TP – Fondamentaux : boucles et instructions conditionnelles

Objectifs

◇

1. Boucles

1.1 Boucle `for`

Exercice 1 – Calculer les termes d'une suite récurrente

Question 1. On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} u_0 = 1, \\ \forall n \in \mathbb{N}, \quad u_{n+1} = 2u_n + 3. \end{cases}$$

- **1.1.** Préciser les valeurs de u_n pour $n \in \{1, 2, 3, 4, 5\}$.
- **1.2.** Écrire une fonction `suite_u` qui calcule la valeur u_n pour un rang n donné. On prendra garde à appliquer toutes les bonnes pratiques de programmation (signature, documentation, préconditions...etc)
- **1.3.** Dessiner un tableau de suivi de variables pour l'appel `suite_u(5)`. On précisera à quel endroit de la boucle on effectue la « photographie » des variables.
- **1.4.** Vérifier la valeur finale déduite de votre suivi de variable en écrivant un test simple.
- **1.5.** Calculer u_{100}.

Exercice 2 – Sommer, multiplier et compter

Question 1. On considère la fonction suivante.

```
def somme_entiers(n : int) -> int:
    """ A completer. """
    s = 0
    for k in range(1, n + 1):
        s = s + k
    return s
```

- **1.1.** Déterminer la valeur renvoyée par l'appel de `somme_entiers(5)` en complétant le tableau suivant, qui détaille les valeurs successives prises par les variables `k` et `s` au fur et à mesure des passages dans la boucle `for`.

i	k_i	s_i
0	n'existe pas encore	0
1	1	1
2	2	3
3		
4		
5		

- **1.2.** Que se passe-t-il pour $n \leq 0$?
- **1.3.** Quel est le sens à donner à la valeur renvoyée par l'appel `somme_entiers(n)` ? Modifier la documentation de la fonction pour préciser ce qu'elle renvoie.

Question 2. S'inspirer de la fonction précédente pour écrire une fonction `factorielle(n: int) -> int`, où `n` est un entier naturel, et qui renvoie sa factorielle définie par :

$$\begin{cases} n! = 1 \times 2 \times \cdots \times n & \text{si } n > 0, \\ 0! = 1. \end{cases}$$

Vérifiez bien que votre fonction est correcte dans le cas où $n = 0$.

Exercice 3 – Avec instructions conditionnelles

Question 1. Écrire une fonction `nb_couples(n : int) -> int`, où `n` est un entier naturel, renvoyant le nombre de couples d'entiers (x, y) tels que

$$0 \leqslant x \leqslant 5, \qquad 0 \leqslant y \leqslant 5, \qquad x^2 + y^2 \leqslant n.$$

Pensez, comme toujours, aux bonnes pratiques.

Question 2. Tester la fonction `nb_couples` sur plusieurs valeurs de `n`. Que doit-on observer lorsque `n` grandit ?

Exercice 4 – Des étoiles plein les yeux

Écrire des fonctions prenant en argument un nombre de lignes `n` ($n = 5$ dans les exemples ci-dessous) et permettant l'affichage les motifs suivants avec le nombre de lignes imposé. On pourra éventuellement se souvenir de la syntaxe dite *par*

compréhension en observant le résultat de l'instruction `"*" * 3 + " " * 2` dans la console.

Question 1. Écrire une fonction `rectangle` qui affiche le motif suivant avec un nombre de lignes choisi. Quel est le type de la valeur renvoyée ?

```
★ ★ ★ ★ ★
★ ★ ★ ★ ★
★ ★ ★ ★ ★
★ ★ ★ ★ ★
★ ★ ★ ★ ★
```

Question 2.

```
★ ★ ★ ★ ★
★ ★ ★ ★
★ ★ ★
★ ★
★
```

Question 3. A faire à la maison pour s'entraîner

```
★
★ ★
★ ★ ★
★ ★ ★ ★
★ ★ ★ ★ ★
```

