

# Langage Python





- **Introduction**
- **Les différents modes**
- **Programmation : notions de base**
- **Structures de contrôle**
- **Les fonctions**
- **Les classes et les objets**



- **Portable, interprété**
- **Gratuit, sous licence LGPL**
- **Syntaxe très simple**
- **Extensible, programmation par modules**
- **Orienté-objet (optionnellement)**
- **Gestion des exceptions**
- **Evolutif**



- Python est inventé par Guico Van Rossum en 1991
- Il l'a commencé depuis 1989 dans le cadre d'un projet lui servant d'occupation durant les vacances de Noël
- Le nom Python est inspiré de la série «Monty Python's Flying Circus»
- Destiné à être un langage de script sur l'OS Amoeba
- Python est influencé par ABC and Modula-3
- La première livraison 0.9.0 est sortie en février 1991, postée sur le forum Usenet alt.sources



- Project started, name picked - Dec 1989
- First public release (USENET) - Feb 1991
- python.org website - 1996 or 1997
- 2.0 released - 2000
- Python Software Foundation - 2001
- ...
- 2.4 released - 2004
- 2.5 released - 2006
- 2.6.1 - 4 Décembre 2008
- 3.0.1 - 13 Février 2009



- **Conçu pour être simple mais puissant**
- **La programmation modulaire**
- **Il met l'accent sur la lisibilité**
- **Le développement rapide d'application**
- **La facilité d'extension et d'intégration d'autres langages**



## ● Vs perl

- *Plus facile à apprendre*
- *Plus lisible*
- *Moins d'effets secondaires*
- *Moins de partialité Unix*

## ● Vs Tcl

- *Beaucoup plus rapide*
- *Moins de besoin d'extensions C*
- *Une meilleure intégration de java*



## ● Vs java

- *Code plus compact*
- *Affectation de type dynamique*
- *Exécution lente mais développement rapide*
- *Pas de compilation native de code*
- *Intégration de Java avec Jython*





## ● Vs lisp

- *Indexation des tableaux au lieu de listes chaînées*
- *Utilisation de Whitespace à la place des parenthèses 😊*
- *Compilation toujours plus lente : génération de bytecode seulement*
  - *Plus voir <http://www.norvig.com/python-lisp.html>*

## ● Vs relfun

- *Objets mutants plutôt que de relations logiques*
- *Plus : <http://www.cs.unb.ca/~boley/FLP>*



## ● Python Homepage

- <http://www.python.org/>

## ● Python Tutorial

- <http://www.python.org/tut>

## ● Python documentation

- <http://www.python.org/doc>



- **Il existe deux modes de programmations**
  - *le mode classique*
  - *le mode interactif*
- **Cela offre plus de souplesse pour le développement**
- **Permet de tester des fonctions sans lancer tout le programme**



## ● Python possède un interpréteur de commandes

```
C:\Documents and Settings\laurent>python
Python 2.3.3 (#51, Dec 18 2003, 20:22:39) [MSC v.1200 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>> 2 + 3
5
>>> (3 * 4) / 2
6
>>> for i in range (10) :
...     print i
```

- Un programme python peut entièrement être écrit en mode interactif
- les « >>> » représentent le « prompt »



- Plusieurs environnement de développement
- IDLE, Scite, SPE, Kdevelop, PyCharm, Eclipse
- La plupart sont libres et gratuits
- Ils intègrent souvent une interface pour le mode interactif

```
Shell | Locals | Session | Find | Breakpoints
1 Portions Copyright 2003, 2004 by
2 Please donate if you find this
3 Python 2.3.3 (#51, Dec 18 2003)
4 Type "help", "copyright", "credits() or "license()" for more
5 >>> 1 + 1
6 2
7 >>>
```



- Les variables
- Les types
- Les affectations
- Les opérateurs
- La composition



- Des noms arbitraires
- Alphanumériques
- Commencent toujours par une lettre
- Attention à la casse !

```
>>> jean = 2
>>> Jean = 45
>>> print jean
2
>>> print Jean
45
```



- Les variables simples sont typées dynamiquement
- La variable prend le type associé à son initialisation
- Pas de conversion automatique
- Il existe des types complexes : listes, dictionnaires et tuples





- On utilise l'opérateur " = "
- Pour les types simples, affectations par copie
- Pour les types complexes, affectations par référence



# Les affectations : exemples

```
>>> a = 'toto'
>>> print a
toto
>>> b = a                #affectation par copie
>>> print b
toto
>>> b = 'tata'
>>> print "a :", a, " - b :", b
a : toto - b : tata
>>> c = [1, 2, 3]
>>> d = c                #affectation par référence
>>> d[1] = 'a'
>>> print c
[1, 'a', 3]
```



## *Les types simples : exemples*

```
>>> a = 2
```

```
>>> b = 'des caracteres'
```

```
>>> c = 1.4
```

```
>>> print a + c
```

```
2.4
```

```
>>> print a + b
```

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

**File "<stdin>", line 1, in ?**

**TypeError: unsupported operand type(s) for +: 'float' and 'str'**



- Les listes sont des tableaux
- Les chaînes de caractères peuvent être utilisées comme des tableaux invariants
- Manipulables facilement
- A la fois des structure du langage et des objets
- Indicage (indexation) négatif = en partant de la fin



# Les listes : exemples

```
>>> jours = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']
```

```
>>> chaine = 'samedi'
```

```
>>> print jours[2]
```

```
'mercredi'
```

```
>>> print chaine[3]
```

```
e
```

```
>>> jours.remove('mercredi')
```

```
>>> print jours
```

```
['lundi', 'mardi', 'jeudi', 'vendredi']
```

```
>>> jours[2] = chaine
```

```
>>> print jours
```

```
['lundi', 'mardi', 'samedi', 'vendredi']
```

```
>>> chaine[3] = 'a'
```

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

**File "<stdin>", line 1, in ?**

**TypeError: object doesn't support item assignment**

```
>>> jours[-2]
```

```
'samedi'
```



## ● Construction directe :

```
>>> liste = ['a', 'b', 'x', 1, 'des lettres', 1.44]
```

- *On peut mélanger les types dans une même liste*

## ● Construction avec initialisation :

```
>>> liste = [0.0] * 10
```

## ● Construction avec range ()

```
>>> liste = range (1, 15, 1)
```



- **Construit une liste automatiquement**

- **Paramètres :**

- *start : debut de la liste*
- *end : fin de la liste*
- *step : pas*

```
>>> range (5)
[0, 1, 2, 3, 4]
>>> range (1, 5)
[1, 2, 3, 4]
>>> range (1, 6, 2)
[1, 3, 5]
>>> range (10, 0, -1)
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```



- Accès à des sous-parties de listes
- Intercalage d'éléments dans une liste
- syntaxe : `liste[début:fin]`
- On peut omettre *début* ou *fin*
- Indexation négative possible





# *Le slicing : exemples*

```
>>> liste = range (1, 6)
>>> print liste
[1, 2, 3, 4, 5]
>>> liste[1:3]
[2, 3]
>>> liste[:4]
[1, 2, 3, 4]
>>> liste[2:]
[3, 4, 5]
>>> liste[:-1]
[1, 2, 3, 4]
>>> liste[1:-1]
[2, 3, 4]
>>> liste[-1:]
[5]
>>> liste[-2:]
[4, 5]
```



- **liste.append()** : ajout d 'éléments en fin
- **liste.remove()** : suppression d'un élément  
(del(liste[indice]))
- **liste + liste** : concaténation
- **len(liste)** : longueur de la liste
- **liste.pop(indice)** : suppression d'un élément ayant l'indice comme position
- **liste.sort()** : tri
- **liste.reverse()** : inversion des éléments
- *élément in liste* : teste si *élément* appartient à *liste*



## *Les types complexes - les tuples*

- Fonctionnent comme les listes
- Ne sont pas modifiables
- Construction différente
  - *tuple = (1, 2, 3)*
- Les parenthèse sont optionnelles
- Le slicing fonctionne de la même manière
- Les tuples peuvent être imbriqués



# *Les tuples : exemples*

```
>>> tuple = (1, 2, 3)
>>> print tuple
(1, 2, 3)
>>> tuple[1]
2
>>> tuple = 'a', 'b'
>>> print tuple
('a', 'b')
>>> tuple = (1, 3, ('a', 'b'), 4)
>>> print tuple[2][1]
'b'
>>> print (1, 2, 3, 4)[2:]
(3, 4)
```



- Associent un nom à une valeur

- Construction :

```
>>> dico = {}  
>>> dico = {'numero' : 324, 'Nom' : 'Duval', 'Prenom' : 'Robert'}
```

- Accès aux valeurs

```
>>> dico['nom'] = 'Duval'  
>>> print dico['nom']  
'Duval'
```

- Comme les listes, ils sont modifiables



## *Les dictionnaires : les méthodes*

- **Dico.keys() : renvoie la listes des clés**
- **Dico.values() : renvoie les valeurs**
- **Dico.items() : renvoie une liste de tuples**
- **Dico.copy() : renvoie une copie du dictionnaire**
- **Dico.has\_key() : teste si la clé fait partie du dictionnaire**



# Les dictionnaires : exemples

```
>>> dico = {'numero':324, 'Nom':'Duval', 'Prenom':'Robert'}
>>> print dico.keys()
['numero', 'Nom', 'Prenom']
>>> print dico.values()
[324, 'Duval', 'Robert']
>>> print dico.items()
[('numero', 324), ('Nom', 'Duval'), ('Prenom', 'Robert')]
>>> print dico.has_key('Nom') # 'Nom' in dico
True
>>> print dico.has_key('Age')
False
>>> dico1 = dico.copy()
```



- Opérations arithmétiques :  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$
- Opérations binaires:  $<<$ ,  $>>$ ,  $|$ ,  $\&$ ,  $\wedge$ ,  $\sim$
- Opérations de comparaisons :  $\leq$ ,  $<$ ,  $>$ ,  $\geq$ ,  $==$ ,  $!=$
- Puissance :  $**$
- Opérateurs logiques : and, or, not
- Test d'appartenance : in
- On peut comparer les types simples, les listes et les tuples





- Permet d'affecter des valeurs à plusieurs variables simultanément
- S'appuie sur le fonctionnement des tuples

```
>>> valeurs = (1, 3, 'a')  
>>> a,b,c = valeurs  
>>> print "a : ",a, " - b : ", b, " - c : ", c  
a : 1 - b : 3 - c : a
```



- **Les blocs**
- **Conditions**
- **La boucle for et les séquences**
- **La boucle while**



## ● Structure :

en-tête :


```
instruction 1  
instruction 2  
...  
instruction n
```

## ● Il est possible de les imbriquer



## Les blocs : exemple

