Fonctions génériques

`help(object)`	Fournit de l'aide sur `object`.
`type(object)`	Retourne le type de `object`.
`dir(object)`	Affiche des informations sur `object`.

Opérations sur des objets itérables

`sum(iterable, start=0)`	Retourne la somme du nombre `start` et des termes (numériques) de `iterable`.
`math.fsum(iterable)`	Fonction similaire à `sum` mais spécialisée dans les flottants, améliore la précision du résultat.
`max(iterable, key=None)`	Retourne la plus grande valeur de `iterable` ou `ValueError` s'il est vide.
`max(a, b, c..., key=None)`	Retourne la plus grande valeur parmi `a`, `b`, `c`....
`min(iterable, key=None)`	Retourne la plus petite valeur de `iterable` ou `ValueError` s'il est vide.
`min(a, b, c..., key=None)`	Retourne la plus petite valeur parmi `a`, `b`, `c`....
	Pour chacune des fonctions précédentes, si une fonction `key` est fournie (impérativement par mot-clé), l'interpréteur Python effectue le test `key(a)<key(b)` pour comparer deux valeurs `a` et `b`. Le premier élément maximal (resp. minimal) est retourné en cas d'égalité.
`all(iterable)`	Retourne `True` si tous les éléments de `iterable` sont évalués à `True` et `False` dans le cas contraire (arrêt de l'évaluation au premier `False` rencontré, retourne `True` si l'itérable est vide).
`any(iterable)`	Retourne `True` si au moins un des éléments de `iterable` est évalué à `True` et `False` dans le cas contraire (arrêt de l'évaluation au premier `True` rencontré, retourne `False` si l'itérable est vide).

Fonctions utisables sur entiers, flottants, complexes

`abs(x)`	Retourne la valeur absolue de `x` ou son module dans le cas d'un complexe.
`pow(x, y[, z])`	Retourne `(x**y) % z` si `z` est fourni, et `x**y` dans le cas contraire.

Fonctions spécifiques aux entiers

`math.gcd(a, b)`	Retourne le plus grand diviseur commun de a et b.
`math.factorial(n)`	Retourne $n!$.

Fonctions spécifiques aux flottants

`round(x, ndigits=0)`	Retourne le flottant avec au plus `ndigits` décimales le plus proche de `x`.
`int(x)`	Retourne l'entier correspondant à l'arrondi de `x` vers 0.
`math.ceil(x)`	Retourne $\lceil x \rceil$.
`math.floor(x)`	Retourne $\lfloor x \rfloor$.
`math.trunc(x)`	Retourne la troncature de `x`.
`math.fabs(x)`	Retourne $\lvert x \rvert$.
`math.fmod(x, y)`	Retourne le reste de la division entière de x par y.
`isclose(x, y, *, rel_tol=1e-9, abs_tol=0.0)`	Retourne `True` si $x \simeq y$ (les tolérances relatives et absolues pouvant être précisées). Nécessite Python ≥ 3.5.
`math.exp(x)`	Retourne e^x.
`math.log(x[, base])`	Retourne $\log_{\text{base}}(x)$ si `base` est fournie, et $\ln(x)$ sinon.
`math.log2(x)`	Retourne $\log_2(x)$.
`math.log10(x)`	Retourne $\log_{10}(x)$.
`math.pow(x, y)`	Retourne x^y.
`math.sqrt(x)`	Retourne $\sqrt{x}$.
`math.gamma(x)`	Retourne $\Gamma(x)$.
`math.erf(x)`	Retourne $\text{erf}(x)$.
`math.pi`	Nom désignant la valeur flottante approchée de π.
`math.e`	Nom désignant la valeur flottante approchée de e.
`math.sin(x)` `math.cos(x)` `math.tan(x)`	Fonctions trigonométriques.
`math.asin(x)` `math.acos(x)` `math.atan(x)`	Fonctions trigonométriques inverses.
`math.sinh(x)` `math.cosh(x)` `math.tanh(x)`	Fonctions hyperboliques.
`math.asinh(x)` `math.acosh(x)` `math.atanh(x)`	Fonctions hyperboliques inverses.
`math.degrees(x)`	Convertit en degrés l'angle `x` exprimé en radians.
`math.radians(x)`	Convertit en radians l'angle `x` exprimé en degrés.
`math.hypot(x, y)`	Retourne la norme du vecteur de composantes (x, y), soit $\sqrt{x^2+y^2}$.
`math.atan2(y, x)`	Retourne l'angle, en radians dans $]-\pi, \pi]$, entre le vecteur de composantes (x, y) et $\vec{u}_x$. Attention à l'ordre des paramètres !

