

Introduction à l'algorithme - L3 Info Com

F. AIT SALAHT & F. DELBOT

farah.aitsalaht@u-paris10.fr

Université Paris Ouest, Nanterre La Défense

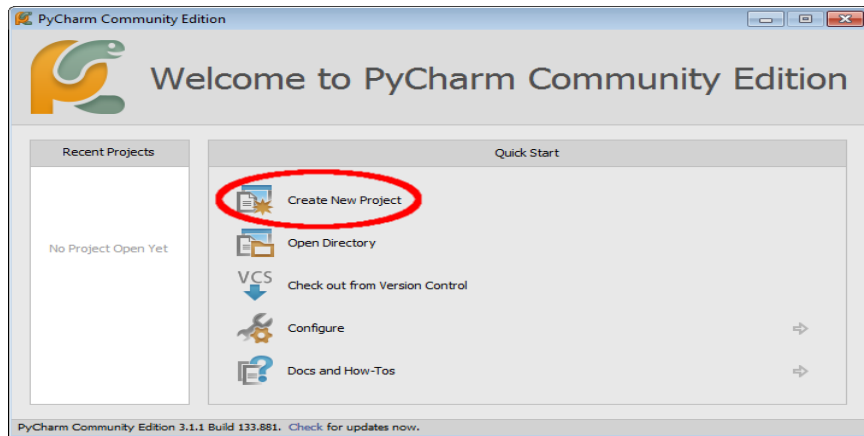
UFR de Sciences Economiques Gestion Mathématiques Informatique

2015-2016

- 1 Environnement PyCharm
- 2 Qu'est-ce qu'un algorithme ?
- 3 Comment décrire un algorithme
- 4 Introduction à Python
 - Variables
 - Opérateurs
 - Tests
 - Boucles
 - Types structurés : les séquences
- 5 Fonctions et procédures
- 6 Fichiers

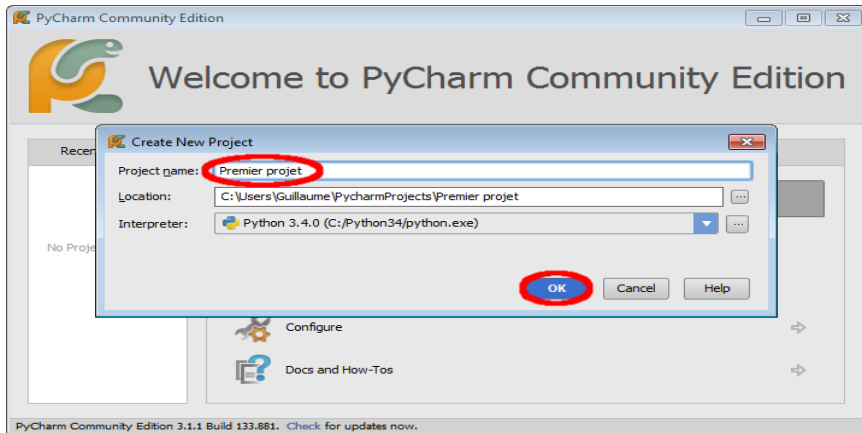
Environnement PyCharm

- Créer un projet contenant un programme Python puis exécuter ce programme afin de vérifier que tout fonctionne bien.



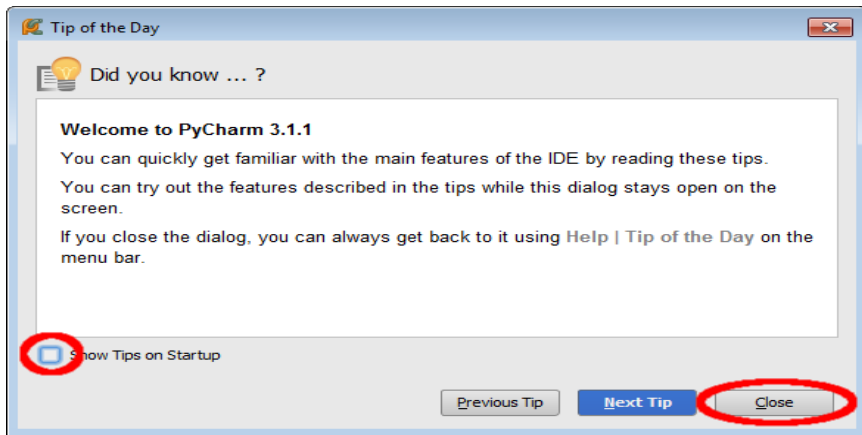
Sur l'écran d'accueil de PyCharm, cliquez sur "Create New Project".

Environnement PyCharm



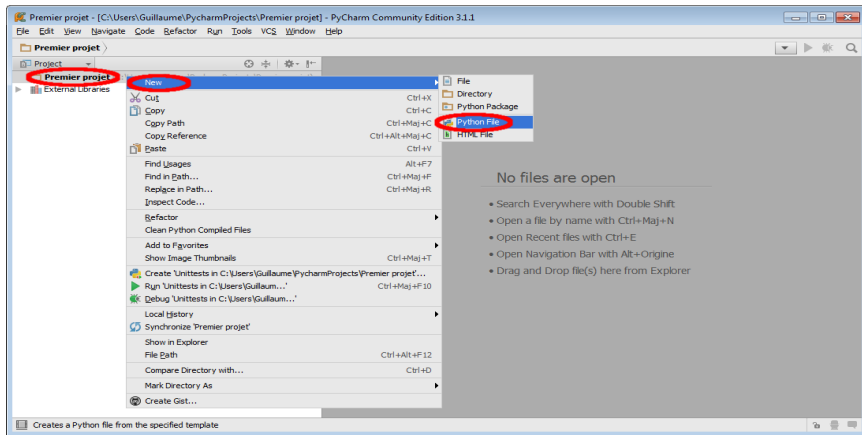
Entrez un nom pour le nouveau projet, puis cliquez sur "OK".

Environnement PyCharm



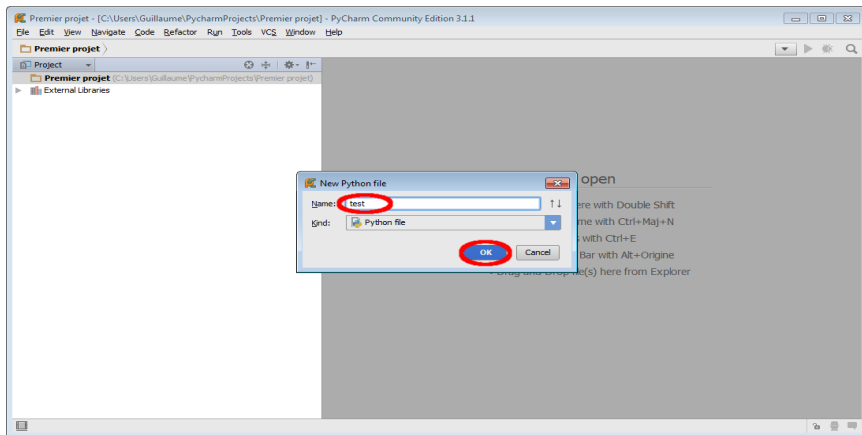
Décochez "Show Tips on Startup" (sous si vous voulez voir cette fenêtre à chaque fois), et cliquez sur "Close".

