GOVERNMENT POLYTECHNIC

PATNA-7

BRANCH : Computer Science and Engineering

NAME : Harsh Deep

CLASS ROLL No. : 46/CSE/21

BOARD ROLL No. : 411181221046 GROUP

SESSION : 2021 - 2024 YEAR/SEMESTER : 2nd / V

SUBJECT : Swayam (T.W) - 2018405

Professor's Signature

Name	Harsh Deep	Year	2nd
Subject	Swayam (T.W)- 2018409	~~Class~~ Branch	CSE
Semester	Fourth (IVth)	Roll No.	46/CSE/21

INDEX

INDEX

Sr. No.	Experiment Description	Experiment Date	Submission Date	Remarks / Signature
07.	Class and object in python.	23		
08.	Dynamic programming in python	24		

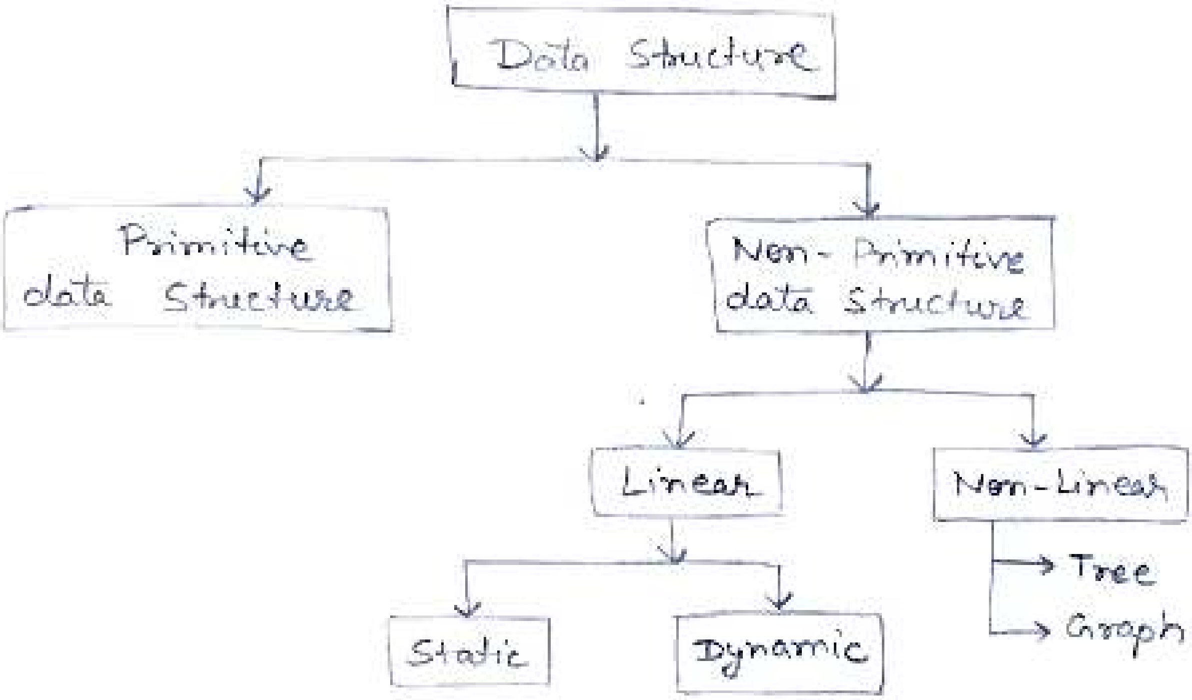


Fig : Data Structure classification

1. Introduction to programming, data structure and algorithm using python.

• Data Structure

→ Data Structure are fundamental concepts of computer science which helps in writing efficient program in any languages. Python is a high-level, interpreted, intractive and object-oriented language using which we can study the fundamentals of data structure in a simpler way as compaired to other programming languages

• Why Learn Data Structure

1> Data Structure and algorithm are two of two most important aspect to the computer science.

2> Data Structure allows as to organize and store data, why algorithm allow store process that data in a meaningful way.

3> Learning data Structure and algorithm, we help us to become a better programmer.

- **Algorithm**

→ Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithm are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

- **Characteristics of an Algorithm**

1) input : An algorithm can be given a value other than take 0 or more input.

2) output : It is released at least one quantity.

