A Summer Internship Project Report on

Data Structures with Python

Submitted in the partial fulfillment for the

Award of Degree in

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING
TO

Rajiv Gandhi University of Knowledge Technologies - SKLM

Submitted by

4th Year B. Tech 1st Semester

K.Monika Ganga S180148

Under The Esteemed Guidance of

Mr. T. Anil Kumar Asst.prof, M.Tech(ph.D)

Department of CSE
S.M. Puram (V), Etcherla (M), Srikakulam (Dt) – 532410

2018-2024

CERTIFICATE

This is to certify that the report entitled "**Data structures with python**" was successfully completed by Ms. "Karri Monika Ganga(s180148)" in partial fulfillement of the Requirements for the summer internship project in computer science and engineering of Rajiv Gandhi University Of Knowledge Technologies is a record bonafide work carried out by her.

The result embodided in this report have not been submitted to any other university for the award of any degree.

Mr.T.Anil Kumar,Asst.Prof,M.tech,(ph.D)
Summer Internship Review Guide
RGUKT SRIKAKULAM

Mrs.Ch.LakshmiBala,Asst.Prof,
Head of the Department CSE
RGUKT SRIKAKULAM

INTERNSHIP CERTIFICATE

UNIVERSITY OF MICHIGAN

COURSE CERTIFICATE

Aug 7, 2023

Monika Ganga Karri

has successfully completed

Python Data Structures

an online non-credit course authorized by University of Michigan and offered through Coursera

Charles Severance
Clinical Professor, School of Information
University of Michigan

Verify at:
https://coursera.org/verify/7Z342UMY8TK4

Coursera has confirmed the identity of this individual and their participation in the course.

DECLARATION

I declared that this thesis Course titled **"Data Structures with Python"** is carried out by us during the year 2023 in fulfillment of the requirements for the Summer Internship Project in **Computer Science and Engineering.**

I further declare that this dissertation has not been submitted elsewhere for any Degree. The matter embodied in this dissertation report has not been submitted elsewhere for any other degree.The work contained in the project report is original and has been done by myselves under the guide. Furthermore, the technical details furnished in various chapters of this thesis are purely relevant to the above project and there is no deviation from the theoretical point of view for design, development and implementation.

K.Monika Ganga [S180148]

ACKNOWLEDGEMENT

We would like to articulate my profound gratitude and indebtedness to my project guide **Mr. T.Anil Kumar Assistant Professor,M.Tech,(ph.D)** who has always been a constant motivation and guiding factor throughout the project time. It has been a great pleasure for us to get an opportunity to work under her guidance and complete the thesis work successfully.

We wish to extend my sincere thanks to Ch.LakshmiBala, Head of the Computer Science and Engineering Department, for his constant enouragement throughout the project.

We thank one and all who have rendered help to me directly or indirectly in the completion of my thesis work. We are also grateful to other members of the department without their support my work would have been carried out so successfully.

Project Associate

K.Monika Ganga [S180148]

ABSTRACT

Data structures are essential tools in computer science, and Python offers a diverse range.

This overview covers key Python data structures:

Lists are Ordered, mutable collections Tuples are Immutable collections.

Sets are Unordered, unique value collections.

Dictionaries are Key-value pairs for efficient data retrieval.

Stacks and Queues are Linear structures for specific ordering.

Linked Lists are Dynamic structures for efficient insertions and deletions.

Trees and Graphs are Hierarchical and complex relationship representations.

Heaps are Specialized trees for priority queues.

Understanding these structures is vital for efficient Python programming, as they enable developers to choose the right tool for various tasks, improving code efficiency.

Keywords:

Data Structures

Python

Lists

Tuples

Data Management

Table of Contents

1.INTRODUCTION

1.1Introduction

Data Structures with Python" on Coursera is an informative and comprehensive course designed to teach students the essential concepts of data structures using the Python programming language. This course provides a solid foundation in fundamental data structures such as lists, stacks, queues, and trees, and demonstrates how to implement and use them effectively in Python. Through hands-on coding exercises and practical examples, participants learn how to optimize data storage, retrieval, and manipulation, making it an invaluable resource for both beginners and experienced Python programmers.
Whether you're looking to enhance your programming skills or gain a deeper understanding of data structures, this Coursera course is a valuable step towards mastering Python and computer science fundamentals.

