Basic data structures in Python

Unit: IV

Problem Solving using Python (ACSE0101)

Course Details
(B Tech 1st Sem)

Sachin Kumar
(Asst. Professor)
IT Department

Content

Unit Objective

- In This semester, the student will
  - Learn Python Basic Data Structures to solve complex problem.
  - Study the concept of packing and unpacking, mutable and immutable data types.
  - Gain the understanding of built-in methods of string, list, tuple, set and dictionary.

Course Outcome

Course Outcome (CO)	At the end of course , the student will be able to:	Bloom's Knowledge Level (KL)
CO1	Analyse and implement simple python programs.	K3, K4
CO2	Develop Python programs using decision control statements.	K3, K6
CO3	Implement user defined functions and modules in python.	K2
CO4	Implement python data structures –string, lists, tuples, set, dictionaries.	K3
CO5	Perform input/output operations with files in python, apply exception handling for uninterrupted execution.	K3, K4

Program Outcomes (PO’s)

Engineering Graduates will be able to:

- **PO1 : Engineering Knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
- **PO2 : Problem Analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

Program Outcomes (PO's)

- **PO3 : Design/Development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate considerations for the public health and safety, and the cultural, societal and environmental considerations.
- **PO4 : Conduct Investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.

Program Outcomes (PO’s)

- **PO5 : Modern tool usage**: Create, select and apply appropriate techniques, resources and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations
- **PO6 : The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and consequent responsibilities relevant to the professional engineering practice.

NIET Greater Noida GET FUTURE READY Estd.2001 AN AUTONOMOUS INSTITUTE

Program Outcomes (PO's)

- **PO7 : Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development
- **PO8 : Ethics**: Apply the ethical principles and commit to professional ethics, responsibilities, and norms of engineering practice.
- **PO9 : Individual and teamwork**: Function effectively as an individual, and as a member or leader in diverse teams and multidisciplinary settings.

Program Outcomes (PO’s)

- **PO10 : Communication**: Communicates effectively on complex engineering activities with the engineering community and with society such as being able to comprehend and write effective reports and design documentation, make effective presentations and give and receive clear instructions.
- **PO11 : Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in team, to manage projects and in multidisciplinary environments.
- **PO12 : Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadcast context of technological change.

CO-PO Mapping

CO.K	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	3	3	3	2	2	1	-	1	-	2	2
CO2	3	3	3	3	2	2	1	-	1	1	2	2
CO3	3	3	3	3	3	2	2	-	2	1	2	3
CO4	3	3	3	3	3	2	2	1	2	1	2	3
CO5	3	3	3	3	3	2	2	1	2	1	2	2
AVG	3.0	3.0	3.0	3.0	2.6	2.0	1.6	0.4	1.6	0.8	2.0	2.4

Prerequisite and Recap

- Conditionals
- Loops
- Functions
- Module

Recap:

- Decision control statements are used for branching and iterate a set of statements repeatedly.

Data types (CO4)

- Every value in Python has a datatype, that are used to define the operations possible on them and the storage method for each of them.
- Since everything is an object in Python programming, data types are classes and variables are instance (object) of these classes

Standard Data types (CO4)

Data Types	Keyword
Text Type	str
Numeric Types	int, float, complex
Sequence Types	list, tuple, range
Mapping Type	dict
Set Types	set
Boolean Types	bool
Binary Types	bytes

strings

STRINGS

Objective of Topic String

- To be aware of basic operations of string in python such as
  - Appending, concatenating on strings
  - Indexing and slicing of strings
  - Comparing Strings

Recap

Let's

Recap

Decision control statements are used for branching and iterate a set of statements repeatedly.

- For loop
- While loop

Introduction to String [CO4]

In Python, Strings are arrays of bytes representing Unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Basic operations [CO4]

Traversing:

A string can be traversed by accessing character(s) from one index to another.

Concatenating:

Concatenate means joining two strings. We use + operator to concatenate two given strings.

Appending:

Append means add something to the end of string. We use + = operator to add one string at the end of another string.

Multiplying:

Means repeating a string n number of times

Program to traverse a string using indexing [CO4]

```
msg="Hello!"
index=0;
for i in msg:
        print("msg[", index,"]=", i)
        index+=1
```

Output:

```
msg[0]=H
msg[1]=e
msg[2]=l
msg[3]=l
msg[4]=o
msg[5]=!
```

Program to concatenate two string using + operator [CO4]

```
str1=“Hello!”
str2=“world”
str3=str1+str2
print(“Concatenated string is :”, str3)
```

Output:

```
Concatenated string is :  Hello! world
```

Program to append a string using + operator [CO4]

```
str="Hello!"
Name=input("\nEnter your name:")
str+=name
str+= ".Welcome to python."
print(str)
```

Output:

```
Enter your name: Sachin
Hello! Sachin. Welcome to python.
```

Program to repeat a string using * operator [CO4]

```
str="Hello!"
print(str * 3)
```

Output:

```
Hello! Hello! Hello!
```

String formatting operator [CO4]

String formatting operator is one of the exciting features of python.

The % operator takes a format string on the left (%d, %s etc) and corresponding value in a tuple on the right.

The format operator % allows users to construct strings, replacing parts of the strings with the data stored in variables.

Syntax:

“<FORMAT>”%(<VALUES>)

Program to use format sequences [CO4]

```
name =“Abhishek”
Age=9
print(“Name = %s and age = %d” %(name, age))
print(“Name = %s and age = %d” %(“avi”, 8))
```

Output:

```
Name= Abhishek and age = 9
Name = avi and age = 8
```

Formatting symbols [CO4]

Operator	Purpose
%c	Character
%d or %i	Signed decimal integer
%s	String
%u	Unsigned decimal integer
%o	Octal integer
%x or %X	Hexadecimal Integer
%e or %E	Exponential Notation
%f	Floating point number
%g or %G	Short numbers in floating point or exponential notation

Indexing [CO4]

- Individual characters in a string are accessed using the subscript ([]) operator. The expression in bracket is called index.
- The index specifies a member of an ordered set and here it specifies the character we want to access from the given set of characters in the string.
- the first character has the position 0 and last character has n-1, where n is number of characters

If you try to exceed the bounds (below 0 or above n-1) an error is raised

Example: Get the character at position 1

```
a = "Hello, World!"
print(a[1])
```

Output:

```
e
```

Slicing [CO4]

A substring of a string is called slice. The slice operation is used to refer to sub-parts of strings.

