جامعة الحسن الأول
UNIVERSITÉ HASSAN 1ER

UNIVERSITE HASSAN 1ER

ÉCOLE NATIONALE DES SCIENCES APPLIQUEES – BERRECHID

ENSA Ecole Nationale des Sciences Appliquées BERRECHID

DIAPORAMAS

(SUPPORT ELECTRONIQUE DE COURS)

PROGRAMMATION PYTHON

ROFESSEUR : LAHCEN MOUMOUN

DEPARTEMENT GENIE INFORMATIQUE & MATHEMATIQUES

Module : Programmation Python

Chapitre 1: Les éléments de base du langage Python

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

Département Génie Informatique & Mathématiques

Présentation de Python

- Développé en 1989 par Guido van Rossum (la première version publique a été publiée en 1991);
- Orienté objet et open-source;
- Portable et extensible (tourne sur la plupart des plateformes);
- Support pour l'intégration d'autres langages;
- Très bon support pour le calcul scientifique (alternative à Matlab).

http://www.python.org

Environnement Python

- Télécharger puis installer Python depuis le site officiel de Python. Il existe 2 célèbre versions de Python : 2.x et 3.x (Python 3.x n'est pas une amélioration ou extension de Python 2.x , la version 3 de Python est désormais le standard);
- Utiliser un environnement de développement convivial :
  - PyCharm (version Community)
  - Spyder;
  - Eclipse + plugin PyDev;
  - Anaconda
  - …

IDLE

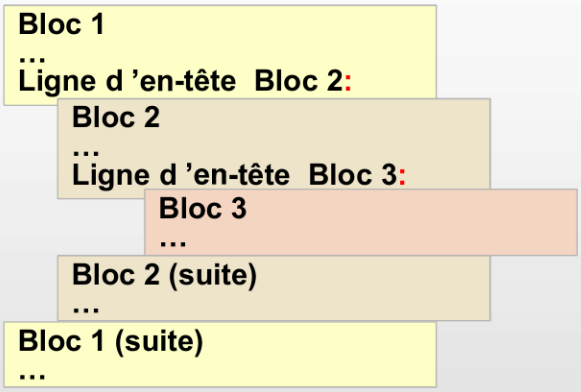
PyCharm

Le langage Python

- En Python, la casse est significative (les caractères majuscules et minuscules sont distingués);
- En Python, les commentaires sont préfixés par le caractère dièse (#), et peuvent être placés en fin de ligne ou prendre une ligne complète;
- Pour placer une instruction sur plusieurs lignes, il est nécessaire d'utiliser :
  - un saut de ligne dans une séquence d'éléments entre parenthèses, accolades ou crochets ;
  - des antislash (\).

Le langage Python

Python utilise l'indentation (caractère d'espacement ou tabulation) pour délimiter des blocs de code.



- Les blocs d'instructions sont toujours associés à une ligne d'en-tête contenant une instruction spécifique se terminant par un double point.
- Les blocs sont délimités par l'indentation : toutes les lignes d'un même bloc doivent être indentées exactement de la même manière.

Modèle de données de Python

- Le modèle de données de Python est basé sur les objets (toute donnée manipulée est un objet avec un identifiant, un type et une valeur);
- Les primitives permettant de lire les attributs d'objets associés aux données sont:
  - id(…) : renvoie l'identifiant d'un objet;
  - type(…) : renvoie le type d'un objet;
  - dir(…) : liste l'ensemble des fonctionnalités d'un objet.

Exemple (interpréteur python)

```
a = 1  # la variable a devient une référence à  un objet
id(a)        ⟶  134536624
type(a)      ⟶  <type 'int'>
dir(a)       ⟶  ['__abs__', … , '__mod__', ... ,
                 '__oct__', … , '__pow__', … ,
                 '__xor__']
```

Modèle de données de Python

- Sous Python, il suffit d'assigner une valeur à une variable pour en déterminer le type correspondant (aucune déclarations);
- Le type d'une variable dépend de sa valeur.

```
var1 = 5   # entier codé sur 32 bits
var3 = 3.0    # un réel codé sur 64 bits
var4 = 10+5 J   # un complexe avec var4.rea désigne la partie réel et
                  # var4.imag désigne la partie imaginaire
chaine = "  Toto  "  # une chaine de caractères
```


Les types numériques

- En Python, Il existe 3 types de base pour représenter les nombres (objets immuables : les objets dont le type et la valeur sont identiques sont définit par une seule instance, la modification de l'objet entraînera la création d'une copie) :
  - Le type **int** pour les nombres entiers (sous Python 3, le type int fonctionne sans limites de valeur);
  - Le type **float** pour les nombres à virgule flottante (réels à double précision);
  - Le type **complex** pour les nombres complexes (couple de nombres à virgule flottante).
- Les constructeurs int(), float() et complex() peuvent être utilisé pour produire des nombres d'un type spécifique.

Types de données

- En Python, un entier (type int) peut être représenté sous forme décimale, binaire (préfixe par 0b ou 0B) , octale (préfixe par 0o ou 0O) ou hexadécimale (préfixe par 0x ou 0X) . **oct** et **hex** permet d'obtenir ces représentations pour un entier;
- Le type **bool** est un sous type qui est utilisé pour représenter les valeurs booléennes (les constantes littérales **False** ou **True** pouvant être remplacées par les nombres 0 ou 1);
- **None** permet de spécifier une absence de valeur (comparable au NULL du JAVA).

Les opérateurs de base

- Opérateurs addition, soustraction, multiplication et division (x / y renvoie par défaut un float cependant x // y retourne le quotient entier de *x* et *y*) :

  +, -, *, /, %, //
- Opérateurs de comparaisons :

  ==, is, !=, is not, >, >=, <, <=
- Opérateurs logiques :

  or, and, not
- Opérateur abrégés :

  +=, -=, *= et /=
- Opérateurs puissance, valeur absolue, ...

  **, pow, abs, ...

Les fonctions de base : print et input

- La fonction print évalue une expression et affiche le résultat;
- La fonction input interrompt l'exécution d'un programme pour renvoyer une valeur saisie au clavier. Elle peut accepter le message à afficher à l'utilisateur comme un paramètre facultatif.

Exemple

```
x = int(input ("Entrez une entier :"))
absX=abs(x)
print("x=", x, "\n|x|=", absX )
print("(x)8=",oct(x), "(x)16=", hex(x))
```


Les fonctions de base

https://docs.python.org/fr/3/library/functions.html

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()		

Les séquences

- Une séquence s est une collection finie d'éléments indexés par des nombres positifs (varient de 0 à n-1 pour une séquence de n éléments).
  - s[i-1] se réfère au i^{ème} élément de la séquence s ;
  - s[-n] et s[-1] se référent respectivement au 1^{ier} et au dernier élément de s;
  - s[u:v] est une sous-partie de s allant de l'élément d'index u inclus à l'élément d'index v exclus (exemple : s[1:-1] est la tranche contenant tous les éléments sauf le premier et le dernier).
- Python fournit plusieurs primitives de manipulation communes à tous les types de séquences:
  - len() : permet de récupérer le nombre d'éléments de la séquence;
  - min() et max() : renvoient les éléments de valeurs minimum et maximum;
  - sum() : renvoie la somme des éléments, lorsque tous les éléments de la liste ont des types qui peuvent être additionnés.

Structure de contrôle

Les structures conditionnelles sont des regroupements de lignes délimités par un niveau d'indentation et dont le contenu est exécuté en fonction d'une ou plusieurs conditions (**elif** est la forme contractée de else if).

- L'instruction if

```
if expression1:
    bloc de lignes
elif expression2:
    bloc de lignes
…
else:
    bloc de lignes
```

Exemple

```
if a > 0:
    print ("a est positif .")
elif a < 0:
    print ("a est négatif .")
else:
    print ("a est nul .")
```

Structure de contrôle

- Syntaxe de la boucle while

```
while expression1:
    bloc de lignes
else:  # optionnel
    bloc de lignes else
```

Exemple while

```
x = y / 2
while x > 1:
  if y % x == 0:
    print (y, 'est facteur de', x)
    break
  x = x-1
else:
  print y, 'est premier'
```

- Syntaxe de la boucle for

```
for element in sequence:
    bloc de lignes
else:
    bloc de lignes
```

Exemple for

```
prod = 1
for p in range(1, 10):
# xrange pour un grand nombre d'éléments
    prod *= p
```

- **else** est exécutée en fin de boucle lorsque celle-ci se termine normalement;
- **break** permet de sortir de la boucle sans passer par else;
- **continue** remonte au début de la boucle;
- **pass** ne fait rien.

Exercice 1

Les nombres de Amstrong appelés parfois nombres cubes sont des nombres entiers qui ont la particularité d'**être égaux à la somme des cubes de leurs chiffres**.

Par exemple, 153 est un nombre de Amstrong car on a : $153=1^3+5^3+3^3$

Afficher tous les nombres de Amstrong sachant qu'ils sont tous compris entre 100 et 499.

Indication : Les nombres de Amstrong sont : 153, 370, 371 et 407.

Exercice 1

Les nombres de Amstrong appelés parfois nombres cubes sont des nombres entiers qui ont la particularité d'**être égaux à la somme des cubes de leurs chiffres**.
Par exemple, 153 est un nombre de Amstrong car on a : $153=1^3+5^3+3^3$

Afficher tous les nombres de Amstrong sachant qu'ils sont tous compris entre 100 et 499.
Indication : Les nombres de Amstrong sont : 153, 370, 371 et 407.

```
for nombre in range(100,500):
    nb = nombre
    s = 0
    while nb != 0:
        ch = nb % 10
        s += ch**3
        nb = nb // 10
    if nombre == s:
        print(nombre)
```


Les chaînes de caractères

- Les chaînes de caractères sont des séquences **immuables** (non modifiable après création) délimitées par des guillemets ou des apostrophes(les chaînes entre triple guillemets peuvent couvrir plusieurs lignes);
- Une chaîne de caractères est représentée par le type string dont l'encoding par défaut en python 3 est utf-8. Aucun type n'est spécifique pour un caractère (utiliser un string de longueur 1);
- Les objets de type string possèdent des opérateurs de formatage sous la forme « % précision_formatage » :
  - %d (%u) pour entier décimal signé (non signé) ;
  - %o (%x ou %X) pour octal non signé (hexadécimale préfixée par 0x ou 0X);
  - %f ou F% pour réel;
  - %s pour une chaine de caractère;
  - %c pour un caractère (sous la forme d'un string ou d'un entier).

Exemple:

```
x=32
nom="Python"
print ("nom=%s , x=%d , x%%=%.2f" % (nom , x, x/100))
```

Les chaînes de caractères

- Les chaînes peuvent être créés à partir d'autres objets à l'aide du constructeur str (la fonction **len** renvoie la taille d'une chaîne de caractères);
- Python offre plusieurs méthodes pour la manipulation des chaîne de caractères:
  - s.**lower**() et s.**upper**() renvoient respectivement le minuscule et le majuscule de s (**capitalize** met uniquement la première lettre en majuscule);
  - s.**split**("separateur") découpe s en plusieurs mot, en utilisant un séparateur;
  - s.**count**(sc) compte le nombre d'occurrences dans s de la sous chaîne de caractères sc;
  - s.**find**(sub[, start[, end]]) : recherche une sous-chaîne sub dans s (retourne l'index du début de sub ou -1 si sub n'a pas été trouvée);
  - s.**replace**(old, new) : substitue une sous chaîne de caractères par une autre;
  - s.**startswith**(sc) vérifie si s commence la sous chaine sc;
  - s.**rstrip**() : enlève les espaces situés sur les bords (début et fin) de s;
  - s.**isdigit** : renvoie True si la chaîne s ne contient que des nombres(sinon False);
  - …

Les chaînes de caractères

Exemple

```
ch = input("Donner une chaine de caractère :")
print("Affichage du nombre d'occuronce des chiffres")
print("============================================")
chiffres="0123456789"
for chiffre in chiffres:
    nbOcc=ch.count(chiffre)
    if nbOcc!=0:
        print(  chiffre, "--------->", nbOcc )
print("\nAffichage du nombre d'occuronce des lettres")
print("============================================")
code=ord('a')
while code <=ord('z'):
    nbOcc=ch.count(chr(code))
    if nbOcc!=0:
        print(chr(code), "--------->", nbOcc)
    code+=1
```

Les Listes

- Une liste est une séquence **muables** (modifiable) dont les éléments sont séparés par une virgule et l'ensemble est entouré par des crochets (une liste vide se note **[ ]**);
- Le constructeur list() permet de créer une liste vide. D'autres méthodes sont applicables aux listes :

Nom	Description
append(e)	Ajout d'un élément e en fin de liste
extend(L)	Ajout des éléments d'une liste L en fin de liste
insert(p, e)	Insérer un élément e à une position p (la position 0 correspond au début de liste)
remove(e)	Retire le premier élément e (si aucun élément n'est trouvé, une erreur est retournée)
pop(i)	Retire l'élément d'index i de la liste et le renvoie (« pop() » retire le dernier élément)
index(e)	Renvoie l'index du premier élément e (une erreur est renvoyée si e n'est pas trouvé)
count(e)	Indique le nombre d'occurrences de l'élément e
reverse()	Renverse la liste. Le premier élément devient le dernier, le deuxième l'avant-dernier, etc.

help(list)

Les Listes

Exemple

```
ch = input("Donner une chaine de caractère :")
sepateur =[" ", ".", ",", ";", ":", "?", "!", "-", "(",")","..."]
mots=list()
debut=0
while debut < len(ch):
    index=debut
    while index < len(ch):
       if ch[index] in sepateur:
            break
       index+=1
    fin=index
    mot=ch[debut:fin]
    if (mot is not None and len (mot)!=0):
        if (len(mots)!=0):
            mots.append("-")
        mots.append(mot)
    debut=fin+1
print ("Affichage des mots de la chaine")
print("".join(mots))# concaténer les éléments de la liste
```

Les tuples

- Les **tuples** sont des séquences **immuables** (éléments non modifiables après création) dont les éléments sont de types hétérogènes;
- Les éléments d'un tuple sont séparés par une virgule et l'ensemble est défini par des **parenthèses** (pour un tuple composé d'un seul élément, il est nécessaire d'ajouter une virgule après cet élément);
- Un tuple vide peut être crée en utilisant le constructeur « tuple() » alors que « tuple(sequence) » permet de créer le tuple correspondant à une séquence.

Exemple

```
colorNote = ('rouge', 12, 'vert', 14, 'bleu', 9)
colors = colorNote[::2]
print(colors)# affiche ('rouge', 'vert', 'bleu')
colorNote = colorNote + ('bizar',)
print(colorNote)# affiche('rouge',12,'vert',14,'bleu',9,'bizar')
colorNote[6]='noir'#erreur'tuple'object does not support item assignment
colorNote = colorNote + ([2],)
print(colorNote) #affiche ('rouge',12,'vert',14,'bleu',9,'bizar',[2])
colorNote[7]=2 #erreur 'tuple'object does not support item assignment
colorNote[7].append (10)
print(colorNote)# affiche ('rouge',12, 'vert',14,'bleu',9,'bizar',[2,10])
```


Le type dictionnaire

- Le type de mapping dict (appelé dictionnaire), est une collection modifiable dont les éléments sont identifiés par des clés uniques (de type **immuable**). Le constructeur dict() permet de créer un dictionnaire vide (noté **{}**);
- Chaque élément d'un dictionnaire est définit par une valeur préfixée de sa clé suivie de deux points. Les éléments, ainsi, formés sont séparés par des virgules et l'ensemble est entouré par des accolades (**les parenthèses délimitent les tuples, les crochets délimitent les listes et les accolades {} délimitent les dictionnaires**);

  Exemple : Un dictionnaire composé de deux éléments cle1: e1 et et cle2: e2 se note: {cle1: e1, cle2: e2}
- Pour accélérer les accès aux éléments, python utilise la méthode hash, qui transforme la clé d'un élément en une valeur constante.
  - L'instruction e = dic[cle] affecte à e la valeur de l'élément du dictionnaire dic associé à la clé cle;
  - L'instruction dic[cle]= e permet d'ajouter au dictionnaire dic l'élément « cle:e » si la clé n'existe pas. Sinon, l'ancienne valeur associée à la clé cle est remplacée par la nouvelle valeur e.

Les méthodes applicables aux dictionnaires

Nom	Description
clear()	Supprime tous les éléments du dictionnaire
copy()	Renvoie une copie par références du dictionnaire
has_key(cle)	Renvoie vrai si la clé fournie existe. Équivalent à la notation : cle in dictionnaire
items()	Renvoie sous la forme d'une liste de tuples, des couples (clé, valeur) du dictionnaire. Les objets représentant les valeurs sont des copies complètes et non des références.
keys() et values()	Renvoie respectivement sous la forme d'une liste l'ensemble des clés et les valeurs du dictionnaire (l'ordre de renvoi des éléments n'a aucune signification)
iteritems()	Fonctionne comme items() mais renvoie un itérateur sur les couples (clé, valeur)
iterkeys() et itervalues()	Renvoie respectivement un itérateur sur les clés et les valeurs (fonctionne comme keys() et values())
get(cle, default)	Renvoie la valeur identifiée par la clé cle. Si la clé n'existe pas, renvoie la valeur default fournie. Si aucune valeur n'est fournie, renvoie None
pop(cle, default)	Renvoie la valeur identifiée par la clé cle et retire l'élément du dictionnaire. Si la clé n'existe pas, pop se contente de renvoyer la valeur default. Si le paramètre default n'est pas fourni, une erreur est levée
popitem()	Renvoie le premier couple (clé, valeur) du dictionnaire et le retire. Si le dictionnaire est vide, une erreur est renvoyée. L'ordre de retrait des éléments correspond à l'ordre des clés retournées par keys()
setdefault(cle , default)	Fonctionne comme get() mais si cle n'existe pas et default est fourni, le couple (cle, default) est ajouté à la liste

Exercice 2

Soit la séquence nucléotidique suivante :

ACCTAGCCATGTAGAATCGCCTAGGCTTTAGCTAGCTCTAGCTAGCTG

En utilisant un dictionnaire, écrire un programme qui répertorie tous les mots de 2 lettres (successifs) qui existent dans la séquence (AA, AC, AG, AT, etc.) avec leur nombre d'occurrences puis qui les affiches à l'écran.

Exercice 2

Soit la séquence nucléotidique suivante :

ACCTAGCCATGTAGAATCGCCTAGGCTTTAGCTAGCTCTAGCTAGCTG

En utilisant un dictionnaire, écrire un programme qui répertorie tous les mots de 2 lettres (successifs) qui existent dans la séquence (AA, AC, AG, AT, etc.) avec leur nombre d'occurrences puis qui les affiches à l'écran.

```
phrase=input("donner une phrase  :")
dictMot=dict()
for i in range(1, len(phrase)):
    mot=phrase[i - 1:i + 1]
    if mot in dictMot:
        dictMot[mot]+= 1
    else:
        dictMot[mot]=1
print(dictMot)
```


Les fonctions en Python

- La syntaxe Python pour la définition d'une fonction est la suivante :

  def nomDeLaFonction**(**liste de paramètres**):**

  bloc d'instructions
- La directive return précise le résultat explicite renvoyé par la fonction. Une fonction qui ne renvoie pas explicitement de valeur renvoie un objet **None**;
- La liste de paramètres spécifie les informations à fournir en guise d'arguments lors de l'appel de la fonction (les parenthèses peuvent parfaitement rester vides si la fonction ne nécessite pas d'arguments).