3) Difiniteness : Each instruction must be clear, well-define and precise.

4) Effectiness ~~Finiteness~~ : This means that operation must be simple and are carried out in a finite at one or more level of complexity.

5) Finiteness : Algorithm must terminate after a finite number of step.

6) **Feasibility** : Should be feasible with the available resources.

7) **Uniqueness** : For a specific task, there can be multiple algorithms, but each algorithm should provide a unique solution approach.

8) **Correctness** : An algorithm is correct if it produces the expected output for all valid inputs. Providing the correctness of an algorithm is a critical aspect of its design and analysis.

—— x ——

2. Basic of python (Data types), function

- What is python?

→ Python is a very popular general-purpose interpreated, intractive, object-oriented and high level programming language.

→ Python is open source which means its available free of cost.

→ It is simple and easy to learn.

- Python Data types

→ Data types are the classification or categorization of data items. It Identify the type of data value a variable can hold.

1) Numeric values:

→ Numeric come in two types:

(a) int (integers): It contain's positive or negative whole numbers. In python there is no limit to how long an integer value can be.

Ex : 178 , -3 , 428328946 are values of type int.

(b) float : It is a real number with floating point representation. It is specified by a decimal point.

Ex : 37.82 , -0.01 , 28.7998 are values of type float.

2> String :

→ A string is a collection of one or more characters ~~of~~ put in a single quote, double quote and triple quote.
It is represented by Str class

Ex : A = " Welcome To Patna - 7. "
print (A)

Output :

```
Welcome To Patna - 7.
```

3> Booleam

→ Data type with one of the two built in values, True or False. Booleam objects that are equal to True are truthly and those equal to False and Falsy (False).

It is denoted by the class bool.

Ex : print (type (True))

Output :

```
<class 'bool'>
```

- Function

→
```
def F (a,b,c) :
        Statement - 1
        Statement - 2
        .....
        return (variable)
        .....
```

⇒ Function name, arguments / parameters
⇒ Body is indented
⇒ return () statement exits and return a value.

Passing value to function :

* Argument value is substituted for name

```
def power (x,n) :
        ans = 1
      For i in range (0,n):
              ans = ans* x
        return (ans)
```

→ like an implicit assignment statement

Example :

```
def update (l, i, v) :
    if i>=0 and i< len(l) :
        l[i] = v
        return (True)
    else :
        v = v+1
        return (False)
```

→ Return value may be ignored.

→ If there ~~a~~ is no return (), Function ends when last statement is reached.

- Recursive function

→ A function can call itself - recursively.

```
def factorial (n):
    if n <= 0
        return (1)
    else :
        val = n* Factorial (n-1)
    return (val)
```

—x—

3. Lists, inductive function definitions

- More about range()

→ range (n) has an ~~term~~ implicity from 0 to n-1.

→ range (i, j, K) produces sequence in step of K.
Negative K counts down.

→ Sequence produced by range () is not a list.
Use list (range (..)) to get a list.

- Lists

→ This list is created by using square brackets.

Ex : D = [1, 2, 3, "GP", "Patra"]

→ List is a collection of different values or different types of item.

→ To extend list in place, use D.append (), D.extend ().

→ Can also assign new value, in place to a slice.

→ In list inserting or deleting an element is easy.

- Extending a list

→ list 1 = [1, 3, 5, 6]
list 2 = list 1
list . append (12)

~~list2 o~~ list 1 is now [1, 3, 5, 6, 12]
list 2 is also [1, 3, 5, 6, 12]

- Inductive definitions

→ Many arithmetic functions are naturally defined inductively.

→ Factorial :
$0! = 1$
$n! = n \times (n-1)!$

→ Multiplication : repeated addition
$m \times 1 = m$
$m \times n = m + (m \times (n-1))$

→ Define one or more cases
→ Inductive step definition $F(n)$ in terms of smaller arguments.

- Recursive Computation

→ Inductive definitions naturally give rise to recursive programs.

```
def factorial (n) :
        if n = = 0  :
            return (1)
        else :
        return (n* factorial (n-1))
```

——x——

4. Sorting , Tuples and Dictionaries

- **Sorting**

→ Sorting is a process of arrangeing the data in a particular formed.

→ Sorting can be done in ascending or desending order. It arrange the data in a sequence which makes searching easier.

