# CHAPITRE 2

## MANIPULER LES DONNÉES

1- Manipulation des fonctions/lambda

2- Listes, tuples, dictionnaires, ensembles (set)

3- Fichiers de données (textes, csv…)

4- Bibliothèques standards (math, chaines de caractères, rand…)

# Manipulation des fonctions

Une fonction est un bloc de code qui ne s'exécute que lorsqu'elle est appelée.  Vous pouvez transmettre des données, appelées paramètres, à une fonction.

Une fonction peut renvoyer des données en conséquence.

Pour appeler une fonction, utilisez le nom de la fonction suivi de parenthèses:

```
>>>def my_function():
>>>      print("Hello from a function")

>>>my_function()
```

# Manipulation des fonctions

Les informations peuvent être transmises aux fonctions comme arguments.

Les arguments sont spécifiés après le nom de la fonction, entre parenthèses.

Vous pouvez ajouter autant d'arguments que vous le souhaitez, séparez-les simplement par une virgule.

```
>>>def my_function(fname):
>>>        print(fname + " Refsnes")

>>>my_function("Emil")
>>>my_function("Tobias")
>>>my_function("Linus")
```

# Manipulation des fonctions

## Valeur de paramètre par défaut

```
>>>def my_function(country = "Norway"):
>>>        print("I am from " + country)

>>>my_function("Sweden")
>>>my_function("India")
>>>my_function()
>>>my_function("Brazil")
```

## Fonction de retour

```
>>>def my_function(x):
 >>> return 5 * x
>>>print(my_function(3))
 >>>print(my_function(5))
 >>>print(my_function(9))
```

# Fonction Lamda

## Définition

En Python, le mot clé **Lamda** est utilisé pour déclarer une fonction anonyme (sans nom), raison pour laquelle ces fonctions sont appelées « fonction Lamda » .

Une fonction Lamda est comme n'importe quelle fonction Python normale, sauf qu'elle n'a pas de nom lors de sa définition et qu'elle est contenue dans une ligne

Tout comme la définition d'une fonction normale par l'utilisateur à l'aide du mot clé 'def', une fonction Lamda est définie à l'aide du mot clé 'Lamda '.

Une fonction Lamda peut avoir 'n' nombre d'arguments mais une seule expression

# Fonction Lamda

## Syntaxe


Exemple

Voici le code d'une fonction normale pour calculer la somme de deux valeurs en Python

```
def sommeClassique( a , b ):
  return a + b
```

Avec une fonction Lamda le code devient

```
sommeLambda = lambda a,b : a+b
```

# CHAPITRE 2

## MANIPULER LES DONNÉES

1- Manipulation des fonctions/lambda

**2- Listes, tuples, dictionnaires, ensembles (set)**

3- Fichiers de données (textes, csv…)

4- Bibliothèques standards (math, chaines de caractères, rand…)

# Les tableaux dynamiques(Liste)

Une liste est une collection qui est commandée et modifiable.

En Python, les liste sont écrites entre crochets.

```
>>>thislist = ["apple", "banana", "cherry"]
>>>print(thislist)
['apple', 'banana', 'cherry']
>>>print(thislist[1])
banana

>>>thislist = ["apple", "banana", "cherry"]
>>>print(thislist[-1])#afficher la valeur de la position -1 (cycle)
cherry
>>>thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
>>>print(thislist[2:5])#afficher les valeurs de la position 2 jusqu'à 5 (5 non inclus)
['cherry', 'orange', 'kiwi']
```

# Les tableaux dynamiques(Liste)

**Modification de la valeur d”un élément du tableau**

```
>>>thislist = ["apple", "banana", "cherry"]
>>>thislist[1] = "blackcurrant“#modifier la valeur de la 1ere position
>>>print(thislist)
['apple', 'blackcurrant', 'cherry']
```

