FUNCTIONS

CS 3080: Python Programming

UCCS University of Colorado Colorado Springs

Functions

- A major purpose of functions is to group code that gets executed multiple times.

```
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
```

Functions

- A **function** is like a mini-program inside a program.
  - *Major purpose: to group code that gets executed multiple times*
- We create a function by using the **def** statement.
- In code, a function call is just the function's name followed by parentheses, possibly with some arguments.
  - *The first line is the **def** statement, which defines a function named hello()*
  - *The code in the block that follows the **def** statement is body of function*
  - *The code inside the function is executed when the function is called, not when the function is first defined.*

Functions

- In general, you always want to avoid duplicating code, because if you ever decide to update the code—if, for example, you find a bug you need to fix—you'll have to remember to change the code everywhere you copied it.
- As you get more programming experience, you'll often find yourself deduplicating code, which means getting rid of duplicated or copy and pasted code. Deduplication makes your programs shorter, easier to read, and easier to update.

Functions with arguments

- Values passed into a function are called **arguments**
  - *They are typed between the parentheses of the calling code*
- A **parameter** is a variable that an argument is stored in when a function is called
- The value stored in a parameter is forgotten when the function returns.

Define, Call, Pass, Argument, Parameter

- To **define** a function is to create it
- The myAge('29') line **calls** the now-created function, sending the execution to the top of the function's code
  - *Also known as* passing *the string value '29' to the function*
- The value passed to a function in a function call is an **argument**
- The argument '29' is assigned to a local variable named *age*.
- Variables that have arguments assigned to them are **parameters**.

```
def myAge(age)
    print('My age is ' + age)
myAge('29')
```

Return statement

- The value that a function call evaluates to is called the **return value** of the function
- When creating a function using the **def** statement, you can specify what the return value should be with a **return** statement
- A return statement consists of the following:
  - *The **return** keyword*
  - *The value or expression that the function should return*
- We can pass return values as an argument to another function call.

None value

- Represents the absence of a value.
- Other programming languages might call this value null, nil, or undefined.
- It is a value-without-a-value
- Must be typed with a capital N.
- Behind the scenes, Python adds return **None** to the end of any function definition with no return statement

Keyword arguments

- Keyword arguments are identified by the keyword put before them in the function call.
- Keyword arguments are often used for optional parameters.
- Can add keyword arguments to the functions we write but first we need to learn about list and dictionary data types (next two chapters)

Positional arguments vs Keyword arguments

Most arguments are identified by their position in the function call

random.randint(1, 10) vs random.randint(10, 1)

Keyword arguments identified by keyword, not the position

sum(a = 5, b = 10) vs sum(5, 10)

Keyword arguments Positional arguments

Default arguments

- Default values indicate that the function argument will take that value if no argument value is passed during function call.

Local and Global scope

- Parameters and variables that are assigned in a called function are said to exist in that function's **local scope** >>> **Local variable**
- Variables that are assigned outside all functions are said to exist in the **global scope** >>> **Global variable**
- Variables must be one or the other; it cannot be both local and global
- When a scope is destroyed, all the values stored in the scope's variables are forgotten.
- A local scope can access global variables.
- Code in a function's local scope cannot use variables in any other local scope.
- You can use the same name for different variables if they are in different scopes.
- While using global variables in small programs is fine, it is a bad habit to rely on global variables as your programs get larger.

Global statement

- Local variables in one function are completely separate from the local variables in another function
- If you need to **modify** a global variable from within a function, use the **global** statement.

Function Attributes

- In Python, every function is a first-class object, which gives us the ability to pass them as arguments, assign to variables, compare to each other and return from other functions.

- Every function has a number of additional attributes which can be accessed using dot syntax

Exception handling

- Getting an error, or exception, in a Python program means the entire program will crash
- Prefer that program detects errors, handles them, and then continues to run
- Example
  - Divide-by-zero error = ZeroDivisionError

- Errors can be handled with **try** and **except** statements.
- The code that could potentially have an error is put in a try clause.
- The program execution moves to the start of a following except clause if an error happens.
- After running that code, the execution continues as normal.