Programmation - Introduction

- Objectifs :
  - Apprendre à raisonner à partir d'algorithmes.
  - Traduire ces raisonnements en programmes informatiques.
- Outils :
  - Langage de programmation Python.
  - Editeur de texte **Emacs** (environnememt de programmation)
  - Environnement**Unix - Linux** .

Qu'est ce qu'un programme ?

- Un fichier ***source***.
- Un ensemble de ***déclarations*** et d'***instructions***.
- Des librairies chargées dynamiquenment ou statiquement.
- Un fichier ***machine*** : bytecode, exécutable.

Vous pouvez générer :

- un programme exécutable directement par un utilisateur
- un programme nécessitant une ***machine virtuelle*** - interpréteur
- ou un autre programme pour exécuter votre programme (applet java exécutée par un navigateur Web).

Respect des règles lexicales et syntaxiques

- Le programme source doit impérativement respecter les ***règles syntaxiques et lexicales*** du langage de programmation :
  - respect des mots réservés,
  - respect du parenthèsage et de l'indentation,
  - respect des règles idiomatiques.

Raisonner différemment

- Ce cours de programmation reprend des éléments du cours d'algorithmique.
- Traduire un problème en ***procèdures automatisables***.
- ***Construire un raisonnement*** qui est reproductible quelque soit le langage utilisé.

Le langage Python - 1

- Ce langage est développé depuis 1989 par *Guido van Rossum* et de nombreux collaborateurs bénévoles.
- Python est *OpenSource* et gratuit
- Pyhton est portable
- La syntaxe de Python est simple mais offre la possibilité de construire et manipuler des types de données évolués.
- C'est un langage orienté objet : possibilité d'héritage, de surcharghe des opérateurs,

Le langage Python - 2

- La gestion de la mémoire est automatisée, appel au `Garbage Collector`
- il est interfaçable avec de nombreux langage : Perl, Java (JPython), C, ainsi qu'avec les SGBD.
- De nombreuses librairies de programmes sont disponibles : TKinter (graphisme), NumPy, Pickle (objets persistants), gadfly (SQL).

Bibliographie

Les livres :

- *Apprendre à programmer avec Python* Gerard Swinnen, ed O'Reilly, 2005.
- *Python précis et concis* Mark Lutz, ed O'Reilly, 2000.
- *Core Python programming*Wesley J. Chun, ed Prentice Hall, 2001.

Bibliographie

Les sites Web :

- Le site internet : `//www.python.org`
- *Note de cours pour l'apprentissage de la programmation Python* Gérard Swinnen, `http://inforef.be/swi/python.htm`
- ...à chercher dans : `www.google.fr`.

Mon premier programme Pyhon

- Respectons la tradition : Affichage de “Hello World”
- Code du programme :

```
print("Hello World")
```

- ou encore en utilisant une variable :

```
c="Hello World"
\\
print(c)
```

Comment exécuter un programme Python

- Lancer l'interpréteur `python` et
  taper son code à l'apparition du `prompt` `>>>`.
  A chaque fois que vous faites `Enter` le code est interprété.
- Ecrire un fichier dans un éditeur de texte, puis lancer son exécution avec `python nomfichier.py` depuis une fenêtre de `shell`.

Règles d'écriture - 1

- La *casse* est significative.
- Le typage des variables est dynamique.
- Une instruction doit commencer en ***première*** colonne.
- L'identation est obligatoire pour marquer les blocks.
- Les instructions composées ont
  - une entête suivie de " : "
  - des instructions indentées.

Règles d'écriture - 2

- Si une instruction dépasse la taille d'une ligne il est possible soit d'écrire un caractère de continuation ou de mettre l'instruction entre `()`.

```
if a == b and c == d and \
    d==e :
      print ok
```

```
if (a == b and c == d and
    d==e) :
      print ok
```

Codage Binaire

- Dans une machine binaire, il n'existe que 2 valeurs possible : 0 et 1.
- Ceci nécessite une traduction de chaque variable, instruction . . .
- La plus petite unité de valeur dans un ordinateur binaire est le *bit*. C'est lui qui prend la valeur 0 ou 1.
- Un mot machine est un ensemble de 8 bits : un *octet*.

