INFO-H-100 - Informatique

Séance d'exercices 2
Introduction à Python
Scripts, fonctions et tests

Université libre de Bruxelles
École polytechnique de Bruxelles

2013-2014

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.

```
length = len('SPAM')
```

On peut voir une fonction comme une boîte 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'instructions 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 clef `return` interrompt la fonction et définit son résultat.

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

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

Bonne habitude : documenter clairement ses fonctions.

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

Conseils

Dans une fonction, n'utilisez que les valeurs passées en paramètres et les variables locales à celle-ci. Une fonction est une boîte noire. Pour donner un résultat, elle ne doit avoir besoin de rien d'autre.

Donnez des noms explicites et simples à vos fonctions, qui indiquent ce qu'elles font.

Documentez clairement et succinctement ce que font vos fonctions à l'aide des docstring.

Utilisez l'anglais pour tous les noms de variables, fonctions ainsi que pour les commentaires.

Expressions booléenes

Une expression booléenne est une expression dont la valeur est soit vraie (`True`) , soit fausse (`False`). Ces expressions sont de type `bool`.

```
>>> 5 == 5
True
>>> 5 != 5
False
```

Elle peut se composer d'opérandes et de comparateurs :
`<` `<=` `>` `>=` `!=` `==`

Ne pas confondre = (assignation) et == (comparaison d'égalité).

L'instruction `if`

L'instruction `if` permet de tester une condition et d'exécuter du code si cette condition est vérifiée.

```
x = 2
if x >= 0:
  print('x est positif')
```

Le code exécuté est constitué du code indenté qui suit l'instruction `if`.

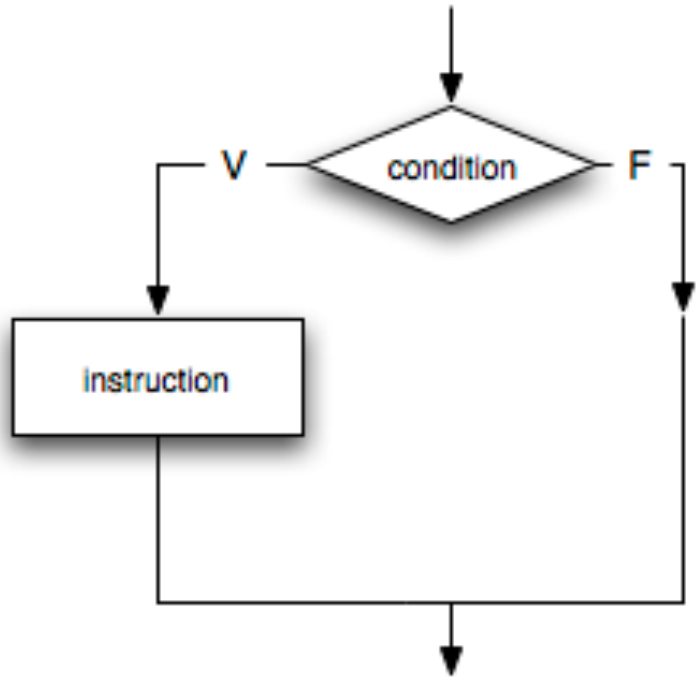
Une deuxième forme d'instruction `if` est disponible :

```
x = 2
if x % 2 == 0:
  print('x est pair')
else:
  print('x est impair')
```

