Brève introduction à Python 2.x

Valable en grande partie aussi pour Python 3.x (attention à la syntaxe du *print* cependant)

UE 3i025 - IA et RO (2015-2016)

UE LV348 - Bioinformatique (2014-2015)

UPMC SORBONNE UNIVERSITÉS

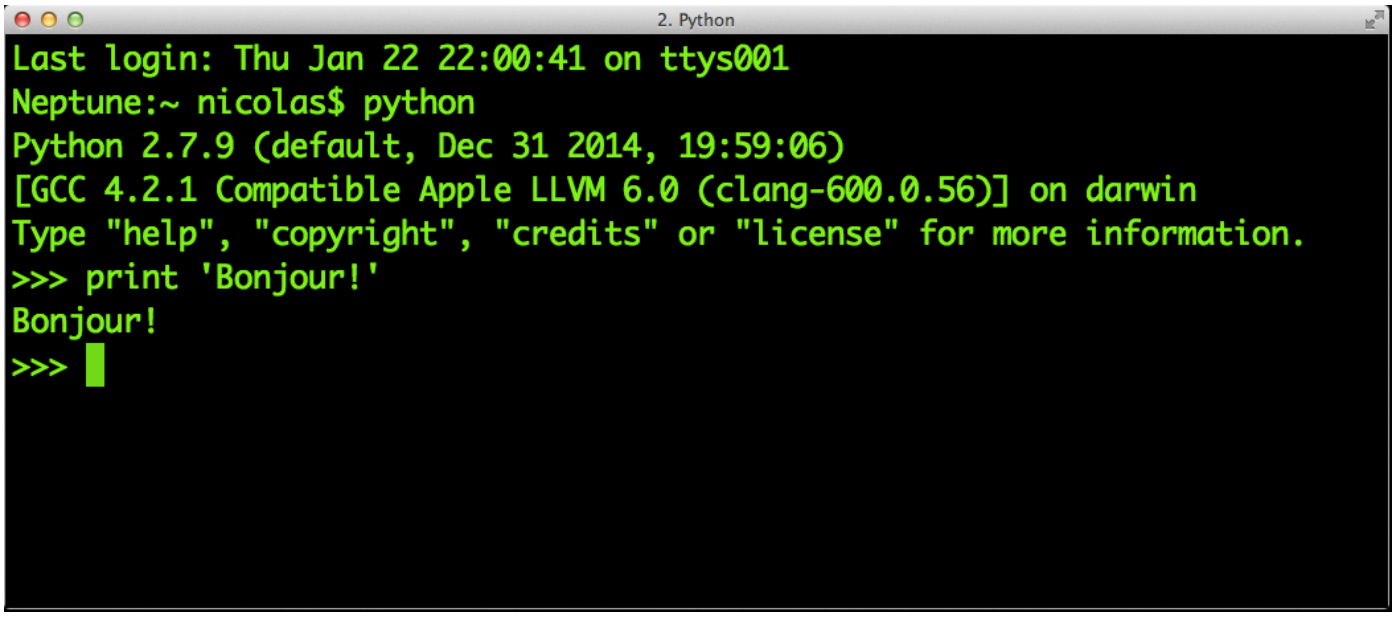
N. Bredeche (nicolas.bredeche@upmc.fr)

Définitions

- Python
  - Un langage interprété
  - De (très) nombreuses librairies
  - Très utilisé dans de nombreux domaines
  - Plusieurs implémentations: Python 2.7, Python 3.3

- Python interactif
  - On tape « python » dans un terminal
  - Pour sortir: CTL+D ou quit()

- Programme python
  - On utilise un éditeur de texte (ex.: *gedit*)
  - On enregistre un fichier avec l'extension *.py*
  - Dans le terminal, on tape: *python monfichier.py*
    - remarques:
      - on peut lancer directement le programme si on place la ligne suivante au début: *#!/usr/bin/python*
      - *pour éviter les problèmes d'accents: # -*- coding: utf-8 -*-*