Question 4. A faire à la maison pour s'entraîner

```
★ ★ ★ ★ ★
  ★ ★ ★ ★
    ★ ★ ★
      ★ ★
        ★
```

Question 5. A faire à la maison pour s'entraîner

```
* * * * * * * * *
  * * * * * * * *
    * * * * * *
      * * * *
        * *
          *
```

Exercice 5 – Calculer un maximum

On considère dans cet exercice que l'on dispose d'une fonction `f(x : float) -> float` définie pour $n \in \mathbb{N}$ On souhaite écrire une fonction `maximum` renvoyant la plus grande des valeurs parmi $\{f(0), \dots, f(n)\}$ pour `n` un entier naturel donné.

Question 1. Dessiner le diagramme entrée-sortie de la fonction `maximum`.

Question 2. On suppose que la fonction f vérifie :

$$f(0) = 5, \quad f(1) = 2, \quad f(2) = 3, \quad f(3) = 7, \quad f(4) = 1, \quad f(5) = 4.$$

Dérouler l'exécution de l'algorithme envisagé pour l'appel `maximum(f, 5)`. On pourra essayer de faire un tableau de suivi de variables.

Question 3. Écrire la fonction `maximum(f, n : int) -> float` en respectant toutes les bonnes pratiques vues en cours (signature, préconditions, documentation, choix des noms, lisibilité). Attention, le nom `max` existe déjà par défaut en Python.

Question 4. Déterminer la plus grande valeur parmi les $f_1(k)$ pour $k \in [\![0, 100]\!]$, où

$$f_1 : x \longmapsto -(x-42)^2 + 1000.$$

Quel est l'entier k pour lequel ce maximum est atteint ? Vérifier votre réponse à l'aide d'un programme Python. Il faudra définir la fonction `f1`.

Question 5. Même question pour $f_2(x) = 100 - |x - 35|$. On rappelle que la fonction valeur absolue se nomme `abs` en Python.

Question 6. Même question pour

$$f_3(x) = \begin{array}{ll} x & \text{si } x \leq 50, \\ 100 - x & \text{sinon.} \end{array}$$

Question 7. En vous inspirant de la fonction précédente, écrire une fonction `maximum_atteint_en(f, n : int) -> tuple[int, int]` renvoyant un couple d'entiers $(k, f(k))$ où $0 \leqslant k \leqslant n$ soit l'un des entiers tel que $f(k)$ est maximal parmi $\{f(0), \dots, f(n)\}$. *La fonction ne devra utiliser qu'une seule boucle* `for`.

Question 8. Déterminer la valeur maximum parmi $\{f(0), \dots, f(200)\}$ ainsi que l'une de ses localisations pour $f_4(x) = 50 - min(|\, x - 20\,|, |\, x - 60\,|)$

Question 9. Copier et modifier les fonctions précédentes pour écrire les fonctions `minimum` et `minimum_atteint_en`.

Exercice 6 – Les nombres parfaits

On dit qu'un entier naturel non nul n est un nombre *parfait* s'il est égal à la moitié de la somme de ses diviseurs positifs.

- Par exemple, 6 est un nombre parfait car les diviseurs positifs de 6 sont 1, 2, 3, 6 et $1 + 2 + 3 + 6 = 12 = 2 \times 6$.
- Par contre, 8 n'est pas un nombre parfait car les diviseurs positifs de 8 sont 1, 2, 4, 8, mais $1 + 2 + 4 + 8 = 15 \neq 2 \times 8$.

Question 1. Écrire une fonction `est_parfait(n : int) -> bool` renvoyant le booléen `True` si `n` est un nombre parfait et `False` sinon.

Question 2. Écrire une fonction `affiche_parfaits` qui afficher dans la console tous les nombres parfaits compris entre 1 et n. Quel est le type de sa sortie ?