Exemple

```
def table_par_7 ():
    nb = 7
    i = 0
    while i < 10:
        print (i + 1, "*", nb , "=", (i + 1) * nb)
        i += 1
```

Exécution

```
1 * 7 = 7
2 * 7 = 14
3 * 7 = 21
...
10 * 7 = 70
```

Documentation strings (docstrings)

- Les objets docstrings sont des chaînes de caractères délimités par triple guillemets et placées au début du corps des fonctions;
- Les objets docstrings sont utilisées pour fournir des informations sur l'objectif et le détail des paramètres d'une fonction. Ils sont automatiquement associés à la variable __doc__ de cette fonction.

Exemple:

```
def pi():
    """Renvoie une approximation du nombre Pi. """
    return 3.14
help(pi) # ou print(pi.__doc__)
```

Exécution

```
Help on function pi in module __main__:
pi()
Renvoie une approximation du nombre Pi.
(END)
```


Contexte d'exécution global et local

- Lors de sa définition, une variable est insérée :
  - Dans le contexte local si elle est définie dans un bloc (Accessibles uniquement à l'intérieur du bloc : boucle, fonction...). Ce contexte est un dictionnaire accessible à un instant donné par la directive locals()
  - Dans un contexte global si elle est définie en dehors de tout bloc. Ce contexte est un dictionnaire accessible par le biais de la primitive globals()
- Lorsque une variable est invoquée, elle est recherchée dans le contexte local puis global. À l'intérieur d'un bloc, il est possible de spécifier qu'une variable est dans le contexte global en utilisant la directive global.

Exemple

```
name = 'Joe'
name2 = 'Sam'
def home(name):
    global name2
    print(locals())
    print('Bonjour %s' % name)
    name2=name
```

Exécution

```
home('Tarek')
```

⇨ {'name': 'Tarek'}
Bonjour Tarek

```
print(globals())
```

⇨ {…,'name': 'Joe', 'name2': 'Tarek' }

Paramètres d'une fonction

- Le passage de paramètres d'une fonction est un passage par référence
- Il existe trois types de paramètres d'une fonction :
  - Les paramètres **explicites** sont définis par des identifiants séparés par des virgules. Chacun de ces paramètres peut en outre être enrichi d'une valeur par défaut et devenir optionnel;
  - Les paramètres **non explicites** (fournir les valeurs nommées lors de l'appel de la fonction);
  - Les paramètres **arbitraires** (fournir des valeurs non nommées lors de l'appel de la fonction).

Les paramètres explicites d'une fonction

- Dans le cas des paramètres explicites, le code appelant peut définir ou non la valeur de chaque paramètre sans avoir à respecter un ordre précis (nommage des paramètres: notation nom=valeur).

  Exemple

```
def sub(a, b):
    return a – b
sub(10, 5)  # ou sub(a=10, b=5) ou sub(b=5, a=10)
```

- Les valeurs par défaut ne sont interprétées qu'une seule fois (au moment de la lecture de la définition de la fonction)

  Exemple

```
def param():
    print('param() appelé')
    return [1, 2, 3]
def add_element(element, list_=param() ):
    list_.append(element)
    return list_
print(add_element(4)) # affiche : param() appelé
                      # [1, 2, 3, 4]
print(add_element(5))
# affiche [1, 2, 3, 4, 5]
def param():
    print('param() appelé')
    return [5]
print(add_element(8))
# affiche  [1, 2, 3, 4, 5, 8]
```

Exercice 3

Sachant qu'une année bissextile est divisible par 4 (mais non par 100) ou par 400 et que le mois de février correspondant compte 29 jours.

- Écrire la fonction qui prend un entier correspondant à une année en paramètre et qui retourne vrai ou faux si celle-ci est ou non bissextile.
- En utilisant les listes, écrire une deuxième fonction qui retourne le nombre de jours à partir de deux paramètres : le nom du mois et l'entier correspondant à l'année.
- Réécrire cette deuxième fonction en utilisant le dictionnaire dnbjm = {'janvier': 31, 'février': 28, ...}

Exercice 3

```
def bissextile (annee):
    if annee % 400==0:
        return True
    if annee % 100==0:
        return False
    if annee % 4==0:
        return True
    return False
def nbJour(mois, annee):
    lnj = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    lnm = ['janvier', 'février', 'mars', 'avril', 'mai', 'juin', ' juillet', 'août', 'septembre',
                'octobre', 'novembre','décembre']
    if mois not in lnm:
        return -1
    indx=lnm.index(mois)
    if indx!=1 or not bissextile(annee):
        return lnj[indx]
    return 29
m=input("Donner mois  : ")
an=int(input("Donner année  : "))
print(nbJour(m,an))
```

Exercice 3

```
def bissextile (annee):
    if annee % 400==0:
        return True
    if annee % 100==0:
        return False
    if annee % 4==0:
        return True
    return False
def nbJour2(mois, annee):
    dnbjm = {'janvier' : 31 , 'février' : 28 , 'mars' : 31 , 'avril' : 30 , 'mai' : 31 , 'juin' :
                30 ,'juillet' : 31 , 'août' : 31 , 'septembre' : 30 , 'octobre' : 31 ,
                'novembre' : 30 ,'décembre' : 31 }
    if mois not in dnbjm:
        return -1
    if mois=='février' and bissextile(annee):
        return 29
    return dnbjm[mois]
m=input("Donner mois  : ")
an=int(input("Donner année  : "))
print(nbJour2(m,an))
```


Les paramètres non explicites d'une fonction

Définir dans la liste des paramètres de la fonction un dictionnaire (précédé son nom de deux étoiles) pour y placer les valeurs des arguments fournis lors de l'appel la fonction.

Exemple

```
def team(name, leader='non défini', **players):
    print('Equipe %s' % name)
    print('Capitaine: %s' % leader)
    for name, value in players.items():
        print('%s: %s' % (name, value))
team('Les bleus')
```

⇨
```
Equipe Les bleus
Capitaine: non défini
```

```
team('Les vaillants', 'Robert', gardien='André', attaquant='Micheline')
```

⇨
```
Equipe Les vaillants
Capitaine: Robert
attaquant: Micheline
gardien: André
```

les paramètres arbitraires d'une fonction

- Les paramètres arbitraires sont équivalents aux paramètres non explicites sauf qu'ils ne sont pas nommés;
- Les paramètres arbitraires sont regroupés dans un tuple nommé (préfixé d'une seule étoile et déclaré après les paramètres explicites et avant les paramètres non explicites) à passer à la fonction.

Exemple

```
def format(sentence, *args):
    print(sentence % args)
format('%d fois plus de %s possibles', 2, 'combinaisons')
# affiche    2 fois plus de combinaisons possibles
```


Les fonctions anonymes

- Le mot-clé **lambda** permet une définition de fonctions anonymes : fonction plus courte dont les instructions possibles sont souvent limitées;
- Les paramètres de cette fonction se situent entre le mot-clé lambda et : (séparés par des virgules) et le résultat de la dernière instruction de la fonction sera automatiquement renvoyé (aucun utilisation de return);
- Ces fonctions sont souvent stockées dans des variables (ou passées en paramètre à une autre fonction).

Exemple

```
def test(f, a, b=None):
    if b != None:
        r = f(a, b)
    else : r = f(a)
    if r:
        print("Test passé avec succès.")
    else:
        print("Echec du test.")
pair = lambda a: a % 2 == 0
divise = lambda a, b: a % b == 0
test(pair, 6) # affiche Test passé avec succès.
test(divise, 6,4)# affiche Echec du test.
```

Les modules

- Certaines fonctionnalités de Python ne sont pas chargées par défaut (fonctions non intégrées). Pour utiliser ces fonctionnalités, il faut importer les modules correspondants (fichier *.py). L'import est réalisé grâce à :
  - l'instruction **import** (l'utilisation d'une déclaration nécessite de préfixer la déclaration par le nom du module);
  - l'instruction **from .. import ..** (l'utilisation direct d'une déclaration);
- L'utilisation de * permet d'importer toutes les déclarations d'un module (à éviter)
- l'utilisation de as permet de renommer localement un module ou une déclaration d'un module.
- La recherche du module à importer est effectuée dans le répertoire courant, puis dans la liste des répertoires définis dans la variable d'environnement PYTHONPATH et enfin dans le répertoire d'installation de Python qui contient tous les modules fournis avec l'interpréteur (cette liste de répertoires peut être retrouvée dans la liste path du module sys).

Les modules

Un module respecte toujours la même structure, soit :

- Un en-tête composé d'un bloc de commentaires (commun à tous les modules d'un projet). Le docstring général du module se place juste après ce bloc, il contient un descriptif de tous les éléments (leur utilisation et dépendances);
- Des clauses d'importations (éléments des bibliothèques standards puis éléments de bibliothèques utilitaires et en fin les importations spécifiques au projet);
- Des variables globales (peuvent être réunies par thèmes séparés par un saut de ligne);
- Des fonctions et classes (l'ordre est en général guidé par la logique les interactions).

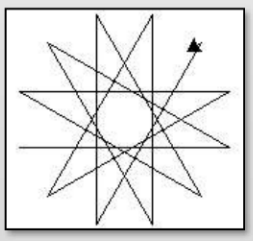
Les paquets

- Un paquet est une structurer en arborescence de répertoires utilisée pour regrouper des modules (utilisation dans une directive d'importation au même titre qu'un module;
- Chaque répertoire faisant partie d'un paquet doit posséder un fichier __init__.py pour que l'interpréteur le prenne en compte. Ce fichier peut être vide ou contenir du code d'initialisation qui est exécuté dès que le répertoire est trouvé dans une directive ;
- Lorsqu'un appel à from Paquet import * est fait, l'interpréteur n'importe que les éléments trouvés dans le fichier __init__.py du répertoire. Pour importer tous les modules du répertoire, il faut les définir explicitement dans une variable globale __all__ dans le fichier __init__.py du répertoire d'importation.

Les modules

Exemple: Le module « turtle » permet de réaliser des « graphiques tortue » (déplacements par contrôler à l'aide d'instructions simples).

```
from turtle import *
reset()
a = 0
while a <12:
    a = a +1
    forward(150)
    left(150)
```



Principales fonctions du le module turtle	
reset()	On efface tout et on recommence
goto(x, y)	Aller à l'endroit de coordonnées x, y
forward(distance)	Avancer d'une distance donnée
backward(distance)	Reculer
up()	Relever le crayon (pour pouvoir avancer sans dessiner)
down()	Abaisser le crayon (pour recommencer à dessiner)
color(couleur)	couleur peut être une chaîne prédéfinie ('red', 'blue', etc.)
left(angle)	Tourner à gauche d'un angle donné (exprimé en degrés)
right(angle)	Tourner à droite
width(epaisseur)	Choisir l'épaisseur du tracé
fill(1)	Remplir un contour fermé à l'aide de la couleur sélectionnée
write(texte)	texte doit être une chaîne de caractères

Module : Programmation Python

Chapitre 2: programmation objet en Python

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

Département Génie Informatique & Mathématiques

Les Classes en Python

- Le mot réservé **class** sert à définir un modèle en associant des attributs et des méthodes (placées dans le niveau d'indentation de la classe) à un nom

```
class <identifiantClass>:
    """ Documentation """
    attribut1=valeur1  # attributs de classe
    …
    def __init__(self[, param1, param2, ...]):
        """ Documentation """
        self.attr1 = ...
        self.attr2 = ...
    def identifiantMethode1(self[, param1, param2, ...]):
        …
```

- Les paramètres d'une méthode commencent obligatoirement par l'objet sur lequel s'applique la méthode (on utilise généralement **self**).
- Le constructeur « **__init__** » est utilisé pour initialiser les attributs d'objets de la classe
- La fonction **dir** renvoie une liste comprenant le nom des attributs et méthodes de l'objet qu'on lui passe en paramètre.

Les Classes en Python

- Une classe est souvent définit dans un module (un module peut contenir plusieurs classes).
- Les attributs d'instance sont définis dans le constructeur (« __init__ »).

Exemple

```
class Point2D:
    def __init__(self, x, y, identifiant=None):
        self._x = x
        self._y = y
        self._id = identifiant
    def get_x(self):
        return self._x
    def set_x(self, x):
        self._x = x
    def get_y(self):
        return self._y
    def set_y(self, y):
        self._y = y
    def get_id(self):
        return self._id
```


Instanciation et appel de méthodes

- La création d'un objet (instanciation) est obtenue par l'identifiant de la classe suivi, entre parenthèses, des paramètres effectifs à donner au constructeur (méthode spéciale « __init__ »);

Exemple :

```
pt=Point2D(10,20)
for st in dir(pt):
    if not st.startswith("__"):
        print(st)
```

- Pour invoquer une méthode, il est possible d'utiliser une des notation suivantes :
  - instanceObjet.identifiantMethode (...)
  - IdentifiantClass.identifiantMethode (instanceObjet, …) # Appel fonction

Exemple :

```
print( pt.get_x())
print(Point2D.get_x(pt))
```

Les attributs d'instance

- Les attributs d'instance (utilisation de self), crées dans les méthodes, restent spécifiques aux instances.
- Python se réfère à la notion d'espace de noms, ou namespace, crée lorsque l'instance de classe est utilisée. Cet espace est un dictionnaire propre à l'instance de classe et comporte les correspondances entre noms d'attributs et valeurs de ces attributs;
- Par défaut, tous les objets d'une classe posséderont un attribut spécial __dict__. Cet attribut est un dictionnaire qui contient en guise de clés les noms des attributs et, en tant que valeurs, les valeurs des attributs;

  instance.nAttribut= valeur est équivalent à instance.dict ["nAttribut"]= valeur
- Lorsque un attribut d'une instance est utilisé dans le cadre d'une affectation de valeur et que cet attribut est inexistant, il sera ajouté dans la liste des attributs de l'instance. Les autres instances ne profitent pas de ce nouvel attribut (assignation dynamique des variables);

Les attributs d'instance

- La fonction **hasattr(instanceObjet, nameAttibut)** retourne True si instanceObjet possède un attribut nameAttibut. Cette fonction fait appel à getattr et retourne False en cas d'erreur.
- La fonction **delattr(instanceObjet, nameAttibut)** Effacer l'attribut nameAttibut de l'instance désigne par **instanceObjet**

Exemple:

```
class A:
    def __init__(self, at): self.at=at
    def addAttrib(self, nameAt, valAt): self.name=valAt
    def getAt(self):
        if hasattr(self,"at" ): return self.at
        else:   return "Attribut inexistant"
    def getAllAttrib(self): return self.__dict__
a=A(5)
b=A(6)
a.addAttrib("atA1",10)
a.atA2=11
print(a.getAllAttrib()) # affiche {'at': 5, 'name': 10, 'atA2':11}
print(b.getAllAttrib())# affiche {'at': 6}
delattr(a,'at')
# print(a.at)# AttributeError: 'A' object has no attribute 'at'
print(a.getAt())# affiche Attribut inexistant
```

Les attributs et les méthodes de classe

- Les attributs placés au niveau de la classe (à l'extérieur des méthodes) sont appelés attributs de classe, ils ont des valeurs communes à toutes les instances. L'accès à ces attributs est donné par : nomDeClasse.nomAttribut
- Les méthodes de classe travaillent indépendamment de toutes les instances. L'usage veut que leur premier paramètre soit noté **cls** en lieu et place de **self**.
- Les méthodes de classe peuvent être déclarée explicitement en utilisant la syntaxe: nomMethode= classmethod(nomMethode)

Exemple:

```
class CompteBancaire:
  decouvert=0 #attribut de classe ( CompteBancaire.__dict__ )
  def __init__(self,solde=0): self.solde=solde #att instance
  def retirer(self,montant):
      if(self.solde-montant>=CompteBancaire.decouvert):
        self.solde=self.solde-montant
      else: print ("solde insuffisent")
  def setDecouvert(cls, seuil):
      cls.decouvert = -seuil
  setDecouvert = classmethod(setDecouvert)
c1=CompteBancaire(1000) ; c2=CompteBancaire(800)
c2.retirer(1000)#soldeinsuffisant
c1.setDecouvert(300); c2.retirer(1100)#succés
```


Les décorateurs

- Les méthodes statiques sont très proches des méthodes de classe, mais sont plus à considérer comme des fonctions utilitaires au sein d'une classe. Contrairement aux méthodes de classe, les méthodes statiques ne recevront pas le paramètre cls.
- Les méthodes statiques peuvent être déclarée explicitement en utilisant la syntaxe: nomMethode= staticmethod(nomMethode).
- Pour simplifier l'écriture de définition des méthodes de classe et les méthodes statiques, il est possible d'utiliser les décorateurs : **classmethod** et **staticmethod.**

```
class CompteBancaire:
    ...
    @classmethod
    def _getDecouvert(cls):
        return cls.decouvert
    @staticmethod
    def setDecouvert(seuil):
      CompteBancaire.decouvert = -seuil
```

```
c1=CompteBancaire(1000) ;
c2=CompteBancaire(800)
c2.retirer(1000)#soldeinsuffisant
print(c1._getDecouvert())
CompteBancaire.setDecouvert(300)
print(c1._getDecouvert())
c2.retirer(1100)#succé
```

Les attributs en Python

- Il n'y a pas de mot clé pour la visibilité des méthodes ou des attributs, mais si l'identifiant commence par un :
  - _ alors il est protégé : n'apparaît pas dans la documentation (help)
  - __ alors il est non accessible
- Les propriétés permettent de contrôler l'accès à certains attributs d'une instance(à définir dans le corps de la classe).

  nom_propriete = property(methode_accesseur, methode_mutateur).
- L'utilisation de « objet.nom_propriete » permet de lire (appel d'accesseur) ou de modifier (appel du mutateur) l'attribut protégé.

