Programmation
Python:
Les fonctions, Les modules et les fichiers

Pr. BENDAHMANE

Les Fonctions

Les fonctions dans Python

- Il y a deux sortes de fonctions dans Python.

  > Les fonctions intégrées qui sont fournies par Python - raw_input(), type(), float(), int() ...

  > Les fonctions que nous définissons et utilisons nous même

- On considère le nom des fonctions intégrées comme de "nouveaux" mots réservés (on ne les utilise pas pour nommer des variables)

La définition d'une fonction

- En Python, une fonction est du code réutilisable qui prend un ou des argument(s) en entrée, réalise des actions, et renvoie un ou des résultats
- On définie une fonction à l'aide du mot réservé def
- On appelle une fonction en utilisant son nom, des parenthèses, et un/des argument(s) dans une expression

Créer une fonction

En algorithmique

Fonction SommeCarre (x : réel, y: réel) : réel
variable z : réel
z ← x*x+y*y
retourne (z)
FinFonction

En Python

```
def SommeCarre(x,y) :
    z=x**2+y**2
    return z
```

Créer une fonction

Créer une fonction

En Python, une fonction est définie à l'aide du mot-clé `def` :

Exemple

```
def my_function():
  print("Hello from a function")
```

Appeler une fonction

Pour appeler une fonction, utilisez le nom de la fonction suivi de parenthèses:

Exemple

```
def my_function():
  print("Hello from a function")

my_function()
```

6

Les arguments

Arguments

Les informations peuvent être transmises aux fonctions en tant qu'arguments.

Les arguments sont spécifiés après le nom de la fonction, entre parenthèses. Vous pouvez ajouter autant d'arguments que vous le souhaitez, séparez-les simplement par une virgule.

L'exemple suivant a une fonction avec un argument (fname). Lorsque la fonction est appelée, nous transmettons un prénom, qui est utilisé à l'intérieur de la fonction pour afficher le nom complet:

Exemple

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

7

Les arguments arbitraires

Arguments arbitraires, * args

Si vous ne savez pas combien d'arguments seront passés à votre fonction, ajoutez un `*` avant le nom du paramètre dans la définition de la fonction.

De cette façon, la fonction recevra un *tuple* d'arguments et pourra accéder aux éléments en conséquence:

Exemple

Si le nombre d'arguments est inconnu, ajoutez un `*` avant le nom du paramètre:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

8

Les arguments de mots clés

Arguments de mots clés

Vous pouvez également envoyer des arguments avec la syntaxe *clé = valeur* .

De cette façon, l'ordre des arguments n'a pas d'importance.

Exemple

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Essayez vous-même »

Les arguments de mots clés arbitraires

Arguments de mots-clés arbitraires, ** kwargs

Si vous ne savez pas combien d'arguments de mot-clé seront passés à votre fonction, ajoutez deux astérisques: `**` avant le nom du paramètre dans la définition de la fonction.

De cette façon, la fonction recevra un *dictionnaire* d'arguments et pourra accéder aux éléments en conséquence:

Exemple

Si le nombre d'arguments de mot-clé est inconnu, ajoutez un double `**` avant le nom du paramètre:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Exemples

```
def somme(*args):
    s=0
    for i in args:
        s=s+i
    return s
somme(1,2,3,4,5) # retourne 15.
def exemple2(**kargs):
    print(kargs)
exemple2(i=1,j=2,text="coucou") # affiche {'j': 2, 'i': 1, 'text': 'coucou'}
```

Paramètres par défaut

```
1  def somme(a,b):
2      return a+b
3
4  x=somme(2,4)
5  y=somme(a=3,b=5)
6  print(x,y)
```

Shell

```
6 8
>
```

```
1  def soustraction(a,b=3):
2      return a-b
3
4  x=soustraction(4,2)
5  y=soustraction(4)
6  z=soustraction(6,b=2)
7  #t=soustraction(b=2,7)      ==> Erreur
8  print(x,y,z)
9
```

Shell

```
2 1 4
>
```

Fonction renvoyant plusieurs valeurs (1)

Renvoyer plusieurs valeurs avec return

return peut envoyer plusieurs valeurs simultanément.
La récupération passe par une affectation multiple.

```
#écriture de la fonction
def extreme(a,b):
    if (a < b):
        return a,b
    else:
        return b,a

#appel
x = 10
y = 15
vmin,vmax = extreme(x,y)
print(vmin,vmax)
```

vmin =10 et vmax=15

Remarque : Que se passe-t-il si nous ne mettons qu'une variable dans la partie gauche de l'affectation ?

```
#ou autre appel
v = extreme(x,y)
print(v)
#quel type pour v ?
print(type(v))
```

v = (10, 15)

<class 'tuple'>
v est un « tuple », une collection de valeurs, à voir plus tard.

