

# Informatique pour le calcul scientifique

Alexandre Pinlou

[Alexandre.Pinlou@umontpellier.fr](mailto:Alexandre.Pinlou@umontpellier.fr)

Polytech' Montpellier

MI4 2017-2018



# Plan

## Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Structures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

IDE Spyder : le débogueur

Les dictionnaires

Les fichiers

Expressions régulières

# Introduction : le langage Python

- Créé par Guido van Rossum en 1991 (Google puis Dropbox) ;
- Nom en hommage à la troupe de comiques les « Monty Python » ;
- Multiplateforme (linux, windows, mac os, android, . . .) ;
- Libre et gratuit ;
- Doté de nombreuses librairies ;

## Deux manières d'interagir avec Python

- Langage de programmation interprété (machine virtuelle) par opposition aux langages compilés.

Interaction avec un interprète de commandes :

```
$ python3.6
Python 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016,17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Saisie d'instructions → l'interprète Python répond directement.

```
>>> a = 4
>>> b = a + 2
>>> print(a,b)
4 6
>>>
```

- Saisie de l'ensemble des instructions dans un fichier (script) avec l'extension `.py`.

Mettre en première ligne l'entête suivant (+ droits d'exécution) :

```
#!/usr/bin/python3
```

# Jouons avec l'interprète de commande

```
$ python3.6
Python 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016,17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 42
42
>>> 1.5
1.5
>>> 22 + 20
42
>>> 3 * 5
15
>>> 10 / 3          # division réelle
3.3333333333333335
>>> 10 // 3         # division entière
3
>>> 10 % 3          # modulo (reste de la division entière)
1
>>> 10 ** 3         # 10 puissance 3
1000
```

# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Strucutures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

IDE Spyder : le débogueur

Les dictionnaires

Les fichiers

Expressions régulières

# IDE Spyder : description

## L'explorateur

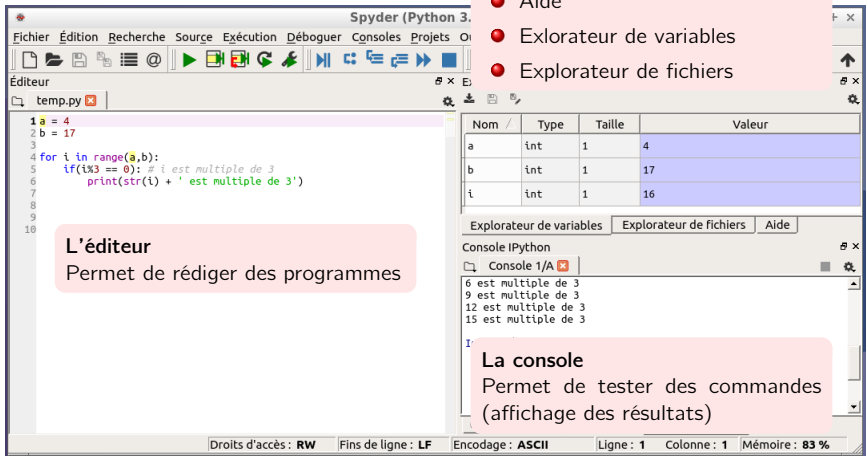
- Aide
- Explorateur de variables
- Explorateur de fichiers

## L'éditeur

Permet de rédiger des programmes


## La console

Permet de tester des commandes  
(affichage des résultats)



# IDE Spyder : avantages

Cet environnement facilite la rédaction d'un programme en :

- imposant des indentations lorsque c'est nécessaire ;
- mettant en couleur les fonctions, les mots clés et les chaînes de caractères (entre guillemets) ;
- proposant une aide pour la gestion des parenthèses ;
- indiquant par un panneau  les erreurs éventuelles de syntaxe ;
- affichant une aide contextuelle pour l'utilisation des fonctions Python.

Possibilité d'ouvrir plusieurs fichiers `.py` dans l'éditeur.



# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Strucutures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

IDE Spyder : le débogueur

Les dictionnaires

Les fichiers

Expressions régulières

# Variables : affectation

- Syntaxe : `nom = valeur`  
`nom` peut contenir lettre, chiffre, `_` (ne commence pas par un chiffre).  
Python sensible à la casse (`nom`  $\neq$  `Nom`).
- Exemples :

