Key Points About `*args` and `**kwargs` in Python

- **`*args` (Non-Keyword Arguments)**:
  - Allows a function to accept a variable number of **positional arguments**.
  - These arguments are accessible as a tuple within the function.
- **`**kwargs` (Keyword Arguments)**:
  - Allows a function to accept a variable number of **keyword arguments**.
  - These arguments are accessible as a dictionary within the function, where keys are argument names and values are their corresponding values.

Using `*args`:

1. **Definition**:

```
def sum_of(*args):
    total = 0
    for x in args:
        total += x
    return total
```

2. **Example Usage**:

```
print(sum_of(4, 5))         # Output: 9
print(sum_of(4, 5, 6))      # Output: 15
print(sum_of(1, 2, 3, 4))   # Output: 10
```

Using `**kwargs`:

1. **Definition**:

```
def total_bill(**kwargs):
    total = 0
    for key, value in kwargs.items():
        total += value
    return round(total, 2)
```

2. **Example Usage**:

```
print(total_bill(coffee=2.99, cake=4.55, juice=2.99))  # Output: 10.53
print(total_bill(sandwich=5.99, salad=7.49))          # Output: 13.48
```

Combined Use:

You can use `*args` and `**kwargs` together in the same function:

```
def display_order(*args, **kwargs):
    print("Positional Arguments (args):", args)
    print("Keyword Arguments (kwargs):", kwargs)
```

Example Usage:

```
display_order(1, 2, 3, coffee=2.99, cake=4.55)
# Output:
# Positional Arguments (args): (1, 2, 3)
# Keyword Arguments (kwargs): {'coffee': 2.99, 'cake': 4.55}
```

Key Differences:

Feature	*args	**kwargs
Accepts	Positional arguments	Keyword arguments
Access Type	Tuple	Dictionary
Use Case	Variable-length positional parameters	Variable-length named parameters

With these tools, you can write more flexible and dynamic functions.