Introduction à la Programmation Python

On utilisera l'éditeur en ligne : Future Coder

A chaque utilisation de la fonction **`print`** ou **`input`**, un chaton meurt. Faîtes attention s'il vous plaît.

I Manipulation des objets élémentaires

Les exercices seront à réaliser *dans la console*.

Exercice 1. *Entiers*

Calculer le quotient et le reste de la division euclidienne par 1454586 de

$$(567 \times 34 - (222 + 103 \times (890 - 888)) \times 5 \times 3^{3127-5^5})^4 + 17471268$$

Exercice 2. *Opération Booléennes*

Observer l'ordre de priorité des opérations booléennes sur ces exemples.

1. Calculer `False` **`and`** `True` **`or`** `True`
2. Calculer `False` **`and`** `(` `True` **`or`** `True` `)`
3. Calculer `(` `False` **`and`** `True` `)` **`or`** `True`
4. Calculer **`not`** `False` **`or`** `True`, puis **`not`** `False` **`and`** `False`.

Exercice 3. *Flottants*

1. Calculer $\frac{(530{,}4709375*42)^{\frac{1}{6}}}{3{,}5}$
2. Comparer `150 // 12` et `150 / 12`.
3. Calculer $\sqrt{2}$
4. Calculer $\sqrt{2}^2$

Exercice 4. *Néant*

Taper `None` dans la console et observer l'absence de retour.

Exercice 5. *Comparaisons*

Observer le résultat des expressions booléennes suivantes.

```
3 < 4 and 4 < 3
3 < 4 or 4 < 3
3 < 3
3 <= 3
3 == 4 or 3 == 3
3 == 3.0
1/3 * 3 == 1
1/3 - 0.33333333333333333 == 0 # (17 chiffres après la virgule)
1 == 1 - 1e-17
```

```
1 == 0.99999999999999999 # (17 chiffres après la virgule)
0 == 1e-17
```

Observer le résultat de `False == True` et `True == True`. Quelque soit l'expression booléenne `B`, est-il pertinent d'écrire `B == True` ? Trouver une écriture plus simple de `B == False`.

II Nommage

Exercice 6. *Variables*

Essayez de prévoir, puis observez, le résultat dans la console des instructions suivantes. L'ordre des instructions doit être préservé.

```
variable1 = 12
x = variable1
x == variable1
x + x
x == 12
x = x * x
x == 12
variable1 == 12
x != variable1
variable2 = x < variable1
variable2
z = False
y = variable2 or x > variable1 and z
y == True
y
nadine = None
nadine
```

Exercice 7. *Fonctions, Appels*

Dans l'éditeur, ajoutez les définitions des fonctions suivantes nommées `nadine` et `dopamine` et possédant trois paramètres de type entier.

```
def nadine(parametre1, parametre2, parametre3):
    variable1 = parametre1 + parametre2
    return variable1 ** parametre3

def dopamine(parametre1, parametre2, parametre3):
    variable1 = (parametre1 * parametre2) % parametre3
```

Exécutez le code de l'éditeur, puis, dans la console, appelez la fonction `nadine` sur les entrées suivantes :

- `3, 4, 2`
- `4, 2, 3`
- `0, -5, 10`

Toujours dans la console, essayez de prévoir, puis observez, le résultat des instructions suivantes.

```
variable1 = 12
nadine(6,7,8)
variable1
parametre1 = 56
nadine(6,7,8)
nadine(56,7,8)
nadine(nadine(3,4,2), 10, nadine(1,1,nadine(1,2,1)))
dopamine(3,4,2)
dopamine(2,4,5) == None
```

Exercice 8. *Fonction, Définition*

Définissez les fonctions suivantes. Vous devrez appeler vos fonctions sur *plusieurs jeux* d'entrées bien choisies pour vérifier le résultat.

1. `racine` qui prend en entrée un flottant, `x`, et un entier naturel non nul, `n`, et qui renvoie la racine $n^{\text{ème}}$ de x.
2. `ordonnées` qui prend en entrée quatre entier, `a`, `b`, `c` et `d` et qui renvoie un booléen vrai si et seulement si `a < b < c < d`.
3. `quasiegal` qui prend en entrée deux flottants `x`, `y` et un flottant positif `e` et qui renvoie un booléen vrai si et seulement si $|x - y| < e$

Exercice 9. *Fonctions, Environnement*

Dans l'éditeur, ajoutez les définitions des fonctions suivantes nommées `f` et `g` et possédant des paramètres de type entier. On ajoutera aussi la définition de la variable globale `z`.

```
z = 12
x = 42

def f(x,y) :
    z = 2
    return z * x ** g(y)

def g(x) :
    return z - x

y = f(2,10)
```

Essayez de prévoir les valeurs de `x`, `y` et `z`, puis exécutez le code de l'éditeur et observez la valeur de `y`. Simulez le code de l'éditeur dans `Python Tutor` et observez les environnements.

Combien d'environnements existeront simultanément au plus lors de l'exécution de la fonction

`f(f(g(12),4),f(f(1,2),g(5)))`? Vérifiez en utilisant `Python Tutor`.

III Flot d'exécution

Exercice 10. *Appels*

Dans l'éditeur, écrivez le code suivant.

```
def f(x,y) :
    return h(x) + g(h(y))

def g(x) :
    return h(x*2)

def h(x) :
    return x + 5

y = f(12,21)
```

Listez les lignes exécutées de l'éditeur, dans l'ordre d'exécution. Certaines lignes seront exécutées plusieurs fois, elles apparaîtront à chaque exécution dans la liste.

Vérifiez avec `Python Tutor` ou `snoop`.

Exercice 11. *Conditionnelles*

Dans l'éditeur, écrivez le code suivant.

```
def f(x,y) :
    z = x*y
    if z == 12 or z > 100 :
        z = z * 2
    elif z < 50 :
        if z < 0 :
            z = - z
        z = z * 3
    else :
        z = z * 12
    return z
```

Listez les lignes exécutées de cette fonction, dans l'ordre d'exécution, lors de son appel sur les entrées suivantes :

1. 10 et 20
2. 5 et 6
3. 3 et 4
4. 7 et 10
5. -3 et 2

Vérifiez avec `Python Tutor` ou `snoop`.

Exercice 12. *Bissextile*

Définissez une fonction `bissextile`, qui prend en entrée un entier positif `annee`, et qui renvoie un booléen vrai si et seulement si l'année est bissextile, selon les règles actuelles. *i.e* si l'un des deux critères suivants est respecté :

1. L'année est divisible par 4 mais pas par 100
2. L'année est divisible par 400

`https://fr.wikipedia.org/wiki/Ann%C3%A9e_bissextile`

Exercice 13. *Boucles*

Dans l'éditeur, écrivez le code suivant.

```
def f(x) :
    z = 0
    r = 0
    while z < x :
        if z % 5 == 0 :
            r = r + 1
        z = z + 1
    n = x
    while r != 0 :
        n = n - 5
        r = r - 1
    return n
```

Listez les lignes exécutées de cette fonction, dans l'ordre d'exécution, lors de son appel sur les entrées suivantes :

1. 12
2. 21
3. 0
4. -5
5. 30

Vérifiez avec `Python Tutor` ou `snoop`.

Que fait la fonction `f` ?

IV Etude d'un algorithme

On considère l'algorithme écrit en pseudo-code ci-dessous :

Algorithme F

ENTRÉES : n un entier naturel

1. $f \leftarrow 1$
2. $i \leftarrow 0$
3. **tant que** $i < n$ **faire**
4. $\quad i \leftarrow i + 1$
5. $\quad f \leftarrow f * i$
6. **fin tant que**
7. **renvoyer** f

Exercice 14. *Simulation Papier*

On simule l'algorithme F sur l'entrée 5.

1. Combien y a-t-il de tours de boucles ? (nombre de fois que l'on exécute le bloc de la boucle).
2. Quelles sont les valeurs de i et f avant chaque exécution de la ligne 3 ?
3. Donnez une équation vérifiée par i et f avant chaque exécution de la ligne 3 ?

Exercice 15. *Spécification de* F

Décrire la sortie de F sur une entrée quelconque n, entier naturel.

Exercice 16. *Vérification*

Implémenter l'algorithme sur `Future Coder` et faire un appel sur l'entrée 5. A l'aide du bouton `snoop` d'abord, puis du bouton `Python Tutor`, vérifiez les résultats de votre simulation.

Exercice 17. *Somme des entiers*

Implémenter une fonction `somme_entiers` qui prend en entrée un entier naturel n et qui renvoie $\sum_{i=0}^{n} i$.

V Diviser ou ne pas Diviser ?

