CFRS 772: Forensic Artifact Extraction
Homework Project 1
Rachel B. Gully

1. Confirm Python on the lab system
- Start:Search:python
  or
- Install Python 3.x if needed (e.g., personal computer)

2. Open terminal
- run "IDLE (Python 3.x GUI)" or similar
- you do not need to run this as Administrator

3. Check python version installed:
- IDLE window (before the Python command prompt) should start with "Python 3.x.x"
or
```
>>> import sys
>>> print(sys.version)
```

4. Set command history in IDLE:
- Options:ConfigureIDLE:Keys
  - Scroll down and select "history-next"
    - Click on "Get new keys"
      - Scroll down to "Down Arrow" and click OK
  - Name the Custom Key Set "CFRS772"
  - Repeat for "history-previous" with "Up Arrow"
  - Apply, OK

5. Basic interaction:
- Which of the below work? (put y/n in the blanks)

```
>>> print("hello")       _Y__
>>> print('hello')       _Y__
>>> print("hello')       _N__
>>> print(`hello`)       _N__
>>> print "hello"        _N__
>>> print 'hello'        _N__
```

- Try the following commands: (put result in the blanks)

```
>>> print(x)             __NameError: name 'x' is not defined__
>>> x=10
>>> print(x)             __10__
>>> print(X)             __NameError: name 'X' is not defined__
>>> y=3
>>> print(y)             __3__
>>> print(x+y)           _13__
>>> print(x*y)           _30__
>>> print(x/y)           __3.3333333333333335__
>>> print(x-y)           __7__
>>> print(x//y)          __3__
>>> print(x%y)           __1__
>>> z=x+y
>>> print(z)             _13__
```

6. Basic script creation:
- In IDLE, go to File:New File
- Enter the following in the Editor Window:

```
x=10
y=3
print(x+y)
```

- File:SaveAs
  test1.py (note the location)
- Run:RunModule or F5

check IDLE console for output

- In console
  Try:
  >>> test1.py
  Did this work? → **YES**
  Try:
  >>> exec(open("test1.py").read())

```
================ RESTART: C:/Users/vselabs/Downloads/test1.py ================
13
>>> exec(open("test1.py").read())
13
>>>
```

- At Windows command line
  Start:Search:cmd
  Run "cmd.exe" (not as Administrator)
  Change to directory c:\python3x
  C:\Python3x>python <path>/test1.py

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64>python.exe C:\Users\vselabs\Downloads\test1.py
13

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64>
```

7. Comments and conditionals:

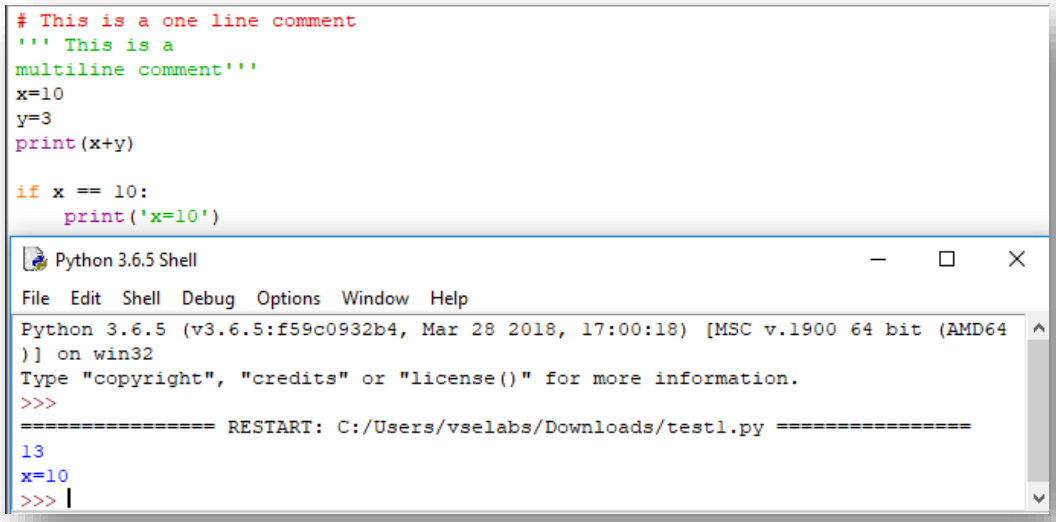
- In IDLE, File:Open
  test1.py
- Add comments:

