



Introduction au Python

Mickaël BOLNET



Objectifs



- Maîtriser la syntaxe du langage Python
- Acquérir les notions essentielles de la programmation objet
- Connaître et mettre en œuvre les différents modules Python
- Concevoir des interfaces graphiques
- Mettre en œuvre les outils de test et d'évaluation de la qualité d'un programme Python

Prérequis : connaissance de base en programmation



Organisation



- 1^{er} Jour : Syntaxe et élément de base en Python
- 2^e Jour : Programmation Orientée Objet (POO)
- 3^e Jour : POO suite + IHM
- 4^e Jour : Un peu de Web (Base de donnée / Flask)
- 5^e Jour : Qualité - Interface Python/C



Historique



- **HISTORIQUE**
- Créé en 1989 par Guido van Rossum
- 1991 : première version publique (0.9.0)
- 2001 : Fondation Python
- 2008 : Python 3
- 2005 : Guido Van Rossum rejoint Google
- 2012 : Guido Van Rossum rejoint Dropbox



Introduction au Python

Mickaël BOLNET



Instagram

Qu'est-ce que Python ?

- Open Source
- Langage interprété
- Multiplate-formes
- Multi-paradigmes
- Haut niveau
- 2 fois « programming language of the year » TIOBE (2007 et 2010)



Particularité

Python 2.7 ou Python 3 ?



Premiers pas

Pour s'entraîner CodeWars ou HackerRank

Installation

- Python
- Pip
- Jupyter
- Anaconda
- PyCharm / Sublime Text





Premier programme

```
print('hello world!')
```



Ajouter des commentaires

```
# ceci est un commentaire  
print('hello world!')  
""" Ceci  
est un commentaire  
sur plusieurs ligne  
-> une docstring  
"""
```



Les variables

```
a = 10
```

```
b = 11
```

```
print(a)
```

```
print('a')
```

```
print(b)
```

```
print(a + b)
```



Convention de nommage

```
ma_variable = 4
```

```
MA_CONSTANTE = 'toto'
```

```
class MaClass:  
    pass
```



Les types

- Int (1, 2, 3 ...)
 - Float (1.2, 3.14 ...)
 - Complex (1+2i, 2+2i ...)
 - Bool (True or False)
-
- Chaine de caractères ('10', "10")
 - List : ie [1,2,3,4]
 - Tuple : ie (2,2)
 - Dictionary : {'nom' : 'BOLNET', 'prenom' : 'Mickael'}



Opérations

$x + y$	Addition
$x - y$	Soustraction
$x * y$	Multiplication
x / y	Division
$x // y$	Division entière
$x \% y$	Reste
$-x$	Opposé
$+x$	
$x ** y$	Puissance

Opérateurs binaires

$x \mid y$	Ou binaire
$x \wedge y$	Ou exclusif
$x \& y$	Et binaire
$x \ll y$	Décalage à gauche
$x \gg y$	Décalage à droite
$\sim x$	Inversion



Opérateurs sur les séquences

x not in s	False si s contient x, sinon True
s1 + s2	Concaténation
s * n	Répétition
s[i]	Élément à l'indice ou clef i
len(s)	Taille de la chaîne
min(s)	Plus petit élément de la séquence
max(s)	Plus grand élément de la séquence
s.index(x)	Indice de la première occurrence de x
s.count(x)	Nombre total d'occurrences de x

Les séquences

- Accès à un caractère

```
ma_chaine = "Bonjour"
```

```
print(ma_chaine[0])
```

```
print(ma_chaine[1])
```

- Modification

```
mon_tableau = [3, 5, '1', False]
```

```
mon_tableau[0] = "D"
```

ATTENTION: Les chaînes de caractères et les tuples ne sont pas modifiables

Les séquences

Slicing

```
mon_tableau = [3, 5, '1', False]
print(mon_tableau[0:2])
print(mon_tableau[:2])
print(mon_tableau[2:])
print(mon_tableau[2:-1])
print(mon_tableau[0:4:2])
print(mon_tableau[::-1])
print(mon_tableau[4:0:-1])
```

Interactions et affichage

```
name = input('Quel est votre nom ? ')\nage = int(input("quel est votre âge ? "))
```

```
"Ma variable : %type" % var
```

```
"Mes variables : %type, %type" % (var1, var2)
```

```
"Resultat : %(val)type %(unit)type" % {'val':var1, 'unit':var2}
```

type est d : entier - f : flottant - s : chaîne de caractère - c : caractère - o : octal - x : hexadécimal - C : caractère

Précision pour les float :

- "Resultat: %.2f" % 3.141592653589793

Avec format

- Syntaxe : `string.format(*args)`
- `"Résultat : {}".format(var)`
- `"Résultat : {}, {}".format(var1, var2)`
- `"Résultat : {1} {0}".format(var1, var2)`
- `"Résultat : {value} {unit}".format(unit=var1, value=var2)`
- `"Résultat : {:5.2f}".format(var)`
- `"Résultat : {value:5.2f} {unit}".format(unit=var1, value=var2)`

Les expressions régulières

```
import re
```

```
print(re.match("GR(.)?S", "GRIS"))
```

```
print(re.findall("([0-9]+)", "Bonjour 111 Aurevoir 222"))
```

```
print(re.sub("([0-9]+)", "Bonjour ", "Bonjour 111 Aurevoir 222"))
```

Les expressions régulières : symboles

- . Le point correspond à n'importe quel caractère.
- ^ Indique un commencement de segment mais signifie aussi "contraire de"
- \$ Fin de segment
- [xy] Une liste de segment possible. Exemple [abc] équivaut à : a, b ou c
- (x|y) Indique un choix multiple type (ps|ump) équivaut à "ps" OU "UMP"
- \d le segment est composé uniquement de chiffre, ce qui équivaut à [0-9].
- \D le segment n'est pas composé de chiffre, ce qui équivaut à [^0-9].
- \s Un espace, ce qui équivaut à [\t\n\r\f\v].
- \S Pas d'espace, ce qui équivaut à [^ \t\n\r\f\v].
- \w Présence alphanumérique, ce qui équivaut à [a-zA-Z0-9_].
- \W Pas de présence alphanumérique [^a-zA-Z0-9_].
- \ Est un caractère d'échappement

Les expressions régulières : répétition

$Z\{3\}$: la lettre Z (en majuscule) se répète 3 fois consécutives.

$(AZ)\{1,6\}$: le segment AZ se répète de 1 à 6 fois consécutives.