Environnement PyCharm



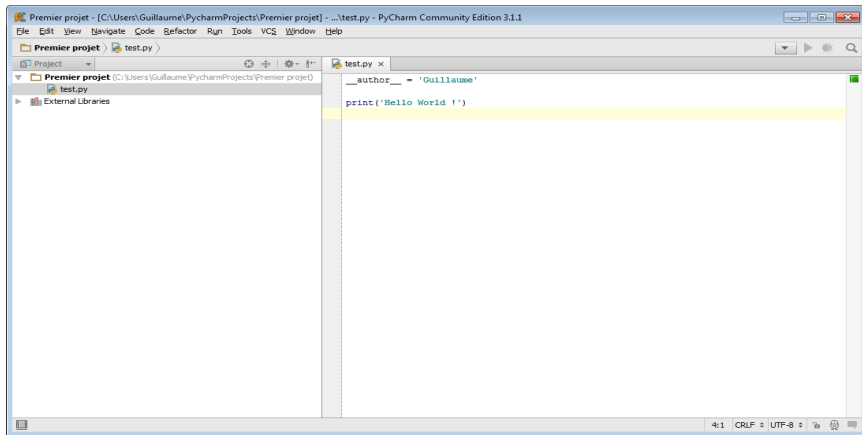
Ajouter un fichier Python à votre projet. Pour cela, faites un clic droit sur le nom du projet, puis cliquez sur "New", puis sur "Python File".

Environnement PyCharm



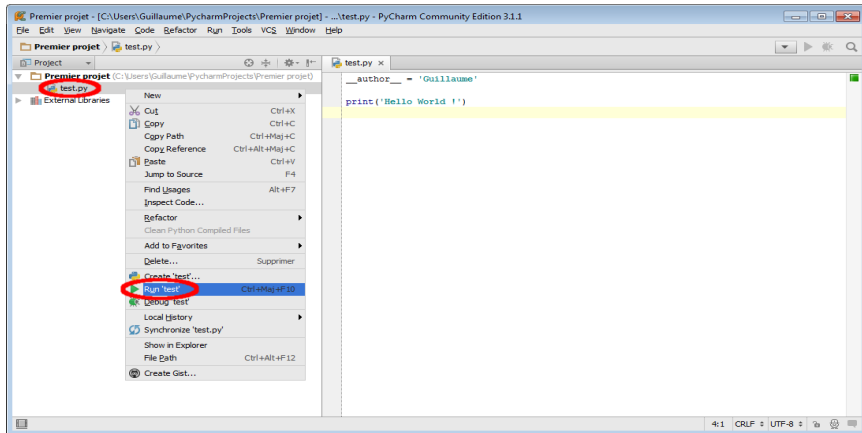
Entrez le nom du fichier.

Environnement PyCharm



Editez le fichier en ajoutant par exemple une commande **print** comme sur la capture.

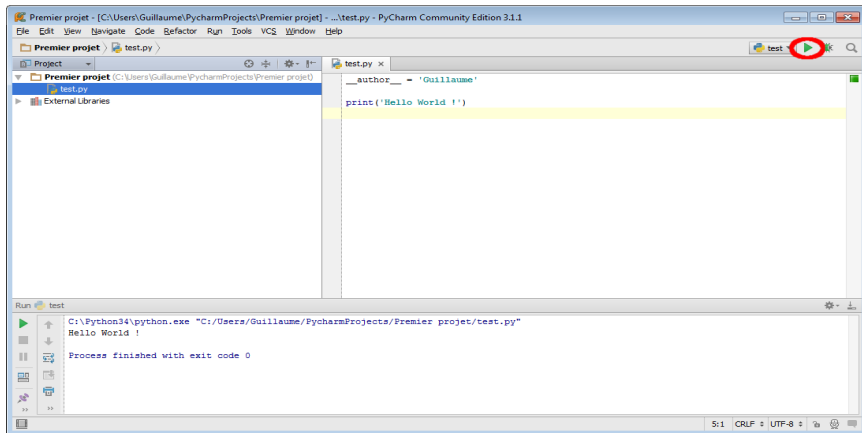
Environnement PyCharm



Clic droit sur le nom du projet, puis cliquez sur "Run 'test'".

Environnement PyCharm

- Exécution du programme Python.



En bas est apparu un nouveau cadre contenant le résultat de l'exécution du programme.

Qu'est-ce qu'un algorithme ?

Qu'est-ce qu'un algorithme ?

- ▶ Un algorithme définit un schéma de calcul ou de résolution de problème
- ▶ Il définit quel comportement suivre selon la situation rencontrée

Algorithme : définition

Série d'instructions qui doit être exécutée par un programme.

Définition de Wikipedia :

- ▶ Processus systématique de résolution d'un problème permettant de décrire les étapes vers le résultat ;
- ▶ Suite finie et non-ambiguë d'instructions permettant de donner la réponse à un problème.

Qu'est-ce qu'un algorithme ?

- ▶ Un algorithme est comme une recette de cuisine
 - ▶ Entrées (150g de sucre)
 - ▶ Instructions
 - ▶ Sortie (Caramel)

Caramel.

végétal à la sauce soja.

INGRÉDIENTS :
 POUR UN POT D'ENVIRON 20 CL.

- 150G SUCRE CASSONNADE
- 10 CL LAIT DE SOJA TIÉDI
- 5 CL EAU
- 1 C. À SOUPE SAUCE SOJA (15 ML)

1 DANS UNE CASSEOLE SUR FEU MOYEN, PORTER À ÉBULLITION LE SUCRE ET L'EAU. LAISSER BOUILLONNER JUSQU'À CE QUE LE MÉLANGE PRENNE UNE TEINTE DORÉE.

2 DÈS QUE LE CARAMEL EST BLOND CLAIR (FONCÉ, IL DEVIENT AMER), POSER LA CASSEOLE DANS L'ÉVIER ET VERSER D'UN COUP LE LAIT TIÉDI ET LA SAUCE SOJA. GARE AUX ÉCLAUSSURES, LE MÉLANGE VA MOUSSER ET PÉTILLER.

3 REMETTRE SUR FEU DOUX ET FOUETTER VIVEMENT POUR DISSOUDRE TOUTS LES MORCEAUX DE CARAMEL.

4 VERSER LE CARAMEL À LA SAUCE SOJA DANS UN POT EN VERRE. SE CONSERVE À TEMPÉRATURE AMBIANTE PLUSIEURS MOIS.

ATTENTION, NE SURTOUT PAS TOUCHER AU MÉLANGE, MAIS REMUER LA CASSEOLE PAR DES MOUVEMENTS CIRCULAIRES.