1.2 Statement of the problem

In the realm of computer science education, there exists a prevalent issue where learners face challenges in comprehending and practically applying data structures within the Python programming language. This discrepancy points to a pressing need for an inclusive and hands-on online course that not only elucidates the core concepts of data structures but also empowers students with the practical skills required to implement them effectively in Python-based programming scenarios.

1.3 Objective

Conceptual Clarity: Ensure a solid grasp of Python data structures such as lists and dictionaries.

Practical Proficiency: Empower students to use data structures effectively in real-world Python projects.

Efficient Coding: Teach optimization techniques for writing efficient Python code.

1.4 Goals

- Accurate Categorization
- To create a high-quality labeled dataset of news headlines
- Learning Opportunity

1.5 scope

The scope of data structures in a Python course typically includes introducing fundamental data structures and their operations to efficiently store and manipulate data. Topics covered usually include lists, tuples, dictionaries, sets, and their various methods, as well as an understanding of when to choose one data structure over another based on specific use cases.
This scope equips students with the ability to work with different data types effectively and optimize their code for performance.

Week-1 Task:

Week-1 we learn about the importing the libraries and basic data structures.

Important Reading: Using Python in this Class

We strongly encourage you to install Python on your computer if you have a desktop or laptop. There are even some Python applications available for iPhone and Android phones.

But if you do not have a computer where you can install Python, you can still complete this class because we have a version of Python that runs in your browser that is sufficient to do the assignments for this class.

You will see various links that allow you to develop, turn in and auto-grade each of the programming assignments throughout the class. You can also use the "Python Playground" to experiment with writing your own Python applications using only your browser.

We feel that you learn the most if you develop your applications using the actual Python environment and then use our browser-based system to turn in your applications after they are completed. But we do understand that not everyone can install Python on their computers.

This is the last course in our "Python for Everybody" specialization before you are required to use Python to run your programs. You can complete **this** course without a desktop or laptop computer but for the **next** course in the specialization, you will be required to install and use Python on your own computer as the assignments become more

Run Python | Reset Code

```
fh = open("romeo.txt", "r")

count = 0
for line in fh:
    print(line.strip())
    count = count + 1

print(count,"Lines")
```

Week1 we learn about the file operations and basic programs about the file handling process.

Week-2 tasks:

7.1 Write a program that prompts for a file name, then opens that file and reads through the file, and print the contents of the file in upper case. Use the file **words.txt** to produce the output below.

