Programming Lesson 2: Anonymous functions; div and mod

Use Python Idle to complete this assignment.

A. Anonymous functions

If we want to give a name to a function, we can define it:

```
def double(n):
   return 2*n

double(5)                returns 10
```

Sometimes, we want to write code to apply any given function:

```
def foo(myfun, mynumber):
      print(myfun(mynumber))
```

Then we can call foo(double, 3) to get 6.

Or we can call foo(lambda x:3*x, 5) to get 15.

In this last example, we used an anonymous function. It is called anonymous since we don't give the function a name. The expression

```
lambda x:3*x
```

defines a function f by f(x) = 3*x without naming that function.

Lambda functions are primarily used to define little functions that will only be used once and so really don't need a name. The term lambda comes from Alonzo Church's lambda calculus. The syntax is

```
lambda argumentlist : expression
```

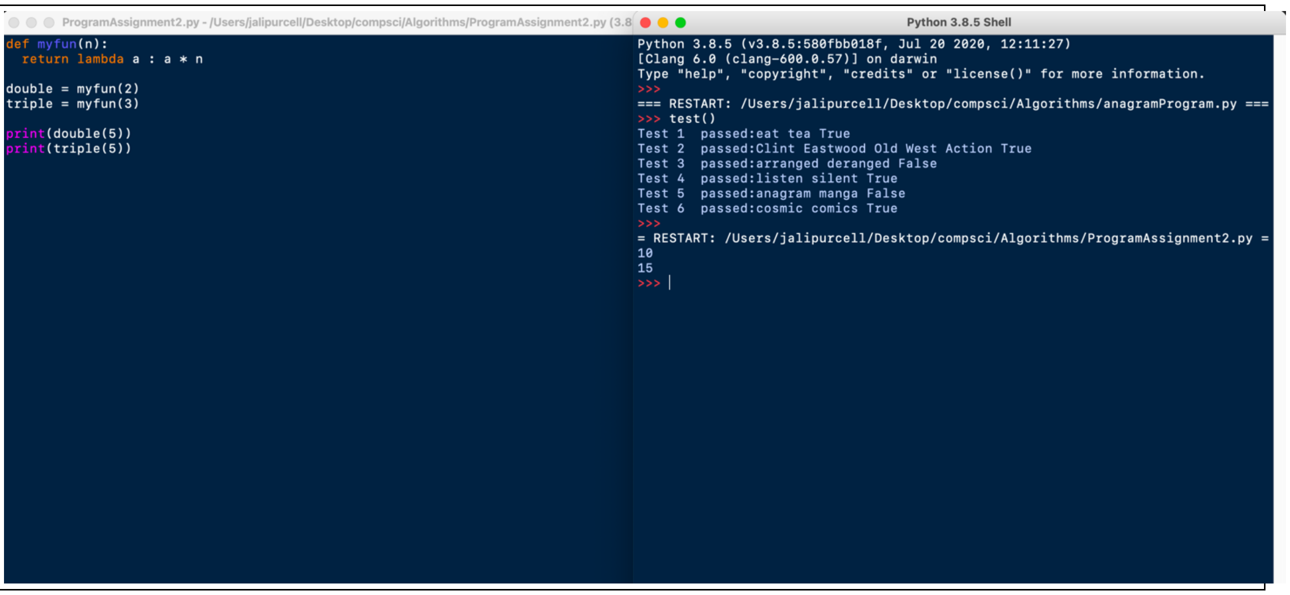
First we have the word lambda meaning "here is a function" followed by a list of variables that represent the inputs. Then we type a colon and the formula for the result.

We can then write a function to create a function!

