**Logo

Description automatically generated**

**San Francisco Bay University**

**Python Programming**

**Homework Assignment #2**

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

**Name: Khandoker Samiul Hoque ID: 19837**

**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)**

**Github url :** <https://github.com/KhandokerSamiulHoque/CE450-LEC-ASSIGN-2.git>

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*

*"""*

**Answer:**

def abndnt(n):

    divisors = []

    for i in range(1, int(n\*\*0.5) + 1):

        if n % i == 0:

            divisors.append(i)

            if i != n // i:

                divisors.append(n // i)

    divisors.sort()

    for i in range(len(divisors)):

        for j in range(i, len(divisors)):

            if divisors[i] \* divisors[j] == n:

                print(divisors[i], "\*", divisors[j])

    div\_sum = sum(divisors) - n

    return div\_sum > n

pos\_num = int(input("Enter a positive integer: "))

abndnt\_state = abndnt(pos\_num)

print(abndnt\_state)

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)*

*>>> replace(10)*

*0*

*Buzz!*

*2*

*3*

*4*

*Buzz!*

*6*

*7*

*8*

*9*

*"""*

**Answer:**

def fancy\_prnt(n):

    def replace(m):

        if not isinstance(m, int) or m <= 0:

            raise ValueError("Invalid input, Please enter a positive integer.")

        for num in range(m):

            if num % n == 0:

                print("Buzz!")

            else:

                print(num)

    return replace

try:

    n = int(input("Enter n: "))

    range\_end = int(input("Enter the the range (input-1): "))

    replace = fancy\_prnt(n)

    replace(range\_end)

except ValueError as error:

    print(error)

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*

*"""*

**Answer:**

def adder(f1, f2):

  def var\_func(x):

    try:

      return f1(x) + f2(x)

    except (TypeError, ValueError):

      print("Invalid input, Please enter only number ")

  return var\_func

def identity(n):

  return n

def square(n):

  return n \*\* 2

while True:

  try:

    x = int(input("Enter the number: "))

    break

  except ValueError:

    print("Invalid input, Please enter only number")

a1 = adder(identity, square)

print(a1(x))

a2 = adder(a1, identity)

print(a2(x))

a3 = adder(a1, a2)

print(a3(x))

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#mode=edit)

**Answer:**

The code defines a function called `combine\_funcs`. This function takes an operator as input and returns a new function that combines the results of two other functions using the operator.The `add\_func` variable is assigned the result of calling `combine\_funcs` with the `add` operator. This means `add\_func` is a function that adds the results of its two input functions.The `h` variable is assigned the result of calling `add\_func` with the `abs` and `neg` functions as the input functions.When `h(-5)` is called, it applies the `abs` and `neg` functions to `-5` and then adds the results. The `abs` function returns the absolute value of its input, so `abs(-5)` is 5. The `neg` function negates its input, so `neg(-5)` is -(-5), which is 5.Therefore, `h(-5)` is 5 + 5, which is 10.

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*

*"""*

**Answer:**

def intscts(f, x):

  def g(function):

    return f(x) == function(x)

  return g

def square(n):

  return n \*\* 2

def triple(n):

  return 3 \* n

def increment(n):

  return n + 1

def identity(n):

  return n

def main():

  f\_case = input("Enter square, triple, increment or identity for 1st case: ")

  x = int(input("Enter x: "))

  g\_case = input("Enter square, triple, increment or identity for 2nd case: ")

  try:

    f = {

      "square": square,

      "triple": triple,

      "increment": increment,

      "identity": identity,

    }[f\_case]

  except KeyError:

    print(f"Invalid function name. Please enter one of the following: square, triple, increment, identity.")

    exit()

  try:

    g = {

      "square": square,

      "triple": triple,

      "increment": increment,

      "identity": identity,

    }[g\_case]

  except KeyError:

    print(f"Invalid function name. Please enter one of the following: square, triple, increment, identity.")

    exit()

  x\_val = intscts(f, x)

  print(x\_val(g))

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

  main()

1. Complete the following function

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

*"""*

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

*3*

*"""*

*# Your Program*

**Answer:**

def f():

    return lambda: lambda x: lambda: x

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

print(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*

*"""*

**Answer:**

def smth(g, dx):

    try:

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

    except Exception as error:

        return lambda x: f"Error: {error}"

try:

    g\_val = input("Enter the function square =  ")

    g = eval(g\_val)

    dx = float(input("Enter the value of dx: "))

    if not callable(g):

        raise ValueError("Error!! input the correct function")

    smth\_g = smth(g, dx)

    x = float(input("Enter a value for x: "))

    result = smth\_g(x)

    if isinstance(result, str):

        print(result)

    else:

        print(round(result, 3))

except Exception as error:

    print(f"Error: {error}")

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))))))*

*"""*

**Answer:**

def cyc(g1, g2, g3):

    def int\_arg\_func(n):

        def main\_func(x):

            if n == 0:

                return x

            elif n == 1:

                return g1(x)

            elif n == 2:

                return g2(g1(x))

            elif n == 3:

                return g3(g2(g1(x)))

            else:

                result = x

                for \_ in range(n // 3):

                    result = g1(result)

                    result = g2(result)

                    result = g3(result)

                if n % 3 == 1:

                    result = g1(result)

                elif n % 3 == 2:

                    result = g2(result)

                return result

        return main\_func

    return int\_arg\_func

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)

print(h(5))

h = my\_cyc(2)

print(h(1))

h = my\_cyc(3)

print(h(2))

h = my\_cyc(4)

print(h(2))

h = my\_cyc(6)

print(h(1))