Résumé des fonctions standards de python

Une petite note pour aider les dev python à avoir en mains un résumé de leurs fonctions les plus utilisées

📌 FONCTIONS BUILT-IN (disponibles sans import)

Ce sont celles que tu peux utiliser directement :

Fonction	Description
`print()`	Affiche un texte
`len()`	Longueur d'un objet
`type()`	Type d'un objet
`int()`, `float()`, `str()`	Conversion de types
`range()`	Génère une séquence de nombres
`input()`	Lit une entrée utilisateur
`sum()`	Somme d'une liste

`max()`, `min()`	**Valeur max / min**
`abs()`	**Valeur absolue**
`sorted()`	**Trie une séquence**
`reversed()`	**Inverse une séquence**
`enumerate()`	**Donne index + valeur dans une boucle**
`zip()`	**Regroupe plusieurs listes**
`map()`, `filter()`, `reduce()`	**Programmation fonctionnelle**
`any()`, `all()`	**Teste des booléens dans un itérable**
`dir()`	**Liste les attributs et méthodes d'un objet**
`help()`	**Affiche l'aide intégrée**
`eval()`	**Exécute une expression Python**

🧰 `math` — Mathématiques

```
import math
```

Fonction	Description
`math.sqrt(x)`	Racine carrée
`math.pow(x, y)`	Puissance
`math.floor(x)`	Arrondi vers le bas
`math.ceil(x)`	Arrondi vers le haut
`math.pi`, `math.e`	Constantes célèbres
`math.sin()`, `math.cos()`	Trigonométrie

Fonction	Description	Exemple	Résultat
`abs(x)`	Valeur absolue	`abs(-5)`	`5`
`round(x)`	Arrondir	`round(3.7)`	`4`
`max(a, b)`	Plus grand	`max(3, 8)`	`8`
`min(a, b)`	Plus petit	`min(3, 8)`	`3`
`pow(x, y)`	Puissance	`pow(2, 3)`	`8`

`sum([a, b, ...])`	**Somme d'une liste**	`sum([1, 2, 3])`	`6`
`math.sqrt(x)`	**Racine carrée**	`math.sqrt(16)`	`4.0`
`math.factorial(x)`	**Factorielle**	`math.factorial(5)`	`120`
`math.ceil(x)`	**Arrondir vers le haut**	`math.ceil(4.2)`	`5`
`math.floor(x)`	**Arrondir vers le bas**	`math.floor(4.8)`	`4`
`math.log(x, base)`	**Logarithme**	`math.log(8, 2)`	`3.0`
`math.exp(x)`	**Exponentielle (e^x)**	`math.exp(1)`	`2.718...`
`math.sin(x)`	**Sinus (en radians)**	`math.sin(math.pi / 2)`	`1.0`
`math.cos(x)`	**Cosinus**	`math.cos(0)`	`1.0`
`math.tan(x)`	**Tangente**	`math.tan(math.pi / 4)`	`1.0`
`math.pi`	**Constante π**	`math.pi`	`3.1415...`
`math.e`	**Constante e**	`math.e`	`2.718...`

🎲 `random` — Aléatoire

```
import random
```

Fonction	Description
`random.random()`	Float entre 0 et 1
`random.randint(a, b)`	Entier aléatoire entre a et b
`random.choice(liste)`	Choix aléatoire dans une séquence
`random.shuffle(liste)`	Mélange une liste

📦 `functools` — Fonctions avancées

```
from functools import cache, lru_cache, reduce
```

Fonction	Description
`@cache`	Mémoïsation automatique (depuis Python 3.9)

`@lru_cache`	**Version + ancienne, avec limite de taille**
`reduce(f, iterable)`	**Applique une fonction cumulée**

📐 `itertools` — Itérateurs puissants

```
import itertools
```

Fonction	**Description**
`itertools.count()`	**Compteur infini**
`itertools.product()`	**Produit cartésien (utile pour QCM, combinaisons)**
`itertools.permutations()`	**Permutations d'une liste**
`itertools.combinations()`	**Combinaisons d'une liste**

📊 `statistics` — Statistiques de base

python

CopyEdit

```
import statistics
```

Fonction	Description
`mean()`	Moyenne
`median()`	Médiane
`stdev()`	Écart type

🧱 heapq — File de priorité (tas)

```
import heapq
```

Fonction	Description
`heapify()`	Transforme une liste en min-heap
`heappush()`	Ajoute un élément