Fonctions liées aux listes (type `list`)

Si `lst` est une liste, alors `list.append(lst, elem)` peut également s'écrire `lst.append(elem)`, et il en est de même pour toutes les fonctions ci-dessous commençant par « `list.` ».

`len(lst)`	Retourne le nombre d'éléments de `lst`.
`list.append(lst, elem)` ou `lst += [ elem ]`	Ajoute `elem` à la fin de `lst`.
`list.extend(lst, iterable)` ou `lst += iterable`	Ajoute les éléments de `iterable` à la fin de `lst`.
`list.insert(lst, index, elem)`	Insère `elem` dans `lst` juste avant l'élément à la position `index`.
`list.pop(lst, index=-1)`	Retire et retourne l'élément de `lst` à la position `index` (par défaut, le dernier).
`list.remove(lst, elem)`	Retire de `lst` la première occurrence de `elem` (`ValueError` si absent).
`del lst[index]`	Retire de `lst` l'élément à la position `index`.
`list.clear(lst)`	Retire tous les éléments de `lst`.
`list.index(lst, elem[, start[, stop]])`	Retourne la première occurrence de `elem` dans `lst` (`ValueError` si absent) ; `start` et `stop` restreignent la recherche à une partie de `lst`.
`list.count(lst, elem)`	Retourne le nombre d'occurrence de `elem` dans `lst`.
`list.copy(lst)`	Crée et retourne une nouvelle liste contenant les éléments de `lst`.
`copy.deepcopy(lst)`	Crée et retourne une copie profonde de `lst` (ou d'un autre conteneur).
`list.sort(lst, key=None, reverse=False)`	Tri (en place, stable) de `lst` par valeurs croissantes ; si une fonction `key` est fournie, `a` est considéré plus petit que `b` si `key(a)<key(b)` ; l'ordre est inversé si `reverse=True`.
`random.sample(lst, k)`	Retourne une liste de `k` éléments pris au hasard sans remise dans `lst` (ou dans toute séquence fournie en premier argument).
`random.shuffle(lst)`	Mélange (en place) les éléments de `lst`.
`list.reverse(lst)`	Inverse l'ordre des éléments dans `lst` (en place).
`list(iterable)`	Crée une liste à partir des éléments de l'itérable passé en argument.

Génération de valeurs pseudo-aléatoires

`random.seed(a=None)`	Initialise le générateur pseudo-aléatoire avec l'élément fourni (permet de faire de l'aléatoire reproductible pour les tests). En l'absence d'argument, l'initialisation se fait à partir d'une source d'entropie du système.
`random.randint(a, b)`	Retourne un entier aléatoire dans l'intervale $[a, b]$ où `a` et `b` sont entiers.
`random.randrange(a, b)`	Retourne un entier aléatoire dans l'intervale $[a, b[$ où `a` et `b` sont entiers.
`random.random()`	Retourne un flottant choisi aléatoirement dans $[0, 1[$.
`random.uniform(a, b)`	Retourne un flottant choisi aléatoirement dans l'intervale $[a, b]$.
`random.gauss(mu, sigma)`	Retourne un flottant obtenu aléatoirement dans une distribution gaussienne de moyenne `mu` et d'écart-type `sigma`.
`random.lognormvariate(mu, sigma)`	Retourne un flottant obtenu aléatoirement dans une distribution log-normale de moyenne `mu` et d'écart-type `sigma`.
`random.expovariate(lambd)`	Retourne un flottant obtenu aléatoirement dans une distribution exponentielle de paramètre `lambd`.

Conversions valeurs numériques/chaînes de caractères

`bin(n)`	Retourne une chaîne contenant la représentation binaire de l'entier `n`.
`hex(n)`	Retourne une chaîne contenant la représentation hexadécimale de l'entier `n`.
`oct(n)`	Retourne une chaîne contenant la représentation octale de l'entier `n`.
`int(ch, base=10)`	Interprète la chaîne `ch` comme une représentation d'un entier dans la `base` spécifiée et retourne cet entier.
`float(ch)`	Interprète la chaîne `ch` comme un nombre flottant et retourne le résultat.
`chr(i)`	Retourne le caractère dont l'index en Unicode est `i`.
`ord(c)`	Retourne l'index Unicode de la chaîne de caractères (réduite à un seul caractère) `c`.

Entrées et sorties

`input(prompt)`	Affiche la chaîne `prompt`, puis lit une entrée de l'utilisateur au clavier et la retourne sous forme de chaîne.
`print(elem1, elem2, ..., end='\n', sep=' ')`	Affiche les objets `elem1`, `elem2`, ... fournis en paramètres, en insérant la chaîne `sep` entre chaque éléments et en terminant par la chaîne `end`.

