

CIS 61 :: Lab 01 - Expressions and Functions

Student Name:

Instructions:

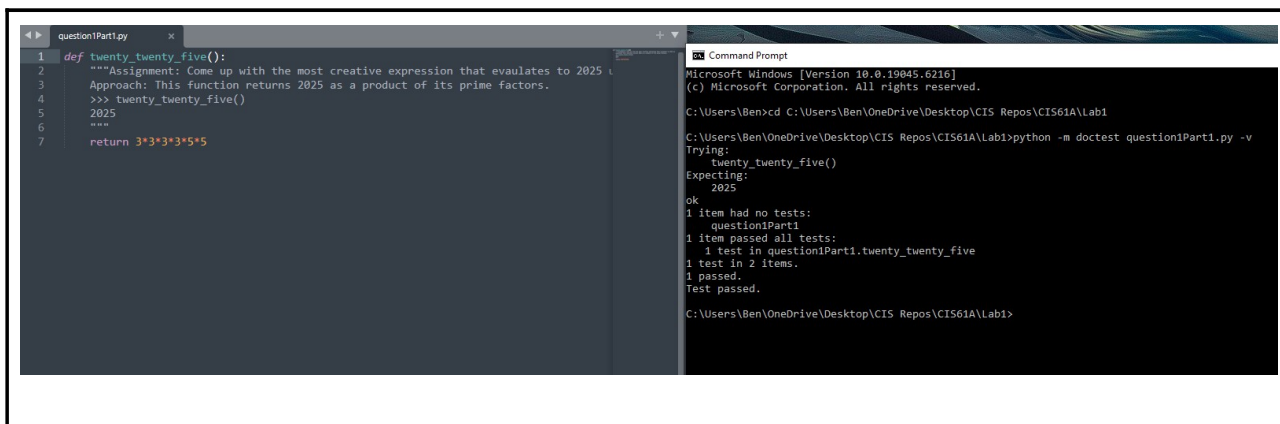
1. Make a copy of the assignment template. Go to File => Make a copy (or download as a Word file).
2. Attach Snipping Photos for each question.
3. Place your name in the Title of each Assignment
 - a. For Example: CIS 61 - Lab 01 - Expression and Names - Irfan O.
4. **Submission:** When done, go to File -> Download as -> Microsoft Word and then upload the file to Canvas.

Lab 1 - Expressions and Functions

Instructions: Use Sublime text editor to write your code and use Python shell to execute the below programs. Attach Snipping photos of **your source code** and **executions of the code in Python shell**.

Question 1: Twenty-Twenty-Five

Part 1: Come up with the most creative expression that evaluates to 2025, using only numbers and the `[+, *, -]` operators. Use integers. Do not use the round function..



The screenshot shows a Python script named `question1Part1.py` in a text editor. The script defines a function `twenty_twenty_five()` that returns the expression `3*3*3*3*5*5`. A docstring explains the assignment and approach. The script is then executed in a Command Prompt using `python -m doctest question1Part1.py -v`. The output shows that the function returns 2025 and passes all tests.

```
1 def twenty_twenty_five():
2     """Assignment: Come up with the most creative expression that evaluates to 2025.
3     Approach: This function returns 2025 as a product of its prime factors.
4     >>> twenty_twenty_five()
5     2025
6     """
7     return 3*3*3*3*5*5
```

```
Microsoft Windows [Version 10.0.19045.6216]
(c) Microsoft Corporation. All rights reserved.

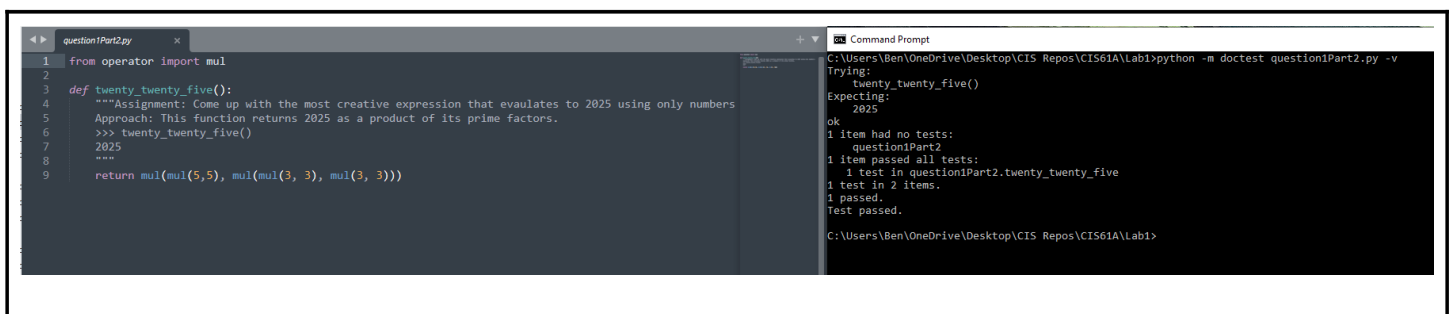
C:\Users\Ben>cd C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1

C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>python -m doctest question1Part1.py -v
Trying:
    twenty_twenty_five()
Expecting:
    2025
ok
1 item had no tests:
    question1Part1
1 item passed all tests:
   1 test in question1Part1.twenty_twenty_five
1 test in 2 items.
1 passed.
Test passed.

C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>
```

