

Séminaire technique Python

Pauline POMMERET

Encadrée par Olivier IFFRIG

11 décembre 2012

Petit historique

- pendant les vacances de Noël 1989, Guido VAN ROSSUM s'ennuie,
- il ne disposait pas d'un langage de programmation qui s'interfaçait correctement avec ses outils,
- il écrit une première version d'un langage de programmation,
- en fan des MONTHY PYTHON il l'appelle python.

Petit historique

- pendant les vacances de Noël 1989, Guido VAN ROSSUM s'ennuie,
- il ne disposait pas d'un langage de programmation qui s'interfaçait correctement avec ses outils,
- il écrit une première version d'un langage de programmation,
- en fan des MONTHY PYTHON il l'appelle python.

Petit historique

- pendant les vacances de Noël 1989, Guido VAN ROSSUM s'ennuie,
- il ne disposait pas d'un langage de programmation qui s'interfaçait correctement avec ses outils,
- il écrit une première version d'un langage de programmation,
- en fan des MONTHY PYTHON il l'appelle python.

Petit historique

- pendant les vacances de Noël 1989, Guido VAN ROSSUM s'ennuie,
- il ne disposait pas d'un langage de programmation qui s'interfaçait correctement avec ses outils,
- il écrit une première version d'un langage de programmation,
- en fan des MONTHY PYTHON il l'appelle python.

- langage de programmation multi-paradigme :
 - programmation impérative (séquence d'instruction donnée au fur et à mesure)
 - programmation orientée objet (on verra plus tard)
- typage dynamique (typage au besoin à l'exécution du programme : type de l'objet peut être celui des classes dont il hérite, ça veut dire qu'on a le droit de comparer des chaînes de caractères et des entiers)
- système de gestion d'exception.

- langage de programmation multi-paradigme :
 - programmation impérative (séquence d'instruction donnée au fur et à mesure)
 - programmation orientée objet (on verra plus tard)
- typage dynamique (typage au besoin à l'exécution du programme : type de l'objet peut être celui des classes dont il hérite, ça veut dire qu'on a le droit de comparer des chaînes de caractères et des entiers)
- système de gestion d'exception.

- langage de programmation multi-paradigme :
 - programmation impérative (séquence d'instruction donnée au fur et à mesure)
 - programmation orientée objet (on verra plus tard)
- typage dynamique (typage au besoin à l'exécution du programme : type de l'objet peut être celui des classes dont il hérite, ça veut dire qu'on a le droit de comparer des chaînes de caractères et des entiers)
- système de gestion d'exception.

- langage de programmation multi-paradigme :
 - programmation impérative (séquence d'instruction donnée au fur et à mesure)
 - programmation orientée objet (on verra plus tard)
- typage dynamique (typage au besoin à l'exécution du programme : type de l'objet peut être celui des classes dont il hérite, ça veut dire qu'on a le droit de comparer des chaînes de caractères et des entiers)
- système de gestion d'exception.

- langage de programmation multi-paradigme :
 - programmation impérative (séquence d'instruction donnée au fur et à mesure)
 - programmation orientée objet (on verra plus tard)
- typage dynamique (typage au besoin à l'exécution du programme : type de l'objet peut être celui des classes dont il hérite, ça veut dire qu'on a le droit de comparer des chaînes de caractères et des entiers)
- système de gestion d'exception.

Pourquoi utiliser python ?

- parce que les vieux l'ont décidé,
- parce que c'est pas trop enquiainant à apprendre,
- parce que c'est pas trop enquiainant à écrire.

Pourquoi utiliser python ?

- parce que les vieux l'ont décidé,
- parce que c'est pas trop enquiquinant à apprendre,
- parce que c'est pas trop enquiquinant à écrire.

Pourquoi utiliser python ?

- parce que les vieux l'ont décidé,
- parce que c'est pas trop enquiquinant à apprendre,
- parce que c'est pas trop enquiquinant à écrire.