```
>>> longueur = 12
>>> longueur
12
>>> largeur = 10
>>> aire = longueur * largeur
>>> print(aire)
120
>>> a, b = 1, 2
>>> a
1
>>> b
2
>>> a = b = 3
>>> a
3
>>> b
3
>>> a = a + 1
>>> a
4
>>> a += 2
>>> a
6
```

```
>>> debut = 'Je_maitrise'
>>> fin = "le_Python"
>>> print(debut + '_' + fin + '.')
Je maitrise le Python.
```

# Types de données

- Types numériques : `int`, `float`, `complex`.  
Opérations : `+`, `-`, `*`, `/`, `//`, `%`.
- Booléens : `bool` sous-type de `int` ; val : `True` (ou 1), `False` (ou 0).  
Opérations : `and`, `or`, `not`. Comparaisons : `<`, `<=`, `>`, `>=`, `==`, `!=`.
- Chaînes de caractères : `str` (encadrées par apostrophes ou guillemets)  
Ce sont des **objets** → manipulation avec la notation « pointée »

```
>>> a = "Hello_world!"
>>> b = 'Hello_world!'
>>> print(a)
Hello world!
>>> print(b)
Hello world!
# antislash : caractère d'échappement
>>> c = 'aujourd\'hui'
>>> print(c)
aujourd'hui
>>> len(c)
11
```

```
>>> print(c.upper())
AUJOURD'HUI
# il existe aussi lower
>>> mois = 'janvier,février,mars'
>>> a,b,c = mois.split(',')
>>> print(a)
janvier
>>> print(b)
février
>>> print(c)
mars
```

# Types de données

- Connaître le type d'une variable : `type(var)`
- Tester le type d'une variable : `isinstance(var, type)`

```
>>> a = "Hello_world!"
>>> type(a)
<class 'str'>
>>> a = 3
>>> type(a)
<class 'int'>
>>> isinstance(a, int)
True
>>> isinstance(a, float)
False
```

# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Structures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

IDE Spyder : le débogueur

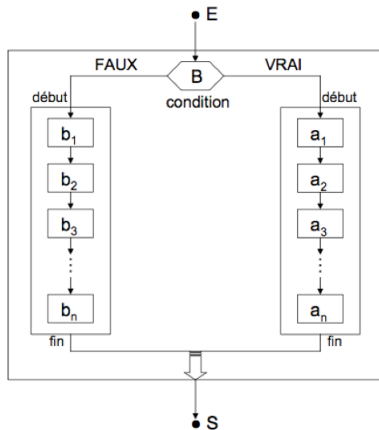
Les dictionnaires

Les fichiers

Expressions régulières

# Structures conditionnelles

- La conditionnelle est une instruction ;
- Elle permet de faire le **choix** entre différentes séquences d'instructions suivant une **condition**.



# Conditionnelle

- La conditionnelle manipule une expression booléenne (la condition);
- Syntaxe :

## Sans alternative

```
if cond :  
    inst1  
    inst2
```

## Avec alternative

```
if cond :  
    inst1  
    inst2  
else :  
    inst3  
    inst4  
    inst5
```

- Exécution :
  - ▶ La condition (expression) `cond` est évaluée et donne une valeur `v` ;
  - ▶ Si la valeur `v` est égale à `True` , alors l'instruction (ou le bloc d'instructions) `inst1` est exécuté ;
  - ▶ Sinon, si la valeur `v` est égale à `False`, alors :
    - dans le cas « sans alternative », aucune instruction n'est exécutée ;
    - dans le cas « avec alternative », l'instruction (ou le bloc d'instructions) `inst2` est exécuté.

# Structures conditionnelles

Tester le dépassement d'une valeur de seuil :

seuil.py

```
#!/usr/bin/python3
v = input('Entrez une vitesse de vol : ')
v = int(v) # conversion en entier
seuilDecrochage = contraintes(x,y,z,v)
if v < seuilDecrochage + 25:
    print('Vitesse trop faible !')
    diff = seuilDecrochage - v
    print('Augmentez votre vitesse de ' + str(diff) + ' km/h')
```

Terminal

```
mi4@polux:~> ./seuil.py
Entrez une vitesse de vol : 132
Vitesse trop faible !
Augmentez votre vitesse de 35 km/h
```



# Structures conditionnelles

Tester si un nombre est positif :

positif.py

```
#!/usr/bin/python3
x = input('x=')
x = int(x)
if x >= 0:
    print(str(x) + ' est positif')
else:
    print(str(x) + ' est négatif')
```

Terminal

