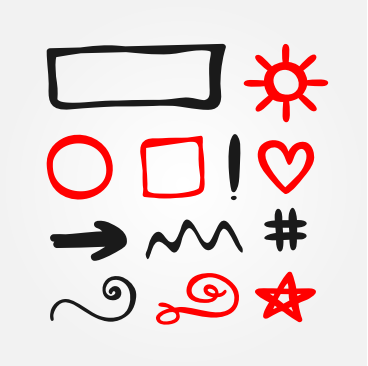

# CPSC 5061—Programming I for Business

## Worksheet 13

### Doctests

*Doctest Example*

For testing a function, **decimalToBinary**, we might run these tests in the console:

```
>>> decimalToBinary(100)
'1100100'
>>> decimalToBinary(127)
'1111111'
>>> decimalToBinary(30938201831)
'11100110100000011110111111011100111'
>>> decimalToBinary(0)
'0'
```

We can get Python to automatically run this console dialog and report to us if the outcomes are different than what we see above. We accomplish this by including the literal dialog above *inside* the docstring for the function. These are called *doctests*.

```
def decimalToBinary(n):
    """ Convert a number into its binary representation (returned
    as a string).

    >>> decimalToBinary(100)
    '1100100'
    >>> decimalToBinary(127)
    '1111111'
    >>> decimalToBinary(30938201831)
    '11100110100000011110111111011100111'
    >>> decimalToBinary(0)
    '0'
    """
    b = decimalToBinaryR(n)
    if n > 0:
        b = b[1:]
    return b
```

This allows a reader to see examples of how your function works *and* has the bonus of providing executable tests for your function. This is an alternative to putting in a test function like the author does in the Guttag textbook and is what is often done in a professional environment.

To run the tests, you have to run the python interpreter in a terminal shell (not the Python shell) which will depend on your environment. For example, on my Mac it looks like this in the ***Terminal*** application:

```
$ python3 -m doctest dec2bin.py
```

My Python code is in the file **dec2bin.py** and I've navigated to that directory before running the terminal command above. You can use the **cd** command at the terminal shell to delve into directories (like clicking on a folder in the File Finder window).

On a PCs, make sure you've installed Python for everyone and to work with the Command Prompt. You may have to reinstall it for this. Once Python is installed correctly, you have to open the ***Command Prompt*** application and use the python command:

```
> python -m doctest dec2bin.py
```

If all the tests produce the output you showed in your doctest (i.e., success), then you'll get no output. If it fails, you'll get something like what follows below. Here, for example, I haven't taken out the leading zero yet, so we expect to get a nice binary number starting with a 1, but it returns one always starting with a 0. I've artificially highlighted the interesting parts of the output, which won't happen in real life where it will all be spit out in one color.

```
**********************************************************************
File "/Users/klundeen/Documents/dec2bin.py", line 5, in dec2bin.decima
lToBinary
Failed example:
    decimalToBinary(100)
Expected:
    '1100100'
Got:
    '01100100'
**********************************************************************
File "/Users/klundeen/Documents/dec2bin.py", line 7, in dec2bin.decima
lToBinary
Failed example:
    decimalToBinary(127)
Expected:
    '1111111'
Got:
    '01111111'
```

```
**********************************************************************
File "/Users/klundeen/Documents/dec2bin.py", line 9, in dec2bin.decima
lToBinary
Failed example:
    decimalToBinary(30938201831)
Expected:
    '11100110100000011110111111011100111'
Got:
    '01110011010000001111011111011100111'
**********************************************************************
1 items had failures:
   3 of   4 in dec2bin.decimalToBinary
***Test Failed*** 3 failures.
```

### *Rewriting Functions*

Let's try this for the functions from PQ3 (see Canvas).

Once we have the doctests in place where we can, we can feel more at ease in changing the implementation of the functions and keeping the same functionality without *regressing*.

As an exercise, rewrite the functions to use lists or dictionaries where that would make for a better implementation.