You can download the sample data at http://www.py4e.com/code3/words.txt

Check Code | Reset Code

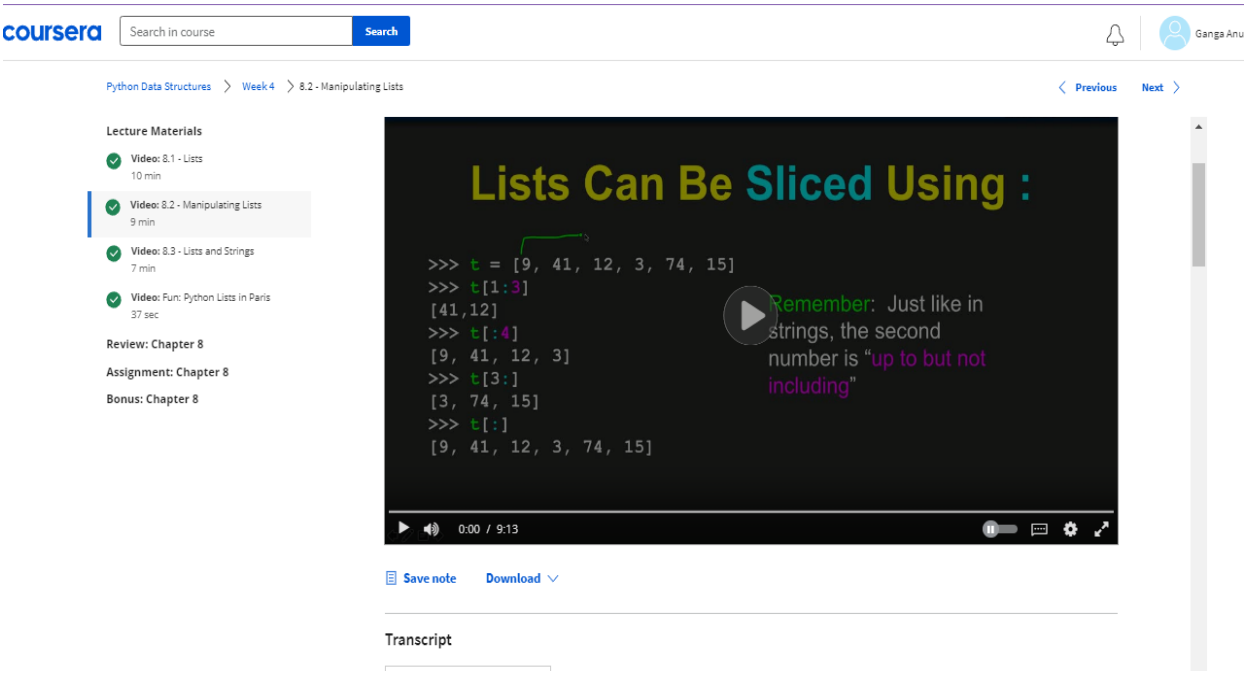
```
# Use words.txt as the file name
fname = input("Enter file name: ")
fh = open(fname)
for line in fh.readlines():
    print(line.upper().strip())

```

Week2 we learn about the file operations programs regarding the files in upper case.

Week-3 tasks:

Dictionary: A dictionary is a data structure in programming that stores a collection of key-value pairs, allowing efficient retrieval of values based on their associated keys.

Tuples: Tuples, on the other hand, are ordered collections of elements in Python that are immutable, meaning their values cannot be changed after creation, making them suitable for representing fixed data sets.



Week-3 tasks we learn about the lists and tuples and dictionary operations.

Week-4 Tasks:

Linked List: A linked list is a linear data structure in programming where elements, called nodes, are connected through pointers, facilitating dynamic insertion and deletion operations with O(1) complexity for certain cases.

Double Linked List: A doubly linked list extends the concept of a linked list by equipping each node with pointers to both its previous and next nodes, enabling efficient traversal in both directions at the cost of increased memory usage compared to a singly linked list.

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class LinkedList:
    def __init__(self):
        self.head=None
        self.tail=None
    def add_element_begin(self,data):
        new_node=Node(data)
        if(self.head==None):
            self.head=new_node
            self.tail=new_node
        else:
            new_node.next=self.head
            self.head=new_node

    def add_end(self,data):
        new_node=Node(data)
        self.tail.next=new_node
        new_node=self.tail

    def add_middle(self,data,value):
        new_node=Node(data)
        temp=self.head
        while temp is not None and temp.data != value:
               temp = temp.next

        if temp is not None:
             new_node.next = temp.next
             temp.next = new_node
        else:
             print("No element is found in the specific location")


    def add_middle_before(self,data,value):
        new_node=Node(data)
        temp=self.head
        rtemp=None
        while(temp is not None and temp.data is not  value):
```

```
def add_middle_before(self,data,value):
    new_node=Node(data)
    temp=self.head
    rtemp=None
    while(temp is not None and temp.data is not  value):
        rtemp=temp
        temp=temp.next


    if (temp!=None):
        new_node.next=rtemp.next
        rtemp.next=new_node

    else:
        print("No element is found in the specific location")

def del_at_first(self):
    self.head=self.head.next

def del_at_end(self):
    temp=self.head
    prev=None
    while(temp.next!=None):
        prev=temp
        temp=temp.next
    self.tail=prev
    self.tail.next=None

def del_at_middle(self,value):
    temp=self.head
    prev=None
    while(temp.data!=value):
        prev=temp
        temp=temp.next
    prev.next=temp.next
```

