Exam 2 Practice Problems

The following problems are for practice when studying for Exam 2. It is recommended that you attempt them using pencil and paper (NOT an IDE) like you will for the exam. After attempting a problem, check your solution by typing it into your favorite IDE and debug. **If you think there is a problem with a question, please notify your instructor.**

The format for exam 2 is the same as exam 1. Several problems (multiple choice, true/false, fill in the blank, etc) will be autograded and no partial credit will be available. Partial credit will be available for code writing problems so **please comment your code**.

During the exam calculators are not allowed, you won't need one anyway. In addition, you may NOT use your phone, the web to search for additional information, your laptop, your book, your notes, lectures on Canvas, or any electronic device.

Remember to review all lecture examples and quizzes. Also review zyBook examples as well as Participation (orange) and Challenge (blue) activities.

Autograded style problems:

Go back and review your quizzes for additional autograded style problems including fill in the blank, multiple choice, multiple answer, and true/false type questions.

For the following problems, write the output to the code. If there is an error, explain the error. Don't forget `[]()\{\}` and/or `,` as needed.

1.
```
grades = [77, 82, 85, 88, 96, 97]
for i in range(1, 2):
    print(grades[i:-i], end="")
```

2.
```
i = 0
j = -1
while i < 3:
    if j > 1:
        continue
        i += 2
    else:
        i += 1
    j += 1
print(f"{i} + j")
```

3.
```
sum = 1
for i in range(1, 5):
    while i < 3:
        sum += i
        i += 1
        if sum % 2 == 0:
            sum += 1
            break
    else:
        sum *= i
    sum -= 2
print(sum)
```

4.
```
a, i = 0, 3
while a <= 3:
    for i in range(1, 3):
        if i == 1:
            i += 1
            a += 1
        else:
            a += 11
print("\"A\" is for", f"{a}", "Apples", sep=" ", end="!")
```

5.
```
j = 1
for i in range(0, 4, 2):
    print(f"{i+1}", f"{j+1}", sep=", ", end=", ")
    j += 2
print("and that's it", end="...or is it? ")
```

6.
```
mydict = {"Ann" : 18, "Bob" : 20, "Charlie" : 19}
if "Joe" in mydict:
    print("Joe is here")
elif "Ann" in mydict:
    print("Hi Ann")
else:
    print("Anyone?")
```

7.
```
name = {"Two" : "Rosewood"}
name[4] = "Lever"
name[2] = "Calendar"
name["Two"] = "Cartograph"
name["Two"] = "Sunspot"
print(name["Two"], name[2], sep=">")
```

8.
```
mydict = {}
mylist = [1, 2, 3, 4, 5]
mydict["Length"] = len(mylist)
mydict["Max"] = max(mylist)
mydict["Min"] = min(mylist)
mydict["Crazy"] = mylist[1] * mylist[3] - mylist[-1]
for key in mydict:
    print(f"{key}: {mydict[key]}")
```

9.
```
mydict = {"apple" : 2, "orange" : 3}
mydict["banana"] = 4
mylist = []
i = 0
print("Total fruit inventory equals: ", end="")   # empty string
for key in mydict:
    mylist.append(mydict[key])
    if i < len(mydict) - 1:
        print(mylist[i], key + "s", end=" + ")
    else:
        print(mylist[i], key + "s", end=" = ")
    i += 1
print(sum(mylist), "fruit")
```

10.
```
mylist = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
mydict = {}
mysum = 0
for i in range(0, len(mylist) - 1, 2):
    if mylist[i] + mylist[i+1] <= 2 * mylist[i] + 4:
        mydict[i] = i + 1
    else:
        mydict[i] = i
for item in mydict:
    mysum += mydict[item]
print(mysum, mydict[4], sep=" : ")
```

11.
```
mylist = ["apples", 4.5, True]
mytuple = ("oranges", 3, False)
mydict = {"bananas" : 2, "strawberries" : 12}
for key in mydict:
    mylist += [mydict[key]]
mytuple = [mytuple, mylist]
mylist[4] = 13
print(mytuple[0][1], mytuple[1][4], sep="00", end="1")
```

12.
```
vector1 = (1, 2, 3)
vector2 = (2, 3, 4)
dotp = vector1 * vector2
print(f"The dot product is {dotp}")
```

13.
```
mystr = "Howdy"
mylist = [2, 0, 2, 0]
mytuple = (mystr, mylist)
mytuple[1][3] = 9
print(mytuple)
```

14.
```
def myfunc(x):
    a = 1
    print(x)
a = 5
myfunc(a)
print(a)
```

15.
```
def plus1_3(x):
    return x + 1, x + 3
print(plus1_3(2)[0])
```

16.
```
def myfun():
    """This function prints a message."""
    print("Gig 'em Aggies!")
help(myfun)
```

