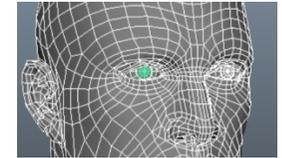
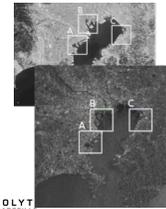
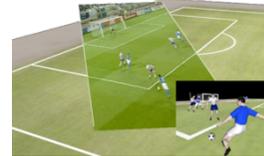


# Algorithmique et Programmation en Python

Sébastien MAVROMATIS & Alain SAMUEL  
<https://mavromatis.org/enseignement>

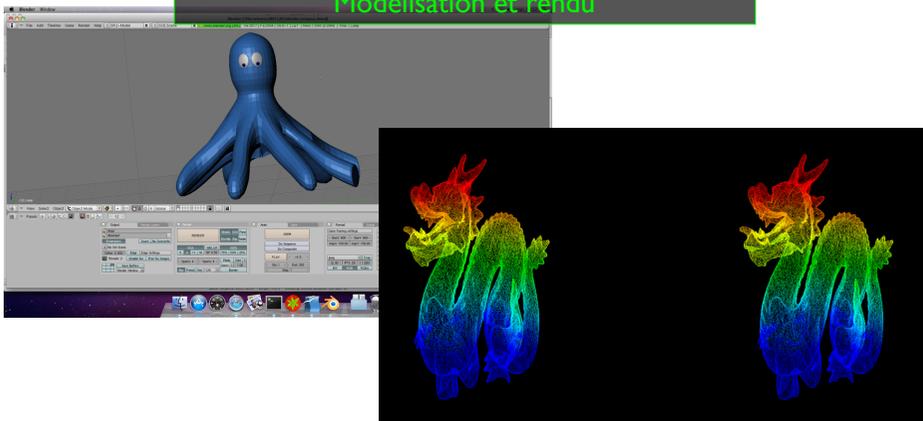


## Sébastien Mavromatis



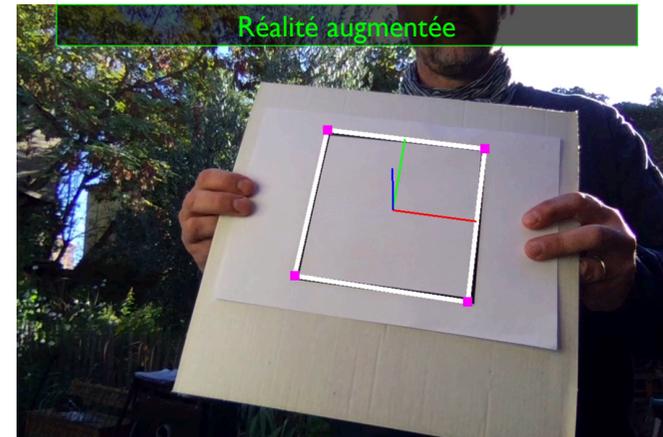
ReVA

Modélisation et rendu



ReVA

Réalité augmentée





# Evaluation

- Une note d'examen
- Une note de contrôle continu
- Présence obligatoire à tous les cours, TD, TP
  
- Une moyenne pondérée de tout cela définira votre note finale

# Environnement matériel

- L'**ordinateur** est un automate déterministe à composants électroniques
- L'ordinateur comprend entre autres :
  - Un microprocesseur avec une Unité de Contrôle, une Unité Arithmétique et Logique, une horloge ...
  - De la mémoire volatile (RAM) qui contient des instructions et des données nécessaires à l'exécution de **programmes**.
  - Des périphériques d'entrées/sorties comme par exemple le disque dur (mémoire permanente), les interfaces réseaux ...

# Les programmes

- Pour faire simple :
  - Le système d'exploitation
    - C'est un ensemble de programmes qui gèrent les ressources matérielles et logicielles. Il propose une aide au dialogue entre l'utilisateur et l'ordinateur (textuelle ou graphique).
  - Les programmes applicatifs
    - Ils sont formés d'une série d'instructions contenues dans un **programme source** qui est transformé pour être exécuté par l'ordinateur. Ce sont des programmes dédiés à des tâches particulières.

# Les langages de programmation

- Le langage machine
  - Le seul langage directement utilisable par l'ordinateur
  - Il est formé de 0 et de 1 : **10110000 01100001** très difficile de traduire pour l'homme !
  - Chaque processeur possède son propre langage
- Le langage d'assemblage
  - Il est formé de codes alphanumériques
  - Il est traduit en langage machine par un assembleur : **movb \$0x61,%al** pas évident à traduire pour l'homme !
    - Ecrire le nombre 97 (la valeur est donnée en hexadécimal) dans le registre AL
  - Chaque architecture de processeur possède son propre langage

# Les langages de programmation

- Les langages de haut niveau
  - Ils sont formés à l'aide d'instructions « compréhensibles » par l'homme :
    - `print ("Bonjour")` en langage python affiche le texte bonjour à l'écran
  - Ils sont normalisés
  - Ils sont portables
  - Ils sont traduits en langage machine par un compilateur ou un interpréteur

# Les langages de programmation

- Un très bref historique
  - Années 50 (approches expérimentales) : FORTRAN, LISP, COBOL ...
  - Années 60 (langages universels) : PL/1, Smalltalk, Basic ...
  - Années 70 (génie logiciel) : C, PASCAL, ADA ...
  - Années 80 (programmation objet) : C++, Objective-C, Eiffel, Perl ...
  - Années 90 (langages interprétés objet) : Java, Ruby, Python ...
  - Années 2000 (langages commerciaux propriétaires) : C#, VB.NET...

# Les langages de programmation

- Que de langages !

