CSC 120
Computer Science for the Sciences
Week 2 Lecture 3

UofT St. George

January 18, 2016

Writing Computer Programs

- What is a computer program?

➔ A *computer program* is an (ordered) set of instructions that can be executed by a *computer*

Defining a function

- Form of a Python function definition:

```
def function_name(parameters):
  <function_body>
```

`def`: a Python reserved keyword

parameters: 0 or more parameters, comma separated

function_body: 1 or more statements

Calling a Function

- Calling a function → asking Python to execute it (carry it out).
- Form of a function call: `function_name(arguments)`:
- How it is executed:
  1. Each argument is an expression. These expressions are evaluated in order. (The value of each expression is a memory address.)
  2. Those memory addresses are stored in the corresponding parameters.
  3. The body of the function is executed.

Function Design Recipe -1

- Examples of calls to your function and the expected returned values.

```
'''
>>> is_even(2)
True
>>> is_even(17)
False
'''
```

- Type contract: the types of the parameters and any return values

```
(str) -> int
(str, bool) -> NoneType
(list of int, tuple of (str,int)) -> list
```

Function Design Recipe -2

- **Header** `def is_even(value):`
- **Description:** what the function does (mention each parameter by name).

  `Return True iff value is evenly divisible by 2.`

- **Body** `return value % 2 == 0`
- **Test**

Function Design Recipe -3

```
def is_even(value):

    ''' (int) -> bool
    Return True iff value is evenly divisible by 2.
    >>> is_even(2)
    True
    >>> is_even(17)
    False
    '''

    return value % 2 == 0
```

Nested Function calls

- You can call functions from within functions:

```
def f(x):
    return g(x)
```

Or: `f(g(x))`

Variable Scope -1

- If you declare/use a variable within a function, its ***scope*** is limited to within that function.

```
def h():
    x = 3
    return x

if __name__ == "__main__":
     print(x)
```

- Output: Traceback (most recent call last):
  File "None", line 8, in <module>
  builtins.NameError: name 'x' is not defined

Variable Scope -2

```
def g():
    x = 3
    return x

if __name__ == "__main__":
     print(g())
```

- Output: 3

- **Note**: we can also define global variables, visible to all functions in our program

Return Values

- When a function is called, it returns a value.

- When writing a function, provide a line at the end that says

```
return (something)
```

→*this will tell Python to end the function and give the return value back to the line of code that called it.*

- **Note**: If there is no return statement in a function, Python will still return a value! By default it returns **None**, which is a placeholder value. None has the type **NoneType** and cannot be used in mathematical expressions

Types

```
def f(x)
     y = x + 3
     return y
```

→ What will the following calls give us?

```
>>> type( f )
<class 'function'>

>>>type( f(3) )
<class 'int'>

>>>type( f(3.5) )
<class 'float'>
```

Docstrings

- When writing a function (or module), we can provide documentation.

- This documentation consists of:

  - **annotations** specifying the types of the parameters and the return value;

  - a **comment** describing what the function does. This comment is called a Docstring , and appears in triple-quotes.

  → *The **doctest** module allows us to automatically run those examples and test whether the associated function behaved as the examples dictate.*

Doctest -1

- What will this code do?

```
import doctest
def average(x, y, z):
    '''(num, num, num) -> num
    Calculate the average of x, y, and z.
    >>> average(0, 0, 9)
    3.0
    >>> average(1, 2, 3)
    2.0
    '''
    return x + y + z / 3

doctest.testmod()
```

Doctest -2

```
Failed example:
    average(1, 2, 3)
    Expected:
    2.0
    Got:
    4.0
**********************************************************************
**********************
1 items had failures:
    1 of   2 in __main__.average
***Test Failed*** 1 failures.
```

Doctest -3

Corrected function:

```
import doctest
def average(x, y, z):
    '''(num, num, num) -> num
    Calculate the average of x, y, and z.
    >>> average(0, 0, 9)
    3.0
    >>> average(1, 2, 3)
    2.0
    '''
    return (x + y + z) / 3

doctest.testmod()
```

Basic Functions
`print -1`

- The **`print`** function, i.e., a function that prints output.

- Form of the **print** statement

```
print(list_of_items)
```

→ `list_of_items` is a comma-separated list of expressions: strings, variables, numbers, function results, etc

Basic Functions
`print -2`

```
>>>print("Hello!")
Hello!
```

→ “Hello” is a **string**. We'll learn more about strings soon.

```
>>> print("average(73, 80)")
average(73, 80)

>>> print("The average is", average(78, 90))
Traceback (most recent call last):
  File "None", line 13, in <module>
builtins.TypeError: average() missing 1 required positional argument: 'x3'
```

Problems -1

Using the Function Design Recipe, write functions for the following and test them with doctest:

1. Convert a temperature in Fahrenheit to Celsius, which is done with

$$temp_c = \frac{temp_F - 32}{1.8}$$

2. Convert a temperature in Celsius to Fahrenheit

3. Write a function that will return 1 if a given number x is odd, and 0 if it is even

Problems -2

- The number of days that the Gregorian calendar is ahead of the Julian calendar is given by

$$D = H - \frac{H}{4} - 2$$

→ where H is the hundreds digit of the year (e.g., for 1924, H is 19). The division here is integer division.

In the year 2014, the Gregorian calendar is 13 days ahead of the Julian calendar; in 1614, it was 10 days ahead