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 :* $\text{sgn}(r) = \text{sgn}(b)$
- arrondi et troncature : `round`, `int`
- autres fonctions math basiques : `max`, `min`, `abs`

2. Variables et affectations

- types & conversions : `int`, `float`, `list`, `str`,...
- typage dynamique : pas de déclaration
- affectation simple : `ma_variable = -5`
- instructions sur la même ligne : `a = 1 ; b = 2`
- affectations multiples : `a = b = 0`
- affectations parallèles : `a, b = 1.2, -4`
- permutation de variables : `a, b, c = b, c, a`
- affectations combinées : `+=`, `-=`, `*=`, etc.
  `x += 1` ($\Longleftrightarrow$ `x = x + 1`)
- affectation conditionnelle : `m = a if a > b else b`

3. Listes

- liste : `L = [-1, 3.2, 1/3, 1e-3]`
- liste vide : `[ ]`
- multiplication de liste : `[0]*3` → `[0, 0, 0]`
- élément d'une liste : `L[1] = 5`
  doit déjà exister, l'indice commence à 0 !
- fonctions de liste : `max`, `min`, `len`, `sum`
- ajouter un élément à une liste : `L.append(-2.5)`
- ajouter une liste à une liste : `L.extend([5, 2, -1])`
- insérer x à la position i : `L.insert(i, x)`
- compter le nombre de x : `n = L.count(x)`
- renvoyer l'indice du 1er x : `i = L.index(x)`
- supprimer un élément : `del L[i]`
- tranchage : `L[i:j]` (i *inclus* → j *exclu*)
- attention aux affectations de liste : `L1 = L2`
  ☞ *pointent sur le même objet*
- listes par compréhension (ou par description) :

```
[ x/10 for x in range(-5, 6) ]
[ k**2 for k in range(10) if k % 3 != 0 ]
sum([n[i]*x[i] for i in range(len(x))])/sum(n)
```

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 !
- méthodes nombreuses (voir aide) :
  `count`, `find`, `replace`, `split`, `join`, `format`, ...
- code ascii/unicode : `ord("A")` ; `chr(65)`
- couper une ligne avec chaîne trop longue : \

5. Entrées/sorties

- sorties : `print(valeur,... , sep='␣', end='\n')`
  `valeur` *de type quelconque,* `sep` *et* `end` *optionnels*
- entrées : `texte = input("message")`

```
a, b = eval(input("message"))
```

6. Commentaires

- en ligne : `#`
- bloc : `'''...'''` ou `"""..."""`
- encodage (1re ligne) ☞ *pour caractères accentués* :
  `# coding: utf-8` ou `# coding: latin_1`

7. Structures de contrôle

- 1re ligne finit par ':', lignes indentées, pas de end
- tests simples : `==`, `!=`, `<`, `<=`, `>`, `>=`, `in`
- tests combinés : `and`, `or`, `not`, `a < b < c`
- structure alternative :

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

- boucle while (boucle conditionnelle) :

```
while condition:
    instructions
```

- boucle for (boucle inconditionnelle) :

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

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

  `range(i, j, k)` : i *inclus* → j *exclu*, k = *pas*
- quitter/sauter une boucle : `break (else)`, `continue`

8. Fonctions

- définition de fonction :

```
def f(x, y):
    return x**2 + y**2
```

- renvois multiples : `return a, b` ou `return liste`
- fonction définie à l'appel : `return eval(expression)`
- portée des variables : *locale si modifiée (sauf listes)*
  → `global x`
- paramètres (formels) : noms utilisés dans la définition
- arguments : valeurs/expressions utilisées à l'appel
- fonction récursive : fonction s'appelant elle-même
- effet de bord : action de la fct (≠ valeur renvoyée)
- méthode : fonction attachée à un objet
  syntaxe : `objet.methode()`

II Modules et extensions

1. Vocabulaire

- module : code Python dans fichier.py
- chargement en mémoire : `import fichier` (sans .py)
- script : module 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 + dépendances
  paquets disponibles sur un dépôt
- anaconda/miniconda : distribution
- conda : gestionnaire de paquets
  `conda install numpy pyqt matplotlib scipy sympy`
  `conda update conda; 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 sqrt, exp`
- 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`
- espace de travail (explorateur de variables) :
  contient variables et modules/fonctions importées
- aide : `objet.methode?` ou `module.fonction?`

3. Modules de la bibliothèque standard

- math : `sqrt, sin, log, exp, e, pi, floor, gcd`
- random : `random(), randint(1, 6), uniform(0, 2)`
- time : `time(), sleep(3), strftime('format')`
- fractions : `Fraction('2/3')` ou `Fraction(-3, 4)`
- statistics : `mean(L), median(L), pvariance(L), pstdev(L)` (*population standard deviation*)
- turtle : `goto(x, y), left(angle), forward(dist), backward(dist), up(), down(),..., 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*

5. Courbes

- courbe : `plot(Xlist, Ylist, options)`
- grille : `grid()`
- titre : `title("mon␣joli␣dessin")`
- effectuer le dessin (*ne pas l'oublier*) : `show()`
- fenêtre : `xlim(xmin, xmax) ; ylim(ymin, ymax)`
  ou `axis([xmin, xmax, ymin, ymax])`
- texte : `text(posx, posy, "string")`

6. Autres graphiques classiques

- nuage de points (foule d'options, s = size (20)) :
  `scatter(Xlist, Ylist, s=20, mark='o', color='red')`
- découpage d'intervalle : `Xlist = np.linspace(a, b, n)`
- diagramme à barres : `bar(Xlist, Ylist)`
- étiquettes : `xticks(Xpos, Xlabels)`
- histogrammes : `hist(datalist, bins)`
  avec `bins=nb_de_classes` ou `liste_des_classes`
- boites à moustaches : `boxplot(dataL1, dataL2,...)`
- diagramme circulaire : `pie(datalist, labels=stringlist)`

7. Matrices

- paquet `numpy` : `import numpy as np`
- définir une matrice : `A = np.array([[1, 0], [-1, 2]])`
- produit : `A.dot(A)` ou `np.dot(A, A)`
- autres fonctions : `identity, zeros, diag, shape, ...`
- puissance : `numpy.linalg.matrix_power(A, 5)`
  ☞ `from numpy.linalg import matrix_power as mp`
- inverse : `numpy.linalg.inv(A)`

8. Intégration, stats et proba

- paquet `scipy` (qui étend `numpy`)
- intégrale : `scipy.integrate.quad(f, a, b)`
  ☞ *renvoie aussi l'erreur*
- l'infini : `numpy.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)`
- 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 : `limit, diff, integrate, Matrix, plot, ...`