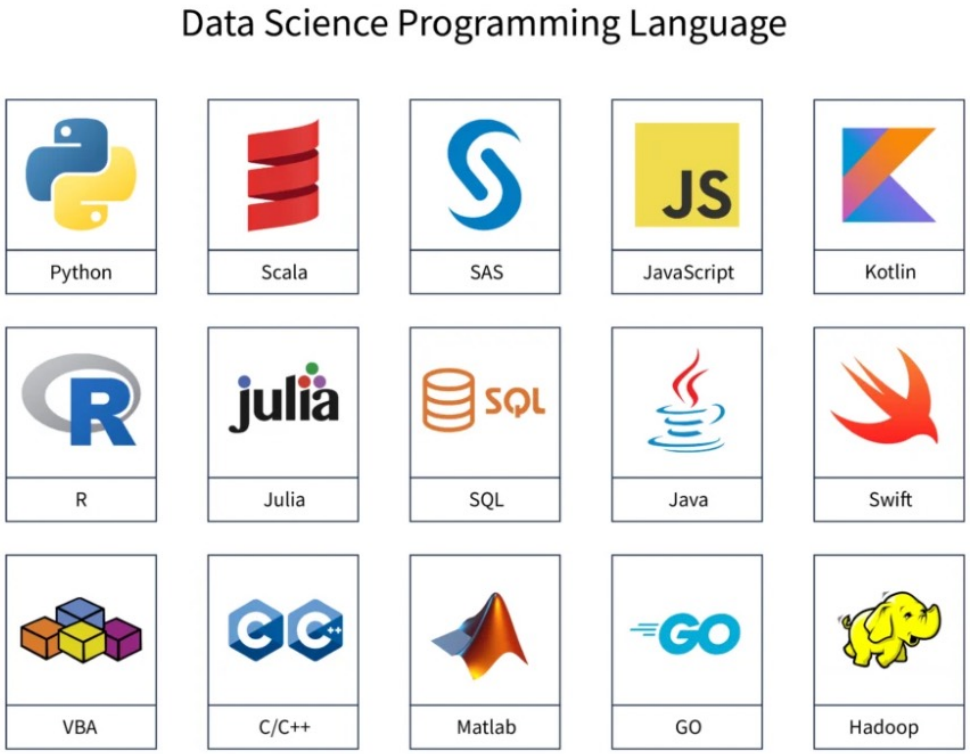
Chapitre I

Introduction sommaire au Python

Langages de programmation pour les statistiques et la science des données



https://www.scaler.com/blog/programming-languages-for-data-science/

Pourquoi Python est Numéro 1

- Python est simple et facile à apprendre
- Il est lisible est intuitif.
- Il est célèbre par son large choix de bibliothèques.
- Multiplateformes
- Sa polyvalence permet une utilisation pour différentes tâches de la science des données.
  - La manipulation des données
  - L'analyse des données
  - Le développement de modèles statistiques
  - La visualisation
- C'est un langage productif, le temps de développement est réduit par rapport aux langages comme C,C++,etc
- La popularité de Python garantit une communauté très impliquée et des ressources abondantes.

Introduction sommaire au Python

- Historique du langage
- Structure d'un programme Python
- Les variables / Les types / Les conversions de type
- Les expressions et les opérateurs
- Les entrées sorties
- Les structures de contrôle
- Les structures de données
- Les fonctions

format page 32

Historique du langage Python

- Le langage Python a été créé en 1989 par Guido Van Rossum
- Il a beaucoup évolué au fil des années (Python1.0 en 1994, 2.0, en 2000, 3.0 en 2008)
- C'est un langage gratuit sous licence libre
- L'origine de l'appellation du Python est liée au groupe comique Britannique « Monty Python »
- Le nom de l'interpréteur IDLE est inspiré du nom d'un membre fondateur du groupe « Eric IDLE »

