

## Outils Info. TP4: Fonctions

### Rendu : sous forme d'archive

À la fin, vos réponses seront rendues dans une archive zip : cette archive contiendra un fichier `compte-rendu.txt` (avec votre **nom**, **groupe**, et les réponses aux questions précédées d'une apostrophe) et vos fichiers pythons.

Pour faire une **archive zip** de votre TP pour le rendu, exécutez, dans le terminal :

```
cd ~/outils-info/  
zip -r tp4.zip tp4/
```

### Important : mise en place

- travaillez dans un dossier dédié au TP, lui même dans `~/outils-info`,
- créez un fichier `compte-rendu.txt` pour écrire la date, vos noms et les réponses aux questions précédées d'une apostrophe,
- mais vous pouvez travailler directement dans le dossier `tp4` (sans sous-dossier par exercice).

## Exercice 1 – Introduction et téléchargement

Dans ce TP, des fichiers de base vous sont donnés et vous allez utiliser la bibliothèque `qtido`. Il vous faudra donc télécharger une archive contenant les fichiers et une archive contenant la bibliothèque (`qtido.py` et un exemple pour vérifier que la bibliothèque fonctionne). Dans les fichiers donnés, certaines parties ne doivent pas être modifiées.

**Q1)** Téléchargez et décompressez (extrayez le contenu) les archives `ressources-tp4.zip`, ainsi que la bibliothèque graphique. Il vous sera demandé par la suite de copier certains de ces fichiers en les renommant.

## Exercice 2 – Bien comprendre les fonctions

Dans ce TP, vous allez écrire des fonctions. Voici un exemple de **définition** d'une fonction :

- dont le nom `bout_a_bout`,
- qui reçoit en **paramètre** deux valeurs (que l'on supposera ici être des chaînes de caractères),
- qui *affiche* la longueur de chaque chaîne et la somme des longueurs,
- qui **renvoie** la concaténation des chaînes (calculée avec l'opérateur `+`),

```
1 def bout_a_bout(ch1, ch2):  
2     print("L1 :", len(ch1))  
3     print("L2 :", len(ch2))  
4     l = len(ch1) + len(ch2)  
5     print("Total :", l)  
6     return ch1 + ch2  
7     print("Jamais")
```

**'Q2)** Que se passerait-il si on mettait cette définition dans un fichier Python et qu'on l'exécutait (le lançait) ?

**'Q3)** Que se passerait-il si on ajoutait, à la fin du fichier, sans indentation, les lignes suivantes ?

```
1 a = "Toto"
2 b = bout_a_bout(a, "Titi")
```

Q4) Que se passerait-il si on ajoutait, en plus, les lignes suivantes ?

```
1 c = bout_a_bout(a, b)
2 print()
3 print(b)
4 print(c)
```

Q5) Testez vos hypothèses en créant un fichier `boutabout.py` avec la définition de la fonction et les lignes ajoutées dans les questions précédentes.

## Exercice 3 – Fonction sans valeur de retour

Q6) Copiez le fichier fourni `couleur-triangles.py` en tant que `ex3.py`. Attention à bien copier le fichier et à ne pas modifier le fichier d'origine.

Q7) Le programme utilise une fonction `triangle` qui n'existe hélas pas. Modifiez `ex3.py` pour y définir la fonction `triangle`. Cette fonction doit recevoir 7 paramètres : la fenêtre dans laquelle tracer, et les coordonnées  $x, y$  des trois sommets du triangle à tracer.

Aide :

- utilisez la fonction `ligne(f, x1, y1, x2, y2)` de la bibliothèque `qtido` pour tracer les 3 cotés du triangle.
- la fonction `triangle` ne doit pas renvoyer de valeur

## Structure typique d'un programme

Sur le modèle de l'exercice précédent, il vous est demandé de bien respecter la structure typique d'un programme, qui est la suivante (exemple qui trace 3 carrés) :

```
1
2 # 1) importation de bibliothèque(s)
3 from qtido import *
4
5 # 2) définition des fonctions
6 def carre(fen, x, y, taille):
7     rectangle(fen, x, y, x+taille, y+taille)
8
9 # 3 et 4) programme principal
10 f = creer(600, 500)
11 carre(f, 100, 100, 100)
12 carre(f, 400, 100, 100)
13 carre(f, 225, 300, 150)
14 exporter_image(f, "toto.png")
15 attendre_pendant(f, 1000)
```

## Exercice 4 – Fonctions composées

Q8) Copiez le fichier fourni `volume.py` en tant que `ex4.py`. Attention à bien copier le fichier et à ne pas modifier le fichier d'origine.

