Northwestern Polytechnic University

Python Programming
Homework Assignment #3

Due day: 10/11/2021

Instruction:

1. Push the source code to Github or answer sheet in word file
2. Please follow the code style rule like programs on handout.
3. Overdue homework submission could not be accepted.
4. Takes academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)

1. Write a function to take a positive integer *x* as input and print all ways of forming positive integer *x* by multiplying two positive integers together, ordered by the first term. Then, return whether the sum of the proper divisors of *x* is greater than *x*.

```
def abndnt(n):
    """
    >>> abndnt(12)      # 1 + 2 + 3 + 4 + 6 is 16, which is larger than 12
    1 * 12
    2 * 6
    3 * 4
    True
   >>> abndnt (14)      # 1 + 2 + 7 is 10, which is not larger than 14
    1 * 14
    2 * 7
    False
    >>> abndnt (16)
    1 * 16
    2 * 8
    4 * 4
    False
    >>> abndnt (20)
    1 * 20
    2 * 10
    4 * 5
    True
    >>> abndnt (22)
    1 * 22
    2 * 11
    False
    >>> r = abndnt(24)
    1 * 24
    2 * 12
    3 * 8
```

```
  4 * 6
  >>> r
 True
"""
def abndnt(x):
    sum=0
    for i in range (1,x):
        if x%i==0 and i*i<x:
            q=x/i
            print(str(i) + ' * ' + str(int(q)))
        if x%i==0:
            sum=sum+i
    if sum>x:
        print(True)
    elif sum<x:
        print(False)
    #print(ans)
abndnt(16)
```

2. Define a high-order function to implement the following operations

```
def  fancy_prnt (n):
"""
    A function prints numbers in a specified range except those divisible by
    n, and print it with “Buzz!”

    Assume that the following example is to print numbers from 0 to (10-1),
    and print “Buzz!” at the location of the number divisible by 5

    >>> replace = fancy_prnt(5)
    >>> replace(10)
    0
    Buzz!
    2
    3
    4
    Buzz!
    6
    7
    8
    9
    """
def fancy_prnt(x):
    def replace(y):
        for i in range(0, y):
            if i%x != 0:
                print(i)
            else:
                print("Buzz!")
    return replace
```

```
re = fancy_prnt(8)
re(5)
```

3. Create a high-order function to implement the following calculations

```
def adder(f1, f2):
    """
    Return a function that takes in a single variable x, and returns
    f1(x) + f2(x). You can assume the result of f1(x) and f2(x) can be
    added together, and they both take in one argument.

    def  identity(n):
            return n

   def  square(n):
            return n**2

    >>> a1 = adder(identity, square)
    >>> a1(4)                                # x + x^2 = 4 + 4^2 = 20
    20
    >>> a2 = adder(a1, identity)
    >>> a2(4)     # a1(4) + identity(4) = identity(4)+ square(4)+ identity(4)
    24
    >>> a2(5)
    35
    >>> a3 = adder(a1, a2)          # (x + x^2) + (x + x^2 + x)
    >>> a3(4)
    44
    """
```

```
def adder(f1, f2):
    def addition(x):
        return f1(x) + f2(x)
    return addition
    def identity(n):
     return n

def square(n):
    return n**2

a1 = adder(identity,square)
print (a1(4))

a2 = adder(a1, identity)
print(a2(4))

print(a2(5))

a3 = adder(a1, a2)
print(a3(4))
```

4. What is printed? And explain WHY

```
from operator import add

def  combine_funcs(op):
   def  combined(f, g):
      def  val(x):
         return   op(f(x), g(x))
      return   val
   return   combined


>>>add_func = combine_funcs(add)

>>>h = add_func(abs, neg)
>>>print(h(-5))
```

**notice that python visualization online tool is good software to either observe program execution process or debug your program at http://pythontutor.com/visualize.html#mode=edit*

