All of these activities are completed on **IDLE in addition to PowerPoint/Word for documentation**

Programming Practical Problems

Task 1- Pythagoras examples

Mrs Ghalichi-Tabriz

Note to read:

- Please note that the evidence of your documentation for each task (similar to NEA work) **must be in detailed** following the guidance given to you in lessons and on the website below:

- http://www.computing.outwood.com/NEA/python/index.html

Task 1

Skills Required:

- *Using variables, operators, inputs, outputs and assignments*
- *Using sequences, selection and iteration*
- *Using count controlled loops (for) and condition controlled loops (while)*
- *Using different types of data, i.e. integer, string, float and Boolean*
- *Basic string manipulation*
- ~~*Basic file handling operations*~~
- ~~*Using lists*~~
- *Using subroutines*
- Create a program to solve Pythagoras problems.
- The program should calculate the length of the hypotenuse, given the length of the two short sides. It should also be able to calculate the length of a missing short side given the length of the hypotenuse and a known short side.
- The program should have a menu system with 3 options, including one to quit the program, and should repeat the program until the user chooses to quit.
- Analyse the requirements for this system and design, develop, test and evaluate a program for solving Pythagoras problems.
-

Task 1

Extension

- Add an extra menu option to run as a Pythagoras tester. The extra menu option should run a subroutine that randomly generates 10 questions and asks the user to input the answers – telling the user if they got each answer right or wrong.

Further Extension

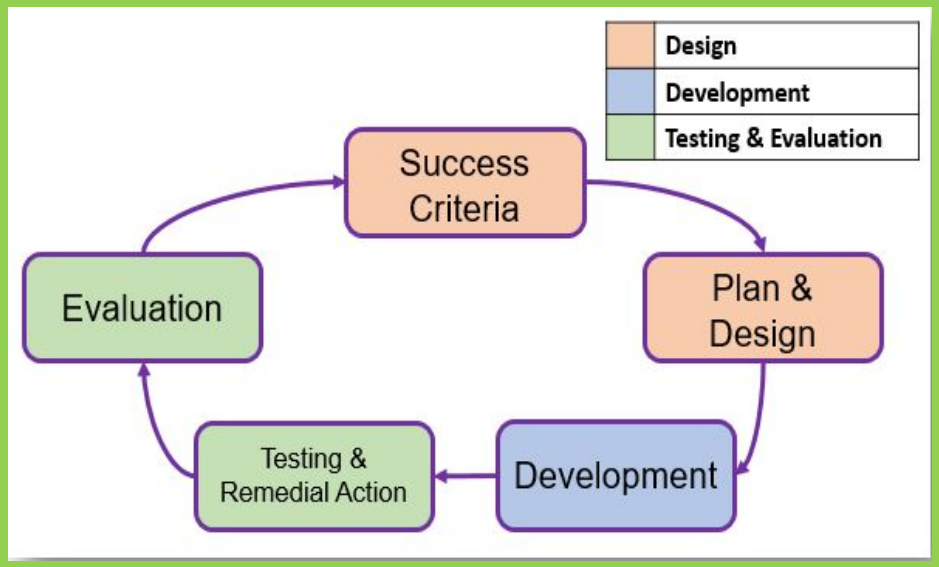
- Save the results of each test, including the user's name, in a file.
- Allow the user to display the names and scores of the three highest scorers.

Guidance

NOTE

- You should have completed a detailed **evidence** of **your work**, program solution and relevant documentations (e.g. flowchart /pseudocode planning, design, testing and etc)
- Refer to **RESOURCE BANK** website to help you with the **documentation** of your work

RESOURCE BANK- EVIDENCING



You will need to cover all the following stages always in your **documentation:**

- **Analysis**
- **Design**
- **Development**
- **Testing**
- **Evaluation**

You can find example of how to show the evidence for the above sections in the resource bank > evidencing section.