Comment écrire

- python en mode « interactif »
 - avec `python`, `ipython`,
 - avec `Python` command line sous windows
- python dans un script
 - `python programme.py`
 - `./programme.py`, s'il commence par `#!/usr/bin/env python` et qu'il est executable.

Comment écrire

- python en mode « interactif »
 - avec `python`, `ipython`,
 - avec `Python` command line sous windows
- python dans un script
 - `python programme.py`
 - `./programme.py`, s'il commence par `#!/usr/bin/env python` et qu'il est exécutable.

Comment écrire

- python en mode « interactif »
 - avec `python`, `ipython`,
 - avec Python command line **sous windows**
- python dans un script
 - `python programme.py`
 - `./programme.py`, s'il commence par `#!/usr/bin/env python` et qu'il est exécutable.

Comment écrire

- python en mode « interactif »
 - avec `python`, `ipython`,
 - avec Python command line sous windows
- python dans un script
 - `python programme.py`
 - `./programme.py`, s'il commence par `#!/usr/bin/env python` et qu'il est executable.

Comment écrire

- python en mode « interactif »
 - avec `python`, `ipython`,
 - avec Python command line sous windows
- python dans un script
 - `python programme.py`
 - `./programme.py`, s'il commence par `#!/usr/bin/env python` et qu'il est executable.

Comment écrire

- python en mode « interactif »
 - avec `python`, `ipython`,
 - avec Python command line sous windows
- python dans un script
 - `python programme.py`
 - `./programme.py`, s'il commence par `#!/usr/bin/env python` et qu'il est executable.

Pour l'encodage

Tester le script suivant :

```
print "é è à à à à à à à à à à"
```

On l'exécute et :

```
In [2]: run /usr/_local/bin/python.py  
File "/usr/bin/python.py", line 1  
SyntaxError: non-ASCII character 'é' in file /usr/bin/python.py on  
line 1, but no encoding declared: see http://www.python.org/doc/2.6.3.html  
for details
```

Comment faire pour que python soit gentil

Il suffit d'ajouter :

```
#-*- coding:utf8 -*-
```

Et on obtient ça :



```
#!/usr/bin/env python  
#-*- coding:utf8 -*-
```

Les différents types

Il existe plusieurs types sous python :

- types numériques :

- `int`, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
- `long`, entier arbitrairement long,
- `float`, nombre à virgule ;
- `complex`,

- chaînes de caractères :

- `str`, chaînes de caractère,
- `unicode`, chaîne de caractère unicode.

- `tuple`, liste de longueur de fixe,

- `list`, liste de longueur variable,

- `dict`, dictionnaire,

- `file`, fichier,

- `bool`, booléen.

Les chaînes de caractères

Travailler avec les fichiers

Créer ses propres jouets : définir une fonction

Les modules

Les différents types

Il existe plusieurs types sous python :

- **types numériques :**
 - `int`, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
 - `long`, entier arbitrairement long,
 - `float`, nombre à virgule ;
 - `complex`,
- **chaînes de caractères :**
 - `str`, chaînes de caractère,
 - `unicode`, chaîne de caractère unicode.
- `tuple`, liste de longueur de fixe,
- `list`, liste de longueur variable,
- `dict`, dictionnaire,
- `file`, fichier,
- `bool`, booléen.

Les chaînes de caractères
Travailler avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Les différents types

Il existe plusieurs types sous python :

- **types numériques :**

- `int`, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
- `long`, entier arbitrairement long,
- `float`, nombre à virgule ;
- `complex`,

- **chaînes de caractères :**

- `str`, chaînes de caractère,
- `unicode`, chaîne de caractère unicode.

- `tuple`, liste de longueur de fixe,
- `list`, liste de longueur variable,
- `dict`, dictionnaire,
- `file`, fichier,
- `bool`, booléen.