# Les tableaux dynamiques(Liste)

**Afin de parcourir une liste**

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

**Afin de chercher un élément dans une liste**

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

# Les tableaux dynamiques(Liste)

**Afin de retrouver la longueur d'une liste**

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

**Afin d'ajouter un élément à la liste**

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

# Les tableaux dynamiques(Liste)

**Afin d'ajouter un élément à une position de la liste**

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

**Afin de supprimer  un élément de la liste**

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

# Les tableaux dynamiques(Liste)

**Afin de supprimer  le dernier élément de la liste**

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

**Afin de supprimer  un élément de la liste**

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

# Les tableaux dynamiques(Liste)

**Afin de joindre deux listes**

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
# une deuxième façon
list1.extend(list2)
print(list1)
```

# Les tableaux dynamiques(Liste)

## Autres méthodes

| Method | Description |
|---|---|
| append() | Adds an element at the end of the list |
| clear() | Removes all the elements from the list |
| copy() | Returns a copy of the list |
| count() | Returns the number of elements with the specified value |
| extend() | Add the elements of a list (or any iterable), to the end of the current list |
| index() | Returns the index of the first element with the specified value |
| insert() | Adds an element at the specified position |
| pop() | Removes the element at the specified position |
| remove() | Removes the item with the specified value |
| reverse() | Reverses the order of the list |
| sort() | Sorts the list |

# Les tableaux statiques (tuple)

Un tuple est une collection ordonnée et non changeable.  En Python, les tuples ont écrits avec des crochets ronds.

```
>>>thistuple = ("apple", "banana", "cherry")
>>>print(thistuple)
('apple', 'banana', 'cherry')
>>>print(thistuple[1])#accéder à la valeur 1
banana
```

Convertissez le tuple en liste pour pouvoir le modifier

```
>>>x = ("apple", "banana", "cherry")
>>>y = list(x)#convertir le tuple en une liste
>>>y[1] = "kiwi"#changer la valeur de la 1er position de la liste
>>>x = tuple(y)#convertir la liste en un tuple
>>>print(x)
```

# Les tableaux statiques (tuple)

**Afin de retrouver la longueur un tuple**

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

**Afin de parcourir un tuple**

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

**Afin de vérifier si un élément est dans un tuple**

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

# Les tableaux statiques (tuple)

**Afin de supprimer un tuple**

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

**Afin de joindre deux tuples**

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

# Les tableaux statiques (tuple)

## Autres méthodes

| Method | Description |
|---|---|
| count() | Returns the number of times a specified value occurs in a tuple |
| index() | Searches the tuple for a specified value and returns the position of where it was found |

# Les tableaux statiques (set)

Un set est une collection non ordonnée et non indexée.  En Python, les sets sont écrits avec des accolades.

**Afin de créer un set**

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

**Afin de parcourir un set**

```
for x in thisset:
  print(x)
```

# Les tableaux statiques (set)

### Afin de vérifier si un élément est dans un set

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Une fois qu'un ensemble est créé, vous ne pouvez pas modifier ses éléments, mais vous pouvez ajouter de nouveaux éléments.

### Afin d'ajouter à un set

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
```

# Les tableaux statiques (set)

**Afin d'ajouter plusieurs éléments à un set**

```
thisset = {"apple", "banana", "cherry"}
thisset.update(["orange", "mango", "grapes"])

print(thisset)
```

**Afin de supprimer un élément d'un set**

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
#ou bien
thisset.discard("banana")
print(thisset)
```

# Les tableaux statiques (set)

**Afin de supprimer le dernier élément d'un set**

```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

**Afin de joindre deux sets**

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)

#Ou bien

set1.update(set2)
print(set1)
```

# Les tableaux statiques (set)

**Afin de vider un set**

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()

print(thisset)
```

**Afin de supprimer un set**

```
thisset = {"apple", "banana", "cherry"}
del thisset

print(thisset)
```

# Les tableaux statiques (set)

Autres méthodes:

| Method | Description |
|---|---|
| add() | Adds an element to the set |
| clear() | Removes all the elements from the set |
| copy() | Returns a copy of the set |
| difference() | Returns a set containing the difference between two or more sets |
| difference_update() | Removes the items in this set that are also included in another, specified set |
| discard() | Remove the specified item |
| intersection() | Returns a set, that is the intersection of two other sets |
| intersection_update() | Removes the items in this set that are not present in other, specified set(s) |
| isdisjoint() | Returns whether two sets have a intersection or not |
| issubset() | Returns whether another set contains this set or not |
| issuperset() | Returns whether this set contains another set or not |
| pop() | Removes an element from the set |
| remove() | Removes the specified element |

