

Langage évolué & Business Intelligence

– Le langage Python : Notions de base

Plan du Chapitre

1. Historique du langage
2. Présentation générale de Python
3. Environnement de développement intégré de Python
4. Mode interactif & mode script
5. Commentaires, Aide et modules en Python
6. Types élémentaires
7. Affectation des variables et mots réservés
8. Opérations d'entrée/sortie
9. Structures conditionnelles et itératives

1) Historique du langage

Python est un langage de programmation développé depuis 1989 par le développeur néerlandais **Guido Van Rossum** et de nombreux collaborateurs.

Le nom Python revient à une série de comédie des années 70 de la BBC nommée “Monty Python’s Flying Circus”

Principales versions officielles parues :

Python 1.0 – Janvier 1994, Python 2.0 – Octobre 2000

Python 2.4 – Novembre 2004, Python 2.5 – Septembre 2006

Python 2.6 - Octobre 2008, Python 2.7 – Juillet 2010

Révolution du langage python et apparition de la v3

- Python 3.0 - Décembre, 2008
- Python 3.5 – **Septembre, 2015**



1) Historique du langage

Classement TIOBE (Septembre 2016)

Septembre 2016	Avril 2015	Avril 2014	Langage de Programmation	Ratings	Evolution
1	1	2	Java	18.236%	-1.33%
2	2	1	C	10.955%	-4.67%
3	3	4	C++	6.657%	-0,13%
4	5	5	C#	5.493%	+0.58%
5	8	8	PYTHON	4,302%	+0.64%
6	6	9	JAVASCRIPT	3.297	+0,59%
7	7	7	PHP	3.009	+0.32%
9	12	13	PERL	2,33%	+0,43%

1) Historique du langage

Classement PYPL (Septembre 2016)

Le classement PYPL (Popularity of Programming Language Index) se base sur l'analyse du nombre de fois où un tutoriel pour un langage spécifique est recherché sur Google.

Rank	Change	Language	Share	Trend
1		Java	23.6 %	-0.6 %
2	↑	Python	13.3 %	+2.4 %
3	↓	PHP	10.0 %	-0.8 %
4		C#	8.6 %	-0.3 %
5	↑↑	Javascript	7.6 %	+0.6 %
6	↓	C++	7.0 %	-0.6 %
7	↓	C	6.8 %	-0.7 %
8		Objective-C	4.5 %	-0.7 %
9	↑↑	R	3.3 %	+0.7 %
10		Swift	3.1 %	+0.4 %

2)Présentation générale de Python

C'est un langage qui inclut tous les **types de données**, les **branchements** conditionnels, les **boucles**, l'organisation du code en procédures et **fonctions**, **objets** et classes, découpage en **modules**.

Mode d'exécution : transmettre à l'interpréteur Python le fichier script **« .py »**

Python est associé à de très nombreuses librairies très performantes, notamment des librairies de calcul scientifique (Numpy, SciPy, Pandas, etc.). De fait, il est de plus en plus populaire, y compris auprès des data-scientists. Il est plus généraliste que R qui est vraiment tourné vers les statistiques.

2)Présentation générale de Python

Mode compilé vs. mode interprété

- Langage **interprété** : + portabilité application ; - lenteur (ex. R, VBA, **Python**...)
- Langage compilé : + rapidité ; - pas portable (ex. Lazarus)
- Langage pseudo-compilé : + portabilité plate-forme ; - lenteur (ex. Java)

Python est interprété, il est irrémédiablement lent, mais... on peut lui associer des bibliothèques intégrant des fonctions compilées qui, elles, sont très rapides.

2)Présentation générale de Python

Avantages(1/4)

Python est un logiciel libre « free » : utilisation sans restriction dans les projets commerciaux;

Python est un langage portable (peut fonctionner sur différentes plateformes OS (operating system));

Python convient aussi bien à des scripts d'une dizaine de lignes qu'à des projets complexes de plusieurs dizaines de milliers de lignes;

La syntaxe de Python est très simple et, combinée à des types de données évolués;

Python est un Langage de haut niveau (faire beaucoup avec peu de code, un programme python est 3 à 5 fois moins cours qu'un programme C).

2)Présentation générale de Python

Un exemple !

- C

```
#include <stdio.h>

int main(int argc, char ** argv)
{
    printf("Hello, World!\n");
}
```

- Java

```
public class Hello
{
    public static void main(String argv[])
    {
        System.out.println("Hello, World!");
    }
}
```

- now in Python

```
print "Hello, World!"
```

2)Présentation générale de Python

Avantages(2/4)

Python est un langage orienté objet qui supporte l'héritage et la surcharge des opérateurs;

Python est Dynamiquement typé:

- tout objet manipulable par le programmeur possède un type bien défini à l'exécution,
- qui n'a pas besoin d'être déclaré à l'avance.

Python est un langage inter-opérable (avec C Cython, Java Jython, C++, Fortran F2Py...)

Python gère ses ressources (mémoire, descripteurs de fichiers...) sans intervention du programmeur,

- par un mécanisme de comptage de références,
- il intègre un système de gestion de mémoire automatique (ramasse miette ou garbage collector)

2)Présentation générale de Python

Avantages(3/4)

Python intègre, comme Java ou les versions récentes de C++, un système d'exceptions,

- permettant de simplifier considérablement la gestion des erreurs,
- Lorsqu'une exception se produit, l'exécution normale du programme est interrompue et l'exception est traitée !!

Multi paradigmes, supportant les principaux styles de programmation :

- impératif, procédural, orienté objet...

Evolutif, Python est un langage qui continue à évoluer, grâce à une communauté d'utilisateurs très actifs

2)Présentation générale de Python

Avantages (4/4)

Langage polyvalent : Nous pouvons presque tout faire avec Python grâce à ses bibliothèques variées

Utilisé par de grands acteurs dans le monde: La NASA, Google, Youtube, Mozilla...

De plus en plus utilisé dans la recherche, l'enseignement et l'industrie.

2)Présentation générale de Python

Domaines d'applications

Python, est un langage de programmation de plus en plus populaire utilisé entre autres:

- WEB: Google (pure Python et Django*), Youtube, Mozilla, Yahoo, eBay (Plone**), Nokia (Plone**)
- Scientifique : la NASA (pure Python et Plone), la CIA (Plone)... la liste est bien trop longue.
- L'enseignement principalement dans plusieurs pays

*: Django est un Framework (Plateforme de développement Web) écrit en Python.

** : Plone permet à des personnes n'ayant pas de connaissance technique de créer et mettre à jour des informations sur un site web public ou sur un intranet en utilisant un simple navigateur web

3) Environnement de développement intégré de Python

Plusieurs EDI sont disponibles,

IDLE : est un environnement de développement intégré fourni avec Python (bon choix pour débutant), IDLE propose un certain nombre d'outils :

- un éditeur de texte (pour l'écriture de script) avec une coloration syntaxique, une **indentation** automatique et l'auto-complétion*.
- un interpréteur (pour exécuter le programme)
- un débogueur (pour tester le programme)

C'est l'EDI Le plus populaire, soit 23% des utilisateurs de python

*: Auto-complétion – ou complément automatique – est une fonctionnalité d'un logiciel qui propose à l'utilisateur des compléments de réponses pouvant convenir aux premières mots ou aux premières phrases qu'il a commencé à taper dans le champ de saisie.