We use [] operator also called slicing operator to take subset of a string from original string.

P	Y	T	H	O	N
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

Fig: Indices in a string

slicing [CO4]

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

Example

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])
```

Output:

```
llo
```

Program to demonstrate slicing operation [CO4]

```
str = “PYTHON”
print(“str [1:5] = ”, str[1:5])
print(“str [:6] = ”, str[:6])  #defaults to start of string
print(“str [1:] = ”, str[1:]) #defaults to end of string
print(“str [:] = ”, str[:]) #defaults to entire string
print(“str [1:20]  =”, str[1:20]) #truncates to length of the string
```

Output:

```
str [1:5] = YTHO
str [:6] = PYTHON
str [1:] = YTHON
str[ : ] = PYTHON
str [1:20] = YTHON
```

Slice using Negative Indexing [CO4]

Use negative indexes to start the slice from the end of the string:

Example

Get the characters from position 5 to position 1, starting the count from the end of the string:

```
b = "Hello, World!"
print(b[-5:-2])
```

Output:

```
orl
```

Program to demonstrate slicing operation [CO4]

```
str = “PYTHON”
print(“str [-1] = “, str[-1])       #last character is accessed
print(“str [-6] = “, str[-6])       #first character is accessed
print(“str [-2:] = “, str[-2:])     #Second last and last characters
print(“str [:-2] = “, str[:-2])     #All except last two
print(“str [-5:-2] = “, str[-5:-2]) #from second upto second last
```

Output:

```
str [-1] = N
str [-6] = P
str [-2:] = ON
str[ : -2] = PYTH
str [-5:-2] = YTH
```

NIET Greater Noida
GET FUTURE READY
Estd.2001
AN AUTONOMOUS INSTITUTE

Comparing strings [CO4]

Operator	Description	Example
= =	If two strings are equal, it returns true	>>> “ABC” == “ABC” True
!=	If two strings are not equal, it returns true	>>> “AbC” != “ABC” True
>	If first string is greater than the second, it returns true	>>> “aBC” > “ABC” True
<	If first string is smaller than the second, it returns true	>>> “ABC” < “aBC” True
>=	If first string is greater than or equal to the second, it returns true	>>> “aBC” >= “ABC” True
<=	If first string is smaller than or equal to the second, it returns true	>>> “ABC” <= “aBC” True

Escape Character [CO4]

- An escape character is a backslash \ followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example

```
txt = “Aman asks, \“How are you?\” ”
print(txt)
```

Output:

```
Aman asks, “How are you? ”
```

Escape characters [CO4]

Escape character	Description
\'	Single Quote
\”	Double Quote
\\	Backslash
\n	New Line
\t	Tab
\b	Backspace

STRING METHODS [CO4]

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
endswith()	Returns true if the string ends with the specified value
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string

NIET Greater Noida GET FUTURE READY Estd.2001 AN AUTONOMOUS INSTITUTE

STRING METHODS [CO4]

Method	Description
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
upper()	Converts a string into upper case

STRING METHODS [CO4]

Method	Description
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
rjust()	Returns a right justified version of the string
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
strip()	Returns a trimmed version of the string

Daily Quiz

(1) Index of first character in string is

(a) 0 (b) 1 (c) n-1 (d) n

Ans: (a) 0

(2) Which error is generated when index is not an integer

(a) IndexError

(b) NameError() (c) TypeError() (d) BoundError()

Ans: (d) BoundError()

(3) Which operator is used to repeat a string n number of times

(a) + (c) *

(b) [] (d) +=

Ans: *

REGULAR EXPRESSION [CO4]

REGULAR EXPRESSION

Objective of Regular expression in String

- To train students on Regular expression for pattern matching

Python RegEx [CO4]

- A RegEx, or Regular Expression, is a special sequence of characters that defines a search pattern for complex string-matching mechanism.
- Python has a built-in module re that can be used to work with Regular Expressions.
- Importing the re module:

Syntax

```
import re
```

RegEx Functions [CO4]

Function	Description
match()	Returns a match object if there is a match in the beginning of the string
search()	Returns a match object if there is a match anywhere in the string
findall()	Returns a list containing all matching pattern string
split()	Returns a list where the string has been split at each matching pattern
sub()	Returns a string where each matching string is replaced by another substring one or more times.

RegEx Example [CO4]

Search the string to see if it starts with “Python":

```
import re
msg = “Python is an object-oriented language."
x = re.match(“Python", msg)
if (x):
        print(“Pattern is matched.")
else:
        print("No match")
```

Output:

```
Pattern is matched.
```

RegEx Example [CO4]

Search the string to see if it starts with "The" and ends with "Spain":

```
import re
#Check if the string starts with "The" and ends with "Spain":
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
if (x):
        print("YES! We have a match!")
else:
        print("No match")
```

Output:

```
YES! We have a match!
```

The search() Function [CO4]

- The search() function searches the string for a match and returns a match object if there is a match.
- If there is more than one match, only the first occurrence of the match will be returned.

Example: Search for the first white-space character in the string:

```
import re
txt = "The rain in Spain"
x = re.search("\s", txt)
print("The first white-space character is located in position:", x.start())
```

Output:

```
The first white-space character is located in position: 3
```

Note: If no matches are found, the value None is returned.

The search() Function [CO4]

Example: Make a search that returns no match:

```
import re
txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
```

Output:

```
None
```

sub() function [CO4]

The sub() function replaces the matches with the text of your choice:

Example: Replace every white-space character with the number 9:

```
import re
txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

Output:

```
The9rain9in9Spain
```

Note: You can control the number of replacements by specifying the count parameter:

sub() function [CO4]

Example: Replace the first 2 occurrences:

```
import re
txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2)
print(x)
```

Output:

```
The9rain9in Spain
```

NIET Greater Noida GET FUTURE READY Estd.2001 AN AUTONOMOUS INSTITUTE

findall() function [CO4]

The findall() function returns a list containing all matches.

