Pytest and Pydoc[1]

Robin Dawes

September 21, 2021

[1] Pytwo Pyvery Pyuseful Pytools

Explains the basics of using the Pytest and Pydoc utilities.

.

Pydoc

Pydoc is a tool for automatically generating a nicely formatted page of external documentation for a Python program. The page can be produced as an html document for easy posting.

Pydoc is now included when Python is installed. You don't need to install anything extra. Here is the official Pydoc documentation, but it is not very informative : `https://docs.python.org/3/library/pydoc.html`

Preparing Your Python Program

Pydoc operates by scanning your program source file and extracting information from specific points:

- The start of the file - here you place useful information such as
  - Version number
  - Date of most recent update to the code
  - Author name and contact information
  - Copyright and licensing information
  - Description of the purpose and method of the program
- The start of each function - here you place information such as
  - Expected type of each parameter
  - Type of each return value
  - Description of the purpose and method of the function

This level of documentation is excessive for CISC-121! You can just put the date, your name, and the description of the program.

- ☾ The main body of the program - here you don't need to place any particular information. Pydoc will identify and record the names and values of all global variables. Giving your variables informative names will pay off here.

The descriptive information at the start of the program and at the start of each function should be bracketed by **triple quote** markers like this ''' : three consecutive single quotes (you can use three consecutive double quotes too, but single quotes are usually used).

These are sometimes called DOCSTRING markers.

EXAMPLE:

```
'''
Version : 1.0
Date : 20210922
Author :  R. Dawes    dawes@queensu.ca
Copyright :  R. Dawes
Licensing :  Licensed for non-commercial use under Creative Commons licensing.

This program illustrates the use of pydoc
'''


def function_1(x, y):
   '''Parameters:
         x : integer
         y : string

     Returns:
        list of strings

     Description:
        returns the result of applying various rotations to parameter y
   '''
   result = []
   for i in range(x):
   z = i % len(y)
   result.append(y[z:]+y[:z])
   return result

def function_2(a):
   '''Parameters:
         a : integer

      Returns:
         None

      Description:
         prints the positive factors of parameter a
   '''
   factors = []
   for i in range(1, abs(a)+1):
   if a % i == 0:
      factors.append(i)
   print("The positive factors of",a,"are",factors)
```

```
# these comments will not be detected by pydoc
'''main'''


some_letters = 'abcdef'
some_numbers = [7,4,10]

for i in some_numbers:
   print(function_1(i,some_letters))
   function_2(i)
```

Running Pydoc

PYDOC IS RUN from the command line. There are several options for what it should do with the extracted information (see the full documentation) but the most useful is to have the information stored as an html document. For this we use the "-w" flag.

EXAMPLE:

Suppose the program shown above is stored in a source file called EXAMPLE.PY. The command to extract and save the documentation is

```
pydoc -w Example
```

I think this is a design flaw in the pydoc program - it should accept the source file name with or without the extension.

Notice that we DO NOT include the ".py" extension of the file name.

Running the command shown above will create a file called EXAMPLE.HTML

If the file name has spaces in it, put quotes around it.

Pytest

PYTEST IS A UTILITY that lets us confirm that the functions we write give the expected results for test cases that we specify.

When we run pytest, it looks for test files in the current directory and executes all of them.

My preference is to write a test file for each function we want to test. The test file is a Python program. To be identified as a test file, the filename must end with "_test.py"

It may also be acceptable to start the name with "test_" but I prefer the "_test.py" naming convention.

Within the test file, we define the function to be tested, and we define the tests to run. Each test is defined as a function whose name must either begin with "test_" or end with "_test". The test function makes an ASSERTION about what the return value from the function being tested should be, given specific input.

It is possible to combine multiple tests into a single test function. I prefer to keep them separate.

When pytest finds a test file, it looks inside to find all the test functions, and reports how many it found. It then executes all the tests and checks to see if the assertions are true. Pytest reports the details of any tests that failed. If all tests are passed, pytest just reports that.

Example:

```
def count_memberships(list_of_lists, target):
   '''takes a list of lists and an object (any type)

      returns the number of lists the object is in

      returns -1 if there is a problem with the list of lists'''
   count = 0
   if type(list_of_lists) is not list:
      return -1
   else:
      for lst in list_of_lists:
         if type(lst) is not list:
            return -1
         elif target in lst:
            count += 1
      return count



list1 = [1, 2, 4, 8]
list2 = ['a', 'b', 8, 'dog', 'cat', False]
list3 = ['salt', 'pepper', 'oregano', "za'atar", "basil", "turmeric"]
list4 = ['basil', False, list1]
list5 = [8, 8, 8, 8]
some_lists = [list1, list2, list3, list4, list5]

def test_count_memberhips_1():
   '''test 'normal' operation'''
   assert count_memberships(some_lists, 8) == 3

def test_count_memberships_2():
   '''test when one of the lists is empty'''
   assert count_memberships([[],[1]], 1) == 1

def test_count_memberships_3():
   '''test when list_of_lists is empty'''
   assert count_memberships([],1) == 0

def test_count_memberships_4():
   '''test when no element of list_of_lists is a list'''
   assert count_memberships(list5, 8) == -1
```

```
def test_count_memberships_5():
   '''test when at least one element of list_of_lists is not a list'''
   assert count_memberships([list1,4], 4) == -1

def test_count_memberships_6():
   '''test when the target value is a string'''
   assert count_memberships(some_lists, "basil") == 2

def test_count_memberships_7():
   '''test when list_of_lists is just a value'''
   assert count_memberships(1, 4) == -1
```

It is impossible (or at least very difficult) to test every eventuality. Basically we have to try to imagine all the things that could go wrong, and then test to confirm that they don't go wrong (or that our function handles them properly).

One of the huge benefits of writing tests like this is that it often suggests improvements that we could make. For example, when writing the tests just shown it occurred to me that rather than "give up" if list_of_lists contains some non-list elements, the function should just ignore those and check the elements of list_of_lists that *are* lists.

Running Pytest

Pytest is a Python module that is not part of the standard Python distribution. We need to install it, usually with a terminal window command such as

```
pip3 install pytest
```

The easiest way (in my opinion) to run pytest is to create a two-line Python program called something like "Run pytest.py"

Your preference may be different, and that's ok!

The content of this Python program is this:

```
import pytest

pytest.main(["--capture=sys"])
```

Executing this program starts the process described above: pytest finds all the test files, looks in them for test functions, runs the tests, and reports the results.