Obtenir de l'aide avec `help()`

In [1]:
```
# La fonction help() permet d'obtenir des informations sur une fonction ou un module
help(round)
```

```
Help on built-in function round in module builtins:

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwise
    the return value has the same type as the number.  ndigits may be negative.
```

Définir des fonctions

In [2]:
```
# Définir une fonction qui retourne la plus petite différence entre trois nombres
def least_difference(a, b, c):
    """Retourne la plus petite différence entre deux nombres parmi a, b et c."""
    return min(abs(a - b), abs(b - c), abs(a - c))

# Obtenir de l'aide sur la fonction définie
help(least_difference)
```

```
Help on function least_difference in module __main__:

least_difference(a, b, c)
    Retourne la plus petite différence entre deux nombres parmi a, b et c.
```

Valeurs par défaut et fonctions d'ordre supérieur

In [3]:
```python
# Une fonction avec un argument par défaut
def greet(who="Colin"):
    print("Bonjour,", who)

# Appels avec et sans argument
greet()
greet("Alice")
```

```
Bonjour, Colin
Bonjour, Alice
```

In [4]:
```python
# Exemple de fonction d'ordre supérieur
def call(fn, arg):
    """Appelle la fonction fn avec l'argument arg"""
    return fn(arg)

# Fonction simple qui multiplie un nombre par 5
def mult_by_five(x):
    return x * 5

# Appel de la fonction d'ordre supérieur
call(mult_by_five, 3)
```

Out[4]: 15

Opérations booléennes et conditionnelles

In [5]:
```python
# Fonction qui vérifie si un nombre est impair
def is_odd(n):
    return n % 2 == 1

# Vérification de deux exemples
is_odd(10)
is_odd(11)
```

Out[5]: True

In [6]:
```python
# Fonction avec des instructions conditionnelles pour inspecter un nombre
def inspect(x):
    if x == 0:
        print("zéro")
    elif x > 0:
        print("positif")
    else:
        print("négatif")

# Inspection de deux exemples
inspect(0)
inspect(15)
```

```
zéro
positif
```

Listes

In [7]:
```python
# Une liste de planètes
planets = ['Mercure', 'Vénus', 'Terre']
# Accéder aux éléments d'une liste par leur index
print(planets[0])  # Mercure
print(planets[-1])  # Terre
```

```
Mercure
Terre
```

In [8]:
```python
# Utilisation d'une liste en compréhension pour générer les carrés de 0 à 9
squares = [n**2 for n in range(10)]
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Chaînes de caractères

In [9]:
```python
# Utilisation des méthodes de chaînes de caractères
claim = "Pluton est une planète!"
print(claim.upper())  # PLUTON EST UNE PLANÈTE!
```

```
PLUTON EST UNE PLANÈTE!
```

In [10]:
```python
# Formatage de chaînes de caractères
planet = "Pluton"
position = 9
print("{}, tu seras toujours la {}ème planète.".format(planet, position))
```

```
Pluton, tu seras toujours la 9ème planète.
```

Dictionnaires

In [11]:
```python
# Dictionnaire simple associant des nombres à des chaînes
numbers = {'un': 1, 'deux': 2}
print(numbers['un'])  # 1
```

```
1
```

In [12]:
```python
# Parcourir un dictionnaire avec ses clés et valeurs
for key, value in numbers.items():
    print(f"{key} = {value}")
```

```
un = 1
deux = 2
```

Boucles

In [13]:
```python
# Exemple de boucle for sur une liste
planets = ['Mercure', 'Vénus', 'Terre']
for planet in planets:
    print(planet)
```

```
Mercure
Vénus
Terre
```

In [14]:
```python
# Exemple de boucle while
i = 0
while i < 3:
    print(i)
    i += 1
```

```
0
1
2
```

Importation de modules

In [15]:
```python
# Importation du module math et utilisation de ses fonctions
import math
print(math.pi)  # 3.14159
```

```
3.141592653589793
```

In [16]:
```python
# Utilisation d'un alias pour le module
import math as mt
print(mt.pi)
```

```
3.141592653589793
```

Gestion des fichiers

In [17]:
```python
# Exemple d'écriture dans un fichier
with open('example.txt', 'w') as file:
    file.write("Bonjour, monde !")
```

Ensembles (sets)

In [18]:
```python
# Exemple d'ensemble avec des éléments uniques
fruits = {"pomme", "orange", "banane"}
fruits.add("cerise")  # Ajoute "cerise" à l'ensemble
print(fruits)
```

```
{'banane', 'orange', 'cerise', 'pomme'}
```

Exceptions

In [19]:
```python
# Gestion des erreurs avec try et except
try:
    print(1 / 0)
except ZeroDivisionError:
    print("Erreur : division par zéro !")
```

```
Erreur : division par zéro !
```

Les fonctions lambda

In [20]:
```python
# Exemple d'utilisation d'une fonction lambda pour doubler un nombre
double = lambda x: x * 2
print(double(5))
```

```
10
```

Générateurs et `range()`

In [21]:
```python
# Utilisation de range() dans une boucle for
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Objets et méthodes

In [22]:
```python
# Exemple d'utilisation de méthodes sur des objets
x = 12
print(x.bit_length())  # 4
```

```
4
```

In [23]:
```python
# Méthodes de liste
planets = ['Mercure', 'Vénus', 'Terre']
planets.append('Mars')  # Ajoute Mars
print(planets)
planets.pop()  # Supprime Mars
print(planets)
```

```
['Mercure', 'Vénus', 'Terre', 'Mars']
['Mercure', 'Vénus', 'Terre']
```

Tuples

In [24]:
```python
# Exemple de tuple pour stocker plusieurs valeurs de retour
numerator, denominator = 0.125.as_integer_ratio()
print(numerator, denominator)
```

```
1 8
```

Décorateurs

In [25]:
```python
# Exemple de décorateur pour ajouter un comportement à une fonction
def debug(func):
    def wrapper(*args, **kwargs):
        print(f"Appel de {func.__name__}")
        return func(*args, **kwargs)
    return wrapper

# Décorateur utilisé pour la fonction add
@debug
def add(a, b):
    return a + b

print(add(3, 5))
```

```
Appel de add
8
```