Codage des caractères

- Pour chaque caractère il existe une valeur numérique correspondante.
- C'est le code ASCII : *American Standard Code for Information Interchange*.
- Ce code est écrit sur 7 bits soit 127 valeurs possibles - donc 128 caractères.
- Les codes 0 à 31 représentent les caractères de contrôle.
- Les codes 65 à 90 représentent les majuscules.
- Les codes 97 à 122 représentent les minuscules.

Type des objets - 1

Type d'objet	Exemple
Nombres	124 3.02 9999L 4.0e+2 3+4j
Chaines	'spam' "d'guido"
Listes	[1,[2,'trois'],4]
Tuples	(1,'spam',4,'U',0)
Dictionnaires	{'nourriture' : 'confiture', 'gout': 'miam'}

Type des objets - 2

- `integer` :
  - Taille : 4 octets (32 bits) de -2 147 483 648 à 2 147 483 649
  - si débordement l'erreur `overflow error` est levée.
- `float` :
  - Taille : 8 octets (64 bits) de 10^{-308} à 10^{308}
  - si débordement pas d'erreur mais affichage de `Inf` (infinity).
- `long` :
  - Pas de limite sauf celle de la mémoire disponible sur la machine.

Type des objets - 3

- `String` :
  - Collection ordonnée de caractères.
  - Si une chaine est encadrée par des ”’ elle peut s´étendre sur plusieurs lignes.
  - c=” désigne la chaine vide.
  - Pas de type `char` (caractère).
  - Pas de gestion de la mémoire par l’utilisateur.

Les opérateurs - 1

- Affectation :
  - $a = 3$
- Arithmétique :
  - $+ \quad - \quad * \quad / \quad \% \quad **$
- Comparaison :
  - $< \quad > \quad <= \quad >= \quad == \quad != \quad !$
- Logique :
  - or, and, not.
- Opérations sur les bits :
  - $<< \quad >> \quad | \quad \&$

Manipulation simple sur les nombres

- Diviser un entier par 2 revient à décaler d'un bit vers la droite.
- Multiplier ce nombre par 2 revient à le décaler vers la gauche en ajoutant un bit à 0.
- Exemple :
  - Codage de 4 = 100
  - Codage de 2 = 10
  - Codage de 8 = 1000

Manipulation simple sur les nombres

- Echanger deux variables

```
print 'donnez un nombre'
nb1=input()
print 'donnez un autre nombre'
nb2=input()
nb1=nb2
nb2=nb1
```

- Où est l'erreur ?

Manipulation simple sur les nombres

- Echanger deux variables

```
print 'donnez un nombre'
nb1=input()
print 'donnez un autre nombre'
nb2=input()
temp=nb1
nb1=nb2
nb2=temp
```

Typage dynamique des variables

- Pour les types simples prédéfinis, pas de déclaration des variables :

```
item=1 # type  entier
print item
item='la maison' # type chaine de caracteres
print item
```

Typage dynamique des variables

- Ne pas mélanger les types :

```
item='5'
item=item+1
```

- Cette instruction provoquera une erreur : Refus d'additionner un entier et une chaine de caractères.

Type de Données - Les opérations sur les chaines

- Surcharge des opérateurs :
  - Concaténation (ajout) + 'spam'+'42'
  - Répétition *
  - Formattage %
- Accès par indiçage :
  `mot="chaine"`
  - mot[2] permet d'obtenir `'a'`
  - mot[:2] permet d'obtenir `'ch'`
  - mot[2:] permet d'obtenir `'aine'`
  - mot[2:4]permet d'obtenir `'ai'`
  - mot[-2] permet d'obtenir `'chai'`

Les chaines de caractères

- Les chaines sont non modifiables sur place.
  - `mot[2]='0'` déclenche une erreur.
- On accède aux caractères de cette chaine en le désignant par sa position.
  - `mot="anticonstitutionnellement"` l'appel à `mot[2]` renvoie le caractère `t`.