1.2 Boucle `while`

Exercice 7 – Premier positif

Soit f une fonction à valeurs réelles. Écrire une fonction `premier_positif(f)` qui renvoie le premier entier $k \in \mathbb{N}$ tel que $f(k) > 0$

Exercice 8 – Une histoire de lions

Question 1. Dans une réserve naturelle, une population de lions est suivie par des biologistes. Au début de l'année 2026, la réserve compte 120 lions. On suppose que, grâce à de bonnes conditions environnementales, la population augmente de 8 % par an.

- **1.1.** Écrire la relation de récurrence mathématique de la suite (p_n), p_n étant le nombre de lions au début de l'année n (l'année 0 est l'année 2026).
- **1.2.** Écrire une fonction `depasse` qui calcule le nombre d'années à partir duquel la population de lions dépassera une limite `lim` donnée.
- **1.3.** Au bout de combien d'année la population dépassera 1000 lions ?

Exercice 9 – Approximation d'un racine carrée

L'utilisation de la fonction `sqrt` du module `math` est interdite. Si a est un réel

strictement positif, on peut démontrer que la suite définie par :

$$\begin{cases} u_0 = a, \\ \forall n \geqslant 0, \quad u_{n+1} = \dfrac{u_n^2 + a}{2u_n} \end{cases} \qquad \text{converge vers } \sqrt{a}.$$

Question 1. Calculer u_4 pour $a = 2$, puis comparer à une valeur approchée de $\sqrt{2}$.

Question 2. Écrire une fonction `suite_u(a : float, n : int) -> float`, où `n` est un entier naturel, renvoyant la valeur u_n.

Question 3. Écrire une fonction `approx_rac_carree(a : float, c : float) -> float`, où $a \in \mathbb{R}^+$ et $e \in \mathbb{R}^{+*}$, calculant une valeur approchée de $\sqrt{a}$ à c chiffres significatifs. Pour s'arrêter à c chiffres significatifs, on utilisera le test normalisé $\dfrac{|u_{n+1} - u_n|}{|u_n|} \leqslant 10^{-c}$.

Question 4. Tester votre fonction en essayant d'obtenir une valeur approchée de $\sqrt{7}$ à au moins 6 chiffres significatifs. Vous pourrez valider votre code en comparant avec la valeur renvoyée par la fonction `sqrt` présente dans le module `math` que vous pourrez utiliser après avoir rajouté, en dehors de la fonction, la ligne

```
from math import sqrt
```

Remarque –

En réalité, faire des comparaisons entre nombres flottants et créer des cas d'arrêts très précis pour ce genre de situations est une chose complexe, à cause des erreurs d'arrondis et de la représentation binaire des flottants. Nous en reparlerons à la fin de l'année.

On rappelle que :

- `a//b` pour a et $b \neq 0$ deux `int` renvoie le le quotient de la division euclidienne de a par b.
- `a % b` pour a et $b \neq 0$ deux `int` renvoie le reste **signé modulo** b de la division euclidienne de a par b.

Par exemple,

- `25//3` renvoie 8 et `25%3` renvoie 1 puisque :

$$25 = 3 \times 8 + 1 \quad \text{avec} \quad 0 \leqslant 1 < 8.$$

- mais attention, `(-25)//3` renvoie -8 et `(-25)%8` renvoie -1 alors que, la division euclidienne mathématique impose un reste positif et donc

$$-25 = 3 \times (-9) + 2 \quad \text{avec} \quad 0 \leqslant 2 < 8.$$

`%` et `//` effectuent donc la division euclidienne sans se préocupper des signes, et rajoute les signes ensuite sur le quotient et le reste.

Exercice 10 – La mystérieuse suite de Syracuse

On considère la suite de Syracuse définie par $u_0 = a$, où a est un entier naturel non nul, et, pour tout $n \in \mathbb{N}$, par :

$$u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair,} \\ 3u_n + 1 & \text{sinon.} \end{cases}$$

Question 1. Préciser les valeurs de u_n pour $n \in \{1, 2, 3, 4, 5\}$ dans le cas où $a = 6$.