Un script (Programme) Python

- Un script Python est une suite de définitions et de commandes; les définitions sont évaluées et les commandes sont exécutées par l'interpréteur Python.
- Les espaces sont significatifs en Python : en particulier l'indentation et le placement des nouvelles lignes. Les deux points et l'indentation permettent la définition de blocs.
  - Exemple:

```
>>> heure = 21
>>> if heure > 20 :
...     print("Il est probablement la nuit")
... else :
...     print("Il est probablement le jour")
```

- Un script peut comporter des commentaires
  - Un commentaire commence par un « # »
  - # voici un exemple d'un commentaire
  - Sur les versions récentes, on peut avoir des commentaires sur plusieurs lignes en les entourant par ''' ou """
- Les docstrings sont des commentaires sur une ou plusieurs lignes qu'on trouve au début d'une fonction, d'une classe ou d'un module, ils sont entourés des symboles ''' ou """

""" Ce module comporte des fonctions de mesures statistiques """	''' Cette fonction calcule la moyenne '''

Les variables en Python

- Un nom de variable est composé de lettres, chiffres et de blancs soulignés et ne peut commencer par un chiffre.
- Les noms de variables sont sensibles à la casse
- Les types de variables n'ont pas besoin d'être déclarés; Python décide lui-même du type de la variable
- Par convention les noms de variables sont en minuscule
- La variable est créée lors de la première affectation. Le nom comportera la référence d'un objet (une référence à une adresse mémoire)
- L'objet référencé par la variable comporte trois composantes
  - Un « id »
  - Un « type »
  - Une « valeur »
- Les variables peuvent être mutables ou non mutables « immuable »

Exemples

- *Objet immuable*

```
>>> x = 3
>>> x
3
>>> id (x)
140397133219944
>>> type(x)
<type 'int'>
```

- *Objet mutable*

```
>>> lis = [ ]
>>> id(lis)
 4417685624
>>> lis.append(10)
>>> lis
[10]
>>> id(lis)
4417685624
```

- *Objet immuable*

```
>>> x = 5
>>> id (x)
140568923530088
>>> x = x + 1
>>> id(x)
140568923530064
>>> x = x - 3
>>> id(x)
140397133219944
```

Les types de variables

- *Les types de base (primitifs)*
  - *Les nombres*
    - *Les entiers*
    - *Les réels*
    - *Les complexes*
  - *Les chaînes de caractères.*
  - *Les types à valeur unique*
  - *Les booléens*
    - *Deux objets uniques True et False*
    - *Les éléments qui sont équivalents à Faux sont :*
      - *False*
      - *0 ; 0.0*
      - *" "*
      - *[] ou {} ou ()*
      - *None*
- *Les types conteneurs ou collection « non primitifs »*
  - *Les séquences*
    - *Les listes*
    - *Les tuples*
  - *Les ensembles*
  - *Les dictionnaires*

Conversion de type « CAST »

- Cela permet de convertir un objet d'un certain type vers un autre type

```
>>> float(3)          # permet de convertir l'entier 3 au réel 3.0
>>> float('3.14')     # permet de convertir une chaîne de caractères en un réel 3.14
>>> int(3.9)          # permet de tronquer le réel 3.9 en un entier 3
>>> int('45')         # permet de convertir une chaîne de caractères en un entier 45
>>> str(24)           # permet de convertir un entier en une chaîne de caractères '24'
>>> str(4.38e4)       # permet de convertir le réel 4.38e4 en une chaînes de caractères '43800.0'
```

```
>>> bin(13)           # permet de convertir l'entier en binaire '0b1101'
>>> oct(13)           # permet de convertir l'entier en représentation octale '015'
>>> hex(2345)         # permet de convertir l'entier en représentation hexadécimale '0x929'
>>> bool ("mot")      # permet de convertir une chaîne non vide en True
```