```
# This is a one-line comment
''' This is a
multiline comment'''
```

- Add an IF statement to the end of test1.py:

```
// The if statement contents, i.e., the stuff after the
// "if" statement, *must* be indented (use tab key),
// like this:
if x==10:
      print ('x=10')
```

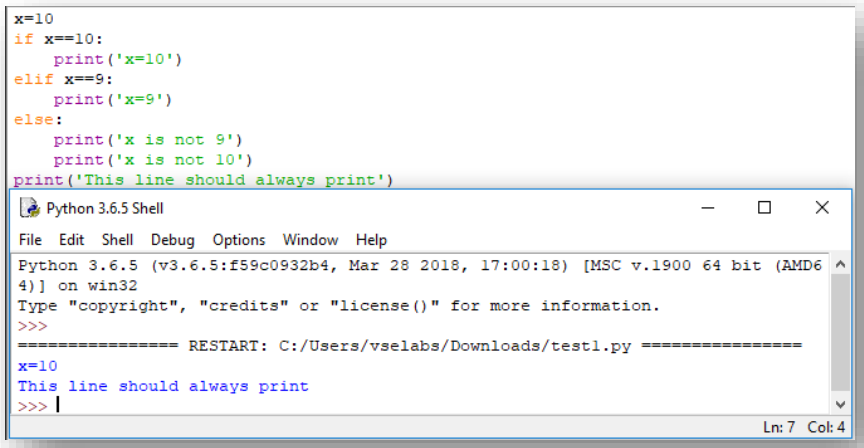
- Run the script



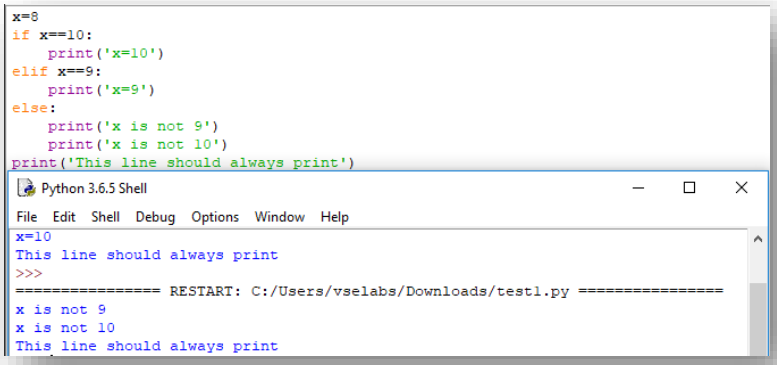
- Edit the script to have only this content:

```
x=10
if x==10:
      print ('x=10')
elif x==9:
      print ('x=9')
else:
      print ('x is not 9')
      print ('x is not 10')
print ('This line should always print')
```

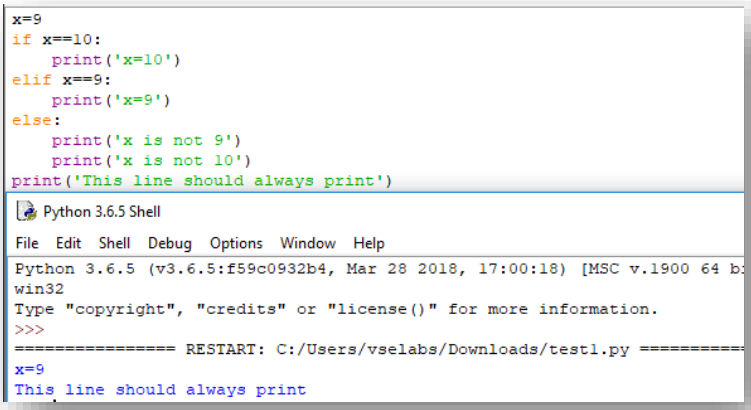
- Run the script



- Change x=8 and run the script



- Change x=9 and run the script



8. Explore context:

- Start a new IDLE session