```
mi4@polux:~> ./ positif.py
x = -2
-2 est négatif
```

# Structures conditionnelles ... en cascade

## Enchaînement de conditionnelles

- Syntaxe :

```
if cond1:  
    inst1  
elif cond2:  
    inst2  
elif cond3:  
    inst3  
else:  
    inste
```

- Exécution :

- ▶ La condition (expression) `cond1` est évaluée et donne une valeur  $v_1$  ;
- ▶ Si la valeur  $v_1$  est égale à `True`, alors l'instruction (ou le bloc d'instructions) `inst1` est exécuté ;
- ▶ Sinon on évalue la conditionnelle :  
`if cond2 inst2 elseif cond3 inst3 else inste end.`

# Structures conditionnelles

Tester si un nombre est dans 1 des intervalles  $]-\infty, -1[$ ,  $[-1, 1]$ ,  $]1, +\infty[$  :

intervalle.py

```
#!/usr/bin/python3
x = input('x=')
x = float(x)
if x < -1:
    print(str(x) + ' est strictement inférieur à -1')
elif x <= 1:
    print(str(x) + ' est compris entre -1 et 1')
else:
    print(str(x) + ' est strictement supérieur à 1')
```

Terminal

```
mi4@polux:~> ./intervalle.py
x = 0.5
0.5 est compris entre -1 et 1
```

# Structures conditionnelles

TP Python / Exercice 1

TP Python / Exercice 2

# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Structures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

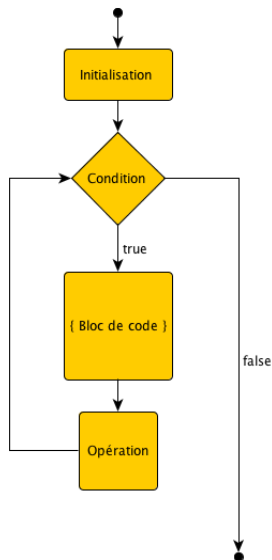
IDE Spyder : le débogueur

Les dictionnaires

Les fichiers

Expressions régulières

# Principe d'une structure répétitive (boucle)



On veut répéter une séquence d'instructions

- soit tant qu'une condition est vérifiée,
- soit un certain nombre de fois (fixé au départ),

# La boucle « Pour »

Utilisée pour répéter une suite d'instructions **un nombre de fois fixé**.

Syntaxe :

```
for i in seq:  
    instruction 1  
    instruction 2  
    instruction 3
```

```
for i in range(n):  
    instruction 1  
    instruction 2  
    instruction 3
```

*range(n) : renvoie une  
« sequence » de 0 à  $n-1$*

1. La variable `i` va prendre successivement l'ensemble des valeurs de `seq` ;
2. Pour chacune des valeurs de `i`, l'ensemble des instructions présentes dans la boucle sont exécutées.

1. La variable `i` va prendre successivement les valeurs de 0 à  $n-1$  ;
2. Pour chacune des valeurs de `i`, l'ensemble des instructions présentes

# La boucle « Pour » : séquences d'entiers

seq1.py

```
#!/usr/bin/python3
for i in range(5):
    print(i)
```

Terminal

```
mi4@polux:~> ./seq1.py
0
1
2
3
4
```

seq2.py

```
#!/usr/bin/python3
for i in range(2,5):
    print(i, end='␣')
```

Terminal

```
mi4@polux:~> ./seq2.py
2 3 4
```

seq3.py

```
for i in range(2,11,3):
    print(i, end='␣')
```

Terminal

```
mi4@polux:~> ./seq3.py
2 5 8
```



# La boucle « Pour » : séquences de caractères

seq4.py

```
#!/usr/bin/python3
s = "Hello"
for i in s:
    print(i)
```

Terminal

```
mi4@polux:~> ./seq4.py
H
e
l
l
o
```

# La boucle « tant que »

Utilisée pour répéter une suite d'instructions **tant qu'une condition est vérifiée**.

Syntaxe :

```
while condition :  
    instruction 1  
    instruction 2  
    instruction 3
```

1. La `condition` est une expression logique renvoyant un booléen (True / False); elle est évaluée **avant** chaque tour de boucle.
2. Si la condition est **vraie**, l'ensemble des instructions présentes dans la boucle sont exécutées;
3. L'étape 2 se répète tant que la condition reste **vraie**.
4. Lorsque la condition devient **fausse**, on exécute l'instruction qui se trouve après la boucle.

## La boucle « tant que » : Quelques remarques

- La `condition` implique (presque) toujours des **variables**.
- Les instructions présentes au sein de la boucle font évoluer la/les variable(s) de la condition.

### Exemple

Trouver le plus petit facteur premier.

facteur.py