$(ARZ)\{,7\}$: le segment ARZ ne soit pas présent du tout ou présent jusqu'à 7 fois consécutives.

$(TO)\{1,\}$: le segment TO soit présent au moins une fois.

$?$: 0 ou 1 fois

$+$: Au moins une fois

$*$: 0, 1 ou autant de fois qu'on le trouve

Echappement

$Z\{3\}$: la lettre Z (en majuscule) se répète 3 fois consécutives.

$(AZ)\{1,6\}$: le segment AZ se répète de 1 à 6 fois consécutives.

$(ARZ)\{,7\}$: le segment ARZ ne soit pas présent du tout ou présent jusqu'à 7 fois consécutives.

$(TO)\{1,\}$: le segment TO soit présent au moins une fois.

$?$: 0 ou 1 fois

$+$: Au moins une fois

$*$: 0, 1 ou autant de fois qu'on le trouve

Les expressions régulières : compilation

```
regex = re.compile(r"GR(.)?S")  
regex.match("GRIS")
```

Augmente les performances!

Les conditions

Structure conditionnelle

```
name = 'Mickael'
if name == 'Mickael':
    print('Bonjour Mickael')
elif name == 'Laetitia':
    print('Bonjour Laetitia')
else:
    print('Vous n\'avez pas le droit de rentrer')
```

Attention à l'indentation des BLOCS !

Opérateurs de comparaison

<	Strictement inférieur à
>	Strictement supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égal à
!=	Différent de

Logique booléenne

- OR
- NOT
- AND

BONUS : opérateur ternaire

```
gender = 'masculin' if name == 'Mickael' else 'feminin'
```

Les boucles

Les boucles (while)

```
nb = 7
```

```
i = 0
```

```
while i < 10:
```

```
    print(i + 1, "*", nb, "=", (i + 1) * nb)
```

```
    i += 1
```

Les boucles (for)

```
for i in range(5):  
    print(i)
```

```
for i in range(3, 6):  
    print(i)
```

```
for i in range(4, 10, 2):  
    print(i)
```

```
for i in range(0, -10, -2):  
    print(i)
```

Comprehensive list

```
circ = [i*2 for i in range(20)]  
x = [math.cos(i) for i in circ]
```

Break and continue

```
while 1:
    lettre = input("Tapez 'Q' pour quitter : ")
    if lettre == "Q":
        print("Fin de la boucle")
        break
    elif lettre == "N":
        print("Vous avez tapé N")
        continue
```

Les fonctions



Les fonctions (définition)

```
def dire_bonjour():  
    print('Bonjour Monsieur!')
```

```
def dire_bonjour(name):  
    print('bonjour ' + name)
```

```
def dire_bonjour(name, name2="", name3='toto'):  
    print('bonjour ' + name + ' ' + name2)
```



Les fonctions (appel)

```
dire_bonjour() #'Bonjour Monsieur!'
```

```
dire_bonjour('toto') #bonjour toto
```

```
def dire_bonjour(name, name2="", name3='toto'):  
    print('bonjour ' + name + ' ' + name2)
```

Portée des variables

```
foo = 1
def test_local():
    foo = 2 # new local foo
def test_global():
    global foo
    foo = 3 # changes the value of the global foo
```


Documentation des fonctions

```
def ajouter(a, b):  
    """  
    Ajoute deux nombres l'un à l'autre et retourne  
    le résultat.  
    """  
    return a + b  
  
help(ajouter)
```

Fonctions lambda

```
def print_result(var, function):  
    print(function(var))
```

```
print_result(4, lambda x: x * 2)
```

Fonctions génératrices

- Elles ne peuvent être parcourues qu'une seule fois
- On ne peut accéder à un élément par un indice

```
def countfrom(x):  
    while True:  
        yield x  
        x += 1  
  
for n in countfrom(10):  
    print n  
    if n > 20: break
```

Récapitulatif / erreurs fréquentes

- Différence chaîne de caractères vs variables :
 - `'mon_nom'` vs `mon_nom` vs `'Bonjour mon_nom'` vs `'Bonjour %s'%(mon_nom)`
- Tableau, accès par indice:
 - `mon_tableau = []` vs `mon_tableau[i] = 'hello'`
- Définition méthode/fonction utilisation:
 - `def ma_methode(self):` vs `instance.ma_methode()`
- La boucle for:
 - `for i in range(5):` / `for i in [0, 1, 2, 3, 4]:` / `for element in mon_tableau:`

Gestion des fichiers

Ouvrir, lire et écrire dans un fichier

```
fichier = open("data.txt", "r")  
print(fichier.read())  
fichier.close()
```

```
fichier = open("data.txt", "a")  
fichier.write("Bonjour monde")  
fichier.close()
```

```
with open("data.txt", "r") as fichier :  
    print(fichier.read())
```

Types d'ouvertures

- **r**, pour une ouverture en lecture (READ).
- **w**, pour une ouverture en écriture (WRITE), à chaque ouverture le contenu du fichier est écrasé. Si le fichier n'existe pas python le crée.
- **a**, pour une ouverture en mode ajout à la fin du fichier (APPEND). Si le fichier n'existe pas python le crée.
- **b**, pour une ouverture en mode binaire.
- **t**, pour une ouverture en mode texte.
- **x**, crée OBLIGATOIREMENT un NOUVEAU fichier et l'ouvre pour écrire

Les répertoires

- `os.mkdir(chemin, mode)` : crée répertoire, mode UNIX
- `os.remove(chemin)` : supprime fichier
- `os.removedirs(chemin)` : supprime répertoires récursivement
- `os.rename(chemin_old, chemin_new)` : renomme fichier ou répertoire
- `os.rename(chemin_old, chemin_new)` : renomme fichier ou répertoire en créant les répertoires si ils n'existent pas
- `os.chdir(chemin)` : change le répertoire de travail
- `os.getcwd()` : affiche répertoire courant

Les répertoires

- `os.path.exists(chemin)` : est-ce que le fichier ou répertoire existe
- `os.path.isdir(chemin)` : est-ce un répertoire
- `os.path.isfile(chemin)` : est-ce un fichier
- `os.listdir(chemin)` : liste un répertoire