```
if i < 3 :  
    print i  
    if j == 4 :  
        print i + j  
    elif j == 2 :  
        print j
```





- Mots clés « if, elif, else »
- Opérateurs de comparaison (rappel) :  
<, <=, >, >=, ==, !=
- Structure :

```
if test :  
    ...  
[elif test :] *  
    ...  
[else :]  
    ...
```



- Les boucles permettent des instructions répétitives
- Deux instructions pour les boucles :
  - *for*
  - *while*



- Parcours d'un ensemble d'éléments prédéfini
- Syntaxe :

```
for element in sequence :  
    instruction 1  
    instruction 2  
    ...  
    instruction n
```



- Les séquences peuvent être parcourues
- Une fonctionnalité implémentée par certaines structures du langage :
  - *les listes*
  - *les tuples*
  - *les dictionnaires*
- Le développeur peut créer ses propres séquences





# *La boucle for : exemples*

```
>>> for i in [1, 2, 4] :  
...     print i  
...  
1  
2  
4  
>>>  
  
>>> liste = ('a', 'b', 'c')  
>>> for lettre in liste :  
...     print lettre  
...  
'a'  
'b'  
'c'
```



- Exécution répétitive d'instructions
- La boucle s'exécute *tant que* la condition est vraie
- Syntaxe :

```
while test :  
    instruction 1  
    instruction 2  
    ...  
    instruction n
```



## La boucle while : exemples

```
>>> i = 0
>>> while i < 10 :
...     print i
...     i = i + 1
>>>
>>> personnes = ['Laurent', 'Thomas' , 'Muriel', 'Antoine']
>>> i = 0
>>> while ((personnes[i] != 'Muriel') && (i < len(personnes))) :
...     print 'La personne est ', personnes[i]
...     i += 1
... 
```



- Ecrivez un programme qui affiche les 20 premiers termes de la table de multiplication par 9
- Ecrivez un programme qui affiche les 10 premiers termes de la suite de Fibonacci

*« Chaque terme est égal à la somme de ses deux précédents »*



# Corrections

```
>>> i = 0
>>> while i < 20 :
...     print i * 9,
...     i+= 1
...
0 9 18 27 36 45 54 63 72 81 90 99 108 117 126 135 144 153 162 171
```

```
>>> i = 1
>>> while i < 6 :
...     print i / 7.0,
...     i+= 1
...
0.142857142857 0.285714285714 0.428571428571 0.571428571429 0.714285714286
```

```
>>> u0,u1,compteur = 1,1,1
>>> while compteur < 11:
...     Print u1,
...     u0, u1, compteur = u1, u0 + u1, compteur + 1
...
1 2 3 5 8 13 21 34 55 89
```



- **Introduction**
- **Les fonctions simples (procédure)**
- **Les valeurs par défaut**
- **Utilisation des étiquettes**
- **Les « vraies » fonctions**



- **Elles évitent la redondance du code**
- **Elles favorisent la réutilisation**
- **Elles clarifient le code des programmes**
- **Le langage comporte un grand nombre de fonctions prédéfinies (l'API)**



## ● Syntaxe :

```
def nomFonction ( param 1, param 2, param n) :  
    instruction 1  
    instruction 2  
    ...  
    instruction n
```

## ● Les noms des fonctions répondent aux mêmes règles que les variables





## *Les fonctions simples : exemple*

```
>>> def afficherMessageErreur (nombreErreur,  
    nomUtilisateur) :  
...     print ' Vous avez fait ', nombreErreur, ' erreur(s)  
    de saisie ', nomUtilisateur, ' ! '  
...  
>>> afficherMessageErreur(3, ' Vincent '  
Vous avez fait 3 erreur(s) de saisie Vincent !
```



- Il est possible de déclarer des variables dans les fonctions
- Les variables peuvent porter le même nom que d'autres, définies ailleurs
- On peut aussi accéder aux autres à l'aide de la fonction « global »



# Portée des variables : exemple

```
>>> def uneFonction () :  
...     a = 3  
...  
>>> def uneDeuxiemeFonction () :  
...     global a  
...     a = 3  
...  
>>> a = 2  
>>> uneFonction()  
>>> print a  
2  
>>> uneDeuxiemeFonction()  
>>> print a  
3
```



- Elles permettent de spécifier les valeurs des paramètres
- Elles peuvent raccourcir la syntaxe d'appel des fonctions
- Syntaxe :

```
def nomFonction ( param1, param2=val2, paramn=valn) :  
    instruction 1  
    instruction 2  
    ...  
    instruction n
```



## *Les valeurs par défaut : exemple*

```
>>> def afficherVolumePiece(superficie, hauteur = 2.2) :  
...     print "Le volume est de ", superficie * hauteur, " m3"  
...  
>>>  
  
>>> afficherVolumePiece (30, 2.35)  
Le volume est de 70.3 m3  
  
>>> afficherVolumePiece (35)  
Le volume est de 77.0 m3
```



- Elles permettent de modifier l'ordre des arguments
- Elles spécifient les noms et les valeurs des arguments lors de l'appel de la fonction



## Les étiquettes : exemple

```
>>> def afficherPersonne(nom, prenom, age) :  
...     print "Nom : ", nom  
...     print "Prenom : ", prenom  
...     print "Age : ", age  
...  
>>> afficherPersonne('Durant','Jean',34)  
Nom : Durant  
Prenom : Jean  
Age : 34  
>>> afficherPersonne(age='29', prenom='Robert',nom='Martin')  
Nom : Martin  
Prenom : Robert  
Age : 29
```



## ***Les fonctions avec retour de valeurs***

- Elles permettent de récupérer le résultat de calcul
- Ce sont de « vraies » fonctions, pas de procédures
- On utilise le mot-clé « return » pour renvoyer une valeur





## Retours de valeurs : exemples

```
>>> def calculerPuissance(valeur, puissance) :  
...     resultat = valeur**puissance  
...     return resultat  
...  
>>> valeur = calculerPuissance (3, 3)  
>>> print valeur  
27  
  
>>> def genererListe (debut, fin) :  
...     return range (debut,fin)  
...  
>>> print genererListe (3, 6) :  
[3, 4, 5]
```



- Il est possible de définir une fonction avec un nombre variable d'arguments

```
>>> def moyenne(*params):  
...     print "params", params  
...  
>>> moyenne(1,4,6,8)  
params (1, 4, 6, 8)  
>>>  
>>>  
>>> def moyenne(*params):  
...     print sum(params)/len(params)  
...  
>>> moyenne(1,4,6,8)  
4
```



## ● Exemple paramètres nommés

```
>>> demarrer(ip="192.168.1.1",hostname="mw100")
kwargs {'ip': '192.168.1.1', 'hostname': 'mw100'}
>>> def demarrer(**kwargs):
...     if "ip" in kwargs:
...         print "demarrer avec ip", kwargs['ip']
...     elif "hostname" in kwargs:
...         print "demarrer avec hostname", kwargs['hostname']
...     else:
...         print "ip ou hostname manquant"
...
>>> demarrer(ip="192.168.1.1",hostname="mw100")
demarrer avec ip 192.168.1.1
>>> demarrer(hostname="mw100")
demarrer avec hostname mw100
>>> demarrer(dns="mw100")
ip ou hostname manquant
```



- On veut manipuler la structure de donnée point (2 dimensions)
  - Implémentez une fonction permettant d'initialiser les coordonnées d'un point.
  - Implémentez une fonction permettant d'afficher les coordonnées d'un point.
  - Implémentez une fonction permettant de déplacer un point.
  - Implémentez une fonction permettant de calculer la distance entre deux points

*Pour cela, placer l'instruction « `from math import sqrt` » en début de script et utiliser la fonction `sqrt()` (racine carrée)*

# Programmation Orientée Objet





- **Introduction**
- **Les classes**
- **L'héritage**
- **Le polymorphisme**
- **La surcharge**



- **Encapsulation des données**
- **Regroupement des fonctions en unité fonctionnelles**
- **Réutilisation des fonctionnalités**
- **Extension des fonctionnalités**



- **Structure de données**
- **Contient des attributs**
- **Contient des méthodes agissant sur les attributs**
- **Peuvent être créés n'importe où dans le code**





```
class MaClasse :  
    instruction
```

- Il est possible d'ajouter dynamiquement des méthodes et des attributs
- Une classe en tant que telle ne sert à rien, il faut l'instancier

```
monObjet = MaClasse()
```



# Les classes : exemple

```
>>> class MaClasse :  
    "Une classe d'exemple"  
...  
...  
>>> monObjet = MaClasse()  
>>> monObjet.x = 3  
>>> monObjet.y = 4  
>>> print monObjet.x * monObjet.y  
12  
>>> print MaClasse.x * MaClasse.y  
Traceback (most recent call last):  
  File "<stdin>", line 1, in ?  
AttributeError: class MaClasse has no attribute 'x'
```



- **Les méthodes sont des fonctions qui s 'appliquent à des objets**
- **Elles attendent toujours la variable *self* en premier paramètre**
- **Elles peuvent accéder aux attributs de la classe**



# *Les méthodes : syntaxe*

```
class MaClasse :  
    def uneMethode(self, param1, paramn) :  
        instructions  
    ...  
  
def uneAutreMethode(self, param1, paramn) :  
    instructions
```



## *Les méthodes : exemple*

```
>>> class Rectangle :  
...     def calculerSurface (self) :  
...         return self.longueur * self.largeur  
...  
>>> rect = Rectangle()  
>>> rect.longueur = 15  
>>> rect.largeur = 10  
>>> print rect.calculerSurface()  
150  
>>>
```



- Permet de déclarer et d'initialiser les attributs
- Est appelé automatiquement lors de l'instanciation
- C'est une méthode particulière : `__init__(self)`



## Le constructeur : exemple

```
>>> class Rectangle :  
...     def __init__ (self, long = 13, larg = 3) :  
...         self.longueur = long  
...         self.largeur = larg  
...         print "Construction de la classe..."  
...     def calculerSurface (self) :  
...         return self.longueur * self.largeur  
...  
>>>  
  
>>> rect = Rectangle(3,5)  
Construction de la classe...  
  