Part 2: Try to rewrite the same expression, this time entirely with call expressions. You can just use the terminal as well.

(using function calls: add, mul, sub, etc... You can use `from operators import add, mul, sub`)



The screenshot shows a Python script named `question1Part2.py` in a text editor. The script imports `mul` from the `operator` module and defines a function `twenty_twenty_five()` that returns the expression `mul(mul(5,5), mul(mul(3, 3), mul(3, 3)))`. A docstring explains the assignment and approach. The script is then executed in a Command Prompt using `python -m doctest question1Part2.py -v`. The output shows that the function returns 2025 and passes all tests.

```
1 from operator import mul
2
3 def twenty_twenty_five():
4     """Assignment: Come up with the most creative expression that evaluates to 2025 using only numbers
5     Approach: This function returns 2025 as a product of its prime factors.
6     >>> twenty_twenty_five()
7     2025
8     """
9     return mul(mul(5,5), mul(mul(3, 3), mul(3, 3)))
```

```
C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>python -m doctest question1Part2.py -v
Trying:
    twenty_twenty_five()
Expecting:
    2025
ok
1 item had no tests:
    question1Part2
1 item passed all tests:
   1 test in question1Part2.twenty_twenty_five
1 test in 2 items.
1 passed.
Test passed.

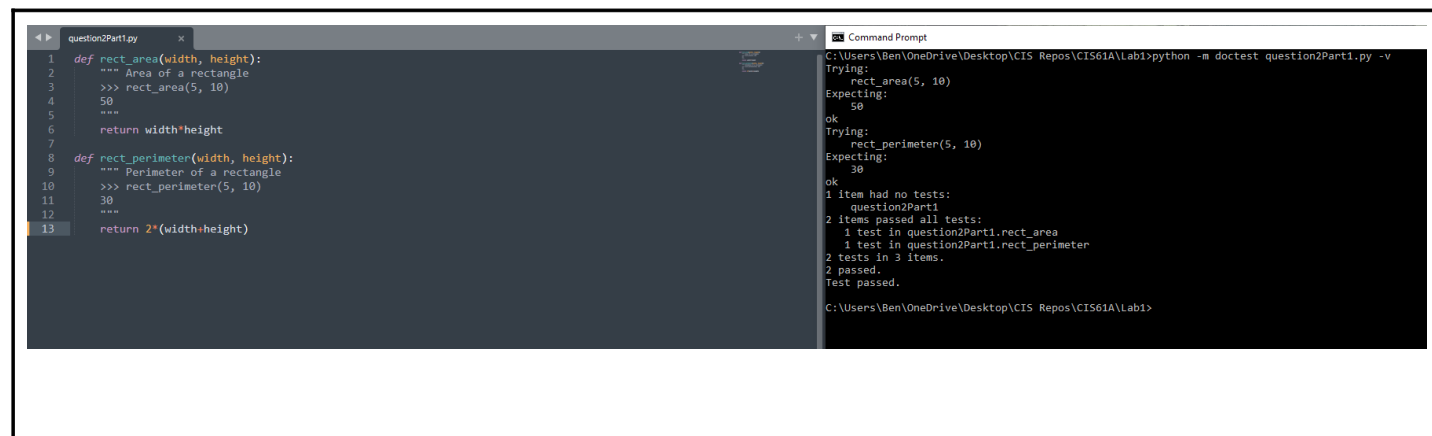
C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>
```

Question 2: Area

Part 1: Write definitions for these functions:

`rect_area(width, height)` returns the area of a rectangle giving its dimensions.