⇨ **It displays error reason being "Neg" function is not defined**

5. Write a function to implement intersects, which takes a one-argument function "*f*" and argument *"x",* returns a function "g". It returns *True* if *f(x)=g(x),* otherwise *False*.

```
def  intscts(f, x):

    """Returns a function that returns if f intersects g at x.

    >>> at_three = intscts (square, 3)
    >>> at_three(triple)                  # triple(3) == square(3)
    True
    >>> at_three(increment)
    False
    >>> at_one = intscts (identity, 1)
    >>> at_one(square)
    True
    >>> at_one(triple)
    False
    """
```

```
def intscts(f,x):
  def operation(g):
    if f(x)==g(x) :
      return True
    else :
      return False
  return operation
def square(x):
  return x * x

def triple(x):
  return 3 * x

def identity(x):
  return x

def increment(x):
  return x + 1
```

6. Complete the following function

```
def   f():
  """
  >>> f()()(3)()
  3
  """
  # Your Program
def f(x=0):
   if x!=0:
       print(x)
   return f
f()()(3)()
```

7. Define a function *"smth"* that takes a function *g* and a value to use for *dx* and returns a function that computes the smoothed version of *g*. Do NOT use any *"def"* statements inside of *"smth",* but use *"lambda"* expressions instead.

```
def   smth(g, dx):
   """Returns the smoothed version of g, f where
   f(x) = (g(x - dx) + g(x) + g(x + dx)) / 3

   >>> square = lambda x: x ** 2
   >>> round(smth(square, 1)(0), 3)
   0.667
   """
square = lambda x: x ** 2



def smth(g,dx):
    def f(x):
        return (g(x-dx)+g(x)+g(x+dx))/3
    return f

round( smth(square, 1) (0), 3)
```

8. Define a function *"cyc"* that takes in three functions *g1*, *g2,* and *g3* as arguments. *"cyc"* will return another function that should take in an integer argument *n* and return another function. That final function should take in an argument *x* and cycle through applying *g1*, *g2,* and *g3* to *x*, depending on what *n* was. Here's what the final function should do to *x* for a few values of *n*:

- *n = 0*, return *x*
- *n = 1*, apply *g1* to *x*, or return *g1(x)*
- *n = 2*, apply *g1* to *x* and then *g2* to the result of that, or return *g2(g1(x))*
- *n = 3*, apply *g1* to x, *g2* to the result of applying *g1*, and then *g3* to the result of applying *g2*, or *g3(g2(g1(x)))*
- *n = 4*, start the cycle again applying *g1*, then *g2*, then *g3*, then *g1* again, or *g1(g3(g2(g1(x))))*
- And so forth.

**Hint: most of the work goes inside the most nested function.*

```
def  cyc(g1, g2, g3):
    """ Returns a function that is itself a higher order function
    >>> def add_one(x):
    ...        return x + 1

    >>> def times_two(x):
    ...        return x * 2

    >>> def add_three(x):
    ...        return x + 3

    >>> my_cyc = cyc(add_one, times_two, add_three)
    >>> h= my_cyc(0)
    >>> h(5)
    5

    >>> h = my_cyc(2)
    >>> h(1)              # times_two (add_one (1))
    4
    >>> h = my_cyc(3)
    >>> h(2)              # add_three (times_two (add_one (2)))
    9

    >>> h = my_cyc(4)
    >>> h(2)              # add_one (add_three (times_two (add_one (2))))
    10

    >>> h = my_cyc(6)
    >>> h(1)
    19      #add_three(times_two (add_one (add_three (times_two (add_one (1))))))
    """
     def  cyc(g1, g2, g3):
```

```
    def wrapper(n):
        def operation(x):
            ans = x
            for i in range(1, n+1):
                r = i%3   #print("i {} r {} n {}".format(
i, r, n))
                if r == 1:
                    ans = g1(ans)
                elif r == 2:
                    ans = g2(ans)
                else:
                    ans = g3(ans)
            return ans
        return operation
    return wrapper

def add_one(x):
    return x + 1

def times_two(x):
    return x * 2

def add_three(x):
    return x + 3

my_cyc = cyc(add_one, times_two, add_three)
h = my_cyc(4)
print(h(2))
```