Les chaînes de caractères

Travailler avec les fichiers

Créer ses propres jouets : définir une fonction

Les modules

Les différents types

Il existe plusieurs types sous python :

- **types numériques :**
 - `int`, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
 - `long`, entier arbitrairement long,
 - `float`, nombre à virgule ;
 - `complex`,
- **chaînes de caractères :**
 - `str`, chaînes de caractère,
 - `unicode`, chaîne de caractère unicode.
- `tuple`, liste de longueur de fixe,
- `list`, liste de longueur variable,
- `dict`, dictionnaire,
- `file`, fichier,
- `bool`, booléen.

Les chaînes de caractères
Travailler avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Les différents types

Il existe plusieurs types sous python :

- **types numériques :**

- `int`, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
- `long`, entier arbitrairement long,
- `float`, nombre à virgule ;
- `complex`,

- **chaînes de caractères :**

- `str`, chaînes de caractère,
- `unicode`, chaîne de caractère unicode.

- `tuple`, liste de longueur de fixe,
- `list`, liste de longueur variable,
- `dict`, dictionnaire,
- `file`, fichier,
- `bool`, booléen.

Les chaînes de caractères

Travailler avec les fichiers

Créer ses propres jouets : définir une fonction

Les modules

Les différents types

Il existe plusieurs types sous python :

- **types numériques :**

- `int`, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
- `long`, entier arbitrairement long,
- `float`, nombre à virgule ;
- `complex`,

- **chaînes de caractères :**

- `str`, chaînes de caractère,
- `unicode`, chaîne de caractère unicode.

- `tuple`, liste de longueur de fixe,

- `list`, liste de longueur variable,

- `dict`, dictionnaire,

- `file`, fichier,

- `bool`, booléen.

Les chaînes de caractères

Travailler avec les fichiers

Créer ses propres jouets : définir une fonction

Les modules

Les différents types

Il existe plusieurs types sous python :

- **types numériques :**
 - `int`, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
 - `long`, entier arbitrairement long,
 - `float`, nombre à virgule ;
 - `complex`,
- **chaînes de caractères :**
 - `str`, chaînes de caractère,
 - `unicode`, chaîne de caractère unicode.
- `tuple`, liste de longueur de fixe,
- `list`, liste de longueur variable,
- `dict`, dictionnaire,
- `file`, fichier,
- `bool`, booléen.

Les chaînes de caractères
Travailler avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Les différents types

Il existe plusieurs types sous python :

- **types numériques :**
 - `int`, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
 - `long`, entier arbitrairement long,
 - `float`, nombre à virgule ;
 - `complex`,
- **chaînes de caractères :**
 - `str`, chaînes de caractère,
 - `unicode`, chaîne de caractère unicode.
- `tuple`, liste de longueur de fixe,
- `list`, liste de longueur variable,
- `dict`, dictionnaire,
- `file`, fichier,
- `bool`, booléen.

Les chaînes de caractères
Travailler avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Les différents types

Il existe plusieurs types sous python :

- types numériques :
 - int, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
 - long, entier arbitrairement long,
 - float, nombre à virgule ;
 - complex,
- chaînes de caractères :
 - str, chaînes de caractère,
 - unicode, chaîne de caractère unicode.
- tuple, liste de longueur de fixe,
- list, liste de longueur variable,
- dict, dictionnaire,
- file, fichier,
- bool, booléen.

Les chaînes de caractères
Travailler avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Les différents types

Il existe plusieurs types sous python :

- types numériques :

- `int`, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
- `long`, entier arbitrairement long,
- `float`, nombre à virgule ;
- `complex`,

- chaînes de caractères :

- `str`, chaînes de caractère,
- `unicode`, chaîne de caractère unicode.

- `tuple`, liste de longueur de fixe,

- `list`, liste de longueur variable,

- `dict`, dictionnaire,

- `file`, fichier,

- `bool`, booléen.