Les opérations sur les chaines

- Il est possible d'obtenir la longueur d'une chaine : `len(mot)` renvoie le nombre de caractères contenu dans le mot.
- `chaine1 in chaine2` renvoie vrai si `chaine2` contient `chaine1`
- `chaine1 not in chaine2` renvoie vrai si `chaine2` ne contient pas `chaine1`
- `for i in chaine1` énumère tous les caractères contenus dans `chaine1`

Les opérations sur les chaines

- Transformer une chaine en nombre :

```
mot=''3''
numero=int(mot)
mot2=''3.2''
nombre=float(mot2)
```

- Transformer un nombre en chaine de caractères :

```
num=3
mot=str(num)
```

Les opérations sur les chaines

- Comparer deux chaines de caractères :

```
s1=''bleu''
s2=''bleu''
s3=''Bleu''
s1==s2 renvoie 1 - vrai
s1==s3 renvoie 0 - faux
```

Les structures de contrôle

- Tester une condition afin de controler la suite d'instructions à exécuter : `if`
  - Syntaxe :
    `if` (test booléen) :
        instructions si vrai
    `else` :
        instructions si faux.
  - Exemple :
    `if ( a % 2 == 0):`
        print “a est pair”
    `else` :
        print “a est impair”

Utilisation des opérateurs booléens

- Pour les besoins d'une enquête de santé publique, vous avez un échantillon de population où chaque individu est défini 2 variables `sexe` et `taille`.
- La variable `sexe` peut prendre la valeur 'F'ou 'M'.
- La variable `taille` est un entier
- Pour intégrer un individu dans votre étude, il doit soit être une femme (valeur 'F') et avoir une taille supérieur à 180, soit être un homme (valeur 'M') et avoir une taille inférieur à 160.

Utilisation des opérateurs booléens

```
print 'donnez une valeur pour le sexe'
sexe=raw_input()
print 'donnez une valeur pour la taille'
taille=input()

if (sexe=='F' and taille>180) or
   (sexe=='M' and taille<160):
   print 'vous etes integre'
else:
    print 'desole vous ne correspondez pas
           au profil de l etude '
```

Exercice de logique - 1

- Sélectionner les animaux dont l'âge est supérieur a 3 jours :
- La négation de ceci est : animaux dont l'âge est inférieur ou égale à 3 jours.
- Code python :

```
if age > 3 :
    print "selection"
else:
    print "ce n'est pas le bon age"
```

Sélection multiple

- Sélectionner les animaux dont l'âge est supérieur à 3 jours et inférieur à 5 jours:
- La négation de ceci est : animaux dont l'âge est inférieur ou égale à 3 jours ou supérieur ou égale à 3 jours.
- Code python :

```
if age > 3 and age < 5:
    print "selection"
else:
    print "trop jeune ou trop vieux"
```

Sélection multiple

- Parmi les animaux qui ont plus de 3 jours, affecter le code A à ceux qui sont des males et le code B à ceux qui sont des femelles.
- Code python :

```
if age > 3 :
    if sexe == 'M':
            code = 'A'
    else:
      if sexe == 'F':
            code = 'B'
```

- Caractéristiques de ceux qui n'ont pas reçu de code ?

Sélection multiple

- Gestion des erreurs :

```
if age > 3 :
    if sexe == 'M':
            code = 'A'
    else:
      if sexe == 'F':
            code = 'B'
      else :
         print ``valeur de sexe incorrecte''
```

- Où est l'erreur ?

Sélection multiple

```
print "age de votre animal en nb jours ?"
age = input()
print "donnez le sexe de cet animal"
sexe=raw_input()
if age > 3 :
    if sexe != 'M' and sexe != 'F':
        print "vous devez saisir F ou M"
    else :
        if sexe== 'M':
            code = 'A'
        else:
            if sexe == 'F':
                code = 'B'
print 'le code est ', code
```

Problèmes ?