Les expressions & les opérateurs

- Une expression est formée en combinant des objets et des opérateurs , elle a une valeur qui a un type. < objet > < operateur > < objet >
- L'ordre d'interprétation des opérateurs dit « PEDMAS » est :
  - Parenthèses « () » ; Exposants « ** » ; Division « / » ; Multiplication « * » ; Addition | Soustraction « + | - »
- Les types d'opérateurs.
  - Opérateur d'affectation : « = »

x = 5	*y , z = 3 , 'Toto'*	*(y , z) = (z , y)*	*a = b = 21*	*w , k = [5 , 9]*

  - Opérateurs arithmétiques
    - « + » , « - » , « * » , « / » , « // » , « % » , « ** »
    - « += », « -= », « *= » , « /= » , « //= », « %= », « **= »
  - Opérateurs de comparaison : > , >= , < , <= , == , !=
  - Opérateurs logiques : and , or , not
  - Opérateurs binaires : ~ , << , >> , & , ^ , |
  - Opérateurs d'identité : is , is not
  - Opérateur d'appartenance : in , not in

Ecriture / Lecture

Pour l'affichage, on utilise la fonction « print »	Pour la saisie, on utilise la fonction « input »

>>> x = 1
>>> print (x)
1
>>> print("mon numéro favori est", x, ".", "x =", x)
mon numéro favori est 1 . x = 1
>>> print("mon numéro favori est" + str(x) + "." + "x =" + str(x))
mon numéro favori est1. x =1
 	
>>> print(' "Python!", C\'est facile ')
"Python!", C'est facile
>>> print(''' "Python!", C'est facile ''')
"Python!", C'est facile
>>> print(""" "Python!", C'est facile """)
"Python!", C'est facile
>>> print(" \"Python!\", C'est facile")
"Python!", C'est facile
	>>> z = input ()
	23
	
	>>> v = input (" Entrez une valeur ")
	Entrez une valeur 25
	
	>>> num = int(input(" Entrez une valeur "))
	Entrez une valeur 5
	
	>>> print(5*num)
	25

Les structures conditionnelles

Nom de la structure	Syntaxe	Exemple 1	Exemple 2
if	if condition : ⟷action(s)	if note >= 10 : print(' Validé ')	if temp_J >= 31 and temp_N > 21 : print(' Canicule ') nb_JC += 1
if else	if condition : ⟷action(s)_if else : ⟷action(s)_else	if note >= 10 : print(' Validé ') else : print(' A refaire ')	if a > 0 : print (a, "est strictement positif") else : if a < 0 : print (a, "est strictement négatif") else : print (a, "est nul")
if elif else	if condition1 : ⟷action(s)_if elif condition2 : ⟷action(s)_elif .... elif conditionN : ⟷action(s)_elif : else : ⟷action(s)_else	if note >= 16 : if note == 20 : print("Excellent") else : print("Très Bien") elif note >= 12 : print("Bien") elif note >= 9 : print("Moyen") else : print("Faible")	if a > 0 : print (a, "est strictement positif") elif a < 0 : print (a, "est strictement négatif") else : print (a, "est nul")

Les boucles

Nom de la boucle	Syntaxe	Exemple 1	Exemple 2
while	initialisation while condition(s) : ⟷action(s) ⟷incrémentation	x = 0 while x < 10 : print(x) x += 1	x = 1 while x < 10 : y = 1 while y < 10 : print(x * y) y = y + 1 x = x + 1
for	for iterator in element : ⟷action(s)	for car in "chaine" : print(car)	for i in range(5) : print(i)

Les chaînes de caractères

- Une liste de caractères
  - Une chaîne de caractères est une séquence ordonnée de caractères.
  - Elle est codée par défaut (depuis Python 3) en Unicode.
- Une chaîne de caractère est un objet non mutable (le même id).

```
ch = 'texte'
ch[2] = 'n'  // erreur.
```

- Déclaration d'une chaîne

```
>>> ch1 = 'test'
>>> ch2 = " Voici un exemple "
>>> ch3 = ''' une chaîne de caractères peut être
                écrite sur plusieurs lignes '''
>>> ch4 = """ un autre exemple d'une chaîne de
              caractères, écrite sur plusieurs
              lignes """
```

Intégration de valeurs à chaînes de caractères

- *L'opérateur « + » permet la concaténation de deux chaînes pour créer une troisième*

  ch1 = 'Le python' ;

  ch2 = ch1 + 'est facile'

  ch2 : 'Le python est facile'