Melinda.

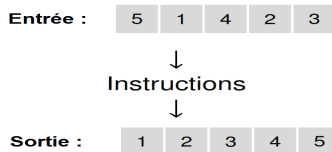
www.cuisinebandouliere.com

Qu'est-ce qu'un algorithme ?

- ▶ Un algorithme est comme une recette de cuisine
 - ▶ Entrées (150g de sucre)
 - ▶ Instructions
 - ▶ Sortie (Caramel)

▶ Exemple de l'informatique

▶ Tri



Caramel.

végétal à la sauce soja.

INGRÉDIENTS :
POUR UN POT D'ENVIRON 20 CL.

- 150G SUCRE CASSONNADE
- 10 CL LAIT DE SOJA TIÉDI
- 5 CL EAU
- 1 C. À SOUPE SAUCE SOJA (15 ML)

1) DANS UNE CASSEROLE SUR FEU MOYEN, PORTER À ÉBULLITION LE SUCRE ET L'EAU. LAISSER BOUILLONNER JUSQU'À CE QUE LE MÉLANGE PRENNE UNE TEINTE DORÉE.

2) DÈS QUE LE CARAMEL EST BLOND CLAIR (FONCÉ, IL DEVIENT AMER), POSER LA CASSEROLE DANS L'ÉVIER ET VERSER D'UN COUP LE LAIT TIÉDI ET LA SAUCE SOJA. GARE AUX ÉCLAUSSURES, LE MÉLANGE VA MOUSSER ET PÉTILLER.

3) REMETTRE SUR FEU DOUX ET FOUETTER VIVEMENT POUR DISSOUDRE TOUTS LES MORCEAUX DE CARAMEL.

4) VERSER LE CARAMEL À LA SAUCE SOJA DANS UN POT EN VERRE. SE CONSERVE À TEMPÉRATURE AMBIANTE PLUSIEURS MOIS.

ATTENTION, NE SURTOUT PAS TOUCHER AU MÉLANGE, MAIS REMUER LA CASSEROLE PAR DES MOUVEMENTS CIRCULAIRES.

Melinda.

www.cuisinebandauliere.com

Comment décrire un algorithme

Définition de Wikipedia :

- ▶ L'**algorithmique** est l'ensemble des règles et des techniques qui sont impliquées dans la définition et la conception d'algorithmes.

Algorithmique

Formalisme permettant de décrire la série d'instructions exécutée par un programme indépendamment d'un langage de programmation en particulier.

Les algorithmes sont décrits en **pseudo-code**, compréhensible par le lecteur humain mais assez précis pour transcrire les structures et les instructions de l'algorithme.

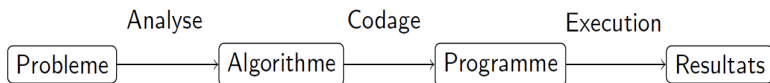
Exemple :

```
Demande l'age d'une personne
Si son age est <21
    ecrire "tu es un petit bebe"
Si son age est >30
    ecrire "qu'est ce que tu es vieux !"
Si entre les deux "ca va venir, patience... tout le monde devient vieux !"
```

Généricité des algorithmes

Les algorithmes sont écrits dans un langage de descriptions des algorithmes (pseudo-code)

- ▶ Indépendant du langage de programmation
→ Un algorithme peut être implémenté dans n'importe quel langage



Généricité des algorithmes

Une fois conçu, on traduit l'algorithme dans un langage de programmation (**Python**, Matlab, C, etc...) qui convertit les instructions en langage machine et permet ainsi à l'ordinateur d'exécuter l'algorithme.

Langage Python

Choix du langage Python :

- ▶ Langage «simple»
- ▶ Une syntaxe impliquant l'écriture d'un code lisible proche d'un langage algorithmique naturel
- ▶ Un accès aux interfaces graphiques facilité

De plus Python est un logiciel libre que chacun pourra se procurer et utiliser, et ceci sous n'importe quel environnement MS-Windows, Linux, MacOS...

Les bases de Python

1. Affichage

On affiche avec l'instruction `print`

- ▶ Le terme à afficher entre parenthèses
- ▶ Les chaînes de caractères sont données entre guillemets, sinon elles sont interprétées comme des noms de variables

```
>>> 2+3
5
>>> print (9)
9
>>> print ("Vive les L3 info-com !")
Vive les L3 info-com !
>>> 5+3
>>> 2 - 9      # les espaces sont optionnels
>>> 7 + 3 * 4   # la hiérarchie des opérations mathématiques # est-elle respectée ?
>>> (7+3)*4
>>> 20 / 3      # attention : ceci fonctionnerait différemment sous Python 2
>>> 20 // 3
>>> 20.5 / 3
>>> 8,7 / 5     # Erreur !
```

Les bases de Python

2. Commentaires en Python

- ▶ Pour commenter une ligne de code on utilise #
 - ▶ Une ligne commentée n'est pas exécutée
 - ▶ Commente une et une seule ligne : le commentaire s'arrête à la fin de la ligne
- ▶ Pour commenter plusieurs lignes, on encadre la section à commenter par ''' (triple quote) ou """(triple double quote)

```
>>> # un commentaire sur une ligne
>>> # Vive les L3 info-com !
>>> '''
un commentaire
sur plusieurs lignes
'''
```

Les bases de Python

3. L'affectation : *(nom de) variable = expression*

```
>>> a = 2+3
>>> print (a)    # affiche 5 à l'écran
>>> a, b = 7.3, 12
>>> y = 3*a + b/5
```

4. Entrée/sortie de base :

```
""" programme perroquet """
a = input (' entrez un message : ')
print (a)
```

Attention, le message est mémorisé comme une chaîne de caractères. Si vous voulez saisir un nombre pour faire des calculs, il faut une conversion :

```
>>> """ programme addition """
>>> a = input (' entrez un entier a : ')
>>> a = int(a)    # je remplace la chaîne par un entier
>>> b = int( input (' entrez un entier b : '))
>>> print (' a + b vaut : ', a+b)
```

Les bases de Python

Exemple : On écrit le programme intitulé age contenant les lignes suivantes :

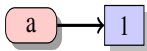
```
>>> age=eval(input("Tu as quel âge ?")) # On définit la variable age qui demande notre âge  
>>> print("Tu as donc",age,"ans.") # On retourne notre âge avec le texte adéquat
```

Variables

- ▶ Variable en mathématiques
 - ▶ Caractère non spécifié, ou inconnu
 - ▶ Exemple : $f(x) = x^2$ (quelle est la valeur de la variable x ?)
- ▶ Variable en informatique et en programmation n'a pas la même signification

Variables : noms et valeurs

- ▶ Variables (en informatique / programmation)
 - ▶ Un nom / identificateur ("a") associé à une valeur ("1")
 - ▶ Affectation : associer une variable avec une valeur ("a = 1")
 - ▶ La valeur d'une variable peut changer