Evidencing

- You need to also present detailed evidence of the following stages in your documentation on **Word** or **PowerPoint**

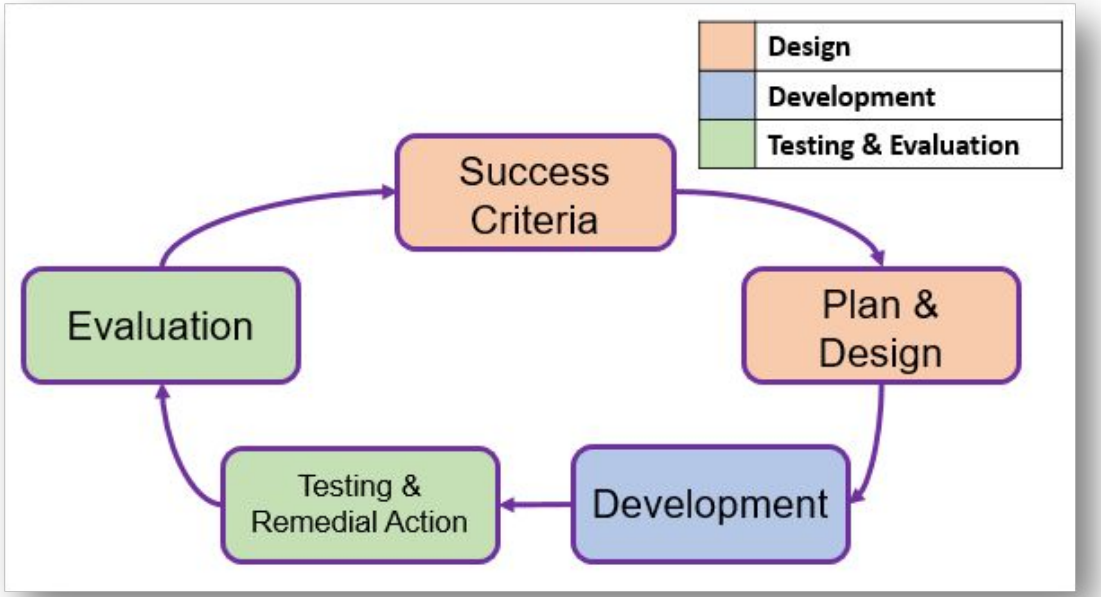
- **Analysis**
- **Design**
- **Development**
- **Testing**
- **Evaluation**

Analysis Section

This should include:

- an explanation of the task that you need to complete, saying what it will need to do
- a break down of the main task, listing the smaller sub-tasks

Process of Success



Each sub-task identified in the analysis should follow the process of success and be broken down into **three** key sections, these are **design**, **development** and **testing**.

Design Section

This should include:

- **Success Criteria** - a list of success criteria for the requirement.
- **Approach to be Used** - this should include an explanation of the programming techniques you will use to complete the task.
- **IPOD table** - showing the inputs, processes, outputs and decisions that will be needed.
- **Variables to be Used** - this should include the data type and what each one will be used for.
- **Validation to be Used** - this should explain the different validation that could/will be added to the solution.
- **Pseudocode** & **Flowchart** of the solution to be created.
- **Test plan** - this should detail the tests that will be carried out after development.

IPOD Table

- The IPOD table will help break down the task by identifying the **inputs**, **processes**, **outputs** and **decisions**. You could write it as a list or use a table like the example below:

Input(s)	Process(es)	Output(s)	Decision(s)
Staff Name Staff Monthly sales (x12)	Calculate the total sales (monthly sales added together) Calculate the average sales (total divided by 12)	If they get a bonus or not	Whether they have entered 12 monthly sales Whether the average is enough to get a bonus

Variables to be Used

- Create a table that lists all of the **variable names**, **their data types** and **what they will be used for**, an example is shown below:

Variable Name	Data Type	Purpose
Name	String	To store the name of the staff member that has been entered
Total	Integer	To store the running total of sales as each month is entered by the user.
Sales	Integer	To store the sales for each individual month as they are entered by the user.
Average	Decimal	To store the average sales after the calculation has been made.

Validation to be Used

- Make **a list of the different validation** you will try to include in your solution to prevent invalid input. You can either make this as a set of bullet points or use a table like the example below:

Validation Type	Where	Reason
Presence check	Sales	To make sure that each time the number of sales for each month is entered rather than having blank entries.
Presence check	Name	To make sure that a staff member's name is entered
Format check	Sales	To make sure that the sales are a numerical value

Guidance

Pseudocode

- You need to develop pseudocode to show the design of your algorithm as covered in your lessons, click below to see the **pseudocode guide** supplied by OCR to help you.

