Programming for DS.

Python DS vs DS vs RAM Memory

Check List

Tuple

* Immutable, ordered, Indexed, allows duplicates, Heterogeneous, light weight, hashable (tuple can be used as dictionary key), nestable.

```
info = ("Swetha", True, (10, 20))      to access tuple we use []
                                                    ↑
                                                  index.
empty = ()
info = ("Sasikala", 2025, True, (10, 20))

print(len(info)) = 4
print(len(empty)) = 0
print(len(info[-1])) = 2.

combined = info + ("AI", "Python")
o/p: ('Sasikala', 2025, True, (10, 20), AI, Python)

print('Sasikala' in info)  True.

name, year, bol, coords = info
print(name) → Sasikala

name, *rest = info  ⟶ Is name, rest a tuple.

result = info(2 : : -1)
result = info[: : -2]
```

data = [(1, 'b'), (3, 'a'), (2, 'c')]

sorted(data) ⟶ sorts by first element.

sorted(data, key = lambda x : x[1]) # sorts by second element

data = ((3, 'c'), (4, 'a')) → sort not possible bcz tuple.

data = tuple(list1) → converting list to tuple.

Q) Create a tuple of tuples containing student names & scores. Sort tuple by scores in descending order.

Student = (('A', 80), ('B', 95), ('C', 70))

~~sorted(data, key = lambda~~

sorted_tuple = tuple(sorted(student, key = lambda x : x[1], reverse = True))

32) From a list of tuples (product, price) filter out products priced below 500.

34) Access 7 from nested = (1, (2, 3, (4, 5), (6, 7)))

nested[1][2][2][1] or nested[-1][-1][-1][-1]

35) & 36) HW.

8) fn [illegible] args & returns them as tuple.

```
def pack_into_tuple(*args):
    return args;

l = []
for i in args:
    l.append(i)

return tuple(l)
```

9) Perform XOR (element wise) b/w 2 tuples of equal length.

. t1 = (1, 2, 3) t2 = (3, 2, 1)

o/p : (2, 0, 2)

```
result = tuple(a^b for a, b in zip(t1, t2))
```

18/7/25

Dictionary is a built in DS that stores key value pairs.

Characteristics : ordered, mutable, indexed by keys, keys must be unique & immutable, any data type.

Common operations :

dict-name[key] - access

dict-name["key"] - value — add item.

merge = d1 | d2.

```
d = {}  for i in range(5):
            sq = i**2
            d[i] = sq
```
} do append(x : x*2)

grades = {'A': 85, 'D': 92, 'k': 78}

passed = {k:v for k, v in grades.items() if v >= 80}

for k in grades:

    if grades[k] >= 80:

        passed.append(k: grades[k])

g[k]

access values using index number.

value_at_index1 = list(d.values())[1]

HW

Q) Create last 3 days temp avg & store atleast 5 readings/day using dictionary (Date is key) (P/o both dict)

Ex: d = {'18/7.25': [30, 38, 36, 33, 35]

:

}

Q) Find best rating & lowest price of amazon products. Get product id, rating & price

Sets

Sets are definate, unordered

A = {0, 2, 4, 4, 6, 8}

B = {1, 2, 3, 4, 5}

Union: A|B = 0, 1, 2, 3, 4, 5, 6, 8

Intersection: A&B = 2, 4

Symmetric diff: A^B = 0, 1, 3, 4, 5, 6, 8

present twice in A

Frozen Set - once created cannot be changed

Multiset - allows duplicates, unordered, hashable

pip install multiset

List - ordered, mutable, allow duplicates, heterogeneous

Adding elements to list - append(), extend(), insert

Removing elements - remove(), pop(), del

List Comprehension - concise way of creating a list

squares = [x**2 for x in range(1,6)]

Q) prices = [7,10,1,3,6,9,2]

Task is to find the maximum profit possible by buying & selling stocks on different days.

A)
```
prices = [7, 10, 1, 3, 6, 9, 2]
min_profit = prices[0]
max_profit = 0
for i in range(1, len(arr)):
    profit = arr[i] - min_profit
    min_profit = min(arr[i], min_profit)
    max_profit = max(max_profit, profit)
print(max_profit)
```

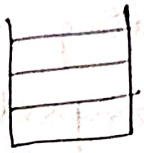
1 → 9 : 8

1. Python Basics
2. Python DS
} 10 to 12 Marks

3. ~~Python~~ DS (RAM & all) ['Python is convenient way of handling stack, que']

Memory (not cont)

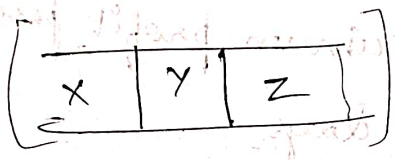
How to append element at the front



stack.insert(0, 'D')

stack = [] , queue = [] ⇒ list but we use as stack, queue.

queue = []



pop(0) → first element

deque from collections.

Code

Implementing stack using deque

```
from collections import deque
stack = deque()
stack.append('A')
stack.append('B')
stack.append('C')
print(stack)# deque([A, B, C])

# pop
stack.pop() → C
```

to insert an element at the start we can directly use appendleft

Insert is for list implementation

```
from collections import deque
# initalize
dq = deque()
dq.append(10)
dq.append(20)
dq.appendleft(5)
dq.pop()
dq.popleft()
```

10 | 20

5 | 10 | 20

20

5.

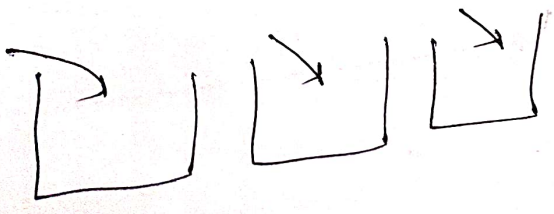
reverse(), remove(), rotate(n), extend(list),
extendleft(list)

Q) For Bread arrange it in the rack as per increasing order of expiry date (shortest)

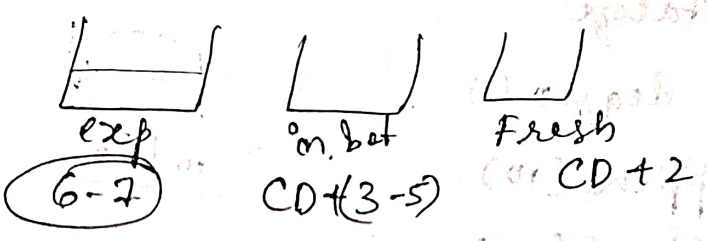
Deck

```
from collections import deque
dq.deque()
if data > expiry
    dq.append(date)
elif date <
    dq.appendleft(date)
```

Q) Three boxes according to manufacturing dates, seperate



if expiry > 0:
max_exp = max(exp)
min_exp = min(exp)
left
stack.append(max_exp)
stack.append(min_exp)



7 items

Stack1 = deque()
Stack2 = deque()
Stack3 = deque()

current_date = get in python.

expiry = date - current data

---

from datetime import datetime
print(datetime.today().date())

O/P: YYYY-MM-DD.

25/7/25

31) count vowels.

```
count = 0
word = input()
word = word.lower
for char in word:
    if (char == 'a' or char == 'i' or char == 'o' or
        char == 'e' or char == 'u':
        count += 1
print(count)
```

vowels = 'aeiou'

Q2)
```
word = "good morning"
word.replace("morning", "evening")
print(word)
```

Q3)
```
word = "safe"
sentence = "wearable device for safety"
if word in sentence:
    print("Present")    # Present
```

Q4)
```
sentence = "Programming for data science"
res = sentence.find("Data")
word = "Data"
res = word.lower()
```

Q5)
```
l = ['Hi', 'Welcome', 'Hello']
res = '-'.join(l)
print(res)    # Hi-Welcome-Hello
```

Q6)
```
sentence = "-- Programming for data science --"
Sentence.skip()
print(sentence)    # Programming for data Science
```

Q7)
```
pin = 560090
print(pin.isdigit())    # True
```

Q8)
```
PAN = 'GXWZ431G'
print(PAN.isalnum())    # True
```

Inheritance is (super()) → base class & derived class

Multiple inheritance with Polymorphism

```
# Base class
    class Hello :
        def func(self) :
            print("Hello")

# derived class
    class Python(Hello) :
        de
```

Diamond prob. how to access base class fn()

OOP in Python

Class = blueprint for creating objects

Syntax:

```
class ClassName :
    # constructor
    def __init__(self, ...) :
        # instance variables
        self.var1 = ...

    # method
    def some_method(self) :

# creating obj
```

(runs automatically when an obj is created)

(variables specific to each object.)

Ex:

```
class Rectangle :
    def __init__(self, width, height) :        # const
        self.width = width
        self.height = height                   # instance variables
```

```
def area(self):
    return self.width * self.height
```

5 × 4

```
# creating obj
rect = Rectange(5,4)
print(rect.area()) = 20
```

} outside the class.

Instance variables are available throughout the class - in all methods - but only through the object self.

Access Modifiers.

	Syntax	Access Scope	Access outside class
Public	self.var	everywhere	✓
Protected	self._var	class + subclass	✓ (Not recommended)
Private	self.__var	only inside class	(Not recommend)

can be done with name mingling.

Name mingling : obj._ClassName__var

Single Inheritance

* A child class (derived class) inherits from one parent class (base class). Child gets access to all the attributes & methods of the parent.

Ex :

```
Class Animal :
    def speak(self) :
        print ("Animal speaks")

Class Dog (Animal) :          # Single inheritance
    def bark (self) :
        print ("Dog barks")

# create object of child class
d = Dog()
d.speak()  # inherited from Animal
d.bark()
```

Multiple Inheritance

Ex :

```
Class Father :
    def skills (self) :
        print ("Father : Gardening, carpentry")

Class Mother :
    def skills (self) :
        print ("Mother : Cooking, Painting")

Class Child (Father, Mother) :  # Multiple inheritance
    def my-skills (self) :
        print ("Child : Coding")
        Father.skills (self)
        Mother.skills (self)

c = Child()
```

If both methods have same name then Python uses the first method. (Method Resolution order)
(MRO)

NOTE:
Name Mangling - isn't required if we have a public method to access the private variable.

But if not : self.__security_code = "ABC123" # Private var
can be accessed as : self._Sensor__security_code.

NOTE:
To access parent methods, multiple inheritance : use super()

1/8/25

Natural Language Toolkit (NLTK)

```
# tokenize & stemmer
from nltk.tokenize import word_tokenize, sent_tokenize

# PosTagging
#entity recognition
# corpora
# spacy
```

2/8/25

(In class exercise)

Q) Define a Base class User with attribute name & email

```
Class User:
    def __init__(self, name, email):          # encapsulation
        self.__name = name
        self.__email = email                  # instance variables
    def display_info(self):                   # instance variables must also be written as self.__
        print("Name : {self.__name}")
        print("Email : {self.__email}")
    def get_name(self):                       # (Getter method) # Public method to access private variable
        return self.__name
```

Internally in Python every Private variable becomes -

Ex: self.__name = name

Translates to :- self._User__name ← Name mingling

(or) self._ClassName__varName

3) Implement child class using super()

Create a class Student inheriting from User. Add a new attribute course & use super() to call parent initializer. Override display_info() to include course information & use super() to call parent class

```
Class Student (User):
    def __init__(self, name, email, course):
        self.course = course
        super().__init__(name, email)
                ↖ calling parent initializer.

    def display_info(self):
        print("Student Info")
        print("Name : {self.get_name()}")   ← accessing private var
        print("Email : {self._email}")
        print("Course : {self.course}")
```

← independent class

```
Class Mentor :
    def __init__(self, specialization):
        self.specialization = specialization

    def display_info(self):
        print("Mentor info")
        print("Specialization : {self.specialization}")
```

Multiple inheritance.

```
~~Class Mentor~~ :
Class TeachingAssistant (Student, User):
    def __init__(self, name, email, course,
                 specialization):
        super().__init__(name, email, course)
        self.specialization = specialization
```

calling parent classes.

```
    def display_info(self):
        print("Teaching assistant info")
        print(f"Name : {self.get_name()}")
        print(f"Email : {self._email}")
        print(f"Course : {self.course}")
        print(f"Specialization : {self.specialization}")

def show_details(obj):
    obj.display_info()

u = User("Alice", "@gmail")
s = Student("Bob", "@yahoo", "AI")
ta = TeachingAssistant("Joe", "@hotmail", "DS", "ML")
m = Mentor("DS")

show_details(u)
show_details(s)
show_details(ta)
show_details(m)
```

} each class prints diff values for same fn

Polymorphism

O/P:

```
Alice
@gmail
Bob
@yahoo
AI

Mentor:
DS

TA:
Joe
@hotmail
DS
ML
```

display_info ↙
Method overriding

When a child class defn a method with same name as a method in its parent class, & modifies its behavior it is called

Method overring

5/8/25 for CAT & FAT

8] Binary Image Processing & Text Processing – Scipy & Numpy (application based)

Numpy

np.arange(10, 20, 2)

np.linspace(0, 1, 5) ← no of elements

CSE = np.array([1, 2, 3, 4, 5])

MML = np.array([6, 7, 8, 9, 10, 11])

np.concatenate ⇒ to merge 2 arrays.

Random numbers : np.random.rand(3, 3) ⇒ Array of rand no

Create 16×16 matrix

matrix = np.random.randint(0, 100, (4, 4)) ↓ array

Store all 2×2 submatrices

sub_matrices = []

for i in range(4):

    for j in range(4):

        sub_matrix = matrix[i:i+2, j:j+2]

        sub_matrices.append(sub_matrix)

print("Total 2×2 submatrices : ", len(sub_matrices))

which is affected [0 1; 1 ;] aff [1 ; 1 ;] aff [; ; ; ;]

Let blocks = $\begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{bmatrix}$

Sentiment analysis

np.char.count(arr, 'an')

from scipy.linalg import solve

A = np.array

B = np.array

7/8/25

CAT - 1 Syllabus

Module 1

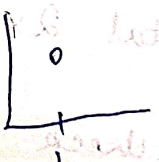
- Condition & Branching statements
- List, Tuple, Dictionary, set, ~~user defn~~
- User defn: stack, Queue, Priority queue
- String handling
- OOPS
- NLTK; ChatterBot

30

Module 2

- Pandas, DataFrame of, NumPy, SciPy, Pandas, Matplotlib

20



Matplotlib.

1. Box
2. Scatter } — outliers.
3. Box
4. Histogram
5. Area
6. Piechart - Imbalanced data.

8/8/25

Priority Queue

1 ← highest priority (general)

- can be implemented using heapq or from queue import priorityqueue.
- Chatbot related answers.

Power BI

Trends → Line / Area chart

Comparision → Bar / column

Relationships → Scatter or Bubble

Composition → Pie, Donut, Tree map

Maps → maps or filled map.

Waterfall → think step by step changes in value

Funnel → narrowing process stages

race tracks, positions change over time

14/8/25

Tutorial

Q1) Given a list of book titles & IDs, classify each book into Mechanical (Mech), Computer science (CSE) or Electronics (ECE) based on keywords in the title.

Q2) Write atleast 5 NLTK & PF operations.

A) book = [['Automobile Engg', '01'], ['Signals & Systems', 2], ['Algorithms', 4], ['Fluid dynamics', 5], ['Digital electronics', 6], ['Machine learning', 7]]

Keyword = {'Mech' : 'Automobile', 'Fluid',
'CSE' : 'Algorithms', 'Machine learning',
'ECE' :}

Mech = ['Automobile', 'Fluid']
CSE = ['learning', 'Algorithms']
ECE = ['Digital', 'Signals']

Steps:
* Create keywords
* Take book titles as input from user (10 books) & ID
* Then for each title perform NLTK (tokenization & POS tagging) then append into list segregating CSE, Mech, ECE.

NLTK operations

NLP

1. Tokenization - Splitting text into parts.

   * Breaking down a piece of text into smaller units (tokens) like words or sentences.
   * word tokenization → split into words / punctuation

   Ex: text = "Hello. my name is"
   print(word_tokenize(text))

   O/P: ['Hello', '.', 'my', 'name', 'is']

   * Sentence tokenize → splits into sentences.

   Ex: print(sent_tokenize(text))

2. Stopword Removal - Removing common filler words.

   * Stopwords are very common words (like 'is', 'the', 'and') that usually don't add much meaning. Removing them helps focus on keywords.

NOT in syllabus

3. Stemming - Cutting words to ~~their~~ their root

   * Chops off word endings to reduce them to a base form.
   * Does not guarantee a real world

   Ex: flies —stemming→ fli

Ex:
```
from nltk.stem import PorterStemmer
ps = PorterStemmer()        <- applies a set of rules to remove endings
print(ps.stem("running"))   → run
print(ps.stem("flies"))     → fli   fly
```

4. Lemmatization - Reducing words to actual dictionary form.

* Converts words to their base form based on grammar & vocal
* unlike stemming, returns valid words

Ex: from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("running", pos="v"))
↳ run

fly

5. POS Tagging - Identifying grammatical roles.

* Parts of speech tagging assigned to each word

Ex: import nltk
words = word_tokenize("NLTK is great for tasks")
print(nltk.pos_tag(words))

Ex: [('NLTK', 'NNP'), ('is', 'VBZ'), ('great', 'JJ'), ('for', 'IN'), ('tasks', 'NNS')]

6. Named Entity Recognition (NER) - Finding names, places, date

NER detects real world named entities like people, organizat

locations.

Sentence = "Apple is buying a UK startup for 1 $ billion"

Apple → Organization
UK → Detected as geo political entity
1 $ billion → Money

Extra : & word_tokenize() internally calls sent_tokenize()
& sent_tokenize() relies on pre-trained sentence boundaries detection model called punkt.

word_tokenize() → expects only one string at a time

---

text = ["Hi, My name is Swetha. I am studying NLP."]

tokenization < word_tokenize
sent_tokenize

NNP
verb

sent = sent_tokenize(text)
= word_tokenize(text)

sent = [Hi
[I

word_to ['Hi', ',', 'My', 'nam

Stemming - running → run

POS tag
↓
Parts of sp