CSC1015F Assignment 8

Testing and Arrays

Assignment Instructions

This assignment involves constructing tests, assembling automated 'doctest' test scripts for statement and path coverage, and constructing Python programs that manipulate lists, dictionaries and strings using arrays. You can use the following built-in functions in Python: `map( )`, `math.sqrt( )`, `split( )`.

READ APPENDIX A on pages 8 to 10 before attempting Question 1 and 2.

NOTE Your solutions to this assignment will be evaluated for correctness and for the following qualities:

- Documentation
  - Use of comments at the top of your code to identify program purpose, author and date.
  - Use of comments within your code to explain each non-obvious functional unit of code.
- General style/readability
  - The use of meaningful names for variables and functions.
- Algorithmic qualities
  - Efficiency, simplicity

These criteria will be manually assessed by a tutor and commented upon. In this assignments, up to 10 marks will be deducted for deficiencies.

Question 1 [15 marks]

This question concerns devising a set of tests for a Python function that cumulatively achieve path coverage.

The Amathuba page for this assignment provides a Python module called '`numberutil.py`' as an attachment. The module contains a function called '`aswords`'. The function accepts an integer (0-999) as a parameter and returns the English language equivalent.

For example, `aswords(905)` returns the string `'Nine hundred and five'`.

Your task:

1. Develop a set of 8 test cases that achieve ***path coverage.***
2. Code your tests as a `doctest` script suitable for execution within Wing IDE.
3. Save your `doctest` script as '`testnumberutil.py`'.

NOTE: make sure the docstring in your script contains a blank line before the closing """. (The automarker requires it.)

NOTE: the `aswords()` function is believed to be error free.

Question 2 [15 marks]

This question concerns devising a set of tests for a Python function that cumulatively achieve ***path coverage.***

The Amathuba page for this assignment provides a Python module called '`palindrome.py`' as an attachment. The module contains a function called '`is_palindrome()`'. The purpose of this function is to accept a string value as a parameter and determine whether it is a palindrome or not.

A palindrome is a word, phrase, number, or other sequence of symbols or elements, whose meaning may be interpreted the same way in either forward or reverse direction (see Wikipedia).

Examples of palindromes are (note that white spaces are ignored – see the last examples):

```
989
a
aba
racecar
a santa at nasa
```

Examples of strings that are not palindromes (note case sensitivity – last two examples):

```
34563
ab
Racecar
A Santa at NASA
```

Your task:

4. Develop a set of 5 test cases that achieve ***path coverage.***
5. Code your tests as a `doctest` script suitable for execution within Wing IDE (like the `testchecker.py` module described in the appendix).
6. Save your doctest script as '`testpalindrome.py`'.

NOTE: make sure the docstring in your script contains a blank line before the closing """. (The automarker requires it.)

NOTE: the `palindrome()` function is believed to be error free.

Question 3 [15 marks]

Write a program, `gridio.py`, that allows a user to input integer values and query a `2`–dimensional array of size `9 X 9`. Your program should then ask the user for a pair of coordinates, `(x, y)`, separated by a space and return the value at the position specified by the given coordinates.

For instance, `0 3` should return the value `7` (the value at `row one, column four` – bearing in mind that array indices start at `zero`).

Assume that each integer is a single digit from `1 - 9`. Enter `-1` for either coordinate to end the program.

Sample IO *(The input from the user is shown in* ***bold*** *font – do not program this):*

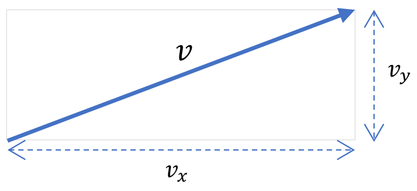
```
Enter an array:
359716482
867345912
413928675
398574126
546281739
172639548
984163257
621857394
735492861

Enter coordinates:
0 3
Value = 7
Enter coordinates:
5 5
Value = 9
Enter coordinates:
8 8
Value = 1
Enter coordinates:
-1 -1
DONE
```

Question 4 [45 marks]

Write a program called '`vectormath.py`' to do basic vector calculations in 2 dimensions: addition, dot product and cross product.

This exercise concerns the construction of a Python program to represent a vector in two dimensions. A vector has magnitude and direction. The following is a graphical depiction of a vector, v:



A two-component vector can be represented by its horizontal and vertical components i.e. $v = (v_x, v_y)$.

There are a number of operations that can be performed on vectors:

Operation	*Description*
Obtain magnitude	The magnitude of a vector $v = (v_x, v_y)$ is $\sqrt{{v_x}^2 + {v_y}^2}$. Print the output to **one decimal place**.
Vector add	Given two vectors $v = (v_x, v_y)$ and $v' = (v'_x, v'_y)$, the result of adding them together is the vector $(v_x + v'_x, v_y + v'_y)$.
Scalar Multiply	The multiplication of a vector $v = (v_x, v_y)$ by a value m produces a vector $v' = (mv_x, mv_y)$.