Exemple:

```
class Personne :
  def __init__ (self , nomPrenom, id=None ):
     self.__id = id
     self.nomPrenom = nomPrenom
  def _getId( self ):
     print("Appel getId")
     return self .__id
  def _setId (self , nouvId ):
     self .__id = nouvId
     print("Appel setId")
  id = property ( _getId ,_setId)
```

```
p=Personne("python","CI")
p._setId("11" )
print(p.__id) --->erreur
print(p._getId())
print(help(Personne))
p.id= "11"
Print(p.id)
p.nat="Maroc"
```


style de code 'pythonic'

- Les getters et setters ne sont pas justifiés, car les données membres sont forcément accessibles (pas de notion de contrôle d'accès).
- Le style de code 'pythonic' propose d'accéder directement aux données membres depuis l'objet.
- Le style de code 'pythonic' ne préserve pas l'encapsulation. Python propose un mécanisme dit de 'properties' pour traiter ce concept.

```
class Personne :
  ...
  def _getId( self ):
     ...
  def _setId (self , nouvId ):
     ...
  id = property ( _getId ,_setId)

p=Personne("python", "CI")
p.id="11"   ------> Appel setId
print(p.id) ------> Appel getId
print(p.__dict__)
```

Méthodes spéciales : Accès aux attributs de l'objet

- Python propose des méthodes spéciales (à définir dans d'une classe) pour pouvoir exécuter des actions personnalisées sur certains objets.
- Lorsque une écriture de type « objet.attribut » est rencontrée, l'interpréteur utilise le dictionnaire interne __dict__ pour rechercher cet attribut.

Méthode	Utilisation
__del__(self)	Appelée au moment de la destruction de l'objet.
__str__(self)	Utilisée pour convertir un objet en chaîne de caractère (afficher l'objet avec print par exemple). Par défaut, si aucune méthode __str__ n'est définie, Python appelle la méthode __repr__ de l'objet.
__getattr__(self, identifiantAttribut)	Appelée lors d'un accès en lecture d'un attribut (la méthode reçoit en paramètre le nom de l'attribut, sous la forme d'une chaîne de caractères).
__setattr__(self, identifiantAttribut, valeurAttribut)	Appelée lors de la modification d'un attribut (instruction objet.nom_attribut = nouvelle_valeur)
__delattr__(self, identifiantAttribut)	Appelée quand on souhaite supprimer un attribut de l'objet (exécuter del objet.attribut par exemple).

Méthodes spéciales : Accès aux attributs de l'objet

```
class Person:
  def __getattr__(self, name):
    print('getattr %s' % name)
    if name in self.__dict__:
      return self.__dict__[name]
    else:
      print("attribut '%s' inexistant" % name)
  def __setattr__(self, name, valeur):
    print('set %s: %s' % (name, str(valeur)))
    self.__dict__[name] = valeur
  def __delattr__(self, name):
    print('del %s' % name)
    if name in self.__dict__:
      del self.__dict__[name]
    else: print("attribut '%s' inexistant" % name)
```

```
p = Person()
p.first_name = "CI"
p.age = 2
print(p.last_name)
p.last_name = 'ENSAB'
del p.age
print(p.age)
```

```
set first_name:CI
set age: 2
getattr last_name
attribut 'last_name' inexistant
set last_name: ENSAB
del age
getattr age
attribut 'age' inexistant
```

Méthodes spéciales : Utilisation de l'objet comme conteneur

Méthode	Utilisation
__len__(self)	Retourne la taille d'un objet conteneur (grâce à la fonction len)
__contains__(self, objet)	Tester si un objet se trouve dans un conteneur. En cas d'existence, la méthode doit renvoyer True ; sinon False.
__getitem__(self, index), __setitem__(self, index) et __delitem__(self, index)	Utilisée lorsque on exécute (respectivement) une instruction de type objet[index], objet[index] = valeur et del objet[index]. Avec **__setitem__**, les mappings ajoutent automatiquement la clé lorsqu'elle n'existe pas, contrairement aux séquences qui retournent une erreur si la l'index n'existe pas.

Exemple:

```
class ZDict :
    def __init__ ( self ):
        self.__dictionnaire = {}
    def __getitem__ (self , index ):
        return self.__dictionnaire [ index ]
    def __setitem__ (self , index , valeur ):
        self.__dictionnaire [ index ] = valeur
    def __str__ (self  ):
        return str(self.__dictionnaire.items())
```

```
zd=ZDict()
zd['a']=11
zd['b']=22
print(zd['a'])
```

dict_items([('a', 11), ('b', 22)])

Méthodes spéciales de surcharge des opérateurs

- Les méthodes de surcharge des opérateurs **+**, **-**, *, /, // (division entière), **%** (modulo), ** (puissance) sont respectivement les suivantes: **__add__** , **__sub__** , **__mul__** , **__truediv__** , **__floordiv__** , **__mod__** , **__pow__**
- Pour toutes ces méthodes, l'instance de leur classe est située à gauche de l'opérateur: l'instruction « objet opérateur other » est équivalent à objet.surchargeMethode (other).

Exemple:

```
class Duree :
  def __init__(self, min=0, sec=0):
    self.min = min  # Nombre de minutes
    self.sec = sec  # Nombre de secondes
  def __str__(self):
    return "{0:02}:{1:02}".format(self.min,self.sec)#forme MM:SS
  def __add__ (self , objet_a_ajouter ):
    nouvelle_duree = Duree ()
    nouvelle_duree.min = self.min
    nouvelle_duree.sec = self.sec
    nouvelle_duree.sec += objet_a_ajouter
    if nouvelle_duree.sec >= 60:
            nouvelle_duree.min += nouvelle_duree.sec // 60
            nouvelle_duree .sec = nouvelle_duree .sec % 60
    return nouvelle_duree
```

```
d1 = Duree (12 , 8)
print (d1)
d2 = d1 + 54
print (d2)
```

Méthodes spéciales de surcharge des opérateurs

- La variation R des méthodes spéciales (**__radd__ , __rsub__** , etc):
  - Ces méthodes permettent de définir des opérateurs inversés (l'instance de la classe est située à droite de l'opérateur):
  - un appel de « other opérateur objet » déclenche objet.Rmethode(other).
- La variation I des méthodes spéciales (**__iadd__** , **__isub__** , etc) :
  - Elles sont Utilisées dans le cas des operateurs arithmétiques abrégés
  - « instance += objet2 » est équivalent à « instance. iadd (objet2) »
- Les méthodes spéciales utilisées pour les opérateurs unaires sont:
  - **__pos__** (self) pour l'operateur +
  - **__neg__** (self) pour l'operateur -
  - **__abs__** (self) pour la fonction abs
- L'utilisation d'une fonction de transtypage numérique appelle une méthode spéciale
  - **__int__** (self) pour la fonction int
  - **__round__** (self) pour la fonction round
  - **__float__** (self) pour la fonction float
- Tout objet dont la classe possède la méthode **__call__** (self[,arg, ...]) peut être utilisée comme une fonction: pour pt=Point2D(1,2); pt(10,20) est équivalente à pt.__call__(10,20)

Méthodes spéciales de surcharge des opérateurs

Opérateur	Méthode spéciale	Résumé
==	def __eq__(self, objet_a_comparer)	Opérateur d'égalité (equal). Renvoie True si self et objet_a_comparer sont égaux, False sinon.
!=	def __ne__(self, objet_a_comparer)	Différent de (non equal). Renvoie True si self et objet_a_comparer sont différents, False sinon.
>	def __gt__(self, objet_a_comparer)	Teste si self est strictement supérieur (greater than) à objet_a_comparer.
>=	def __ge__(self, objet_a_comparer)	Teste si self est supérieur ou égal (greater or equal) à objet_a_comparer.
<	def __lt__(self, objet_a_comparer)	Teste si self est strictement inférieur (lower than) à objet_a_comparer.
<=	def __le__(self, objet_a_comparer)	Teste si self est inférieur ou égal (lower or equal) à objet_a_comparer.

Exemple:

```
class Duree :
  …
 def __eq__ (self , autre_duree ):
   """ Test si self et autre_duree sont égales """
   return self.sec==autre_duree.sec and self.min==autre_duree.min
 def __gt__ (self , autre_duree ):
   nb_sec1 = self .sec + self .min * 60
   nb_sec2 = autre_duree .sec + autre_duree.min * 60
   return nb_sec1 > nb_sec2
```

Méthodes spéciales de surcharge des opérateurs

- La méthode **__cmp__** peut être utilisée par tous les opérateurs de comparaison lorsque l'objet est impliqué. Elle doit renvoyer :
  - un entier négatif si self est inférieur à other ;
  - un entier positif si self est supérieur à other ;
  - zéro en cas d'égalité.
- La méthode **__hash__** est appelée par la primitive hash() ou par un objet dictionnaire lorsque l'objet est utilisé comme clé. Elle doit renvoyer un entier de 32 bits. Si deux objets sont définis comme égaux, par __cmp__(), __eq__() ou __ne__(), __hash__() doit renvoyer la même valeur pour ces deux objets.
- La methode **__nonzero__** est appelée par la primitive bool() et par la comparaison avec True ou False. Elle doit renvoyer True ou False. Lorsque cette méthode n'est pas définie, c'est __len__() qui est utilisée. __len__() représente la taille de l'objet. Si aucune des deux méthodes n'est présente, l'objet est toujours considéré comme vrai.

L'Héritage en Python

- Syntaxe de définition d'une classe fille (sous classe ou clase dérivée) :
  class <identifiantClassFille>(<**identifiantClassMere**>):
  …
- L'accès à une méthode de la super classe est obtenu en préfixant son identifiant de l'identifiant de la super classe et en mettant self en premier paramètre effectif.
  Exemple:

```
class Personne:
    def __init__ (self , nom , prenom):
        self.nom = nom
        self.prenom = prenom
    def __str__ ( self ):
        return "{0} {1}".format ( self.nom, self.prenom  )
class AgentSpecial ( Personne ):
    def __init__ (self , nom , prenom, matricule ):
        Personne.__init__(self, nom, prenom)
        self.matricule = matricule
    def __str__ ( self ):
        return " Agent {0} matricule {1}".format (
                    Personne.__str__(self), self.matricule )
ag=AgentSpecial("ENSAB","GI",2019)
print(ag)
```

L'Héritage en Python

- Règle de surcharge :
  - Suite à une invocation, la méthode sélectionnée est toujours celle qui est la plus proche (au sens de l'héritage) de la classe de l'objet (de même pour les attributs).
  - Indépendamment de cette règle, il est possible d'invoquer une méthode spécifique en précisant la classe de cette méthode : ClasseDeBase.methode(self, parametre1,…).
- La fonction **issubclass** vérifie si une classe est une sous-classe d'une autre classe (exemple : « issubclass (AgentSpecial , Personne) » renvoie True);
- La fonction **isinstance** permet de savoir si un objet est issu d'une classe ou de ses classes filles;
- La classe au sommet de l'héritage est la classe Object qui doit être précisée si **c'est la superclasse de la classe à définir** (python 3)
- En python, il n'y a pas de classe abstraite (non définis dans le cœur de Python)

Le polymorphisme en Python

Dans le contexte de la programmation orientée objet, le polymorphisme est un concept qui fait référence à la capacité d'un objet à prendre plusieurs formes.

Exemple :

```
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)
photo1 = rectangle(3,4)
print(photo1.surface())
photo2 = paveDroit(3,4,10)
photo2= rectangle(3,4)
print(photo2.surface())
```

⇨

```
12
12
```

L'héritage multiple

- Python supporte l'héritage multiple en laissant la possibilité de lister plusieurs classes parentes dans la définition. On peut connaitre les superclasses d'une classe à l'aide de son attribut spécial __bases__
- Dans le cas de l'héritage multiple, le mécanisme de recherche des attributs et méthodes est appliquée à chacune des classes de base, de gauche à droite. l'interpréteur parcourt l'arbre de dérivation d'une de ces classes comme dans le cas d'un héritage simple, puis passe à la classe de base suivante si l'attribut n'a pas été trouvé.
- Lorsque des classes parentes ont une classe de base commune, il devient difficile de maîtriser les enchaînements d'appels et d'avoir une bonne visibilité (l'utilisation de l'héritage multiple est délicate et fortement déconseillée dans la plupart des cas).

L'héritage multiple

```
class A:
  def __init__(self):
    print("Début init A")
    super().__init__()
    print(super().__str__())
    print("Fin init A")
  def __str__(self):
    return "Classe A"
class B:
  def __init__(self):
    print("Début init B")
    super().__init__()
    print(super().__str__())
    print("Fin init B")
  def __str__(self):
    return "Classe B"
class C(A, B):
  def __init__(self):
    print("Début init C")
    super().__init__()
    print("Fin init C")
class D(B, A):
  def __init__(self):
    print("Début init D")
    super().__init__()
    print("Fin init D")
```

```
c=C()
d=D()
```

```
Début init C
Début init A
Début init B
<__main__.C object at 0x00B25350>
Fin init B
Classe B
Fin init A
Fin init C
Début init D
Début init B
Début init A
<__main__.D object at 0x00B25370>
Fin init A
Classe A
Fin init B
Fin init D
```

Les classes abstraites

- Les classes abstraites ne font pas partie du coeur de Python, mais sont disponibles via un module de la bibliothèque standard, abc (Abstract Base Classes). Ce module contient notamment la classe ABC et le décorateur abstractmethod, pour définir respectivement une classe abstraite et une méthode abstraite de cette classe.
- Une classe abstraite doit donc hériter d'ABC, et utiliser le décorateur cité pour définir ses méthodes abstraites.

Exemple

```
import abc
class MyABC(abc.ABC):
    @abc.abstractmethod
    def myMethodAbstact(self):
        pass
```


Les exceptions en python

- Une exception correspond à une erreur qui s'est produite (l'interpréteur stoppe l'exécution du programme en cours et affiche l'erreur).
- Le message affiché contient en général le traceback (pile d'appel : le chemin parcouru) , le type d'exception levée et un message explicite sur le problème rencontré.
- Python permet de séparer le code de gestion des erreurs avec le code métier (code plus lisible)
- Une exception est dite levée lorsqu'une erreur apparaît alors qu'elle est dite capturée lorsqu'elle est gérée et traitée.

Lever d'une exception

- Une exception est levée à l'aide l'instruction **raise** suivie de la création d'une instance d'une sous classe de Exception;
- Le constructeur de la sous classe de Execption prend au moins en paramètre une chaîne de caractères explicitant l'erreur.

Exemple

```
class ClasseAbstraite():
  def __init__(self):
    raise NotImplementedError("Instanciation classe abstraite")
```

```
c=ClasseAbstraite()
```

⇨ NotImplementedError: Instanciation classe abstraite

Intercepter une exception

- L'instruction **try** est utilisée pour délimitée la suite d'instructions susceptible de lever une exception;
- Le bloc **try** comporte au moins une clause **except** (possibilité d'enchaîner plusieurs blocs **except**) suivie de la classe de l'exception devant être interceptée (avec possibilité de définir l'instance de l'exception: séparation par virgule ou as);
- La clause **else** (placée après toutes les clauses **except**) est exécutée lorsque aucune exception des clauses **except** n'a été interceptée;
- La clause finally termine l'instruction try. Son code est toujours exécuté à la fin (qu'il y ait eu une exception ou pas).

Exemple

```
class ClasseAbstraite():
  def __init__(self):
    raise NotImplementedError("Instanciation classe abstraite")
try:
    c=ClasseAbstraite()
except NotImplementedError as nImp:
    print(nImp.__str__())
```

Les assertions

- Les assertions sont un moyen simple de s'assurer, avant de continuer, qu'une condition est respectée (utilisé généralement dans des blocs try ... Except);
- L'instruction « **assert** test » permet de poursuivre l'exécution du programme normalement si test renvoie True, sinon, une exception « **AssertionError** » est levée.

Exemple 1:

```
import datetime
anneeCourant = datetime.datetime.now().year
try:
  annee = int( input(" Saisissez l'année"))
  assert annee == anneeCourant or annee == anneeCourant-1
except ValueError :
  print (" Vous n'avez pas saisi un nombre.")
except AssertionError :
  print ("L'année saisie doit être encours ou précédente ")
```


Les assertions

Exemple 2:

```
class CompteBancaire :
    def __init__(self, banque, titulaire, solde):
        assert isinstance(banque,str)and len(banque)>0
        assert isinstance(titulaire,str)and len(titulaire)>0
        assert isinstance(float(solde),float)and solde>=0
        self.banque=banque
        self.titulaire=titulaire
        self.solde=solde
...
try:
    cb=CompteBancaire(...)
except Exception as e:
    print(e)
```

Les exceptions prédéfinies

La classe Exception est la classe de base de toutes les exceptions (qui hérite elle-même de BaseException). On distingue deux types d'exceptions :

- Les erreurs qui provoquent l'arrêt de l'exécution du code et doivent être interceptées par une directive try … except.
- Les avertissements, dérivés de l'exception Warning, utilisés avec la fonction warn du module warnings, et qui se contentent dans ce cas d'afficher un message d'avertissement sans interrompre l'exécution du programme (provoquent l'arrêt du programme si elles sont utilisées directement avec une directive raise).

Hiérarchie des exceptions

https://docs.python.org/fr/3/library/exceptions.html

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
```

```
+-- Exception
     +-- OSError
     |    +-- BlockingIOError
     |    +-- ChildProcessError
     |    +-- ConnectionError
     |    |    +-- BrokenPipeError
     |    |    +-- ConnectionAbortedError
     |    |    +-- ConnectionRefusedError
     |    |    +-- ConnectionResetError
     |    +-- FileExistsError
     |    +-- FileNotFoundError
     |    +-- InterruptedError
     |    +-- IsADirectoryError
     |    +-- NotADirectoryError
     |    +-- PermissionError
     |    +-- ProcessLookupError
     |    +-- TimeoutError
     +-- ReferenceError
     +-- RuntimeError
     |    +-- NotImplementedError
     |    +-- RecursionError
     +-- SyntaxError
     |    +-- IndentationError
     |         +-- TabError
```

```
+-- Exception
     +-- SystemError
     +-- TypeError
     +-- ValueError
     |    +-- UnicodeError
     |         +-- UnicodeDecodeError
     |         +-- UnicodeEncodeError
     |         +-- UnicodeTranslateError
     +-- Warning
          +-- DeprecationWarning
          +-- PendingDeprecationWarning
          +-- RuntimeWarning
          +-- SyntaxWarning
          +-- UserWarning
          +-- FutureWarning
          +-- ImportWarning
          +-- UnicodeWarning
          +-- BytesWarning
          +-- ResourceWarning
```

Création d'exceptions personnalisées

- Pour créer un type d'exception il faut créer une sous classe (nom de préférence se terminant par Error ou Erreur) de la classe Exception (importée par défaut en python 3).
- La classe d'exception crée comporte souvent (utilisation de pass si c'est vide) : un constructeur et une méthode __str__ (le constructeur prend généralement en paramètre un message d'erreur que la méthode __str__ renvoie).

Exemple

```
class MonException ( Exception ):
    """ Exception levée dans un certain contexte... à définir """
    def __init__ (self , message ):
        """ On se contente de stocker le message d'erreur """
        self.message = message
    def
... __str__ ( self ):
        return self.message
try:
    raise MonErreur ("OUPS ... ")
except MonErreur as er:
    print(er)
```


Université Hassan 1er
École Nationale des Sciences Appliquées – Berrechid

Module : Programmation Python

Chapitre 3: Calcul scientifique en Python

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

Calcul scientifique en python

- numpy est une bibliothèque numérique apportant le support efficace de larges tableaux multidimensionnels, et de routines mathématiques de haut niveau (fonctions spéciales, algèbre linéaire, statistiques, etc.)
  http://www.numpy.org/
- Scipy est un ensemble qui comprend de nombreux modules utiles pour le calcul scientifique (cluster , ga : genetic algorithms, integrate , optimize …). Scipy est construit à partir de Numpy, ce qui signifie qu'il faut avoir le module Numpy pour faire fonctionner le module Scipy.
  https://scipy.org - https://docs.scipy.org/doc/
- Le module matplotlib permet de visualiser en 2D des données (chargé de tracer les courbes). Matplotlib est la librairie de graphiques scientifiques de scipy.
  https://matplotlib.org
- La librairie Pandas est spécialisée dans l'analyse des données.
  https://pandas.pydata.org - https://pandas.pydata.org/docs/

NB. Pour installer un package dans un environnement , utiliser la commande : pip install package

Les tableaux array de NumPy

- La structure de base du numpy est les tableaux (homogène : même type, en général numérique) multidimensionnels ndarray (signifie «n-dimensional array»)
- Il est possible de construire des tableaux à partir de listes avec la fonction array.

