1 — Syntaxe

Ex. 1 — Prévoir les résultats des expressions suivantes, puis les vérifier grâce à l'interpréteur interactif de Python.

1. `(1,2)`
2. `(1)`
3. `(1,)`
4. `(,)`
5. `()`
6. `()+()`
7. `()+() == ()`
8. `(1,2) + (3)`
9. `(1,2) + 3`
10. `len((1,2))`
11. `len(())`
12. `(1,2,3)[1] = -1`

Ex. 2 — Prévoir les résultats des expressions suivantes, puis les vérifier grâce à l'interpréteur interactif de Python.

1. `"abba"`
2. **abba**
3. `"" == " "`
4. `"12" + 3`
5. `"12" + (3,)`
6. `"12" + ("3",)`
7. `"12" + ("3")`

Ex. 3 — Pour chaque séquence d'instruction, prévoir son résultat et le vérifier.

1.
```
t = (2, "abba", 9, 6*9, 22)
t
type(t)
print(t)
type(print(t))
t[0]
t[-1]
t[1]
t[1] = "beatles"
t[1] == "abba"
```

2.
```
z = (45, 5)
x, y = z
(x, y) == x, y
(x, y) == (x, y)
x, y = y, x
y == 5
```

3.
```
t = "Bonjour tout le monde !"
t[0]
t[0] = "b"
"a" in t
"onjo" in t
```

1.1 — Boucle `for`

Ex. 4 — Calculer, à l'aide de **Python**, $\sum_{k=831}^{944} k^{10}$.

Ex. 5 — Petite astuce : avec l'option `end = ""` la fonction `print` ne met pas de retour chariot à la fin d'une liste, mais un simple espace. Le `print` suivant s'affiche donc à la suite de la ligne. Par exemple `print(test, end = "")`.
Créer une boucle qui affiche

1. 12 zéros séparés par des espaces
2. `42 41 40 ... 26 25 24` (il s'agit de remplir les `...` et pas à la main...)
3. `0 0 0 2 2 2 ... 10 10 10`
4. `1 22 333 ... 10101010`

5. `1 21 321 ... 987654321`

Ex. 6 — Projet Euler 1 Si on liste tous les entiers naturels inférieurs à 10 qui soient soit multiple de 3, soit multiple de 5, on obtient 3, 5, 6 et 9 ; leur somme vaut $3+5+6+9=23$.
Écrire la fonction sans argument liprobleme1() qui renvoie la somme de tous les entiers multiples de 3 ou de 5 qui soient strictement inférieurs à 1000.

Ex. 7 — Écrire une fonction qui prend en paramètre deux entiers **n** et **p** et qui renvoie $\binom{n}{p}$. Cette fonction fera au maximum $2p$ opérations élémentaires.

Ex. 8 — On définit les deux suite $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ par

$$u_0=1, \qquad \text{et} \qquad \forall n\in\mathbb{N}, \qquad u_{n+1}=\frac{1}{2}\left(u_n+\frac{1}{u_n}\right) \qquad v_n=\sum_{k=0}^{n}\frac{1}{u_k^5}$$

Écrire une fonction qui calcule le terme d'ordre n de la suite v_n. Cette fonction doit pouvoir calculer v_{10^6} en une poignée de secondes.

Ex. 9 — Écrire deux fonctions **Python** pour calculer les sommes

$$\sum_{1\leqslant i,j\leqslant n}\frac{1}{i+j^2} \qquad \text{et} \qquad \sum_{1\leqslant i<j\leqslant n}\frac{1}{i+j^2}$$

Ex. 10 — Écrire une fonction **Python** qui renvoie le nombre de points à coordonnées entières dans le cercle de centre O et de rayon $R>0$.

Ex. 11 — 1. Proposer une fonction **Puissance(x, n)** qui renvoie x^n en effectuant uniquement des multiplications.

2. Proposer une fonction **double_flèche(x,n)** qui calcule $x\uparrow\uparrow n$ défini par

$$x\uparrow\uparrow n=\underbrace{a^{a^{\cdot^{\cdot^{a}}}}}_{n \text{ fois}}$$

Ex. 12 — Autour des nombres premiers On dit qu'en entier naturel $n\in\mathbb{N}^*$ est *premier* ssi il n'est divisible que par 1 et par lui-même. Un entier qui n'est pas premier est dit *composé*.
Un entier composé n peut donc s'écrire comme le produit ab de deux entiers différents de 1 et n. L'un de ces deux entiers est nécessairement inférieur ou égal à $\sqrt{n}$.
En tirant partie de la remarque précédente, écrire une fonction **estPremier(n)** qui renvoie **True** si **n** est premier et **False** sinon. Cette fonction doit effectuer au plus $\sqrt{n}$ calculs.

Ex. 13 — On considère la suite définie par

$$u_0=1 \qquad \forall n\in\mathbb{N}, \quad u_{n+1}=\sum_{k=0}^{n}(nk+1)u_k$$

Écrire une fonction qui calcule l'élément d'indice n de cette suite.
Vérification : $u_6=107\,632$

Ex. 14 — Nombres d'Armstrong Afficher tous les nombres de trois chiffres qui sont égaux à la somme des cubes de leurs chiffres (en base 10).

