

Programmation procédurale

RACHID ALILI

Principe général d'une fonction

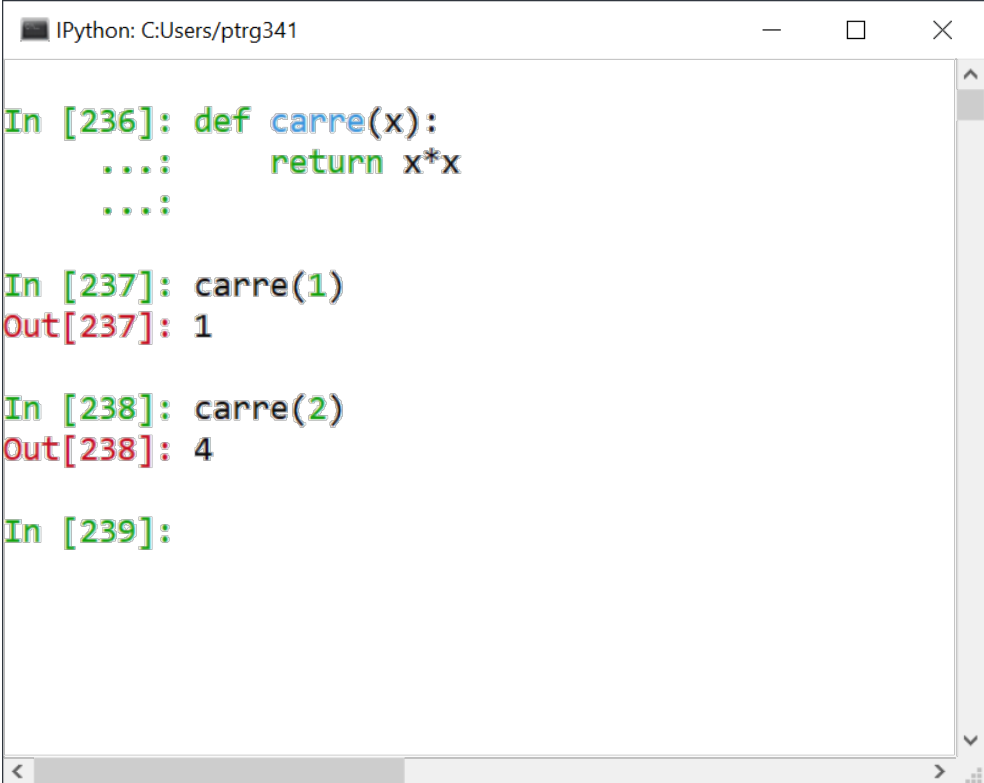
Exemple : Définir une fonction qui calcule le carré d'un nombre :

```
def carre(x):  
    return x*x
```

Son utilisation (appel) est plus familière :

```
carre(1) = 1  
carre(2) = 4  
etc...
```

⇒



```
IPython: C:\Users\ptrg341  
  
In [236]: def carre(x):  
...:     return x*x  
...:  
  
In [237]: carre(1)  
Out[237]: 1  
  
In [238]: carre(2)  
Out[238]: 4  
  
In [239]:
```

Principe général d'une fonction

Moins évident pour la fonction $\sin(x)$:

- Sa définition nécessite de réfléchir à l'implémentation ou à l'algorithme (serie de Taylor ou cordic)
- Il faut s'assurer qu'elle renverra toujours un résultat cohérent

Les fonctions se rapportent aussi à n'importe quel problème non scientifique. Là aussi il faut réfléchir à son implémentation et toujours s'assurer qu'elle renverra toujours un résultat cohérent.

Définition d'une fonction en python

- Structure générale d'une fonction

```
def nomDeLaFonction(liste de paramètres):
```

```
    # bloc d'instructions indenté
```

```
# suite du programme
```

Définition d'une fonction en python

- Pas de **mots réservés** du langage ou **de caractères spéciaux ou accentués**.
- Utiliser plutôt **des lettres minuscules**,
 - Surtout au début du nom
 - Le caractère souligné « _ » est permis et encouragé.
- La signature se termine obligatoirement par un ":" qui introduit un bloc d'instructions.
- La *liste de paramètres* (peut être vide), fournie respectera certaines règles.