Terminal

```
Neptune:~ nicolas$ python
```



```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
```



```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
>>>
```

```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
>>> print a
```



```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
>>> print a
hello
>>>
```

```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
>>> print a
hello
>>> for i in range(5):
```



```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
>>> print a
hello
>>> for i in range(5):
...
```

```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
>>> print a
hello
>>> for i in range(5):
...
```

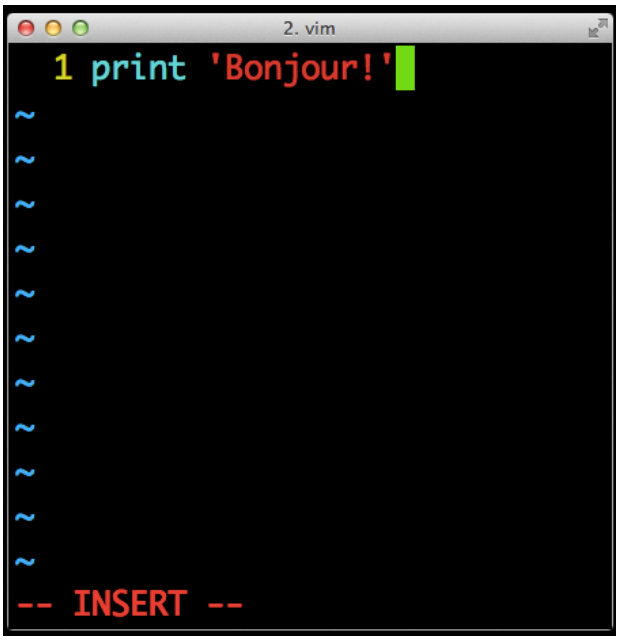


```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
>>> print a
hello
>>> for i in range(5):
...     print a
```

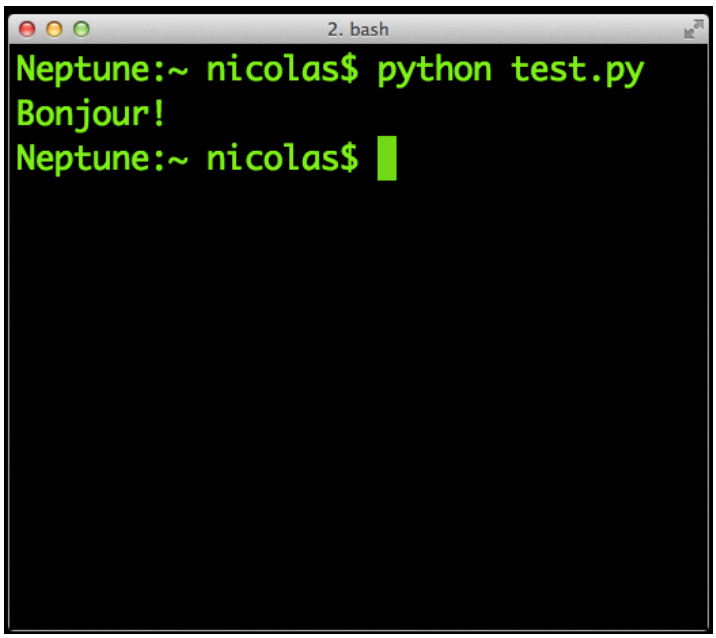
```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
>>> print a
hello
>>> for i in range(5):
...     print a
...
```



```
Neptune:~ nicolas$ python
Python 2.7.9 (default, Dec 31 2014, 19:59:06)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "hello"
>>> print a
hello
>>> for i in range(5):
...     print a
...
hello
hello
hello
hello
hello
>>>
```



Editeur de texte



Terminal

Types de données

La partie sur les listes est en partie repris du cours 2I015 (UPMC, L2 SdV)

- entier, réel
- chaînes de caractères
- listes
- dictionnaires

Entiers et réels


```
i = 3
a = 3.14
c = i/2
```

- Opérations
  - opérations de base: +, -, *, /
  - modulo : %
  - puissance: **

- Remarque sur l'encodage
  - un entier est encodé exactement
  - un entier divisé par un entier donne un entier *(c vaut 1!)*
  - un réel est encodé *approximativement*

