Fiche de révision Python — Niveaux 1 & 2

1. Types de données

Niveau 1 —

```
int     125, -102, 1253202
float   15.6, 2.3e10, 1e-5
bool    True, False
str     "Python", 'Monty'
list    [2,3,5], ["P","Q"]
        [1,'pi',[1,2]]
```

Niveau 2 —

```
complex   1+1j, complex(1,1)
tuple     (6,2,3), ('P','Q')
set       {1,3,8}, set()
dict      {'P':13, 'Q':9}
```

Mutables : listes, dict, sets, tableaux numpy.
Immuables : int, float, bool, str, tuple.
Aliasing : `B = A` ne copie pas la liste !
Copie superficielle : `L.copy()`, `list(A)`, `A[:]`
Copie profonde :

```
import copy ; copy.deepcopy(A)
```

2. Variables

```
A = 12
Nom = 'Paul'
x = 3.14
L = ['Paul', 12]
# affectation :
A = A**2
Nom = Nom + 'e'
```

Noms valides : lettres, chiffres, _ (pas de chiffre en 1er).

3. Opérations

Entiers — `+ - * // % **`

Flottants — `+ - * / **`

Booléens — `or and not`

Chaînes et listes — `+` (concat.), `*` (rép.)

Complexes — `+ - * /`

Sets — `|` (union), `&` (intersection)

Dicts — `|` (fusion, Python 3.9+)

Priorités (haute → basse) :

1 `()` parenthèses
2 `**` (droite→gauche)
3 `+x -x` unaire
4 `* / // %`
5 `+ -`
6 `== != < <= > >= in`
7 `not`
8 `and`
9 `or`

Ex. : `2+3*6 = 20` `2<=a<8` valide en Python.

Slicing (niv. 2) — `L[start:stop:step]`
S'applique aux listes, chaînes, tuples.

```
L = [-1, 0, 2, 4, 8, 12]

L[1:4]    # [0, 2, 4]
L[::2]    # [-1, 2, 8]
L[::-1]   # [12, 8, 4, 2, 0, -1]
T = "Hello world"
T[0:5]    # "Hello"
```

4. Fonctions utiles

Entiers — `abs()` `divmod()` `bin()` `type()`

Flottants — `abs()` `round()` `int()` `type()`

Chaînes — `len(t)` `list(t)` `t.lower()` `t.upper()`

Listes — `len(L)` `sorted(L)`
`L.append()` `L.sort()` `L.remove()`

Complexes (niv. 2) — `z.real` `z.imag` `abs(z)` `z.conjugate()`

Tuples (niv. 2) — `len(t)` `list(t)` `max(t)` `sum(t)`

Sets (niv. 2) — `s.add()` `s.remove()` `s1.union(s2)`

Dicts (niv. 2) — `d.keys()` `d.values()` `d.items()`
`d.pop()` `del d[k]` `d.update(...)`

Fonctions natives générales :

Conversion	`int()` `float()` `str()` `list()`
Maths	`abs()` `pow()` `round()`
	`max()` `min()` `sum()`
Itér.	`len()` `range()`
	`sorted()` `reversed()`
Types	`type()` `id()`
E/S	`print()` `input()` `open()`
Utiles	`any()` `all()`

`any()` `all()` —

```
notes = [12, 15, 10, 18]

all(n >= 10 for n in notes)   # True
all(n >= 15 for n in notes)   # False

any(n >= 15 for n in notes)   # True
any(n < 0 for n in notes)     # False
```

5. Entrées / Sorties

print et input (niv. 1) —

```
print("Hello World")
print(A)
print('Res.', A + 6)
c = input('Texte : ')
x = float(input('Nombre : '))
```

Fichiers texte (niv. 2) —

```
with open("f.txt", "r") as f:
    contenu = f.read()
with open("f.txt", "w") as f:
    f.write(contenu)
```

Images (niv. 2) —

```
import numpy as np
from PIL import Image
img = Image.open("image.png")
np_img = np.array(img)
# modifier np_img ...
img = Image.fromarray(np_img)
img.save("sortie.png")
```

6. Structures de contrôle

if / else (niv. 1) —

```
if condition:
    bloc1
else:
    bloc2
```

elif (niv. 2) —

```
if cond1:
    bloc1
elif cond2:
    bloc2
else:
    bloc3
```

