ICC, partie Programmation — Série d'exercices 3

GC/MX, 4 octobre 2019

À faire individuellement ou par petits groupes de deux ou trois.

Exercice 1. Boucles et fonctions (I)

(a) Écrivez une fonction **`calculate_volume`** qui calcule le volume d'une sphère d'un rayon donné. Quel est (ou quels sont) le ou les paramètres que vous devez déclarer? Quel sera le type de retour de cette fonction?

(b) Appelez votre fonction pour calculer le volume d'une sphère de rayon 1, 5 et 10, et affichez les valeurs sur la console avec **`print(...)`**.

(c) Changez votre code pour afficher le volume des sphères avec des rayons de 1 à 20 (avec un incrément de 1) en utilisant une boucle **`while`**, puis une boucle **`for-in`**.

Exercice 2. Boucles et fonctions (II)

(a) Écrivez une fonction **`is_prime`** qui détermine si l'argument de type **`int`** qu'on lui donne est un nombre premier et qui retourne un **`bool`**. Vous aurez besoin d'utiliser une boucle.

Aide: pour tester si un nombre n est divisible par m, vous pouvez utiliser cette condition: **`n % m == 0`**.
L'opérateur **`%`** renvoie ici la valeur de n modulo m, c'est-à-dire le reste de la division en nombres entiers de n par m. Si vous trouvez un nombre m différent de 1 et de n pour lequel le reste est zéro, alors n n'est pas premier, comme m en est un diviseur.

Indice (à lire si vous êtes bloqué·e): vous devrez tester dans une boucle tous les diviseurs potentiels. Si vous trouvez un diviseur différent de 1 et de n, vous savez que n n'est pas premier. Vous pouvez confirmer qu'il est premier uniquement après obtenu confirmation qu'aucun autre diviseur que 1 et n existe.

(b) Testez votre code et vérifiez que 2, 3 et 29 sont bien premiers, mais pas 4, 9 ou 27.

(c) Avec une boucle, écrivez du code qui trouve les 10 premiers nombres premiers.

Indice: vous devrez gérer deux variables de boucle: l'une pour rechercher des nombres premiers de plus en plus grands, l'autre pour compter le nombre de nombres premiers trouvés.

Exercice 3. Compréhension de code

(a) Depuis la page Moodle du cours, copiez et collez ce code dans un nouveau fichier.

Que fait chaque ligne de ce programme? Comment (et pourquoi) est-ce que la boucle fonctionne?

Indice: la méthode **`find()`** fait la même chose que la méthode **`index()`** de la semaine passée, sauf qu'elle retourne **`-1`** quand elle ne trouve pas le string recherché plutôt qu'une erreur.

```
1  def show_word(word: str) -> None:
2      print(f"Mot: '{word}'")
3
4  my_string = "This is fun!"
5
6  last_space_position: int = -1
7  next_space: int = my_string.find(" ")
8
9  while next_space != -1:
10     word: str = my_string[last_space_position + 1:next_space]
11     show_word(word)
12     last_space_position = next_space
13     next_space = my_string.find(" ", last_space_position + 1)
14
15 last_word: str = my_string[last_space_position + 1:]
16 show_word(last_word)
```

(b) Insérez une espace de trop entre deux mots dans la valeur de **`my_string`**, par exemple:

```
"This  is fun!"  # deux espaces entre This et is
```

Que fait le programme actuel? Modifiez-le pour ne pas afficher les mots «vides» en introduisant une condition avec un `if` dans le code de la fonction **`show_word`**.