Utiliser le module `glob` qui permet l'utilisation de wildcards

- `glob.glob(pattern)` : liste le contenu du répertoire en fonction du `pattern`

Les répertoires

- `shutil.move(src, dest)` : déplace ou renomme un fichier ou un répertoire
- `shutil.copy(src, dest)` : copie un fichier ou un répertoire
- `shutil.copy2(src, dest)` : copie un fichier ou un répertoire avec les métadonnées
- `os.chmod(path, mode)` : change les permissions
- `os.path.dirname(path)` : retourne l'arborescence de répertoires
- `os.path.basename(path)` : retourne le nom du fichier
- `os.path.split(path)` : retourne un tuple des deux précédents
- `os.path.splitext(path)` : retourne un tuple pour obtenir l'extension

Les exceptions en bref

```
annee = input()
try: # On essaye de convertir l'année en entier
    annee = int(annee)
except:
    print("Erreur lors de la conversion de
l'année.")
finally:
    print("S'affiche de toute manière.")
```

Les exceptions en bref

```
raise TypeDeLException("message à afficher")
```

Les exceptions en bref

```
try:
    resultat = numerateur / denominateur
except NameError:
    print("La variable numerateur ou denominateur
n'a pas été définie.")
except TypeError:
    print("La variable numerateur ou denominateur
possède un type incompatible avec la division.")
except ZeroDivisionError:
    print("La variable denominateur est égale à 0.")
```



Modules et Packages



Modules et Packages

Commencent par :

```
#!/usr/bin/env python
```

```
# -*- coding: utf-8 -*-
```

Doit contenir un `__init__.py` :

MyPackage/

`__init__.py`

 MyModule.py

 MyModule2.py

```
__all__ = [ 'MyModule', 'MyModule2']
```



Importer les modules

```
import MyModuleLibrary.MyModule  
import MyModuleLibrary.MyModule2
```

```
MyModuleLibrary.MyModule.function_welcome()  
MyModuleLibrary.MyModule2.function_welcome_bis()
```




Module `__name__`

```
if __name__ == '__main__':  
    giveAnswer()
```



Empaqueter son module

```
setup.py
src/
  mypkg/
    __init__.py
    module.py
  data/
    tables.dat
    spoons.dat
    forks.dat
```



Empaqueter son module : setup.py

```
#!/usr/bin/env python

from distutils.core import setup

setup(name='Distutils',
      version='1.0',
      description='Python Distribution Utilities',
      author='Greg Ward',
      author_email='gward@python.net',
      url='https://www.python.org/sigs/distutils-sig/',
      packages=['mypkg'],
      requires=['numpy'],
      package_dir={'mypkg': 'src/mypkg'},
      package_data={'mypkg': ['data/*.dat']},
      )
```



Empaqueter son module : setup.py

```
python setup.py install
```



ArgParse : la ligne de commande

```
parser = argparse.ArgumentParser(  
    description="This script does something.")  
parser.add_argument("who", help="Who are you ?")  
parser.add_argument("many", type=int)  
args = parser.parse_args()  
for i in range(args.many):  
    print("Hello " + args.who)
```



ArgParse : la ligne de commande

```
parser = argparse.ArgumentParser(  
    description="This script does something.")  
parser.add_argument("--who", help="Who are you ?")  
parser.add_argument("--many", type=int)  
args = parser.parse_args()  
for i in range(args.many):  
    print("Hello " + args.who)
```



La Programmation Orientée Objet (POO)

Les paradigmes de programmation

Il s'agit des différentes façons de raisonner et d'implémenter une solution à un problème en programmation.

- La programmation impérative : paradigme originel et le plus courant
- La programmation orientée objet (POO) : consistant en la définition et l'assemblage de briques logicielles appelées objets
- La programmation déclarative consistant à déclarer les données du problème, puis à demander au programme de le résoudre
- Fonctionnelle ...



Les Objets

Caractérisés par

- Un état : ses attributs
- Des comportements : ses méthodes



Les Classes

Sont les définition des objets

- Un objet est une instance d'une classe
- En POO, nous définissons des classes
- En POO, nous manipulons des instances des classes
- Le type d'un objet est sa classe

Les Classes

```
class Personne:
```

```
    """Classe définissant une personne caractérisée par :
```

- son nom
- son prénom
- son âge
- son lieu de résidence"""

```
def __init__(self):
```

```
    """Pour l'instant, on ne va utiliser qu'un seul paramètre"""
```

```
    self.nom = "Dupont«
```

```
    self.prenom = "Jean"
```

```
    self.age = 33
```

```
    self.lieu_residence = "Paris"
```

Les Classes

```
class Personne:
```

```
    """Classe définissant une personne caractérisée par :
```

- son nom
- son prénom
- son âge
- son lieu de résidence"""

```
def __init__(self, nom, prenom):
```

```
    """constructeur"""
```

```
    self.nom = nom
```

```
    self.prenom = prenom
```

```
    self.age = 33
```

```
    self.lieu_residence = "Paris"
```

Attention à ne PAS OUBLIER self !

Les Objets (sont des instances de classe)

```
personne= Personne("Martin","Jean")  
print(personne.nom)
```



Les Classes

```
class Compteur:
```

```
    """Cette classe possède un attribut de classe qui s'incrémente à chaque  
    fois que l'on crée un objet de ce type"""
```

```
    objets_crees = 0 # Le compteur vaut 0 au départ
```

```
    def __init__(self):
```

```
        """À chaque fois qu'on crée un objet, on incrémente le compteur"""
```

```
        Compteur.objets_crees += 1
```

Méthodes spéciales

- `__init__(self)` : initialiseur appelé juste après l'instanciation d'un objet
- `__del__(self)` : destructeur, appelé juste avant la destruction de l'objet
- `__str__(self) -> str` : est appelé par la fonction de conversion de type `str()` et par la fonction `print()`. Elle doit donc retourner une chaîne de caractères représentant l'objet.
- `__repr__(self) -> str` : est appelé par la fonction `repr()` et doit retourner une chaîne de caractères contenue entre des chevrons et contenant non, type de l'objet et informations additionnelles.

