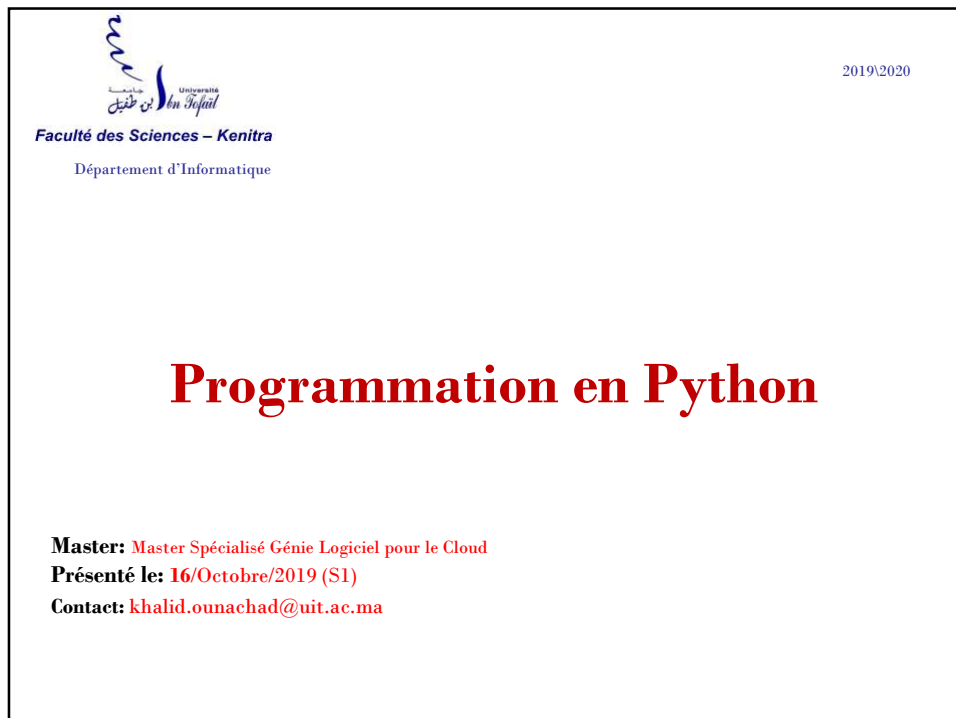




1



2

Les éléments de base en langage Python

- Introduction,
- Types et opérateurs de base ,
- Instructions de base ,
- Choix ,
- Boucles,
- Fonctions.

3

- Introduction,

4

Introduction

- Python est développé depuis 1989 par **Guido van Rossum** et de nombreux contributeurs bénévoles.
- Python est un langage :
 - *gratuit*, et *open source*
 - *portable*,
 - *dynamique*,
 - *extensible*,
 - qui permet (sans l'imposer) une approche modulaire et *orientée objet* de la programmation.
 - Graphique...ect



5

Introduction



- Python 3.x n'est pas une simple amélioration ou extension de Python 2.x.



6

Introduction

Les différentes implémentations:

- **CPython** Implémentation de base basé sur le langage C
ANSI
- **Jython** Implémentation permettant de mixer Python et
java dans la même JVM
- **IronPython** Implémentation permettant d'utiliser
Python pour Microsoft .NET
- **PyPy** Implémentation de Python en Python
- **CLPython** Implémentation de Python en Common Lisp

7

Introduction

Que peut-on faire avec Python ?

- **bases de données**
MySQL, PostgreSQL, Oracle, ...
- **réseaux**
TwistedMatrix, PyRO, ...
- **représentation graphique**
gnuplot, matplotlib, VTK, ...
- **calcul scientifique**
numpy, scipy, sage, ...
- **Traitement d'image:**
open cv, pil
- ...

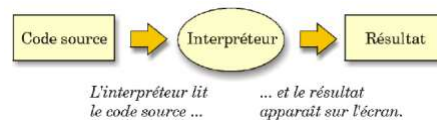
8

Introduction

Mon code source ?

Il existe 2 techniques principales pour effectuer la traduction en langage machine de mon code source :

- **Interprétation**



- **Compilation**



Figures tirées du livre « Apprendre à programmer avec Python »

9

Introduction

Mon code source Python ?



Figure tirée du livre « Apprendre à programmer avec Python »

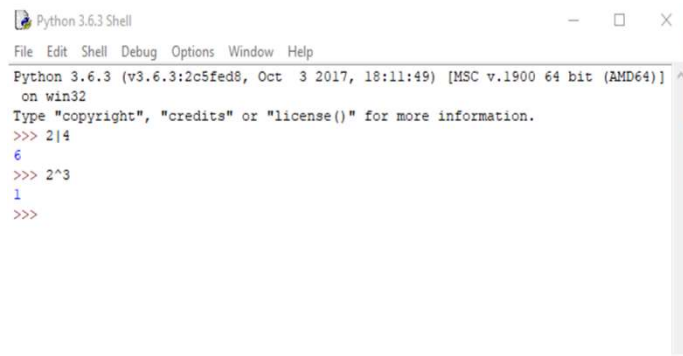
- + interpréteur permettant de tester n'importe quel petit bout(script) de code,
- peut être lent.

10

Introduction

L'interpréteur Python?

Sous Windows :



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2|4
6
>>> 2^3
1
>>>
```

11

- Types et opérateurs de base

12

Types et opérateurs de base

- Mots réservés python:

and as assert break class continue def
del elif else except False finally for from global
if import in is lambda None nonlocal
not or pass raise return True try while
with yield

13

Types et opérateurs de base

- Types de base: simples

Valeur	Type python	exemple
$\in \mathbb{Z}$	int	12,+20,-20
$\in \mathbb{IR}$	float	12., 20.0, 1.23e-6
$\in \mathbb{C}$	complex	2+9j, 2+9J
caractère	Chr	'1','f'
Valeur logique	bool	True ,False

Type permet d'avoir le type ou la
classe du type

14

Types et opérateurs de base

- bases de numérotation et codage(conversions)

En python	exemple	Résultat
bin	bin(2)	0b10
oct	oct(12)	0o14
hex	hex(15)	0xf
int	int(0b10)	2
ord	ord('a')	97(code ASCII..)
chr	chr(97)	'a'

15

Types et opérateurs de base

- Affectation: =

```
>>> # ceci est un commentaire
```

```
>>> i = 30 # i vaut 30
```

```
>>> a, pi = True, 3.141596
```

```
>>> k = r = 2.156
```

16

Types et opérateurs de base

- **Affectation: chaîne de caractères**

```
>>> x = 'hello '

>>> y = "world!"

>>> z = '''hello
... world'''
```

17

Types et opérateurs de base

- **Opérateurs arithmétiques (ou et fonctions)**

opérateur	Syntaxe python
$X + y$	$X + y$
$X - y$	$X - y$
$X \times y$	$X * y$
X / y	X / y
x/y division euclidienne	$X // y$
x^y	$X ** y$, <code>pow(x,y)</code>
$ x $	<code>abs(x)</code>
Reste de la division euclidienne	<code>%</code>

18

Types et opérateurs de base

- Opérateurs abrégés

opérateur	Syntaxe python abrégée
$X = X + y$	$X += y$
$X = X - y$	$X -= y$
$X = X \times y$	$X *= y$
$X = X / y$	$X /= y$
$X = X // y$	$X //= y$
$X = X ** Y$	$X **= y$
$X = X \% y$	$X \% = y$

19

Types et opérateurs de base

- Opérateurs de comparaisons:

opérateur	Syntaxe python
= (égalité)	$==$
\neq	$!=$
\leq	$<=$
\geq	$>=$
$>$	$>$
$<$	$<$
5 est réel	5 is float
	is not

20

Types et opérateurs de base

- Opérateurs logiques:

opérateur	Syntaxe python
et	and
ou	or
non	not

- Opérateurs d'appartenance:

opérateur	Syntaxe python
€	in
∉	not i

21

Types et opérateurs de base

- Opérateurs binaire(la base binaire):

opérateur	Syntaxe python	Exemple
Et logique	&	>>> 2&3 2
Ou exclusif	^	>>> 2^3 1
Ou logique		>>> 2 3 3
Décalage à gauche	<<	>>> 2<<3 16
Décalage à droite	>>	>>> 2>>3 0

22

- **Instructions de base,**

23

Instructions de base

- **Lecture:**

- **Syntaxe:**

```
input(prompt=None, /)
```

Read a string from standard input. The trailing newline is stripped.

The prompt string, if given, is printed to standard output without a trailing newline before reading input.

If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError. On *nix systems, readline is used if available.

tirée de la documentation de « python3.6.3 »

- **Exemple:**

```
X=float(input(«donner un nombre réel: »))
```

24

Instructions de base

- **Affichage:**

- **Syntaxe:**

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

tirée de la documentation de « python3.6.3 »

- **Exemple:**

```
print(« le langage Python »)
print(2**10)
print(« resultat= »,5/20+30**8)
```

25

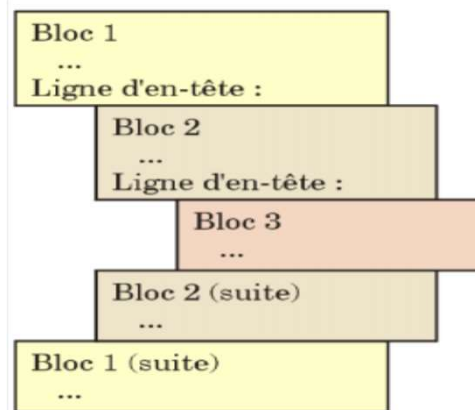
- **Choix(sélections),**

26

Choix

- **Indentation générale**

- **Fonctionnement par bloc**



Figures tirées du livre « Apprendre à programmer avec Python »

27

Choix

- **Simple:**

- **Syntaxe:**

if conditions :

blocs d'instructions

- **Exemple:**

a = -150

if a < 0:

 print ('a est négatif')

28

Choix

- **Double:**

- **Syntaxe:**

if conditions :

blocs d'instructions 1

else:

blocs d'instructions 2

- **Exemple:**

if a < 0:

print ('a est négatif')

else:

print('a est positif ')

29

Choix

- **Multiple:**

- **Syntaxe:**

if conditions1 :

blocs d'instructions 1

elif conditotions2 :

blocs d'instructions 2

elif conditions I :

...

else:

autres instructions

30

Choix

- **Multiple:**

- **Exemple:**

