004: Data structures

Learning Outcome:	String, Collections: list, tuple, dictionary, sets

Definitions/Concepts	

Data structures	• Data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied on it.
	• Some in-built data structures of python are:
	  ○ List
	  ○ Tuple
	  ○ Sets
	  ○ Dictionary

String	• Any collection of numbers, letter, special character or space enclosed within double or single quotes is data of type String in Python

Important functions of String	

len():
• It returns the length of string.
• *Syntax: **len**(string)*
	`>>> a="toppr"`
	`>>> len(a)`
	`5`

.upper():
• It returns the string in upper case.
• *Syntax: string.**upper**()*
	`>>> b="codr"`
	`>>> b.upper()`
	`'CODR'`

.lower():
• It returns the string in lower case.
• *Syntax: string.**lower**()*
	`>>> c="TopprCodr"`
	`>>> c.lower()`
	`'topprcodr'`

• **Concatenation** means joining.
• **Plus(+)** sign can be used to concatenate two String variables.
	`>>> a="Toppr"`
	`>>> b="Codr"`
	`>>> c=a+b`
	`>>> print(c)`
	`ToprrCodr`

Lists

- It is a collection of data values, which can be of different data types.
- It is created using square brackets **[ ]**
- It is ordered, and each element of list has a unique address called index
- It is mutable, which means we can change its elements.
- To access any element of a list, we can use its index, this process is called indexing.

Indexing

0, 1, 2, 3 a_list = [1, 3.5, "new", True]	In this list **a_list,** the first element of is **1** and its index is **0.** Similarly, the second element is **3.5** and its index is **1** and so on.

To access a single element of the list we can simply use square brackets.

```
>>> a_list[3]
True
```

Note: Index of the 4th element of the list is 3

We can even access more than one element of the list.eg.

```
>>> a_list[0:2]
[1, 3.5]
```

Note: Elements with index starting from 0 and less than 2 are returned.

We can use negative indexes too.
a_list = [1, 3.5, "new", True]
-4, -3, -2, -1
	`>>> a_list[-3]`
	`3.5`
	`>>> a_list[-3:-1]`
	`[3.5, 'new']`

Tuples **e.g.**

```
a_tuple=("a",34,False,'cat')
```

- Similar to list, it is a collection of data values, which can be of different data types.
- It is created using round brackets **( )** or parentheses.
- It is ordered, and each element of tuple has a unique address called index
- It is immutable, which means we can't change its elements.
- To access any element of a tuple, we can use indexing.

Sets **e.g.**

```
a_set={'we',45,3.6}
```

- It is a collection of data values, which can be of different data types.
- It is created using curly brackets **{ }**.
- It is unordered, so indexing can't be done on sets.
- It is mutable, which means we can change its elements.
- Only unique elements can be stored in a set.

Dictionary **e.g.**

```
a_dict={1:'a',2:'b',3:'c'}
```

- It is a collection of **key-value** pairs.
- It is created using curly brackets **{ }.**
- The key-value pairs can be of any data type.
- The keys in a dictionary should be unique.

Important functions of Lists

.append():

- It is used to add an element at the end of the list
- *Syntax: list**.append**(element)*

```
>>> a=[1,2,3]
>>> a.append(7)
>>> a
[1, 2, 3, 7]
```

.insert():
• It is used to add an element at the specified index.
• *Syntax: list**.insert**(index,element)*
	`>>> b=['c','o','d','r']`
	`>>> b.insert(1,0)`
	`>>> b`
	`['c', 0, 'o', 'd', 'r']`

del:
• It is used to delete the whole list or a particular element using its index
• *Syntax: **del** list or **del** list[index]*
	`>>> del b[2]`
	`>>> b`
	`['c', 0, 'd', 'r']`

.remove():
• It is used to delete an element by its value
• *Syntax: list**.remove**(element)*
	`>>> b.remove('d')`
	`>>> b`
	`['c', 0, 'r']`

Important functions of Lists and Tuples	
len(): • It gives the number of elements present in it. • *Syntax: **len**(list)* or *__len__(tuple)*	`a=[1,2,3]` `b=('w','e')`

.index():
• It gives the index no.of first occurrence of an element.
• *Syntax: list**.index**(element)* or *tuple**.index**(element)*
	`>>> s=['t','o','p','p','r']`
	`>>> s.index('p')`
	`2`
	`>>> s=('t','o','p','p','r')`
	`>>> s.index('p')`
	`2`

.count():
• It gives the count of an element in a list or tuple
• *Syntax: list**.count**(element) or tuple**.count**(element)*
	`>>> s=['t','o','p','p','r']`
	`>>> s.count('o')`
	`1`
	`>>> s=('t','o','p','p','r')`
	`>>> s.count('p')`
	`2`

