Introduction to Programming

Spring 2022

Functions

- The Function of Functions
- Functions Informally
- **Functions and Parameters: The Executing Details**
- **Functions That Return Values**
- **Functions that Modify Parameters**
- **Functions and Program Structure**
- **Class Work**

Functions and Parameters

- Scope of a variable
  - A variable created inside a function is only visible inside that function
- They are accessible inside the body of the function
- Variables with same name but inside different bodies of functions are different
  - Example

Functions and Parameters

- We can pass data to a function using parameters
- A function definition looks like this:
-
```
def <name>(<formal-parameters>):
      <body>
```
- The name of the function must be an identifier
  - Just like variables
- Formal-parameters is a (possibly empty) list of variable names

Functions and Parameters

- Formal parameters
  - Are only accessible in the body of the function.
  - They are similar to variables created inside the body of the function.
  - Variables with identical names elsewhere in the program are distinct from the formal parameters.

Functions and Parameters

- A function is called by using its name followed by a list of actual parameters or arguments.
- `<name>(<actual-parameters>)`
- When Python comes to a function call, it initiates a four-step process.

Functions and Parameters

- The calling program suspends execution at the point of the call.
- The formal parameters of the function get assigned the values supplied by the actual parameters in the call.
- The body of the function is executed.
- Control returns to the point just after where the function was called.
- Example

Functions and Parameters

- The formal variable disappears after the end of the function
- The memory occupied by local function variables is reclaimed when the function exits.
- Local variables do not retain any values from one function execution to the next.

Functions and Parameters

- We can also have multiple parameters.
- Formal and actual parameters are matched up based on position
  - The first actual parameter is assigned to the first formal parameter
  - The second actual parameter is assigned to the second formal parameter
  - and so on.

Functions and Parameters

- Passing parameters provides a mechanism for initializing the variables in a function.
- Parameters act as inputs to a function.
- We can call a function many times and get different results by changing its parameters.

Functions That Return Values

- We've already seen numerous examples of functions that return values to the caller.
- `sq_root = math.sqrt (a)`
- The value `a` is the actual parameter of `math.sqrt`.
- We say `sqrt` returns the square root of its argument.

Functions That Return Values

- We can use `return` key word to return a value from the function
- When Python encounters `return`, it exits the function and returns control to the point where the function was called.
- In addition, the value(s) provided in the `return` statement are sent back to the caller as an expression result.
- Example

Functions That Return Values

- Sometimes a function needs to return more than one value.
- To do this, simply list more than one expression in the return statement.
- Example

Functions That Return Values

- All Python functions return a value:
  - Whether they contain a return statement or not.
  - Functions without a return hand back a special object, denoted None.
- A common problem is writing a value-returning function and omitting the `return`!
- If your value-returning functions produce strange messages, check to make sure you remembered to include the return!

Functions that Modify Parameters

- The formal parameters of a function only receive the values of the actual parameters.
  - The function does not have access to the variable that holds the actual parameter.
- Python is said to pass all parameters by value.
- Exception:
  - If the value of the variable is a mutable object (like a list), then changes to the state of the object will be visible to the calling program.

Functions and Program Structure

- So far, functions have been used as a mechanism for reducing code duplication.
- Another reason to use functions is to make your programs more modular.
- As the algorithms you design get increasingly complex, it gets more and more difficult to make sense out of the programs.
  - It is also harder to test

Functions and Program Structure

- One way to deal with this complexity is to break an algorithm down into smaller subprograms, each of which makes sense on its own.
- We can test each of the subprogram independently
  - Write func1 to perform certain task and test it to make sure it works
  - Write func2 to perform certain task and test it to make sure it works
  - …
  - When done with all functions write code for main and test the whole program

Class Work

1)Write a function that will read a set of integers in a file and return the list. File name is passed as a parameter.

2)Write a function that adds all the number in a list and returns the result. List is passed as a parameter.

3)Write a main function that will ask:

1)User for name of input file.

2)Calls function in part 1 to read numbers from input file

3)Calls function in part 2 to calculate sum of integers

4)Displays the sum of integers