String:

```
slicing ->collecting group of character->syntax:->starting point to ending
point(sp:ep).end point is n-1= n is end point
```

Python string is Mutable Or Immutable=>strings are immutable meaning that once a string is created, it cannot be altered

Mutable-> we can increase and decrease memory size

Immutable-> we cant increase and decrease memory size

String Built-in Function:-

Function	Meaning	Syntax	Example
`capitalize()`	Capitalizes first letter, rest lowercase	`string.capitalize()`	`"hello".capitalize() → "Hello"`
`title()`	Capitalizes first letter of each word	`string.title()`	`"hello world".title() → "Hello World"`
`lower()`	Converts entire string to lowercase	`string.lower()`	`"HELLO".lower() → "hello"`
`upper()`	Converts entire string to uppercase	`string.upper()`	`"hello".upper() → "HELLO"`

`replace(old, new)`	Replaces substring with another	`string.replace(old, new)`	`"hello world".replace("world", "Python") → "hello Python"`
`islower()`	Checks if all characters are lowercase	`string.islower()`	`"hello".islower() → True`
`isupper()`	Checks if all characters are uppercase	`string.isupper()`	`"HELLO".isupper() → True`
`isalpha()`	Checks if all characters are letters (A-Z, a-z)	`string.isalpha()`	`"Hello".isalpha() → True`
`isnumeric()`	Checks if all characters are numbers	`string.isnumeric()`	`"12345".isnumeric() → True`
`isalnum()`	Checks if all characters are letters or numbers	`string.isalnum()`	`"Python3".isalnum() → True`
`startswith(substring)`	Checks if string starts with substring	`string.startswith(substring)`	`"Hello world".startswith("Hello") → True`
`endswith(substring)`	Checks if string ends with substring	`string.endswith(substring)`	`"Hello world".endswith("world") → True`

`count(substring)`	Counts occurrences of substring	`string.count(substring)`	`"banana".count("a") → 3`
`index(substring)`	Returns index of first occurrence	`string.index(substring)`	`"banana".index("a") → 1`
`split(separator)`	Splits string into list by separator (default space)	`string.split(separator)`	`"hello world".split() → ['hello', 'world']`
`join(iterable)`	Joins iterable into string with separator	`separator.join(iterable)`	`" ".join(['hello','world']) → "hello world"`
`in`, `not in`	Membership operators to check substring presence	`'sub' in string`	`"hello" in "hello world" → True`

`ASCII Concept`	Every character has a unique ASCII number	`ord('A') → 65`
		`chr(65) → 'A'`
			`ord('A') → 65`
			`chr(65) → 'A'`

split()=> used to break ->method splits a string into a list.=>output come in form of list

string.split(separator, maxsplit)=>

- `separator Optional. Specifies the separator to use when splitting the string. By default any whitespace is a separator`
- `maxsplit Optional. Specifies how many splits to do. Default value is -1, which is "all occurrences`

join() ->

join() method is a string method used to concatenate elements of an iterable (like a list, tuple, or set) into a single string, with a specified separator between each element.

`=========*******************************************************==========`

Tuple

Python tuple concept is going to represent with -> ()

- `python tuple concept can accept and data type`
- `tuple is an immutable, ordered collection of elements.`
- `Tuples are similar to lists but differ in that they cannot be modified after creation.`

```
python tuple is immutable
```

python Tuple we have only 2 built-in functions

Function	**Meaning**	**Syntax**	**Example**
`count()`	Returns the **number of times** a specified value appears in the tuple.	`tuple.count(value)`	`t = (1, 2, 2, 3)` `t.count(2) → 2`
`index()`	Returns the **index of the first occurrence** of the specified	`tuple.index(value)`	`t = (1, 2, 3)`

	value. Raises an error if not found.	`t.index(2)` → 1

compair List and Tuple

- `List` =>[],memory allocation,mutable,index,slicing,15 built-in function,skipping,
- `Tuple` =>(),memory allocation,immutable,index,slicing,skipping,2 built-infunctions

---

keyboard_arrow_down

Tuples Advantages:

- Only tuple is the concept which can hold multiple input to a single variable
- packing
- Unpacking