17.
```
def myfunc(b, a):
    a = 3
    c = b + a
    return c, a
a, b = 3, 2
a, b = myfunc(a, b)
print(a, f"{b} = {a / b:.0f}", sep=" / ")
```

18.
```
def add1(x):
    if x + 1 < 10:
        return x + 1
    else:
        return "Too big!"
print(add1(1), add1(29))
```

19.
```
def myfunction(a, b=5, c):
    return a * b * c
print(myfunction(1, 2, 3))
```

20.
```
def clip(v, lo=0, hi=10):
    if v < lo:
        return lo
    if v > hi:
        return hi
    return v
print(clip(15), clip(-1), clip(7))
```

21.
```
def myfunc1(x=10, y=3, z=False):
    if z:
        var = x * y
    else:
        var = x / y
    return var
def myfunc2(myvar):
    if myvar > 5:
        return True
    elif myvar < 5:
        return False
    else:
        return 30
print(myfunc2(myfunc1()))
```

22.
```
def myfunc(x, y="Green", z=100, Flag=False):
    if z == 100:
        z += 10
        return z, x
    elif y == "Green":
        Flag = True
        print(x, z)
        return x, z
    return Flag
out1, out2 = myfunc(10)
print(out1, out2)
```

23.
```
def aplus(i):
    j = 0
    while i != 1:
        if i % 2 == 0:
            i /= 2
        else:
            i = i * 3 + 1
        j += 1
    return j
print(aplus(20))
```

24.
```
def drawStar(a=6):
    print("*" * int(a))
a = "4"
drawStar(a)
```

25.
```
def go(x):
    s = ""
    for i in range(1, x + 1):
        for j in range(1, i + 1):
            s += "# "
        s += "=\n"
    return s
print(go(2))
```

26.
```
def combo(n):
    s, i = 0, 0
    while i < n:
        j = 0
        while j < 3:
            s += 1
            if j == 1:
                break
            j += 1
        else:
            s += 10
        i += 1
    return s
print(combo(2))
```

27.
```
def A_function(one, two, three):
    list1 = []
    list2 = []
    for i in one:
        if i <= len(one) - 1:
            list1.append(one[i])
        if i % two == 0:
            list1.append(i)
        elif i % 3 == 1:
            list2.append(i + 1)
        return list1, list2
mylist = [4, 3, 22, 15, 7]
a, b = 3, 2
x, y = A_function(mylist, a, b)
print(y + x)
```

28.
```
def f(a, b=2):
    return a * b
print(f"{f(3)}{f(3, 3)}")  # no space
```

29.
```
def B_function(a, b):
    for i in a:
        if i == 5:
            continue
        if a[i] == "Cat":
            print(f"#{str(b.index(10))} {a[i]} {i}", end=" ")
            x = 2010
            break
a = [5, 10, 15, 20, 25]
b = {"Quinn" : "Cat", "Juliet" : "Dog", "Banjo" : "Dog"}
x = "2017"
B_function(b, a)
print(x)
```

30.
```
x = 4
y = timestwo(x)
def timestwo(x):
    return x * 2
print(y)
```

31.
```
number = input("Enter a number: ")  # assume the user enters 5.0
try:
    x = int(number)
    print(f"x is the integer {x}")
except:
    x = float(number)
    print(f"x is the float {x}")
```

32.
```
mystr = "Howdy! Welcome to Texas A&M Engineering!"
mylist = mystr.split()
newstr = ""  # empty string
for i in range(3, len(mylist)):
    newstr += mylist[i] + " "
print(mylist[0][:5] + " " + newstr[:-2] + " students! ")
```

33.
```
a, b = 3, 0
countStr = ""  # empty string
while a > b:
    try:
        c = a / b
        countStr += str(a)
    except:
        countStr += str(b)
        a -= 1
countList = list(countStr)
for i in range(len(countList)):
    countList[i] = float(countList[i])
print(sum(countList), countList.count(0), sep=":", end="!")
```

34.
```
mystring = "   \t\n123abc   \n"
out_value = "::".join(mystring.strip().split("23a"))
print(out_value)
```

35.
```
mystring = "AXVRL:2025/11/07"
mylist = mystring.split(":")
mylist[1] = mylist[1].split("/")
for i in range(len(mylist)):
    if i == 0:
        print(mylist[i][-3:], "\t", sep="", end="")  # both empty str
    else:
        print(f"{mylist[i][1]}\t{mylist[i][2]}\t{mylist[i][0]}")
```