Exercice 18. *Divisibilité*

Implémenter une fonction `est_diviseur` qui prend en entrée deux entiers naturels a et b et qui renvoie un booléen vrai ssi a est un diviseur de b. (et qui renvoi donc faux si a n'est pas un diviseur de b, quoi d'autre ?)

Exercice 19. *Premier*

Un nombre entier naturel plus grand que 2 est premier ssi il ne possède aucun diviseurs parmi les entiers naturels strictement plus grands que 1 *et plus petits ou égaux à sa racine carrée.*

Voici une méthode simple, décrite en langue naturelle française, pour savoir si un nombre est premier.

```
On considère que le nombre, n, est premier, jusqu'à preuve du contraire.
On part de l'entier 2.
Tant qu'on a trouvé un diviseur, ou qu'il reste des nombres entiers à tester
entre 2 et la racine carrée de n : on vérifie si l'entier courant est un
diviseur de n et si c'est le cas, on note que le nombre n'est finalement pas
premier.
```

Complétez et implémentez l'algorithme suivant pour qu'il renvoie un booléen vrai ssi n est premier.

L'algorithme essaye de reproduire fidèlement la description précédente. Vous *devrez* utiliser la fonction `est_diviseur`.

Algorithme P

entrées : n un entier naturel plus grand ou égal à 2

```
1. p ← ...
2. d ← 2
3. tant que ... faire
4.     si ... alors
5.         p ← ...
6.     fin si
7.     d ← d + 1
8. fin tant que
9. renvoyer p
```

Simulez l'algorithme sur `Python Tutor` sur l'entrée 11. Combien d'environnements sont créés au total ? Combien y en a-t-il au plus existants simultanément ?

Exercice 20. *Euclide*

Implémenter en Python l'algorithme d'Euclide qui prend en entrée deux entiers a et b et qui renvoie le `PGCD` de ces deux entiers.

`https://fr.wikipedia.org/wiki/Algorithme_d%27Euclide`

VI Suites récurrentes

Exercice 21. *Méthode de Halley pour le calcul de $\sqrt{2}$*

Implémenter en Python la fonction `halley` qui calcule le $n^{\text{ème}}$ terme de la suite récurrente suivante :

$$\mathrm{H}_0 = 1\,,\ \mathrm{H}_{n+1} = \mathrm{H}_n \cdot \frac{(\mathrm{H}_n^2 + 6)}{(3\,\mathrm{H}_n^2 + 2)}$$

Exercice 22. *Suite de Fibonacci*

Implémentez en Python la fonction `fibonacci` qui calcule le $n^{\text{ème}}$ terme de la suite récurrente d'ordre 2 suivante :

$$\mathrm{F}_0 = 0\,,\ \mathrm{F}_1 = 1\,,\ \mathrm{F}_{n+2} = \mathrm{F}_{n+1} + \mathrm{F}_n$$

Exercice 23. *Syracuse*

Les mathématiques ne sont pas encore prêtes pour de tels problèmes – Paul Erdõs

Une suite de Syracuse est définie par la relation de récurrence suivante :

$$\mathrm{S}_{n+1} = \begin{cases} \dfrac{\mathrm{S}_n}{2} & \text{si } \mathrm{S}_n \text{ est pair,} \\ 3\mathrm{S}_n + 1 & \text{si } \mathrm{S}_n \text{ est impair.} \end{cases}$$

La conjecture de Syracuse stipule que, quelque soit $\mathrm{S}_0 \in \mathbb{N}^*$, la suite de Syracuse débutant par S_0 atteint 1. On notera qu'une fois la valeur 1 atteinte, la suite boucle sur les valeurs $1, 4, 2, 1, \ldots$.

Implémentez une fonction `syracuse` qui prend en entrée un entier strictement positif s_0 et qui renvoie le premier rang auquel la suite atteint la valeur 1.

Est-ce que la fonction `syracuse` termine sur toutes les entrées ?

VII Interlude – Pour s'entraîner

Exercice 24. *Suites adjacentes*

Implémentez en Python la fonction `adjacentes` qui prend en argument un flottant ϵ et qui calcule le premier rang n termes des suites adjacentes suivantes, à partir duquel $|a_n - b_n| < \epsilon$:

$$b_{n+1} = \frac{a_n + b_n}{2}\,,\ a_{n+1} = \frac{2}{b_{n+1}}\,,\ a_0 = 1, b_0 = 2$$

Exercice 25. *Méthode de Héron*

La suite x^a définie si dessous converge vers $\sqrt{a}$.

$$\forall n \in \mathbb{N} \quad x^a_{n+1} = \frac{x^a_n + \frac{a}{x^a_n}}{2}\,,\ x^a_0 = a$$

Implémentez une fonction `heron` qui prend en argument deux entiers a et n positifs et renvoie x^a_n.