>>> print rect.calculerSurface()  
39
```



- Définir une classe Point
- Ajouter un constructeur pour initialiser l'abscisse et l'ordonnée
- Créer une méthode permettant de déplacer le point.
- Créer une méthode permettant d'afficher les coordonnées.
- Créer une méthode permettant de calculer la distance entre deux points

*Pour cela, placer l'instruction « `from math import *` » en début de script et utiliser la fonction `sqrt()` (racine carrée)*





## Les classes : correction

```
>>> from math import *
>>> class Point
...     def __init__ (self) :
...         self.x = 0
...         self.y = 0
...     def distance (self, point) :
...         distanceX = self.x - point.x
...         distanceY = self.y - point.y
...         return sqrt(distanceX ** 2 + distanceY ** 2)
... def deplacer (self, deplX, deplY) :
...     self.x += deplX
...     self.y += deplY
... 
```



- Les trois termes désignent le même concept
- Permet de créer des classes en se basant sur d'autres classes
- Spécifie le fonctionnement d'une classe au besoin
- Une classe peut hériter de plusieurs classes simultanément



# L'héritage : exemple

```
>>> class Vehicule :
...     def __init__ (self) :
...         self.nombreRoues = 4
...     self.placesAssises = 2
...     self.consomation = 8
...     def calculConso (self, distance) :
...         return float(self.consomation) * distance / 100
...
>>> class Autobus (Vehicule) :
...     def __init__ (self) :
...         self.nombreRoues = 8
...     self.placesAssises = 30
...     self.consomation = 25
...
>>> vehic = Vehicule()
>>> bus = Autobus()
>>> print vehic.calculConso (30)
2.4
>>> print bus.calculConso (30)
7.5
```



- Attention : l'ordre de l'héritage multiple est ultra important (droite -> gauche)

```
>>>
>>> class Mere1:
...     attribut = "mere1 attribut"
...
>>> class Mere2:
...     attribut = "mere2 attribut"
...
>>> class Fille(Mere1,Mere2):
...     def methode(self):
...         print self.attribut
...
>>>
>>> m1 = Mere1()
>>> m1.attribut
'mere1 attribut'
>>> m2 = Mere2()
>>> m2.attribut
'mere2 attribut'
>>> f1 = Fille()
>>> f1.methode()
mere1 attribut
>>> class Fille(Mere2,Mere1):
...     def methode(self):
...         print self.attribut
...
>>>
>>> f1 = Fille()
>>> f1.methode()
mere2 attribut
```



- **Veillez définir une classe *Forme***
  - *Attributs : point origine*
  - *Méthodes : `calculerDistance()`, `calculerPerimetre()`, `afficher()`*
- **Veillez définir les classes *Rectangle* et *Cercle* qui héritent de la classe *Forme***
- **Veillez redéfinir les méthodes `calculerDistance()` et `calculerPerimetre()` au niveau des classes filles**



- La surcharge permet d'alléger l'écriture du code
- Il suffit de définir des fonctions spécifiques dans les classes
- $+$ ,  $-$ ,  $*$ ,  $/$  : `__add__`, `__sub__`, `__mul__`, `__div__`
- $<$ ,  $>$ ,  $==$ ,  $!=$  : `__lt__`, `__gt__`, `__eq__`, `__ne__`
- Tous les opérateurs peuvent être surchargés



## Surcharge : exemple

```
>>> class Personne :  
...     def __init__ (self,nom, age) :  
...         self.nom = nom  
...         self.age = age  
...     def __lt__ (self, personne) :  
...         return self.age) < personne.age  
...  
>>> a = Personne ('Roger',20)  
>>> b = Personne ('Pierre',30)  
>>> print a < b  
  
True
```



- **Sur l'exercice précédent, utiliser la surcharge pour implémenter la fonction de calcul de distance**







- Afin de gérer la cohérence des données avec python:
  - Héritage explicite de la classe `object`
  - rendre les attributs privés (notation `__idAttribut`)
  - Utilisation de la fonction `property(getter,setter,del,'Commentaire')`

```
class MaClasse(object) :  
...     def __init__(self,attr1):  
...         self.__attr1 = attr1  
...     def setAttr1(self,attr1):  
...         print "setter"  
...         self.__attr1 = attr1  
...     def getAttr1(self):  
...         return self.__attr1  
...     attr1 = property(getAttr1,setAttr1,None,"commentaire de l'attribut attr1")  
...  
>>> o1 = MaClasse(111)  
>>> o1.attr1 = 222  
setter  
>>> print o1.attr1  
222
```



- Il est possible d'implémenter le comportement des propriétés a travers le décorateur @property

```
class Compte(object):

    def __init__(self, numero, solde = 0):
        self.__numero = numero
        self.__solde = solde

    @property
    def solde(self):
        print "get_solde"
        return self.__solde

    @solde.setter
    def solde(self, somme):
        print "set_solde"
        if somme > 0:
            self.crediter(somme)
        else:
            self.debiter(somme)
```



- Il est possible d'ajouter le comportement d'itérateur à une classe.

```
class Reverse:
    "Iterator for looping over a sequence backwards"
    def __init__(self, data):
        self.data = data
        self.index = 0
    def __iter__(self):
        self.index = len(self.data)
        return self
    def next(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]
...
>>> for char in Reverse('spam'):
...     print char
...
m
a
p
s
```

```
>>> s = 'abc'
>>> it = iter(s)
>>> it.next()
'a'
```



- Les Generators sont des outils puissants et simples de création des itérateurs
- Il sont formulé comme des expressions régulières mais utilise `yield` pour retourner les données
- à la place des méthodes `__iter__` et `next`

```
def reverse(data):  
    for index in range(len(data)-1, -1, -1):  
        yield data[index]  
  
...  
>>> for char in reverse('golf'):  
...     print char  
f  
l  
o  
g
```



## ● Examples

```
>>> sum(i*i for i in range(10)) # sum of squares
285
>>> xvec = [10, 20, 30]
>>> yvec = [7, 5, 3]
>>> sum(x*y for x,y in zip(xvec, yvec)) # zip crée des liste de tuple(ai,bi)
260

>>> from math import pi, sin
>>> sine_table = dict((x, sin(x*pi/180)) for x in range(0, 91))
>>>
>>> valedictorian = max((student.gpa, student.name) for student in graduates)

>>> data = 'golf'
>>> list(data[i] for i in range(len(data)-1,-1,-1))
['f', 'l', 'o', 'g']
```



- Veuillez définir une classe Pile avec le contrat de service suivant :
  - *empiler(objet) : ajouter un élément à la pile*
  - *depiler () : récupérer le dernier élément empilé*
  - *\_\_str\_\_()*
  - *\_\_len\_\_() : le nombre d'elements de la pile*
  - *\_\_str\_\_()*
  - *la taille est accessible en consultation*
- La pile ne peut pas dépasser une taille max initialisée lors de l'instanciation



## *Les classes : les exceptions*

- Permettent la gestion des erreurs
- Les exceptions sont des objets
- Elles peuvent être traitées individuellement, ou massivement
- Elles provoquent l'arrêt des traitements jusqu'à ce qu'elles soient interceptée
- Syntaxe :

```
try :  
    instruction #provoquant une erreur  
except [type d 'erreur] as e :  
    traitement en cas d'echec
```



## *Les exceptions : exemple*

```
>>> try :  
...     3 / 0  
...     print "La division est passee"  
...     a = UneClasseInexistante ()  
... except ZeroDivisionError :  
...     print "Une division par 0"  
... except :  
...     print "Toutes les autres erreurs"  
Une division par 0
```





## ***Les exceptions : envoi d'exceptions***

- Il est possible d'en envoyer soit même
- N'importe quelle classe peut-être une exception
- On utilise l'instruction « raise »



## *Envoi d'exception : exemple*

```
>>> class MaClasse :  
...pass  
...  
>>> def envoiException () :  
...     raise MaClass()  
...  
... try :  
...     envoiException()  
... except MaClass as e:  
...     print "Une erreur s 'est produite »,e
```



## ● Définition d'une nouvelle exception

```
class MyException(Exception):
...     def __init__(self, code, message):
...         Exception.__init__(self)
...         self.code = code
...         self.message = message
...     def __str__(self):
...         return "MyException, code :{0},
message:{1}".format(self.code, self.message)
...
>>>
>>> try:
...     print "exemple de bloc nominal"
...     print "declencher une exception"
...     raise MyException(1000, "message de l'exception")
... except Exception as e:
...     print "erreur", e
```

● Le traitement des exceptions est à la charge de l'appelant

● Le fournisseur de service déclenche les exceptions



# Exception - Exemples

Exemple I :

```
-----  
import sys  
  
try:  
    f = open('myfile.txt')  
    s = f.readline()  
    i = int(s.strip())  
except IOError as (errno, strerror):  
    print "I/O error({0}): {1}".format(errno, strerror)  
except ValueError:  
    print "Could not convert data to an integer."  
except:  
    print "Unexpected error:", sys.exc_info()[0]  
    raise SystemExit # exit()
```

Exemple II

```
-----  
for arg in sys.argv[1:]:  
    try:  
        f = open(arg, 'r')  
    except IOError:  
        print 'cannot open', arg  
    else:  
        print arg, 'has', len(f.readlines()), 'lines'  
        f.close()
```



## Exception - Méthodes avancées

- Dans la clause Except, une variable peut être ajoutée pour référencer l'exception
- Il est possible de récupérer les arguments ainsi que le message de l'exception

- `__getitem__()`

- `__str__()`

```
try:
...   raise Exception('spam', 'eggs')
... except Exception as inst:
...   print type(inst) # the exception instance
...   print inst.args # arguments stored in .args
...   print inst # __str__ allows args to be printed directly
...   x, y = inst # __getitem__ allows args to be unpacked directly
...   print 'x =', x
...   print 'y =', y
...
<type 'exceptions.Exception'>
('spam', 'eggs')
('spam', 'eggs')
x = spam
y = eggs
```

# Gestion des associations





## ● Caractéristiques d'une association

- *Une association est navigable dans les 2 sens*
- *une association a 2 extrémités qui lient les 2 classes associées*
- *Chaque extrémité porte une multiplicité et éventuellement un rôle*



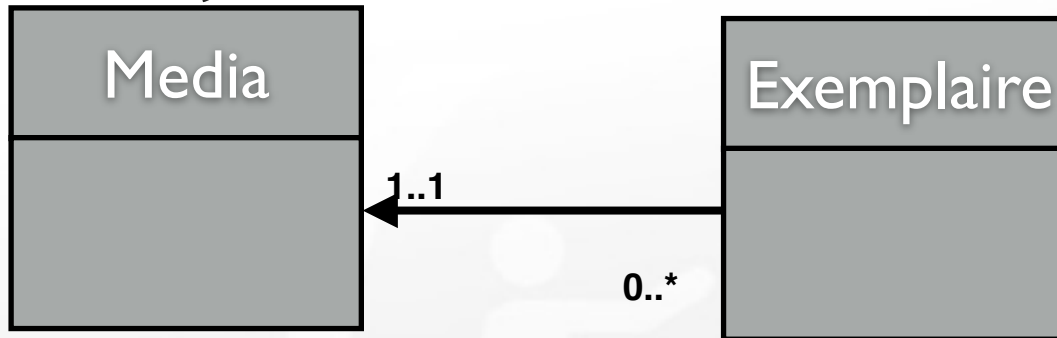


- **Implémenter une association consiste :**
  - *ajouter un attribut pour chaque extrémité*
  - *ajouter les accesseurs (getter, setter...)*
  - *ajouter les méthodes add/remove pour les extrémités ayant une multiplicité plusieurs (0..\*, 1..\*)*





## ● Cas I : Many to One



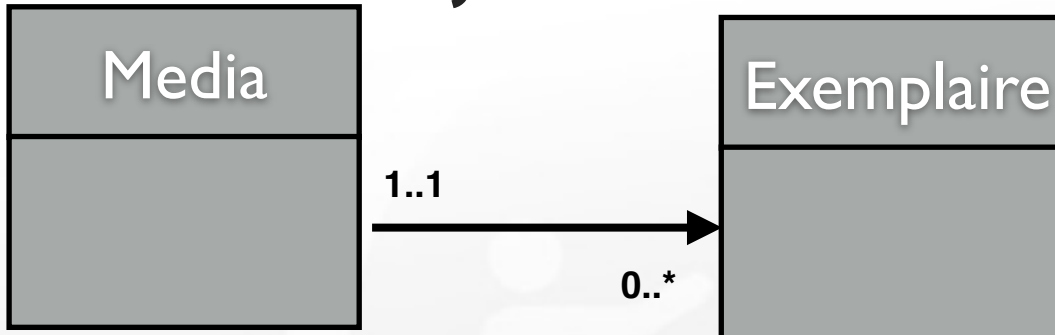
```
class Exempleire(object):
    """Exempleire d'un media"""
    def __init__(self, ident, etat , media):
        self.__ident = ident
        self.__etat  = etat
        self.__media = None
        self.set_media(media)

    def get_media(self):..