```
a = 10.  
if a > 0:  
    print( 'a est strictement positif' )  
    if a >= 10:  
        print( 'a est un nombre' )  
    else:  
        print( 'a est un chiffre' )  
    a += 1  
elif a is not 0:  
    print( 'a est strictement négatif' )  
else:  
    print( 'a est nul' )
```

31

- **Boucles(Répétitions),**

32

Boucles

- **While:**

- **syntaxe:**

`while` conditions:
instructions

- **Instructions particuliers:**

break : sort de la boucle

continue : remonte au début de la boucle,

pass : ne fait rien,

33

Boucles

- **While:**

- **exemple: y est-il premier ?**

`x = y / 2`

`while x > 1:`

`if y % x == 0:`

`print (y, 'est facteur de', x)`

`break`

`x = x-1`

`print (y, 'est premier')`

34

Boucles

- **For:**

- **Syntaxe:**

```
for cible in séquence d'objets:
    bloc instructions
```

- **range:**

- **Syntaxe**

range(start,stop+1,step)

- **Exemple:**

range(6) donne la séquence(liste) [0,1,2,3,4,5]

range(1,6)----->[1,2,3,4,5]

range(1,6,3)----->[1,4]

35

Boucles

- **For:**

- **Exemple:**

```
prod = 1
for p in range(1, 10):
    prod *= p
    print(prod)
```

- **Exécution:**

```
1
2
6
24
120
720
5040
40320
362880
```

36

Boucles

- **remarque:**

- **zip** : permet de parcourir plusieurs séquences en parallèle
- **map** : applique une méthode sur une ou plusieurs séquences **Exemple:**

```
L1 = [1, 2, 3]
```

```
L2 = [4, 5, 6]
```

```
for (x, y) in zip(L1, L2):
```

```
    print( x, y, '--', x + y)
```

Exécution:

```
1 4 -- 5
```

```
2 5 -- 7
```

```
3 6 -- 9
```

37

- **Fonctions**

38

Fonctions

- **Fonction prédéfinies:**

- Syntaxe:

`from module import fct1,fct2,`

`from module import * # pour charger toutes les fonctions du module`

`import module`

Exemple

`from math import sin,sqrt`

`print("la racine carré de « ,x,"est",sqrt(x),"et sin("x,")="sin(x))`

Ou bien

`import math`

`print(math,sin(x))`

39

Fonctions

- **Quelques fonctions prédéfinies: math**

Fonction	Syntaxe python
$ x $	<code>fabs(x)</code>
\sqrt{x}	<code>sqrt(x)</code>
X^y	<code>pow(x,y)</code>
e^x	<code>exp(x)</code>
ln	<code>log</code>
log	<code>log10</code>
Log a base 2	<code>log2</code>
Sin,cos,tg	<code>sin,cos,tan</code>
sinh	<code>Sinh...</code>

40

Fonctions

- **Remarque fonctions prédéfinies:**
- **dir** permet de voir les objets et méthodes disponibles
- **help** permet d'avoir une aide
- **Exemple:**

```
>>> dir(math)
['_doc_', '_loader_', '_name_', '_package_', '_spec_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',
'log', 'log10', 'logip', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
>>> help(math.log2)
Help on built-in function log2 in module math:

log2(...)
|   log2(x)
|
|   Return the base 2 logarithm of x.
```

41

Fonctions

- **fonctions propres:**
- **Syntaxe:**

```
def NomFonction(arg1, arg2,... argN):
```

```
    ...
```

```
    bloc d'instructions
```

```
    ...
```

```
    return valeurs_resultats
```

42

Fonctions

- **fonctions propres:**

- **Exemple:**

```
def est_premier(n):  
    for i in range(2,n):  
        if n%i==0:  
            return False  
    return True
```

43

Fonctions

- **Fonction réursive:**

- **Exemple:**

```
def f(n):  
    if n==0:  
        return 1  
    else:  
        return n*f(n-1)
```

44

Fonctions

- **Fonction lambda :**

- **Syntaxe:**

lambda argument1,... argumentN : expression utilisant les arguments

Exemple :

```
f = lambda x, i : x**i
```

```
f(2, 4)
```

45

MERCI

46

Travaux Pratiques:

- TP1



جامعة ابن توفيل
UNIVERSITÉ IBN TOFÄÏL
Faculté des Sciences

MASTER D'UNIVERSITÉ SPÉCIALISÉ
... Informatique ...

**GÉNIE LOGICIEL POUR
LE CLOUD**

Exercez les métiers de demain


Business Intelligence

BIG DATA

Cloud Computing

Coordinateur
Pr. Moulay Youssef Hadi
Département d'informatique
hadiyoussef@gmail.com 0694 500 534

1



2019\2020

Faculté des Sciences – Kenitra

Département d'Informatique

P00 en Python

Master: Master Spécialisé Génie Logiciel pour le Cloud
Présenté le: 30/Octobre/2019 (S1)
Contact: khalid.ounachad@uit.ac.ma

2

POO en langage Python

- Introduction,
- Classe,
- Encapsulation,
- Heritage,
- Polymorphism et surcharge ,

3

- Introduction,

4

Introduction

- Dans les années 1970, la programmation se développe et de nouvelles technologies logicielles apparaissent: mode multi use, interfaces graphiques,....
- Taille des programmes augment: 10000 lignes de code pour un logiciel de bonne taille dans les années 1970(et des millions de nos jours)
- La **crise du logiciel** des années 70 est provoquée par:
 - L'impossibilité de maitriser la sureté des logiciels(pannes++)
- De bonnes **techniques de développement logiciel** deviennent **nécessaire**.

5

Introduction

- Programmation objet est inventée dans les années 1970 (Extention de la programmation procédurale)
- Idée: concevoir les programmes comme des **objets** qui se **dialoguent**.
- Fin des années 60(Norvège): Simula 64 est le premier langage à utiliser ce paradigme(mais de façon limitée)
- Smalltalk: developpé par **Alan kay** 1971 et publié en 1980 est le premier vrai langage objet.
- **Python en 1991**

6

- **Class,**

7

Classe

Définition:Class

Une classe définit :

- une **Unité d'encapsulation** :
 - elle **regroupe la déclaration des attributs et la définition des méthodes** associées dans une même construction syntaxique
 - attributs = stockage d'information (état de l'objet).
 - méthodes = unités de calcul (sous-programmes : fonctions ou procédures).
 - C'est aussi un **espace de noms** : deux classes différentes peuvent avoir des membres de même nom
- un **TYPE** qui permet de :
 - créer des objets (la classe est un moule, une matrice) ;

Les méthodes et attributs définis sur la classe comme « unité d'encapsulation » pourront être appliqués à ses objets

8

Classe

Syntaxe:

```
class (superclass,...):
    donnee = valeur
    def methode(self,...):
        self.membre = valeur
```

Classe Etudiant	
Attributs :	
Nom : char[]	nom
Prenom : char[]	prenom
Age : int	a
Méthodes :	
Lecture-etudiant()	
Ecriture-etudiant()	
Calcul-age()	

```
class MaClasse:
    "Une classe simple pour exemple "
    i = 12345
    def f(self):
        return 'bon'
```

MaClasse.i : référence d'attribut valide ; renvoie un entier

MaClasse.f : référence d'attribut valide ; renvoie un objet
fonction

9

Classe: Objet

Définition: Objet

-Objet : donnée en mémoire qui est le représentant d'une classe.
• l'objet est dit **instance** de la classe .

-Un objet est caractérisé par :

un état : la valeur des attributs (x, y, etc.) ;

un comportement : les méthodes qui peuvent lui être appliquées

une identité : identifie de manière unique un objet (p.ex. son adresse en mémoire).

-Un objet est créé à partir d'une classe (en précisant les paramètres effectifs de son constructeur, sauf le premier, self)

-Les identités des objets sont conservées dans des variables (des noms)

10

Classe: Attributs

Définition: Attributs

On distingue :

- les **attributs d'instance** : spécifique d'un objet
- les **attributs de classe** : attaché à la classe et non aux objets

class A:

 ac = 10 # Attribut de classe

 def __init__(self, v):

 self.ai = v # ai attribut d'instance

11

Classe: Méthodes

Méthodes

-Une méthode d'instance est un sous-programme qui

exploite l'état d'un objet (en accès et/ou en modification).

- Le premier paramètre désigne nécessairement l'objet. Par convention, on l'appelle **self**.

-Une méthode de classe est une méthode qui travaille sur la classe (et non l'objet).

- Elle est décorée **@classmethod**.
- Son premier paramètre est nommé **cls** par convention.

12

Classe: Méthodes

Méthodes d'instance/de classe

Exemple:

```
class A:
    nb = 0

    def __init__(self):
        print("creation objet de type A")
        A.nb = A.nb + 1
        print("il y en a maintenant ", A.nb)

