Python3

Action: a une *durée* finie et un *effet* attendu et bien défini(*instruction*). **Objet**: cible de l'action et sur lequel on observe l'*effet* de celle-ci. **Valeur**: ce que vaut un objet. **Type**: domaine de valeurs. **Variable**: < *identificateur, valeur*>. **Expression**: désigne le calcul d'une valeur à partir d'autres valeurs et d'opérations. **Prédicat**: expression → booléenne. **Enoncé**: description d'actions.

Eléments de base

Symboles *(sinon voir PEP 3131)*:
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_]*

Mots clés

⚠ *(jamais comme identificateurs).*
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."""

Affectation augmentée

Exp	*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*.

Exp	*Signification*
x or y	*True si x ou y est True*
x and y	*True si x et y sont True*
not y	*True si y est False*

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

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

opérateur ternaire: ...if ... else...
expr_1 if prédicat else expr_2
Exemple :
x = -1; y = -12; x *if* x < y *else* y →-12

Fonction

définition
*def nom_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_f(paramètres_effectifs)*
Exemple :
y= succ(7 + 5), y→13 | y= f(7 + 5), y→None

fonction *anonyme*
lambda [liste_params]: *expression*
Exemple :
f = *lambda* x: x+1, f(7+5) → 13

autres *types* de fonctions

def generer([p]):
<instructions>
yield [exp]
<instructions>
...

def closure([p]):
<instructions>
def inner([t]):
...
[*return*[*inner*]]

Exemple :

def generer(x=0):
while x < 10:
yield x
x += 1
if x == 7:
return

def closure(x):
def inner():
nonlocal x
x += 1
return x
return inner

g = generer(); next(g)→0, f = closure(3), f()→4

Opérateurs relationnels

Exp	*Signification*
x == y	*True si x = y sinon False*
x < y	*True si x < y sinon False*
x <= y	*True si x ≤ y sinon False*
x > y	*True si x > y sinon False*
x >= y	*True si x ≥ y sinon False*
x != y	*True si x ≠ y sinon False*

Exemple :
1 < 2 < 3→*True*
1 < 2.5 > 2 != 0 < 0.1→*True*

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

while...else

while prédicat:
<instructions>
[*else* :
<instructions>]

Exemple:
x = 0; s = 0; n = 11
while x <= n:
s = s + x
x = x + 1
else:
print("s = ", s)

for

for var in itérable:
<instructions>

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

for...else

for var in itérable:
<instructions>
[*else* :
<instructions>]

Exemple:
s = 0
for x in range(10):
s = s + x
else:
print('x égal 10')

Indentation

module|class|def|if...else|while|for→nécessite 1 block: alignement vertical d'instruction(s)

tab ⟹ *<instruction(s)>* ... *block*

Assertion

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

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')]

Arithmétique des entiers

Exp	*Signification*
x | y	*ou logique (or)*
x ^ y	*ou exclusif (xor)*
x & y	*et logique (and)*
x ≪ n	*décalage n bits à gauche*
x ≫ n	*décalage n bits à droite*
∼x	*complément à 2*

Exemple :
0x11 & 0xb → 1, 0x11 ^ 0xb→ 26
a = 2; ∼a → -3, *a → -a-1 → a*

Aide Mémoire - Djamel Aouane - Grenoble-INP

Opérateurs numériques

Exp	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

Programmation Orientée objets

classe: implémentation d'un TAD.

```
class NomDeLaClasse[([liste_ancêtres])]:
    # Attributs de classe
    def __init__(self[,paramètres]):
        <instruction>
    def __del__(self):
        <instruction>
```

objet: occurrence d'une *classe.*
obj = NomDeLaClasse(params)

Entrées/Sorties

Entrée:
input([expression])→ *str*

Exemple :
raw = input("Saisir: ")→ Saisir: []

Sortie:
print([v_0][,v_i...][,sep=][,end=][,file=])

Exemple :
print("val: ", 19, 0x3F) —*affiche*→ *val: 19 63*

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")
with e, s:
    texte = e.read(80)
    nbcar = s.write(texte)
# les fichiers e et s sont auto fermés par with
# sans with il faut: e.close() et s.close()
```

Module

un *module m* fichier Python sans .py

structure d'un module

