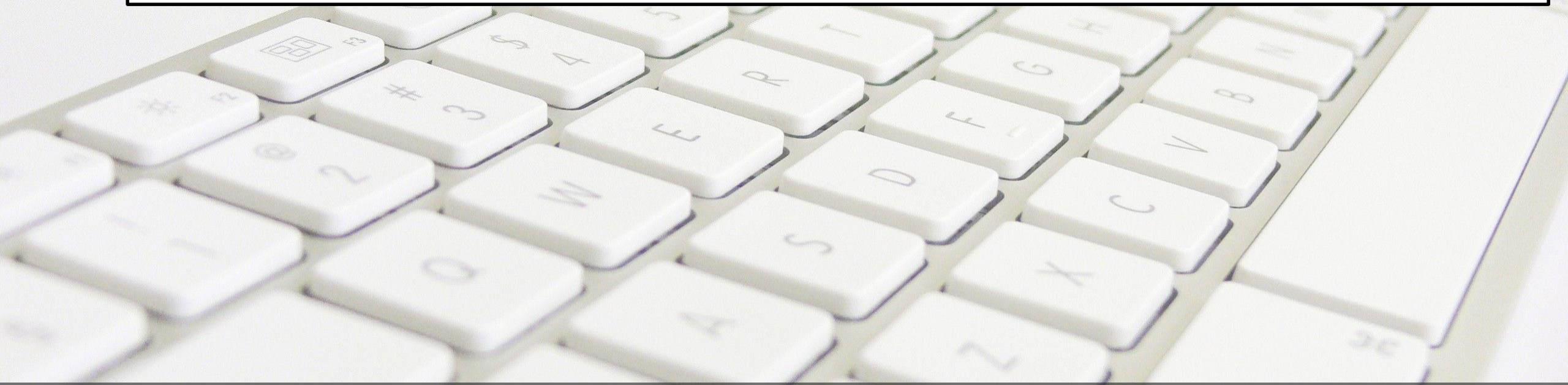
ABES
Estd. 2000

3.Python Collections and Sequences



General Guideline

Objective of Collections/Sequences

- To describe the importance of list , string, tuple , dictionary and set in python
- To uses to store multiple data values.
- To use the index to update, add, and remove items
- To use the data structures with out indexes
- To explain the difference among different Collections
- To select collections built-in functions in Python to write programs in Python.

Topics Covered

Day 1	Day 2	Day 3	Day 4

3.1 Introduction
• 3.2 String
  • Creation of String
  • Accessing the String
	3.2 String
	• Updation
	• Deletion
	• Built-in-methods
	• Operations
	• String Formatters
	• Loops with Strings
		3.3 List
		• Creation
		• Accessing
		• Update
		• Built in methods
			3.3 List
			• Loops
			• Nested List
			• List Comprehensions

Topics Covered

Day 5	Day 6	Day 7	Day 8
3.4 Tuple • Creation • Accessing • Modification • Built in Methods • Operations	**3. 5 Dictionary** • Creation • Accessing • Modification	**3.5 Dictionary** • Built in Methods • Loops and Conditions	**3.6 Set** • Creation • Assessing • Modification • Built in methods • Operators • Loops • Frozen set

Session Plan - Day 1

3 Introduction to Python Collections and Sequences

3.1 String

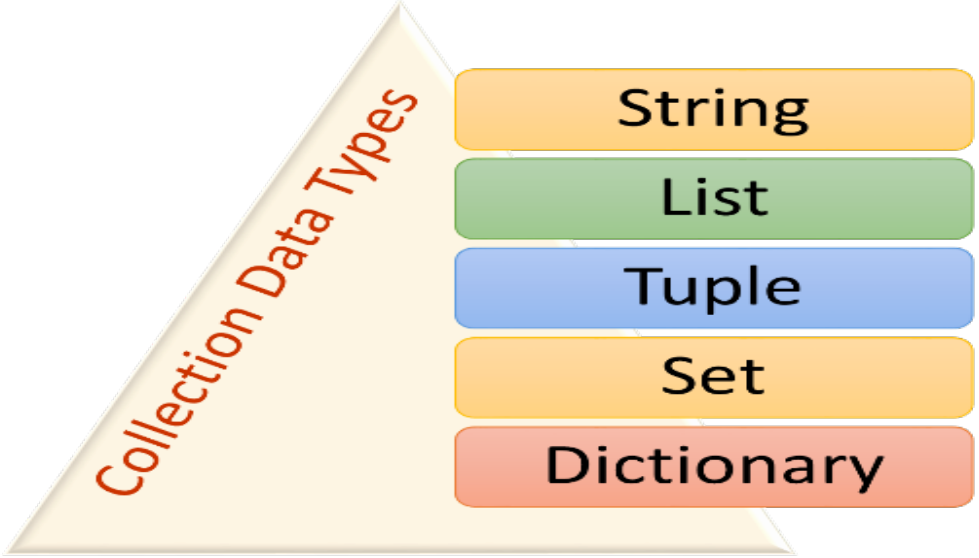
- Creation of String
- Assessing of String
- Updation of String
- Examples
- Review Questions

Introduction

In real life we need to store data like-

- **Name of students –** A sequence of alphabet character
- **Marks of n students –** A sequence of either integer or float value
- **Student's record –** A sequence of name, branch, roll number, address etc

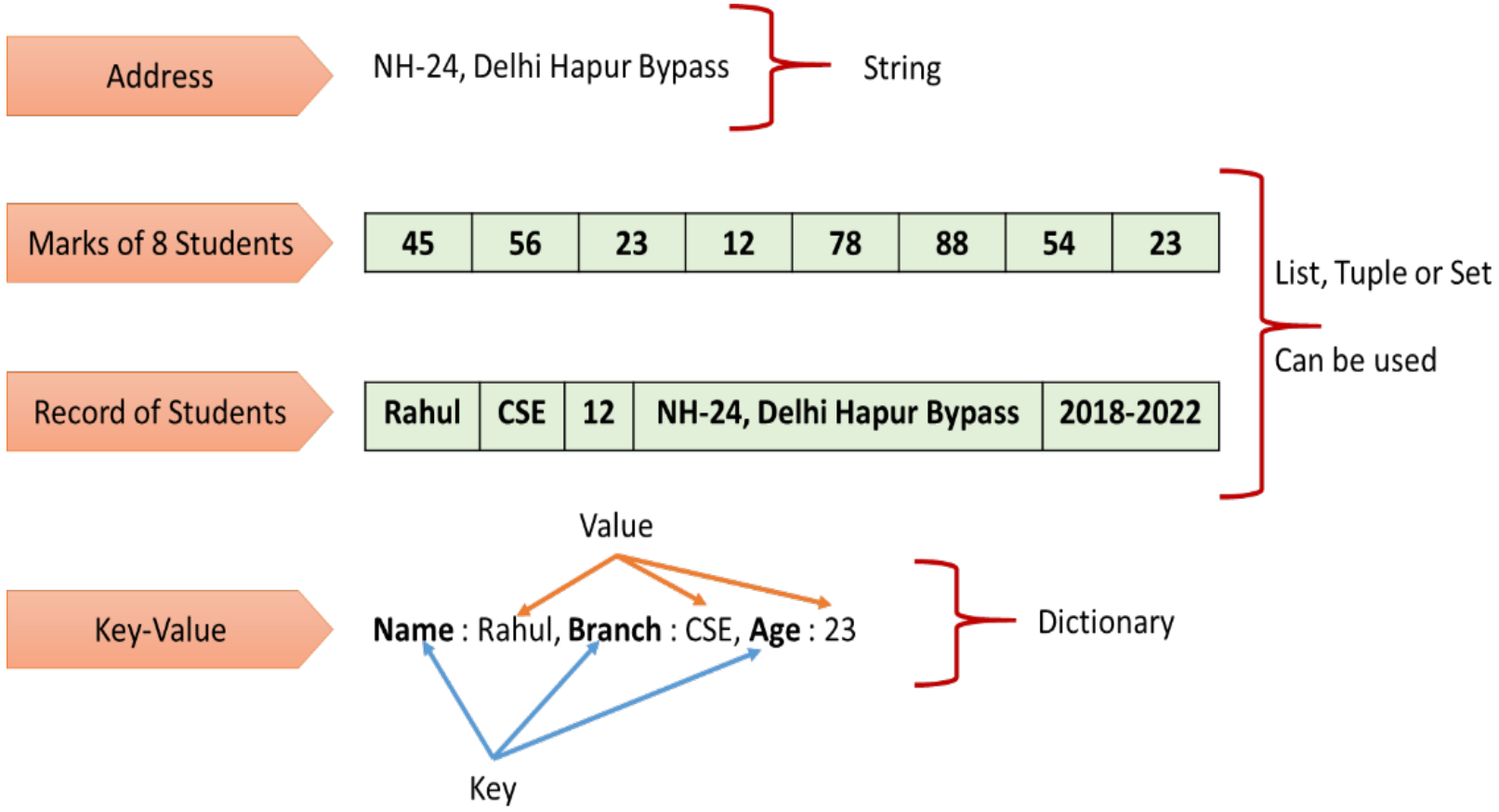
So to represent these type of data in python provides us some built in collections.



Contd…

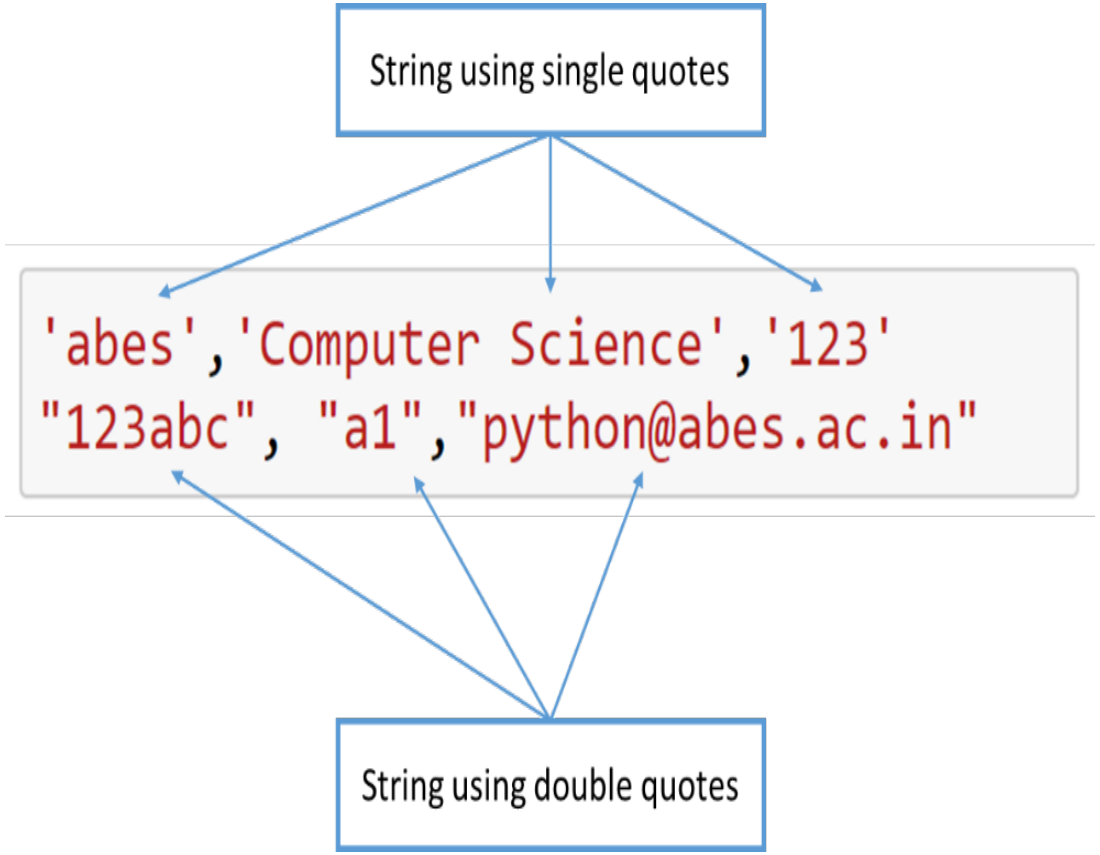
Representation of Python Collections with the help of Figure:

- If data is the combination of numeric data, alphanumeric or of different types than we go for List, Tuple or set .
- If data is in the form of key-value pair than we go for dictionary.



String

- A string is a sequence of alphanumeric and special character.
- String is the collection of characters, it may compose of alphanumeric and special characters.
- Strings are created in many ways using single quotes or double-quotes.



Alphanumeric characters: a-z, A-Z, 0-9; **Special symbol**: *, -, +, $, @, whitespace …. etc

Creation of Empty String

Empty String

- An empty string is a string having no character.
- We have three ways to create an empty string as shown in syntax.
- An Empty String is created by single quotes, double quotes and str().

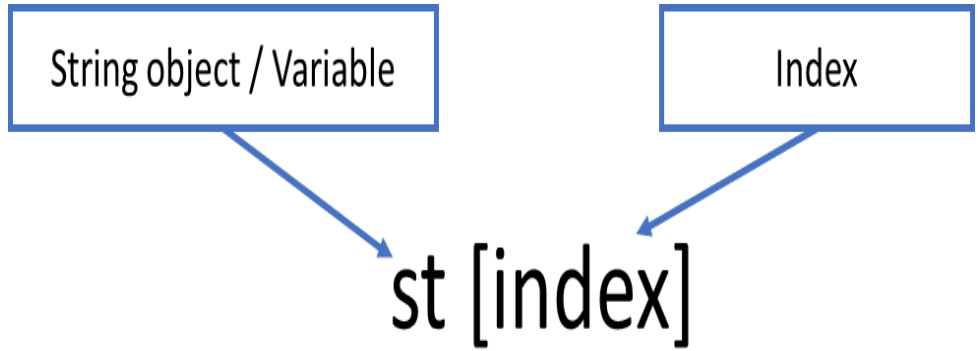
Creation of Non Empty String

- A non-empty string is a sequence of alphanumeric character with at least one character.
- A non-empty string is created by using single quotes and double quotes

```
st1 = 'abes'    #String creation using single quotes
st2 = "abes"    #String creation using double quotes
```

Assessing the String

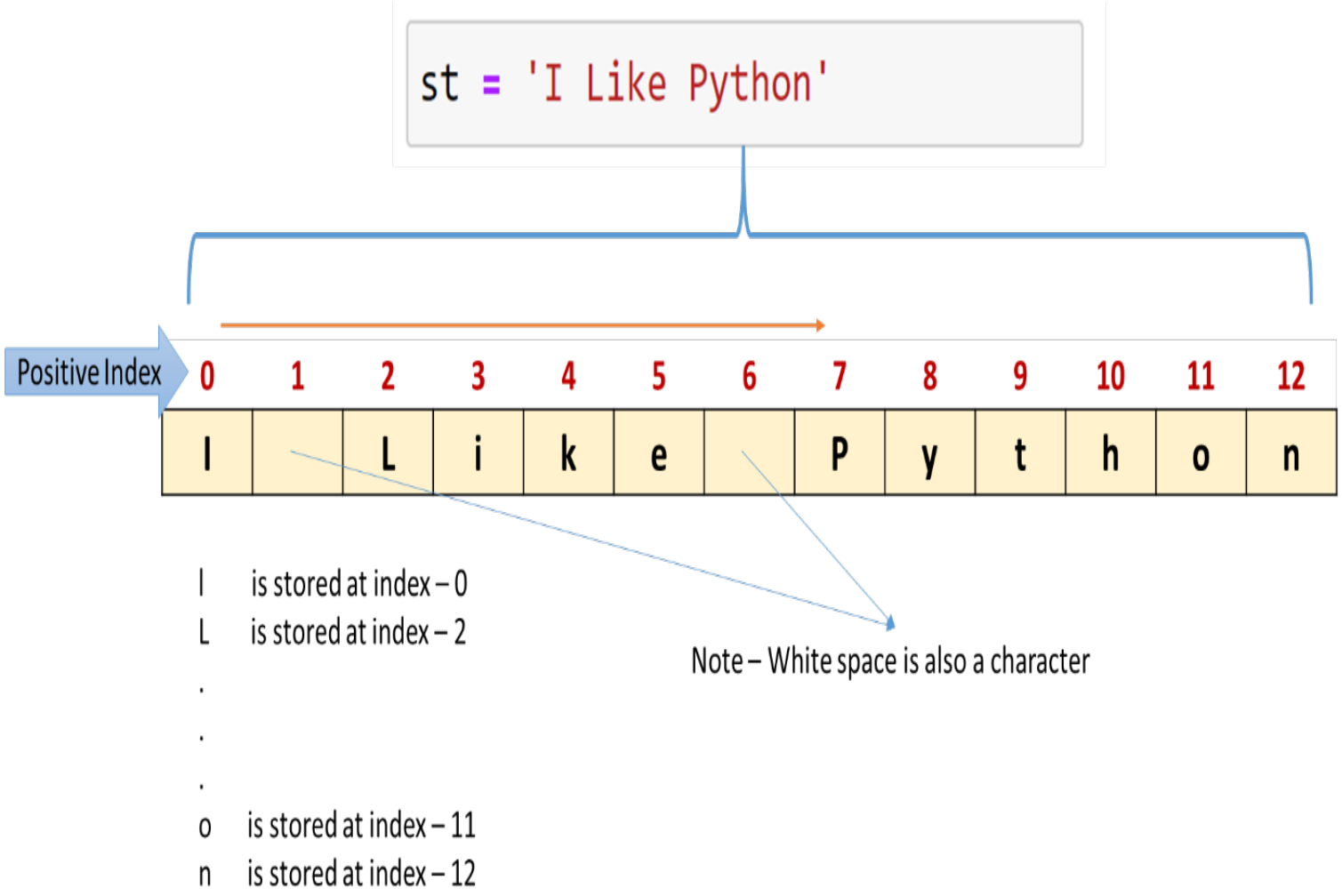
- String Element can be accessed using Square Bracket.
- These Square brackets [ ] take an index as input.
- In Python we have two types of Indexing.
  - Positive Indexing
  - Negative Indexing



Positive Indexing

- Positive indexing start from Zero (0) and from left hand side i.e. first character store at index 0, second at index 1 and so on.

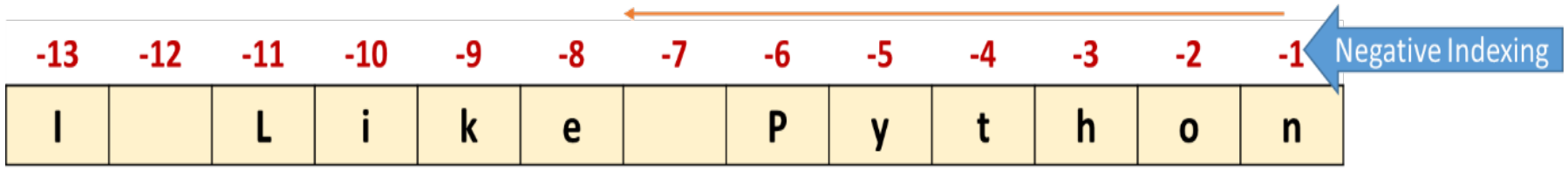
- Every character of string has some index value.



Negative Indexing

- Negative Indexing starts from negative indexing start from -1 and from right-hand side.
- Last character store at index -1 and as we move towards the left, it keeps increasing like -2, -3, and so on.

```
st = 'I Like Python'
```



n is stored at index → -1
o is stored at index → -2
.
.
.
.
I is stored at index → -13

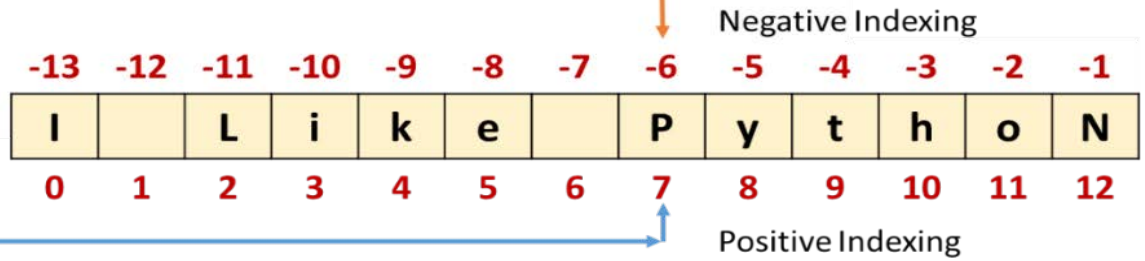
Example

- Write a program to print character 'P' and 'L' using positive indexing and negative indexing.