Méthodes spéciales

Méthode	Opération
<code>__lt__(self, other)</code>	$x < y$
<code>__le__(self, other)</code>	$x \leq y$
<code>__eq__(self, other)</code>	$x == y$
<code>__ne__(self, other)</code>	$x \neq y$
<code>__ge__(self, other)</code>	$x \geq y$
<code>__gt__(self, other)</code>	$x > y$

Méthodes spéciales

Méthode	Opération
<code>__neg__</code>	$-x$
<code>__add__</code>	$x + y$
<code>__sub__</code>	$x - y$
<code>__mul__</code>	$x * y$
<code>__div__</code>	x / y

Encapsulation

L'objet est une « boîte noire »

- Faciliter la modification interne sans perturber l'utilisateur
- Gérer la complexité en interne
- Sécuriser l'utilisation d'un objet



L'encapsulation :

attribut privé / setters & getters

```
class Personne:
    def __init__(self, nom, prenom):
        """constructeur"""
        self._nom = nom
        self._prenom = prenom

    def getNom(self):
        """getter nom"""
        return self._nom

    def setNom(self, nom):
        """setter nom"""
        self._nom = nom
```



Héritage Abstraction

- Propriété de généraliser ou spécialiser des états ou comportements
- Généralisation : définition unique, évite duplication
- Spécialisation : adapter caractéristiques et comportements
- Abstraction
- Polymorphisme : « une méthode pour les gouverner toutes »

Le polymorphisme





Le Polymorphisme

compte_suisse = mon_compte + ton_compte

mon_compte_credite = mon_compte + 10



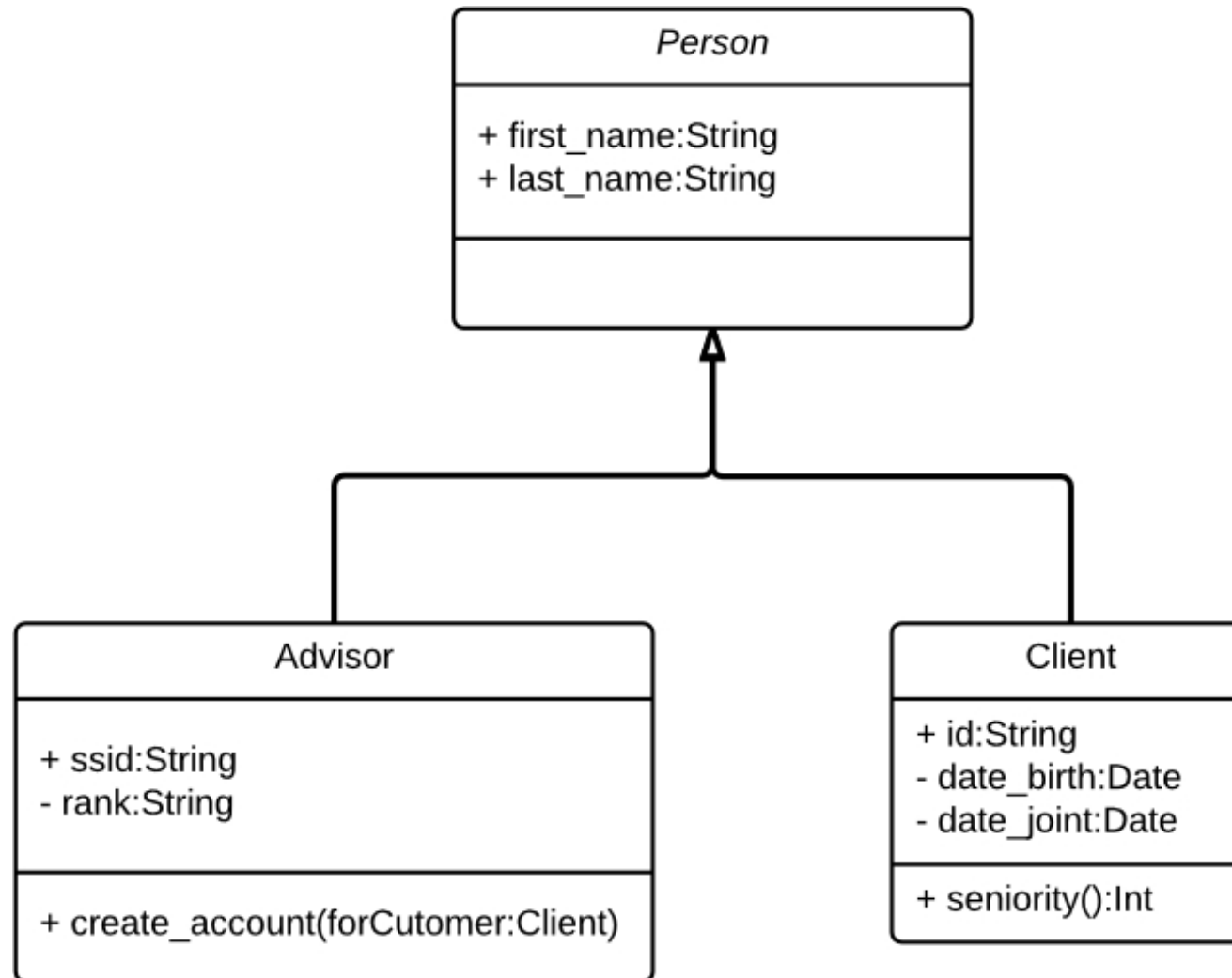
Polymorphisme

- Possibiliter de redéfinir « a posteriori » un comportement »
- Le système choisit dynamiquement la méthode à exécuter sur l'objet en cours, en fonction de son type réel.

Exemple :

- Pour Mercedes, accélère() augmente la vitesse de 10 km/h
- Pour Clio, accélère() augment la vitesse de 2km/h

Héritage Abstraction



Héritage

```
class Client(Personne):
```

```
    """Classe définissant une personne caractérisée par :
```

- son nom
- son prénom
- son âge
- son lieu de résidence"""

```
def __init__(self):
```

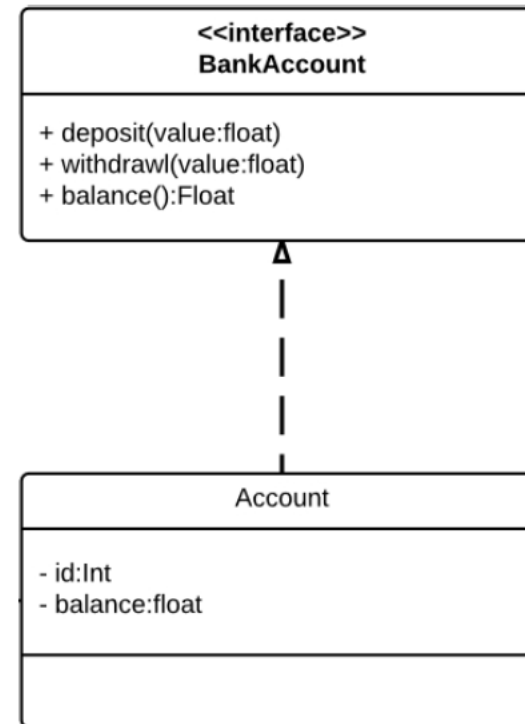
```
    """Pour l'instant, on ne va définir qu'un seul attribut"""
```

```
    Personne.__init__(self)
```