    def set_media(self, media):..
```



## ● Cas II : One to Many



```
class Media(object):
    """ Media """
    def __init__(self, ident, nom, date_pub , auteur):
        self.__ident = ident
        self.__nom = nom
        self.__date_pub = date_pub
        self.__auteur = auteur
        self.__exemplaires = []

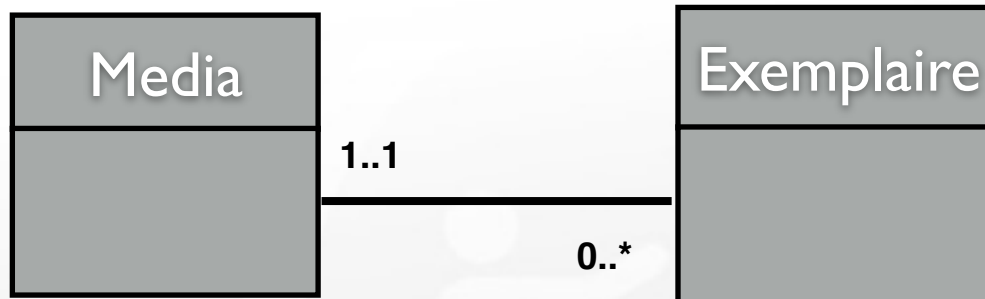
    def get_exemplaires(self):
        return self.__exemplaires

    def add_exemplaire(self, e):
        if e not in self.__exemplaires :
            self.__exemplaires.append(e)

    def remove_exemplaire(self,e):
        if e in self.__exemplaires :
            self.__exemplaires.remove(e)
```



## ● Cas III : Many to One / One to Many



```
class Media(object):
    """ Media """
    def __init__(self, ident, nom, date_pub , auteur):
        self.__ident = ident
        self.__nom = nom
        self.__date_pub = date_pub
        self.__auteur = auteur
        self.__exemplaires = []

    def get_exemplaires(self):
        return self.__exemplaires
    def add_exempleire(self, e):
        if e not in self.__exemplaires :
            self.__exemplaires.append(e)
            e.set_media(self)
    def remove_exempleire(self,e):
        if e in self.__exemplaires :
            self.__exemplaires.remove(e)
            e.set_media(None)
```



- **Ils organisent le code**
- **Ils sont représentés par les fichiers et les répertoires**
- **Ils permettent de créer des bibliothèques de fonctions**



- Un fichier est un module
- Il faut *importer* le contenu du module dans le code courant pour y accéder
- On peut importer tout ou partie d'un module
- On utilise « from » et « import » pour importer le contenu d'un module
  - *from cheminModule import \**
  - *import cheminModule.fonction*
- Pour accéder à des sous-modules, on accède aux sous-répertoires :
  - *import rep1.sousrep.rep2.nomFichier.nomFonction*



# Les modules : exemple

```
E:\imc\zope\python\moduleExemple.py

1 def uneFonction() :
2     print "Une fonction"
3
4 def uneAutreFonction () :
5     print "Une autre fonction"
6
7 class UneClasse :
8     def __init__ (self) :
9         print "Une classe"
10
```

```
>>> import moduleExemple.uneFonction
>>> uneFonction()
Une fonction
>>> from moduleExemple import *
>>> maClasse = UneClasse()
Une classe
```



- **Stockage permanent des données**
- **Interopérabilité entre applications**
- **Souplesse de fonctionnement**
  - *Edition des données de l'application*
  - *Paramétrage des applications (fichiers de configuration)*
- **Les fichiers sont traités sous forme d'objets**



- Fonction open(« chemin », « mode »)
- Le chemin comprend le nom du fichier
- Le mode peut être :
  - *"r" : Lecture seule, au début du fichier.*
  - *"r+" : Lecture/écriture, au début du fichier.*
  - *"w" : Ecriture seule, vide le fichier ou le crée*
  - *"w+" : Lecture/écriture au début du fichier, vide le fichier ou le crée*
  - *"a" : Ecriture seule, en fin de fichier, Essaye de créer le fichier*
  - *"a+" : Lecture/écriture, en fin de fichier. Essaye de créer le fichier.*





- On utilise la méthode `write`(« chaine »)
- Retour à la ligne = « `\n` »
- Exemple :

```
>>> fichier = open ('c:\monrep\monfichier.txt ', 'w')  
>>> fichier.write ('Première ligne\n')  
>>> fichier.write ('Deuxième ligne\nTroisième ligne')  
>>> fichier.close()
```

- Méthodes avancées:
  - *`fichier.tell()` : la position courante dans le fichier*
  - *`fichier.seek(offset,from_what)` : avancer la position*
    - *`from_what= 0 {début}, 1 {position courante} et 2 {Fin}`*



- On utilise les méthodes « `readline()` » et « `readlines()` »
- `fichier.closed = True` | `False`
- Un fichier est une séquence (à utiliser avec `for`)

```
try:
    file = open("exemple1_base_type.py", 'r')
    for ligne in file:
        print ligne,
        # print (ligne, end='') # python 3
except IOError as e :
    print e
finally:
    #cloture de fichier qlq soit le scenario
    if not file.closed:
        file.close()
```



## ● L'instruction with simplifier la gestion des ressources

```
try:
    with open("exemple1_base_type.py_", 'r') as file :
        for ligne in file:
            print ligne,
            # print (ligne, end='') # python 3
except IOError as e :
    print e
```



- **L 'écriture en mode texte ne permet pas de conserver les types des données**
- **Il existe des moyens de le faire (transtypage, fonctions `int()`, `float()`, etc...)**
- **On utilise un module spécialisé « pickle »**



## Lecture/écriture binaire : exemple

```
>>> import pickle
>>> f = open('Monfichier', 'w')    # wb pour python3
>>> pickle.dump(a, f)
>>> pickle.dump(b, f)
>>> pickle.dump(c, f)
>>> f.close()
>>> f = open('Monfichier', 'r')    # rb pour python3
>>> t = pickle.load(f)
>>> print t, type(t)
5 <type 'int'>
>>> t = pickle.load(f)
>>> print t, type(t)
2.83 <type 'float'>
>>> t = pickle.load(f)
>>> print t, type(t)
67 <type 'int'>
>>> f.close()
```



- **Ecrivez un programme qui filtre les lignes d'un fichier commençant par « # »**
- **Ecrivez un programme qui stocke des personnes dans un fichier (nom, prenom, adresse,...)**
- **Ecrivez un programme qui extrait les lignes différentes entre deux fichiers et les stocke dans un troisième**



## Correction exercice 1

```
def filtre(source,destination):  
    "recopier un fichier en éliminant les lignes de  
    remarques"  
    fs = open(source, 'r')  
    fd = open(destination, 'w')  
    while 1:  
        txt = fs.readline()  
        if txt == '':  
            break  
        if txt[0] != '#':  
            fd.write(txt)  
    fs.close()  
    fd.close()  
    return
```



## Correction exercice 2

```
import pickle

def nouvellePersonne (fichier, nom, prenom, age) :

    fd = open (fichier, 'a+')

    pickle.dump (nom, fd)

    pickle.dump(prenom, fd)

    pickle.dump(age, fd)

    fd.close()

def lirePersonne (fd) :

    print "Nom : ", pickle.load(fd)

    print "Prenom : ", pickle.load(fd)

    print "Age : ", pickle.load(fd)

nouvelle = 'o'

compteur = 0

while nouvelle == 'o' :

    nom1 = raw_input("Saisissez le nom : ")

    prenom1 = raw_input ("Saisissez le prenom : ")

    age1 = int(raw_input("Saisissez l'age : "))

    nouvellePersonne ('c:/personnes.bin', nom1, prenom1, age1)

    compteur += 1

    nouvelle = raw_input("Continuer ? (o/n)")

fd = open ("c:/personnes.bin")

for i in range (compteur) :

    lirePersonne(fd)
```





## ● L'interface de Système d'exploitation

```
>>> import os
>>> os.system('ls -la')
0
>>> os.getcwd() # Return the current working directory
'C:\\Python26'
>>> os.chdir('/server/accesslogs')

>>> dir(os)
<returns a list of all module functions>
>>> help(os)
<returns an extensive manual page created from the module's docstrings>

#For daily file and directory management tasks, the shutil module
provides a higher level interface that #is easier to use:
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db')
>>> shutil.move('/build/executables', 'installdir')
```



- **File Wildcards : Fournit le mécanisme de recherche basé sur le caractère \***

```
>>> import glob
>>> glob.glob('*.py')
['primes.py', 'random.py', 'quote.py']
```

- **Les arguments de commandes en ligne : Les arguments de script sont stockés dans sys.argv**

```
python demo.py one two three
>>> import sys
>>> print sys.argv
['demo.py', 'one', 'two', 'three']
```



- Le module `math` donne accès aux fonctions de la bibliothèque C

```
>>> import math
>>> math.cos(math.pi / 4.0)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

- Le module `random` définit les méthodes de création de sélection aléatoire

```
import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.sample(xrange(100), 10) # Echantillonnage sans
replacement
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random() # random float
0.17970987693706186
>>> random.randrange(6) # random integer chosen from range(6)
4
```



## ● Le module datetime fournit les méthodes de gestion des dates

```
# dates are easily constructed and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2003, 12, 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

# dates support calendar arithmetic
>>> birthday = date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```



- **Contrôle de Qualité : L'approche est basée sur l'intégration des méthodes de test**
- **Le module doctest fournit les moyens pour scanner et valider les tests des programmes**