- L'opérateur « * » permet la répétition d'une chaine

  3 * 'Voici' => 'VoiciVoiciVoici'

- La chaine peut comporter des caractères d'espace

```
ch_s = "C'est une chaine \"spéciale\" \n\tavec des
       espaces!"
```

Parcours d'une chaine

```
ch = 'Exemple'
for ind in range(len(ch)):
    # traitement de l'élément ch[ind]
for char in ch:
    # traitement de char
```

Intégration de valeurs à chaînes de caractères

- Il est possible de remplacer des valeurs dans une chaine de caractères de trois façons différentes

```
chaine_complete = ’’ Python’’
```

```
print(’’ %s pour la statistique ’’ % chaine_complete)
```

```
print(’’ {} pour la statistique ’’.format(chaine_complete))
```

```
print(f’’{chaine_complete} pour la statistique ’’)
```

Exemple: ch = ’Voici un exemple’

chaine	V	o	i	c	i		u	n		e	x	e	m	p	l	e
index_p	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
index_n	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

chaine[#index] = élément & le slicing chaine[début:fin:pas]

ch[0] : V
ch[-16] : V
ch[7] : n
ch[-9] : n
ch[15] : e
ch[-1] : e

ch[0:3] : Voi
ch[-16:-13] : Voi
ch[6:8] : un
ch[-10:-8] : un
ch[9:12] : exe
ch[-7:-4] : exe

ch[::2] : Viiu xml
ch[::-1] : elpmexe un icioV
ch[0:5:2] : Vii
ch[:-8:-2] : epee
ch[1::5] : oue
ch[-7:-12:-4] : e

’un’ in ch : True
’z’ in ch : False
’el’ in ch : False

ch.index(’un’) : 6
ch.index(’e’) : 9
ch.index(’y’) : Erreur

ch.find(’un’) : 6
ch.find(’ici’) : 2
ch.find(’el’) : -1

len(ch) : 16

ch.endswith(’el’) : False
ch.endswith(’le’) : True

new_ch1 = ch.upper()
new_ch1 : VOICI UN EXEMPLE
ch.lower() : voici un exemple

new_ch2 = ch.replace(’Voici’, ’C\’est’)
new_ch2 : C'est un exemple

ch.count(’e’) : 3
ch.count(’ci’) : 1
ch.count(’el’) : 0

Les structures de données

- Il existe différentes collections en python qui comportent des éléments de différents types
- Les collections ordonnées (les séquences)
  - Les tuples
  - Les listes
- Les collections non ordonnées
  - Les ensembles « set »
  - Les dictionnaires

Les tuples vs les listes « Les différences »

- **Le tuple** : C'est une séquence ordonnée immuable, une fois créé, ne peut pas être modifié.
  - Pour créer un tuple vide
    - t_v = () ou t_v = tuple()
  - Avec un seul élément
    - t = (5 ,)
  - Avec éléments
    - t = (2, 'mot' , 3 , 9)
    - t = 2, 'mot' , 3 , 9
  - Modification d'éléments : impossible
    - t[2] = 4 // erreur ; objet immuable
  - Ajout ou suppression d'un élément : impossible.
  - Tri des éléments du tuple : impossible

- **Les listes** : C'est une séquence ordonnée mutable, il est possible de modifier son contenu
  - Pour créer une liste vide
    - l_v = [] ou l_v = list()
  - Avec éléments
    - l = [2, 'mot' , 3 , 9]
  - Modification d'éléments
    - l[2] = 4 **# C'est correct**
    - l[2:4] = [45,78,36,55] **# l'objet est mutable**
  - Ajout d'un élément
    - l.append (9) # Ajout à la fin
    - l.insert (1, 'test') # Ajout à une position
  - Tri des éléments de la liste
    - l.sort()
    - l.reverse()