36.
```
mylist = [5, 3, 7, 9, 1, 2]
mylist.pop()
mylist.insert(2, 4)
mylist.sort()
for num in mylist:
    print(num, end = " ")
```

37.
```
def output(mystr, num):
    outstring = mystr
    with open("results.csv", "w") as outfile:
        for i in range(num):
            outfile.write(outstring[-1] + ",")
            outstring = outstring[:-1]
        outfile.write(mystr[0])
    return outstring
print(output("tacocat", 6))
```

38.
```
import numpy as np
mygrid = np.arange(1, 10).reshape(3, 3)
for i in range(len(mygrid)):
    mygrid[i][i] = 5
print(mygrid)
```

39.
```
mystr = "2-g:gig'em\n3-s:aggies\n1-o:whoop"
mylist = mystr.split("\n")
good = 0
for item in mylist:
    key, word = item.strip().split(":")
    key = key.split("-")
    if word.count(key[1]) >= int(key[0]):
        good += 1
print(good)
```

40.
```
import numpy as np
x = np.linspace(1.0, 10.0, 10)
y = x ** 2 - 1
with open("zfile.txt", "w") as zfile:
    zfile.write("x\ty\n")
    for i in range(len(x)):
        mystr = str(x[i]) + "\t" + str(y[i])
        zfile.write(mystr + "\n")
with open("zfile.txt") as myfile:
    all_of_it = myfile.read().split("\t")
output = ",".join(all_of_it)
print(output)
```

41. The file `scores.csv` contains the following text:
```
ID,Score1,Score2,Score3
121,30,50,90
045,90,70,80
217,60,60,85
```

Write the output of the following code.
```
with open("scores.csv") as file1:
    count = 0
    score_dict = {}
    for i in file1:
        i = i.strip().split(",")
        score_dict[i[0]] = i[1:]
del(score_dict["ID"]
avg1, avg2, avg3 = 0, 0, 0
for i in score_dict:
    avg1 += int(score_dict[i][0])
    avg2 += int(score_dict[i][1])
    avg3 += int(score_dict[i][2])
value = [avg1/len(score_dict), avg2/len(score_dict), avg3/len(score_dict)]
print(value)
```

42. What are the contents of the file `afile2.csv` after the following lines of code have executed?
```
nextfile = open("afile2.csv", "w")
nextfile.write("\t".join("1,2,3\n".split(",")))
nextfile.write("\t".join("4,5,6\n".split(",")))
list1 = ".".join(["5", "6", "7", "8", "9"])
nextfile.write(list1)
nextfile.close()
```

43.
```
mystr = "1.1".join("1,2,3".split(","))
mystr2 = mystr.split(".")
mysum = 0
for num in mystr2:
    mysum += int(num)
print(mysum)
```

44. Given the list: `my_list = [1, 2, 3, 4, 5]`
Which of the following code snippets below will output the following? Choose all that apply.
`1, 3, 5`

A)
```
for i in range(0, 5, 2):
    print(my_list[1], end=", ")
```

B)
```
i = 0
out = my_list[i]
while i < 4:
    print(out, end=", ")
    i += 2
    out = my_list[i]
print(out)
```

C)
```
for i, value in enumerate(my_list):
    if value % 2 == 1 and i < 4:
        print(my_list[i], end=", ")
print(my_list[-1])
```

D)
```
i = 1
out = my_list[i]
while i < 5:
    print(out, end=", ")
    i += 2
    out = my_list[i]
print(out)
```

E)
```
for i in range(len(my_list)):
    if i % 2 == 0:
        print(my_list[i], end="")
        if i + 1 < len(my_list):
            print(", ", end="")
```

F)
```
count = 0
for i in my_list:
    if i % 2 == 1 and count < 4:
        print(i, end=", ")
    elif count == 4:
        print(i)
    count += 1
```

G)
```
for i in my_list:
    print(my_list[i], end=", ")
```

45. Which of the code snippets below produces the same output as the following loop? Choose all that apply.

```
for i in range(4):
    print(f"{i+1}", end="")
```

A)
```
for j in range(4):
    print(f"{j+1}", end="")
```

B)
```
i = 0
while i <= 3:
    print(f"{i+1}", end="")
    i += 1
```

C)
```
for i in range(1, 5):
    print(i, end="")
```

D)
```
for i in range(5):
    if i == 4:
        break
    print(i + 1, end="")
```

46. After executing the two lines of code below, which of the following potential third lines of code will **NOT** result in an error? Choose all that apply.

```
# Execute the following two lines of code first
the_list = [3, "5", 0]
the_str = "A1B2C3"
## third line of code goes here ##
```

