seance_2020-12-08_com

December 9, 2020

1 A propos des fonctions: valeur de retour

Vision possible de la fonction en python: *procédure*: regroupement d'instructions auquel on donne un nom: on enrichit en quelque sorte le répertoire des commandes possibles, par exemple ci dessous en ajoutant l'instruction "`affichage()`".

[1]:
```
def affichage():
    print("Bonjour")
    print("Il fait beau aujourd'hui")

affichage()
```

```
Bonjour
Il fait beau aujourd'hui
```

On peut de plus *paramétrer* ces "nouvelles instructions": ci dessous un exemple de procédure qui comporte un paramètre. Au lieu d'avoir créé une instruction (`affichage()`), on crée ainsi comme une famille d'instructions (`affichage(3)`, `affichage(5)`, `affichage(254)`, *etc.*).

[2]:
```
def affichage(n):
    print("Bonjour")
    print("Il fait beau depuis %d jours"% n)

affichage(4)
```

```
Bonjour
Il fait beau depuis 4 jours
```

Autre vision de la fonction : la notion familière de *fonction mathématique*, qui est celle d'un calcul qui dépend d'un paramètre (au moins) et qui *renvoie* ou *retourne* un résultat. Prenons par exemple la fonction f définie par l'équation $f(x) = 3x + 6$. On sait que $f(0)$ *vaut* 6, que $f(1)$ vaut 9, *etc.* Pour le dire encore autrement, un fonction selon ce point de vue *associe* une valeur à chacun des paramètres possibles.

Le langage python permet de s'approcher de cette notion mathématique, en ajoutant à la définition d'une fonction la possibilité, avec l'instruction `return`, de renvoyer un résultat. Ainsi, alors qu'une instruction comme `print()` (fonction prédéfinie) ou comme `affichage()` (telle que définie plus haut) n'a pas de *valeur* particulière, un appel à une fonction plus proche du sens mathématique va **avoir une valeur**, que l'on peut utiliser par exemple dans une affectation:

[3]:
```
def fmath(x):
    return 3 * x + 6

print(fmath(4))
y = fmath(7)
print(y)
maliste = [fmath(2),fmath(3),fmath(4)]
print(maliste)
```

```
18
27
[12, 15, 18]
```

Il existe en python beaucoup de fonctions prédéfinies qui ont ainsi une valeur (on parle souvent de *valeur de retour*). Par exemple la fonction `sum()` qui renvoie/retourne/vaut la somme des nombres de la liste donnée en paramètre.

[4]:
```
f = sum([3,8,9])
print(f)
```

```
20
```

Par exemple la fonction mathématique *cosinus* (disponible dans la bibliothèque `math`): (ci dessous deux cellule illustrant, par parenthèse, les deux façons que l'on peut avoir d'importer cette fonction `cos()` depuis la bibliothèque mathématique.)

[5]:
```
import math
y = math.cos(2*3.14159263)
print(y)
```

```
0.9999999999999989
```

[6]:
```
from math import cos
y = cos(2*3.14159263)
print(y)
```

```
0.9999999999999989
```

Nouvel exemple de fonction, correspondant bien à la notion mathématique: la fonction suivante, qui convertit une température donnée en degrés Celsius vers une température donnée en degrés Farenheit:

[7]:
```
def farenheit(t):
    f = (9./5 * t) + 32
    return f
# Variante d'écriture, plus compacte:
def farenheit(t):
    return (9./5 * t) + 32
```

[8]:
```
x = farenheit(37.5)
#print(x)
```

Telle qu'elle est définie, la fonction `farenheit()` n'affiche pas à l'écran son calcul: c'est seulement en affichant la variable dans laquelle le résultat a été stocké qu'on peut vérifier le résultat. Par contraste, on peut définir une fonction `farenheit_p()` qui fait le même calcul, mais ne le *renvoie* pas, elle ne fait que l'afficher.