```
#!/usr/bin/python3
#-*- coding: utf-8 -*-
""" 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
class nom_Class[([liste_ancêtres])]:
    pass
def fonction([paramètres]):
    pass
if __name__ == '__main__':
    <instructions>
```

Module math

pi⟶ π 15 chiffres significatifs
sin(x) ⟶ sinus de x radians
sqrt(x) ⟶ racine de x
exp(x) ⟶ e^x
log(x[,b])⟶ log de x base b
floor(x)⟶ + grand int <= x
...
e ⟶ à 15 chiffres significatifs

Module sys

path→chemins modules	*stdin→Entrée*
maxsize→2^{31} -1 | 2^{63} -1	*stdout→Sortie*
version→interpreter	*stderr→Erreurs*
exit([arg])→Exit python	*argv→args cmd line*
float_info→info sur réel	*modules→chargés*

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.

Exp	Signification
t + s	*concaténation de t et s*
*s * n*	*concaténer n fois s*
x in t	*True si $x \in t$ sinon False*
len(s)	*nombres 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)

Tranches

t: séquence, *i*: int, *t[i]*: élément de *t*, *d*: début, *f*: fin et *p*: périodicité

p > 0 —*parcours*→ *gauche–>droite*

t[d:f:p] —*désigne*→ *t[d] ... t[f -1]*

	0	1	2	3	4	5	6
t	*P*	*y*	*t*	*h*	*o*	*n*	
	−6	−5	−4	−3	−2	−1	

p < 0 ←*parcours*— *droite–>gauche*

t[d:f:p] —*désigne*→ *t[f] ... t[d -1]*

↳ si *i < 0* alors *t[i]* ⇔ *t[len(t) + i]*

Exp	Signification
t[i:j]	*tranche de t selon [i,j[*
t[i:j:k]	*" de t avec période de k*

Exemple :
i = -4, t[i]→t, t[len(t) + i]==t[2]→t
t[1:6:2]→ "yhn", t[-1:-6:-2]→ "nhy"
t[::-1]→"nohtyP"⇔ t[len(s):-len(t)-1:-1]

Opérations sur listes

s, t des listes; *i,j,k* des entiers.

Exp	Signification
t[i:j] = ['t']	*t[i],...,t[j-1] = ['t']*
del t[i:j]	*supprime t[i],...,t[j-1]*
t.append(x)	*ajout de x à t*
t[i:j:k]=[...]	*voir exemple*

Exemple :
t[1::3]=['Y','O'], t→['P','Y', 't', 'h','O', 'n']
t[-2::-3]=['y','o'], t→['P','y', 't', 'h','o', 'n']

Module os

path→manip de chemins	*environ→variables d'environnement*
uname()→info système	...

Aide Interactive

help()→aide interactive
vars([objet])→affiche toutes les variables
help(m), où m: module \|fonction \|objet \|...
dir(m), où m: module \|fonction \|objet \|...

Opérations sur chaînes

s.methode	Signification
join(seq)	*concaténation*
split([sep])	*suite chaînes selon sep*

Exemple :
s = '' ; r = s.join(str([1,2,3])), r→"[1, 2, 3]"
s = "liste de chaînes"; r = s.split(sep=' ')
r→['liste', 'de', 'chaînes']

Dictionnaires

collection *mutable non ordonnée* de couples <clé, valeur>.
constructeurs: dict ([k_v]), {[k_v]}

d.methode	Signification
items()	*itér: dict_items(...)*
keys()	*itér: dict_keys(...)*
values()	*itér: dict_values(...)*
update()	*fusion de dicts*

Exemple :
d = {}; d['x']=10 ; dic['y']=15
d→{'x': 10, 'y': 15}, list(d.keys())→['x', 'y']
list(d.values())→[20, 30]

Ensemble

collection *mutable non ordonnée*:
constructeur: *set()*

Exp	Signification
|, &, -, ^	∪,∩, *diff, diff-sym*

Exemple :
X = set("python"), X→{'p','y','h','o','t','n'}
Y = set("pyt"), X \| Y→{'p','y','h','o','t','n'}
X & Y→{'p','y','t'}, X ^ Y→{'h', 'o', 'n'}
Y - X→set()

list|set|dict|tuple en compréhension

"["*exp for i in itérable* [*if exp*]"]"
"{"*exp for* j *in itérable*[*if exp*] "}"
"{"*exp:exp for* k *in itérable*[*if exp*]"}"

Exemple :
liste = [i*i for i in range(1,99) if x%10 == 7]
ens = {j*j for j in range(1,99) if x%10 == 7}
dico = {k:k*k for k in range(1,99) if x%17 }
uplet = tuple(k**k for k in range(1,99) if x%3)

Exception

exception, *"interruption"* résultant d'une exécution. E_i: UneException
lever exception:
raise E_i([arguments])
traiter exception:

```
try :                    try :
    <instructions>           <instructions>
except [Ei]:             except [Ei as m]:
    <instructions>           <instructions>
[else :                  [finally :
    <instructions>]          <instructions>]
```

Exemple: (ZDE: ZeroDivisionError)

```
try:                     try:
    0/1                      1/0
except Exception:        except ZDE as m:
    print("except")          print("ZDE")
else:                    finally:
    print("else")            print("finally")
```

affiche→else | affiche→ZDE+finally

Qq Exceptions : ZeroDivisionError, Exception, SyntaxError, IndexError, ImportError, TypeError, IOError, NameError, KeyError, IndentationError

Extra: Spyder3

fichier:	*edition:*
Ctrl+N→Nouveau	*Ctrl+C→copier*
Ctrl+Q→Quitter	*Ctrl+V→coller*
Ctrl+S→Sauver	*Ctrl+X→couper*
recherche:	*Ctrl+Z→undo*
Ctrl+L→ligne	*exécution:*
Ctrl+F→texte	*F5→programme*
Ctrl+H→remplacer	*F9→lignes choisies*