Définition d'une fonction en python

■ Autres exemples :

```
def ouEstCharlie(nom):  
    if nom == 'Charlie':  
        print('Bravo')  
    else:  
        print('perdu')
```

```
def ouEstCharlie(nom):  
    return nom == 'Charlie':
```

```
def ouEstCharlie(nom):  
    if nom == 'Charlie' : return True  
    else : return False
```

Appel d'une fonction en python

- L'appel est identique à une instruction quelconque.
- Il est constitué du nom de la fonction suivi des arguments entre les parenthèses
- Les arguments sont renseignés aux valeurs souhaitées et transmis à la fonction.
- il faudra fournir une **valeur pour chacun des paramètres non prédéfinis**

Appel d'une fonction en python

```
Sélectionner IPython: C:Users/ptrg3...  
  
In [82]: def ouEstCharlie(nom):  
...:     return nom=='Charlie'  
...:  
  
In [83]: def ouEstCharlie(nom):  
...:     return nom=='Charlie'  
...:
```

```
IPython: C:Users/ptrg341  
  
In [85]: carre(10)  
Out[85]: 100  
  
In [86]: ouEstCharlie('toto')  
Out[86]: False  
  
In [87]: ouEstCharlie('Charlie')  
Out[87]: True  
  
In [88]:
```


Notion de variables locales et globales

--- test 1 ---

```
def f():  
    print("dans f() : ", a)
```

#----

```
a = 2  
f()  
print("après f() : ", a)
```



dans f() : 2
après f() : 2

--- test 2 ---

```
def f():  
    a = 3  
    print("dans f() : ", a)
```

#----

```
a = 2  
f()  
print("après f() : ", a)
```



dans f() : 3
après f() : 2

--- test 3 ---

```
def f():  
    global a  
    a = 3  
    print("dans f() : ", a)
```

#----

```
a = 2  
f()  
print("après f() : ", a)
```



dans f() : 3
après f() : 3

Passage de paramètres par valeur et référence

```
Sélectionner IPython: C:Users/ptrg341

In [276]: # passage par valeur
...: def f(x):
...:     x += '1'
...:     print(x)
...:

In [277]: x = 'toto'
...: f(x) # je passe "toto" et non x
...: print(x)
...: # x est inchangé à l'extérieur

toto1
toto
```

La référence passée est la chaîne de caractères

```
IPython: C:Users/ptrg341

In [279]: # passage par référence de l'objet
...: def f(maliste):
...:     maliste.append('2')
...:     print('fonction : id = ', id(maliste))
...:

In [280]: #-----
...: # avant appel
...: maliste = [1, 2, 3]
...: print('principal : id = ', id(maliste))
...:
...: # appel
...: f(maliste)
...: print(maliste)
...:

principal : id = 2139731530624
fonction : id = 2139731530624
[1, 2, 3, '2']

In [281]:
```

Passage de paramètres obligatoire / facultatif

- Python permet aux arguments de fonction d'avoir une **valeur par défaut**, si la fonction est appelée sans l'argument ce dernier prend sa valeur par défaut.
- Les arguments nommés peuvent être listés dans n'importe quel ordre .
- Exemple :

```
def f(x, y=10, z=1) :
```

...

- `y` et `z` sont optionnels car des valeurs par défaut sont définies.
- `x` est obligatoire car il n'a pas de valeur par défaut.
- Si la fonction `f()` est appelée avec un seul argument, `y` prend la valeur 10 et `z` la valeur 1.

