# Variable Number of Arguments Scope

In Python, you can use `*args` and `**kwargs` to define functions that accept a variable number of arguments. These constructs allow you to create functions that can take an arbitrary number of positional and keyword arguments, respectively. Let's explore their usage and scope with examples:

### Variable Number of Positional Arguments (`*args`):

```python
def print_args(*args):
    for arg in args:
        print(arg)

# Calling the function with different numbers of arguments
print_args(1, 2, 3)        # Output: 1 2 3
print_args('a', 'b')       # Output: a b
print_args(10, 'hello', 3.14, [1, 2, 3])  # Output: 10 hello 3.14 [1, 2, 3]
```

In this example, `*args` collects any number of positional arguments into a tuple named `args`. The function then prints each argument. You can call this function with any number of arguments, and they will be handled as elements of the `args` tuple.

### Variable Number of Keyword Arguments (`**kwargs`):

```python
def print_kwargs(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

# Calling the function with different keyword arguments
print_kwargs(name='Alice', age=30)  # Output: name: Alice, age: 30
print_kwargs(city='Wonderland', job='Engineer')  # Output: city: Wonderland, job: Engineer
```

Here, `**kwargs` collects any number of keyword arguments into a dictionary named `kwargs`. The function then iterates over the dictionary and prints each key-value pair. You can call this function with any number of keyword arguments.

### Combining `*args` and `**kwargs`:

```python
def print_args_and_kwargs(*args, **kwargs):
    for arg in args:
        print(arg)
    for key, value in kwargs.items():
        print(f"{key}: {value}")

# Calling the function with a mix of positional and keyword arguments
print_args_and_kwargs(1, 'hello', name='Alice', age=25)
# Output:
# 1
# hello
# name: Alice

# age: 25
```

In this example, the function `print_args_and_kwargs` accepts both positional arguments (`*args`) and keyword arguments (`**kwargs`). It prints each positional argument and each key-value pair in the keyword arguments.

### Scope of `*args` and `**kwargs`:

The names `args` and `kwargs` are just conventions. You can use any valid variable names, but the usage of `args` and `kwargs` is a common convention for clarity.

```python
def example_function(*custom_args, **custom_kwargs):

    for arg in custom_args:

        print(arg)

    for key, value in custom_kwargs.items():

        print(f"{key}: {value}")


# Calling the function with custom argument names

example_function(1, 'custom', name='Bob', age=30)
```

In summary, `*args` and `**kwargs` provide a powerful way to create flexible functions that can accept a variable number of arguments. Understanding their usage allows you to design functions that accommodate diverse input scenarios.