CIS 61 :: Lab 01 - Expressions and Names

Student Name: Jonathan Adrianto Saleh (CCCID : CAU9032)

Instructions:

1. *Make a copy of the assignment template. Go to File => Make a copy (or download as a Word file.)*
2. Complete definitions and attach Snipping Photos where appropriate
3. *Place your name in the Title of each Assignment*
    a. *For Example: CIS 61 - Lab 00 - Introduction to Python - Irfan O.*
4. Use the book or do online research to find answers.
5. **Write your answers using a different font color. Find your own unique color.**
6. Write answers in your own words. **DO NOT COPY & PASTE** from anywhere.
7. **Submission:** When done, go to **File -> Download as -> Microsoft Word** and then upload the file to Lab-00 on Canvas.

*The lines in the triple-quotes `"""` are called a **docstring**, which is a description of what the function is supposed to do. When writing code in 61A, you should always read the docstring!*

*The lines that begin with `>>>` are called **doctests**. Recall that when using the Python interpreter, you write Python expressions next to `>>>` and the output is printed below that line. Doctests explain what the function does by showing actual Python code: "if we input this Python code, what should the expected output be?"*

Don't forget to save your assignment after you edit it! In most text editors, you can save by navigating to File > Save or by pressing Command-S on MacOS or Ctrl-S on Windows.

Below command will help you test the python file

```
> python3 -m doctest -v filename.py
```

Lab 1 - Expressions and Functions – Jonathan Saleh

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

Question 1: **Twenty-Twenty-One**

Part 1: Come up with the most creative expression that evaluates to 2021, using only numbers and the +, *, and - operators.

You should replace the underscores in `return _____` with the expression that evaluates to 2020.

```
def twenty_twenty_one():
    """Come up with the most creative expression that evaluates
    to 2021,using only numbers and the +, *, and - operators.
    (no call expressions)
    >>> twenty_twenty_one()
    2021
```

```
    """

    return 2*(10*(50+50))+(1*2*3*4)-(6-3)
```

```
>>> def twenty_twenty_one():
        return 2*(10*(50+50))+(1*2*3*4)-(6-3)

>>> twetny_twenty_one()
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    twetny_twenty_one()
NameError: name 'twetny_twenty_one' is not defined
>>> twenty_twenty_one()
2021
```

Part 2: Try to rewrite the same expression, this time entirely with call expressions (using function calls: add, mul, pow, etc..)

```
>>> def twenty_twenty_one():

     return add(mul(4,5),add(1,mul(2,(pow(10,(mul(1,(add(1,2)))))))))
```

```
>>> def twenty_twenty_one():
        return add(mul(4,5),add(1,mul(2,(pow(10,(mul(1,(add(1,2)))))))))

>>> twenty_twenty_one()
2021
```

Question 2: **Area**

Part 1: Write definitions for these functions:

sphereArea(radius) returns the surface area of a sphere having the given radius.

sphere Volume (radius). Returns the volume of a sphere having the given radius.

Write two test cases for each function. Be careful! These functions will return decimal numbers (floats).

```
from math import pi
```

```
def sphere_area(r):
    """Area of a sphere with radius r.
>>> sphere_area(5)
    314.14159
>>> sphere_area(10)
    1256.6370

    """
    return 4* pi * r**2


def sphere_volume(r):
    """Volume of a sphere with radius r.
>>> sphere_area(5)
    523.59877
>>> sphere_area(10)
    4188.79020
    """
    Return (4/3) * pi * pow(r,3)
```

```
>>> from math import pi
>>> pi
3.141592653589793
>>> def sphere_area(r):
        return 4 * pi * r**2

>>> sphere_area(5)
314.1592653589793
>>> def sphere_volume(r):
        return (4/3) * pi * pow(r,3)

>>> sphere_volume(5)
523.5987755982989
>>>
```

Part 2: Rewrite 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.

```
sphere_area = lambda r: 4* 3.14 * r**(2)
```

```
sphere_volume = lambda r:(4/3)*3.14* pow(r,3)
```

```
>>> sphere_area = lambda r: 4 * pi * r**2
>>> sphere_area
<function <lambda> at 0x7fd76ac9a0d0>
>>> sphere_area(5)
314.1592653589793
>>> sphere_volume = lambda r: (4/3) * pi * pow(r,3)
>>>
>>> sphere_volume
<function <lambda> at 0x7fd76a833ee0>
>>> sphere_volume(5)
523.5987755982989
>>>
```

```
>>> sphere_area
<function <lambda> at 0x7fd76ac9a0d0>
>>> sphere_area(5)
314.1592653589793
>>> sphere_area(10)
1256.6370614359173
>>>
```

```
>>>
>>> sphere_volume
<function <lambda> at 0x7fd76a833ee0>
>>> sphere_volume(5)
523.5987755982989
>>> sphere_volume(10)
4188.790204786391
>>>
```

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.

```
def wears_jacket(temp, raining):
    """
    >>> wears_jacket(90, False)
    False
    >>> wears_jacket(40, False)
    True
    >>> wears_jacket(100, True)
    True
    """
    return temp<60 or raining
```

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

```
>>> def wears_jacket(temp, raining):
        return temp<60 or raining

>>> wears_jacket(90,False)
False
>>> wears_jacket(40,False)
True
>>> wears_jacket(100,True)
True
```

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

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

```
>>> wears_jacket = lambda temp,raining: temp<60 or raining
>>> wears_jacket
<function <lambda> at 0x7fd76ac9a430>
>>> wears_jacket(90,False)
False
>>> wears_jacket(40,False)
True
>>> wears_jacket(100,True)
True
>>>
```

