Python Online Free Ramzan Course 2025
Taught by: Shaida Muhammad

# Functions

## Day 11 - Python Basics

Shaida Muhammad

Monday, March 17, 2025

ShaidaSherpao@gmail.com

# Agenda



1 What are functions?

2 Defining and calling functions

3 Function parameters and arguments

4 Return values

5 Default arguments

6 Keyword arguments

7 Lambda functions (anonymous functions)

8 Variable scope (local vs. global)

9 Hands-on Practice

# What are Functions?

- **Definition:** A function is a reusable block of code that performs a specific task.
- **Why use functions?**
  - **Reusability:** Write once, use multiple times.
  - **Modularity:** Break down complex problems into smaller, manageable parts.
  - **Abstraction:** Hide implementation details, focus on functionality.
- **Example:**

```
def greet():
    print("Pa Khair Raaghly!")
```

# Defining and Calling Functions



- **Syntax:**

```
def function_name():
    # Code to execute
```

- **Example:**

```
def say_hello():
    print("Starry Mashy!")
```

- **Calling a Function:**

```
say_hello()  # Output: Starry Mashy!
```

- **Key Points:**
  - Use the `def` keyword to define a function.
  - Function names should be descriptive and follow snake_case.
  - Call a function using its name followed by parentheses `()`.

# Function Parameters and Arguments

- **Parameters:** Variables listed in the function definition.
- **Arguments:** Values passed to the function when calling it.
- **Example:**

```
def greet(name):  # 'name' is a parameter
    print(f"Pa Khair, {name}!")
greet("Ali")  # Ali is an argument
greet("Khan")   # Khan is an argument
```

- **Multiple Parameters:**

```
def add(a, b):
    print(a + b)
add(3, 5)  # Output: 8
```

# Return Values

- **Purpose:** Functions can return a value using the `return` statement.
- **Example:**

```
def add(a, b):
    return a + b
result = add(3, 5)
print(result)  # Output: 8
```

- **Key Points:**
  - The `return` statement exits the function and sends back a value.
  - If no `return` is specified, the function returns None.

# Default Arguments

- **Definition:** Default values for parameters if no argument is provided.
- **Example:**

```
def greet(name="Melma"):
    print(f"Pa Khair, {name}!")
greet()           # Output: Pa Khair, Melma!
greet("Ali")    # Output: Pa Khair, Ali!
```

- **Key Points:**
  - Default arguments make functions more flexible.
  - Parameters with default values must come after parameters without defaults.

# Keyword Arguments



- **Definition:** Arguments passed by explicitly naming the parameter.
- **Example:**

```
def greet(name, message):
    print(f"{message}, {name}!")
greet(name="Ali", message="Hi")
```

- **Key Points:**
  - Keyword arguments improve readability and allow out-of-order arguments.

# Lambda Functions

- **Definition:** Small, one-line functions defined without a name.
- **Syntax:**

```
lambda arguments: expression
```

- **Example:**

```
square = lambda x: x ** 2
print(square(5))  # Output: 25
```

- **Key Points:**
  - Lambda functions are useful for short, throwaway functions.
  - Often used with `map()`, `filter()`, and `sorted()`.

# Variable Scope



- **Local Variables:** Variables defined inside a function. They are accessible only within that function.

```
def my_function():
    x = 10  # Local variable
    print(x)
my_function()  # Output: 10
print(x)  # Error: x is not
defined outside the function
```

- **Global Variables:** Variables defined outside a function. They are accessible everywhere.

```
x = 20  # Global variable
def my_function():
    print(x)  # Accessing
global variable
my_function()  # Output: 20
```

- **Modifying Global Variables:** Use the `global` keyword.

```
x = 10
def modify_global():
    global x
    x = 20
modify_global()
print(x)  # Output: 20
```

# Hands-On Practice



- **Task 1:** Write a function `greet()` that prints "Pa Khair, Halko!".

```
def greet():
    print("Pa Khair,Halko!")
greet()
```

- **Task 2:** Write a function `add(a, b)` that returns the sum of two numbers.

```
def add(a, b):
    return a + b
print(add(3, 5))  # Output: 8
```

- **Task 3:** Write a function `is_even(num)` that checks if a number is even and returns `True` or `False`.

```
def is_even(num):
    return num % 2 == 0
print(is_even(4))
print(is_even(5))
```

- **Task 4:** Write a function `calculate_area(length, width)` that returns the area of a rectangle.

```
def calculate_area(length, width):
    return length * width
print(calculate_area(5, 10))
# Output: 50
```

- **Task 5:** Write a lambda function to calculate the square of a number.

```
square = lambda x: x ** 2
print(square(5))  # Output: 25
```

# Recap

- Functions are reusable blocks of code that perform specific tasks.
- Use def to define a function and return to send back a value.
- Parameters are placeholders; arguments are actual values passed to the function.
- Default arguments and keyword arguments make functions more flexible.
- Lambda functions are useful for short, throwaway operations.
- Variables can have local or global scope.

# Homework

1. Write a function find_max(num1, num2) that returns the larger number.
2. Write a function `find_max_l(lst)` that returns the maximum value in a list.
3. Write a function multiply(a, b) that returns the product of two numbers.
4. Write a function convert_to_celsius(fahrenheit) that converts Fahrenheit to Celsius.
5. Experiment with default and keyword arguments in custom functions.
6. Write a function `is_prime(n)` that checks if a number is prime.
7. Use a lambda function with `map()` to square all numbers in a list.

# Q&A

- Do you have any questions?
- Share your thoughts.

# Closing

## Next class: Strings