Python Functions Cheat Sheet

Basic Function Definition and Call

Function Definition

python Copy code

```python
def function_name(parameters):
    """
    Function docstring (optional)
    """
    # Function body
    # Code to be executed
    return value  # Optional
```

Function Call

python Copy code

```python
result = function_name(arguments)
```

Example

python Copy code

```python
def add(a, b):
    """Returns the sum of a and b."""
    return a + b

result = add(5, 3)
print(result)  # Output: 8
```

Function Arguments

Positional Arguments

python Copy code

```python
def greet(name, message):
    return f"{message}, {name}!"

print(greet("Alice", "Hello"))  # Output: Hello, Alice!
```

Keyword Arguments

python Copy code

```python
def greet(name, message):
    return f"{message}, {name}!"

print(greet(name="Alice", message="Hello"))  # Output: Hello, Alice!
print(greet(message="Hi", name="Bob"))       # Output: Hi, Bob!
```

Default Arguments

python Copy code

```python
def greet(name, message="Hello"):
    return f"{message}, {name}!"

print(greet("Alice"))          # Output: Hello, Alice!
print(greet("Bob", "Hi"))      # Output: Hi, Bob!
```

Return Values

python Copy code

```python
def add(a, b):
    return a + b

result = add(10, 5)
print(result)  # Output: 15
```

Docstrings

python Copy code

```python
def add(a, b):
    """
    Adds two numbers and returns the result.

    Parameters:
    a (int): The first number
    b (int): The second number

    Returns:
    int: The sum of the two numbers
    """
    return a + b

print(add.__doc__)
```

Lambda Functions

Basic Syntax

python Copy code

```python
lambda arguments: expression
```

Example

python Copy code

```python
add = lambda x, y: x + y
print(add(5, 3))  # Output: 8

# Using lambda with built-in functions
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x ** 2, numbers))
print(squared)  # Output: [1, 4, 9, 16, 25]
```

Recursion

python Copy code

```python
def factorial(n):
    """Returns the factorial of n."""
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```

Variable-Length Arguments

*args

python Copy code

```python
def sum_all(*args):
    return sum(args)

print(sum_all(1, 2, 3, 4))  # Output: 10
```

**kwargs

python Copy code

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

print_info(name="Alice", age=25, city="New York")
# Output:
# name: Alice
# age: 25
# city: New York
```

Built-in Higher-Order Functions

map()

python Copy code

```python
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x ** 2, numbers))
print(squared)  # Output: [1, 4, 9, 16, 25]
```

filter()

python Copy code

```python
numbers = [1, 2, 3, 4, 5]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # Output: [2, 4]
```

reduce()

python Copy code

```python
from functools import reduce

numbers = [1, 2, 3, 4, 5]
product = reduce(lambda x, y: x * y, numbers)
print(product)  # Output: 120
```

nonlocal Keyword

python Copy code

```python
def outer_function():
    x = "local"

    def inner_function():
        nonlocal x
        x = "nonlocal"
        print("Inner:", x)

    inner_function()
    print("Outer:", x)

outer_function()
# Output:
# Inner: nonlocal
# Outer: nonlocal
```

Decorators

Basic Decorator

python Copy code

```python
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
# Output:
# Something is happening before the function is called.
# Hello!
# Something is happening after the function is called.
```

Decorator with Arguments

python Copy code

```python
def repeat(num_times):
    def decorator_repeat(func):
        def wrapper(*args, **kwargs):
            for _ in range(num_times):
                result = func(*args, **kwargs)
            return result
        return wrapper
    return decorator_repeat

@repeat(num_times=3)
def greet(name):
    print(f"Hello, {name}!")

greet("Alice")
# Output:
# Hello, Alice!
# Hello, Alice!
# Hello, Alice!
```

Scope

Local Scope

python Copy code

```python
def my_function():
    x = 10  # Local variable
    print(x)

my_function()  # Output: 10
```

Global Scope

python Copy code

```python
x = 10  # Global variable

def my_function():
    print(x)

my_function()  # Output: 10
```