- http://www.computing.outwood.com/NEA/python/evidencing.html

- **Click above** and **scroll down** to be able to read this information

Variables

Variables are assigned using the = operator.
x=3 name="Bob"
A variable is declared the first time a value is assigned. It assumes the data type of the value it is given.
Variables declared inside a function or procedure are local to that subroutine.
Variables in the main program can be made global with the keyword global. global userid = 123

Casting

Variables can be typecast using the int str and float functions.
str(3) returns "3"
int ("3.14") returns 3
float ("3.14") returns 3.14

Outputting to Screen

print(string/variable)
Example print("hello")

Taking Input from User

variable=input(prompt to user)
Example name=input("Please enter your name")

Selection

Selection will be carried out with if/else
if entry=="a" then print("You selected A") elseif entry=="b" then print("You selected B") else print("Unrecognised selection") endif
switch/case
switch entry: case "A": print("You selected A") case "B": print("You selected B") default: print("Unrecognised selection") endswitch

Iteration – Count Controlled

for i=0 to 7 print("Hello") next i
Will print hello 8 times (0-7 inclusive).

Iteration – Condition Controlled

while answer!="computer" answer=input("What is the password?") endwhile
do answer=input("What is the password?") until answer=="computer"

String Handling

To get the length of a string: stringname.length
To get a substring: stringname.subString(startingPosition, numberOfCharacters)
NB The string will start with the 0th character.
Example
someText="Computer Science" print(someText.length) print(someText.substring(3,3))
Will display 16
put

Arrays

Arrays will be 0 based and declared with the keyword array.
array names[5] names[0]="Ahmad" names[1]="Ben" names[2]="Catherine" names[3]="Dana" names[4]="Elijah" print(names[3])
Example of 2D array:
Array board[8,8] board[0,0]="rook"

Reading to and Writing from Files

To open a file to read from openRead is used and readLine to return a line of text from the file. The following program makes x the first line of sample.txt
myFile = openRead("sample.txt") x = myFile.readLine() myFile.close()
endOfFile() is used to determine the end of the file. The following program will print out the contents of sample.txt
myFile = openRead("sample.txt") while NOT myFile.endOfFile() print(myFile.readLine()) endwhile myFile.close()
To open a file to write to openWrite is used and writeLine to add a line of text to the file. In the program below hello world is made the contents of sample.txt (any previous contents are overwritten).
myFile = openWrite("sample.txt") myFile.writeLine("Hello World") myFile.close()

Subroutines

function triple(number) return number*3 endfunction
Called from main program
y=triple(7) procedure greeting(name) print("hello"+name) endprocedure
Called from main program
greeting("Hamish")

Flowchart

- You also need to **create a flowchart** when designing your program, use the symbols below to do this:

Symbol	Description
Start Stop	This shapes indicates the start or end of a flowchart
Process	A rectangular box represents a process, this is doing something. E.g. total = num1+num2
Input/ Output	A parallelogram represents input or output. E.g. Input num1 Output total
Decision	A diamond shape represents a decision, YES or NO e.g. is it a weekday?
Subroutine	This can be used to represent a link to a smaller sub-task from a main flowchart

Development Section

This should include:

- **explanation** showing how you developed the code bit by bit to get to your finished solution.
- **screenshots and evidence of problems** you had and changes you made to solve them.
- **screenshots and evidence of any tests** you carried out as you developed the program to test that sections were working.
- **full annotation of the final code** with screenshots of it.

Testing & Evaluation Section

This should include:

- **Test plan** has been fully completed with screenshots showing the tests being carried out.
- If any tests failed, these **should be corrected** and **re-tested**.

- **Evaluation of your solution** against the success criteria, explaining how you have met each one.
- What did you find difficult while working on this project
- What did you learn while working on this project
- If you were to do this again, what would you improve or do differently.

Test Plan

See this website for more guidance of good testing:
http://www.computing.outwood.com/NEA/python/testing.html

- Use the table layout below when creating your test plan. The test plan should have the **Actual Outcome** and **Pass/Fail** left blank until it is completed in the testing section at the end.
-

Test Number	Test Type	Test Data	Reason	Expected Outcome	Actual Outcome	Pass/ Fail?
1						
2						
3						
4						