Example: Print a list of all matches:

```
import re
txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

Output:

```
['ai', 'ai']
```

The split() Function [CO4]

The split() function returns a list where the string has been split at each match:

Example: Split at each white-space character:

```
import re
txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

Output:

```
['The', 'rain', 'in', 'Spain’]
```

Note: You can control the number of occurrences by specifying the maxsplit parameter.

The split() Function [CO4]

Example

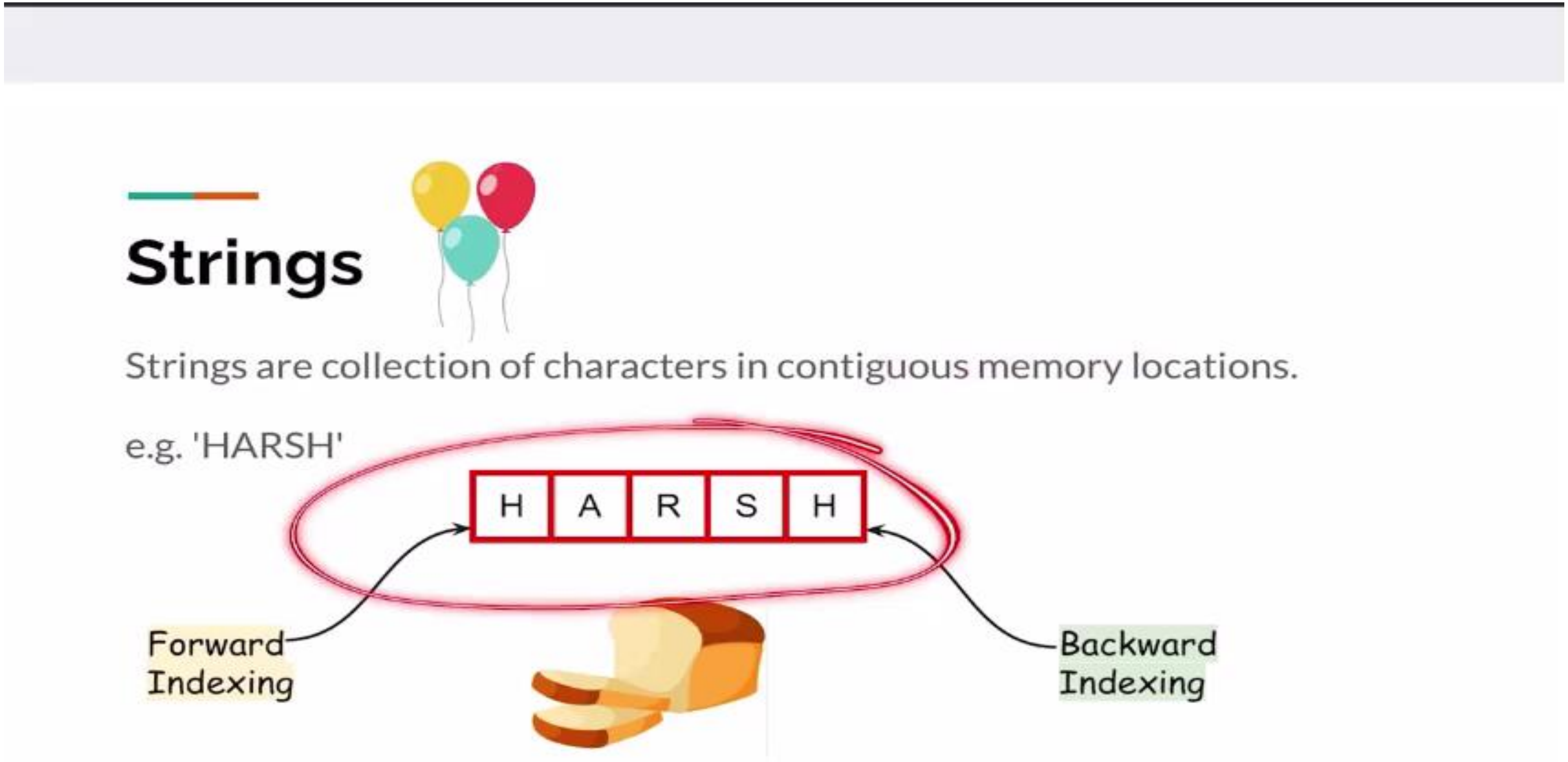
Split the string only at the first occurrence:

```
import re
txt = "The rain in Spain"
x = re.split("\s", txt, 1)
print(x)
```

Output:

```
['The', 'rain in Spain']
```

String [CO4]



Strings

Strings are collection of characters in contiguous memory locations.

e.g. 'HARSH'

H
A
R
S
H
Forward Indexing
Backward Indexing

Daily Quiz

(1) Which regular expression is not equivalent to others

(a) (a|b|c|d|e) (b) [abcde] (c) [a-f] (d) None of these

Ans: (c)

(2) [a^]* would match, all strings with

(a) zero or more repetition of any character

(b) zero or more repetition of a

(c) zero or more repetition of ^

(d) both b and c

Ans: (b)

Weekly Assignment

- Explain various methods on regular expression.
- Write a program to swap two strings.
- differentiate between
- (i) find() and rfind()
- (ii) find() and index()
- Write a python program to demonstrate difference between pass by reference and value.
- Given a String comprising of many words separated by space, write a Python program to iterate over these words of the string.
- Python program that takes a text file as input and returns the number of words of a given text file.