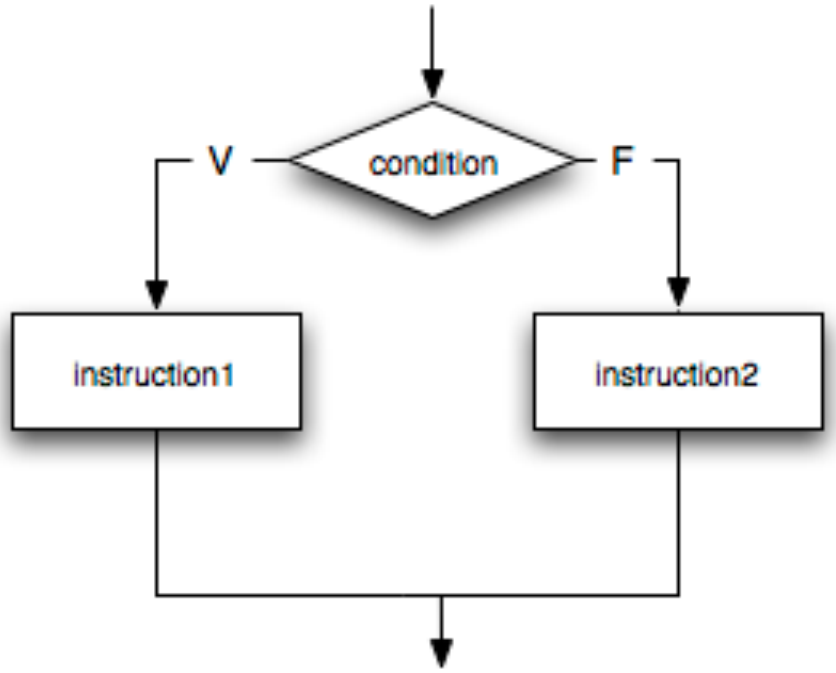
Dans ce cas, les instructions indentées après le `else` sont exécutées uniquement si la condition du `if` est fausse.

Tests simples

```
if condition:
  instruction
```




```
if condition:
  instruction_1
else:
  instruction_2
```



Tests et fonctions

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

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):
  even = False
  if number % 2 == 0:
    even = True
  return even
```

Visualiser le fonctionnement

Le site `http://www.pythontutor.com/visualize.html` fournit un outil très pratique pour visualiser le fonctionnement du code étape par étape.

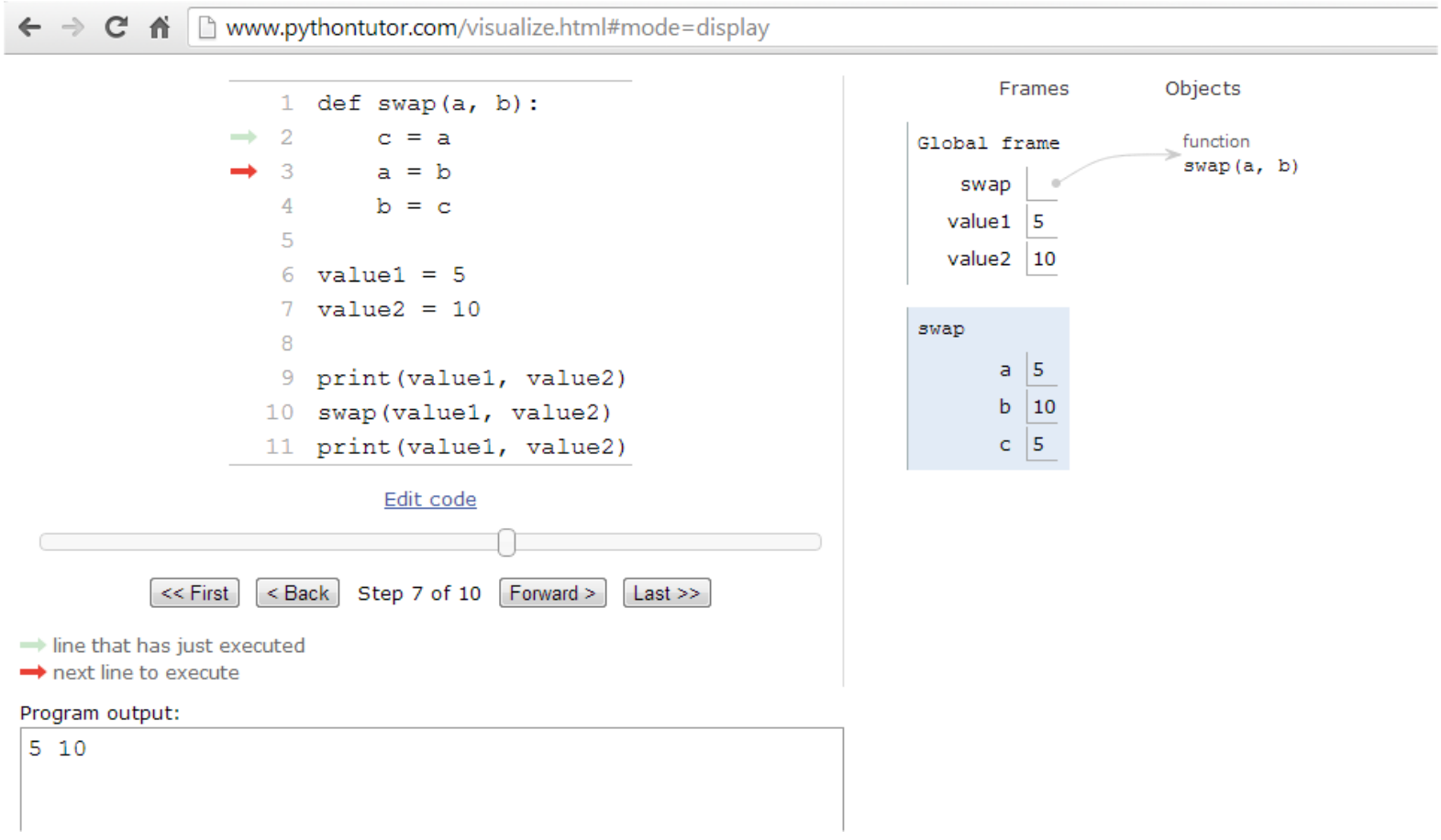


Tableau de séquence des valeurs des variables

Il s'agit d'un tableau permettant de décrire le comportement d'un système avec ses transitions.

Il existe différents formalismes. Ici, nous utilisons un système de tableaux pour représenter les états successifs du système.