A) `1_var = {the_list[1]:the_str}`

B) `myvar = the_list[the_str[1]]`

C) `a_var = the_list[0] + int(the_str[-1])`

D) `some_var = the_str, the_list`

E) `the_var = the_str[the_str[2]]`

F) `a = the_str[the_list[2]]`

47. Which of the following lines of code correctly open an existing file? Choose all that apply.

A) `myfile = open("grades.csv", "r")`

B) `with open("data.dat", "r+") as my-file:`

C) `myfile = open("stats.csv", "w")`

D) `with open("datafile.txt", "a") as myfile:`

E) `this_file = open("weather.odb", a+)`

F) `with open("barcodes.txt") as bfile:`

48. Which of the following code snippets below will execute without error?

A)
```
listA = [1, 2, 3, 4, 5]
listB = [4, 6, 8, 10, 12]
for i in range(5):
    listC += [listA[i] + listB[i]]
print(listC)
```

B)
```
listA = [6, 5, 4, 3, 2, 1]
listB = [1, 3, 5, 7, 9, 11]
listC = []
for i in range(1, 7):
    listC += [listA[i] + listB[i]]
print(listC)
```

C)
```
myList = [7, 8, 6, 4, 2]
for num in myList:
    if num < 3
        print(item)
```

D)
```
data_list = []
numbers.open("data.txt", "r")
for line in numbers:
    data_list.append(line.strip())
print(data_list)
```

E)
```
def myfunc(a, b=7, c):
    out1 = a + b / c
    out2 = a - c * b
    return out1, out2
a, b = myfunc(5, 2, 4)
```

F)
```
from math import sqrt
def func2(sigma, n):
    error = sigma / sqrt(n)
    return error
def func1(x):
    resid = 0
    avg = sum(x) / len(x)
    for i in x:
        resid += (i - avg) ** 2
    stdev = resid / (len(x) - 1)
    error = func2(stdev, len(x))
    return error
data = [2, 5, 3, 7, 4, 9, 1, 2, 5, 8, 9, 3, 4]
print(func1(data))
```

G)
```
from math import sqrt
mylist, square = [1, 3, 5, 7, 9], []
for i in mylist:
    square += i ^ 2
print(sum(square))
```

49. What are the contents of the file `results.csv` after the following lines of code have executed?

```
def output(mystr, num):
    outstring = mystr
    with open("results.csv", "w") as outfile:
        for i in range(num):
            outfile.write(outstring[-1] + ",")
            outstring = outstring[:-1]
        if len(mystr) - num == 1:
            outfile.write(mystr[0])
        elif num < len(mystr):
            for i in range(len(mystr) - (num + 1), -1, -1):
                outfile.write(outstring[i] + ",")
        return outstring
output("tacocat", 5)
```

50. Which of the code snippets below will execute without error? Choose all that apply.

A)
```
from numpy import *
a = arange(15).reshape(3, 5)
print(a)
```

B)
```
import numpy
a = arange(15).reshape(3, 5)
print(a)
```

C)
```
from math import *
a = math.sqrt(25)
print(a)
```

D)
```
import math
a = math.sqrt(25)
print(a)
```

51. Write a single line of code to import either an entire module or just the required functions in such a way that the following code executes without error.

```
## Your single line of code here ##
import numpy as np
x = np.linspace(0, 10, 21)
y = 2 * x ** 2 - 6 * x - 4
z = x ** (1 / 2) + 6
a = x
pyp.plot(x, y, "k-", x, z, "rv", x, a, "b1")
pyp.xlabel("Time [ns]")
pyp.ylabel("Space [ly^3]")
pyp.title("Plot of Something Important")
pyp.show()
```

52. Write a single line of code to import either an entire module or just the required functions in such a way that the following code executes without error.

```
## Your single line of code here ##
x = arange(0, 10.5, 0.5).reshape(7, 3)
```

Code writing problems:

53. Write a Python program that uses a loop to sum the numbers 1 to 10 (inclusive) and print the sum.

54. Create a top-down hierarchy for the following problem statement:
A birdwatcher wants a program that will allow them to input their bird sightings and output requested statistical information. Inputs include the species/family, number of individuals, date, time, and location of sighting. Outputs include the total number of species seen (overall and in a particular location), total number of places or dates birded, number of sightings of a particular species, and family with the highest number of species seen.

55. Write a Python program to take as input 5 birthdays from 5 users (1 each) and output them in chronological order. Dates should be entered with the month and day (not year) in the format "`June 6`" as a single input per user. Format the output as shown below. You MUST use a dictionary.

