1. Iteration and recursion:
   - give an iterative definition for $gcd(m, n)$ and provide a walk-thru to test it for a typical value. What happens if m is smaller than n?
   - give a recursive definition for $gcd(m, n)$ and provide a stack trace to test it for a typical value. What happens it m is smaller than n?
   - give an iterative definition for $n!$ and provide a walk-thru table to text if for typical value. Test if for special cases.
   - give a recursive definition for $n!$ and implement it with a python recursive function definition. Provide a stack trace to test it for a typical value of n and for any special cases.
2. Demonstrate now to use the python module ***numpy*** by writing a simple program that will:
   - prompt the user for the degree of a polynomial
   - use the numpy function ***zeros*** to allocate memory for the polynomial, all coefficients set to zero, Explicitly set the type to ***float***.
   - prompt the user for coefficients and assign them to the polynomial array (overwrite the zeros).
   - now create a second polynomial by first prompting the user for the degree. Then starting with an empty list $p = []$ prompt the user for coefficients
   - print the polynomial.
   - one at a time and append them to the list. Now use numpy's array method to convert the list of coefficients (can have ints mixed with floats) to an array of floats. Explicitly set the type to ***float*** (i.e. don't use the default setting). Print the polynomial.
3. Here is a definition of a simple function:

   ```
   def prod(m,n):
      return m*n
   ```

   Give an equivalent definition of ***prod*** using an anonymous function. Give an example of how to use it.
4. Use the polynomial $f(x) = x^2 - 5x + 6$ to demonstrate how to use newtons method to find roots of polynomials. Use an initial guess of $x0 = 10$ and a tolerance of $tol = .1$. Use a graphical calculator to do the calculations. Write down all calculations and results.
5. Define a python class named Circle to represent circle objects. The class has the following:
   - three instance variables: r, c and a for radius, circumference and area.

- a constructor that can be called with $\boldsymbol{Circle()}$ or $\boldsymbol{Circle(r)}$ where $\boldsymbol{r}$ evaluates to a radius. If the class is called with no parameters then the the default radius of **1.0** should be used.
- a definition of $\boldsymbol{\_\_str\_\_}$ that will return a tuple with the radius, circle and area of circle object.
- an accessor method named $\boldsymbol{getRadius}$ that will return the radius as a float.
- an accessor method named $\boldsymbol{getCirc}$ that will return the circumference as a float.
- an accessor method named $\boldsymbol{getArea}$ that will return the area as a float.
- a mutater method named $\boldsymbol{putRadius}$ that will replace the objects radius with a new value passed in as a parameter. Note that the objects attributes $\boldsymbol{a}$ and $\boldsymbol{c}$ will need to be updated.
- assume that your class definition is in a module named circle.py. Include a main function in the same module that will test your class definition and all of its methods. Include code so your function named main will not conflict with with other functions named main in the modules that import your class named Circle.

6. Assume that a polynomial of degree n is represented by $\boldsymbol{P(x) = a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n}$. Define a python function named $\boldsymbol{evalPoly()}$:
   Input: $\boldsymbol{(a, deg, x)}$ an array of coefficients, the degree of the polynomial and the value to evaluate the polynomial. Assume that the array of coefficients is represented as a python list of floats.
   Output: return the the value of the polynomial evaluated at x. For example the function could be called with: $\boldsymbol{fx = evalPoly(a, n, x)}$
   Other requirements: The factoring method must be used to evaluate the polynomial.

7. Assume that a polynomial of degree n is represented by $\boldsymbol{P(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n}$. Define a python function named polyPrime():
   Input: $\boldsymbol{(a, deg)}$ an array of coefficients and the degree of the polynomial. The array of coefficients is implemented as a python list of floats.
   Output: the function should return a list of coefficients and the degree of the derivative polynomial as a python tuple. For example the function might be called with $\boldsymbol{d, dDeg = polyPrime(a, aDeg)}$
   Other requirements: the function must not modify the original list of coefficients. If the input polynomial has $\boldsymbol{deg = 0}$ then return a polynomial with degree zero and $\boldsymbol{d_0 = 0.0}$.

8. Define a function named $\boldsymbol{array2dSum()}$ that when passed an mxn list, the function returns the sum of all of the values in the list. Do not use the python built-in $\boldsymbol{sum()}$ function. Use either a double for() loop or double while() loop to iterate through the 2-dim array. A typical function call might be $\boldsymbol{tot = array2dSum(a, 3, 9)}$

9. A data file named data.dat contains real numbers representing values collected from an experiment. There are two data values in each row. Each data value can begin and/or end with any sequence of white space characters including space and tab characters. Each row in the file ends with a newline character. There may be any number of blank

lines (any white space characters ending with newline) at the end of the file but not between two rows with data values.