```
#!/usr/bin/python3
x = int(input("x_="))
i = 2
while x % i != 0:
    i += 1
print(i)
```

Terminal

```
mi4@polux:~> ./facteur.py
x = 25
5
```

# La boucle « tant que »

## Exemple

Trouver le plus petit entier  $n$  tel que  $2^n > 132$ .

puissance.py

```
#!/usr/bin/python3
cible = 132
n = 0
while 2**n <= cible:
    print('2^' + str(n) + ' <= ' + str(cible))
    n += 1 # équivalent à n = n + 1
print(str(n) + ' plus petit entier t.q. 2^' + str(n) + ' > ' + str(cible))
```

Terminal

```
mi4@polux:~> ./puissance.py
2^0 <= 132
2^1 <= 132
2^2 <= 132
2^3 <= 132
2^4 <= 132
2^5 <= 132
2^6 <= 132
2^7 <= 132
8 plus petit entier t.q. 2^8 > 132
```

# Structures conditionnelles

TP Python / Exercice 3

TP Python / Exercice 4

# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Strucutures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

IDE Spyder : le débogueur

Les dictionnaires

Les fichiers

Expressions régulières

# Sous-programmes : procédures / fonctions

- Intérêt : un code utilisé plusieurs fois sera écrit une **unique** fois
- Définition :

```
def nom(arg1, arg2, ..., argn):  
    instruction1  
    instruction2  
    instruction3
```

- Utilisation :

```
a = 36  
nom(3, 6.5, a, 'seuil')
```

- Dans le cas d'une **fonction** : contient le mot-clé `return`
- Les variables d'une fonction sont **locales** ;
- Modifier les variables globales : mot-clé `global` (en général, on évite).

# Exemples de fonctions et procédures

- Fonction (avec `return`) :

racine.py

```
#!/usr/bin/python3

from math import *

def racines_reelles(a,b,c):
    delta=b**2 - 4*a*c
    x1 = (-b-sqrt(delta))/(2*a)
    x2 = (-b+sqrt(delta))/(2*a)
    return x1,x2

res=input('Saisir 3 valeurs (séparées par des espaces) : ')
x,y,z=res.split()
r1,r2=racines_reelles(float(x),float(y),float(z))
print(r1,r2)
```

Terminal

```
mi4@polux:~> ./racine.py
Saisir 3 valeurs (séparées par des espaces) : 2 4 1
-1.7071067811865475 -0.2928932188134524
```



# Exemples de fonctions et procédures

- Procédures (sans/avec effet de bord) :

sans\_effet\_bord.py

```
#!/usr/bin/python3
x = 1
y = 2
def increment(y):
    x = x + y
    y = y + 1
increment(y)
print(x,y)
```

effet\_bord.py

```
#!/usr/bin/python3
x = 1
y = 2
def increment(y):
    global x
    x = x + y
    y = y + 1
increment(y)
print(x,y)
```

Terminal

```
mi4@polux:~> ./sans_effet_bord.py
1 2
```

Terminal

```
mi4@polux:~> ./effet_bord.py
3 2
```

# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Structures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

IDE Spyder : le débogueur

Les dictionnaires

Les fichiers

Expressions régulières

# Importation d'un module

- Module = fonctions / variables stockées dans un fichier
- Nombreux modules disponibles (`math`, `numpy`, `scipy`, ...)
- Ajout d'un module :
  - `import <module>`
  - `from <module> import <fonction/variable>`

```
>>> sqrt(16)
NameError: name 'sqrt' is not defined
>>> math.sqrt(16)
NameError: name 'math' is not defined
>>> import math
>>> math.sqrt(16)
4
>>> sqrt(16)
NameError: name 'sqrt' is not defined
>>> from math import sqrt
>>> sqrt(16)
4
>>> from math import *
>>> pi
3.141592653589793
```