```
list=LinkedList()
list.add_element_begin(50)
list.add_element_begin(40)
list.add_element_begin(30)
list.add_element_begin(20)
list.add_element_begin(10)
list.add_middle_before(70,30)
list.add_middle(5,10)
list.add_end(60)
list.del_at_first()

list.traverse()
list.del_at_end()
list.traverse()
list.del_at_middle(30)
list.traverse()
```

Double Linked List:

```
class node:
    def __init__(self,data):
        self.data=data
        self.prev=None
        self.next=None
class doublelinkedlist:
    def __init__(self):
        self.head=None
        self.tail=None
    def add_begin(self,data):
        new_node=node(data)
        if(self.head==None):
            self.head=new_node
            self.tail=new_node
        else:
            self.head.prev=new_node
            new_node.next=self.head
            self.head=new_node

    def add_end(self,data):
        new_node=node(data)
        temp=self.head
        while(temp.next!=None):
            temp=temp.next
        temp.next=new_node
        new_node.next=temp
        self.tail=new_node
        self.tail.next=None

    def add_middle(self,data,value):
        new_node=node(data)
        temp=self.head
        while( temp!= None and temp.data!=value):
            temp=temp.next
        new_node.next=temp.next
        if(temp.next!=None):
            temp.next.prev=new_node
```

```
        self.tail.next=None

    def add_middle(self,data,value):
        new_node=node(data)
        temp=self.head
        while( temp!= None and temp.data!=value):
            temp=temp.next
        new_node.next=temp.next
        if(temp.next!=None):
            temp.next.prev=new_node
        new_node.prev=temp
        temp.next=new_node
    def del_at_begin(self):
        self.head=self.head.next
        self.head.prev=None

    def del_at_end(self):
        temp=self.head
        rtemp=None
        while(temp.next!=None):
            rtemp=temp
            temp=temp.next
        rtemp.next=None

    def del_at_middle(self,value):
        temp=self.head
        rtemp=None
        while(temp.data!=value):
            rtemp=temp
            temp=temp.next
        rtemp.next=temp.next
        temp.next.prev=rtemp
```

```
    def del_at_middle(self,value):
        temp=self.head
        rtemp=None
        while(temp.data!=value):
            rtemp=temp
            temp=temp.next
        rtemp.next=temp.next
        temp.next.prev=rtemp



    def traverse(self):
        temp=self.head
        while(temp!=None):
            print(temp.data,'-->',end=' ')
            temp=temp.next
        print('Null')

