else :
    print("b est égal à a")
```

```
b est strictement inférieur à a
```

```
temperature = 25

eau_etat_solide = ( temperature < 0 )
eau_etat_gazeux = ( temperature >= 100 )

if eau_etat_solide is True:
    print( "à {} °C, l'eau est en état solide".format( temperature ) )
elif not eau_etat_gazeux:
    print( "à {} °C, l'eau est en état liquide".format( temperature ) )
else :
    print( "à {} °C, l'eau est en état gazeux".format( temperature ) )
```

```
à 25 °C, l'eau est en état liquide
```

Boucles while

```
i = 0
while i < 5:
    print(i)
    i = i + 1
print("Terminé")
```

```
0
1
2
3
4
Terminé
```

Boucles for

Itérations sur une liste

```
for x in [1, 2, 3]:
```

```
    print(x)
```
```
1
2
3
```

```
for personnage in ['riri', 'fifi', 'loulou']:
    print(personnage)
```
```
riri
fifi
loulou
```

création de liste avec for

```
liste1 = [x ** 2 for x in range(0,5)]
print (liste1)

# est la version courte de :
liste2 = list()
for x in range(0, 5):
    liste2.append(x ** 2)
print(liste2)
```
```
[0, 1, 4, 9, 16]
[0, 1, 4, 9, 16]
```

itérations sur une séquence

```
# début 0 par défaut, 4 est exclu
for x in range(4):
    print(x)
```
```
0
1
2
3
```

```
# de -3 à 3 exclu
for x in range(-3,3):
    print(x)
```
```
-3
-2
-1
0
1
2
```

```
# de 0 à 8 par pas de 2
for x in range(0,10,2):
    print(x)
```
```
0
2
4
6
8
```

Parfois il est utile d'accéder à la valeur et à l'index de l'élément. Il faut alors utiliser enumerate:

```
# Pour acceder à l'index et à la valeur en même temps
for index, valeur in enumerate(range(-3,3)):
    print(index, valeur)
```
```
0 -3
```

```
1 -2
2 -1
3 0
4 1
5 2
```

itérations sur une chaine de caractères

```
for lettre in "Python":
    print(lettre)
```

```
P
y
t
h
o
n
```

Itérations sur un dictionnaire

```
message = {
    "solide": "l'eau est en état solide",
    "liquide": "l'eau est en état liquide",
    "gazeux": "l'eau est en état gazeux",
}

print("Méthode 1")
for key in message:
    print ( key, " = ", message[key] )

print("Méthode 2")
for key, value in message.items():
    print (key, " = ", value)
```

```
Méthode 1
solide  =  l'eau est en état solide
liquide  =  l'eau est en état liquide
gazeux  =  l'eau est en état gazeux
Méthode 2
solide  =  l'eau est en état solide
liquide  =  l'eau est en état liquide
gazeux  =  l'eau est en état gazeux
```

Exercice : compter le nombre d'occurences de chaque charactère dans la chaîne de caractères "HelLo WorLd!!" On renverra un dictionaire qui à la lettre associe son nombre d'occurences.

```
s = "HelLo WorLd!!"
```

Exercice : message codé par inversion de lettres

Ecrire un programme de codage et de décodage

```
def codage( phrase ):
    codage = ""

    for lettre in phrase:
        if lettre in code:
```

```
            codage += code[lettre]
        else:
            codage += lettre

    return(codage)

def decodage( phrase ):
    decodage = ""

    for lettre in phrase:
        trouve = False
        for key in code:
            if trouve == False and lettre == code[key]:
                decodage += key
                trouve = True

        if not trouve:
            decodage += lettre

    return(decodage)


code = {'e':'a', 'l':'m', 'o':'e'}
s = 'Hello world!'

s1 = codage(s)
print( "Codage de \"{}\" => \"{}\" ".format(s, s1) )

s2 = decodage(s1)
print( "Décodage de \"{}\" => \"{}\" ".format(s1, s2) )
```

```
Codage de "Hello world!" => "Hamme wermd!"
Décodage de "Hamme wermd!" => "Hello world!"
```

Les fonctions

Une fonction en Python est définie avec le mot clé def, suivi par le nom de la fonction, la signature entre parenthèses (), et un :.

Exemples:

```
# définition de la fonction
def test():
    print("fonction test")

print("programme principal")
test() # Appel effectif
print("programme principal")
```

```
programme principal
fonction test
programme principal
```

Documenter une fonction (docstring):

```
def func1(s):
    """
    Affichage d'une chaine et de sa longueur
    """
    print(s, "est de longueur", len(s))
    return

help(func1)
func1("longueur")
```

```
Help on function func1 in module __main__:

func1(s)
    Affichage d'une chaine et de sa longueur

longueur est de longueur 8
```

Retourner une valeur avec return:

```
def square(x):
    """
    Returne le carré de x.
    """
    return(x ** 2)

print(square(4))
```

```
16
```

Retourner plusieurs valeurs:

```
def powers(x):
    """
    Retourne les premières puissances de x.
    """
    return( x ** 2, x ** 3, x ** 4 )

print( powers(3) )
print( type(powers(3)) )

x2, x3, x4 = powers(3)
print("x2 =",x2, "x3 =",x3, "x4 =",x4)
```

```
resultats = powers(3)
for valeur in resultats:
    print(valeur)
```

```
(9, 27, 81)
<class 'tuple'>
x2 = 9 x3 = 27 x4 = 81
9
27
81
```

Arguments par défault

Il est possible de fournir des valeurs par défault aux paramètres:

```
def puissance(x, p=2, debug=False):
    """
    Calcule le resultat de x à la puissance p
    parametres:
    x opérande
    p exposant (valeur par défaut 2)
    debug affichage des paramétres (valeur par défaut False)
    """
    if debug:
        print ("debug: fonction puissance avec x =", x, "et l'exposant p =", p)

    return( x ** p )

help(puissance)
print( puissance(3,3) )
print( puissance(3) )
print( puissance(x=5, debug=True))
```

```
Help on function puissance in module __main__:

puissance(x, p=2, debug=False)
    Calcule le resultat de x à la puissance p
    x opérande
    p exposant (valeur par défaut 2)
    debug affichage des paramétres (valeur par défaut False)

27
9
debug: fonction puissance avec x = 5 et l'exposant p = 2
25
```

Classes

Les classes sont les éléments centraux de la programmation orientée objet

Classe: structure qui sert à représenter un objet et l'ensemble des opérations qui peuvent êtres effectuées sur ce dernier.

Dans Python une classe contient des *attributs* (variables) et des *méthodes* (fonctions). Elle est définie de manière analogue aux fonctions mais en utilisant le mot clé class. La définition d'une classe contient généralement un certain nombre de méthodes de classe (des fonctions dans la classe).

Le premier argument d'un méthode doit être self: argument obligatoire. Cet objet self est une auto-référence.

Certains noms de méthodes ont un sens particulier, par exemple :

__init__: nom de la méthode invoquée à la création de l'objet

__str__ : méthode invoquée lorsque une représentation de la classe sous forme de chaîne de caractères est demandée, par exemple quand la classe est passée à print

Exemple

```
class Point (object):
    """
    Classe pour représenter un point dans le plan.
    """
    def __init__(self, x, y):
        """
        Creation d'un nouveau point en position x, y.
        """
        self.x = x
        self.y = y
    def translation(self, dx, dy):
        """
        Translation du point de dx et dy.
        """
        self.x += dx
        self.y += dy
    def __str__(self):
        return "Point: [ %f , %f ]" % (self.x, self.y)

p1 = Point(x=0, y=0) # appel à __init__
print( type(p1) )
print( p1.x )
print( p1.y )
print( "%s" % p1 ) # appel à la méthode __str__
p1.translation( dx=1, dy=1 )
print( p1 )
```

Pour invoquer une méthode de la classe sur une instance p de celle-ci:

```
In [163]: p2 = Point(1, 1)
p1.translate(0.25, 1.5)
print (p1)
print (p2)
Point: [1.250000, 2.500000]
Point: [1.000000, 1.000000]
```

Remarques

L'appel d'une méthode de classe peut modifier l'état d'une instance particulière. Cela n'affecte ni les autres instances ni les variables globales.

Exceptions

Dans Python les erreurs sont gérées à travers des "Exceptions". Une erreur provoque une Exception qui interrompt l'exécution normale du programme. L'exécution peut éventuellement

reprendre à l'intérieur d'un bloc de code try – except.

Une utilisation typique: arrêter l'exécution d'une fonction en cas d'erreur

```
def affiche_age(age):
    if age < 0:
        raise Exception("Erreur: l'age ne peut être négatif")
    # et on continue
    print( "Votre age est ", age )

affiche_age(5)
affiche_age(-5)
```

```
Votre age est  5
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-19-43e2b0d7bd68> in <module>
      6
      7 affiche_age(5)
----> 8 affiche_age(-5)

<ipython-input-19-43e2b0d7bd68> in affiche_age(age)
      1 def affiche_age(age):
      2     if age < 0:
----> 3         raise Exception("Erreur: l'age ne peut être négatif")
      4     # et on continue
      5     print( "Votre age est ", age )

Exception: Erreur: l'age ne peut être négatif
```

```
age = 999
while ( age != 0 ):
    saisie_utilisateur = input("Saisir votre age : ")
    age = int(saisie_utilisateur)
    affiche_age(age)
```

```
Saisir votre age : 42
Votre age est  42
Saisir votre age : -5
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-21-1cc0ceae1ffb> in <module>
      3     saisie_utilisateur = input("Saisir votre age : ")
      4     age = int(saisie_utilisateur)
----> 5     affiche_age(age)
      6

<ipython-input-19-43e2b0d7bd68> in affiche_age(age)
      1 def affiche_age(age):
      2     if age < 0:
----> 3         raise Exception("Erreur: l'age ne peut être négatif")
      4     # et on continue
      5     print( "Votre age est ", age )

Exception: Erreur: l'age ne peut être négatif
```

On utilise try et expect pour maîtriser les erreurs:

```
age = 999
while ( age != 0 ):
    saisie_utilisateur = input("Saisir votre age : ")
    age = int(saisie_utilisateur)
    try:
        affiche_age(age)
    except:
        print("Erreur: age incorrect.")
```

```
Saisir votre age : 42
Votre age est  42
Saisir votre age : -5
Erreur: age incorrect.
```

Pour obtenir de l'information sur l'erreur: accéder à l'instance de la classe Exception concernée:

```
try:
    affiche_age(-3)
except Exception as e:
    print ("Une exception à été générée:", e)
```

```
Une exception à été générée: Erreur: l'age ne peut être négatif
```