Les chaînes de caractères

Travailler avec les fichiers

Créer ses propres jouets : définir une fonction

Les modules

Les différents types

Il existe plusieurs types sous python :

- types numériques :
 - int, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
 - long, entier arbitrairement long,
 - float, nombre à virgule ;
 - complex,
- chaînes de caractères :
 - str, chaînes de caractère,
 - unicode, chaîne de caractère unicode.
- tuple, liste de longueur de fixe,
- list, liste de longueur variable,
- dict, dictionnaire,
- file, fichier,
- bool, booléen.

Les chaînes de caractères
Travailler avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Les différents types

Il existe plusieurs types sous python :

- types numériques :
 - `int`, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
 - `long`, entier arbitrairement long,
 - `float`, nombre à virgule ;
 - `complex`,
- chaînes de caractères :
 - `str`, chaînes de caractère,
 - `unicode`, chaîne de caractère unicode.
- `tuple`, liste de longueur de fixe,
- `list`, liste de longueur variable,
- `dict`, dictionnaire,
- `file`, fichier,
- `bool`, booléen.

Les chaînes de caractères
Travailler avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Les différents types

Il existe plusieurs types sous python :

- types numériques :
 - int, entier entre $-2^{31} + 1$ et $2^{31} - 1$,
 - long, entier arbitrairement long,
 - float, nombre à virgule ;
 - complex,
- chaînes de caractères :
 - str, chaînes de caractère,
 - unicode, chaîne de caractère unicode.
- tuple, liste de longueur de fixe,
- list, liste de longueur variable,
- dict, dictionnaire,
- file, fichier,
- bool, booléen.

Les chaînes de caractères
Travailler avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Obtenir le type d'un objet

Pour cela on utilise `type`.

- `type(1.34)` donne `float`
- `type("1.34")` donne `str`
- `type("plop")` donne `str`
- `type(1337)` donne `int`
- ...

Obtenir le type d'un objet

Pour cela on utilise `type`.

- `type(1.34)` **donne** `float`
- `type("1.34")` **donne** `str`
- `type("plop")` **donne** `str`
- `type(1337)` **donne** `int`
- ...

Obtenir le type d'un objet

Pour cela on utilise `type`.

- `type(1.34)` **donne** `float`
- `type("1.34")` **donne** `str`
- `type("plop")` **donne** `str`
- `type(1337)` **donne** `int`
- ...

Obtenir le type d'un objet

Pour cela on utilise `type`.

- `type(1.34)` donne `float`
- `type("1.34")` donne `str`
- `type("plop")` donne `str`
- `type(1337)` donne `int`
- ...

Obtenir le type d'un objet

Pour cela on utilise `type`.

- `type(1.34)` donne `float`
- `type("1.34")` donne `str`
- `type("plop")` donne `str`
- `type(1337)` donne `int`
- ...

Qu'est ce que c'est ?

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

définition

Une liste est un objet pouvant contenir d'autres objets. Ce sont des séquences d'objets.

Comment créer une liste :

- `maliste = list()` crée une liste vide,
- `maliste = []` crée aussi une liste vide,
- `maliste = [0, 2.5, "plop", [1,2]]` crée une liste non-vide.

Accéder aux éléments d'une liste

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Accéder aux éléments d'une liste :

- `maliste[0]` donne le premier élément d'une liste, ici 0
- `maliste[n]` donne le *nième* élément d'une liste,
- `maliste[-1]` donne le dernier élément d'une liste, ici
[1, 2]

Accéder aux éléments d'une liste

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Accéder aux éléments d'une liste :

- `maliste[0]` donne le premier élément d'une liste, ici 0
- `maliste[n]` donne le *nième* élément d'une liste,
- `maliste[-1]` donne le dernier élément d'une liste, ici
[1, 2]

Accéder aux éléments d'une liste

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Accéder aux éléments d'une liste :

- `maliste[0]` donne le premier élément d'une liste, ici 0
- `maliste[n]` donne le *nième* élément d'une liste,
- `maliste[-1]` donne le dernier élément d'une liste, ici `[1, 2]`

Modifier les éléments d'une liste

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

- `maliste[1]=2.2` réaffecte au deuxième élément de `maliste` la valeur `2.2`
- `maliste.append(42)` ajoute `42` à la fin de la liste,
- `maliste.insert(3, "bidule")` insère `"bidule"` à la position `3`.