    @classmethod
    def get_nb(cls):
        return A.nb

print("Partie 1 : nb objets = ", A.get_nb())
a = A()
print("Partie 2 : nb objets = ", A.get_nb())
b = A()
print("Partie 3 : nb objets = ", A.get_nb())
```

13

Classe: Méthodes

Méthodes

-Une méthode statique est une méthode définie dans l'espace de nom de la classe mais est indépendante de cette classe.

-Elle est décorée **@staticmethod**

Exemple:

class Date:

...

@staticmethod

def est_bissextile(annee):

return annee % 4 == 0 and (annee % 100 != 0
or annee % 400 == 0)

14

Classe: Méthodes

Méthodes spéciales:

`__init__(self, ...)` : le constructeur : méthode d'initialisation nécessairement appelée quand on crée un objet. Voir aussi `__new__`. C'est le constructeur qui devrait définir les attributs d'instance de la classe.

`__new__(cls, ...)` : méthode implicitement statique qui crée l'objet (et appelle `__init__`).

`__del__(self)` : le destructeur, appelé quand une instance est sur le point d'être détruite.

`__bool__(self)` : utilisée quand l'objet est considéré comme

15

Classe: Méthodes

Méthodes spéciales:

`__bool__(self)` : utilisée quand l'objet est considéré comme booléen et avec la fonction prédéfinie `bool()`

. ... Voir: <https://docs.python.org/3/reference/datamodel.html#special-method-names>

16

- **Constructeur,**

17

Classe: Constructeur

Initialiseur: Constructeur

class Robot:

""" Robot qui sait avancer d'une case et pivoter à droite de 90°. Il est repéré par son abscisse x, son ordonnée y et sa direction. """

def __init__(self, x, y, direction): """ Initialiser le robot self à partir de sa position (x, y) et sa direction. """

self.x = x

self.y = y

self.direction = direction

18

Classe: Constructeur

Exemple: instance

```
>>> class Complexe:
...     def __init__(self, reel, imag):
...         self.r = reel
...         self.i = imag ...
>>> x = Complexe(3.0, -4.5)
>>> x.r, x.i
3.0, -4.5
```

19

- Encapsulation,

20

Classe: Encapsulation

Syntaxe: `_NomAtt` ou `__NomAtt` (de même pour les méthodes)

Accès:

• On utilise les getters et les setters:

- les **accesseurs** (en anglais accessor) qui fournissent des informations relatives à l'état d'un objet, c'est-à-dire aux valeurs de certains de ses attributs (généralement privés) sans les modifier ;
- les **mutateurs** (en anglais mutator) qui modifient l'état d'un objet, donc les valeurs de certains de ses attributs.

On rencontre souvent l'utilisation de noms de la forme

`get_XXXX()` pour les accesseurs et `set_XXXX()` pour les mutateurs, y

21

Classe: Encapsulation

Exemple:Accès

```
class Point:
    def __init__(self, x, y):
        self.__x = x
        self.__y = y

    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y

    def set_x(self, x):
        self.__x = x

    def set_y(self, y):
        self.__y = y

a = Point(3, 7)
print("a : abscisse =", a.get_x())
print("a : ordonnee =", a.get_y())
a.set_x(6)
a.set_y(10)
print("a : abscisse =", a.get_x())
print("a : ordonnee =", a.get_y())
```

22

Classe: Encapsulation

Exemple2: Autre façon

```
class Date:
    def __init__(self, jour, mois, annee):
        self.__jour = jour
        self.__mois = mois
        self.__annee = annee

    @property
    # accès en lecture à mois, comme si c'était un attribut
    def mois(self):
        return self.__mois

    @mois.setter
    # accès en écriture à mois, comme si c'était un attribut
    def mois(self, mois):
        if mois < 1 or mois > 12:
            raise ValueError
        self.__mois = mois
```

23

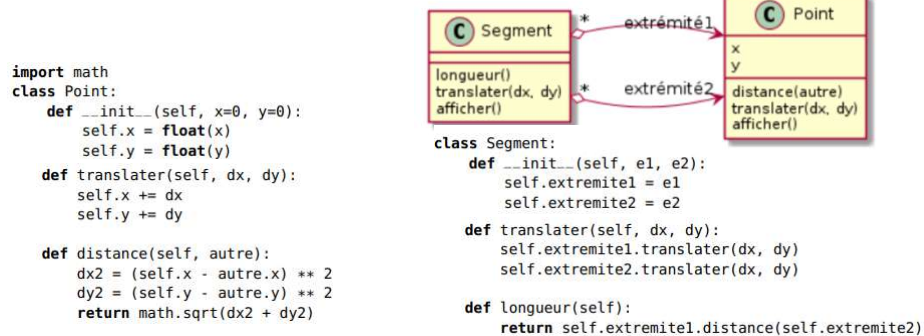
- Utilisation,

24

Classe: relation d'utilisation

Exemple où:

Une classe utilise une autre classe (en général, ses méthodes)



25

Classe: relation d'utilisation

Exemple:

```

def exemple():
    # créer les points sommets du triangle
    p1 = Point(3, 2)
    p2 = Point(6, 9)
    p3 = Point(11, 4)

    # créer les trois segments
    s12 = Segment(p1, p2)
    s23 = Segment(p2, p3)
    s31 = Segment(p3, p1)

    # créer le barycentre
    sx = (p1.x + p2.x + p3.x) / 3.0
    sy = (p1.y + p2.y + p3.y) / 3.0
    barycentre = Point(sx, sy)
  
```

26

- Héritage,

27

Classe: Héritage

Principe: définir une nouvelle classe par spécialisation d'une (ou plusieurs) classes existantes.

Exemple:

Définir une classe PointNommé sachant que la classe Point existe. La classe

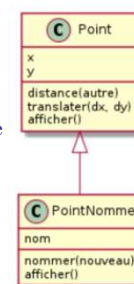
PointNommé :

hérite (récupère) tous les éléments de la classe Point

ajoute un nom et les opérations pour manipuler le nom

redéfinit la méthode afficher (pour afficher le nom et les coordonnées du point)

Point est la **super-classe**, PointNommé la **sous-classe**



28

Classe: Héritage

Exemple:

```
class PointNomme(Point):    # La classe PointNommé hérite de Point
    def __init__(self, nom, x=0, y=0):
        super().__init__(x, y) # initialiser la partie Point du PointNommé
        self.nom = nom         # un nouvel attribut

    def nommer(self, nouveau_nom):    # une nouvelle méthode
        self.nom = nouveau_nom
```

super() est recommandé pour appeler la méthode des super classes. ici

Ancienne solution : **Point.__init__(self, x, y)**

La classe **object** C'est l'ancêtre commun à toutes les classes.

Quand on ne précise aucune superclasse, la classe hérite implicitement de object

29

Classe: Héritage Multiple

Il suffit de lister toutes les super classes séparées par des virgules

class S(A, B, C): # la classe S hérite de A, B et C

```
class carnivore:
    def __init__(self,p):
        self._poidsViande = p
    def devorer(self):
        print("Je mange",self._poidsViande,"kilogs de steack par jour")

class herbivore:
    def __init__(self,p):
        self._poidsHerbe = p
    def brouter(self):
        print("Je mange",self._poidsHerbe,"kilogs de gazon par jour")

class omnivore(carnivore,herbivore):
    def __init__(self,pv,ph,h):
        carnivore.__init__(self,pv)
        herbivore.__init__(self,ph)
        self._humain = h

teddy = omnivore(10,5,False)
teddy.devorer()
teddy.brouter()

Je mange 10 kilogs de steack par jour
Je mange 5 kilogs de gazon par jour
```

30

- polymorphisme,

31

Classe: Polymorphisme

Rappel: mécanisme qui permet à une classe fille de redéfinir une méthode dont elle a hérité de sa classe mère, tout en gardant la même signature.

```
class rectangle:
    def __init__(self,x,y):
        self._x = x
        self._y = y
    def surface(self):
        return self._x*self._y

class paveDroit(rectangle):
    def __init__(self,x,y,z):
        super().__init__(x,y)
        self._z = z
    def surface(self):
        return 2*(self._x*self._y+self._x*self._z+self._y*self._z)

photo = rectangle(3,4)
print(photo.surface())
weston = paveDroit(3,4,10)
print(weston.surface())
```

Console

```
12
164
```

32

- **Surcharge des opérateurs**

33

Classe: Surcharge

Rappel: Surcharge des opérateurs: Lorsque l'on crée des classes, il est naturel de vouloir adapter les opérateurs usuels afin de pouvoir les appliquer sur des objets de nos propres classes.

Cette création de nouvelles versions s'appelle une **surcharge**.