Type de données : remarques générales

- Fonctions utiles
  - len(*variable*)
  - type(*variable*)
- Conversion (si c'est possible)
  - int(*variable*)
  - float(*variable*)
  - str(*variable*)

Chaînes de caractères


```
machaine = 'hippopotame'
```

- Déclaration et affectation
  - animal = 'hippopotame' ou animal = "hippopotame"
- Accès à un (ou plusieurs) élément(s)
  - animal[4] => 'o'
  - for k in animal:
    - print k
- Extraction d'une sous-chaîne
  - print animal[6:10]
    - résultat: 'otam'

- somme
  - m = ‘bla’
  - n = ‘ireau’
  - animal = m + n
    - attention à l’ordre!
  - print animal
    - résultat: ’blaireau’

- produit
  - print animal*3
    - résultat: ‘blaireaublaireaublaireau’

Listes


```
maliste = [ 19, 17, 20, 17, 18, 19, 21, 18 ]
maliste2 = [ ‘upmc’, ‘LV348’, 2015 ]
maliste3 = []
```

- Créer une liste
  - l = []
    - *liste vide*
  - l = [0,1,2,5]
    - résultat: *l* contient [0,1,4,5]
  - l = range(0,100,2)
    - résultat: *l* contient [0,2,4,…,96,98]

- Fonction spéciale: *range(a, b, delta)*
  - construit une liste contenant tout [a,b[avec intervalle delta
  - notation légère: *range(b) <=> range (0,b,1)*

- Ajouter un élément à une liste
  - l = [0,1,2]
  - l.append(5)
    - résultat: *l* contient [0,1,2,5]

- Modifier un élément
  - l[2] = 4
    - résultat: *l* vaut [0,1,4,5]

- *on peut aussi enlever un élément, mais c'est déconseillé*

- On peut faire des listes de listes
  - et aussi: des listes de listes de listes de n'importe quoi

```
>>> maliste3.append(maliste2)
>>> maliste3.append(maliste2)
>>> maliste3.append(2)
>>> maliste3
[[ 'upmc', 'LV348', 2015 ], [ 'upmc', 'LV348', 2015 ], 2]
>>> len(maliste3)
3
>>> len(maliste3[1])
3
>>> len(maliste3[2])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no len()
>>>
```

Listes (cont.)

- Obtenir une sous-liste
  - jours = [‘lundi’,’mardi’,(…),’samedi’,’dimanche’]
  - troisjours = jours[1:4]
    - résultat: *troisjours* vaut [’mardi’,’mercredi’,’jeudi’]
  - weekend = jours[5:7]
    - ou weekend = jours[5:]
    - ou weekend = jours[-2:]
    - résultat: *weekend* vaut [’samedi’,’dimanche’]
  - debutsemaine = jours[:2]
    - résultat: *debutsemaine* vaut [’lundi’,’mardi’]
  - dernierjour = l[len(jours)-1:] *ou* dernierjour = l[-1:]
    - résultat: *dernierjour* vaut [’dimanche’]

de 5 à *(fin)*

de *(2 avant la fin)* à *(fin)*

de *(début)* à 2, exclus

Listes (cont.)

- Addition / Concaténation
  - l1= [0,1,2]
  - l2 = [3,4,5]
  - l3 = l1 + l2
    - résultat: l3 contient [0,1,2,3,4,5]
      - attention, l1+l2 est différent de l2+l3
- Multiplication
  - l4 = l1 * 3
    - résultat: l4 contient [0,1,2,0,1,2,0,1,2]