```
import numpy as np
A = np.array([[1 , 2, 3], [4, 5, 6]])
```

⇒ `[[1, 2, 3], [4, 5, 6]]` $\quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

A est une matrice à 2 lignes et 3 colonnes

B= np.array([1, 4, 5, 8], float) ⟶ array([1., 4., 5., 8.])

- Les attributs ndim, shape et dtype permettent de connaître respectivement le nombre de dimensions d'un tableau, le nombre d'éléments dans chaque dimension (renvoie un tuple) et le type des éléments du tableau

  A.shape # retourne (2, 3) ⟶ 2 lignes et 3 colonnes
- Le référencement des éléments d'un tableau (slicing) est relativement similaire à celui des listes:
  - Si T est un tableau 1d , T[:n] est le sous-tableau d'indices 0 à n-1
  - Si U est un tableau 2d, U[:,1] désigne un tableau 1d correspondant à la colonne d'indice 1 de U alors que U[0,1:] référence le tableau 1d correspondant à la ligne d'indice 0 et aux colonnes à partir de la colonne d'indice 1 de U.

Les tableaux array de NumPy

Fonction	Description
linspace(a,b,n)	Créer un tableau 1d de n valeur uniformément reparties entre a et b
arange(a,b,inc)	Créer un tableau 1d comme range(a,b,inc) , les arguments peuvent être des flottants
zeros(nbl|(nbl, nbc)), ones(nbl | (nbl, nbc))	Créer des tableaux contenants des 0 ou des 1 (l'argument tuple définit les dimensions : nbLigne, nbColone, …)
zeros like(T) , ones like(T)	Initialiser des tableaux de 0 ou de 1 ayant les mêmes dimensions que le tableau T
eye(nb1 [, nb2])	Permet d'obtenir des tableaux \identites" np.eye(3) → array([[1., 0., 0.],[0., 1., 0.],[0., 0., 1.]]) np.eye(3,2) # surjection canonique de R^2 → R^3 → array([[1.,0.],[0.,1.],[0.,0.]])
random.rand (nbl [, nbc]) **random.randint (n1, n2, tupleDim)**	Le sous-module random permet d'obtenir des nombres aleatoires: np.random.rand(3) → array([-0.15114084, 0.09718998, -.81857774]) np.random.randint(0,9,5) # 5 entiers de [0,9[→ array([4, 2, 5, 4, 5]) np.random.randint(0,9,**(3,3)**) ⟶ array([[3, 6, 8], [6, 4, 0], [5, 3, 4]])

reshape(T, tupleDim)
T.flatten()
	Reshape change les dimensions d'un tableau à condition que le nombre total de composantes ne change pas (valeurs partagées avec le tableau initial , utiliser la méthode copy pour obtenir une copie de ce tableau).
	Flatten transforme (avec copie) un array 2D en un array 1D
	T = arange(12) # array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])
	TT = reshape(T, (3,4)) # array([[0,1,2,3], [4,5,6,7], [8, 9, 10, 11]])

Les tableaux array de NumPy

Fonction	Description
concatenate((T1,T2) [,axis=n])	Concaténation de tableaux de même nombre de dimension (mis dans un tuple comme premier argument avec un assemblage horizontale si axis=1) A= np.ones((2,2)) ; B = np.zeros((2,2)) np.concatenate((A,B)) ⟶ array([[1., 1.],[1., 1.],[0., 0.],[0., 0.]]) np.concatenate((A,B),axis=1) ⟶ array([[1., 1., 0., 0.],[1., 1., 0., 0.]])

diag (T, [k=n])	Construire des tableaux 2d en spécifiant une diagonale sous forme d'un tableau 1d ou bien d'extraire une certaine diagonale d'un tableau 2d sous la forme d'un tableau 1d:
	diag(ones(3)) # creation du tableau array([[1., 0., 0.],[0., 1., 0.],[0., 0., 1.]])
	diag(ones(3),k=1) # ici on précise la diagonale (par defaut k=0)
	# [[0., 1., 0., 0.],[0., 0., 1., 0.],[0., 0., 0., 1.],[0., 0., 0., 0.]]
	diag(array([[1,2,3],[4,5,6],[7,8,9]])) # extraction de array([1, 5, 9])
	diag(array([[1,2,3],[4,5,6],[7,8,9]]), k=-1) # extraction de array ([4, 8])

transpose(T) **T.transpose()**	Transposition d'un tableau (non modifié). Le tableau transpose partage ses éléments avec l'ancien tableau . A = random.rand (2,3) ⟶ array([[0.19865153, 0.01653091, 0.44841961], [0.33224291, 0.24958435, 0.68242376]]) B = A.transpose() ⟶ array([[0.19865153, 0.33224291], [0.01653091, 0.24958435], [0.44841961, 0.68242376]])

Opérations sur les tableaux array de NumPy

- L'opération arithmétique (opérateurs : +, -, *, /, …) entre deux tableaux réalise un calcul au sens élément par élément (dans le cas 2d, C = A*B est le tableau dont les éléments sont $C_{i;j}$ = Ai;j*Bi;j). Les deux tableaux opérandes doivent avoir la même forme (dimension et taille des dimensions)
- La fonction **dot** assure la multiplication matricielle de deux tableaux
  - Si A et B sont deux tableaux 2d, dot(A,B) réalise un produit matricielle si le nombre de colonnes de A est égal au nombre de lignes de B.
  - Si v est un vecteur (array 1d) et A une matrice (array 2d) alors np.dot(A,v) renvoie le produit Av (un tableau 1d en un vecteur colonne) tandis que np.dot(v,A) renvoie v^t A (un tableau 1d en un vecteur ligne)

    a=np.arange(4).reshape(2,2) ; v=np.array([-3,2])

    np.dot(a,v) ⟶ array([2, 0])

    np.dot(v,a) ⟶ array([4, 3])
  - Si x et y sont deux tableaux 1d de même taille np.dot(x,y) effectue le produit scalaire (x|y).

Opérations sur les tableaux array de NumPy

- L'opération relationnelle (opérateurs : ==, !=, < , <=, >, >=) entre deux tableaux réalise une comparaison au sens élément par élément. Ces opérateurs sont aussi disponible sous les noms de np.equal(a,b), np.not_equal(a,b), np.less(a,b), np.less_equal(a,b), np.greater(a,b), np.greater_equal(a,b)

  a=np.arange(16).reshape(4,4) → array([[0,1,2,3],[4,5,6,7],[8,9,10,11],[12,13,14,15]])

  b=6*np.eye(4,4)-np.diag(4+np.arange(3),1)+2*np.ones((4,4))

  → array([[8, -2, 2, 2], [2, 8, -3, 2], [2, 2, 8, -4], [2, 2, 2, 8]])

  (a<b) → array([[True, False, False, False], [False, True, False, False],

  [False, False, False, False], [False, False, False, False]])

  b[b<4] → array([-2, 2, 2, 2, -3, 2, 2, 2, -4, 2, 2, 2])
- Numpy permet de "vectoriser", i.e. appliquer une fonction à un vecteur/matrice et éviter les boucles.

  a=np.arange(4, dtype=float) → array([0., 1., 2., 3.])

  np.cos(a) → array([1. , 0.54030231, -0.41614684, -0.9899925])

  b = np.arange(4).reshape((2,2)) # → array([[0, 1], [2, 3]])

  np.amin(b) → élément minimum 0

  np.amin(b, axis=0) → minimum selon colonne : array([0, 1])

  np.amin(b, axis=1) → minimum selon ligne : array([0, 2])

  c=6*np.eye(4,4)-np.diag(4+np.arange(3),1)+2*np.ones((4,4))

  np.argmin(c,axis=1) → tableau d'indices des minimums : array([1, 2, 3, 0])

Exercice 1

On considère la matrice $A \in \mathcal{M}_{3,4}(\mathrm{R})$ définie par :

$$\begin{pmatrix} 4 & 6 & -2 & 3 \\ 2 & -1 & 0 & 1 \\ -7 & 0 & 1 & 12 \end{pmatrix}.$$

(a) Définir la matrice A comme un `np.array()`.

(b) Modifier la matrice A pour que ses deux premières lignes soient multipliées par 2 et que sa dernière colonne soit divisée par 3.

(c) Créer une nouvelle matrice B définie par

$$\begin{pmatrix} 4 & 5 & 6 \\ 5 & 10 & 15 \\ 1 & 1 & 1 \end{pmatrix},$$

en utilisant le fait que les lignes 1 et 2 sont composées des éléments successifs de deux suites arithmétiques (voir fonction `np.arange` et `np.ones()`).

(d) Créer la matrice $C \in \mathcal{M}_{3,3}(\mathrm{R})$ extraite de A telle que pour $1 \leqslant i, j \leqslant 3$, $c_{ij} = a_{ij}$.

(e) Différents produits matriciels

- Réaliser le produit matriciel D de B et A (`np.dot()`).
- Réaliser le produit d'Hadamard E de B et de C.

Pour mémoire, le produit d'Hadamard $E \in \mathcal{M}_{3,3}(\mathrm{R})$ des matrices $B \in \mathcal{M}_{3,3}(\mathrm{R})$ et $C \in \mathcal{M}_{3,3}(\mathrm{R})$ est défini par

$$\forall 1 \leqslant i, j \leqslant 3, \quad e_{ij} = c_{ij} b_{ij}.$$

Calcul numérique matriciel

- **linalg.matrix_rank** retourne le rang d'une matrice
  a=np.arange(15).reshape(3,5)
  → array([[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14]])
  np.linalg.matrix_rank(a) → 2
- **linalg.inv** retourne l'inverse d'une matrice carrée (tester si c'est inversible)
  a=np.array([2,4,6,8],float).reshape(2,2)
  np.linalg.inv(a) → array([[-1. , 0.5], [0.75, -0.25]])
- **linalg.solve (a,b)** assure la résolution d'un système linéaire pour lequel a est une matrice carrée et b un vecteur ou une matrice (avec condition de compatibilité, Il faut tester l'existence de solution)
  a=np.array([2,4,6,8],float).reshape(2,2)
  b=np.array([1, 4],float)
  np.linalg.solve(a,b) → array([1. , -0.25])
- **np.linalg.det** calcul le déterminant d'une matrice carrée
- **np.linalg.eig** calcul des valeurs propres et vecteurs propres d'une matrice carrée(1ier argument retourné : valeurs propres, 2ème argument =matrice des vecteurs propres en colonne càd une colonne est un vecteur propre).
  a=np.array([2,4,6,8],float).reshape(2,2)
  np.linalg.eig(a) → (array([-0.74456265, 10.74456265]), array([[-0.82456484, -0.41597356], [0.56576746, -0.90937671]]))

Exercice 2

On considère la matrice $A \in \mathcal{M}_{4,4}(\mathrm{R})$ suivante

$$\begin{pmatrix} 4 & 5 & 6 & -1 \\ 5 & 10 & 15 & 2 \\ 6 & 15 & 1 & 4 \\ -1 & 2 & 4 & -2 \end{pmatrix}.$$

(a) Pourquoi A est-elle diagonalisable ? À l'aide de la fonction `np.linalg.eig()`, calculer avec Python ses valeurs propres et donner une base de vecteurs propres.

(b) Calculer de deux manières l'inverse de A en utilisant le résultat précédent et la fonction `np.linalg.inv`. Comparer les résultats obtenus

Exercice 3

Exercice 3 On s'intéresse à la matrice $A \in \mathcal{M}_{3,5}(\mathrm{R})$

$$\begin{pmatrix} 1 & -1 & 2 & 1 & 2 \\ -1 & 2 & 3 & -4 & 1 \\ 0 & -1 & 1 & 0 & 0 \end{pmatrix}.$$

1. Quel est le rang de la matrice A? Utiliser la fonction `np.linalg.matrix_rank` afin de retrouver ce résultat.
2. En définissant :

$$A = \begin{pmatrix} 1 & -1 & 2 \\ -1 & 2 & 3 \\ 0 & -1 & 1 \end{pmatrix}, \text{ et } b = \begin{pmatrix} 3 \\ -7 \\ 1 \end{pmatrix},$$

résoudre à l'aide de la fonction `np.linalg.solve()` le système $Ax = b$.

La bibliothèque scipy

scipy est une bibliothèque numérique d'algorithmes et de fonctions mathématiques, basée sur les tableaux numpy.ndarray, complétant ou améliorant (en termes de performances) les fonctionnalités de numpy.

Sous modules	Description
constants	Constantes physiques ou mathématiques
fftpack	Routines de FFT (Transformation de Fourier Rapide)
special	Fonctions spéciales (fonctions de Bessel, gamma, etc.).
integrate	Intégration numérique et résolution d'équations différentielles ordinaires
optimize	Routines d'optimisation (minimisation, moindres-carrés, zéros d'une fonction,
interpolate	Routines d'interpolation et de lissage (smoothing splines)
linalg	Algèbre linéaire
signal	Traitement du signal (convolution, corrélation, filtrage, ondelettes, etc.).
stats	Fonctions et distributions statistiques
ndimage	Traitement d'images multidimensionnelles.

La bibliothèque graphique matplotlib

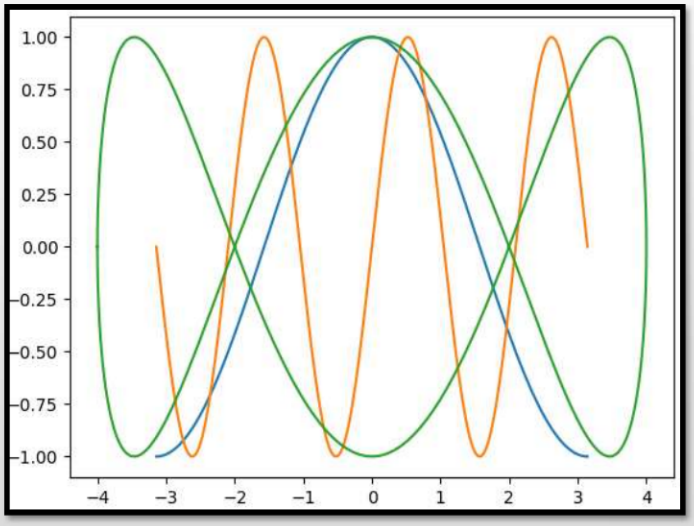
- matplotlib est une bibliothèque graphique de visualisation 2D (avec un support pour la 3D, l'animation et l'interactivité)
- Le module pyplot de matplotlib est l'un de ses principaux modules. Il regroupe un grand nombre de fonctions qui servent à créer des graphiques et les personnaliser (travailler sur les axes, le type de graphique, sa forme et même rajouter du texte)

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

- Les fonctions de base de pyplot sont:
  - figure(nom_figure) définit une fenêtre graphique.
  - plot(…): permet de représenter de nouvelles fonctions en reliant par une lignes les couples de points correspondant à leurs abscisses et ordonnées (transmit successivement sous forme de tableaux).
  - show(): déclenche l'affichage à l'écran (arrêt de l'exécution du script en cours jusqu'à la fermeture de la fenêtre d'affichage).
  - savefig(fichier): permet de sauvegarder une figure dans un fichier (le format par défaut est PNG)
  - clf(): efface et réinitialise la figure en cours (clf pour « CLear Figure »)
  - close(): sert à fermer la fenêtre qui s'est ouverte avec show

La bibliothèque graphique matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
# échantillonnage de l'axe des x: debut, fin , pas
X = np.linspace(-np.pi,np.pi,256)
# utilisation des fonctions de numpy
C = np.cos(X)
S = np.sin(3*X)
# Fonctions matplotlib
plt.figure("figures graphiques")
plt.plot(X,C)
plt.plot(X,S)
plt.plot(4*C,S)
plt.show()
plt.savefig('cos_sin.png')
plt.clf()
```

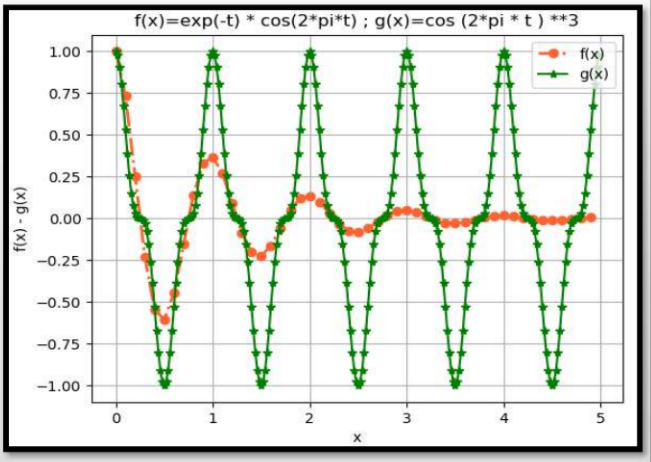


La bibliothèque graphique matplotlib

- Pour gérer le style d'un tracé, la fonction **plot** possède les paramètres supplémentaires suivants (il est possible de combiner la valeur de ces paramètres sous forme d'un 3ème argument de la fonction):
  - Le paramètre **color** pour changer la couleur du tracé (utiliser un nom ou une abréviation pour les couleurs primaires ou un tuple des valeurs RGB de la couleur)
  - Le paramètre **linestyle** pour modifier le style du tracé (valeur : "-" , "--" , "-.").
  - Le paramètre **marker** ajoute des marqueurs au tracé (par exemple : *, +, o)
  - Le paramètre **lw** (linewidth) permet de changer l'épaisseur du tracé.
- La fonction **grid** affiche un quadrillage en pointillés dont le style peut être défini par les paramètres suivants:
  - Le paramètre axis pour choisir l'axe de quadrillage (valeur: both, x ou y)
  - Des paramètres color, linewidth et linestyle (tout comme plot)
- La fonction title("…") définit le titre du graphique alors que la fonction legend("…",…) permet d'attribuer des légendes aux différents tracés.
- Les fonctions xlabel et ylabel permet de donner respectivement un nom à l'axe des abscisses et à l'axe des ordonnées.