- **Different types of Sorting in python**

1) Bubble sort
2) Merge sort
3) insertion sort
4) Selection sort

1) **Bubble sort** : It is comparison - based algorithm in which each pair of adjacent element is compared and the element are swapped if they are swapped only if they are not in order.

Example :

```
def bubblesort(list):
    for i in range (len(list) -1,0,1):
        for j in range (i) :
            if list [i] > list [j+1] :
                temp = list [j]
                list[j] = list [j+1]
                list[j+1] = temp

list = [19, 2, 31, 45, 6, 11, 121, 27]
bubble sort (list)
 print (list)
```

Output :

```
[2, 6, 11, 19, 27, 31, 45, 121]
```

- Time complexity

→ This algorithm has a worst case time complexity of $O(n^2)$.

- Space complexity

→ The bubble sort has a space complexity of $O(1)$.

2> **Merge sort** : It divides the collection into smaller sub-collections and sorts them individually.

It ~~olt~~ utilizes a divide and-conquer strategy and create a final sorted collection by merging the sorted sub-collection.

3> **Insertion Sort** : Builds the sorted portion by iteratively inserting elements from the unsorted portion into their correct positions.

Each new element is compared in the sorted portion and inserted where it belongs.

4> **Selection Sort** : Divides the collection into a sorted and an unsorted portion.

It finds the minimum (or maximum) element from the unsorted portion and swaps it with the first unsorted element. The sorted portion gradually expands.

It has an average and worst case time complexity of $O(n^2)$.

- Tuples

→ A tuple is a collection of different values which is ordered and unchangeable.

→ In python , tuples are written with rounded /small brackets.

example :

```
Tuple 1 = (1,2, "GP", "Patna")
 print (Tuple 1)
```

Output :

```
(1 , 2 , "GP", "Patna")
```

- Dictionaries

* Python dictionary

→ Any immutable value can be a Key.

→ Can update dictionaries in place - mutable, like list.

* Empty dictionary is {} , not[]

→ ~~initi~~ initization : test 1 = {}

* <u>Keys can be any immutable values</u>

→ int, float, bool, string, tuple

→ But not list, or dictionary.

for example:

```
Score = {"Rohit": 84, "Virat": 100}
```

——x——

5. Exception handling, input/output, file handling and String processing

- Exception handling

→ Exception handling allows us to gracefully deal with run time errors.

→ It can check type of error and take appropriate action based on type.

→ Can change coding style to exploit exception handling.

```
try :
    : : :   ← code where error may occur
except IndexError :
    . . .   ← what to do if index Error
              occur
except (NameError, KeyError) :
    . . .   ← Common code to handle multiple
              error
except :
    . . .   ← catch all other exceptions
else :
    . . .   ← Execute if try terminates normally
              no error
```

• Input / output

* Interacting with the user :

→ Program needs to intract with the user

↳ Receive input
↳ Display output

→ Standard input and output

↳ input from keyboard
↳ output from screen

→ Read from keyboard using input()
→ Can also display a message

```
User data = input ("Enter a number :")
```

→ print to screen using print()

```
print (x,y)
print (a,b,c)
```

→ Can control format of print() output

↳ optional arguments end = "...", sep = "..."
↳ more precise control later.

- File handling

* Dealing with files :

→ Standard input and output is not convenient for large volumes of data.

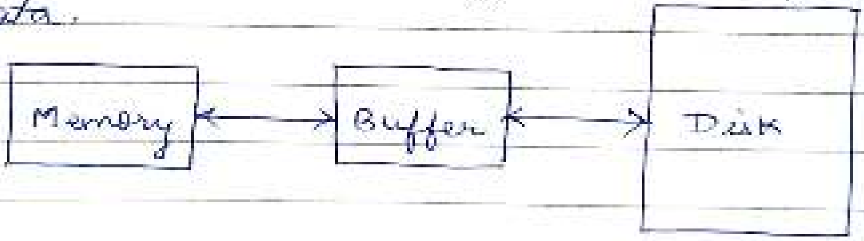
→ Instead, read and write files on the disk.

→ Disk read/write ~~and~~ is much slower than memory.

* Disk buffers :

→ Disk data is read/written in large blocks.

→ "Buffer" is temporary place for disk data.