```
length = 4
width = 6
area = a * b

width = 2
area = a * b
```

instruction	length	width	area
`length = 4`	4		
`width = 6`	4	6	
`area = length * width`	4	6	24
`width = 2`	4	2	24
`area = length * width`	4	2	8

Trouver de la documentation

Le site `docs.python.org` contient la documentation de tous les modules officiels.

La fonction `help()` donne l'aide d'une fonction ou d'un module :

```
>>> import math
>>> help(math.sqrt) #affiche l'aide de math.sqrt()
                    #(q pour quitter)
>>> help(math)      #affiche l'aide de tout le module math
```

La fonction `dir()` affiche le contenu d'un objet :

```
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>> a = 3
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'a']
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh',
'ceil', # etc.
```

Derniers conseils

Programmez étape par étape et testez continuellement votre code. Ça ne sert à rien de programmer 200 lignes d'affilée sans les tester.

Vérifiez que vos fonctions renvoient bien ce que vous attendez.

Faites un code clair et lisible. Ne laissez pas de code mort.

Des fonctions existent déjà au sein du langage ou de ses bibliothèques, utilisez-les ! Ça ne sert à rien de réinventer la roue.

Faites des recherches dans la documentation pour comprendre le fonctionnement des fonctions que vous utilisez. Recherchez également sur votre moteur de recherche favoris, beaucoup d'autres personnes ont certainement déjà rencontré les mêmes problèmes et une réponse y a déjà été apportée sur internet.

Exercices

- Exercices conseillés : 1, 2, 3, 4, 5, 7, 9, 10, 12, 13 et 14.
- Pour les exercices 4 à 11, écrire la ou les fonctions nécessaires et les tester à l'aide de valeurs pertinentes.