La bibliothèque graphique matplotlib

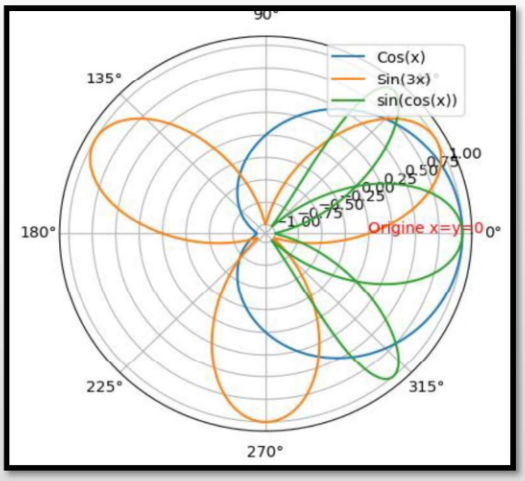
```
def f(t):
  return np.exp(-t) * np.cos(2*np.pi*t)
def g(t):
  return np.cos (2*np.pi * t ) **3
plt.figure ("Exemple de tracage de fonction")
plt.title ("f(x)=exp(-t) * cos(2*pi*t) ; g(x)=cos (2*pi * t ) **3")
plt.xlabel("x");
plt.ylabel("f(x) - g(x)")
plt.grid("both")
t1 = np.arange(0.0, 5.0, 0.1);
t2 = np.arange(0.0, 5.0, 0.02)
line1= plt.plot(t1, f(t1),
  color = (1, 100/255, 50/255) ,
  linestyle='-.', marker="o",
  lw=2 , label="f(x)" )
line2, =plt.plot(t2, g(t2),
  'g-*',label="g(x)")
plt.legend(handles=[line1[0], line2],
   loc='upper right')
plt.show()
```



La bibliothèque graphique matplotlib

- La fonction axis permet de réaliser plusieurs opérations sur les axes: Utiliser axis([xmin, xmax, ymin, ymax]) pour un cadrage et axis("equal") pour un échelle isometrique.
- La fonction text(x, y, text, color = ...) permet de placer du texte dans le dessin (x et y sont les coordonnées du premier caractère de text)

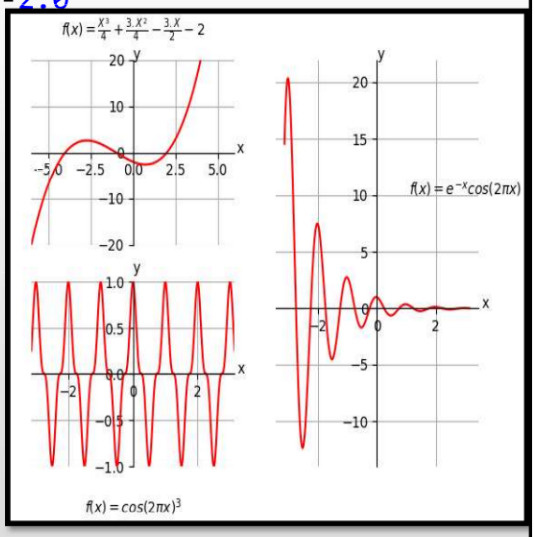
```
X = np.linspace(-np.pi,np.pi,256)
C = lambda X: np.cos(X)
S = lambda X: np.sin(3*X)
# Passage aux coordonnées polaires
plt.axes(polar=True)
line1,=plt.plot(X, C(X),
   label="Cos(x)")
line2,= plt.plot(X, S(X),
   label="Sin(3x)")
line3,= plt.plot(C(X), S(X),
   label="sin(cos(x))")
plt.legend(handles=[line1, line2,line3], loc='upper right')
plt.text(0, 0, "Origine x=y=0", color ="r")
plt.show()
```



La bibliothèque graphique matplotlib

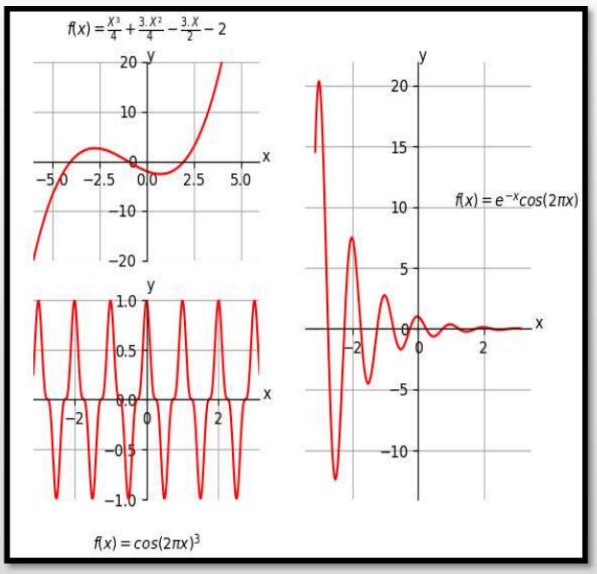
Une fenêtre graphique peut contenir plusieurs graphiques disposés dans une grille de lignes et colonnes. Chaque sous-graphique est , ainsi, introduit par la fonction subplot(NbLignes, NbColonnes, numeroGraphique).

```
t = np.linspace(-6.0,4.0,256); x = np.linspace(-np.pi,np.pi,256)
f1= lambda t: t**3/4.0+3.0*t**2/4.0-3*t/2.0-2.0
f2= lambda x:np.cos(2*np.pi*x)**3
f3= lambda x:np.exp(-x)*np.cos(2*np.pi*x)
def style(ax):
  ax.spines['left'].set_position('zero')
  ax.spines['right'].set_color('none')
  ax.spines['bottom'].set_position('zero')
  ax.spines['top'].set_color('none')
  ax.grid(True); maxX=ax.get_xlim()[1];
  maxY = ax.get_ylim()[1]
  ax.text(maxX + 0.1, 0.0, r'x')
  ax.text(0.0, maxY+0.1, r'y')
ax= plt.subplot(2, 2, 1)
ax.plot(t, f1(t), 'r')
ax.set_xlim(-6, 6); ax.set_ylim(-20, 20)
style(ax); ax.text(0.0,25.0, r'$f(x)=\frac{X^3}{4}+\frac{3.X^2}{4}-
       \frac{3.X}{2}-2$', horizontalalignment='center', fontsize=10)
```



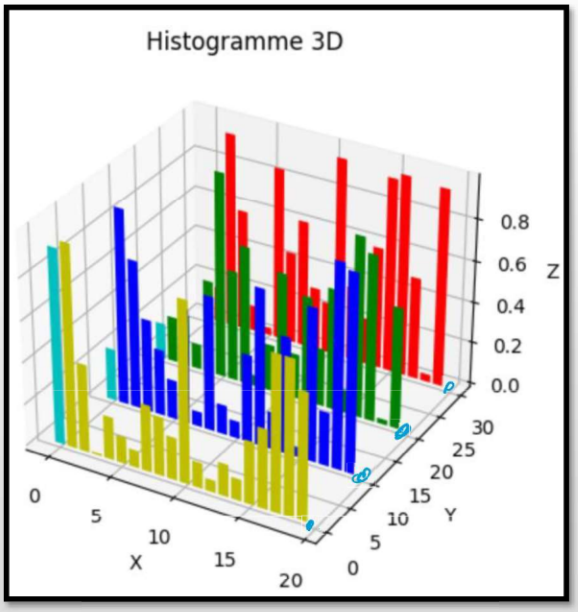
La bibliothèque graphique matplotlib

```
ax=plt.subplot(2, 2, 3); ax.plot(x, f2(x), 'r')
ax.set_xlim(-np.pi, np.pi); ax.set_ylim(-1.0, 1.0)
style(ax)
ax.text(0.0,-1.5,r'$f(x)=cos(2\pi x)^3 $',
  horizontalalignment='center',
  fontsize=10)
ax=plt.subplot(1, 2, 2)
ax.plot(x, f3(x), 'r')
style(ax)
ax.text(3.0,10,
       r'$f(x)=e^{-x} cos(2\pi x) $',
       horizontalalignment='center',
       fontsize=10)
plt.show()
```



La bibliothèque graphique matplotlib

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d  # Fonction pour la 3D
import numpy as np
# Tracé du résultat en 3D
fig = plt.figure()
ax = fig.gca(projection='3d')
# Construction des histogrammes et
# affichage barre par barre
for c, z in zip(['r', 'g', 'b', 'y'], [30, 20, 10, 0]):
    x = np.arange(20)
    y = np.random.rand(20)
    cs = [c] * len(x)
    cs[0] = 'c' # Ici la première barre est en cyan.
    ax.bar(x, y, z, zdir='y', color=cs)  # Ajout barre
plt.title("Histogramme 3D")
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```



La bibliothèque Pandas

- Pandas est une librairie Python spécialisée dans l'analyse des données (relationnelles ou étiquetées) :
  - Manipuler des données aisément et efficacement (sous forme de tableau)
  - Lire et écrire des données depuis fichiers CSV,tableurs Excel,...
  - Fusion et jointure de large volume de données
  - Représentation graphique
- La librairie est très largement documentée (https://pandas.pydata.org/docs/).

Le type Dataframe de Pandas

- Les dataframes correspondent à des tableaux à deux dimensions avec des étiquettes pour nommer les lignes et les colonnes.
- Un dataframe est créé avec la fonction DataFrame() à laquelle on fournit les arguments suivant:
  - L'argument columns et index indiquent les listes des noms des colonnes et des lignes.
  - L'argument data fournit le contenu du dataframe, sous la forme d'une liste de valeurs correspondantes à des lignes.

Exemple:

```
df=pd.DataFrame(columns=["c1","c2","c3","c4"],
                index=["l1","l2","l3"],
      data=[np.arange(10,14),np.arange(20,24),np.arange(30,34)])
```

	c1	c2	c3	c4
l1	10	11	12	13
l2	20	21	22	23
l3	30	31	32	33

Le type Dataframe de Pandas

- Un dataframe peut aussi être créé avec la fonction pd.DataFrame.from_dict() à laquelle on passe un dictionnaire qui contient les données en colonnes (à chaque colonne est associée le nom de la colonne comme clé) . Dans ce cas, les étiquettes des lignes peuvent être définit avec l'attribut index

Exemple:

```
data2={"c2":[3,4],c4":[5,6]}
df2=pd.DataFrame.from_dict(data2)
df2.index = ["l1", "l3"]
```

	c2	c4
l1	3	5
l3	4	6

- Un dataframe possède plusieurs attributs, à titre d'exemple:
  - shape : Dimensions d'un dataframe
  - index, columns : Noms des lignes et des colonnes, elles permettent aussi de renommer les lignes et les colonnes d'un dataframe
  - values: Représente les valeurs d'un dataframe
  - dtypes: type de données de chaque colonne

Le type Dataframe de Pandas

Les mécanismes de sélection fournis avec pandas sont très puissants.

Exemple:

	c1	c2	c3	c4
l1	10	11	12	13
l2	20	21	22	23
l3	30	31	32	33

- Sélection de colonnes : df [["c2", "c3"]], df["c2"][0:3]
- Sélection de lignes: df.loc["l2"] - df.iloc [1] - df.iloc [[1,0]] - df.iloc[0:2]
- Sélection sur les lignes et les colonnes: df.loc [["l3", "l1"], ["c3", "c4"]]

	c3	c4
l3	32	33
l1	12	13

- Sélection par condition: df[df > 0]

  df [(df ["c3"] >15) & (df ["c2"] >25)]

  df[(df ["c3"] >15) | (df ["c2"] >25)]

	c1	c2	c3	c4
l3	30	31	32	33

	c1	c2	c3	c4
l2	20	21	22	23
l3	30	31	32	33

Opérations sur les objets Dataframe

- La fonction pd.to_datetime (df [“nomColonne"]) permet de convertir les donnée d’une colonne en format date.
- La fonction concat(…) permet de concaténer des dataframes fournis sous forme de liste.

```
pd.concat ([ df , df2 ])) # concat par lignes
pd.concat ([ df , df2 ], axis=1)
df3=pd.concat ([ df , df2], join ="inner")
```

	c2	c4
l1	11	13
l2	21	23
l3	31	33
l1	3	5
l3	4	6

- La fonction crosstab(…) réalise des calculs statistiques croisés (table de fréquence : semblable aux tableaux croisés d'Excel)

```
pd.crosstab(df3["c2"],df3["c4"]))
```

- La function read_csv(…) permet charger le contenu d’un fichier csv dans un dataframe : df = pd. read_csv ("nameFile.csv")

Quelques méthodes d'un Dataframe

Méthode	Description
head(n), tail(n)	renvoie les n lignes du haut et du bas du dataframe (par défaut, n vaut 5)
transpose	Transposer les données
Sort_index(axis=1, ascending=False)	Tri par un axe (axis=0 par colonnes , axis=1 par lignes),. « ascending=False » correspond à un tri en ordre décroisant. La méthode sort_values() permet de retourner les valeurs triées d'une colonne.
describe()	Faire une statistique sommaire et rapide des données (count, mean, std : écart-type, min, max,…) Exemple : df.describe() **c1 c2 c3 c4** **count** 3.0 3.0 3.0 3.0 **mean** 20.0 21.0 22.0 23.0 **std** 10.0 10.0 10.0 10.0 **…** … … … …
groupby(["col"]). mean()	Regroupe les données par colonne et effectue des calcul selon la fonction statistique utilisée(mean, std, min, max, median…)

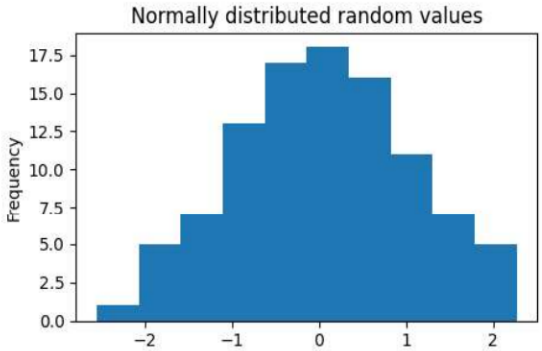
10 minutes pour les pandas — documentation pandas 1.3.4 (pydata.org)

Le type Series de Pandas

- Le type de données séries de pandas correspond à un vecteur d'une dimension. Chaque élément d'une série de données peut posséder une étiquette. L'accès aux valeurs d'une série se fait soit en précisant leur position ou leur étiquette.
- Les types des valeurs d'une série peuvent être différentes et qu'il peut y avoir des répétitions dans les étiquettes.

Exemple

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

np.random.seed(0) # réinitialisation
values = np.random.randn(100)
s = pd.Series(values) # générer une série pandas
s.plot(kind='hist',title='Normally distributed random values')
# ou directement s.hist()
plt.show()
```



Module : Programmation Python

Chapitre 4: Base de données et programmation graphique en Python

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

Les bases de données en python

- Python peut gérer différents types de bases de données. Pour chacun de ces types, une API différente existe: ODBC, Pysqlite, PyMysql, …
  - Créer un objet connexion à la base de données en utilisant la méthode **connect** puis définir un curseur qui fonctionnera avec la connexion crée.
  - Utiliser la méthode **execute** du curseur pour interagir avec la base de données et, si nécessaire , valider les modifications à l'aide de la méthode de validation de l'objet de connexion.
- Accéder à une base de données MySQL

```
import pymysql
class Dbconnect(object):
    def __init__(self):
        self.dbconection = pymysql.connect(host='host_example',
                                           port=3306, user='user_example',
                                           passwd='pass_example', db='base_example'  )
        self.dbcursor = self.dbconection.cursor()
    def commit_db(self):
        self.dbconection.commit()
    def close_db(self):
        self.dbcursor.close() ; self.dbconection.close()
```

Manipuler une base de données MySQL

- Exécution d'une requête SQL

```
db = Dbconnect()
db.dbcursor.execute('SELECT * FROM …')
```

- Récupération du résulat d'une requette

```
#Utilisation de l'objet curseur comme générateur pour récupérer tous les résultats
results = db.dbcursor.fetchall()
for individual_row in results:
    first_field = individual_row[0]
    …
#Utilisation directe du générateur
for individual_row in db.dbcursor:
    first_field = individual_row[0]
```

- Valider des modifications apportées à la base de données:

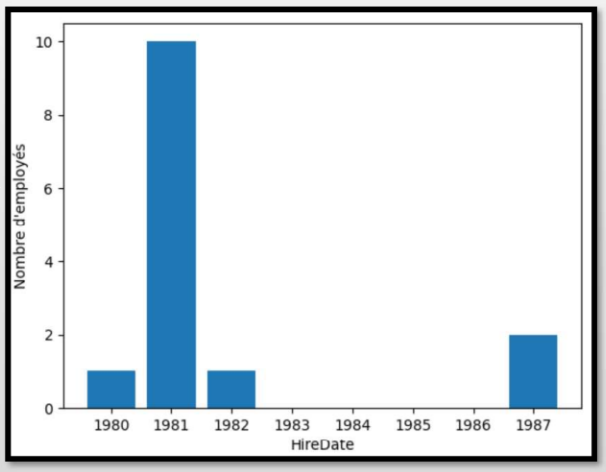
```
db.commit_db()
```

- Fermeture de la connection

```
db.close_db()
```


Manipuler une base de données MySQL

```
import pymysq # PyLab permet d'utiliser aisément  NumPy et matplotlib
from pylab import *
…
try :
  db = Dbconnect()
  db.dbcursor.execute('SELECT * FROM emp')
  for emp in db.dbcursor.fetchall():  print(emp)
  sql="select year(hiredate), count(*) from emp group by year(hiredate)"
  db.dbcursor.execute(sql) ;
  res=array(db.dbcursor.fetchall())
  annee=res[:, 0]
  nbEmp=res.transpose()[1]
  fig=figure()
  bar(annee,nbEmp)
  ylabel("Nombre d'employés")
  xlabel("HireDate")
  show()
  db.close_db()
except Exception as err:
  print ("Erreur detected:" + str(err) )
```



Manipuler une base de données SQLLite

- SQLite est une base de données légère basée sur disque (ne nécessite pas de serveur de base de données séparé)
- SQLite supporte nativement les types suivants: NULL, INTEGER, REAL, TEXT, BLOB.

Type python	Type SQLLite
None	NULL
int	INTEGER/INT
float	REAL/FLOAT
str	TEXT/VARCHAR(n)
bytes	BLOB

- Avec sqlite3, les clefs étrangères ne sont pas activées de base. Pour les actives il faut exécuter le code suivant :

```
curseur.execute("PRAGMA foreign_keys = ON")
```


Manipuler une base de données SQLLite

Exemple de manipulation d'une base de données

```
import sqlite3
conn = sqlite3.connect("users.db")
cur = conn.cursor()
cur.execute("CREATE TABLE user (name text, age integer)")
cur.execute("INSERT INTO user VALUES ('User A', 42)")
cur.execute("INSERT INTO user VALUES ('User B', 43)")
name = "ENSAB" ;  age = 37
cur.execute("insert into user values (?, ?)", (name, age)) # requête paramétrée
conn.commit() # conn.rollback() annule la transaction
cur.execute("SELECT * FROM user")
print(cur.fetchall())
conn.close()
```

Manipuler une base de données SQLLite

- La méthode « executescript(sql_script) » permet d'exécuter plusieurs instructions SQL à la fois (une instruction COMMIT est exécutée avant le script SQL)