3) Environnement de développement intégré

Il existe d'autres IDE pour Python :

- Eclipse/Pydev (17,31%),
- NetBeans,
- Spyder,
- Eric...

Pour les non débutants d'**autres IDE complet** :

- **winPython ,**
- **anaconda V3,**
- **Python(x,y)..**

4) Mode interactif & modescript

Une fois installé Python (*la dernière version 3.5.0*) peut être utilisé en deux modes, mode interactif ou en mode script:

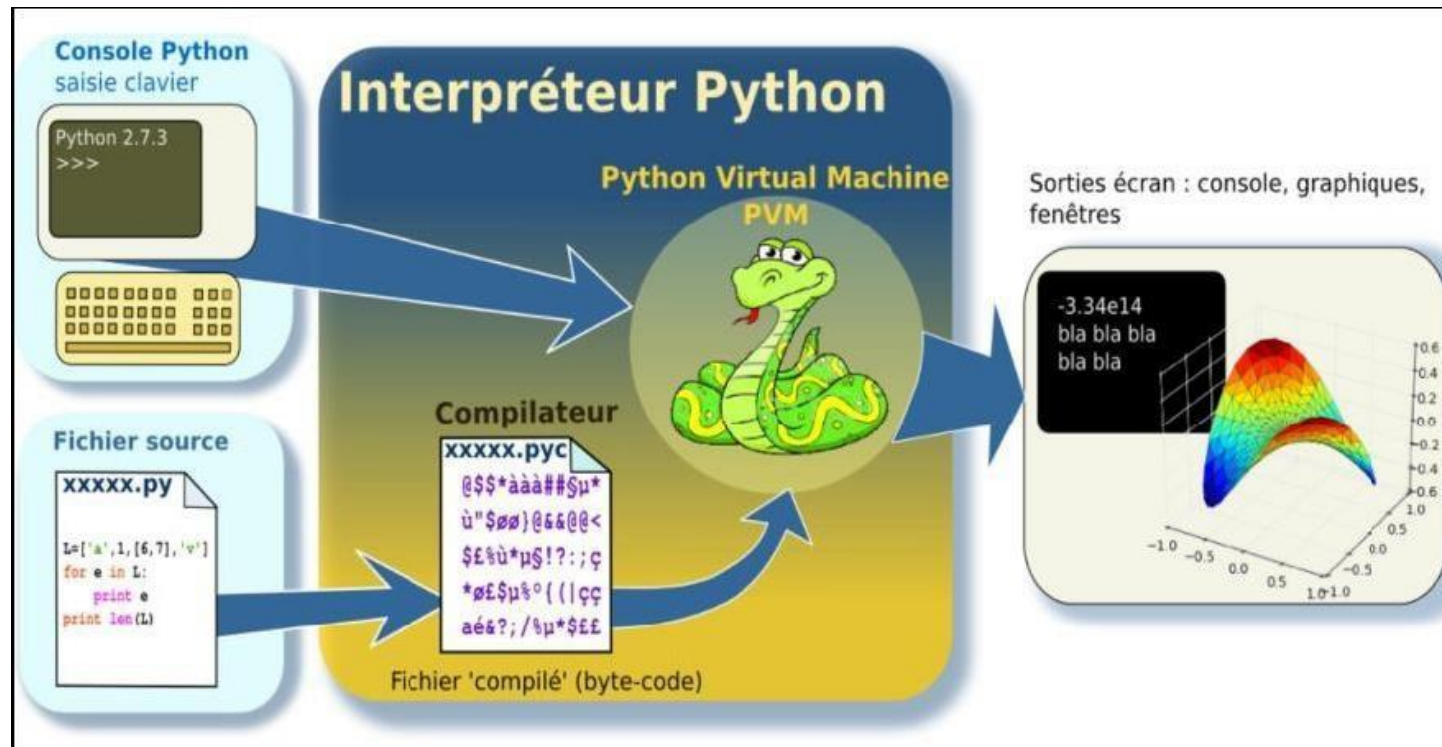
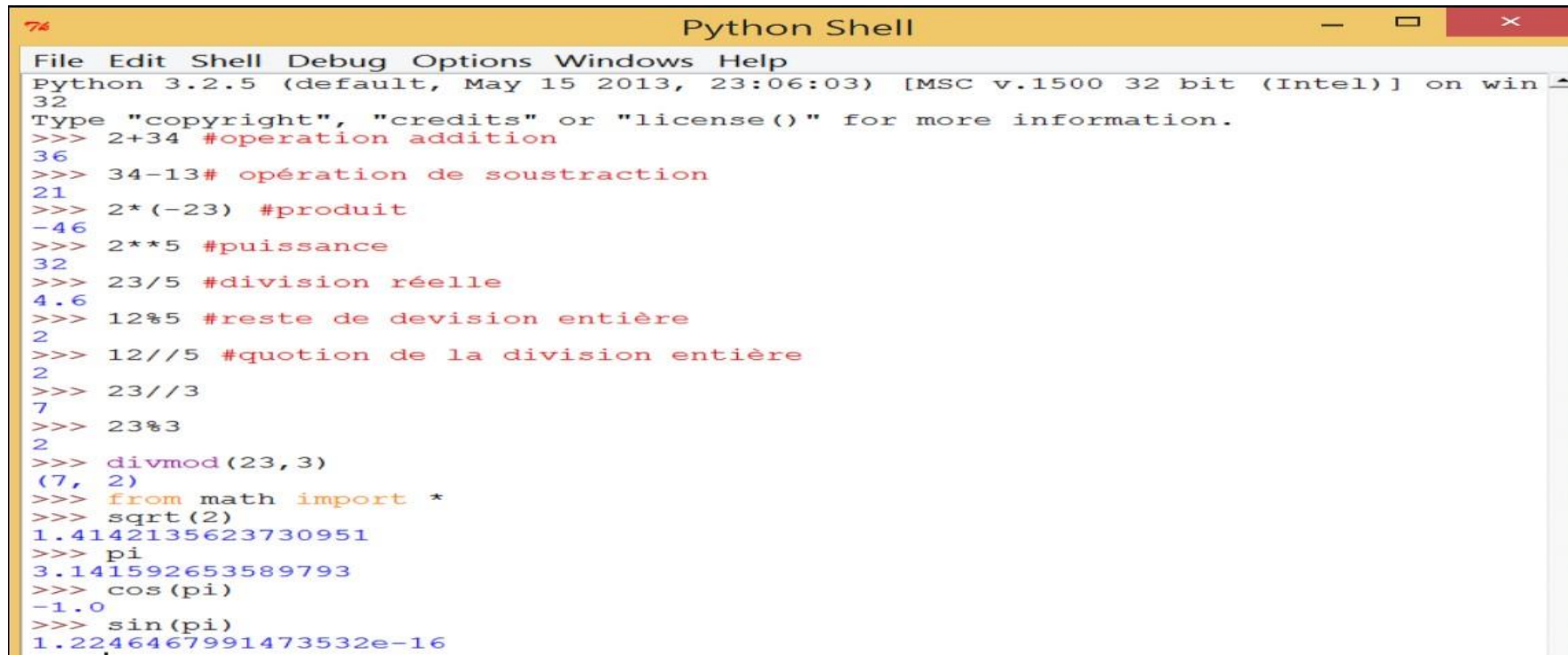