`heappop()`	Retire le plus petit élément

📜 `collections` — Structures utiles

```
from collections import Counter, defaultdict, deque, namedtuple
```

Élément	Description
`Counter()`	Compte les occurrences
`defaultdict()`	Dictionnaire avec valeur par défaut
`deque()`	File double
`namedtuple()`	Tuple nommé

🔧 `os` — Système d’exploitation

```
import os
```

Fonction	Description

`os.getcwd()`	Dossier courant
`os.listdir()`	Liste les fichiers d'un dossier
`os.path.join()`	Joint des chemins

📁 `pathlib` — Manipulation de fichiers (moderne)

```
from pathlib import Path
```

Fonction / classe	Description
`Path()`	Représente un chemin
`.exists()`	Vérifie l'existence
`.read_text()`	Lire un fichier texte
`.write_text()`	Écrire dans un fichier

🕑 `datetime` — Dates et heures

```
import datetime
```

Fonction / classe	Description

`datetime.now()`	**Date et heure actuelles**
`timedelta()`	**Durée entre deux dates**
`date()`	**Créer une date**

Segment Tree

```python
class segmentTree:

    def __init__(self, arr):

        self.n = len(arr)

        self.Tree = [0] * (4*self.n)

        self.build(arr, 0, 0, self.n-1)

    def build(self, arr, node, start, end):

        #Si l'interval contient un seul element

        if start == end:

            self.Tree[node] = arr[start]

        else:

            mid = (start+end)//2

            left_node = node*2+1

            right_node = node*2+2

            self.build(arr, left_node, start, mid)

            self.build(arr, right_node, mid+1, end)

            self.Tree[node] = self.Tree[left_node] + self.Tree[right_node]
```

```
    def update(self, idx, val, node, start, end):
        if start == end:
            self.Tree[node]+=val
        else:
            mid = (start + end)//2
            left_node = node*2+1
            right_node = node*2 + 2
            if start<=idx<=mid:
                self.update(idx, val, left_node, start, mid)
            else:
                self.update(idx, val, right_node, mid+1, end)
            self.Tree[node] = self.Tree[left_node] + self.Tree[right_node]
    def query(self, L, R, node, start, end):
        if start>R or L>end:
            return 0
        if L<=start and R>=end:
            return self.Tree[node]
        mid = (start + end)//2
        left_node = node * 2 + 1
        right_node = node*2 + 2
        left_query = self.query(L, R, left_node, start, mid)
        rigth_query = self.query(L, R, right_node, mid+1, end)
        return left_query + rigth_query
```

Tries

```
class TrieNode():

    def __init__(self):

        self.children = {}

        self.is_word = False


class Trie():

    def __init__(self):

        self.root = TrieNode()

    def insert(self, word):

        node = self.root

        for c in word:

            if c not in node.children:

                node.children[c] = TrieNode()

            node = node.children[c]

        node.is_word = True

    def query(self, word):

        node = self.root

        for c in word:

            if c not in node.children:

                return False

            node = node.children[c]

        return node.is_word
```

```
    def ispref(self, word):
        node = self.root
        for c in word:
            if c not in node.children:
                return False
            node = node.children[c]
        return True


if __name__ == '__main__':
    node = Trie()
    node.insert("abab")
    node.insert("abcd")
    node.insert("fofo")
    node.insert("papara")
    print(node.query("papa"))
    print(node.ispref("abc"))
    print(node.query("aba"))
```

Disjoint Union

```
class DisjointUnion:
    def __init__(self, n):
        self.parent = [i for i in range(n)]
```

```
        self.rang = [1] * n


    def find(self, x):
        if self.parent[x]!=x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]
    def union(self, x, y):
        rootX = self.parent[x]
        rootY = self.parent[y]
        if self.rang[rootX]>self.rang[rootY]:
            self.parent[rootY] = rootX
        elif self.rang[rootX]<self.rang[rootY]:
            self.parent[rootX] = rootY
        else:
            self.parent[rootX] = rootY
            self.rang[rootY]+=1
    def areConnected(self, x, y):
        return self.find(x) == self.find(y)
#[[0,2,7],[0,1,15],[1,2,6],[1,2,1]]55[[0,1,7],[1,3,7],[1,2,1]]
if __name__ == '__main__':
    du = DisjointUnion(5)
    du.union(1,3)
    du.union(0,1)
    du.union(1,2)
```

```
print(du.areConnected(3,4))
```