object=doublelinkedlist()
object.add_begin(50)
object.add_begin(40)
object.add_begin(30)
object.add_begin(20)
object.add_begin(10)
object.add_begin(5)
object.traverse()
object.add_end(60)
object.traverse()
object.add_middle(80,30)
object.traverse()
object.del_at_begin()
object.traverse()
object.del_at_end()
object.traverse()
object.del_at_middle(30)
object.traverse()
```

Week-5 Task:

In Python, a stack is a fundamental data structure that follows the Last-In-First-Out (LIFO) principle. It's implemented using lists and supports two primary operations: "push," which adds an element to the top of the stack, and "pop," which removes and returns the top element. Stacks are commonly used for managing function calls, tracking program flow, and solving problems like expression evaluation and backtracking algorithms.

Stack program:

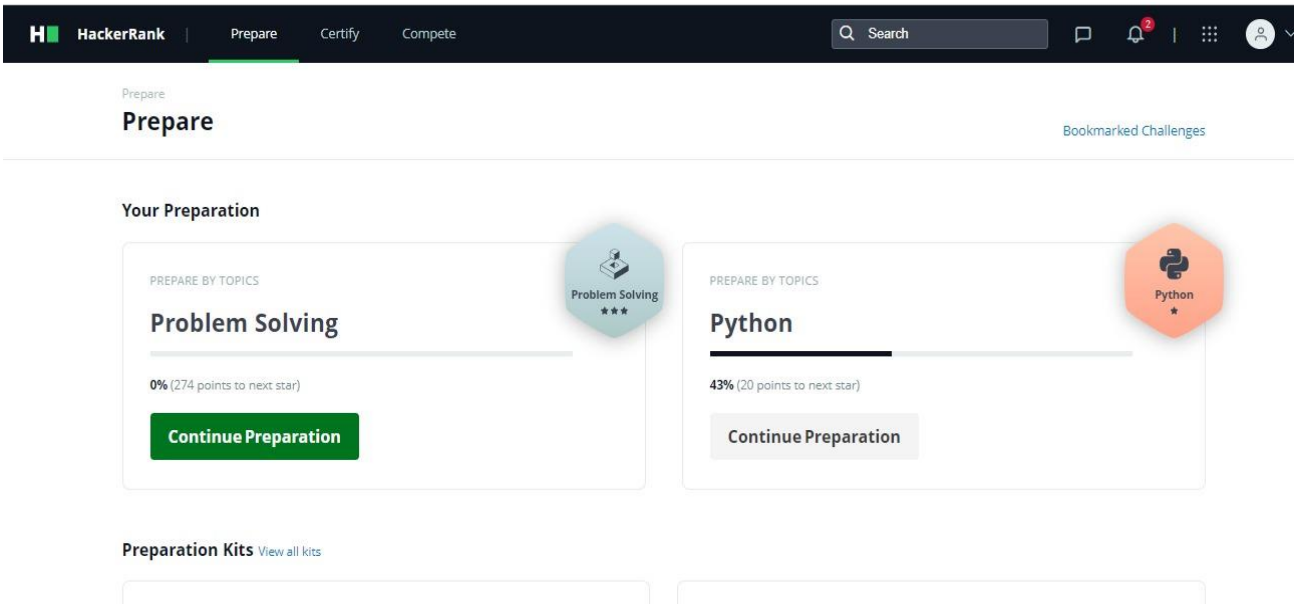
```
class stack:
    def __init__(self):
        self.items=[]
    def is_empty(self):
        return len(self.items)==0
    def push(self,items):
        self.items.append(items)
    def pop(self):
        if not self.is_empty():
            self.items.pop()
        else:
            print("stack is empty")
    def peek(self):
        if not self.is_empty():
            return self.items[-1]
        else:
            print("stack is empty")
    def size(self):
        return len(self.items)

