Aide Mémoire

Eléments de base

Symboles :
A..Z a..z — 0 1 2 3 4 5 6 7 8 9
Identificateur : nom d'objet,
[a–zA–Z_] suivi d'un ou de plusieurs *[a–zA–Z0–9_]* *(voir PEP 3131)*

Mots clés

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

Types de base

float : nombres réels,
0.0 -1.2 2e-9 (2×10^{-9}),
Exemple :
2e-9.as_integer_ratio()→num, dénom
int : entiers (base : 10, 2, 8, 16),
-15 0b11001 0o667 0x3ff
Exemple :
int(x[,base]), *int(7.21)*→7, hex(31) →0x1f
complex : nombres complexes,
-15j 0.0 1+4j,
Exemple :
complex(r[,img]), (1-9j).conjugate() →1+9j
bool : booléen→ *True/False*,
Exemple :
bool(x) == *False*, si x : None|0|séquences vides.
NoneType : `None` objet nul unique,
Exemple :
x=None ; *x* `is` None→ True

Affectation

opérateur d'affectation : =
x = y, se lit *x* prend pour valeur *y*.
Exemple :
x= -19 ; type(x)→*int*, x=3e-6 ; type(x)→*float*

Enoncés composés

succession : ';'
Exemple :
t = x ; x = y ; y = t
accès : '.'
module|classe|objet|...fonction|.attributs
Exemple :
math.cos(math.pi)→ -1.0

Commentaire/Documentation

Exemple :
 # le commentaire est sur une seule ligne.
""" la doc est sur une ou plusieurs lignes."""

Assertion

`assert` *prédicat* [, *expression*]
Exemple :
assert 3 == 4→AssertionError (rupture)
assert 3 == 4, 'Pb' →AssertionError : Pb

Enoncés conditionnels

if

```
if prédicat:
    <instructions>
```

Exemple:
```
if x < y:
    print('x < y')
```

if...else

```
if prédicat:
    <instructions>
[else:
    <instructions>]
```

Exemple:
```
if x < y:
    print('x < y')
else:
    print('x ≥ y')
```

if...elif

```
if prédicat:
    <instructions>
[ elif prédicat:
    <instructions>
...
else:
    <instructions>]
```

Exemple:
```
if x == 0:
    print('x = 0')
elif x > 0:
    print('x > 0')
else:
    print('x < 0')
```

Expression conditionnelle

expression `if` *prédicat* `else` *expression*
Exemple :
x = -1 ; y = -12 ; x `if` x < y `else` y →-12

Entrées/Sorties

Entrée :
input([expression])→ str
Sortie :
print([v0][,vi...][,sep=][,end=][,file=])
Exemple :
raw = input("Saisir : ")→ Saisir : []
print("val : ", 19, 0x3F)→val : 19 63

Intervalle

range([d,] f [,p])→plage de valeurs où d, f, p :entiers ,d = 0 et p=1 par défaut.
Exemple :
range(1,4)→ 1, 2, 3 *range(4,1,-1)→4, 3, 2*

Enumération

enumerate(seq[, deb])→itérable
Exemple :
list(enumerate('pyt')→[(0, 'p'),(1, 'y'),(2, 't')]

Affectation augmentée

Opérateurs	*Signification*
x += y	*plus : -=, *=, %=,.../=*
x=y=z	*x,y et z→même valeur*
x,y=1,'e'	*affectation simultanée*

Exemple :
x , y = 1 , 'e' ⟶ x vaut 1 et y vaut 'e'

Opérateurs booléen

False, None, 0, séquences vides :→False.