```
class rectangle:
    def __init__(self,x,y):
        self.__x = x
        self.__y = y
    def surface(self):
        return self.__x*self.__y
    def __mul__(self, other):
        return rectangle(self.__x*other.__x,self.__y*other.__y)
    def __imul__(self, other):
        self = self*other
        return self

a = rectangle(3,4)
b = rectangle(2,5)
c = a*b
print(c.surface())
c *= b
print(c.surface())
```

Console

```
120
1200
```

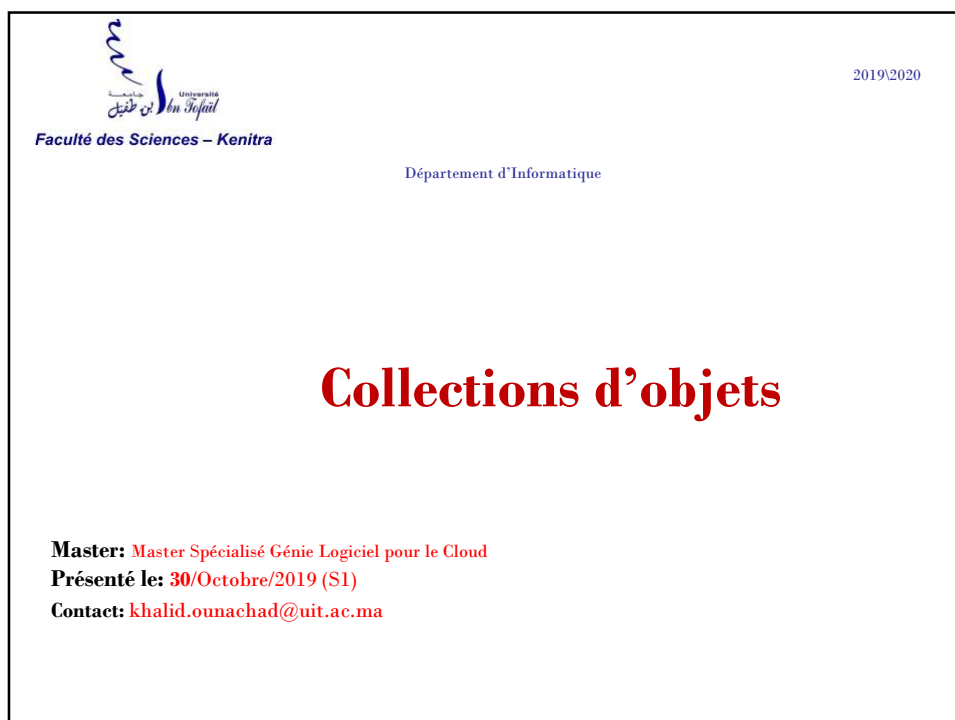
Surcharge des opérateurs * et *= dans la classe "rectangle"

On définit ici le résultat de la multiplication de deux rectangles comme étant le rectangle dont la largeur (resp. longueur) est le produit des largeurs (resp. longueurs) :

34



35



36

Collections d'objets

- Liste,
- Tuple,
- dictionnaire,
- Ensemble.

37

List

Exemples

- `[1,2,3,4,5,6]` # e n t i e r s
- `[3.14, 0.693, 2.78]` # f l o t t a n t s
- `[True, True, False]` # b o o l é e n s
- `['a', 'bra', 'ca', 'da', 'bra']` # c h a î n e s d e a r a t è r e s
- `[1, 1.609, 'bonjour', None]`
- `range(10), range(2, 10, 2)`

38

List

Listes vides

- `L=[]`
- `L1=list()`

Longueur: Le nombre d'éléments d'une liste est la longueur de la liste. La longueur de la liste **L** est donnée par **len(L)**.

- Pour une liste de longueur **N**, les indices valides varient entre $-N$ et $(N - 1)$ au sens large .
- L'utilisation d'un indice non valide provoque une erreur.

IndexError: list index out of range

39

List

Accès à un élément et modification

On considère une liste **L** de longueur **N**.

- L'indice du premier élément est nul.
- L'indice du dernier élément est égal à $(N - 1)$.
- L'élément d'indice **k** est **L[k]**.
- **L[i]=v** # modification du ième par v

Exemples:

```
jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
print(jour[2])
mercredi
jour[-3]=200;print(jour[-3])
200
```

40

List

Sous-listes:

On suppose que i et j sont positifs.

- La tranche $L[i:j]$ est la sous-liste

$$(L_k)_{i \leq k < j} \quad [L[i], L[i+1], \dots, L[j-1]]$$

- La tranche $L[i:]$ est la sous-liste

$$(L_k)_{i \leq k} = (L_k)_{i \leq k < N}$$

- La tranche $L[:j]$ est la sous-liste

$$(L_k)_{k < j} = (L_k)_{0 \leq k < j}$$

- La tranche $L[i:j:p]$ est la sous-liste de la tranche $L[i:j]$ constituée des éléments $L[k]$ pour lesquels l'indice k

vérifie $k = i \bmod p$.

- En particulier, $L[::-1]$ est une copie en miroir de L .

41

List

Concaténation

```
>>> sept_zeros = [0]*7; sept_zeros
```

```
[0, 0, 0, 0, 0, 0, 0]
```

```
>>> L1, L2 = [1, 2, 3], [4, 5]
```

```
>>> L1 + L2
```

```
[1, 2, 3, 4, 5]
```

Opérations algébriques

On suppose que les éléments de la liste L sont comparables : nombres ou chaînes de caractères.

min(L).

max(L).

sum(L). Si les éléments de la liste L sont des nombres, on obtient la somme de ces nombres avec l'instruction

42

List

Copie d'une liste **ATTENTION !**

```
>>> L = ['Dans', 'python', 'tout', 'est', 'objet']
>>> T = L
>>> T[4] = 'bon'
>>> T ['Dans', 'python', 'tout', 'est', 'bon']
>>> L ['Dans', 'python', 'tout', 'est', 'bon']
>>> L = T[:]
>>> L[4] = 'objet'
>>> T; L
['Dans', 'python', 'tout', 'est', 'bon'] ['Dans', 'python', 'tout', 'est', 'objet']
```

Correction:

```
>>> from copy import copy; T = copy( L )#copie superficielle
>>> L1= [[1, 2, 3], [4, 5]]
>>> L2 = copy(L1)# même problème qu'avant
>>> from copy import deepcopy; T =deepcopy( L )#copie profonde
```

43

List

Une liste L a ses propres méthodes

len(L) : taille de la liste

L.sort : trier la liste L

L.append : ajout d'un élément à la fin de la liste L

L.reverse : inverser la liste L

L.index : rechercher un élément dans la liste L

L.remove : retirer un élément de la liste L

L.pop : retirer le dernier élément de la liste L

L.pop(i) : retirer le ième élément de la liste L

del(L[i]): supprime le ième élément de L

L.count : compter les occurrences dans la liste L

La commande sorted(L) renvoie une liste triée par ordre croissant

Voir (help(list)) ou dir(list) pour autres méthodes..

44

Tuple

Initialisation

- (),
- tuple(),
- (2,), 'a', 'b', 'c', 'd',
- ('a', 'b', 'c', 'd')

Concaténation

```
>>> (1, 2)*3
(1, 2, 1, 2, 1, 2)
>>> t1, t2 = (1, 2, 3), (4, 5)
>>> t1 + t2
(1,2,3,4,5)
```

45

Tuple

• Modification

- un tuple **n'est pas modifiable**

```
>>> t = 'a', 'b', 'c', 'd'
>>> t[0] = 'master'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item assignment
>>> t = ('master',) + t[1:]
>>> t
('master', 'b', 'c', 'd')
```

46

Tuple

```
>>> t = (1, 2, [3, 4], 6)
```

```
>>> t[2][0] = 1
```

```
>>> t
```

```
(1, 2, [1, 4], 6)
```

Un tuple `t` a ses propres méthodes

(`help(tuple)`)

- Sont presque les mêmes que pour les listes (sauf modification)

47

dict

- Un tuple `t` a ses propres méthodes

(`help(tuple)`)

- Sont presque les mêmes que pour les listes (sauf modification)

48

dict

Initialisation

```
{}, dict(),
>>>dico={'point': 1, 'ligne': 2, 'triangle': 3}
>>> dico['quad'] = 4
>>> dico
{'quad': 4, 'ligne': 2, 'triangle': 3, 'point': 1}
>>> dico['point'] = 3;dico
{'quad': 4, 'ligne': 2, 'triangle': 3, 'point': 3}
```

49

dict

Attention!:Remarques

- un dictionnaire n'est pas une séquence
- un dictionnaire est constitué de clés et de valeurs
- on ne peut pas concaténer un dictionnaire avec un autre

50

dict

Copie d'un dictionnaire