Image extraite du cours de Jean-Luc Charles, Eric Ducasse, Art et Métiers Paris Tech

4) Mode interactif & mode script

Mode interactif

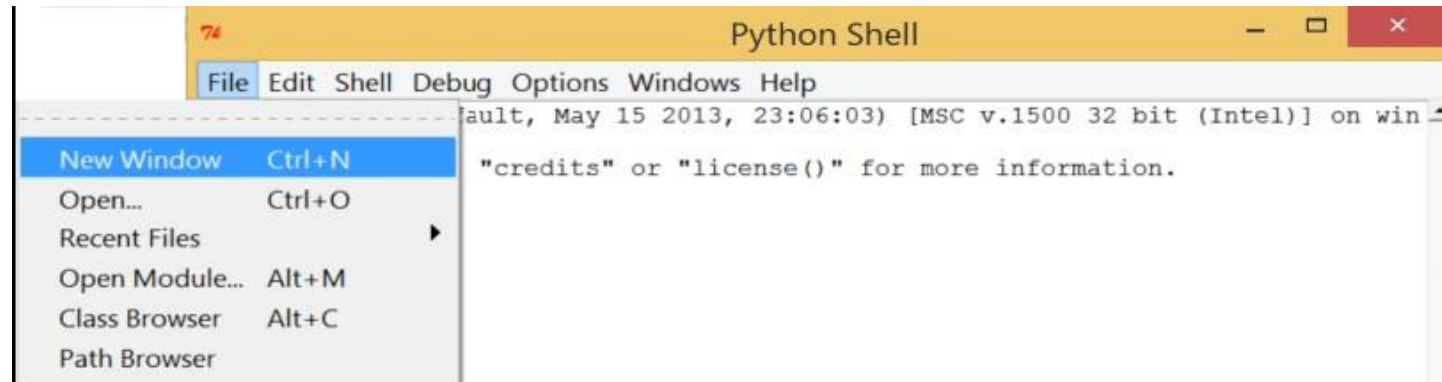
Les instructions tapées sont exécutées directement par l'interpréteur python, c'est aussi le mode calculette .

A screenshot of a 'Python Shell' window. The title bar is yellow and contains the text 'Python Shell' and standard window control buttons (minimize, maximize, close). The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Windows', and 'Help'. The main text area shows the following content:

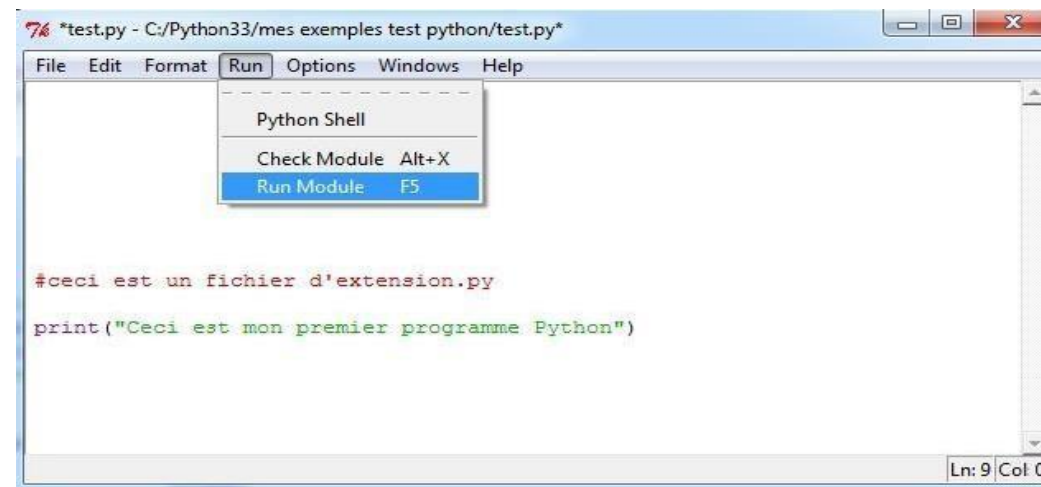
```
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 2+34 #operation addition
36
>>> 34-13# opération de soustraction
21
>>> 2*(-23) #produit
-46
>>> 2**5 #puissance
32
>>> 23/5 #division réelle
4.6
>>> 12%5 #reste de devision entière
2
>>> 12//5 #quotion de la division entière
2
>>> 23//3
7
>>> 23%3
2
>>> divmod(23,3)
(7, 2)
>>> from math import *
>>> sqrt(2)
1.4142135623730951
>>> pi
3.141592653589793
>>> cos(pi)
-1.0
>>> sin(pi)
1.2246467991473532e-16
```

4) Mode interactif & mode script

Mode script (1/2)

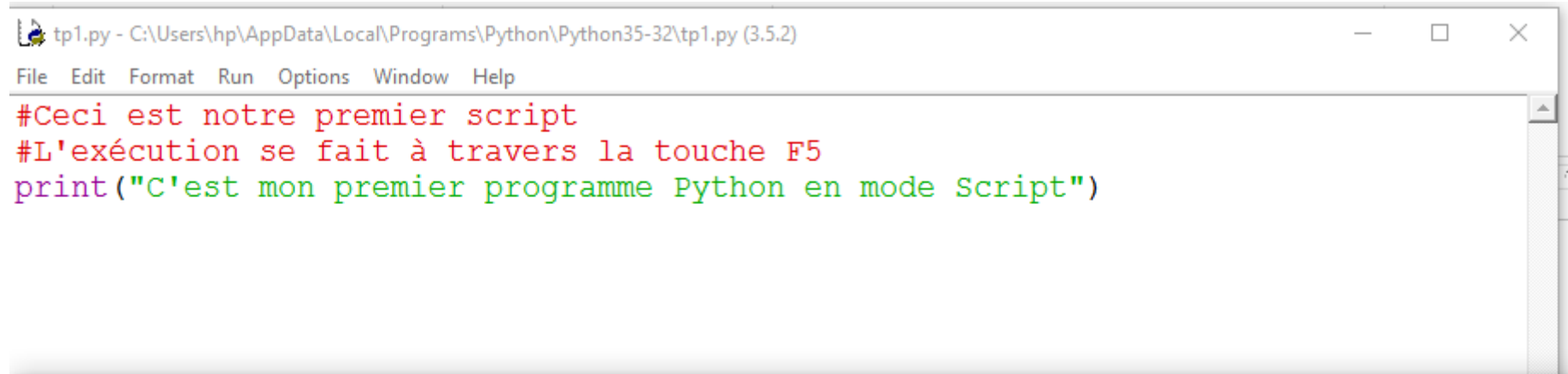


Une nouvelle fenêtre s'ouvre , vous écrivez votre code Python. Il faut enregistrer dans un fichier d'extension « .py »