Language Popularity Index - Web queries done on: 2013/07/01 18:49

| Language category: any 123 entries. |             |         |                    |                   | Language category: general-purpose 48 entries. |             |         |  |  | Language category: script 49 entries. |             |         |  |  | Language category: other 26 entries. |              |         |  |  |
|-------------------------------------|-------------|---------|--------------------|-------------------|--|-------------|---------|--|--|---------------------------------------|-------------|---------|--|--|--------------------------------------|--------------|---------|--|--|
| Rank                                | Name        | Share   | Last month's share | Last year's share | Rank   | Name        | Share   |  |  | Rank                                  | Name        | Share   |  |  | Rank                                 | Name         | Share   |  |  |
| 1                                   | C           | 17.668% | 15.868%            | 16.825%           | 1  | C           | 26.623% |  |  | 1                                     | PHP         | 19.801% |  |  | 1                                    | Logo         | 14.778% |  |  |
| 2                                   | Java        | 14.720% | 15.450%            | 20.381%           | 2  | Java        | 22.182% |  |  | 2                                     | Python      | 16.458% |  |  | 2                                    | COBOL        | 14.597% |  |  |
| 3                                   | Objective-C | 8.230%  | 8.516%             | 9.221%            | 3  | Objective-C | 12.401% |  |  | 3                                     | Perl        | 11.545% |  |  | 3                                    | Prolog       | 10.099% |  |  |
| 4                                   | C++         | 6.770%  | 7.544%             | 7.912%            | 4  | C++         | 10.202% |  |  | 4                                     | Ruby        | 8.628%  |  |  | 4                                    | PL/SQL       | 8.939%  |  |  |
| 5                                   | Basic       | 5.457%  | 5.955%             | 7.592%            | 5  | Basic       | 8.223%  |  |  | 5                                     | JavaScript  | 5.399%  |  |  | 5                                    | SAS          | 8.619%  |  |  |
| 6                                   | PHP         | 4.401%  | 4.144%             | 4.247%            | 6  | C#          | 4.926%  |  |  | 6                                     | R           | 4.898%  |  |  | 6                                    | LabView      | 8.317%  |  |  |
| 7                                   | Python      | 3.658%  | 3.363%             | 3.616%            | 7  | Pascal      | 1.906%  |  |  | 7                                     | Lisp/Scheme | 3.960%  |  |  | 7                                    | ABAP         | 6.482%  |  |  |
| 8                                   | ...         | ...     | ...                | ...               | 8  | Ada         | 1.688%  |  |  | 8                                     | MATLAB      | 3.488%  |  |  | 8                                    | Transact-SQL | 2.837%  |  |  |

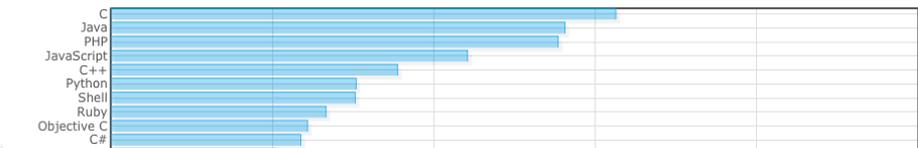
<http://lang-index.sourceforge.net>

# Les langages de programmation

- Que de langages !

## Normalized Comparison

This is a chart showing combined results from all data sets, listed individually below.



<http://langpop.com>

# Les langages de programmation

- Que de langages !

| Feb 2014 | Feb 2013 | Change | Programming Language | Ratings | Change |
|----------|----------|--------|----------------------|---------|--------|
| 1        | 2        | ▲      | C                    | 18.334% | +1.25% |
| 2        | 1        | ▼      | Java                 | 17.316% | -1.07% |
| 3        | 3        |        | Objective-C          | 11.341% | +1.54% |
| 4        | 4        |        | C++                  | 6.892%  | -1.87% |
| 5        | 5        |        | C#                   | 6.450%  | -0.23% |
| 6        | 6        |        | PHP                  | 4.219%  | -0.85% |
| 7        | 8        | ▲      | (Visual) Basic       | 2.759%  | -1.89% |
| 8        | 7        | ▼      | Python               | 2.157%  | -2.79% |
| 9        | 11       | ▲      | JavaScript           | 1.929%  | +0.51% |

<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>



- Un langage **interprété**
- Un langage **simple** à utiliser mais complet
- Une programmation **modulaire** (et objet)
- De nombreuses **bibliothèques associées** (image, cryptographie ...)
- Un des points forts : la **lisibilité**

## Démo !

```
mavromatis ~$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Bonjour")
Bonjour
>>> 1 + 2
3
>>> exit()
mavromatis ~$
```

## Démo !

```
mavromatis ~$ cat test.py
1 # Mon premier test d'un script python
2 print("Bonjour")
3 print(1+2)
4

mavromatis ~$ python test.py
Bonjour
3
mavromatis ~$
```

# Démo !

```
test_diazo_1.py
print ("Bonjour")
print ("-----")
a = 1
b = 2
print ('a + b = ', a+b)
print ("-----")

Run test_diazo_1
/System/Library/Frameworks/Python.framework/Versions/2.7/bin/python /Users/mavromatis/PyCh...
Bonjour
-----
('a + b = ', 3)
-----
```

# Démo !

```
Last login: Wed Mar 5 12:14:47 on tty000
mavromatis@styx000 ~ % system
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 3.0 (clang-400.0.68)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> print ("Bonjour")
Bonjour
>>> 1 + 2
3
>>> exit()
mavromatis@styx000 ~ %

test.py
1 # Mon premier test d'un script python
2 print ("Bonjour")
3 print 1+2
4

mavromatis@styx000 ~ % python test.py
Bonjour
mavromatis@styx000 ~ %
```

# Production de programmes

- Plusieurs techniques

- La **compilation**



- L'**interprétation**



- L'interprétation d'un **bytecode** compilé



# Algorithmes et programmes

- Un **algorithme** est un ensemble des étapes qui permet d'atteindre un but en répétant un nombre fini de fois un nombre fini d'instructions. Un algorithme se termine en **un temps fini**.
- Un **programme** est la **traduction d'un algorithme** en un langage qui est compilé ou interprété par un ordinateur. Il est souvent écrit en plusieurs parties dont une qui pilote les autres : le programme principal.

## Par exemple

- Algorithme d'Euclide : Etant donnés deux entiers, retrancher le plus petit au plus grand et recommencer jusqu'à ce que les deux nombres soient égaux. La valeur obtenue est le plus grand diviseur commun.
- Reprenons l'idée : Prendre deux nombres. Tant qu'ils ne sont pas égaux, soustraire le plus petit au plus grand.
- Description par étapes :
  - a et b deux nombres
  - **Répéter** tant que a et b sont différents
    - Si a est le plus grand, les deux nombres deviennent a-b et b
    - **Sinon**, les deux nombres deviennent a et b-a
  - Le pgcd est a

## Par exemple

- Ce qui peut, par exemple, s'exprimer ainsi

```
fonction pgcd(a, b : entiers)
  répéter tant que a ≠ b
    si a > b alors a = a - b
    sinon b = b - a
  retourner a
```

- En python

```
def pgcd(a, b):
  while a != b:
    if a > b: a = a - b
    else: b = b - a
  return a
```



## Commentaires

- Dans un script, tout ce qui suit le caractère **#** est ignoré jusqu'à la fin de la ligne. Cette ligne est un **commentaire**.
  - La première ligne de votre script peut déroger à cette règle.
- Les commentaires sont indispensables pour annoter un code source.



```
#!/usr/bin/env python
# Mon premier test d'un script python
print "Bonjour"
# L'instruction suivante affiche le resultat de l'addition de 1 et de 2
print 1+2
```

# Indentation

- En programmation, il est courant de répéter un certain nombre d'instructions (boucles) ou de faire des choix (tests)

Instruction\_qui\_indique\_qu'il\_faut\_réaliser\_50\_fois\_ce\_qui\_suit :