Aspect	**Packing**	**Unpacking**
Meaning	Putting multiple values into a single tuple.	Assigning tuple elements to multiple variables.
Action	**Combining** values into one tuple.	**Separating** tuple values into variables.
Syntax	`t = 1, 2, 3`	`a, b, c = t`
Example	`t = (1, 2, 3)`	`a, b, c = t`

====***==========

Set=>python set concept is going to represent with {}

- `Python set is an unorderes pair->which means it will not follow any order`
- `Since python set is unordered pair-> it will not follow Index->Python set concept will not have index->If there is no index->no slicing | no skipping`
- `python set concept wont allow duplicates`
- `python set concept is Mutable->but it will not allow mutable data type inside it`

Python set is Mutable or Immutable->prove it

- Since we dont have index in set concept
- Manually we cant increase or decrease memory size
- So we need to take help of Buil-in function

Built_in Function:

Increase

- add
- update

Decrease

- pop
- discard
- remove

other

- copy
- clear
- union
- intersection

Function	**Meaning**	**Syntax**	**Example**

`add()`	Adds a **single element** to the set.	`set.add(element)`	`s = {1,2}`
			`s.add(3)` → `{1,2,3}`

`update()`	Adds **multiple elements** (from list, tuple, set) to the set.	`set.update(iterable)`	`s = {1,2}`
			`s.update([3,4])` → `{1,2,3,4}`

`pop()`	**Removes and returns** a random element.	`set.pop()`	`s = {1,2,3}` `s.pop()` → Randomly removes 1 element

`discard()`	Removes a specific element if present. **No error** if not found.	`set.discard(element)`	`s = {1,2,3}`
			`s.discard(2)` → `{1,3}`

`remove()`	Removes a specific element. **Raises error** if not found.	`set.remove(element)`	`s = {1,2,3}`
			`s.remove(2)` → `{1,3}`

`copy()`	Returns a **shallow copy** of the set.	`set.copy()`	`s1 = {1,2}`
			`s2 = s1.copy()`

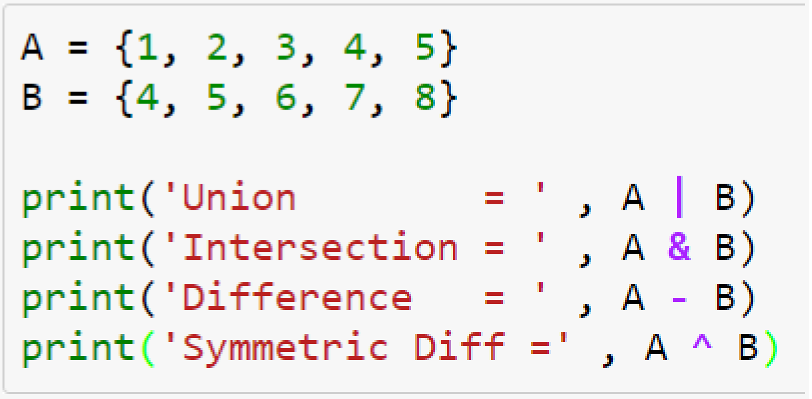
`clear()`	Removes **all elements** from the set (makes it empty).	`set.clear()`	`s = {1,2,3}`
			`s.clear()` → `set()`

`union()`	Returns a new set containing **all elements** from both sets (no duplicates).	`set1.union(set2)`	`{1,2}.union({2,3}) → {1,2,3}`
`intersection()`	Returns a set containing **common elements** of both sets.	`set1.intersection(set2)`	`{1,2}.intersection({2,3}) → {2}`

Set operations:

- Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference.
- We can do this with operators or methods.

Method	Operator	
union	|	
intersection	&	
difference	-	
symmetric_difference	^	

eg: 