Opérateurs	*Signification*
x or y	*True si x ou y est Vrai*
x and y	*True si x et y sont Vrai*
not y	*True si y est Faux*

Exemple :
x = -2 ; y = 7, 0 and x or y →7

Enoncés répétitifs

while

```
while prédicat:
    <instructions>
```

Exemple:
```
x = 0;      s = 0
while x < 100:
    x = x + 1
    s = s + x
```
s → 4950 x → 100

while...break

```
while prédicat:
    <instructions>
[ break]
<instructions>
```

Exemple:
```
x = 0;      s = 0
while x < 100:
    if x > 81:
        break
    x = x + 1
    s = s + x
```
x → 82 s → 3321

while...continue

```
while prédicat:
    <instructions>
[ continue]
<instructions>
```

Exemple:
```
x = 0;      s = 0
while x < 100:
    if x < 21:
        x = x + 1
        continue
    x = x + 1
    s = s + x
    if x > 41:
        break
```
x → 42 s → 672

for

```
for var in itérable:
    <instructions>
```

Exemple:
```
s = 0
for x in range(10):
    s = s + x
```
x → 9 s → 45

Séquences

liste (séquence *mutable*) :
constructeurs : *list* (), [ ]
Exemple :
p = *list* () ; L = [ ] ; s = [1,-2,[ ],9]
s[i] $i^{ème}$ élément, $0 \leq i < n$ (n = taille de s)
s[0] → 1, s[3] → 9
ls = list(range(1,5)), ls → [1, 2, 3, 4]
chaîne (séquence *immuable*) :
constructeurs : *str* (), ""
Exemple :
p = *str* () ; L = "" ; s = "abDf"
s[0] → 'a', s[3] → 'f'
n-uplet (séquence *immuable*) :
constructeurs : *tuple* (), ()
Exemple :
p = *tuple* ()→(), L = ()→()
p = (5) ; type(p)→ <type 'int'>
L = (5,) ; type(L)→ <type 'tuple'>

Opérations sur Séquences

s, t des séquences ; *i,j,k,n* des entiers.

Opérateurs	*Signification*
t + s	*concaténation de t et s*
*s * n*	*concaténer n fois s*
x in t	*True si x∈t sinon Faux*
len(s)	*nbr d'éléments de s*

Exemple :
t = [1, 2, 3, 11, 5, 4, 9, 6, 7] ; -1 in t →False
s = "LaTex" ; s*3→"LaTexLaTexLaTex"
tu = tuple([1,-2,[ ],9]) ; tu→(1, -2, [], 9)

Fonctions

définition

```
def nom_de_f([s][, b=0][, *t][, **d]) :
    <instructions>
    [ return [expression]]
```

ordre des paramètres à respecter :
s : paramètre simple peut être optionel
b : paramètre par défaut ""
t : paramètres sous forme de tuple ""
d : paramètres sous forme de dict ""
Exemple :

```
def f(x) :
    x = x + 1
```

```
def succ(x) :
    return x + 1
```

appel
y = *nom_de_f(paramètres_effectifs)*
Exemple :
y= succ(7 + 5), y→13 | y= f(7 + 5), y→None
fonction *anonyme*
lambda [liste_paramètres] : *expression*
Exemple :
f = *lambda* x : x+1, f(7+5) → 13

Opérateurs numériques

Opérateurs	*Signification*
x + y, x - y	*som et diff*
*x * y, x / y*	*produit et quotient*
*x ** y*	x^y
x // y, x % y	*quotient et reste*

Exemple :
2017 // 13 → 155, 2**8 → 256
2017 / 13 → 155.15384, 255 % 7→ 3

Fichiers

un *fichier f* est une séquence d'objets.
ouverture / fermeture :
open(f [,mode][,encoding]) | *close()*
lecture / ecriture :
read([n])→str | *write("string")*→int
Exemple :

```
e = open("ent.txt", "rt", encoding="utf-8")
s = open("sor.txt", "wt", encoding="utf-16")
texte = e.read(80)
nbcar = s.write(texte)
e.close() ; s.close()
```

Module

structure d'un module

```
""" documentation module """

# charge et exécute le(s) module(s)
import module [" as" name], module
from module import m[" as" name] | *

vars_Globales = 'V'      # portée module

def fonction([paramètres]) :
    return <valeur>
if __name__ == '__main__' :
    <instructions>
```