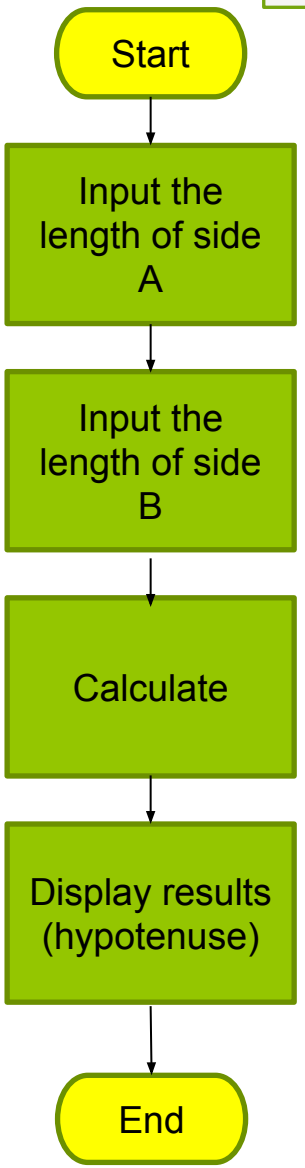
Show screenshot evidence of run time

```
Enter an email address: a@b.c
VALID
>>>
```

```
Enter an email address: ab23@f45.d3
VALID
>>>
```

Flowchart-V1

Plan



Version 1- Code

Final Screenshot of your Final code and testing evidence in run time

```
##########
#
#    File:    pythag.py
#    Author:  pgOnline
#    Date:    January 2017
#    Notes:   A program for solving Pythagoras problems
#             Version 1 - Only deals with finding the hypotenuse
#
##########

def findHypotenuse():
    sideOne = float(input("Enter the length of side A: "))
    sideTwo = float(input("Enter the length of side B: "))
    hypotenuse = (sideOne ** 2 + sideTwo ** 2) ** (1/2)
    print("The hypotenuse is: " + str(hypotenuse))

findHypotenuse()
```

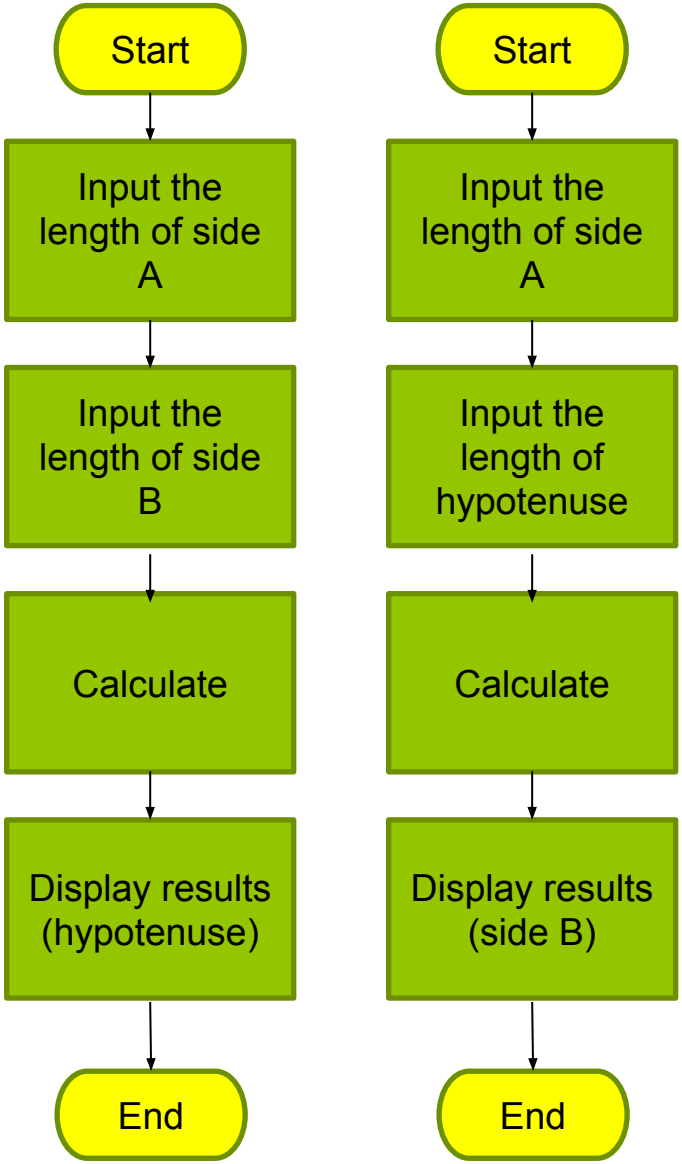
You will need to provide **screenshot of each stage** of your code writing **with evidence of testing** and fixing error in addition to final code and successfully running of it at the end.