Toutes les questions de cet exercice doivent être réalisées dans ce même fichier. Quand on parle ici de paramètres, ce sont des paramètres de fonctions. À la fin de votre fichier python, vous devez ajouter des appels à vos fonctions (et affichez le résultat) pour être sûr que vos fonctions sont correctes.

**Q9)** Écrivez une fonction `carre` qui reçoit un nombre en paramètre et renvoie son carré.

**Q10)** Écrivez une fonction `cube` qui reçoit un nombre en paramètre et renvoie son cube.

**Q11)** Écrivez une fonction `logbase` qui calcule le logarithme en base  $b$  de  $v$ , défini mathématiquement par  $\log_b(v) = \frac{\log_{10}(v)}{\log_{10}(b)}$ . En pratique, `logbase` est une fonction (Python) qui reçoit deux nombres `v` et `b` en paramètre et renvoie le rapport des `log10` (du module `math`) de ses deux paramètres.

**Q12)** Écrivez une fonction `aire_disque` qui renvoie l'aire d'un disque à partir d'un rayon reçu en paramètre, à l'aide de la constante `pi` du module `math`. L'aire du disque est mathématiquement définie par  $\mathcal{A}_{disque}(r) = \pi \cdot r^2$ .

**Q13)** Écrivez une fonction `volume_cylindre` qui renvoie le volume d'un cylindre calculé à partir d'un rayon et d'une hauteur reçus en paramètre (le volume du cylindre est l'aire du disque multipliée par la hauteur). IMPORTANT : il est interdit d'utiliser `pi` dans cette fonction.

**Q14)** Dans votre programme principal, calculez et affichez la valeur correspondant à la phrase suivante : « le cube du log (en base  $z$ ) du volume d'un cylindre de hauteur 2 et de rayon 10, où  $z$  est l'aire d'un disque de rayon 42.42 ».

## Exercice 5 – Fonctions Simples ...

Attention : Ne confondez pas les paramètres des fonctions avec ce que l'on demande de taper à l'utilisateur avec `input`.

**Q15)** Dans un fichier `ex5.py`, définissez une fonction `produit(a,b)` qui affiche `Appel de la fonction produit` et renvoie le produit de ses paramètres.

**Q16)** Que se passe-t-il en exécutant `python3 ex5.py` ?

**Q17)** Définissez une fonction `somme(a,b)` qui affiche `Appel de la fonction somme` et renvoie la somme de ses paramètres.

**Q18)** Le résultat de l'exécution `python3 ex5.py` a-t-il changé ?

**Q19)** Dans le programme principal, affichez maintenant `Programme principal`.

**Q20)** Que se passe-t-il en exécutant `python3 ex5.py` ?

**Q21)** Dans le programme principal, définissez maintenant deux variables `var1` et `var2` et initialisez-les respectivement aux valeurs `10` et `1.5`.

**Q22)** Complétez le programme principal pour qu'il calcule et affiche le produit et la somme des deux variables, en utilisant les fonctions définies dans les questions précédentes.

**Q23)** Que se passe-t-il en exécutant `python3 ex5.py` ?

**Q24)** Toujours dans le programme principal, définissez et initialisez deux nouvelles variables et affichez leurs somme et leurs produit.

## Exercice 6 – ... et Conditions

Vous allez maintenant modifier le programme pour qu'il affiche le produit ou la somme des valeurs tapées par l'utilisateur. Par exemple, si on exécute `python3 ex6.py` et tape `produit 2.2 3`, le programme doit afficher le produit `6.6` et si on tape `somme 2.2 1`, le programme doit afficher la somme `3.2`.

**Q25)** Dans un fichier `ex6.py`, réécrivez les fonctions `somme` et `produit` de l'exercice précédent.

**Q26)** Dans le programme principal, faites que le programme demande à l'utilisateur de taper une commande (comme par exemple `produit 2.2 3`) et qu'il appelle une fonction appelée `calculer` avec comme paramètre la ligne tapée par l'utilisateur. Cette fonction sera définie dans la question suivante (et ne renvoie aucune valeur).

**Q27)** Définissez la fonction `calculer` qui doit découper en morceaux (selon les espaces) la chaîne reçue en paramètres. Ensuite, la fonction doit afficher `multiplication` si le premier morceau est `produit` ou `addition` si le premier morceau est `somme`.

**Q28)** Qu'affiche le programme si on l'exécute et tape `produit 11 100` ?

**Q29)** Modifiez la fonction pour qu'elle affiche, en plus, le produit des valeurs si le premier paramètre est `produit` ou la somme si le premier paramètre est `somme`.

**Q30)** Qu'affiche le programme si on l'exécute et tape `somme 11 100` ?