```
>>> import matplotlib.pyplot
>>> matplotlib.pyplot.show()
>>> from matplotlib.pyplot import show
>>> show() # fonctionne mais ...
>>> import matplotlib.pyplot as plt
>>> plt.show()
```

# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Structures répétitives

Procédures / Fonctions

Importation de modules

**Les séquences**

IDE Spyder : le débogueur

Les dictionnaires

Les fichiers

Expressions régulières

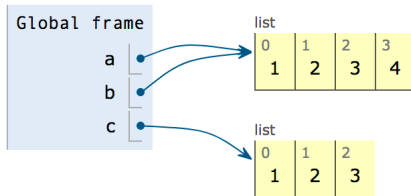
# Les séquences

- Chaînes de caractères
- Objets créés par la fonction `range`
- Listes :
  - ▶ création d'une liste vide : `l = []` ou `l = list()`
  - ▶ création d'une liste pré-remplie : `l = [1, 2, 3]`
  - ▶ insertion d'un élément à la fin : `l.append(4)`
  - ▶ insertion d'un élément à une position donnée : `l.insert(1,5)`
  - ▶ suppression du dernier élément : `l.pop()`
  - ▶ suppression d'un élément à une position donnée : `l.pop(2)`
- Tuples : `a,b,c` ou `(a,b,c)` (séquence non modifiable)
- Opérations sur les séquences :
  - ▶ Longueur : `len(s)` ;
  - ▶ Appartenance : `x in s`, `x not in s` (`x` élément, `s` séquence) ;
  - ▶ Concaténation : `s + t`, `s * k` (`s,t` séquences, `k` entier) ;
  - ▶ Accès par indice : `s[i]` (commence à 0) ;
  - ▶ Parties : `s[d:f]`, `s[d:]`, `s[:f]`, `s[d:f:i]`.

# Les listes : quelques remarques

- Les listes sont des **objets**

```
>>> a = [1,2,3]
>>> b = a
>>> c = a[:] # ou c = list(a)
>>> a.append(4)
>>> print(a)
[1,2,3,4]
>>> print(b)
[1,2,3,4]
>>> print(c)
[1,2,3]
```



- On peut **itérer** les listes

```
>>> lettres = ['a','e','i','o','u']
>>> n = len(lettres)
>>> for i in range(n):
...     print(lettres[i])
a
e
i
o
u
```

```
>>> lettres = ['a','e','i','o','u']
>>> for v in lettres:
...     print(v)
a
e
i
o
u
```

# Séquences et fonctions

TP Python / Exercice 5

TP Python / Exercice 6

# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Structures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

IDE Spyder : le débogueur

Les dictionnaires

Les fichiers

Expressions régulières



# IDE Spyder : le débogueur

- Vos programmes comportent parfois des erreurs.
- Spyder intègre un module de déboguage :
  - ▶ Exécution de votre code pas à pas depuis le début ou à partir d'un point d'arrêt.
  - ▶ Suivi de l'évolution de valeurs des variables.

# IDE Spyder : le débogueur

Lubuntu17 [Running]

**Spyder (Python 3.5)**

Fichier Édition Recherche Source Exécution Débugger Consoles Projets Outils Affichage Aide

Éditeur temp.py

```
1 a = 4
2 b = 17
3
4 for i in range(a,b):
5     if(i%3 == 0): # i est multiple de 3
6         print(str(i) + ' est multiple de 3')
7
```

Explorateur de variables

Nom	Type	Taille	Valeur
a	int	1	4
b	int	1	17
i	int	1	6

Explorateur de vari... Explorateur de fi... A...

Console IPython

Console 1/A

```
temp.py(4)<module>()
2 b = 17
3
----> 4 for i in range(a,b):
5     if(i%3 == 0): # i est multiple de
3
6         print(str(i) + ' est multiple
de 3')

ipdb>
ipdb>
```

Console Python Historique Console IPython

Droits d'accès : RW Fins de ligne : LF Encodage : ASCII Ligne : 4 Colonne : 1 Mémoire : 90 %

osboxes@osb... Spyder (Python 3.5) MACINTOSH

# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Structures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

IDE Spyder : le débogueur

Les dictionnaires

Les fichiers

Expressions régulières

# Dictionnaires

- Chaînes, listes, tuples : séquences ordonnées (accès par index).
- Dictionnaires :
  - ▶ Semblables aux listes (modifiables, éléments de n'importe quel type).
  - ▶ Les éléments ne sont pas ordonnés.
  - ▶ Accès aux éléments par une **clé** qui pourra être alphabétique, numérique, ou même d'un type complexe.

# Dictionnaires