Dot product	Given two vectors $v = (v_x, v_y)$, and $v' = (v'_x, v'_y)$, their dot product is the value of $(v_x \times v'_x + v_y \times v'_y)$.
Cross product	Given two vectors $v = (v_x, v_y, v_z)$, $v' = (v'_x, v'_y, v'_z)$ their cross product is the new vector $v'' = (v_y v'_z - v_z v'_y, v_z v'_x - v_x v'_z, v_x v'_y - v_y v'_x)$.

NOTE: For the vector cross product, we need two 3-component vectors of the form $v = (v_x, v_y, v_z)$.

Sample IO *(The input from the user is shown in* ***bold*** *font – do not program this):*

(See Appendix B (page 11) for a complete Sample I/O covering all options)

```
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
2
Enter the components of the first vector separated by spaces:
4 5
Enter the components of the second vector separated by spaces:
2 1 5
Error: Vectors must have the same length.
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
2
Enter the components of the first vector separated by spaces:
4 2
Enter the components of the second vector separated by spaces:
7 1
Sum of the vectors is: (11, 3)
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
```

```
6
Exiting...
```

Sample IO *(The input from the user is shown in **bold** font – do not program this)*:

```
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
5
Enter the components of the first 3-dimensional vector
separated by spaces:
4 1 5
Enter the components of the second 3-dimensional vector
separated by spaces:
3 5 1
Cross product of the vectors is: (-24, 11, 17)
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
x
Invalid choice. Please choose a valid option.
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
6
Exiting...
```

Question 5 [10 marks]

Write a program called ‘`histogram.py`’ that takes in a list of marks (separated by spaces) and outputs a histogram representation of the marks according to the mark categories at UCT:

- fail < 50%
- 50% ≤ 3rd < 60%
- 60% ≤ lower 2nd < 70%
- 70% ≤ upper 2nd < 75%
- 75% ≤ first

From the given marks, count the number of marks that satisfy the conditions of each category. For example, for the following list of marks: 32 67 54 90 77, there is one fail, one 3rd, one lower 2nd, no upper 2nds and two firsts. Then print out horizontal bars, using "X", that correspond to the counters, along with some hard-coded axes and labels to represent a histogram.
You should use an array/list to store the counters.

Sample IO *(The input from the user is shown in* ***bold*** *font – do not program this):*

```
Enter a space-separated list of marks:
88 90 75 76 92 78 98 100 81 70 70 73 74 72 73 72 66 67 60 63 65 62
64 60 52 50 55 56 54 56 57 58 59 50 42 40 40 34 23 43 42 32
1 |XXXXXXXXX
2+|XXXXXXX
2-|XXXXXXXX
3 |XXXXXXXXXX
F |XXXXXXXX
```

Submission

Create and submit a Zip file called ‘`ABCXYZ123.zip`’ (where ABCXYZ123 is YOUR student number) containing `testnumberutil.py, testpalindrome.py, gridio.py, vectormath.py` and `histogram.py`.

Appendix A: Testing using the doctest module

The doctest module utilises the convenience of the Python interpreter shell to define, run, and check tests in a repeatable manner.
Say, for instance, we have a function called 'check' in a module called 'checker.py':

```
1 # checker.py
2
3 def check(a, b):
4     result=0
5     if a<25:
6         result=result+3
7     if b<25:
8         result=result+2
9     return result
```

We've numbered the lines for convenience.
The function is supposed to behave as follows: if *a* is less than 25 then add 1 to the *result*. If *b* is less than 25 then add 2 to the *result*. Possible outcomes are 0, 1, 2, or 3.
For the sake of realism, that there's an error in the code. Line 6 should be 'result=result+1'.)
Here are a set of tests devised to achieve path coverage:

test #	*Path(lines executed)*	*Inputs (a,b)*	*Expected Output*
1	3, 4, 5, 6, 7, 8, 9	(20, 20)	3
2	3, 4, 5, 6, 7, 9	(20, 30)	1
3	3, 4, 5, 7, 8, 9	(30, 20)	2
4	3, 4, 5, 7, 9	(30, 30)	0

Note that we analyse the code to identify paths and select inputs that will cause that path to be followed. Given the inputs for a path, we study the function *specification* (the description of its intended behaviour) to determine the expected output.

