رواد مصر الرقمية

وزارة الاتصـــــالات
وتكنولوجيا المعلومات

Python Programming

Session 3: (Functions)

Agenda:

1	Defining and Using Functions
2	Advanced Function Concepts
3	Practical Examples

رواد مصر الرقمية

وزارة الاتصـــــالات
وتكنولوجيا المعلومات

Defining and Using Functions

Defining and Using Functions

Functions are blocks of reusable code that perform a specific task. They help organize code into smaller, manageable pieces and can be used multiple times throughout a program.

Defining and Using Functions

Basic Function Definition and Call

```
def greet():
    print("Hello, world!")

greet()
```

```
Hello, world!
```

A function named greet() is defined with no parameters.
When greet() is called, it prints "Hello, world!".

Defining and Using Functions

Function with Parameters

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

result = add(5, 3)
print("Sum:", result)
```

```
Sum: 8
```

This function add(a, b) takes two parameters and returns their sum. Calling add(5, 3) results in 8 being printed.

Defining and Using Functions

Function with Default Parameter Values

```
def greet(name="Guest"):
    print(f"Hello, {name}!")

greet("Alice")
greet()
```

```
Hello, Alice!
Hello, Guest!
```

The function greet(name="Guest") has a default parameter value. If no argument is provided, it defaults to "Guest".

Defining and Using Functions

Function Returning Multiple Values

```
def divide(a, b):
    quotient = a // b
    remainder = a % b
    return quotient, remainder

q, r = divide(10, 3)
print("Quotient:", q)
print("Remainder:", r)
```

```
Quotient: 3
Remainder: 1
```

The divide(a, b) function returns both the quotient and remainder. These values are unpacked into q and r.

Defining and Using Functions

Using Docstrings for Function Documentation

```
def multiply(a, b):
    """
    Multiplies two numbers and returns the result.
    """
    return a * b

print(multiply(4, 5))
print(multiply.__doc__)
```

```
20

    Multiplies two numbers and returns the result.
```

This function multiply(a, b) includes a docstring to describe what the function does. The __doc__ attribute prints the docstring.

Advanced Function Concepts

Lambda Functions

Lambda functions are small, anonymous functions defined using the lambda keyword. They are used for short, simple operations.

Lambda Functions

Basic Lambda Function

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

```
25
```

A lambda function is defined to square a number. The output is 25.

Lambda Functions

Lambda Function with Multiple Arguments

```
multiply = lambda x, y: x * y
print(multiply(3, 4))
```

```
12
```

This lambda function takes two arguments and multiplies them. The output is 12.

Lambda Functions

Using Lambda in map() Function

```
numbers = [1, 2, 3, 4, 5]
squares = list(map(lambda x: x ** 2, numbers))
print(squares)
```

```
[1, 4, 9, 16, 25]
```

The map() function applies the lambda function to each element of the list. The result is [1, 4, 9, 16, 25].

Lambda Functions

Using Lambda in filter() Function

```
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)
```

```
[2, 4, 6]
```

The filter() function uses the lambda to keep only even numbers. The output is [2, 4, 6].

Recursive Functions

Recursive functions are functions that call themselves to solve a problem by breaking it down into smaller subproblems.

Recursive Functions

Calculating Factorial Using Recursion

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

print(factorial(5))
```

```
120
```

The factorial(n) function calls itself with decremented values of n. It calculates 5! = 5 * 4 * 3 * 2 * 1 = 120.

Recursive Functions

Fibonacci Sequence Using Recursion

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

print(fibonacci(6))
```

```
8
```

This function calculates the Fibonacci number at position n by summing the previous two Fibonacci numbers. The output is 8.

Higher-Order Functions

Higher-order functions are functions that can take other functions as arguments or return them as results.

Higher-Order Functions

Using a Function as an Argument

```
def apply_operation(a, b, operation):
    return operation(a, b)

def add(x, y):
    return x + y

print(apply_operation(5, 3, add))
```

```
8
```

The apply_operation(a, b, operation) function takes another function (add) as an argument and applies it. The output is 8.

Higher-Order Functions

Returning a Function from Another Function

```
def outer_function(message):
    def inner_function():
        print(message)
    return inner_function

greet = outer_function("Hello!")
greet()
```

```
Hello!
```

The outer_function() returns an inner_function that prints a message. When greet() is called, it prints "Hello!".

Practical Examples

Calculating Average of Numbers Using Functions

Calculating Average of Numbers Using Functions