1. Run the following code snippet. Then write full sentences to explain each line:

```
def myfun(n):
  return lambda a : a * n

double = myfun(2)
triple = myfun(3)

print(double(5))
print(triple(5))
```

Your code and transcript:



ProgramAssignment2.py - /Users/jalipurcell/Desktop/compsci/Algorithms/ProgramAssignment2.py (3.8

```
def myfun(n):
  return lambda a : a * n

double = myfun(2)
triple = myfun(3)

print(double(5))
print(triple(5))
```

Python 3.8.5 Shell

```
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: /Users/jalipurcell/Desktop/compsci/Algorithms/anagramProgram.py ===
>>> test()
Test 1  passed:eat tea True
Test 2  passed:Clint Eastwood Old West Action True
Test 3  passed:arranged deranged False
Test 4  passed:listen silent True
Test 5  passed:anagram manga False
Test 6  passed:cosmic comics True
>>>
= RESTART: /Users/jalipurcell/Desktop/compsci/Algorithms/ProgramAssignment2.py =
10
15
>>> |
```

Your explanation of each line:

Line 1. Defining the name of a function as myfun, and adding a parameter of n
Line 2. The function returns the anonymous function with argument a, which computes a*n
Line3. Assigns variable double to myfun(2), or myfun with parameter of 2.
Line3. Assigns variable triple to myfun(3), or myfun with parameter 3.
Line4. Print the result of double(5), also known as myfun(2) with the argument a assigned to 5. [10 is printed]
Line5. Print the result of triple(5), also known as myfun(3) with the argument a assigned to 5. [15 is printed]

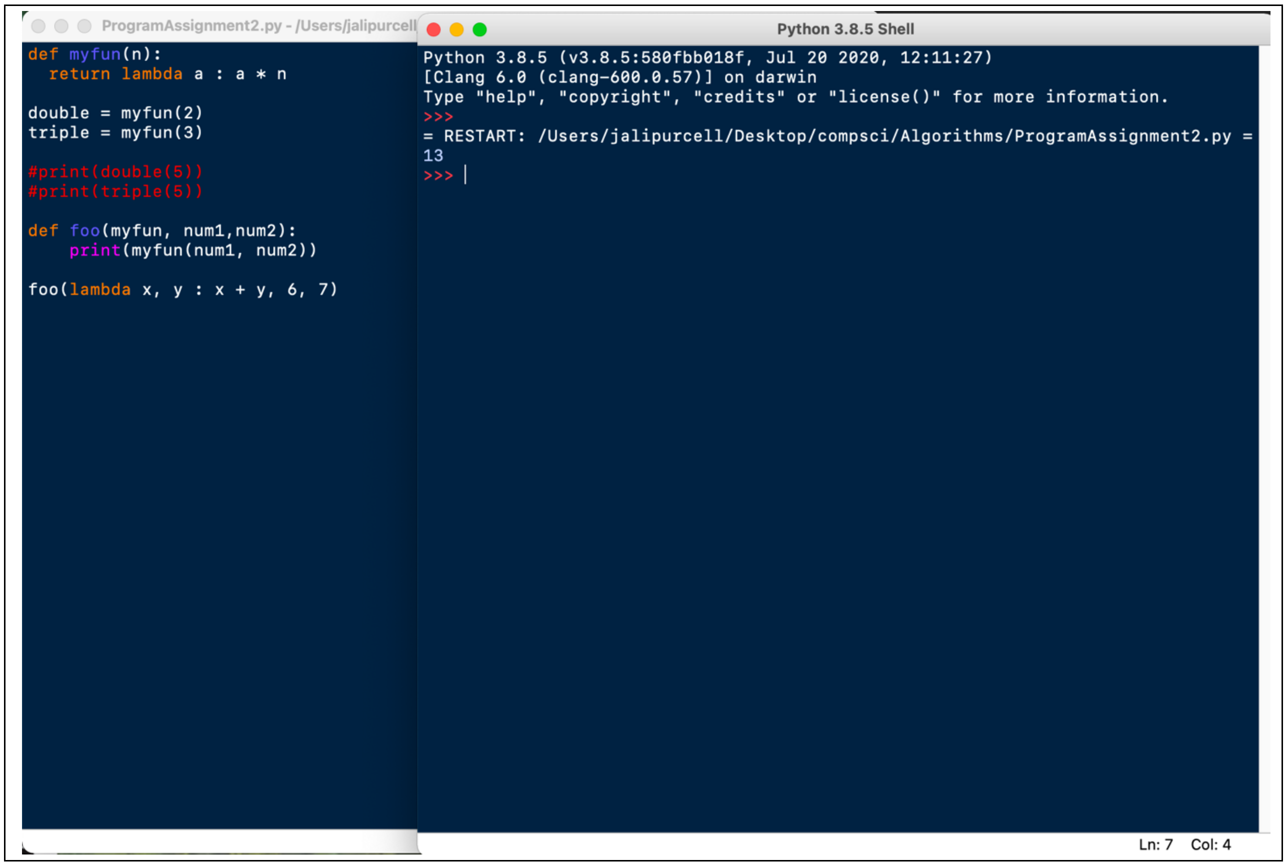
2. An anonymous function can have multiple inputs.