```
>>> dico = {'computer':'ordinateur', 'mouse':'souris',
'keyboard':'clavier'}
>>> dico2 = dico
>>> dico3 = dico.copy()
>>> dico2['printer'] = 'imprimante'
>>> dico2
{'computer': 'ordinateur', 'mouse': 'souris', 'printer': 'imprimante',
'keyboard': 'clavier'}
>>> dico
{'computer': 'ordinateur', 'mouse': 'souris', 'printer': 'imprimante',
'keyboard': 'clavier'}
>>> dico3
{'computer': 'ordinateur', 'mouse': 'souris', 'keyboard': 'clavier'}
```

51

dict

Un dictionnaire a ses propres méthodes (help(dict))

- **len(dico)** : taille du dictionnaire
- **dico.keys** : renvoie les clés du dictionnaire sous forme de liste
- **dico.values** : renvoie les valeurs du dictionnaire sous forme de liste
- **dico.has_key** : renvoie True si la clé existe, False sinon
- **dico.get** : donne la valeur de la clé si elle existe, sinon une valeur par défaut

52

dict

Tableaux associatifs

Un tableau associatif (dictionnaire) est une collection modifiable de couples <clé non modifiable, valeur modifiable> permettant un accès à la valeur si on fournit la clé. On peut le voir comme une liste dans laquelle l'accès à un élément se fait par un code au lieu d'un indice. L'accès à un élément est optimisé en Python.

Création de nouveaux tableaux associatifs

avec `A = {'a': 1, 'b': 2, 'c': 3, 'd': 4}`

Description	Opération	exemple	résultat
tableau associatif vide	<code>{}</code>	<code>{}</code>	<code>{}</code>
	<code>dict</code>	<code>dict()</code>	<code>{}</code>
en extension	<code>{...}</code>	<code>{'a': 1, 'b': 2, 'c': 3}</code>	<code>{'c': 3, 'a': 1, 'b': 2}</code>
par itérable	<code>dict</code>	<code>dict([('a', 1), ('b', 2)])</code>	<code>{'a': 1, 'b': 2}</code>
nommé	<code>dict</code>	<code>dict(a=1, b=2, c=3)</code>	<code>{'c': 3, 'a': 1, 'b': 2}</code>
par copie superficielle	<code>copy</code>	<code>A.copy()</code>	<code>{'d': 4, 'c': 3, 'a': 1, 'b': 2}</code>
à partir d'un dict	<code>fromkeys</code>	<code>dict.fromkeys(A, 0)</code>	<code>{'d': 0, 'c': 0, 'a': 0, 'b': 0}</code>

53

dict

Informations/consultations

avec `A = {'a': 1, 'b': 2, 'c': 3, 'd': 4}` et `B = {'x': 24, 'y': 25, 'z': 26}`

Description	Opération	exemple	résultat
cardinal	<code>len</code>	<code>len(A)</code>	<code>4</code>
appartenance de clé	<code>in</code>	<code>'c' in A</code>	<code>True</code>
égalité	<code>==</code>	<code>A == B</code>	<code>False</code>
inégalité	<code>==</code>	<code>A != B</code>	<code>True</code>
accès à une valeur (erreur si absent)	<code>[]</code>	<code>A['b']</code>	<code>2</code>
accès à une valeur (défaut si absent)	<code>get</code>	<code>A.get('a', 0); A.get('z', 17)</code>	<code>1 17</code>
itérateur sur les couples	<code>items</code>	<code>[x for x in A.items()]</code>	<code>[('a', 1), ('c', 3), ('d', 4), ('b', 2)]</code>
itérateur sur les clés	<code>keys</code>	<code>[x for x in A.keys()]</code>	<code>['a', 'c', 'd', 'b']</code>
itérateurs sur les valeurs	<code>values</code>	<code>[x for x in A.values()]</code>	<code>[1, 3, 4, 2]</code>

54

dict

Modifications

avec `A = {'a': 1, 'b': 2, 'c': 3, 'd': 4}`

Description	Opération	exemple	résultat
vidage	<code>clear</code>	<code>A.clear(); A</code>	<code>{}</code>
suppression	<code>del</code>	<code>del A['c']; A</code>	<code>{'d': 4, 'b': 2, 'a': 1}</code>
modification d'une valeur	<code>[]</code>	<code>A['b']={'r', 'v', 'b'}; A</code>	<code>{'d': 4, 'c': 3, 'b': {'r', 'b', 'v'}, 'a': 1}</code>
retrait et valeur	<code>pop</code>	<code>A.pop('a', 9); A.pop('a', 9); A</code>	<code>1 9 {'d': 4, 'c': 3, 'b': 2}</code>
retrait et couple	<code>popitem</code>	<code>A.popitem(); A</code>	<code>('d', 4) {'c': 3, 'b': 2, 'a': 1}</code>
accès et/ou modification d'une valeur	<code>setdefault</code>	<code>A.setdefault('b', 9); A.setdefault('z', 9); A</code>	<code>2 9 {'z': 9, 'd': 4, 'c': 3, 'b': 2, 'a': 1}</code>
modification plurielle	<code>update</code>	<code>A.update({'a': 7}, {'x': 9}, {'y': 8}); A</code>	<code>{'y': 8, 'd': 4, 'b': 2, 'x': 9, 'c': 3, 'a': 7}</code>

55

set

Création de nouveaux ensembles

avec `A = {0, 1, 2, 3, 4}` et `B = {3, 4, 5, 6}`

Description	Opération	exemple	résultat
ensemble vide	<code>set</code>	<code>set()</code>	<code>set()</code>
ensemble non vide	<code>{...}</code>	<code>{1, 2, 3, 4}</code>	<code>{1, 2, 3, 4}</code>
itérable	<code>set</code>	<code>set("ESIEE PARIS")</code>	<code>{'I', 'A', 'P', 'S', 'E', ' ', 'R'}</code>
copie superficielle	<code>copy</code>	<code>A.copy()</code>	<code>{0, 1, 2, 3, 4}</code>
intersection	<code>&</code>	<code>A & B</code>	<code>{3, 4}</code>
	<code>intersection</code>	<code>A.intersection(B)</code>	<code>{3, 4}</code>
union	<code> </code>	<code>A B</code>	<code>{0, 1, 2, 3, 4, 5, 6}</code>
	<code>union</code>	<code>A.union(B)</code>	<code>{0, 1, 2, 3, 4, 5, 6}</code>
différence	<code>-</code>	<code>A - B</code>	<code>{0, 1, 2}</code>
	<code>difference</code>	<code>A.difference(B)</code>	<code>{0, 1, 2}</code>
différence symétrique	<code>^</code>	<code>A ^ B</code>	<code>{0, 1, 2, 5, 6}</code>
	<code>symmetric_difference</code>	<code>A.symmetric_difference(B)</code>	<code>{0, 1, 2, 5, 6}</code>

56

set

Informations/consultations

avec $A = \{0, 1, 2, 3, 4\}$ et $B = \{3, 4, 5, 6\}$

Description	Opération	exemple	résultat
cardinal	<code>len</code>	<code>len(A)</code>	5
appartenance	<code>in</code>	<code>1 in A</code>	True
inclusion	<code><=</code>	<code>{1, 2} <= A</code>	True
	<code>issubset</code>	<code>A.issubset(A)</code>	True
inclusion stricte	<code><</code>	<code>A < A</code>	False
contenance	<code>>=</code>	<code>A >= {1, 2}</code>	True
	<code>issuperset</code>	<code>A.issuperset(A)</code>	True
contenance stricte	<code>></code>	<code>A > {1, 2}</code>	True
égalité	<code>==</code>	<code>A == B</code>	False
inégalité	<code>!=</code>	<code>A != B</code>	True
intersection vide	<code>isdisjoint</code>	<code>A.isdisjoint(B)</code>	False

57

set

Modification d'un ensemble

avec $A = \{0, 1, 2, 3, 4\}$ et $B = \{3, 4, 5, 6\}$

Description	Opération	exemple	résultat
ajout	<code>add</code>	<code>A.add(2014); A</code>	$\{0, 1, 2, 3, 4, 2014\}$
vidage	<code>clear</code>	<code>A.clear(); A</code>	<code>set()</code>
extraction (retourne et supprime un élément)	<code>pop</code>	<code>A.pop(); A</code>	$\{0, 1, 2, 3, 4\}$
suppression (erreur si absent)	<code>remove</code>	<code>A.remove(2); A</code>	$\{0, 1, 3, 4\}$
suppression (sans effet si absent)	<code>discard</code>	<code>A.discard(4); A</code>	$\{0, 1, 2, 3\}$
union	<code> =</code>	<code>A = B; A</code>	$\{0, 1, 2, 3, 4, 5, 6\}$
	<code>update</code>	<code>A.update(B); A</code>	$\{0, 1, 2, 3, 4, 5, 6\}$
intersection	<code>&=</code>	<code>A &= B; A</code>	$\{3, 4\}$
	<code>intersection.update</code>	<code>A.intersection.update(B); A</code>	$\{3, 4\}$
différence symétrique	<code>^=</code>	<code>A ^= B; A</code>	$\{0, 1, 2, 5, 6\}$
	<code>symmetric.difference.update</code>	<code>A.symmetric.difference.update(B); A</code>	$\{0, 1, 2, 5, 6\}$
différence	<code>-=</code>	<code>A -= B; A</code>	$\{0, 1, 2\}$
	<code>difference.update</code>	<code>A.difference.update(B); A</code>	$\{0, 1, 2\}$

58

MERCI

59

Travaux Pratiques:

- TP2

60



جامعة ابن توفيل
UNIVERSITÉ IBN TOFÄÏL
Faculté des Sciences

MASTER D'UNIVERSITÉ SPÉCIALISÉ
... Informatique ...

GÉNIE LOGICIEL POUR LE CLOUD

Exercez les métiers de demain


Business Intelligence

BIG DATA

Big Data

Cloud Computing

Coordinateur
Pr. Moulay Youssef Hadi
Département d'informatique
hadiyoussef@gmail.com 0694 500 534



2019/2020

Département d'Informatique

Faculté des Sciences – Kenitra

Tableaux en Python

(Données multimédia)

Master: Master Spécialisé Génie Logiciel pour le Cloud
Présenté le: 04/Decembre/2019 (S1)
Contact: khalid.ounachad@uit.ac.ma

Plan

- Généralités,
- Manipulations de base(numpy , matplotlib.pyplot),
- Images et python ,
- Son et video en python.
- Annexe

- Généralités,

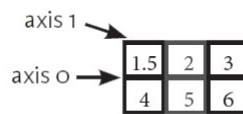
Généralités

- **Introduction:**
- Numpy est un package pour Python spécialisé dans la manipulation des tableaux (array), pour nous essentiellement les vecteurs et les matrices.

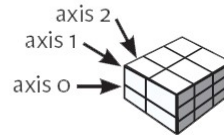
1D array



2D array



3D array



- Les tableaux « numpy » ne gère que les objets de même type.

Généralités

- **Introduction:(par rapport aux listes)**
- Le package propose un grand nombre de routines pour un accès rapide aux données (ex. recherche, extraction), pour les manipulations diverses(ex. tri), pour les calculs (ex. calcul statistique)
- Les tableaux « numpy » sont plus performants (rapidité, gestion de la volumétrie) que les collections usuelles de Python
- Les tableaux « numpy » sont sous-jacents à de nombreux packages dédiés au calcul scientifique sous Python.

Généralités

- **Introduction:(par rapport aux listes)**
- Une matrice est un tableau (array) à 2 dimensions
- Il n'est pas possible de tout aborder dans ce support. Pour aller plus loin, voir:

<https://docs.scipy.org/doc/numpy/reference/>

Généralités

- **Installation:(autres modules python)**
- Pip est le gestionnaire de paquets standard pour Python .
Il vous permet d'installer et de gérer des packages supplémentaires qui ne font pas partie de la bibliothèque standard Python .
- Pour se faire:
 - Recherche de packages publiés dans Python Package Index (PyPI)

Généralités

- **Installation:(autres modules python)**

- Installation de paquets supplémentaires/

- Désinstallation des packages et de leurs dépendances.

Voir(<https://realpython.com/what-is-pip/>)

Exemple: dans C:\Windows\System32>

pip install numpy

pip install matplotlib

- **Manipulation de base,**

Manipulations de base

Fonctions particulières:

Commande	Description (rapide)
<code>np.zeros(n) ((n,p))</code>	vecteur nul de taille n , matrice nulle de taille n,p
<code>np.eye(n) ((n,p))</code>	matrice de taille n (n,p) avec des 1 sur la diagonale et des zéros ailleurs
<code>np.ones(n) ((n,p))</code>	vecteur de taille n , matrice de taille n,p remplie de 1
<code>np.diag(v)</code>	matrice diagonale dont la diagonale est le vecteur v
<code>np.diag(v,k)</code>	matrice dont la 'diagonale' décalée de k est le vecteur v (k est un entier relatif)
<code>np.random.rand(n) ((n,p))</code>	vecteur (taille n), matrice (taille n,p) à coefficients aléatoires uniformes sur $[0,1]$

Manipulations de base

Exemples:

```
>>> v=np.ones(5)
>>> v
array([ 1.,  1.,  1.,  1.,  1.])
>>> np.diag(v,2)
array([[ 0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.]])
>>> np.diag(v,-1)
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.]])
```

Manipulations de base

Exemples:

```
>>> b=np.ones((4,4),float)+np.eye(4,4)
>>> print b
[[ 2.  1.  1.  1.]
 [ 1.  2.  1.  1.]
 [ 1.  1.  2.  1.]
 [ 1.  1.  1.  2.]]
>>> np.reshape(b,16)
array([ 2.,  1.,  1.,  1.,  1.,  2.,  1.,  1.,  1.,  1.,  2.,  1.,  1.,
        1.,  1.,  2.])
>>> np.reshape(b,(2,8))
array([[ 2.,  1.,  1.,  1.,  1.,  2.,  1.,  1.],
       [ 1.,  1.,  2.,  1.,  1.,  1.,  1.,  2.]])
>>> b.flatten()
array([ 2.,  1.,  1.,  1.,  1.,  2.,  1.,  1.,  1.,  1.,  2.,  1.,  1.,
        1.,  1.,  2.])
```

Manipulations de base

Création: Une première méthode consiste à convertir une liste en un tableau via la commande array:

import numpy as np

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1.,  4.,  5.,  8.])
>>> type(a)
<type 'numpy.ndarray'>
```

Un tableau peut être multidimensionnel ;

```
>>> a=np.array([[[1,2],[1,2]],[[1,2],[1,2]]])
>>> a
array([[ [1, 2],
         [1, 2]],
       [[1, 2],
         [1, 2]]])
>>> type(a)
<type 'numpy.ndarray'>
>>> a[1,1,1]
2
```

Manipulations de base

Création:

Linspace:

```
>>> np.linspace(1., 4., 6)
array([ 1. ,  1.6,  2.2,  2.8,  3.4,  4. ])
```

Conversion en matrice:

```
>>> a=np.array([1, 2, 4])
>>> np.mat(a)
matrix([[1, 2, 4]])
```

Manipulations de base

Création via une saisie manuelle:

```
a = np.array([[1.2,2.5],[3.2,1.8],[1.1,4.3]])
```

Noter le rôle des `[]` et `[]` pour délimiter les portions de la matrice

```
#type de la structure
print(type(a)) #<class 'numpy.ndarray'>
#type des données
print(a.dtype) #float64
#nombre de dimensions
print(a.ndim) #2 (car c'est une matrice)
#nombre de lignes et col, shape renvoie un tuple
print(a.shape) #(3,2) → 3 lignes et 2 colonnes
#nombre totale de valeurs
print(a.size) #6, nb.lignes x nb.colonnes
```

Manipulations de base

Typage des données:

```
#print de l'ensemble
print(a)
```



```
[[ 1.2  2.5]
 [ 3.2  1.8]
 [ 1.1  4.3]]
```

```
#création et typage implicite
a = np.array([[1,2],[4,7]])
print(a.dtype) #int32
```

```
#création et typage explicite – préférable !
a = np.array([[1,2],[4,7]],dtype=float)
print(a.dtype) #float64
```

Manipulations de base

Création d'une matrice à partir d'une séquence de valeurs:

```
#création à partir d'une séquence
#attention les dim. doivent être compatibles
a = np.arange(0,10).reshape(2,5)
print(a)
```

`arange()` génère une séquence de valeurs, 0 à 9.
`reshape()` se charge de les réorganiser en matrice
 2 lignes et 5 colonnes.

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

```
#un vecteur peut être converti en matrice
a = np.array([2.1,3.4,6.7,8.1,3.5,7.2])
print(a.shape) # (6,)
#redim. en 3 lignes x 2 col.
b = a.reshape(3,2)
print(b.shape) # (3, 2)
print(b)
```

```
[[ 2.1  3.4]
 [ 6.7  8.1]
 [ 3.5  7.2]]
```

```
#matrices de valeurs identiques
#ex. pour une initialisation
a = np.zeros(shape=(2,4))
print(a)
```

```
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
```

```
#plus généralement
a = np.full(shape=(2,4),fill_value=0.1)
print(a)
```

```
[[ 0.1  0.1  0.1  0.1]
 [ 0.1  0.1  0.1  0.1]]
```

Manipulations de base

Chargement à partir d'un fichier – Conversions

Les données peuvent être stockées dans un fichier texte (`loadtxt` pour charger, `savetxt` pour sauver)

	age	poids	taille
1	45	62	168
2	34	68	187
3	53	85	159

La première ligne doit être ignorée dans ce fichier, d'où le symbole # en début de 1^{ère} ligne.

Remarque : si besoin, modifier le répertoire par défaut avec la fonction `chdir()` du module `os` (qu'il faut importer au préalable)

#charger à partir d'un fichier, typage explicite

#séparateur de colonne = tabulation « \t »

`a = np.loadtxt("matrice.txt", delimiter="\t", dtype=float)`

`print(a)`

```
[[ 45.  62. 168.]
 [ 34.  68. 187.]
 [ 53.  85. 159.]]
```

Conversion d'une collection (type standard Python) en type array de « numpy »

#liste de valeurs

`lst = [1.2, 3.1, 4.5, 6.3]`

`print(type(lst))` # <class 'list'>

#conversion à partir d'une liste : 2 étapes `asarray()` et `reshape()`

`a = np.asarray(lst, dtype=float).reshape(2,2)`

`print(a)`

```
[[ 1.2  3.1]
 [ 4.5  6.3]]
```

Manipulations de base

Redimensionnement:

`a =`

```
[[ 1.2  2.5]
 [ 3.2  1.8]
 [ 1.1  4.3]]
```

#matrice de valeurs

`a = np.array([[1.2, 2.5], [3.2, 1.8], [1.1, 4.3]])`

Accoler le vecteur **b** en tant que nouvelle ligne (axis = 0) de la matrice a

#ajouter une ligne – marche pour la concaténation de matrices

`b = np.array([4.1, 2.6])`

`c = np.append(a, b, axis=0)`

`print(c)`

```
[[ 1.2  2.5]
 [ 3.2  1.8]
 [ 1.1  4.3]
 [ 4.1  2.6]]
```

Accoler le vecteur **d** en tant que nouvelle colonne (axis = 1) de la matrice a

#ajouter une colonne

`d = np.array([[7.8], [6.1], [5.4]])`

`print(np.append(a, d, axis=1))`

```
[[ 1.2  2.5  7.8]
 [ 3.2  1.8  6.1]
 [ 1.1  4.3  5.4]]
```

Insertion de **b** en tant que nouvelle ligne (axis = 0) à la position n°1

#insertion

`print(np.insert(a, 1, b, axis=0))`

```
[[ 1.2  2.5]
 [ 4.1  2.6]
 [ 3.2  1.8]
 [ 1.1  4.3]]
```

Suppression de la ligne (axis = 0) via son indice (n°1)

#suppression

`print(np.delete(a, 1, axis=0))`

```
[[ 1.2  2.5]
 [ 1.1  4.3]]
```

Redimensionnement d'une matrice

#modifier la dimension d'une matrice existante

#parcourt les données lignes par ligne

`h = np.resize(a, new_shape=(2,3))`

`print(h)`

```
[[ 1.2  2.5  3.2]
 [ 1.8  1.1  4.3]]
```


Manipulations de base

Statistiques:

Principe : les calculs sont réalisés selon un axe (0 : traitement des valeurs en ligne pour chaque colonne, 1 : inversement)

$v = \begin{bmatrix} 1.2 & 2.5 \\ 3.2 & 1.8 \\ 1.1 & 4.3 \end{bmatrix}$

#moyenne des lignes pour chaque colonne
`print(np.mean(v,axis=0))` # [1.833 2.867]

#moyenne des colonnes pour chaque ligne
`print(np.mean(v,axis=1))` # [1.85 2.5 2.7]

#somme cumulée des valeurs en ligne pour chaque colonne
`print(np.cumsum(v,axis=0))`

$\begin{bmatrix} 1.2 & 2.5 \\ 4.4 & 4.3 \\ 5.5 & 8.6 \end{bmatrix}$

#matrice de corrélation
 #rowvar = 0 pour indiquer que les variables
 #sont organisés en colonnes
`m = np.corrcoef(v,rowvar=0)`
`print(m)`