Remarques: l1 et l2 peuvent être modifiés sans changer l3 ou l4

Parcourir une liste

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

i = 0
while i < len(jours):                    # parcours par indice (1)
    print jours[i]
    i = i + 1

for i in range(0,len(jours),1):          # parcours par indice (2)
    print jours[i]

for j in jours:                          # parcours par éléments
    print j
```


Chaînes de caractères vs. listes

- Une chaîne n'est pas une liste
  - machaine = 'chapeau'
  - maliste = ['c','h','a','p','e','a','u']
- Modification d'un élément
  - machaine = machaine[0:3] + 't' + machaine[4:7]
    - alors que *maliste[3] = 't'* suffit pour une liste
- Etendre d'un élément
  - machaine = machaine + 'x'
    - alors que pour une liste: maliste.append('x')

Liste et tableaux

- Comment faire un tableau?
  - en Python, pas de tableau…mais des listes de listes!
    - Exemple pour un tableau 2D d'entier

```
liste = []
liste.append([0,1,2])
liste.append([3,4,5])
liste.append([6,7,8])
print liste
```

```
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
```

```
print liste[1][2]
```

```
5
```

```
for ligne in liste:
    for cellule in ligne:
        print cellule,
    print                    # retour à la ligne
```

```
0 1 2
3 4 5
6 7 8
```

… même raisonnement pour la 3D, la 4D, etc.

Dictionnaires

```
dico = {}
```

```
dico['nom'] = 'Woolf'
dico['prenom'] = 'Virginia'
dico['date de naissance'] = 1882
dico['livres'] = []
```

```
dico['livres'].append('Mrs Dalloway')
dico['livres'].append('To the lighthouse')
dico['livres'].append('Orlando')
dico['livres'].append('A room of One\'s own')
```

```
print 'Contenu du dictionnaire: '
print dico
```

```
Contenu du dictionnaire:
{'nom': 'Woolf', 'livres': ['Mrs Dalloway', 'To the lighthouse', 'Orlando', "A room of One's own"], 'prenom': 'Virginia', 'date de naissance': 1882}
```

```
print 'Keys: '
print dico.keys()
```

```
Keys:
['nom', 'livres', 'prenom', 'date de naissance']
```

```
print 'Values: '
print dico.values()
```

```
Values:
['Woolf', ['Mrs Dalloway', 'To the lighthouse', 'Orlando', "A room of One's own"], 'Virginia', 1882]
```

```
if dico.has_key('livres') == True:
    print dico['prenom'] + ' ' + dico['nom'] + ' a ecrit des livres'
```

```
Virginia Woolf a ecrit des livres
```


- On peut faire des listes de dictionnaires…

```
dico1 = { 'nom':'Holmes', 'occupation':'detective' }   # initialisation+affectation
dico2 = { 'nom':'Watson', 'occupation':'docteur' }     # initialisation+affectation
binome = [ dico1, dico2 ]                              # création d'une liste
print binome
```

```
[{'nom': 'Holmes', 'occupation': 'detective'}, {'nom': 'Watson', 'occupation': 'docteur'}]
```

- Un tuple
  - monTuple = (0,1,2,'abc')
  - print monTuple[0]
    - résultat: affiche 0
  - monTuple[0]=3
    - résultat: ERREUR! on ne peut pas modifier un tuple
- Comparaison avec une liste
  - Jamais modifiable après initialisation
  - Avantages (parce qu'il y en a)
    - Permet de protéger les données qu'on ne veut voire modifiée
    - Depuis une fonction, permet de renvoyer facilement des données
    - Un tuple peut servir de clé pour un dictionnaire

Entrées/Sorties

```
a = 3
print a                                                    3
print 2 + ' pommes et ' + a + ' poires'                    ##ERREUR##
print str(2) + ' pommes et ' + str(a) + ' poires'          2 pommes et 3 poires
print 2, 'pommes et ', 3, 'poires'                         2 pommes et 3 poires
```



```
>>> print 'j\'utilise des \"guillemets\" dans\ncette phrase.'
j'utilise des "guillemets" dans
cette phrase.
>>>
```

- Caractères spéciaux
  - \n saut de ligne
  - \t tabulation
  - \' apostrophe (single quote)
  - \" guillemet (double quote)