- Problème de programmation : on accède à la variable `code` alors que peut-être elle n'est pas définie, cela peut provoquer une erreur d'exécution du programme.
- Problème de gestion des données : si l'age est inférieur ou égale à 3 aucun traitement n'est effectué, dans la suite des traitements il y aura des animaux pour lequel il n'y a pas de `code`

Une version possible

```
print "age de votre animal en nb jours ?"
age = input()
print "donnez le sexe de cet animal"
sexe=raw_input()
code =''
if age > 3 :
    if sexe != 'M' and sexe != 'F':
        print "vous devez saisir F ou M"
    else :
        if sexe== 'M':
            code = 'A'
        else:
            if sexe == 'F':
                code = 'B'
```

Une version possible

```
if code != '':
    print 'le code est ', code
else :
    print 'aucun code affecte'
```

La notion de blocs - 1

- Les blocs sont délémités par l'indentation, c.-à-d. par la mise en page.
- Les commentaires (commençant par #) sont ignorés.
- Il est possible de composer - d'imbriquer, des instructions.
  - Bloc d'instructions :

        Ligne d'entête :

            première instruction du bloc

            ... ...

            dernière instruction du bloc.

La notion de blocs - 2

- Instructions imbriquées :

```
if embranchement == ''vertebres'':
   if classe == ''mammiferes'':
      if ordre == ''carnivores'':
         if famille == ''felins'':
           print ''c'est peut-etre un chat''
      print ''c'est en tout cas un mammifere''
   elif classe == ''oiseaux'':
       print ''c'est peut etre un canari''
print (''la classification des animaux
         est complexe'' )
```

Boucles et Itérations

- Objectifs : répéter un certain nombre de fois le même traitement.
- Exemple : Demander à l'utilisateur de donner 10 nombres pour les additionner.
- Code Python des instructions à répéter:

```
print 'donnez un nombre'
nombre = input()
```

- Instructions de répétition :
  - Répéter tant que : `while <test>:`

Boucles et Itérations

- Code Python :

```
compteur = 1
while compteur <=10:
      print 'donnez un nombre'
      nombre = input()
      somme = somme+nombre
      compteur=compteur+1
#La boucle est finie
print 'la somme des 10 nombres est ``, somme
```

Boucles et Itérations

- Construction de factoriel : $f(x) = f(x-1) * x$

```
print "quel factoriel voulez-vous calculer ? "
fact = input()
if fact >= 1 :
  resultat = 1
  i=1
  while i<=fact :
    resultat = resultat * i
    i=i+1
  print "le factoriel de ",fact," est ",
  print resultat
else:
  print "le factoriel de ",fact,
  print " ne peut etre calcule"
```

Boucles et Itérations

- Construction de la suite de Fibonacci :
  $f(x) = f(x-1) + f(x-2)$
- Code Python :

```
a ,b,c =1,1,1
while (c<11):
    print b,
    a,b,c=b,a+b,c+1
```

Boucles et Itérations

- Itérations : `for <cible> in <objet>:`
- Code Python :

```
jours=('lundi','mardi','mercredi')
for item  in jours:
    print item,
```

- Exécution :

```
lundi mardi mercredi
```

Boucles et Itérations

- Affichage de toutes les valeurs comprises entre $0 - 9$:

```
for i in range (10):
        print i
```

- La fonction `range` crée une liste de toutes les valeurs dans l'intervalle $[0 - 9]$
- Elle peut s'écrire `range(valDebut,valFin,pas)`
- Code Python :

```
for i in range (2,10,3):
        print i,
```

- Ceci affiche : 2 5 8

Boucles imbriquées

- Faire la somme de 10 valeurs de poids pour une liste d'animaux donnée
- Code Python :

```
animaux=('rat', 'souris', 'cobaye')
for animal in animaux:
    resultat=0
    for i in range(10):
        print 'donnez poids pour', animal
        poids=input()
        resultat=resultat+ poids
    print 'Somme des poids pour ',animal,
    print 'est ', resultat
print 'fini'
```

Transformer une chaine ADN en ARN

- Saisir la chaine

```
chaine='acgtcgagctgagagcccaa'
```

- La transformer en liste

```
adn=list(chaine)
```

- Convertir les caractères :

```
for i in range (len(adn)):
    if  adn[i] == 'a':
        adn[i]='u'
    elif  adn[i] == 'c':
        adn[i]='g'
    elif  adn[i] == 'g':
        adn[i]='c'
    else :
        adn[i]='a'
```

Transformer une chaine ADN en ARN

- Même chose avec une chaine de caractères et les fonctions associées au type `string`
- Importer la bibliothèque : `import string`

```
>>>adn='acgtcgagctgagagcccaa'
>>>string.join(string.split(adn,'a'),'u')
'ucgtcgugctgugugcccuu'
```

- Attention, pour modifier la cahine adn de facon durable il faut écrire :

```
>>>adn=string.join(string.split(adn,'c'),'g')
```

- Il faut se protéger du fait que si vous modifiez les 'c' en 'g' sans précaution, vous ne pourrez plus différencier les 'g' initiaux et les 'g' qui ont déjà été transcrits.