- **Les listes** : C'est une séquence ordonnée mutable,
  - Concaténer deux listes avec la fonction extend
    - l.extend([3.14, 'Python',9])
  - Suppression d'éléments
    - del(l[1]) # supprime l'élément d'indice 1
    - l.pop() # supprime le dernier élément
    - l.pop(1) # supprime l'élément d'indice 1
    - l.remove(2) # supprime la première occurrence de 2
    - l.clear() # supprime tous les éléments de la liste

Les tuples vs les listes « Les similarités »

- Nombre d'éléments dans un tuple ou une liste:
  - len(t)
  - len(l)
- Concaténer deux tuples ou des listes avec l'opérateur « + »
  - t1 =(1,2,3) ; t2 = (4,5,6)
  - t3 = t1 + t2 => (1,2,3,4,5,6)
  - l1 =[1,2,3] ; l2 = [4,5,6]
  - l3 = l1 + l2 => [1,2,3,4,5,6]
- Répéter le contenu d'un tuple ou d'une liste
  - t4 = t1 * 3 => (1,2,3,1,2,3,1,2,3)
  - l4 = l1 * 3 => [1,2,3,1,2,3,1,2,3]
- Tri dans un nouveau tuple ou liste
  - t1 = sorted(t)
  - l1 = sorted(l)

- Accès aux éléments d'un tuple ou d'une liste

Index positif	Index négatif	par slicing
t[0] : 2 l[1] : 'mot'	t[-2] : 3 l[-1] : 9	t[1:3] : ('mot', 3) t[::3] : (2,9) l[:2:2] : [2] l[1:2] : 'mot'

- L'index d'un élément
  - t1.index(2) : 1
  - l2.index(4) : 0
- La présence d'un élément
  - 'mot' in t : True
  - 4 not in t : True
  - 99 in l : False
  - 2 not in l : False
- Le nombre d'occurrences
  - t1.count(3) : 1
  - l2.count(4) : 1

Les ensembles

- Un ensemble : une collection non ordonnée d'éléments uniques
- Il existe deux versions de « set », une version mutable et une autre immuable
  - Pour un ensemble vide : s_v = set()
  - Avec éléments
    - s = {2, 5.7 , 2 , 9} ou s = set([2, 5.7 , 2 , 9]) # ensemble mutable
    - sf = frozenset([2, 5.7 , 2 , 9]) # ensemble non mutable
  - Nombre d'éléments dans un ensemble : len(s)
  - Ajout d'un élément
    - s.add(10) # C'est correct
    - sf.add(5) # Erreur , objet immutable ; la méthode « add » est non définie
  - Suppression d'un élément
    - s.remove(10) # C'est correct
    - sf.remove(5) # Erreur , objet non mutable ; la méthode « remove » est non définie
- *Opération sur les ensembles :*
  - *L'appartenance d'un élément à l'ensemble « in ou not in»*
  - *La différence et la différence symétrique entre deux ensembles « - »*
  - *L'union de deux ensembles « | »*
  - *L'intersection de deux ensembles « & »*

Les dictionnaires

- Les dictionnaires : une collection non ordonnée d'éléments sous forme de paires « clé : valeur »
- Les clés sont uniques dans un dictionnaire et de type immuable.
- Les valeurs peuvent être redondantes et de n'importe quel type.

- Pour créer un dictionnaire vide
  - d_v = {}
  - d_v = dict()
- Un dictionnaire avec éléments
  - d = {'age' : 22 , 'taille' : 1.75 , 'poids' : 65 }
- Nombre d'éléments dans le dictionnaire : len(d)
- Modification d'un élément
  - d['poids'] = 67
- Ajout d'un élément
  - d['genre'] = 'masculin'
- Suppression d'un élément
  - del(d['poids'])
  - d.pop('age')
  - v = d.pop('taille'). => v comporte 1.75

- Accès aux éléments
  - d[0] # Erreur « KeyError » ; Pas d'accès par indice
  - d['taille'] : 1.75
  - d['genre'] # Erreur « KeyError »
  - d.get('genre','Introuvable') : Introuvable
- Vérification de la présence d'une clé
  - 'genre' in d : False
  - 'adresse' not in d : True
  - 'age' in d : True
- Récupérer la liste des clés ou des valeurs ou des deux
  - d.keys() : ['age', 'taille', 'poids']
  - d.values() : [22, 1.75, 65]
  - d.items() : [('age', 22), ('taille', 1.75), ('poids', 65)]