4) Mode interactif & mode script

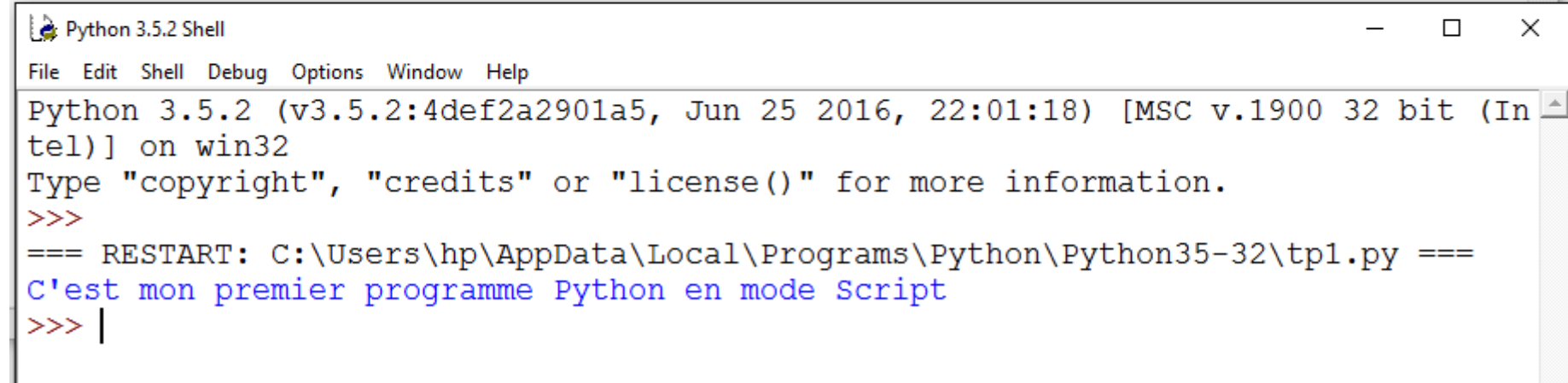
Mode script (2/2)



A screenshot of a text editor window titled "tp1.py - C:\Users\hp\AppData\Local\Programs\Python\Python35-32\tp1.py (3.5.2)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code content is as follows:

```
#Ceci est notre premier script
#L'exécution se fait à travers la touche F5
print("C'est mon premier programme Python en mode Script")
```

Exécution :



A screenshot of the "Python 3.5.2 Shell" window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The output shows the Python version and architecture, followed by a restart command and the execution of the script, which prints the message "C'est mon premier programme Python en mode Script".

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: C:\Users\hp\AppData\Local\Programs\Python\Python35-32\tp1.py ===
C'est mon premier programme Python en mode Script
>>> |
```

5) Commentaires, Aide et modules en Python

Commentaires

Tout ce qui suit le caractère # est considéré comme un commentaire et ne sera jamais évalué.

5) Commentaires, Aide et modules en Python

Utilisation de l'aide

L'utilisation de l'aide en ligne se fait par la commande ***help(identificateur)***

Exemple :

```
>>>help(int)
Help on class int in module builtins:
class int(object)
| int(x=0) -> integer
| int(x, base=10) -> integer
...
Methods defined here:
| _abs_(...)
| x._abs_() <==> abs(x)
| _add_(...)
| x._add_(y) <==> x+y
| _and_(...)
| x._and_(y) <==> x&y
| _bool_(...)
| x._bool_() <==> x != 0
```

5) Commentaires, Aide et modules en Python

Import de modules (1/4)

Trois manières d'import de modules

1^{ère} manière:

```
>>>import math
>>>dir(math)
['_doc_', '_name_', '_package_', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'copysign', 'cos',....., 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'trunc']
>>> math.ceil(7.8989) #partie entière supérieure
8
>>> math.floor(7.8989) #partie entière inférieure
7
>>>help(math.ceil)
Help on built-in function ceil in module math:
ceil(...)
ceil(x)
    Return the ceiling of x as an int.
    This is the smallest integral value >= x.
```

5) Commentaires, Aide et modules en Python

Import de modules (2/4)

2ème manière: Utilisation d'un alias

```
>>>import math as m
```

```
>>>dir(m)
```

```
['_doc_', '_name_', '_package_', 'acos', 'acosh', 'asin', 'asinh',  
'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', '.....', 'pi', 'pow',  
'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

```
>>> m.sqrt(2)
```

```
1.4142135623730951
```

```
>>> m.tan(m.pi)
```

```
-1.2246467991473532e-16
```

5) Commentaires, Aide et modules en Python

Import de modules (3/4)

3^{ème} Manière : Importation de toutes les fonctions d'un module

```
>>>from math import *
```

```
>>>dir(math)
```

```
->Erreur
```

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

```
File "<pyshell#1>", line 1, in <module>
```

```
dir(math)
```

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

Mais les fonctions du module sont directement accessibles

```
>>>sqrt(2)
```

```
1.4142135623730951
```

```
>>> help(abs)
```

```
Help on built-in function abs in module builtins:
```

```
abs(...)
```

```
abs(number) -> number
```

```
Return the absolute value of the argument.
```


5) Commentaires, Aide et modules en Python

Import de modules (4/4)

La troisième manière représente l'avantage d'accéder directement aux fonctions mais représente également l'inconvénient:

- d'encombrement de l'espace de noms réservé et,
- la possibilité de conflit entre deux fonctions ayant le même identificateur provenant de deux modules différents !!!

Exemple

```
>>>from math import *
```

```
>>>from numpy import *
```

- Les deux contiennent la fonction ***sqrt*** , l'une définie pour les réels l'autre sur les tableaux!
- Laquelle sera utilisée ????

