LAB 01: Revision of Basic Programming Constructs and ADTs

CS211 – Data Structures and Algorithms Usman Institute of Technology

Fall 2021

* **How to submit:**
  + Submit lab work in a **single** .py file on MS Teams. (No other format will be accepted)
  + Lab work file name should be saved with your roll number (e.g. 19b-001-SE\_LW.py)
  + Submit home work in a **single** .py file on MS Teams. (No other format will be accepted)
  + Lab work file name should be saved with your roll number (e.g. 19b-001-SE\_HW.py)

**Write functions in Python whose parameters and return value are given below.**

1. Write a short Python function, **is multiple(n, m)**, that takes two integer values and returns True if is a multiple of , that is, for some integer , and False otherwise.
2. Write a short Python function, **is even(k)**, that takes an integer value and returns **True** if is even, and **False** otherwise. However, your function cannot use the multiplication, modulo, or division operators.

**def is\_multiply**(m,n):

// your code goes here

**def is\_even**(k):

// your code goes here

1. Create a function **EvenList** that takes a parameter **n** to input **n** number from users and returns the list of only even numbers.

**def EvenList**(n):

// your code goes here

Example: if user gives the input 1,2,3,4,5,6 then list contains only 2,4,6

1. Write a short Python function, **minmax(data)**, that takes a sequence of one or more numbers, and returns the smallest and largest numbers, in the form of a tuple of length two. Do not use the built-in functions min or max in implementing your solution.

**def minmax**(data):

// your code goes here

1. Write a short Python function that takes a positive integer and returns the sum of the squares of all the positive integers smaller than .

**def sumsquares():**

// your code goes here

1. Write a short Python function that takes a positive integer and returns the sum of the squares of all the odd positive integers smaller than n.

**def sumoddsquares(k):**

// your code goes here

1. Write a short Python function that takes a sequence of integer values and determines if there is a distinct pair of numbers in the sequence whose product is odd.

**def distinctoddpairgen(array):**

// your code goes here

1. Create a function **Reverse** that takes a list as a parameter and returns a list which is reverse of the original list.

**def Reverse(list):**

// your code goes here

Example: if given list is [2,4,6] then function must return [6,4,2]

1. Create a function **Unique** that takes a list as a parameter and returns a list containing only unique elements i.e. duplicate elements should be removed.

**def Unique(list):**

// your code goes here

Example: if given list contains [2,4,4,6,6] then function must return [2,4,6]

1. Create a function **UserNumbers** that takes 10 number as input from the users but stores only even numbers. The function also prints the following from given number.
   1. The last element of the list
   2. The maximum value
   3. The minimum value
   4. The second last element of the list

**def UserNumbers(list):**

// your code goes here

1. Create a function **ShowExcitement** that returns the string “A quick brown fox jumps over the lazy dog” 5 times. Make sure to separate the sentence with space every time. Don’t copy paste the sentence 5 times.

**def ShowExcitement(list):**

// your code goes here

1. Create a function **Greater** which takes three numbers as parameters and returns the largest numbers.

**def Greater(n1, n2, n3):**

// your code goes here

1. Create a function **Divide** that takes two numbers, *dividend* and *divisor*, as parameters and returns the *quotient* and *reminder*.

**def divide**(dividend, divisor):

// your code goes here

**Quotient:** In arithmetic, a quotient is the quantity produced by the division of two numbers. **Remainder:** In mathematics, the remainder is the amount "left over" after performing some computation

Example: Divide(10,3) must return (3,1) because 3 is quotient and 1 is remainder.

1. Create a class **Person** which takes two parameters to initialize: name and age. The class should also have a function **birthday()** which increases the age by 1 year.

**class Person**:

**def init** (self, name, age):

// your code goes here

**def birthday**():

// your code goes here

**Home Task**

* 1. A click counter is a small hand-held device that contains a push button and a count display. To increment the counter, the button is pushed and the new count shows in the display. Clicker counters also contain a button that can be pressed to reset the counter to zero. Design and implement the Counter ADT that functions as a hand-held clicker.
  2. A Grab Bag ADT is similar to the Bag ADT with one difference. A grab bag does not have a remove() operation, but in place of it has a grabItem() operation, which allows for the random removal of an item from the bag. Implement the Grab Bag ADT.
  3. A Counting Bag ADT is just like the Bag ADT but includes the numOf(item) operation, which returns the number of occurrences of the given item in the bag. Implement the Counting Bag ADT and defend your selection of data structure.
  4. The use of the Student File Reader ADT makes it easy to extract student records from a text file no matter the format used to store the data. Implement a new version of the ADT to extract the data from a text file in which each record is stored on a separate line and the individual fields are separated by commas. For example, the following illustrates the format of a sample file containing three student records:

10015, John, Smith, 2, 3.01

10334, Jane, Roberts, 4, 3.81

10208, Patrick, Green, 1, 3.95

* 1. In the lecture 1, we defined and implemented the Student File Reader ADT for extracting student records from an external source. We can define and use a similar ADT for output.

1. Design a Student File Writer ADT that can be used to display, or store to an output device, student records contained in a Student Record object.
2. Provide an implementation of your ADT to output the records by displaying them to the terminal in a neatly formatted fashion.
3. Provide an implementation of your ADT to output the records to a text file using the same format described in the text.
4. Design and implement a complete program that extracts student records from a text file, sorts them by either student id or student name, and displays them to the terminal using your ADT. The choice of sort keys should be extracted from the user.
   1. A line segment is a straight line bounded by two endpoints. The Line Segment ADT, whose operations are described below, represents a line segment defined by points in the two-dimensional Cartesian coordinate system. Use the Point class from Appendix D and implement the Line Segment ADT.

* LineSegment(ptA,ptB): Creates a new Line Segment instance defined by the two Point objects.
* endPointA(): Returns the first endpoint of the line.
* endPointB(): Returns the second endpoint of the line.
* length (): Returns the length of the line segment given as the Euclidean distance between the two endpoints.
* toString (): Returns a string representation of the line segment in the format (Ax, Ay)#(Bx, By).
* isVertical(): Is the line segment parallel to the y-axis?
* isHorizontal(): Is the line segment parallel to the x-axis?
* isParallel(otherLine): Is this line segment parallel to the otherLine?
* isPerpendicular(otherLine): Is this line segment perpendicular to the otherLine?
* intersects(otherLine): Does this line segment intersect the otherLine?
* bisects(otherLine): Does this line segment bisect the otherLine?
* slope(): Returns the slope of the line segment given as the rise over the run. If the line segment is vertical, None is returned.
* shift(xInc,yInc): Shifts the line segment by xInc amount along the x-axis and yInc amount along the y-axis.
* midpoint(): Returns the midpoint of the line segment as a Point object.