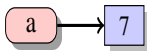


```
a = 1          # Attribuer la valeur '1' à la variable 'a'  
print(a)
```

```
C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables1.py  
1
```

Variables : noms et valeurs

- ▶ Variables (en informatique / programmation)
 - ▶ Un nom / identificateur ("a") associé à une valeur ("1")
 - ▶ Affectation : associer une variable avec une valeur ("a = 1")
 - ▶ La valeur d'une variable peut changer

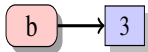
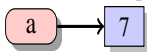


```
a = 1          # Attribuer la valeur '1' à la variable 'a'  
print(a)  
a = 7          # Attribuer une nouvelle valeur à la variable 'a'  
print(a)
```

```
C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables1.py  
1  
7
```


Variables : noms et valeurs

- ▶ Variables (en informatique / programmation)
 - ▶ Un nom / identificateur ("a") associé à une valeur ("1")
 - ▶ Affectation : associer une variable avec une valeur ("a = 1")
 - ▶ La valeur d'une variable peut changer

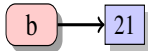
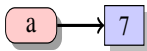


```
a = 1      # Attribuer la valeur '1' à la variable 'a'  
print(a)  
a = 7      # Attribuer une nouvelle valeur à la variable 'a'  
print(a)  
b = 3      # Attribuer la valeur '3' à la variable 'b'  
print(b)
```

```
C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables1.py  
1  
7  
3
```

Variables : noms et valeurs

- ▶ Variables (en informatique / programmation)
 - ▶ Un nom / identificateur ("a") associé à une valeur ("1")
 - ▶ Affectation : associer une variable avec une valeur ("a = 1")
 - ▶ La valeur d'une variable peut changer



```

a = 1          # Attribuer la valeur '1' à la variable 'a'
print(a)
a = 7          # Attribuer une nouvelle valeur à la variable 'a'
print(a)
b = 3          # Attribuer la valeur '3' à la variable 'b'
print(b)
b = a * b      # Attribuer la valeur 'a*b' (=7*3) à la variable 'b'
print(b)
  
```

```

C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables1.py
1
7
3
21
  
```

Type de variables

Définition

Le **Type** d'une variable est le type de donnée qui pourra être contenu dans cette variable.

Exemples :

- ▶ Entier : tout nombre entier
- ▶ Booléen : vrai ou faux
- ▶ Caractère
- ▶ Nombre réel
- ▶ Chaîne de caractères
- ▶ Tableau...

On ne peut pas mettre une donnée d'un type dans une variable d'un autre type

- ▶ Exception : un transtypage est parfois possible (entier dans réel...)

En Python : pas besoin de déclaration de type (pas besoin de dire "*n* est un entier compris entre ...")

- ▶ le typage se fait automatiquement (typage dynamique)

Type de variables

Exemple :

```
Nombre_entier = 1  
Nombre_reel = 1.540  
Booleen = True  
Caractere = "Bonjour"  
print(Nombre_entier)  
print(Nombre_reel)  
print(Booleen)  
print(Caractere)
```

```
C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables1.py  
1  
1.54  
True  
Bonjour
```

Type de variables

Exemple : • Déterminer le type de variable avec "`type ()`"

```
Nombre_entier = 1  
Nombre_reel = 1.540  
Booleen = True  
Caractere = "Bonjour"  
print(type(Nombre_entier))  
print(type(Nombre_reel))  
print(type(Booleen))  
print(type(Caractere))
```

```
C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables1.py  
<class 'int'>  
<class 'float'>  
<class 'bool'>  
<class 'str'>
```

Opérateurs

- ▶ Opérateurs d'affectation : =

```
a = 1          # Opérateurs d'affectation
print(a)
a = 3          # Opérateurs d'affectation
print(a)
```

```
C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables1.py
1
3
```

Opérateurs

- ▶ Opérateurs d'affectation : =
- ▶ Opérateurs arithmétiques : + - * \ ** %

```
a = 4 + 3          # Addition
print(a)
print( 4 - 3 )    # Soustraction
print( 4 * 3 )    # Multiplication
print( 4 / 3 )    # Division
print( 4 ** 3 )   # Puissance
```

```
C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables1.py
7
1
12
1.3333333333333333
64
```

Opérateurs

- ▶ Opérateurs d'affectation : =
- ▶ Opérateurs arithmétiques : + - * \ ** %

```
# Division
```

```
print( 9 / 2 )      # => 4.5 (float)  
print( 9 // 2 )    # => 4 (int)
```

```
# Reste de division (opérateur modulo)  
print( 9 % 9 ) print( 9 % 7 )
```

```
C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables1.py
```

```
4.5  
4  
0  
2
```


Opérateurs

- ▶ Opérateurs d'affectation : =
- ▶ Opérateurs arithmétiques : + - * \ ** %
- ▶ Opérateurs de comparaison : == != < > <= >=

```
print( 7 == 2 )      # Égal ?  
print( 7 != 2 )     # Inégal ?  
print( 7 < 2 )      # Inférieure ?  
print( 7 > 2 )      # Supérieure ?  
a = (7==2)          # Affecte le résultat de l'opérateur à la variable 'a'  
print(a)
```

```
C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables1.py  
False  
True  
False  
True  
False
```

Opérateurs

- ▶ Opérateurs d'affectation : =
- ▶ Opérateurs arithmétiques : + - * \ ** %
- ▶ Opérateurs de comparaison : == != < > <= >=
- ▶ Les opérateurs logiques : *and* ou *not* (& ou |)

```
print( 1>0 and 1>2 )      # Les deux sont vrais : Non
print( 1>0 or 1>2 )      # L'un d'eux est vrai : Oui !
print( not (1>0 and 1>2) )  # Pas vrai que les deux sont vrai ? Oui !
print( not not 1>0 )      # Double négation
```

```
C : Python34 python.exe C :/Users/FARAH/PycharmProjects/untitled/ variables 1.py
False
True
True
True
```

Tests

Un test est une structure conditionnelle : les instructions exécutées dépendent de la réalisation ou non d'une condition.

Définition

Un test définit une condition et un comportement à suivre si elle est réalisée. Optionnellement, il peut définir un comportement à suivre dans le cas contraire.

```
si (Expression Booléenne) alors  
instructions
```

La condition d'un test est une expression booléenne

- ▶ Valeurs possibles : VRAI ou FAUX

Elle est évaluée pour décider quel bloc d'instructions exécuter.