    Example output (input in bold, red text):

```
User 1 please enter a birthday: December 12
User 2 please enter a birthday: January 15
User 3 please enter a birthday: April 12
User 4 please enter a birthday: November 25
User 5 please enter a birthday: April 1
-----------------------------------------------
January 15
April 1
April 12
November 25
December 12
```

56. A schematic for converting phone letters to digits mapping is shown in the image below. Write a Python program that prompts the user to enter a 10-character phone number in this format XXX-XXXXXXX. Your program should replace the last seven alphabetic characters by their equivalent digits and display the entered phone number in this format XXX-XXX-XXXX. For example, if the user enters `800-GOFEDEX`, your program output would convert the number to `800-463-3339`. You may assume that the last seven characters are alphabetic characters from A to Z. You MUST use a dictionary. You may find section 5.2 in zyBooks to be helpful.

    Example output for input **800-GOFEDEX**:

```
Enter a phone number in this format XXX-XXXXXXX: 800-GOFEDEX
800-GOFEDEX is equivalent to 800-463-3339
```

57. Write a Python program that takes as input from the user the names, blood pressure, pulse, and blood glucose values of patients. Have your program prompt the user for the number of patients first, and then take as input that many names and their corresponding information. Store the information for each patient in a dictionary using the names as keys. Then, take as input from the user a name and print their information using the format shown below.

    Example output:
    ```
    Enter the number of patients: 5
    Enter the next name and information: Amari 120/75 78 102
    Enter the next name and information: Blake 134/82 73 116
    ...
    Enter a name: Cameron
    Here is information on Cameron:
    Blood pressure: 118/76, Pulse: 75, Blood glucose: 93
    ```

58. Write a Python function named `numcondition` which takes in as parameters two integers. If the integers add to more than 10 and multiply to more than 20, return `True`. If neither condition is satisfied, return `False`. If one condition is satisfied, return which condition. **Don't forget to document your function!**

    Examples:
    `numcondition(2, 3)` returns `False`
    `numcondition(20, 28)` returns `True`
    `numcondition(12, -1)` returns `addition only`
    `numcondition(3, 7)` returns `multiplication only`

59. Assume a function `isprime()` is available for you to use in a module called `ENGR102` which determines whether or not a number is a prime number. The function `isprime()` takes in as a parameter a single integer, and returns either `True` or `False`. Write a Python program that takes as input from the user two integers. If the user gives bad input, continue to prompt them to try again until they enter two integers. Then, test only the odd numbers between and including those two numbers, to check if they are prime using the `isprime()` function. Have your program print a list of the prime numbers found. If no prime numbers were found, have your program print a message stating that. Start your code with: `from ENGR102 import isprime`. You do not have to write the function `isprime()`, you only need to call it.

    Example output (bad input):
    ```
    Enter an integer: no
    Bad input! Try again: 1
    Enter another integer: 1.5
    Bad input! Try again: 5
    Primes: [3, 5]
    ```

    Example output (good input):
    ```
    Enter an integer: 8
    Enter another integer: 10
    No primes found!
    ```

60. Write a Python function named `max_min` that takes in as parameters six (6) integers and returns three (3) values in the following order: a list that includes the six (6) integers ordered from largest to smallest, the sum of the six (6) integers, and a count of the number of even integers. **Don't forget to document your function!**

    Examples:
    `max_min(1, 6, 2, 5, 3, 4)` returns `[6, 5, 4, 3, 2, 1]`, `21`, and `3`
    `max_min(8, 6, 7, 5, 3, 9)` returns `[9, 8, 7, 6, 5, 3]`, `38`, and `2`
    `max_min(102, 67, 234, 216, 185, 217)` returns `[234, 217, 216, 185, 102, 67]`, `1021`, and `3`

61. Write a Python function that takes in as parameters two `numpy` arrays A and B, checks that the inner dimensions of A and B match, and if they do calculate and return C = AB (matrix multiplication (not elementwise multiplication)). If they don't match, have your function return an empty `numpy` array. The inner dimensions of two matrices multiplied together are the two interior numbers. For matrices with dimensions 2x3 and 3x4, the inner dimensions match (both 3). For matrices with dimensions 3x2 and 3x4, the inner dimensions do not match (2 and 3). **Don't forget to document your function!**

    Examples:
    Array A: `[[0 1 2] [3 4 5]]`
    Array B: `[[ 0  1  2  3] [ 4  5  6  7] [ 8  9 10 11]]`
    Function returns `[[20 23 26 29] [56 68 80 92]]`

    Array A: `[[0 1] [2 3]]`
    Array B: `[[ 0  1  2  3] [ 4  5  6  7] [ 8  9 10 11]]`
    Function returns `[]`