5) Commentaires, Aide et modules en Python

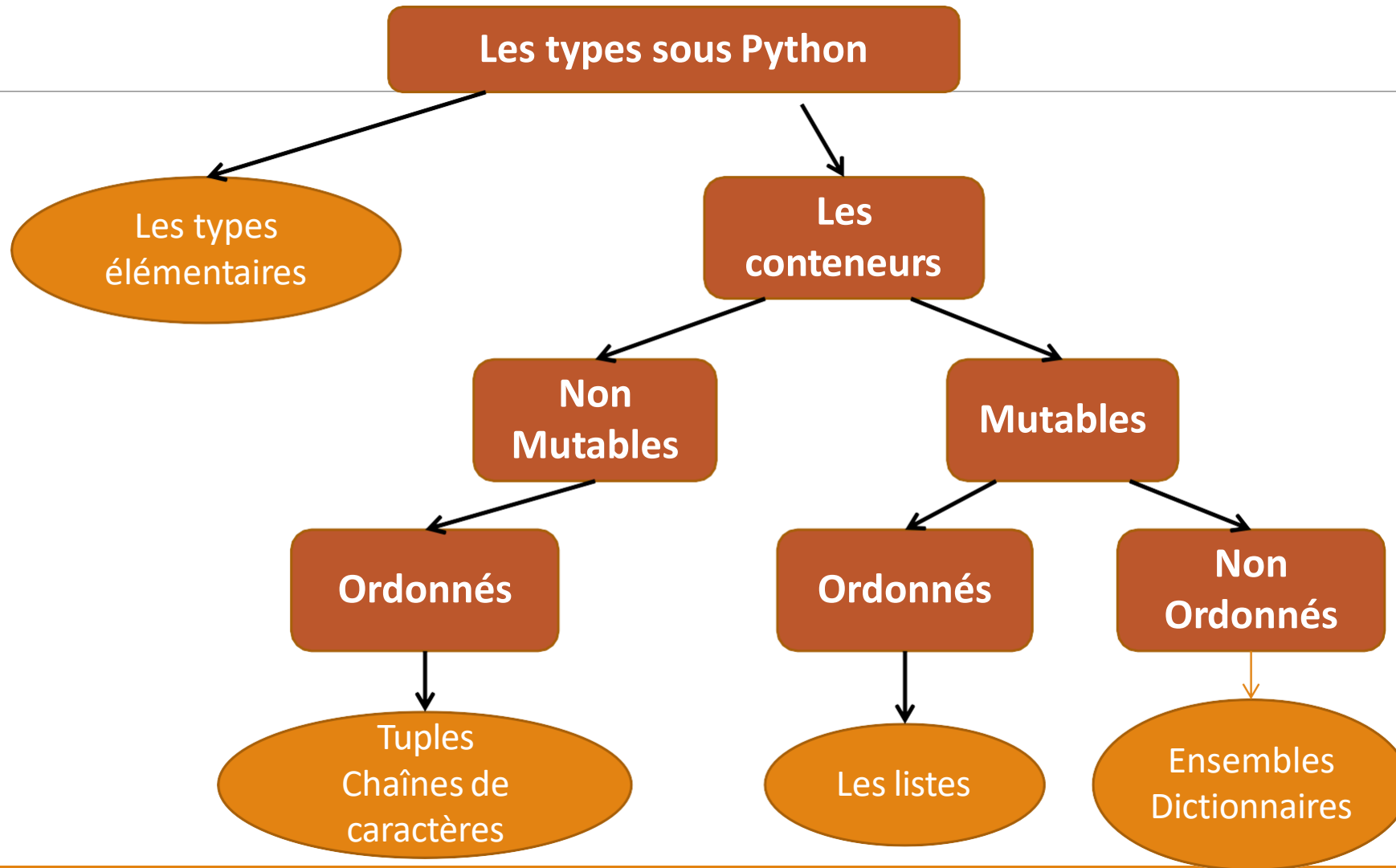
Module *math*

Parmi les fonctions les plus utiles du module `math`, on trouve :

- ❑ Racine carrée : `sqrt`
- ❑ Factorielle : `factorial`
- ❑ Fonctions trigonométriques : `cos`, `sin`, `tan`, `asin`, `acos`, `atan`
- ❑ Conversion des angles : `degrees`, `radians`
- ❑ `log`, `exp` : `log`, `log10`, `log2`
- ❑ fonctions d'arrondissement : `floor`, `ceil`
- ❑ troncature : `trunc`

NB : Le module `math` stocke aussi la valeur de π et de e .

6) Les types sous Python (1/9)



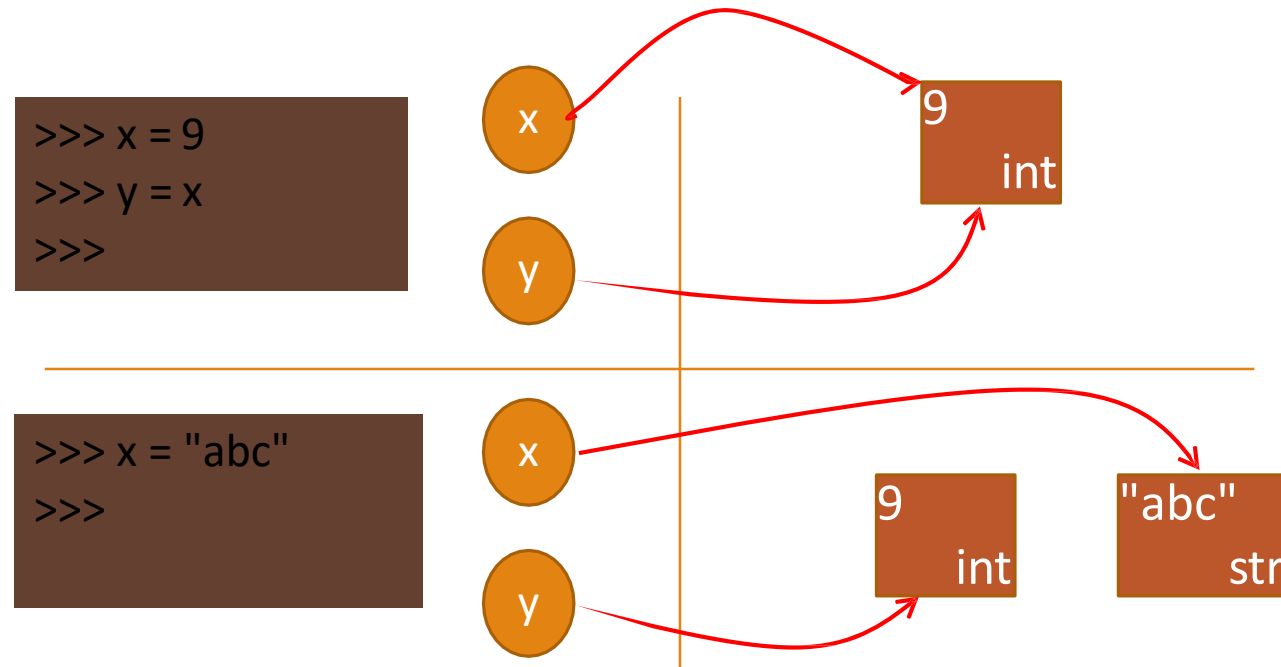
Liste des types	
<i>int</i>	Nombre entier optimisé
<i>long</i>	Nombre entier de taille arbitraire
<i>float</i>	Nombre à virgule flottante
<i>complex</i>	Nombre complexe
<i>str</i>	Chaîne de caractère
<i>unicode</i>	Chaîne de caractère unicode
<i>tuple</i>	Liste de longueur fixe
<i>list</i>	Liste de longueur variable
<i>dict</i>	dictionnaire
<i>file</i>	Fichier
<i>bool</i>	Booléen
<i>NoneType</i>	Absence de type
<i>NotImplementedType</i>	Absence d'implémentation
<i>function</i>	fonction
<i>module</i>	module

6) Les types élémentaires (3/9)

Un objet ne peut chang   ni d'identit   ni de type !

Quand un objet n'a plus de nom (nombre de r  f  rence nul), il est d  truit automatiquement:

-    m  canisme automatique de "ramasse miettes", ou *garbage collector*.



6) Les types élémentaires (4/9)

Les types élémentaire intrinsèques (built-in)

- Le ***NONETYPE*** : seule valeur possible ***None***
 - c'est la valeur retournée par une fonction qui ne retourne rien,
- Le type ***bool*** : deux valeurs possible ***True*** et ***False*** (1/0)
- Les types numériques ***int***, ***float***, ***complex*** :
 - Le type ***int x = 898***, Le type ***float x = 8.98***
 - Le type ***complex z = 8+1j*8***

Le type d'un objet détermine :

- les ***valeurs*** : domaine de définition
- les ***opérations*** possibles (+, -, /, ...)

