Les fonctions

Cours

Lycée Jean Zay – 21 rue Jean Zay – 63300 Thiers – Académie de Clermont-Ferrand

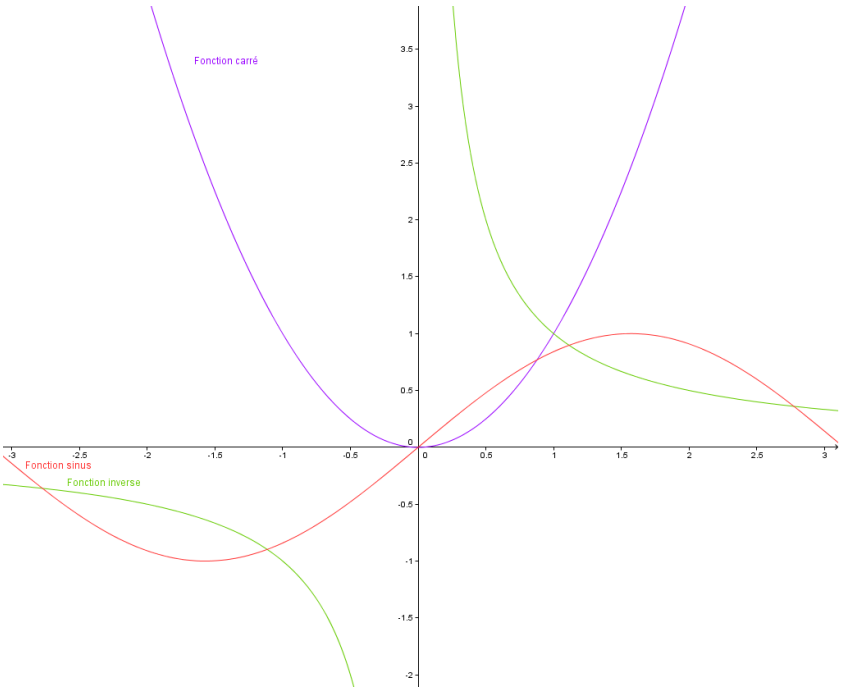

Table des matières

Préambule

Une fonction est une portion de code qui peut être « appelée », exécutée, à n'importe quel endroit d'un code informatique, à partir du moment où elle a été définie, c'est-à-dire lue une fois et mise en mémoire par l'ordinateur. Elle permet d'exécuter une succession d'opérations, prenant comme entrée des arguments et renvoyant en sortie d'autres variables ou arguments.

1 Notion de fonction

Définition d'une fonction en langage Python

Une fonction est une suite d'instructions qui dépend de paramètres, qui est définie par

- son nom,
- ses arguments (éventuels),
- (éventuellement) une valeur de retour communiquée par le programme en fin d'exécution,

et dont la syntaxe en Python est :

```
1
2 def nom_de_la_fonction (argument_1,...,argument_n) :
3     bloc_d_instructions
4     return valeur_de_retour
```

Chaque instruction `nom_de_la_fonction (···)` est un appel de la fonction.

Attention *Nom des fonctions*

Il faut faire attention aux noms des fonctions qui ressemblent à des variable. Par exemple après avoir définit la fonction `Maximum`, écrire `Maximum=Maximum(Liste)` ne stocke pas le Maximum de la liste `Liste`, mais renvoie une erreur car python cherche l'élément `Liste` dans l'objet `Maximum`. Il est donc important de différentier les noms de variable et les noms de fonction.

2 Arguments

Les arguments sont les entrées des fonctions, les variables dont elle va se servir pour exécuter les blocs d'instructions.

2.1 Aucun argument

Les fonctions Python sans arguments et/ou sans valeur de retour ne sont pas à proprement parler des fonctions (au sens mathématique de transformation d'un objet en un autre objet). Elles peuvent néanmoins être utiles lorsqu'une suite d'instructions est répétée fréquemment. Dans d'autres langages, on parlerait de procédures ou de routines.

```
1 def Message():
2     print('Veuillez patienter !')
```

```
>>> Message()
Veuillez patienter
```

2.2 Arguments obligatoires

Si des arguments sont définis dans la fonction, ils sont non-optionnels et seront donc attendus lors de l'appel de la fonction.

```
1 def f(x,y,z):
2     Norme = sqrt(x**2+y**2+z**2)
3     return Norme
```

```
>>> f(1,2,3)
3.7416573867739413

>>> f(1)
Traceback (most recent call last):
File "", line 1 , in
TypeError: f( () missing 2 required positional arguments: ' y
    ' and ' z '
```

Attention *Ordre des arguments*

Il faut faire attention à l'ordre des arguments. Les variables x, y et z ne concernent ici les instructions qui sont à l'intérieur de la fonction.

```
1 x=1
2 y=2
3 z=3
4 def f(x,y,z):
5     Norme = sqrt(x**2+y**2+z**2)
6     return Norme
```

```
>>> f(z,y,x)
3.7416573867739413
```

La fonction va associer la valeur z à x, y à y et x à z !!!