Fonction renvoyant plusieurs valeurs (2)

Utilisation des listes et des dictionnaires

Nous pouvons aussi passer par une structure intermédiaire telle que la liste ou le dictionnaire d'objets. Les objets peuvent être de type différent, au final l'outil est très souple. (nous verrons plus en détail les listes et les dictionnaires plus loin)

```
#écriture de la fonction
def extreme_liste(a,b):
    if (a < b):
        return [a,b]
    else:
        return [b,a]

#appel
x = 10
y = 15
res = extreme_liste(x,y)
print(res[0])
```

```
#écriture de la fonction
def extreme_dico(a,b):
    if (a < b):
        return {'mini' : a,'maxi' : b}
    else:
        return {'mini' : b,'maxi' : a}

#appel
x = 10
y = 15
res = extreme_dico(x,y)
print(res['mini'])
```

Les deux fonctions renvoient deux objets différents
Notez l'accès à la valeur minimale selon le type de l'objet

Visibilité (portée) des variables

Variables locales et globales

1. Les variables définies localement dans les fonctions sont uniquement visibles dans ces fonctions.
2. Les variables définies (dans la mémoire globale) en dehors de la fonction ne sont pas accessibles dans la fonction
3. Elles ne le sont uniquement que si on utilise un mot clé spécifique

```
#fonction
def modif_1(v):
    x = v

#appel
x = 10
modif_1(99)
print(x) → 10
```

x est une variable locale, pas de répercussion

```
#fonction
def modif_2(v):
    x = x + v

#appel
x = 10
modif_2(99)
print(x)
```

x n'est pas assignée ici, l'instruction provoque une ERREUR

```
#fonction
def modif_3(v):
    global x
    x = x + v

#appel
x = 10
modif_3(99)
print(x) → 109
```

On va utiliser la variable globale **x**. L'instruction suivante équivaut à x = 10 + 99

Récursivité

Exemple

```
def factoriel(nombre) :
    """
    Cette fonction retourne le factoriel d'un nombre.
    nombre : Le nombre dont on veut le factoriel.
    return : nombre!
    """
    if nombre == 1 or nombre == 0 :
        return 1
    else :
        return nombre * factoriel(nombre - 1)
```

Travaux dirigés

Travaux dirigés

1. Ecrire une fonction **longueur_chaine(ch)**qui reçoit en argument une chaine de caractères ch, et qui retourne sa taille (sans utiliser la fonction Len)
2. Ecrire une fonction **nbr_occurence(ch,e)** qui reçoit en arguments une chaine de caractères **ch** et un caractère **e**, la fonction retourne le nombre d'occurrences du caractère **e** dans la chaine **ch**.
3. Ecrire une fonction **supp_espace(ch)** qui reçoit en argument une chaine de caractères **ch**, la fonction retourne la chaine **ch** après avoir supprimé tous les caractères espaces s'ils existent au début de cette chaine.
4. Ecrire une fonction **Moyenne()**qui calcule la moyenne des valeurs entrées en paramètres de la fonction quelque soit leurs nombres

Création et utilisation des modules

LES MODULES

Importer un module ou des fonction d'un module

```
#Méthode 1
import math
x=math.sqrt(25)
print(x)

#Méthode 2
import math as m
x=m.sqrt(25)
print(x)

#Méthode 3
from math import sqrt
x=sqrt(25)
print(x)
```

```
import math,random              #importer 2 modules
from math import sqrt,sin       #importer 2 fonctions
from math import *              #importer toutes les  fonctions
```

Le module random

main.py Run

```
1  from random import *
2  x=random()     #retourne une valeur aléatoire x :0 < x < 1
3  x=randrange(n)   #retourne un entier aléatoire x : 0<= x < n
4  x=randrange(n,m) #retourne un entier aléatoire x : n<= x < m
5  x=randrange(n,m,p)
6  #retourne un entier aléatoire x :  n<= x < m  avec un pas de  p
7  x=randint(n,m))  # randint(n,m)=randrange(n,m+1) càd n<= x <=m
```

La gestion des fichiers

La fonction clé pour travailler avec des fichiers en Python est la `open()` fonction.

La `open()` fonction prend deux paramètres; *nom de fichier* et *mode* .

Il existe quatre méthodes (modes) différentes pour ouvrir un fichier:

`"r"` - Lire - Valeur par défaut. Ouvre un fichier en lecture, erreur si le fichier n'existe pas

`"a"` - Ajouter - Ouvre un fichier à ajouter, crée le fichier s'il n'existe pas

`"w"` - Ecrire - Ouvre un fichier pour l'écriture, crée le fichier s'il n'existe pas

`"x"` - Créer - Crée le fichier spécifié, renvoie une erreur si le fichier existe

De plus, vous pouvez spécifier si le fichier doit être traité en mode binaire ou texte

`"t"` - Texte - Valeur par défaut. Mode texte

`"b"` - Binaire - Mode binaire (par exemple images)

Ouvrir et fermer un fichier

Syntaxe

Pour ouvrir un fichier en lecture, il suffit de spécifier le nom du fichier:

```
f = open("demofile.txt")
```

Le code ci-dessus est le même que:

```
f = open("demofile.txt", "rt")
```

Étant donné `"r"` que `"t"` les valeurs par défaut pour la lecture et pour le texte sont, vous n'avez pas besoin de les spécifier.

