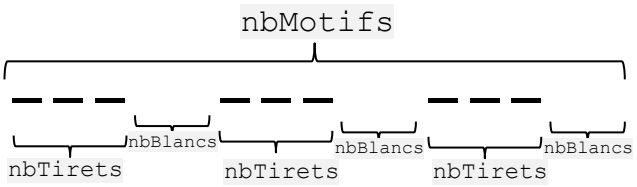
NSI	Langages et programmation Fonctions et conditions	TP – **Fonctions et conditions**

Objectifs :

- ⇨ Découvrir la notion d'exécution conditionnelle et la mettre en œuvre
- ⇨ Découvrir la notion de fonction et la mettre en œuvre

I – Echauffement



On se propose de créer un programme pour afficher une ligne séparatrice.
Cette ligne discontinue est composée de tirets et d'espaces comme présenté sur la figure ci-contre.

La ligne est donc composée de `nbTirets` fois le caractère tiret `-` suivi de `nbBlancs` fois le caractère espace. Le motif doit se répéter `nbMotifs` fois. Le motif présenté sur la figure ci-dessus a été obtenu avec `nbTirets = 3`, `nbBlancs = 3` et `nbMotifs = 2`.

Q1 : Réaliser le programme permettant d'afficher une ligne séparatrice en utilisant les variables citées. Vous pouvez choisir de compléter le programme à trou « Echauffement.py » ou partir de zéro suivant votre niveau de maîtrise (dans ce dernier cas, copier le programme dans son espace personnel et compléter les parties avec des pointillés).

Imaginons que l'on veuille tracer plusieurs lignes séparatrices à différents moments de notre programme. Il est fastidieux de copier-coller le code précédent à chaque fois. De plus si on change la présentation (remplacer les tirets par des étoiles par exemple), il faudra changer à tous les endroits du programme où on se sert de ce code.

Pour utiliser le même bout de code à différents endroits d'un programme, les langages permettent de définir des **fonctions**.

II – Les fonctions et les procédures

1) Qu'est-ce qu'une fonction ?

Une fonction[1] est une entité informatique qui encapsule une portion de code (une séquence d'instructions) effectuant un traitement spécifique bien identifié (asservissement, tâche, calcul, etc.) relativement indépendant du reste du programme, et qui peut être réutilisé dans le même programme, ou dans un autre.
Une fonction possède une quantité quelconque de variable d'entrée (éventuellement aucune) et une seule variable de sortie (le « résultat » de la fonction).

2) Syntaxe en python

Les fonctions en python se présentent de la façon suivante :

Ce mot-clé indique que l'on définit une fonction

Ne pas oublier les « : »

```
def nom_fonction(argument1, argument2, …) :
    .
    .
    Instructions…
    .
    .
    return resultatDeLaFonction ;
```

Ce mot-clé indique que l'on sort de la fonction et que l'on revient au programme appelant en renvoyant comme résultat l'expression qui suit l'instruction `return`.

[1] On parle aussi de *routine* ou de *sous-programme* en informatique.

Remarques :

- Comme pour une boucle la portion de code qui correspond à la fonction est indentée par rapport à la colonne du `def`.
- **La définition d'une fonction doit forcément précéder son utilisation** dans le programme. Du coup on déclare toutes les fonctions au début du programme (juste après les imports de modules et avant le programme principal).

Exemple :

```
def somme(a, b) :
    c = a + b
    return c
```

La fonction `somme` prend comme argument 2 nombres et renvoie la somme des deux.

L'appel d'une fonction s'effectue simplement en écrivant son nom avec entre parenthèse les arguments. S'il n'y a pas d'argument, on met simplement « `()` » pour indiquer qu'il s'agit d'une fonction.

Tout se passe dans le programme comme si la portion où la fonction est appelée était remplacée par le résultat de la fonction :

```
x = 5
y = 23
print("Le résultat est :" + somme(x,y))
```

est équivalent à :

```
x = 5
y = 23
print("Le résultat est :" + 28)
```

3) Qu'est-ce qu'une procédure ?

Une **procédure** est tout simplement le nom que l'on donne à une **fonction qui ne renvoie rien** (En python elle renvoie « None »). Si elle ne renvoie rien, cela ne signifie pas pour autant qu'elle ne fait rien, mais qu'elle a des **effets de bords**[2] qui nous intéressent. C'est le cas de la fonction `print()` que nous avons déjà vu et qui ne renvoi rien, mais qui a pour effet de bord d'afficher quelque chose sur la console.
Lorsque python arrive à la fin du bloc de code d'une fonction sans rencontrer le mot-clé « `return` », il sort tout de même de la fonction (ou plutôt de la procédure), et renvoie « `None` ». En d'autres termes, l'instruction « `return` » est optionnelle pour les procédures.

4) Définition d'une maison

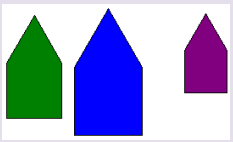
On utilise maintenant le code source ci-contre qui permet de dessiner une maison de largeur « `largeur` » et de couleur « `couleur` ». Ce code se trouve déjà dans le fichier « Question2.py ».

```
pencolor("black")
fillcolor(couleur)
down()
begin_fill()
forward(largeur)
left(90)
forward(largeur)
left(30)
forward(largeur)
left(120)
forward(largeur)
left(30)
forward(largeur)
end_fill()
left(90)
up()
```

[2] En programmation, un *effet de bord* correspond à tous les effets que peut avoir une fonction qui ne font pas partie du résultat qu'elle renvoie.