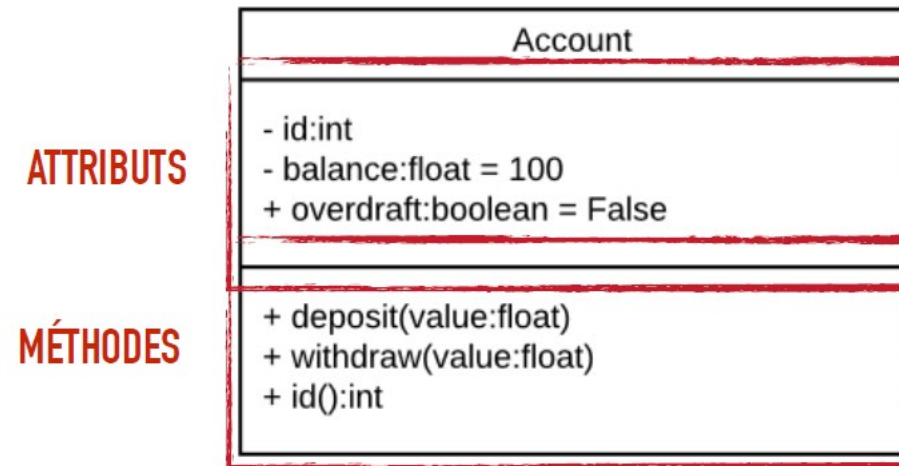


Interface

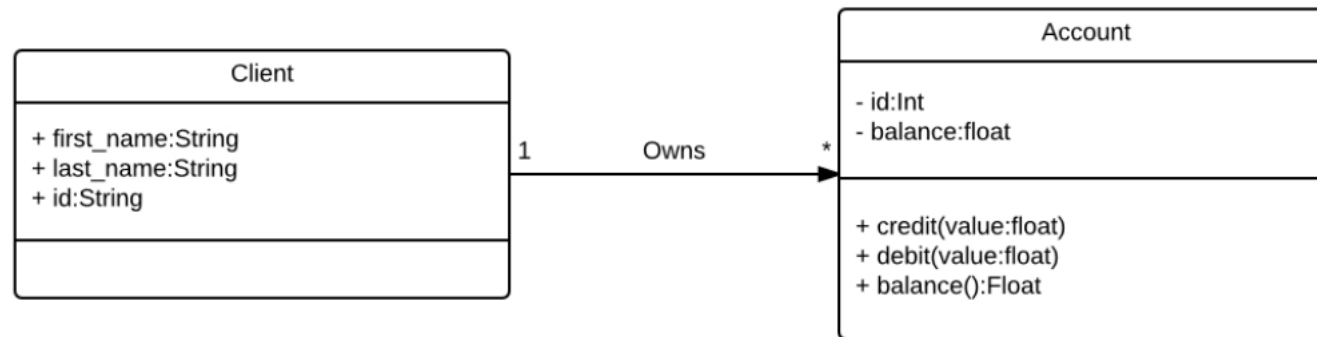
Faciliter le développement de classe
devant interagir avec un même
concept



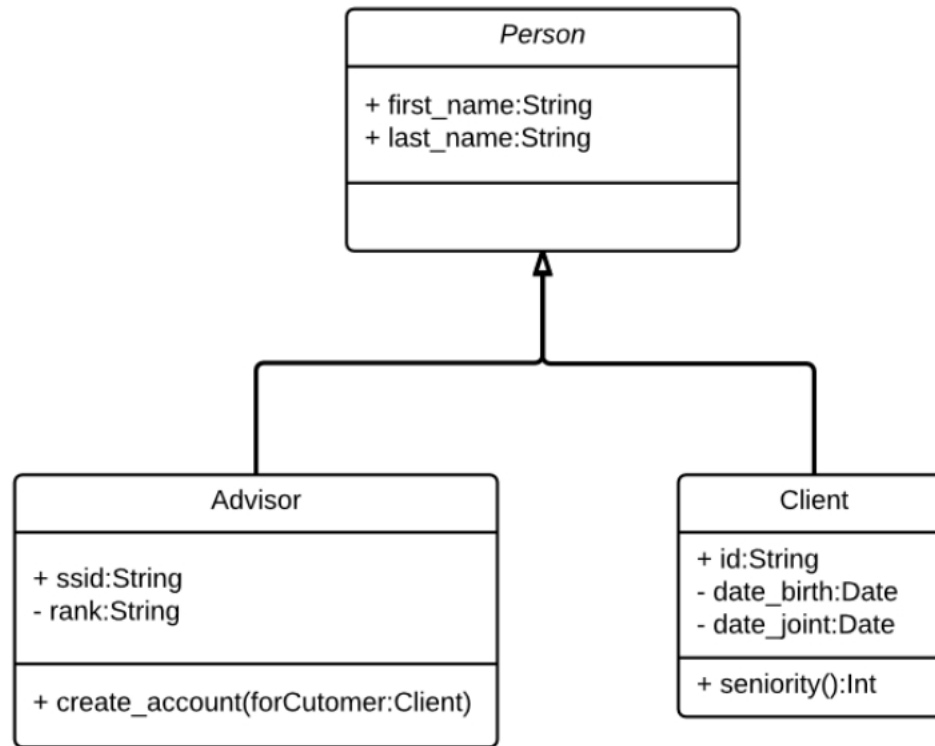
Représentation UML : diagramme de classe