Question 2.

Question 3. Écrire une fonction `terme_syracuse` qui, pour un rang n choisi, calcule le terme de rang n de la suite de Syracuse dont le terme initial est également choisi. On respectera toutes les bonnes pratiques (signature, préconditions, documentation)

Question 4. Tester votre code en calculant la valeur de u_5 lorsque $a = 6$ et en vérifiant à la main

Question 5. Une célèbre conjecture, jamais démontrée à ce jour, affirme que cette suite finit toujours par atteindre la valeur 1, quel que soit son premier terme a, puis boucle sur les valeurs $1, 4, 2, 1, 4, 2, \ldots$.
Par exemple, pour $a = 6$, les premières valeurs de la suite sont :

$$u_0 = 6,\ u_1 = 3,\ u_2 = 10,\ u_3 = 5,\ u_4 = 16,\ u_5 = 8,\ u_6 = 4,\ u_7 = 2,\ \boxed{u_8 = 1}, \ldots$$

Écrire une fonction `syracuse_premier_1(a : int) -> int`, où $a \in \mathbb{N}^*$, renvoyant le plus petit entier naturel n tel que $u_n = 1$.

Question 6. La tester pour $a = 127$.

Question 7. Tester pour plusieurs autres valeurs et constater que la conjecture semble vraie (si vous savez la prouver, vous êtes riche. [a]

a. L'entreprise japonaise Bakuage offre un prix de 120 millions de yens (environ 750 000 euros) à quiconque réussira à démontrer la conjecture de Syracuse.

Exercice 11 – Algorithme d'Euclide

Si a et b sont deux entiers, on rappelle que $\text{pgcd}(a,b)$ est le plus grand diviseur **positif** qui est commun aux deux entiers a et b. Il peut se calculer grâce à l'algorithme d'Euclide.

- Si a et b sont tous les deux nuls, $\text{pgcd}(a,b)$ n'est pas défini
- Si $b = 0$, tout nombre divise b et alors $\text{pgcd}(a,0) = |a|$ car $|a|$ est le plus grand diviseur positif de a. De manière symétrique, si $a = 0$, $\text{pgcd}(0,b) = |b|$
- Sinon, on calcule le reste r de la division euclidienne de a par b et on peut démontrer que $\text{pgcd}(a,b) = \text{pgcd}(b,r)$; on remplace donc la valeur de a par celle de b, et la valeur de b par celle de r.

On itère le procédé jusqu'à obtenir $b = 0$ et on retombe sur le cas $\text{pgcd}(a,0) = |a|$

Question 1. On prend $a = 192$ et $b = 138$. Faire fonctionner à la main l'algorithme d'Euclide en complétant le tableau suivant. La première étape est déjà remplie, 54 étant le reste de la division euclidienne de 192 par 138.

a	b	r
192	138	54
138	54	...
...	...	...
...	...	...
...	...	...
...	...	...

Préciser la valeur de $\text{pgcd}(192, 138)$.

Question 2. On souhaite écrire l'algorithme d'Euclide sous la forme d'une fonction `pgcd(a : int, b : int) -> int`. Le code suivant, à compléter, vous est donné :

```
def pgcd(a : int, b : int) -> int:
    """Renvoie le pgcd de a et b."""
    assert(...)
    while ...:
        r = a % b
        a = ...
        b = ....
    return a
```

Question 3. Tester votre fonction sur des cas simples pour lesquels vous savez calculer rapidement à la main le PGCD

Question 4. Utiliser cette fonction pour calculer pgcd(51940, 6201).

Question 5. Écrire une fonction `premiers_entre_eux` qui indique si deux nombre sont premiers entre eux. On pensera à toutes les bonnes pratiques.