   a. Run the following code and explain the results:

```
def foo(myfun, num1,num2):
   print(myfun(num1, num2))
```

foo(lambda x, y : x + y, 6, 7)

Your code and transcript:



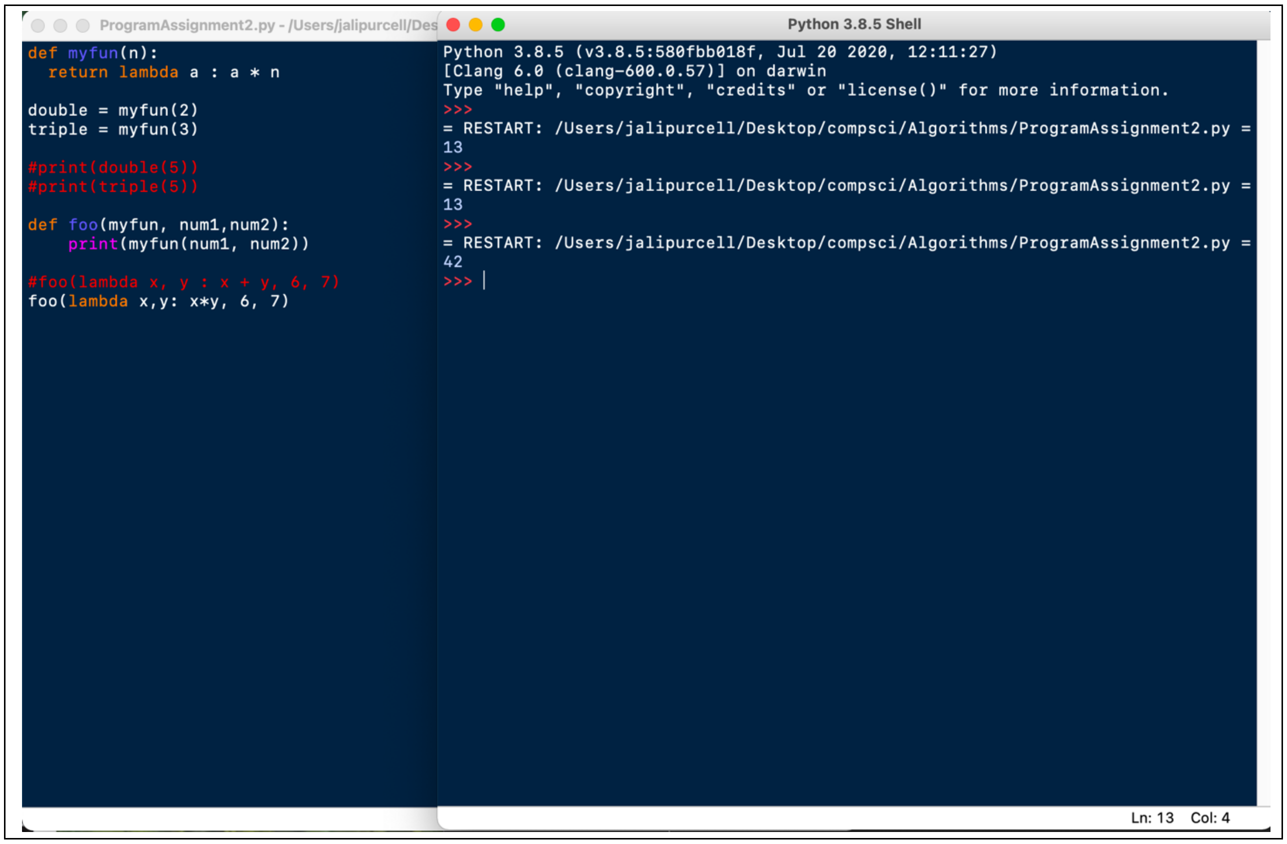
```
def myfun(n):
  return lambda a : a * n

double = myfun(2)
triple = myfun(3)

#print(double(5))
#print(triple(5))

def foo(myfun, num1,num2):
    print(myfun(num1, num2))

foo(lambda x, y : x + y, 6, 7)
```

```
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/jalipurcell/Desktop/compsci/Algorithms/ProgramAssignment2.py =
13
>>> 
```