6) Les types élémentaires (5/9)

Le type entier : <class int>

```
>>> x=3
>>> y=6
>>> type(x), type(y)
(<class 'int'>, <class 'int'>)
Les opérations arithmétiques +, -, *, **, /, //, %
>>> z=x+y          z= x.__add__(y)
9
>>> z
9
>>> z=x-y          z= x.__sub__(y)
-3
>>> x*y
18
>>> x**y # puissance
729
```

Opérateur	Déscription
x or y	ou logique
x and y	et logique
not x	négation logique
<, <=, >, >=, ==, <>, !=	opérateurs de comparaison
is, is not	Test d'identité
in, not in	Appartenance à une séquence
x y	ou bits-à-bits
x ^ y	ou exclusif bits-à-bits
x & y	et bits-à-bits
x << y, x >> y	Décalage de x par y bits
x + y, x - y	addition ou concaténation / soustraction
x * y	multiplication ou répétition
x / y, x % y	division / reste de la div. (modulo)

6) Les types élémentaires (6/9)

Le type entier : <class int>

```
>>> pow(x,y) # calcul de x à la puissance y
729
>>> x/y # division réelle
0.5
>>> x//y #quotient de la division entière
0
>>> x%y #reste de la division entière
3
```

Les opérateurs de comparaison <, <=, !=, ==, >, >=

```
>>>x==y
False
>>>x>=y
False
>>>x<=y
True
>>>x!=y
True
```


6) Les types élémentaires (7/9)

Le type réel: <class float>

```
>>> x=12/7; y=4.  
>>> x;y  
1.7142857142857142  
4.0  
>>> type(x); type(y)  
<class 'float'> <class 'float'>
```

Les opérations arithmétiques

```
>>> x+y  
5.714285714285714  
>>> x-y  
-2.2857142857142856  
>>> x*y  
6.857142857142857  
>>> x/y  
0.42857142857142855
```

6) Les types élémentaires (8/9)

Le type réel: <class float>

```
>>> x**y
8.636401499375259
>>> x/y
0.42857142857142855
>>> x//y
0.0
>>> x%y
1.7142857142857142
>>> x=12/5
>>> x
2.4
>>> int(x) #Passage de réel en entier l'objet retourné est un nouvel objet
2
```

6) Les types élémentaires (9/9)

Le type booléen : <class bool>

La classe « bool » hérite de la classe « int »

```
>> x=3 ; y=4 ; z=3
>>> B=x==y
>>> B
False
>>> E=x<y
>>> E
True
>>> B and E
False
>>> B or E
True
```

```
>>>int (True)
1
>>>int (False)
0
```

Les opérations logiques :
and, or, not...

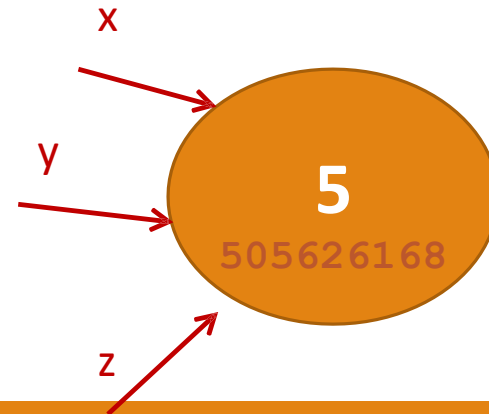
7) Affectation des variables et mots réservés (1/4)

Une affectation crée un nom (identificateur, variable) qui référence un objet (les identifiants au format `__nom__` sont réservés à l'interpréteur Python)

C'est l'objet qui porte le **type et les données (valeur, pour un objet numérique)**.

Un même objet peut être référencé sous plusieurs noms (alias).

```
>>>x=5
>>>y=5
>>>x,id(x),type(x)
(5, 505626168, <class 'int'>)
>>>y,id(y),type(y)
(5, 505626168, <class 'int'>)
```



7) Affectation des variables et mots réservés (2/4)

Nom des variables

Les noms de variables sont des noms qu'on choisit assez librement

Quelques règles pour les noms de variables sous Python :

1. Un nom de variable est une séquence de lettres (a à z , A à Z) et de chiffres, qui doit toujours commencer par une lettre.
2. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que \$, #, @, etc. sont interdits, à l'exception du caractère _ (souligné).
3. La casse est significative, **Attention** : *Mariam*, *mariaem* , *MARIEM* sont donc des variables différentes. Soyez attentifs !
4. Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre). Il s'agit d'une simple convention, mais elle est largement respectée. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans *TableDesMatières*.

7) Affectation des variables et mots réservés (3/4)

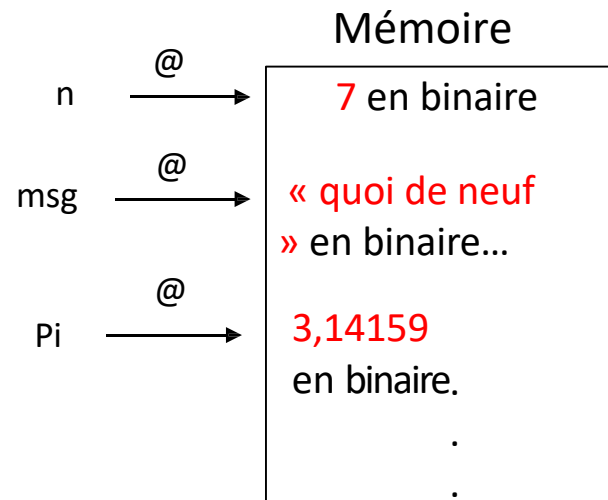
Assignation

En Python comme dans de nombreux autres langages, l'opération d'affectation est représentée par le signe *égal* :

```
>>> n = 7 # donner à n la valeur 7
```

```
>>> msg = "Quoi de neuf ?" # affecter la valeur "Quoi de neuf ?" à msg
```

```
>>> pi = 3.14159 # assigner sa valeur à la variable pi
```



7) Affectation des variables et mots réservés (4/4)

Mots réservés

En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme noms de variables les 29 « mots réservés » au langage ci-dessous :

and	assert	break	class	continue	def
del	elif	else	except	exec	finally
for	from	global	if	import	in
is	lambda	not	or	pass	print
raise	return	try	while	yield	

8) Les opérations d'entrée/sortie (1/4)

Opération d'affichage : print()

Opération d'affichage : print()

```
>>> print('ceci est un message')
ceci est un message
>>> print("ceci est un message")
ceci est un message
>>> print("ceci un message \n avec retour à la ligne")
ceci un message
avec retour à la ligne
>>> print(""" Ceci est un message
sur plusieurs lignes
avec beaucoup d'espaces et des sauts de ligne""")
Ceci est un message
sur plusieurs lignes
avec beaucoup d'espaces et des sauts de ligne
```


8) Les opérations d'entrée/sortie (2/4)

Opération d'affichage : print()

```
>>> x=10;y=10;z=10;
>>> print (x, y, z, sep=' ');
10 10 10
>>> print (x, y, z, sep=';');
10;10;10
>>> print (x, y, z , sep='\n');
10
10
10
>>> print ('x =',x, 'y =',y, 'z =', z, sep= ' ', end =';');
x = 10 y = 10 z = 10;
```

sep: désigne le caractère de séparation

end: désigne le caractère de marquage de fin

8) Les opérations d'entrée/sortie (3/4)

Opération de lecture : input()

Opération de lecture : input()

```
>>> x=input("saisir : ")
```

```
Saisir : 3498392483
```

```
>>> print("la saisie",x, "est de type",type(x))
```

```
la saisie 3498392483 est de type <class 'str'>
```

8) Les opérations d'entrée/sortie (4/4)

Opération de lecture : input()

Il est toutefois possible de convertir la quantité saisie en entier, réel ou même booléen au moyen de `int()`, `float()`, et `bool()`

```
>>> x=int(input("saisir un entier"))
```

```
saisir un entier 12
```

```
>>> print(x, "de type", type(x))
```

```
12 de type <class 'int'>
```