Instruction\_itérée\_1

Instruction\_itérée\_2

Instruction\_itérée\_3

Instruction\_suivante

Bloc  
d'instruction

- On reviendra là-dessus plus tard

# Variables

- Une variable est une zone mémoire dans laquelle une valeur est stockée
- Pour le programmeur, elle est définie par un nom
- Pour l'ordinateur, elle est définie par une adresse (zone mémoire particulière)
- La déclaration et l'initialisation d'une variable se fait en même temps

```
mavromatis ~ bash
Last login: Tue Feb 25 14:54:31 on ttyv000
mavromatis:~$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 2
>>> a
2
>>> 
```

# Noms des variables

- Premier caractère
  - Lettres minuscules (a-z), majuscules (A-Z), caractère \_
- Ensuite
  - Lettres minuscules (a-z), majuscules (A-Z), nombre (0-9), caractère \_
- Certains sont réservés
- Par exemple : **prixHT** ou bien **prix\_ht**

# Types

- Trois types principaux pour une variable : entier (int), réel (float), chaîne de caractères (str)
- Les variables sont typées dynamiquement

```
mavromatis ~ bash
mavromatis:~$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 2
>>> a
2
>>> b = 3.14
>>> b
3.14
>>> c = 'abodef'
>>> c
'abodef'
>>> type ( a )
<type 'int'>
>>> type ( b )
<type 'float'>
>>> type ( c )
<type 'str'>
>>> 
```

- En réalité, plus de types ... on y reviendra

# Variables

- Deux actions possibles :
  - Evaluer : c'est à dire lire sa valeur
  - Affecter : c'est à dire modifier sa valeur
    - L'affectation est réalisée par le signe =

```
a = 1
nom = "mavromatis"
b = 2
a = b
```

# Variables

- Par exemple :

```
riri = "fifi"
fifi = "loulou"
```

```
riri = "fifi"
fifi = riri
```

- Evaluons les variables

# Variables

- Par exemple :

```
a = 10
a = 20
b = a
```

```
a = 10
b = a
a = 20
```

- Evaluons les variables

# Opérateurs

- Un **opérateur** est un signe qui agit sur une ou plusieurs valeurs (**opérandes**), pour produire un résultat
  - Opérateurs arithmétiques
  - Opérateurs alphanumériques
  - Opérateurs de comparaison
  - Opérateurs logiques

# Opérateurs

- Opérateurs arithmétiques : + , - , \* , / , \*\* , %

```
mavromatis:~$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> a + 2
7
>>> b = 2.5
>>> a + b
7.5
>>> ( a * 100 ) / b
200.0
>>> []
```

# Opérateurs

- Opérateurs alphanumériques : + , \* , len

```
mavromatis:~$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> chaine = "Bonjour"
>>> chaine
'Bonjour'
>>> chaine + " tout le monde"
'Bonjour tout le monde'
>>> chaine * 3
'BonjourBonjourBonjour'
>>> len ( chaine )
7
>>> []
```

# Opérateurs

- On ne peut pas faire n'importe quoi !

```
mavromatis:~$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> "titi" + 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> "titi" + str ( 3 )
'titit3'
>>> a = 3
>>> "titi" = a
File "<stdin>", line 1
SyntaxError: can't assign to literal
>>> "titi" + str ( a )
'titit3'
>>> []
```

- L'instruction **str** convertit une valeur numérique en une chaîne de caractère

# Entrées / Sorties

- Il est naturel que l'utilisateur interagisse avec le programme

- Un programme peut **afficher** des informations : **print (a)**
- Affichage « formaté »

```
Last login: Wed Mar  5 12:14:53 on ttye001
mavromatis:~$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> nombreDiapos = 1000
>>> print ("Il y a %d diapositives dans ce cours !" %(nombreDiapos))
Il y a 1000 diapositives dans ce cours !
>>> []
```

# Entrées / Sorties

- Il est naturel que l'utilisateur interagisse avec le programme
  - Un programme peut **lire** des informations : **a = input ("Entrez un nombre : ")**
  - Les informations saisies sont considérées comme une chaîne de caractères
  - Deux fonctions de conversion :
    - **int ( a )** convertit la chaîne de caractère a en nombre entier
    - **float ( a )** convertit la chaîne de caractère a en nombre réel

# Opérateurs de comparaison

- Les opérateurs de comparaison
  - **==** : égal à
  - **!=** : différent de
  - **<** : strictement plus petit
  - **>** : strictement plus grand
  - **<=** : plus petit ou égal
  - **>=** : plus grand ou égal
- Le résultat d'un opérateur de comparaison est de type **bool** (booléen). Sa valeur est soit **True** soit **False**.

# Opérateurs logiques

- Parfois, il n'est pas possible d'exprimer une condition simple
  - Un âge est compris entre 20 et 30 : (age >=20) **and** (age <= 30)
- Opérateurs logiques
  - **and** : la condition A **and** B est **True** si à la fois A est **True** et B est **True**
  - **or** : la condition A **or** B est **True** si A ou bien B ou bien les deux sont **True**
  - **not** : la condition **not** A est **True** si la condition A est **False** (et inversement)

# Opérateurs logiques

- Les opérations logiques sont souvent représentées dans une table de vérité
- On utilise souvent 1 pour représenter **True** et 0 pour représenter **False**

| A | B | not A | A and B | A or B |
|---|---|-------|---------|--------|
| 0 | 0 | 1     | 0       | 0      |
| 0 | 1 | 1     | 0       | 1      |
| 1 | 0 | 0     | 0       | 1      |
| 1 | 1 | 0     | 1       | 1      |

# Opérateurs logiques

- Distributivité
  - $A \text{ and } (B \text{ or } C) == (A \text{ and } B) \text{ or } (A \text{ and } C)$
  - $A \text{ or } (B \text{ and } C) == (A \text{ or } B) \text{ and } (A \text{ or } C)$
- Priorité des opérateurs logiques
  - and est prioritaire sur or
  - not est prioritaire sur or et sur and
  - **Le plus simple reste d'utiliser les parenthèses !**

# Opérateurs logiques et conditions

- Ces opérations permettent d'exprimer un « état » en fonction de conditions
- Par exemple, on veut savoir si on doit acheter un gâteau au chocolat. On choisit de décider en fonction de l'argent qu'il nous reste, de notre faim, de la gentillesse du pâtissier qui souhaite nous offrir un gâteau

# Opérateurs logiques et conditions

- Nos variables booléennes
  - a indique si on a de l'argent
  - b indique si on a faim
  - c indique si le pâtissier est sympa
  - d indique si on va se régaler
- Au final, on va se régaler si on a faim et qu'on a de l'argent ou bien juste si le pâtissier est sympa (même si on a plus faim !)
- $d = (a \text{ and } b) \text{ or } c$

| a | b | c | d |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

# Affecter n'est pas comparer

- Une affectation a un effet (change la valeur de la variable) et n'a pas de valeur (ne renvoie pas de valeur).
- Une comparaison a une valeur (renvoie une valeur True, False) et n'a pas d'effet (aucune variable n'est modifiée).

```
mavromatis: $ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> a = 1
>>> print a
1
>>> b = a
>>> print b
1
>>> print a
1
>>> a == b
True
>>> b = 2
>>> a == b
False
>>> []
```

## Affectations multiples

- Affecter plusieurs variables par la même valeur.

