Python Class – 3b and 4b

Strings and Dictionaries

Strings

In Python, **Strings** are arrays of bytes representing Unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string. Indexing in a string is done in the same way as in a list.

Strings are immutable in nature. That is if you declare a string the value of it cannot be changed after declaration.

```
a="Hello"
a[1]="b"
```

This will throw an error as Strings are immutable.

The id() function returns the id of a particular object in Python. Refer this for more details: https://www.geeksforgeeks.org/id-function-python/

Note that the membership function “is” compares the id of two objects.

Single and double quotes are used for single line strings and triple quotes for multiline strings.

```
# String with single quotes
print('Welcome to the ISTE SMP')

# String with double quotes
print("Welcome to the ISTE SMP")

# String with triple quotes
print('''Welcome to the ISTE SMP''')
```

Accessing individual characters and string slicing is same as that in the list so refer to that for details.

One important thing to note abt strings is that while using different type of quotes in one string only the outermost one shall be considered as "wrapping" the string.

```
a="Hello 'ISTE' Hello"
print(a)
 #Shall print Hello ‘ISTE’ Hello without showing any errors
```

To print the same type of quotes as the "wrapping" quotes \ is used before it. (Escape sequence)

```
a='Hello \'Hi\' '
print(a)
#This shall print Hello ‘Hi’ without throwing any errors.
```

Converting a str to a list

To convert the entire str without any changes use the list()

```
A=”Hello”
print(list(A))
```

To split using a separation parameter the split() function is used.

```
a="a b ab aa"
print(a.split("ab"))
print(a.split("b"))
print(a.split())
```

By default, if no parameter is passed ' ' (single white space) is taken. The passed separation parameter can be of any length.

Converting a list to a str

To convert a list to a str we use the join() function.

The syntax is like this

```
"<separation_parameter>".join([<str1>,<str2>,...])
```

Example:

```
print(" ".join(['Python','is','yeet']))
```

Updating a String

```
a="Hello"
#Changing the l at index 2 to L
a=a[0:2]+"L"+a[3:]
print(a)
```

String formatting

The format function is an extremely useful function which can be used for, as it's name suggests, string formatting.

There are 3 basic ways to use it.

```
# Default order
String1 = "{} {} {}".format('ISTE', 'SMP', 'Python')
print("Print String in default order: ")
print(String1)

# Positional Formatting
String1 = "{1} {0} {2}".format('ISTE', 'SMP', 'Python')
print("\nPrint String in Positional order: ")
print(String1)

# Keyword Formatting
String1 = "{l} {f} {g}".format(g = 'ISTE', f = 'SMP', l = 'Python')
print("\nPrint String in order of Keywords: ")
print(String1)
```

The {} denotes that a string object is to be filled in that position.

Integers such as Binary, hexadecimal, etc. and floats can be rounded or displayed in the exponent form with the use of format specifiers.

```
# Formatting of Integers
String1 = "{0:b}".format(16)
print("\nBinary representation of 16 is ")
print(String1)

# Formatting of Floats
String1 = "{0:e}".format(165.6458)
print("\nExponent representation of 165.6458 is ")
print(String1)

# Rounding off Integers
String1 = "{0:.2f}".format(1/6)
print("\none-sixth is : ")
print(String1)
```

b	-	binary representation
e	-	exponential representation
.2f	-	2 digit in decimal part representation

A string can be left() or centre(^) justified with the use of format specifiers, separated by colon(:).

```
# String alignment
String1 = "|{:<10}|{:^10}|{:>10}|".format('ISTE','SMP','Python')
print("\nLeft, center and right alignment with Formatting: ")
print(String1)

# To demonstrate aligning of spaces
String1 = "\n{0:^16} was founded in {1:<4}!".format("ISTESMPPython", 2012)
print(String1)
```

Note that the given amount of space is the total amount of characters in which the passed string will be aligned. If the passed string is greater than the allotted space then the string is printed as such.

Dictionaries

Dictionary is not an indexed data structure but one which accesses data based on a key. The passed key shall determine what data is returned. Dictionaries are mutable and heterogeneous in nature (can be homogeneous in nature).

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds `key:value` pair.

You use the 'key' to access the 'value'.
Note that a certain key can have only one value.
But multiple keys can access the same 'value.