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

print('Union        = ' , A | B)
print('Intersection = ' , A & B)
print('Difference   = ' , A - B)
print('Symmetric Diff =' , A ^ B)
```

- **Isdisjoint**– This method will return True if two set have a null intersection
- **Issubset** – This method reports whether another set contains this set
- **Issuperset** – This method will report whether this set contains another set

===========*************************************============

Python List

python list concept is going to represent with->[]

- `we can pass any data type into the list`

Python List is Mutable or Immutable :->

Python, lists are **mutable**. This means you can modify a list's contents after it has been created, including adding, removing, or changing elements

- `Try to increase the memory`
- `Try to decrease the memory`

List Built-in Function

To Increase Memory

- `append`
- `extend`
- `insert`

To Decrease Memory

- `pop`
- `remove`
- `count`

Some other Operations

- `index`
- `copy`
- `clear`
- `sort`
- `reverse`

Function	**Meaning**	**Syntax**	**Example**
`append()`	Adds a **single element** at the end of the list.	`list.append(element)`	`l = [1,2]` `l.append(3) → [1,2,3]`
`extend()`	Adds **multiple elements** from another iterable (list, tuple, set).	`list.extend(iterable)`	`l = [1,2]` `l.extend([3,4]) → [1,2,3,4]`
`insert()`	Inserts an element at a **specific position**.	`list.insert(index, element)`	`l = [1,2,4]` `l.insert(2, 3) → [1,2,3,4]`

`pop()`	Removes and returns the **last element** by default, or by index.	`list.pop([index])`	`l = [1,2,3]` `l.pop() → [1,2]`
`remove()`	Removes the **first occurrence** of the specified value.	`list.remove(value)`	`l = [1,2,3]` `l.remove(2) → [1,3]`
`count()`	Returns the **number of times** a value appears.	`list.count(value)`	`l = [1,2,2,3]` `l.count(2) → 2`
`index()`	Returns the **index** of the first occurrence of a value.	`list.index(value)`	`l = [1,2,3]` `l.index(2) → 1`
`copy()`	Returns a **shallow copy** of the list.	`list.copy()`	`l1 = [1,2]` `l2 = l1.copy()`
`clear()`	Removes **all elements** from the list (makes it empty).	`list.clear()`	`l = [1,2,3]` `l.clear() → []`
`sort()`	Sorts the list in **ascending order** by default.	`list.sort()`	`l = [3,1,2]` `l.sort() → [1,2,3]`
`reverse()`	**Reverses** the elements of the list.	`list.reverse()`	`l = [1,2,3]` `l.reverse() → [3,2,1]`

Shallow copy and Deep copy

Shallow Copy: A shallow copy creates a new object, but does not create copies of nested objects within the original. Instead, it copies references to these nested objects.

Deep Copy: A deep copy creates a new object and recursively copies all nested objects, ensuring that the new object is entirely independent of the original.

```
======*************************************************===========
```

Python Dictionary:

- `Represent with ->{}->but in a key_value pair concept`
- `key means column name | value means->Data inside the column`

```
#dict is used to prepare the data

#if we give to pandas library we can excel sheet
```

```
in place of keys-> we cant pass mutable
```

Python Dict is Mutable or immutable-> Mutable

- Python Dictionary is mutable, and indexed by keys (not by position)

Python Dictionary Built-in function

Purpose	**Functions**
Access elements safely	`get()`, `keys()`, `values()`, `items()`
Modify dictionary	`update()`, `pop()`, `popitem()`
Other operations	`clear()`, `copy()`

Function	**Meaning**	**Syntax**	**Example**
get()	Returns the **value** for a given key. Returns None if key not found (no error).	dict.get(key, default)	d = {'a':1} d.get('a') → 1 d.get('b') → None

update()	Updates the dictionary with elements from another dictionary or key-value pairs.	dict.update(other_dict)	d = {'a':1}
			d.update({'b':2}) → {'a':1, 'b':2}

pop()	Removes and returns the value for a specified key. Raises error if key not found.	dict.pop(key)	d = {'a':1, 'b':2} d.pop('a') → 1

popitem()	Removes and returns the **last inserted** key-value pair.	dict.popitem()	d = {'a':1, 'b':2}
			d.popitem() → ('b',2)

keys()	Returns a view object of **all keys** in the dictionary.	dict.keys()	d = {'a':1, 'b':2}
			d.keys() → dict_keys(['a','b'])

values()	Returns a view object of **all values** in the dictionary.	dict.values()	d = {'a':1}
			d.values() → dict_values([1])

`items()`	Returns a view object of **(key, value) pairs**.	`dict.items()`	`d = {'a':1}` `d.items() → dict_items([('a',1)])`
`clear()`	Removes **all key-value pairs** from the dictionary.	`dict.clear()`	`d = {'a':1}` `d.clear() → {}`
`copy()`	Returns a **shallow copy** of the dictionary.	`dict.copy()`	`d1 = {'a':1}` `d2 = d1.copy()`