`rect_peri (width, height)` returns the perimeter of a rectangle giving its dimensions.



The screenshot shows a Python IDE window titled 'question2Part1.py' with the following code:

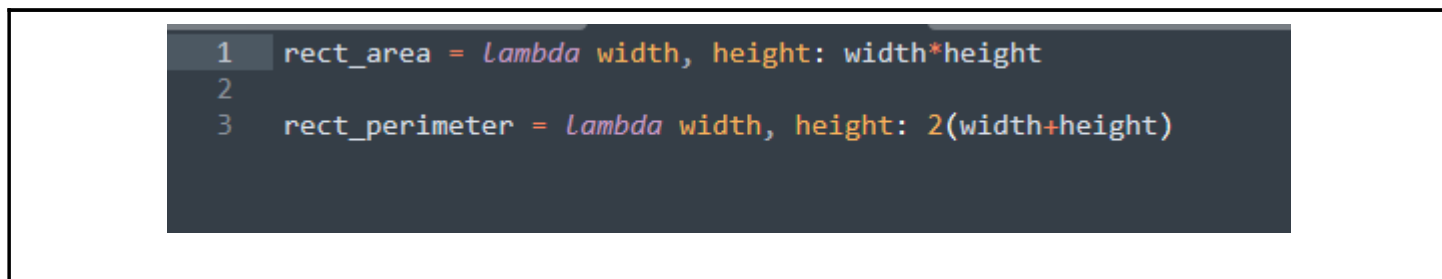
```
1 def rect_area(width, height):
2     """ Area of a rectangle
3     """
4     >>> rect_area(5, 10)
5     50
6     """
7     return width*height
8
9 def rect_perimeter(width, height):
10    """ Perimeter of a rectangle
11    """
12    >>> rect_perimeter(5, 10)
13    30
14    """
15    return 2*(width+height)
```

Next to it is a Command Prompt window showing the execution of doctest:

```
C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>python -m doctest question2Part1.py -v
Trying:
    rect_area(5, 10)
Expecting:
    50
ok
Trying:
    rect_perimeter(5, 10)
Expecting:
    30
ok
1 item had no tests:
    question2Part1
2 items passed all tests:
   1 test in question2Part1.rect_area
   1 test in question2Part1.rect_perimeter
2 tests in 3 items.
2 passed.
Test passed.

C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>
```

Part 2: Rewrite the above functions with lambda expressions and assign them to respective names. You can just use a Python shell and take the screenshot of the code.



The screenshot shows a Python shell with the following code:

```
1 rect_area = lambda width, height: width*height
2
3 rect_perimeter = lambda width, height: 2*(width+height)
```

Question 3: Rain or Shine

Part 1: Alfonso will only wear a jacket outside if it is below 60 degrees or it is raining.

Write a function that takes in the current temperature and a boolean value telling if it is raining and it should return True if Alfonso will wear a jacket and False otherwise.

Try solving this problem with a single line of code.

```
1 def wears_jacket(temp, raining):
2     """
3     >>> wears_jacket(90, False)
4     False
5     >>> wears_jacket(40, False)
6     True
7     >>> wears_jacket(100, True)
8     True
9     """
10    return temp < 60 or raining
```

```
C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>python -m doctest question3Part1.py -v
Trying:
    wears_jacket(90, False)
Expecting:
    False
ok
Trying:
    wears_jacket(40, False)
Expecting:
    True
ok
Trying:
    wears_jacket(100, True)
Expecting:
    True
ok
1 item had no tests:
    question3Part1
1 item passed all tests:
    3 tests in question3Part1.wears_jacket
3 tests in 2 items.
3 passed.
Test passed.
C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>
```

Note that it should either return **True** or **False** based on a single condition, whose truthiness value will also be either True or False.