Q2 :

1) Ecrire une procédure « `traceMaison` » qui prend deux arguments : la largeur de la maison (comme premier argument) et sa couleur (comme deuxième argument).
2) En utilisant la procédure précédente, tracer 3 maisons :
   - ✓ Une maison bleue de 80 pixels de large en position 0,0
   - ✓ Une maison violette (« `purple` ») de 50 pixels de large en position 130, 50
   - ✓ Une maison verte de 65 pixels de large en position -80, 20



Aide : la procédure `goto(x,y)` du module `turtle` permet de se déplacer au point de coordonnées x, y.

3) A la toute fin du programme, rajouter la ligne « `traceMaison("orange", 60)` » pour tracer une maison orange de 60 pixels de large. Que remarque-t-on lorsque l'on exécute le programme ? Pourquoi ?

III – Conditions

1) Le problème du mölkky

Au jeu de mölkky (jeu de quilles finlandais), chaque joueur marque à son tour de jeu entre 0 et 12 points (valeur qui dépend du numéro et du nombre de quilles renversées), qui viennent s'ajouter à son score précédent. Le premier à atteindre le score de 50 gagne. Mais quiconque dépasse le score cible de 50 points revient immédiatement à 25 points.
On souhaite écrire un programme demandant un score, un nombre de points marqués et qui affiche le nouveau score et indique éventuellement la victoire.
Tant que le score de 50 n'est pas dépassé, on peut facilement compter les points, mais comment afficher la victoire *uniquement si le score obtenu est de 50*. De même comment gérer le cas où on dépasse ce nombre ?

On a besoin pour ce faire d'utiliser des instructions de contrôle de flux pour exécuter une portion de programme uniquement si une condition et vérifiée. Voyons rapidement comment cela se fait en python.

2) Syntaxe des instructions conditionnelles

3 instructions permettent de faire de l'exécution conditionnelle en python : **`if`**, **`elif`** et **`else`**.

```
if expression logique :
    bloc d'instructions
    si l'expression logique est vraie
elif deuxième expression logique :
    bloc d'instructions
    si la deuxième expression logique est vraie et la première est fausse
else :
    bloc d'instructions
    si toutes les expressions logiques sont fausses
suite du programme
```

Remarques :

- L'expression logique correspond exactement à ce que l'on a vu pour l'instruction `while`. Elle est évaluée par python au moment du passage par le `if`. Si elle vaut `True` le bloc d'instructions juste en dessous est exécuté. Si elle vaut `False` il est sauté et le bloc juste en-dessous du `else` est exécuté.
- On a encore, comme pour les boucles ou les fonctions des blocs d'instructions délimités par leur indentation (décalage par rapport au début de la ligne).
- Le bloc `elif` (pour `else if`) est optionnel et il peut y en avoir un nombre quelconque.

- Le `else` et le bloc d'instruction qui va avec est optionnel. Le bloc qui va avec le `else` n'est exécuté que si toutes les autres conditions (du `if` et des `elif`) sont fausses.
- On peut utiliser un `if` seul. Dans ce cas si la condition est vrai le bloc sous le `if` est exécuté, si elle est fausse, il ne se passe rien et le programme continu juste après le bloc d'instructions du `if`.

Exemple :

```
age = int(input("Quel age avez-vous ? "))
if age < 15:
     print("Vous êtes trop jeune pour la conduite accompagnée.")
elif age < 18:
     print("Il faut patienter encore un peu pour avoir le permis...")
else:
     print("Vous pouvez passer le permis de conduire !")
print("Fin du programme")
```

3) Application

Q3 :

1) Ecrire un programme pour le mölkky qui demande le score actuel, le nombre de point marqué et qui affiche le nouveau score et indique, le cas échéant, si il y a victoire.
2) On veut maintenant un programme qui gère les scores de 2 joueurs. Les scores commencent à 0 et chacun à tour de rôle indique le nombre de points qu'il a marqués. Le programme s'arrête lorsqu'un des deux joueurs (que l'on indiquera) a gagné. Ecrire ce programme en partant du code précédent.
3) Améliorer le programme précédent en détectant une entrée incorrecte des points marqués (supérieur à 12 ou inférieur à 0) et en faisant en sorte que le programme redemande au joueur le nombre de points marqués jusqu'à ce que la réponse soit correcte. Implémentez cette fonctionnalité sous la forme d'une fonction `demande_gain()` qui ne renvoie le gain saisi par l'utilisateur que lorsque celui-ci est correct.

PROLONGEMENT :

Ecrire un programme qui choisit un nombre entier aléatoire entre 1 et 20 et qui demande à l'utilisateur de trouver ce nombre en un minimum d'essais. A chaque proposition du joueur, le programme indique si le nombre proposé est plus grand ou plus petit que le nombre que l'ordinateur a choisi. Quand c'est gagné, le programme l'affiche et se termine.
Vous pouvez ensuite modifier le programme pour qu'il affiche le nombre d'essais à la fin.

Aide : Pour générer un nombre entier aléatoire, on peut utiliser la fonction `randint` du module `random`.