```
def average(values):  
    """Computes the arithmetic mean of a list of numbers.  
    >>> print average([20, 30, 70])  
    40.0  
    """  
  
    return sum(values, 0.0) / len(values)  
import doctest  
doctest.testmod() # automatically validate the embedded tests
```



## ● Python fournit un module «unittest» pour simplifier les tests unitaires

```
import unittest

class PileTestCase(unittest.TestCase):
    def setUp(self):
        print "set_up : Initialisation des tests unitaires"
        self.pile = Pile()

    def tearDown(self):
        print "tearDown : liberation des ressources »
        self.pile = None

    def testEmpiler(self):
        self.pile.empiler("elem1")
        resultat_attendu = 1
        resultat_calcule = len(self.pile)
        self.assertEqual(resultat_attendu, resultat_calcule)

if __name__ == "__main__":
    #import sys;sys.argv = ['', 'ContratTestCase.testEmpiler']
    unittest.main()
```



- Gestion des traces : python offre un module de gestion de logs "logging" flexible
- Par défaut les logs sont redirigés vers un fichier ou `sys.stderr`

```
import logging
logging.debug('Debugging information')
logging.info('Informational message')
logging.warning('Warning:config file %s not found',
'server.conf')
logging.error('Error occurred')
logging.critical('Critical error -- shutting down')
```

#Le résultat est:

```
WARNING:root:Warning:config file server.conf not found
ERROR:root:Error occurred
CRITICAL:root:Critical error -- shutting down
```



## ● **La démarche de gestion des traces :**

- *Configuration de logger*
- *Configuration des gestionnaires*
- *Configuration des layouts*
- *Ecriture de messages*





## ● Les Loggers offrent:

### ● *les méthodes de traces*

```
• debug(log_message, [*args[, **kwargs]])  
• info(log_message, [*args[, **kwargs]])  
• warning(log_message, [*args[, **kwargs]])  
• error(log_message, [*args[, **kwargs]])  
• critical(log_message, [*args[, **kwargs]])  
• exception(message[, *args])  
• log(log_level, log_message, [*args[, **kwargs]])
```

### ● *Déterminent quels sont les messages à afficher*

### ● *Diffusion des messages vers les gestionnaires enregistrés*



- Les gestionnaires de messages écrivent les logs sur un support particulier
- Exemples de Gestionnaires (Handler)
  - *StreamHandler / FileHandler*
  - *RotatingFileHandler / TimedRotatingFileHandler*
  - *SocketHandler / DatagramHandler*
  - *SysLogHandler / NTEventLogHandler*
  - *SMTPHandler / MemoryHandler / HTTPHandler*



## ● Le layout des traces est défini avec :

<code>%(name)s</code>	Name of the logger (logging channel)
<code>%(levelno)s</code>	Numeric logging level for the message (DEBUG, INFO, WARNING, ERROR, CRITICAL)
<code>%(levelname)s</code>	Text logging level for the message ("DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL")
<code>%(pathname)s</code>	Full pathname of the source file where the logging call was issued (if available)
<code>%(filename)s</code>	Filename portion of pathname
<code>%(module)s</code>	Module (name portion of filename)
<code>%(lineno)d</code>	Source line number where the logging call was issued (if available)
<code>%(created)f</code>	Time when the LogRecord was created (time.time() return value)
<code>%(asctime)s</code>	Textual time when the LogRecord was created
<code>%(msecs)d</code>	Millisecond portion of the creation time
<code>%(relativeCreated)d</code>	Time in milliseconds when the LogRecord was created, relative to the time the logging module was loaded (typically at application startup time)
<code>%(thread)d</code>	Thread ID (if available)
<code>%(process)d</code>	Process ID (if available)
<code>%(message)s</code>	The result of record.getMessage(), computed just as the record is emitted

<code>%(message)s</code>	The result of record.getMessage(), computed just as the record is emitted
<code>%(process)s</code>	Process ID (if available)
<code>%(thread)s</code>	Thread ID (if available)



## ● Exemple d'utilisation

```
import logging, logging.handlers
# create logger
logger = logging.getLogger("log_sample")
logger.setLevel(logging.DEBUG)
# create console handler and set level to debug
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
# un autre handler : rotting : Add the log message handler to the logger
rh = logging.handlers.RotatingFileHandler("biblio.log", maxBytes=200, backupCount=5)
# create formatter
formatter = logging.Formatter("%(asctime)s - %(name)s - %(levelname)s - %(message)s")
# add formatter to ch
ch.setFormatter(formatter)
rh.setFormatter(formatter)
# add ch to logger
logger.addHandler(ch)
logger.addHandler(rh)
# "application" code
logger.debug("debug message") logger.info("info message")
logger.warn("warn message")
logger.error("error message")
logger.critical("critical message")
```



## ● Exemple de fichier de configuration : logging.conf

```
[loggers]  
keys=root,logger1
```

```
[handlers]  
keys=consoleHandler,fileHandler
```

```
[formatters]  
keys=simpleFormatter
```

```
[logger_root]  
level=DEBUG  
handlers=consoleHandler
```

```
[logger_logger1]  
level=INFO  
handlers=consoleHandler,fileHandler  
qualname=logger1  
propagate=0
```

```
[handler_consoleHandler]  
class=StreamHandler  
level=DEBUG  
formatter=simpleFormatter  
args=(sys.stdout,)
```

```
[handler_fileHandler]  
class=handlers.RotatingFileHandler  
level=INFO  
formatter=simpleFormatter  
args=("biblio.log", 200, 5)
```

```
[formatter_simpleFormatter]  
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s  
datefmt=
```

```
import logging  
import logging.config  
  
logging.config.fileConfig("logging.conf")  
# create logger  
logger = logging.getLogger("logger1")  
# "application" code  
logger.debug("debug message")  
logger.info("info message")  
logger.warn("warn message")  
logger.error("error message")  
logger.critical("critical message")
```



## ● Multi-Threading : Le Threading est une technique permettant l'exécution des tâches en parallèles

```
import threading, zipfile

class AsyncZip(threading.Thread):
    def __init__(self, infile, outfile):
        threading.Thread.__init__(self)
        self.__infile = infile
        self.__outfile = outfile

    def run(self):
        f = zipfile.ZipFile(self.__outfile, 'w', zipfile.ZIP_DEFLATED)
        f.write(self.__infile)
        f.close()
        print 'Finished background zip of: ', self.infile

background = AsyncZip('mydata.txt', 'myarchive.zip')
background.start()
print 'The main program continues to run in foreground.'
background.join() # Wait for the background task to finish
print 'Main program waited until background was done.'
```



## ● python fournit plusieurs classes pour faciliter la gestion de la concurrence entre threads:

- *Lock,*
- *RLock,*
- *Semaphore*
- ...

```
class DepilerThread(threading.Thread):
    """
    depiler dans une pile
    """
    def __init__(self, pile, nombre, lock):
        threading.Thread.__init__(self)
        self.__pile = pile
        self.__nombre = nombre
        self.__lock = lock

    def run(self):
        print "DepilerThread {0} démarre".format(self.getName())
        for elem in xrange(self.__nombre):
            self.__lock.acquire()
            print ">>>>>DepilerThread {0} debut depiler".format(self.getName())
            elem = self.__pile.depiler()
            print "-----DepilerThread {0} depile {1}".format(self.getName(), elem)
            self.__lock.release()
            print "-----DepilerThread {0} termine".format(self.getName())

if __name__ == "__main__":
    _NBR = 1000
    _P1 = piles.Pile(_NBR)
    _COND = threading.RLock()
    _TH_E2 = DepilerThread(_P1, _NBR, _COND)
    _TH_E3 = EmpilerThread(_P1, _NBR, _COND)
    _TH_E4 = DepilerThread(_P1, _NBR, _COND)
    print "Fin programme principal"
```



# threading : la synchronization

## la classe Condition facilite la synchronisation entre threads:

- wait
- notify
- notifyAll

```
if __name__ == "__main__":  
    _Nombre = 1000  
    _Pile = piles.Pile(_Nombre)  
    _Lock = threading.RLock()  
    _Cond = threading.Condition(_Lock)  
    _EmpilerTh1 = EmpilerThread(_Pile, _Nombre, _Cond)  
    _EmpilerTh2 = EmpilerThread(_Pile, _Nombre, _Cond)
```

```
class DepilerThread(threading.Thread):  
    """  
    depiler dans une pile  
    """  
    def __init__(self, pile, nombre, lock):  
        threading.Thread.__init__(self)  
        self.__pile = pile  
        self.__nombre = nombre  
        self.__lock = lock  
  
    def run(self):  
        print "DepilerThread {0} démarre".format(self.getName())  
        for elem in xrange(self.__nombre):  
            self.__lock.acquire()  
            while len(self.__pile) <= 0:  
                # pile est vide  
                print ">>>>>DepilerThread {0} depile => je s'attends"  
                self.__lock.wait()  
            print ">>>>>DepilerThread {0} depiler => je retire"  
            elem = self.__pile.depiler()  
            self.__lock.notify()  
            print "-----DepilerThread {0} depile {1}".format(self.getName(), elem)  
            self.__lock.release()  
        print "-----DepilerThread {0} termine".format(self.getName())
```





## ● Démarche d'accès aux Base de données

- *Importation de module d'accès aux BD*
- *Ouverture d'une connexion*
- *Exécution des requêtes*
- *Libération des ressources*



- L'API python d'accès aux BdD fournit à l'image de l'API Java JDBC, des interfaces génériques indépendante de SGBD
- Il est nécessaire d'installer un pilote de la BdD choisie
- MySQLdb est une implémentation de DB-API pour MySQL

```
>>> import MySQLdb
Traceback (most recent call last):
File "<stdin>", line 1, in ?
ImportError: No module named MySQLdb
```



## ● Exemples de scripts d'accès à BdD

```
# server_version.py -  
#retrieve and display database server version  
  
import MySQLdb  
  
conn = MySQLdb.connect ( host = "localhost",  
                        user = "testuser",  
                        passwd = "testpass",  
                        db = "test")  
  
cursor = conn.cursor ()  
  
cursor.execute ("SELECT VERSION()")  
  
row = cursor.fetchone ()  
  
print "server version:", row[0]  
  
cursor.close ()  
  
conn.close ()
```



## ● Exemples : Création de table et insertion de données

```
# Création d'une table
# Insertion des données dans la BdD
import MySQLdb
conn = MySQLdb.connect ( host = "localhost",...)