* Opening a file :

```
Fh = open("gcd.py","r")
```

→ First argument to open is file name
- Can give a full path

→ Second argument is mode for opening file

- Read , "r" : open a file for reading only
- Write , "W" : creates an empty file to write to
- Append , "a" : append to an existing File

* **Reading file :**

→ Reading is a sequential operation

- When file is opened, point to position 0
- Each successive readline () moves forward

→ Fh. seek(n) → moves pointer to position n

* **End of file :**

→ ~~Bo~~ The follow both signal end of file

- Fh . read () returns empty string " "
- Fh. readline () returns empty string " "

- **Closing a file :**

→ Fh. close ()

- **Copying a file :**

```
→ infile = open ("input. text ","r" )
  outfile = open ("output. text ","w")
   For line in range infile. readlines :
        outfile.write (line)
    infile.close ()
    outfile.close ()
```

- String processing

→ Easy to read and write text files

→ String processing functions make it easy to analyse and transform contents

→ Search and replace text

* Strip whitespace

→ S.rstrip () removes trailing whitespace

```
    For line in contents :
        S = line.rstrip ()
```

* Searching for text

→ Syntax :

```
        S.find (pattern)
```

* Searching and replace

→ Syntax :

```
        S.replace (From str, tostr)
```

* Joining String :

```
→      date  = "16"
      month  = "08"
       year  = " 2016 "

    today = " _ " .Join ([date , month , year])
```

—— x ——

6. Backtraking in python

- **Backtracking**

→ Backtracking is a form of recursion. But it involves choosing only option out of any possibility. We begin by choosing an option and backtrack from it, if we reach a state where we conclude that this specific option does not give the required solution. We repeat these step by going across each available option until we get the desired solution.

Example :

```
def permute (list, s):
        if list == 1 :
            return S
           else:
           return [
              y+x
              for y in ~~pe~~ permute (1,S)
              for x in permute (list,-1,S)
                  ]
print (permute (1, ["a", "b", "c"]))
print (permute (2, ["a", "b", "c"]))
```

—— x ——

7. Class and object in python

- Class

→ Syntax :

```
class className :
        # statement
```

→ Classes are created by Keyword class.

→ Attributes are the variables that belong to a class

Ex :

```
class Dog :
    pass
```

- Object

→ Syntax :

```
class className :
        # statement... suite
```

→ The object is essential to work with the class attributes.

→ Instantiate is a term used when we create the object of any class, and the instance is also refered to as an object.

——x——

8. Dynamic programming in python

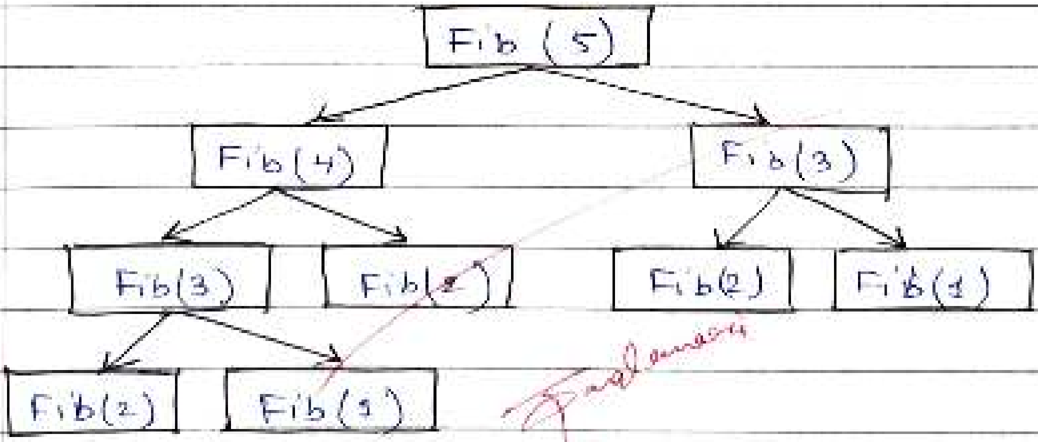
- **Dynamic programming**

→ Dynamic programming is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the Fact that the optimal solution to the overall problem depends upon the optional solution to subproblem

Example : Fibonacci series

* **Optimal Substructure :**

→ Fibonacci (N) = Fibonacci (N-1) + Fibonacci (N-2)



Overlapping Subproblem

—x—