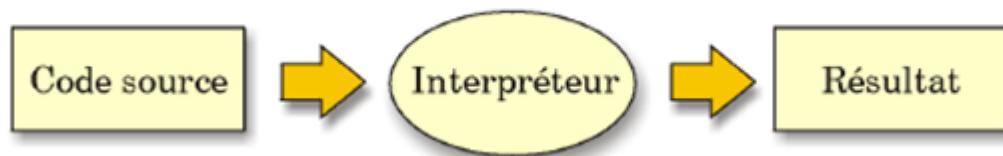


# Python : Introduction

A. Malek TOUMI UV1.1

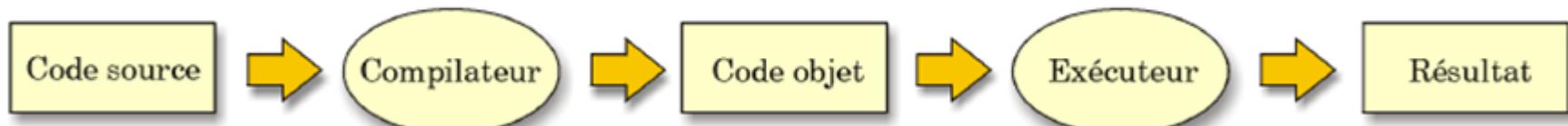
# Introduction : Langage évolué, compilation, interprétation

- un langage évolué permet de représenter
    - les nombres par leur notation usuelle
    - les instructions par des opérateurs et des mots-clés expressifs et faciles à lire
  - avant exécution, instructions et données doivent être converties en binaire
  - interprétation (« code source » est le programme que vous avez écrit) :



*L'interpréteur lit  
le code source ...*

- compilation :



*Le compilateur lit le code source ... ... et produit un code objet (binaire).*

# Introduction : Mode immédiat

- la boucle supérieure (*top-level loop*) :

- *read* : la machine affiche une invite (>>>) et l'utilisateur tape une expression correcte
- *eval* : la machine obtient la *valeur* de cette expression
- *print* : la machine affiche cette *valeur*

- exemple :

```
>>> 2**8      # 2 à la puissance 8
256
>>>
```

- exception : si la valeur est **None** (convention), rien n'est affichée

```
>>> nbr = 8
>>>
```

- mode immédiat : bon moyen d'approcher le langage et tester ses idées

Simplement avec la commande «python» sur une console

*un commentaire*

# Introduction : Données, opérateurs, expressions

- *données* : données initiales, résultats, valeurs intermédiaires
- types primitifs :

- entiers (sur 32 bits)

0

123

-50

- entiers longs

340282366920938463463374607431768211456

1L

- nombres non entiers (on dit « flottants »)

1.5

-0.33333333333333331

0.166054018e-23

- chaînes de caractères

"J'aime Python"

- quelques valeurs conventionnelles

True (vrai), False (faux), None (absence de valeur)

# Introduction : Données, opérateurs, expressions

- plusieurs notations

```
>>> "Bonjour"  
'Bonjour'  
>>> "Bonjour" == 'Bonjour'  
True  
>>> "J'ai dit bonjour"  
"J'ai dit bonjour"  
>>> "J'ai dit \"bonjour\""  
'J\'ai dit "bonjour"'  
>>> print "J'ai dit \"bonjour\""  
J'ai dit "bonjour"  
>>> """Ceci est une  
... longue chaine"""  
'Ceci est une\nlongue chaine'  
>>> print """Ceci est une  
... longue chaine"""  
Ceci est une  
longue chaine  
>>>
```

## Chaîne de caractères

```
st = "langage python"  
st = 'langage python'  
st = 'un guillement '  
st = "un guillement \""  
  
st = st.upper ()  
i = st.find ("PYTHON")  
print (i)  
  
print (st.count ("PYTHON"))  
print (st.count ("PYTHON", 9))
```

# idem  
# chaîne contenant un guillement  
# chaîne contenant un guillement, il faut ajouter  
# pour ne pas confondre avec l'autre guillement  
# mise en lettres majuscules  
# on cherche "PYTHON" dans st  
# affiche 8 Version 3.x, écrire print (i),  
# pour la version 2.x, écrire print i  
# affiche 1 Version 3.x : idem print (...)  
# affiche 0 Version 3.x : idem print (...)

