😊 Args and kwargs in Python

***args and **kwargs in Python**

1. ***args (Non-Keyword Arguments)**
    - Allows a function to accept any number of positional arguments.
    - The arguments are accessible as a **tuple** within the function.
    - Useful when you don't know how many inputs you'll receive.
2. ****kwargs (Keyword Arguments)**
    - Allows a function to accept any number of keyword arguments.
    - The arguments are accessible as a **dictionary** within the function.
    - Useful for named parameters when you don't know all possible keys in advance.

Why Use Them?

- **Flexibility**: They enable writing functions that handle varied and unknown numbers of inputs.
- **Clean Code**: Reduces the need for multiple function overloads or complicated parameter handling.
- **Scalability**: Makes functions adaptable for future changes without breaking existing code.

😄 To begin with the Lab

1. This function takes two numbers, adds them together, and then calculates 5% of that sum. For example, when you call it with 40 and 60, the sum is 100, and 5% of 100 is 5.

```
[1]: def myfunc(a,b):
         return sum((a,b))*.05

     myfunc(40,60)
```

```
[1]: 5.0
```

2. This function accepts up to five numbers, defaulting missing ones to 0. It sums the numbers and then calculates 5% of that total. When you call it with 40, 60, and 20, it computes (40 + 60 + 20) = 120, and 5% of 120 is 6.
3. Obviously, this is not a very efficient solution, and that's where ***args** comes in.

[2]:
```
def myfunc(a=0,b=0,c=0,d=0,e=0):
    return sum((a,b,c,d,e))*.05

myfunc(40,60,20)
```

[2]: 6.0

4. When a function parameter starts with an asterisk, it allows for an *arbitrary number* of arguments, and the function takes them in as a tuple of values. Rewriting the above function:
5. This function accepts any number of numbers, adds them together, and then calculates 5% of the total. For the inputs 40, 60, and 20, the sum is 120, and 5% of 120 is 6.

[3]:
```
def myfunc(*args):
    return sum(args)*.05

myfunc(40,60,20)
```

[3]: 6.0

6. This function uses *spam to gather all arguments into a tuple. It then sums these numbers and calculates 5% of the total. For inputs 40, 60, and 20, the sum is 120, and 5% of 120 is 6.

[4]:
```
def myfunc(*spam):
    return sum(spam)*.05

myfunc(40,60,20)
```

[4]: 6.0

7. Similarly, Python offers a way to handle arbitrary numbers of *keyworded* arguments. Instead of creating a tuple of values, **kwargs builds a dictionary of key/value pairs. For example:
8. This function accepts any keyword arguments as a dictionary. It checks if there is a key called 'fruit'. If so, it prints "My favorite fruit is pineapple". If not, it prints "I don't like fruit".

[5]:
```
def myfunc(**kwargs):
    if 'fruit' in kwargs:
        print(f"My favorite fruit is {kwargs['fruit']}")  # review String Formatting and f-strings if this syntax is unfamiliar
    else:
        print("I don't like fruit")

myfunc(fruit='pineapple')
```

```
My favorite fruit is pineapple
```

[6]:
```
myfunc()
```

```
I don't like fruit
```

9. Here we are combining both *args and **kwargs. This function accepts any positional and keyword arguments. It checks for 'fruit' and 'juice' keys, then prints a message joining the positional arguments, displaying the favorite fruit, and asking for juice. In our call, it prints that you like eggs and spam, your favorite fruit is cherries, and asks for orange juice.
10. In Python, you must list all positional arguments before any keyword arguments. The call you provided puts keyword arguments before positional ones, which causes a syntax error.

[7]:
```
def myfunc(*args, **kwargs):
    if 'fruit' and 'juice' in kwargs:
        print(f"I like {' and '.join(args)} and my favorite fruit is {kwargs['fruit']}")
        print(f"May I have some {kwargs['juice']} juice?")
    else:
        pass

myfunc('eggs','spam',fruit='cherries',juice='orange')
```

```
I like eggs and spam and my favorite fruit is cherries
May I have some orange juice?
```

Placing keyworded arguments ahead of positional arguments raises an exception:

[8]:
```
myfunc(fruit='cherries',juice='orange','eggs','spam')
```

```
  File "<ipython-input-8-fc6ff65addcc>", line 1
    myfunc(fruit='cherries',juice='orange','eggs','spam')
                                           ^
SyntaxError: positional argument follows keyword argument
```