#Ou encore en réel

```
>>> x=float(input("saisir un réel"))
```

```
saisir un réel 23
```

```
>>>>> print(x, "de type", type(x))
```

```
23.0 de type <class 'float'>
```

9) Structures conditionnelles & Itératives

Principe d'indentation

De manière générale, un bloc contient tout le code avec une même indentation.

ceci est le bloc principal

if condition:

 bloc 2

 if condition2:

 bloc 3

 fin du bloc 2

fin du bloc 1

9) Structures conditionnelles & Itératives

Les structures conditionnelles (1/4)

Syntaxe: (Attention à l'indentation !!!)

```
if condition1:
    instruction 1
elif condition2:
    instruction 2
elif condition3:
    instruction 3
    instruction 4
else :
    instruction 5
    instruction 6
```

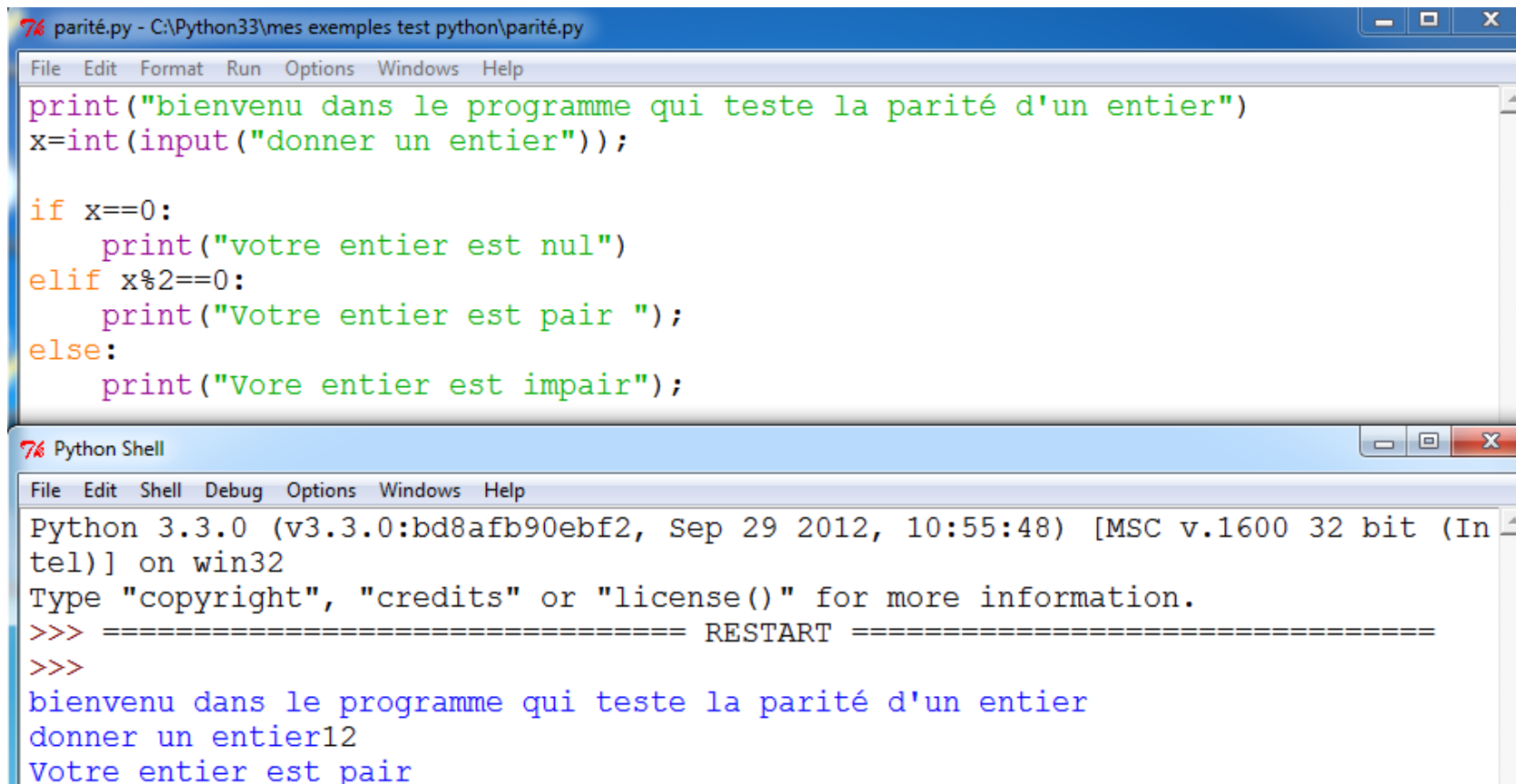


L'indentation après le ":" est obligatoire.

9) Structures conditionnelles & Itératives

Les structures conditionnelles (2/4)

Ecrire un programme qui saisit un nombre et teste si l'entier est nul, pair ou impair



```
76 parité.py - C:\Python33\mes exemples test python\parité.py
File Edit Format Run Options Windows Help
print("bienvenu dans le programme qui teste la parité d'un entier")
x=int(input("donner un entier"));

if x==0:
    print("votre entier est nul")
elif x%2==0:
    print("Votre entier est pair ");
else:
    print("Vore entier est impair");

76 Python Shell
File Edit Shell Debug Options Windows Help
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:55:48) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
bienvenu dans le programme qui teste la parité d'un entier
donner un entier12
Votre entier est pair
```

9) Structures conditionnelles & Itératives

Les structures conditionnelles (3/4)

Application: Résolution d'équation de 2nd degré

Ecrire un programme qui saisi trois entiers a, b et c et résout dans l'ensemble C l'équation de second degré **$ax^2+bx+c=0$** .

On discutera tous les cas possibles pour a, b et c !

9) Structures conditionnelles & Itératives

Les structures conditionnelles (4/4)

```
from math import *
print("Résolution equation de second degré dans C")
a=int(input("saisir le coefficient a :"))
b=int(input("saisir le coefficient b :"))
c=int(input("saisir le coefficient c :"))
if a==0:
    if b==0:
        if c==0:
            print("L'ensemble C")
        else :
            print("impossible ")
    else :
        print("Equation de premier degré de solution: ", -c/b)
else :
    delta=b**2-4*a*c
    if delta==0:
        print("solution double X1=X2=", -b/(2*a))
    elif delta >0:
        X1=(-b-sqrt(delta))/(2*a)
        X2=(-b+sqrt(delta))/(2*a)
        print("2 solutions réelles X1=", X1, "et X2=", X2)
    else :
        Z1=(-b-1j*sqrt(-delta))/(2*a)
        Z2=(-b+1j*sqrt(-delta))/(2*a)
        print("2 solutions complexes Z1=", Z1, " et Z2=", Z2)
```


9) Structures conditionnelles & Itératives

Les structures itératives : Boucle For (1/5)

Syntaxe:

```
for i in range(a) :  
    instructions
```

```
for i in range(a,b) :  
    instructions
```

```
for i in range(a,b,c) :  
    instructions
```

```
For i in iter :  
    instructions
```

range (a): désigne l'intervalle [0,a[

range (a,b): désigne l'intervalle [a,b[

range (a,b,c): désigne l'intervalle [a,b[par pas entier égal à c

Le quatrième « **iter** » cas est un parcours par élément que nous pourrons effectuer avec les itérables tels que les listes, les tuples, les chaînes de caractères ou même les fichiers...