```
>>> print(x)
>>> x=7
>>> print(x)
```

Run the test1.py script (it should have "x=9" in it)

```
>>> print(x)
```

- What does this result tell you? ____**9**_____

```
>>> x=6
>>> print(x)
```

Remove the "x=..." line from your test1.py script

Run the test1.py script

Why didn't that work? ____**x isn't defined**_____

```
>>> x=10
>>> print(x)
```

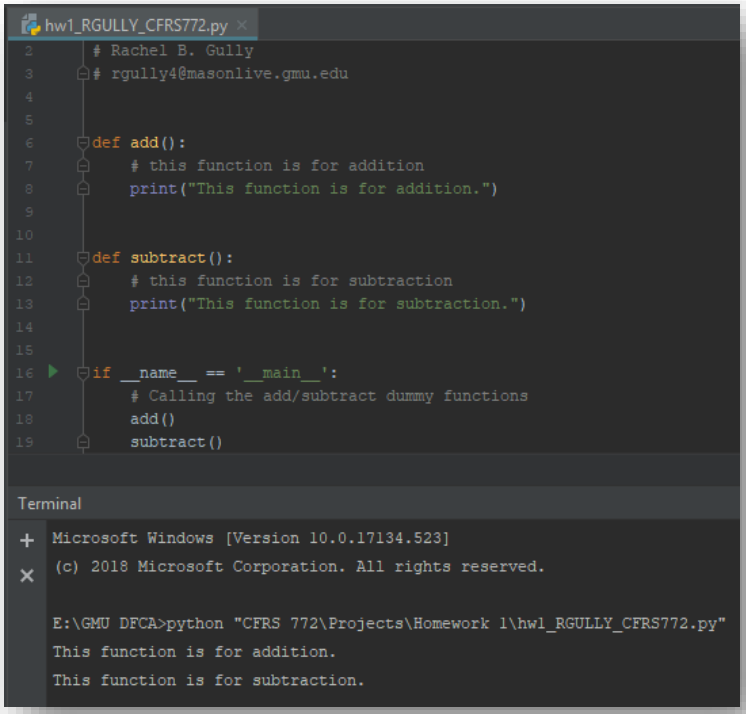
Run the test1.py script

- Why didn't that work? **_It does work because x is now defined, and assigned a value of 10_**

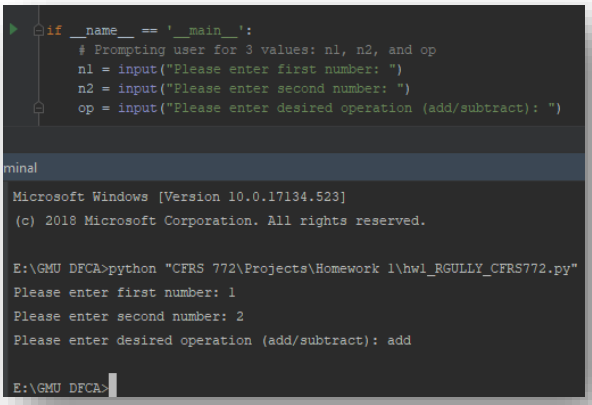
9. Create a Python script

Create a new file (script) in IDLE; name the file hw1_yourname.py

Create a simple program with a main routine that calls two functions, add() and subtract(); for each function, include comments about its purpose but just print a short message and return



Prompt for user input in the main routine; create three prompts: number 1, number 2, and operation



Add code (IF block) to the main routine to check the value of the 3rd user argument and call the appropriate function with the first two user arguments

```python
if __name__ == '__main__':
    # Prompting user for 3 values: n1, n2, and operation
    n1 = int(input("Please enter first number: "))
    n2 = int(input("Please enter second number: "))
    op = input("Please enter desired operation (add/subtract): ")

