Functions

Positional and Keyword Arguments

Positional Arguments:

Positional arguments are the most common type of arguments in Python. They are passed to a function in the same order as they are defined in the function's parameter list.

```python
def greet(name, age):
    print(f"Hello, {name}! You are {age} years old.")


# Calling the function with positional arguments
greet("Alice", 30)
```

Keyword Arguments:

Keyword arguments are passed to a function with a specific keyword identifier. They do not rely on the order of parameters defined in the function.

```python
def greet(name, age):
    print(f"Hello, {name}! You are {age} years old.")

# Calling the function with keyword arguments
greet(age=25, name="Bob")
```

Default Values:

You can also provide default values for function parameters, which allows the function to be called with fewer arguments.

```python
def greet(name="Anonymous", age=18):
    print(f"Hello, {name}! You are {age} years old.")


# Calling the function with default values
greet()
```

Mixing Positional and Keyword Arguments:

You can mix positional and keyword arguments in a function call, but positional arguments must come before keyword arguments.

```python
def greet(name, age):
    print(f"Hello, {name}! You are {age} years old.")


# Mixing positional and keyword arguments
greet("Charlie", age=35)
```

Understanding these concepts will help you effectively pass arguments to functions in Python, providing flexibility and clarity in your code.

* args

In Python, when $*$ is used as a prefix for a parameter in a function definition, it indicates that the parameter is a variable-length argument list, often referred to as "arbitrary positional arguments" or "varargs". This means that the function can accept any number of positional arguments, and they will be packed into a tuple.

```python
def my_function(*args):
    print(args)


my_function(1, 2, 3, 4)
```

In this example, `*args` is used to collect all the positional arguments passed to `my_function()` into a tuple named `args`. When you call `my_function(1, 2, 3, 4)`, the output will be `(1, 2, 3, 4)`.

You can also combine `*args` with other regular parameters:

```python
def my_function(a, b, *args):
    print("Regular parameters:", a, b)
    print("Extra positional arguments:", args)


my_function(1, 2, 3, 4, 5)
```

Here, `a` and `b` are regular parameters, while `*args` collects any additional positional arguments into a tuple.

This feature is particularly useful when you want to create functions that can handle a variable number of arguments, providing flexibility in your code.

Putting * as the first argument force manual typing of the argument when called

```python
def place_order(* ,item, price, quantity)
    print(f" {quantity} unitys of {item} at {price} price")


def what_could_go_wrong():
    place_order(item="SPX", price=4500, quantity=10000)
```