9) Structures conditionnelles & Itératives

Boucle For (2/5)

Exemples:

```
>>> for i in range(5):  
    if i**2==4:  
        continue  
    else :  
        print(i)
```

```
0  
1  
3  
4
```

(a)

```
>>> for i in range(1,5):  
    print(i**2)
```

```
1  
4  
9  
16
```

(b)

```
>>> for i in range(1,11,2):  
    print(i)
```

```
1  
3  
5  
7  
9
```

(c)

```
>>> L=[1, 'bb', -1, "bonjour", (1,2)]  
>>> for elt in L:  
    print(elt)
```

```
1  
bb  
-1  
bonjour  
(1, 2)
```

(d)

9) Structures conditionnelles & Itératives

Boucle For (3/5)

L'instruction **for in** : permet d'itérer sur le contenu d'une liste, d'un tuple, les caractères d'une chaîne ou même un fichier ...

```
>>>L=list(range(5))
>>>L
[0,1,2,3,4]
>>>L1=[]
>>>for k in L:
    L1.append(k**2)
>>>L1
L1[0, 1, 4, 9, 16]

>>>ch="azerty"
>>>ch1=''
>>>for c in ch:
    ch1=ch1+c*2
aazzeerrttyy
```

```
>>> a = ['Zero', 'Un', 'Deux',
'Trois', 'Quatre']
```

```
>>> for i in range(len(a)):
    print(i, a[i])
```

...

```
1 Zero
2 Un
3 Deux
4 Trois
5 Quatre
```

9) Structures conditionnelles & Itératives

Boucle For (4/5)

#utilisation de l'instruction « **continue** »

```
for num in range(2, 10):  
    if num % 2 == 0:  
        print("Un nombre paire", num)  
        continue  
    print("Un nombre impaire", num)
```

#utilisation de l'instruction « **break** » et « **else** »

```
for n in range(2, 10):  
    print("n=", n)  
    for x in range(2, n):  
        print("x=", x)  
        if n % x == 0:  
            print(n, '=', x, '*', n//x)  
            break  
    else: # exécuté à la fin normale de la boucle  
        print(n, 'est un nombre premier')
```

9) Structures conditionnelles & Itératives

Boucle For (5/5)

Construction de listes par compréhension:

```
>>> L= [i for i in range(1,21,2)]
```

```
>>>L
```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

```
>>>L=[(i, j) for i in range (1,5) for j in range (1,5)]
```

```
>>>L
```

```
[(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), (3, 1), (3, 2), (3, 3), (3, 4), (4, 1), (4, 2), (4, 3), (4, 4)]
```

```
>>>L=[(i, j) for i in range (1,5) for j in range (1,5) if (i+j) % 2 ==0]
```

```
>>>L
```

```
[(1, 1), (1, 3), (2, 2), (2, 4), (3, 1), (3, 3), (4, 2), (4, 4)]
```

9) Structures conditionnelles & Itératives

Boucle tant que

Syntaxe:

```
while condition:  
    instructions1
```

```
else:  
    instructions2
```

Exemple :

```
i=1  
while i<=5:  
    print(i)  
    i +=1  
else:  
    print("Fin boucle")
```

❑ Remarque:

Il existe l'instruction « pass » pour rien faire

❑ Exemple:

```
while True:
```

```
    pass # Busy-wait pour une  
    interruption clavier (Ctrl+C) ...
```

9) Structures conditionnelles & Itératives

Utilisation du break, continue, pass

Ces trois instructions permettent à l'utilisateur d'avoir un plus grand contrôle de la boucle.

- ❑ Comme en C, l'instruction « **break** » permet de sortir de la boucle instantanément et de passer à la suite. Elle annule le passage dans la boucle else.
- ❑ L'instruction « **continue** » saute au début de la boucle la plus imbriquée.
- ❑ « **pass** » : Si à un endroit on a syntaxiquement besoin d'un bloc mais qu'il n'y a rien à faire, on peut utiliser l'instruction pass, qui justement ne fait rien

❑ Exemple

if condition:

pass

else:

instruction ...

Application: Jeu de devinette (1/2)

Ecrire un programme devinette qui permet de deviner un entier généré par le programme de façon aléatoire (compris entre 1 et 100), on indiquera à l'utilisateur si l'entier introduit est plus grand ou plus petit que l'entier caché.

Un bravo sera affiché lorsqu'il aurait deviné l'entier, on lui indiquera également au bout de combien de tentatives il est parvenu à trouver le nombre.

Remarque:

pour générer un entier aléatoire il faut importer le package **random**, puis utiliser la fonction **randint(a,b)** où a et b désigne l'intervalle [a,b] dans lequel l'entier aléatoire sera tiré.

Application: Jeu de devinette (2/2)

```
print("Jeu de devinette")
from random import *
c=randint(1,100)
x=int(input("devinez un entier entre 1 et 100: "))
i=1
while x!=c:
    if x<c:
        print("votre entier est plus petit que
l'entier caché")
    else:
        print("votre entier est plus grand que
l'entier caché")
    i+=1
    x=int(input("devinez un entier entre 1 et 100:
"))
print("Bravoooo, vous avez deviné au bout de ", i,
"tentatives")
```

```
Jeu de devinette
devinez un entier entre 1 et 100: 50
votre entier est plus petit que l'entier caché
devinez un entier entre 1 et 100: 80
votre entier est plus grand que l'enier caché
devinez un entier entre 1 et 100: 65
votre entier est plus petit que l'entier caché
devinez un entier entre 1 et 100: 72
votre entier est plus petit que l'entier caché
devinez un entier entre 1 et 100: 76
votre entier est plus grand que l'enier caché
devinez un entier entre 1 et 100: 74
Bravoooo, vous avez deviné au bout de 6 tentatives
>>>
```

Application 2 (1/2)

Modifier le programme précédant en ajoutant 3 niveaux de difficulté (difficile (4 tentatives), Moyen (7 tentatives) ou facile (12 tentatives))

Le joueur choisi un niveau de difficulté, un nombre d'essai lui est alors attribué, si ce dernier épuise ses essais il aurait échoué et un message sera affiché.

Remarque: On lui indiquera à chaque fois le nombre de tentatives restantes

Application 2 (2/2)

```
print("devinette 2")
from random import *
c=randint(1,100)
print("Jeu de devinette")
print("""choisir niveau de difficulté, taper:
    F: pour facile (12 tentatives)
    M: pour intermediaire (7 tentatives)
    D: pour difficile (4 tentatives)""")
rep=input("Niveau: F, M ou D: ")
while rep!='F' and rep!='M' and rep!='D':
    rep=input("Niveau: F, M ou D: ")
if rep=='F':
    tentatives=12
elif rep=='M':
    tentatives=7
else:
    tentatives=4
print("vous avez", tentatives, " tentatives")

x=int(input("devinez un entier entre 1 et 100: "))
i=1
while x!=c and i<tentatives:
    print("il vous reste ", tentatives-i, " tentatives")
    if x<c:
        print("votre entier est plus petit que l'entier caché")
    else:
        print("votre entier est plus grand que l'enier caché")
    i+=1
    x=int(input("devinez un entier entre 1 et 100: "))
if x==c:
    print("Bravoooo, vous avez deviné au bout de ", i, "tentatives")
else:
    print("GAME OVER, vous avez épuisé vos tentatives, c'est le ", c)
```