    # Check user's desired operation and call the appropriate function.
    if 'add' in op:
        add(n1, n2)
    else:
        subtract(n1, n2)
```

Modify the two functions so they take two arguments (the two numbers entered by the user) and compute and return the sum or difference; add try-except structures to catch errors

```python
def add(value1, value2):
    """
    This function returns value1 + value2
    :param value1:
    :param value2:
    :return:
    """
    try:
        return int(value1) + int(value2)
    except Exception as e:
        print("An error has occurred: " + str(e.__class__.__name__) + ". Please enter new numbers.")


def subtract(value1, value2):
    """
    This function returns value1 - value2
    :param value1:
    :param value2:
    :return value1-value2:
    """
    try:
        return int(value1) - int(value2)
    except Exception as e:
        print("An error has occurred: " + str(e.__class__.__name__) + ". Please enter new numbers.")
```

Modify the main routine to print out the user input and result

```python
if __name__ == '__main__':
    # Prompting user for 3 values: n1, n2, and operation
    n1 = input("Please enter first number: ")
    n2 = input("Please enter second number: ")
    op = input("Please enter desired operation (add/subtract): ")

    # Check user's desired operation and call the appropriate function.
    if 'add' in op:
        result = add(n1, n2)
        if type(result) is not None:
            print('Result: {0}'.format(result))
    elif 'subtract' in op:
        result = subtract(n1, n2)
        if type(result) is not None:
            print('Result: {0}'.format(result))
    else:
        print("Operation not recognized.")
```

Run a few different examples, including ones that generate errors

```
E:\GMU DFCA>python "CFRS 772\Projects\Homework 1\hw1_RGULLY_CFRS772.py"
Please enter first number: 11
Please enter second number: aa
Please enter desired operation (add/subtract): add
An error has occurred: ValueError. Please enter new numbers.

E:\GMU DFCA>python "CFRS 772\Projects\Homework 1\hw1_RGULLY_CFRS772.py"
Please enter first number: aa
Please enter second number: bb
Please enter desired operation (add/subtract): subtract
An error has occurred: ValueError. Please enter new numbers.

E:\GMU DFCA>python "CFRS 772\Projects\Homework 1\hw1_RGULLY_CFRS772.py"
Please enter first number: 11
Please enter second number: 22
Please enter desired operation (add/subtract): add
Result: 33

E:\GMU DFCA>python "CFRS 772\Projects\Homework 1\hw1_RGULLY_CFRS772.py"
Please enter first number: 11
Please enter second number: 22
Please enter desired operation (add/subtract): subtract
Result: -11
```

Add code to use the debugger; run your script in debugging mode, step through your program, and check values at various points

```
E:\GMU DFCA>python "CFRS 772\Projects\Homework 1\hw1_RGULLY_CFRS772.py"
> e:\gmu dfca\cfrs 772\projects\homework 1\hw1_rgully_cfrs772.py(8)<module>()
-> def add(value1, value2):
(Pdb) break 41
Breakpoint 1 at e:\gmu dfca\cfrs 772\projects\homework 1\hw1_rgully_cfrs772.py:41
(Pdb) continue
Please enter first number: 11
Please enter second number: 22
Please enter desired operation (add/subtract): add
> e:\gmu dfca\cfrs 772\projects\homework 1\hw1_rgully_cfrs772.py(41)<module>()
-> if 'add' in op:
(Pdb) n1
'11'
(Pdb) n2
'22'
(Pdb) op
'add'
(Pdb) break 43
Breakpoint 2 at e:\gmu dfca\cfrs 772\projects\homework 1\hw1_rgully_cfrs772.py:43
(Pdb) continue
> e:\gmu dfca\cfrs 772\projects\homework 1\hw1_rgully_cfrs772.py(43)<module>()
-> if result is not None:
(Pdb) result
33
(Pdb)
```

On BlackBoard, submit (1) a single PDF document with all screenshots (numbered) and answers to questions (all the blanks, also numbered; you can insert answers into this document and save it as PDF for submission), and (2) your final code for part 9 with debugging disabled.