Introduction to Programming

Functions

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**

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_Example_9.py
  - (Change return statement)

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_Example_10.py

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_Example_11.py

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