Functions in Python

By

Prof. Muhammad Iqbal Bhat

Department of Higher Education

Government Degree College Beerwah

Topics:

1. What are Functions?
2. Advantages of Using Functions
3. Syntax for Defining and Calling Functions

Functions in Python

1. Functions are an essential part of programming in Python
2. They are used to group a set of related statements and execute them multiple times
3. Functions make code more organized, efficient, and reusable

Advantages of Functions:

Modularity: Functions allow you to break down a program into smaller, more manageable pieces. This makes it easier to read, understand, and maintain the code.

Reusability: Functions can be reused in multiple places in your code. This can save you time and effort, and also reduces the amount of code you need to write.

Abstraction: Functions allow you to abstract away the details of how a task is performed. This makes it easier to reason about the code and to modify it in the future.

Debugging: Functions can help make debugging easier by isolating specific parts of the program. You can test and debug individual functions without affecting the rest of the code.

Readability: Functions make the code more readable and understandable. By giving each task its own function, you can create self-contained blocks of code that are easier to comprehend.

Syntax for Defining and Calling Functions:

- The "def" keyword is used to define a function in Python

```
def greet(name):
    print("Hello, " + name)
```

- Function names should be descriptive and follow the naming conventions of Python
- Function parameters are optional, but if used, they should be enclosed in parentheses

```
greet("Iqbal")
```

- Function blocks are indented and end when the indentation level returns to the previous level
- Functions are called using the function name followed by parentheses

Syntax for Defining and Calling Functions:

- Functions can also have default values for their arguments. This means that if no value is passed in for the argument, it will use the default value.

```
def greet(name="World"):
    print("Hello, " + name)
greet()              # Output: Hello, World
greet("iqbal")       # Output: Hello, John
```

- we can return values from a function using the "return" keyword

```
def add_numbers(x, y):
    return x + y

result = add_numbers(3, 4)
print(result)   # Output: 7
```

Variations of Functions in Python

1. Functions with Required Arguments:

- Required arguments are the arguments that have to be passed to a function in a particular order. If the required arguments are not provided in the function call, it will raise an error.

```
def greet(name, age):
    print("Hello, my name is", name, "and I'm", age, "years old.")

greet(“iqbal", 35)
```

2. Functions with Default Arguments:

- Default arguments are the arguments that take a default value if no value is passed to them. This helps in making the function more flexible. .

```
def greet(name, age=25):
    print("Hello, my name is", name, "and I'm", age, "years old.")

greet("John")
```

3.1 Functions with Variable-Length Arguments:

- Sometimes, you might want to pass a variable number of arguments to a function. In such cases, you can use variable-length arguments. There are two types of variable-length arguments: *args and **kwargs.

```
def sum_numbers(*args):
    sum = 0
    for num in args:
        sum += num
    return sum

result = sum_numbers(1, 2, 3, 4)
print(result)   # Output: 10
```

3.2 Functions with Variable-Length Arguments:

- **kwargs is used to pass a variable-length dictionary of named arguments to a function.

```
def print_info(**kwargs):
    for key, value in kwargs.items():
        print(key + ': ' + value)

print_info(name='iqbal', age='25', city='New York')
```

4. Lambda Functions:

- Lambda functions are anonymous functions that can have any number of arguments, but can only have one expression.
- They are useful when you need a small function for a short period of time

```
add_numbers = lambda x, y: x + y
result = add_numbers(3, 4)
print(result)   # Output: 7
```

- Lambda functions are commonly used in combination with other functions like "map", "filter", and "reduce".

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

5. Recursive Functions:

- Recursive functions are functions that call themselves. They are used to solve problems that can be broken down into smaller, similar problems.

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

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

6. Higher-Order Functions:

- Higher-order functions are functions that take other functions as arguments and/or return functions as output. They are used to write more abstract and reusable code..

```
def apply_operation(num, func):
    return func(num)

def square(x):
    return x * x

result = apply_operation(3, square)
print(result)   # Output: 9
```

Questions?