Python data structure

Python data structure

Content

- Sequence
- Unpacking Sequences
- Mutable Sequences
- Lists
- List Comprehension
- Looping in lists
- Tuples
- Sets
- Dictionaries

Recap

Let's

Recap

- Sets are collection of elements.
- Sets are unordered and does not contains duplicates.
- We use list or tuples whenever need something different from sets

Topic Objective

- To Introduce students about various data structures in Python
- To teach students about list, tuple and dictionary data structures and compare them.

Sequence [CO4]

- In Python, a sequence is the ordered collection of similar or different data types. It is a generic term for ordered set.
- In sequence, each element has a specific index that starts from 0 and is automatically incremented for next element in the sequence.
- Sequence Sequences allow storing multiple values in an organized and efficient fashion. There are several sequence types in Python –
  - ❖String
  - ❖List
  - ❖Tuple

Types of Sequence [CO4]

- Sequence types: strings, Unicode strings, lists, tuples, and range objects.
- String's literals are written in single or double quotes: 'xyzzy', "frobozz".
- Lists are constructed with square brackets, separating items with commas: [a, b, c].
- Tuples are constructed by the comma operator (not within square brackets), with or without enclosing parentheses, but an empty tuple must have the enclosing parentheses, e.g., a, b, c or (). A single item tuple must have a trailing comma, e.g., (d,).
- range objects are used to create a sequence of numeric values.

Packing Sequences [CO4]

- Packing is a technique in python with which we put several values into a single iterator. If we talk of packing in literal terms, Just like we pack certain items into a box in the real world, In python we pack certain variables in a single iterable.

How to perform packing in Python ?

- We can perform packing by using simple syntax for declaration of iterables like list or tuples or we can use asterisk operator * for packing. The * operator is used as an operator for unpacking the iterables into variables but it allows us to pack a number of variables into a single variable.
- For example, suppose we have three variables num1, num2,num3. We can pack them into a tuple as follows.

```
num1=1
num2=2
num3=3
myTuple=(num1,num2,num3)
```

How to perform packing in Python ?

- Or we can pack them into a list as follows.

```
num1=1
num2=2
num3=3
myList=[num1,num2,num3]
```

How to perform packing in Python ?

Alternatively, we can use * operator to pack the numbers together as follows.

```
num1=1
num2=2
num3=3
*num,=num1,num2,num3
```

How to perform packing in Python ?

- We have used a comma “,” after the *num because the left hand side of the assignment operation must be a tuple or list otherwise error will be encountered.
- In the left side, when we use * operator, we can also have other variables which can be assigned the values. For example, we can pack two numbers into a variable and a third number can be assigned to another variable as follows.

```
num1=1
num2=2
num3=3
*num,myNum=num1,num2,num3
```

How to perform packing in Python ?

In the above example, we have to remember that myNum is a mandatory variable and it must be assigned some value whereas *num can be assigned no value. Here, num3 will be assigned to myNum and num1 and num2 will be packed in the list num.

Unpacking Sequences [CO4]

- Let we have a function that receives four arguments. Also, we have a list of size 4 that has all arguments for the function. When we make call to this function and simply pass list to the function, the call doesn’t work.

Python program to understand need of unpacking [CO4]

```
# A sample function that takes 4 arguments and prints them.
def func(a, b, c, d):
        print(a, b, c, d)

#driver Code
my_list = [1, 2, 3, 4]
func(my_list)  # This doesn't work
```

Python program to understand need of unpacking [CO4]

Unpacking

We can use * to unpack the list so that all elements of it can be passed as different parameters.

```
# A sample function that takes 4 arguments and prints them
def fun(a, b, c, d):
        print(a, b, c, d)
# Driver Code
my_list = [1, 2, 3, 4]
fun(*my_list)        # Unpacking list into four arguments
```

Output

```
1 2 3 4
```

Mutable Sequences [CO4]

- Mutable Sequences:

Sequence that can change their state or contents are called mutable sequence.

These are of type list, dict, set . Custom classes are generally mutable.

Mutable Sequences [CO4]

```
# Python code to test that lists are mutable
color = ["red", "blue", "green"]
print(color)

color[0] = "pink"
color[-1] = "orange"
print(color)
```

Output

```
['red', 'blue', 'green']
```

Output

```
['pink', 'blue', 'orange']
```

Lists [CO4]

- Lists are like arrays, declared in other languages.
- A single list may contain data types such as Integers, Strings or Objects.
- The elements in a list are indexed according to a definite sequence.
- indexing of a list is done with 0 being the first index. It is represented by list class.

Python program to understand Creation of List [CO4]

```
# Creating a List
List = [ ]
print(List)
```

Output

```
[ ]
```

```
# Creating a list of strings
List = ['Python Programming', 'Python']
print(List)
```

Output

```
['Python Programming', 'Python']
```

```
# Creating a Multi-Dimensional List
List = [['Python', 'For'], ['Beginners']]
print(List)
```

Output

```
[['Python', 'For'], ['Beginners']]
```

Python program to Access elements from the List [CO4]

Use the index operator [ ] to access an item in a list. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3].

```
List = [1, 2, 3, 4, 5, 6]
# accessing a element
print(List[0])
print(List[2])
```

Output

```
1
3
```

```
# Negative indexing
print(List[-1])      # print the last element of list
print(List[-3])      # print the third last element of list
```

Output

```
6
4
```

NIET Greater Noida GET FUTURE READY Estd.2001 AN AUTONOMOUS INSTITUTE

List methods [CO4]

Method	Description	Syntax	Example	output

append()	Appends an element to the list	list.append(obj)	list=[1,2,3,4,5]
			list.append(8)
			print(list)
				[1,2,3,4,5,8]

count()	count number of times element appears in the list	list.count(obj)	list=[1,2,3,4,5]
			print(list.count(3))
				1

index()	Returns lowest index of element in list	list.index(obj)	list=[1,2,3,4,5]
			print(list.index(3))
				2

List methods [CO4]