# Introduction : Données, opérateurs, expressions

- affecter plusieurs variables par la même valeur

```
>>> a = b = c = 0  
>>>
```

- affecter plusieurs variables en même temps

```
>>> a, b = 0, 1  
>>>
```

- plus fort :

```
>>> a, b = b, a + b  
>>>
```

*couple de variables*

*couple de valeurs*

Variables,  
affectation

# Introduction : Données, opérateurs, expressions

- arithmétiques :

+      -      \*      \*\*      /      %

Opérateurs

```
>>> 2 ** 3      # puissance  
8  
>>> 17 % 5      # modulo (reste du quotient)  
2  
>>>
```

- comparaison :

==      !=      <      <=      >      >=

```
>>> 2 ** 3 == 8  
True  
>>>
```

- logiques :

and      or      not

```
>>> x = 14  
>>> x >= 10 and x <= 20  
True  
>>>
```

- quelques bizarries : + (entre chaînes), % (avec une chaîne), etc.

Opérateurs :  
priorité,  
associativité,  
parenthèses

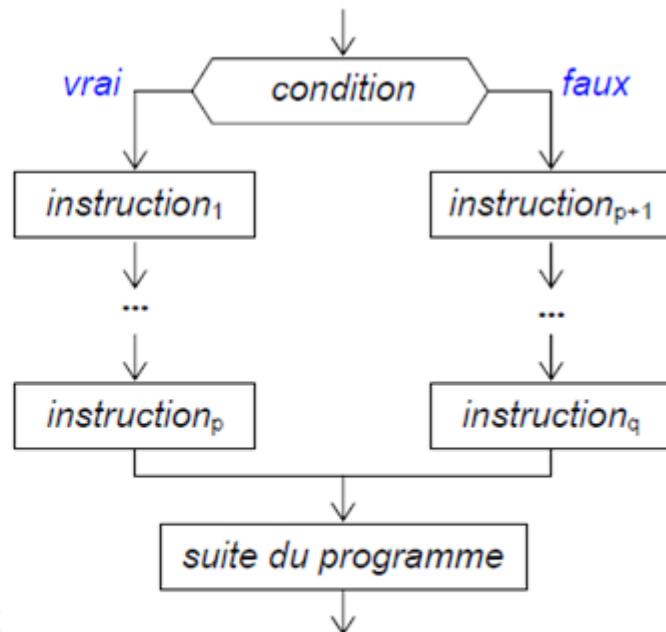
# Modules et bibliothèques

```
>>> sqrt(4)                                sqrt appartient au module math
[...]
NameError: name 'sqrt' is not defined
>>> math.sqrt(4)                            le module math n'est pas connu
[...]
NameError: name 'math' is not defined
>>> import math
>>> math.sqrt(4)
2.0
>>> sqrt(4)                                 importer ne dispense pas
[...]                                       de préfixer, sauf si...
NameError: name 'sqrt' is not defined
>>> from math import *
>>> sqrt(4)
2.0
>>>
```

# Instructions

# Instruction conditionnelle : if, else, elif

```
if condition :  
    instruction1  
    ...  
    instructionp  
else :  
    instructionp+1  
    ...  
    instructionq  
suite du programme
```



- condition est une expression booléenne : sa valeur est **True OU False**
- l'indexation (marge à gauche) joue un rôle syntaxique
  - elle est obligatoire
  - elle suit des règles strictes

# Instruction conditionnelle : Exemple

- exemple :

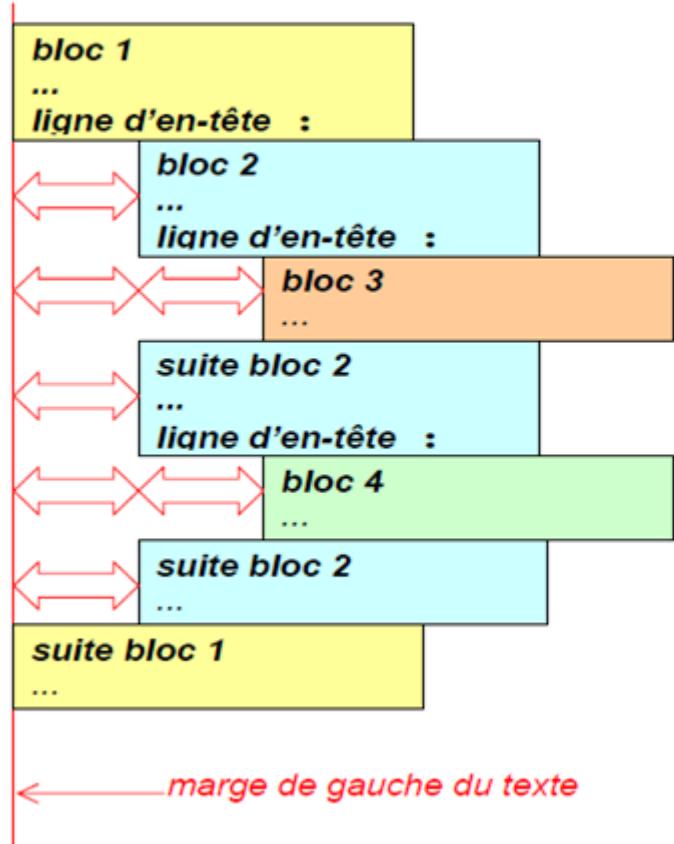
```
racine.py
import math
x = float(raw_input("x? "))
if x >= 0:
    y = math.sqrt(x)
    print "La racine de", x, "est", y
else:
    print "On ne peut prendre la racine d'un nombre négatif!"
print "Au revoir"
```