You must show **various input you used** to test your program, e.g. integer, float, text, etc.

```
>>>
Enter the length of side A: 3
Enter the length of side B: 4
The hypotenuse is: 5.0
>>> |
```

Flowchart-V

Plan



Version 2- Code

Final Screenshot of your Final code and testing evidence in run time

pythag v2.py - K:\Computer Science\Admin\ASGH\A\Lessons\Year 10s\PG Online\Practical programming skills in Python\Python Skills Practical problems\Pythagoras ex

File Edit Format Run Options Window Help

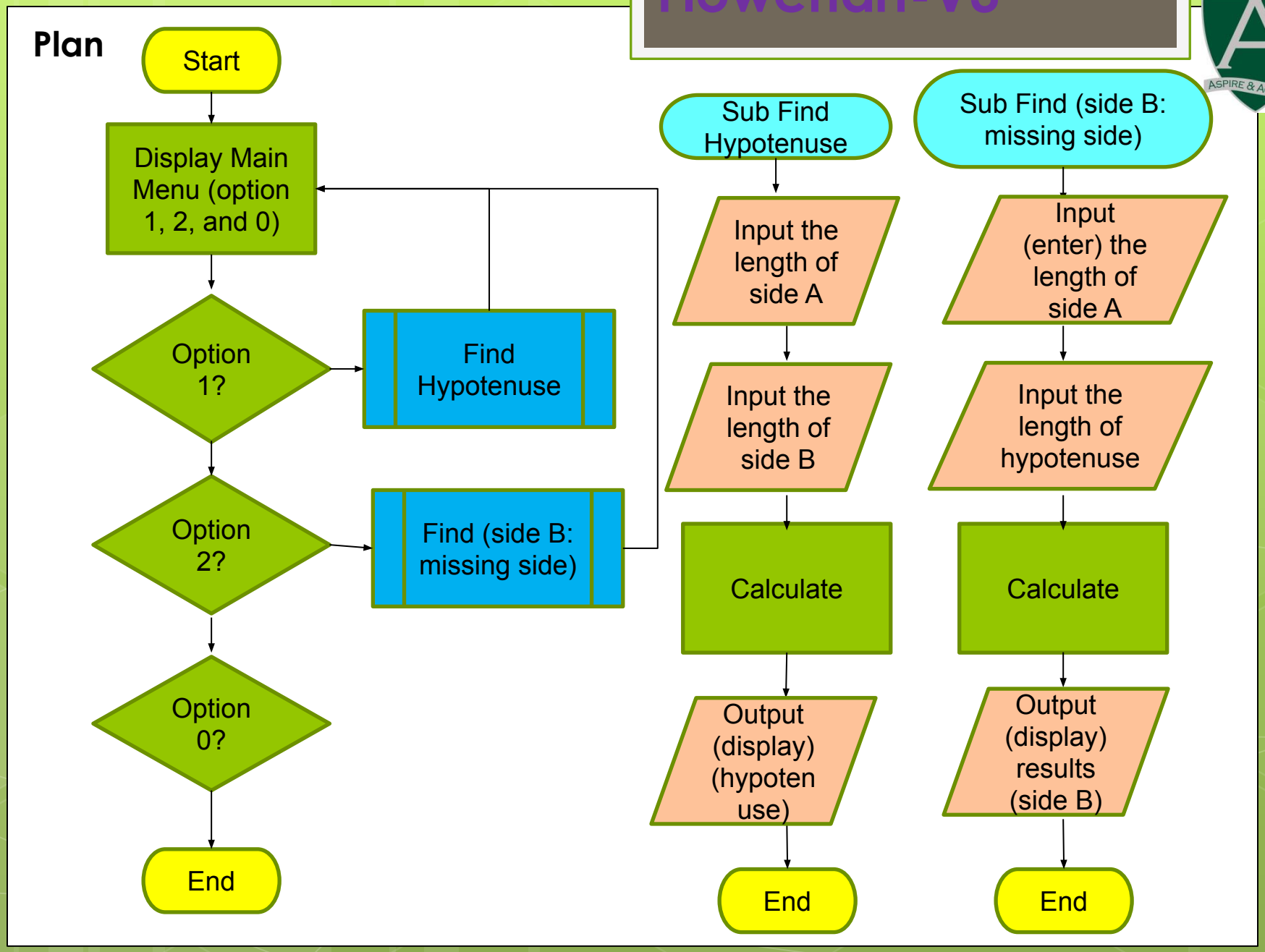
```
##########
#
#    File:   pythag.py
#    Author: pgOnline
#    Date:   January 2017
#    Notes:  A program for solving Pythagoras problems
#            Version 2 - Now finds a missing short side
#
##########

def findHypotenuse():
    sideOne = float(input("Enter the length of side A: "))
    sideTwo = float(input("Enter the length of side B: "))
    hypotenuse = (sideOne ** 2 + sideTwo ** 2) ** (1/2)
    print("The hypotenuse is: " + str(hypotenuse))

def findSide():
    sideOne = float(input("Enter the length of side A: "))
    hypotenuse = float(input("Enter the length of the hypotenuse: "))
    sideTwo = (hypotenuse ** 2 - sideOne ** 2) ** (1/2)
    print("The missing side is is: " + str(sideTwo))

findSide()
```

```
>>>
Enter the length of side A: 3.4
Enter the length of the hypotenuse: 5.5
The missing side is is: 4.323193264243457
>>>
```

