**Logo

Description automatically generated**

**San Francisco Bay University**

**Python Programming**

**Homework Assignment #2**

**Due day: 6/24/2023**

*Melvin Divine Pritchard :19857*

**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*

*"""*

***Program***

*def abndnt(x):*

*# Print all ways of forming x by multiplying two positive integers*

*for i in range(1, x + 1):*

*if x % i == 0:*

*j = x // i*

*if j >= i:*

*print(i, "\*", j)*

*# Calculate the sum of proper divisors*

*divisor\_sum = sum(i for i in range(1, x) if x % i == 0)*

*# Return whether the sum of proper divisors is > x*

*return divisor\_sum > x*

*def main():*

*# User input for the positive integer*

*x = int(input(" Please enter a positive integer: "))*

*# function testing based on userinput*

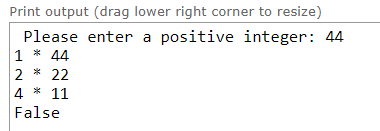
*result = abndnt(x)*

*print(result)*

*if \_\_name\_\_ == '\_\_main\_\_':*

*main()*

***Function testing result, input: 44***



1. 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)(10)*

*#print number from 0 to 10*

*# if the number is divisible by 5, replace with “Buzz”*

*>>> replace(10)*

*Buzz!*

*1*

*2*

*3*

*4*

*Buzz!*

*6*

*7*

*8*

*9*

*"""*

**Program**

def fancy\_prnt(n):

def replace(num):

for i in range(num):

if i % n == 0:

print("Buzz!")

else:

print(i)

return replace

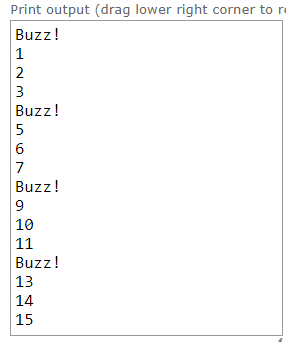
#print from 0 to (16-1)

#print “Buzz!” at the location of the number divisible by 4

replace = fancy\_prnt(4)

replace(16)

**Result**



1. 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 + = 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*

*"""*

**Program**

def adder(f1, f2):

def added\_functions(x):

return f1(x) + f2(x)

return added\_functions

def identity(n):

return n

def square(n):

return n \*\* 2

# testing

a1 = adder(identity, square)

print(a1(6))

a2 = adder(a1, identity)

print(a2(6))

print(a2(8))

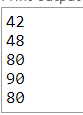
a3 = adder(a1, a2)

print(a3(6))

a4 = adder(a2, identity)

print(a2(8))

**Result**



1. 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*

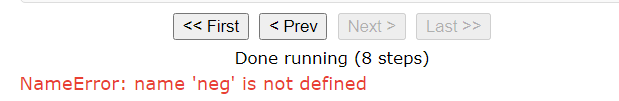
*return combined*

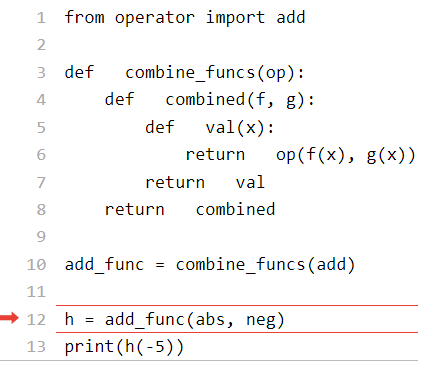
*>>>add\_func = combine\_funcs(add)*

*>>>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" \l "mode=edit)*





*The error is being shown because “ the function “neg” is being refer to but it is not defined. Defining or importing the func “neg” will fixed the error. See below for updated code.*

*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*

*def neg(n):*

*return -n*

*add\_func = combine\_funcs(add)*

*h = add\_func(abs, neg)*

*print(h(-5))*

1. 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*

*"""*

**program**

*def intscts(f, x):*

*def g(g\_func):*

*return f(x) == g\_func(x)*

*return g*

*def test\_intscts():*

*square = lambda n: n \*\* 2*

*triple = lambda n: 3 \* n*

*increment = lambda n: n + 1*

*identity = lambda n: n*

*# Test case 1*

*at\_four = intscts(square, 4)*

*print(at\_four(triple))*

*# Test case 2*

*print(at\_four(increment))*

*# Test case 3*

*at\_one = intscts(identity, 1)*

*print(at\_one(square))*

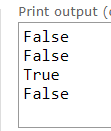
*# Test case 4*

*print(at\_one(triple))*

*if \_\_name\_\_ == "\_\_main\_\_":*

*test\_intscts()*

**Result**



1. Complete the following function

***def***  ***f():***

*"""*

*>>> f()()(3)()*

*3*

*"""*

***Program***

*def f():*

*return lambda: lambda x: lambda: x*

*# Example usage*

*result = f()()(3)()*

*print(result)*

***Result***



1. 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*

*"""*

**Program**

*def smth(g, dx):*

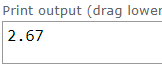
*return lambda x: (g(x - dx) + g(x) + g(x + dx)) / 3*

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

*result = round(smth(square, 2)(0), 2)*

*print(result)*

**Result**



1. 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))))))*

*"""*

***Program***

*def cyc(g1, g2, g3):*

*def inner\_func(n):*

*def apply\_functions(x):*

*for \_ in range(n):*

*if n % 3 == 0:*

*x = g1(x)*

*elif n % 3 == 1:*

*x = g2(x)*

*else:*

*x = g3(x)*

*return x*

*return apply\_functions*

*return inner\_func*

*def add\_one(x):*

*return x + 1*

*def times\_two(x):*

*return x \* 2*

*def add\_three(x):*

*return x + 3*

*# Example usage*

*my\_cyc = cyc(add\_one, times\_two, add\_three)*

*# Test case 1*

*h = my\_cyc(0)*

*result = h(5)*

*print(result)*

*# Test case 2*

*h = my\_cyc(1)*

*result = h(1)*

*print(result)*

*# Test case 3*

*h = my\_cyc(2)*

*result = h(1)*

*print(result)*

*# Test case 4*

*h = my\_cyc(3)*

*result = h(2)*

*print(result)*

*# Test case 5*

*h = my\_cyc(4)*

*result = h(2)*

*print(result)*

***Result***