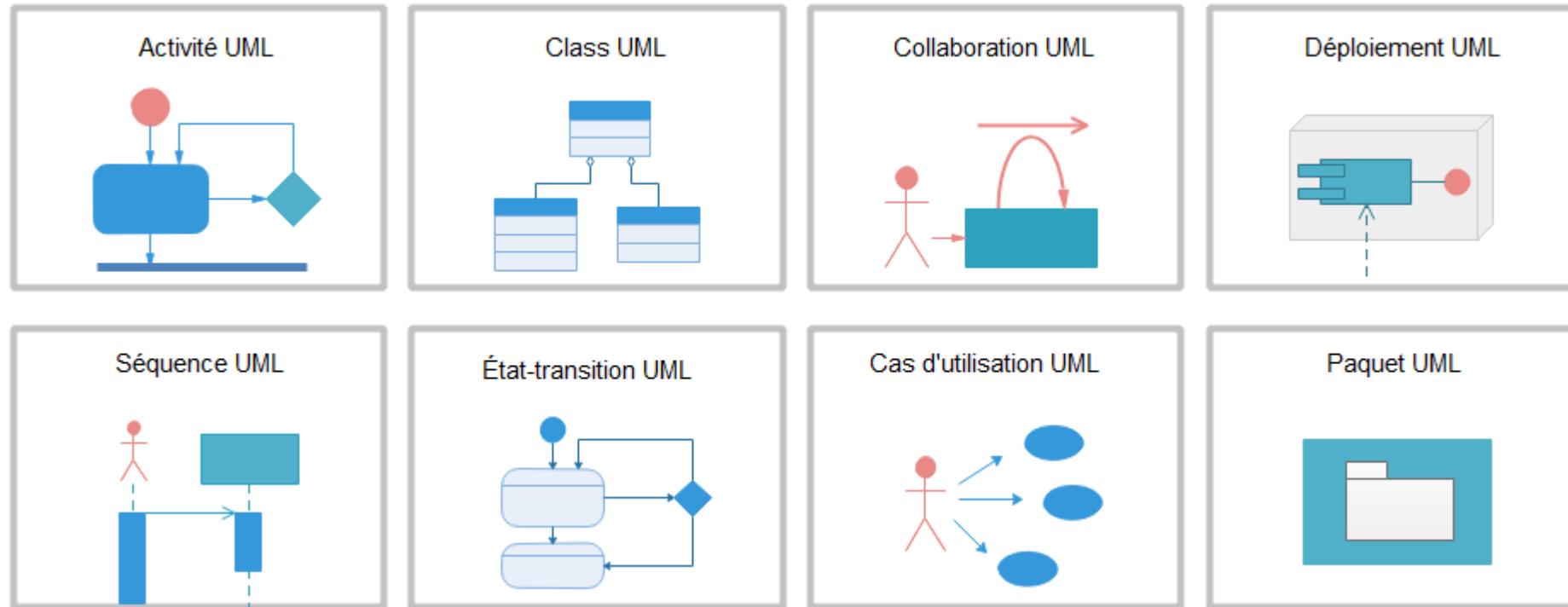
Représentation UML : association



Représentation UML : héritage



Représentation UML : séquences, activité, use case...





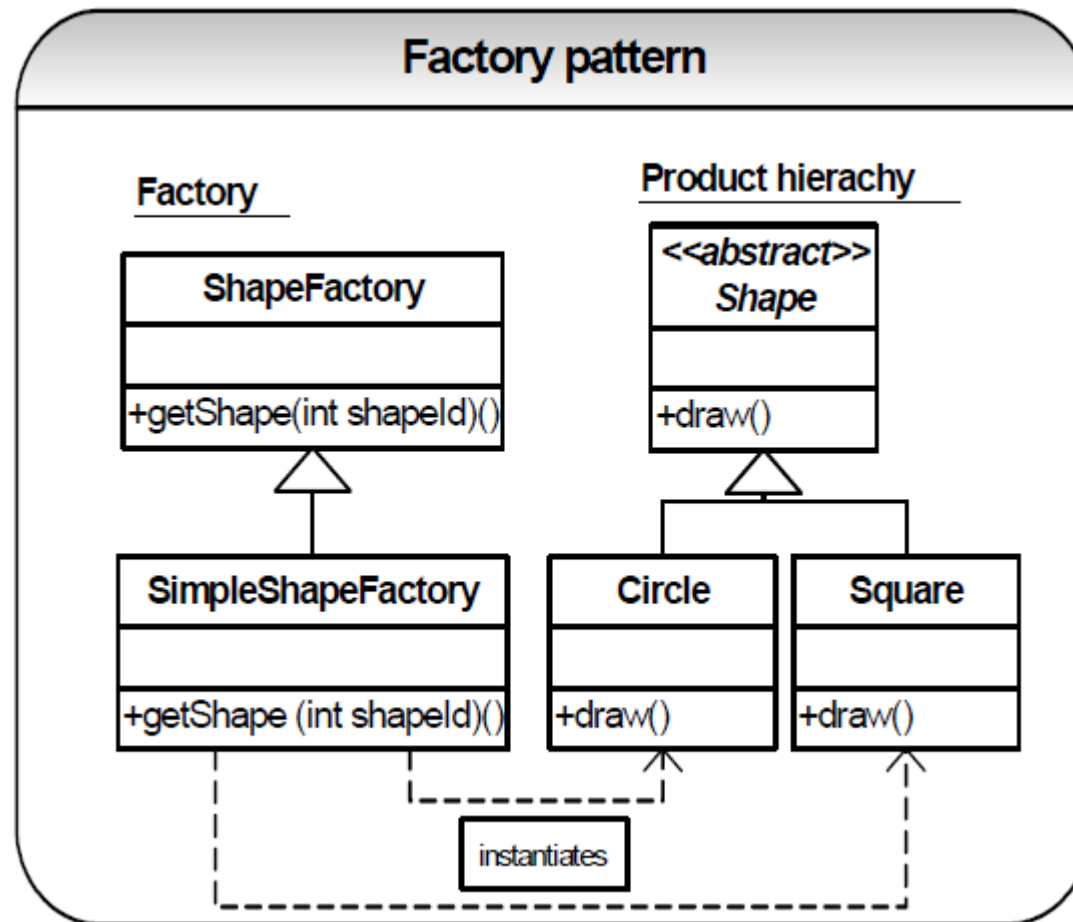
Design pattern

Conceptualiser des situations de développement récurrente.

Sécuriser le développement

- Factory: création d'objet centralisé
- Mediator: Souvent consiste en la création d'un manager
- Facade: fournir une interface unifiée à un système complexe
- Iterator: gérer une liste d'éléments avec next() et hasNext()
- Prototype: Objet de référence copié pour fournir des semblables
- Adapter: Communication entre des objets non-lié

Design pattern : factory



TKinter

Interface graphique (Tkinter) : la fenêtre

```
#!/usr/bin/env python  
# -*- coding: utf-8 -*-  
  
from tkinter import *  
  
fenetre = Tk()  
  
label = Label(fenetre, text="Hello World")  
label.pack()  
  
fenetre.mainloop()
```

Les composants : widgets

- Boutons : Button
- Labels : Label
- Inputs : Entry
- Checkboxes / RadioButtons : Checkbutton / Radiobutton
- SpinBox : Spinbox
- Listes : Listbox
- Canvas : Canvas
- Scale : Scale
- Frame : Frame