Tests

Deux formes possibles pour un test :

```
si (Expression Booléenne) alors  
instructions
```

```
if ( <condition> ) :  
<instructions>
```

```
if (x < 0) :  
    print("x est négatif")
```

Tests

Deux formes possibles pour un test :

```
si (Expression Booléenne) alors
instructions
```

```
if (x < 0) :
    print("x est négatif")
```

```
si (Expression Booléenne) alors
    instructions 1
sinon
    instructions 2
```

```
if ( <condition> ) :
<instructions>
```

À remarquer :

- ▶ les **:** en fin de ligne
- ▶ l'indentation : **else** doit être aligné avec **if** les deux "actions" (ici print) sont aussi alignées



```
if(x<0) :
    print('x est négatif')
else :
    print('x est positif')
```

Exercice : Lire un entier représentant l'heure et afficher "il est avant midi" ou "il est après midi" selon sa valeur.

Tests imbriqués

On peut imbriquer des tests, c'est-à-dire qu'un test peut être effectué dans le corps d'un autre test.

Syntaxe : if elif else

```
if ( <condition> ) :           # si vraie
    <instructions>
elif ( <condition> ) :       # si non, on teste autre condition
    <instructions> else :    # si aucune condition
    <instructions>
```

Exemple :

```
Python Shell
File Edit Shell Debug Options Windows Help
>>> if (n < 0):
        print("n est négatif")
    elif (n == 0): # n est égal zéro ?
        print("n est zéro")
    else:
        print("n est positif")

n est négatif
>>> |
Ln: 163 Col: 4
```

```
max.py - F:\Dropbox\mat_peda\algorithmique\Python\coursPptEvolution2012...
File Edit Format Run Options Windows Help
#-----
# Calculer le maximum de deux entiers
#-----
# var a, b, max : entiers
a = int(input("donnez un entier ? "))
b = int(input("donnez un entier ? "))
if (a > b) :
    max = a
else :
    max = b
print("Le maximum entre ", a, " et ", b, " est : ", max)

Python Shell
File Edit Shell Debug Options Windows Help
>>> ----- RESTART -----
>>>
donnez un entier ? 5
donnez un entier ? 12
Le maximum entre 5 et 12 est : 12
>>>
```

Boucles

- ▶ Que faire si vous voulez exécuter la même instruction à plusieurs reprises ?

Boucles

- ▶ Que faire si vous voulez exécuter la même instruction à plusieurs reprises ?

Condition d'arrêt

La condition d'arrêt d'une boucle est une condition (une expression booléenne) qui détermine le moment où une boucle doit arrêter d'exécuter le bloc d'instructions.

Une boucle sert à **répéter** un bloc d'instructions tant qu'une condition de continuation est satisfaite ou que la condition d'arrêt n'est pas satisfaite. Exemples d'utilisation :

- ▶ Parcours d'un tableau, calcul itératif...

Une boucle utilise un bloc d'instructions : c'est tout le bloc d'instructions correspondant qui est répété.

Il est possible que le bloc d'instructions ne soit pas exécuté du tout (si la condition d'arrêt est déjà satisfaite) ou un nombre infini de fois (souvent un bug).

Boucle for ("boucle pour")

On définit un compteur et :

- ▶ Une initialisation de ce compteur
 - ▶ $i = 0$
- ▶ Une condition d'arrêt pour sortir de la boucle
 - ▶ à 9
- ▶ Un pas qui modifie le compteur à la fin de chaque itération
 - ▶ pas de 1

On termine le bloc avec fin-pour

Exemple : algorithme d'affichage d'une suite de 0 à 10

```
pour  $i = 0$  à 9+1 pas 1 faire  
    écrire (i)
```

- Syntaxe **for** Python :

Syntaxe **for**

```
for <variable> in range(<debut>,<fin>,<pas>) :  
    <instructions >
```

Boucle for ("boucle pour")

- ▶ Boucle **pour** : assigne des valeurs dans une liste à une variable, l'un après l'autre
- ▶ La fonction **range**(n_1, n_2) génère des séquences de nombres
 - ▶ Attention, **range**(n_1, n_2) s'arrête à $n_2 - 1$
 - ▶ **range**(n) commence à 0 et s'arrête à $n - 1$

```
nb = 0
print(7 * nb)
nombre = 1
print(7 * nb)
nombre = 2
print(7 * nb)
nombre = 3
print(7 * nb)
```

```
for nombre in range(0, 4) :
    print(7 * nombre)
```

```
for nombre in range(4) :
    print(7 * nombre)
```

```
commline> python3 for loop0.py
0
7
14
21
```

```
commline> python3 for loop0.py
0
7
14
21
```

Indentation et blocs de codes

- ▶ L'alignement est très important dans python
- ▶ Bloc de code ("bloc d'instructions")
 - ▶ Une section de code est regroupé.
 - ▶ Code basé sur l'indentation des groupes en Python
- ▶ Indentation : 4 espaces

```

for nombre in [0, 1, 2] :
  # À l'intérieur du bloc de code
  print(7 * nombre)

  # À l'intérieur du bloc de code
  print("abc")
  
```

```

commline> python3 indentat ion1 . py
0
abc
7
abc
14
abc
  
```

```

for number in [0, 1, 2] :
  # À l'intérieur du bloc de code
  print(7 * nombre)

# En dehors du bloc de code
print("abc")
  
```

```

commline> python3 indentat ion1 . py
0
7
14
abc
  
```

Boucle imbriquée

- ▶ Boucle imbriquée : une boucle dans une boucle

```
for nombre1 in [4000, 1000] :  
    for nombre2 in [20, 30] :  
        print(number1 + number2)
```

```
commline> python3 for loopnested . py  
4020  
4030  
1020  
1030
```

Boucle imbriquée

- ▶ Boucle imbriquée : une boucle dans une boucle

```
for nombre1 in [4000, 1000] :  
    for nombre2 in [20, 30] :  
        for nombre3 in [6, 7] :  
            print(number1 + number2 + number3)
```

```
commline> python3 for loopnested2 . py  
4026  
4027  
4036  
4037  
1026  
1027  
1036  
1037
```

Exemple : Algorithme "factorielle"

- ▶ Implementation factorielle : $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$
 - 1 Version de base (pour $f(4)$, *i.e.* $f(4) = 1 * 2 * 3 * 4 = 24$)

```
# Algorithme : Calcul du factorielle = 4! = 1 * 2 * 3 * 4 = 24
```

```
factorial = 1           # 1  
factorial = 2 * factorial # 2 <- 2 * 1  
factorial = 3 * factorial # 6 <- 3 * 2  
factorial = 4 * factorial # 24 <- 4 * 6
```

```
print(factorial)      # Output
```

```
commline> python3 factorial2.py  
24
```

Exemple : Algorithme "factorielle"

- ▶ Implementation factorielle : $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$
 - 1 Version de base (pour $f(4)$, i.e. $f(4) = 1 * 2 * 3 * 4 = 24$)
 - 2 Avec la boucle **for** (pour $f(4)$)

```
# Algorithme : Calcul du factorielle = 4! = 1 * 2 * 3 * 4 = 24
factorial = 1                # 1
for i in [2, 3, 4] :        # 2, 3, 4
    factorial = i * factorial    # 2 <- 2 * 1, 6 <- 3 * 2, 24 <- 4 * 6

print(factorial)           # Output
```

```
commline> python3 factorial2.py
24
```

