Python 08 - Une Fonction, une Tâche

I – Transfert d'informations : arguments et paramètres

Lors d'un **appel,** les valeurs qu'on envoie en entrée se nomment les **arguments**

En utilisant la **déclaration** mise en mémoire, on stocke les **arguments** dans les **paramètres**.

DOC 1 : Appels, déclaration et transfert d'informations

```
01  def nouveau_stylo(ecriture:str, fond:str, largeur:int) -> Turtle:   # DECLARATIION
02    feutre = trt.Turtle()
03    feutre.color(ecriture)
04    feutre.fillcolor(fond)
05    feutre.pensize(largeur)
06    feutre.speed(5)
07    return feutre
08
09  s1 = nouveau_stylo( "red", "black" , 4 )   # APPEL
```

Avant l'appel, les paramètres n'existent pas et ne contiennent donc rien.

Lors de cet appel (L09 vers L01) :

ecriture reçoit **"red",**

fond reçoit **"black" et**

largeur reçoit **4.**

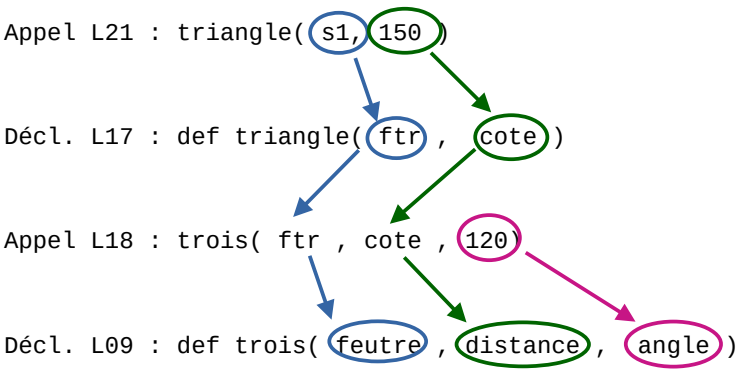
Après la rencontre du **return** en ligne 07, la réponse de la fonction va alors être récupérée dans **s1**,

On peut également réaliser des appels plus compliqués où une fonction appelle une autre fonction en lui fournissant comme argument ce qu'elle a elle-même reçu dans l'un de ses paramètres.

DOC 2 : Transfert du contenu des paramètres en tant qu'argument à une autre fonction

```
09  def trois(feutre, distance, angle):
10    feutre.forward(distance)
11    feutre.left(angle)
12    feutre.forward(distance)
13    feutre.left(angle)
14    feutre.forward(distance)
15    feutre.left(angle)
16
17  def triangle(ftr, cote):
18     trois(ftr, cote, 120)
19
20  s1 = nouveau_stylo("green","black", 4)
21  triangle(s1, 150)
```

On a donc les transferts ci-dessous :



Important à comprendre : l'observation des transferts permet de comprendre que, lors de cette succession d'appels :

→ On envoie **s1** à **triangle()** qui le stocke dans **ftr** de **trois()**

→ On envoie ce **ftr** à **trois()** qui le stocke dans **feutre**

Lors du déroulement de cet appel, **s1, ftr** et **feutre** caractérise donc le même objet.

De la même façon, 150 est stocké dans **cote**, puis dans **distance**,

II – Une fonction, une tâche

Puisqu'une fonction peut transférer le contenu de ses propres paramètres, on peut ne pas concevoir des fonctions longues : il suffit de déléguer certaines parties de l'action à d'autres fonctions.

Chaque fonction ne doit pas faire plus de 10-20 lignes.

Sa lecture doit être le plus claire possible.

Comment réaliser une fonction réalisant une tâche moyenne ?

Ses instructions incorporent des appels à des fonctions réalisant des tâches simples.

Comment réaliser une fonction réalisant une tâche complexe ?

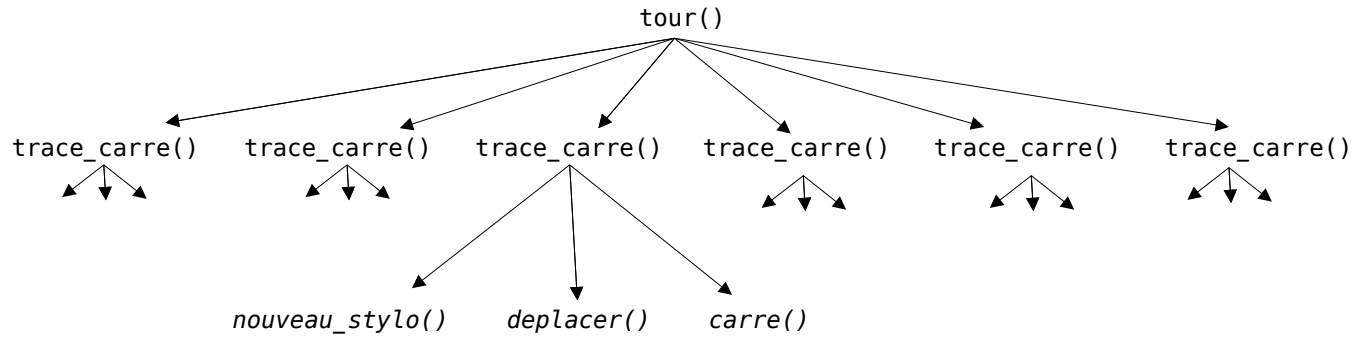
Ses instructions incorporent des appels à des fonctions réalisant des tâches moyennes.

Exemple :

La fonction **tour()** réalise une tâche complexe : tracer une tour.

Pour réaliser cette tâche, elle utilise plusieurs appels à la fonction **trace_carre()**, qui lance à son tour des appels à d'autres fonctions, ect...

DOC 3 : L'encapsulation des tâches



Bien entendu, chaque fonction transfère souvent l'un de ses propres paramètres à l'une des fonctions qu'elle appelle.

DOC 1 : Appels, déclaration et transfert d'informations

```
01  def nouveau_stylo(ecriture:str, fond:str, largeur:int) -> Turtle:   # DECLARATIION
02    feutre = trt.Turtle()
03    feutre.color(ecriture)
04    feutre.fillcolor(fond)
05    feutre.pensize(largeur)
06    feutre.speed(5)
07    return feutre
08
09  s1 = nouveau_stylo( "red" , "black" , 4 )   # APPEL
```

DOC 2 : Transfert du contenu des paramètres en tant qu'argument à une autre fonction

```
09  def trois(feutre, distance, angle):
10    feutre.forward(distance)
11    feutre.left(angle)
12    feutre.forward(distance)
13    feutre.left(angle)
14    feutre.forward(distance)
15    feutre.left(angle)
16
17  def triangle(ftr, cote):
18     trois(ftr, cote, 120)
19
20  s1 = nouveau_stylo("green","black", 4)
21  triangle(s1, 150)
```

On a donc les transferts ci-dessous :

```
Appel L21 : triangle( s1, 150 )

Décl. L17 : def triangle( ftr ,  cote )

Appel L18 : trois( ftr , cote , 120)

Décl. L09 : def trois( feutre , distance ,  angle )
```

DOC 3 : L'encapsulation des tâches