Placement par layouts : Pack() / Grid()

- Le placement par la méthode pack() divise le conteneur en deux zones et place le widget dans la zone indiquée par le paramètre side.
 - Exercice : Faire une fenêtre avec un input (en haut à gauche), un bouton validé (en haut à droite), un label (en bas)
- Le placement par la méthode grid() place les éléments selon leur indices dans une grille matricielle.
 - Exercice : faire un pavé numérique

Placement : Pack() / Grid()

```
Canvas(fenetre, width=250, height=50, bg='ivory').pack(side=LEFT, padx=5, pady=5)
Button(fenetre, text='Bouton 1').pack(side=TOP, padx=5, pady=5)
Button(fenetre, text='Bouton 2').pack(side=BOTTOM, padx=5, pady=5)
```

```
for ligne in range(5):
    for colonne in range(5)
        Button(fenetre, text='L%s-C%s' % (ligne, colonne), borderwidth=1).grid(row=ligne, column=colonne)
```

Les fichiers avec tkinter

```
filepath = askopenfilename(title="Ouvrir une image",filetypes=[('png files','*.png'),('all files','*.*)'])
photo = PhotoImage(file=filepath)
canvas = Canvas(fenetre, width=photo.width(), height=photo.height(), bg="yellow")
canvas.create_image(0, 0, anchor=NW, image=photo)
canvas.pack()
```

```
filename = askopenfilename(title="Ouvrir votre document",filetypes=[('txt files','*.txt'),('all files','*.*)'])
fichier = open(filename, "r")
content = fichier.read()
fichier.close()
Label(fenetre, text=content).pack(padx=10, pady=10)
```

Les événements et tkinter

```
def clavier(event):  
    touche = event.keysym  
    print(touche)
```

```
canvas = Canvas(fenetre, width=500, height=500)  
canvas.focus_set()  
canvas.bind("<Key>", clavier)  
canvas.pack()
```

Les Bases de données



Base de données

Principe général

- Établir une connexion
- Créer un curseur et lui attribuer une requête
- Exécuter la requête
- Itérer sur les éléments retournés
- Fermer la connexion

Base de donnée SQL

```
import MySQLdb
try:
    conn = MySQLdb.connect(host='localhost',
        user='test user',
        passed='test pass',
        db='test')
    cursor = conn.cursor()
    cursor.execute("SELECT VERSION()")
    row = cursor.fetchone()
    print('server version', row[0])
finally:
    if conn:
        conn.close()
```

Quelques requêtes PostGre courantes

```
CREATE TABLE COMPANY( ID INT PRIMARY KEY NOT NULL,  
    NAME TEXT NOT NULL,  
    AGE INT NOT NULL,  
    ADDRESS CHAR(50),  
    SALARY REAL,  
    JOIN_DATE DATE );
```

```
INSERT INTO COMPANY (NAME,AGE,ADDRESS,SALARY,JOIN_DATE)  
    VALUES ('Mark', 25, 'Rich-Mond ', 65000.00, '2007-12-13' ),  
    ('David', 27, 'Texas', 85000.00, '2007-12-13');
```

```
DELETE FROM COMPANY WHERE ID = 2;
```

```
DELETE FROM COMPANY;
```

```
SELECT column1, column2, columnN FROM table_name LIMIT 10 OFFSET 20 ORDER BY AGE ASC
```

```
UPDATE COMPANY SET salary = 15000 WHERE ID = 2;
```

Quelques requêtes PostGre courantes

```
SELECT NAME, SUM(SALARY) FROM COMPANY GROUP BY NAME ORDER BY NAME;
```

Autres fonctions : COUNT / MAX / MIN / AVG

```
SELECT EMP_ID, NAME, DEPT FROM COMPANY  
INNER JOIN DEPARTMENT ON COMPANY.ID = DEPARTMENT.EMP_ID;
```

Autres JOIN : LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN

Initiation à Flask

Hello world avec Flask

```
pip install Flask
```

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

Hello world avec Flask

```
set FLASK_APP = main.py  
set FLASK_DEBUG = 1  
flask run
```

Hello world avec Flask

```
Pip install flask-sqlalchemy
```


A parte sur pip

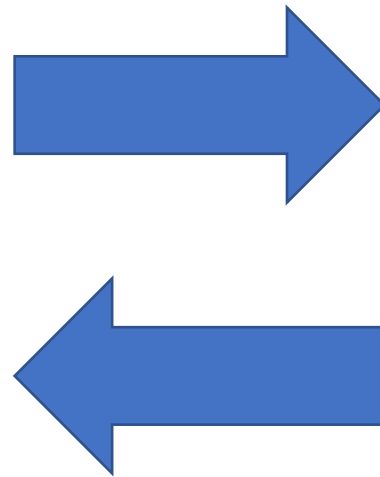
```
pip freeze > requirements.txt
```

```
pip install -r requirements.txt
```

```
Virtualenv
```

Afficher une page web

Coté client : Navigateur



Coté server : Backend





Page HTML de base

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Titre</title>
  </head>

  <body>
    <h1>Hello tout le monde</h1>
  </body>
</html>
```



Coté server

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

@app.route('/base')
def hello():
    return render_template('base.html')

if __name__ == '__main__':
    app.run()
```



Formulaire : server

```
from flask import Flask, render_template
app = Flask(__name__)

...

@app.route('/form', methods=['POST'])
def form():
    print(request.form.get('nom', 'valeur par défaut'))
    return 'Validé'

...

if __name__ == '__main__':
    app.run()
```



Formulaire : client

```
<form method="post">  
  <label for="nom">Mon nom</label>  
  <input type="text" name="nom">  
</form>
```

Qualité



La qualité ? Pour quoi ?

- Conformité aux exigences et aux attentes établies
- Ensemble des actions permettant d'assurer la fiabilité, la maintenance et l'évolutivité du logiciel
- Suivie par l'ensemble des mesures mises en place



La qualité ? Pour quoi ?

- les délais de livraison des logiciels sont rarement tenus, le dépassement de délai et de coût moyen est compris entre 50 et 70 %
- la qualité du logiciel correspond rarement aux attentes, le logiciel ne correspond pas aux besoins, il consomme plus de moyens informatiques que prévu, et tombe en panne
- les modifications effectuées après la livraison d'un logiciel coûtent cher, et sont à l'origine de nouveaux défauts.
- il est rarement possible de réutiliser un logiciel existant pour en faire un nouveau produit de remplacement



La qualité ? Que faire ?