Method	Description	Syntax	Example	output

insert()	Insert object at the specified index in the list	list.insert(index,obj)	list=[1,2,3,4,5]
			list.insert(3, 10)
			print(list)
				[1,2,3,10,4,5]

pop()	Removes element at specified index from list(by default last)	list.pop(index)	list=[1,2,3,4,5]
			print(list.pop())
			print(list)
				5
				[1,2,3,4]

List methods[CO4]

Method	Description	Syntax	Example	output

remove()	removes the elements in the list	list.remove(obj)	list=[1,2,3,4,5]
			list.remove(1)
			print(list)
				[2,3,4,5]

reverse()	reverse the elements in the list	list.reverse()	list=[1,2,3,4,5]
			list.reverse()
			print(list)
				[5, 4, 3, 2, 1]

List methods[CO4]

Method	Description	Syntax	Example	output

sort()	sorts the elements in the list	list.sort()	list=[5,2,4,1,3]
			list.sort()
			print(list)
				[1,2,3,4,5]

extend()	adds the elements in a list to the end of another list	list1.extend(list2)	list1=[1,2,3,4,5]
			list2=[6,7]
			list1.extend(list2)
			print(list1)
				[1,2,3,4,5,6,7]

Python program to demonstrate Addition of elements in a List Using append(), insert() and extend() [CO4]

```
# Creating a List
List = []
# Using append()
List.append(5)
List.append(6)
print(List)
```

Output

```
[5, 6]
```

```
# Using insert()
List.insert(3, 12)
List.insert(0, 'Begin')
print(List)
```

Output

```
['Begin', 5, 6, 12]
```

```
# Using extend()
List.extend([18, 'keep', 'smiling'])
print(List)
```

Output

```
['Begin', 5, 6, 12, 18, 'Keep', 'smiling']
```

Program to demonstrate removal of elements from a List Using remove() and pop() [CO4]

```
# Creating a List
List = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
# using Remove() method
List.remove(5)
List.remove(6)
print(List)
```

Output

```
[1, 2, 3, 4, 7, 8, 9, 10, 11, 12]
```

```
# using pop()
List.pop()
print(List)
```

Output

```
[1, 2, 3, 4, 7, 8, 9, 10, 11]
```

List Comprehension [CO4]

- Python supports computed lists called list comprehensions having following syntax:
- List =[expression for variable in sequence]
- Where the expression is evaluated once, for every item in the sequence.
- List comprehension help programmers to create list in a concise way.
- This is mainly beneficial to make new lists where each element is obtained by applying some operations to each member of another sequence or iterable.
- List comprehension is also used to create a subsequence of those elements that satisfy a certain condition.

Program to make a list of cubes [CO4]

```
cubes = [ ]
for i in range (11):
        cubes.append(i**3)
print(“cubes of numbers from 1-10: ”, cubes)
```

Output:

```
cubes of numbers from 1-10 :
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Program to make a list of cubes using iterable [CO4]

Iterable:

An iterable is an object that can be used repeatedly in subsequent loop statements, e.g. for loop

Rewriting program to make list of cubes using iterable:

```
>> cubes = [i**3 for i in range(11)]
>> print(cubes)
```

Output:

```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Program to combine and print elements of two list [CO4]

```
print([(x, y) for x in [10, 20, 30]  for y in [30, 10, 40] if x!=y])
```

Output:

```
[(10, 30), (10, 40), (20, 30), (20, 10), (20, 40), (30, 10), (30, 40)]
```

Looping in lists
#using for loop [CO4]

There are multiple ways to iterate over a list in Python.

Method #1: Using For loop

```
list = [1, 2, 3, 4, 5]
for i in list:
        print(i)
```

Output

```
1
2
3
4
5
```

Looping in lists
Method #2: For loop and range() [CO4]

Using traditional for loop that iterates from number x to number y.

```
# iterating over a list
list = [1, 2, 3, 4, 5]
length = len(list)
for i in range(length):
        print(list[i])
```

Output

```
1
2
3
4
5
```

Looping in lists
Method #3: Using while loop [CO4]

```
list = [1, 2, 3, 4, 5]
length = len (list)
i = 0
while i < length:
        print(list[i])
        i += 1
```

Output

```
1
2
3
4
5
```

Looping in lists
Method #4: Using list comprehension [CO4]

```
list = [1, 2, 3, 4, 5]
[print(i) for i in list]
```

Output

```
1
2
3
4
5
```

Looping in lists
Method #5: Using enumerate [CO4]

```
list = [11, 12, 13, 14, 15]
for i, val in enumerate(list):
        print (i, ",",val)
```

Output

```
0 , 11
1 , 12
2 , 13
3 , 14
4 , 15
```

Program to find sum and mean of elements in a list [CO4]

```
list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sum=0
for i in list:
        sum+= i
print (”sum of elements in list=”, sum)
length=len(list)
print ( ”mean of elements in list=”, sum/length)
```

Output

```
sum of elements in list= 55
mean of elements in list= 5.5
```

Daily Quiz

What will be the output of the following Python code?

```
a=[10,23,56,[78]]
b=list(a)
a[3][0]=95
a[1]=34
print(b)
```

a) [10,34,56,[95]]
b) [10,23,56,[78]]
c) [10,23,56,[95]]
d) [10,34,56,[78]]

Answer: c

Daily Quiz

What will be the output of the following Python code?

```
import copy
a=[10,23,56,[78]]
b=copy.deepcopy(a)
a[3][0]=95
a[1]=34
print(b)
```

a) [10,34,56,[95]]
b) [10,23,56,[78]]
c) [10,23,56,[95]]
d) [10,34,56,[78]]
Answer: b

Tuples [CO4]

Tuple

- A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.
- A tuple is an immutable sequence of Python objects.

Tuples vs lists [CO4]

- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists.
- Tuples use parentheses, whereas lists use square brackets.
- Creating a tuple is as simple as putting different comma-separated values.

Creating Tuple [CO4]

For creating a tuple, one need to just put the different comma separated values within a parentheses.

Tup = (val1, val2, …….) where val can be integer, floating number, character or a string.