$\begin{bmatrix} 1. & -0.74507396 \\ -0.74507396 & 1. \end{bmatrix}$

Manipulations de base

Calcul matriciel et algèbre linéaire:

$x = \begin{bmatrix} 1.2 & 2.5 \\ 3.2 & 1.8 \\ 1.1 & 4.3 \end{bmatrix}$

$y = \begin{bmatrix} 2.1 & 0.8 \\ 1.3 & 2.5 \end{bmatrix}$

#transposition
`print(np.transpose(x))`



$\begin{bmatrix} 1.2 & 3.2 & 1.1 \\ 2.5 & 1.8 & 4.3 \end{bmatrix}$

#multiplication
`print(np.dot(x,y))`



$\begin{bmatrix} 5.77 & 7.21 \\ 9.06 & 7.06 \\ 7.9 & 11.63 \end{bmatrix}$

#déterminant
`print(np.linalg.det(y))`

4.21

#inversion
`print(np.linalg.inv(y))`



$\begin{bmatrix} 0.59382423 & -0.19002375 \\ -0.3087886 & 0.49881235 \end{bmatrix}$

Manipulations de base

Calcul matriciel et algèbre linéaire:

$$x = \begin{bmatrix} 1.2 & 2.5 \\ 3.2 & 1.8 \\ 1.1 & 4.3 \end{bmatrix} \quad y = \begin{bmatrix} 2.1 & 0.8 \\ 1.3 & 2.5 \end{bmatrix}$$

Solution de `#résolution d'équation`
`Y.a = z`
`z = np.array([1.7,1.0])`
`print(np.linalg.solve(y,z)) # [0.8195 -0.0261]`

On peut faire `#vérification`
`a = Y-1.z`
`print(np.dot(np.linalg.inv(y),z)) # [0.8195 -0.0261]`

`#matrice symétrique avec XTX`
`s = np.dot(np.transpose(x),x)`
`print(s)`

`#val. et vec. propres d'une matrice symétrique`
`print(np.linalg.eigh(s))`

```
[ [ 12.89  13.49]
  [ 13.49  27.98]]
```

```
(array([ 4.97837925, 35.89162075]),
 array([[ -0.86259502,  0.50589508],
        [ 0.50589508,  0.86259502]]))
```

Manipulations de base

Tracé de courbes graphiques:

Commande	Bibl.	Résultat
<code>plot(x,y,c?)</code>	matplotlib.pyplot	Affiche les points définis par les vecteurs <code>x</code> et <code>y</code> , (Option-: <code>c</code> permet de définir le format et la couleur du tracé)
<code>imshow(m,c?)</code>	matplotlib.pyplot	Affiche la matrice <code>m</code> en deux dimensions
<code>show()</code>	matplotlib.pyplot	Affiche la figure courante
<code>savefig(name)</code>	matplotlib.pyplot	Sauvegarde la figure courante dans le fichier <code>name</code>
<code>clf()</code>	matplotlib	Efface la figure courante
<code>legend(array,loc?)</code>	matplotlib.pyplot	Dessine une légende contenant les lignes apparaissant dans <code>array</code> (Option-: <code>loc = (x, y)</code> pour définir l'emplacement)
<code>xlabel(str) ylabel(str)</code>	matplotlib.pyplot	Imprime une légende pour décrire les axes horizontaux et verticaux
<code>axis([xl,xr,yb,yt])</code>	matplotlib.pyplot	Cadre la figure sur le rectangle décrit par les 4 coordonnées.

Manipulations de base

Tracé de courbes graphiques: Exemple

Commande	Bibl.	Résultat
<code>plot(x,y,c?)</code>	<code>matplotlib.pyplot</code>	Affiche les points définis par les vecteurs <code>x</code> et <code>y</code> , (Option-: <code>c</code> permet de définir le format et la couleur du tracé)
<code>imshow(m,c?)</code>	<code>matplotlib.pyplot</code>	Affiche la matrice <code>m</code> en deux dimensions
<code>show()</code>	<code>matplotlib.pyplot</code>	Affiche la figure courante
<code>savefig(name)</code>	<code>matplotlib.pyplot</code>	Sauvegarde la figure courante dans le fichier <code>name</code>
<code>clf()</code>	<code>matplotlib</code>	Efface la figure courante
<code>legend(array,loc?)</code>	<code>matplotlib.pyplot</code>	Dessine une légende contenant les lignes apparaissant dans <code>array</code> (Option-: <code>loc = (x, y)</code> pour définir l'emplacement)
<code>xlabel(str) ylabel(str)</code>	<code>matplotlib.pyplot</code>	Imprime une légende pour décrire les axes horizontaux et verticaux
<code>axis([xl,xr,yb,yt])</code>	<code>matplotlib.pyplot</code>	Cadre la figure sur le rectangle décrit par les 4 coordonnées.

Manipulations de base

Tracé de courbes graphiques: Exemple

Le module Matplotlib est chargé de tracer les courbes :

```
>>> import matplotlib.pyplot as plt
```

```
Commençons par la fonction sinus.
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(-5,5,100)
plt.plot(x,np.sin(x)) # on utilise la fonction sinus de Numpy
plt.ylabel('fonction sinus')
plt.xlabel("l'axe des abscisses")
plt.show()
```

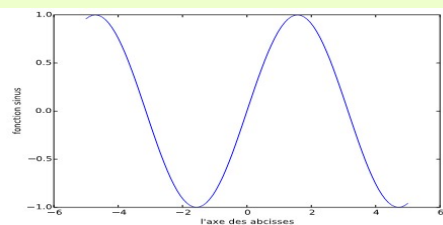


Figure1 : Graphe de la fonction sinus

Manipulations de base

Tracé de courbes graphiques: Exemple2

```
>>> x=np.linspace(-np.pi,np.pi,100)
>>> plt.plot(x,np.sin(x),color="red", linewidth=2.5, linestyle="-", label="sinus")
>>> plt.plot(x,np.cos(x),color="blue", linewidth=2.5, linestyle="-", label="cosinus")
>>> plt.legend(loc='upper left')
>>> plt.axis([-np.pi,np.pi,-1,1])
>>> plt.yticks([-1, 0, +1],
[r'$-1$', r'$0$', r'$+1$'])
>>> plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
[r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
```

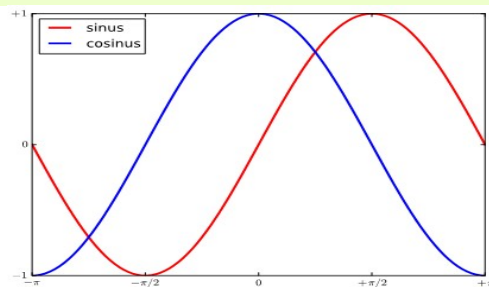


Figure2 : Graphe des fonctions sinus et cosinus

- Images et python,

Images et Python

Principe:

l'objectif sera toujours de transformer une image en tableau numpy, pour pouvoir ensuite la manipuler.

Les modules:

- **matplotlib**
- **pil**
- **Open cv**
- ...

Images et Python

Les modules:

PIL permet de lire une image enregistrée localement dans de nombreux formats.

matplotlib.image ne permet que de charger des .png. Mais vous avez directement un tableau numpy (pas besoin de transformation).

Images et Python

Lecture de l'image:

Avec "matplotlib.image"

```
import matplotlib.image as mpimg
import numpy as np
img = mpimg.imread("monimage.png")
```

Avec PIL

```
from PIL import Image import numpy as np
imgpil = Image.open("monimage.png") #
anciennement np.asarray img =
np.array(imgpil) # Transformation de
L'image en tableau numpy
```

Images et Python

Affichage de l'image :

Matplotlib permet d'afficher une image (si c'est un tableau numpy).

Avec "matplotlib.image"

```
import matplotlib.pyplot as plt
plt.imshow(img)
plt.show()
```

Avec PIL

```
from PIL import Image
im = Image.open("monimage.jpg")
im.show()
```

Images et Python

Sauvgarde de l'image: Pour enregistrer des images, on utilise une méthode similaire au chargement. Matplotlib permet ainsi de sauvegarder directement un tableau numpy au format PNG uniquement. PIL permettra de sauvegarder dans n'importe quel format, pourvu qu'on ait transformé le tableau numpy en Image PIL.

Avec Matplotlib

```
import matplotlib.image as mpimg  
mpimg.imsave("resultat.png", img)
```

Avec PIL

Avec PIL, il faut s'abord transformer le tableau numpy en image PIL.

```
from PIL import Image  
imgpil = Image.fromarray(img) # Transformation du tableau en image PIL  
imgpil.save("resultat.jpg")
```

- Son et Vidéo en Python,

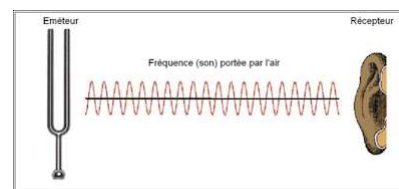
Son et Python

Introduction:

Le son est une sensation auditive provoquée par une vibration.

Trois éléments sont nécessaires à l'existence d'un son :

- une source qui produit le son,
- un milieu qui transmet la vibration,
- un récepteur : l'oreille.