```
print 'Bonjour!'

nom = raw_input('Comment t appelles-tu? ')
nbPommes = raw_input('Combien veux tu de pommes? ')
nbPoires = raw_input('Combien veux tu de poires? ')

nbFruits = nbPommes + nbPoires

print nom + ', Je te donne ', nbFruits, ' fruits!'
```



```
print 'Bonjour!'

nom = raw_input('Comment t appelles-tu? ')
nbPommes = raw_input('Combien veux tu de pommes? ')
nbPoires = raw_input('Combien veux tu de poires? ')

nbFruits = nbPommes + nbPoires

print nom + ', Je te donne ', nbFruits, ' fruits!'
```

QUESTION: combien de fruits si je veux 3 pommes et 3 poires?

```
print 'Bonjour!'

nom = raw_input('Comment t appelles-tu? ')
nbPommes = raw_input('Combien veux tu de pommes? ')
nbPoires = raw_input('Combien veux tu de poires? ')

nbFruits = nbPommes + nbPoires

print nom + ', Je te donne ', nbFruits, ' fruits!'
```

REPONSE: 33 — Attention au TYPE des variables!


```
print 'Bonjour!'

nom = raw_input('Comment t appelles-tu? ')
nbPommes = int(raw_input('Combien veux tu de pommes? '))
nbPoires = int(raw_input('Combien veux tu de poires? '))

nbFruits = nbPommes + nbPoires

print nom + ', Je te donne ', nbFruits, ' fruits!'
```

REPONSE: 6

```
f1 = open("bonjour.txt","w")
f1.write("Bonjour le fichier!\n")
f1.write("Comment ca va?\n")
f1.write("Ca va bien?\n")
f1.close()
```

contenu de bonjour.txt :

```
Bonjour le fichier!
Comment ca va?
Ca va bien?
```



```
f2 = open("bonjour.txt","r")
ligne = f2.readline()
while len(ligne) > 0:
    print ligne
    ligne = f2.readline()
f2.close()
```

```
Bonjour le fichier!

Comment ca va?

Ca va bien?
```

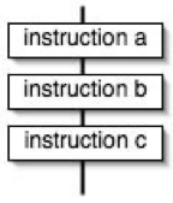
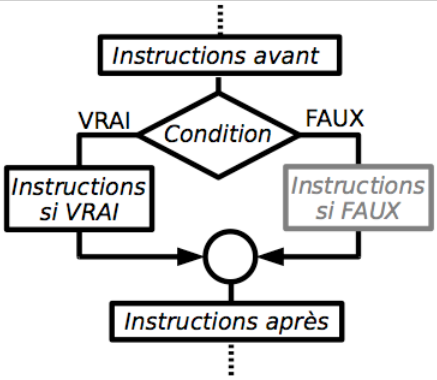
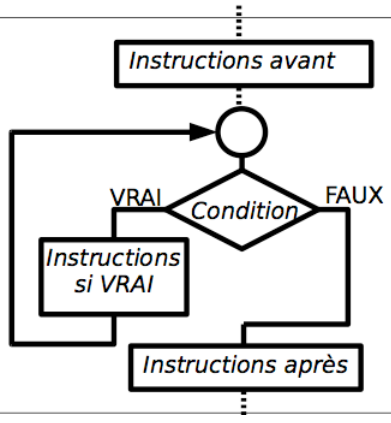
```
f2 = open("bonjour.txt","r")
ligne = f2.readline()
while len(ligne) > 0:
    print ligne[0:-1]
    ligne = f2.readline()      # enlève le '\n'
f2.close()
```

```
Bonjour le fichier!
Comment ca va?
Ca va bien?
```

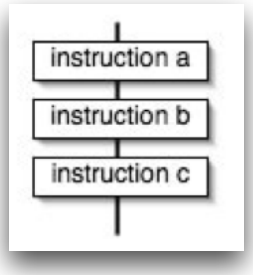
```
f2 = open("bonjour.txt","r")
tout = f2.readlines()    # 'tout' est une liste
f2.close()
for ligne in tout:
    print ligne[0:-1]
```

