1 Docstrings

A **docstring** is a string at the beginning of a function that explains what the function does. A docstring explains *what* a function does, but not necessarily *how* it does it. Collectively, docstrings are the main vehicle for describing your program to another programmer: if I read only the "main" program (the statements at the end of the program that lie outside any function defintion) and the docstrings, I should know what your program does.

A docstring is a triple-quoted string, also known as a multiline string because the triple quotes allow the string to span more than one line. It appears just below the function header and just above the function body.

It has several components, as illustrated by the following example. This function even_sum takes two numbers and returns True if their sum is even and False otherwise. Here is the function definition complete with a docstring:

```
def even_sum(num1, num2):
    ''' (int, int) -> bool
    Return True if and only if the sum of num1 and num2 is
    evenly divisible by 2.
    >>> even_sum(2, 6)
    True
    >>> even_sum(17, 4)
    False
    >>> even_sum(17, 3)
    True
    '''
    return (num_1 + num_2) % 2 == 0
```

Docstring components:

1) Examples. The docstring includes one or two examples of calls to the function and the expected return values. It should include an example of a standard case as well as tricky cases, if there are some.

   It is not always feasible to include an example. Some cases where examples are inappropriate:

   - the function returns a complex object (such as an Image object)
   - the function does not return anything but prints or writes to file
   - the function is random and so its output is unpredictable

2) Type contract. The **type contract** describes the types of the parameters and any return values. The line (int, int) -> bool means that the function takes two int values and returns a bool. In general, the types of the inputs are enclosed in parentheses and the return type comes after the arrow. If the function does not return anything, then the return type is NoneType.

3) Description. Between the examples and type contract, add a description of what the function does. This description *must* mention each parameter by name. It should be terse and precise.

2 Design recipe: steps to writing a new functions

It is often recommended that programmers design a new function by following these steps in this order:

1) Write part of the docstring, starting with examples, then type contract.
2) Write the function header.
3) Write the description. Be sure to use each variable name in description.
4) Write the body of the function. *Notice that you only write the body after you have already worked through several examples by hand.*
5) Test the function. Run the module in IDLE and test the function by trying out the examples specified in the docstring.

3 Exercises

Solutions are presented in class and also included in the moodle version of this handout. Solve each of these problems by writing at least one function. Follow the design recipe above.

1. At a pizzeria, adults order two slices, boys order three slices, and girls order one slice. Each pizza has eight slices. Write a function called `num_pizzas` that takes three parameters representing the number of adults, boys, and girls, and returns the required number of pizzas.
2. You are driving a little too fast, and a police officer stops you. Write a function `caught_speeding` to compute the result, encoded as an int value: 0=no ticket, 1=small ticket, 2=big ticket. If speed is 60 or less, the result is 0. If speed is between 61 and 80 inclusive, the result is 1. If speed is 81 or more, the result is 2. Unless it is your birthday – on that day, your speed can be 5 higher in all cases. Your function should take two parameters: speed and a variable indicating whether or not it is your birthday.
3. Write a program that prints the numbers from 1 to 100. But for multiples of three print “Fizz” instead of the number and for the multiples of five print “Buzz.” For numbers which are multiples of both three and five print “FizzBuzz.”
   (a) Write a function `fizz_or_buzz` that takes in a single number and returns the appropriate print value for that number – it should return either a number, “Fizz,” “Buzz,” or “FizzBuzz.” So that all return values have the same type, have it return the number as a string.
   (b) In the “main” program, call this function within a for loop over the numbers from 1 to 100.
4. Write a function, `count_hi`, that takes a string and returns the number of times that the string `"hi"` appears anywhere in the given string.