Chapter 8 - User-Defined Functions

User-Defined Functions

Parameters

Learning Objective - Parameters

- **Demonstrate how to define a function with parameters**
- **Identify which value is assigned to each parameter**
- **Assign a parameter a value based on its name**
- **Identify which data types can be used as parameters**
- **Check parameter types and provide feedback when there is an error**
- **Explain how optional parameters work**
- **Demonstrate how to declare a function with an undetermined amount of parameters**

Passing Parameters

Parameters

Parameters are required values passed to a function. The function uses the parameters to accomplish the stated goal in the docstring. In the example below, the function adds the parameters together. Parameters are those values in between the parentheses. If your function uses parameters, they need to be present when you define and call the function. Multiple parameters are separated by commas.

```
def addition(num1, num2):
    """Prints the sum of two numbers"""
    print(num1 + num2)

addition(5, 7)
```

Parameters

```
def addition(num1, num2):
    """Prints the sum of two numbers"""
    print(num1 + num2)

addition(5, 7)
```

challenge

What happens if you:

- Change the function header to `def addition(num1):`?
- Change the function call to `addition()`?
- Change the function call to `addition(5, 10, 15)`?

Parameters as Variables

You can think of parameters as variables. They are declared in the function header, and their values are determined by the function call. Because of this, the order of parameters is important. The first parameter in the function call will be the first parameter in the function header, the second parameter from the function call will be the second parameter in the function header, etc.

```
def add_sub(num1, num2, num3):
    """add_sub does the following:
    Add the first two parameters
    Subtract the third paramter
    Print the result"""
    print(num1 + num2 - num3)

add_sub(5, 10, 15)
```

Parameter Order

```
def add_sub(num1, num2, num3):
    """add_sub does the following:
    Add the first two parameters
    Subtract the third paramter
    Print the result"""
    print(num1 + num2 - num3)

add_sub(5, 10, 15)
```

challenge

What happens if you:

- Change the function call to `add_sub(10, 15, 5)`?
- Change the function call to `add_sub(15, 5, 10)`?
- Change the function call to `add_sub(10 + 5, 20 // 4, 5 * 2)`?

Parameter Values

Named Parameters

Typically, parameter values are assigned based on their position in the function call. However, Python allows you to pass a value to a parameter based its name.

```
def subtract(num1, num2):
    """Subtract the second parameter from the first"""
    print(num1 - num2)

subtract(5, 2)
subtract(2, 5)
subtract(num2=2, num1=5)
```

> challenge
>
> ## What happens if you:
>
> - Change the function call to `subtract(num3=2, num1=5)`?
> - Change the function call to `subtract(num1=2, 5)`?
> - Change the function call to `subtract(num1=2, num1=5)`?

Parameter Values

If parameters can be thought of as variables, then they can have the same values as variables: ints, floats, strings, boolean, lists, etc.

```
def parameter_types(param1, param2, param3, param4):
    """Takes four parameters
    Print the type of each element"""
    print("The type of {} is {}".format(param1, type(param1)))
    print("The type of {} is {}".format(param2, type(param2)))
    print("The type of {} is {}".format(param3, type(param3)))
    print("The type of {} is {}".format(param4, type(param4)))

parameter_types(1, 5.9, "Beatles", False)
```

challenge

What happens if you:

- Change the function call to `parameter_types([1,2,3], -6, len, True)`?
- Change the function call to `parameter_types(range(10), "", parameter_types, 45)`?

Checking Parameters

Checking Parameter Data Types

Functions can fail with the wrong data type is passed as a parameter. The function definition below expects two numbers, but the function calls passes a string.

```
def addition(num1, num2):
    """Add the two parameters together"""
    print(num1 + num2)

addition(5, "cat")
```

This code generates a type error. Python says the + operator cannot work with operands (the items being used with +) of type string. The program terminates with a cryptic error message. The try... except keywords allow for a more user-friendly error message.

```
def addition(num1, num2):
    """Add the two parameters together
    Use try/except to catch any errors"""
    try:                                    Run if there are no errors
        print(num1 + num2)
    except:                                 Run if there is an error
        print("There is an error in your code.")
```

Try Except

```
def addition(num1, num2):
    """Add the two parameters together
    Use try/except to catch any errors"""
    try:
        print(num1 + num2)
    except:
        print("There is an error in your code.")

addition(5, "cat")
```

Notice that a green check mark appears even though there is an error

in the function call. `try... except` keeps the program running while providing feedback to the user.

