TD_boucles_for

February 4, 2021

Très important : Dans ce TD, ne pas hésiter à recourir aux outils de "debuggage" pour bien comprendre les boucles et trouver les problèmes : * Thonny en mode debug * Pythontutor en ligne * Pythontutor dans un Notebook jupyter

1 Exercice 1 : niveau facile

1. Pour chaque séquence ci-dessous, écrire l'instruction python permettant de la générer (on utilisera la fonction `range`) :
   - 2, 3, 4, 5, 6, 7, 8
   - 0, 4, 8, 12, 16, 20, 24
   - Les caractères composant une chaîne de caractères appelée `maChaine`. (Par exemple si `maChaine = "informatique"`, on veut obtenir la séquence `'i'`, `'n'`, `'f'`, `'o'`, `'r'`, `'m'`, `'a'`, `'t'`, `'i'`, `'q'`, `'u'`, `'e'`)
   - Les indices des caractères composant une chaîne de caractères appelée `maChaine`. (Par exemple si `maChaine = "informatique"`, on veut obtenir la séquence 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
   - les chiffres de 1 à n inclus (n étant le nombre de caractères d'une chaîne de caractère appelée `maChaine`). (Par exemple si `maChaine = "informatique"`, on veut obtenir la séquence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
2. Parcourir chaque séquence ci-dessus à l'aide d'une boucle `for` afin d'afficher chaque valeur de la séquence

2 Exercice 2 : niveau facile

Le code suivant permet d'afficher **10 fois** le message "Pour progresser en programmation, la pratique est le plus important". Malheureusement, plusieurs erreurs se sont glissées dans le code, corrigez-les et tester !

[ ]:
```
for i in len(10)
print(Pour progresser en programmation, la pratique est le plus important)
print(Pour progresser en programmation, la pratique est le plus important)
print(Pour progresser en programmation, la pratique est le plus important)
print(Pour progresser en programmation, la pratique est le plus important)
print(Pour progresser en programmation, la pratique est le plus important)
print(Pour progresser en programmation, la pratique est le plus important)
print(Pour progresser en programmation, la pratique est le plus important)
```

```
print(Pour progresser en programmation, la pratique est le plus important)
print(Pour progresser en programmation, la pratique est le plus important)
print(Pour progresser en programmation, la pratique est le plus important)
```

3 Exercice 3 : niveau facile

1. Ecrire une boucle `for` permettant d'**afficher** la table de multiplication par 7. L'affichage attendu est donné ci-dessous :

7 14
21
28
35
42
49
56
63
70

2. Modifier le code ci-dessus afin que l'affichage soit celui donné ci-dessous :

7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70

3. "Encapsuler" le code précédent dans une fonction `tableMultiplication` qui :
   - prend en **paramètre** l'entier dont on veut la table de multiplication
   - **affiche** la table de multiplication

Un exemple d'appel de la fonction `tableMultiplication` est donné ci-dessous :

[7]:
```
tableMultiplication(9)
```

```
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
```

```
9 * 9 = 81
9 * 10 = 90
```

4. Écrire une fonction `plusieursTablesMultiplication` qui :
   - prend deux entiers a et b en **paramètre**
   - **affiche** toutes les tables de multiplication pour les entiers de a à b.

Un exemple d'appel de la fonction `plusieursTablesMultiplication` est donné ci-dessous : *(Consigne: utiliser la fonction* ***`tableMultiplication`*** *pour écrire le code de* ***`plusieursTablesMultiplication`****)*