- Mise en place de tests unitaires
- Mise en place de règles de programmation
- Mise en place de métriques liées à l'analyse du code (couverture de code)
- Mise en pratique et validation sur une plate-forme d'intégration continue

De la documentation et des tests !



Documenter son code

- Informer de ce que fait le code
- Informer pourquoi le code est écrit de cette manière
- Informer sur le comportement du code (des fonctions, objets...)





Les commentaires pour la doc courte

```
# x is set to 10
```

```
x = 10
```

```
# x is set to the last list element
```

```
x = ma_liste[-1]
```

```
# account number is the last element of bank infos
```

```
numero_compte = infos_bancaire[-1]
```



La docstring pour une description complète

```
def add(a, b):  
    """  
        Adds two numbers and returns the result.  
        :param a: The first number to add  
        :param b: The second number to add  
        :type a: int  
        :type b: int  
        :return: The result of the addition  
        :rtype: int  
        .. seealso:: sub(), div(), mul()  
        .. warnings:: This is a completely useless function. Use it only in a  
        tutorial unless you want to look like a fool.  
    """  
    return a + b
```



Les tests

- Montrer que le code fonctionne
- Montrer que le code répond aux attentes
- Illustrer l'usage du code
- Montrer que le code fonctionne toujours



Les test dans la docstring

```
def add(a, b):  
    """  
        :Example:  
        >>> add(1, 1)  
        2  
        >>> add(2.1, 3.4)  
        5.5  
    """  
    return a + b  
  
if __name__ == "__main__":  
    import doctest  
    doctest.testmod()
```



Les tests unitaires

- Un test unitaire doit tester une fonctionnalité et une seule
- Un test unitaire doit être indépendant et isolé du système



Les test unitaire avec unittest

```
import unittest
from training.poo.bank import bank
class TestDeposit(unittest.TestCase):
    def setUp(self):
        self.account = bank.BankAccount('012345', 500)
    def testBasicDeposit(self):
        self.account.deposit(100)
        self.assertEqual(600, self.account.balance())
    def tearDown(self):
        del self.account
```



Unittest assertions

- `assertEqual(a, b)` -> `a == b`
- `assertNotEqual(a, b)` -> `a != b`
- `assertTrue(x)` -> `x` is `True`
- `assertFalse(x)` -> `x` is `False`
- `assertIs(a, b)` -> `a` is `b`
- `assertIsNot(a, b)` -> `a` is not `b`
- `assertIsNone(x)` -> `x` is `None`
- `assertIsNotNone(x)` -> `x` is not `None`
- `assertIn(a, b)` -> `a` in `b`
- `assertNotIn(a, b)` -> `a` not in `b`
- `assertIsInstance(a, b)` -> `isinstance(a, b)`
- `assertNotIsInstance(a, b)` -> `not isinstance(a, b)`



Unittest les Exception vérifiée

```
class TestDeposit(unittest.TestCase):  
    def testNegativeDeposit(self):  
        with self.assertRaises(ValueError):  
            self.account.deposit(-100)
```



Pour aller plus loin

- TestSuite (aggregation)
- coverage run -m tests.montest (pour tester la couverture de test)
- python -m pdb monfichier.py (le débbugger)
- PyLint le linter
- cProfile : python -m cProfile -s cumtime mon_script.py

Testons ces outils sur notre code !



Pour aller plus loin : les commande du débbugger

- l : (list) liste quelques lignes de code avant et après
- n : (next) exécute ligne suivante
- s : (step in) entre dans la fonction
- r : (return) sort de la fonction
- unt : (until) si dernière ligne boucle, reprend jusqu'à l'exécution boucle
- q : (quit) quite brutalement le programme
- c : (continue) reprend l'exécution



Pour aller plus loin : utiliser coverage

```
$ coverage run my_program.py arg1 arg2
```

```
$ coverage run --source=dir1,dir2 my_program.py arg1 arg2
```

```
$ coverage report
```

Name	Stmts	Miss	Cover
------	-------	------	-------

test.py	15	0	100%
---------	----	---	------



Pour aller plus loin : utiliser PyLint

```
pylint test.py
```

```
No config file found, using default configuration
```

```
***** Module test
```

```
C: 1, 0: Missing module docstring (missing-docstring)
```

```
C: 1, 0: Invalid constant name "ma_variable1" (invalid-name)
```

```
Global evaluation
```

```
-----
```

```
Your code has been rated at 3.33/10
```



Pour aller plus loin : cProfile

```
python -m cProfile -o profilage.txt test.py
```

Ordered by: standard name

```
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
      1   0.000    0.000    0.000    0.000 <string>:1(<module>)
      1   0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
```




La TDD : Test Driven Development

- Commencer par faire les cas d'usage (cf UML)
 - On en déduit les cas de test
 - Développer les test
 - Développer la fonctionnalité
- > Oblige à prévenir les bugs liés aux cas particuliers

Ex : La classe voiture !

Python / C



CTypes

- Appeler depuis python des “shared libraries”
- Récupérer les types C en python
- `c_bool` , `c_char`, `c_int`, `c_float`



Chargement du C en python

```
from ctypes import *  
  
print windll.kernel32  
  
libc = CDLL("libc.so.6")  
libc.printf("Hello, %s\n", "World!")  
i = c_int(42)
```



Chargement du Python en C

```
gcc -I/usr/include/python2.7 prog.c -lpython2.7 -o prog -Wall && ./prog
```

```
def coucou(arg):  
    return arg.upper() + ' !!!'
```

```
#include <Python.h>
```

```
int main () {  
    PyObject *retour, *module, *fonction, *arguments;  
    char *result;  
  
    Py_Initialize();
```



Chargement du C en python

```
PySys_SetPath(".");  
module = PyImport_ImportModule("test.py");  
fonction = PyObject_GetAttrString(module, "coucou");  
arguments = Py_BuildValue("(s)", "Hello guys");  
// Appeler la fonction  
retour = PyEval_CallObject(fonction, arguments);  
// Conversion du PyObject obtenu en string C  
PyArg_Parse(retour, "s", &result);  
printf("Retour: %s\n", result);  
Py_Finalize();  
return 0;  
}
```



Pyrex ou Cython

`python setup.py build_ext --inplace`

`cythonize -a -i module.pyx`

```
def say_hello_to(name):  
    print("Hello %s!" % name)
```

```
from distutils.core import setup  
from Cython.Build import cythonize  
  
setup(  
    name = 'Hello world app',  
    ext_modules = cythonize("hello.pyx"),  
)
```