Exemple : Algorithme "factorielle"

- ▶ Implementation factorielle : $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$
 - 1 Version de base (pour $f(4)$, i.e. $f(4) = 1 * 2 * 3 * 4 = 24$)
 - 2 Avec la boucle **for** (pour $f(4)$)
 - 3 Avec la fonction **range** (pour $f(4)$)

```
# Algorithme : Calcul du factorielle = 4! = 1 * 2 * 3 * 4 = 24
factorial = 1                                # 1
for i in range (2, 4+1) :                    # 2, 3, 4
    factorial = i * factorial                 # 2 <- 2 * 1, 6 <- 3 * 2, 24 <- 4 * 6

print(factorial)                            # Output
```

```
commline> python3 factorial2.py
24
```


Exemple : Algorithme "factorielle"

- ▶ Implementation factorielle : $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$
 - 1 Version de base (pour $f(4)$, i.e. $f(4) = 1 * 2 * 3 * 4 = 24$)
 - 2 Avec la boucle **for** (pour $f(4)$)
 - 3 Avec la fonction **range** (pour $f(4)$)

```

n = 4           # Input
# Algorithme : Calcul du factorielle = n! = 1 * 2 * *(n - 2) * (n - 1) * n
factorial = 1   # 1
for i in range (2, n+1) :           # 2, 3, 4
    factorial = i * factorial        # 2 <- 2 * 1, 6 <- 3 * 2, 24 <- 4 * 6

print(factorial)                   # Output

```

```

commline> python3 factorial2.py
24

```

Exemple : Algorithme "factorielle"

- ▶ Implementation factorielle : $f(n) = n! = 1 \cdot 2 \cdot \dots \cdot (n-2) \cdot (n-1) \cdot n$
 - 1 Version de base (pour $f(4)$, i.e. $f(4) = 1 * 2 * 3 * 4 = 24$)
 - 2 Avec la boucle **for** (pour $f(4)$)
 - 3 Avec la fonction **range** (pour $f(4)$)

```

n = 10          # Input
# Algorithme : Calcul du factorielle = n! = 1 * 2 * ... * (n-2) * (n-1) * n
factorial = 1   # 1
for i in range (2, n+1) :           # 2, 3, 4
    factorial = i * factorial        # 2 <- 2 * 1, 6 <- 3 * 2, 24 <- 4 * 6

print(factorial)                   # Output

```

```

commline> python3 factorial2.py
3628800

```

Boucle While ("boucle tant que")

- ▶ **Combine une boucle avec une condition**
 - ▶ **utile lorsque vous ne connaissez pas le numéro d'itérations à l'avance**

La boucle **while** exécute un bloc d'instructions tant qu'une condition est vraie : c'est la condition de boucle.

- ▶ La condition est évaluée avant d'exécuter le bloc d'instructions
- ▶ Si elle n'est jamais réalisée, on n'entre jamais dans la boucle
- ▶ Condition suivie de deux points :
- ▶ Le bloc à exécuter est indenté d'un cran

Attention aux boucles infinies !

- ▶ Il faut que la condition de boucle finisse par être invalidée...

Exemple : algorithme de calcul des puissances de 2 inférieures à 100.

```
début
  puissance : Entier
  puissance ← 1
  tant que puissance < 100 faire
    afficher( puissance )
    puissance ← puissance * 2
  fintq
fin
```

Syntaxe **while**

```
while ( <condition> ) :
  <instructions >
  # fin "while"
```

Boucle While ("boucle tant que")

Exemple 1 : Afficher le premier multiple de 7 qui est plus grand que 10 000

```
nombre = 0
while (nombre <= 10000) :
    nombre = nombre + 7
print(nombre)
```

```
commline> python3 factorial2.py
10003
```

► Exemple de boucles imbriquées. Plusieurs tables de multiples :

```

multiplesImbriques.py - F:/Dropbox/mat_peda/algorithmique/Python/coursPptEvolut...
File Edit Format Run Options Windows Help
# Multiples
#-----
# var n, nb, i : entiers
nb = int(input("Multiples de combien de nombres ? "))
n = 2
while (n <= nb):
    i = 1
    while (i <= 10):
        print(i*n, "\t", end=" ")
        i = i + 1
    print() # saut de ligne
    n = n + 1
print("fin table")

Python Shell
File Edit Shell Debug Options Windows Help
Multiples de combien de nombres ? 4
2      4      6      8      10     12     14     16     18     20
3      6      9      12     15     18     21     24     27     30
4      8      12     16     20     24     28     32     36     40
fin table
>>> |
  
```

Ruptures de séquences

1 Interrompre une boucle : **break**

Sort immédiatement de la boucle **for** ou **while** en cours contenant l'instruction :

```
>>> for x in range(1, 11) :
...     if x == 5 :
...         break
...     print(x, end=" ")
...
1 2 3 4
>>> print("Boucle interrompue pour x =", x)
Boucle interrompue pour x = 5
```

2 Court-circuiter une boucle : **continue**

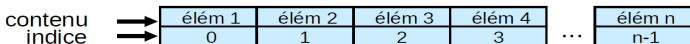
Passer immédiatement à l'itération suivante de la boucle **for** ou **while** en cours contenant l'instruction ; reprend à la ligne de l'en-tête de la boucle :

```
>>> for x in range(1, 11) :
...     if x == 5 :
...         continue
...     print(x, end=" ")
...
1 2 3 4 6 7 8 9 10
>>> # la boucle a sauté la valeur 5
```

Types structurés : les séquences

On a souvent besoin de regrouper dans une seule variable plusieurs valeurs. En python, on utilise pour cela les séquences.

Une *séquence* est un conteneur ordonné d'éléments accessibles à travers un indice entier.



Python dispose de trois types prédéfinis de séquences :

- ▶ les chaînes de caractères ;
- ▶ les listes (*on parlera de **tableaux***) ;
- ▶ les tuples (Similaires aux listes mais non modifiables. Ils ne seront pas abordés).