1.2 — Boucles `while`

Ex. 15 — Écrire une fonction **Python** qui demande à l'utilisateur un nombre plus grand que 1. La fonction continuera de poser la question tant que l'utilisateur n'a pas entré un nombre valide.

Ex. 16 — Pierre-Feuille-Ciseau Implémenter le jeu « Pierre-Feuille-Ciseau ». Le programme demande à l'utilisateur son choix parmi `"P"`, `"F"` ou `"C"` (et continue à demander jusqu'à obtenir une réponse valide), joue au hasard et affiche le vainqueur.

Ex. 17 — On note $u_n = \sum_{k=1}^{n} k^5$. Quelle est la plus petite valeur de n pour laquelle $u_n > 1000$. On écrira une fonction ; il y a au moins trois implémentations différentes possibles (deux avec une `while` et une avec `for`).

Ex. 18 — Écrire une fonction qui renvoie le plus grand diviseur d'un entier **n** passé par paramètre.

Ex. 19 — Écrire une fonction **Somme_Nombre(n)** qui calcule la sommes des chiffres de la réprésentation en base 10 de l'entier n. On se propose d'appliquer l'algorithme suivant :

EntrÃľe : $n \in \mathbb{Z}$
  $s \leftarrow 0$
  tant que $n \neq 0$:
    s = s + n%10
    n = n//10
  fin tant que
  renvoie s

1. Appliquer cet algorithme à la main sur l'entier 15423 et vérifier qu'il est correct. Prouver qu'il se termine.
2. L'implémenter et en déduire **Somme_Nombre(10!)**.

Ex. 20 — 1. Écrire une fonction qui renvoie la liste des n premiers carrés parfaits (1, 4, 9, etc.).
2. Écrire une fonction qui renvoie la liste des carrés parfaits plus petits que l'entier n.
3. Écrire une fonction qui tire au hasard un nombre entier entre 1 et n, qui s'arrête dès qu'un carré parfait est obtenu et renvoie le nombre de tirages effectués.

2 — Listes

Ex. 21 — Algorithmes de base sur les listes Écrire une fonction...

1.a) ... **RechercheElement(L,x)** qui renvoie `True` si l'élément **x** est dans la liste **L**, **False** sinon. Y-a-t'il un équivalent **Python** déjà prévu ?

b) ... **IndiceElement(L,x)** qui renvoie le premier indice de l'élément **x** s'il est dans la liste **L**, **False** sinon. Y-a-t'il un équivalent **Python** déjà prévu ?

c) ... **IndiceElement(L,x)** qui renvoie la liste des indices de l'élément **x** s'il est dans la liste **L**, **False** sinon. Y-a-t'il un équivalent **Python** déjà prévu ?

2.a) ... **MinListe(L)** qui renvoie la valeur maximale de la liste **L**. Y-a-t'il un équivalent **Python** déjà prévu ?

b) ... **MinListe(L)** qui renvoie la valeur maximale de la liste **L** ainsi que la liste des indices où se maximum est présent. Y-a-t'il un équivalent **Python** déjà prévu ?

3.a) ... **SommeListe(L)** qui renvoie la somme des éléments de la liste **L**. Y-a-t'il un équivalent **Python** déjà prévu ?

4.a) ... **AplliqueFonction(L, f)** qui applique la fonction **f** à chaque élément de la liste **L** et renvoie le résultat. Y-a-t'il un équivalent **Python** déjà prévu ?

Ex. 22 — 1. Calculer avec **Python** ces différentes listes :

```
[i for i in range(10)]
[j**2 for j in range(7)]
[(i, -i+1) for i in range(-5, 5)]
```

2. Sur ce modèle, obtenir de manière synthétique
   a) la liste des 20 premiers entiers naturels impairs ;
   b) la liste de tous les multiples de 5 entre 100 et 200 (inclus) ;
   c) les coordonnées des quatre voisins d'un point de coordonnées (i, j) sur un quadrillage entier ;
   d) les coordonnées des huit voisins d'un point de coordonnées (i, j) sur un quadrillage entier.

Ex. 23 — Carnet d'adresse On programme en **Python** un mini-carnet d'adresse. Chaque contact est représenté par une liste de la forme

```
[Prénom, numéro de téléphone, adresse e-mail]
```

Ainsi le carnet d'adresse est une liste contenant plusieurs listes, comme par exemple

```
[ [ "Audrey", 0621123113, "audrey@gmail.com"],
  [ "Ben", 0661165115,  ""],
  [ "Didier", 0621124112,  "dede@hotmail.com"]]
```

1. Écrire une fonction **coordonnée**(**C**, **nom**) qui renvoie les coordonnées de la personne **nom** si elle est dans le carnet **C** et qui renvoie un message d'erreur sinon.
2. Écrire une fonction **ajout**(**C**, **E**) qui ajoute l'entrée **E** au carnet **C**.
3. Écrire une fonction **doublon**(**C**) qui recherche les personnes présentes plusieurs fois dans le carnet **C**, qui tente de fusionner les entrées si possibles, et qui renvoie le carnet **C** modifié ainsi que la liste des doublons non résolus.