object=stack()
object.push(1)
object.push(2)
object.push(3)
object.push(4)
object.push(5)
object.push(7)
print(object.size())
print(object.peek())
object.pop()
object.pop()
print(object.size())
```

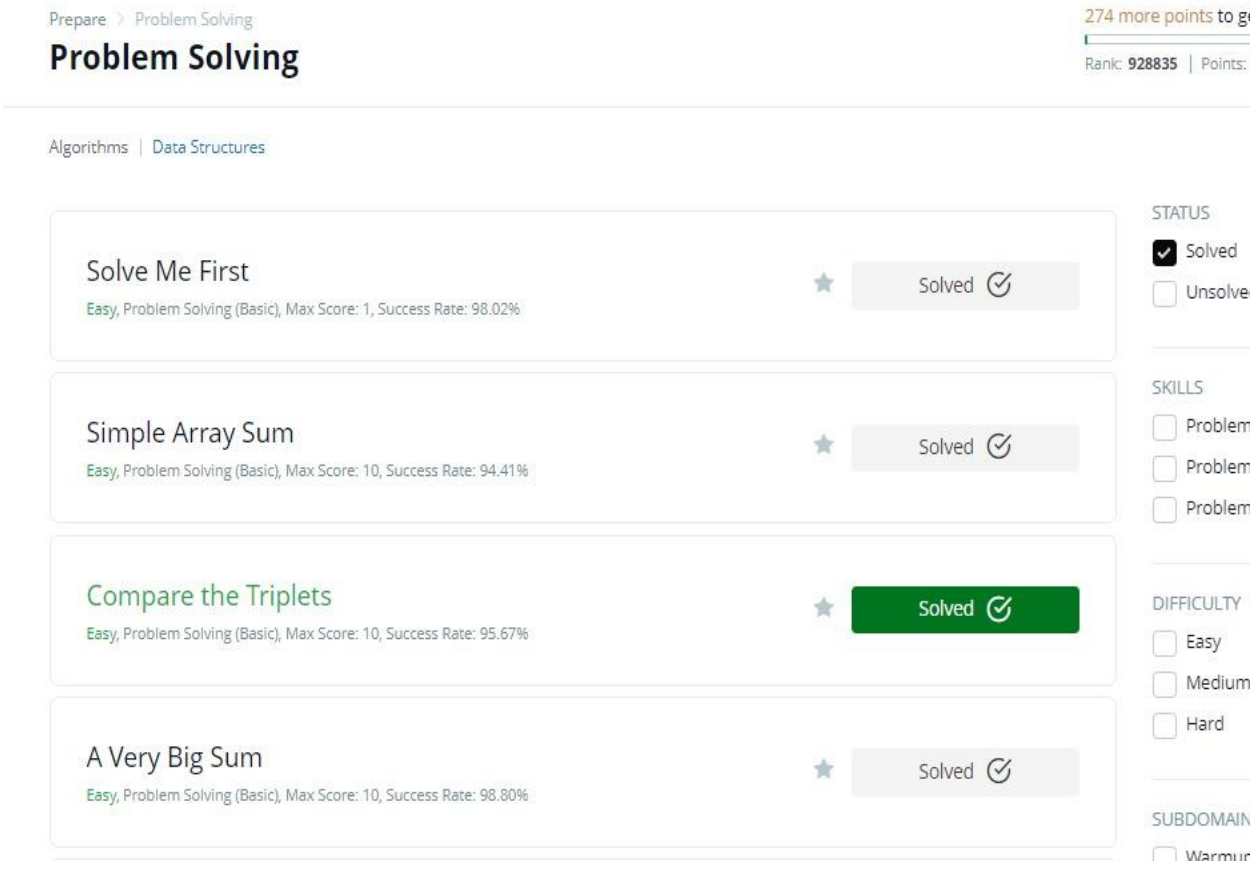
Week-6 Task:

A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle, where elements are added at the back and removed from the front. It represents a collection of items with two main operations: "enqueue" to add elements to the rear end, and "dequeue" to remove elements from the front end. This structure is often used to manage tasks in the order they were added, like in a waiting line or task processing system.

```
class queue:
    def __init__(self):
        self.items=[]
    def enqueue(self,items):
        self.items.append(items)
    def is_empty(self):
        return len(self.items)==0
    def dequeue(self):
        if  not self.is_empty():
            self.items.pop(0)
        else:
            print("queue is empty")
    def size(self):
        return len(self.items)
object=queue()
object.enqueue(2)
object.enqueue(3)
object.enqueue(4)
object.enqueue(5)
object.enqueue(6)
object.enqueue(7)
print(object.size())
object.dequeue()
object.dequeue()
object.dequeue()
print(object.size())