Modifier les éléments d'une liste

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

- `maliste[1]=2.2` réaffecte au deuxième élément de `maliste` la valeur `2.2`
- `maliste.append(42)` ajoute `42` à la fin de la liste,
- `maliste.insert(3, "bidule")` insère `"bidule"` à la position `3`.

Modifier les éléments d'une liste

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

- `maliste[1]=2.2` **réaffecte** au deuxième élément de `maliste` **la valeur 2.2**
- `maliste.append(42)` **ajoute** 42 à la fin de la liste,
- `maliste.insert(3, "bidule")` **insère** "bidule" à la **position 3**.

Concaténer les listes

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Pour concaténer des listes :

- `liste1.extend(liste2)`
- `liste1 += liste2`

Concaténer les listes

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Pour concaténer des listes :

- `liste1.extend(liste2)`
- `liste1 += liste2`

Suppression d'éléments d'une liste

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

- `del maliste[2]` retire le 3ème élément de la liste
- `maliste.remove("bidule")` retire l'élément bidule de la liste.

Suppression d'éléments d'une liste

- `del maliste[2]` retire le 3ème élément de la liste
- `maliste.remove("bidule")` retire l'élément bidule de la liste.

Énumérer les éléments d'une liste

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

- avec une boucle `while`
- avec une boucle `for`
- avec `enumerate` qui prend en paramètre une liste et sort les couples (indices, éléments).

```
for i, elt in enumerate(maliste):  
    print("À l'indice {} se trouve l'élément  
{})".format(i, elt))
```

Énumérer les éléments d'une liste

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

- avec une boucle `while`
- avec une boucle `for`
- avec `enumerate` qui prend en paramètre une liste et sort les couples (indices, éléments).

```
for i, elt in enumerate(maliste):  
    print("À l'indice {} se trouve l'élément  
{})".format(i, elt))
```

Énumérer les éléments d'une liste

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

- avec une boucle `while`
- avec une boucle `for`
- avec `enumerate` qui prend en paramètre une liste et sort les couples (indices, éléments).

```
for i, elt in enumerate(maliste):  
    print("À l'indice {} se trouve l'élément  
{})".format(i, elt))
```

Qu'est ce que c'est

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

définition

Les tuples sont des listes immuables, que l'on ne peut modifier.

En fait, on les utilisait déjà grâce à la fonction `enumerate`.
On peut les définir ainsi :

- `tuplevide=()`
- `tuplenonvide=(1,)`
- `autretuple=(1,2,3,4,5,6)`

Qu'est ce que c'est ?

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

définition

Le dictionnaire est un objet conteneur non ordonné. Pour accéder aux objets contenus dans le dictionnaire, on utilise des clés.

Voici comment créer un dictionnaire :

- `mondico=dict()` crée un dictionnaire vide,
- `mondico={}` crée un dictionnaire vide,

Qu'est ce que c'est ?

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

définition

Le dictionnaire est un objet conteneur non ordonné. Pour accéder aux objets contenus dans le dictionnaire, on utilise des clés.

Voici comment créer un dictionnaire :

- `mondico=dict()` crée un dictionnaire vide,
- `mondico={}` crée un dictionnaire vide,

Remplir et regarder le contenu d'un dictionnaire

Pour remplir un dictionnaire, on s'y prend de la façon suivante :

- `mondico["plop"] = 42`

Pour regarder ce qu'il y a dans un dictionnaire :

- `mondico["plop"]` affiche la valeur associée à la clé "plop".