Parcours des collections de données

- Parcours d'un ensemble

```
s = {45, 'rouge' , 'bleu', 45, (1,5)}
for elem in s:
        # traitement de l'élément elem
```

- Parcours d'une liste

```
f = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
for ind in range (len(f)):
        # traitement de l'élément f[ind]
for elem in f:
    # traitement de l'élément elem
```

- Parcours d'un dictionnaire

```
dico = {"nom" : "Anne" , "note" : 13.5 }
for k in dico:
    # traitement des clés
for cle in dico.keys():
    # traitement des clés
for valeur in dico.values():
    # traitement des valeurs
for cle, valeur in dico.items():
    # traitement des enregistrements : cle, valeur
```

- Parcours d'un tuple

```
t = (5 , 'mot', -6 )
for ind in range(len(t)):
        # traitement de l'élément t[ind]
for elem in t:
    # traitement de l'élément elem
```

Conversion d'une chaine de caractères vers d'autres collections

- Liste vers String

```
lis =['Un', 'exemple']
ch1 = ' '.join(lis)
ch1 = '{}'.format(*lis)
```

- Tuple vers String

```
tup = ('Un', 'exemple')
ch2 = " ".join(tup)
ch2 = '{}'.format(*tup)
```

- Set vers String

```
se = {'Un', 'exemple}
ch3 = " ".join(se)
ch3 = '{}'.format(*se)
```

- String vers liste
  - l = list('test')
  - l = [] + 'test'
  - ch = "un exemple de chaine"

    lis = ch.split()

- String vers set
  - ch = "un exemple de chaine"
  - s = set(ch)
  - s1 = {*ch}

D'une collection à une autre

- Liste vers tuple
  - lis= ['E1', 'E2', 'E3']

```
tup = tuple(lis)
tup = (*lis,)
t = tuple(enumerate(l1))
t = tuple(zip(range(len(lis)),lis))
```

- Tuple vers liste
  - tup= ('E1', 'E2', 'E3')

```
lis = list(tup)
lis=[*tup4]
```

- Liste vers dictionnaire
  - lis= ['E1', 'E2', 'E3']

```
dic1={}.fromkeys(lis, 0) # dict.fromkeys
dic2 = dict(enumerate(lis))
dic3 = dict(zip(range(len(lis)),lis))
```

  - cles = ['E1', 'E2', 'E3']

```
valeurs = [1, 2, 3]
dic4 = dict(zip(cles, valeurs))
```

  - `k_and_v = [[1, 'a'], [2, 'b']]`

```
dict(k_and_v)
```

- Tuple vers dictionnaire
  - tup= ('E1', 'E2', 'E3')

```
dic1=dict.fromkeys(tup, 0) # {}.fromkeys
dic2 = dict(enumerate(tup))
dic3 = dict(zip(range(len(tup)),tup))
```

  - cles = ('E1', 'E2', 'E3')

```
valeurs = (1, 2, 3 )
dic4 = dict(zip(cles, valeurs))
```

- Liste ou tuple vers ensemble
  - lis= ['E1', 'E2', 'E3', 'E2']
  - tup= ('E1', 'E2', 'E3', 'E2')
  - s1 = set(lis)
  - s2 = {*lis}
  - s3 = set(tup)
  - s4 = {*tup}

- Ensemble vers liste ou tuple
  - se= {'E1', 'E2', 'E3'}
  - tup = tuple(se)
  - tup = (*s1,)
  - lis = list(se)
  - lis = [*se]

Les collections de compréhension

- Liste de compréhension
  - res1 = [x+1 for x in une_seq]
  - res1 = [x+1 for x in une_seq if x > 23]
  - res1 = [x+y for x in une_seq for y in une_autre]
  - l = [[x**2,x**3] for x in range(4)]