**a = b = c = 0**

- Affecter plusieurs variables en même temps.

**a , b = 0, 1**

**m , n = n , m + n**

## Opérateurs et expressions

- Expression
  - Assemblage de constantes, de variables et d'opérateurs
  - Synthèse précise
  - Renvoie une valeur

volume = **surfaceBase \* hauteur / 3**

- Tous les éléments d'une expression doivent-êtré définis

## Opérateurs : priorité, parenthèses

- Les opérateurs sont hiérarchisés (\* et / plus forts que + et - ; **and** plus fort que **or** ...).
- Les parenthèses rendent l'expression qu'elles encadrent plus prioritaire que le reste.

**10 + 5 \* 3**

**( 10 + 5 ) \* 3**

## Opérateurs : priorité, parenthèses

- Opérateurs de même priorité
  - Associativité de chaque opérateur

**100 - 10 - 1**

**100 - ( 10 - 1 )**

**a = b = c**

**a = b = c**

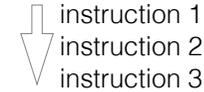
# Importer des modules

```
mavromatis:~$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> sqrt(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
>>> import math
>>> sqrt(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
>>> math.sqrt(4)
2.0
>>> []

Last login: Thu Mar 6 21:51:25 on tty001
mavromatis:~$ python
Python 2.7.5 (default, Aug 25 2013, 00:04:04)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from math import *
>>> sqrt(4)
2.0
>>> []
```

# Flux d'instructions

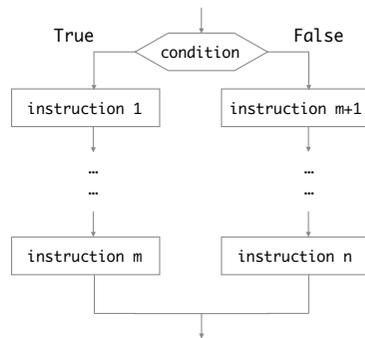
- Les instructions sont exécutées dans l'ordre où elles sont écrites (séquentiellement).



- Il existe des instructions de rupture de séquences (ou de contrôle).
- **if** : selon une condition, choisir un groupe d'instruction parmi deux.
- **while** : tant qu'une condition est vraie, exécuter un groupe d'instructions.
- **for** : pour chaque membre d'une collection, exécuter un groupe d'instructions.

# Instruction conditionnelle

```
if condition :
    instruction 1
    ...
    instruction m
else :
    instruction m+1
    ...
    instruction n
```



- Expression booléenne : True / False
- L'indentation définit un bloc d'instructions

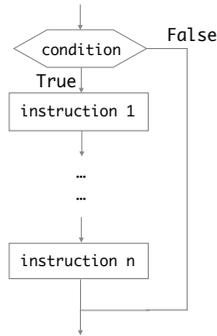
# Par exemple

```
racine_carre.py
1 # coding=utf8
2
3 import math
4 a = float ( input ( "a ? " ) )
5 if a >= 0 :
6     r = math.sqrt ( a )
7     print ( " La racine de %f est %f." %(a,r))
8 else :
9     print ( " La valeur saisie ne peut pas être négative.")
10 print ( "Fin du programme." )
11

mavromatis:~$ python racine_carre.py
a ? 23
La racine de 23.000000 est 4.795832.
Fin du programme.
mavromatis:~$ python racine_carre.py
a ? -10
La valeur saisie ne peut pas être négative.
Fin du programme.
mavromatis:~$
```

# Instruction conditionnelle

```
if condition :  
    instruction 1  
    ...  
    instruction n  
...
```



# Par exemple

```
min_max.py  
1 # coding=utf8  
2  
3 min = int ( input ( "Minimum ? " ) )  
4 max = int ( input ( "Maximum ? " ) )  
5 if min > max :  
6     print ( "Le minimum est plus grand que le maximum !!!" )  
7 print ( "Fin du programme." )  
  
Bureau - bash  
mavromatis@marie:~$ python min_max.py  
Minimum ? 10  
Maximum ? 20  
Fin du programme.  
mavromatis@marie:~$ python min_max.py  
Minimum ? 50  
Maximum ? 20  
Le minimum est plus grand que le maximum !!!  
Fin du programme.  
mavromatis@marie:~$
```

# Par exemple

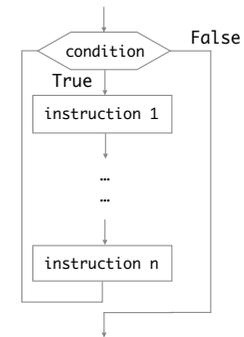
- Si ... Si .. Si ... Si ... Sinon Si ...
- Merci l'indentation !

```
conditions_embriquées.py  
1 # coding=utf8  
2  
3 embranchement = input ( "Embranchement ? " )  
4 classe = input ( "Classe ? " )  
5 ordre = input ( "Ordre ? " )  
6 famille = input ( "Famille ? " )  
7  
8 if embranchement == "vertébrés":  
9     if classe == "mammifères":  
10         if ordre == "carnivores":  
11             if famille == "félins":  
12                 print ( "C'est peut-être un chat" )  
13             print ( "C'est en tous cas un mammifère" )  
14         elif classe == "oiseaux":  
15             print ( "C'est peut-être un canari" )  
16 print ( "La classification des animaux est complexe." )
```

# Instruction répétitive

- Boucle **Tant que ... Faire ...**

```
while condition :  
    instruction 1  
    ...  
    instruction n  
...
```



## Par exemple

- $k \geq 1$  donné, on cherche le plus grand nombre  $2^n$  tel que  $2^n \leq k$
- On examine les nombres  $p = 1, 2, 4, 8, 16 \dots$  tant que  $p \leq k$

```
ppp_gron.py
1 # coding=utf8
2
3 k = int ( input ( "k ? " ))
4 p = 1
5 while p <= k :
6     p = 2 * p
7     p = p / 2 # p > k, la valeur précédente est la bonne
8     print ( "p : %d" % (p))
```

## Par exemple

- $k \geq 1$  donné, on cherche le plus grand nombre  $n$  tel que  $2^n \leq k$
- On examine les nombres  $p = 1, 2, 4, 8, 16 \dots$  tant que  $p \leq k$ , on compte le nombre de tours dans la boucle.

```
ppp_gron_tours.py
1 # coding=utf8
2
3 k = int ( input ( "k ? " ))
4 p = 1
5 n = 0 # n est notre compteur
6 while p <= k :
7     p = 2 * p
8     n = n + 1
9     p = p / 2 # p > k, la valeur précédente est la bonne
10    n = n - 1
11    print ( "p : %d et n : %d" % (p,n))
```