Le type liste

- Une liste est un ensemble ordonné d´éléments.
- Les éléments peuvent être hétérogènes en terme de type.
- Une liste peut être contenue dans une autre liste.
- Il n'y a pas de limitation (autre que celle de la taille de la mémoire) au nombre de dimmensions d'une liste.

Manipulation des listes

- Création : 2 façons de créer une liste
  1. initialisation avec valeurs :
     ```
     maliste = [“A”,”un animal”,3]
     ```
  2. initialiser une liste vide :
     ```
     autreliste =[]
     ```
  3. initialiser une liste à 2 dimmensions :

```
nouvelleliste = [[1,3,9],[2,4,6],[1,5,8,8]]
```

- Ajout avec création d’un nouvel emplacement à la fin de la liste :
  ```
  maliste.append(“b”)
  ```
- Accès à l´élément dans la case numéro 2 :
  ```
  print maliste[2]
  ```
- Accès dans une liste à 2 dimmensions :
  ```
  print nouvelleliste[0][1]
  ```

e chapitre 5 du tutoriel de la documentation python pour l’ensemble des fonctions disponibles.

Manipulation des listes

- Insertion et création d'un nouvel emplacement entre l'élément dans la case numéro 0 et celui de la case numéro 1 :

  `maliste[1:1] =['j']`
- Suppression :
  1. en citant le numéro de la case :

     `maliste.pop(1)`

     - cette fonction renvoie l'élément supprimé de la liste.
  2. en citant la valeur de l'élément :

     `maliste.remove("un animal")`

Le type dictionnaire

- Déclaration : `dict ={}` ou `dict={'jambon':2,'oeufs':3,'pain':1}`
- Nombre d'éléments : `len(dict)`
- ATTENTION ! `dict[2]` donne une erreur.
- Accès aux éléments `dict['jambon']` renvoie 2.
- Test d'existence : `dict.has_key('pain')` répond 1 pour vrai
- Obtenir la liste des clés : `dict.keys()`
- Obtenir le contenu du dictionnaire sous forme de tuples `dict.items()`

Exemple

- Mémoriser les poids relevés pour 5 souris identifiées par un code :

```
print 'programme gestion des poids'
DictPoids={}
for i in range(5):
    print ' donnez le code'
    code=raw_input()
    print 'donnez le poids'
    poids=input()
    DictPoids[code]=poids
print ' Affichage des poids'
for i in DictPoids.keys():
    print 'la souris', i,
    print ' a le poids', DictPoids[i]
```

Exemple

- Demander les informations sur une souris :

```
print ' de quelle souris voulez-vous le poids ?'
code=raw_input()
if DictPoids.has_key(code):
    print 'la souris',code,
    print 'a le poids', DictPoids[code]
else:
    print "desolee cette souris n'existe pas"
```

Le type dictionnaire

- **Attention** il n'est pas possible de gérer l'ordre dans lequel le dictionnaire est rangé
- Il n'est pas possible de demander un accès par position
- **Piège** : `diction2 = DictPoids` ne crée pas une 2ième copie, il faut utiliser

```
diction2 = DictPoids.copy()
```

Ecrire une fonction

- Définition :

```
def nomfonction(listeparametres):
        liste d'instructions
```

- La définition se termine automatiquement lorsque l'indentation est supprimée.
- Appel de la fonction :

```
nomfonction(listeparametres)
```

- Si la fonction renvoie une valeur, il est possible de la récupérer dans une variable:

```
variable=nomfonction(listeparametres)
```

Exemple

```
def maximum(nb1,nb2):
    if nb1>nb2:
        return nb1
    else :
        return nb2
# Programme principal
print 'Donnez le premier nombre'
nb1=input()
print 'Donnez le deuxieme nombre'
nb2=input()
max=maximum(nb1,nb2)
print 'le maximum de ces deux nombres est',
print max
```

