Définition

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nom_Fonction(parametres):
    instructions
    return resultats
```

Définition

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nom_Fonction(parametres):
    instructions
    return resultats
```

Remarques

- Le nom d'une fonction la désigne de manière unique. En Python, il est impossible de définir deux fonctions du même nom. Une nouvelle définition écrase l'ancienne.

Définition

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nom_Fonction(parametres):
    instructions
    return resultats
```

Remarques

- Le nom d'une fonction la désigne de manière unique. En Python, il est impossible de définir deux fonctions du même nom. Une nouvelle définition écrase l'ancienne.
- Les valeurs des ***paramètres*** ne seront connues qu'à l'appel de la fonction.

Définition

La syntaxe Python pour la définition d'une fonction est la suivante :

```
def nom_Fonction(parametres):
    instructions
    return resultats
```

Remarques

- Le nom d'une fonction la désigne de manière unique. En Python, il est impossible de définir deux fonctions du même nom. Une nouvelle définition écrase l'ancienne.
- Les valeurs des ***paramètres*** ne seront connues qu'à l'appel de la fonction.
- Idéalement, une fonction effectue une seule tâche.

Appel d'une fonction

- Pour appeler une fonction, on doit préciser les valeurs prises par les paramètres appelés ***arguments***.

Appel d'une fonction

- Pour appeler une fonction, on doit préciser les valeurs prises par les paramètres appelés ***arguments***.

```
def somme(x,y):
        return x+y
somme(5,3)
```

Appel d'une fonction

- Pour appeler une fonction, on doit préciser les valeurs prises par les paramètres appelés ***arguments***.

```
def somme(x,y):
        return x+y
somme(5,3)
```

Ici on appelle la fonction somme avec les arguments 5 et 3 : x prend la valeur 5 et y prend la valeur 3 dans la fonction

Valeurs par défaut des paramètres

On peut préciser des valeurs par défaut pour certains paramètres de la fonction. Dans ce cas, il n'est pas obligatoire de donner leurs valeurs en appelant leur fonction.

Valeurs par défaut des paramètres

On peut préciser des valeurs par défaut pour certains paramètres de la fonction. Dans ce cas, il n'est pas obligatoire de donner leurs valeurs en appelant leur fonction.

```
def fct(a, b, x=0, y=0, z=0):
    return a, b, x, y, z
```

Valeurs par défaut des paramètres

On peut préciser des valeurs par défaut pour certains paramètres de la fonction. Dans ce cas, il n'est pas obligatoire de donner leurs valeurs en appelant leur fonction.

```
def fct(a, b, x=0, y=0, z=0):
    return a, b, x, y, z
```

Ici les deux premiers arguments sont obligatoires, pas les suivants.

Valeurs par défaut des paramètres

On peut préciser des valeurs par défaut pour certains paramètres de la fonction. Dans ce cas, il n'est pas obligatoire de donner leurs valeurs en appelant leur fonction.

```
def fct(a, b, x=0, y=0, z=0):
    return a, b, x, y, z
```

Ici les deux premiers arguments sont obligatoires, pas les suivants.

```
print(fct(1,1))#donne (1,1,0,0,0)
print(fct(1,1,2))#donne (1,1,2,0,0)
print(fct(1,1,2,4,5))#donne (1,1,2,4,5)
print(fct(1,1,x=1,z=2))#donne (1,1,1,0,2)
```

Valeurs renvoyées

- On sort de la fonction au premier *return* rencontré.

Valeurs renvoyées

- On sort de la fonction au premier *return* rencontré.

```
def test(x):
    if x>20:
        return x+5
    return x-3
```

Valeurs renvoyées

- On sort de la fonction au premier *return* rencontré.

```
def test(x):
    if x>20:
        return x+5
    return x-3
```

Si la valeur de x est > à 20, la fonction renvoie x+5, sinon elle renvoie x-3.

Valeurs renvoyées

- On sort de la fonction au premier *return* rencontré.

```
def test(x):
    if x>20:
        return x+5
    return x-3
```

Si la valeur de x est > à 20, la fonction renvoie x+5, sinon elle renvoie x-3.

- En l'absence de *return*, la fonction renvoie *none* après la dernière ligne d'instruction.

Valeurs renvoyées

- On sort de la fonction au premier *return* rencontré.

