```
import unittest

class TestVariables(unittest.TestCase):
    def test_variables(self):
        """
        This is a test method you will need to fill in the content
        where it says and then run it from the proper directory using
        terminal (mac) or the command prompt (windows):
        (ie ``python exer01_variables.py``).
        or run from IDLE or another IDE.
        If it doesn't complain, then you pass- If it complains- fix it!
        """

        # create the variable ``diffusion_rate`` and assign it a float value
        # of 6.0
        # ***********************************************
        diffusion_rate = 6.
        self.assertEqual(diffusion_rate, 6.)
        self.assert_(isinstance(diffusion_rate, float))

        # assign ``cohort_size`` to an integer value of 84
        # ***********************************************
        cohort_size = 84
        self.assertEqual(cohort_size, 84)
        self.assert_(isinstance(cohort_size, int))

        # create a variable ‘species_name‘ ,
        # and assign it to ’Pieza kake’
        # ****************************************
        species_name = 'Pieza kake'
        self.assertEqual(species_name, "Pieza kake")
        self.assertTrue(isinstance(species_name, str))

# from terminal or DOS shell
if __name__ == '__main__':
    unittest.main()
```

```
import unittest

class TestStrings(unittest.TestCase):
    def test_strings(self):
        """
        A basic introduction to strings
        """
        str_dir =  ['__add__', '__class__', '__contains__', '__delattr__',
'__doc__', '__eq__', '__format__',
                    '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__',
                    '__init__', '__le__', '__len__', '__lt__', '__mod__', '__mul__',
'__ne__', '__new__', '__reduce__',
                    '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
'__setattr__', '__sizeof__', '__str__',
                    '__subclasshook__', '_formatter_field_name_split',
'_formatter_parser', 'capitalize', 'center', 'count',
                    'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format',
'index', 'isalnum', 'isalpha', 'isdigit', 'islower',
                    'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
                    'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
                    'zfill']
        # Create the variable ``hola`` and assign 'hello world'
        #***********************************************
        hola = 'hello world'
        self.assertEqual(hola, """hello world""")
        self.assert_(isinstance(hola, str))

        # Create a string, ``hola2`` that equals ``hola`` multiplied by 2
        #***********************************************
        hola2 = hola * 2
        self.assertEqual(hola2, 'hello worldhello world')

        # Triple quoted strings (with ''' or """) allow embedded of
        # both single and double quotes.
        # Create a triple quoted string
        # ``darwin_quote2`` that has the following content:
        # Darwin said, "There is grandeur in this view of things."
        #***********************************************
        darwin_quote2 = 'Darwin said, "There is grandeur in this view of things."'
        self.assertEqual(darwin_quote2, 'Darwin said, "There is grandeur in this
view of things."')

        # Assign the method names of a string to a variable ``string_methods``
        # use ``dir()`` to list them
        #***********************************************
        string_methods = dir(str)
        self.assertEqual(string_methods, str_dir)

        # Create a variable where_is_gra that has holds the index of the
        # substring "gra" in the string darwin_quote2.  (Find a string method to
        # figure it out)
        #***********************************************
        where_is_gra = darwin_quote2.find("gra")
        self.assertEqual(where_is_gra, 23)


if __name__ == '__main__':
    unittest.main()
```

```
import unittest

class TestLists(unittest.TestCase):
    def test_lists(self):
        """
        A basic introduction to lists
        """
        # Create the variable ``bird_list`` and assign to an empty list
        # ************************************************
        bird_list = []
        self.assertEquals(bird_list, [])

        # Append 'American redstart' and 'Arctic tern' to ``bird_list``
        # ************************************************
        bird_list.append('American redstart')
        bird_list.append('Arctic tern')
        self.assertEquals(bird_list, ['American redstart','Arctic tern' ])

        # Sort ``bird_list`` in reverse order (use help())
        # ************************************************
        bird_list.sort(reverse=True)
        self.assertEquals(bird_list,  ['Arctic tern', 'American redstart'])

        # ``extend`` the list ``bird_list`` with ['Northern parula', 'Hooded
warbler']
        # ************************************************
        bird_list.extend(['Northern parula', 'Hooded warbler'])
        self.assertEquals(bird_list, ['Arctic tern', 'American redstart', 'Northern
parula', 'Hooded warbler'])

        # create a variable ``warbler_id`` with the index of 'Hooded warbler' in
        # ``bird_list`` using list methods.
        # ************************************************
        warbler_id = bird_list.index('Hooded warbler')
        self.assertEquals(warbler_id, 3)


if __name__ == '__main__':
    unittest.main()
```

```
import unittest

class TestDicts(unittest.TestCase):
    def test_dicts(self):
        """
        A basic introduction to dictionaries
        """
        # Create the variable ``common_to_latin`` and assign to an empty dict
        #************************************************
        common_to_latin={}
        self.assertEquals(common_to_latin, {})

        # map the string 'Capuchin monkey' to an empty list
        #************************************************
        common_to_latin['Capuchin monkey']=[]
        self.assertEquals(common_to_latin['Capuchin monkey'], [])
        self.assert_('Capuchin monkey' in common_to_latin)

        # map the string 'Squirrel monkey' to the list ['Saimiri sciureus', 'Saimiri
oerstedi']
        #************************************************
        common_to_latin['Squirrel monkey']=['Saimiri sciureus', 'Saimiri oerstedi']
        self.assertEquals(common_to_latin['Squirrel monkey'], ['Saimiri sciureus',
'Saimiri oerstedi'])

        # map the string 'Capuchin monkey' to a list with one element ['Cebus
capucinus']
        #************************************************
        common_to_latin['Capuchin monkey']=['Cebus capucinus']
        self.assertEquals(common_to_latin['Capuchin monkey'], ['Cebus capucinus'])

        # use ``in`` to see if 'Howler monkey' is there.
        # assign the results to variable ``howler``
        #************************************************
        howler = 'Howler monkey' in common_to_latin
        self.assertEquals(howler, False)

        # use ``del`` to remove 'Capuchin monkey'
        #************************************************
        del common_to_latin['Capuchin monkey']
        self.assert_('Capuchin monkey' not in common_to_latin)

if __name__ == '__main__':
    unittest.main()
```

```
import types
import unittest

class TestFunctions(unittest.TestCase):
    def test_functions(self):
        """
        A basic introduction to functions
        """
        # Define a function called ``add_2`` that returns 2 more than
        # what is passed into it.  Normally function are global to a
        # module, but they can also be defined in the scope of another
        # function.  You can either put it below, or outside of the
        # TestFunctions class.
        # ================================
        def add_2(num):
            return num +2
        self.assert_(isinstance(add_2, types.FunctionType))
        self.assertEquals(add_2(4), 6)

        # Write a function ``add_3`` that has the following docstring:
        # "Adds 3 to the input"
        # ================================
        def add_3(num):
            "Adds 3 to the input"
            return num + 3
        self.assertEquals(add_3.__doc__, 'Adds 3 to the input')

        # Default Parameters.
        # Write a function ``mul_n`` that takes one or two parameters (the second
parameter named ``x``).
        # If it has 2 parameters it multiplies them.  If it takes one
        # parameter, it multiplies it by 5
        # ================================
        def mul_n(num,x=5):
            return num*x

        self.assertEquals(mul_n(5,1), 5)
        self.assertEquals(mul_n(5), 25)
        self.assertEquals(mul_n.func_defaults, (5,))



if __name__ == '__main__':
    unittest.main()
```

```
import unittest

class TestLoops(unittest.TestCase):
    def test_loops(self):
        # Using ``range``
        # Write variable ``nests`` that holds 0 to 5 (use the ``range`` function)
        # ================================
        nests = range(6)
        self.assertEquals(nests, [0,1,2,3,4,5])

        # ``range`` 2
        # Create variable ``sample_id`` that holds values from 3 to 11
        # ================================
        sample_id = range(3,12)
        self.assertEquals(sample_id, [3,4,5,6,7,8,9,10,11])

        # Write a function ``even`` that takes a list of
        # number and returns a list of even numbers
        # ================================
        def even(some_list):
            return_list = []
            for item in some_list:
                if item%2==0:
                    return_list.append(item)
            return return_list

        self.assertEquals(even(nests), [0, 2, 4])
        self.assertEquals(even(sample_id), [4, 6, 8, 10])

        # Write a function ``even_index`` that takes a list
        # of numbers and returns those that are in an even
        # index position (hint: enumerate)
        # ================================
        def even_index(some_list):
            return_list = []
            for index,value in enumerate(some_list):
                if index%2==0:
                    return_list.append(value)
            return return_list
        self.assertEquals(even_index(nests), [0, 2, 4])
        self.assertEquals(even_index(sample_id), [3, 5, 7, 9, 11])


if __name__ == '__main__':
    unittest.main()
```