cursor = conn.cursor ()
cursor.execute ("DROP TABLE IF EXISTS animal")
cursor.execute ("""
    CREATE TABLE animal(
        name CHAR(40),
        category CHAR(40) ) """)
cursor.execute ("""
    INSERT INTO animal (name, category)
    VALUES
    ('snake', 'reptile'),
    ('frog', 'amphibian'),
    ('tuna', 'fish'),
    ('raccoon', 'mammal')""")

print "Number of rows inserted: %d" % cursor.rowcount
```



## ● Exemples : Récupération de données

```
#Récupération des données avec DB-API

# perform a fetch loop using fetchone()

cursor.execute ("SELECT name, category FROM animal")

while (1):
    row = cursor.fetchone ()
    if row == None:
        break
    print "%s, %s" % (row[0], row[1])

print "Number of rows returned: %d" % cursor.rowcount

# perform a fetch loop using fetchall()
cursor.execute ("SELECT name, category FROM animal")
rows = cursor.fetchall ()
for row in rows:
    print "%s, %s" % (row[0], row[1])
print "Number of rows returned: %d" % cursor.rowcount
```



## ● Exemples : Récupération des données en utilisant les noms de colonnes

```
#Récupération des données avec DB-API en utilisant les noms de colonnes
try :
    cursor.execute ("""UPDATE animal SET name = %s
                      WHERE name = %s""", ("snake", "turtle"))
    print "Number of rows updated: %d" % cursor.rowcount
    # create a dictionary cursor so that column values
    # can be accessed by name rather than by position
    cursor.close ()
    cursor = conn.cursor (MySQLdb.cursors.DictCursor)
    cursor.execute ("SELECT name, category FROM animal")
    result_set = cursor.fetchall ()

    for row in result_set:
        print "%s, %s" % (row["name"], row["category"])
        print "Number of rows returned: %d" % cursor.rowcount

    cursor.close ()
except MySQLdb.Error, e:
    print "Error %d: %s" % (e.args[0], e.args[1])
sys.exit (1)
```

# Python - GUI avec Tkinter





- Tkinter correspond à tk interface
- Il s'agit d'un tool kit graphique qui permet la programmation des interfaces graphiques
- Il facilite la création des composants graphique nommés widget:
  - *Frame, Button, Label, Entry*
  - *RadioButton*
  - *Autres*



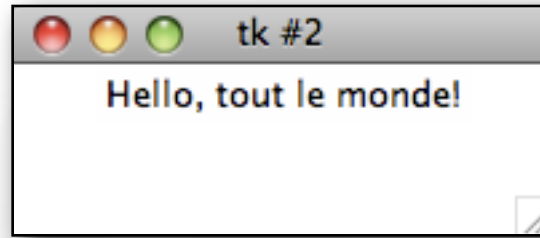


- **tkinter est un module python**
- **Afin d'utiliser tkinter dans un programme python, il faut l'importer!!!**
  - *from Tkinter import \**
- **Les exemples de ce chapitre ne constitue pas la liste exhaustive de l'ensemble des widget du module**



- L'exemple illustre la création d'un label dans une fenêtre

- Exemple



```
# tkinterhello.py
# creates a tk widget
from Tkinter import *
root = Tk()
w = Label(root, text = "Hello, tout le monde!")
w.pack()
root.mainloop()
```



- **root = Tk()**

- *Crée un widget racine qui correspond à une fenêtre avec une barre de titre à la base constituée d'un titre et des boutons*

- **w = Label(root, text='...')**

- *Correspond à widget permettant l'affichage d'un libellé*

- **w.pack()**

- *permet de recalculer la taille de la fenêtre par rapport au contenu et d'afficher la fenêtre*

- **root.mainloop()**

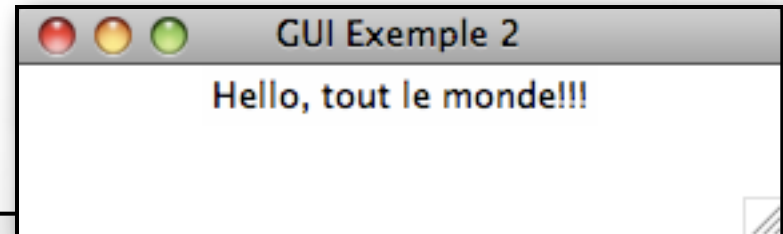
- *ajouter un écouteur d'événement*





- L'exemple illustre la création d'un label dans une fenêtre avec un titre

- Exemple



```
# tkinterhello.py
# creates a window with a title
# adds a title to the window
from Tkinter import *
root = Tk()
root.title("GUI Exemple 2")
w = Label(root, text = "Hello, tout le monde!!!")
w.pack()
root.mainloop()
```

- Il est possible de préciser la taille de la fenêtre

- ***root.minsize(width=250, height=25)***

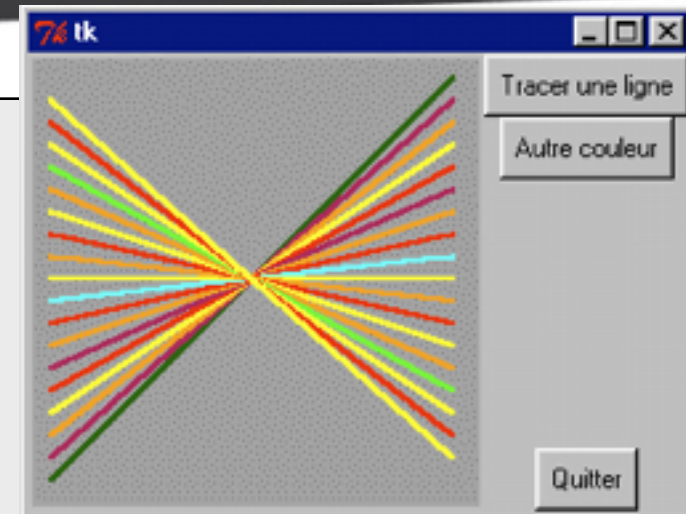


## tkinter - Exemple III

```
# Petit exercice utilisant la bibliothèque graphique Tkinter
from Tkinter import *
from random import randrange
# définition des fonctions gestionnaires d'événements :
def drawline():
    "Tracé d'une ligne dans le canevas can1"
    global x1, y1, x2, y2, coul
    can1.create_line(x1,y1,x2,y2,width=2,fill=coul)
    # modification des coordonnées pour la ligne suivante :
    y2, y1 = y2+10, y-10

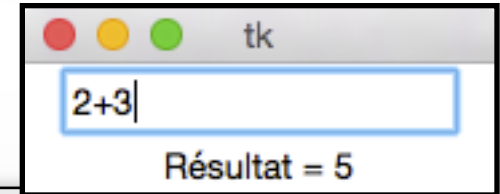
def changecolor():
    "Changement aléatoire de la couleur du tracé"
    global coul
    pal=['purple','cyan','maroon','green','red','blue','orange','yellow']
    c = randrange(8) # => génère un nombre aléatoire de 0 à 7
    coul = pal[c]

#Programme principal #
les variables suivantes seront utilisées de manière globale :
x1, y1, x2, y2 = 10, 190, 190, 10 # coordonnées de la ligne
coul = 'dark green' # couleur de la ligne
# Création du widget principal ("maître") :
fen1 = Tk()
# création des widgets "esclaves" :
can1 = Canvas(fen1,bg='dark grey',height=200,width=200) can1.pack(side=LEFT)
bou1 = Button(fen1,text='Quitter',command=fen1.quit) bou1.pack(side=BOTTOM)
bou2 = Button(fen1,text='Tracer une ligne',command=drawline) bou2.pack()
bou3 = Button(fen1,text='Autre couleur',command=changecolor) bou3.pack()
fen1.mainloop() # démarrage du réceptionnaire d'événements
fen1.destroy() # destruction (fermeture) de la fenêtre
```





## ● Calculatrice



```
# Exercice utilisant la bibliothèque graphique Tkinter et le module math
```

```
from Tkinter import *  
from math import *
```

```
# définition de l'action à effectuer si l'utilisateur actionne  
# la touche "enter" alors qu'il édite le champ d'entrée :
```

```
def evaluer(event):  
    chaine.configure(text = "Résultat = " + str(eval(entree.get())))
```

```
# Programme principal :
```

```
fenetre = Tk()  
entree = Entry(fenetre)
```

```
entree.bind("<Return>", evaluer)
```

```
chaine = Label(fenetre)
```

```
entree.pack()
```

```
chaine.pack()
```

```
fenetre.mainloop()
```



# tkinter - les widgets

Widget	Description
Button	Un bouton classique, à utiliser pour provoquer l'exécution d'une commande quelconque.
Canvas	Un espace pour disposer divers éléments graphiques. Ce widget peut être utilisé pour dessiner, créer des éditeurs graphiques, et aussi pour implémenter des widgets personnalisés.
Checkbutton	Une « case à cocher » qui peut prendre deux états distincts (la case est cochée ou non). Un clic sur ce widget provoque le changement d'état.
Entry	Un champ d'entrée, dans lequel l'utilisateur du programme pourra insérer un texte quelconque à partir du clavier.
Frame	Une surface rectangulaire dans la fenêtre, où l'on peut disposer d'autres widgets. Cette surface peut être colorée. Elle peut aussi être décorée d'une bordure.
Label	Un texte (ou libellé) quelconque (éventuellement une image).
Listbox	Une liste de choix proposés à l'utilisateur, généralement présentés dans une sorte de boîte. On peut également configurer la Listbox de telle manière qu'elle se comporte comme une série de « boutons radio » ou de cases à cocher.
Menu	Un menu. Ce peut être un menu déroulant attaché à la barre de titre, ou bien un menu « <i>pop up</i> » apparaissant n'importe où à la suite d'un clic.
Menubutton	Un bouton-menu, à utiliser pour implémenter des menus déroulants.
Message	Permet d'afficher un texte. Ce widget est une variante du widget Label, qui permet d'adapter automatiquement le texte affiché à une certaine taille ou à un certain rapport largeur/hauteur.
Radiobutton	Représente (par un point noir dans un petit cercle) une des valeurs d'une variable qui peut en posséder plusieurs. Cliquer sur un « bouton radio » donne la valeur correspondante à la variable, et "vide" tous les autres boutons radio associés à la même variable.
Scale	Vous permet de faire varier de manière très visuelle la valeur d'une variable, en déplaçant un curseur le long d'une règle.
Scrollbar	« ascenseur » ou « barre de défilement » que vous pouvez utiliser en association avec les autres widgets : Canvas, Entry, Listbox, Text.
Text	Affichage de texte formaté. Permet aussi à l'utilisateur d'éditer le texte affiché. Des images peuvent également être insérées.
Toplevel	Une fenêtre affichée séparément, « par-dessus ».

Toplevel	Une fenêtre affichée séparément, « par-dessus ».
Text	Affichage de texte formaté. Permet aussi à l'utilisateur d'éditer le texte affiché. Des images peuvent également être insérées.

# PyGTK







- **PyGTK est un module python qui facilite l'intégration de la bibliothèque GTK+ GUI**
- **PyGTK a été initié par James Henstridge**
- **GTK+ (GIMP Toolkit) est une riche boîte à outils de composants graphique développée en C, distribuée sous la licence GNU LGPL 2.1**
- **GTK+ fait partie du projet GNOME**



## ● PyGTK est composé de plusieurs modules:

GObject

\* Les classes , attributs et fonctions de base

ATK

\* Accessibilité

GTK

\* Composants graphique

Pango

\* Internationalisation

Cairo

\* Composants 2D

Glade

\* Création de GUI à partir d'une description XML



## ● Example