Séquence : chaîne de caractères

Trois notations disponibles :

- ▶ Les guillemets permettent d'inclure des apostrophes :

```
chaine1 = "L'eau vive"
```

- ▶ Les apostrophes permettent d'inclure des guillemets :

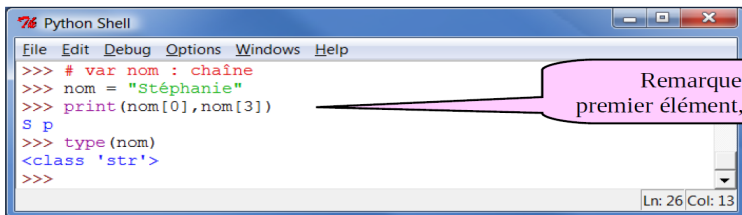
```
chaine2 = ' est "froide" !'
```

- ▶ Les triples guillemets ou triples apostrophes conservent la mise en page (lignes multiples) :

```
chaine3 = """Usage :  
-h : help  
-q : quit"""
```

Séquence : chaîne de caractères

Accès aux caractères individuels d'une chaîne. Exemple :

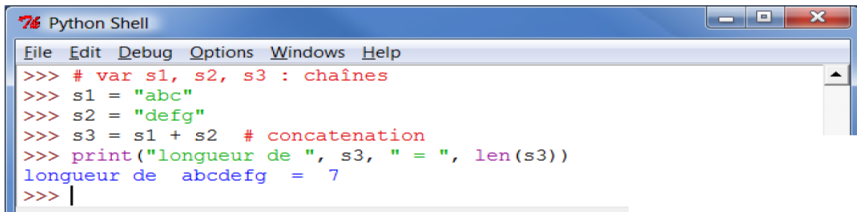


```
Python Shell
File Edit Debug Options Windows Help
>>> # var nom : chaîne
>>> nom = "Stéphanie"
>>> print(nom[0],nom[3])
S p
>>> type(nom)
<class 'str'>
>>>
```

Remarquez : premier élément, indice 0

Ln: 26 Col: 13

Python comporte de nombreuses fonctions permettant de manipuler les chaînes de caractères. Par exemple : *len*, *int*, *str*, +



```
Python Shell
File Edit Debug Options Windows Help
>>> # var s1, s2, s3 : chaînes
>>> s1 = "abc"
>>> s2 = "defg"
>>> s3 = s1 + s2 # concatenation
>>> print("longueur de ", s3, " = ", len(s3))
longueur de abcdefg = 7
>>> |
```


Séquence : chaîne de caractères

Exemple : affichage des caractères d'une chaîne

```
Python Shell
File Edit Debug Options Windows Help
>>> # var message : chaîne
>>> # var i       : entier
>>> message = "hello tous"
>>> i = 0      # initialisation
>>> while (i < len(message)):
    print(i, " - ", message[i])
    i = i + 1  # incrementation

0 - h
1 - e
2 - l
3 - l
4 - o
5 - 
6 - t
7 - o
8 - u
9 - s
```

```
Python Shell
File Edit Debug Options Windows Help
>>> # var message : chaîne
>>> # var i       : entier
>>> message = "hello tous"
>>> for i in range(0, len(message), 1):
    print(i, " - ", message[i])
```

avec "for"

Séquence : tableaux

Les tableaux sont la représentation informatique de la notion de vecteurs et de matrices mathématiques.

Tableaux

Un tableau est un type particulier de variable. Il contient un ensemble de variables de même type, stockées de façon contiguë en mémoire.

Exemple : Tableau d'entiers

3	9	12	7	29	0	4	70
---	---	----	---	----	---	---	----

Caractéristiques d'un tableau

- ▶ Un tableau a une taille fixe
- ▶ Il contient un certain type de données

Accès aux données d'un tableau

- ▶ Les cases du tableau sont numérotées de 0 à $N - 1$ (si N est la taille du tableau)
- ▶ On accède aux données d'un tableau en utilisant l'indice dans ce tableau

```
>>> tab=[1, 3, 2, 7, 5]
>>> print tab[2]
2
```

Les tableaux

1 Indices positifs et négatifs

```
>>> t = (23, 'abc', 4.56, (2,3), 'hello')
```

Indice positif : compter à partir de la gauche, en commençant par 0.

```
>>> t[1]
'abc'
```

Recherche négative : compter de droite, en commençant par -1

```
>>> t[-3]
4.56
```

Sélectionner un sous-ensemble du tableau

```
>>> t[1 :4]
('abc', 4.56, (2,3))
```

Vous pouvez également utiliser des indices négatifs lors de votre sélection.

```
>>> t[1 :-1]
('abc', 4.56, (2,3))
```

Les tableaux

2 Opérateurs sur les tableaux

- Opérateur *in* -

```
# Test boolean si une valeur est dans un tableau :
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
# Pour les chaînes, les tests pour les chaînes :
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'ac' in a
False
```

- Opérateur *+* -

L'opérateur de *+* produit un nouveau tuple, une nouvelle liste ou chaîne dont le résultat est la concaténation de ses arguments.

```
>>> (1, 2, 3) + (4, 5, 6)
(1, 2, 3, 4, 5, 6)
>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]
>>> "Hello" + " " + "World"
'Hello World'
```

Opérateur * - :** L'opérateur *** produit une "Répétition" du contenu original.

```
>>> (1, 2, 3) * 3
(1, 2, 3, 1, 2, 3, 1, 2, 3)
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> "Hello" * 3
'HelloHelloHello'
```

Les tableaux

2 Opérations sur les tableaux

```

>>>li = [1, 11, 3, 4, 5]
>>> li.append('a')           # Ajouter une element en fin de tableau
>>> li
[1, 11, 3, 4, 5, 'a']
>>> li.insert(2, 'i')       # Insérer 'i' en position 2 (li.pop(j) : Supprime l'element en position j)
>>>li
[1, 11, 'i', 3, 4, 5, 'a']
>>> li.extend([9, 8, 7])    # Ajouter une liste d'elements en fin de tableau
>>>li
[1, 11, 'i', 3, 4, 5, 'a', 9, 8, 7]
>>> li = ['a', 'b', 'c', 'b']
>>>li.index('b')           # indice de la premiere occurrence
1
>>> li.count('b')          # nombre d'occurrences
2
>>>li.remove('b')         # retirer premiere occurrence
>>> li
['a', 'b', 'c']
>>> li = [5, 2, 6, 8]
>>> li.reverse()          # inverse le tableau
>>> li
[8, 6, 2, 5]
>>> li.sort()              # Tri le tableau
>>> li
[2, 5, 6, 8]

```

Fonctions et procédures

Définition d'une fonction

- ▶ Les **fonctions** permettent de décomposer les programmes en sous-programmes et de réutiliser des morceaux de programmes.
- ▶ Une fonction est un programme Python défini à partir de paramètres d'entrées qui retourne éventuellement une valeur de sortie.
- ▶ La définition d'une fonction est introduite par le mot-clé **def**
 - ▶ Suivi du nom de la fonction
 - ▶ Entre parenthèses, ses arguments (parenthèses vides si aucun)
 - ▶ Enfin, la ligne est terminée par deux points :

Le corps de la fonction est un bloc : on l'**indente** donc d'un cran vers la droite

```
def maFonction ( a ) :  
    print " parametre : " + str ( a ) + " de type " + str ( type ( a ) )  
    return type ( a )
```

Fonctions et procédures

Définition d'une fonction

On peut sortir d'une fonction de deux façons :