## Reprenons le PGCD

- Algorithme d'Euclide : Etant donnés deux entiers, retrancher le plus petit au plus grand et recommencer jusqu'à ce que les deux nombres soient égaux. La valeur obtenue est le plus grand diviseur commun.

```
mavromatis:~/Documents/Polytech/MARSEILLE/python$ python pgcd.py
Calcul de PGCD
Entrez un nombre : 600
Entrez un second nombre : 225
Le PGCD de 600 et de 225 est 75
mavromatis:~/Documents/Polytech/MARSEILLE/python$
```

```
pgcd.py
1 print ("Calcul de PGCD")
2
3 a = int (input ("Entrez un nombre : "))
4 b = int (input ("Entrez un nombre : "))
5
6 while a != b :
7     if a > b :
8         a = a - b
9     else :
10        b = b - a
11
12 print ("Le PGCD de %d et de %d est %d" % (a, b, c))
13
```

## Traitement d'une suite de données

- Appliquer un même traitement à des données acquises successivement jusqu'à la rencontre d'une donnée invalide (qui signale la fin de la saisie).
- Acquérir la première donnée
- Tant que la donnée est valide
  - Réaliser le traitement
  - Acquérir la donnée suivante

## Par exemple

- Calculer la somme d'une suite de nombres positifs saisis au clavier. Un nombre négatif signale la fin de la saisie.

```
somme_positifs.py
1 # coding=utf8
2
3 somme = 0
4 n = int ( input ( "n ? " ) )
5 while n > 0 :
6     somme = somme + n
7     n = int ( input ( "n ? " ) )
8 print ( "Somme : %d" %(somme))
9
```

## Par exemple

- Calculer la somme d'une suite de nombres positifs saisis au clavier, compter combien de nombre ont été saisis et combien de ces nombres étaient plus grand que 100. Un nombre négatif signale la fin de la saisie.

```
somme_positifs_plus.py
1 # coding=utf8
2
3 somme = 0
4 nombreTotal = 0
5 nombreGrands = 0
6
7 n = int ( input ( "n ? " ) )
8 while n > 0 :
9     somme = somme + n
10    nombreTotal = nombreTotal + 1
11    if n > 100 :
12        nombreGrands = nombreGrands + 1
13    n = int ( input ( "n ? " ) )
14
15 print ( "Somme : %d" %(somme))
16 print ( "Vous avez saisi %d nombre(s) dont %d nombre(s) plus grand(s) que 100." \
17        %(nombreTotal, nombreGrands))
```

## Parcourir une collection

- Collection : string, tuple, list ... range
- On reviendra sur les structures de données plus tard
- Une liste : données séquentielles en mémoire
  - maListeDeNombres = [ 10, 20, -5, 1.3, 100 ]
  - La longueur de la liste est obtenue par l'appel à : len (maListeDeNombres)
  - L'accès est indexé de 0 à len (maListeDeNombres) - 1
  - Plusieurs opérations possibles sur les listes ...

## Parcourir une collection

- Boucle **Pour chaque élément** d'une collection **faire ...**

```
i = 0
while i < len ( collection )
    traiter l'élément i de ma collection
    i = i + 1
```

```
for e in collection
    traiter e
```

# Parcourir une collection

- Boucle **Pour chaque élément** d'une collection **faire ...**

```
1 #coding=utf8
2
3 # Calcul de la somme des éléments d'une liste de nombres
4 maListeDeNombres = [ 10, 20, -5, 1.3, 100 ]
5
6 somme = 0
7 n = len( maListeDeNombres )
8 i = 0
9
10 while i < n :
11     somme = somme + maListeDeNombres[i]
12     i = i + 1
13
14 print( "La somme des éléments est : %f" %(somme))
```

```
1 #coding=utf8
2
3 # Calcul de la somme des éléments d'une liste de nombres
4 maListeDeNombres = [ 10, 20, -5, 1.3, 100 ]
5
6 somme = 0
7
8 for e in maListeDeNombres :
9     somme = somme + e
10
11 print( "La somme des éléments est : %f" %(somme))
```

# Parcourir une collection

- Boucle **Pour i de n à m par pas de p faire ...**

- range ( [start,] stop [,step] )      **itérateur**

```
Python 3.3.5 (v3.3.5:62cf4e77e785, Mar 9 2014, 01:12:15)
[[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> range(10)
range(0, 10)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

```
1 # C'est beau !
2 for i in range(8) :
3     print(" * " * i, "Joli", " " * (7-i) * 2, "Joli!")
4
```

# Parcourir une collection

- Recherche dans une liste
- Déterminer **la présence** et le **rang** d'une valeur dans une liste

```
1 #coding=utf8
2
3 # Cherche une valeur dans une liste et détermine son rang
4 maListeDeNombres = [ 10, 20, -5, 1.3, 100, 9, 1, 72 ]
5 valeur = 9
6
7 rang = 0
8 n = len( maListeDeNombres )
9 while rang < n and valeur != maListeDeNombres[rang] :
10     rang = rang + 1
11
12 if rang < n :
13     print("La valeur %f existe au rang %d." %(valeur, rang))
14 else :
15     print("La valeur n'a pas été trouvée !")
```

```
1 #coding=utf8
2
3 # Cherche une valeur dans une liste et détermine son rang
4 maListeDeNombres = [ 10, 20, -5, 1.3, 100, 9, 1, 72 ]
5 valeur = 9
6 rang = -1
7
8 for i in range(len(maListeDeNombres)) :
9     if (valeur == maListeDeNombres[i]) :
10         rang = i
11         break
12
13 if rang == 0 :
14     print("La valeur %f existe au rang %d." %(valeur, i))
15 else :
16     print("La valeur n'a pas été trouvée !")
```

# Structures de données

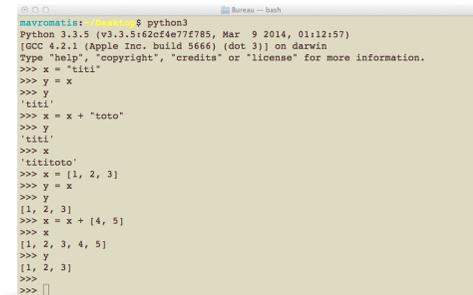
- Entiers, réels, booléens
- Absence de valeur : None, sont type NoneType
- string : chaîne de caractères (immuable)
- **list** : liste d'éléments. Accès indexé. [ a, b, c ]
- **tuple** : n-uplet. Immuable, Accès indexé. Optimisé en mémoire. ( a, b, c )
- **set** : ensemble d'éléments uniques. Pas d'indexation. set ( [ a, b, b ] )
- **dict** : couples (clé, valeur). Accès par clé. { r : Riri, f : Fifi, l : Loulou }

# Gestion de la mémoire

- Python s'en occupe, la mémoire semble infinie
- Python s'occupe donc de gérer le manque de mémoire

```
maListe = [ 1, 5, 4 , 10, 8399, 10]  
...  
...  
maListe = [ "Riri" , "Fifi" , "Loulou" ]  
...  
...
```

# Différent d'autres langages