Exemple

- Donner la table de multiplication d'un nombre

```
print ``Donnez un nombre''
nb=input()
print 'voici la table de ', nb
for i in range (1,11):
    result=nb*i
    print nb , ''*'' ,i ,''='',result
```

Version avec Fonction

```
def multiplication(nb):
    print 'voici la table de ', nb
    for i in range (1,11):
        result=nb*i
        print nb , ''*'' ,i ,''='',result
# Programme principal
print ``Donnez un nombre''
nb=input()
multiplication(nb)
```

Exemple

```
def calculTVA(prix):
    result=prix *19.8
    return result
# Programme principal
print ``Donnez le premier prix''
prix1=input()
print ``Donnez le second prix''
prix2=input()
total = calculTVA(prix1) + calculTVA(prix2)
```

Visibilité des variables

- Une variable est locale à son bloc de définition.
- Une variable globale doit être déclarée au moyen du mot clé `global`.
- Les variables sont passées en paramètres par copie : valeur pour les types simples et adresse pour les types liste, tuples, dictionnaires.

Exemple

```
def echange(a,b):
    print 'dans la fonction echange'
    print 'a =',a
    print 'b =',b
    a,b=b,a
    print "echange fait "
    print 'a =',a
    print 'b =',b
```

Exemple

```
print 'donnez a'
a=input()
print 'donnez b'
b=input()
print 'premiere situation'
print 'valeur de a =',a
print 'valeur de b =',b
echange(a,b)
print'retour dans le programme principal'
print 'valeur de a =',a
print 'valeur de b =',b
print 'deuxieme cas'
echange(b,a)
print'retour dans le programme principal'
print 'valeur de a =',a
print 'valeur de b =',b
```

Un Plus sur les fonctions

- Donner des paramètres par défaut :

```
def demande_ok(question, tentatives=4,
               plainte='Oui ou non, svp!'):
    while 1:
        ok = raw_input(question)
        if ok in ('o', 'yes', 'oui'):
                return 1
        if ok in ('n', 'no', 'non', 'niet'):
                return 0
        tentatives = tentatives - 1
        if tentatives < 0:
                raise IOError, 'utilisateur ref
        print plainte
```

Un Plus sur les fonctions

-
- Appel de la fonction de 2 façons différentes :

```
demande_ok('Etes vous sur de
            vouloir quitter?')
ou
demande_ok('OK pour ecrasement
            du fichier?', 2)
```

‘

Un Plus sur les fonctions

- La valeur par défaut est évaluée une seule fois :

```
def f(a, l = []):
    l.append(a)
    return l
print f(1)
print f(2)
print f(3)
```

- A la fin `l` contient `[1,2,3]`

Un Plus sur les fonctions

- Utilisation de mots clés :

```
def perroquet(voltage, etat=''c'est du solide'',
                  action='voom',
                  type='Bleu Norvegien'):
  print "-- Ce perroquet ne fera pas",action,
  print "si vous le mettez sous", voltage, "Volts."
  print "-- Beau plumage, le", type
  print "-- Ca", etat, "!"
```

Un Plus sur les fonctions

- Appel de la fonction `perroquet` :

```
perroquet(1000)
perroquet(action = 'VOOOOOM', voltage = 1000000)
perroquet('un millier',
     etat = 'fait bouffer les pissenlits par la rac
perroquet('un million',
          'vous degoute de la vie', 'de bonds')
```

Appels incorrects :

```
perroquet()      # manque un argument obligatoire
perroquet(voltage=5.0, 'rend mort')
                 # un argument non-mot-cle suit un mo
perroquet(110, voltage=220)
                 # doublon de valeurs pour un argumen
perroquet(acteur='John Cleese')
                 # mot-cle inconnu
```

Un Plus sur les fonctions

- Exemple :

```
def fonc(x,y,z):
    return x+y+z
fonc(2,3,4)
```

- Création d'une focnction anonyme équivalente : `lambda`

```
f=lambda x,y,z:x+y+z
```