2.3 Arguments optionnels prédéfinis

Il est simple de définir des arguments optionnels. Ils sont nécessairement introduits à la fin de la parenthèse, après les arguments obligatoires. Leur valeur est définie par défaut :

```
1 def f(x,a=0,b=0):
2     return a*x+b
```

```
>>> f(2)
0

>>>f(2,1,1)
3
```

Ainsi, si les arguments ne sont pas définis lors de l'appel de la fonction, ils seront égaux à la valeur de base, ici 0. On veillera à mettre le plus à droite les arguments toujours optionnels. Dans l'exemple ci-dessus, si l'on veut préciser la valeur de b, il faudra aussi préciser a.

2.4 Arguments optionnels en quantité quelconque

On peux définir à l'aide le la commande * une quantité quelconque d'éléments. La fonction attend au moins 1 élément.

```
1 def f(*args):
2     Somme = 0
3     for arg in args:
4         Somme += arg
5     return Somme
```

```
>>> f(2,1)
3

>>>f(2,1,1)
4
```

3 Contenu

3.1 Notion de variable locale/globale

Une fois les arguments définit, il faut indenter le contenu de la fonction. Le contenu doit comporter uniquement les variables locales, c'est à dire celle qui sont définis uniquement pour la fonction, elles "n'existent pas" avant ni après l'exécution de la fonction.

Définition *Variable locale*

Une variable est locale si :
- Elle est présente dans la parenthèse des arguments
- Elle est créée dans une fonction.

Une variable locale n'existe que dans la fonction dans laquelle elle est créée. On fait ce que l'on veut des variables locales tant que l'on reste dans le cadre de la fonction considérée. Elles n'existent plus à la sortie des fonctions.

Définition *Variable globale*

Toutes les variables créées dans le code principal, autrement dit toute variable qui n'est pas créée dans une fonction, est une variable globale que l'on retrouvera dans le « Workspace ».

Le contenu de la fonction comporte alors des suites d'instructions utilisant uniquement des variables locales, si on souhaite utiliser une variable globale, il faut l'indiquer dans les arguments.

3.2 Renvoie d'objets

Si l'on souhaite que la fonction renvoie un résultat, il suffit d'inscrire à la fin : `return Objet`. Si l'on souhaite retourner plusieurs objets, il faut écrire `return [objet_1,objet_2]`.

Attention

Attention, ce que vous mettrez après ce return ne sera pas exécuté, return marque la fin de la fonction.

Attention à ne pas confondre le `return` d'une fonction avec l'instruction `print` :

- `print` ne fait qu'afficher dans l'interpréteur la valeur d'une variable, le résultat d'un calcul... Cette valeur n'est stockée nulle part.
- `return` retourne le résultat de la fonction, qui peut alors être appelée. Il est alors souvent utile d'affecter ce résultat dans une variable. Exemple : si la fonction mafonction a une ligne `return`(...), alors on peut dans le corps du programme écrire x = mafonction(*variables*), et utiliser plus tard la variable x.

3.3 Assignation des renvoies

Lors de l'appel d'une fonction `Fonction` qui renvoie deux objets en sortie, on écrira lors de l'appel de la fonction :

```
a,b=Fonction(...)
```

```
v=Fonction(...)
a=v[0]
b=v[1]
```

```
v=Fonction(...)
a,b=v
```

Ces trois exécutions sont équivalentes !

Attention

ATTENTION : si vous appelez uniquement la fonction `Fonction(...)`, les variables a et b ne seront pas créées!!!

```
1 def f(x):
2     a = 2*x
3     b = 3+x
4     return a,b
```

```
>>>f(2)
(4, 5)

>>>a
Traceback (most recent call last):
  File "", line 1, in <cell line: 1>
    a
NameError: name 'a' is not defined


>>>b
Traceback (most recent call last):
 File "", line 1, in <cell line: 1>
    b
NameError: name 'b' is not defined
```

3.4 Aucun renvoie

Une fonction peut réaliser un travail sans rien renvoyer, par exemple une modification de liste :

```
1 def Ajout(x,L):
2     L.append(x)
```

```
>>> L=[]
>>> Ajout(1,L)
>>> L
[1]
```