Trouvez le rang à partir duquel x^2_n approche $\sqrt{2}$ à 10^{-10} prêt (≈ 1.4142135623).

Exercice 26. *Nombre parfait*

Un nombre parfait est un nombre entier positif égal à la somme de ses diviseurs stricts (lui-même exclu).

Implémentez une fonction `parfait` qui prend en entrée un entier et renvoie vrai s'il est parfait.

VIII Interlude – Pour aller plus loin

Exercice 27. *Logarithme entier*

On rappelle que pour tout $n \geq 1$ et $b \geq 2$, le logarithme entier en base b de n est l'unique entier l tel que

$$b^l \leq n < b^{l+1}$$

Implémentez en Python la fonction ayant les spécifications suivantes :

Algorithme `logarithme_entier`

entrées : $n \geq 1$ et $b \geq 2$ deux entiers
sorties/effets : un entier. Le logarithme entier en base b de n

Exercice 28. *Nombres Chanceux d'Euler*

Un nombre chanceux d'Euler est un nombre $n > 1$ tel que pour tout $0 \leq i \leq n-2$, $i^2 + i + n$ est premier.

Implémentez une fonction `chanceux_euler` qui prend en entrée un entier n et renvoie vrai ssi n est chanceux.

Exercice 29. *Fonction 91 de McCarthy*

La fonction 91 de McCarthy est définie pour $n \in \mathbb{N}$ par la formule suivante :

$$f(n) = \begin{cases} n - 10 & \text{si } n > 100 \\ f(f(n+11)) & \text{sinon.} \end{cases}$$

Implémentez une fonction `mccarthy` qui prend en entrée un nombre entiers n positifs et renvoie $f(n)$. Observer le retour de la fonction pour les valeurs plus petites ou égales à 101.

IX Manipulation des listes

Exercice 30. *Création*

1. Affecter à la variable `l0` la liste `[1,3,6]`.
2. Essayer d'accéder à l'indice 3 de la liste `l0`.
3. Affecter à la variable `x` la dernière valeur de `l0`.
4. Affecter à la variable `l1` la liste contenant 144 entiers `12`.
5. Affecter à la variable `l2` la concatenation de `l0` et `l1`.
6. Affecter à la variable `l0` une liste vide.
7. Affecter à la variable `l1` une sous-liste de `l2` contenant les éléments d'indice `1,2,3` et `4`.

Exercice 31. *Modification*

1. Modifier l'indice `0` de `l1` pour être égal à 1.
2. Modifier l'indice `1` de `l1` pour être égal au quotient de la division. euclidienne de la valeur à l'indice `2` par la valeur à l'indice `1`.
3. Enlever le dernier élément de la liste `l1` et l'affecter à la variable `x`.
4. Modifier l'indice `2` de `l1` pour être égal à la longueur de la liste `l1`.
5. Ajouter le reste de la division euclidienne de 40 par `x` à la fin de la liste `l1`.

X Etude d'un algorithme

Algorithme `nadine`

entrées : l une liste d'entiers naturels

```
1.  n ← len(l)
2.  si n = 0 alors
3.      renvoyer None
4.  fin si
5.  m ← l[0]
6.  i ← 1
7.  tant que i < n faire
8.      si m < l[i] alors
9.          m ← l[i]
10.     fin si
11.     i ← i + 1
12. fin tant que
13. renvoyer m
```

On considère dans les exercices suivants l'algorithme écrit en pseudo-code ci-dessus.

Exercice 32. *Simulation Papier*

On simule l'algorithme `nadine` sur l'entrée `[3,2,4,1,2,5,0,10,7,8]`.

1. Combien y a-t-il de tours de boucles ? (nombre de fois que l'on exécute le bloc de la boucle).
2. Quelles sont les valeurs de i et m avant chaque exécution de la ligne 7 ?
3. Donnez une expression de m en fonction de i et l, valide avant chaque exécution de la ligne 7.

Exercice 33. *Spécification de* `nadine`

Décrire la sortie de `nadine` sur une entrée quelconque l, liste d'entiers naturels.

