SDEV 1001

Programming Fundamentals

Modules and Functions - 1 and 2

A LEADING POLYTECHNIC COMMITTED TO YOUR SUCCESS

Expectations - What I expect from you

- No Late Assignments

- No Cheating

- Be a good classmate

- Don’t waste your time

- Show up to class

Agenda

On the right is what we will cover today.

- Introduction to Functions and Modular Code
- Defining a Function
- Functions with Parameters
- Returning Values
- Organizing Code with Functions
- The Main Guard
- Default Parameter Values
- Keyword Arguments
- Importing Functions from Other Files
- Summary

Introduction to Functions and Modular Code

What is a function.

A function is a reusable block of code that performs a specific task. Functions help organize code into logical sections, making it easier to read, maintain, and debug. Analogy: Think of a function as a tool in a toolbox. Each tool (function) has a specific purpose, and you can use it whenever you need to perform that task.

Why use functions?

- As programs grow, organizing code into functions makes it easier to read, maintain, and debug.
- Functions let you break your code into reusable, logical pieces.
- This is in every programming language and is a fundamental concept in programming.

Defining a Function

A function is a block of code that performs a specific task. You define a function using the `def` keyword followed by the function name and parentheses.

Analogy: Think of a function as a recipe. You define the steps (code) to make a dish (task), and you can use that recipe whenever you want to make that dish.

```
# function definition
def greet_user():
    print("Hello, welcome to the program!")

# function call
greet_user()
```

Notes on the above example - Use `def` to define a function, you can name a function anything you want (as long as it follows Python's naming rules). Here we named it `greet_user`. - Call the function by its name followed by parentheses.

Functions with Parameters

Functions can take inputs called parameters, allowing you to pass data into the function.

- Note parameters are defined in the function definition, and you provide values (arguments) when calling the function.

Analogy: Think of parameters as ingredients in a recipe. You can change the ingredients to make different versions of the dish. For example a pizza recipe can take different toppings as parameters.

```
def add_numbers(a, b): # a and b are parameters
    return a + b

result = add_numbers(3, 7) # 3 and 7 are arguments
print("Sum:", result)
```

Here, `a` and `b` are parameters. When we call `add_numbers(3, 7)`, it passes the values 3 and 7 to the function, which then returns their sum.

Returning Values

The `return` statement sends a value back to the caller.

Analogy: Think of a function as a machine that processes inputs and gives you an output.

```
def get_favorite_color():
    color = input("What's your favorite color? ")
    return color

user_color = get_favorite_color()
print("You like", user_color)
```

Here the variable `color` only exists with in the function scope, but we can `return` it to use outside the function. So when we call `get_favorite_color()`, it returns the user's input which we can store in `user_color`.

Organizing Code with Functions

Functions help separate concerns and make code easier to test and reuse.

Example

```
def get_temperature():
    return float(input("Enter the temperature: "))

def is_hot(temp):
    return temp > 30

temp = get_temperature()
if is_hot(temp):
    print("It's a hot day!")
else:
    print("It's not too hot.")
```

The Main Guard

Code under `if __name__ == "__main__":` only runs when the file is executed directly, not when imported.

- For example if the file is named `example.py` and you run it directly with `python example.py` in the terminal, the code under the main guard will execute.

Example

```
def main():
    print("This code runs when the script is executed directly.")

if __name__ == "__main__":
    main()
```

You'll be using this pattern to ensure that your code runs correctly whether it's imported as a module or run as a script.

Default Parameter Values

You can provide default values for parameters, making them optional.

```
def power(base, exponent=2):
    return base ** exponent

print(power(3))      # 9 (3 squared)
print(power(2, 3))   # 8 (2 cubed)
```

Note on the above example:

- If you don’t provide a value, the default is used.
  - so in the first call `power(3)`, the `exponent` defaults to 2.
  - In the second call `power(2, 3)`, you override the default by providing a value for `exponent`.

Keyword Arguments

You can specify arguments by name, making your code clearer.

```
def describe_pet(animal, name):
    print(f"I have a {animal} named {name}.")

describe_pet(animal="dog", name="Buddy")
describe_pet(name="Whiskers", animal="cat")
```

- Keyword arguments can be given in any order.

Important Note: You always have to order your keyword arguments after positional arguments, so if you have a function that takes both positional and keyword arguments, the positional arguments must come first.

- We we'll see this a lot more as we program more complex functions.

Importing Functions from Other Files

You can organize your code by putting functions in separate files (modules) and importing them.

Suppose you have a file `math_utils.py`:

```
def square(n):
    return n * n
```

You can use it in another file:

```
from math_utils import square

print(square(6))  # 36
```

We'll talk more about modules and how to set up a project structure later, but for now, this is how you can import functions from other files.

- There's more to importing than just this, but this is a good start to conceptualize how to use functions from other files.

Summary

- Functions make your code modular, readable, and reusable.
- Use functions to break up tasks, accept arguments, and return results.
- The main guard helps control script execution and importing.
- Functions with parameters are powerful and flexible.
- Use multiple, default, and keyword arguments to make your functions easy to use.
- Organize your code by importing functions from other files (modules).

Example

Let’s go run a few examples together