```
>>> d = {} # création d'un dictionnaire vide
>>> effectifs = {'mi3' : 26, 'mi4' : 29, 'mi5' : 19} # création non vide
>>> d['computer'] = 'ordinateur' # ajout d'élément dans le dictionnaire
>>> d['mouse'] = 'souris'
>>> d['keyboard'] = 'clavier'
>>> print(d)
{'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
>>> print(d['keyboard']) # accès par clé
clavier
>>> d.pop('keyboard') # suppression par clé
clavier
>>> print(d)
{'computer': 'ordinateur', 'mouse': 'souris'}
>>> d['mouse'] = 'mulot' # modification
>>> print(d)
{'computer': 'ordinateur', 'mouse': 'mulot'}
```

# Dictionnaires : les méthodes importantes

- `keys` : retourne la liste des clés
- `values` : retourne la liste des valeurs
- `in` : retourne vrai/faux si la clé est présente
- `items` : retourne la liste de tuple (clé, valeur)

```
>>> d = {'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
>>> print(d.keys())
['computer', 'keyboard', 'mouse']
>>> print(d.values())
['ordinateur', 'clavier', 'souris']
>>> print('keyboard' in d)
True
>>> print('screen' in d)
False
>>> print(d.items())
[('computer', 'ordinateur'), ('keyboard', 'clavier'), ('mouse', 'souris')]
```

# Dictionnaires : Les parcours

```
>>> d = {'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}

>>> for cle in d:      # équivalent à      for cle in d.keys():
...     print(cle)
computer
keyboard
mouse

>>> for val in d.values():
...     print(val)
ordinateur
clavier
souris

>>> for cle, val in d.items():
...     print(cle + "=>" + val)
computer => ordinateur
keyboard => clavier
mouse => souris
```

# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Structures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

IDE Spyder : le débogueur

Les dictionnaires

**Les fichiers**

Expressions régulières



# Fichiers

- `open` : ouverture d'un fichier (lecture, écriture, ajout)
- `write` : écrire dans un fichier
- `readline` : lecture d'une ligne (avec caractère de fin de ligne)
- `close` : fermer le fichier

```
>>> fic = open('bonjour.txt','r') # ouverture en lecture
>>> fic.readline() # lecture ligne par ligne
'Hello_MI4!'
>>> fic.readline()
"Python, un langage d'avenir"
>>> fic.readline()
''
>>> fic.close() # fermeture du fichier
>>> fic = open('fichier.txt','w') # ouverture en écriture
>>> fic.write('Test\n') # écrase le contenu
>>> fic.close() # fermeture du fichier
>>> fic = open('fichier.txt','a') # ouverture en écriture (ajout)
>>> fic.write('Test\n') # ajoute à la fin
```

bonjour.txt

Hello MI4!

Python, un langage d'avenir

# Fichiers : Boucle de lecture

- Un fichier est « itérable » à l'aide d'une boucle `for`

```
>>> fic = open('bonjour.txt', 'r')
>>> for ligne in fic:
...     print(ligne.rstrip('\n')) # rstrip : retire '\n' à la fin
Bonjour MI4 !
Python, un langage d'avenir
```

- C'est l'occasion de parler des fonctions de chaînes de caractères.
  - ▶ `strip` : Retire des extrémités la liste des caractères en paramètre
  - ▶ `lstrip` : Retire de l'extrémité gauche la liste des caractères en paramètre
  - ▶ `rstrip` : Retire de l'extrémité droite la liste des caractères en paramètre
  - ▶ `split` : Découpe la chaîne de caractères suivant le symbole en paramètre
  - ▶ `replace` : Remplace un motif par un autre dans la chaîne de caractères

# Chaînes de caractères

C'est l'occasion de parler des fonctions de chaînes de caractères.

- ▶ `strip` : Retire des extrémités la liste des caractères en paramètre
- ▶ `lstrip` : Retire de l'extrémité gauche la liste des caractères en paramètre
- ▶ `rstrip` : Retire de l'extrémité droite la liste des caractères en paramètre
- ▶ `split` : Découpe la chaîne de caractères suivant le symbole en paramètre
- ▶ `replace` : Remplace un motif par un autre dans la chaîne de caractères