#Bubble sort#

a=[6,8,10,2,1,3]
for i in range(len(a)):
    for j in range(i,len(a)):
        if(a[i]>a[j]):
            a[i],a[j]=a[j],a[i]
print(a)
```

Hacker Rank problems:





Diagonal Difference

Easy, Problem Solving (Basic), Max Score: 10, Success Rate: 95.95%

Solved

Plus Minus

Easy, Problem Solving (Basic), Max Score: 10, Success Rate: 98.38%

Solved

Staircase

Easy, Problem Solving (Basic), Max Score: 10, Success Rate: 98.36%

Solved

Mini-Max Sum

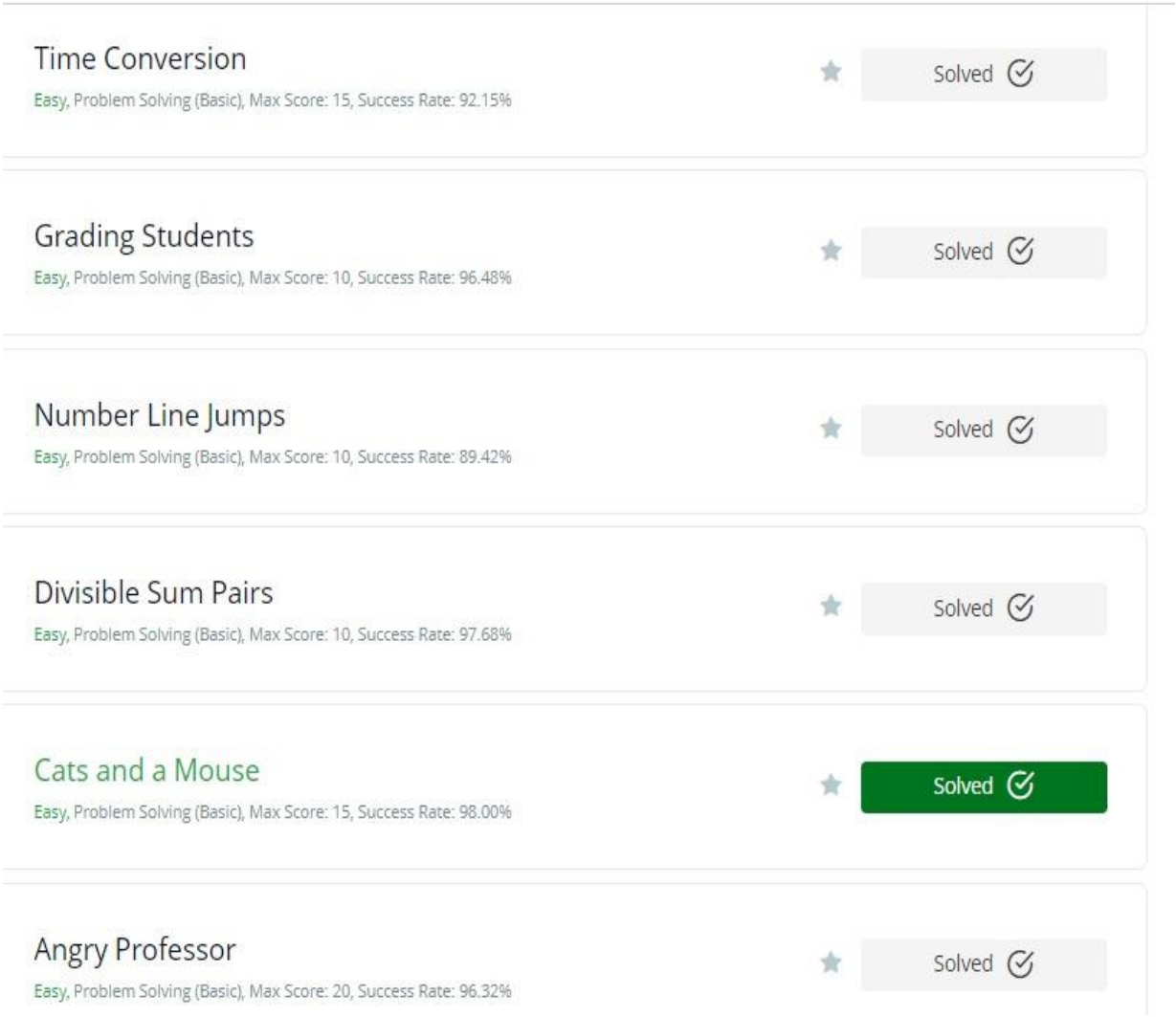
Easy, Problem Solving (Basic), Max Score: 10, Success Rate: 94.37%

Solved

Birthday Cake Candles

Easy, Problem Solving (Basic), Max Score: 10, Success Rate: 97.07%

Solved



Conclusion:

Studying data structures using Python is a rewarding pursuit. Python's simplicity and versatility make it an ideal language for learning and implementing various data structures. Mastering these structures empowers you to efficiently organize, manage, and manipulate data, fostering problem-solving skills essential for software development and computer science. Through Python, you can grasp fundamental concepts while building a strong foundation for tackling complex programming challenges with elegance and effectiveness.