```
maurmatias: ~ $ python3  
Python 3.3.5 (v3.3.5:62cf4e7f785, Mar  9 2014, 01:12:57)  
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>> x = "titi"  
>>> y = x  
>>> y  
'titi'  
>>> x = x + "toto"  
>>> y  
'titi'  
>>> x  
'tittoto'  
>>> x = [1, 2, 3]  
>>> y = x  
>>> y  
[1, 2, 3]  
>>> x = x + [4, 5]  
>>> x  
[1, 2, 3, 4, 5]  
>>> y  
[1, 2, 3]  
>>> []
```

# list

- Démo indexation, concaténation, sous-liste, modification ...

# list

- Quelques méthodes sur les listes

# tuple

- Démonstration

# set

- Démonstration

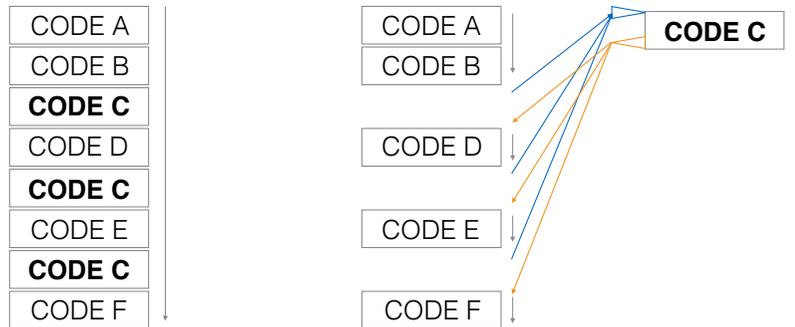
# dict

- Démonstration

# Notion de fonction

- Fonction, procédure, méthode
- Partie de code informatique « séparée »
- Avantages :
  - Mettre en évidence un traitement en précisant les données et les résultats
  - Faciliter la « maintenance » du code
    - Eviter des erreurs liées aux modifications d'un code « dupliqué »

# Notion de fonction



# Notion de fonction

- Par exemple :

```
premiereFonction.py
1 def proportion ( chaine, caractere ) :
2     """Fréquence d'appartion du caractere dans la chaine.
3
4     chaine est une chaine de caractere non vide
5     caractere est un caractere unique"""
6
7     n = len( chaine )
8     k = chaine.count( caractere )
9
10    return k/n
11
```

# Notion de fonction

- Par exemple :

```
testPremiereFonction.py
1 def proportion ( chaine, caractere ) :
2     """Fréquence d'appartion du caractere dans la chaine.
3
4     chaine est une chaine de caractere non vide
5     caractere est un caractere unique"""
6
7     n = len( chaine )
8     k = chaine.count( caractere )
9
10    return k/n
11
12    texte = "Bonjour à tous !"
13    nbO = proportion( texte, 'o')
14    nbU = proportion( texte, 'u')
15
16    print( "Texte : ", texte)
17    print( "Proportion de o : ", nbO)
18    print( "Proportion de u : ", nbU)
19
20
```

# Notion de fonction

- Avantages :

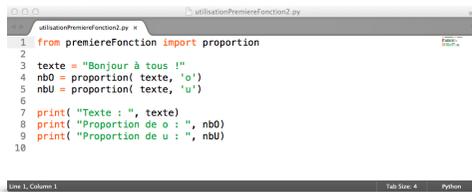
- ...
- Réutiliser un même code pour plusieurs programmes

```
utilisationPremiereFonction.py
1 import premiereFonction
2
3 texte = "Bonjour à tous !"
4 nbO = premiereFonction.proportion( texte, 'o')
5 nbU = premiereFonction.proportion( texte, 'u')
6
7 print( "Texte : ", texte)
8 print( "Proportion de o : ", nbO)
9 print( "Proportion de u : ", nbU)
10
```

# Notion de fonction

- Avantages :

- ...
- Réutiliser un même code pour plusieurs programmes

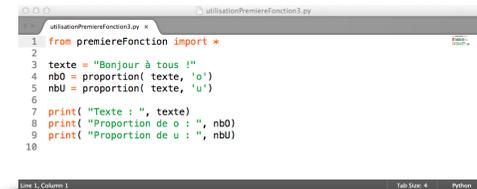


```
1 from premiereFonction import proportion
2
3 texte = "Bonjour à tous !"
4 nbO = proportion( texte, 'o')
5 nbU = proportion( texte, 'u')
6
7 print( "Texte : ", texte)
8 print( "Proportion de o : ", nbO)
9 print( "Proportion de u : ", nbU)
10
```

# Notion de fonction

- Avantages :

- ...
- Réutiliser un même code pour plusieurs programmes

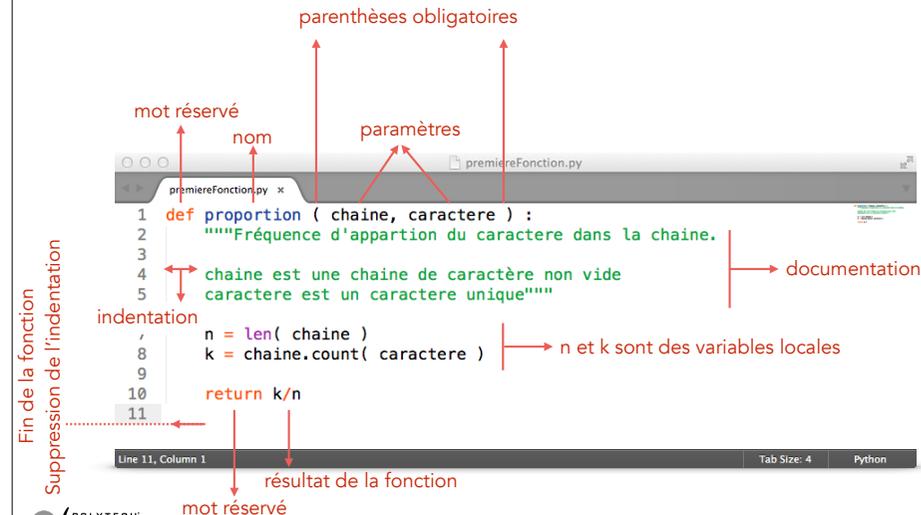


```
1 from premiereFonction import *
2
3 texte = "Bonjour à tous !"
4 nbO = proportion( texte, 'o')
5 nbU = proportion( texte, 'u')
6
7 print( "Texte : ", texte)
8 print( "Proportion de o : ", nbO)
9 print( "Proportion de u : ", nbU)
10
```

# Notion de fonction

- Avantages :

- Mettre en évidence un traitement en précisant les données et les résultats
- Faciliter la « maintenance » du code
  - Eviter des erreurs liées aux modifications d'un code « dupliqué »
- Réutiliser un même code pour plusieurs programmes
- Décomposer un problème en sous-tâches