Example

```
tup = ("apple", "banana", "cherry")
print(tup)
```

Output:

```
('apple', 'banana', 'cherry')
```

Access Tuple Items [CO4]

You can access tuple items by referring to the index number, inside square brackets:

Example

print the second item in the tuple:

```
tup1 = ("apple", "banana", "cherry")
tup2 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print(tup1[1])
print(tup2[3:6])
print(tup2[:4])
print(tup2[4:])
print(tup2[:])
print(tup2[-1])  // beginning from end
print(tup2[-3:-1])
```

Output

```
banana
(4, 5, 6)
(1, 2, 3, 4)
(5, 6, 7, 8, 9, 10)
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
10
(8, 9)
```

Change Tuple Values [CO4]

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example: Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Output:

```
("apple", "kiwi", "cherry")
```

Loop Through a Tuple [CO4]

You can loop through the tuple items by using a for loop.

Example: Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")

for x in thistuple:
  print(x)
```

Output:

```
apple
banana
cherry
```

Tuple Assignment [CO4]

It allows tuple of variables on left hand side of the assignment operator to be assigned values from a tuple on the right hand side of the assignment operator.

Each value is assigned to its respective variable.

Example:

```
(a, b, c)= (1, 2, 3)
print (a, b, c)
Tup1=(100, 200, 300)
(a, b, c)= Tup1
print (a, b, c)
(a, b, c) = (3 + 4, 7 / 3 + 4, 9 % 6)
print (a, b, c)
```

Output

```
1 2 3
100  200  300
7   6.333333   3
```

Check if Item Exists In Tuples [CO4]

To determine if a specified item is present in a tuple use the in keyword:

Example: Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

output:

```
Yes, 'apple' is in the fruits tuple
```

Tuple Length [CO4]

To determine how many items a tuple has, use the len() function:

Example

Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

```
3
```

Tuple Methods [CO4]

Python has two built-in methods that you can use on tuples.

count(): returns the number of times a specified value occurs in a tuple

```
Tup=(1,2,3,3,5,6,3,8)
print(Tup.count(3))
```

Output

```
3
```

index(): Searches the tuple for a specified value and returns the position of where it was found

```
Tup=(1,2,3,4,5,6,7,8)
print(Tup.index(4))
```

Output

```
3
```

Join Two Tuples [CO4]

To join two or more tuples, you can use the + operator:

Example

Join two tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

Output

```
('a', 'b', 'c', 1, 2, 3)
```

Remove Items in tuple [CO4]

Note: You cannot remove items in a tuple.

Tuples are unchangeable, so you cannot remove items from it, but you can delete the tuple completely:

Example: The del keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple  no longer exists
```

Tuples returning multiple values [CO4]

A function can return only a single value. But when we need to return more than one value from a function, we group them as tuple and return

```
def max_min(*vals):
        x = max(vals)
        y = min(vals)
        return (x, y)
a, b = max_min(23,45,78,12,34,91)
print(a, b)
```

Output

```
91  12
```

Nesting of Tuples [CO4]

Process of defining a tuple inside another tuple is called Nesting of tuple.

```
# Code for creating nested tuples
tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'programming')
tuple3 = (tuple1, tuple2)
print(tuple3)
```

Output :

```
((0, 1, 2, 3), ('python', 'programming'))
```

Daily Quiz

1. Which of the following is a Python tuple?
a) [1, 2, 3]
b) (1, 2, 3)
c) {1, 2, 3}
d) {}
Answer: b

2. Suppose t = (1, 2, 4, 3), which of the following is incorrect?
a) print(t[3])
b) t[3] = 45
c) print(max(t))
d) print(len(t))
Answer: b

Daily Quiz

What will be the output of the following Python code?

```
>>>t=(1,2,4,3)
>>>t[1:3]
```

a) (1, 2)

b) (1, 2, 4)

c) (2, 4)

d) (2, 4, 3)

Answer: c

What will be the output of the following Python code?

```
>>>t=(1,2,4,3)
>>>t[1:-1]
```

a) (1, 2)

b) (1, 2, 4)

c) (2, 4)

d) (2, 4, 3)

Answer: c

Weekly Assignment

1. How are tuples important data structures?
2. Create a tuple that has just one element which in turn may have three elements ‘a’, ‘b’,’c’. Also print the length of tuple.
3. How can you return more than one values from a function. Explain with the help of program.
4. What do you understand by variable length arguments?
5. Python program to create an instance of an Ordered Dict using a given dictionary. Sort the dictionary during the creation and print the members of the dictionary in reverse order.
6. Explain Sets and Dictionaries with built-in methods.

SETS

SETS

SETS [CO4]

Definition:

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

A set is mutable i.e., we can easily add or remove items from a set

Set are same as list but with a difference that sets are list with no duplicate entries.

Creating a SET [CO4]

A set is created by placing all the elements inside curly brackets.

Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

```
{'cherry', 'apple', 'banana'}
```

Access Items in a SET [CO4]

- Items in a set can’t be accessed by referring to an index, since sets are unordered the items has no index.
- Using for loop the items in the set can be iterated.
- Example

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
    print(x)
```

Output

```
cherry
banana
apple
```

SETS [CO4]

Change Items

Once a set is created, items of the set can’t be changed but a new item can be added.

Add Items to a SET [CO4]

- To add one item to a set, use the add() method.
- To add more than one item to a set use the update() method.
- Example

Add an item to a set, using the add() method:

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

Output:

```
{'apple', 'orange', 'cherry', 'banana'}
```

Add Multiple Items to a SET [CO4]

Example

Add multiple items to a set, using the update() method:

```
thisset = {"apple", "banana", "cherry"}
thisset.update(["orange", "mango", "grapes"])
print(thisset)
```

Output:

```
{'banana', 'apple', 'cherry', 'orange', 'grapes', 'mango'}
```

Get the Length of a Set [CO4]

To determine how many items a set has, use the len() method.

Example

Get the number of items in a set:

```
thisset = {"apple", "banana", "cherry"}
print(len(thisset))
```

Output:

```
3
```

Remove Item from a Set [CO4]

To remove an item in a set, use the remove(), or the discard() method.

Example: Remove "banana" by using the remove() method:

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")    # thisset.discard("banana")
print(thisset)
```

Output:

```
{'cherry', 'apple’}
```

Note: If the item to remove does not exist, remove() will raise an error while discard does not give error

Join Two Sets [CO4]

- You can use the union() method that returns a new set containing all items from both sets, or
- the update() method that inserts all the items from one set into another:

Example

The union() method returns a new set with all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

Output

```
{'c', 'b', 2, 3, 1, 'a'}
```

Join Two Sets [CO4]

The update() method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}

set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

Output

```
{'c', 3, 'a', 'b', 1, 2}
```

Note: Both union() and update() will exclude any duplicate items.