You will need to provide **screenshot of each stage** of your code writing **with evidence of testing** and fixing error in addition to final code and successfully running of it at the end.

You must show **various input you used** to test your program, e.g. integer, float, text, etc.

Flowchart-V3



Version 3- Code

Final Screenshot of your Final code

```
##########
#
#   File:   pythag.py
#   Author: pgOnline
#   Date:   January 2017
#   Notes:  A program for solving Pythagoras problems
#           Version 3 - Added a menu
#
##########

# Procedure to find the long side of a right angled triangle
def findHypotenuse():
    # Get the two short sides
    sideOne = float(input("Enter the length of side A: "))
    sideTwo = float(input("Enter the length of side B: "))
    # Add the squares of the two short sides, then square root
    hypotenuse = (sideOne ** 2 + sideTwo ** 2) ** (1/2)
    print("The hypotenuse is: " + str(hypotenuse))

# Procedure to find a short side of a right angled triangle
def findSide():
    # Get the known short side an the hypotenuse
    sideOne = float(input("Enter the length of side A: "))
    hypotenuse = float(input("Enter the length of the hypotenuse: "))
    # Calculate the length of the missing side
    sideTwo = (hypotenuse ** 2 - sideOne ** 2) ** (1/2)
    print("The missing side is is: " + str(sideTwo))

# Procedure to display the menu
def displayMenu():
    print("")
    print("==========")
    print("Pythagoras Problem Sovler")
    print("")
    print("1. Find the hypotenuse")
    print("2. Find a short side")
    print("0. Quit")
    print("")

# MAIN PROGRAM
```

Procedures

You will need to provide **screenshot of each stage** of your code writing **with evidence of testing** and fixing error in addition to final code and successfully running of it at the end.

You must show **various input you used** to test your program, e.g. integer, float, text, etc.

```
# MAIN PROGRAM

# Set the choice so that the while loop will run
choice = ""

# Repeat until the user chooses 0
while choice != 0:
    # Display menu and validate the user's choice
    displayMenu()
    choice = int(input("Choose an option: "))
    while choice not in (0, 1, 2):
        print("Invalid option")
        displayMenu()
        choice = int(input("Choose an option: "))

    # Run the procedure chosen by the user
    if choice == 1:
        findHypotenuse()
    elif choice == 2:
        findSide()
```

Version 3- Code

Final Screenshot of your final testing evidence to day it works in run time

```
>>>

==========
Pythagoras Problem Sovler

1. Find the hypotenuse
2. Find a short side
0. Quit

Choose an option: 2
Enter the length of side A: 3.2
Enter the length of the hypotenuse: 4.2
The missing side is is: 2.7202941017470885

==========
Pythagoras Problem Sovler

1. Find the hypotenuse
2. Find a short side
0. Quit

Choose an option: 1
Enter the length of side A: 3
Enter the length of side B: 4
The hypotenuse is: 5.0

==========
Pythagoras Problem Sovler

1. Find the hypotenuse
2. Find a short side
0. Quit

Choose an option: 0
>>>
```

You will need to provide **screenshot of each stage** of your code writing **with evidence of testing** and fixing error in addition to final code and successfully running of it at the end.

You must show **various input you used** to test your program, e.g. integer, float, text, etc.

Testing must have a **TEST TABLE** as well.

See:

http://www.computing.outwood.com/NEA/python/testing.html