6. Boucles

while —

```
while condition:
    bloc
```

for —

```
for k in range(n):            # k va de 0 à n-1
    bloc

for k in range(a, b, pas):    #de a à b-1 par pas
    bloc

for x in L:    # Parcours en valeurs d'une liste
    bloc       # ou d'une chaine de caractères
```

Listes en compréhension. (niv. 2) —

```
L = [expr for x in iter]
L = [expr for x in iter  if condition]
# exemples :
car = [k**2 for k in range(10)]
pai = [k for k in L if k%2 == 0]
```

Fiche de révision Python — Niveaux 1 & 2 (suite)

7. Définition de fonctions

Syntaxe de base —

```
def nom(param1, param2):
    instructions
    return objet
```

Exemple —

```
def f(x, y):
    z = x**2 + y**2
    return z
print(f(1, 2))    # affiche 5
```

Arg. par défaut (niv. 2) —

```
def subdivision(a, b, n=50):
    L = [a + k*(b-a)/n
         for k in range(n+1)]
    return L
```

Effet de bord volontaire (niv. 2) —

```
def premier_dernier(L):
    a = L.pop(0)   # modifie L !
    L.append(a)
```

Des fonctions sans `return` qui modifient les variables mutables passées en argument.

Attention aux effets de bord involontaires : les listes passées en argument sont modifiées en place. Penser à faire une copie pour l'éviter.

Variables locales / globales —

```
z = 3           # variable globale
def f(x, y):
    z = x**2    # z local != global
    return z
print(f(1,2))   # 1  (z global = 3)
```

8. Module math

```
import math as m

# ou : from math import *
m.sin(x)   m.cos(x)   m.tan(x)
m.exp(x)   m.log(x)   m.sqrt(x)
m.acos(x)  m.floor(x) m.pi
m.comb(n,k)  m.factorial(n)
```

9. Module numpy

```
import numpy as np

np.array(L)     np.zeros(n)
np.ones(n)      np.eye(n)
np.exp(A)       np.sin(A)
np.shape(A)     np.mean(A)
np.std(A)       np.polyfit(x,y,deg)
```

numpy.linalg (niv. 2) —

```
import numpy.linalg as la

la.inv(A)           # inverse
la.eig(A)           # vect. propres
la.eigvals(A)       # val. propres
la.matrix_rank(A)   # rang
```

10. Module matplotlib.pyplot

```
import matplotlib.pyplot as plt

plt.close('all')
plt.figure("Nom")
plt.plot(x, y, 'r', label="...")
plt.plot(x, z, '--+')
plt.legend()
plt.grid(True)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Titre")
plt.show()
```

11. Module random

```
import random as rd

rd.random()         # U[0,1[
rd.randint(a, b)    # entier [a,b]
rd.choice(L)        # element de L
rd.shuffle(L)       # melange L
rd.gauss(mu, sig)   # loi normale
rd.seed(42)         # graine fixee
```

12. Module scipy.stats (niv. 2)

```
from scipy.stats import norm

norm.pdf(x)       # densite N(0,1)
norm.cdf(x)       # Fonction de rep.
norm.ppf(p)       # quantile d'ordre p
```

13. numpy.random vs random (niv. 2)

numpy.random	random	sortie
np.random.rand(n)	rd.random()	tableau / scalaire
np.random.randint(a,b)	rd.randint(a,b)	tableau / scalaire
np.random.normal(m,s)	rd.gauss(m,s)	tableau / scalaire
np.random.uniform(a,b)	rd.uniform(a,b)	tableau / scalaire

Attention aux deux randint.

14. Exemples complets

Équation du second degré —

```
import math

def resoudre(a, b, c):
    d = b**2 - 4*a*c
    if d < 0:
        return 'Pas de racine reelle'
    if d == 0:
        return -b / (2*a)
    x1 = (-b - math.sqrt(d))/(2*a)
    x2 = (-b + math.sqrt(d))/(2*a)
    return x1, x2
```

Simulation (espérance empirique) —

```
import random as rd

def simul_X():
    R = []
    for k in range(10):
        R.append(rd.randint(1, 6))
    return sum(R)
N = 1000
s = 0
for k in range(N):
    s += simul_X()
print("Moyenne :", s/N)
```

Tracé de courbe —

```
import matplotlib.pyplot as plt

a, b, N = -2, 2, 100
x, y = [], []
for k in range(N+1):
    t = a + k*(b-a)/N
    x.append(t)
    y.append(t**2)
plt.plot(x, y, 'r')
plt.grid(True)
plt.show()
```