Your explanation of the results:

Line 1. Defining a function foo, with parameters myfun, num1, and num2
Line2. Prints the result of myfun, with parameters num1, and num2.
Line3. Calls foo, giving the anonymous function arguments x, y, and commanding it to add x and y, then passing 6 and 7 to x and y
Returned: 13. So, this means myfun was passed x+y, x=6, and y=7. 6+7=13

b. Write a lambda expression for an anonymous function that multiplies two given numbers. Run foo with this lambda expression and explain your result.
Your code and transcript:


ProgramAssignment2.py - /Users/jalipurcell/Des

```
def myfun(n):
  return lambda a : a * n

double = myfun(2)
triple = myfun(3)

#print(double(5))
#print(triple(5))

def foo(myfun, num1,num2):
    print(myfun(num1, num2))

#foo(lambda x, y : x + y, 6, 7)
foo(lambda x,y: x*y, 6, 7)
```

Python 3.8.5 Shell

```
Python 3.8.5 (v3.8.5:580fbb018f, Jul 20 2020, 12:11:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/jalipurcell/Desktop/compsci/Algorithms/ProgramAssignment2.py =
13
>>>
= RESTART: /Users/jalipurcell/Desktop/compsci/Algorithms/ProgramAssignment2.py =
13
>>>
= RESTART: /Users/jalipurcell/Desktop/compsci/Algorithms/ProgramAssignment2.py =
42
>>> |
```

Your explanation of the results:

I called foo, assigning myfun to x*y with arguments x and y, assigned to x=6, and y=7. I chose the same numbers as previously to compare. The value returned is 42, which is 6*7