Opérations sur les chaînes de caractères (type `str`)

Si `ch` est une chaîne de caractères, alors `str.split(ch, sep)` peut aussi s'écrire `ch.split(sep)`, et il en est de même pour toutes les fonctions ci-dessous commençant par « `str.` ».

`len(ch)`	Retourne le nombre de caractères constituant la chaîne de caractères `ch`.
`str.split(ch)`	Retourne une liste de chaînes de caractères obtenue en découpant la chaîne de caractères `ch` à chaque espacement.
`str.split(ch, sep)`	Même chose que précédemment, mais en utilisant la chaîne `sep` comme séparateur.
`str.splitlines(ch)`	Retourne une liste de chaînes de caractères correspondant aux différentes « lignes » (privées des caractères marquant les fins de ligne) présentes dans la chaîne de caractères `ch`.
`str.join(ch, iterable)`	Retourne une chaîne de caractère correspondant à la concaténation des éléments de `iterable` (lesquels doivent être des chaînes de caractères), entre lesquels on a intercalé la chaîne de caractère `ch`.
`str.lstrip(ch, ch2=None)`	Retourne une chaîne correspondant à `ch` privée des caractères situés en tête de `ch` présents dans `ch2`. Par défaut, les caractères retirés sont les espacements, tabulations et sauts de ligne.
`str.rstrip(ch, ch2=None)`	Même chose que `lstrip`, mais pour les caractères situés à la fin de `ch`.
`str.strip(ch, ch2=None)`	Même chose que `lstrip` et `rstrip` pour les caractères situés aux deux extrémités de `ch`.
`str.count(ch, ch2, start=None, end=None)`	Retourne le nombre d'occurrences (sans recouvrement) de `ch2` dans `ch`.
`str.find(ch, ch2, start=None, end=None)`	Retourne l'indice à partir duquel on peut trouver la chaîne `ch2` à l'intérieur de `ch` (et -1 si `ch2` n'est pas une sous-chaîne de `ch`).
`str.index(ch, ch2, start=None, end=None)`	Même chose que la fonction précédente, mais renvoie une erreur si `ch2` n'est pas présente dans `string`.
`str.startwith(ch, ch2, start=None, end=None)`	Retourne `True` si le début de `ch` est identique à `ch2`.
`str.endwith(ch, ch2, start=None, end=None)`	Retourne `True` si la fin de `ch` est identique à `ch2`. Pour toutes ces fonctions, il est possible de préciser deux indices `start` et `end` pour réduire la chaîne `string` à la sous-chaîne comprise entre ces deux indices. Il existe des fonctions `rfind` et `rindex` qui fonctionnent comme `find` et `index` mais effectuent la recherche en partant de la fin.

Opérations sur les chaînes de caractères (suite)

`str.isalnum(ch)`	Retourne `True` si la chaîne `ch` n'est pas vide et ne contient que des caractères alphanumériques, et `False` sinon.
`str.isalpha(ch)`	Retourne `True` si la chaîne `ch` n'est pas vide et ne contient que des « lettres », et `False` sinon.
`str.isdecimal(ch)`	Retourne `True` si la chaîne `ch` n'est pas vide et ne contient que des caractères « chiffres », et `False` sinon.
`str.isspace(ch)`	Retourne `True` si la chaîne `ch` n'est pas vide et ne contient que des caractères de type « espacements » (espaces, tabulations, etc.), et `False` sinon.
`str.islower(ch)`	Retourne `True` si la chaîne `ch` n'est pas vide et ne contient que des caractères minuscules, et `False` sinon.
`str.isupper(ch)`	Retourne `True` si la chaîne `ch` n'est pas vide et ne contient que des caractères majuscules, et `False` sinon. Note : la classification des caractères est basée sur celle d'Unicode. Il existe d'autres fonctions similaires non citées ici.
`str.upper(ch)`	Retourne une chaîne de caractère correspondant à `ch` dans laquelle les minuscules ont été transformées en majuscules.
`str.lower(ch)`	Identique à `upper`, mais en transformant les majuscules en minuscules.
`str.ljust(ch, width, fillchar=' ')`	Crée une chaîne de caractères de longueur égale à `width` en ajoutant à `ch` le caractère `fillchar` en fin de chaîne.
`str.rjust(ch, width, fillchar=' ')`	Crée une chaîne de caractères de longueur égale à `width` en ajoutant à `ch` le caractère `fillchar` au début de la chaîne.
`str.center(ch, width, fillchar=' ')`	Crée une chaîne de caractères de longueur égale à `width` en ajoutant à `ch` le caractère `fillchar` des deux côtés. Si la chaîne `ch` est plus longue que `width`, ces fonctions renvoient simplement la chaîne `ch`
`str.format(ch, ...)`	Retourne une chaîne de caractères similaire à `ch`, mais dans laquelle les « jetons » (identifiés par des accolades) ont été remplacés par les autres arguments de la fonction, après avoir été mis en forme.
`eval(ch)`	Analyse le contenu de la chaîne de caractères `ch` et retourne l'objet Python correspondant (entier, flottant, liste...).
`string.ascii_lowercase`	Chaîne contenant l'ensemble des minuscules non accentuées.
`string.ascii_uppercase`	Chaîne contenant l'ensemble des majuscules non accentuées.
`string.digits`	Chaîne contenant l'ensemble des chiffres.