62. Write a Python function named `myinsert` that takes in as parameters a sorted string of arbitrary length and a character (string of length one). Have your function insert the character so the new string remains sorted, and return it. Do NOT use the `sort` or `sorted` functions. **Don't forget to document your function!**

    Examples:
    `myinsert("abde", "c")` returns `"abcde"`
    `myinsert("abdehijmnqr", "y")` returns `"abdehijmnqry"`
    `myinsert("123589", "7")` returns `"1235789"`

    Now write a second Python function named `myinsert2` that takes in as parameters a sorted list of arbitrary length and a value. Have your function insert the value so the new list remains sorted, and return it. Do NOT use the `sort` or `sorted` functions. **Don't forget to document your function!**

    Examples:
    `myinsert2([1, 2, 4, 5], 3)` returns `[1, 2, 3, 4, 5]`
    `myinsert2([1, 4, 5, 12, 20, 28], 0)` returns `[0, 1, 4, 5, 12, 20, 28]`
    `myinsert2(["a", "b", "d", "e"], "c")` returns `["a", "b", "c", "d", "e"]`

63. Write a Python function that takes in as a parameter a list of numbers then calculates and returns the mean ($\bar{x}$) and sample variance (s^2) of the numbers. Do NOT use the `numpy` or `statistics` modules. Instead, calculate the values using the equations below. You MUST use a loop. **Don't forget to document your function!**

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Now rewrite your function using the `numpy` module.

64. An Armstrong number is a positive integer, where the sum of its digits, each raised to the power of the number of digits, is equal to the integer itself. For example, 371 is an Armstrong number because `(3 ** 3) + (7 ** 3) + (1 ** 3) = 371`. Write a Python function called `Armstrong_number` that takes in as parameters two positive integers and returns a list containing all the Armstrong numbers between and including these two integers. Next, write main code that takes as input from the user two integers. If the user gives bad input, continue to prompt them to try again until they enter two integers that are both positive values. Then, call your function and print the resulting list. **Don't forget to document your function!**

Example output (bad input):
```
Enter an integer: -1
Need a positive integer: 0.5
Bad input! Try again: 4
Enter another integer: 2.7
Bad input! Try again: -29
Need a positive integer: 29
Armstrong numbers: [4, 5, 6, 7, 8, 9]
```

Example output (good input):
```
Enter an integer: 100
Enter another integer: 400
Armstrong numbers: [153, 370, 371]
```

65. A *perfect number* is a positive integer greater than 1 that is equal to the sum of its proper divisors. The smallest perfect number is 6 since the sum of the proper divisors for 6 (1, 2, and 3) equals 6. The integer 28 is also a perfect number since the divisors for 28 are 1, 2, 4, 7, and 14 and the sum of these divisors equals 28. Write a Python function that takes as input a positive integer greater than or equal to 1, and returns `True` if the number is a perfect number or `False` if it is not. Next, write main code that takes as input from the user one integer greater than or equal to 1. If the user gives bad input, continue to prompt them to try again until they enter a positive integer. Then, call your function and print if the number is a perfect number (or not). **Don't forget to document your function!**

Hint: The proper divisors of a positive integer N are those numbers, other than N itself, that divide N without remainder. For N > 1 they will always include 1, but for N == 1 there are no proper divisors. The proper divisors of 6 are 1, 2, and 3. The proper divisors of 100 are 1, 2, 4, 5, 10, 20, 25, and 50.

Example output (bad input):
```
Enter an integer: -5
Need a positive integer: 0.5
Bad input! Try again: a
Bad input! Try again: 28
28 is a perfect number
```

Example output (good input):
```
Enter an integer: 1
1 is not a perfect number
```

Example output (good input):
```
Enter an integer: 6
6 is a perfect number
```

66. The function below is very important in probability and statistics. Write a Python function that takes in as parameters individual values for μ and σ and a list of values for x and returns a list of the values of the function evaluated at those values. **Don't forget to document your function!**

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$$

In your main code, call your function, then plot the results. Be sure to include a title and axis labels and show the plot.

67. The Taylor series to approximate $\cos(x)$ for all values of x is defined as

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$$

Write a Python function named `tay_term` that takes in as parameters values for n and x, then calculates and returns the value of the term for **only** that value of n. For example, if the values $n = 3$ and $x = 0.5$ are passed to the function, it would return

$$\frac{(-1)^3}{(2 \cdot 3)!} 0.5^{2 \cdot 3} = -0.00002170138$$

In your main code take as input from the user a value for x, then approximate the value for cosine of x by starting at $n = 0$ and repeatedly call the function `tay_term` with increasing values of n, and sum those returned values until the absolute value returned by the function is less than 10^{-7}. Display your result with seven (7) decimal places. **Don't forget to document your function!**