- exécution :

```
$ python racine.py
x? 25
La racine de 25.0 est 5.0
Au revoir
$ python racine.py
x? -5
On ne peut prendre la racine d'un nombre négatif!
Au revoir
$
```

# Indentation

## L'imbrication des blocs et l'indentation



```
if condition :  
    instruction1  
    ...  
    instructionp  
else :  
    instructionp+1  
    ...  
    instructionq  
suite du programme
```

# Instruction répétitive : While

## Ex. : Parcourir une liste

- données séquentielles logées dans la mémoire : listes

```
liste = [ 20, 0, -12, 5.0, 51 ]
```

- problème : effectuer un traitement sur chaque élément d'une liste.  
Exemple : calculer la somme des termes d'une liste de *nombres*

```
somme = 0
```

```
n = len(liste)
i = 0
print i, somme
while i < n :
    somme = somme + liste[i]
    i = i + 1
    print i, somme
```

*si on est curieux...*

*len = 5*

```
print "somme:", somme
```

20	0	-12	5.0	51
0	1	2	3	4

# Instruction répétitive : for

## Ex. : recherche dans une liste, autre manière

- déterminer la présence et le rang d'une valeur dans une liste

```
liste = [ 20, 0, 12, 5.0, 51, 17, 24, 9, 13 ]
valeur = 12

rang = -1
for i in range(len(liste)):
    if liste[i] == valeur:
        rang = i
        break

if rang >= 0 :
    print "valeur presente - rang:", rang
else :
    print "valeur absente"
```

# Fonction

(ou procédure, méthode)

# Syntaxe

```
mot réservé      nom de la fonction      paramètres formels      parenthèses (obligatoires)
                ↓                         ↓                         ↓                         ↓
def proportion ( chaine, base ) :
    """Fréquence d'apparition de base dans chaine.

    chaine est une chaîne de caractères,
    base doit être un caractère unique.
    Attention: Chaîne ne doit pas être vide
    """
    n = len(chaine)
    k = chaine.count(base)

    return float(k) / n
...
mot réservé      résultat de la fonction
```

indentation

variables locales

chainé de documentation (docstring)

la fin de la définition de la fonction est indiqué par le retour à l'indentation normale

# Utilisation

- définition

```
def proportion ( chaine, base ) :  
    n = len(chaine)  
    k = chaine.count(base)  
    return float(k) / n
```

- appel

```
...  
y = proportion ( adn, "t" )  
...
```

*paramètres ou paramètres formels*

*arguments ou paramètres effectifs*

```
n = len(adn)  
k = chaine.count("t")  
return float(k) / n
```

*précisément, comment **chaine** et **base**  
sont-ils substitués par **adn** et "t" ?*

# Fonction sans résultats (procédure)

- fonction « pure » : renvoie un résultat et n'a aucun effet secondaire
- procédure : ne renvoie pas de résultat, le seul intérêt est l'effet secondaire
- procédure en Python :
  - pas d'instruction `return`, ou de la forme « `return` » (sans expression)
  - la fonction renvoie alors la valeur conventionnelle `None`

```
def lister(uneListe) :
    """Affichage 'vertical' de la liste donnée"""
    i = 0
    for x in uneListe :
        print i, ":", x
        i = i + 1
```

- le programmeur n'a pas indiqué de résultat  $\Rightarrow$  la fonction renvoie `None`.  
Emploi erroné :

```
n = lister([11, 22, 33, 44, 55])      # étourderie !
```

la plupart des utilisations de la valeur de `n` provoqueront une erreur

Suite ... UV 3.1