SETS [CO4]

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have an intersection or not

SETS [CO4]

Method	Description
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Daily Quiz

Which of these about a set is not true?
a) Mutable data type
b) Allows duplicate values
c) Data type with unordered values
d) Immutable data type
Answer: d

2. Which of the following is not the correct syntax for creating a set?
a) set([[1,2],[3,4]])
b) set([1,2,2,3,4])
c) set((1,2,3,4))
d) {1,2,3,4}
Answer: a

Daily Quiz

What will be the output of the following Python code?

```
nums = set([1,1,2,3,3,3,4,4])
print(len(nums))
```

a) 7

b) Error, invalid syntax for formation of set

c) 4

d) 8

Answer: c

Which of the following statements is used to create an empty set?

a) { }

b) set()

c) [ ]

d) ( )

Answer: b

Weekly Assignment

Q1 Make two sets of random integers and apply all set operations on them.

Q2 Write a program that generates a set of prime numbers and another set of odd numbers. Demonstrate the result of union, intersection, difference and symmetric difference operations on these sets.

Q3 Explain any 5 methods on set.

Dictionaries

Dictionaries

Topic Objective

- To Introduce students about dictionaries in Python
- To train about methods of Directories and how to buid applications using them

Recap

Let's

Recap

- Sets are collection of elements.
- Sets are unordered and are not indexed
- We use dictionaries as they are unordered, changeable and indexed

Dictionaries [CO4]

Dictionary

- A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.
- Each key is separated from its value by a colon (:) and consecutive items are separated by commas.
- Entire items in dictionary are enclosed in curly brackets ({}).

Dictionaries [CO4]

Syntax:

dictionary_name = {key1:value1, key2:value2, key3:value3}

Value of key can be of any type

Dictionary keys are case sensitive. Two keys with same name but in different case are not same in Python.

Dictionaries [CO4]

- Creating an empty dictionary

```
dict = {}
print(dict)
```

Output

```
{}
```

- Create and print a dictionary:

```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964 }
print(thisdict)
```

Output

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Accessing Items in Dictionaries [CO4]

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example: Get the value of the "model" key:

```
thisdict ={
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

Output:

```
Mustang
```

Accessing Items in Dictionaries [CO4]

There is also a method called get() that will give you the same result:

Example: Get the value of the "model" key:

```
thisdict ={"brand": "Ford", "model": "Mustang", "year": 1964 }
x = thisdict.get("model")
print(x)
```

```
Mustang
```

Modifying an item in Dictionary [CO4]

- The value of a specific item can be changed by referring to its key name:

Example

Change the "year" to 2018:

```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964 }
thisdict["year"] = 2018
print(thisdict)
```

Output:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Loop Through a Dictionary [CO4]

- When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

Example:

Print all key names in the dictionary, one by one:

```
thisdict ={ "brand": "Ford", "model": "Mustang", "year": 1964}
for x in thisdict:
    print(x)
```

Output

```
brand
model
year
```

Check if Key Exists [CO4]

- To determine if a specified key is present in a dictionary use the in keyword.
- Check if "model" is present in the dictionary:

```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964 }
if "model" in thisdict:
        print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Output:

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

Adding Items [CO4]

- Adding an item to the dictionary is done by using a new index key and assigning a value to it:
- Example

```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964 }
thisdict["color"] = "red"
print(thisdict)
```

Output:

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Removing Items from Dictionaries [CO4]

Method1

The pop() method removes the item with the specified key name:

```
thisdict = {"brand": "Ford","model": "Mustang","year": 1964}
thisdict.pop("model")
print(thisdict)
```

Output:

```
{'brand': 'Ford', 'year': 1964}
```

Removing Items from Dictionaries [CO4]

Method 2

The del keyword removes the item with the specified key name:

```
thisdict = {"brand": "Ford","model": "Mustang","year": 1964}
del thisdict["model"]
print(thisdict)
```

Output:

```
{'brand': 'Ford', 'year': 1964}
```

Removing Items from Dictionaries [CO4]

Method 3

The del keyword can also delete the dictionary completely:

```
thisdict = {"brand": "Ford","model": "Mustang","year": 1964}
del thisdict
print(thisdict)
```

Output:

#print statement will cause an error because "thisdict" no longer exists.

Removing Items from Dictionaries [CO4]

Method 4

The clear() method empties the dictionary:

```
thisdict = {"brand": "Ford","model": "Mustang","year": 1964 }
thisdict.clear()
print(thisdict)
```

Output:

```
{}
```

Nested Dictionaries [CO4]