Un Plus sur les fonctions

- *Fabrique* de fonction

```
def fabrique_incrementeur(n):
    return lambda x, incr=n: x+incr

f=fabrique_incrementeur (3)
print f(2)
```

Les entrées - sorties

- Ouverture d'un fichier : `open(nomfichier,mode)`

  ```
  fic=open("texte","r")
  ```

  Les modes possibles sont : `"r","w","a"`
- Lecture séquentielle du fichier : `readline()`

  ```
  nom=fic.readline()
  ```
- Ecriture séquentielle dans un fichier : `write(chaine)`

  ```
  ficsortie=open("sauv","w")
  ficsortie.write("ce que je veux ecrire")
  ```

Les entrées - sorties

- Exemple d'écriture :

```
def ecrire(dict):
    fic=open("sauv","w")
    for cle in dict.keys():
        fic.write(cle)
        fic.write('\n')
        fic.write(str(dict[cle]))
        fic.write('\n')
    fic.close()
```

Les entrées - sorties

- Exemple de lecture :

```
def lire(dict):
    print "donnez un nom de fichier "
    nom=raw\_input()
    fic=open(nom,"r")
    for ligne in fic.readlines():
        if not ligne:
            break
        enz=ligne.split()
        print enz
        nom=enz[0]
        poids=string.atoi(enz[1])
        dict[nom]=poids
    fic.close()
```

Gestion des Exceptions

- Erreurs d'exécution
- Traitement grâce au type `Exception`
- Exemple :

```
while 1:
   try:
       x = int(raw_input("Veuillez entrer un nombre: "))
     break
   except ValueError:
     print "Aille! Ce n'était pas un nombre valide.
            Essayez encore..."
```

Les exceptions en Python

- Lever une exception avec l'instruction `raise`
- Exemple :

```
try:
        raise NameError('HiThere')
except NameError:
        print 'Exception levée'
else:
        print 'ok'
```

- NB : Il est possible de définir ses propres exceptions, en définissant de nouvelles classes.

Gestion des Exceptions

```
try:
  f = open('monfichier.txt')
  c = f.readline()
  i = int(string.strip(c))
except IOError, (errno, strerror):
  print "Erreur(s) E/S: c"  (errno, strerror)
except ValueError:
  print "N'a pas pu convertir la donnée en entier."
else:
  print "Erreur non prévue:", sys.exc_info()[0]
  raise
```

Les modules Python

- La variable `PYTHONPATH` détermine l'environnement de recherche des fonctions et des noms utilisés.
- Il est possible d'**importer** des fonctionalités supplémentaires afin d'étendre le langage :
- Exemple de modules fournis en général par les implémentations :

```
import sys, string
```

Qu'est-ce qu'une classe ?

- Une classe d'objet :décrit les caractéristiques générales d'une entité - c-à-d les attributs.
- Les instances de la classe : les objets qui partagent les mêmes caractéristiques.
- Les méthodes de la classe : les comportements de ces objets - c-à-d les fonctions.

Décrire des classes

- Utilisation du mot clé `class`
- Chaque méthode est déclarée avec le mot clé `def`
- Une classe désigne ses instances avec le mot clé `self`
- `variable= NomClasse()` permet d'obtenir une variable du type `NomClasse`

Les objets en Python - Syntaxe

- Définition :

```
class nomdelaclasse:
    attributs
    def fonctionClasse(self,autreparam):
        instructions de la fonction
```

Exemple - 1

```
class Point:
    x=0
    y=0
    def change_x(self,valeur):
        self.x=valeur
    def change_y(self,valeur):
        self.y=valeur
    def affiche(self):
        print 'x=',self.x , 'y=',self.y
```

Exemple - 1

```
p1=Point()
p2=Point()
p1.change_x(3)
p2.change_x(4)
p1.change_y(2)
p1.affiche()
p2.affiche()
```

Exemple de la classe Animal

```
class Animal:
    nom=''
    poids=0
# les fonctions dont le nom est
# encadré par __ ont un statut particulier.
    def __init__(self, nom):
        self.nom=nom
    def changepoids(self,valeur):
        self.poids=valeur
    def affiche(self):
        print 'je m'appelle', self.nom,
        print 'je pese', self.poids
```