Exercice 12 – Persistance multiplicative

On se donne un entier naturel n. Si n s'écrit avec plusieurs chiffres en vbase 10, alors on calcule le produit de ses chiffres, ce qui donne un nouvel entier naturel. On recommence ce procédé sur le nombre obtenu jusqu'à obtenir un nombre à un seul chiffre.
Par exemple, en partant de $n = 377$, on trouve successivement :

- $3 \times 7 \times 7 = 147$ à la première étape ;
- $1 \times 4 \times 7 = 28$ à la deuxième étape ;
- $2 \times 8 = 16$ à la troisième étape ;
- $1 \times 6 = 6$ à la quatrième et dernière étape.

On appelle *persistance* de l'entier n le nombre d'étapes nécessaires pour le réduire à un seul chiffre. Sur notre exemple, la persistance de 377 vaut donc 4. Si n est déjà formé d'un seul chiffre, alors sa persistance vaut 0.

Question 1. Écrire une fonction `prod_chiffres(n : int) -> int`, où n est un entier naturel, renvoyant le produit des chiffres (en base 10) de n. *On remarquera que* `n%10` *donne le chiffre des unités de* n*, et que* `n//10` *donne le nombre obtenu en enlevant ce dernier chiffre.*

Question 2. Tester votre fonction sur le nombre 377

Question 3. Écrire une fonction `persistance(n : int) -> int`, où n est un entier naturel, et qui renvoie sa persistance.

Question 4. Calculer la persistance de 277777788888899.

Question 5. Une question légitime consiste à rechercher des entiers naturels ayant une persistance élevée. Une conjecture célèbre émet l'hypothèse que *la persistance maximale que l'on peut obtenir est seulement égale à* 11.

5.1. Écrire une fonction `max_persistance(n : int) -> int` qui calcule les persistances de tous les nombres inférieurs ou égaux à n et renvoie la persistance maximale trouvée.

5.2. Utiliser la fonction pour essayer de vérifier la conjecture. *Les calculs deviennent bien sûr de plus en plus longs quand* n *augmente...*

Question 6. Question exploratoire pour ceux qui aiment les mathématiques : nous avons étudié la persistance en base 10. Que dire sur la persistance en base 2 ? Et dans d'autres bases ?

Exercice 13 – Des nombres bien rangés

Pour finir, cet exercice nettement plus difficile s'adresse à celles et ceux d'entre vous les plus à l'aise en programmation. L'énoncé est extrait des phases qualificatives 2017 du concours de programmation **Google Code Jam** et vous fera en outre travailler votre anglais.

Tatiana likes to keep things tidy. Her toys are sorted from smallest to largest, her pencils are sorted from shortest to longest and her computers from oldest to newest. One day, when practicing her counting skills, she noticed that some integers, when written in base 10 with no leading zeroes, have their digits sorted in non-decreasing order. Some examples of this are 8, 123, 555, and 224488. She decided to call these numbers tidy. Numbers that do not have this property, like 20, 321, 495 and 999990, are not tidy.

She just finished counting all positive integers in ascending order from 1 to N. What was the last tidy number she counted ?

Comme vous l'aurez compris, un entier d'écriture décimale $c_{n-1}c_{n-2}\dots c_1c_0$ est dit « bien rangé » lorsque :

$$c_{n-1} \leqslant c_{n-2} \leqslant \cdots \leqslant c_1 \leqslant c_0.$$

Par exemple, l'entier 3356889 est bien rangé, mais 3354889 ne l'est pas.

L'objectif est d'écrire une fonction `tidy(n : int) -> int`, où n est un entier naturel non nul, renvoyant le plus grand entier bien rangé inférieur ou égal à n. Les fonctions à écrire pour le concours Google Code Jam doivent obéir à un souci d'efficacité en fournissant leurs réponses dans un temps raisonnable. Par exemple, le résultat de l'appel `tidy(1122220001222556799)` devra être obtenu instantanément, en moins d'une seconde. Enfin, nous n'utiliserons aucune structure de données complexe, comme les listes, les chaînes de caractères ou les dictionnaires, ce qui aurait pu être approprié mais fera l'objet de TPs ultérieurs.