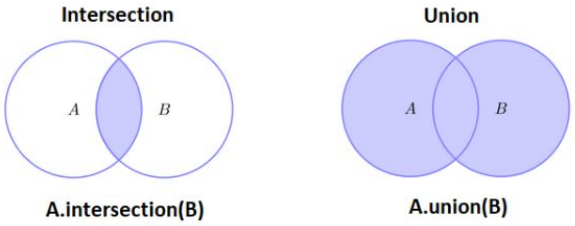
Important functions of Sets

.add()
- It is used to add an element in the set.
- Syntax: *set.**add**(element)*

```
>>> a={2,'r',5.7,11}
>>> a.add(78)
>>> a
{2, 11, 78, 'r', 5.7}
```

Let's take two sets:

```
setA={2,4,6,8,10,12,14}
setB={3,6,9,12,15}
```



.intersection()
- It returns the common elements of the two sets.
- Syntax: *setA.**intersection**(setB)*

```
>>> setA.intersection(setB)
{12, 6}
```

.union()
- It returns all the elements of the two sets.
- Syntax: *setA.**union**(setB)*

```
>>> setA.union(setB)
{2, 3, 4, 6, 8, 9, 10,
, 12, 14, 15}
```

Important functions of Dictionary

.values():
- It is used to display all the keys of the dictionary.
- *Syntax: dictionary.**values**()*

```
>>> d={'a':20,'b':300,'z':45}
>>> d.values()
dict_values([20, 300, 45])
```

.keys():
- It is used to display all the values of the dictionary.
- *Syntax: dictionary.**keys**()*

```
>>> d={'a':20,'b':300,'z':45}
>>> d.keys()
dict_keys(['a', 'b', 'z'])
```

.items()

- It is used to display all the key-value pairs of the dictionary.
- *Syntax: dictionary.**items**()*

```
>>> d={'a':20,'b':300,'z':45}
>>> d.items()
dict_items([('a', 20), ('b', 300), ('z', 45)])
```

Activity links and Solutions	
Student Activity 1: Strings	

#Q1: Convert the string **s** into upper case
`>>> s="I am learning to code in Python"`
`>>> s.upper()`
`'I AM LEARNING TO CODE IN PYTHON'`	

#Q2: Convert the string **t** into lower case	`>>> t="myTopprCodr"`
	`>>> t.lower()`
	`'mytopprcodr'`

#Q3: Create a string and display its length	`>>> a_string="CODR-toppr2020"`
	`>>> len(a_string)`
	`14`

#Q4: Extract two words from **w**	`>>> w="determination"`
	`>>> w[2:6]`
	`'term'`
	`>>> w[7:13]`
	`'nation'`

Student Activity 2: List and Tuples
#Q1: Create a list with elements of different data types. `>>> new_list=['toppr',2020,'codr',9.12]`
#Q2: Access the third element from the beginning and the third element from the end.

`>>> new_list[2]`
`'codr'`
For the 3rd element from beginning the index will be 3-1,i.e. 2
	`>>> new_list[-3]`
	`2020`
	For the 3rd element from end the index will be -3

#Q3: Create a tuple with elements of different data types.

```
>>> new_tuple=('12',345,True,7.8,'u')
```

#Q4: Access the second element from beginning and second element from the end.

`>>> new_tuple[1]`
`345`
For the 2nd element from beginning the index will be 2-1,i.e. 1
	`>>> new_tuple[-2]`
	`7.8`
	For the 2rd element from end the index will be -2

#Q5:Find the lengths of both list and tuple and display which is larger. {hint: if else conditional will be used}

```
>>> list_len=len(new_list)
>>> tuple_len=len(new_tuple)
>>> if(list_len>tuple_len):
        print("List is greater")
elif(tuple_len>list_len):
        print("Tuple is greater")
else:
        print("Both List and Tuple are of same length")
```

Student Activity 2: Set and Dictionary

#Q1: Create two sets, one containing first six multiples of 2 and the other containing first five multiples of 3

```
>>> m_2={2,4,6,8,10,12}
>>> m_3={3,6,9,12,15}
```

#Q2: Find the intersection and union of both the sets

```
>>> m_2.union(m_3)
{2, 3, 4, 6, 8, 9, 10, 12, 15}
>>> m_2.intersection(m_3)
{12, 6}
```

#Q3: Create a dictionary of your choice

```
>>> new_dict={'India':'New Delhi','Pakistan':'Islamabad',
'Nepal':'Kathmandu','Bangladesh':'Dhaka'}
```

#Q4: List all the keys and values of the dictionary separately.

```
>>> new_dict.keys()
dict_keys(['India', 'Pakistan', 'Nepal', 'Bangladesh'])
>>> new_dict.values()
dict_values(['New Delhi', 'Islamabad', 'Kathmandu', 'Dhaka'])
```

Fun-fact

Tim Peters, a major contributor to the Python community, wrote a poem 'The Zen of Python' to highlight the philosophies of Python. If you type in "import this" in your Python IDLE, you'll find this poem.