Functions

1. Defining Functions

Functions are reusable blocks of code that perform specific tasks.

```
def greet(name):
    """Simple function with one parameter"""
    return f"Hello, {name}!"

# Call the function
message = greet("Alice")
print(message)

message = greet("Bob")
print(message)
```

```
Hello, Alice!
Hello, Bob!
```

2. Parameters and Default Values

Functions can have multiple parameters and default values.

```
def calculate_area(length, width=1):
    """Calculate rectangle area with default width"""
    return length * width

# Using both parameters
print(f"Area(5, 3): {calculate_area(5, 3)}")

# Using default width
print(f"Area(5): {calculate_area(5)}")

# Named arguments
print(f"Area(width=4, length=6): {calculate_area(width=4, length=6)}")
```

```
Area(5, 3): 15
Area(5): 5
Area(width=4, length=6): 24
```

3. Multiple Return Values

Functions can return multiple values as a tuple.

```
def get_stats(numbers):
    """Return min, max, and average"""
```

```
    min_val = min(numbers)
    max_val = max(numbers)
    avg = sum(numbers) / len(numbers)
    return min_val, max_val, avg

nums = [1, 2, 3, 4, 5]
minimum, maximum, average = get_stats(nums)
print(f"Stats of {nums}:")
print(f"  Min: {minimum}")
print(f"  Max: {maximum}")
print(f"  Avg: {average}")
```

```
Stats of [1, 2, 3, 4, 5]:
  Min: 1
  Max: 5
  Avg: 3.0
```

4. Variable Arguments

Use *args for variable positional arguments and **kwargs for variable keyword arguments.

```
def print_args(*args, **kwargs):
    """Variable number of arguments"""
    print(f"Args: {args}")
    print(f"Kwargs: {kwargs}")

print_args(1, 2, 3, name="Alice", age=20)
```

```
Args: (1, 2, 3)
Kwargs: {'name': 'Alice', 'age': 20}
```

5. Lambda Functions

Lambda functions are small anonymous functions defined in one line.

```
# Lambda function
square = lambda x: x**2
print(f"Lambda - square(5): {square(5)}")

# Use with map
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
print(f"Map: {squared}")

# Use with filter
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(f"Filter: {evens}")
```

```
Lambda - square(5): 25
Map: [1, 4, 9, 16, 25]
Filter: [2, 4]
```

6. Scope

Variables have different scopes: local (inside function) and global (outside function).

```
# Global variable
x = 10

def modify_x():
    # Local variable
    x = 20
    print(f"Inside function: x = {x}")

modify_x()
print(f"Outside function: x = {x}")

# Using global keyword
def modify_global():
    global x
    x = 30
    print(f"Modified global: x = {x}")

modify_global()
print(f"After modification: x = {x}")
```

```
Inside function: x = 20
Outside function: x = 10
Modified global: x = 30
After modification: x = 30
```

7. Docstrings

Docstrings document what a function does.

```
def calculate_bmi(weight, height):
    """
    Calculate Body Mass Index

    Parameters:
    weight (float): Weight in kilograms
    height (float): Height in meters

    Returns:
    float: BMI value
    """
    return weight / (height ** 2)

# Access docstring
print(calculate_bmi.__doc__)
```

```

    Calculate Body Mass Index

    Parameters:
    weight (float): Weight in kilograms
    height (float): Height in meters

    Returns:
    float: BMI value
```

Key Takeaways

- Functions promote code reusability
- Use def to define functions
- Default parameters provide flexibility
- *args and **kwargs for variable arguments
- Lambda functions for simple one-line functions
- Understand local vs global scope
- Document functions with docstrings