  Assume a string **st = 'I Like Python',** is given.

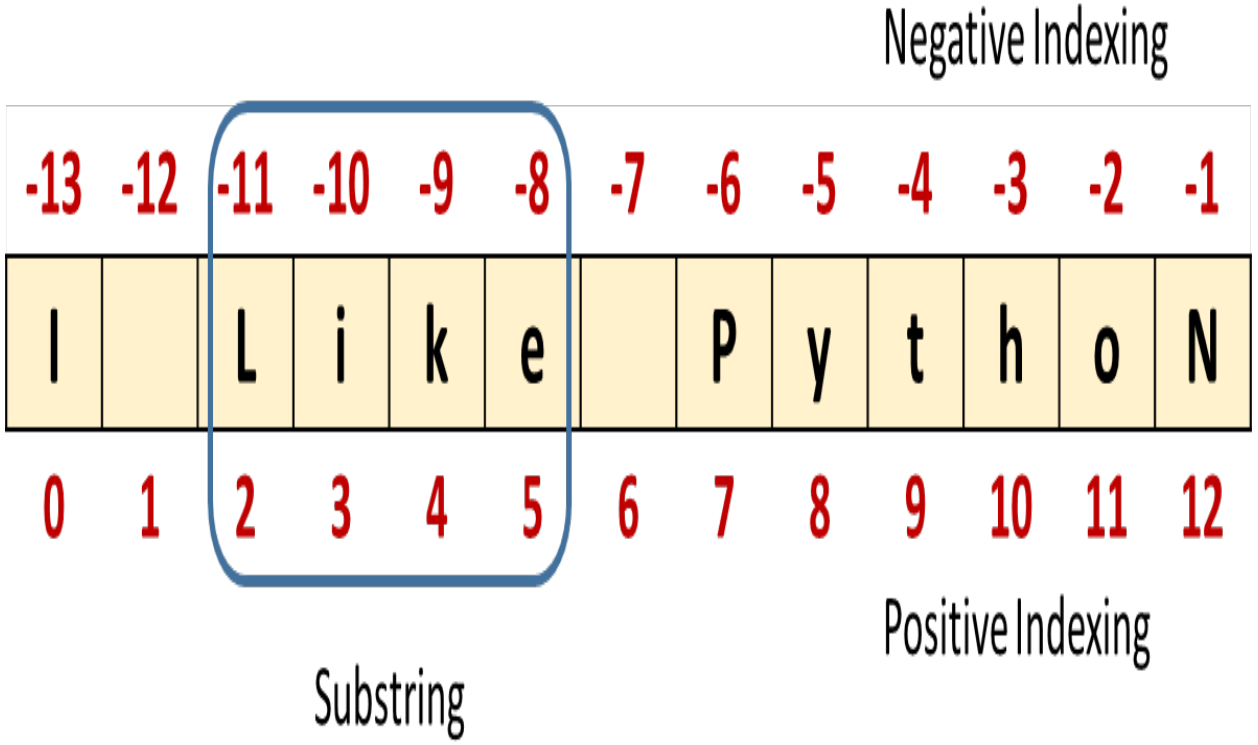
```
st = 'I Like Python' # A given string

print(st[2])    #print 'L' using postive index
print(st[-11])  #print 'L' using negative index

print(st[7])    #print 'P' using postive index
print(st[-6])   #print 'P' using negative index
```

```
L
L
P
P
```



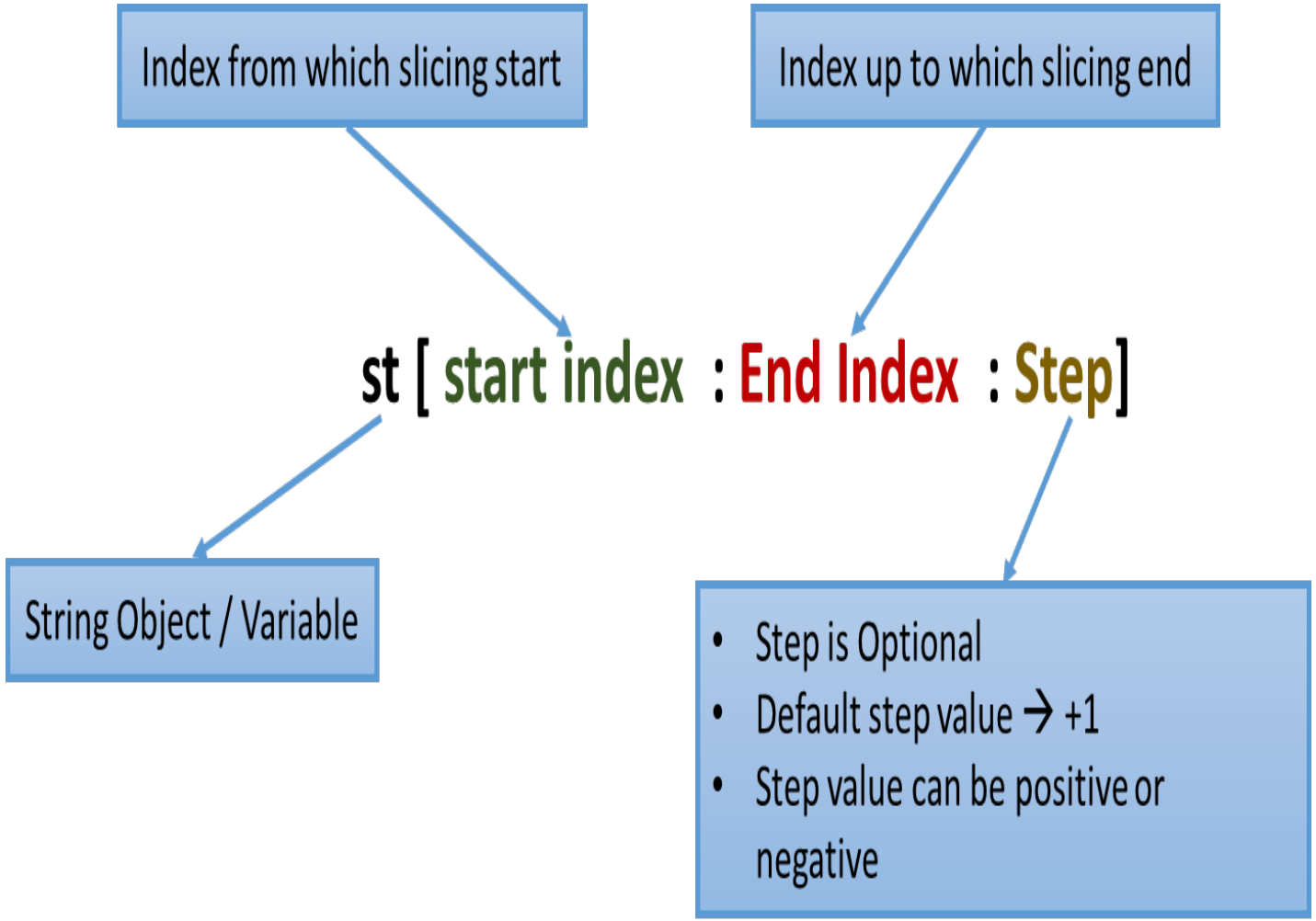
String Slicing

- **String slicing is the way of selection of substring.**
- String **'Like'** is substring in given 'I Like Python' string.
- [] we can access string characters by giving indexes if we use colon inside square bracket like **[:]** it becomes a slicing operator in Python.



Syntax of String Slicing

- Start index: Index from which slicing starts.
- End Index: Index up to which slicing end.
- Step value is optional.



Examples of String Slicing

Example	Explanation	Syntax

st[2 : 6]	It starts with the 2nd index and ending with (6It starts with the 2nd index and ending with (6-1=5)th index.-1=5)th index.	`st = 'I Like Python'`
		`print(st[2:6])`
		`Like`

st[0 : 6 : 2]	It starts with index 0th and end with (6-1=5) th. Step is 2 So, it will give value at index 0,2,4	`st = 'I Like Python'`
		`print(st[0:6:2])`
		`ILk`

st[12 : 6 : -1]	It starts slicing from 12th index up to 6-1=5th index. 12th to 6th in opposite direction because step is negative.	`st = 'I Like Python'`
		`print(st[12:6:-1])`
		`nohtyP`

Contd..

Example	Explanation	Syntax

st[-11 : -7]	It starts with -11th index and ending with (-7-1 = -8)th index.	`st = 'I Like Python'`
		`print(st[-11:-7])`
		`Like`

[:]	The operator gives the complete original string.*	`st = 'I Like Python'`
		`print(st[:])`
		`I Like Python`

[: 6]	Start index is missing and step is +1, So it will start from 0 till 6-1=5th index.*	`st = 'I Like Python'`
		`print(st[:6])`
		`I Like`

Contd..

Example	Explanation	Syntax

st[7 :]	It will start from index 7 to last index (Last index missing)*	`st = 'I Like Python'`
		`print(st[7:])`
		`Python`

st[: : -1]	Step is -1, So start index will be the last index and end index will be first index when start and end is missing.	`st = 'I Like Python'`
		`print(st[::-1])`
		`nohtyP ekiL I`

st[2 : 6 : -1]	This is explaining in Note Section ** Output – Empty string	`st = 'I Like Python'` `print(st[2:6:-1])`

Important Points to Remember

Note about *, **

- * When start index is missing it will start from either first character or from last. It depends on step sign (positive or negative).
- * When end index is missing it will execute till last character or first character. It depends on step sign (positive or negative).
- * When we take negative steps it will scan from start to end in opposite direction.
- ** In the case of step is positive or negative the slicing will be done as below given algorithm.

Comparison

When Step is Positive	When Step is negative

st [start : end : +1]
Working of slicing
i = start
while i<end:
//do scanning
i = i +1
	st [start : end : -1]
	Working of slicing
	i = start
	while i>end:
	//do scanning
	i = i -1

Example –
st[6:2:1]
i = 6
while i < 2:
//do scanning
i = i +1
Expression i<2 will evaluate to False and loop will terminate and empty string will be slice.
	Example –
	st[2:6:-1]
	i = 2
	while i > 6:
	//do scanning
	i = i -1
	i>6 will evaluate to False and loop will terminate and empty string will be slice.

Updation in String

- Strings in python are Immutable(un-changeable) sequences, which means it does not support new element assignment using indexes.

- Lets try to understand this concept with the help of an example

```
st = 'I Like Python'
st[0] = 'i'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-ee7a862518bd> in <module>
      1 st = 'I Like Python'
----> 2 st[0] = 'i'

TypeError: 'str' object does not support item assignment
```

In the above example string does not allow assigning a new element, because item assignment does not support by string.

Contd..

1. What will be the output of the following code snippets?

```
message="welcome to Mysore"
word=message[-7:]
if(word=="mysore"):
    print("got it")
else:
    message=message[3:14]
    print(message)
```

A. come to Myso

B. come to Mys

C. lcome to Mys\

D. lcome to Myso

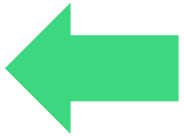
Contd..

2. What will be the output of the following code Comparison?

```
print("ABES" > "AKTU")
print("AKTU" < "ABES")
```

A. True
   False

B. False
   False



Contd..

3. What will be the output of the following code snippet?

```
example = "ABES Engineering College"
print("%s" % example[2:7])
```

A. ES

B. En

C. ES En

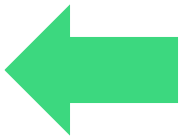
D. ESEn

Contd..

4. What will be the output of the following code snippet?

```
s1 = 'hellohow'
for i in range(len(s1)):
    print(s1,end="")
    s1 = 'a'
```

A. hellohowa
B. hellohowaaaaa
C. hellohowaaaaaaa
D. Error



Session Plan - Day 2

3.1 String

- Built in Methods
- Basic Operations
- String Formatters
- Loops with Strings
- Review Questions
- Practice Exercises

Deletion

- In String, we can't reassign the string characters but we can delete the complete string using **del command.**

- Lets try to understand this concept with the hep of an example

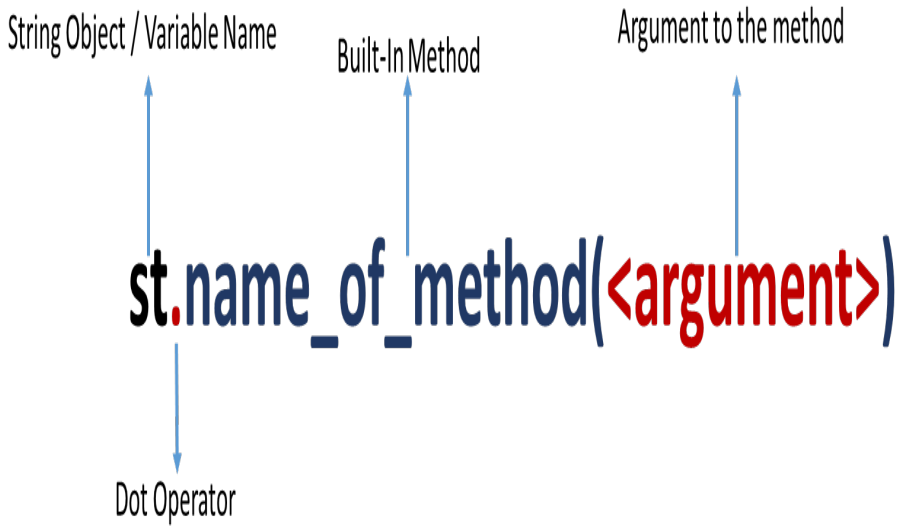
```
st = 'I Like Python'
print(st)
del st
print(st)
```

```
I Like Python

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-933f7d08d696> in <module>
      2 print(st)
      3 del st
----> 4 print(st)

NameError: name 'st' is not defined
```

In the above example , after deletion when we try to print the string st it gives NameError : st not defined.

String Built in methods:

- String supports a variety of Built-In methods for achieving different types of functionalities.
- All built-in methods provide us to use as plug and play, we do not have to implement it.
- In python methods are called using dot(.) operator



String Built in methods:

Method Name	Explanation	Code Snippet

capitalize()	Return a string with first letter capital
	e.g.
	i like python → I Like Python
		`st = 'i like python'`
		`stnew = st.capitalize()`
		`print(stnew)`
		`I like python`

casefold()	Return string in lowercase	`st = 'i Like pythoN'`
		`stnew = st.casefold()`
		`print(stnew)`
		`i like python`

count()	Return the number of occurrence of substring.
	In the first example "**i**" occurred 3 times.
	In the second example "**like**" occurred 2 times.
		`st = 'i like what you like'`
		`n = st.count('i')`
		`print(n)`
		`3`
		`st = 'I like what you like'`
		`n = st.count('like')`
		`print(n)`
		`2`

String Built in methods:

Method Name	Explanation	Code Snippet

endswith()	Return **True** if the string ends with the given substring otherwise return **False.**	`st = 'abes.ac.in'`
		`print(st.endswith('in'))`
		`True`

startswith()	Returns **True** if the string starts with the given substring otherwise return **False.**	`st = 'abes.ac.in'`
		`print(st.startswith('abes'))`
		`True`
		
		`st = 'abes.ac.in'`
		`print(st.startswith('abesec'))`
		`False`

find()	Return the lowest index in String where substring sub is found.
	In example – there are two “like”.
	1st “like” is at index-2 and 2nd at index-16. It return 2.
		`st = 'I like what you like'`
		`n = st.find('like')`
		`print(n)`
		`2`

String Built in methods:

Method Name	Explanation	Code Snippet

lower()	Convert and return a string into lower case	`st = 'I Like Python'`
		`stnew = st.lower()`
		`print(stnew)`
		`i like python`

upper()	Convert and return a string into upper case	`st = 'I Like Python'`
		`stnew = st.upper()`
		`print(stnew)`
		`I LIKE PYTHON`

swapcase()	In swaps cases, the lower case becomes the upper case and vice versa	`st = 'I Like Python'`
		`stnew = st.swapcase()`
		`print(stnew)`
		`i lIKE pYTHON`

String Built in methods:

Method Name	Explanation	Code Snippet

title()	Converts the first character of each word to upper case.	`st = 'i like python'`
		`stnew = st.title()`
		`print(stnew)`
		`I Like Python`

replace()	Returns a string after replacing old substring with new substring.	`st = 'I like python'`
		`stnew = st.replace('python','java')`
		`print(stnew)`
		`I like java`

split()	Splits the string from given separator and returns a list of substring.
	Note –
	By default, value of separator is a whitespace
		`st = 'I Like Python'`
		`stnew = st.split()`
		`print(stnew)`
		`['I', 'Like', 'Python']`
		
		`st = 'Sunday,Monday,Tuesday'`
		`stnew = st.split(',')`
		`print(stnew)`
		`['Sunday', 'Monday', 'Tuesday']`

String Built in methods:

Method Name	Explanation	Code Snippet

join()	Concatenate any number of strings. The string whose method is called is inserted in between each given string. The result is returned as a new string	`st = 'Python'`
		`stnew = '-'.join(st)`
		`print(stnew)`
		
		`P-y-t-h-o-n`

strip()	Return a copy of the string with leading and trailing whitespace removed.	`st = '  I Like Python  '`
		`stnew = st.strip()`
		`print(stnew)`
		
		`I Like Python`

isalnum()	Return true if the string is alphanumeric (Combination of a-z, A-Z, 0-9)
	Note – It will return false only if string contains special character.
		`st = 'Python3'`
		`print(st.isalnum())`
		
		`True`
		
		`st = 'Python-3'`
		`print(st.isalnum())`
		
		`False`

String Built in methods:

Method Name	Explanation	Code Snippet

isalpha()	Return true if the string contains only alphabets(Combination of a-z, A-Z)	`st = 'Python'`
		`print(st.isalpha())`
		`True`

isdecimal()	Returns True if all characters in the string are decimals	`st = '123'`
		`print(st.isdecimal())`
		`True`
		`st = 'Python34'`
		`print(st.isdecimal())`
		`False`

islower()	Returns True if all characters in the string are lower case	`st = 'python'`
		`print(st.islower())`
		`True`
		`st = 'Python'`
		`print(st.islower())`
		`False`

String Built in methods:

Method Name	Explanation	Code Snippet

len()	Return length(No of character) of a given string. This is a generic function.	`st = 'Python'`
		`print(len(st))`
		`6`

Basic Operations in String:

- Strings in Python support basic operations like **concatenation**, **replication and membership**

  - **Concatenation** means joining/combining two string into one.
  - **Replication** means repeating same string multiple times.
  - **Membership** tells a given string is member of another string or not.

Operations in String:

Method Name	Explanation	Code Snippet

+	Concatenation
	
	It will merge/join second string at the end of first string and return.
		`st1 = 'Go'`
		`st2 = 'ing'`
		`print(st1+st2)`
		
		`Going`

*	Multiply (Replicas)
	It will repeat same string multiple times and return.
	Note – multiply string only with integer number.
		`st = 'Python'`
		`print(st*3)`
		
		`PythonPythonPython`

in	Membership
	It will check a given substring is present in another string or not.
	Note – gives bool value (True/False)
		`st = 'I Like Python'`
		`print('Python' in st)`
		
		`True`
		
		`st = 'I Like Python'`
		`print('Java' in st)`
		
		`False`

Operations in String:

Method Name	Explanation	Code Snippet

Not in	It's reverse of Membership	`st = 'I Like Python'`
		`print('Java' not in st)`
		
		`True`

String Formatters:

- String formatting is the process of infusing things in the string dynamically and presenting the string.

Why to use String Formatter??

- For different types of requirement to print the string in a formatted manner.
- Here, format implies that in what look and feel we want our strings to get printed.
- Python string provides a number of options for formatting.

String Formatters:

- String formatting is the process of infusing things in the string dynamically and presenting the string.

Why to use String Formatter??

- For different types of requirement to print the string in a formatted manner.
- Here, format implies that in what look and feel we want our strings to get printed.
- Python string provides a number of options for formatting.