```
conn = sqlite3.connect(":memory:") # créer une BD temporaire en RAM
cur = conn.cursor()
cur.executescript(""" create table book( title, author, published );
    insert into book(title, author, published) values ( 'Python', 'ENSAB',2019); """)
```

- La méthode « executemany(sql, seq_of_parameters) » permet d'exécuter une commande SQL en utilisant une séquence de paramètres

```
L = [('title 1', 'author 1', 2019), ('title 2', 'author 2', 2019),('title 3', 'author 3', 2019)]
conn.executemany("insert into User values (?, ?, ?)", L)
donnees = ( {"tit":"t4","aut": "a4", "pub": 2019}, {"tit":"t4","aut": "a4", "pub": 2019}, ...)
curseur.executemany("INSERT INTO book (title, author, published) VALUES (:tit,
        :aut , :pub)", donnees)
```

- La méthode fetchone() récupère la ligne suivante de l'ensemble de résultats de requête alors que « fetchmany(size=cursor.arraysize) » renvoie une liste du prochain ensemble de lignes de ce résultat (spécifié par taille),

```
cur.execute ("select * from book"); print(cur.fetchmany(2)); bk = cur.fetchone()
while(bk):
    print(bk) ; bk = cur.fetchone()
```


Manipuler une base de données

Attribut	Description
connection. in_transaction	Vaut True si des modifications ont été apportées sans être validées et False sinon.
connection. total_changes	Nombre de modifications (ajouts, mises à jour ou suppressions) apportées depuis la connexion à la base
cursor. rowcount	Nombre de lignes impactées par une exécution. S'il n'y a eu aucune exécution ou que le nombre de de lignes ne peut pas être déterminé, il vaut -1.

Les interfaces graphiques en Python

- Python fournit diverses options pour développer des interfaces graphiques (GUI):
  - Tkinter : interface Python de la bibliothèque GUI Tk livrée avec Python.
  - wxPython : implémentation en Python libre et open source Python de l'interface de programmation wxWidgets.
  - PyQt : interface Python pour une bibliothèque d'interface graphique Qt populaire multiplate-forme.
  - …
- La création d'une application graphique suit généralement les étapes suivantes :
  - Créer la fenêtre principale de l'application graphique.
  - Ajouter à l'interface les widgets (contrôles) graphiques nécessaires.
  - Définir les actions pour répondre à un évènement déclenché par l'utilisateur

Les widgets Tkinter

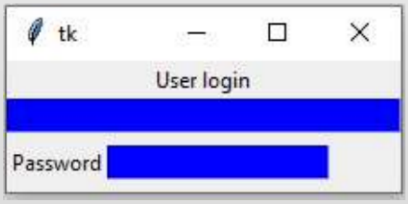
widgets	Description
Button	Bouton simple, qui permet de lancer une commande
Canva	Widget générique qui offre une surface de dessin (lignes, ovales, …)
Checkbutton	Cases à cocher (l'utilisateur peut sélectionner plusieurs options à la fois)
Radiobutton	Options sous forme de boutons radio (l'utilisateur ne peut sélectionner qu'une option à la fois)
Label	Légende ou description pour les autres widgets (peut contenir des images)
Entry	Champ de texte d'une seule ligne
Frame	widget conteneur pour organiser d'autres widgets.
Listbox	Liste d'options de choix déroulant
menubutton	Les menus dans votre application.
Menu	Les diverses commandes contenues dans Menubutton
Message	Un label évolué de texte multilignes
Scale	widget Echelle qui fournit un widget à curseur.
Scrollbar	Barre de défilement qui est utilisée pour ajouter une fonctionnalité de défilement à divers widgets, tels que les zones de liste.
Text	Champs de texte sur plusieurs lignes.
Toplevel	Conteneur de fenêtre séparé (gére une fenêtre modale autonome).
Spinbox	Variante du widget standard Tkinter Entry, qui peut être utilisé pour sélectionner un nombre fixe de valeurs.
PanedWindow	Conteneur pouvant contenir un nombre quelconque de volets, disposés horizontalement ou verticalement.

Les interfaces graphiques **Tkinter**

- La classe Tk du module Tkinter est un widget spécial qui, lorsqu'elle est instanciée, représente la fenêtre principale de l'application
- La méthode pack() assure le positionnement d'un widget dans un widget conteneur (rend le widget visible).
- Chaque widget possède un certain nombre de propriétés, appelées options, peuvent être lues et configurées comme des éléments de dictionnaire, ou spécifiées en paramètres du constructeur.
- La méthode mainloop() assure l'affichage de la fenêtre principale et se place en attente des événements.

Exemple

```
from  tkinter import *
maFenetre = Tk()
lblUser = Label (maFenetre , text = "User login" )
txtUser = Entry (maFenetre)
txtUser['bg']="blue"
lblUser.pack(); txtUser.pack(fill=X)
lblPass = Label (maFenetre )
lblPass['text'] = "Password"
txtPass = Entry (maFenetre ,bg="blue" )
lblPass.pack(side=LEFT); txtPass.pack(side=LEFT)
maFenetre.mainloop ()
```



Exemple d'options du le widget Entry

Option	Description
bg , fg	définit la couleur du texte et celui de l'arrière-plan
bd	définit la taille des bordures. La valeur par défaut est 2 pixels.
command	définit et associe une commande
cursor	définit le motif du curseur de la souris
font	définit la police qui sera utilisée pour le texte.
highlightcolor	couleur de surbrillance à utiliser pour le rectangle qui est dessiné autour du widget lorsqu'il a le focus.
justify	spécifie comment le texte est organisé s'il contient plusieurs lignes.
relief	spécifie le type de bordure. Sa valeur par défaut est FLAT
selectbackground	la couleur d'arrière-plan du texte sélectionné.
selectborderwidth	définit la largeur de la bordure à utiliser autour du texte sélectionné. La valeur par défaut est un pixel.
selectforeground,	définit la couleur de premier plan du texte sélectionné.
show	utilisé pour afficher ou masquer les caractères. Exemple pour un mot de passe on utilise show = "*".
state	utiliser state =DISABLED pour masquer le contrôle et le rendre inactif. la valeur par défaut est state = NORMAL,
width	définit la largeur du widget
xscrollcommand	ajoute une barre de défilement au widget.

Exemple de méthodes associés au widget Entry

Méthode	Description
delete (first, last = None)	supprime les caractères du widget, en commençant par celui qui est à l'index first, jusqu'au dernier last
get()	renvoie le texte actuel de l'entrée sous forme de variable chaîne.
icursor(index)	place le curseur d'insertion juste avant le caractère à l'index donné.
index(index)	décale le contenu de l'entrée de sorte que le caractère à l'index donné soit le caractère visible le plus à gauche (n'a aucun effet si le texte est entièrement contenu dans l'entrée).
insert(index, s)	insère la chaîne s avant le caractère à l'index donné.
select_clear ()	efface la sélection. S'il n'y a pas actuellement de sélection, n'a aucun effet.
select_form(index)	définit la position de l'index d'ancrage sur le caractère spécifié par l'index.
select_present()	s'il y a une sélection, renvoie vrai, sinon renvoie faux.
select_range(début, fin)	sélectionne les caractères qui existent dans la plage spécifiée.
select_to(index)	sélectionne tous les caractères du début à l'index spécifié.
xview(index)	utilisé pour lier le widget de saisie à une barre de défilement horizontale.
xview_scroll()	utilisé pour faire défiler l'entrée horizontalement.

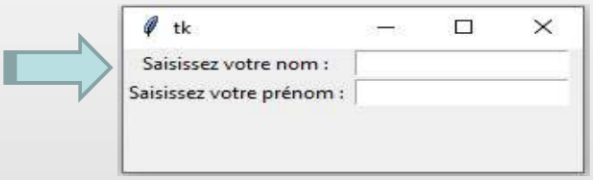
Gestion des dispositions géométriques des widgets

- Le widget Frame (cadre) est utilisé pour le processus de regroupement et d'organisation des autres widgets de manière conviviale.
- Tkinter possède les méthodes de gestionnaire de géométrie suivantes :
  - La méthode **pack**() qui place les widgets l'un au-dessous des autres selon l'ordre d'application de la méthode pack() avec un emplacement centré par défaut.
  - La méthode **grid**() organise les widgets dans une structure de type table dans le widget parent.
  - La méthode **place**() organise les widgets en les plaçant à des positions spécifiques dans le widget parent suivant leurs coordonnées et leurs dimensions

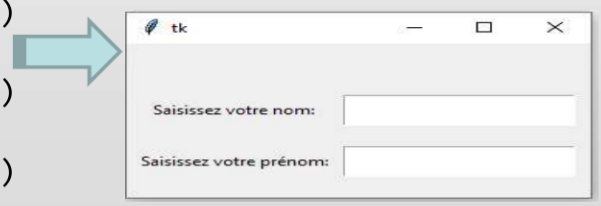
Gestion des dispositions géométriques des widgets

```
f = Tk(); f.geometry( "300x150")
Nom = Label ( f , text = "Saisissez votre nom : " )
Prenom = Label ( f , text="Saisissez votre prénom : " )
champNom = Entry ( f );
champPrenom = Entry ( f )

Nom.pack(); champNom.pack();
Prenom.pack(); champPrenom.pack()
```


```
Nom. grid (row=0, column=0)
Prenom. grid (row=1, column=0)
champNom. grid (row=0, column=1)
champPrenom. grid (row=1, column=1)
```




```
Nom.place(x=10,y=50,width=120,height=30)
champNom.place(x=140,y=50,width=150,
                                height=30)
Prenom.place(x=10,y=100,width=120,
                                height=30)
champPrenom.place(x=140,y=100,
            width=150,height=30)
```



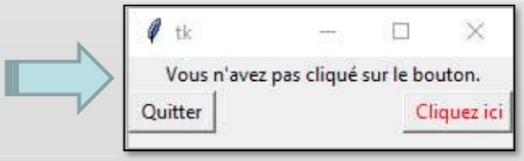
Gestion des événements

- Les applications graphiques sont pilotées par les événements. Un réceptionnaire d'événements scrute, ainsi, sans cesse tous les périphériques (clavier, souris, horloge, etc.) et réagit immédiatement lorsqu'un événement y est détecté.
- Dans ce cas, le réceptionnaire envoie un message spécifique au programme python pour réagir en appelant une fonction afin d'effectuer le travail que l'on attend en réponse à l'événement.

Actions manipulant des widgets Tkinter

Exemple

```
class Interface(Frame):
  def __init__(self, fenetre, **kwargs):
     Frame.__init__(self, fenetre, width=768, height=576, ** kwargs)
     self.pack(fill=BOTH);
     self.nb_clic = 0
     # Création des widgets
     self.msg=Label(self, text=" Vous n'avez pas cliqué sur le bouton.")
     self.msg.pack()
     self.btnQuit= Button(self, text=" Quitter ",command=self.quit)
     self.btnQuit.pack(side="left")
     self.btnCliq=Button(self,text=" Cliquez ici",command= self.cliquer )
     self.btnCliq.pack(side="right")
  def cliquer(self):
     self.nb_clic += 1
     self.msg["text"]= " Vous avez cliqué{} fois .".format(self.nb_clic)
fenetre = Tk()
interface = Interface(fenetre)
interface.mainloop()
interface.destroy()
```

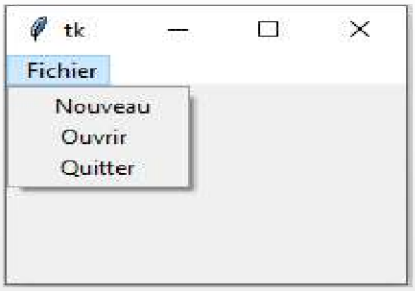


L'option **commande** définie l'action à exécuter lors d'un événement click sur un bouton

Menu Tkinter

Le rôle de ce widget est de permettre de créer toutes sortes de menus utilisables par une applications.

Exemple




```
master = Tk()
# étape 1: Création de la barre des menu
menuBar = Menu(master)
# étape 2: Création du menu principal
menuFichier = Menu(menuBar, tearoff = 0)
#Menu non achable (tearoff= 0); Menu détachable (tearoff=1)
menuBar.add_cascade( label=" Fichier", menu=menuFichier)
# étape 3: Création des sous menus
menuFichier.add_command( label="Nouveau" )
menuFichier.add_command( label=" Ouvrir" )
menuFichier.add_command( label=" Quitter" , command=quit )
# étape 4: Configuration de la barre des menus
master.config (menu=menuBar)
```

Actions manipulant des widgets Tkinter

- Chaque widget peut associer une fonction à **un événement qu'elle reçoit**, par le biais de la méthode **bind :**

```
widget.bind(event, fonctionCallback )
```

- Le premier argument de la méthode bind est un événement. On le définit par une chaîne de caractères de la forme : "<modificateur-type-détail>"

modificateur	Description
Alt, Control, Shift	Touche Alt , Ctrl ou Maj enfoncée
Double, Triple	événement s'est produit 2 ou 3 fois dans un cours laps de temps.
Lock	Vrai si l'utilisateur a verrouiller le mode Majuscule.

Exemple de type	Description
Button	clic de souris.
ButtonRelease	Bouton de la souris relâché
KeyPress	Touche clavier a été enfoncé
KeyRelease	Touche clavier a été relâché
Enter , Leave	Déplacement de la souris dans (enter) ou hors (leave) du Widget lié à l'événement

Exemple

"<KeyPress-s>" : Touche S enfoncée ; "<Double-Button-1>" : Double clic gauche de souris
<Control-Shift-KeyPress-H>: Les touches Control, Maj et H sont simultanément enfoncées

- Le callback est la fonction qui sera appelée lorsque l'événement se produit. Cette fonction doit accepter en argument un objet **event**.

```
def fonctionCallback(event): ...
```


Actions manipulant des widgets Tkinter

Lorsque l'événement est intercepté, L'argument event de la fonction callback permet de fournir des informations sur l'événement qui a été déclenché à travers différents attributs.

Attribut	Description
type	le type d'événement
char	le code du caractère sous forme de chaîne (événement clavier)
keycode	le code de touche (événement clavier)
keysym	le symbole de touche (événement clavier)
Height, width	la nouvelle hauteur , largeur (événement configuration)
x, y	la position horizontale , verticale de la souris
x_root, y_root	la position horizontale, verticale de la souris, relative au coin supérieur gauche
num	le numéro de bouton (événement souris)
widget	un lien vers l'instance de widget liée à l'événement

```
def evenementClavier(event):
  print ("Type event:", event.type )
  print ("Touche:",event.char,"code=",event.keycode, "sym=",event.keysym)
def evenementSouris(event):
  print ("Type event:", event.type ); print("numéro bouton:", event.num)
  print("position relative: x=", event.x, " y=", event.y)
  print("position relative: x=", event.x_root, " y=", event.y_root)
racine = Tk(); racine.bind('<KeyPress>',evenementClavier)
racine.bind('<Button-1>',evenementSouris) ; racine.mainloop ()
```

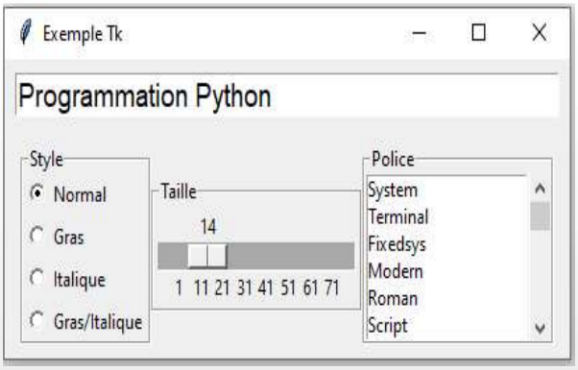
Actions manipulant des widgets Tkinter

Evènement clavier	Description
<Enter>, <Return>	touche Entrée
<Escape> , <Cancel>	touche Échappement , combinaison des touches Ctrl+C
<Alt_L>, <Cntrl_L>, <Shift_L>	touches Alt, Ctrl, Shift (L pour Left, R pour Right)
<Up>, <Left> , <Down>, <Right>, <Home>,<End>,<Next>,<Prior>,<Tab>	flèche haut, gauche, bas, droite et touches Home, Fin, page down, page up, Tabulation
<Delete>, <Insert>, <BackSpace>	touche Suppression, Insertion, retour arrière (backspace)
<Caps_Lck>, <Num_Lck>	touche de verrouillage majuscules , verrouillage numérique
<Pause>, <Print>, <FN>	touche Pause, Impression, de fonctions F1, F2, F3...
<Key>	touche quelconque

Evènement souris	Description
<[Button|ButtonPress]-n>	un des boutons de la souris est appuyé. n vaut 1 (bouton gauche), 2 (bouton du centre) ou 3 (bouton de droite)
<Bn-Motion>	la souris est déplacée au-dessus du widget, avec un bouton appuyé (n vaut 1, 2 ou 3)
<ButtonRelease-n>	le bouton n est lâché.
<Configure>	la taille du widget est modifiée.
<Double-Button-n>,<Triple-Button-n>	équivalent à Button, mais pour un double ou triple-clic.
<Enter> , <Leave>	la souris entre sur le widget, sort du widget

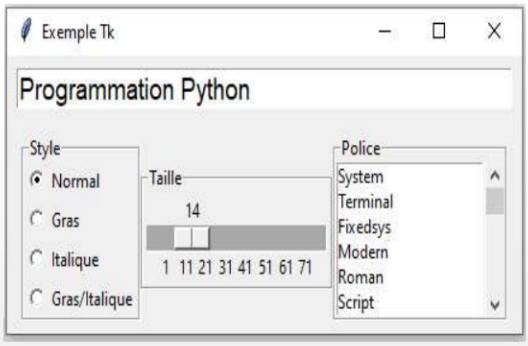
Exemple