Write a **complete** python program that will:

- open the file for reading
- read the data values from the file one row at at a time. You must use a loop that iterates over the reference to the file object.
- find the sum of the row values and append the row-sum to a list.
- open a file named "sums.txt" for writing only and write all the values in the list to the file. Each value must on a new line.
- close both files before exiting your program.
- additional requirements:
  - your program must have a function named main().
  - your program must define and call at least one function other than main()

10. Define a python function named **listMin()**:
Input: a python list of numbers
Output: return the minimum value in the list. Example: **mn = listMin(a)**
Other requirements: You must use a for loop or a while loop to find the minimum. You cannot use the python min() function.

11. Define a function named **newList2d** that when passed the dimensions of a 2-dim list, mxn, it will use a nested list comprehension to allocate memory for the mxn list, initialize the list to zeros, and return the list. For example after executing **A = newList2d(2,3)**, ***A*** will be a reference to a list with 2 rows and 3 columns with all elements set to zero.

12. Define a function named **rowSum** with:
Input: $\boldsymbol{a, m, n}$ where $\boldsymbol{a}$ is a 2-dim list with $\boldsymbol{m}$ rows and $\boldsymbol{n}$ columns.
Output: 1-dim list containing the sums of the rows of list $\boldsymbol{a}$.
Note, the function will need to allocate a new list for the row sums.
Typical call: **b = rowSum(a,33,45)**

13. Define a function named **colSum** with:
Input: $\boldsymbol{a, m, n}$ where $\boldsymbol{a}$ is a 2-dim list with $\boldsymbol{m}$ rows and $\boldsymbol{n}$ columns.
Output: 1-dim list containing the sums of the columns of list $\boldsymbol{a}$.
Note, the function will need to allocate a new list for the column sums.
Typical call: **c = colSum(a,33,45)** where a has 33 rows and 45 columns.

14. Define a python function named **listMax()**:
Input: a python list of numbers
Output: return the maximum value in the list. Example: **mn = listMax(a)**
Other requirements: You must use a for loop or a while loop to find the maximum. You cannot use the python max() function.

15. Define a function named **rowMul** with:
Input: $\boldsymbol{a, b, m, n}$ where $\boldsymbol{a}$ is a 2-dim list with $\boldsymbol{m}$ rows and $\boldsymbol{n}$ columns and $\boldsymbol{b}$ is a one dimensional list with $\boldsymbol{m}$ elements. Note, memory has already been allocated for $\boldsymbol{b}$ but you do not know if it's elements are preset to zero.
Output: On return the original list $\boldsymbol{b}$ will contain the products of the rows of $\boldsymbol{a}$.
Typical call: **rowSum(a,b,33,45)**

16. The first row of a file data file named data.dat is an integer that is the number of real values that are on the remaining rows of the file. There is one number per row.

    Write a **complete** python program that will:

    - open the file for reading
    - read the integer from the first row and save it in a variable named $\boldsymbol{n}$
    - create a list large enough to hold the $\boldsymbol{n}$ data values in the file
    - read the data values from the file into the list
    - find the sum of all values in the file and print the sum to the screen along with an appropriate message.
    - close the file when finished
    - additional requirements:
        - your program must have a function named main().
        - your program must call at least one function other than main()

17. Here is a program to test a gcd function that does not work (named badgcd()). (a) Do a hand calculation to find the correct value for gcd(18,30). (b) Use a table to do a walk-thru for the function call in main to badgcd(). Make sure your walk-thru table shows what the function actually does and not what you think it should do.

```
#!/usr/bin/env python
def badgcd(a,b):
    while(b!=0):
        a=b
        r=a%b
        b=r
        print a,b,r
    return r
def main():
    a = 18
    b = 30
    g = badgcd(a,b)
    print a,b,g
main()
```

18. Give a good function definition for gcd() to replace badgcd() in the above code and repeat the walk-thru for the call gcd(18,30)

19. Assume that integers are stored in one byte memory locations.

    (a) If the integers were stored as unsigned integers, what are the largest and smallest values that can be stored? You can give your answers in powers of two. No need for a calculator.

    (b) If the integers were stored as signed integers, what are the largest and smallest values that can be stored?

20. Assume that integers are stored as signed integers in one byte memory locations.

    (a) Give the binary form for the number 16. You must show all 8 bits.

(b) Give the binary form for the number (-6). Assume that the two's compliment is used to represent negative values. You must show all 8 bits.

(c) Demonstrate that when you add the two 8-bit binary numbers that the resulting binary sum is what you would expect from the difference **16 − 6** in base 10.

(d) Now assume that the signed integers are stored in 4-bit memories (and not 8-bit). Demonstrate and explain what overflow is by converting the base 10 numbers to 4-bit binary numbers and adding them.

21. Convert the base two number **10011110** to:

(a) base 16

(b) base 8

(c) base 10 , no need for a calculator.

(d) convert the base ten number **13** to base **2** by repeated division by **2** and collecting the remainders. Demonstrate that your answer is correct by converting it back to base 10.