Part 2: Rewrite the above function with a lambda expression. You can just use a Python shell and take a screenshot of the code.

```
1 wears_jacket = lambda temp, raining: temp < 60 or raining
```

Question 4: Sum of the first N natural numbers:

Part 1: Write a function sumNaturals (n) that returns the sum of the first n natural numbers. You can use this formula $1 + 2 + \dots + n = n(n+1) / 2$. Make sure that the function returns an integer.

Do not use a for loop or a while loop.

```
1 def sumNaturals(n):
2     """ Sum all the first n natural numbers.
3     >>> sumNaturals(3) # 1 + 2 + 3 = 6
4     6
5     >>> sumNaturals(5) # 1 + 2 + 3 + 4 + 5 = 15
6     15
7     """
8     return n*(n+1)//2
```

```
C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>python -m doctest question4Part1.py -v
Trying:
    sumNaturals(3) # 1 + 2 + 3 = 6
Expecting:
    6
ok
Trying:
    sumNaturals(5) # 1 + 2 + 3 + 4 + 5 = 15
Expecting:
    15
ok
1 item had no tests:
    question4Part1
1 item passed all tests:
    2 tests in question4Part1.sumNaturals
2 tests in 2 items.
2 passed.
Test passed.
C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>
```

Part 2: Define a lambda expression that takes n and returns the sum of the first n natural numbers, using the above formula. You can just use a Python shell and take a screenshot of the code.

```
question4Part2.py x
1 sumNaturals = lambda n: n*(n+1)//2
```

Q5: A Plus Abs B

```
question5.py x Command Prompt
1 from operator import add, sub
2
3 def a_plus_abs_b(a, b):
4     """Return a+abs(b), but without calling abs.
5     >>> a_plus_abs_b(2, 3)
6     5
7     >>> a_plus_abs_b(2, -3)
8     5
9     """
10
11     if b < 0:
12         f = sub
13     else:
14         f = add
15
16     return f(a, b)

C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>python -m doctest question5.py -v
Trying:
    a_plus_abs_b(2, 3)
Expecting:
    5
ok
Trying:
    a_plus_abs_b(2, -3)
Expecting:
    5
ok
1 item had no tests:
  question5
1 item passed all tests:
  2 tests in question5.a_plus_abs_b
  2 tests in 2 items.
  2 passed.
Test passed.

C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>
```

Question 6: You Define a Function

Part 1: Write a function that takes in one or two input parameters and returns an output. The function should return the output of a **one-line expression**. Write at least three test cases for your function in the docstring. Use the command line to test your function against the test cases. Take a screenshot of your code and the result of your test. Also, write the function in the below box as well.

Make sure your function has just one line of code

```
question6Part1.py x Command Prompt
1 from operator import mul, sub
2
3 def multiplyButFrustrating(variableOne, variableTwo):
4     """Multiplies variableOne by variableTwo
5     >>> multiplyButFrustrating(2,6)
6     11
7     >>> multiplyButFrustrating(0,3)
8     -1
9     >>> multiplyButFrustrating(5,-1)
10    -6
11    """
12    return sub(mul(variableOne, variableTwo), 1)

C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>python -m doctest question6Part1.py -v
Trying:
    multiplyButFrustrating(2,6)
Expecting:
    11
ok
Trying:
    multiplyButFrustrating(0,3)
Expecting:
    -1
ok
Trying:
    multiplyButFrustrating(5,-1)
Expecting:
    -6
ok
1 item had no tests:
  question6Part1
1 item passed all tests:
  3 tests in question6Part1.multiplyButFrustrating
  3 tests in 2 items.
  3 passed.
Test passed.

C:\Users\Ben\OneDrive\Desktop\CIS Repos\CIS61A\Lab1>
```

Part 2: Write the same function as a lambda function.

```
question6Part1.py  ×  question6Part2.py  ×  
1  from operator import mul, sub  
2  
3  multiplyButFrustrating = lambda variableOne, variableTwo: sub(mul(variableOne, variableTwo), 1)
```