exer07_fileIO_key.py

1 Read a file
1.1 Read StateoftheUnion.txt
1.2 Output each word on its own line

2 Sum a file
2.1 Read IntegerFile.txt, which has many numbers on one line.
2.2 Calculate the sum, and the mean. Hint, the int() function
turns a string into a number.

3 File copier
3.1 Copy http://www.ubertool.org/index.html to a file on your local disk.

```
import unittest

class TestClass(unittest.TestCase):
    def test_classes(self):
        # Create a class called ``Coyote``
        # Accept a name in the constructor
        # Create a coyote object called wilee with the name 'Wile E. Coyote'
        # ================================
        class Coyote(object):
            def __init__(self, name):
                self.name=name
        wilee = Coyote('Wile E. Coyote')
        self.assert_(isinstance(wilee, Coyote))

        # Add a method ``speak`` to a Coyote class that
        # accepts a string and returns:
        # '${name} said, "${string}"
        # where "${string} is an argument passed into the method
        # ================================
        class Coyote(object):
            def __init__(self, name):
                self.name=name
            def speak(self, message):
                return self.name + ' said, "' + message + '"'
        wilee = Coyote('Wile E. Coyote')
        self.assertEquals(wilee.speak('Being a genius certainly has its
advantages.'),
                          'Wile E. Coyote said, "Being a genius certainly has its
advantages."')

        # Subclass - create a subclass ``RoadRunner``
        # of ``Coyote`` that has the same constructor
        # but ``beep`` returns:
        # ${name} said, "${string}"
        # ================================
        class RoadRunner(Coyote):
            def beep(self, beeps):
                return self.name + ' said, "' + beeps + '"'
        rr = RoadRunner("Road Runner")
        self.assertEquals(rr.beep('Beep. Beep.'), 'Road Runner said, "Beep. Beep."')


if __name__ == '__main__':
    unittest.main()
```