- Set de compréhension
- {s for s in [1, 2, 3] if s % 2}
- {(m, n) for n in range(2) for m in range(3, 5)}

- Dictionnaire de compréhension
- {i: i*2 for i in range(5)}
- k_and_v = [[1, 'a'], [2, 'b']]
- {k: v for k, v in k_and_v}
- {k+1: v*2 for k, v in k_and_v}

Les fonctions help et dir

- La fonction « dir » : affiche les attributs et les méthodes de la classe ou du type de son argument
  - >>> *dir("chaine")*

['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__','__getattribute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__','__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__','__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__','__str__', '__subclasshook__', '_formatter_field_name_split', '_formatter_parser', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index','isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

- La fonction « help »
  - >>> *help([].append)*

Help on built-in function append:

append(...)

L.append(object) -- append object to end

Fonction utilisateur en python

- *Une fonction est caractérisée par :*
  - *Le mot clef def*
  - *Son nom*
  - *Ses paramètres (0 ou plus), avec ou sans valeur par défaut*
  - *: + indentation*
  - *Sa description (optionnelle mais recommandée)*
  - *Son corps*
  - *Son retour « Par défaut la valeur None »*

```
def  Nom_fonction (ses_paramètres) :
    ⟷ Sa description « DocString »
    ⟷ Son corps
    ⟷ Son retour
```

```
def  func_vide () :
    """
    Exemple d'une fonction vide
    """
    pass

>>>print (func_vide())
    None

>>>help(func_vide)
    Exemple d'une fonction vide
```

```
def  func_Affichage (a,b) :
    """
    Exemple d'une fonction d'affichage
    """
    print(a,b)

>>>print(func_Affichage("Well","Done"))
    Well Done
    None

>>>help(func_Affichage)
    Exemple d'une fonction d'affichage
```

```
def incr (a, n=1) :
    """
    Incremente de n ou 1 par defaut
    """
    return a + n
>>> print(incr(2))
    3

>>> print(incr(5, 3))
    8

>>>help(incr)
    Incremente de n ou 1 par defaut
```

```
def  add_sub (a,b) :
    """
    Calcul somme et difference
    """
    return (a+b, a-b)

>>>print(add_sub(27473,22398))
    (49871, 5075)

>>>print(add_sub(b=4,a=10))
    (14, 6)

>>> help(add_sub)
    Calcul somme et différence
```

La portée des variables

- *À l'intérieur d'une fonction, il est possible d'accéder, en lecture, à une variable définie à l'extérieur*
- *Il n'est pas possible de modifier une variable définie à l'extérieur*
- *Il est possible d'utiliser les variables globales mais pas bien vu comme style de programmation*

def f(y) :
  x = 1
  x += 1
  print (x)
 	
x = 5
f(x)
print(x)
	def g(y) :
	  print(x)
	  print(x + 1)
	
	x = 5
	g(x)
	print(x)
		def h(y) :
		  x += 1
		
		x = 5
		h(x)
		print(x)
			def fonc(y) :
			  global x
			  x += 2
			  z =x + y
			  return z
			
			x=99
			print(fonc(1)) # retourne 102

Le passage des paramètres

- En python, le passage de paramètres se fait par valeur.
- Les noms des variables comportent des références vers les objets.
- de données, donc ce sont ces références qui sont passées aux fonctions et non les objets eux-mêmes.
- Si l'objet passé est immuable, il sera impossible de le modifier même dans la fonction.
- Pour les objets mutables, les modifier directement dans les fonctions impactera les objets initiaux.

Les fonctions statistiques en Python

- La fonction sum
  - sum([1, 3, 2])
  - sum([1, 3, 2], 10)
- La fonction min
  - min([1, 3, 2])
  - min(1, 3, 2)
  - min("abc")
  - def absolute_value(x):
        return abs(x)
    min(-3, 2, key=absolute_value)
- La fonction max
  - max([1, 3, 2])
  - max(1, 3, 2)
  - max("abc")
  - def absolute_value(x):
        return abs(x)
    max(-3, 2, key=absolute_value)