Tuples

1. Tuples are denoted by parentheses () ###identifier ---
2. Tuple is immutable
3. Duplicate elements are allowed in tuple
4. Multiple datatypes are allowed in tuple
5. Indexing is supported and accessing of elements is done with index position

```
a = (1,2,5.9,True)
type(a)
```

```
tuple
```

Replacing values in Python Tuple

```
b = (10,20,[40.6,True,8j,75],-1,20,30)
type(b)
```

```
tuple
```

```
b[2][2] = False
b
```

```
(10, 20, [40.6, True, False, 75], -1, 20, 30)
```

Concatenating 2 Python Tuples

```
tuple1 = (0, 1, 2, 3)
tuple2 = ('Zahid', 'Shaikh')
```

```
print(tuple1 + tuple2)
```

```
(0, 1, 2, 3, 'Zahid', 'Shaikh')
```

Nesting of Python Tuples

```
tuple1 = (0, 1, 2, 3)
tuple2 = ('Zahid', 'Shaikh')

tuple3 = (tuple1, tuple2)
print(tuple3)
```

```
((0, 1, 2, 3), ('Zahid', 'Shaikh'))
```

Repetition Python Tuples

```
tuple1 = ("Zahid ") * 7
tuple1
```

```
'Zahid Zahid Zahid Zahid Zahid Zahid Zahid '
```

Slicing Tuples in Python

```
b = (10,20,[40.6,True,8j,75],-1,20,30)
b[1:7:2]
```

```
(20, -1, 30)
```

Multiple Data Types With Tuple

```
tuple_obj = ("immutable",True,23)
print(tuple_obj)
```

```
('immutable', True, 23)
```

Packing & Unpacking

```
a=1,2.5,3,'hello',True
a
```

```
(1, 2.5, 3, 'hello', True)
```

```
b,c,d,e,f=a
f
```

```
True
```

String

Strings in python are surrounded by either single quotation marks, or double quotation marks.

```
a='hi today is tuesday date is 12/12/23.'
type(a)
```

```
str
```

```
a = "Zahid Salim Shaikh"
for i in a:
    print(i)
```

```
Z
a
h
i
d

S
a
l
i
m

S
h
a
i
k
h
```

```
b = "23"
type(b)
```

```
str
```

```
c = "78"
d = "25"
print(c+d)
```

```
7825
```

Upper(), Lower() & Swapcase()

```
a='Hi Today Is Tuesday Date Is 12/12/23.'
print("This is Upper case:",a.upper())
print("This is Lower case:",a.lower())
print("This is Swap case:",a.swapcase())
```

```
This is Upper case: HI TODAY IS TUESDAY DATE IS 12/12/23.
This is Lower case: hi today is tuesday date is 12/12/23.
This is Swap case: hI tODAY iS tUESDAY dATE iS 12/12/23.
```

Strip()

- The strip() method removes any leading, and trailing whitespaces.
- Leading means at the beginning of the string, trailing means at the end.
- You can specify which character(s) to remove, if not, any whitespaces will be removed.

```
txt = ",,,,,zzttbb.....grapes....iii"

x = txt.strip(",.zbit")

print(x)

grapes
```

split()

- The split() method splits a string into a list.
- You can specify the separator, default separator is any whitespace.

```
txt = "hello, my name is Zahid, I am 23 years old"

x = txt.split(", ")

print(x)

['hello', 'my name is Zahid', 'I am 23 years old']
```

Learning Progress:

Tuples:

- I have comprehensively explored the concept of tuples in Python. I understand their properties and limitations, including being immutable and ordered collections of elements.
- I have mastered various operations associated with tuples, including accessing elements by index, slicing, and tuple packing/unpacking.
- I can effectively compare tuples and identify duplicates or subsets.

Strings:

- I have thoroughly grasped the concept of strings in Python, including their manipulation and formatting techniques.
- I am proficient in utilizing various string methods like len(), upper(), lower(), strip(), split(), and more.
- I can effectively iterate through characters in a string and perform operations based on their position.

Next Steps:

Program Development:

I am ready to build a program that counts the occurrence of uppercase letters, lowercase letters, and spaces in a given string. I will utilize string methods and loop structures to achieve this objective.