▶ **Failing Gracefully**

challenge

What happens if you:

- Change the function call to `addition(5, 3)`?

Error Types

The code above provides an error message that is easy to understand, but it is not very helpful. What exactly is the problem? Python allows you to customize the exception messages based on the type of error. Trying to add a string to an int causes a type error, so the exception and message should reflect this.

```
def addition(num1, num2):
    """Add the two parameters together
    Use try/except to catch any type errors"""
    try:
        print(num1 + num2)
    except TypeError:
        print("Please pass the function two numbers")

addition(5, "cat")
```

▶ **Python Errors**

challenge

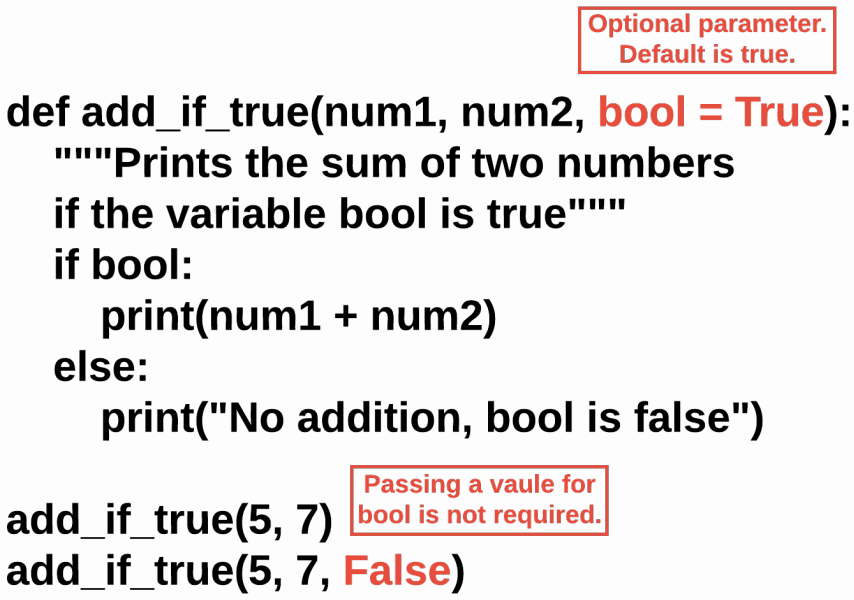
What happens if you:

- Change the print statement in the `try` block to `print(num1 + num3)`?

Parameters - Advanced Topics

Optional Parameters

Python allows you to create functions with optional parameters. They are considered to be optional because the function call can state the parameter or not. However, the function declaration will name this parameter and give it a default value.



Optional Parameters

```
def add_if_true(num1, num2, bool = True):
    """Prints the sum of two numbers
    if the variable bool is true"""
    if bool:
        print(num1 + num2)
    else:
        print("No addition, bool is false")

add_if_true(5, 7)
add_if_true(5, 7, False)
```

challenge

What happens if you:

- Add a function call that looks like this `add_if_true(29, 45, True)`?

Variable Parameter Lists

It is possible to declare a function with a list of variables of an undetermined length. The function below will find the sum for any number passed as a parameter. There can be two parameters or twenty, but it is not necessary to write out each parameter. Instead, use a * before a single parameter name. This creates a list of parameters.

```
def calc_sum(*nums):
    """Calculate the sum of all of the parameters"""
    total = 0
    for num in nums:
        total += num
    print(total)

calc_sum(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

challenge

What happens if you:

- Change the function call to `calc_sum(4)`?
- Change the function call to `calc_sum()`?