```
Bonjour le fichier!
Comment ca va?
Ca va bien?
```

Structure de données

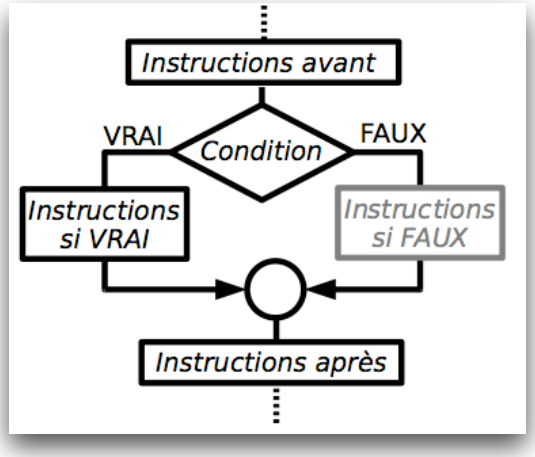
Séquence	Sélection simple ou avec alternative	Répétition (Boucle)
		

Séquence




```
print "Bonjour"
a = 2
a = a + 2
print a
```

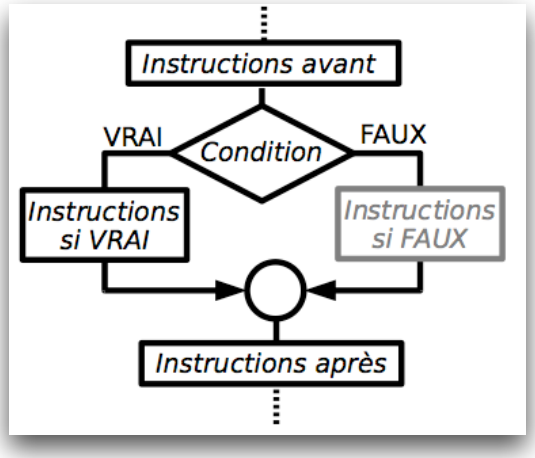

Sélection




```
if temperature < 20:
    print "il fait frais"
```

```
if temperature < 20:
    print "il fait frais"
else:
    print "il fait chaud"
```

Sélection

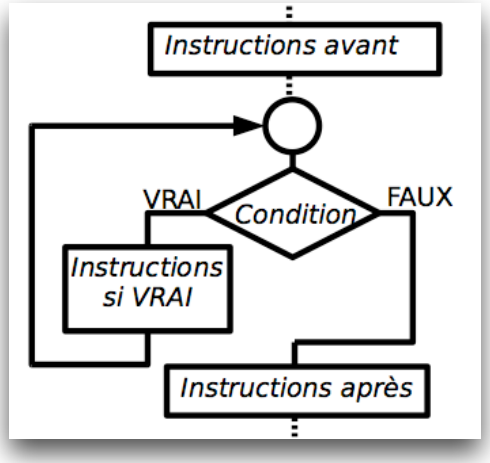


[!!!] Attention à la tabulation

```
if temperature < 20:
    print "il fait frais"
```

```
if temperature < 20:
    print "il fait frais"
else:
    print "il fait chaud"
```


Séquence




```
i = 0
while i<10:
    print "blabla"
```

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

```
R = raw_input("Combien font  3x4 ? ")
R = int(R)

while R!=12:
    print "Faux"
    if R<12:
        print "Essayez une valeur plus grande"
     else:
        print "Essayez une valeur plus petite"
     R = raw_input( "Encore : 3x4 ? ")
     R = int(R)

 print "Bravo !"
```


Les fonctions

- Fonctions
  - Certaines sont disponibles par défaut
    - range()
  - Certaines sont chargés via les librairies
    - import math
    - math.sin(1)
  - Certaines sont définies par vos soins
    - def *nom_de_ma_fonction*(*argument(s)*): # argument(s facultatif(s)
      instructions
      return *variable(s)* # retour facultatif

- Avantages: code modulaire et réutilisable


