Overview

We have now input values, processed values, and output values. That is a large part of the computer science curriculum.

The other large part comes from breaking the different processes you will program into modules. In this course, the modules we will be focusing on are functions.

Important Information

A function is a block of code that only runs when it is called.

Every function has:

- A name
- the `def` keyword
- a pair of parenthesis `()`
- a colon `:` to signify where the block of code begins
- an indented block of code beneath it

Code defined in a function is reusable, and can be added in the program as many times as the user sees fit.

Definition

Let's look at an example of a function definition.

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

- `def` tells us this is a function
- the function is named `greet`
- double parenthesis and a colon are required `():`
- `print("Hello World!")` is the indented block of code

Indentation Is Required

In Python, **every function must have an indented block of code** underneath it.

If a function does not have an indented block, Python will raise an `IndentationError`.

This code will cause an error:

```
def greet():
```

Python expects at least one indented line after the colon. If it does not find one, the program will not run.

The `pass` Keyword

Sometimes you want to define a function before you are ready to write its code. In this case, you can use the `pass` keyword.

`pass` tells Python when a function is intentionally left empty.

Example:

```
def greet():
    pass
```

The above function is valid, and does not cause any errors.

Using `pass` is useful when planning your program or writing placeholder functions.

Calling Functions

If we wanted to call this function later it would look like this:

```
greet()
```

Putting it all together we get code that looks like this:

```
# Define the function here
def greet():
    print("Hello, World!")

# Call the function here
greet()
```

Order Matters

Another thing that is important to note about functions is that in python, functions ALWAYS need to be defined before they are called. In some languages it doesn't matter where you put your functions, but in python your functions need to be written above the place they are called in your code.

The following code will run just fine:

```
# Function definition before function is called
def greet():
    print("Hello, World!")

# Function is called after being defined
greet()
```

However, this code will not:

```
# Function called before being defined, results in a NameError
greet()

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

When a function is called before it is defined, we get a `NameError` because the function is trying to be called before it is defined. The easiest fix for this is to ensure the function is defined above it is called.

Function Scope

Sometimes you will want to use variables inside of functions. In python, functions are a special part of the code that treat variables differently than other parts of the code. This is called variable scope.

At this point, we will focus on local and global scope for variables.

Variables that are declared inside of functions have a local scope. This means that the only place in your code that the variable can be accessed, assigned, compared, or seen by your code is within the function. Variables from a function cannot be accessed outside of that function.

```
def greet():
    # message is a local variable to the greet function
    message = "Hello, World!"
    # message can be accessed here because it is in the function
    print(message)

# message cannot be accessed here because it is outside of the function. This will result in a
print(message)
```

Variables that are declared outside of functions have global scope, meaning that they can be accessed anywhere in your code.

```
# message is declared as a global variable
message = "Hello, World!"

def greet():
    print(message)

# message is global so this line does not result in an error
print(message)

# when greet is called we get a NameError even though message is a global variable...
greet()
```

By default, global variables can't be reassigned inside of functions. To reassign a global variable inside of a function, you need to use the `global` keyword.

```
# message is declared as a global variable
message = "Hello, World!"

def greet():
    # by specifying that we want to use the global message variable, python interprets this var
    global message
    message = "Goodbye now!"
    print(message)

# message is global so this line does not result in an error
print(message)

# no more errors!
greet()
```

Copy, Change, Challenge – Defining and Calling Functions

Copy

```
def greet():
    print("Hello from a function!")

greet()
```

Change

Change the message inside the `print()` statement so the function prints a different greeting.

Challenge

Create a new function called `farewell` that prints a goodbye message, and then call it after `greet()`.

Copy, Change, Challenge – Using an Empty Function

Copy

```
def not_ready_yet():
    pass

not_ready_yet()
```

Change

Replace `pass` with a `print()` statement so the function displays a message.

Challenge

Create a function that initially uses `pass`, then later add an indented block of code that prints at least two lines when the function is called.

Copy, Change, Challenge – Using a Local Variable

Copy

```
def show_score():
    score = 10
    print(score)

show_score()
```

Change

Change the value of the `score` variable to a different number and run the program again.

Challenge

Add a second local variable inside the function and print both values on separate lines.

Copy, Change, Challenge – Using the `global` Keyword

Copy

```
total = 5

def add_to_total():
    global total
    total = total + 3

add_to_total()
print(total)
```

Change

Change the value being added to `total` so the final printed value is different.

Challenge

Create a second function that also uses `global total` and changes the value in a different way before printing it.