[9]:
```
def farenheit_p(t):
    f = (9./5 * t) + 32
    print(f)
    return None
```

[10]:
```
x = farenheit_p(37.5)
#print(x)
```

```
99.5
```

[11]:
```
farenheit_p(38)
```

```
100.4
```

1.1 Illustration de l'utilisation des fonctions ayant une valeur de retour:

Ci-dessous plusieurs ré-écritures successives d'un programme qui demande une température à l'utilisation, la convertit et l'affiche:

[12]:
```
t = float(input("Donnez une température "))
f = (9./5 * t) + 32
print("Température convertie : %f" % f)
```

```
Donnez une température 45
Température convertie : 113.000000
```

[13]:
```
def conversion_i():
    t = float(input("Donnez une température "))
    f = (9./5 * t) + 32
    print("Température convertie : %f" % f)
```

[14]:
```
conversion_i()
```

```
Donnez une température 39.5
Température convertie : 103.100000
```

[15]:
```
def conversion_i():
    t = float(input("Donnez une température "))
    f = farenheit(t)
    print("Température convertie : %f" % f)
```

[16]:
```
conversion_i()
```

```
Donnez une température 37.6
Température convertie : 99.680000
```

[17]:
```
def conversion_i():
    f = farenheit(float(input("Donnez une température ")))
    print("Température convertie : %f" % f)
```

[18]:
```
conversion_i()
```

```
Donnez une température 37.7
Température convertie : 99.860000
```

[19]:
```
print("Température convertie : %f" % farenheit(float(input("Donnez une␣
 ↪température "))))
```

```
Donnez une température 37.8
Température convertie : 100.040000
```

1.2 Exemple de fonction booléenne

[20]:
```
def appartient(x,l):
    for elt in l:
        if elt == x:
            return True
    return False
```

[21]:
```
if appartient(2,[2,4,6]):
    print("trouvé")
else:
    print("non trouvé")
```

```
trouvé
```

2 Fonctions prédéfinies sur les listes

En reprenant les fonctions présentées dans le cadre correspondant du memento, on présente successivement ci dessous: 1. Des fonctions pour ajouter des éléments à la fin d'une liste * `append()` qui ajoute un élément à une liste * `extend()` qui ralonge une liste avec une autre liste 2. Des fonctions agissent sur la liste en utilisant les indices (*positions*) * `insert()` qui insère un élément à une position donnée * `pop()` qui retourne l'élément se trouvant à une position donnée, et le supprime 3. Des fonctions plus complexes * `remove()` qui supprime la première occurrence d'un élément donné (fonction qui suppose un parcours de la liste pour rechercher l'occurrence) * `sort()` (et `reverse()`) qui trie (ou inverse) sur place la liste

Définitions de quelques variables pour faciliter les manipulations ultérieures

[22]:
```
toto = [3, 7, 9, 1, 0, 8, 8, 8]
titi = [9, False, 8.34, True, 22, 1e-3, 'a', -1]
tutu = [[0,1], ['a', 'b', 'c'], 78.5]
vecteur = [0]*100
lv = []
roman = ['Nous', 'étions', 'à', 'l'Étude,', 'quand', 'le', 'Proviseur',␣
 ↪'entra,',
         'suivi', 'd'un', 'nouveau', 'habillé', 'en', 'bourgeois', 'et', 'd'un',
         'garçon', 'de', 'classe', 'qui', 'portait', 'un', 'grand', 'pupitre.',
         'Ceux', 'qui', 'dormaient', 'se', 'réveillèrent,', 'et', 'chacun',␣
 ↪'se',
         'leva', 'comme', 'surpris', 'dans', 'son', 'travail.']
```

Exercice: écrire une fonction qui renvoie le minimum d'une liste

[23]:
```
def minimum(l):
    min = l[0]
    for elt in l[1:]:
        if elt < min:
            min = elt
    return min
```