Utiliser `.get()`

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

- `mondico.get(5)` affiche la valeur associée à la clé 5
- `mondico.get(5, "raté")` affiche la valeur associée à la clé 5 ou "raté" si elle n'existe pas.

Utiliser `.get()`

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

- `mondico.get(5)` affiche la valeur associée à la clé 5
- `mondico.get(5, "raté")` affiche la valeur associée à la clé 5 ou "raté" si elle n'existe pas.

Supprimer des clés d'un dictionnaire

On peut utiliser 2 méthodes :

- `del mondico["plop"]` supprime la clé "plop" et sa valeur associée
- la méthode `pop` : `mondico.pop("plip")` qui supprime la clé "plip" et sa valeur associée, 24 et donne en sortie 24, la valeur supprimée.

Supprimer des clés d'un dictionnaire

On peut utiliser 2 méthodes :

- `del mondico["plop"]` supprime la clé "plop" et sa valeur associée
- la méthode `pop` : `mondico.pop("plip")` qui supprime la clé "plip" et sa valeur associée, 24 et donne en sortie 24, la valeur supprimée.

Différents types d'instruction

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

- les tests conditionnels pour faire des tests,
- les boucles `for` et `while`.

Forme minimale

La forme minimale d'un test conditionnel se présente sous la forme :

```
if condition:  
    instruction
```

Par exemple :

```
a=2  
if a<10:  
    print "a est plus petit que 10"  
a est plus petit que 10
```

Forme complète

C'est pas très pratique si l'on veut tester plusieurs conditions.
C'est pourquoi on utilise `else`.

```
age=20
if age>=18:
    print "félicitation, tu es
majeur-e"
else:
    print "tu es mineur-e"
félicitation, tu es majeur-e
```


elif

Oui, mais si on veut faire plus de distinctions ?

```
a=0
if a>0:
    print "a est positif"
elif a<0:
    print "a est négatif"
else:
    print "a est nul"
a est nul
```

Opérateurs de comparaison

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Opérateur	Signification
<	strictement inférieur à
>	strictement supérieur à
<=	inférieur ou égal à
>=	supérieur ou égal à
==	égal à
!=	différent de

and, or et not

Pour savoir si `a` est compris entre 5 et 10 :

```
a=2
if a>=5:
    if a<=10:
        print "dans la fenêtre"
    else:
        print "pas dans la fenêtre"
else:
    print "pas dans la fenêtre"
pas dans la fenêtre
```

C'est lourd.

and, or et not

Avec `and` et `or` ça devient plus facile à écrire :

```
if a >= 5 and a <= 10:  
    print "in"  
else:  
    print "out"
```

```
if a < 5 or a > 10:  
    print "out"  
else:  
    print "in"
```

and, or et not

not permet de rendre ce que l'on écrit plus clair :

```
a, heureux=12, False
if a<10:
    if heureux is not False:
        print "c'est bizarre"
    else:
        print "heureux =",heureux
else:
    if heureux is False:
        print "étrange"
    else:
        print "heureux =",heureux
étrange
```

A quoi sert-elle ?

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

La boucle `while`, c'est pour les fainéants.

définition

Elle sert à répéter un bloc d'instruction tant qu'une condition est vraie.

```
i=0
n=8
while i<10:
    print i+1, "x", n, "=", (i+1)*n
    i+=1
```

A quoi sert-elle ?

Les chaînes de caractères
Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

définition

`for` permet de parcourir des séquences variées (des chaînes de caractères, des listes)/

```
sequence="AATCGC"  
for element in sequence:  
    print element
```

`element` est « instancié » tout seul en devenant successivement toutes les valeurs de `sequence`, pas besoin de lui dire !

quote, double-quote, triple-quote

On a mis les chaînes de caractères entre guillemets. On peut définir une chaîne de caractères de plein de façons différentes :