```
def disBonjour():
    print "Bonjour"
disBonjour()
```

Bonjour

```
def disBonjour(nom):
    print "Bonjour " + nom
disBonjour("Nicolas")
```

Bonjour Nicolas

```
def disBonjour(nom):
    print "Bonjour " + nom
disBonjour(raw_input("Qui es-tu? "))
```

Qui es-tu? <*nom*>
Bonjour <*nom*>

```
def carre(x):
    return x**2
print carre(3)
```

9

```
def exp(x,y):
    return x**y
print exp(3,2)
```

9

```
def exp2(x,y):
    return x**y, y**x
print exp2(3,2)
```

(9, 8)

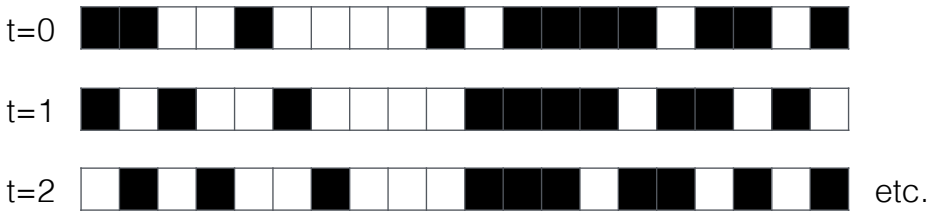
Etude de cas

- Compter les occurrences de groupe de lettres
  - lire une entrée clavier, dictionnaire

Bonjour, je compte les lettres d'une phrase.
Entrez une phrase: *un et un font deux*
{'e': 2, 'd': 1, 'f': 1, 'o': 1, 'n': 3, 'u': 3, 't': 2, 'x': 1} ← *lettres seules*
{'on': 1, 'ux': 1, 'de': 1, 'un': 2, 'eu': 1, 'et': 1, 'nt': 1, 'fo': 1} ← *mots de 2 lettres*

- Un automate cellulaire: l'embouteillage
  - liste 1D, boucle, affichage

si case libre : avance avec p=1 ← *arbitraire*
sinon : attend



compteurdelettres.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

print 'Bonjour, je compte les lettres d\'une phrase.'
phrase = raw_input('Entrez une phrase: ')

dico = {}

# compte le nombre d'occurrences de chaque lettre

for i in range(len(phrase)):
        if phrase[i] != ' ':
                if dico.has_key(phrase[i]) == True:
                        dico[phrase[i]] = dico[phrase[i]] + 1
                else:
                        dico[phrase[i]] = 1

print dico

# compte le nombre d'occurrence de chaque mot de deux lettres

dico = {}

for i in range(len(phrase)-1):
        if phrase[i] != ' ' and phrase[i+1] != ' ':
                if dico.has_key(phrase[i:i+2]) == True:
                        dico[phrase[i:i+2]] = dico[phrase[i:i+2]] + 1
                else:
                        dico[phrase[i:i+2]] = 1

print dico
```


nicolas.bredeche@upmc.fr

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import random

ligne = []

for i in range (20):
	if random.random() < 0.5:
		ligne.append('o') # voiture
	else:
		ligne.append('_') # route

for j in range(30):

	# affichage de l'état courant

	for i in range(len(ligne)):
		print ligne[i],
	print

	# mise a jour

	for i in range(len(ligne)-1, -1, -1):
		if ligne[i] == 'o':
			if ligne[(i+1)%len(ligne)] == '_' and random.random()<0.5:
				ligne[i] = '_'
				ligne[(i+1)%len(ligne)] = 'o'
			else:
				ligne[i] = 'o' # inutile, car ne bouge pas
		else:
			ligne[i] = '_'  # inutile, car rien a faire
```


Fin du cours

Pour aller plus loin (accès gratuit, Python 2.x ou 3.x):

http://python.developpez.com/tutoriels/cours-python-uni-paris7/

http://inforef.be/swi/python.htm

…et plein d'autres ressources sur Internet