# Les dictionnaires

Un dictionnaire est une collection non ordonnée, modifiable et indexée.  En Python, les dictionnaires sont écrits avec des accolades, et ils ont des clés et des valeurs.

```
>>>thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
>>>print(thisdict)
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}


>>>x = thisdict["model"] #pour accéder à une valeur précise
>>>print(x)
Mustang
```

# Les dictionnaires

**Afin de parcourir les clés d'un dictionnaire**

```
for x in thisdict:
  print(x)
```

**Afin de parcourir les valeurs d'un dictionnaire**

```
for x in thisdict:
  print(thisdict[x])

#une deuxième façon
for x in thisdict.values():
  print(x)
```

# Les dictionnaires

**Afin de modifier une valeur d'une clé particulière**

```
thisdict["year"] = 2018#modier une valeur
print(thisdict)
```

**Afin de retrouver une valeur particulière d'une clé**

```
x = thisdict.get("model")
```

**Afin de parcourir les clés et les valeurs d'un dictionnaire à la fois**

```
for x, y in thisdict.items():
  print(x, y)
```

# Les dictionnaires

**Afin de vérifier si une clé existe**

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

**Afin de retrouver la longueur**

```
print(len(thisdict))
```

# Les dictionnaires

**Afin d'ajouter à un dictionnaire**

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

**Afin de supprimer  d'un dictionnaire**

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
#ou bien pour supprimer le dernier élément : thisdict.popitem()
```

# Les dictionnaires

**Afin de vider un dictionnaire**

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

**Afin de copier un dictionnaire**

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

# Les dictionnaires

## Autres méthodes

| Method | Description |
|---|---|
| clear() | Removes all the elements from the dictionary |
| copy() | Returns a copy of the dictionary |
| fromkeys() | Returns a dictionary with the specified keys and value |
| get() | Returns the value of the specified key |
| items() | Returns a list containing a tuple for each key value pair |
| keys() | Returns a list containing the dictionary's keys |
| pop() | Removes the element with the specified key |
| popitem() | Removes the last inserted key-value pair |
| setdefault() | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| update() | Updates the dictionary with the specified key-value pairs |
| values() | Returns a list of all the values in the dictionary |

# CHAPITRE 2

## MANIPULER LES DONNÉES

1- Manipulation des fonctions/lambda

2- Listes, tuples, dictionnaires, ensembles (set)

**3- Fichiers de données (textes, csv…)**

4- Bibliothèques standards (math, chaines de caractères, rand…)

# L'utilisation des fichiers

**Introduction**

Python a plusieurs fonctions pour créer, lire, mettre à jour et supprimer des fichiers texte. La fonction clé pour travailler avec des fichiers en Python est open().

Elle prend deux paramètres; nom de fichier et mode.

Il existe quatre méthodes (modes) différentes pour ouvrir un fichier:

1. **"r"** - Lecture -Par défaut. Ouvre un fichier en lecture, erreur si le fichier n'existe pas
2. **"a"** - Ajouter -Ouvre un fichier à ajouter, crée le fichier s'il n'existe pas
3. **"w"** - Écrire -Ouvre un fichier pour l'écriture, crée le fichier s'il n'existe pas
4. **"x"** - Créer -Crée le fichier spécifié, renvoie une erreur si le fichier existe

# L'utilisation des fichiers

**Ouvrir et lire un fichier**

Pour ouvrir le fichier, utilisez la fonction open() intégrée. La fonction open() renvoie un objet fichier, qui a une méthode read() pour lire le contenu du fichier

```
>>>f = open("demofile.txt", "r")
>>>print(f.read())
```

Renvoyez les 5 premiers caractères du fichier:

```
>>>f = open("demofile.txt", "r")
>>>print(f.read(5))
```

Vous pouvez renvoyer une ligne en utilisant la méthode readline ():

```
>>>f = open("demofile.txt", "r")
>>>print(f.readline())
```

# L'utilisation des fichiers

**Ecrire dans un fichier**

Pour écrire dans un fichier existant, vous devez ajouter un paramètre à la fonction open():

```
>>>f = open("demofile2.txt", "a")
>>>f.write("Now the file has more content!")
>>>f.close()
>>>f = open("demofile2.txt", "r")#afficher le ficher après modification
>>>print(f.read())

>>>f = open("demofile3.txt", "w")
>>>f.write("Woops! I have deleted the content!")
>>>f.close()
# ouvrez et lisez le fichier après l'ajout:
>>>f = open("demofile3.txt", "r")
>>>print(f.read())
```

La méthode "w" écrasera tout le fichier.

# L'utilisation des fichiers

**Fermeture d'un fichier**

Il est recommandé de toujours fermer le fichier lorsque vous en avez terminé.

```
>>>f = open("demofile.txt", "r")
>>>print(f.readline())
>>>f.close()
```

**Suppression d'un fichier**

Pour supprimer un fichier, vous devez importer le module OS et exécuter sa fonction os.remove ():

```
>>>import os
>>>os.remove("demofile.txt")
```

# Le format CSV

**Introduction**

Il existe différents formats standards de stockage de données. Il est recommandé de favoriser ces formats car il existe déjà des modules Python permettant de simplifier leur utilisation.

Le fichier Comma-separated values (CSV) est un format permettant de stocker des tableaux dans un fichier texte. Chaque ligne est représentée par une ligne de texte et chaque colonne est séparée par un séparateur (virgule, point-virgule …).

Les champs texte peuvent également être délimités par des guillemets.

Lorsqu'un champ contient lui-même des guillemets, ils sont doublés afin de ne pas être considérés comme début ou fin du champ.

Si un champ contient un signe pouvant être utilisé comme séparateur de colonne (virgule, point-virgule …) ou comme séparateur de ligne, les guillemets sont donc obligatoires afin que ce signe ne soit pas confondu avec un séparateur.

# Le format CSV

Données sous la forme d'un tableau

| Nom | Prenom | Age |
|---|---|---|
| Dubois | Marie | 29 |
| Duval | Julien "Paul" | 47 |
| Jacquet | Bernard | 51 |
| Martin | Lucie;Clara | 14 |

Données sous la forme d'un fichier CSV

```
Nom;Prénom;Age
"Dubois";"Marie";29
"Duval";"Julien ""Paul""";47
Jacquet;Bernard;51
Martin;"Lucie;Clara";14
```

# Le format CSV

Le module csv de Python permet de simplifier l'utilisation des fichiers CSV

**Lecture d'un fichier CSV**

Pour lire un fichier CSV, il faut ouvrir un flux de lecture de fichier et ouvrir à partir de ce flux un lecteur CSV.

**Exemple**

```
import csv
fichier = open("noms.csv", "rt")
lecteurCSV = csv.reader(fichier,delimiter=";")  # Ouverture du lecteur CSV en lui fournissant le caractère séparateur (ici ";")
for ligne in lecteurCSV:
        print(ligne)
fichier.close()
```

```
['Nom ', 'Prenom', 'Age']
['Dubois', 'Marie', '29']
['Duval', 'Julien "Paul" ', '47']
['Jacquet', 'Bernard', '51']
['Martin', 'Lucie;Clara', '14']
```

# Le format CSV

Il est également possible de lire les données et obtenir un dictionnaire par ligne contenant les données en utilisant DictReader au lieu de reader

**Exemple**

```
import csv
fichier = open("noms.csv", "rt")
lecteurCSV = csv.DictReader(fichier,delimiter=";")
for ligne in lecteurCSV:
                print(ligne
fichier.close()
```