{} is used to create a dictionary. Note that when you write the following statement an empty dictionary is made and not an empty set.

```
A={}
```

Creating a dictionary is done like this:

```
# Creating a Dictionary
# with Integer Keys
Dict = {1: 'ISTE', 2: 'SMP', 3: 'Python'}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)

# Creating a Dictionary
# with Mixed keys
Dict = {'Name': 'ISTE', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict)
```

key, value pair is separated by ':'
Each key, value pair separated from another key-value pair by a ,

Also note that dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

One thing to be noted is that while the 'value' can be anything (lists,tuples,sets etc) the 'key' has to be immutable.

That is you cannot use lists, to name a few, as a key. If a tuple has a list as an element, that also cannot be used.

You can also create nested dictionaries.

```
Dict = {1: 'ISTE', 2: 'SMP',  3:{'A' : 'Welcome', 'B' : 'To', 'C' : 'Python'}}
print(Dict[3][‘A’])
```

Adding elements is pretty peaceful as it's very similar to how you do it in an array in C or Java.

```
# Creating an empty Dictionary
Dict = {}
# Adding elements one at a time
Dict[0] = 'ISTE'
Dict[2] = 'SMP'
Dict[3] = 'Python'
print(Dict)
```

To find the length of the dictionary use the len function:

```
my_dict = {1: 'ISTE', 2: 'SMP', 3:'Python'}
z = len(my_dict)
print(z)
```

Deleting a particular key:

```
test_dict = {"Arushi" : 22, "Anuradha" : 21, "Mani" : 21, "Haritha" : 21}
print(test_dict)
del test_dict['Mani']
print(test_dict)
```

You can also use the pop() to delete a particular element.

```
d={1:"ISTE",2:"SMP"}
print(d.pop(1))
print(d)
```

The pop() function shall return the value for the key-value pair it is deleting and then delete the said key-value pair.

The popitem() function can also be used to pop an item out.

```
# Creating Dictionary
Dict = {1: 'ISTE', 'SMP': 'For', 3: 'Python'}
```

```
# Deleting an arbitrary key
# using popitem() function
pop_ele = Dict.popitem()
print(Dict)
print(pop_ele)
```

To pop out the last added key-value pair then popitem() is used.
However, for version before 3.7 it shall remove a random pair

You can clear all the elements from the list using the clear () function.

```
d={1:1,2:2,3:3,4:4}
print(d)
d.clear()
print(d)
```

To update the value for a key:

```
dict={1:'ISTE'}
print(dict)
dict[1]="Python"
print(dict)
```

To update the key for a value, copy the value onto the new key and then delete the old key:

```
d={1:"ISTE"}
print(d)
d[2]=d[1]
print(d)
del d[1]
print(d)
```

The keys() , values() and items() functions are functions which are used to return view objects for a dictionary. These objects cannot be used to access the dictionary elements directly so it is first converted to a list and then accessed. They provide a dynamic view on the dictionary's entries, which means that when the dictionary changes, the view reflects these changes.
Example: dict_values, dict_items and dict_keys.

```
d={1:'ISTE',2:'SMP',3:'Python'}
print(d.values())
print(d.keys())
print(d.items())
```

There are three ways when you use the 'in' operator along with for loop

- Iterate through all keys

- Iterate through all values
- Iterate through all key, value pairs

Iterate through keys

If you just simply use the dictionary as it is in a for loop it will iterate through the keys. You can also use the keys() function.

```
statesAndCapitals = {
                    'Gujarat' : 'Gandhinagar',
                    'Maharashtra' : 'Mumbai',
                    'Rajasthan' : 'Jaipur',
                    'Bihar' : 'Patna'
                   }

print('List Of given states:\n')

# Iterating over keys
for state in statesAndCapitals:
    print(state)
```

Iterate through values

We use the values() function

```
statesAndCapitals = {
                    'Gujarat' : 'Gandhinagar',
                    'Maharashtra' : 'Mumbai',
                    'Rajasthan' : 'Jaipur',
                    'Bihar' : 'Patna'
                   }

print('List Of given capitals:\n')

# Iterating over values
for capital in statesAndCapitals.values():
    print(capital)
```

Iterate through key-value pairs

We use the items () function

```
statesAndCapitals = {
```

```
                 'Gujarat' : 'Gandhinagar',
                 'Maharashtra' : 'Mumbai',
                 'Rajasthan' : 'Jaipur',
                 'Bihar' : 'Patna'
                }

print('List Of given states and their capitals:\n')

# Iterating over values
for state, capital in statesAndCapitals.items():
    print(state, ":", capital)
```