[24]:
```
newl = [5, 8,9,24, 67,3,45]
y = minimum(newl)
print(y)
```

```
3
```

[25]:
```
print(minimum(toto))
```

```
0
```

[26]:
```
print(toto)
print(toto[:5])
```

```
[3, 7, 9, 1, 0, 8, 8, 8]
[3, 7, 9, 1, 0]
```

Exercice: écrire une fonction qui renvoie vrai si un élément de la liste passée en paramètre est en double

[27]:
```
# la fonction suivante ne fonctionne pas
def exo2(liste):
    element = liste[0]
    for elt in liste[1:]:
        if element == elt:
            return True
    return False
```

```
# Proposition de correction:
def contient_doublon(liste):
    for i in range(len(liste)-1):
        element = liste[i]
        for elt in liste[i+1:]:
            if element == elt:
                return True
    return False
```

[28]:
```
x = exo2([3, 7, 9, 7])
print(x)
```

```
False
```

[29]:
```
x = contient_doublon([3, 7, 9, 9, 12])
print(x)
```

```
True
```

[30]:
```
# Exercice: écrire une fonction qui reçoit deux arguments
# et dit si ces deux arguments sont identiques ou différents
```

3 Manipulations de listes

[31]:
```
x = [0]*10
print(x)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

[32]:
```
x[3] = 5
print(x)
```

```
[0, 0, 0, 5, 0, 0, 0, 0, 0, 0]
```

[33]:
```
x.append(4)
print(x)
```

```
[0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 4]
```

[34]:
```
l = []
for n in range(100):
    if n % 3 == 0 :
        l.append(n)
print(l)
```

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60,
63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99]
```

[35]:
```
l = [3,4,5]
l.append(6)
print(l)
```

```
[3, 4, 5, 6]
```

[36]:
```
l = [3,4,5]
l.append([7,8,9])
print(l)
```

```
[3, 4, 5, [7, 8, 9]]
```

[37]:
```
l = [3,4,5]
l.extend([7,8,9])
print(l)
```

```
[3, 4, 5, 7, 8, 9]
```

[38]:
```
l.insert(3,6)
print(l)
```

```
[3, 4, 5, 6, 7, 8, 9]
```

[39]:
```
l.insert(0,0)
print(l)
```

```
[0, 3, 4, 5, 6, 7, 8, 9]
```

[40]:
```
l.pop(1)
```

[40]:
```
3
```

[41]:
```
print(l)
```

```
[0, 4, 5, 6, 7, 8, 9]
```

[42]:
```
print(l.append(6))
```

```
None
```

[43]:
```
print(l)
```

```
[0, 4, 5, 6, 7, 8, 9, 6]
```

[44]:
```
print(l.pop(4))
print(l)
```

```
7
[0, 4, 5, 6, 8, 9, 6]
```

[45]:
```
l.remove(5)
print(l)
```

```
[0, 4, 6, 8, 9, 6]
```

[46]:
```
txt = list("Sois sage ô ma douleur")
```

[47]:
```
print(txt)
```

```
['S', 'o', 'i', 's', ' ', 's', 'a', 'g', 'e', ' ', 'ô', ' ', 'm', 'a', ' ', 'd',
'o', 'u', 'l', 'e', 'u', 'r']
```

[48]:
```
txt.remove('s')
print(txt)
```

```
['S', 'o', 'i', ' ', 's', 'a', 'g', 'e', ' ', 'ô', ' ', 'm', 'a', ' ', 'd', 'o',
'u', 'l', 'e', 'u', 'r']
```

[49]:
```
txt.sort()
print(txt)
```

```
[' ', ' ', ' ', ' ', 'S', 'a', 'a', 'd', 'e', 'e', 'g', 'i', 'l', 'm', 'o', 'o',
'r', 's', 'u', 'u', 'ô']
```

[50]:
```
l.sort()
print(l)
```

```
[0, 4, 6, 6, 8, 9]
```

[51]:
```
l.reverse()
print(l)
```

```
[9, 8, 6, 6, 4, 0]
```