```
>>> s = '  Bonjour!!!  '
>>> s.strip(' ')
'Bonjour!!!'
>>> s.strip(' !')
'Bonjour'
>>> s.lstrip(' !')
'Bonjour!!!  '
>>> s.rstrip(' !')
'  Bonjour'
>>> t = 'lundi;mardi;mercredi;jedi;vendredi'
>>> t.split(';')
['lundi', 'mardi', 'mercredi', 'jedi', 'vendredi']
```

# Plan

Introduction

L'IDE « Spyder »

Variables

Structures conditionnelles

Structures répétitives

Procédures / Fonctions

Importation de modules

Les séquences

IDE Spyder : le débogueur

Les dictionnaires

Les fichiers

Expressions régulières

# Expressions régulières

- On parle d'expressions régulières ou rationnelles ;
- Issues des théories des langages formels des années 1940 ;
- Notations plus ou moins standards (dépend un peu du langage) ;
- Module `re` : `import re` ;
- Recherche élaborée dans une chaîne de caractères ;
- Plusieurs opérations possibles : chercher, remplacer, etc.

# Expressions régulières : Rechercher un motif

```
>>> import re
>>> text='mi4_29'
>>> re.search('mi4',text)
<_sre.SRE_Match object; span=(0, 3), match='mi4'>
>>> re.search('mi3',text)
>>> if re.search('mi4',text):
...     print('Trouvé')
... else:
...     print('Pas trouvé')
```

Trouvé

# Expressions régulières : Rechercher un motif

```
>>> import re
>>> text='mi4_ig4_mat4_45'
>>> re.search('ig[0-9]',text)
<_sre.SRE_Match object; span=(10, 13), match='ig4'>
>>> re.search('mat...',text)
<_sre.SRE_Match object; span=(20, 26), match='mat4_'>
>>> re.search('ig.*',text)
<_sre.SRE_Match object; span=(10, 19), match='ig4_45'>
>>> re.search('[^,]*45',text)
<_sre.SRE_Match object; span=(9, 18), match='_ig4_45'>
>>> re.search('[^,]*45$',text)
<_sre.SRE_Match object; span=(19, 29), match='_mat4_45'>
>>> re.search('^ig',text)
>>>
```

# Expressions régulières : Récapitulatif

expression	matches...
abc	abc (that exact character sequence, but anywhere in the string)
^abc	abc at the <i>beginning</i> of the string
abc\$	abc at the <i>end</i> of the string
a b	either of a and b
^abc abc\$	the string abc at the beginning or at the end of the string
ab{2,4}c	an a followed by two, three or four b's followed by a c
ab{2,}c	an a followed by at least two b's followed by a c
ab*c	an a followed by any number (zero or more) of b's followed by a c
ab+c	an a followed by one or more b's followed by a c
ab?c	an a followed by an optional b followed by a c; that is, either abc or ac
a.c	an a followed by any single character (not newline) followed by a c
a\\.c	a.c exactly
[abc]	any one of a, b and c
[Aa]bc	either of Abc and abc
[abc]+	any (nonempty) string of a's, b's and c's (such as a, abba, acbabacaaa)
[^abc]+	any (nonempty) string which does <i>not</i> contain any of a, b and c (such as defg)
\\d\\d	any two decimal digits, such as 42; same as \\d{2}



# Expressions régulières : Groupes

```
>>> import re
>>> text='mi4_29'
>>> re.search('(.*)_*(.*)',text)
<_sre.SRE_Match object; span=(0, 8), match='mi4_29'>
>>> a = re.search('(.*)_*(.*)',text)
>>> a.group(0)
mi4_29
>>> a.group(1)
mi4
>>> a.group(2)
29
```

# Expressions régulières : Rechercher un motif multiple

```
>>> import re
>>> text='mi3_26,mi4_29,mi5_27'
>>> re.search('mi',text)
<_sre.SRE_Match object; span=(0, 2), match='mi'>
>>> re.finditer('mi',text)
<callable_iterator object at 0x10cde2e48>
>>> for i in re.finditer('mi[0-9]_[0-9]*',text):
...     print(i)
<_sre.SRE_Match object; span=(0, 8), match='mi3_26'>
<_sre.SRE_Match object; span=(10, 18), match='mi4_29'>
<_sre.SRE_Match object; span=(20, 28), match='mi5_27'>
>>> for i in re.finditer('(mi[0-9])_([0-9]*)',text):
...     print(i.group(0) + '_=>' + i.group(1) + '/' + i.group(2))
mi3=26 => mi3 / 26
mi4=29 => mi4 / 29
mi5=27 => mi5 / 27
```

# Fichiers et expressions régulières

TP / Exercice 7