Remarque: assurez-vous que le fichier existe, sinon vous obtiendrez une erreur.

Comment fermer un fichier ?

```
1 f.close()
```

Ecrire dans un fichier : write()

Écrire dans un fichier existant

Pour écrire dans un fichier existant, vous devez ajouter un paramètre à la `open()` fonction:

`"a"` - Ajouter - ajoutera à la fin du fichier

`"w"` - Ecrire - écrasera tout contenu existant

Exemple

Ouvrez le fichier "demofile2.txt" et ajoutez le contenu au fichier:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Ecrire dans un fichier : write()

Exemple

```
fichier = open("test.txt","w")
fichier.write("Salut ! J'essaye d'écrire dans un fichier !")
fichier.close()
```

Vous pouvez appliquer plusieurs fois la fonction ***write()*** à un même fichier. Dans une boucle par exemple, pour écrire une suite de données calculées ou issues d'une séquence.

Exemple

```
fichier = open("test.txt","w")
for indice in range(8) :
    fichier.write("Je suis la ligne " + str(indice + 1) + " !\n")
fichier.close()
```

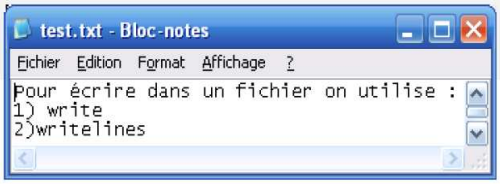
Remarque Si vous oubliez de fermer votre fichier avant la fin de votre programme, il se peut que celui-ci ne soit pas sauvegardé !

Ecrire dans un fichier : writelines()

Pour écrire dans un fichier, il existe aussi la fonction appelée *writelines* (écrire lignes) qui permet d'écrire tous les éléments d'une liste dans un fichier.

Exemple

```
liste = ["Pour écrire dans un fichier on utilise :\n","1) write\n", "2)writelines\n"]
fichier = open("test.txt","w")
fichier.writelines(liste)
fichier.close()
```



Lecture à partir d'un fichier : read()

Apres avoir ouvrir le fichier, le langage Python permet plusieurs méthodes d'accès à un fichier :

- ✓ **read(n)** : lit au plus **n** caractères dans le fichier s'il reste encore plus de **n** caractères de la position courante à la fin de fichier.

Syntaxe

Chaine=fichier.read(n)

Exemple

```
>>>fichier = open("test.txt","r")
>>>chaine=fichier.read(4)
>>>print(chaine)
Pour
>>>fichier.close()
```

L'utilisation de la méthode **read()** sans paramètre permet de lire tout le fichier.

```
>>>fichier = open("test.txt","r")
>>>chaine=fichier.read()
>>>print(chaine)
Pour écrire dans un fichier on utilise :
1) write
2)writelines
>>>fichier.close()
```

Lecture à partir d'un fichier : readline()

Vous pouvez renvoyer une ligne en utilisant la `readline()` méthode:

Exemple

Lisez une ligne du fichier:

```
f = open("demofile.txt", "r")
print(f.readline())
```

En appelant `readline()` deux fois, vous pouvez lire les deux premières lignes:

Exemple

Lisez deux lignes du fichier:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

En parcourant les lignes du fichier, vous pouvez lire l'intégralité du fichier, ligne par ligne:

Exemple

Parcourez le fichier ligne par ligne:

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

Lecture à partir d'un fichier : readlines()

✓ **readlines() :** cette méthode permet de récupérer l'ensemble de contenu de fichier et de le mettre dans une liste qui sera en suite retournée.

fichier = open("test.txt","r")

Exemple

```
>>>fichier = open("test.txt","r")
>>>liste = fichier.readlines()
>>>print(liste[0])
Pour écrire dans un fichier on utilise :
>>>print(liste[1])
1) write
>>>print(liste[2])
2) writelines
>>>fichier.close()
```

```
>>>fichier = open("test.txt","r")
>>>liste = fichier.readlines()
>>>for element in liste :
…       print(element)
Pour écrire dans un fichier on utilise :
1) write
2) writelines
>>>fichier.close()
```

Se déplacer dans un fichier : seek()

La fonction *seek* accepte deux arguments. Le premier indique le nombre d'octet dont on souhaite se déplacer. Le second indique la position à partir de laquelle on souhaite se déplacer (0 pour commencer du début du fichier, 1 pour la position courante ou 2 pour la fin du fichier).

Exemple

```
fichier = open("test.txt","r")
fichier.seek(42,0)    # positionner le curseur au début de la deuxième ligne (après 42 caractères de début)
ligne=fichier.readline()
print(ligne)
fichier.close()
```