- ▶ En utilisant le mot-clé **return**
 - ▶ Soit seul : fin d'une **procédure** (ne retournant rien)
 - ▶ Soit suivi d'une variable qui est retournée par la fonction
- ▶ En arrivant à la fin de la fonction : retour à l'indentation de niveau maximal (complètement à gauche), on parlera d'une **procédure**

```
def Pro(n) :  
    print(n*n) # Procédure
```

Appel d'une fonction

On appelle une fonction **par son nom**, en lui passant ses **paramètres entre parenthèses**

```
def maFonction ( a ) :  
    print " parametre : " + str ( a ) + " de type " + str ( type ( a ) )  
    return type ( a )  
  
# ailleurs dans le programme  
maFonction ( 5 )  
maFonction ( " toto " )
```

L'exécution d'un script Python commence par la première ligne en-dehors de toute fonction

- ▶ On exécute la première ligne du bloc de plus haut niveau qui ne soit pas une définition de fonction

Appel d'une fonction

On appelle une fonction **par son nom**, en lui passant ses **paramètres entre parenthèses**

```
def maFonction ( a ) :  
    print " parametre : " + str ( a ) + " de type " + str ( type ( a ) )  
    return type ( a )  
  
# ailleurs dans le programme  
maFonction ( 5 ) # premiere ligne executee  
maFonction ( " toto " )
```

L'exécution d'un script Python commence par la première ligne en-dehors de toute fonction

- ▶ On exécute la première ligne du bloc de plus haut niveau qui ne soit pas une définition de fonction

Appel d'une fonction

Exemple 1 :

```
# _____  
# Definition fonction  
# _____  
def compter_lettre(lettre, texte) :  
    n=0  
    for c in texte :  
        if c == lettre :  
            n += 1  
    return("nombre d'occurences de la lettre " + lettre + " : " + 'n')  
  
# _____  
# Corps du programme principal  
# _____  
print compter_lettre('e', 'je reviens')  
nombre d'occurences de la lettre e : 3
```

Appel d'une fonction : Exemple 2

```
>>> def pluriel(mot, famille = 'standard') :  
    if famille == 'standard' :  
        return mot + 's'  
    if famille == 's' :  
        return mot  
    if famille == 'oux' :  
        return mot + 'x'  
    if famille == 'al' :  
        return mot[:-1]+'ux'
```

Corps du programme principal

```
>>> print pluriel('maison')  
'maisons'  
>>> print pluriel('souris','s')  
'souris'  
>>> print pluriel('chou','oux')  
'choux'  
>>> print pluriel('cheval','al')  
'chevaux'
```

Variables locales et variables globales

- ▶ Les variables qui sont introduites dans la définition d'une fonction peuvent être utilisées dans la suite de la définition mais pas à l'extérieur de la fonction.
- ▶ Ces variables sont dites **locales** par opposition aux variables globales qui sont introduites à l'extérieur de la définition d'une fonction et qui peuvent être utilisées à l'intérieur comme à l'extérieur de cette définition.
- ▶ Lorsque le même nom est utilisé pour introduire une variable locale et une variable globale, Python distingue bien deux variables différentes mais à l'intérieur de la définition de la fonction, c'est à la variable locale auquel le nom réfère.

Variables locales et variables globales

```
>>> def f(x)
```

```
    y=2
```

```
    return x+y
```

```
>>> print(f(3))
```

```
5
```

```
>>> print(y)
```

```
Traceback (most recent call last) :
```

```
  File "<input>", line 1, in <module>
```

```
NameError : name 'y' is not defined
```

```
>>> u=7
```

```
>>> def g(v) :
```

```
    return u*v
```

```
>>> print(g(2))
```

```
14
```

```
>>> print(u)
```

```
7
```

Fonctions et procédures

Exercice

Un palindrome est une phrase qui peut-être lue de gauche à droite ou de droite à gauche.
Par exemple :

- ▶ le sel
- ▶ Engage le jeu que je le gagne
- ▶ Élu par cette crapule.
- ▶ Zeus a été à Suez.
- ▶ Eh ! ça va la vache ?
- ▶ Tu l'as trop écrasé, César, ce Port-Salut !

Écrivez une fonction palindrome qui détermine si la chaîne passée en argument est un palindrome sans tenir compte de la ponctuation, de la casse ou des espaces. La fonction palindrome renvoie un booléen.

Manipulation de fichiers

La manipulation de fichiers se fait toujours en trois étapes :

- ▶ Ouverture du fichier
- ▶ Manipulation du fichier
- ▶ Fermeture du fichier

Attention à ne pas oublier d'ouvrir le fichier !

Attention à ne pas oublier de fermer le fichier !

Attention à bien gérer les erreurs :

- ▶ Fichier inexistant
- ▶ Disque plein donc écriture impossible
- ▶ Pas les bons droits sur le fichier
- ▶ ... (plein de possibilités d'erreurs)

Donc : gérer les exceptions !

Ouverture de fichier

Fonction `open()`

- ▶ Deux arguments :
 - ▶ **Chemin vers le fichier** (chaîne de caractères)
 - ▶ **Mode d'ouverture** (chaîne de caractères)
- ▶ Retourne un descripteur de fichier

Ouverture du fichier `/etc/hosts` en lecture :

```
fd = open ( '/ etc / hosts ', 'r' )
```

On récupère le descripteur de fichiers `fd` :

```
>>> type ( fd )  
<type 'file '>  
>>> print fd  
<open file '/ etc/ hosts ', mode 'r' at 0 x7facf5f3f390 >
```

C'est sur ce **descripteur de fichier** que l'on va effectuer les manipulations sur le fichier.

Mode d'ouverture de fichier

Le mode d'ouverture précise :

- ▶ Si on ouvre en lecture, en écriture, ou en lecture/écriture
- ▶ Où on se place initialement dans le fichier : début ou fin
- ▶ Si le fichier est tronqué à zéro ou non

<i>Mode</i>	Utilisation
<i>r</i>	Lecture
<i>w</i>	Écriture (créé ou tronqué à 0)
<i>r+</i>	Mise à jour (lecture et écriture)
<i>w+</i>	Tronque le fichier à zéro et l'ouvre en lecture-écriture
<i>a</i>	Ajoute à la fin du fichier
<i>a+</i>	Lit au début du fichier, ajoute à la fin du fichier

- ▶ Avec **r** ou **w** : on est positionné au **début** du fichier
- ▶ Avec **a** : on est positionné à la **fin** du fichier (append)
- ▶ Avec **w** : on tronque le fichier à 0 ou on le crée
- ▶ Pour écrire sans écraser : **a** ou **a+** à la fin, **r+** au début

Fermeture d'un fichier

Fonction `close()` sur le descripteur de fichier :

```
fd. close ()
```

- ▶ On peut fermer un fichier plus d'une fois
- ▶ On ne peut pas fermer un fichier qui n'a pas été ouvert
- ▶ On doit fermer un fichier quand on en a fini avec lui

Des tentatives d'opérations sur un fichier fermé soulèveront l'exception `ValueError`