```
import pygtk
pygtk.require('2.0')
import gtk

class Base:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.show()

def main(self):
    gtk.main()
    print __name__
if __name__ == "__main__":
    base = Base()
    base.main()
```



## ● Exemple

```
class HelloWorld:
    def hello(self, widget, data=None):
        print "Hello World"
    def delete_event(self, widget, event, data=None):
        print "delete event occurred"
        return False
    def destroy(self, widget, data=None):
        gtk.main_quit()

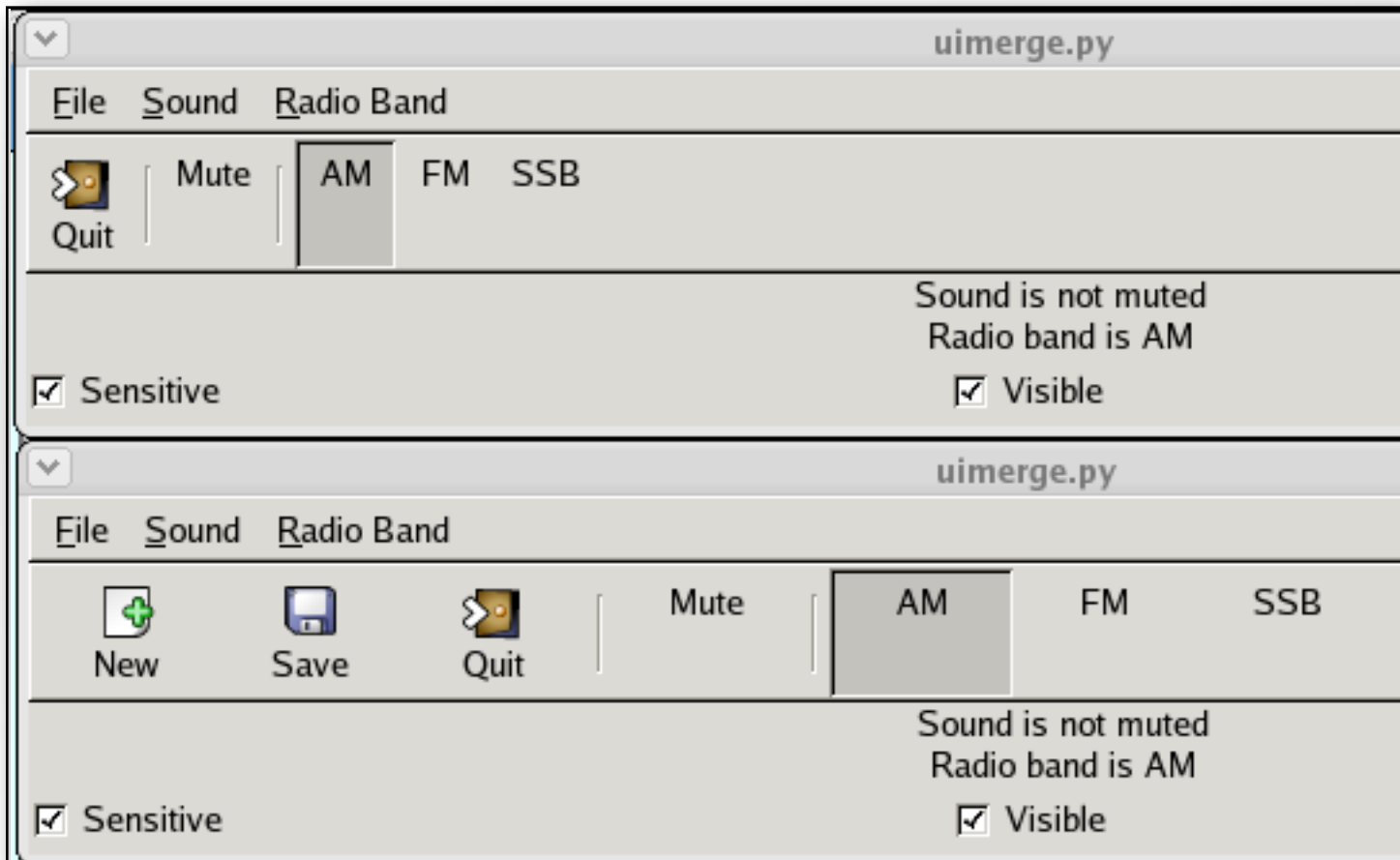
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.connect("delete_event", self.delete_event)
        self.window.connect("destroy", self.destroy)
        self.window.set_border_width(10)
        self.button = gtk.Button("Hello World")
        self.button.connect("clicked", self.hello, None)
        self.button.connect_object("clicked", gtk.Widget.destroy, self.window)
        self.window.add(self.button)
        self.button.show()
        self.window.show()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    hello = HelloWorld()
    hello.main()
```

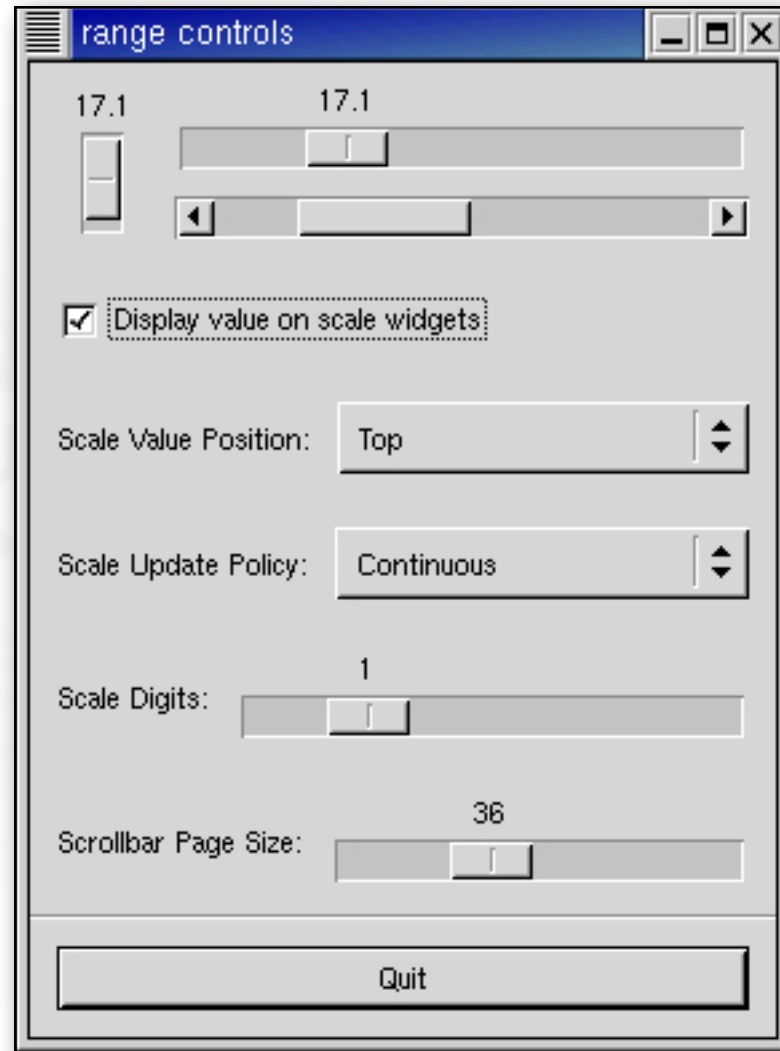


## ● Exemple d'interface graphique





## ● Exemple d'interface





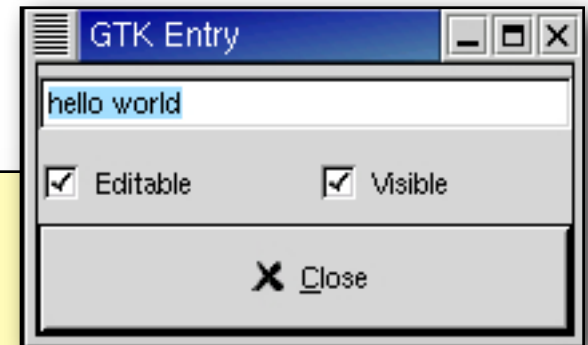
## ● Example

```
class EntryExample:
    """ """
    def enter_callback(self, widget, entry):
        entry_text = entry.get_text()
        print "Entry contents: %s\n" % entry_text

    def entry_toggle_editable(self, checkbutton, entry):
        entry.set_editable(checkbutton.get_active())

    def entry_toggle_visibility(self, checkbutton, entry):
        entry.set_visibility(checkbutton.get_active())

    def __init__(self):
        # create a new window
        window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        window.set_size_request(200, 100)
        window.set_title("GTK Entry")
        window.connect("delete_event", lambda w,e: gtk.main_quit())
        vbox = gtk.VBox(False, 0)
        window.add(vbox)
        vbox.show()
```





## ● Exemple

```
entry = gtk.Entry()
entry.set_max_length(50)
entry.connect("activate", self.enter_callback, entry)
entry.set_text("hello")
entry.insert_text(" world", len(entry.get_text()))
entry.select_region(0, len(entry.get_text()))
vbox.pack_start(entry, True, True, 0)
entry.show()
hbox = gtk.HBox(False, 0)
vbox.add(hbox)
hbox.show()
check = gtk.CheckButton("Editable")
hbox.pack_start(check, True, True, 0)
check.connect("toggled", self.entry_toggle_editable, entry)
check.set_active(True)
check.show()
check = gtk.CheckButton("Visible")
hbox.pack_start(check, True, True, 0)
check.connect("toggled", self.entry_toggle_visibility, entry)
check.set_active(True)
check.show()
```





## ● Example