```
def test(x):
    if x>20:
        return x+5
    return x-3
```

Si la valeur de x est > à 20, la fonction renvoie x+5, sinon elle renvoie x-3.

- En l'absence de *return*, la fonction renvoie *none* après la dernière ligne d'instruction.

- Il est possible de renvoyer plusieurs valeurs séparées par des virgules. Elle sont renvoyées sous forme d'un tuple.

Exercice

Que fait la fonction suivante pour les différentes valeurs de x proposées ci-dessous ?

```
def ma_fonction(x):
    if x>5 :
        return x,x+3
    elif x==2:
            print("ça vaut 2")
    return x-3
```

- Si x=8 :

Exercice

Que fait la fonction suivante pour les différentes valeurs de x proposées ci-dessous ?

```
def ma_fonction(x):
    if x>5 :
        return x,x+3
    elif x==2:
            print("ça␣vaut␣2")
    return x-3
```

- Si x=8 :
  On sort par le premier return, la valeur renvoyée est (8,11)

Exercice

Que fait la fonction suivante pour les différentes valeurs de x proposées ci-dessous ?

```
def ma_fonction(x):
    if x>5 :
        return x,x+3
    elif x==2:
            print("ça␣vaut␣2")
    return x-3
```

- Si x=8 :
  On sort par le premier return, la valeur renvoyée est (8,11)
- Si x=4 :

Exercice

Que fait la fonction suivante pour les différentes valeurs de x proposées ci-dessous ?

```
def ma_fonction(x):
    if x>5 :
        return x,x+3
    elif x==2:
            print("ça␣vaut␣2")
    return x-3
```

- Si x=8 :
  On sort par le premier return, la valeur renvoyée est (8,11)
- Si x=4 :
  On sort par le dernier return sans vérifier les conditions des tests, la valeur renvoyée est 1

Exercice

Que fait la fonction suivante pour les différentes valeurs de x proposées ci-dessous ?

```
def ma_fonction(x):
    if x>5 :
        return x,x+3
    elif x==2:
            print("ça␣vaut␣2")
    return x-3
```

- Si x=8 :
  On sort par le premier return, la valeur renvoyée est (8,11)
- Si x=4 :
  On sort par le dernier return sans vérifier les conditions des tests, la valeur renvoyée est 1
- Si x=2 :

Exercice

Que fait la fonction suivante pour les différentes valeurs de x proposées ci-dessous ?

```
def ma_fonction(x):
    if x>5 :
        return x,x+3
    elif x==2:
            print("ça vaut 2")
    return x-3
```

- Si x=8 :
  On sort par le premier return, la valeur renvoyée est (8,11)
- Si x=4 :
  On sort par le dernier return sans vérifier les conditions des tests, la valeur renvoyée est 1
- Si x=2 :
  On vérifie la deuxième condition, donc les message « ça vaut 2 » s'affiche puis on sort par le dernier return , la valeur renvoyée est -1

Variables locales et globales

La portée des variables indique quand et comment elles sont accessibles.

- Les variables définies dans une fonction sont appelées **variables locales.** Elles ne peuvent être utilisées qu'à l'intérieur de la fonction qui les a définies.
  Le programme suivant génére deux erreurs car les variables x et y n'existent pas en dehors de la fonction.

```
def ma_fonction(x):
    y=x+2
    return y
print(y)
print(x)
```

```
>>> print(y)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'y' is not defined
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Variables locales et globales

- Les variables définie dans le programme principal, en dehors des fonctions sont dites **globales**. Elles sont utilisables par les fonctions mais non modifiables.Le programme suivant génére deux erreurs car les variables x et y n'existent pas en dehors de la fonction.
  Le programme suivant affiche la valeur 7, la valeur de la variable globale y est utilisée.

```
y=3
def f(x):
    return x+y
print(f(4))
```

Variables locales et globales

- Les variables définie dans le programme principal, en dehors des fonctions sont dites **globales**. Elles sont utilisables par les fonctions mais non modifiables.
  Si on tente de modifier la valeur de y, une erreur est renvoyée.

```
y=3
def f(x):
        y=y+2
    return x+y
```

```
>>> f(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
UnboundLocalError: local variable 'y' referenced before assignment
```