Aide-mémoire Python 3

I Les bases du langage Python

1. Calculs

- opérations de base et puissance : `+ - * / **`
- reste et quotient de la division euclidienne : `% //`
  ☞ $a = bq + r$ *avec* $|r| < |b|$, *Python :* $\text{sgn}(r) = \text{sgn}(b)$
- fonctions math basiques : **`max`**, **`min`**, **`abs`**, **`round`**
- nombres complexes : `1+2j` ; **`complex`**`(1, 2)`
- rappel du dernier résultat : `_`

2. Variables et affectations

- types & conversions : **`int`**, **`float`**, **`bool`**, **`list`**, **`str`**
- typage dynamique : pas de déclaration
- renvoyer le type de `x` : **`type`**`(x)`
- affectation simple : `ma_variable = -5`
- instructions sur la même ligne : `a = 1 ; b = 2`
- affectation multiple : `a = b = 0`
- affectation parallèle : `a, b = 1.2, -4`
- échange de variables : `a, b = b, a`
- affectation conditionnelle : `m = a` **`if`** `a > b` **`else`** `b`
- affectations combinées : `+=`, `-=`, `*=`, etc.
  `x += 1` → `x = x + 1`

3. Listes

- liste : `L = [-1, 3.2, 1/3, 1e-3]`
- élément d'une liste (modifiable) : `L[1] = 5`
  ☞ *doit déjà exister, l'indice commence à 0 !*
- fonctions de liste : **`max`**, **`min`**, **`len`**, **`sum`**
- multiplication de liste : `[0]*5` → `[0, 0, 0, 0, 0]`
- supprimer un élément d'une liste : **`del`** `L[i]`
- ajouter un élément à une liste : `L.`**`append`**`(2)`
- ajouter une liste à une liste : `L.`**`extend`**`([5, -1])`
- renvoyer le nombre de `x` : `n = L.`**`count`**`(x)`
- trier une liste : `L.`**`sort`**`()`
- insérer `x` à la position `i` : `L.`**`insert`**`(i, x)`
- affectation de liste : `L1 = L` (même objet !)
- tranchage : `L2 = L[i:j]` (`i` *inclus* → `j` *exclu*)
- listes « en compréhension » :

```
Xlist = [k/10 for k in range(-50, 51)]
Lsup = [x for x in L if x > 0]
```

- indices inversés : `L[-1]` (dernier terme)

4. Chaînes de caractères

- chaîne : `texte = "bonjour"` ou `'bonjour'`
- concaténation : `texte + "␣tout␣le␣monde"`
- élément d'une chaîne : `texte[i]`
  ☞ *non modifiable, l'indice débute à 0 !*
- longueur : **`len`**`(texte)`
- `*`, tranchage + méthodes nombreuses (voir aide) :
  `count`, **`find`**, **`replace`**, **`split`**, **`join`**, **`format`**, ...
- code ascii/unicode : **`ord`**`("A")` ; **`chr`**`(65)`

5. Entrées/sorties

- sorties : **`print`**`(valeur1, ..., sep="␣", end="\n")`
  `valeurs` *de type quelconque,* **`sep`** *et* **`end`** *optionnels*
- entrées : `texte = `**`input`**`("message")` → type **`str`**
  `a, b = `**`eval`**`(`**`input`**`("message"))` ← nbres, expressions
- test de validité : **`assert`** `condition, "erreur␣si␣faux"`

6. Commentaires

- en ligne : `# ...` ; bloc : `""" ... """` ou `''' ... '''`
- encodage (1re ligne) : `# coding: utf-8 (ou latin_1)`

7. Structures de contrôle

- structure alternative :

```
if condition1:
    instructions
elif condition2: # optionnel
    # elif = contraction de else if
    instructions
else: # optionnel
    instructions
```

  ☞ double-points + *indentation*, pas de end
- tests simples : `==`, `!=`, `<`, `<=`, `>`, `>=`, **`in`** → `True/False`
- tests combinés : **`and`**, **`or`**, **`not`**, `a < b < c`
- tester si `x` est entier : **`type`**`(x) == `**`int`**
- boucle while (boucle conditionnelle) :

```
while condition:
    instructions
```

- boucle for (boucle inconditionnelle) :

```
for i in liste: # ou chaîne
    instructions # boucle de parcours
```

```
for i in range(5): # pour i de 0 à 4
    instructions # boucle incrémentale
```

  `range``(1, 5)` : de 1 à 4 (*range = plage*)
  `range``(m, n, p)` : `m` *inclus* → `n` *exclu*, `p` = *pas*
- quitter/sauter une boucle : **`break`** **`(else)`**, **`continue`**

8. Fonctions

- définition de fonction :

```
def f(x, y): # nb quelconque de paramètres
    return x**2 + y**2 # valeur renvoyée par f
```

- paramètres : noms utilisés dans la définition
- arguments : valeurs/expressions utilisées à l'appel
- fonction récursive : fonction s'appelant elle-même
- méthode : fonction attachée à un objet
  ☞ *syntaxe* : `objet.`**`methode`**`()`
- renvois multiples : **`return`** `a, b` ou **`return`** `liste`
- paramètres optionnels : **`def`** `f(x, y=2, z=3):`
- fonction anonyme : **`lambda`** `x: x**2` ($x \mapsto x^2$)
- fonction définie à l'appel : **`def`** `f(x):` **`return`** **`eval`**`(expr)`
- portée des variables : locale si variable *redéfinie*
  ☞ *pour rendre la portée de* `a` *globale* : **`global`** `a`