String format Style:

Escape Sequence	Explanation	Example

\newline	Ignores newline	`st = 'Hello "how" are you.\`
		`I am fine'`
		`print(st)`
		
		`Hello "how" are you.I am fine`

\\	Backslash
	
	Write a backslash in string
		`st = 'https:\\\\abes.ac.in'`
		`print(st)`
		
		`https:\\abes.ac.in`

\'	If we need to use single quotes in our string like
	Good Morning! Mr. 'BOB'
		`st = 'Good Morning! Mr. \'BOB\''`
		`print(st)`
		
		`Good Morning! Mr. 'BOB'`

String format Style

Escape Sequence	Explanation	Example

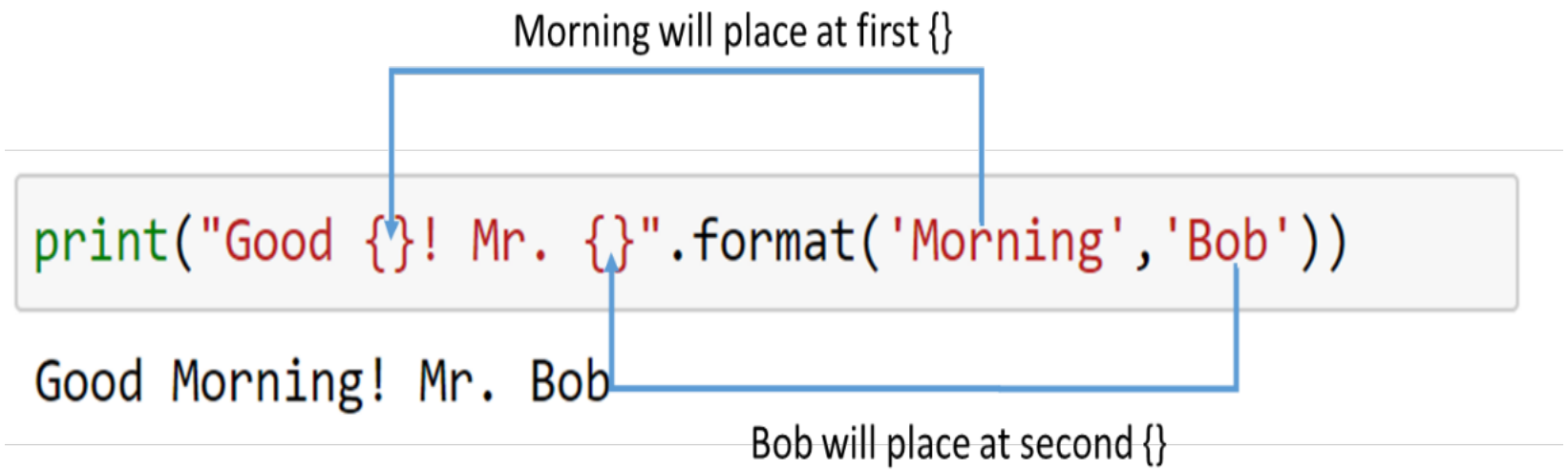
\"	If we need to use double quotes in our string like
	Good Morning! Mr. 'BOB'
		`st = "Good Morning! Mr. \"BOB\" "`
		`print(st)`
		`Good Morning! Mr. "BOB"`

\n	Newline	`st = "Good Morning!\nMr. BOB"`
		`print(st)`
		`Good Morning!`
		`Mr. BOB`

String Format()

Method 1: It is a beneficial method for formatting strings; it uses {} as a placeholder.

- We have placed two curly braces and arguments that give the format method filled in the same output.



String Format()

- If we want to change the order of , we can give an index of parameters of format method starting with 0th index.

```
print("{0} and {1} play football".format('Bob','Ram'))
```

```
Bob and Ram play football
```

0th index 1st index

```
print("{1} and {0} play football".format('Bob','Ram'))
```

```
Ram and Bob play football
```

String Format()

- If we want to change the order of , we can give an index of parameters of format method starting with 0th index.

```
print("{0} and {1} play football".format('Bob','Ram'))
```

```
Bob and Ram play football
```

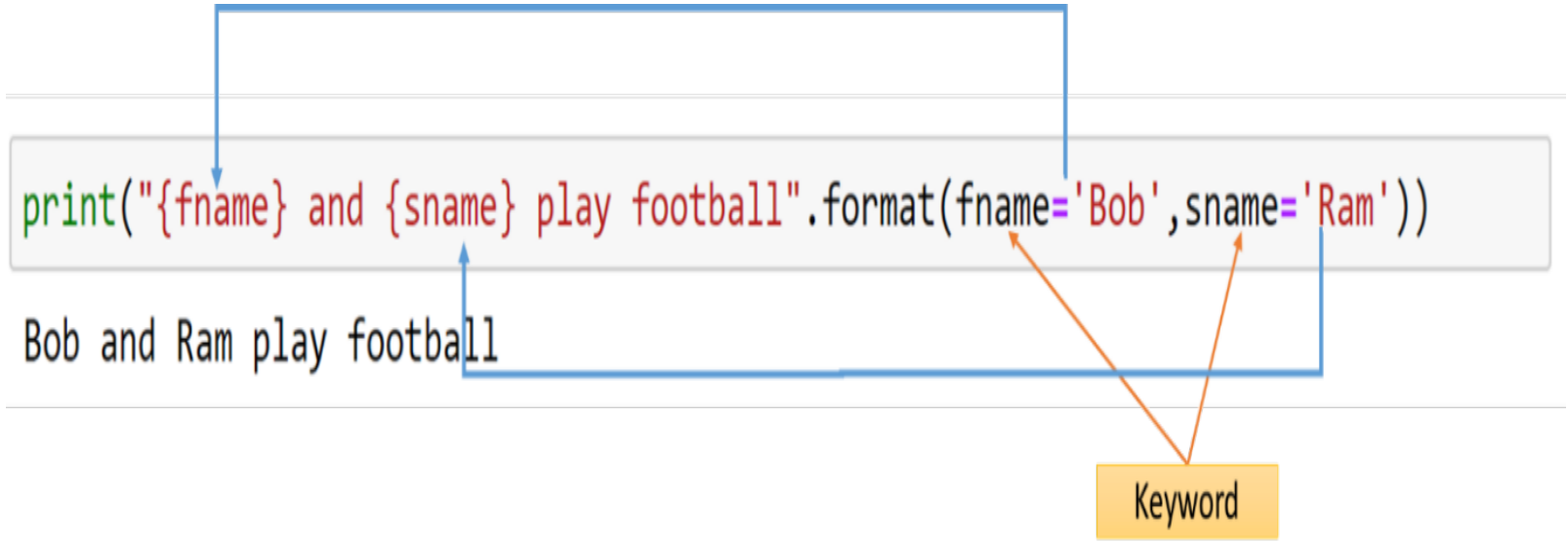
0th index 1st index

```
print("{1} and {0} play football".format('Bob','Ram'))
```

```
Ram and Bob play football
```

String Format()

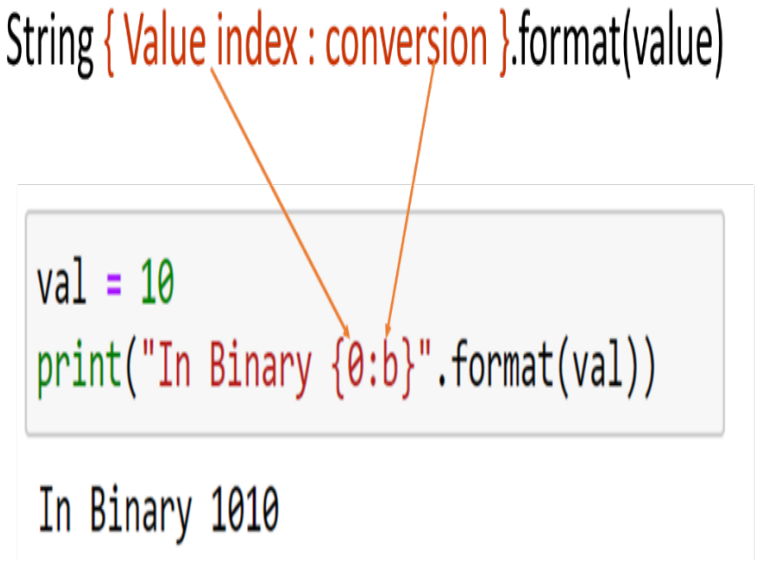
- We can also use keywords arguments in format method, as shown in the following example, as shown in figure.



String Format()

- We can also use format specifier in format method like in language ‘C’. Format specifier are used to do following.

  - **Represent value of amount** = 12.68456 at two decimal place
  - **Represent value of integer in Binary, Octal or Hexadecimal** etc.



Note: In the above example b is used for binary representation.

Format Specifiers

Format Specifier	Explanation	Example

b	Use for Binary	`val = 10`
		`print("In Binary {0:b}".format(val))`
		`In Binary 1010`

o	Use for Octal	`val = 10`
		`print("In Octal {0:o}".format(val))`
		`In Octal 12`

X or x	Use for Hexadecimal	`val = 10`
		`print("In Hexadecimal {0:x}".format(val))`
		`print("In Hexadecimal {0:X}".format(val))`
		`In Hexadecimal a`
		`In Hexadecimal A`

Format Specifiers

Format Specifiers	Explanation	Example

d	Use for Decimal	`val = 10`
		`print("In Decimal {0:d}".format(val))`
		
		`In Decimal 10`

f	Use for Float
	Note – Place **.n** before **f**, for representing floating point precision. n is decimal places.
		`val = 10.8934`
		`print("In Float {0:f}".format(val))`
		`print("Two decimal point {0:.2f}".format(val))`
		
		`In Float 10.893400`
		`Two decimal point 10.89`

Formatting using f-string

- Fstring is the way of formatting as **format method** does but in an easier way.
- We include ‘f’ or ‘F’ as a prefix of string.

```
a = 10
b = 20
c = a+b
print(f"Sum of {a} and {b} = {c}")
```

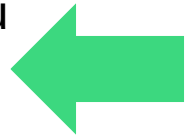
```
Sum of 10 and 20 = 30
```

Can you answer these questions?

1. What will be the output of the following code snippet?

```
line = "Hello how are you"
L = line.split('a')
for i in L:
    print(i, end=' ')
```

A. Hello how are you
B. Hello how are
C. hello how are you
D. Hello how re you



Can you answer these questions?

2. What will be the output of the following code snippet?

```
example="ABES Engineering College"
example[::-1].startswith("A")
```

A. True

B. False

C. Error

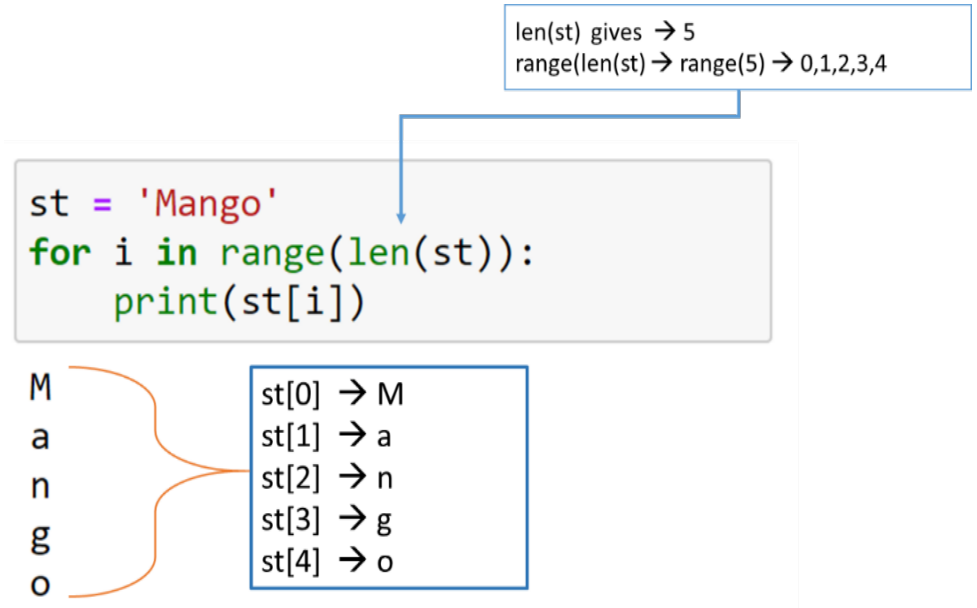
D. None

Loops with Strings:

- Strings are sequence of character and iterable objects, so we can directly apply for loop through string.

Example-1

Scan/Iterate each character of string through index using for loop.



Example -2

Scan/Iterate each character of string directly using for loop

```
st = 'Mango'
for val in st:
    print(val)
```

```
M
a
n
g
o
```

Loops with Strings:

Example 3 – Write a program to print number of alphabets and digits in a given string.

```
st = 'NH-24, Delhi Hapur By pass Vijay Nagar 201009'
alphabet = 0
digit = 0
for val in st:
    if val.isalpha():
        alphabet = alphabet + 1
    if val.isdigit():
        digit = digit +1
print(f"Total No of alphabet = {alphabet}")
print(f"Total No of Digit = {digit}")
```

```
Total No of alphabet = 28
Total No of Digit = 8
```

Loops with Strings:

Example 4

To add 'ing' at the end of a given string (length should be at least 3).

- If the given string already ends with 'ing' then add 'ly' instead.
- If the string length of the given string is less than 3, leave it unchanged.

Sample String : 'abc'

Expected Output : 'abcing'

Sample String : 'string'

Expected Output : 'stringly'

.

```
st = input("Enter any String")
if len(st)>=3:
    if st.endswith('ing'):
        print(st+'ly')
    else:
        print(st+'ing')
else:
    print(st)
```

```
Enter any Stringabc
abcing
```

Session Plan - Day 3

3.2 List

- Creation
- Accessing
- Updation
- Review Questions
- Practice Exercises

List

- ❑ Python **List** is the most commonly used sequence.

❑ Important points about list are as follows.

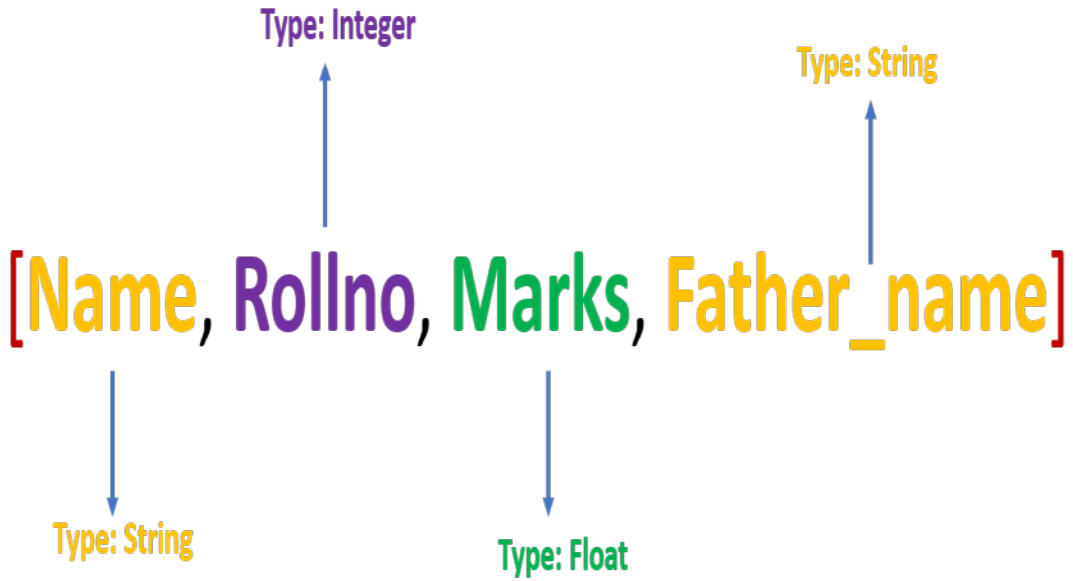
- List elements are enclosed in **square brackets []** and are comma separated.
- List is the sequence of class type '**list'.**
- List can contain elements of different data types.
- List is a mutable(changeable) sequence, would be discussed in detail in 3.2.3
- List allows duplicate elements.
- List elements are ordered, it means it give specific order to the elements, if new element is added, by default it comes at the end of the list.

List

Real World Scenario:

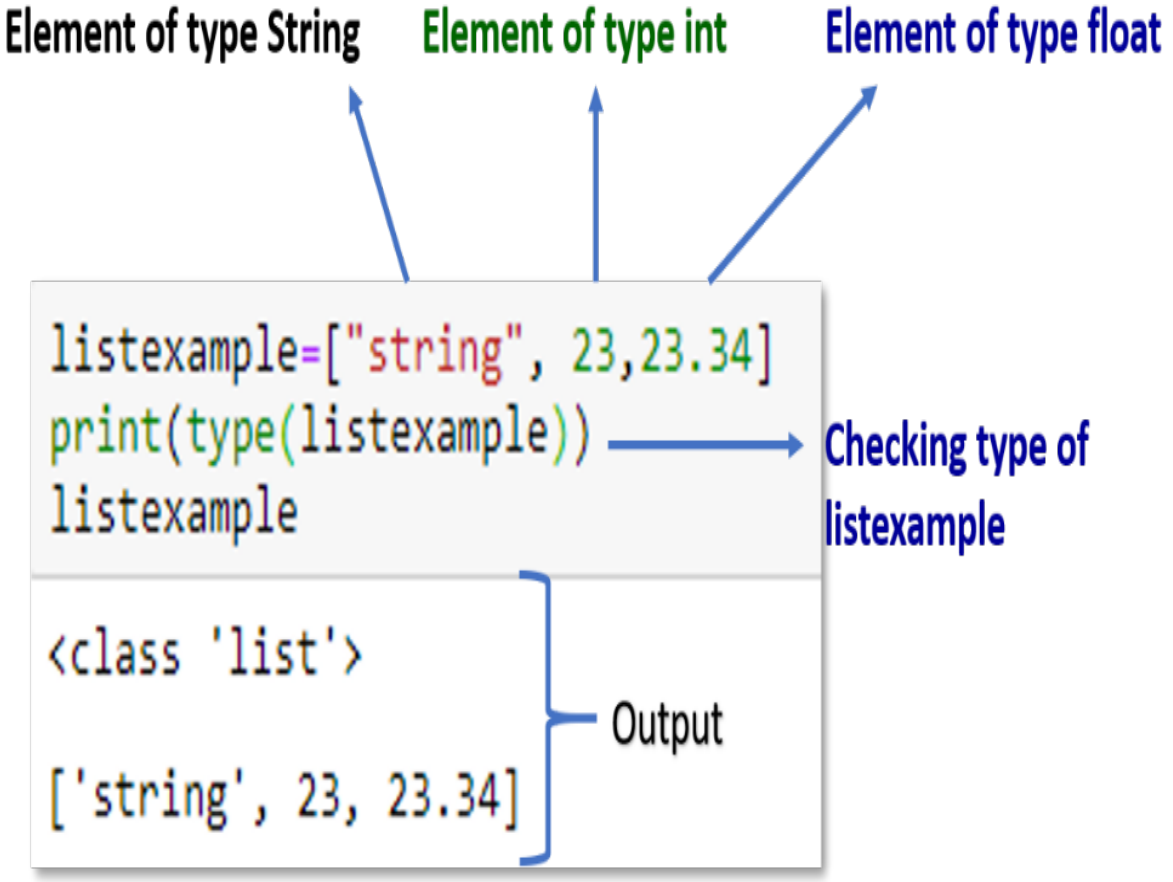
- List is used when there is a possibility of elements of different data type.
- For example record of a particular student, having name as string, roll.no as integer, marks as float, father's name as string.
- To contain this record list is the appropriate sequence.

List Representation



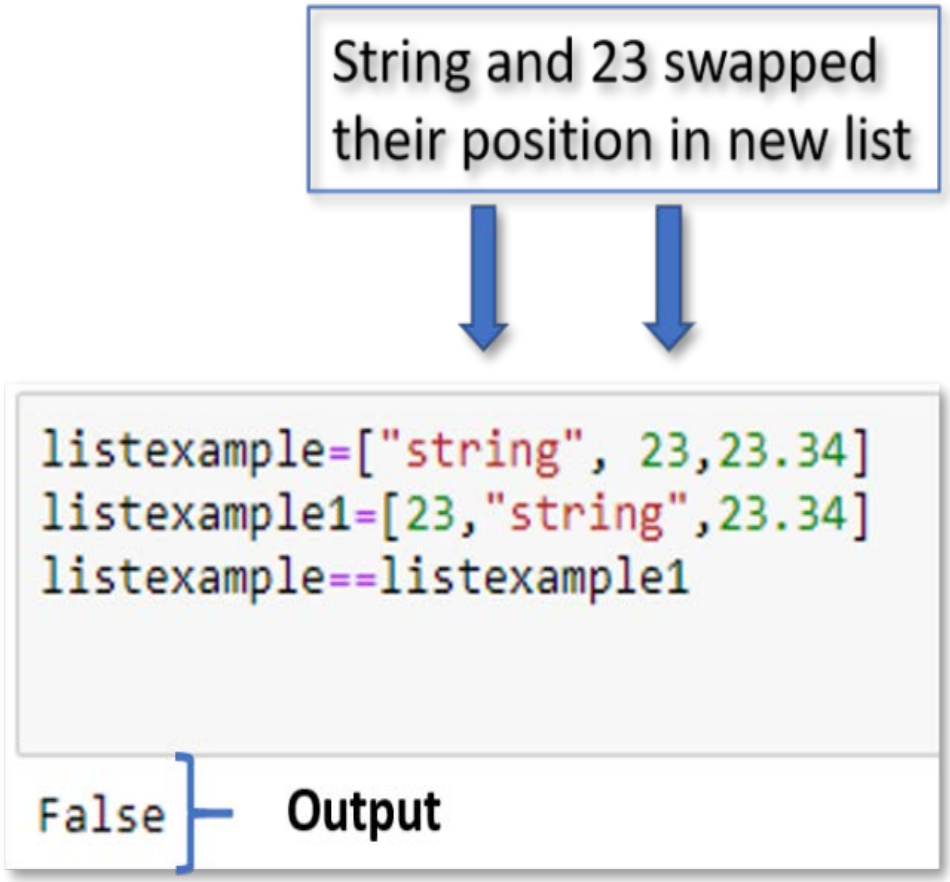
List Example

- In this example a variable named list example has been created and three elements have been assigned.
- As we have enclosed elements in square brackets, this makes list example is of type list.



Ordered Property of List

- Ordered sequence of the list means the order in which elements appear is unique
- The position of every element is fixed.
- If we change the position of any element, then list would not be the same anymore.



Creation of List

List can be created by many ways as follows.

Creation of Empty List

```
s=list()
print(s)
```

[] → Empty list created

Using list class

```
s=[]
print(s)
```

[] → Empty list created

Using empty square brackets

Creation of Non Empty List

```
s=list([1,"Hello"])
s
```

[1, 'Hello'] → list created

Using list class

```
s=[1,"Hello"]
s
```

[1, 'Hello'] → list created

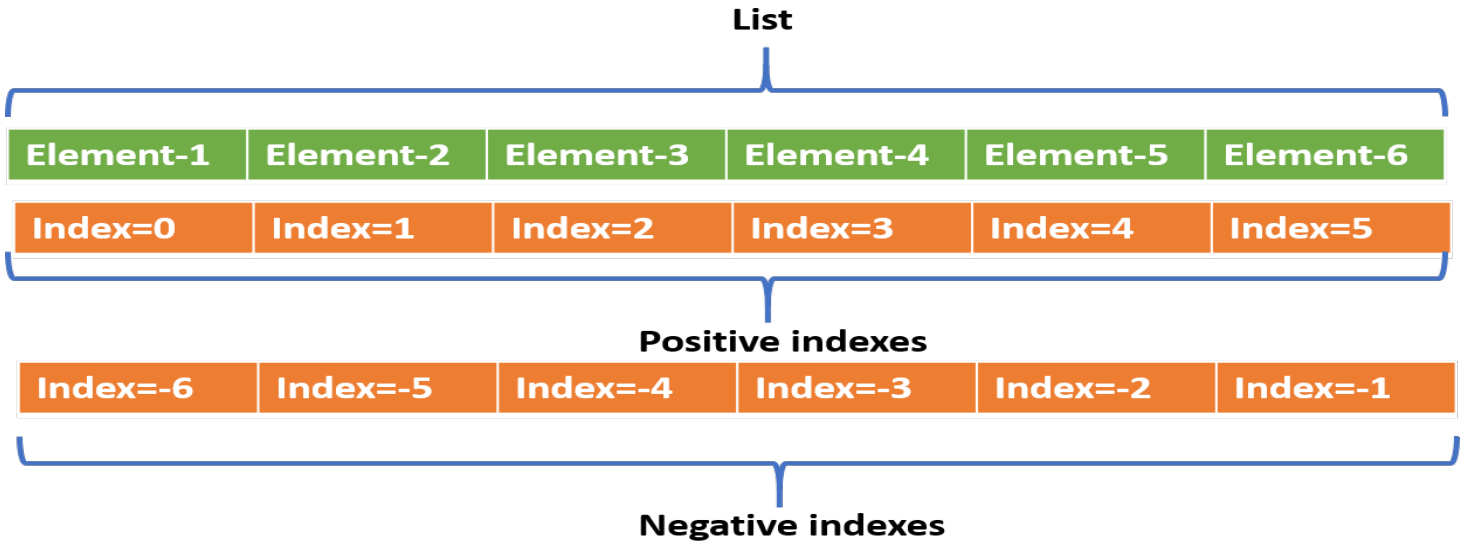
Using square brackets only

Accessing the List

Indexing the List

- In the list index are started from 0, it means first element takes 0 index and it increases like 0,1,2…(n-1).
- List also supports negative indexing.

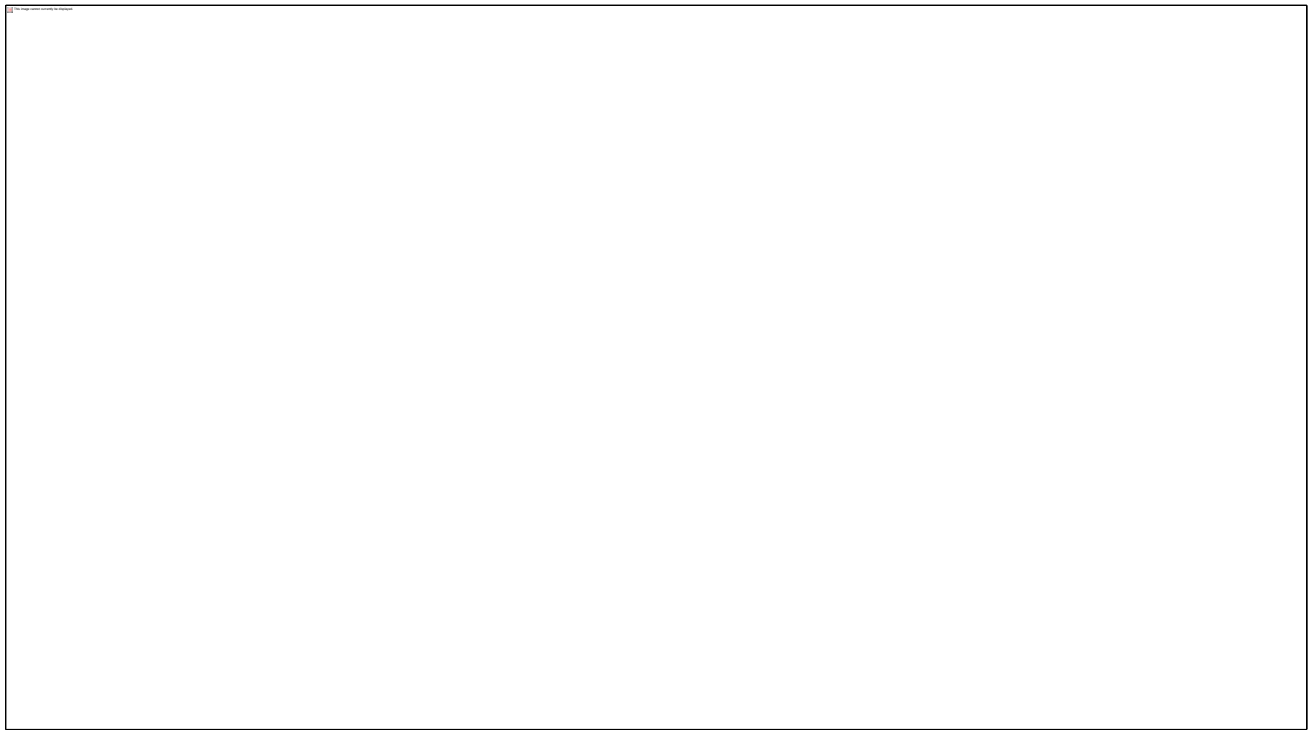
- It means last element can be accessed using -1 index, second last as -2 and so on.



Slicing

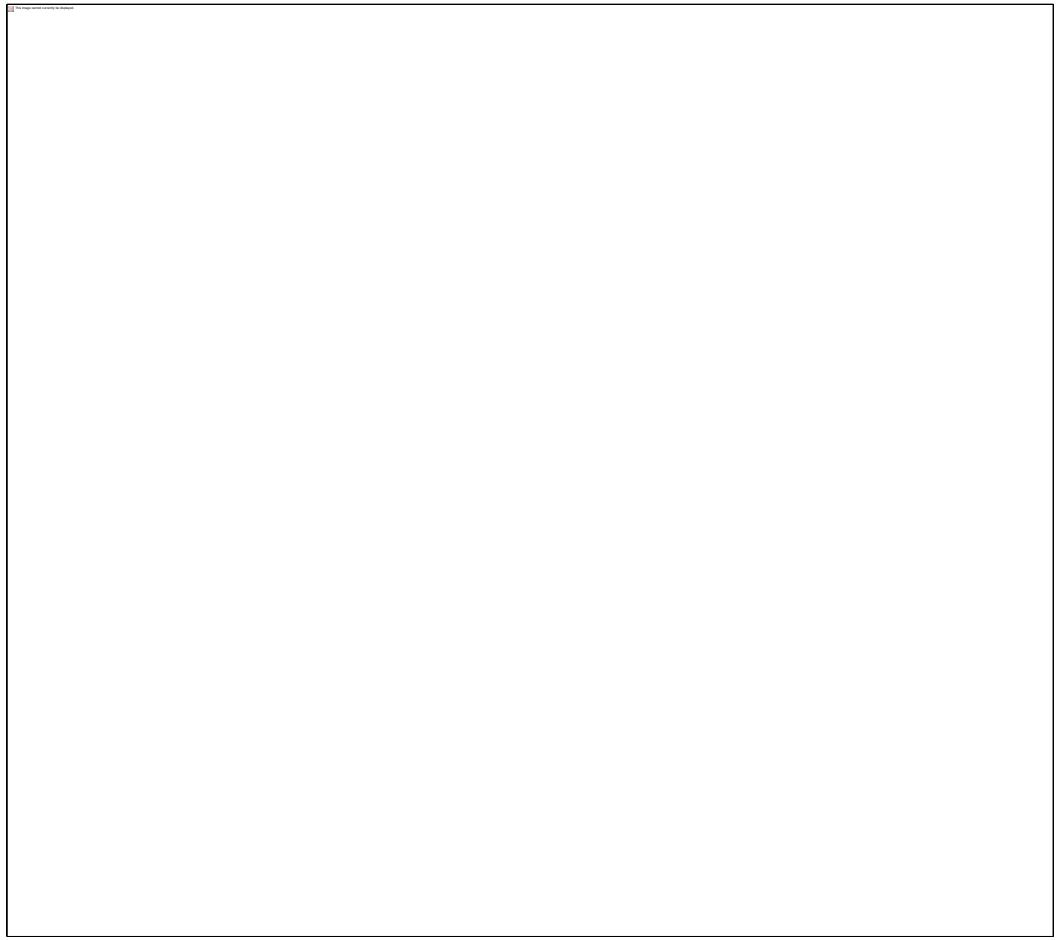
- Slicing is used to get substring from a string.
- We use Slice operator in the list to get a sub list.

Start= Optional argument by default value is 0.
End = Optional argument by default value is number of elements in the list. If any number given, then value is taken as number-1.
Step=Options argument by default value is 1.
Colon 1: Required
Colon 2: Optional



Slicing Example

- Single colon[:]- It is used to print the entire list.
- Double colon[:]- It is used to print the entire list.
- l[2::]-It is used to print the list starting from index 2 till the last element of the list.
- L[2:5:]- We have set start and end both as 2 and 5, so it's starting from 2 and ending with (5-1) and printing values for indexes 2,3 and 4 index.
- L[2:5:2]-end and step has been set as 2,5 and 2. So output substring starting from index 2, ending with 5-1=4 and step counter is 2, so its skipping alternate element.



Examples of List Slicing

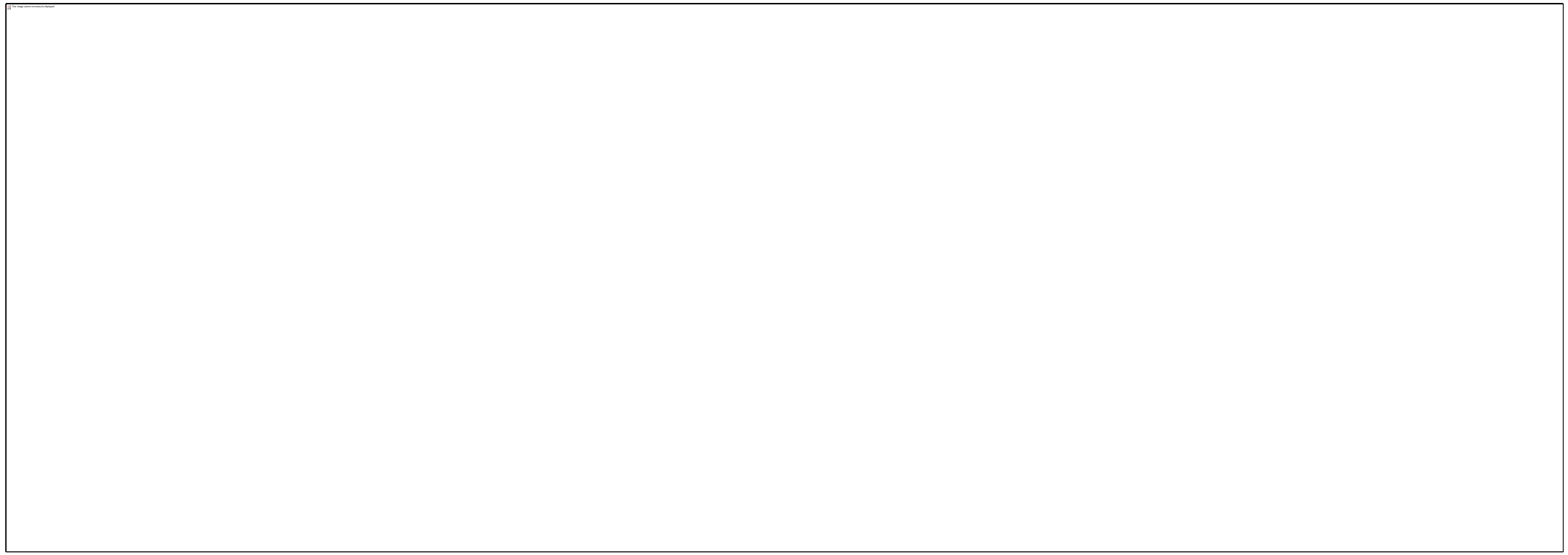
Operator	Explanation	Example
[:]	This gives a complete original list	
[1:3]	It starts with 1th index and ending with (3-1=2) index	
[2:]	Sub list starts with index 2 and ends at last element as nothing specified on right side of the colon	

Examples of List Slicing

Operator	Explanation	Example
[1:4:]	Start=1, stop=4 and by default step=1.	
[1:4:2]	Start=1, stop=4 and by step=2. As the step is 2, it took every 2nd element.	
[::-1]	Starting is 0th index and stopping at last index and step is negative count so that it would print in reverse order	

Updation in List:

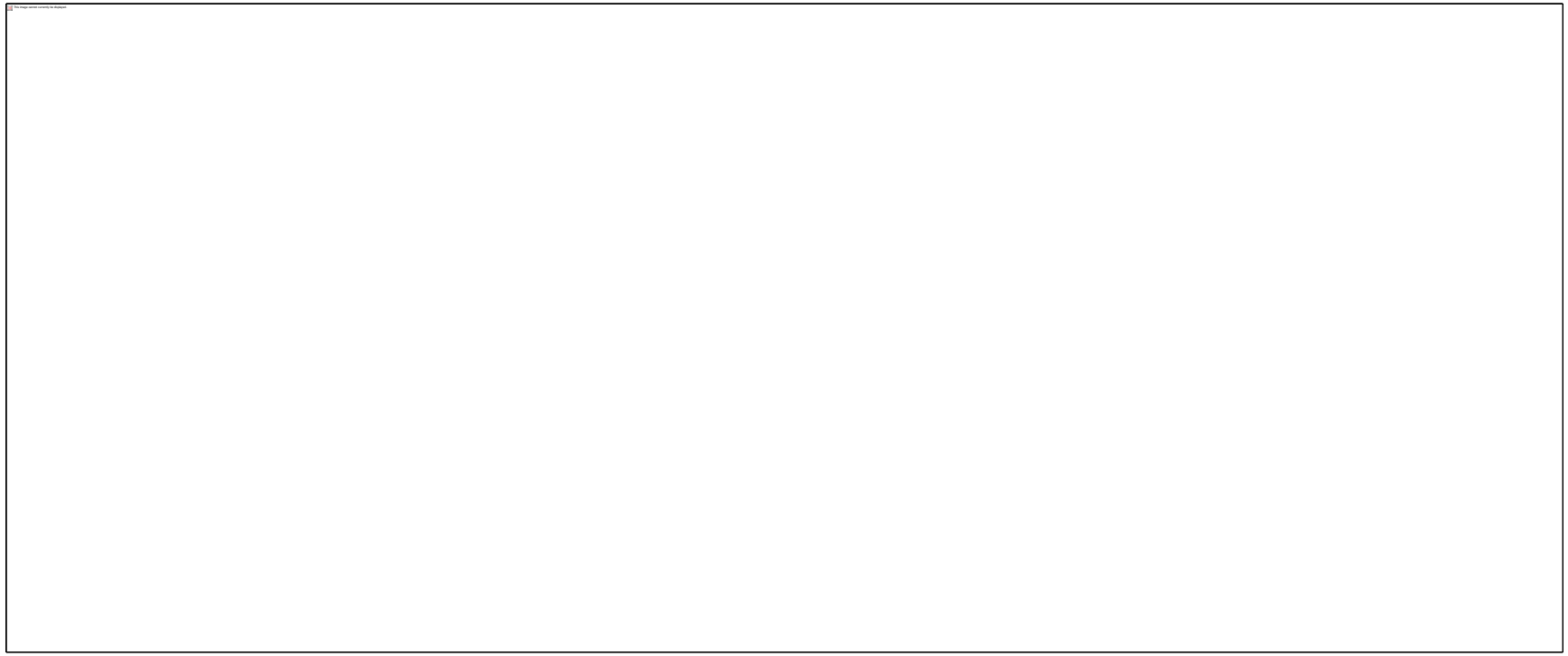
- List updating involves insertion of new elements, deletion of elements or deletion of complete list.
- List is a mutable sequence it means it allows changes in the elements of list.



Updation in List:

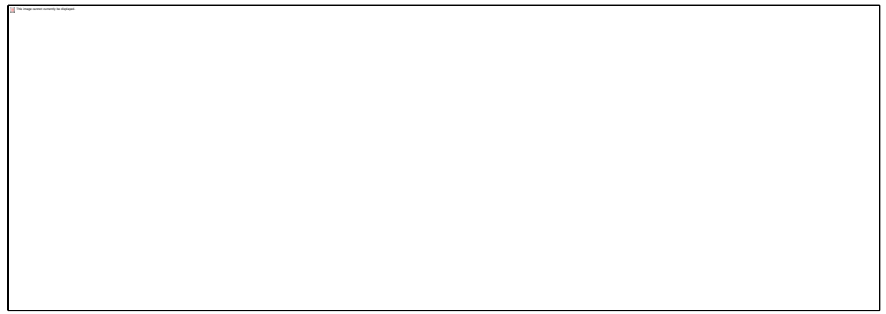
- List updating involves insertion of new elements, deletion of elements or deletion of complete list.

- List is a mutable sequence it means it allows changes in the elements of list.



Can you answer these Questions

1.What is the output of the following code?



A.[‘XY’,’YZ’]

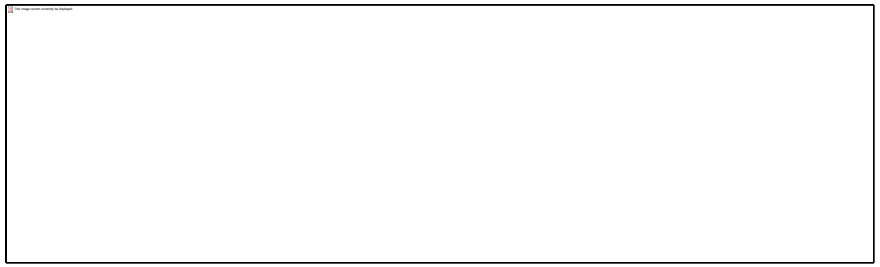
B.[‘xy’,’yz’]

C. None of the above

D. Both of the above

Can you answer these Questions

2.What is the output of the following code?



A. [‘g’,’h’,’k’,’l’,’m’]

B. [‘g’,’h’,’k’,’l’,’m’,9]

C. [‘g’,’h’,’k’,’l’,’m’,8,9]

D. [‘g’,’h’,’k’,’l’,’m’,9,8]

Session Plan - Day 4

3.2 List

- Updation in List
- Operations in List
- Built in Methods
- Operations
- Loops
- Nested list
- List Comprehension
- Review Questions
- Practice Exercises

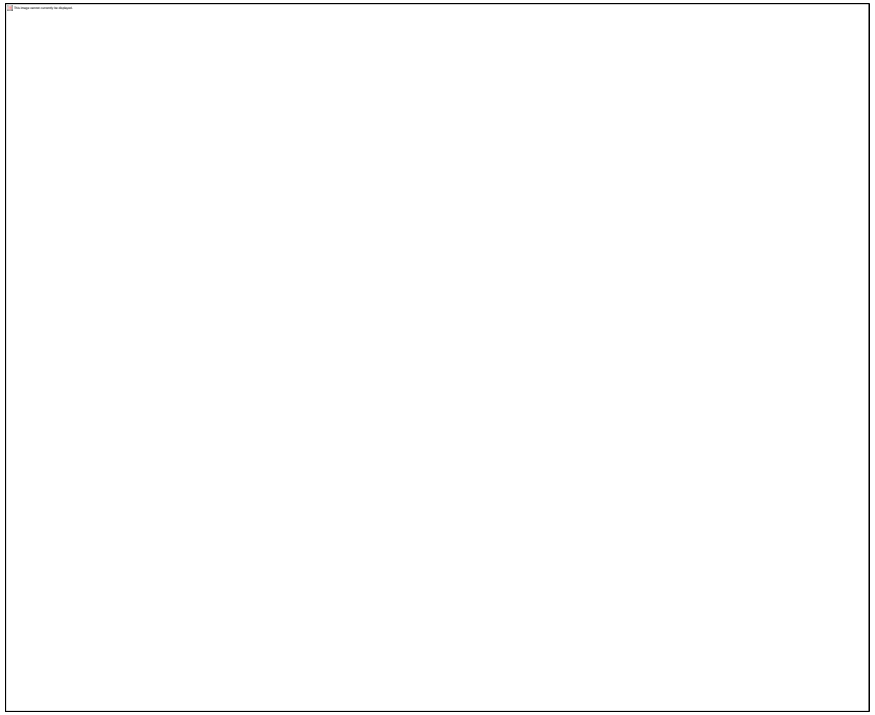
Add new element in List:

- We can add elements to the existing list using append function.
- Append function always add the element at the end of the list.

Syntax:

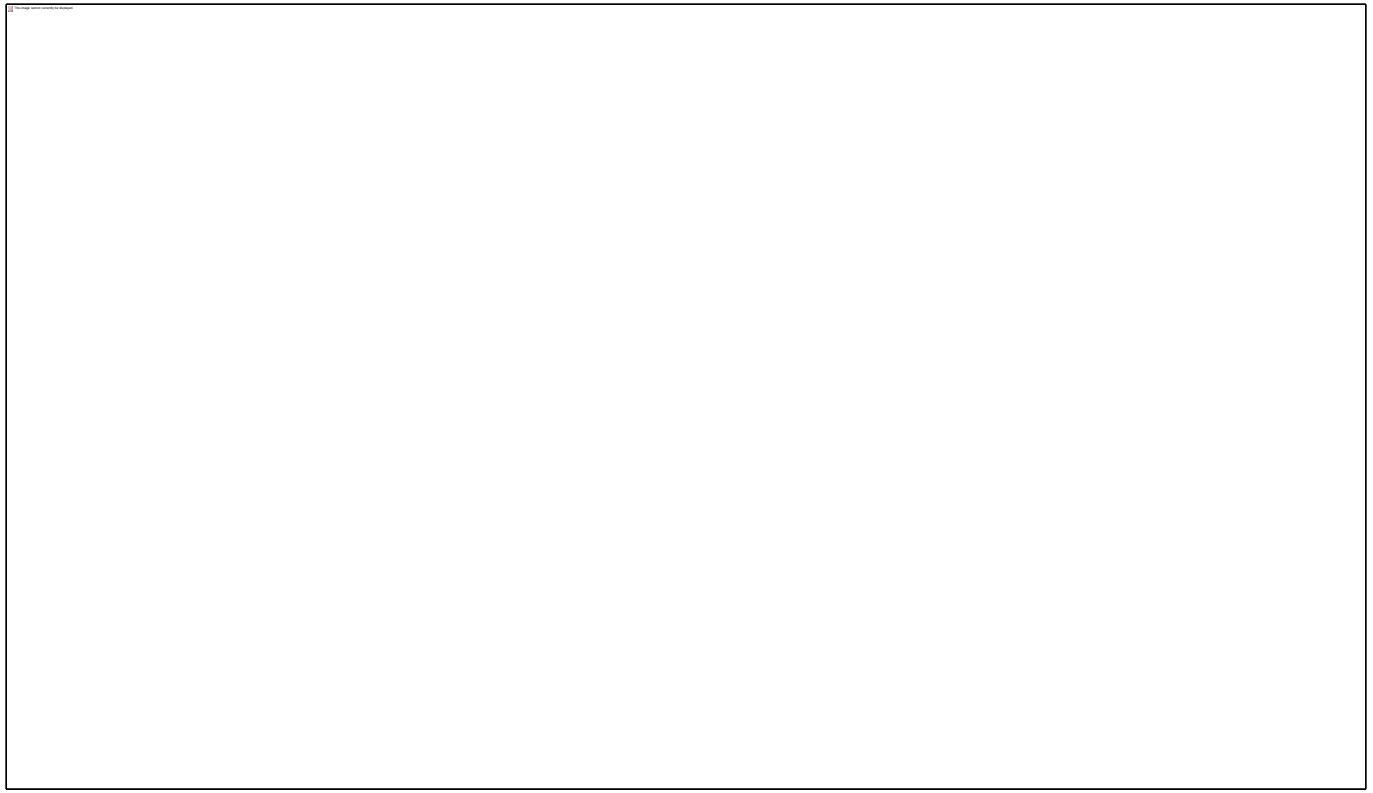
Listname.append (element to be added)

In this example element 5 has been appended in the originally existed list. It's being added in the last



Changing new element in List:

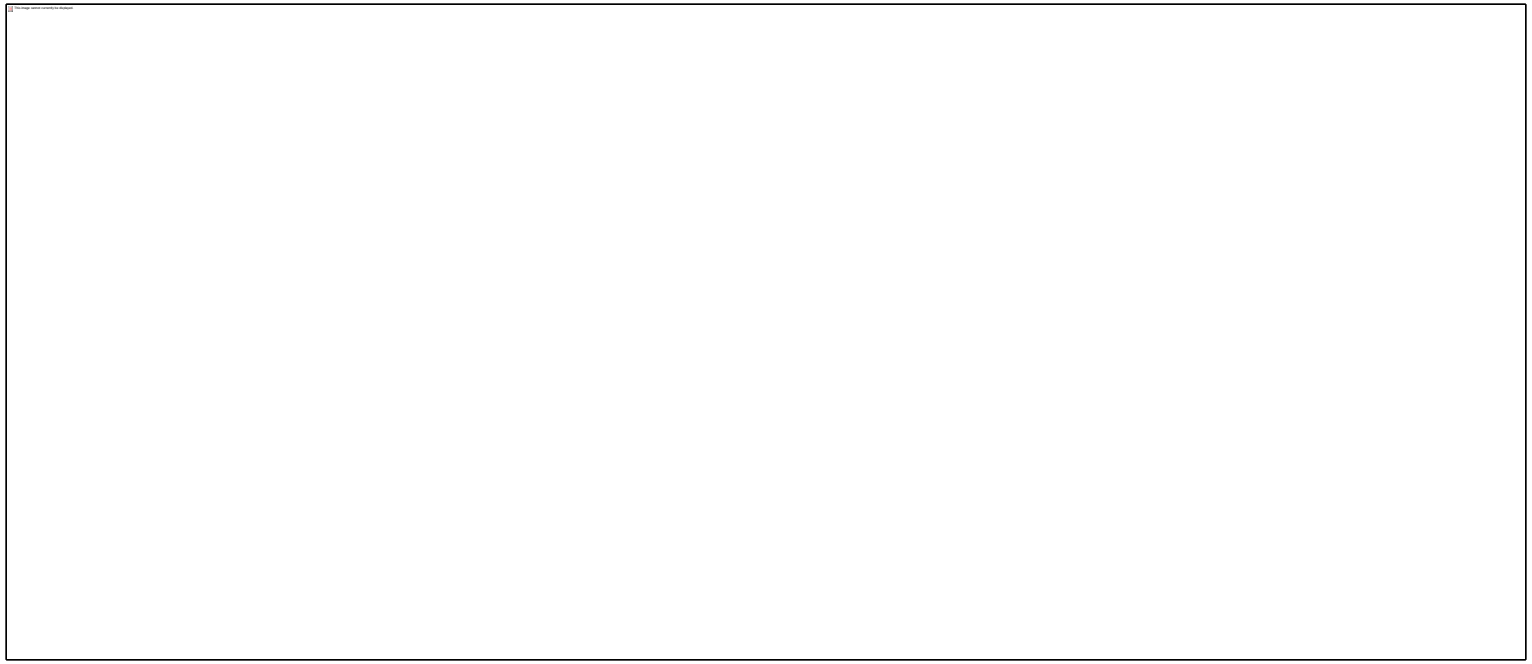
- Value of the element at index 3 has been assigned new value as 5, so element in the output list is changed from 4 to 5.



Deletion in List:

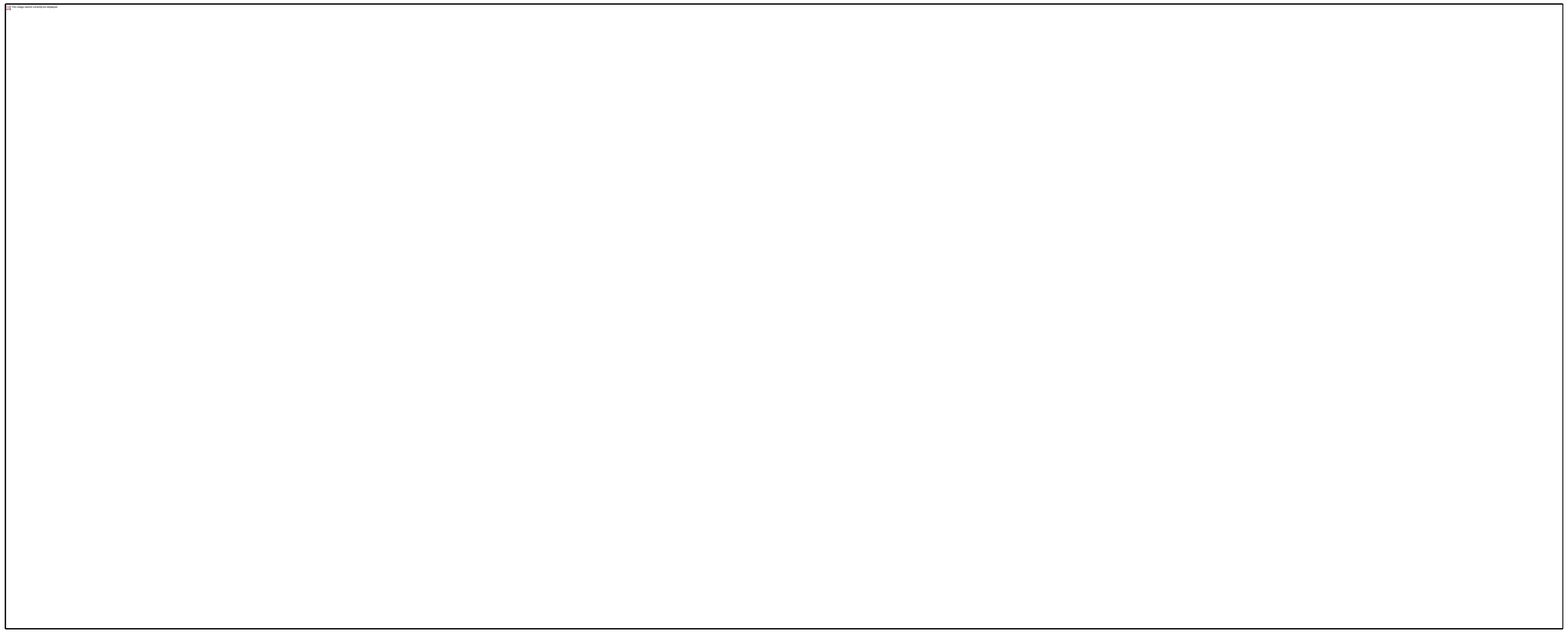
List elements can be deleted.

- Using del command If we know the position(index) of the element which is to be deleted.
- We can use the remove method by giving the specific element as given in the example If we know the position(index) of the element.



Deletion in List:

- list1 has been deleted and then we are trying to print it and its giving error because it does not exist now.



Built in Methods in List

Method	Remark	Example
len()	It calculates the length of the list or the number of elements in the list.	
max(list)	It returns the maximum element from the list	
min(list)	It returns a minimum element from the list	
list(seq)	It converts into any sequence into a list	

Contd..

Method	Remark	Example
pop()	It deletes the last element from the list Note – We can also pass index as argument in pop() to delete a specific index value.	
count()	It counts the occurrences of a particular element in the list	
sort()	Sort the elements of the list in ascending order	
reverse()	Sort the elements of the list in reverse order	

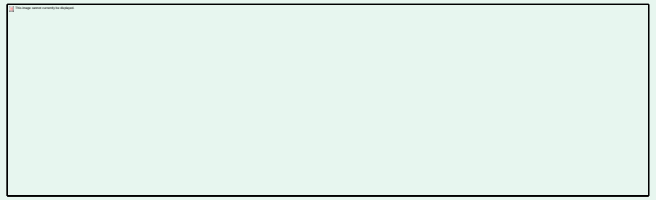
Operations on List

- Python supports variety of operations on the list

Operation	Remark	Example
Concatenation	Operation adds two list elements	
Repitition	It repeats the list specified number of times	

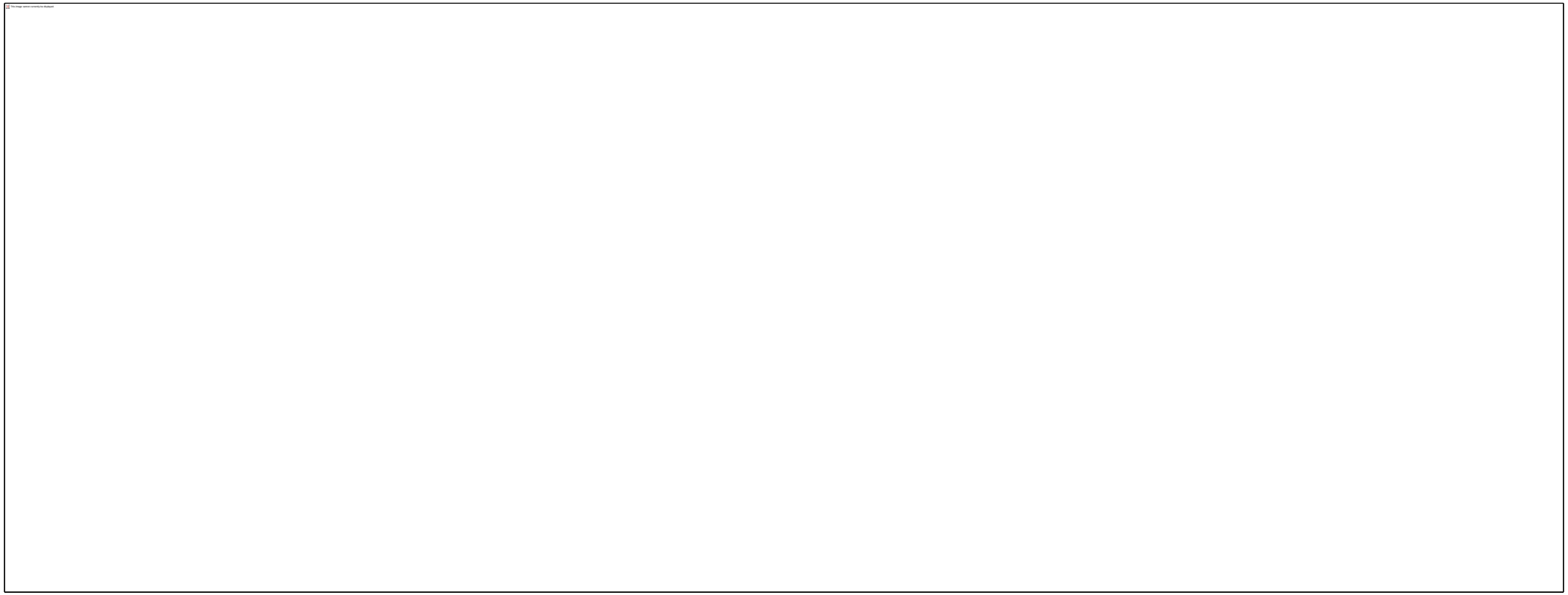
Contd..

- Python supports variety of operations on the list

Operation	Remark	Example
Membership	To check whether an element belongs to the list or not	
Membership not	I return true of an element that does not belong to the list	

Example

- Write Python Program to swap elements in the list.



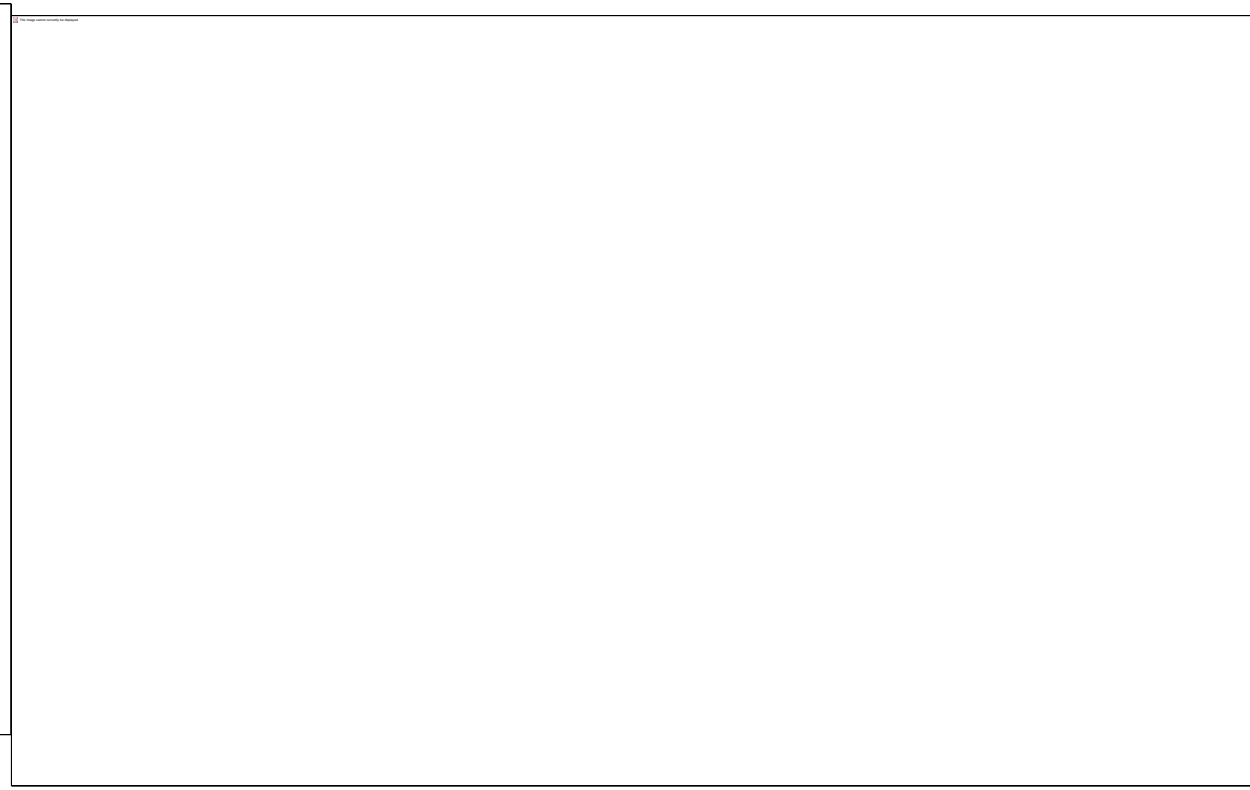
Practice Problems

- Take a list input from user having integer elements and calculate sum and average of the list.
- Take an input list and swap string elements of the list with empty string.

Loops with List

- While loop with list

- For Loop with list



Example

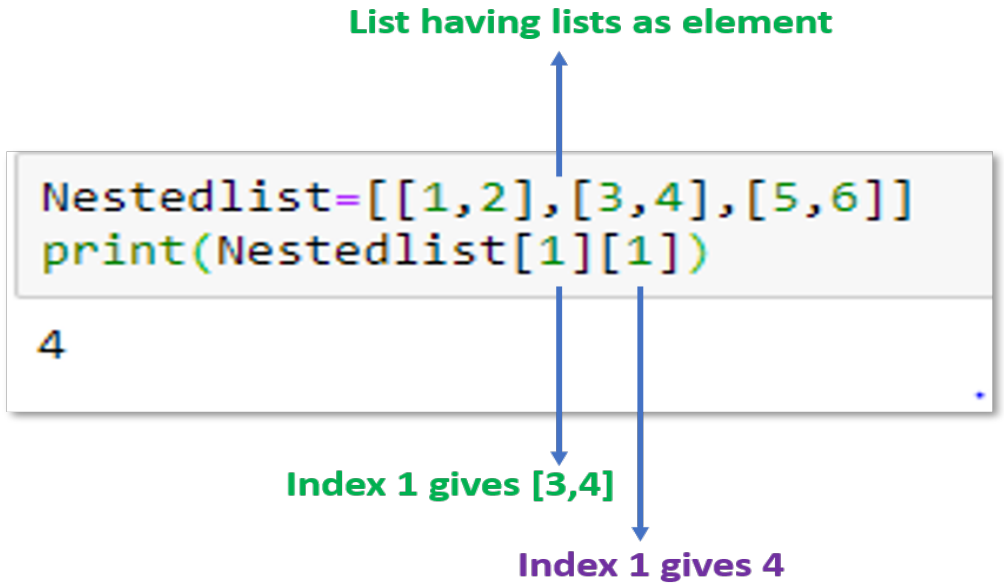
- Write a python program to print all positive number of a list.

Practice Exercise:

- Write a python program to remove duplicate from the list.
- Write a python program to count positive, negative and string type elements.

Nested List

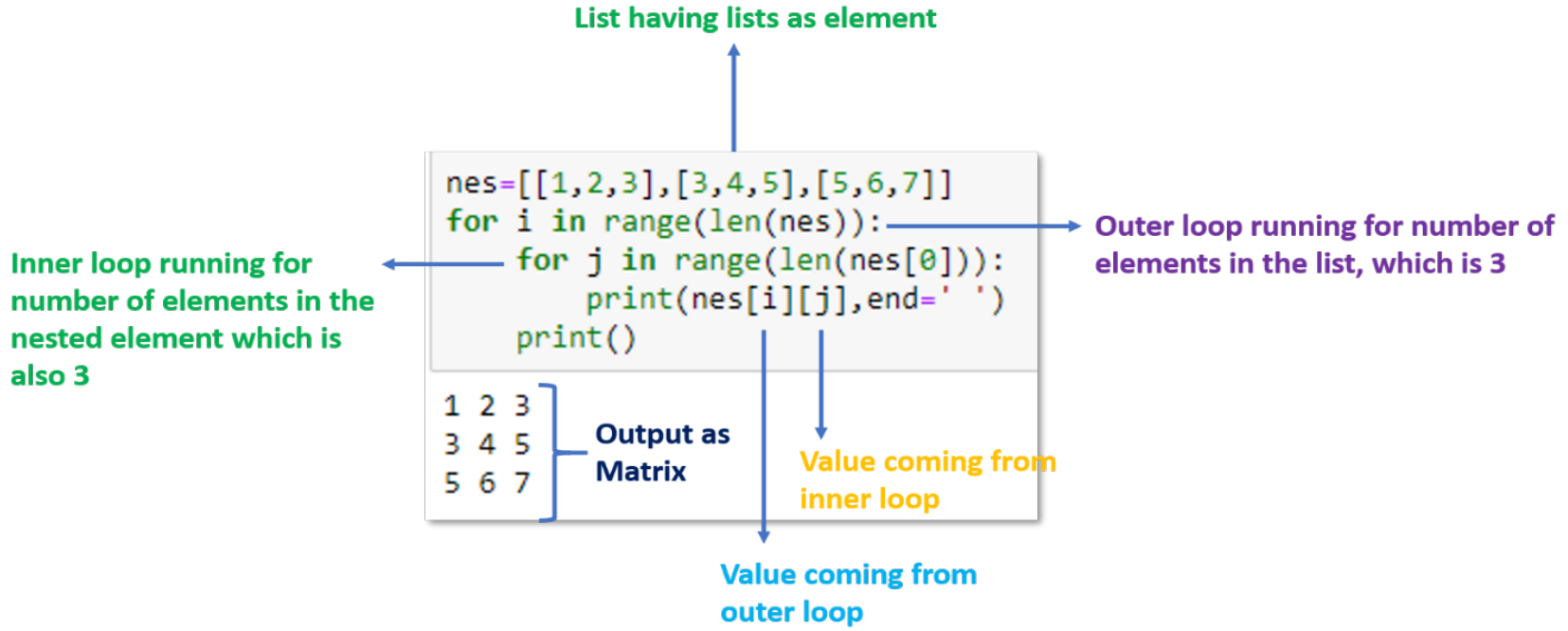
- When we have an element of a list in the form of the list itself, it is called Nested List.
- Elements of the nested list can be accessed using the 2-D indexing access method.



In this example, first [1] denotes [3,4] and second [1] represents 4, so it gives output as 4.

Nested List as a Matrix

- We can represent nested list as matrix also.
- For this we would use nested for loop.
- Outer loop would run for number of elements in the list and inner loop would consider individual element of the nested list as a list.



Example

- Create a flat list from a nested list.

```
nestedlist=[[1,2],[3,4],[5,6,7]]
newlist=[]
for i in range(len(nestedlist)):
    for j in range(len(nestedlist[i])):
        newlist.append(nestedlist[i][j])
print(newlist)
```

Given list

Empty list created for output

Nested for loop to access nested list elements

Printing of new list

```
[1, 2, 3, 4, 5, 6, 7]
```

Output

.

Practice Exercise:

- Using nested list print transpose, of a given matrix
- Print reverse order of a nested list

.

List Comprehensions

- Let's suppose we want to write a program to calculate powers of 3 for a given range.

- Traditional Programming

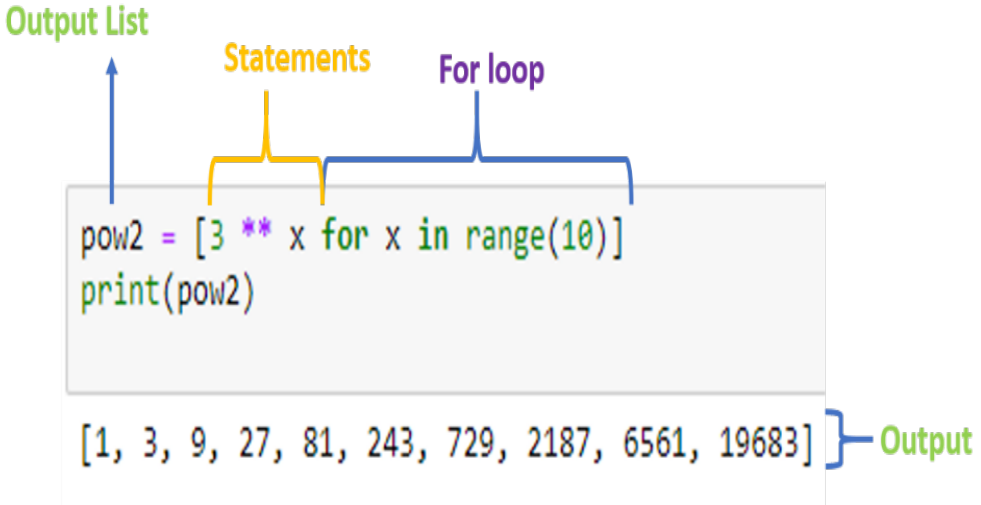
```
pow2 = []
for x in range(10):
    pow2.append( 3** x)
print(pow2)
```

```
[1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683]
```

.

- Python provide us writing iterative programs in much lesser lines called List comprehension.

- The same problem has been solved using list comprehension.

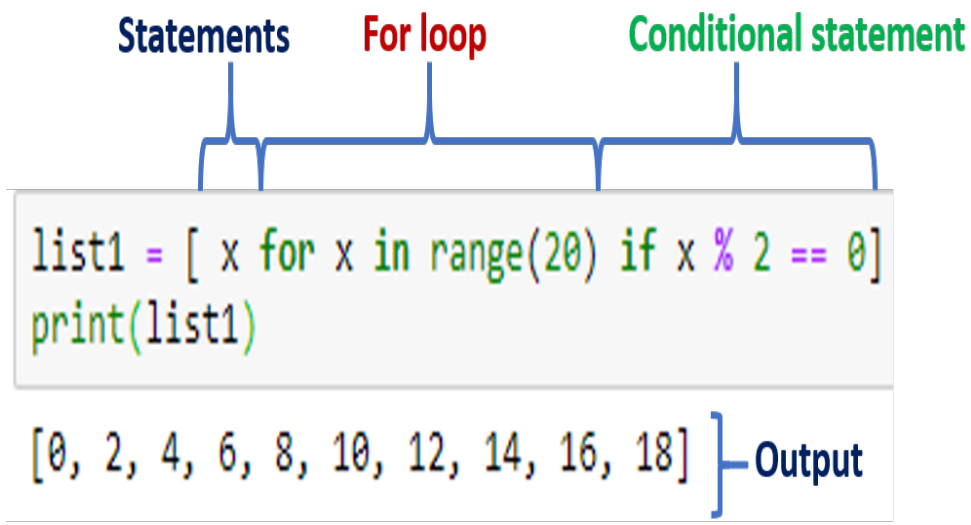


List Comprehension

Syntax:

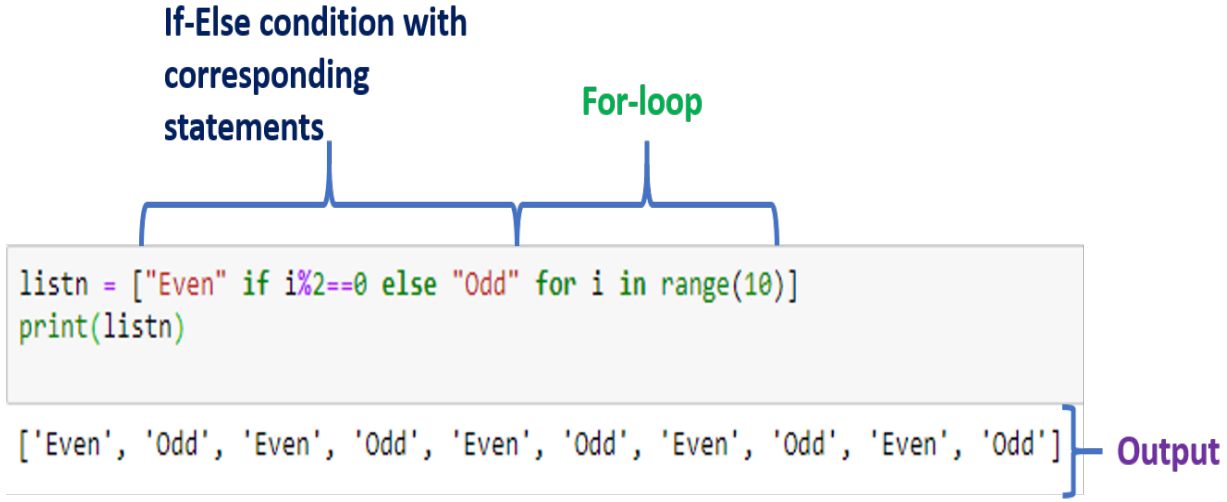
[*statements* for an item in list]

- We have the facility of adding conditionals in the list comprehension.



IF ELSE :

- We should notice one change that because in if-else case output is separated based on the condition.
- if-else and conditions has been written along with statements.



Can you answer these questions?

1. What will be the Output of the following code?

```
t = [p**+1 for p in range(6)]
```

A.[0,1,2,3,4,5]

B.[0,1,2,3,4,5,6]

C.[0,1,2,3]

D. None of the Above

.

Can you answer these questions?

2. What will be the Output of the following code?

```
B=[[3, 4, 5],[6,7,8],[9,10, 11]]
[B[i][len (B)-1-i] for i in range (len (B))]
```

A. [5,7,9]

B.[3,4,5]

C.[6,7,8]

D.[9,10,11]

Session Plan - Day 5

3.3 Tuple

- Creation
- Assessing
- Modification/Updation
- Built in Methods
- Operations
- Review Questions
- Practice Exercises

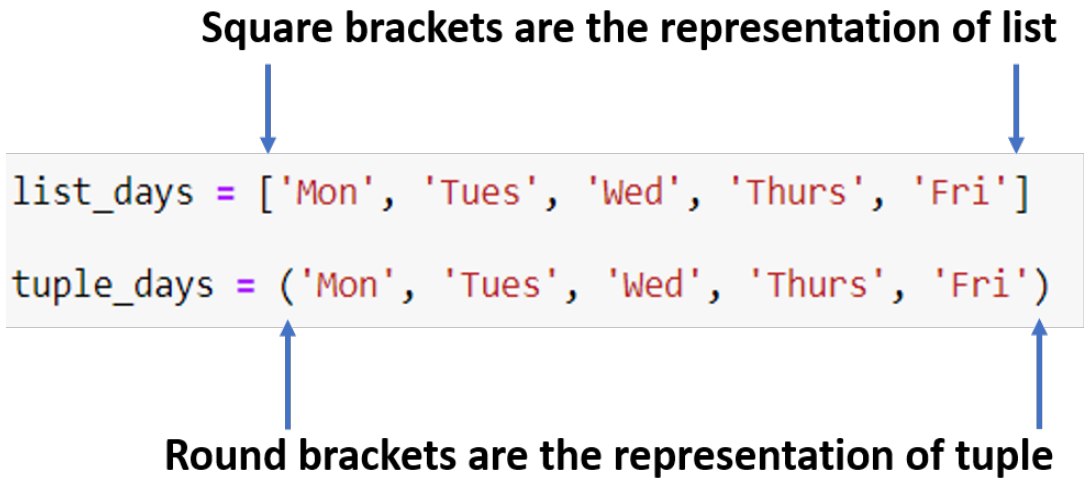
Session Plan - Day 5

3.3 Tuple

- Creation
- Assessing
- Modification/Updation
- Built in Methods
- Operations
- Review Questions
- Practice Exercises

Tuple

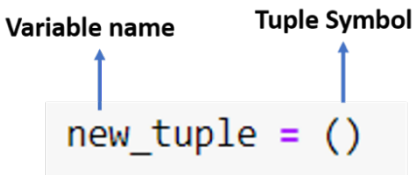
- Tuple is an immutable (unchangeable) ordered collection of elements of different data types.
- Generally, when we have data to process, it is not required to modify the data values.
- Take an example of week days, here the days are fixed. So, it is better to store the values in the data collection, where modification is not required.



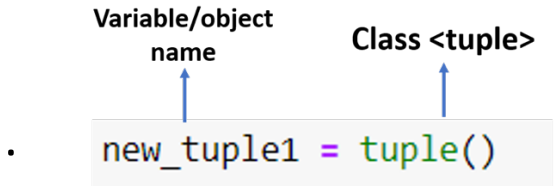
.

Creation of Tuple

Creation of Empty Tuple

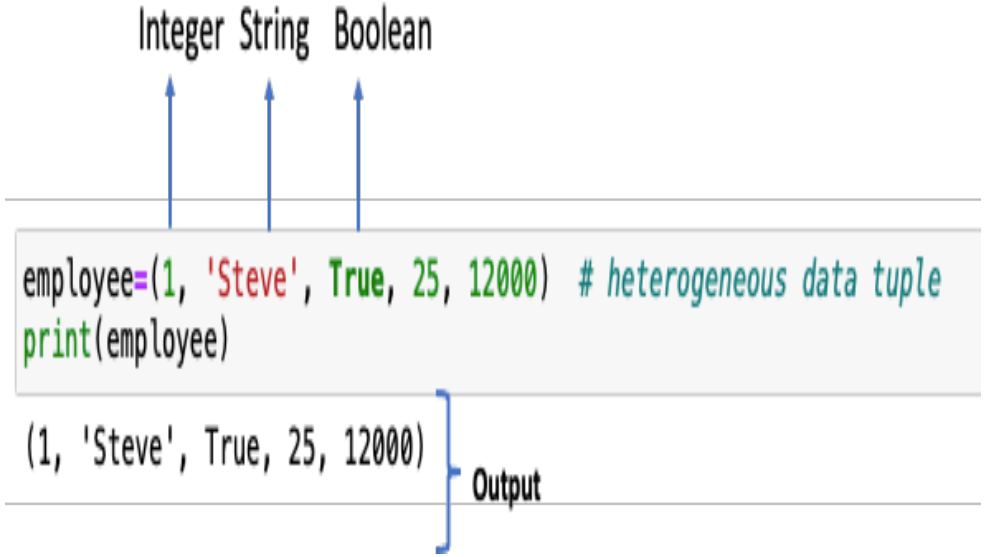


- In above example, the () round brackets are used to create the variable of tuple type.
- We can also call the class to create a tuple object as shown in example below.



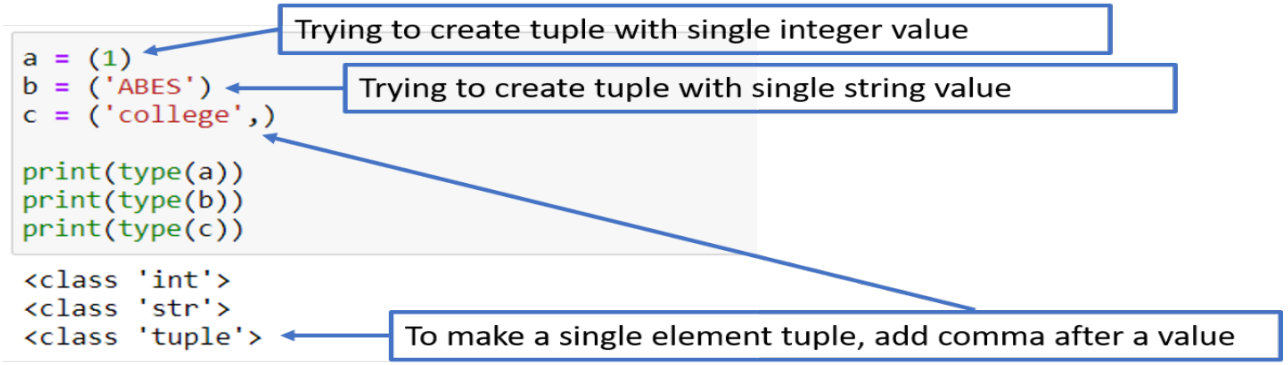
Creation of Non Empty Tuple

- Syntax of Creating Non Empty Tuple



Example

- Write a program in python to create one element in tuple



- If you want to create a tuple with single value, it is required to add a comma after the single value as shown in the above example **c=('college',)**.
- If the comma is not placed, then the single value **a= (1) or b=(ABES)** will be treated as an independent value instead of the data collection.

Packing and Unpacking of Tuple

- Tuple can also be created without using parenthesis; it is known as **packing.**
- Write a program to create tuple without parenthesis.

```
newtup4=3,4,5,"hello"
print(type(newtup4))
print(newtup4)
```

```
<class 'tuple'>
(3, 4, 5, 'hello')
```

In above example 1, **newtup4=3,4,5," hello"** creates a new tuple without parenthesis.

.

Unpacking of Tuple

- **Write a program to unpack elements in tuple**
- Unpacking is called when the multiple variables is assigned to a tuple; then these variables take corresponding values.

```
newtup4=3,4,5,"hello"   #packing
print(newtup4)
a,b,c,d=newtup4         #unpacking
print(a,b,c,d)
```

```
(3, 4, 5, 'hello')
3 4 5 hello
```

Practice Questions:

- Write a Python program to create the colon of a tuple.
- Write a Python program to unpack a tuple in several variables.

Accessing Elements in Tuple

- Tuple elements can be accessed using indexes, just like String and List .
- Each element in the tuple is accessed by the index in the square brackets [].
- An index starts with zero and ends with (number of elements - 1)

Example:

Write a program to access index 1 element from tuple.

```
newtup=("hello",2,3,23.45)
print(newtup[1])
```

Accessing the index one of tuple

```
2
```

Output

In above example 1: tuple ***newtup*** carries multiple types of data in it as “hello” is string, 2 is integer, and 23.45 is float. If we can fetch the index 1 element using **print(newtup[1]).**

Contd…

- Write a program to access index 0 element from tuple.

```
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')

print(tuple_days[-1])
```

Accessing the index -1 of tuple

```
Fri
```

Output

- Basically, the -1 index will return the last value of tuple.
- Indexes start from zero, and it goes up to a number of elements or say -1.
- Take another example to fetch the -1 index from tuple.
- last element is accessed by **print(tuple_days[-1]).**

.

Indexing in Tuple and Slicing

- Slicing in a tuple is like a list that extracts a part of the tuple from the original tuple.
- Write a program to access elements from 0 to 4 index in tuple.

```
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')

print(tuple_days[0:4])
('Mon', 'Tues', 'Wed', 'Thurs')
```

Accessing multiple indexes of tuple

Output

- Write a program to access elements from 1 to 3 index in tuple.

```
newtup=("hello",2,3,23.45)
print(newtup[1:3])
(2, 3)
```

Modification/Updating a Tuple

- If we want to change any of the index value in tuple, this is not allowed and throw an error.
- Tuple object does not support item assignment.
- Here, we are taking the same above example of days and showing the immutable characteristic of tuple

We want to change the value 'Mon' to 'Monday'

```
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')

tuple_days[0] = 'Monday'
```

Trying to change the **Value** of zero index in tuple

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-9c932d3b637c> in <module>
      1 tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')
      2
----> 3 tuple_days[0] = 'Monday'

TypeError: 'tuple' object does not support item assignment
```

.

Practice Exercises

- Write a Python program to get the 4th element and 4th element from last of a tuple.

- Write a Python program to check whether an element exists within a tuple.

.

Built in Methods in Tuple

Tuple has 2 Built in methods

Method name	Remark	Example

count()	It returns the occurrences of the value given	`newtup=("hello",2,3,23.45,2,3)`
		`print(newtup.count(2))`
		`2`

index()	It returns the index of the specified value	`newtup=("hello",2,3,23.45)`
		`print(newtup.index(2))`
		`1`

.

```
tup1=(2,3,4)
3 not in tup1
```

```
False
```

ations on Tuple

Operations	Remarks	Example

Concatenation	Add two tuples	`tup1=(2,3,4)`
		`tup2=(5,6,7)`
		`tup1+tup2`
		`(2, 3, 4, 5, 6, 7)`

Multiplication	Creates copies of the tuple	`tup1=(2,3,4)`
		`tup1*2`
		`(2, 3, 4, 2, 3, 4)`

Membership	To check whether an element belong to tuple or not	`tup1=(2,3,4)`
		`3 in tup1`
		`True`

Not Membership	Would return true if element does not belong to tuple.	`tup1=(2,3,4)`
		`3 not in tup1`
		`False`

Loops and Conditions on Tuples

- Tuple can be traversed using Loop
- Take the tuple named *tuple_days* and print all its elements

```
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')

for i in tuple_days:
    print(i)
```

Iteration on tuple

```
Mon
Tues
Wed
Thurs
Fri
```

Output

- **In this example** The elements are printed while iterating through for loop condition **for in tuple_days** and then we print the values of **i** in each iteration.

Loops and Conditions on Tuples

Example:

- Let have a look to the example where we are placing a condition to return the values having word length greater than 3.

```
tuple_days = ('Mon', 'Tues', 'Wed', 'Thurs', 'Fri')

for i in tuple_days:
    if len(i)>3:
        print(i)
```

Condition in Iteration on tuple

```
Tues
Thurs
```

Output

- **In above example ,** The elements are printed while iterating through for loop condition **for in tuple_days** and then we print the values of **i** in each iteration with the condition **if(len(i)>3)**

Comparison between List and Tuple

List	Tuple
Mutable Sequence	Immutable Sequence
Accession Slower than Tuple because of mutable property	Faster access of elements due to immutable property
It cannot be used as Dictionary keys	Can work as a key of the dictionary
Not suitable for application which needs write protection	It suits such scenarios where write protection is needed.

Practice Exercises:

Exercise: Write Python program to join two input tuples, if their first element is common.

```
tup1=tuple(input("enter first tuple").split())
tup2=tuple(input("enter second tuple").split())

if(tup1[0]==tup2[0]):
    print(tup1+tup2)
else:
    print("First element not same")
```

```
enter first tuple2 4 5 6
enter second tuple2 3
('2', '4', '5', '6', '2', '3')
```

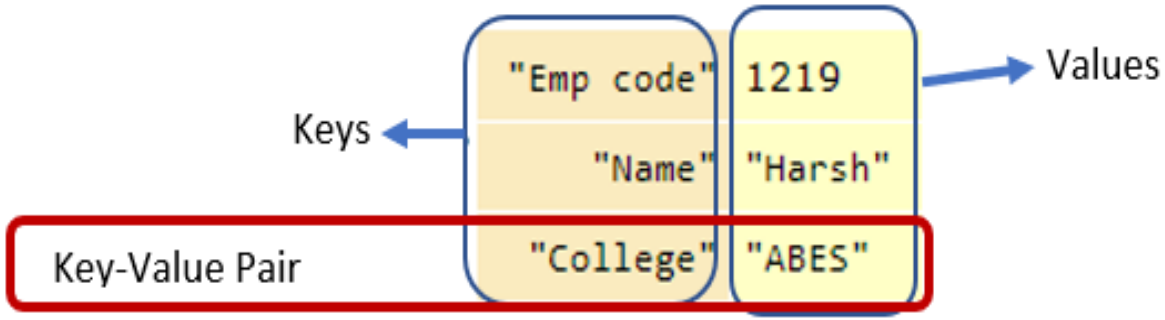
Session Plan - Day 6

3.4 Dictionaries

- Creation
- Assessing
- Modification/Updation
- Nested Dictionary
- Built in Methods
- Review Questions
- Practice Exercises

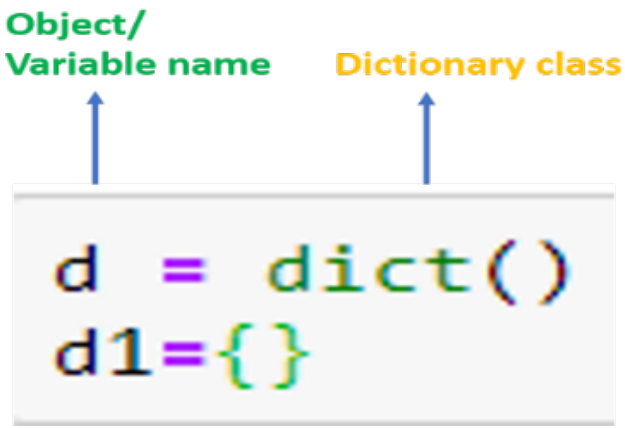
Dictionaries

- Dictionary is a unique data collection of Python which stores the key-value pair.
  - The user can add an element by giving a user-defined index called a **key**.
  - Each key and its corresponding value makes the key-value pair in dictionary.
  - This key-value pair is considered as one item in dictionary.



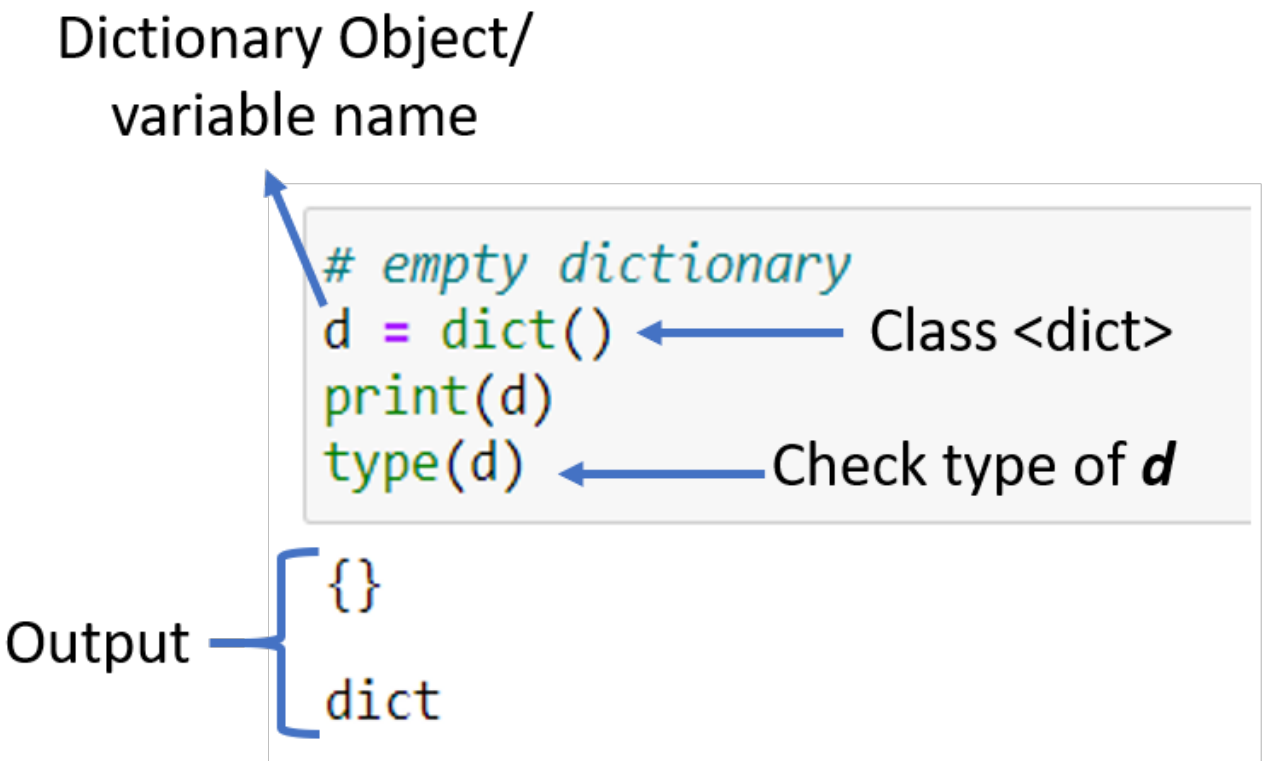
Introduction

- In dictionaries, the indexes are not by default. (such as 0 in string, list, tuple).
- **{} is the representation of Dictionary**
- Creation of Dictionary
  - An empty dictionary can be created as



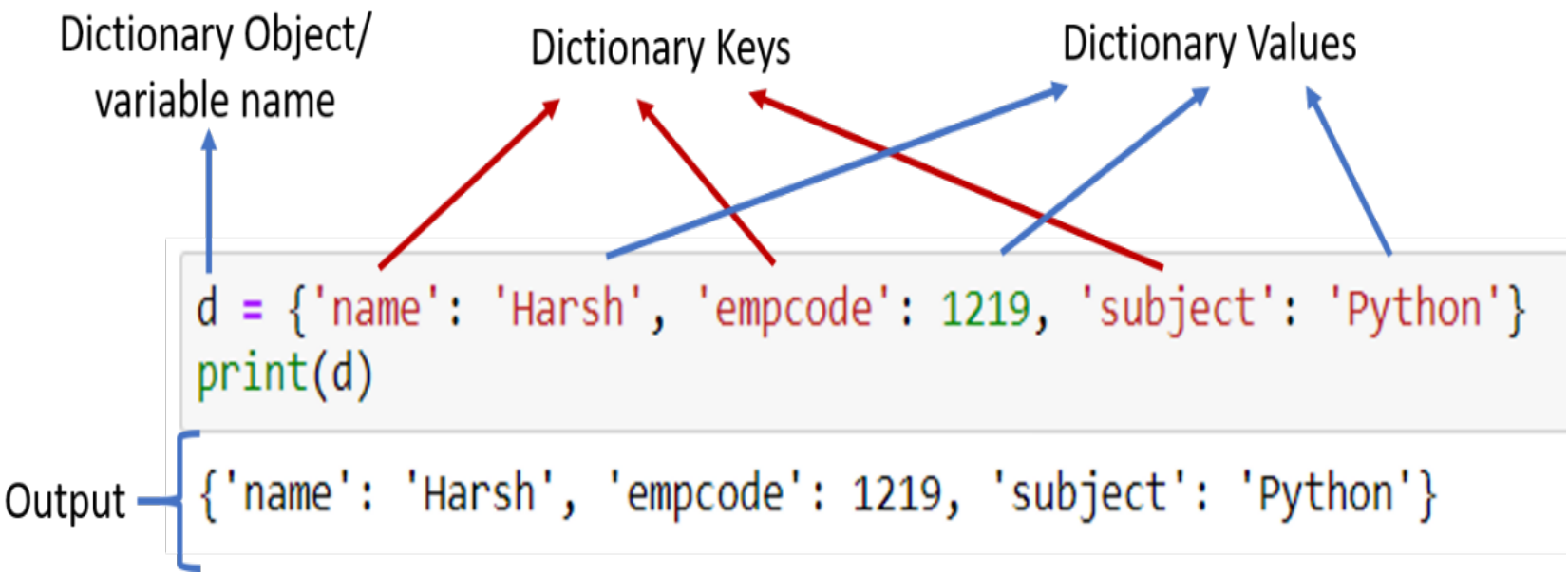
Example-1

- **Creating an empty dictionary**



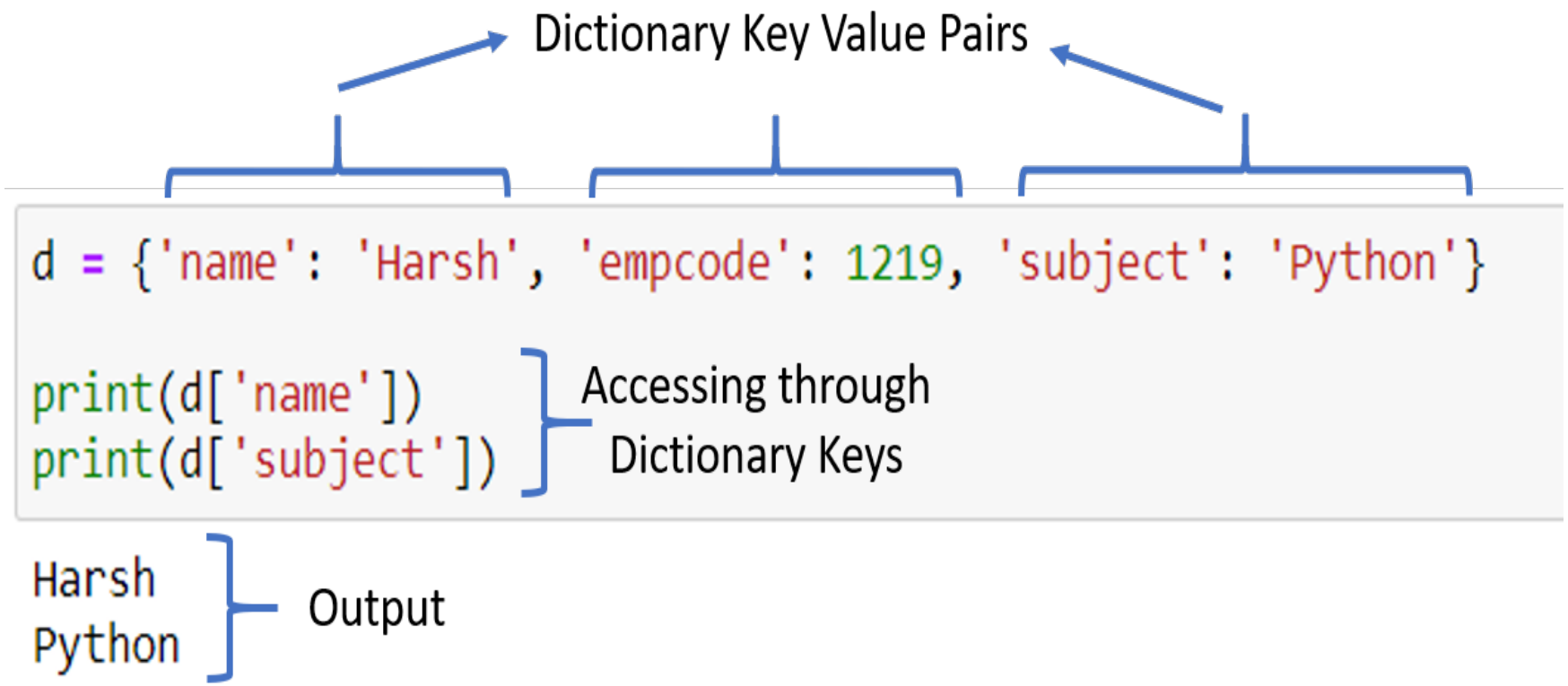
Example-2

- Creating a non-empty dictionary



Accessing of Dictionary

- In dictionary, the items are accessed by the keys.



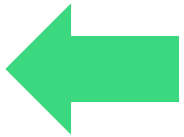
Can you answer these questions?

1. Output of following code?

```
dict1={'opt1':'Python','opt2':'Java','opt3':'C','opt4':'C++'}
print(dict1[1])
```

A) ‘Java’

B) ‘Python’

C) KeyError 

D) None of above

Can you answer these questions?

2. Output of following code?

```
dict1={1:2,2:4,4:8,8:16,16:32}
print(len(dict1))
```

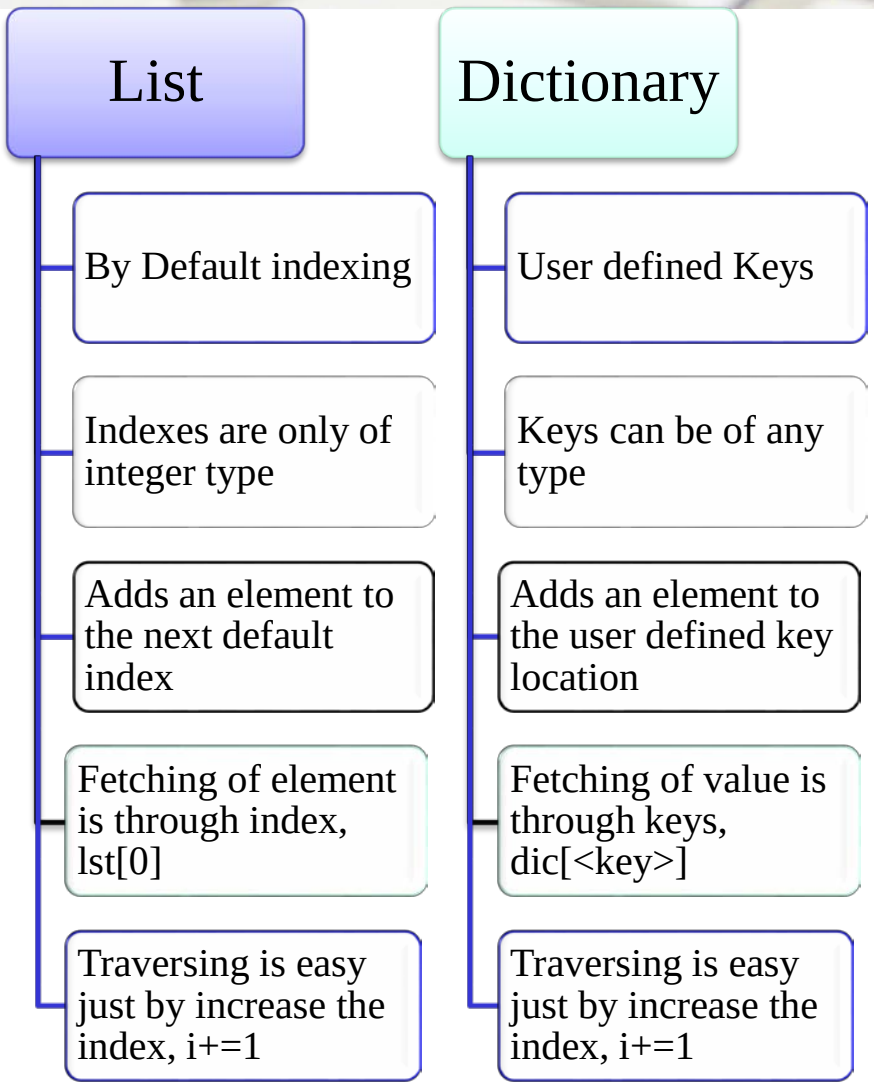
A) 5

B) 10

C) 15

D) None of above

Accessing of Dictionary



Modification in a Dictionary

Dictionary Object/
variable name

```
d = {'name': 'Harsh', 'empcode': 1219, 'subject': 'Python'}
print(d['name'])

# change the name to full name
d['name'] = 'Harsh Khatter'
print(d['name'])
```

← Update the **Value** using **Key**

```
Harsh
Harsh Khatter
```

← Value before modification
← Value after modification

Output

Can you answer these questions?

1. Output of following code?

A) {1:’Store’,2:’Kitchen’}

B) {2:’Store’,1:’Kitchen’}

C)KeyError

D) None of above

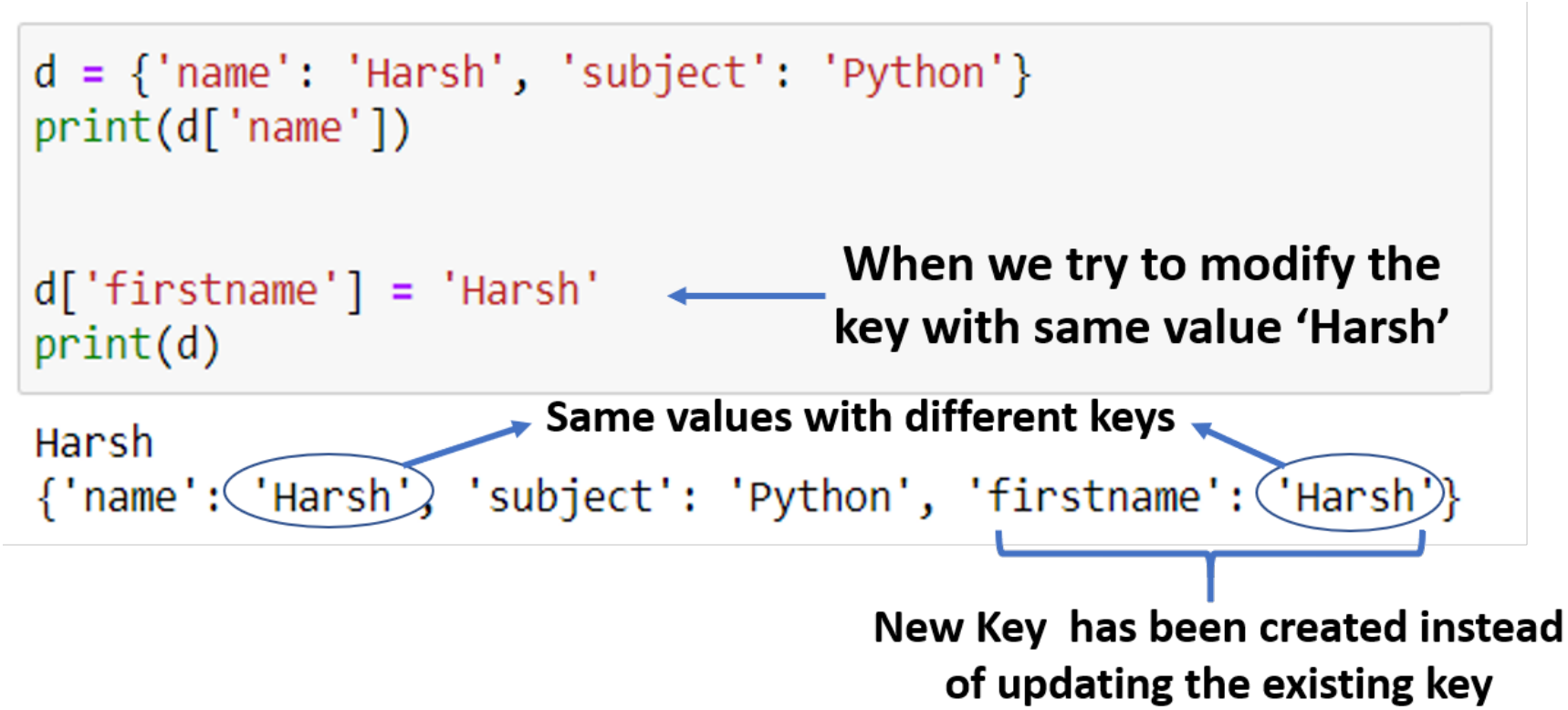
```
room={1:'Store',2:'Kitchen'}
temp=room[1]
room[1]=room[2]
room[2]=temp
print(room)
```

Can you answer these questions?

2. Is it possible to change key in dictionary?

A) True

B) False



Nested Dictionary

- As we have the option of a nested list, similarly, dictionaries can consist of data collections.

```
# Creating a Nested Dictionary
dt = {1: 'Hello', 2: 'to',
        3:{'A' : 'Welcome', 'B' : 'To', 'C' : 2021}}

print(dt)
```

```
{1: 'Hello', 2: 'to', 3: {'A': 'Welcome', 'B': 'To', 'C': 2021}}
```

Session Plan - Day 7

3.4 Dictionary

- Built in Methods in Dictionary
- Loops in Dictionary
- Review Questions
- Practice Exercises

Built-in Methods in Dictionary

Method	Description
copy()	Copying a dictionary to another dictionary
fromkeys()	Create a new dictionary with key in a data sequence list/tuple with value
get()	If the key is present in the dictionary, its value is returned. If the key is not present in a dictionary, then the default value will be shown as output instead of a KeyError.
items()	this will return the key-value pair as an output.
keys()	this will return only the keys as an output.
values()	this will return only the dictionary values as an output.
update()	this adds the one dictionary with another dictionary.
pop()	The pop() method takes an argument as the dictionary key, and deletes the item from the dictionary.
popitem()	The popitem() method retrieves and removes the last key/value pair inserted into the dictionary.

copy()

- copy() method provide a fresh copy with different memory location.

```
floor1={1:'Store',2:'Kitchen'}
floor2=floor1
floor3=floor1.copy()
print(floor1)
print(floor2)
print(floor3)
print(id(floor1)==id(floor2))
print(id(floor1)==id(floor3))
```

Output

```
{1: 'Store', 2: 'Kitchen'}
{1: 'Store', 2: 'Kitchen'}
{1: 'Store', 2: 'Kitchen'}
True
False
```

fromkey()

```
key=(1,2,3,4)
value='xyz'
sqr={}
sqr.fromkeys(key,value)
```

Output

```
{1: 'xyz', 2: 'xyz', 3: 'xyz', 4: 'xyz'}
```

get()

```
std={'name':'Bob','age':24,'phone':[12345,23456]}
print(std.get('name'))
print(std.get('gender')) # default value None
print(std.get('gender','M')) # set default value to 'M'
```

Output

```
Bob
None
M
```

items(),keys(),values()

```
std={'name':'Bob','age':24,'phone':[12345,23456]}
print(std.keys())
print(std.values())
print(std.items())
```

Output

```
dict_keys(['name', 'age', 'phone'])
dict_values(['Bob', 24, [12345, 23456]])
dict_items([('name', 'Bob'), ('age', 24), ('phone', [12345, 23456])])
```

update()

```
A={1:1,2:4,4:8,6:36}
B={3:9,5:25}
A.update(B)
print(A)
print(B)
```

Output

```
{1: 1, 2: 4, 4: 8, 6: 36, 3: 9, 5: 25}
{3: 9, 5: 25}
```

pop(),popitem()

```
A={1:1,2:4,4:8,6:36}
A.pop(6)
print(A)
```

```
{1: 1, 2: 4, 4: 8}
```

Output

```
A={1:1,2:4,4:8,6:36,8:64}
A.popitem()
print(A)
```

```
{1: 1, 2: 4, 4: 8, 6: 36}
```

Loops and conditions on dictionaries

- Loops and conditions can easily apply to dictionaries.

  dict1={1:1,2:4,4:8,6:36}

for key in dict1.keys(): print(key)	for val in dict1.values(): print(val)	for item in dict1.items(): print(item)	for i,j in dict1.items(): print(i,":",j)
OUTPUT			
1 2 4 6	1 4 8 36	(1,1) (2,4) (4,8) (6,36)	1:1 2:4 4:8 6:36

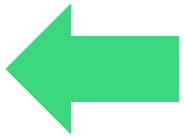
Can you answer these questions?

1. Output of following code?

```
newdict={}
for i in range(1,6):
    newdict[i]=i*i
print(newdict)
```

A) {1:1,2:4,3:9,4:16,5:25,6:36}

B) {1:1,2:4,3:9,4:16,5:25}



C) [1,4,9,16,25]

D) Error

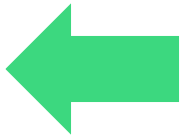
Can you answer these questions?

1. Output of following code?

```
room={1:'Store',2:'Kitchen'}
temp=room[1]
room[1]=room[2]
room[2]=temp
print(room)
```

A) {1:'Store',2:'Kitchen'}

B) {2:'Store',1:'Kitchen'}



C)KeyError

D) None of above

Session Plan - Day 8

3.5 Set

- Creation
- Assessing
- Modification
- Built in methods
- Operators
- Loops
- Frozen Set
- Review Questions

Set

- A set is another data collection data types in python, which stores unique elements in an unordered way.
- Every element in a set is unique and **immutable(unchangeable)**, i.e. no duplicate values should be there, and ***the values can't be changed.***

Creation of empty Set

```
# Creation of an Empty set using set()
s1 = set()
print(s1)
print(type(s1))
```

```
set()
<class 'set'>
```

Output

Creation of non-empty Set

Elements as Integer

```
s1 = {8,3,15,31,8,10,31,9,2,4}
print(s1)
```

```
{2, 3, 4, 8, 9, 10, 15, 31}
```

Output

```
set1 = {"mango", "banana", "orange"}
count=len(set1)
print(count)
```

len() method finds the number of items in set.

```
3
```

Output

String Integer Boolean Float

```
myset = {"name", 2, "age", True , 12.8 }
print(myset)
```

```
{True, 2, 12.8, 'name', 'age'}
```

Output

Creation of non-empty Set

List of Integers

```
set_int = set([1, 2, 3, 4, 5, 6, 7, 7, 7])
print(set_int)
```

```
{1, 2, 3, 4, 5, 6, 7}
```

Output

String

```
set_str = set("ABESEC")
print(set_str)
```

```
{'C', 'B', 'E', 'A', 'S'}
```

Output

Elements as String

```
set_of_fruits = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(set_of_fruits)
```

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

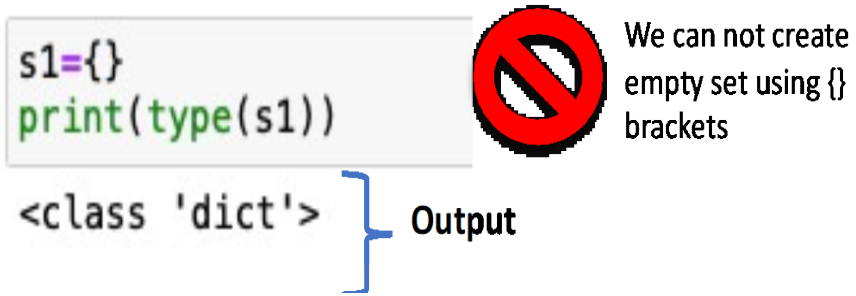
Output

Can you answer these questions?

2. Is it possible to create an empty set using like s1={} ?

A) True

B) False



Accessing set

- Python set's item cannot be accessed using indexes.

```
set_access = {7, 6, 1, 4, 3, 4, 1, 7}
print(set_access[2])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-68579cc997ed> in <module>
      1 set_access = {7, 6, 1, 4, 3, 4, 1, 7}
----> 2 print(set_access[2])

TypeError: 'set' object is not subscriptable
```

Built-in Methods in Dictionary

Method	Description
copy()	Copying a set to another set
clear()	Removes all element from set
add()	Adding a new item in set
update()	If the key is present in the dictionary, its value is returned. If the key is not present in a dictionary, then the default value will be shown as output instead of a Key Error.
remove()	to remove the specified element from the given set.
pop()	used to removes a random element from the set and returns the popped (removed) elements.
remove ()	used to remove the specified element from the given set.
discard ()	used to remove the specified item from the given input set. ***the remove() method will give an error if the specified item does not exist but this method will not.***

copy()

- copy() method provide a fresh copy with different memory location.

```
set1 = {"mango", "banana", "orange"}
set2 = set1.copy()
print(set2)
```

copy() method to copy all **set 1** elements to **set2**

```
{'orange', 'banana', 'mango'}
```

Output

clear()

```
set1 = {"mango", "banana", "orange"}
set1.clear()
print(set1)
```

clear() method to remove all elements

```
set()
```

Output

add()

```
# initializing a set
set_add = {21, 23}
print(set_add)

# adding an item
set_add.add(22)
print(set_add)
```

add() method to add new element

```
{21, 23}
{21, 22, 23}
```

Output

update()

```
# initializing a set
set1 = {21, 23}
print(set1)

# adding multiple elements
set1.update([22, 13, 14])
print(set1)

# adding list and set
set1.update([14, 15], {11, 16, 18})
print(set1)
```

update() method to add new elements

update() method to add new elements as list and set

```
{21, 23}
{13, 14, 21, 22, 23}
{11, 13, 14, 15, 16, 18, 21, 22, 23}
```

Output

remove()

```
set1 = {"mango", "banana", "orange"}
set1.remove("mango")
print(set1)
```

remove() method removes any specific element

```
{'orange', 'banana'}
```

Output

pop()

```
set1 = {"mango", "banana", "orange"}
set1.pop()
print(set1)
```

pop() method randomly removes any element

```
{'banana', 'mango'}
```

Output

update()

```
# initializing a set
set1 = {21, 23}
print(set1)

# adding multiple elements
set1.update([22, 13, 14])
print(set1)

# adding list and set
set1.update([14, 15], {11, 16, 18})
print(set1)
```

update() method to add new elements

update() method to add new elements as list and set

```
{21, 23}
{13, 14, 21, 22, 23}
{11, 13, 14, 15, 16, 18, 21, 22, 23}
```

Output

remove()

```
set1 = {"mango", "banana", "orange"}
set1.remove("mango")
print(set1)
```

remove() method removes any specific element

```
{'orange', 'banana'}
```

Output

discard()

```
set1 = {"mango", "banana", "orange"}
set1.discard("mango")
print(set1)
```

discard() method removes any specific element

```
{'orange', 'banana'}
```

Output

Operators

Set Operation	Description	Operator	Method
Union	All unique elements in set1 and set2	|	union()
Intersection	Elements present in set1 and set2	&	intersection()
Difference	Elements that are present in one set, but not the other	-	difference()
Symmetric Difference	Elements present in one set or the other, but not both	^	symmetric_difference()
Disjoint	True if the two sets have no elements in common	None	isdisjoint()
Subset	True if one set is a subset of the other (that is, all elements of set2 are also in set1)	<=	issubset()
Proper Subset	True if one set is a subset of the other, but set2 and set1 cannot be identical	<	None
Superset	True if one set is a superset of the other (that is, set1 contains all elements of set2)	>=	issuperset()
Proper Superset	True if one set is a superset of the other, but set1 and set2 cannot be identical	>	None

Union

```
set1 = {'abc', 'pqr', 'xyz'}
set2 = {'a', 'b'}
#union
print(set1 | set2)
```

| operator to join two sets

```
{'a', 'abc', 'b', 'pqr', 'xyz'}
```

Output

```
set1 = {'abc', 'pqr', 'xyz'}
set2 = {'a', 'b'}
#union
s_union=set1.union(set2)
print(s_union)
```

Union method to join two sets

```
{'b', 'a', 'xyz', 'abc', 'pqr'}
```

Output

Intersection

```
set1 = {'abc', 'pqr', 'xyz'}
set2 = {'a', 'b', 'abc'}
#intersection
print(set1 & set2)
```

⟸ & operator to intersect two sets

```
{'abc'}
```

Output

Intersection

```
set1 = {'abc', 'pqr', 'xyz'}
set2 = {'a', 'b', 'abc'}
#intersection
print(set1 & set2)
```

& operator to intersect two sets

```
{'abc'}
```

Output

```
set1 = {'abc', 'pqr', 'xyz'}
set2 = {'a', 'b', 'abc'}
#intersection
s_intersection=set1.intersection(set2)
print(s_intersection)
```

Intersection method to intersect two sets

```
{'abc'}
```

Output

Set Difference

```
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
print(set1 - set2)
```

- operator to find difference among two sets

```
{1, 2, 3}
```

Output

```
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
diff_set = set1.difference(set2)
print(diff_set)
```

Difference() method to find difference among two sets

```
{1, 2, 3}
```

Output

Symmetric Difference

```
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
print(set1 ^ set2)
```

^ operator to find symmetric difference among two sets

```
{1, 2, 3, 6, 7, 8}
```

Output

```
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}
symm_diff = set1.symmetric_difference(set2)
print(symm_diff)
```

Method to find symmetric difference among two sets

```
{1, 2, 3, 6, 7, 8}
```

Output

Loops with set

```
# Creating a set using string
set1 = {"Mango","Banana","Orange"}

# Iterating using for loop
for i in set1:
    print(i)
```

Iteration using for loop

```
Banana
Mango
Orange
```

Output

Frozen Set

A frozen set is a special category of the set which is unchangeable once created

```
# Frozen Set
set1 = {"mango", "banana", "orange"}
set2 = frozenset(set1)

print(type(set1))
print(type(set2))
```

frozenset() method

```
<class 'set'>
<class 'frozenset'>
```

Output

Frozen Set

```
set1 = {"Mango", "Banana", "Orange"}
set2 = frozenset(set1)
#add new element to frozen set
new="Grapes"
print(set2.add(new))
```

Trying to add new element in frozen set using **add()** method

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-36-a4b367bff942> in <module>
      3 #add new element to frozen set
      4 new="Grapes"
----> 5 print(set2.add(new))

AttributeError: 'frozenset' object has no attribute 'add'
```

Output

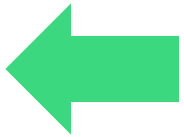
Summary

List	Tuple	Set	Dictionary
List is a collection of values that is ordered.	*Tuple is ordered and unchangeable collection of values*	*Set stores the unique values as data collection*	*Dictionary is a collection of key-value pairs*
Represented by [ ]	*Represented by ( )*	*Represented by { }*	*Represented by { }*
Duplicate elements allowed	*Duplicate elements allowed*	*Duplicate elements not allowed*	*Duplicate keys not allowed, in dictionary values allowed duplicate*
Values can be of any type	*Values can be of any type*	*Values can be of any type*	*Keys are immutable type, and value can be of any type*
Example: *[1, 2, 3, 4]*	*Example:* *(1, 2, 3, 4)*	*Example:* *{1, 2, 3, 4}*	*Example:* *{1:1, 2:2, 3:3, 4:4}*
List is mutable	*Tuple is immutable*	*Set is mutable*	*Dictionary is mutable*
List is ordered	*Tuple is ordered*	*Set is unordered*	*Dictionary is insertion ordered*
Creating an empty list *l=list()* *l = []*	*Creating an empty Tuple* *t = tuple ()* *t=( )*	*Creating a set* *s=set ()*	*Creating an empty dictionary* *d=dict( )* *d={}*

Can you answer these questions?

1. Which of the following Python code will create a set?
   (i) set1=set((0,9,0))
   (ii) set1=set([0,2,9])
   (iii) set1={}

A) iii
B) i and ii
C) ii and iii
D) All of Above



Can you answer these questions?

2. What is the output of following code?

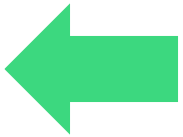
```
set1=set((0,9,0))
print(set1)
```

A) {0,0,9}

B) {0,9}

C) {9}

D) Error



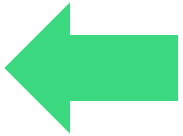
Can you answer these questions?

3. What is the output of following python code?

```
set1={1,2,3}
set1.add(4)
set1.add(4)
print(set1)
```

A) {1,2,3}

B) {1,2,3,4}



C) {1,2,3,4,4}

D) Error