INFO-H-100 - Informatique

Séance d'exercices 3
Introduction à Python
Fonctions et tuples

Université Libre de Bruxelles
Faculté des Sciences Appliquées

2017-2018

Fonctions

Une fonction est une séquence d'instructions qui a un nom. Elle peut recevoir en entrée des arguments et peut renvoyer une valeur de retour.

```
longueur = len('SPAM')
```

On peut voir une fonction comme une boite noire qui effectue un travail.

- Ses arguments sont les informations dont elle a besoin pour faire son travail.
- Sa valeur de retour est le résultat de son travail.

Composition : un argument d'une fonction peut être toute expression compatible :

```
x = math.sin(degrees / 360.0 * 2 * math.pi)
x = math.exp(math.log(x+1))
```

Définition de fonction

Une définition de fonction spécifie le nom, les paramètres (optionnels) et la séquence d'instructions de la fonction.

Chaque ligne de la séquence d'instruction est indentée, c'est à dire décalée vers la droite (par exemple de 4 espaces).

```
>>> def times(x, y):
      return x * y

>>> def pretty_print(a_string):
      print('*' * (len(a_string) + 4))
      print('* ' + a_string + ' *')
      print('*' * (len(a_string) + 4))

>>> y = times(2, 3)
>>> print(y)
6
>>> pretty_print('Python')
**********
* Python *
**********
```

Retour et paramètres

Le mot clé `return` interrompt la fonction et définit son résultat.

```
>>> def get_ratio(x, y):
      return float(x) / y
      print('done.')

>>> get_ratio(3,4)
0.75
```

Ici, l'instruction `print('done.')` n'est jamais exécutée.

L'ordre des arguments est important, pas leur nom.

```
>>> get_ratio(4,3)
1.3333333333333333
```

```
>>> x = 4
>>> y = 3
>>> get_ratio(y, x):
0.75
```

Variables locales

Les paramètres et les variables définies à l'intérieur d'une fonction sont des variables locales à leur fonction, c'est-à-dire qu'ils n'existent pas en dehors de leur fonction.

```
>>> def pretty_print(a_string):
        size = len(a_string) + 4
        print('*' * size)
        print('* ' + a_string + ' *')
        print('*' * size)

>>> pretty_print('Python')
**********
* Python *
**********
>>> a_string
NameError: name 'a_string' is not defined
>>> size
NameError: name 'size' is not defined
```

On parle de portée d'une variable : la zone dans laquelle elle est visible.

Documenter ses fonctions

Un doctring est un commentaire éventuellement multiligne (encadré par des """) placé au début du corps d'une fonction.

```
>>> def get_sum(x, y):
      """ returns the sum of x and y """
      return x + y

>>> help(get_sum)
'Help on function get_sum in module __main__:
get_sum(x, y)
  returns the sum of x and y
```

Bonne habitude : documenter clairement ses fonctions.

Dire ce que la fonction fait et pas comment elle le fait.

Tests et fonctions

```
def is_even(number):
  """ returns True if number is even.
      Otherwise returns False. """
  if number % 2 == 0 :
    test = True
  else :
    test = False
  return test

x = int(input())
if is_even(x):
  print(str(x) + ' est pair')
```

Autres versions :

```
def is_even(number):
  return number % 2 == 0
```

```
def is_even(number):
  test = False
  if number % 2 == 0 :
    test = True
  return test
```

Tuples

En Python, un tuple est une séquence de valeurs formée en séparant ces valeurs par des virgules. L'usage des parenthèses est recommandé.

```
>>> point = (1,2)
>>> print(point)
(1, 2)
```

```
>>> cours = ('INFO','H',100)
>>> print(cours)
('INFO','H',100)
```

Les valeurs sont indexées

```
>>> print(point[0])
1
```

On peut assigner les valeurs d'un tuple dans un autre possédant le même nombre de valeurs.

```
>>> (x,y) = point        #extraction
>>> print(x)
1
>>> print(y)
2
```

Tuples

Une fonction peut prendre un tuple en paramètre.

```
>>> def distance_origine(point):
      (x,y) = point
      return math.sqrt(x**2 + y**2)

>>> distance_origine((0,1))
1.0
```

Attention :

```
>>> distance_origine(0,1)
...
TypeError: distance_origine() takes 1 positional argument but 2 were
```

Une fonction peut aussi renvoyer un tuple, ce qui lui permet par exemple de renvoyer plusieurs valeurs.

```
>>> def divise_modulo(a, b):
      return (a // b, a % b)

>>> (quotient, reste) = divise_modulo(5,2)
>>> print(quotient, reste)
2 1
```

Prochain TP

- **listes, strings**