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+ ... + n = n(n+1) / 2 or you can use a **while loop** with a count and an accumulator variable. Make sure that the function returns an int. **Do not use a for loop**.

```
def sumNaturals(n):
    """Sum all the first n natural numbers.
    >>> sumNaturals(3) # 1 + 2 + 3 = 6
    6
    >>> sumNaturasl(5) # 1 + 2 + 3 + 4 + 5 = 15
    15
    """
     ct = 0
     tot = 0
     while n > ct:
          ct = ct + 1
          tot = tot + ct
     return tot
```

```
>>> def sumNaturals(n):
        ct = 0
        tot = 0
        while n > ct:
                ct = ct + 1
                tot = tot + ct
        return tot

>>> sumNaturals(5)
15
>>>
```

```
>>> def sumNaturals(n):
        return int(n*(n+1)/2)

>>> sumNaturals(5)
15
```

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 the screenshot of the code.

sumNaturals = lambda n: int(n*(n+1)/2)

```
>>> sumNaturals = lambda n: int(n*(n+1)/2)
>>> sumNaturals
<function <lambda> at 0x7fd76ac9a3a0>
>>> sumNaturals(5)
15
```

```
>>> sumNaturals
<function <lambda> at 0x7fd76ac9a3a0>
>>> sumNaturals(3)
6
>>>
```

Question 5: You Define a Function

Part 1: Write a function that takes in one or two inputs 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.

```
def average(n1,n2):
    """Average of two Numbers
>>> average(10,10)
    10.0
>>> average(400,0)
    200.0
>>> average(15,20)
    17.5
    """
    return (n1 + n2)/2
```

CIS61 > lab1.py > average

```
1   def average(n1,n2):
2       """Average of two Numbers
3       >>> average(10,10)
4       10.0
5       >>> average(400,0)
6       200.0
7       >>> average(15,20)
8       17.5
9       """
10      return (n1 + n2)/2
11
```

```
def mynumber(x,y):
    """makes x the power of y and adds 5
>>> mynumber(10,3)
    1005
>>> mynumber(5,4)
    630
>>> mynumber(1,10)
    6
    """
    return add(5,pow(x,y))
```

CIS61 > lab1.2.py > ...

```
1   from operator import add,pow
2
3   def mynumber(x,y):
4       """makes x the power of y and adds 5
5   >>> mynumber(10,3)
6   1005
7   >>> mynumber(5,4)
8   630
9   >>> mynumber(1,10)
10      6
11  """
12      return add(5,pow(x,y))
13
14
```

```
def buy_food(hungry,money):
    """Should buy food
>>> buy_food (True,10)
    False
>>> buy_food (False,100)
    False
>>> buy_food (True,20)
    True
    """
    return hungry and money>10
```

CIS61 > lab1.3.py > buy_food

```
1   def buy_food(hungry,money):
2       """Should buy food
3       >>> buy_food (True,10)
4       False
5       >>> buy_food (False,100)
6       False
7       >>> buy_food (True,20)
8       True
9       """
10      return hungry and money>10
11
```

Program 1

```
(base) joas160703@JoAS160703s-MacBook-Air lab1 % python3 -m doctest -v lab1.py
Trying:
    average(10,10)
Expecting:
    10.0
ok
Trying:
    average(400,0)
Expecting:
    200.0
ok
Trying:
    average(15,20)
Expecting:
    17.5
ok
1 items had no tests:
    lab1
1 items passed all tests:
   3 tests in lab1.average
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
(base) joas160703@JoAS160703s-MacBook-Air lab1 %
```

Program 2

```
(base) joas160703@JoAS160703s-MacBook-Air lab1 % python3 -m doctest -v lab1_2.py
Trying:
    mynumber(10,3)
Expecting:
    1005
ok
Trying:
    mynumber(5,4)
Expecting:
    630
ok
Trying:
    mynumber(1,10)
Expecting:
    6
ok
1 items had no tests:
    lab1_2
1 items passed all tests:
   3 tests in lab1_2.mynumber
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
(base) joas160703@JoAS160703s-MacBook-Air lab1 %
```

Program 3


```
(base) joas160703@JoAS160703s-MacBook-Air lab1 % python3 -m doctest -v lab1_2.py
Trying:
    mynumber(10,3)
Expecting:
    1005
ok
Trying:
    mynumber(5,4)
Expecting:
    630
ok
Trying:
    mynumber(1,10)
Expecting:
    6
ok
1 items had no tests:
    lab1_2
1 items passed all tests:
   3 tests in lab1_2.mynumber
3 tests in 2 items.
3 passed and 0 failed.
Test passed.
(base) joas160703@JoAS160703s-MacBook-Air lab1 %
```


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

average = lambda n1,n2: (n1 + n2)/2

```
>>> average = lambda n1,n2: ( n1 + n2 )/2
>>> average
<function <lambda> at 0x7fd76ac9a5e0>
>>> average(10,10)
10.0
>>> average(400,0)
200.0
>>> average(15,20)
17.5
>>>
```

mynumber = lambda x,y: add(5,pow(x,y))

```
>>> mynumber = lambda x,y: add(5,pow(x,y))
>>> mynumber
<function <lambda> at 0x7fd76ac9a790>
>>> mynumber(10,3)
1005
>>> mynumber(5,4)
630
>>> mynumber(1,10)
6
>>>
```

Ln: 272

buy_food = lambda hungry,money: hungry and money>10

```
>>> buy_food = lambda hungry,money: hungry and money>10
>>> buy_food
<function <lambda> at 0x7fd76ac9a820>
>>> buy_food (True,10)
False
>>> buy_food (False,100)
False
>>> buy_food (True,20)
True
>>>
```

Ln: