(9) Strings

String special operators:

a = 'Hello'
b = 'Python'

a + b = HelloPython

a * 2 = HelloHello
a[1] = e
a[1:4] = ell
H in a = True
M not in a = True

~~String~~

Built in String methods to manipulate strings:

capitalize() — Capitalizes first letter of string.

count(str, beg=0, end=len(string))

— Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.

lower() — Converts all uppercase letters in string to lowercase

upper() — Converts lowercase letters in string to uppercase.

Built-in functions with Strings

len(string) — returns the length of the string.

max(string) — returns the max alphabetical character from the string str.

min(string) — returns the min alphabetical character from the string str.

String Indexing :

0	1	2	3	4	5	6	7	8	9	10	11
H	E	L	L	O		P	Y	T	H	O	N
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Using format() method

```
str = "Welcome to {}"
print(str.format("Tutorialspoint"))
```

Eg 1 Program to find number of vowels in a given string.

```
mystr = "All animals are equal. Some are
        more equal"
vowels = "aeiou"
count = 0
for x in mystr:
    if x.lower() in vowels: count += 1
print("number of vowels:", count)
```

Eg 2 Python program to drop all digits from a string.

```
digits = [str(x) for x in range(10)]
mystr = 'Hel2llo, Pyt0th55on!'
chars = []
for x in mystr:
    if x not in digits:
        chars.append(x)
newstr = ''.join(chars)
print(newstr)
```

classmate
Date
Page

Programs:

1. Program to sort characters in a string.
2. Prog. to write no. of words in a string.
3. Prog. to list unique characters with their count in a string.

Exercise:

1. Python Program that accepts a string and calculates the number of digits and letters.
2. Write a Python program to get a substring from a given string starting from the 3rd character to the 5th character.

Eg: Program to delete character of a string

```
String 1 = "Hello, I'm a geek"
print("Initial String:")
print(String 1)

print("Deleting character at 2nd Index:")
del String 1[2]
print(String 1)
```

Eg:

eg. 1 string = "geeksforgeeks"

```
print(len(string))
```

eg. 2

```
string = "geeks"
print(max(string))
```

output:

s

eg. 3

```
string = "raj"
print(max(string))
```

output:

r

(3) Searching :

(1) word = 'find me if you can'
print(word.find('me'))

5

(2) word = 'geeks for geeks'
print(word.find('ge', 2))

substring is searched in 'eks for geeks'

Output: 10

print(word.find('geeks', 2))

Output: -1

(3) print(word.find('g', 4, 10))

Output: -1 (10 is exclusive search happens from 4 to ending before index 10)

4) string = "geeks for geeks"

new_string = string.center(24, '*')

print("after padding string is:", new_string)

Padding Calculation:

Total padding needed: 24 - 15
= 9 characters

Left padding: 9//2 = 4 stars

Right padding: 9 - 4 = 5 stars.

Output:

After padding string is: ****geeks for geeks*****

5) string = 'random'

print("index of 'and' in string:",
string.index('and'))

output

Index of 'and' in string: 1

(6) test_string = "1234gfg4321"

print(test_string.index("gfg", 4, 8))

Output: 4

(7) string = "Convert ALL to lOWERCASE"

print(string.lower())

output: convert all to lowercase

(8) string = "Hello World"

new_string = string.replace("Hello", "good Bye")

print(new_string)

(2)

String functions:

1. len(string)
2. max(string)
3. min(string)
4. string.upper()
5. string.lower()
6. string.find()
7. string.replace()
8. string.capitalize()

→ string = "Hello"
print(len(string))

→ print(max(string))

→ print(min(string))

→ result = string.upper()

→ result1 = string.lower()

→ string.find('l')

→ string.replace('l', 'j')

→ string.capitalize()
Hello

Introduction to Lists:

~~List~~

List is one of the built-in data types in Python. A python is a sequence of comma separated items, enclosed in square brackets [].

The items in a Python list need not be of the same data type.

list1 = ["Rohan", "Physics", 21, 89.75]

List is an ordered collection of items. Each item in a list has a unique position index, starting from 0.

A Python list is mutable.

① Accessing Values in List:

list1 = ['Physics', 'Chemistry', 1997, 2000]

print(list1[0])

output: Physics

(2) Updating List :

```
list = ['Physics', 'chemistry', 1997, 2000]
print(list[2])
list[2] = 2004
print(list[2])
```

Output: 1997
2004

replaces the element at 2

(3) ~~list~~

```
print(list[1:4])
```

Output: [chemistry, 1997, 2000]

(4) Deleting :

① del list[2]

(2) list.remove(1997)

Python List Operations :

	Results
Concatenation → [1,2,3] + [4,5,6]	[1,2,3,4,5,6]
Repetition → ['Hi'] * 4	['Hi', 'Hi', 'Hi', 'Hi']
→ 3 in [1,2,3]	True

Indexing , Slicing

L = ['spam', 'Spam', 'SPAM']

Python Expression	Result
L[2]	SPAM
L[-2]	Spam
L[1:]	['Spam', 'SPAM']

Python List Methods

① list.append(obj)

```
alist = ['123', 'xyz', 'zara']
alist.append('2009')
print("Updated list :", alist)
```

Output: Updated List: ['123', 'xyz', 'zara', 2009]

append() adds new objects to a list. This method accepts an object as an argument and inserts it at the end of the existing list.

Using this method, a single object is added at the end of the list.

```
→ list = ['Mon', 'Tue', 'Wed']
  print("Existing list", list)
→ list.append('Thu')
  print("Appended one element:", list)
→ list.append(['Fri', 'Sat'])
  print("Append a list:", list)
```

existing list ['Mon', 'Tue', 'Wed']

Appended one element ['Mon', 'Tue', 'Wed', 'Thus']

Appended a list ['Mon', 'Tue', 'Wed', ['Fri', 'Sat']

(11) list.clear()

clears the contents

(3) list.copy()

returns a copy of the list object.

(9) list.count(obj)

returns count of how many times obj occurs in list.

(5) list.extend(seq)

appends the contents of seq to list.

eg:

```
my_list = [1, 2, 3]
print("Before clear:", my_list)
my_list.clear()
print("After clear:", my_list)
```

Output

Before clear : [1, 2, 3, 4, 5]

After clear : []

② eg: original_list = [1, 2, 3, 4]

copied_list = original_list.copy()

copied_list.append(5)

print("Original list:", original_list)

print("Copied list", copied_list)

Output:

Original list : [1, 2, 3, 4]

Copied list : [1, 2, 3, 4, 5]

③ eg: my_list = [1, 2, 2, 3, 4, 2, 5]

count_of_two = my_list.count(2)

print("Count of 2:", count_of_two)

eg. Extend / Append

(1)

```
my_list = [1, 2, 3]
my_list.append([4, 5])
print(my_list)
```

o/p [1, 2, 3, [4, 5]]

```
my_list = [1, 2, 3]
my_list.extend([4, 5])
print(my_list)
```

output: [1, 2, 3, 4, 5]

310

Programs of Python

1. Creating a List
2. Appending an element to a list
3. Insert an element at a specific position
4. Removing an element from the list.
5. Access elements by Index
6. Slicing a List
7. Clear
8. Reverse
9. merging 2 List
10. Check if an element exist in a List

TUPLES

Tuple is one of the built in data types in Python. It is a sequence of comma separated items, enclosed in parenthesis.

tup1 = ('Rohan', 21, 14, 69.75)

empty tuple tup1 = ( )

Accessing Values:

① print (tup1[0])

output Rohan

② tup1[1:3]

output

21, 14

③ Updating Tuples:

IMMUTABLE

tup1 = (12, 34.56)
tup2 = ('ab', 'xyz')

tup3 = tup1 + tup2

print (tup3) output: (12, 34.56, 'ab', 'xyz')

Delete tuple elements

Removing individual tuple elements is not possible.

```
tup = ('Physics', 'Chemistry', 1997, 2000)
del tup
```

Python tuple operations:

1. `(1, 2, 3) + (4, 5, 6)` → `(1, 2, 3, 4, 5, 6)`
   Concatenation
2. `(1, 2, 3) * 4` → `(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)`
   Repitition
3. `3 in (1, 2, 3)` → True
   Membership

Indexing, slicing

```
T = ('spam', 'Spam', 'SPAM')
T[2]  = SPAM
T[-2] = Spam
T[1:] = Spam, SPAM
```

Any set of multiple objects, comma separated, written without identifying symbols. i.e, brackets for lists, parentheses for tuples.

Built-In functions with Tuples

(1) cmp(tuple1, tuple2)

removed in 3.0
does not handle

(2) len(tuple)

(3) max(tuple)

```
tuple = (465, 578, 253, 957)
res = max(tup)
print("maximum element is", res)
```

Maximum element is: 957

(4) min(tuple)

~~output~~ gives minimum element from the tuple

~~slicing~~ ~~using~~

Updating tuples using slicing :

```
T1 = (37, 14, 95, 40)
new_elements = ('green', 'blue', 'red', 'pink')

part1 = T1[:2]
part2 = T2[2:]

updated_tuple = part1 + new_elements + part2

print("Original Tuple"), T1)
print("Updated Tuple"), updated_tuple)
```

Original Tuple: (37, 14, 95, 40)
Updated Tuple: (37, 14, 'green', 'blue', 'red', 'pink', 95, 40)

Updating tuples using List Comprehension

```
T1 = (10, 20, 30, 40)
list_T1 = list(T1)
updated_list = [item + 100 for item in list_T1]

updated_tuple = tuple(updated_list)
print("Original Tuple", T1)
print("Updated Tuple", updated_tuple)
```

Output:

Original tuple : (10, 20, 30, 40)

Updated tuple: (110, 120, 130, 140)

List Comprehension:

It is a concise way to create lists in Python. It allows you to generate a new list by applying an expression to each item in an existing iterable (like a list or a tuple), optionally filtering items with a condition.

eg. ① numbers = [1, 2, 3, 4]

new_list = [n + 10 for n in numbers]

eg. ② numbers = [1, 2, 3, 4, 5, 6]

even_numbers = [n for n in numbers if n%2==0]

print (even_numbers)

Updating tuple using append() function.

```
T1 = (10, 20, 30, 40)
list_T1 = list(T1)
new_elements = [50, 60, 70]
for element in new_elements:
    list_T1.append(element)
updated_tuple = tuple(list_T1)
print("Original Tuple:", T1)
print("Updated Tuple:", updated_tuple)
```

Original Tuple: (10, 20, 30, 40)
Updated Tuple: (10, 20, 30, 40, 50, 60, 70)

Unpack tuple items:

```
tup1 = (10, 20, 30)
x, y, z = tup1
print("x:", x, "y:", y, "z:", z)
```

Unpack Tuple Items Using (*)

```
tup1 = (10, 20, 30)
x, *y = tup1
print("x:", "y:", y)
```

Output:

```
x: y: [20, 30]
```

→ Loop Through Tuple Items with for Loop and while loop.

```
tup = (25, 12, 10, -21, 10, 100)
for num in tup:
    print(num, end=' ')
```

→

```
my_tup = (1, 2, 3, 4, 5)
index = 0
while index < len(my_tup):
    print(my_tup[index])
    index += 1
```

Syntax for using the sum() function to join tuples in Python:

result_tuple = sum((tuple1, tuple2), ())

joined tuple : (10, 20, 30, 40, 'one', 'two', 'three', 'four')

→ tuple.count(obj)

returns count of how many times obj occurs in tuple.

→ tuple.index(obj)

returns the lowest index in tuple that obj appears.

→ eg:

```
tup2 = (10, 20, 45, 10, 30, 10, 55)
print("Tup1:", tup1)
c = tup1.count(10)
print("Count of 10:", c)
```

output: Tup1 : (10, 20, 45, 10, 30, 10, 55)
Count of 10 : 3

Programs Tuple

Program to find Unique numbers in a given tuple.

1) T1 = (1, 9, 1, 6, 3, 4, 5, 1, 1, 2, 5, 6, 7, 8, 9, 2)

```
T2 = ( )
for x in T1:
    if x not in T2:
        T2 += (x,)
print ("original tuple:", T1)
print ("Unique numbers", T2)
```

original tuple: (1, 9, 1, 6, 3, 4, 5, 1, 1, 2, 5, 6, 7, 8, 9, 2)
Unique numbers: (1, 9, 6, 3, 4, 5, 2, 7, 8)

2) Python program to find sum of all numbers in a tuple.

T1 = (1, 9, 1, 6, 3, 4)

```
tt1 = 0
for x in T1:
    tt1 += x
print ("sum of all numbers using
        loop:", tt1)

tt1 = sum (T1)
print ("sum of all numbers
        sum( ) function:", tt1)
```

Dictionaries

Creating

```
Dict = { "name" : "Richa",
         "age"  : 25,
         "gender" : 'F'
         "grade" : A
         "House No" : 265 }
```

Keys → "name", "age", "gender", "grade", "House No"

Values → "Richa", 25, 'F', A, 265

Keys are unique

Value may repeat

```
print(Dict)
```

Dictionaries are Unordered, Mutable

Sequence built in data types having items as key value pairs

Accessing :

~~print(Dict[gender])~~

① `print(Dict["gender"])`

output : F

(2) print(dict.get("gender"))

output: F

(3) ~~print(dict.get~~

(3) print(dict.values())

~~output: dict_values(['ford',~~

~~Output: dict_values(['name', 'age', 'gender~~

Output: dict_values(['Richa', 25, 'F', A, 265])

or print(list(student.values()))

(4) print(dict.keys())

output:

dict_keys(['name', 'age', 'gender', 'grade', 'house no.'])

(II) Change Values:

```
Dict["age"] = 43
```

(III) Adding a new key value pair

```
Dict["Height"] = "56 feet"
```

(IV) Removing item using del, pop()

(I) `del Dict["gender"]`

(II) `Dict.pop("gender")`

(III) `del Dict`

(V) Loop through a dictionary

~~for x in Dict~~
~~print(x)~~ ① Looping through keys

```
for key in Dict:
    print(key)
```

output: name
age
gender
House No.

② Looping through values

```
for value in Dict.values():
    print(value)
```

output: Rich
25
F
A
265

③ Looping through keys and Values

To loop through both keys & values at the same time, use the items() method.

```
for key, values in dict.items():
    print(f"{key}: {values}")
```

Dictionary Methods:

```
dict.clear()
dict.copy()
dict.fromkeys()
dict.has_key(key)
dict.key
dict.get(key, default=None)
dict.items()
dict.pop()
dict.values()
```

→
```
dict.clear()
print("dictionary after clear():", dict)
```

→ dict.fromkey() method is used to create a new dictionary with keys from an iterable (such as a list or tuple) and set all the values to a specified value (which defaults to None if not provided).

dict.fromkeys(iterable, value)

→ dict.fromkeys()

```
seq = ['a', 'b', 'c']

new_dict = dict.fromkeys(seq)
print(new_dict)
```

output

{'a': None, 'b': None, 'c': None}

```
keys = ['a', 'b', 'c']
new_dict = dict.fromkeys(keys, 0)
print(new_dict)
```

{'a': 0, 'b': 0, 'c': 0}

→ dict.has_key(key)

If the key passed as an argument to the get() method is present in the dictionary, it returns its corresponding value.

→ dict.get(key, value)

→ dict.get(key)

If key is not there, None is default value.

→ dict.items()

* It gives you a view of the dictionary, meaning if you update the dictionary after calling items(), the changes will appear in the view object as well.

* It gives you a view of the dictionary which contains all the key-value pairs as tuples.

* This view reflects any changes made to the dictionary, meaning if you update the dictionary after calling items(), the changes will appear in the view object as well.

* It is most commonly used when you want to iterate over both keys and values at the same time.

→ dict.keys()

It is used to retreive the list of all the keys in the dictionary.

Output:

dict_keys(['name', 123, 'class'])

→ dict.popitem()

It is used to remove and return an arbitrary key-value pair from the dictionary. It modifies the dictionary in place.

When you call popitem(), it removes and returns an arbitrary key-value pair as a tuple.

If the dictionary is empty, calling popitem() raises a KeyError.

If the dictionary is not empty, popitem() removes and returns a key value pair. Which pair is removed is not guaranteed and may vary between Python implementations or versions.

In Python 3.7, & later versions, it generally removes the last key-value pair that was added to the dictionary.