```
button = gtk.Button(stock=gtk.STOCK_CLOSE)
button.connect("clicked", lambda w: gtk.main_quit())
vbox.pack_start(button, True, True, 0)
button.set_flags(gtk.CAN_DEFAULT)
button.grab_default()
button.show()
window.show()

def main():
    gtk.main()
    return

if __name__ == "__main__":
    EntryExample()
    main()
```

# Pylint et PyChecker





## ● Pychecker est outil de vérification de code

## ● Exemple d'erreurs

- No global found (e.g., using a module without importing it)
- Passing the wrong number of parameters to functions/methods/constructors
- Passing the wrong number of parameters to builtin functions & methods
- Using format strings that don't match arguments
- Using class methods and attributes that don't exist
- Changing signature when overriding a method
- Redefining a function/class/method in the same scope
- Using a variable before setting it
- self is not the first parameter defined for a method
- Unused globals and locals (module or variable)
- Unused function/method arguments (can ignore self)
- No doc strings in modules, classes, functions, and methods



- **Il s'agit d'un outil de vérification de modules python**
  - *Paramétrable*
  - *Personnalisable (ajouter des plugins)*
- **Il permet à l'image de PyChecker de vérifier le code python**
  - *la taille des lignes*
  - *les noms de variables / règles définie*
  - *respect de Python Style (voir le guide)*
    - <http://www.python.org/doc/essays/styleguide.html>



## ● L'outil pylint nécessite l'installation des dépendances suivantes:

- *logilab-astng (version  $\geq 0.14$ )*
- *ogilab-common (version  $\geq 0.13$ )*
- *optik (only for python  $< 2.3$ ) packages*

## ● Les urls

- *<http://www.logilab.org/projects/astng>*
- *<http://www.logilab.org/projects/common>*
- *<http://optik.sourceforge.net/>*



- **Après installation, il est possible d'analyser les modules python**
  - *pylint mymodule.py (win —reports yes module.py)*
  - *pylint [options] module\_or\_package*
  - *pylint directory/mymodule.py*
  - *pylint-gui nécessite TkInter*



## ● Il est possible de modifier les expressions régulières utilisées pour les conventions de nommage

- `pylint --generate-rcfile > ~/.pylintrc`

- *Section [BASIC]*

- `function-rgx=_?_[a-z][A-Za-z0-9]{1,30}$`
- `method-rgx=_?_[a-z][A-Za-z0-9]{1,30}$`
- `attr-rgx=_?_[a-z][A-Za-z0-9]{1,30}$`
- `argument-rgx=_?[a-z][A-Za-z0-9]{1,30}$`
- `variable-rgx=_?[a-z][A-Za-z0-9]{1,30}$`
- `inlinevar-rgx=_?[a-z][A-Za-z0-9]{1,30}$`



- Les messages de pylint ont le format suivant
  - *MESSAGE\_TYPE: LINE\_NUM:[OBJECT:] MESSAGE*
- Les rapports sont écrit en html ou texte
- Exemples:

```
***** Module pylint.checkers.format
W: 50: Too long line (86/80)
W:108: Operator not followed by a space
      print >>sys.stderr, 'Unable to match %r', line
          ^
W:141: Too long line (81/80)
W: 74:searchall: Unreachable code
W:171:FormatChecker.process_tokens: Redefining built-in (type)
W:150:FormatChecker.process_tokens: Too many local variables (20/15)
W:150:FormatChecker.process_tokens: Too many branches (13/12)
```





- Python intègre un débogueur pdb utilisé en ligne de commande
- Il existe plusieurs IDE facilitant le débogage des modules python
  - *Komodo*
  - *NetBeans*
  - *PyDev (plugin pour Eclipse)*
  - *WingIde*
  - *PyScripter*



## ● Les options de pdb

- *s : step*
- *list : visualiser ligne courante*
- *b : break (définir un breakpoint)*
- *c : continue (avancer jusqu'au breakpoint)*
- *w : where (la pile d'appels)*
- *u : up (remonter d'un niveau dans la pile)*
- *d : down (descendre dans la pile d'appel)*

# Intégration de Java avec Python : Jython





- Jython est une implémentation pure Java du langage dynamique et orienté objet python.
- Il s'intègre d'une façon transparente avec la plateforme Java
- Il permet d'utiliser python sur toute plateforme Java
- Il ajoute le mécanisme de scripting au langage Java



- **Jython, lest you do not know of it, is the most compelling weapon the Java platform has for its survival into the 21st century:-)**
- **L'ancêtre de Jython 'JPython' est certifié 100% pure Java**
- **Jython est gratuit et disponible pour l'usage commercial et non commercial**
- **Un bon complémentaire à Java**



## ● Jython convient pour :

- *Script : offre aux développeurs d'intégrer les bibliothèques Jython pour permettre aux utilisateurs l'usage des script*
- *Interactivité: Jython offre un interpréteur qui peut interagir avec les bibliothèques Java=> possibilité de debugger tout système Java avec Jython*
- *RAD : Rapid Application Development => possibilité de mélanger les 2 langages*



- Jython est lancé par un script qui lance la machine virtuelle Java et de positionner les variables d'environnement (install.path..) et exécute la classe **org.python.util.jython**

- ***jython [options] [-jar jar | -c cmd | file | -] [args]***

- *-jar jar : le programme à exécuter est spécifié dans le \_\_init\_\_.py*
- *- c cmd le programme est passé en ligne de commande*
- *args : les arguments sont stockés dans sys.argv[1:]*
- *--help*
- *--version*



- Jython simplifie l'accès aux bibliothèques Java à partir de python
- Utilisation de la classe `java.util.Random`



```
Terminal — java — 80x21
Type "copyright", "credits" or "license" for more information.
>>> from java.util import Random
>>> r = Random()
>>> r.nextInt()
-2029514076
>>> for i in range(5):
...     print r.nextDouble()
...
0.2568514761123678
0.06415976689240344
0.22799544018571927
0.5576855760743459
0.6731367401476701
>>> 
```





# Mapping Java/Python

Java Types	Allowed Python Types
char	String (must have length 1)
boolean	Integer (true = nonzero)
byte, short, int, long	Integer
float, double	Float
java.lang.String, byte[], char[]	String
java.lang.Class Class or JavaClass	(only if class subclasses from exactly one Java class; mutiple inheritance from more than one Java class is now illegal)
Foo[]	Array (must contain objects of class or subclass of Foo)
java.lang.Object	String->java.lang.String, all others unchanged
org.python.core.PyObject	All unchanged
Foo	Instance->Foo (if Instance is subclass of Foo); JavaInstance -> Foo (if JavaInstance is instance of Foo or subclass)

Returned values from a Java method are also possibly coerced back to an object that is more readily usable in Python. The following table shows those coercions.

Java Type	Returned Python Type
char	String (of length 1)
boolean	Integer (true = 1, false = 0)
byte, short, int, long	Integer
float, double	Float
java.lang.String	String
java.lang.Class	JavaClass which represents given Java class
Foo[]	Array (containing objects of class or subclass of Foo)
org.python.core.PyObject (or subclass)	Unchanged
Foo	JavaInstance which represents the Java Class Foo



## ● Les attributs des JavaBean

```
b = awt.Button()  
b.setEnabled(0)  
  
#Accès direct  
b = awt.Button()  
b.enabled = 0  
  
#Lors de l'instantiation  
= awt.Button(enabled=0)
```

## ● Les Tuples

```
# setSize(length,width)  
frame = awt.Frame(size=(500,100))  
#RGB  
frame.background = 255,255,0
```



## ● Le module jarray fournit le mécanisme de conversion de tableau Java

- *zeros(length, type)*
- *array(sequence, type)*

```
from jarray import array
a = array([ 1 ], 'i')
print a
from java.net import URL
u = URL('http://jython.org')
b = array([], URL)
print b
```

Character Typecode	Corresponding Java Type
z	boolean
c	char
b	byte
h	short
i	int
l	long
f	float
d	double



## ● Exemple

```
from java import awt

class SpamListener(awt.event.ActionListener):
    def actionPerformed(self, event):
        if event.getActionCommand() == "Spam":
            print 'Spam and eggs!'

f = awt.Frame("Subclassing Example")
b = awt.Button("Spam")
b.addActionListener(SpamListener())
f.add(b, "Center")
f.pack()
f.setVisible(1)
```

## ● Surcharge en Jython

Classe Java java.io.InputStream

1. abstract int read()
2. int read(byte[])
3. int read(byte[], int, int)

SuperClass.foo(self) permet d'utiliser la méthode de la super classe

```
from java.io import InputStream

class InfiniteOnes(InputStream):
    def read(self, *args):
        if len(args) > 0:
            # int read(byte[])
            # int read(byte[], int, int)
            return apply(InputStream.read, (self,) + args)
        return 1

io = InfiniteOnes()

for i in range(10):
    print io.read(),
print
```

Résolution de la surcharge



## ● Embedding Python : Utilisation de Python Interpreter

● Voir : [org.python.util.PythonInterpreter](#)

```
import org.python.core.PyException;
import org.python.core.PyInteger;
import org.python.core.PyObject;
import org.python.util.PythonInterpreter;

public class SimpleEmbedded {

    public static void main(String[] args) throws PyException {
        PythonInterpreter interp = new PythonInterpreter();
        interp.exec("import sys");
        interp.exec("print sys");
        interp.set("a", new PyInteger(42));
        interp.exec("print a");
        interp.exec("x = 2+2");
        PyObject x = interp.get("x");
        System.out.println("x: " + x);
    }
}
```



## ● La JSR 223 : Scripting Java

● *PythonEngine* : <https://scripting.dev.java.net/>

```
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;

public class JSR223 {

    public static void main(String[] args) throws ScriptException {
        ScriptEngine engine = new ScriptEngineManager().getEngineByName("python");
        engine.eval("import sys");
        engine.eval("print sys");
        engine.put("a", 42);
        engine.eval("print a");
        engine.eval("x = 2 + 2");
        Object x = engine.get("x");
        System.out.println("x: " + x);
    }
}
```

# Intégration de C





## ● L'intégration d'une bibliothèque C

- *Utilisation de Python.h : Définition des types et méthodes de conversion*
- *Déclaration d'une fonction C*
- *Déclaration du nom de la fonction Python équivalente*
- *Déclaration du module python à importer*
- *Compilation de la bibliothèque*





## ● Déclaration d'une fonction C

```
#include <Python.h>

// Function to be called from Python
static PyObject* py_myFunction(PyObject* self, PyObject* args) {

    char* s = "Vendredi - OrangeLabs a Belford"

    return Py_BuildValue("s", s);

}
```



## ● Déclaration du nom de la fonction Python équivalente

```
// Bind Python function names to our C functions

static PyMethodDef myModule_methods[] = {
    {"myFunctionPython", py_myFunction, METH_VARARGS},

    {"myOtherFunctionPython", py_myOtherFunction, METH_VARARGS},

    {NULL, NULL}
};
```



## ● Déclaration du module python à importer

```
// Python calls this to let us initialize our module  
void initmyModule() {  
    (void) Py_InitModule("myModule", myModule_methods);  
}
```

## ● Compilation de la bibliothèque



## ● Utilisation dans un module python

### ● *import de la bibliotheque c comme module python*

```
# import de la bibliothèque comme module  
# myModule.so  
  
from myModule import *  
  
print myFunctionPython()  
  
print myOtherFunctionPython(8,2)
```