[9]:
```
plusieursTablesMultiplication(3,5)
```

```
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Si vous avez suivi les consignes, vous avez utiliser la fonction `tableMultiplication` pour écrire le code de `plusieursTablesMultiplication`

5. **A faire avec le professeur :** Dans le corps de la fonction `plusieursTablesMultiplication`, remplacer l'appel de la fonction `tableMultiplication` par son code. Tester la fonction `plusieursTablesMultiplication`

On a maintenant une boucle `for` (interne) imbriquée dans une boucle `for` (externe). Exécuter ce code en mode debug ou avec pythontutor.

4 Exercice 4 : niveau intermédiaire

[ ]:
```
def nombreCaracteres(chaine):
    compteur = 0
    for ...
```

Compléter le code de la fonction `nombreCaracteres` :

- qui prend en **paramètres** une chaîne de caractères
- qui **renvoie** la longueur de cette chaîne de caractères

L'utilisation de la fonction `len` est b ien sûr interdire !
(Indication : Parcourir la chaîne de caractères et incrémenter une variable `compteur` à chaque tour de boucle)

Des exemples d'appel de cette fonction sont donnés ci-dessous :

[6]:
```
nombreCaracteres("Python")
```

[6]: 6

[7]:
```
nombreCaracteres("Javascript")
```

[7]: 10

5 Exercice 5 : niveau intermédiaire

Ecrire le code de la fonction `compterOccurence`:

- qui prend en **paramètres** :
  - une chaîne de caractères `chaine`
  - un caractère `c`
- et qui **renvoie** le nombre de fois que le caractère apparaît dans `chaine`.

(Indication : Parcourir `chaine` et incrémenter une variable `compteur` à chaque fois qu'on rencontre le cacractère recherché `c`)

Des exemples d'appel de cette fonction sont donnés ci-dessous :

[2]:
```
compterOccurence("Guido Van Rossum", 'o')
```

[2]: 2

[3]:
```
compterOccurence("Linus Torvalds", 'z')
```

[3]: 0

6 Exercice 6 : niveau intermédiaire

[5]:
```
def rechercheOccurence(chaine, c):
    for lettre in chaine:
        if lettre == c:
            return True
    return False
```

1. Exécuter **à la main (sans ordinateur)** l'appel `rechercheOccurence("Stallmann", "a")`. Combien de fois "êtes-vous entré" dans la boucle `for` ?
2. Exécuter **à la main (sans ordinateur)** l'appel `rechercheOccurence("Richard", "b")`. Combien de fois "êtes-vous entré" dans la boucle `for` ?
3. Pourquoi n'obtient-on pas la même valeur dans les 2 questions précédentes (préciser quelle instruction du code de la fonction est responsable de cette différence)
4. Ecrire le code de la fonction `rechercheIndice`

(Indication : On peut s'inspirer du code de la fonction `rechercheOccurence` *mais comme on recherche un indice, il faudra parcourir les indices de* `chaine` *plutôt que les caractères de* `chaine`*...)*

[ ]:
```
def rechercheIndice(chaine,c):
    """
    Description de la fonction : renvoie l'indice de la première occurence du␣
 ↪caractère c dans chaine
    chaine (str) : chaine de caractère dans laquelle la recherche s'effectue
    c (str) : caractère recherché
    return (int ou None) : indice du caractère c danc chaine (None si le␣
 ↪caractère c n'apparaît pas dans chaine)
    """
    # A compléter
```

Quelques exemples d'appels de la fonvtion `rechercheIndice` :

[9]:
```
rechercheIndice("Stallmann", "a")
```

[9]: 2

[10]:
```
rechercheIndice("Richard", "b")
```

7 Exercice 7 : niveau intermédiaire

On considère la fonction `mystere` ci dessous.

[17]:
```
def mystere(n):
    s = 0
    n = str(n)
    for i in range(len(n)):
        s = s + int(n[i])
    return s
```

1. Exécuter **à la main (sans ordinateur)** l'appel `mystere(63847)`. Pour cela, il faut dresser un tableau regroupant les valeurs successives de `i` et `s`
2. Vérifier en exécutant l'appel `mystere(63847)` en mode debug sous Thonny ou avec python-tutor
3. Maintenant que l'on a découvert le fonctionnement de la fonction, plutôt que "mystere" comment pourrait-on mieux nommer cette fonction ?
4. Ecrire le code d'une fonction `produitChiffres` qui :
   - prend un entier n en **paramètre**
   - **renvoie** le produit de tous les chiffres de n

(Indication : on pourra s'inspirer du code de `mystere`)

Des exemples d'appel de cette fonction sont donnés ci-dessous :

[2]:
```
produitChiffres(4132)
```

[2]: 24

[3]:
```
produitChiffres(9045)
```

[3]: 0

8 Exercice 8 : niveau intermédiaire

Ecrire le code d'une fonction `verlan` qui :

- prend une chaîne de caractères en **paramètre**
- **renvoie** la chaîne de caractères écrite à l'envers

*(Indication : on pourra partir d'une chaîne de caractère vide et la construire au fur et à mesure qu'on parcourt la chaîne passée en paramètre. L'opérateur de **concaténation** peut être utile.... (voir `/premiere/bloc6/initiation_a_python/Cours_Variables_et_types_simples_en_python.ipynb`))*

Des exemples d'appel de cette fonction sont donnés ci-dessous :

[28]:
```
verlan("Python")
```

[28]: 'nohtyP'

[11]:
```
verlan("! reggubed tse'c remmargorP")
```

[11]:
```
"Programmer c'est debugger !"
```

9 Exercice 9 : niveau intermédiaire

[ ]:
```
def repetition(chaine,n):
    m = ""
    for c in chaine:

```

Compléter le code de la fonction `repetition` qui permet de **renvoyer** une chaîne de caractère dont chaque caractère est répété n fois. Ainsi dans l'appel ci-dessous, chaque lettre du mot Python est répétée 4 fois

[26]:
```
repetition("Python",4)
```

[26]:
```
'PPPPyyyytttthhhhoooonnnn'
```

10 Exercice 10 : niveau intermédiaire

> Un **nombre premier** est un entier naturel qui n'admet que deux diviseurs distincts entiers et positifs (1 et lui-même). Par définition 1 n'est pas *premier*.
> Ainsi, parmi les entiers inférieurs à 30 : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 sont des nombres premiers

Ecrire une **fonction prédicat** `est_premier` qui prend un entier en **paramètre** et qui **renvoie** un booléen selon que ce nombre est premier ou pas. Tester votre fonction à l'aide de ce site. N'oublier pas de tester votre fonction pour l'entier 1 et 2....

Indications :

- *Pour savoir si x est un nombre premier, une méthode consiste à tester la divisibilité de x par tous les entiers inférieurs à x.*
- *L'opérateur `%` (voir `/premiere/bloc6/initiation_a_python/Cours_Variables_et_types_simples_en_py.ipynb`) est utile pour tester la divisibilité...)*

Des exemples d'appel de cette fonction sont donnés ci-dessous :

[9]:
```
est_premier(8)
```

[9]:
```
False
```

[10]:
```
est_premier(11)
```

[10]:
```
True
```