```
from tkinter import *
import tkinter.font as tkFt
class Interface(Frame):
  def __init__(self, fenetre, **kwargs):
    Frame.__init__(self, fenetre)
    self.pack(fill=BOTH)

    self.txt=Entry(master=self)
    self.txt.insert(END,"Prog Python")
    self.txt.pack(fill=X,padx=8, pady=8)

    frm1 = Frame(self); frm1.pack()
    radioGroup = LabelFrame(frm1, text="Style")
    radioGroup.grid (row=0, column=0,pady=8)
    stylePoliceFr = ["Normal", "Gras", "Italique", "Gras/Italique"]
    stylePoliceTk = ["normal", "bold", "italic", "bold italic"]
    self.style = StringVar() ; self.style.set(stylePoliceTk[0])
    for n in range(4):
       bout=Radiobutton(master=radioGroup,text=stylePoliceFr[n],
            variable=self.style,value=stylePoliceTk[n],command=self.update)
       bout.pack(anchor=W) # alignement à gauche

    frameTaille = LabelFrame(frm1, text="Taille")
    frameTaille.grid(row=0, column=1, pady=8)
```



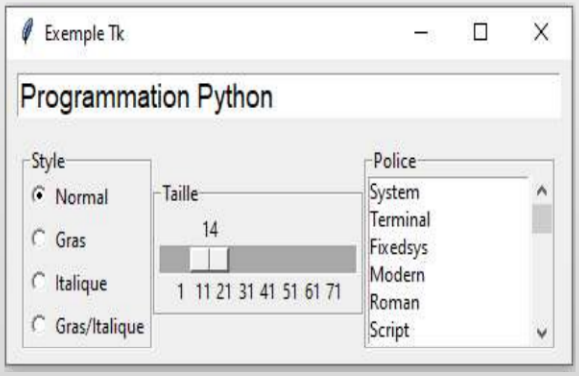
Exemple

```
  def __init__(self, fenetre, **kwargs):
    ...
    self.taille=IntVar()
    self.curseur= Scale(master=frameTaille,
       orient='horizontal',
       troughcolor='dark grey',from_=1,
       to=72,resolution=1,tickinterval=10,
       length=150,variable= self.taille,
       command = self.updateTaille )
    self.curseur.pack(); self.curseur.set(14)
    framePolice = LabelFrame(frm1, text="Police")
    framePolice.grid(row=0, column=2, pady=8);
    sbar = Scrollbar(framePolice)
    self.police=StringVar()
    list = Listbox(framePolice, relief=SUNKEN, height =6)
    sbar.config(command=list.yview);list.config(yscrollcommand=sbar.set)
    # expand : remplir tout espace non utilisé dans le widget parent.
    sbar.pack(side=RIGHT,fill=Y); list.pack(side=LEFT,expand=YES, fill=BOTH)
    for label in tkFt.families():
      list.insert(END, label)
    list.bind('<Double-Button-1>', self.handleList)
    self.listbox = list
```



Exemple

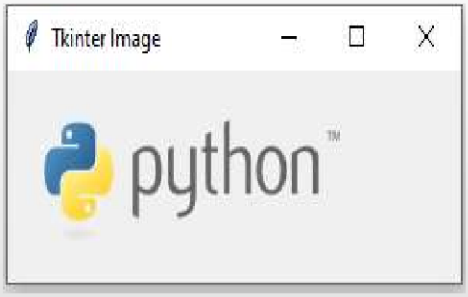
```
class Interface(Frame):
  def __init__(self, fenetre, **kwargs):
    ...
  def handleList(self, event):
    index = self.listbox.curselection()
    self.police = self.listbox.get(index)
    self.update()
  def updateTaille(self,event):
    self.update()
  def update(self):
    self.txt.configure(font=(self.police,self.taille.get(), self.style.get() )

  fenetre = Tk()
  fenetre.title("Exemple Tk")
  interface = Interface(fenetre)
  interface.mainloop()
  interface.destroy()
```



Manipulation d'images sur une fenêtre Tkinter

Pour traiter des images, Python dispose d'une librairie nommée Pillow (projet PIL: Python Imaging Library). Elle est conçue pour offrir un accès rapide aux données contenues dans une image avec la manipulation des différents formats de fichiers images tels que PNG, JPEG, GIF, TIFF et BMP.

Exemple




```
from PIL import Image, ImageTk
root = Tk()
root.title ( "Tkinter Image" )
root.geometry( "300x100" )
# Création de l'objet image
load = Image.open( "images/Python.png" )
# Redimensionner l'image
load.thumbnail((200,100) )
# Création d'une photo à partir de l'objet image
photo = ImageTk.PhotoImage( load )
# Placer l'image dans un label
label_image=Label(root,image=photo); label_image.place(x=20,y=20)
root.mainloop ()
```


Extensions pour Tkinter

Des modules de la bibliothèque standard pour compléter Tkinter

Module	Description
ScrolledText	widget texte doté d'ascenseurs
tkColorChooser	implémente un dialogue de sélection de couleur
tkCommonDialog	classe de base utilisée par tous les dialogues
tkFileDialog	implémente des dialogues de sélection de fichier
tkFont	utilitaires pour travailler avec les polices de caractères
tkMessageBox	dialogues standards d'affichage de messages
tkSimpleDialog	utilitaires et dialogues de base
Tix	widgets supplémentaires pour Tk

Deux extensions complètent cette liste **ttk** et **tix**. On trouve notamment :

Combobox, Notebook, Progressbar, Treeview

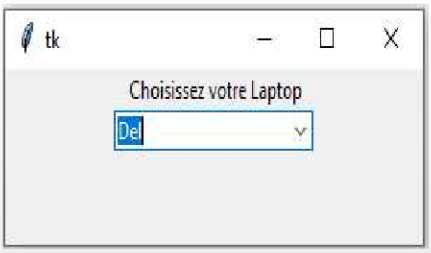
tix propose des widgets un peu plus complexes :

DirTree, FileSelectBox

Le module tkinter.ttk

Le module tkinter.ttk est une extension de la bibliothèque Tkinter qui fournit un accès au jeu de style pour les widgets Tk.

Exemple d'usage de ttk.Combobox

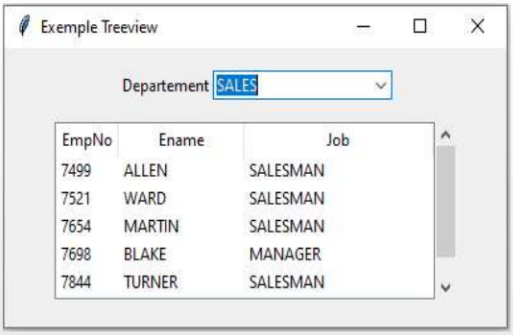



```
from tkinter import ttk
def comboAction(event) :
  select = combo.get ()
  print ( "Vous avez choisie : " , select )
root = tk.Tk(); root.geometry('300x200')
labelTop = tk.Label ( root , text = " Choisissez votre Laptop" )
labelTop.pack()
style = ttk.Style ()
style.configure ( "BW.TCombobox", padding=8, relief="flat",
            foreground="blue")
liste_valeurs=["Acer" , "HP" , "Del" , "Asus" ]
combo=ttk.Combobox(root,values=liste_valeurs,style="BW.TCombobox")
combo.pack();combo.current (0)#élément qui s'affiche par défaut
combo. bind ( "<<ComboboxSelected>>" , comboAction) # bind action
root.mainloop ()
```


L'objet Treeview

L'objet Treeview du module ttk, permet d'organiser et de visualiser sous forme d'une table des données sur une fenêtre Tkinter.

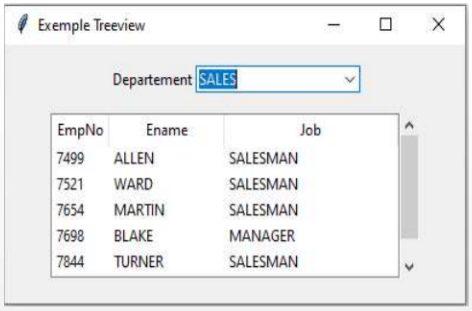
Exemple




```
class Interface(Frame):
  def __init__(self, fenetre, **kwargs):
    self.db = Dbconnect()
    Frame.__init__(self, fenetre)
    self.pack(fill=BOTH)
    frm1 = Frame(self, pady=8)
    frm1.pack()
    label = Label(frm1, text="Departement")
    label.grid(row=0, column=0, pady=8)
    self.db.dbcursor.execute('SELECT dname FROM dept')
    results = self.db.dbcursor.fetchall()
    self.dname=StringVar()
    self.combo = ttk.Combobox(frm1, values=results, textvariable =
                              self.dname )
    self.combo.grid(row=0, column=1, pady=8);self.combo.current(0)
```

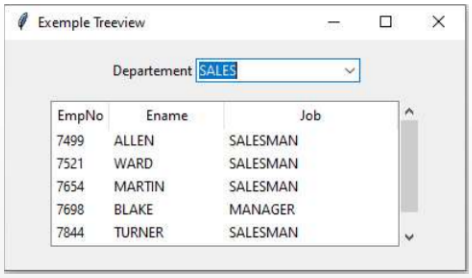
L'objet Treeview

```
  def __init__(self, fenetre, **kwargs):
    ...
    # === Création de l'objet Treeview ===
    frm2 = Frame(self) ; frm2.pack()
    self.tree = ttk.Treeview(frm2,columns=
          (1,2,3),height=5,show="headings")
    self.tree.place(x=50, y=50, width=300)
    self.tree.column(1, width=50); self.tree.column(2, width=100)
    self.tree.column(3, width=150) ;
    self.tree.heading(1,text="EmpNo");
    self.tree.heading(2, text="Ename")
    self.tree.heading(3, text="Job");
    sbar = ttk.Scrollbar(frm2,orient='vertical',
                                 command=self.tree.yview)
    sbar.pack(side=RIGHT, fill=Y)
    self.tree.pack(side=LEFT, expand=YES, fill=BOTH)
    self.tree.configure(yscroll=sbar.set)
    self.combo.bind("<<ComboboxSelected>>" , self.comboAction )
```



L'objet Treeview

```
class Interface(Frame):
  def __init__(self, fenetre, **kwargs):
    ...
  def comboAction (self, event):
    req="SELECT empno, ename, job FROM
        emp inner join dept on emp.deptno=
        dept.deptno where dname = '"+ self.dname.get()+ "'"
    self.db.dbcursor.execute(req )
    results = self.db.dbcursor.fetchall()
    for it in self.tree.get_children():
        self.tree.delete(it)
    for row in results:
        self.tree.insert('','end', values=(row[0], row[1], row[2]))

fenetre = Tk()
fenetre.title("Exemple Treeview"); fenetre.geometry( "400x200" )
interface = Interface(fenetre); interface.mainloop()
```



Module : Programmation Python

Framework Django

Pr LAHCEN MOUMOUN
ensablearn@gmail.com

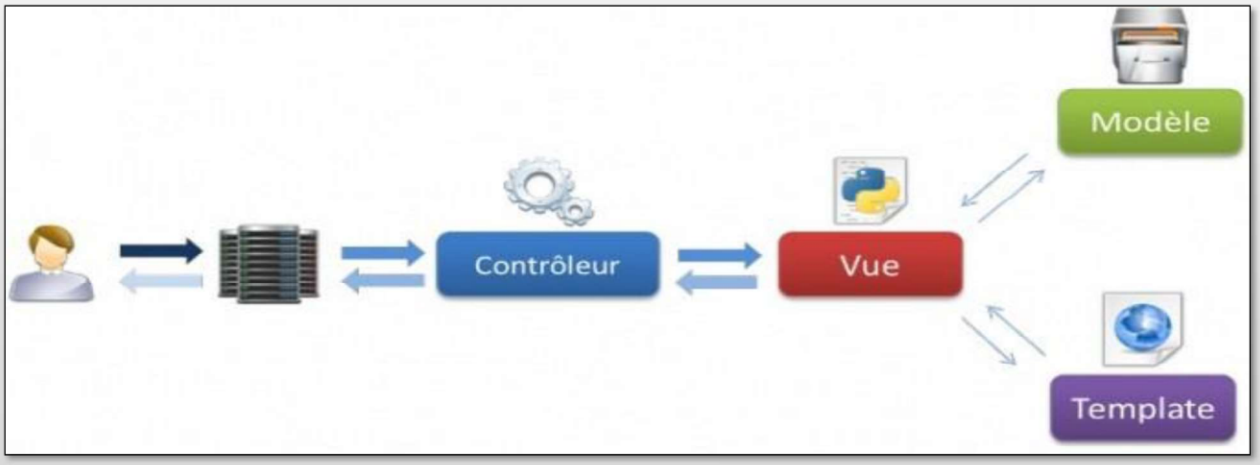
Département Génie Informatique & Mathématiques- FI SIBD

Le Framework Django

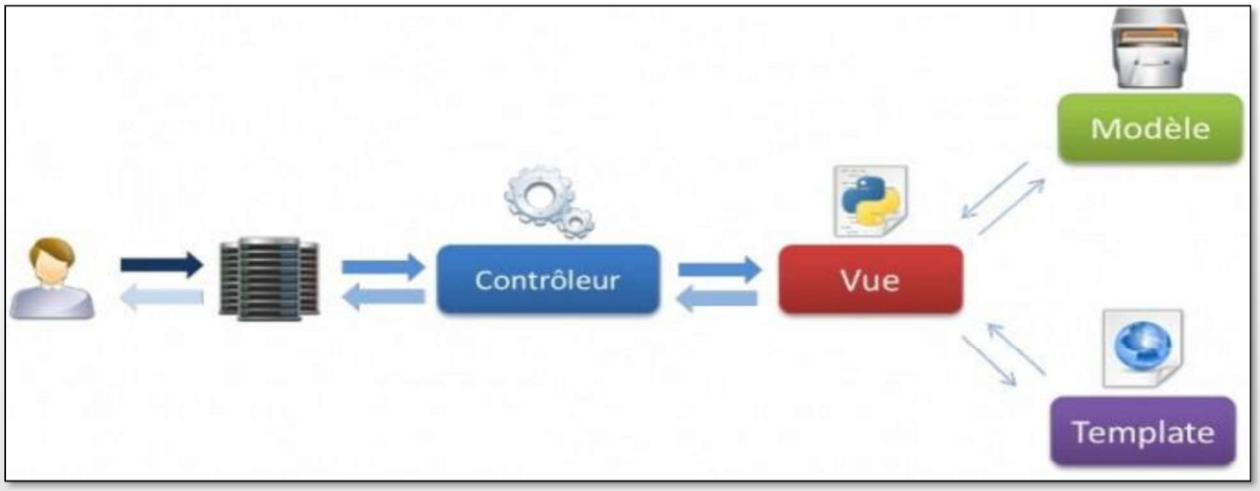
- Django est un framework web écrit en Python
  - Né en 2003 par l'agence de presse Lawrence Journal-World
  - Rendu Open Source sous Licence BSD (Berkeley Software Distribution) depuis 2005
  - Beaucoup d'évolutions depuis, la version actuelle est 4.0
- Django se veut complet tout en facilitant la création d'applications web riches.
  - Interface d'Admin complète, souple et extensible
  - Serveur Web intégré
  - ORM = Object Relational Mapper
  - Documentation très complète (docs.djangoproject.com)
  - Communauté active et expérimentée avec mise en disposition de snippets (portions de code réutilisables) sur djangosnippets.org

Le modèle MVT du Django

- Django utilise une architecture MVT (Modèle-Vue-Template), directement inspirée du modèle MVC ;
- Un template est un fichier HTML qui sera analysé et exécuté par le framework avant d'être récupéré par la vue et envoyé au visiteur.
- Django gère de façon autonome la réception des requêtes et l'envoi des réponses au client (partie contrôleur) ;

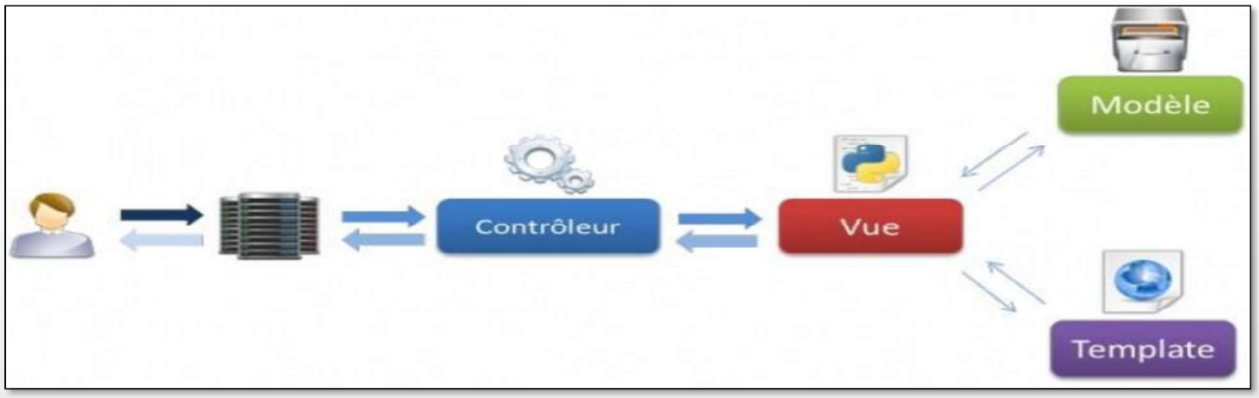


Le modèle MVT du Django



- Quand un utilisateur appelle une page web, la requête est directement prise en charge par le contrôleur de Django qui va chercher la vue qui correspond URL de la requête (règles de routage URL définies).
- Une vue est exécutée pour récupérer des données à partir des modèles et générer un rendu HTML à partir d'une template.
- Le serveur renvoie page générée au navigateur de l'internaute.

Le modèle MVT du Django



Les quatre parties à gérer par un développeur Django sont:

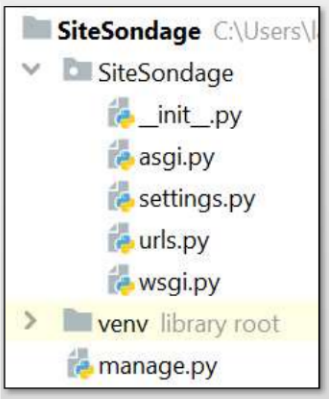
- Le routage des requêtes, en fonction de leur URL ;
- Les modèles : représentation des données dans l'application(classe dont les attributs correspondent aux champs dans la base de données), avec leur gestion (ajout, édition, suppression…);
- les templates permettant d'afficher les données au format HTML;
- les vues (a chacune on assigne une URL précise) représentée chacune par une fonction qui récupère des données dans les modèles et appel un template pour générer le rendu HTML adéquat.

Projets et applications Django

- Un projet ou site web Django (installation via pip install django) est divisé en plusieurs applications ayant chacune un ensemble de vues, de modèles et de schémas d'URL
- À la création du projet, Django déploie un ensemble de fichiers, facilitant à la fois la structuration du projet et sa configuration. La commande de création d'un projet est :

django-admin startproject SiteSondage

- « manage.py » assure l'administration du projet
- « settings.py » définit la configuration du projet
- « urls.py » définit le routage des URLs (fonction à appeler pour chaque URL).
- « __init__.py » , « asgi.py » et « wsgi.py » sont des fichier de chemin et de déploiement et (à ne pas modifier)



Projets et applications Django

- Commande pour lancer un serveur

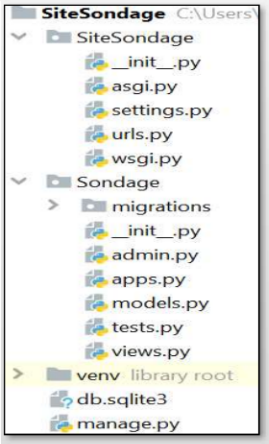
  python manage.py runserver

  Le serveur ne doit pas être utilisé en production (dans « settings.py », DEBUG = True). Le serveur peut être testé sur http://127.0.0.1:8000/

- Commande de création d'application

  python manage.py startapp Sondage

  - « models.py » et « views.py » contiennent resp. les modèles et les vues de l'application
  - « tests.py » permet la création de tests unitaires
  - « admin.py » contient la définition des éléments de l'espace admin
  - « apps.py » définit les applications du projet



Projets et applications Django

- Il faut ajouter le nom de l'application (ajout 'Sondage') à la liste de la variable INSTALLED_APPS du fichier «settings.py».