```
{‘Nom ': 'Dubois', 'Prenom': 'Marie', 'Age': '29'}
{'Nom ': 'Duval', 'Prenom': 'Julien "Paul" ', 'Age': '47'}
{'Nom ': 'Jacquet', 'Prenom': 'Bernard', 'Age': '51'}
{‘Nom ': 'Martin', 'Prenom': 'Lucie;Clara', 'Age': '14'}
```

# Le format CSV

**Écriture dans un fichier CSV**

À l'instar de la lecture, on ouvre un flux d'écriture et on ouvre un écrivain CSV à partir de ce flux :

```
import csv
fichier = open("annuaire.csv", "wt")
ecrivainCSV = csv.writer(fichier,delimiter=";")
ecrivainCSV.writerow(["Nom","Prénom","Téléphone"])       # On écrit la ligne d'en-tête avec le titre des colonnes
ecrivainCSV.writerow(["Dubois","Marie","0198546372"])
ecrivainCSV.writerow(["Duval","Julien \"Paul\"","0399741052"])
ecrivainCSV.writerow(["Jacquet","Bernard","0200749685"])
ecrivainCSV.writerow(["Martin","Julie;Clara","0399731590"])
fichier.close()
```

```
Nom;Prénom;Téléphone
Dubois;Marie;0198546372
Duval;"Julien ""Paul""";0399741052
Jacquet;Bernard;0200749685
Martin;"Julie;Clara";0399731590
```

# Le format CSV

Il est également possible d'écrire le fichier en fournissant un dictionnaire par ligne à condition que chaque dictionnaire possède les mêmes clés.

Il faut également fournir la liste des clés des dictionnaires avec l'argument fieldnames :

```
import csv
bonCommande = [
		{"produit":"cahier", "reference":"F452CP", "quantite":41, "prixUnitaire":1.6},
		{"produit":"stylo bleu", "reference":"D857BL", "quantite":18, "prixUnitaire":0.95},
		{"produit":"stylo noir", "reference":"D857NO", "quantite":18, "prixUnitaire":0.95},
		{"produit":"équerre", "reference":"GF955K", "quantite":4, "prixUnitaire":5.10},
		{"produit":"compas", "reference":"RT42AX", "quantite":13, "prixUnitaire":5.25}
]
fichier = open("bon-commande.csv", "wt")
ecrivainCSV = csv.DictWriter(fichier,delimiter=";",fieldnames=bonCommande[0].keys())
ecrivainCSV.writeheader()		# On écrit la ligne d'en-tête avec le titre des colonnes
for ligne in bonCommande:
		ecrivainCSV.writerow(ligne)
fichier.close()
```

```
reference;quantite;produit;prixUnitaire
F452CP;41;cahier;1.6
D857BL;18;stylo bleu;0.95
D857NO;18;stylo noir;0.95
GF955K;4;équerre;5.1
RT42AX;13;compas;5.25
```

# Le format JSON

Le format JavaScript Object Notation (JSON) est issu de la notation des objets dans le langage JavaScript.

Il s'agit aujourd'hui d'un format de données très répandu permettant de stocker des données sous une forme structurée.

Il ne comporte que des associations clés → valeurs (à l'instar des dictionnaires), ainsi que des listes ordonnées de valeurs (comme les listes en Python).

Une valeur peut être une autre association clés → valeurs, une liste de valeurs, un entier, un nombre réel, une chaîne de caractères, un booléen ou une valeur nulle.

Sa syntaxe est similaire à celle des dictionnaires Python.

# Le format JSON

Exemple de fichier JSON

```
{
   "Dijon":{
                "nomDepartement": "Côte d'Or",
                "codePostal": 21000,
                "population": {
                                "2006": 151504,
                                "2011": 151672,
                                "2014": 153668
                }
   },
   "Troyes":{
                "nomDepartement": "Aube",
                "codePostal": 10000,
                "population": {
                                "2006": 61344,
                                "2011": 60013,
                                "2014": 60750
                }
   }
}
```

# Le format JSON

**Lire un fichier JSON**

La fonction loads (texteJSON) permet de décoder le texte JSON passé en argument et de le transformer en dictionnaire ou une liste.