- avec des guillemets, "entre guillemets, comme ça j'ai des apostrophes"
- avec des apostrophes, 'il dit utilise des apostrophes pour pouvoir citer X quand il dit "plop"'
- avec l'un ou l'autre, en échappant ce qu'il faut, "j'ai dit plop"

quote, double-quote, triple-quote

On a mis les chaînes de caractères entre guillemets. On peut définir une chaîne de caractères de plein de façons différentes :

- avec des guillemets, "entre guillemets, comme ça j'ai des apostrophes"
- avec des apostrophes, 'il dit utilise des apostrophes pour pouvoir citer X quand il dit "plop"'
- avec l'un ou l'autre, en échappant ce qu'il faut, "j'ai dit plop"

quote, double-quote, triple-quote

On a mis les chaînes de caractères entre guillemets. On peut définir une chaîne de caractères de plein de façons différentes :

- avec des guillemets, "entre guillemets, comme ça j'ai des apostrophes"
- avec des apostrophes, 'il dit utilise des apostrophes pour pouvoir citer X quand il dit "plop"'
- avec l'un ou l'autre, en échappant ce qu'il faut, "j'ai dit plop"

Des histoires de casse

Parfois on a envie de modifier la casse d'une chaîne de caractères :

- pour tout mettre en minuscule, `chaine.lower()`
- pour tout mettre en majuscule, `chaine.upper()`
- pour mettre la première en majuscule et le reste en minuscule, `chaine.capitalize()`
- pour mettre une majuscule à chaque mot, `chaine.title()`

Des histoires de casse

Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Parfois on a envie de modifier la casse d'une chaîne de caractères :

- pour tout mettre en minuscule, `chaine.lower()`
- pour tout mettre en majuscule, `chaine.upper()`
- pour mettre la première en majuscule et le reste en minuscule, `chaine.capitalize()`
- pour mettre une majuscule à chaque mot, `chaine.title()`

Des histoires de casse

Parfois on a envie de modifier la casse d'une chaîne de caractères :

- pour tout mettre en minuscule, `chaine.lower()`
- pour tout mettre en majuscule, `chaine.upper()`
- pour mettre la première en majuscule et le reste en minuscule, `chaine.capitalize()`
- pour mettre une majuscule à chaque mot, `chaine.title()`

Des histoires de casse

Parfois on a envie de modifier la casse d'une chaîne de caractères :

- pour tout mettre en minuscule, `chaine.lower()`
- pour tout mettre en majuscule, `chaine.upper()`
- pour mettre la première en majuscule et le reste en minuscule, `chaine.capitalize()`
- pour mettre une majuscule à chaque mot, `chaine.title()`

Formatage de chaînes

Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Pour cela il y a deux méthodes :

- `%s`
- `.format()`



Voici un exemple d'application :

```
"|%s|a%s|plop|%-6s|%12s|"%( "plip", "AAA", 13,  
2)
```

qui donne :

```
'|plip|aAAA|plop|13 | 2|'
```


.format

Jouer avec les fichiers
Créer ses propres jouets : définir une fonction
Les modules

Voici un exemple d'application :

```
for i, elt in enumerate(maliste):  
    print "À l'indice {} se trouve  
{}}.".format(i, elt)
```

À l'indice 0 se trouve 0.

À l'indice 1 se trouve 2.5.

À l'indice 2 se trouve [1, 2].

Ouvrir, fermer

Les chaînes de caractères

Pour ouvrir un fichier :

Créer ses propres jouets : définir une fonction
Les modules

```
f=open(monfichier, "mode")
```

où mode vaut :

- r pour lire,
- w pour écrire,
- a pour écrire à la fin.

Et pour le fermer :

```
f.close()
```

Lire et écrire

Les chaînes de caractères

Créer ses propres jouets : définir une fonction
Les modules

Pour lire :

- tout, `f.read()`
- une ligne, `f.readline()`
- la liste des lignes,
`f.readlines()`

Pour écrire :

- `f.write("mon texte")`

Gérer sa position

Les chaînes de caractères

Créer ses propres jouets : définir une fonction
Les modules

Dans un fichier :

- `f.tell()` donne la position,
- `f.seek(pos)` permet de se rendre à la position,
- `for line in f:` permet de parcourir le fichier.

