**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):***

***i=1***

***sum=1***

***while ((i\*i)<=n):***

***if (n%i==0):***

***print(i,n//i)***

***if (i==n//i):***

***sum+=i***

***elif (i!=n and n//i !=n):***

***sum+=i+n//i***

***i+=1***

***if sum>n:***

***return True***

***else:***

***return False***

***abndnt(12)***

***abndnt(14)***

***abndnt(16)***

***abndnt(20)***

***abndnt(22)***

***abndnt(24)***

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

*A function prints numbers in a specified range except those divisible by n, and prints 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*

*def fancy\_prnt (x):*

*def h(n):*

*i=0*

*for i in range (0, n):*

*if (i==0):*

*print(i)*

*i=i+1*

*elif (x%i==0):*

*print("Buzz!")*

*i=i+1*

*else:*

*print(i)*

*i=i+1*

*return h*

*replace=fancy\_prnt (5)*

*replace (10)*

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

*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 adder(f1,f2):***

***def h(x):***

***return f1(x)+f2(x)***

***return h***

***def identity (n):***

***return n***

***def square (n):***

***return n\*\*2***

***a1= adder (identity, square)***

***a1(4)***

***a2=adder(a1,identity)***

***a2(4)***

***a2(5)***

***a3=adder(a1,a2)***

***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*](http://pythontutor.com/visualize.html#mode=edit)

The above program returns a value of 10 as output.

from operator import add

from operator import neg

def combine\_funcs(op): # (op=add)

def combined(f, g): #(f=abs, g=neg)

def val(x): # val(-5)

return op(f(x), g(x)) # add (abs(-5),neg(-5))= add(5,5)=10

return val # returns 10 to combined

return combined # returns combined value 10 to the function

add\_func = combine\_funcs(add)

h = add\_func (abs, neg)

print(h(-5))

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):*

*def g(x):*

*return triple(x)*

*if f(x)==g(x):*

*return True*

*else:*

*return False*

*def square(n):*

*return n\*n*

*def triple(n):*

*return 3\*n*

*at\_three = intscts(square, 3)*

*at\_three*

6. Complete the following function

*def f():*

*def f1():*

*def f2(x):*

*def f3():*

*return x*

*return f3*

*return f2*

*return f1*

*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.

*Returns the smoothed version of g, f where*

*f(x) = (g(x - dx) + g(x) + g(x + dx)) / 3*

*def smth(f, dx):*

*square = lambda x: x \*\* 2*

*return lambda x: (f(x - dx) + f(x) + f(x + dx)) / 3*

*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):*

*def g(x):*

*def fun\_x(n):*

*if n==0:*

*return x*

*elif n==1:*

*return g1(x)*

*elif n==2:*

*return g2(g1(x))*

*return g*

*return fun\_x*

*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)*

*my\_cyc(1)*