Docstring and Doctests

google.github.io/styleguide/pyguide.html

Docstring for Function

```
def factorial(n):
    """Calculate the factorial of an integer input, n

     Args:
         n (int): input integer value >= 0

     Returns:
         int: the factorial value of n


    """
    return n
```

Docstring and Doctest for Function

```
def factorial(n):
    """Calculate the factorial of an integer input, n
     Args:
         n (int): input integer value >= 0
     Returns:
         int: the factorial value of n
    Examples:
       >>> factorial(5)
       120
       >>> factorial(30)
       265252859812191058636308480000000
    """
    return n
```

Docstring and Doctests

- The lines in the triple-quotes """ are called a docstring, which is a description of what the function is supposed to do.
- The lines that begin with >>> are called doctests.
- When using the Python interpreter, you write Python expressions next to >>>
- The output is printed below that line
- Doctests explain what the function does by showing actual Python code:
    - "if we input this Python code, what should the expected output be?"

Test-Driven Development: Running Doctests

```
python3 -m doctest factorial.py
```

```
**********************************************************************
File "/Users/Stretch/Desktop/factorial.py", line 8, in factorial.factorial
Failed example:
    factorial(5)
Expected:
    120
Got:
    5
**********************************************************************
File "/Users/Stretch/Desktop/factorial.py", line 10, in factorial.factorial
Failed example:
    factorial(30)
Expected:
    265252859812191058636308480000000
Got:
    30
**********************************************************************
1 items had failures:
   2 of   2 in factorial.factorial
***Test Failed*** 2 failures.
```

Fix by Providing Correct Code for Factorial

```
def factorial(n):
    """Calculate the factorial of an integer input, n
    Args:
        n (int): input integer value >= 0
    Returns:
        int: the factorial value of n
    Examples:
        >>> factorial(5)
        120
        >>> factorial(30)
        265252859812191058636308480000000
    """
    if n == 0:
        return 1
    result = 1
    for i in range(1, n+1):
        result = result * i
    return result
```

Running Doctests Again

```
python3 -m doctest factorial.py
```

```
def factorial(n):
    """Calculate the factorial of an integer input, n
    Args:
        n (int): input integer value >= 0
    Returns:
        int: the factorial value of n
    Examples:
        >>> factorial(5)
        120
        >>> factorial(30)
        265252859812191058636308480000000
    """
    if n == 0:
        return 1
    result = 1
    for i in range(1, n+1):
        result = result * i
    return result


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

could also put this at the end of all the function definitions

Running Doctests Again

```
python3 factorial.py
```

Python Exceptions

Extracted from realpython.org

Exceptions versus Syntax Errors

Syntax errors occur when the parser detects an incorrect statement. Observe the following example:

Python

```
>>> print( 0 / 0 ))
  File "<stdin>", line 1
    print( 0 / 0 ))
                   ^
SyntaxError: invalid syntax
```

The arrow indicates where the parser ran into the **syntax error**. In this example, there was one bracket too many. Remove it and run your code again:

Python

```
>>> print( 0 / 0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

This time, you ran into an **exception error**. This type of error occurs whenever syntactically correct Python code results in an error. The last line of the message indicated what type of exception error you ran into.

docs.python.org/3/library/exceptions.html

Raising an Exception

If you want to throw an error when a certain condition occurs using `raise`, you could go about it like this:

Python

```
x = 10
if x > 5:
    raise Exception('x should not exceed 5. The value of x was: {}'.format(x))
```

When you run this code, the output will be the following:

Python

```
Traceback (most recent call last):
  File "<input>", line 4, in <module>
Exception: x should not exceed 5. The value of x was: 10
```

The program comes to a halt and displays our exception to screen, offering clues about what went wrong.

Python Assert Statement

- Want to be sure that assumptions on state of computation are as expected

- Use an assert statement to raise an AssertionError exception if assumptions are not met

- Defensive programming

Assert Statement Example

```
my_list = [1, 2, 3, 4, 5, 6]
for i in range(len(my_list)):
    element = my_list.pop(0)
    print(element)
assert len(my_list) == 0, 'my_list not empty'
```

```
# after repeatedly removing the first element from
# my_list len(my_list) times, it must be empty
```

Assertions VS Exceptions

- Use assert to detect programming errors and conditions that should never occur, e.g., invariants that must be maintain at certain points of program executions
    - Warning: Assertion checks are stripped and not executed if Python is invoked with the -O or --OO option!

- Raise an exception for errors that are caused by invalid user input or other problems with the environment (e.g., network errors or unreadable files)

Revisiting Factorial()

```
def factorial(n):
    """Calculate the factorial of an integer input, n
    Args:
        n (int): input integer value >= 0
    Returns:
        int: the factorial value of n
    Raises:
        TypeError: if n is not an integer
        Exception: if n is negative
    Examples:
        >>> factorial(5)
        120
        >>> factorial(30)
        265252859812191058636308480000000
        >>> factorial(5.2)
        Traceback (most recent call last):
          ...
        TypeError: n is not integer; n value is 5.2 and n type is <class 'float'>
        >>> factorial(-6)
        Traceback (most recent call last):
          ...
        Exception: n must be greater than or equal to zero; n value is -6
    """
    if isinstance(n, int) == False:
        raise TypeError('n is not integer; n value is {} and n type is {}'.format(n, type(n)))
    if n < 0:
        raise Exception('n must be greater than or equal to zero; n value is {}'.format(n))
    if n == 0:
        return 1
    result = 1
    for i in range(1, n+1):
        result = result * i
    return result
```

What We Have Learned

- Docstring and doctests for documentation and unit testing
- Exceptions and assertions to detect error conditions during runtime