```
 import json
fichier = open("villes.json","rt")
villes = json.loads(fichier.read())
print(villes)
fichier.close()
```

```
{'Troyes': {'population': {'2006': 61344, '2011': 60013, '2014': 60750},
'codePostal': 10000, 'nomDepartement': 'Aube'}, 'Dijon': {'population':
{'2006': 151504, '2011': 151672, '2014': 153668}, 'codePostal': 21000,
'nomDepartement': "Côte d'Or"}}
```

# Le format JSON

**Écrire un fichier JSON**

la fonction dumps(variable, sort_keys=False)  transforme un dictionnaire ou une liste en texte JSON en fournissant en argument la variable à transformer.

La variable sort_keys permet de trier les clés dans l'ordre alphabétique.

```
import json
quantiteFournitures = {"cahiers":134, "stylos":{"rouge":41,"bleu":74}, "gommes": 85}
fichier = open("quantiteFournitures.json","wt")
fichier.write(json.dumps(quantiteFournitures))
fichier.close()
```

# CHAPITRE 2

## MANIPULER LES DONNÉES

1- Manipulation des fonctions/lambda

2- Listes, tuples, dictionnaires, ensembles (set)

3- Fichiers de données (textes, csv…)

**4- Bibliothèques standards (math, chaines de caractères, rand…)**

# Bibliothèques standards

Python dispose d'une très riche bibliothèque de modules (classes (types d'objets), fonctions, constantes, …) étendant les capacités du langage dans de nombreux domaines: nouveaux types de données, interactions avec le système, gestion des fichiers et des processus, protocoles de communication (internet, mail, FTP, etc.), multimédia, etc.

Certains des modules importants sont:

- math pour les utilitaires mathématiques
- re pour les expressions régulières
- Json pour travailler avec JSON
- Datetime pour travailler avec des dates

# Le module math

Le module **math** nous fournit un accès à de nombreuses fonctions permettant de réaliser des opérations mathématiques comme le calcul d'un sinus, cosinus, d'une tangente, d'un logarithme ou d'une exponentielle

Les fonctions les plus couramment utilisées sont les suivantes :

- ***ceil() et floor()*** renvoient l'arrondi du nombre passé en argument en arrondissant respectivement à l'entier supérieur et inférieur ;
- ***fabs()*** renvoie la valeur absolu d'un nombre passé en argument ;
- ***isnan()*** renvoie True si le nombre passé en argument est NaN = Not a Number (pas un nombre en français) ou False sinon ;
- ***exp()*** permet de calculer des exponentielles ;
- ***log()*** permet de calculer des logarithmes ;
- ***sqrt()*** permet de calculer la racine carrée d'un nombre ;
- ***cos(), sin() et tan()*** permettent de calculer des cosinus, sinus et tangentes et renvoient des valeurs en radians.

Les fonctions de ce module ne peuvent pas être utilisées avec des nombres complexes. Pour cela, il faudra plutôt utiliser les fonctions **du module cmath**.

Le module math définit également des constantes mathématiques utiles comme pi ou le nombre de Neper, accessibles via **math.pi et math.e.**

# Le module math

```
import math
print(math.ceil(3.1))
print(math.ceil(3.9))
print(math.floor(3.1))
print(math.floor(-4))
print(math.pi)
```

```
4
4
3
-4
3.141592653589793
```

# Le module random

**random** fournit des outils pour générer des nombres pseudo-aléatoires de différentes façons.

**La fonction random()** est le plus utilisée du module. Elle génère un nombre à virgule flottante aléatoire de façon uniforme dans la plage semi-ouverte [0.0, 1.0).

**La fonction uniform()** génère un nombre à virgule flottante aléatoire compris dans un intervalle. Cette fonction a deux arguments : le premier nombre représente la borne basse de l'intervalle tandis que le second représente la borne supérieure.

```
import random
print(random.random())
print(random.random())
print(random.uniform(10 , 100))
```

```
0.28349953961279983
0.007815000784710868
44.309364272132456
```