Program 1

```
string = "Lorem Ipsum Is Simply Dummy Text Of The Printing And
Typesetting Industry."
newstring = ''
upper_count = 0
lower_count = 0
space_count = 0

for i in string:
    if (i.isupper() == True):
        upper_count += 1
        newstring += (i.lower())
    elif (i.islower() == True):
        lower_count += 1
        newstring += (i.upper())
    elif (i.isspace()) == True:
        space_count += 1
        newstring += i

print("In original String : ")
print("Number of Character :",len(string))
print("Uppercase -", upper_count)
print("Lowercase -", lower_count)
print("Spaces -", space_count)
print("After changing cases:")
print(newstring)
```

```
In original String :
Number of Character : 74
Uppercase - 12
Lowercase - 50
Spaces - 11
After changing cases:
lOREM iPSUM iS sIMPLY dUMMY tEXT oF tHE pRINTING aND tYPESETTING
iNDUSTRY
```

DICTIONARY

1. IT IS DENOTED BY CURLY BRACKETS{}.

2. IT IS IN THE FORM OF KEY AND VALUE PAIR.
3. WE CAN ACCESS THE VALUE WITH THE HELP OF KEY.
4. TRY NOT TO REPEAT THE KEY ALTHOUGH VALUE CAN BE REPEATED.
5. MULTIPLE DATATYPES ARE ALLOWED.
6. MUTABLE

```
a={'key1':'value1','key2':'value2','key3':'value3','key1':'value4','key5':'value4'}
a['key1']
```

```
'value4'
```

```
type(a)
```

```
dict
```

Getting the values using keys and indexing

```
b={1:'name',2:['phone number','address'],3:('email id',True)} #It supports multiple datatypes
b[3][1]
```

```
True
```

```
b[2][0] #in first bracket we use key and then in next index
```

```
'phone number'
```

keys() to get the keys in dictionary

```
b.keys() #to get the keys
```

```
dict_keys([1, 2, 3])
```

values() to get the values in dictionary

```
b.values() #to get the values
```

```
dict_values(['name', ['phone number', 'address'], ('email id', True)])
```

items() to get the keys-value pairs in dictionary

```
b.items() #to get the key-values
```

```
dict_items([(1, 'name'), (2, ['phone number', 'address']), (3, ('email id', True))])
```

Replacing values in dictionary

```
b[2][0] = 'mobile number'
b
```

```
{1: 'name', 2: ['mobile number', 'address'], 3: ('email id', True)}
```

Add/Updating values in dictionary

```
b.update({4:'DOB'})
b
```

```
{1: 'name', 2: ['mobile number', 'address'], 3: ('email id', True), 4:
'DOB'}
```

```
a={'rank':1,'percentage':90}
b.update(a)
b
```

```
{1: 'name',
 2: ['mobile number', 'address'],
 3: ('email id', True),
 4: 'DOB',
 'rank': 1,
 'percentage': 90}
```

Removing values from dictionary using keys

pop()

```
b.pop(4)
```

```
'DOB'
```

```
b
```

```
{1: 'name',
 2: ['mobile number', 'address'],
 3: ('email id', True),
 'rank': 1,
 'percentage': 90}
```

popitem()

popitem() method removes the last inserted key-value pair from the dictionary and returns it as a tuple.

```
b.popitem()
```

```
('percentage', 90)
```

del

It is used to delete a specific entry using value

```
del b['rank']
b
```

```
{1: 'name', 2: ['mobile number', 'address'], 3: ('email id', True)}
```

clear()

This will clear whole Dictionary

```
b.clear()
b
```

```
{}
```

```
a={'name1':30,'name2':20}
b={'name3':10,'name4':50}
c={'name5':50,'name6':25}

d={**a,**b,**c}

d
```

```
{'name1': 30, 'name2': 20, 'name3': 10, 'name4': 50, 'name5': 50,
'name6': 25}
```

Typecasting

Type Casting is the method to convert the Python variable datatype into a certain data type in order to perform the required operation by users.

```
a = 78
print(type(a),a)

b = float(a)
print(type(b),b)

c = str(a)
print(type(c),c)
```

```
<class 'int'> 78
<class 'float'> 78.0
<class 'str'> 78
```

```
a = [10,20,40.6,True,8j,75,-1,20,30]
print(type(a),a)

b = set(a)
print(type(b),b)
```

```
c = tuple(a)
print(type(c),c)
```

```
<class 'list'> [10, 20, 40.6, True, 8j, 75, -1, 20, 30]
<class 'set'> {True, 40.6, 10, 75, 20, 30, 8j, -1}
<class 'tuple'> (10, 20, 40.6, True, 8j, 75, -1, 20, 30)
```

Learning Progress:

Dictionaries:

- I have comprehensively explored the concept of dictionaries in Python, understanding their properties and limitations as unordered collections of key-value pairs.
- I have mastered various operations associated with dictionaries, including accessing values using keys, adding new key-value pairs, modifying existing values, and deleting elements.
- I can efficiently iterate through key-value pairs and perform operations based on the data within.

Type Casting:

- I have thoroughly grasped the concept of typecasting in Python, understanding how to convert data types from one form to another.
- I can utilize various built-in functions like int(), float(), str(), and bool() to perform type conversion effectively.
- I understand the importance of choosing the appropriate type conversion function based on the desired outcome and data context.

Next Steps:

- Although I will not be building any projects specific to this module, I will continue to practice and apply my knowledge of dictionaries and type casting in future projects and coding exercises.
- I will actively seek opportunities to utilize these concepts in real-world scenarios to solidify my understanding and enhance my problem-solving skills.

```

```