Ex. 24 — Crible d'Ératosthène avec des listes 1. Créer une fonction **effacemultiple** qui prend en argument une liste **L** et un entier **n** et qui renvoie une liste contenant les éléments de L qui ne sont pas multiples de n.

2. Programmer l'algorithme du crible d'Ératoshène, qui fournit la liste des nombres premiers entre 2 et un entier N.
   Créer tout d'abord la liste des entiers entre 2 et N, puis utiliser de façon répétée la fonction **effacemultiple** de la question précédente.
   Afficher le résultat. Quel est le nombre premier le plus proche de 1000 ?

Ex. 25 — Une liste de chaines de caractères étant donnée, écrire une fonction qui renvoie le nombre de chaînes de cette liste de longueur au moins 2 et dont le premier et le dernier caractères sont identiques.
Exemple de liste d'entrée : `['a', 'xyz', 'abba', 'idem', 'ioioioi']`

Ex. 26 — Deux listes d'entiers triées par ordre croissant étant données, écrire une fonction qui renvoie une liste triée formée du contenu de ces deux listes.
Cette fonction doit fonctionner en « temps linéaire », ne faisant qu'une seule passe sur chaque liste.

Ex. 27 — Une liste d'entiers étant donnée, écrire une fonction qui renvoie une liste identique, mais où tous les éléments consécutifs égaux ont été supprimés.
Par exemple avec la liste `[1,2,2,4,1,2,2]`, la fonction renvoie `[1,2,4,1,2]`

Ex. 28 — Une liste de flottant peut contenir des termes de signes différents. Par exemple

```
[13, 12.1, -5.5, 0, -7, 21.2, 32, 0, -4]
```

Lorsque deux termes successifs non nuls sont de signes distincts, on dit qu'il y a un changement de signe dans la liste. Si un des termes est nul, on « l'oublie » dans le décompte des signes. Ainsi dans la liste précédente, on compte 3 changements de signes : un entre `12.1` et `-5.5`, un entre `-7` et `21.2` et un dernier entre `32` et `-4`.

1. Écrire une fonction Python **`supprime_zéro(L)`** qui prend en paramètre une liste **`L`** et renvoie la même liste dont on supprimé tous les termes égaux à `0`.
2. Écrire une fonction Python **`compte_chgt_signe(L)`** qui prend en paramètre une liste de flottants **`L`** et qui renvoie le nombre de changements de signes de **`L`**.

Ex. 29 — Suite de Conway La suite « audio-active » a été inventé en 1986 par le mathématicien John Conway. Chaque terme de la suite se détermine en énonçant à haute voie les chiffres du terme précédent.
Ainsi, si on commence par « 1 » le terme suivant sera « 11 », puis « 21 », ensuite « 1211 », etc.

1. Écrire une fonction **`SuiteConway(init, n)`** qui renvoie la liste des $n+1$ premiers termes de la suite de Conway commençant par **`init`** (qui sera un nombre écrit avec des « 1 » et des « 2 ».
2. On note L_n la longueur du n–ième terme. « Vérifier » que la suite L_{n+1}/L_n converge vers une limite indépendante du terme de départ.

Ex. 30 — Problème de Josephus N joueurs sont aussi en cercle. Le joueur 1 choisit un entier p. On part du joueur 1 vers la droite, on compte p joueurs et on élimine le joueur sur lequel on tombe. Puis on recommence mais en partant vers la gauche. On réitère le processus, en alternant les sens de parcours. Les joueurs éliminés ne participent évidemment plus à la suite du jeu. Le dernier joueur présent est le gagnant du jeu.

1. Programmer une fonction **`Josephus(n,p)`** qui prend en paramètre **`n`** et **`p`** et qui renvoie l'indice initial du gagnant.
2. Programmer une fonction **`choixp(n)`** qui renvoie, si elle existe, une valeur de **`p`** assurant au joueur 1 de gagner le jeu.

Ex. 31 — Rendu de monnaie On souhaite rendre une certaine somme entière en pièces, mettons **`N`**€ à l'aide de pièce de `1`, `2`, `5` et `10`€.

1. On va utiliser pour cela un algorithme glouton, qui consiste à rendre d'abord le maximum de pièce de 10€, puis des pièces de 5, etc. Implémenter un tel algorithme.
2. La Banque Centrale de Syldavie réfléchit à un jeu de pièce optimal permettant de rendre la monnaie avec un minimum de pièce, en utilisant l'algorithme glouton.
   Recopier puis modifier le code précédent pour qu'il utilise un jeu de pièce quelconque (et pas seulement `[10,5,2,1]`). Le tester avec $S = 8$ et $[6, 4, 1]$. Que remarquez-vous ?
   (Caractériser les jeux de pièces pour lesquels l'algorithme glouton est optimal est un problème ouvert...)

Ex. 32 — Effet de bord Écrire une fonction qui incrémente d'une unité le premier élément d'une liste **`L`**

1. en renvoyant la liste modifiée à l'aide d'un **`return`**
2. sans renvoyer la liste modifiée, donc sans utiliser de **`return`**.