Exercice 34. *Vérification*

Implémenter l'algorithme sur `Future Coder` et faire un appel sur l'entrée `[3,2,4,1,2,5,0,10,7,8]`. A l'aide du bouton `snoop` d'abord, puis du bouton `Python Tutor`, vérifiez les résultats de votre simulation.

XI Algorithmique des listes

Exercice 35. *Minimum de liste*

Implémentez une fonction **`min`** qui prend en entrée une liste d'entier et renvoie le minimum de la liste.

Exercice 36. *Somme de liste*

Implémentez une fonction `somme` qui prend en entrée une liste d'entier et renvoie la somme des éléments de la liste.

Exercice 37. *Maximum, minimum, moyenne, variance*

Implémentez quatre fonctions `imax`, `imin`, `moyenne`, et `variance` qui prennent en argument une liste d'entiers et qui renvoient, respectivement, l'indice du maximum, l'indice du minimum, la moyenne et la variance des valeurs de la liste.

Exercice 38. *Recherche Linéaire*

Implémentez une fonction `recherche` qui prend en entrée une liste d'entiers l et un entier x et qui renvoie l'indice dans la liste de la première occurrence de x dans l. Si x n'apparaît pas dans l, la fonction devra renvoyer -1.

Exercice 39. *Somme de deux listes*

Implémentez une fonction `somme` qui prend en entrée deux listes de même longueur $L_1 = [a_1, \dots, a_n]$ et $L_2 = [b_1, \dots, b_n]$, qui renvoie la liste $L = [a_1 + b_1, \dots, a_n + b_n]$ obtenue en sommant un à un les éléments de L_1 et L_2.

Exercice 40. *Liste des diviseurs*

Implémentez une fonction `diviseurs` qui prend en entrée un entier strictement positif `n` et qui renvoie une liste contenant les diviseurs de `n`.

XII Pour s'entraîner

Exercice 41. *Moyenne pondérée*

Implémentez en Python la fonction `moyenne_ponderee` qui prend en entrée une liste de nombres entiers positifs `L` et une liste de coefficient `C` de même longueur et qui renvoie la moyenne de la liste `L` pondérée par les coefficients de C.

Exercice 42. *Éléments impairs*

Implémentez une fonction `impairs` qui prend en entrée une liste d'entiers et qui renvoie une liste contenant les éléments impairs de la liste d'entrée.

Exercice 43. *Liste des multiples*

Implémentez une fonction `multiples` qui prend en entrée deux entiers naturels non nuls n et m et qui renvoie la liste des multiples de `m` plus petits que `n`.

Exercice 44. *Encadrement*

Implémentez en Python la fonction `encadrement` qui prend en entrée une liste de nombres entiers positifs `L` et un nombre entier `x` et qui renvoie le couple `(a,b)` tel que `a` est le plus grand élément de `L` plus petit ou égal à `x` (-1 s'il n'y en a pas) et `b` est le plus petit élément de `L` plus grand que `x` (-1 s'il n'y en a pas).

XIII Pour aller plus loin

Exercice 45. *Crible d'Eratostene*

Le crible d'Eratostene est un algorithme permettant d'obtenir la liste des nombres premiers plus petit qu'un entier fixé $n \geq 2$. Le crible fonctionne comme ceci :

1. noter tous les nombres entiers de 2 à n
2. prendre le premier nombre qui n'est pas éliminé.
   - il est premier, le noter dans la liste des nombres premiers.
   - éliminer ce nombre et tous ses multiples de la liste initiale.
3. Répéter `2.` tant que tous les nombres de la liste initiale ne sont pas barrés.

Implémentez une fonction `eratostene` qui prend en entrée un entier `n` supérieur à 2 et qui renvoie la liste des nombres premiers plus petits ou égaux à `n` en utilisant le crible d'Eratostene.

Exercice 46. *Tri de liste*

Décrire un algorithme en pseudo-code qui permet de renvoyer une copie triée par ordre croissant d'une liste d'entier.

Vérifiez votre algorithme en le simulant sur des exemples pertinents.

Implémentez l'algorithme en python et réalisez un jeu de test pertinent.

Exercice 47. *Calcul de la Médiane*

Implémentez une fonction `mediane` qui calcule la médiane d'une liste d'entiers.

Si la liste possède un nombre pair d'éléments, la médiane sera le milieu des deux éléments centraux de la liste. *e.g.* la médiane de `[4,3,8,7,5,7]` est le milieu de `5` et `7` soit `6`.