Definitions

1. *Bloc d'instructions*

Une **fonction** permet d'encapsuler un ***bloc d'instructions*** et de lui donner un nom.

Exemple de fonction:

```
1. def somme(a, b):
2.     s = a + b
3.     return s
```

2. *Executer la fonction*

On peut ensuite exécuter ce bloc en utilisant ce nom. On dit qu'on **appelle** cette fonction.

Exemple de fonction:

```
>>> somme(12, 5)
17
```

En pratique: Où écrit-on le script de la fonction? A quel endroit de l'IDE appelle t-on la fonction?
Ici, l'exemple montre que l'on appelle la fonction depuis l'interpréteur de l'IDE Pyzo.
Mais cela peut aussi être depuis le programme lui-même. Dans ce cas, on place les fonctions au début du script, après/avant la déclaration des variables globales. Puis vient le programme du **`main`**.

3. *Paramètres et valeur de retour*

Une fonction peut avoir un ou plusieurs **paramètres** qui permettent de ***transmettre des valeurs*** au bloc d'instruction. A l'intérieur du bloc, les paramètres sont traités comme des variables.

Une fonction peut retourner une valeur, après l'instruction **return.**
Cette valeur retournée peut être **un nombre, une chaine de caractère, un objet …**

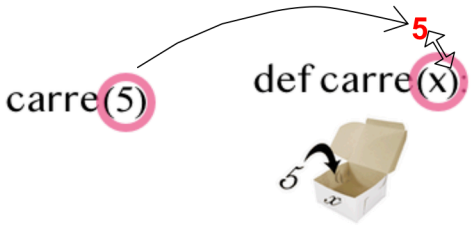
- creation

```
def nom(paramètres):
    bloc
```

- appel

```
nom(arguments)
```

Lors de l'appel, on dit que l'on passe un argument: ici on fait par exemple x = 5



Sur la feuille d'exercices/TD/TP, on a vu:

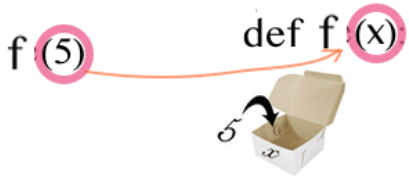
4. *Fonction math et fonction python: retourner un nombre*

On peut programmer une fonction pour qu'elle retourne la valeur y = f(x)

Exemple: soit la fonction mathématique f : $x \rightarrow 3*x^2+2$

Le script python correspondant sera:

```
1. def f(x):
2.   return 3*x**2+2
```



Pour exécuter la fonction avec **x=5**, on fait: **f(5)**

Et cela retourne … $3*5^2+2$, soit 77

5. *retourner une chaine de caractères*

Avec l'exemple vu en TP:

```
1. def etoiles(nb):
2.     return '*' + nb * ' ' + '*'
```

On peut appeler cette fonction plusieurs fois pour obtenir l'image suivante:

```
>>> etoiles(1)
* *
>>> for i in range(3): # rappel: for execute 3 fois le bloc
. . . . print(etoiles(1))
* *
* *
* *
```

Question: Quelle figure donne le script suivant?

```
1. for i in range(4): # rappel: range(4)=> i est le variant 0,1,2,3
2.     print(etoiles(i))
```

6. *Procedure*

Une fonction sans valeur de retour est une **procédure**. voir TP Turtle.

Portée des variables

Lorsqu'une fonction utilise des **variables**, celles-ci sont normalement propres à la fonction et ne sont pas accessibles à l'extérieur de celle-ci. On dit qu'il s'agit de **variables locales**, par opposition aux **variables globales,** du programme principal.

Exemple: si on définit une variable a avant la définition de la fonction, celle-ci sera différente du a de la fonction somme:

```
1. a = 10
2. def somme(a, b):
3.     s = a + b
4.     return s
>>> somme(1,0)
1
>>> a
```

```
10
>>> b
NameError: name 'b' is not defined
```

Python permet d'affecter ***à l'intérieur d'une fonction***, une **variable globale**. Pour cela, il faut déclarer la variable comme étant globale au début de la fonction:

```
1. a = 10
2. def somme(b):
3.     global a
4.     a = a + b
5.     return a
>>> somme(1)
11
```

Modifier une variable globale depuis une fonction s'appelle un **effet de bord**. C'est considéré comme une *mauvaise pratique*, mais parfois indispensable.

Portée de la fonction, modules

Une fonction peut être placée dans un autre fichier. On a alors:
- un fichier principal (le programme main)
- un ou plusieurs fichiers annexes (modules)

Le programme principal doit alors faire référence aux modules.
L'une des manières est la suivante:

```
1. import module
2. from module import *
3. import module as alias
```

Il est recommandé toutefois de ne charger que les fonctions utiles du **module**:

```
1. from module import fonction
```

Seul l'un de ces programmes est exécuté. Cela revient à écrire, dans le **shell**:

```
>>> from programme import *
```

Exemple: supposons que le programme suivant soit dans le fichier *test.py*

```
# contenu du fichier test.py
a = 10
def somme(b):
    s = a + 10
    return s
```

On execute le programme dans le shell:

```
>>> from test import *
>>> a
10
```

```
>>> somme(0)
20
```

Celui-ci (le programme) occupe alors une place privilégiée dans l'espace de noms de l'interpréteur python: c'est le **main**.

On peut mettre les instructions du programme principal dans un bloc sous:

```
10. if __name__ == '__main__':
11.     fen1 = Tk()
12.     # …
```

Conclusion:

Une fonction est un morceau de code qui porte un nom et qui s'exécute lorsqu'on l'appelle. L'avantage est d'avoir un code plus court, lisible, mais aussi d'encapsuler les variables et de limiter leur portée à celle de la fonction.

Dans un projet plus grand, les fonctions peuvent être mises dans des modules (des fichiers séparés). On doit alors les importer pour avoir une extension du langage. Certains modules très utiles: math, turtle, random, …