Comment définir une fonction

Parfois, on réutilise le même bloc d'instructions à plusieurs reprises, et il serait plus simple de pouvoir l'appeler facilement. C'est pour cela que l'on définit des fonctions.

```
def nomdefonction(param1, param2,  
param3) :  
    instruction1  
    instruction2  
    instruction3  
    instruction4  
    instruction5  
    instruction6
```

Exemple

Les chaînes de caractères
Jouer avec les fichiers

Les modules

Faisons une fonction qui affiche les tables de multiplication :

```
def multiplication(n):  
    i=0  
    while i<10:  
        print i+1, "x", n, "=", (i+1)*n  
        i+=1
```

Valeur par défaut

Les chaînes de caractères
Jouer avec les fichiers

Les modules

On peut définir des valeurs pas défauts ainsi :

```
def multiplication(n=8):  
    i=0  
    while i<10:  
        print i+1, "x", n, "=", (i+1)*n  
        i+=1
```

Taper `multiplication()`.

L'instruction `return`

L'instruction `return` permet de faire en sorte que la fonction retourne quelque chose, qui puisse être capturée par une variable.

Exemple :

```
def bissextile(n):  
    if n%400==0:  
        return True  
    elif n%100==0:  
        return False  
    elif n%4==0:  
        return True  
    return False
```


Qu'est ce que c'est ?

Un module est un ensemble de fonctions et de variables rangé dans un fichier que l'on peut appeler pour pouvoir les utiliser.
Pour les appeler :

- `import module`, et on utilise les fonctions du module `math.fonction()`
- `from module import fonction` et on utilise alors `fonction` qui peut écraser la fonction « `fonction` » préexistante
- `from module import *` pour importer toutes les fonctions du module, avec les risques encourus...

Quelques modules

Voici quelques modules rapidement utiles :

- math
- os
- sys
- numpy
- biotools
- ldap
- optparse
- matplotlib.pyplot
- ...

La programmation orientée objet

- On a écrit beaucoup de code
- On a parfois dû surcharger des parties pour prendre en compte de nouveaux cas
- C'est le fouillis dans nos fichiers
- On souhaite regrouper les choses par thème

La programmation orientée objet

- On a écrit beaucoup de code
- On a parfois dû surcharger des parties pour prendre en compte de nouveaux cas
- C'est le fouillis dans nos fichiers
- On souhaite regrouper les choses par thème

La programmation orientée objet

- On a écrit beaucoup de code
- On a parfois dû surcharger des parties pour prendre en compte de nouveaux cas
- C'est le fouillis dans nos fichiers
- On souhaite regrouper les choses par thème

La programmation orientée objet

- On a écrit beaucoup de code
- On a parfois dû surcharger des parties pour prendre en compte de nouveaux cas
- C'est le fouillis dans nos fichiers
- On souhaite regrouper les choses par thème

Concept

- Une classe est la représentation abstraite d'un objet.
- Ce qui concerne l'objet (variables, fonctions) y est regroupé sous forme d'attributs et méthodes.
- Toutes ces données sont désormais dépendantes de l'objet.

Syntaxe

```
Class vecteur:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
  
    def norme():  
        return math.sqrt(self.x**2 + self.y**2)
```


Additionner 2 vecteurs

```
class vecteur:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def norme(self):
        return math.sqrt(self.x**2+ self.y**2)
    def __add__(self, v):
        return vecteur(self.x+v.x, self.y+v.y)
```

Héritage de classe

```
class vecteur3(vecteur):  
    def __init__(self, x, y, z):  
        self.x = x  
        self.y = y  
        self.z = z  
    def norme(self):  
        return  
        math.sqrt(self.x**2+self.y**2+self.z**2)
```