```
def calculate_average(numbers):
    return sum(numbers) / len(numbers)

numbers = [10, 20, 30, 40, 50]
print("Average:", calculate_average(numbers))
```

```
Average: 30.0
```

This function calculates the average of a list of numbers. The output is 30.0.

Calculating Average of Numbers Using Functions

Finding the Longest Word in a List

```
def find_longest_word(words):
    longest = ""
    for word in words:
        if len(word) > len(longest):
            longest = word
    return longest

words = ["apple", "banana", "cherry", "date"]
print("Longest word:", find_longest_word(words))
```

```
Longest word: banana
```

This function iterates over a list of words and returns the longest one. The output is "banana".

Calculating Average of Numbers Using Functions

Checking if a Number is Prime

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

print("Is 17 prime?", is_prime(17))
```

```
Is 17 prime? True
```

The is_prime(n) function checks if a number is prime by testing divisibility up to its square root. The output is True.

Calculating Average of Numbers Using Functions

Converting Temperature Units

```
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

def fahrenheit_to_celsius(fahrenheit):
    return (fahrenheit - 32) * 5/9

print("25°C to Fahrenheit:", celsius_to_fahrenheit(25))
print("77°F to Celsius:", fahrenheit_to_celsius(77))
```

```
25°C to Fahrenheit: 77.0
77°F to Celsius: 25.0
```

Two functions convert temperatures between Celsius and Fahrenheit. The output is 77.0 and 25.0.

Calculating Average of Numbers Using Functions

Generating a List of Squares Using List Comprehension and Functions

```
def square_list(numbers):
    return [x ** 2 for x in numbers]

numbers = [1, 2, 3, 4, 5]
print("List of squares:", square_list(numbers))
```

```
List of squares: [1, 4, 9, 16, 25]
```

The function square_list(numbers) uses list comprehension to generate a list of squares for the given input list. It iterates over each number, squares it, and creates a new list with these squared values. The output is [1, 4, 9, 16, 25].

Calculating Average of Numbers Using Functions

Converting a List of Strings to Uppercase

```
def convert_to_uppercase(words):
    return [word.upper() for word in words]

words = ["hello", "world", "python", "functions"]
print("Uppercase words:", convert_to_uppercase(words))
```

```
Uppercase words: ['HELLO', 'WORLD', 'PYTHON', 'FUNCTIONS']
```

The function convert_to_uppercase(words) uses list comprehension to iterate over each word in the input list, converting each word to uppercase. The output is ['HELLO', 'WORLD', 'PYTHON', 'FUNCTIONS'].

Calculating Average of Numbers Using Functions

Calculating the Total Price Including Tax

```
def calculate_total_price(prices, tax_rate):
    total = sum(prices)
    total_with_tax = total * (1 + tax_rate)
    return total_with_tax

prices = [100, 200, 300]
tax_rate = 0.1
print("Total price with tax:", calculate_total_price(prices, tax_rate))
```

```
Total price with tax: 660.0
```

The function calculate_total_price(prices, tax_rate) calculates the total price of items in the list and then applies the tax rate. The output is 660.0.

Calculating Average of Numbers Using Functions

Finding the Minimum and Maximum Values in a List

```
def find_min_max(numbers):
    return min(numbers), max(numbers)

numbers = [3, 7, 2, 9, 4]
min_value, max_value = find_min_max(numbers)
print("Minimum value:", min_value)
print("Maximum value:", max_value)
```

```
Minimum value: 2
Maximum value: 9
```

The function find_min_max(numbers) uses the built-in min() and max() functions to find the smallest and largest numbers in the input list. The output is 2 (minimum) and 9 (maximum).

Calculating Average of Numbers Using Functions

Checking if a String is a Valid Email Address

```
def is_valid_email(email):
    return "@" in email and "." in email.split("@")[-1]

email = "example@example.com"
print("Is valid email:", is_valid_email(email))
```

```
Is valid email: True
```

The function is_valid_email(email) checks if a given string contains "@" and a "." after the "@", which are basic checks for a valid email format. The output is True.

Calculating Average of Numbers Using Functions

Counting the Frequency of Each Character in a String

```
def count_character_frequency(text):
    frequency = {}
    for char in text:
        if char in frequency:
            frequency[char] += 1
        else:
            frequency[char] = 1
    return frequency

text = "hello world"
print("Character frequency:", count_character_frequency(text))
```

```
Character frequency: {'h': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1}
```

The function count_character_frequency(text) counts the occurrences of each character in the input string and returns a dictionary with characters as keys and their frequencies as values. The output is {'h': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1}.