```
1 def proportion ( chaîne, caractere ) :
2     """Fréquence d'appartion du caractere dans la chaîne.
3
4     chaîne est une chaîne de caractère non vide
5     caractere est un caractère unique"""
6
7     n = len( chaîne )
8     k = chaîne.count( caractere )
9
10    return k/n
11
```

Annotations:

- mot réservé: def
- nom: chaîne, caractere
- paramètres: chaîne, caractere
- parenthèses obligatoires: ( ),
- documentation: """Fréquence d'appartion du caractere dans la chaîne. chaîne est une chaîne de caractère non vide caractere est un caractère unique"""
- n et k sont des variables locales: n = len( chaîne ), k = chaîne.count( caractere )
- return k/n: résultat de la fonction
- mot réservé: return
- Fin de la fonction: 11
- Suppression de l'indentation: indentation

# Appel d'une fonction

```
testPremiereFonction.py
1 def proportion ( chaine, caractere ) :
2     """Fréquence d'appartion du caractere dans la chaine.
3
4     chaine est une chaîne de caractère non vide
5     caractere est un caractère unique"""
6
7     n = len( chaine )
8     k = chaine.count( caractere )
9
10    return k/n
11
12    texte = "Bonjour à tous !"
13    nbO = proportion( texte, 'o')
14    nbU = proportion( texte, 'u')
15
16    print( "Texte : ", texte)
17    print( "Proportion de o : ", nbO)
18    print( "Proportion de u : ", nbU)
19
20
```

# Appel d'une fonction

- Toujours des parenthèses, même sans arguments
- A priori, le même nombre de paramètres et d'arguments
- A priori, dans l'ordre
- L'instruction **return** indique le résultat de la fonction

# Passage des arguments

- Les arguments sont transmis comme si des affectations étaient réalisées :
  - chaine = texte
  - caractere = 'o'
- N'importe quelle expression peut-être un argument.
- Un argument ne peut pas être modifié par la fonction (type simple)

# Passage des arguments

```
mavromatis:~/Desktop$ python3
Python 3.3.5 (v3.3.5:62cf4e7f785, Mar  9 2014, 01:12:57)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> def double(n):
...     n = 2 * n
...     return n
...
>>> p = 5
>>> print(p)
5
>>> double(p)
5
>>> print(p)
5
>>> []

mavromatis:~/Desktop$ python3
Python 3.3.5 (v3.3.5:62cf4e7f785, Mar  9 2014, 01:12:57)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> def double(n):
...     n = 2 * n
...     return n
...
>>> p = 5
>>> print(p)
5
>>> double(p)
10
>>> print(p)
5
>>> p = double(p)
10
>>> print(p)
10
>>> []
```

# Variables locales

- A priori, une variable définie dans une fonction est locale :
- Visible uniquement dans sa fonction (portée réduite).
- Toute variable non locale de même nom est masquée.
- Toute variable locale est détruite lorsque la fonction se termine.

# Variables locales

- A priori, une variable définie dans une fonction est locale :
- Visible uniquement dans sa fonction (portée réduite).
- Toute variable non locale de même nom est masquée.
- Toute variable locale est détruite lorsque la fonction se termine.

# Variables locales

- Portée réduite

```
testPremiereFonction.py
1 def proportion ( chaine, caractere ) :
2     """Fréquence d'appartion du caractere dans la chaine.
3
4     chaine est une chaîne de caractère non vide
5     caractere est un caractère unique"""
6
7     n = len( chaine )
8     k = chaine.count( caractere )
9     return k/n
10
11
12 texte = "Bonjour à tous !"
13 nb0 = proportion( texte, 'o')
14 nbU = proportion( texte, 'u')
15
16 print( "Texte : ", texte)
17 print( "Proportion de o : ", nb0)
18 print( "Proportion de u : ", nbU)
19
20
```

→ n et k ne sont pas accessibles ici

# Variables locales

- Masquage, durée de vie

```
variableLocale.py
1 a = 5
2
3 def jolieFonction ():
4     a = 10
5     print ("jolieFonction --> a = ", a)
6
7
8 print ("Valeur initiale de a : ", a)
9 jolieFonction()
10 print ("Après la fonction a = ", a)
11
```

```
Bureau - bash
mavromatis:~/Bureau$ python3 variableLocale.py
Valeur initiale de a : 5
jolieFonction --> a = 10
Après la fonction a = 5
mavromatis:~/Bureau$
```

# Variables globales

- Les variables globales sont consultables dans une fonction (sauf masquage bien sûr !)

```
variableGlobaleFonction.py
1 # La valeur de la TVA est écrite dans la fonction
2 def calculPrixTTC (prixHT):
3     return prixHT * 1.2
4
5 # La valeur de la TVA est passée à la fonction
6 def calculPrixTTC (prixHT, TVA):
7     return prixHT * (1 + TVA / 100)
8
9 # TVA est une variable globale
10 TVA = 20
11 # La variable globale est utilisée dans la fonction
12 def calculPrixTTC (prixHT):
13     return prixHT * (1 + TVA / 100)
14
```

# Variables globales

- Il est possible d'indiquer que l'on souhaite modifier une variable globale.

```
variableGlobaleFonction2.py
1 nbAppels = 0
2
3 def maFonction():
4     global nbAppels
5     nbAppels = nbAppels + 1
6
7 for i in range(20):
8     maFonction()
9
10 print ("La fonction a été appelée", nbAppels, "fois")
```

# Le mot clé **return**

- Précise le résultat que la fonction renvoie
- Produit la terminaison de la fonction

```
return.py
1 # Recherche d'un élément d'une liste égal à une valeur donnée
2
3 def indice(valeur, liste):
4     """Renvoie l'indice du premier élément de la liste qui a la valeur"""
5
6     for i in range(len(liste)):
7         if valeur == liste[i]:
8             return i
9
10    return -1
11
12 l = [1, 2, 3, 4, 55, 66, 777, 888]
13 print (indice(4, l))
14 print (indice(777, l))
```

# Fonction sans résultat

- **Pas** d'instruction **return**
- Produit un « effet secondaire » ... par exemple on affiche quelque chose.

```
fonctionSansRetour.py
1 def afficherListeVerticale (maListe):
2     i = 0
3     for e in maListe:
4         print (e)
5         i = i + 1
6
7 l = ["un", "deux", "trois", "quatre"]
8 print (l)
9 afficherListeVerticale(l)
```

## Arguments avec mots-clés

- Pour simplifier l'utilisation d'une fonction.
- Il est possible de passer outre l'ordre des arguments
  - Si on utilise cette technique pour un argument, il faut spécifier des mots-clés pour tous les arguments suivants.