Example: Create a dictionary that contain three dictionaries:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
print(myfamily)
```

A dictionary can also contain many dictionaries, this is called nested dictionaries.

Output

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

Dictionary method [CO4]

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Daily Quiz

1. Which of the following statements create a dictionary?
a) d = {}
b) d = {“john”:40, “peter”:45}
c) d = {40:”john”, 45:”peter”}
d) All of the mentioned
Ans d

2. What will be the output of the following Python code snippet?
d = {"john":40, "peter":45}
a) “john”, 40, 45, and “peter”
b) “john” and “peter”
c) 40 and 45
d) d = (40:”john”, 45:”peter”)
Answer: b

Daily Quiz

3. What will be the output of the following Python code snippet?
d = {"john":40, "peter":45}
"john" in d
a) True
b) False
c) None
d) Error
Answer: a

4. Suppose d = {“john”:40, “peter”:45}, to delete the entry for “john” what command do we use?
a) d.delete(“john”:40)
b) d.delete(“john”)
c) del d[“john”]
d) del d(“john”:40)
Answer: c

Faculty Video Links, Youtube & NPTEL Video Links and Online Courses Details

- Strings in Python
  - https://youtu.be/TXw1HseIdJw
  - https://youtu.be/lSItwlnF0eU
- List in Python
  - https://youtu.be/mDlSS_4BOvE
- Tuple in python
  - https://youtu.be/GstQPTWpt88
- Set in Python
  - https://youtu.be/MEPlLAjPvXY
- Dictionary in Python
  - https://youtu.be/rZjhId0VkuY

Weekly Assignment 4.1

1. Give the properties of lists
2. Explain list comprehension with example.
3. Differentiate between append() and insert()
4. Write a program that prints the maximum element in second half of a list
5. Discuss the relation between tuples and lists, tuples and dictionaries in detail.
6. Python program to find and print all prime numbers in a list.
7. Python program to convert a list of tuples into a dictionary.
8. write a function that takes a list of words and returns the length of longest one.
9. Python program to combine each line from first file with the corresponding line in second file.
10. Explain the following by giving suitable code:
    1. List Comprehension
    2. Packing and unpacking in tuples

MCQ s

(1) Identify the right way to close a file

(a) file.close() (b) close(file)

(c) Close(“file”) (d) file.closed

(2) Which method returns a string that includes everything specified in the path?

(i) os.path.dirname(path)

(ii) os.path.basename(path)

(iii) os.path.relpath()

(iv) os.path.abs()

MCQ s

(3) To open a file c:\scores.txt for reading, we use _______________

a) infile = open(“c:\scores.txt”, “r”)

b) infile = open(“c:\\scores.txt”, “r”)

c) infile = open(file = “c:\scores.txt”, “r”)

d) infile = open(file = “c:\\scores.txt”, “r”)

Answer: b

(4) Which of the following statements are true?

a) When you open a file for reading, if the file does not exist, an error occurs

b) When you open a file for writing, if the file does not exist, a new file is created

c) When you open a file for writing, if the file exists, the existing file is overwritten with the new file

d) All of the mentioned

Answer: d

MCQ s

(5) To read two characters from a file object infile, we use

____

a) infile.read(2)
b) infile.read()
c) infile.readline()
d) infile.readlines()
Answer: a

(6) The readlines() method returns ______________
a) str
b) a list of lines
c) a list of single characters
d) a list of integers
Answer: b

MCQ s

(7) Which regular expression is not equivalent to others

(a) (a|b|c|d|e) (b) [abcde] (c) [a-f] (d) None of these

Ans: (c)

(8) [a^]* would match, all strings with

- (a) zero or more repetition of any character
- (b) zero or more repetition of a
- (c) zero or more repetition of ^
- (d) both b and c

Ans: (b)

Old Question Papers

- What does [::-1] do?

  [NIET Autonomous 2020-21 (Odd) Marks-1]

- Name the Mutable built-in type does python provides?

  [NIET Autonomous 2020-21 (Odd) Marks-1]

- Write Python program to sort numbers in a list in ascending order using Merge Sort.

  [NIET Autonomous 2020-21 (Odd) Marks-10]

- Discuss format specifiers and escape sequences with examples.

  [NIET Autonomous 2020-21 (Odd) Marks-6]

Old Question Papers

- Discuss the relation between tuples and lists, tuples and dictionaries in detail. [NIET Autonomous 2020-21 (Odd) Marks-6]
- Write Python Program to count the number of characters in a string using dictionaries. Display the keys and their values in alphabetical Order. [NIET Autonomous 2020-21 (Odd) Marks-10]
- Write Python program to sort numbers in a list in ascending order using Merge Sort. [NIET Autonomous 2020-21 (Odd) Marks-10]
- Explain the following by giving suitable code:
  - i. List Comprehension
  - ii. Packing and Unpacking in tuples

[NIET Autonomous 2020-21 (Odd) Marks-6]

Old Question Papers

- What is the difference between list and tuples in Python?
  [AKTU 2019-2020(odd), 2 marks]
- Explain Unpacking Sequences, Mutable Sequences, and List Comprehension with example. Write a program to sort list of dictionaries by values in Python – Using lambda function.
  [AKTU 2019-2020(odd), 10 marks]

Expected Questions for University Exam

1. Explain various types of data structures in python.
2. Write and explain any 5 builts-in methods of string().
3. Explain list comprehension with suitable example.
4. Explain the looping in data structures.

SUMMARY

- **Python Basic Data Structures:**
  - Sequence
  - Packing and Unpacking Sequences, Mutable Sequences,
  - Strings, Basic operations, Comparing strings, string formatting, Slicing Built-in string methods and function
  - Regular expressions
  - Lists
  - Tuples
  - Sets and Dictionaries with built-in methods
  - List Comprehension
  - Looping in basic data structures

References

1. Allen B. Downey, “Think Python: How to Think Like a Computer Scientist”, 2nd edition, Updated for Python 3, Shroff/O‘Reilly Publishers, 2016.
2. Robert Sedgewick, Kevin Wayne, Robert Dondero, “Introduction to Programming in Python: An Inter-disciplinary Approach” , Pearson India Education Services Pvt. Ltd., 2016.
3. Paul Barry, “Head First: A Brain Friendly Guide” O’Reilly publisher.
4. Reema Thareja, “Python Programming: Using Problem Solving Approach” 2nd Edition, Oxford University Press publisher.
5. Guido van Rossum and Fred L. Drake Jr, “An Introduction to Python”, Revised and updated for Python 3.2, Network Theory Ltd., 2011.

End

Thank You