```
INSTALLED_APPS [
    'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes',
    'django.contrib.sessions','django.contrib.messages', 'django.contrib.staticfiles',
    'Sondage', ]
```

- Applications incluses par défaut :
  - django.contrib.admin – Le site d'administration.
  - django.contrib.auth – Un système d'authentification.
  - django.contrib.contenttypes – Une structure pour les types de contenu (content types).
  - django.contrib.sessions – Un cadre pour les sessions.
  - django.contrib.messages – Un cadre pour l'envoi de messages.
  - django.contrib.staticfiles – Une structure pour la prise en charge des fichiers statiques.

Gestion des vues

- Chaque application possède son propre fichier views.py, regroupant un ensemble de fonctions dont chacune représente une vue (la fonction récupère les données dans les modèles et génère le rendu HTML en utilisant une template)

  Exemple : Vue qui renvoi directement du code HTML au client

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, world. You're at index.")
```

- Toutes les fonctions prendront comme premier argument un objet du type HttpRequest. Elles doivent forcément retourner une instance de HttpResponse (sans quoi Django générera une erreur).

Routage d'URL

- Il faut créer un fichier URLconf « **urls.py** » dans le dossier de chaque application. La paramètre **urlpatterns** de ce fichier permet de définir les associations entre URL et vues.

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'), ]
```

  Un motifURL peut être composée d'arguments qui permettent par la suite de retrouver des informations dans les modèles)

- il faut ensuite ajouter dans la variable urlpatterns du fichier « urls.py » du projet les fichiers « urls.py » des applications (référencer les configurations)

```
from django.urls import include
urlpatterns = [
    ...
    path('sondage/', include('Sondage.urls')),
]
```

Les bases de données et Django

- La configuration de la base de données se fait dans le dictionnaire DATABASES du fichier « settings.py ». La configuration par défaut utilise SQLite (inclus dans Python)

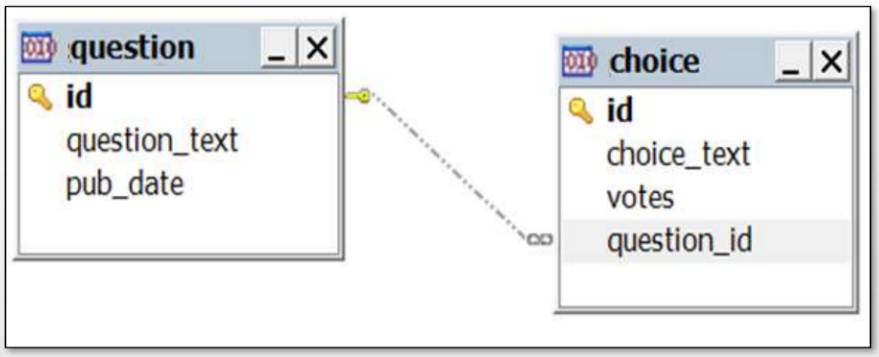
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Avec SQLite, NAME est le chemin absolu complet du fichier de la base. (BASE_DIR est le répertoire du projet).

- Si un autre SGBD est utilisé (prévoir l'installation du connecteur approprié), il faut apporter les réglages nécessaire à DATABASES
  - ENGINE – choisir 'django.db.backends.mysql' , 'django.db.backends.postgresql', 'django.db.backends.oracle' ou autre
  - NAME – définir le nom de la base de données.
  - HOST, USER , PASSWORD, 'PORT'

Base de donnée de l'application de sondage

Application de sondage :



Question et Choice (choix). Une Question possède le texte de la question et une date de mise en ligne. Un choix a deux champs : le texte représentant le choix et le décompte des votes. Chaque choix est associé à une Question.

Création des modèles

- Un modèle qui représente une table dans la base de données, s'écrit par une classe dont les attributs correspondent aux champs de la table.
- Lorsque un modèle est crée (dans le fichier models.py), le framework Django crée automatiquement la table correspondante et permettra d'y enregistrer les mise à jours des données du modèle.

```
from django.db import models
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField(auto_now_add=True, auto_now=False,
                verbose_name='date published')
    #auto_now_add: mise à jour à la création , auto_now : màj à la modification
    # verbose_name : nom humainement compréhensible
class Choice(models.Model):
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
```


Modèles et base de données

- Django propose un système ORM (object-relationnal mapping) pour bénéficier des avantages d'une base de données SQL.
- Les migrations sont le moyen utilisé par Django pour stocker les modifications des modèles. La commande **makemigrations** permet prendre en considération les changements de ces modèles(création de fichiers de migrations sur disque).

python manage.py **makemigrations** Sondage

- La commande **sqlmigrate** accepte des noms de migrations et affiche le code SQL correspondant.

python manage.py **sqlmigrate** Sondage 0001

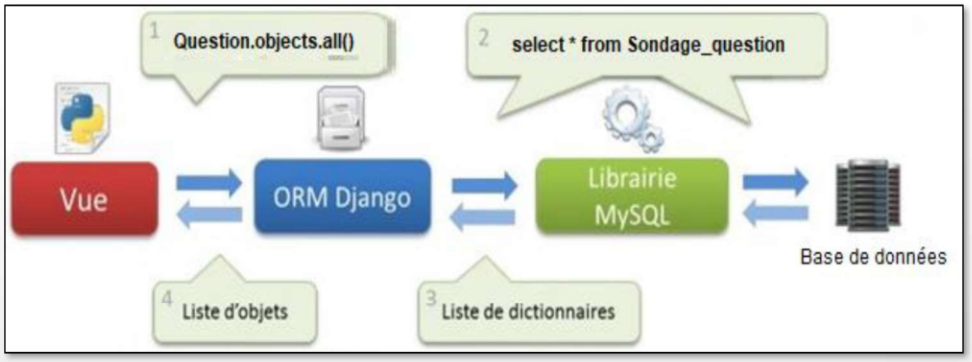
- La commande **migrate** exécute les migrations et gère automatiquement le schéma de base de données

python manage.py **migrate**

Modèles et base de données

```
BEGIN;
--
-- Create model Question
--
CREATE TABLE "Sondage_question" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "question_text" varchar(200) NOT NULL, "pub_date" datetime NOT NULL );
--
-- Create model Choice
--
CREATE TABLE "Sondage_choice" (
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "choice_text" varchar(200) NOT NULL,
    "votes" integer NOT NULL,
    "question_id" bigint NOT NULL REFERENCES "Sondage_question" ("id") );
CREATE INDEX "Sondage_choice_question_id_5780cbec«
          ON "Sondage_choice" ("question_id");
COMMIT;
```


Modèles et API Django



Lancer un shell Python pour manipuler l'API Django
python manage.py shell

```
from Sondage.models import Choice, Question
q = Question(question_text="What's new?") # Create a new Question
q.save()
q.id # Now it has an ID.
# Change values by changing the attributes, then calling save().
q.question_text = "What's up?"
q.save()
Question.objects.all()
```

Amélioration des modèles

```
from django.db import models
import datetime
from django.utils import timezone
class Question(models.Model):
        question_text = models.CharField(max_length=200)
        pub_date = models.DateTimeField(auto_now_add=True,
                        auto_now=False, verbose_name='date published’)
    def __str__(self):
        return (self.question_text)
    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
class Choice(models.Model):
        question = models.ForeignKey(Question, on_delete=models.CASCADE)
        choice_text = models.CharField(max_length=200)
        votes = models.IntegerField(default=0)
    def __str__(self):
        return self.choice_text
```


Modèles et API Django

```
Question.objects.filter(id=1) # keyword arguments
Question.objects.filter(question_text__startswith='What')
from django.utils import timezone
current_year = timezone.now().year
Question.objects.get(pub_date__year=current_year)
q.was_published_recently()
q = Question.objects.get(pk=1) #for primary-key exact lookups
# Create choices.
q.choice_set.create(choice_text='Not much', votes=0)
q.choice_set.create(choice_text='The sky', votes=0)
c = q.choice_set.create(choice_text='Just hacking again', votes=0)
# Display any choices from the related object.
q.choice_set.all()
# Choice objects have API access to their related Question.
c.Question
q.choice_set.count()
Choice.objects.filter(question__pub_date__year=current_year)
c = q.choice_set.filter(choice_text__startswith='Just hacking')
c.delete()
```

Site d'administration de Django

- Django automatise entièrement la création des interfaces d'administration pour les modèles (ajouter, modifier et supprimer du contenu est un travail).
- Création d'un utilisateur administrateur (login, email, motPass)

```
python manage.py createsuperuser
Username: admin - Email address: … - Password: …
```

- Lancer le site d'administration (à redémarrer le serveur) sur http://127.0.0.1:8000/admin/
- Rendre l'application de sondage modifiable via l'interface d'admin. Éditer le fichier Sondage/admin.py pour éditer les objets Question et Choice.

```
from django.contrib import admin
from .models import Question,Choice
admin.site.register(Question)
admin.site.register(Choice)
```


Exemple de vues de l'application sondage

Dans l'application de sondage, considérons les quatre vues suivantes :

- La page de sommaire des questions – affiche quelques-unes des dernières questions.
- La page de détail d'une question – affiche le texte d'une question, sans les résultats mais avec un formulaire pour voter.
- La page des résultats d'une question – affiche les résultats d'une question particulière.
- Action de vote – gère le vote pour un choix particulier dans une question précise.

Vues avec paramètres

Sondage.views

```
from django.http import HttpResponse
from .models import Question

def index(request): # affiche les 5 dernières questions.
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    output = ', '.join([q.question_text for q in latest_question_list])
    return HttpResponse(output)
def detail(request, question_id):
    return HttpResponse("You're looking at question %s." %question_id)
def results(request, question_id):
    response = "You're looking at the results of question %s."
    return HttpResponse(response % question_id)
def vote(request, question_id):
    return HttpResponse("You're voting on question %s." % question_id)
```


Vues avec paramètres

Spécification des URL des nouvelles vues dans Sondage.urls

```
urlpatterns = [
    # ex: http://127.0.0.1:8000/sondage/
    path('', views.index, name='index'),
    # ex: http://127.0.0.1:8000/sondage/5/
    path('<int:question_id>/', views.detail, name='detail'),
    # ex: http://127.0.0.1:8000/sondage/5/results/
    path('<int:question_id>/results/', views.results, name='results'),
    # ex: http://127.0.0.1:8000/sondage/5/vote/
     path('<int:question_id>/vote/', views.vote, name='vote'), ]
```

Les convertisseurs du chemin (motif URL) couramment utilisé :

- str - correspond à n'importe quelle chaîne non vide (à l'exception du séparateur de chemin '/').
- int - correspond à zéro ou un autre nombre entier positif.
- slug - correspond à toute chaîne composée de lettres ou chiffres ASCII, du trait d'union ou du caractère soulignement. Par exemple, construire-votre-1er-site-django.

Les vues et les templates

- Les templates (ou gabarits) sont écrits avec un langage propre à Django avec des tags sous forme d'expressions et des structures de contrôle basiques (if/else, boucle for, etc.). Le moteur de templates transforme les tags qu'il rencontre par le rendu HTML correspondant.
- Le paramètre TEMPLATES du fichier « settings.py » d'un projet Django indique comment le framework va charger et produire les templates (par défaut l'option APP_DIRS=True pour une recherche d'un sous-répertoire «templates» dans chaque application figurant dans INSTALLED_APPS)

Créer un répertoire templates dans le répertoire Sondage et un autre sous répertoire nommé Sondage dans ce répertoire templates

Les vues et les templates

Fonction index du fichier « Sondage/views.py»

```
from django.shortcuts import render
from django.template import loader
def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    context = {'latest_question_list': latest_question_list,}
    # 1ière méthode
    template = loader.get_template('Sondage/index.html')
    return HttpResponse(template.render(context, request))
    # 2ème méthode raccourci
    return render(request, 'Sondage/index.html', context)
```

- La fonction « **loader.get_template** » charge une template à partir du chemin donnés dans « settings.py »
- La fonction « **render** » génère un objet HttpResponse après avoir traité une template en utilisant un contexte. Ce contexte est un dictionnaire qui fait correspondre des objets Python à des noms de variables accessibles dans la template.

Les vues et les templates

Fichier « Sondage/templates/Sondage/index.html »

```
<!DOCTYPE html> <html lang="en">
<head> <meta charset="UTF-8">  <title>Title …</title> </head>  <body>
   {% if latest_question_list %}
     <ul>
     {% for question in latest_question_list %}
       <!-- 1ière méthode : approche codée en dur et fortement couplée
       <li><a href="/Sondage/{{ question.id }}/">{{question.question_text}}</a></li>-->
       <!-- 2ième méthode -->
       <li><a href="{% url 'detail' question.id %}">{{ question.question_tex }}</a></li>
     {% endfor %}
     </ul>
   {% else %}  <p>No polls are available.</p>  {% endif %}
</body> </html>
```

- Le tag {% … %} permet d'insérer un code python (if, for…)
- L'expression {{…}} permet d'afficher directement des données dans la page HTML associée à la template.
- {% url … %} permet de supprimer la dépendance en chemins d'URL en utilisant le paramètre « name » défini par les fonctions path() du module Sondage.urls.

Les vues et les templates

Fonction detail du fichier « Sondage/views.py»

```
from django.http import Http404
from django.shortcuts import get_object_or_404
def detail(request, question_id):
    # 1ière méthode
    try:
        question = Question.objects.get(pk=question_id)
    except Question.DoesNotExist:
        raise Http404("Question does not exist")
    #2ème méthode raccorci
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'Sondage/detail.html', {'question': question})
```

- La vue lève une exception de type Http404 si une question avec l'ID demandé n'existe pas.
- La fonction get_object_or_404() prend un modèle Django comme premier paramètre et un nombre arbitraire de paramètres mots-clés, qu'il transmet à la méthode get() du gestionnaire du modèle. Elle lève une exception Http404 si l'objet n'existe pas.

Les vues et les templates

Fichier « Sondage/templates/Sondage/detail.html »

```
<!DOCTYPE html> <html lang="en">
<head> <meta charset="UTF-8">  <title>Title …</title> </head>
<body>
   <h1>{{ question.question_text }}</h1>
   <ul>
       {% for choice in question.choice_set.all %}
          <li>{{ choice.choice_text }}</li>
       {% endfor %}
   </ul>
</body> </html>
```

L'appel de méthode a lieu dans la boucle {% for %} : question.choice_set.all est interprété comme le code Python question.choice_set.all(), qui renvoie un itérable d'objets Choice et qui convient pour l'utilisation de la balise {% for %}.

Espaces de noms et noms d'URL

- Dans des projets Django comportant plusieurs applications, il est possible de différencier les noms d'URL de ces applications par l'ajout d'espaces de noms à la configuration d'URL.
- Dans le fichier «Sondage/urls.py» , ajouter une variable app_name pour définir l'espace de nom de l'application :

```
app_name = 'Sondage'
urlpatterns = [
   … ]
```

- Dans le template « Sondage/templates/Sondage/index.html » faire référence à l'espace de nom correspondant :

```
<li><a href ="{% url 'Sondage:detail' question.id %}">
        {{ question.question_text }}</a></li>
```

Les formulaires en Django

Fichier Sondage/templates/Sondage/detail.html

```
<!DOCTYPE html> <html lang="en">
<head> <meta charset="UTF-8">  <title>Title …</title> </head> <body>
<form action="{% url 'Sondage:vote' question.id %}" method="post">
    {% csrf_token %}
    <fieldset>
        <legend><h1>{{ question.question_text }}</h1></legend>
        {% if error_message %}<p><strong>{{error_message}}</strong></p>{% endif %}
        {% for choice in question.choice_set.all %}
          <input type="radio" name="choice" id="choice{{ forloop.counter }}"
                  value="{{ choice.id }}">
          <label for="choice{{ forloop.counter }}">{{ choice.choice_text }}</label><br>
        {% endfor %}
    </fieldset>
    <input type="submit" value="Vote">
</form> </body> </html>
```

- {% csrf_token %} permet de s'en protéger des attaques inter-sites (pour POST)
- {% url 'Sondage:vote' question.id %} est l'attribut action du formulaire
- forloop.counter indique combien de fois la boucle for a été exécutée.

Les formulaires en Django

Fonction vote du fichier « Sondage/views.py»

```
from django.http import HttpResponseRedirect
from django.urls import reverse
def vote(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    try:
        selected_choice = question.choice_set.get(pk=request.POST['choice'])
    except (KeyError, Choice.DoesNotExist):  # Redisplay the question voting form.
        return render(request, 'Sondage/detail.html', { 'question': question,
                    'error_message': "You didn't select a choice.", })
    else:
        selected_choice.votes += 1
        selected_choice.save()
        return    HttpResponseRedirect(reverse('Sondage:results',    args=(question.id,)))
```

- request.POST permet d'accéder aux données envoyées par leurs clés.
- l'exception KeyError est levé si choice n'est pas spécifié
- Il faut systématiquement renvoyer une HttpResponseRedirect après avoir correctement traité les données POST.
- La fonction reverse recoit en paramètre la vue vers laquelle nous voulons se rediriger

Amélioration des templates

Fonction results du fichier « Sondage/views.py»

```
def results(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, 'Sondage/results.html', {'question': question})
```

Fichier Sondage/templates/Sondage/ results.html

```
<!DOCTYPE html> <html lang="en">
<head> <meta charset="UTF-8">  <title>Title …</title> </head> <body>
    <h1>{{ question.question_text }}</h1>
    <ul>
    {% for choice in question.choice_set.all %}
        <li>{{choice.choice_text}} -- {{choice.votes}} vote{{choice.votes|pluralize}}
        </li> {% endfor %}
    </ul>
    <a href="{% url 'Sondage:detail' question.id %}">Vote again?</a>
</body></html>
```


Personnalisation de l'apparence des templates

- La module « django.contrib.staticfiles » assure la collection des fichiers statiques des applications pour les mettre dans un seul emplacement qui peut être facilement configuré pour servir les fichiers en production.
- Créer un répertoire nommé static dans le répertoire Sondage. À l'intérieur de ce répertoire static créer un autre répertoire nommé Sondage dans lequel on place un fichier style.css.

Créer un fichier « Sondage/ /static/Sondage/style.css »

```
li a { … }
```

Définir dans les templates un lien vers le fichier de style

```
{% load static %}
<link rel="stylesheet" type="text/css"
                href="{% static 'Sondage/style.css' %}">
```

La balise de gabarit {% static %} génère l'URL absolue des fichiers statiques.