Example output:

```
Enter a value for x: 0.5
The cosine of 0.5 is 0.8775825
```

68. Without knowing the contents of `fileID.txt` and after executing the following three lines of code, write an additional line of code that writes the first character and last character of `file_string` to `myfile` separated by a colon (`:`) and followed by a new line.

```
with open("fileID.txt", "r") as file1:
    file_string = file1.read().strip()
with open("output_file.txt", "w") as myfile:
    ## your additional line of code goes here ##
```

69. Write a Python program that takes as input from the user a filename, opens the file, reads the contents, and counts how many times each letter appears, then prints the results to the screen. Combine uppercase and lowercase letters, and only print the letters that appear in the file. This problem is good practice for using dictionaries.

    Example output:
    ```
    Enter a filename: sample.txt
    sample.txt contains the following letters:
    a: 11, c: 6, d: 7, e: 8, f: 1, g: 2, h: 3, i: 2, l: 1, m: 4, n: 9, o: 8,
    p: 2, r: 8, s: 4, t: 9, u: 4, w: 2, y: 1
    ```

70. Write a Python program that will take as input from the user grades from a recent ENGR 102 quiz, save it to a file named `grades.txt`, and print the average quiz grade to the screen. Have your program prompt the user for the number of students first, and then take as input that many names and their corresponding scores. You may assume the user provides valid input. Write your code using standard file I/O commands. Use the format shown below for your output file, which includes a header line and aligned columns.

    Example output:
    ```
    Enter the number of students: 5
    Enter the next name and score: Amari 85
    ...
    The average grade for this quiz is 81.4
    ```

    Example `grades.txt` file:
    ```
    Name      Score
    Amari     85.0
    Blake     76.5
    Cameron   89.0
    Dylan     59.5
    Emerson   97.0
    ...
    ```

71. A file named `item_cost.dat` is stored on a computer's hard drive. In a text editor the file displays the first few lines of the file below. The rest of the file is not shown. Write a Python program that reads the contents of this file, determines the most expensive item and prints the information about that item and its cost to the screen using the format shown below. Write your code using standard file I/O commands; *do not use* the csv module or any csv reader available in some Python package. *Do not use* any built-in sorting functions. You may assume that all costs are unique (no two items cost the same).

    Contents of `item_cost.dat`:
    ```
    Item,Cost ($)
    Item1,5.75
    Item2,3.50
    Item3,2.75
    Item4,4.50
    ...
    ```

    Example output:
    ```
    The most expensive item is Item67 and costs $8.19
    ```

72. A file named `grades.txt` contains exam grades. There is no header and each grade is on a new line in the file (the file contains only numbers). Write a Python program that opens the file, calculates the average, maximum, and minimum scores, then prints the information to the screen. Write your code using standard file I/O commands; *do not use* the csv module or any csv reader available in some Python package.

73. A sentence is a set of words that are separated by a single space with no leading or trailing spaces. The file `sentences.txt` contains many sentences, one on each line. A sample is shown below, the entire file is over one hundred lines long. Write a Python program to read the contents of this file and print the maximum number of words that appear in a single sentence.

    Contents of `sentences.txt`:
    ```
    This is a sentence.
    This is also a sentence!
    The quick brown fox jumped over the lazy dog.
    How much wood can a woodchuck chuck if a woodchuck could chuck wood?
    ...
    ```

    Example output:
    ```
    The maximum number of words in a single sentence is 29
    ```

74. A text file named `data.dat` is stored on a computer's hard drive. In a text editor the file displays the snippet below. The entire file is over one hundred lines long.

    Contents of `data.dat`:
    ```
    # Created on November 5, 2025
    # Time,Temperature,Windspeed
    # (min),(deg F),(knots)
    0,33.47,1.27
    5,32.59,1.95
    10,33.62,0.76
    15,33.79,1.12
    ...
    ```

    Write a Python program that reads the contents of this file, assigns the header lines to a variable that is a list of strings, and assigns the data to a variable that is a floating point `numpy` array that contains all of the data in the file. (The data should be in an n x 3 array.) Then, find and print the minimum temperature and maximum windspeed recorded in the file. Write this code using standard file I/O commands; *do not use* the csv module or any other csv reader that you may know of in some Python package.

    Example output (using the entire file):
    ```
    Minimum temperature is 31.28 F
    Maximum windspeed is 7.82 knots
    ```

75. A file named `peanut.dat` is stored on a computer's hard drive. In a text editor the file displays the first few lines of the file below. The rest of the file is not shown. Write a Python program that reads the contents of this file and prints the number, major diameter, minor diameter, and diameter ratio of each peanut. Format the output as shown below. You MUST use f-strings; do NOT use the `round()` function. Write your code using standard file I/O commands; *do not use* the csv module or any csv reader available in some Python package.

