
Types de base, listes et boucles

Ceci est le corrigé

Question 1. Qu'affiche le bout de code suivant ?

```
for i in range(3,15,2):  
    print(i)
```

Solution.

L'affichage de ce bout de code (sauts de ligne inclus) est le suivant :

```
3  
5  
7  
9  
11  
13
```

Question 2. Vers quelle valeur s'évalue l'expression `1 / 2` et quel est son type? (On supposera le résultat exact).

Solution.

Cette expression s'évalue vers `0.5` de type flottant.

Question 3. Vers quelle valeur s'évalue l'expression `int(str(int(4.2)))` et quel est son type?

Solution.

L'expression `int(4.2)` s'évalue vers l'entier 4 donc `str(int(4.2))` s'évalue vers la chaîne de caractères "4" et enfin `int(str(int(4.2)))` s'évalue vers l'entier 4.

Question 4. Quelle propriété usuelle de l'addition n'est pas respectée entre flottants en Python? (Si vous n'avez pas le mot, donnez un exemple)

Solution.

L'associativité.

Question 5. Écrire un code Python qui définit une liste `pairs` et la remplit avec les entiers pairs entre 2 (inclus) et 40 (inclus), à l'aide d'une boucle.

Solution.

On propose le code suivant :

```
pairs = []  
for i in range(20):  
    pairs.append(2*(i+1))
```

Remarque

Une autre solution est :

```
pairs = []  
for i in range(1, 21):  
    pairs.append(2*i)
```

Remarque

Encore une autre solution :

```
pairs = []  
for i in range(2, 41, 2):  
    pairs.append(i)
```

Remarque

Et encore une autre :

```
pairs = []  
for i in range(2, 41):  
    if i % 2 == 0:  
        pairs.append(i)
```

Question 6. On suppose qu'on dispose d'une variable `liste` qui contient une liste de nombres. Écrire un code Python qui affiche les carrés des nombres contenus dans `liste`.

Solution.

On utilise l'opérateur `**` pour les puissances en Python :

```
for nombre in liste:
    print(nombre**2)
```

Question 7. Écrire un code python qui demande à l'utilisateur·ice de rentrer un entier n , puis qui affiche les n premiers termes de la suite $(u_k)_{k \in \mathbb{N}}$ définie par $u_0 = 1$ et pour tout $k \in \mathbb{N}$, $u_{k+1} = \frac{k^2 + 1}{u_k^2 + 1} + 1$.

Solution.

On utilise la fonction `int` pour convertir la chaîne entrée par l'utilisateur·ice en un entier.

On utilise une boucle `for` sur `range(n)` pour répéter un code n fois.

```
n = int( input( "Entrez un nombre : " ) )
u = 1
for k in range(n):
    print(u)
    u = (k**2 + 1)/(u**2 + 1) + 1
```

La $k^{\text{ième}}$ itération de notre boucle affiche la valeur de u_k (contenue dans la variable `u`), puis calcule la valeur de u_{k+1} et la stocke dans la variable `u`.

Question 8. Vers quoi s'évaluent les expressions suivantes et quel est leur type? (Vous justifierez chaque réponse)

- `(1 < 2) == (5 == 5)`
- `((2 < 1) or (not(2 < 2))) and (not((2 >= 2) or (-1 < 3)))`
- `(len("bon" + "jour") == ([3, 7] + [2, 5])[1]) or (1 + "bonjour")`
- `[3, 5] * 2`
- `[3, 5] * True`

Solution.

- `(1 < 2) == (5 == 5)` s'évalue vers `True`, de type booléen, car `(1 < 2)` et `(5==5)` s'évaluent tous

les deux vers True.

• $((2 < 1) \text{ or } (\text{not}(2 < 2))) \text{ and } (\text{not}((2 \geq 2) \text{ or } (-1 < 3)))$ s'évalue vers False, de type booléen, car son membre droit s'évalue vers False.

En effet, $(2 \geq 2)$ s'évalue vers True donc $(2 \geq 2) \text{ or } (-1 < 3)$ s'évalue vers True donc $\text{not}((2 \geq 2) \text{ or } (-1 < 3))$ s'évalue vers False.

• La troisième expression s'évalue vers **True**, de type booléen. En effet, $\text{len}(\text{"bon"} + \text{"jour"})$ s'évalue vers 7 tout comme $([3, 7] + [2, 5])[1]$, donc la partie gauche du **or** s'évalue vers **True**. L'opérateur **or** en python étant paresseux, Python ne tente pas d'évaluer l'expression à droite du **or**.

• $[3, 5] * 2$ s'évalue vers $[3, 5, 3, 5]$, qui est une liste (type list).

• $[3, 5] * \text{True}$ s'évalue la liste $[3, 5]$. (Il y a une conversion implicite de **True** vers 1)

Test de divisibilité par 7

Voici un test qui détermine si un nombre entier naturel $n \in \mathbb{N}$ est divisible par 7 :

- On enlève à l'écriture décimale de n le chiffre des unités
- On retranche au nombre ainsi obtenu, le double du chiffre qu'on a enlevé
- On réitère le procédé jusqu'à ce que l'on obtiennent un nombre a entre -20 et 20
- On conclut en regardant si a est divisible par 7

Exemple 1 : $31976 \rightarrow 3197 - 2 \times 6 = 3185 \rightarrow 318 - 2 \times 5 = 308 \rightarrow 30 - 2 \times 8 = 14$

Comme 14 est divisible par 7, on en déduit que 31976 l'est également.

Exemple 2 : $311 \rightarrow 31 - 2 \times 1 = 29 \rightarrow 2 - 2 \times 9 = -17$

Comme -17 n'est pas divisible par 7, on en déduit que 311 ne l'est pas.

Question 9. Écrire un code Python qui demande à l'utilisateur·ice d'entrer un entier n **positif** et qui affiche

- le message `Divisible par 7` si l'entier entré est divisible par 7
- le message `Pas divisible par 7` si l'entier entré n'est pas divisible par 7

Votre code doit utiliser le processus décrit plus haut. En particulier, aucun test de comparaison avec le résultat d'un % ou // ne sera toléré pour cette question. (Mais vous pouvez utiliser % et // si ce n'est pas pour une comparaison).

Solution.

On propose le code suivant :

```
n = int(input("Entrez un nombre :"))
while (n < -20) or (n > 20):
    unites = n % 10
    non_unites = n // 10
    n = non_unites - 2*unites
if n in [-14, -7, 0, 7, 14]:
    print("Divisible par 7")
else:
    print("Pas divisible par 7")
```

Remarque

Le sujet précise qu'il s'agit de nombres positifs.

Défi : (en bonus bien sûr)

Le problème suivant est posé sans questions intermédiaires, vous pouvez le traiter par la méthode de votre choix, néanmoins on s'interdira d'utiliser sans les avoir préalablement défini soi-même des fonctions fournies par python qui rendraient le problème trivial, telles que les fonctions qui renvoient le *pgcd* et *ppcm* de deux entiers.

Question 10. Écrire une fonction python qui prend en entrée deux entiers strictement positifs a et b et affiche la réduction de la fraction $\frac{a}{b}$ avec la phrase suivante (ici sur un exemple) : "42 / 8 = 21 / 4". On ne fera pas attention au fait que les fractions de la forme 4/1 sont maladroites.