```
def calcul (masse, surface, epaisseur, temp)  
r = calcul (100, 2, 0.5, 40)  
r = calcul (surface = 2, masse = 100, temp = 40, epaisseur = 0.5)  
r = calcul (100, 2, temp = 40, epaisseur = 0.5)  
r = calcul (100, surface = 2, 0.5, 40)
```

## Valeur par défaut des paramètres

- Pour simplifier l'utilisation d'une fonction.
- La définition de la fonction indique les valeurs de certains arguments.
  - Si un paramètre a une valeur par défaut alors ceux qui le suivent doivent également avoir des valeurs par défaut.
- Ces arguments peuvent-être omis à l'appel de la fonction.

## Valeur par défaut des paramètres

```
def calcul (masse, surface, epaisseur = 0.1, temp = 37)
```

- Appels possibles

```
r = calcul (100, 2, 0.5, 40)  
r = calcul (100, 2, 0.5)  
r = calcul (100, 2)
```

- Valeurs par défaut + appel avec mot-clé

```
r = calcul(masse=100, surface=2, temp=10)
```

## Quelques liens

- <http://docs.python.org>
- <http://docs.python.org/3/library/index.html>
- <http://docs.python.org/3/tutorial/>
- <http://inforef.be/swi/python.htm>

# Jeu du morpion

- **Initialiser** les éléments du jeu
- Faire jouer le joueur
- Tant que la grille n'est pas pleine et que ce n'est pas un coup gagnant
  - Changer de joueur
  - Faire jouer le joueur
- Dire s'il y a un gagnant

# Jeu du morpion

- **Initialiser** les éléments du jeu
  - Initialiser la grille de jeu, c'est à dire créer un tableau à deux dimensions avec des cases 'vides'
  - Créer deux joueurs, c'est à dire proposer deux symboles pour jouer 'X' ou 'O'
  - Choisir le joueur qui commence par tirage au sort par exemple
  - Afficher la grille

# Jeu du morpion

- **Initialiser** les éléments du jeu
  - **Initialiser la grille de jeu**, c'est à dire créer un tableau à deux dimensions avec des cases 'vides'
  - **Créer deux joueurs**, c'est à dire proposer deux symboles pour jouer 'X' ou 'O'
  - **Choisir le joueur qui commence** par tirage au sort par exemple
  - **Afficher la grille**
- Faire jouer le joueur

# Jeu du morpion

- Faire jouer le joueur
  - Comme la grille est un tableau à deux dimensions, il est nécessaire de **demandé au joueur de saisir deux entiers : un numéro de ligne et un numéro de colonne.**
  - Si la position en libre alors placer le pion
  - **Afficher la grille**

**La case  
désignée par  
la position ne  
contient rien**

**Affecter le  
symbole du  
joueur dans la  
case choisie**

# Jeu du morpion

- **Initialiser** les éléments du jeu
  - **Initialiser la grille de jeu**, c'est à dire créer un tableau à deux dimensions avec des cases 'vides'
  - **Créer deux joueurs**, c'est à dire proposer deux symboles pour jouer 'X' ou 'O'
  - **Choisir le joueur qui commence** par tirage au sort par exemple
  - **Afficher la grille**
- Faire joueur le jouer
  - **Demander au joueur de saisir deux entiers : un numéro de ligne et un numéro de colonne.**
  - **Si la position en libre** (la case désignée par la position ne contient rien) **alors placer le pion** (affecter le symbole du joueur dans la case choisie)
  - **Afficher la grille**

# Jeu du morpion

- **Initialiser** les éléments du jeu
  - **Initialiser la grille de jeu**, c'est à dire créer un tableau à deux dimensions avec des cases 'vides'
  - **Créer deux joueurs**, c'est à dire proposer deux symboles pour jouer 'X' ou 'O'
  - **Choisir le joueur qui commence** par tirage au sort par exemple
  - **Afficher la grille**
- Faire joueur le jouer
  - **Demander au joueur de saisir deux entiers : un numéro de ligne et un numéro de colonne.**
  - **Si la position en libre** (la case désignée par la position ne contient rien) **alors placer le pion** (affecter le symbole du joueur dans la case choisie)
  - **Afficher la grille**
- **Tant que la grille n'est pas pleine et que ce n'est pas un coup gagnant**

# Jeu du morpion

- **Tant que la grille n'est pas pleine et que ce n'est pas un coup gagnant**

↓  
**Il reste au moins une case vide dans la grille**

↓  
**Trois symboles ne sont pas alignés**

# Jeu du morpion

- **Initialiser** les éléments du jeu
  - **Initialiser la grille de jeu**, c'est à dire créer un tableau à deux dimensions avec des cases 'vides'
  - **Créer deux joueurs**, c'est à dire proposer deux symboles pour jouer 'X' ou 'O'
  - **Choisir le joueur qui commence** par tirage au sort par exemple
  - **Afficher la grille**
- Faire joueur le jouer
  - **Demander au joueur de saisir deux entiers : un numéro de ligne et un numéro de colonne.**
  - **Si la position en libre alors placer le pion**
  - **Afficher la grille**
- **Tant que la grille n'est pas pleine (il reste au moins une case vide dans la grille) et que ce n'est pas un coup gagnant (trois symboles ne sont pas alignés)**

# Jeu du morpion

- **Initialiser** les éléments du jeu
  - **Initialiser la grille de jeu**, c'est à dire créer un tableau à deux dimensions avec des cases 'vides'
  - **Créer deux joueurs**, c'est à dire proposer deux symboles pour jouer 'X' ou 'O'
  - **Choisir le joueur qui commence** par tirage au sort par exemple
  - **Afficher la grille**
- Faire jouer le joueur
  - **Demander au joueur de saisir deux entiers : un numéro de ligne et un numéro de colonne.**
  - **Si la position en libre alors placer le pion**
  - **Afficher la grille**
- **Tant que la grille n'est pas pleine (il reste au moins une case vide dans la grille) et que ce n'est pas un coup gagnant (trois symboles ne sont pas alignés)**
  - **Changer de joueur (si c'est le joueur 'X' qui joue alors c'est au joueur 'O' sinon c'est au joueur 'X' de jouer)**
  - **Faire jouer le joueur**
- Dire s'il y a un gagnant

# Des fonctions

- **Initialiser** les éléments du jeu
  - **INITIALISERGRILLE**, c'est à dire créer un tableau à deux dimensions avec des cases 'vides'
  - **Créer deux joueurs**, c'est à dire proposer deux symboles pour jouer 'X' ou 'O'
  - **Choisir le joueur qui commence** par tirage au sort par exemple
  - **AFFICHERGRILLE**
- **Faire jouer le joueur**
  - **Demander au joueur de saisir deux entiers : un numéro de ligne et un numéro de colonne.**
  - **Si ESTLIBRE alors PLACERPION**
  - **AFFICHERGRILLE**
- **Tant que not GRILLEPLEINE (il reste au moins une case vide dans la grille) et que not COUPGAGNANT (trois symboles ne sont pas alignés)**
  - **Changer de joueur (si c'est le joueur 'X' qui joue alors c'est au joueur 'O' sinon c'est au joueur 'X' de jouer)**
  - **Faire jouer le joueur**
- Dire s'il y a un gagnant