Contents of `peanut.dat`:

```
Number    MajorDia    MinorDia
     1       17.61        5.46
     2       16.47        5.87
     3       18.08        6.12
...
```

Example output:

```
Number MajorDia MinorDia Ratio
   1     17.61    5.46   3.23
   2     16.47    5.83   2.81
   3     18.08    6.12   2.95
...
```

76. It's flounder fishing season! The Fishing Pier Company is keeping track of each fisherman's catch in a file named `fishtotals.txt`. A sample is shown below, the entire file is over 100 lines long.

Contents of `fishtotals.txt`:

```
Fisherman A: catfish 12 inches, tarpon 20 inches
Fisherman B: trout 15 inches, flounder 20 inches
Fisherman C: flounder 12 inches, redfish 20 inches, flounder 16 inches
...
```

Write a Python program that reads the contents of `fishtotals.txt`, identifies each flounder caught along with its size, and writes the information to a new file named `flounder.txt`. At the end of the file include the total number of flounders caught. Write your code using standard file I/O commands; *do not use* the csv module or any csv reader available in some Python package.

Contents of `flounder.txt`:

```
Fisherman B: flounder 20 inches
Fisherman C: flounder 12 inches, flounder 16 inches
...
Total: 102 flounders caught
```

77. Draw the plot produced by the following code:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-2, 2, 25)
y1 = x
y2 = x ** 2
plt.plot(x, y1, "g", linewidth = 3)
plt.plot(x, y2, "k", marker = "o", markerfacecolor = "b")
plt.axis([-2, 2, -2, 4])
plt.xlabel("x")
plt.ylabel("f(x) ")
plt.title("Plots for 2 polynomials")
plt.legend(["straight", "curved"], loc = "upper center")
plt.show()
```

Short answer problems for studying:

These review questions are to see how well you understand the concepts. There won't be any short answer problems like this, but if you can answer them, you have a good understanding of the material. If you struggle with any of these questions, keep studying that topic!

78. List 5 good coding practices that have been mentioned in this course.

79. In the following dictionary, identify the keys and values. Write the command to add the key `"Orange"` with value `1` to the dictionary. Write code to loop through the dictionary and check for a particular key or value.
```
mydict = {"Apple" : 3, "Pear" : 5, "Banana" : 2}
```

80. Explain how dictionaries are different from lists, and how lists are different from tuples.

81. Identify and explain 3 ways strings are similar to lists.

82. List all of the data types we have seen in this course and provide an example of each. Identify which ones are mutable and which are immutable.

83. In your own words, explain the difference between the top-down and bottom-up design methods. Name at least one advantage and one disadvantage for each.

84. Describe the various components of a hierarchy. How can we use one in program design?

85. Does it matter if we use the same variable names inside a user defined function and the main program? Why (not)?

86. Does it matter where in our code we write function definitions?

87. Can functions access main memory? How do we pass information into a function? Can functions return multiple values?

88. What type of coding error do you hate the most and why?

89. Explain what happens when the following code is executed: `print(int("5.5"))`

90. When you are trying to fix your code, what are some alternatives to using a debugger?

91. When would it be best to use the `myfile = open(...)` and `myfile.close()` commands for opening files as opposed to the `with open(...) as myfile:` command?

92. When opening files, what is the difference between the designators `r`, `w`, `r+`, and `a`?

93. What are the differences between `myfile.read()`, `myfile.readline()`, `myfile.readlines()`, and `list(myfile)`?

94. What command(s) are used to remove the leading and trailing whitespace in a string? To separate a string into a list of its components with a specified delimiter? Write an example using both.

95. Write the command to print the value of `pi` to 4 decimal places to the screen. Don't forget to import `pi` from the `math` module!

96. Write the command to import the `coolfunc` function from the `neatomod` module in the `funpack` package and rename it to `coolf`. Now write the command to import all functions from the `neatomod` module. What command can we use to display all of the functions inside the module?

97. What are the differences between `np.arange()`, `np.array()`, and `np.linspace()`? How many rows and columns will the matrix `np.arange(15).reshape(5, 3)` have?

98. What are the differences between `plt.plot()`, `plt.bar()`, `plt.hist()`, and `plt.scatter()`? How do you set the color of a line? The marker shapes? The axis labels? The title? The legend?

99. Why is it important to include docstrings when writing functions?

100. Please review all lecture examples and quizzes.

101. Please review zyBook examples as well as Participation (orange) and Challenge (blue) activities.

102. Please review zyBook labs marked "Optional" for more coding practice.