Passage de paramètres obligatoire / facultatif

```
def f(x, y=10, z=1):  
    return x, y, z
```

<code>print(f(10))</code>	<code>(10, 10, 1)</code>
<code>print(f(10, 5))</code>	<code>(10, 5, 1)</code>
<code>print(f(10, 5, 2))</code>	<code>(10, 5, 2)</code>

Exercice

- Question 1 : Ecrire une fonction qui calcule la moyenne de 2 nombres
- Question 2 : généraliser pour des nombres présents dans une liste
- Question 3 : On a maintenant 3 listes liste des étudiants, liste de leur note en math et liste de leur note en français. Calculer la moyenne pour chaque étudiant

```
nom='pierre','Paul','Jacques'  
fr=10,12,14  
mth=16,10,6
```

Exercice

- Ecrire une fonction qui calcule la moyenne de 2 nombres

```
IPython: C:\Users\ptrg341

In [282]: nom='pierre','Paul','Jacques'
...: fr=10,12,14
...: mth=16,10,6

In [283]: # question 1
...: def moy(x,y): return (x+y)/2

In [284]: moy(20,5)
Out[284]: 12.5

In [285]:
```

Exercice

- Question 2 : Généraliser pour des nombres présents dans une liste

```
nom='pierre','Paul','Jacques'  
fr=10,12,14  
mth=16,10,6
```

```
IPython: C:\Users\ptrg341

In [301]: # question 2
...: def moyL1(l):
...:     s=0
...:     for i in range(len(l)) :
...:         s+=l[i]
...:     return s/len(l)
...:
...: def moyL2(l):
...:     return sum(l)/len(l)
...:

In [302]: print(moyL1(mth), moyL2(mth))
10.666666666666666 10.666666666666666

In [303]:
```

Exercice

Question 3 :

On a maintenant 3 listes

- liste des étudiants,
- liste des notes en math
- Liste de notes en français.
- Calculer la moyenne pour chaque étudiant

nom='pierre','Paul','Jacques'

fr=10,12,14

mth=16,10,6

```
IPython: C:\Users\ptrg341

In [303]: # question 3
...: def moyL0(l1,l2):
...:     l=[]
...:     for i in range(len(l1)):
...:         l.append(l1[i]/2+l2[i]/2)
...:     return l
...:
...: def moyL1(l1,l2):
...:     l=[]
...:     for x,y in zip(l1,l2) :
...:         l.append((x+y)/2)
...:     return l
...:
...: def moyL2(l1,l2):
...:     return [(x+y)/2 for x,y in zip(l1,l2)]
...:

In [304]: print(moyL0(fr,mth))
...: print(moyL1(fr,mth))
...: print(moyL2(fr,mth))
[13.0, 11.0, 10.0]
[13.0, 11.0, 10.0]
[13.0, 11.0, 10.0]

In [305]:
```


Valeurs de retour

Python permet de retourner un ensemble de valeurs, elles peuvent être toutes exploitées ou en partie seulement.

Attention `x, y = f()` ⚡

ValueError: too many values to unpack (expected 2)

```
IPython: C:\Users\ptrg341

In [306]: def f():
...:     a = 10
...:     b = 20
...:     c = 'coucou'
...:     return a, b, c
...:
...:     print('retour de la commande f() : ', f())
...:
...:     x, y, _ = f()
...:     print('2 premiers parametres de f() : ', x, y)
...:
retour de la commande f() : (10, 20, 'coucou')
2 premiers parametres de f() : 10 20

In [307]:
```

Fonction anonyme

- Fonction simplifiée sur une ligne et sans déclarer de nom ni de bloc `def ()`
- Pas de `return`, le résultat est passé à une variable

```
IPython: C:Users/ptrg341

In [308]: def f(x, a):
...:     return x**a
...:

In [309]: f(3,4)
Out[309]: 81

In [310]: g = lambda x: x**4
...:     g(4)
Out[310]: 256
```

```
IPython: C:Users/ptrg341

In [311]: (lambda x: x**3)(4)
Out[311]: 64

In [312]: fonction = lambda a, b: a**b

In [313]: fonction(3, 4)
...:
Out[313]: 81
```

TP

- Récupérer le TP1 puis charger le notebook

La correction est fournie, sachant qu'il y a toujours plusieurs façon de faire...