Here is a doctest script for running these tests:

```
>>> import checker
>>> checker.check(20, 20)
3
>>> checker.check(20, 30)
1
>>> checker.check(30, 20)
2
>>> checker.check(30, 30)
0
```

The text looks much like the transcript of an interactive session in the Python shell.
A line beginning with '>>>' is a statement that doctest must execute. If the statement is supposed to produce a result, then the expected value is given on the following line e.g. 'checker.check(20, 20)' is expected to produce the value 3.
NOTE: there must be a space between a '>>>' and the following statement.
It is possible to save the script just as a text file. However, because we're using Wing IDE, it's more convenient to package it up in a Python module (available on the Amathuba assignment page):

```
# testchecker.py
"""
>>> import checker
>>> checker.check(20, 20)
3
>>> checker.check(20, 30)
1
>>> checker.check(30, 20)
2
>>> checker.check(30, 30)
0

"""
import doctest
doctest.testmod(verbose=True)
```

The script is enclosed within a Python docstring. The docstring begins with three double quotation marks and ends with three double quotation marks.

NOTE: the blank line before the closing quotation marks is essential.

Following the docstring is an instruction to import the `doctest` module, followed by an instruction to run the '`testmod()`' function. (The parameter '`verbose=True`' ensures that the function prints what it's doing.)

If we save this as say, '`testchecker.py`', and run it, here's the result:

```
 Trying:
     import checker
 Expecting nothing
 ok
 Trying:
     checker.check(20, 20)
 Expecting:
     3

**********************************************************************
 File "testchecker.py", line 3, in __main__
 Failed example:
     checker.check(20, 20)
 Expected:
     3
 Got:
     5
 Trying:
     checker.check(20, 30)
 Expecting:
     1

**********************************************************************
 File "testchecker.py", line 5, in __main__
 Failed example:
     checker.check(20, 30)
 Expected:
     1
 Got:
     3
 Trying:
     checker.check(30, 20)
 Expecting:
     2
 ok
 Trying:
     checker.check(30, 30)
 Expecting:
     0
 ok

**************************************************************
********
 1 items had failures:
    2 of   5 in __main__
 5 tests in 1 items.
 3 passed and 2 failed.
 ***Test Failed*** 2 failures.
```

As might be expected, we have two failures because of the bug at line 6.
What happens is that `doctest.testmod()` locates the docstring, and looks for lines within it that begin with '>>>'. Each that it finds, it executes. At each step it states what it is executing and what it expects the outcome to be. If all is well, ok, otherwise it reports on the failure.
The last section contains a summary of events.
If we correct the bug at line 6 in the check function and run the test script again, we get the following:

```
 Trying:
     import checker
 Expecting nothing
 ok
 Trying:
     checker.check(20, 20)
 Expecting:
     3
 ok
 Trying:
     checker.check(20, 30)
 Expecting:
     1
 ok
 Trying:
     checker.check(30, 20)
 Expecting:
     2
 ok
```

```
Trying:
    checker.check(30, 30)
Expecting:
    0
ok
1 items passed all tests:
   5 tests in __main__
5 tests in 1 items.
5 passed and 0 failed.
Test passed.
```

APPENDIX B: Complete Sample I/O for Question 4

The sample I/O below covers all options for Vector Maths as asked in question 4.

Sample IO *(The input from the user is shown in **bold** font – do not program this):*

```
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
1
Enter the components of the vector separated by spaces:
7 3
Magnitude of the vector is: 7.6
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
2
Enter the components of the first vector separated by spaces:
4 5
Enter the components of the second vector separated by spaces:
2 1 5
Error: Vectors must have the same length.
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
2
Enter the components of the first vector separated by spaces:
4 2
Enter the components of the second vector separated by spaces:
7 1
Sum of the vectors is: (11, 3)
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
```

```
3
Enter the components of the vector separated by spaces:
5 2
Enter the scalar:
3
Scalar multiplication of the vector is: (15, 6)
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
4
Enter the components of the first vector separated by spaces:
4 4
Enter the components of the second vector separated by spaces:
1 6 2
Error: Vectors must have the same length.
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
4
Enter the components of the first vector separated by spaces:
5 1
Enter the components of the second vector separated by spaces:
3 3
Dot product of the vectors is: 18
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
5
Enter the components of the first 3-dimensional vector
separated by spaces:
3 4
Enter the components of the second 3-dimensional vector
separated by spaces:
2 1 5
Error: Vectors must have the same length and 3-dimensional.
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
```

```
5. Cross product of two 3-dimensional vectors
6. Exit
5
Enter the components of the first 3-dimensional vector
separated by spaces:
5 2 3
Enter the components of the second 3-dimensional vector
separated by spaces:
4 3 5
Cross product of the vectors is: (1, -13, 7)
Choose an option:
1. Magnitude of a vector
2. Vector addition
3. Scalar multiplication
4. Dot product of two vectors
5. Cross product of two 3-dimensional vectors
6. Exit
6
Exiting...
```