Topic 6: Functions

What's a function?

How can we use functions to write better software?

Textbook

- Strongly Recommended Exercises
  - The Python Workbook: 85, 86, 98 and 103
- Recommended Exercises
  - The Python Workbook: 84, 88, 92, 93, 101 and 102
- Recommended Readings
  - Starting Out with Python
    - Chapter 3 and Chapter 6 (2nd Ed.) / Chapter 5 (3rd Ed. and 4th Ed.)

What is a Function?

- What is a function?
  - A named set of statements
  - Perform some task
- Functions:
  - May require parameters
  - May return values
- What functions have you already used?

Motivation

- Ideally, a function should
  - perform a clearly defined specific single purpose
  - hide details from the caller
  - be sufficiently small to be easily understood
  - be well documented

Defining a Function

- Creates a function for later use
  - The function does not execute until it is called
  - Function may be called many times (from different places) after it has been defined

- General form:
  - `def functionName(parameters):`
    statement(s)

Example

- Create a function that draws a music note
  - Head will be a solid oval, 20 pixels wide and 10 pixels high
  - Stem will be 50 pixels tall on the right side
  - Center the head of the note at 100, 100

Calling Our Function

- A function does not execute when it is defined
  - It must be called

- Execution for the entire program begins at the first statement outside of a function

Example

- What's the problem with our function?

- How do we fix it?

Parameters

- Allow us to provide data to a function
  - Values, called arguments, are placed in brackets after the function name when the function is called
  - Parameter variables appear in brackets after the function name in the function definition
  - Arguments are transferred to the parameter variables when the function executes
  - Arguments / parameters are positional

Terminology

- Argument (or Actual Parameter)
  - The value placed in brackets after the function name when the function is called

- Parameter (or Formal Parameter)
  - The name of the parameter variable in the function definition

Example

- Extend our note drawing function so that it takes two parameters that control the position of the note

Named Parameters

- Positional arguments assign arguments to parameter variables in the order that they occur

- Named arguments allow us to assign arguments to parameters in any order
  - Allow for optional parameters / default values for some parameters
  - May still be used in a positional manner

Example

- Extend the note drawing function so that it takes additional parameters that specify the color of the note

Default Parameter Values

- Python permits default values for parameters
  - If the function call does not supply a value then the default is used
  - If the call includes a value for that parameter then the default value is overridden

Functions Can Call Functions

- Create a second function for drawing a note
  - Head will be a solid oval, 20 pixels wide and 10 pixels high
  - Stem will be 50 pixels tall on the right side
  - Flag will be a curve

Functions Can Call Functions

Functions Can Return a Result

- Returning a result allows a value to move from the function to the location where it was called
  - Accomplished using a return statement inside the function
  - When the function is called it is often on the right side of an assignment statement

Functions Can Return a Result

- Write a function that determines the number of real roots of an equation of the form $ax^2 + bx + c = 0$

Variables & Functions

- Variables can be defined inside functions
  - A variable defined inside of a function can only be used inside that function
  - Behaves just like the variables we have used previously

Variables & Functions

- Variables can be defined outside of functions
  - Referred to as global variables
  - Can be read anywhere in the program after it is assigned a value
  - All of the constants we have created are global variables that we choose not to change
  - Use of global variables (other than as constants) is strongly discouraged

Variables & Functions

- Changing global variables
  - By default, an assignment statement inside of a function creates a new variable within that function
    - Even if a global variable with that name already exists
  - Want to change a global variable?
    - Include a global statement at the beginning of the function

Example

- Create a function that computes n-factorial

Returning Multiple Values

- What if we need to return more than one value from a function?
  - Comma separated tuple of values in return statement
  - Comma separated tuple of variables to the left of the equals sign

Scope

- Scope determines the portion of a program where a name can be used
  - Impacts functions, variables, …
- Functions
  - Functions can’t be called before they have been defined
  - Functions in other modules cannot be used until after the import statement for that module

Scope

- Variables
  - Cannot be read before they are given a value
  - Can be used from the point where they are first assigned a value until the end of the function
  - Variables created inside a function are destroyed when the function returns

Formal Parameters

- Formal parameter variables hold values passed to a function from the calling scope
  - Formal parameters are normally read
  - It is also possible to store a new value into a formal parameter
    - We don't usually do this!
    - Value of the variable will change in the called function
    - For the types we have used so far, the value will **not** change in the main program

Why Functions are Useful

- Facilitate Code Reuse
  - Write once, use many times
- Reduce Complexity
  - Low level details are hidden
  - Programmer can concentrate on higher level problems
- Ease Maintenance
  - Bugs only need to be corrected once
  - Functions can be tested separately

Comments

- Every function should begin with a comment
  - Describe the action taken by the function
  - Describe the arguments that need to be provided (if any)
  - Describe the value returned by the function (if any)

Preconditions / Postconditions

- Function comments may also describe
  - Preconditions:
    - Conditions that must be true before the function executes
    - If any precondition is not met, the function may not behave correctly
  - Postconditions:
    - Conditions that are guaranteed to be true after the function executes
    - If the function doesn't make a post-condition true then the function contains a bug that must be fixed

Example

- Addition and multiplication practice:
  - 10 random questions that involve adding or multiplying 2 integers between 1 and 10

Example

Testing

- Test each function you write individually
  - Errors are easier to find
    - Generally only need to look inside the function being tested
  - Only use the function in the rest of your program once you have tested it thoroughly

Design

- How do functions relate to top down design?
  - Use top down design to break the problem into smaller pieces
  - Each smaller piece is a good candidate for a function
  - Look at each function
    - Is it too big?
    - Does it contain repeated code?
    - Should it call other functions?

Modules

- Functions can be placed in modules to promote reuse
  - Place the functions in a different .py file
  - Import it just like math or SimpleGraphics
  - Add an if statement to prevent the main program from running in the imported file

```
if __name__ == "__main__":
  main()
```

Wrapping Up

- Functions
  - A named group of statements that perform a task
  - Allow us to break our program into separate units that each have a specific purpose
  - Ease program creation and debugging

Where Are We Going?

- Now that we can write larger programs we want to be able to manage more data
  - How can we work with many values at the same time in a reasonable way?
  - How do we read and write values in files?