Opérations sur le système de fichier

`os.getcwd()`	Retourne le répertoire courant.
`os.chdir(path)`	Adopte `path` comme répertoire courant.
`os.listdir(path='.')`	Retourne la liste des fichiers et sous-répertoires du répertoire courant.
`os.rename(src, dst)`	Renomme le fichier nommé `src` en `dst`.
`os.mkdir(path)`	Crée le sous-répertoire `path`.
`os.rmdir(path)`	Supprime le sous-répertoire `path`.

Opérations sur les fichiers

`open(nom_fichier, mode='r')`	Ouvre le fichier dans le répertoire courant appelé `nom_fichier` et retourne un objet représentant le fichier. Par défaut, l'ouverture se fait en mode *lecture* (`'r'`). D'autres modes couramment utilisés sont *écriture* (`'w'`) et *ajout* (`'a'`). Dans ces deux derniers cas, le fichier est créé s'il n'existe pas.
`fich.close()`	Referme le fichier désigné par le nom `fich`.
`fich.read()`	Extrait et retourne l'intégralité du fichier désigné par le nom `fich`, à partir de la position courante, sous la forme d'une chaîne de caractères.
`fich.read(size)`	Extrait et retourne `size` caractères du fichier désigné par le nom `fich`, à partir de la position courante, sous la forme d'une chaîne de caractères (ou moins si l'on atteint la fin du fichier).
`fich.readline()`	Extrait à partir de la position courante et retourne une ligne du fichier désigné par le nom `fich`.
`fich.readlines()`	Extrait à partir de la position courante et retourne une liste de lignes dans le fichier désigné par le nom `fich`.
`fich.write(ch)`	Écrit à partir de la position courante la chaîne de caractères `ch` dans le fichier désigné par le nom `fich`.
`fich.tell()`	Retourne la position courante (mesurée en octets à partir du début) dans le fichier désigné par le nom `fich`.
`fich.seek(offset, from_what)`	Déplace la position courante à l'octet `offset` compté à partir du début (si `from_what` est omis ou égal à 0), de la position courante (si `from_what` est égal à 1) ou de la fin (si `from_what` est égal à 2).

Principales structures de contrôle

```
def nom_fonction(arguments) :
    instructions
    return résultat

for nom in itérable :
    instructions

while expression_booléenne :
    instructions    # Exécuté en boucle
                    # tant que expression_booléenne est vraie

if expr_bool_1 :
    instructions    # Si expr_bool_1 est vraie
elif expr_bool_2 :
    instructions    # Si expr_bool_1 est fausse et expr_bool_2 est vraie
else :
    instructions    # Si expr_bool_1 et expr_bool_2 sont fausses
```

Principaux objets itérables

objet « `range` »	séquence	entiers du « range »
liste (type `list`)	séquence	éléments de la liste
chaîne de caractères (type `str`)	séquence	caractères constituant la chaîne
tableau `numpy` 1D	séquence	éléments du tableau
tableau `numpy` 2D	séquence	lignes du tableau
tuple	séquence	éléments du tuple
fichier (texte)	itérable	lignes du fichier
dictionnaire (type `dict`)	itérable	clés du dictionnaire

Les itérables et les séquences peuvent être utilisés avec « `for ... in` » (boucles, compréhensions de listes). Les séquences sont des itérables qui permettent en plus d'accéder aux éléments via un indice, et peuvent faire l'objet de « saucissonage » (*slicing*).

Fonctions retournant des itérables

`enumerate(['B', 'A', 'C'])`	`(0, 'B'), (1, 'A'), (2, 'C')`
`reversed(['B', 'A', 'C'])`	`'C', 'A', 'B'`
`sorted(['B', 'A', 'C'])`	`'A', 'B', 'C'`
`zip(['B', 'A', 'C'], ['x', 'y', 'z'])`	`('B', 'x'), ('A', 'y'), ('C', 'z')`