B. div and mod in Python

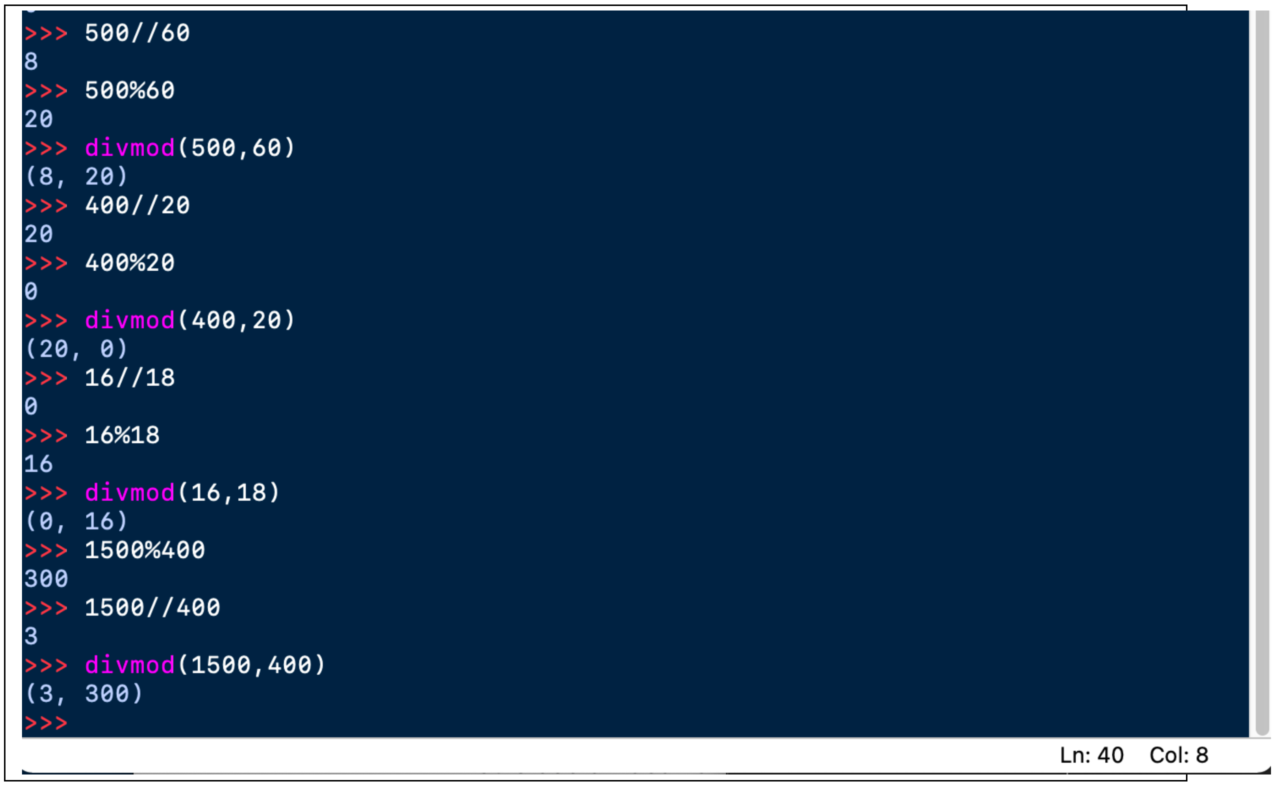
What does each of the following lines of code return? Run the code to make sure you are correct. Explain using complete sentences. Provide additional examples to illustrate these concepts.

1. 500//60. 8, meaning 500 was divided by 60, which has a result of 8 (rounded down to the nearest integer).

2. 500 % 60 20, the modulus returns the remainder left from 500 divided by 60. Since 500//60 returns 8, but 8*60 returns 480, the modulus returns 500-480=20. A zero would be returned if 500 was divisible by 60 without any remainder.

3. divmod(500,60) (8,20): This function takes both of the previous functions, and displays the results for both: with div first, and modulo second. 500//60=8, given in the x slot, and 500%60=20, given in the y slot.

Your code and transcript with additional examples:



```
>>> 500//60
8
>>> 500%60
20
>>> divmod(500,60)
(8, 20)
>>> 400//20
20
>>> 400%20
0
>>> divmod(400,20)
(20, 0)
>>> 16//18
0
>>> 16%18
16
>>> divmod(16,18)
(0, 16)
>>> 1500%400
300
>>> 1500//400
3
>>> divmod(1500,400)
(3, 300)
>>>
```

Your explanation of the results:

Lines 1-3: Explained above.
Line 4: 400//20, 400 divided by 20 is 20 without any remainder.
Line 5: Since 400//20 has no remainder, 400%20 returns 0.
Line 6: divmod(400,20), returns 20 (displaying 400//20) and 0 (400%20)
Line 7: 16//18, 18 cannot divide 16 since it is bigger than 16, so 0 is returned.
Line 8: 16%18=16 because 16//18 returns 0, and 16-0=16
Line 9: 1500%400 returns 300 because 1500//400 returns 3, and 1500- (3x400)=300
Line 10: 1500//400 returns 3 because 400*4 is larger than 1500
Line 11: divmod places 1500//400 in the x slot, and 1500%400 in the right slot