Son et Python

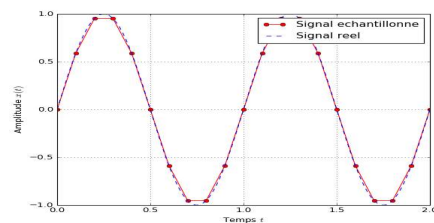
Principe:

Un signal unidimensionnel peut être vu comme une fonction mathématique du type: $x:t \rightarrow x(t)$

```
import numpy as np
import matplotlib.pyplot as plt

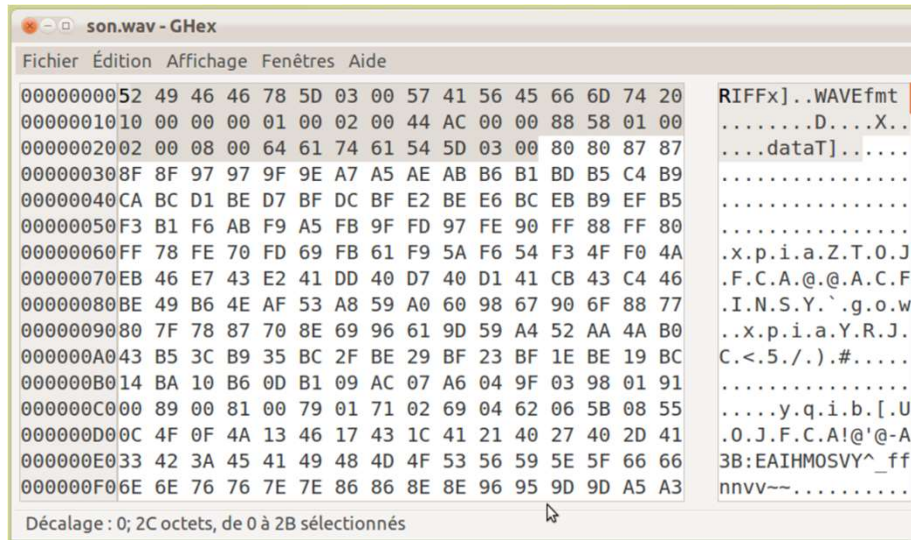
# Signal
T = 1.
def signal(t): return np.sin(2. * np.pi * t / T)

# Echantillonnage
D = 2. # Duree d'observation
fe = 10. # Frequence d'echantillonnage
N = (D * fe) + 1 # Nombre de points enregistrés
te = np.linspace(0., (N-1)/fe, N) # Grille d'echantillonnage
tp = np.linspace(0., D, 1000) # Grille plus fine pour tracer l'allure du signal parfait # Trace du signal
plt.plot(te, signal(te), 'or-', label = u"Signal echantillonne")
plt.plot(tp, signal(tp), 'b--', label = u"Signal reel")
plt.grid()
plt.xlabel("Temps t")
plt.ylabel("Amplitude x(t)")
plt.legend()
plt.show()
```



Son et Python

Contenu: édition hexadécimal



Son et Python

Modules:

- Pyaudio,
- Winsound,
- Pygame,
- Pyglet,
- Pymedia,
- espeak
- cv
- etc.

Son et Python

Exemples: Ecouter le son

```
import pygame
pygame.mixer.init()
pygame.mixer.Sound("son.wav").play()
while pygame.mixer.get_busy():
    # lecture en cours
    pass
```

```
# lecture audio (sortie vers la carte son)
import winsound
winsound.PlaySound('son.wav', winsound.SND_FILENAME)
```

Vidéo et Python

Définition:

La vidéo est une succession d'images animées défilant à une certaine cadence afin de créer une illusion de mouvement pour l'œil humain.

Elle peut être:

- analogique (signal continu d'intensité de luminance)
- ou numérique (suite de trames ou images)

En vidéo, c'est la caméra qui transforme l'information lumineuse (photons) en signal électrique (électrons). En vidéo analogique, l'intensité de ce signal électrique varie de façon continue.

Vidéo et Python

Exemples: Le programme pour lire une webcam branchée sur un port USB

```
1 import numpy as np
2 import cv2 as cv
3 cap = cv.VideoCapture(0)
4 while(True):
5     # Capture image par image
6     ret, img = cap.read()
7     # Préparation de l'affichage de l'image
8     cv.imshow('frame',img)
9     # affichage et saisie d'un code clavier
10    if cv.waitKey(1) & 0xFF == ord('q'):
11        break
12# Ne pas oublier de fermer le flux et la fenetre
13cap.release()
14cv.destroyAllWindows()
```

Vidéo et Python

Changement de la taille

Pour changer la taille on utilise la méthode set, après avoir ouvert le flux video

```
1 cap.set(cv.CAP_PROP_FRAME_WIDTH,1280)
2 cap.set(cv.CAP_PROP_FRAME_HEIGHT,960)
```

- Lecture d'une webcam et enregistrement d'une vidéo

```
1 import numpy as np
2 import cv2 as cv
3 webcam = cv.VideoCapture(0)
4 fourcc = cv.VideoWriter_fourcc('MJPG')
5 fichier = cv.VideoWriter('c:/temp/test.avi',fourcc,20,(640,480))
6 while(True):
7     # Capture image par image
8     ret, img = webcam.read()
9     if ret==True:
10        fichier.write(img)
11        # Preparation de l'affichage de l'image
12        cv.imshow('Ma Webcam',img)
13        # affichage et saisie d'un code clavier
14        if cv.waitKey(1) & 0xFF == ord('q'):
15            break
16    else:
17        break
18# Ne pas oublier de fermer le flux et la fenetre
19webcam.release()
20fichier.release()
21cv.destroyAllWindows()
```

Vidéo et Python

Changement de la taille

Pour changer la taille on utilise la méthode set, après avoir ouvert le flux vidéo

```
1 cap.set(cv.CAP_PROP_FRAME_WIDTH,1280)
2 cap.set(cv.CAP_PROP_FRAME_HEIGHT,960)
```

- Lecture d'une webcam et enregistrement d'une vidéo

```
1 import numpy as np
2 import cv2 as cv
3 webcam = cv.VideoCapture(0)
4 fourcc = cv.VideoWriter_fourcc(*'MJPG')
5 fichier = cv.VideoWriter('c:/temp/test.avi', fourcc, 20, (640, 480))
6 while(True):
7     # Capture image par image
8     ret, img = webcam.read()
9     if ret==True:
10         fichier.write(img)
11         # Preparation de l'affichage de l'image
12         cv.imshow('Ma Webcam',img)
13         # affichage et saisie d'un code clavier
14         if cv.waitKey(1) & 0xFF == ord('q'):
15             break
16     else:
17         break
18 # Ne pas oublier de fermer le flux et la fenetre
19 webcam.release()
20 fichier.release()
21 cv.destroyAllWindows()
```

Vidéo et Python

- Lecture d'une webcam et enregistrement d'une vidéo

```
1 import numpy as np
2 import cv2 as cv
3 webcam = cv.VideoCapture(0)
4 fourcc = cv.VideoWriter_fourcc(*'MJPG')
5 fichier = cv.VideoWriter('c:/temp/test.avi', fourcc, 20, (640, 480))
6 while(True):
7     # Capture image par image
8     ret, img = webcam.read()
9     if ret==True:
10         fichier.write(img)
11         # Preparation de l'affichage de l'image
12         cv.imshow('Ma Webcam',img)
13         # affichage et saisie d'un code clavier
14         if cv.waitKey(1) & 0xFF == ord('q'):
15             break
16     else:
17         break
18 # Ne pas oublier de fermer le flux et la fenetre
19 webcam.release()
20 fichier.release()
21 cv.destroyAllWindows()
```

Vidéo et Python

- **Lecture d'un fichier vidéo:** La lecture d'un fichier vidéo ou du flux d'une webcam fonctionne sur le même principe. La différence est dans l'origine : pour un fichier il faut donner le nom !

```

1 import numpy as np
2 import cv2 as cv
3 fichier = cv.VideoCapture('c:/temp/test.avi')
4 while(True):
5     # Capture image par image
6     ret, img = fichier.read()
7     if ret==True:
8         # Preparation de l'affichage de l'image
9         cv.imshow('Mon fichier',img)
10        # affichage et saisie d'un code clavier
11        if cv.waitKey(1) & 0xFF == ord('q'):
12            break
13    else:
14        break
15 # Ne pas oublier de fermer le flux et la fenetre
16 fichier.release()
17 cv.destroyAllWindows()

```

Annexe

Dialoguer avec Excel en Python : openpyxl:

```

from openpyxl import Workbook
from openpyxl import load_workbook

#ouvrir le fichier excel sample.xlsx (le fichier excel doit
#se trouver dans le même répertoire que le fichier python)
wb = load_workbook(filename = "sample.xlsx")

#charger la feuille excel active dans une variable python
ws = wb.active

#on récupère la valeur de la case B2 puis on l'affiche
a = ws.cell(row=2, column=2).value
print(a)

#on écrit la valeur de la variable b dans la case B1
b = 425.5
ws.cell(row=1, column=2).value = b

#on enregistre le fichier sample.xlsm
wb.save("sample.xlsx")

#ouvrir un nouveau fichier excel
wb2 = Workbook()

#charger la feuille excel active dans une variable python
ws2 = wb2.active

#on écrit la valeur de la variable b dans la case A2
ws2.cell(row=1, column=2).value = b

#on enregistre le nouveau fichier excel en lui donnant le nom sample2.xlsx
wb2.save("sample2.xlsx")

```

MERCI

جامعة ابن توفيل
UNIVERSITÉ IBN TOFAL
Faculté des Sciences

MASTER D'UNIVERSITÉ SPÉCIALISÉ
:: Informatique ::

**GÉNIE LOGICIEL POUR
LE CLOUD**

Exercez les métiers de demain

Business Intelligence

BIG DATA

Big Data

Cloud Computing

Coordinateur
Pr. Moulay Youssef Hadi
Département d'informatique
hadiyoussef@gmail.com 0694 500 534

Travaux Pratiques:

- TP5