II Modules et extensions

1. Vocabulaire

- module : fichier avec code (Python .py ou compilé)
- chargement en mémoire : `import fichier` (sans .py)
- script : module/programme destiné à être exécuté
- paquet (package) : dossier de modules (avec init)
- bibliothèque (library) : module/paquet générique
- bibliothèque standard : modules de base
- extension : paquet optionnel (téléchargement)
- distribution : Python + collection de paquets
  ☞ *paquets disponibles sur un dépôt*
- anaconda/miniconda : distributions
- conda, pip : gestionnaires de paquets
  `conda install <paquets>` ; `conda update <paquets>`

2. Importation et aide interactive

- importer : charger en mémoire
- importation « globale » (module) : `import math`
  ☞ *nécessite un préfixage* : `math.sin(math.pi/2)`
- importation « ciblée » : `from math import sin, pi`
- importation « exhaustive » : `from math import *`
- utilisation d'alias : `from math import log as ln`
- supprimer l'importation : `del log ; del math`
- renommage de fonction : `ln = log ; log = log10`
- aide (shell) : `objet.methode?` ou `module.fonction?`

3. Modules de la bibliothèque standard

- math : `sqrt, sin, exp, log, e, pi, floor, gcd, ...`
- random : `random(), randint(1, 6), uniform(0, 2)`
  `choice("a", "b", "c")`
- time : `perf_counter(), sleep(3)`
- fractions : `Fraction("-2/3"), Fraction(-2, 3)`
- decimal : `Decimal("0.2"), Decimal(str(eval(1/5)))`
- statistics : `mean(L), median(L), pvariance(L),`
  `pstdev(L)` (*population standard deviation*)
- turtle : `forward(dist), left(angle), goto(x, y),`
  `circle(r), up(), down(), pos(), speed(0-10),`
  `color("red"), begin_fill(), ..., mainloop()`

4. La bibliothèque graphique matplotlib

- module pyplot : fonctions graphiques essentielles
- importation globale, avec alias, ou exhaustive :
  `import matplotlib.pyplot as plt`
  ou `from matplotlib.pyplot import *`
  ☞ *évite le préfixage mais pollue l'espace de noms*

5. Courbes

- table des x : `Xlist = np.linspace(a, b, n)`
  ou `Xlist = np.arange(a, b, pas)` (ici `b` est exclu)
- liste des y : `Ylist = [f(x) for x in Xlist]`
- courbe : `plot(Xlist, Ylist, options)`
- afficher le graphique (*ne pas l'oublier*) : `show()`
- grille : `grid()`
- titre : `title("mon␣joli␣dessin")`
- fenêtre : `xlim(xmin, xmax), ylim(ymin, ymax)`
  ou `axis([xmin, xmax, ymin, ymax])` ; `axis("equal")`
- texte : `text(posx, posy, "string")`

6. Autres graphiques classiques

- nuage de points : `scatter(Xlist, Ylist, options)`
- diagramme à barres : `bar(Xlist, Ylist, options)`
  options : `tick_label=liste, align="center", ...`
- histogramme : `hist(datalist, classes, options)`
  avec `classes` = nb de classes ou liste
- boites à moustaches : `boxplot(dataL1, dataL2,...)`
- diagramme circulaire : `pie(datalist, labels=liste)`

7. Matrices

- paquet numpy : `import numpy as np`, module `linalg`
- définir une matrice : `A = np.array([[1, 0], [-1, 2]])`
  `O,I,D = np.zeros((2,2)), np.eye(3), np.diag([1,2,0])`
- produit : `np.dot(A, A)` ou `A.dot(A)` ou `A @ A`
- puissance : `np.linalg.matrix_power(A, 5)`
  ☞ `from numpy.linalg import matrix_power as mp`
- inverse : `np.linalg.inv(A)`
- en fractions :
```
from fractions import Fraction as frac
def frc(x): return str(frac(x).limit_denominator())
np.set_printoptions(formatter={"float":frc})
```

8. Solveur, intégration, stats et proba

- paquet scipy, modules optimize, integrate, stats
- $f(x) = 0$: `scipy.optimize.fsolve(f, valeur_init)`
- intégrale : `scipy.integrate.quad(f, a, b)` → (I, err)
- l'infini : `scipy.inf`
- coefficient binomial : `scipy.special.binom(n, p)`
- loi binomiale : `scipy.stats.binom.pmf(k, n, p)`
  (*probability mass function*)
- loi normale : `scipy.stats.norm.cdf(x, mu, sigma)`
  (*cumulative distribution fct = fct de répartition*)
- loi normale inverse : `scipy.stats.norm.ppf(p, mu, sigma)`
  (*percent point function = fonction fractile*)

9. Calcul formel

- paquet sympy : expressions avec chaînes ou symboles
- symboles formels : `x, y = symbols("x␣y")`
- développer : `expand("(x+y)**2")` ou `expand((x+y)**2)`
- factoriser : `factor(x*y+2*x), collect(3*x+x*y+1, x)`
- simplifier : `simplify(cos(x)**2 + sin(x)**2)`
- affectation : `expr = x**2 + y**2`
- substitution : `expr.subs(x, 3)`
- valeur approchée : `sqrt(75).evalf()`
- autres : `solve, limit, oo, diff, integrate, Sum, ...`
- matrices : `A = Matrix([[0, a], [-1, 0]]) ; A*A ; A**n`