

- **Set:**

- ☐ Sets are used to store multiple items in a single variable.
- ☐ A set is a collection which is unordered, unchangeable (immutable), unindexed and does not allow duplicate values.
- ☐ Unordered means that the items in a set do not have a defined order.
- ☐ Unchangeable means that we can add or remove items from a set after its creation. However, the individual elements themselves are generally immutable. You cannot directly modify an element within a set.
- ☐ Sets cannot have two items with the same values.
- ☐ Sets are typically created by enclosing a comma-separated sequence of items within a curly braces {}.
- ☐ Set supports operations inspired by mathematical set theory.

Set

```
# set creation
Set = {1,2,3,"apple", "raspberry", "banana"}
print(type(Set))
<class 'set'>
```

```
# convert list to set
L = [1,2,3,"apple", "mango", "raspberry"]
print(type(L))
<class 'list'>
S = set(L)
print(type(S))
<class 'set'>
print(S)
{1, 2, 'apple', 3, 'raspberry', 'mango'}
```

```
# convert tuple to set
T = (1,2,3, "apple", "mango", "raspberry")
print(type(T))
<class 'tuple'>
S = set(T)
print(type(S))
<class 'set'>
print(S)
{1, 2, 'apple', 3, 'raspberry', 'mango'}
```

```
S = {1,2,4,2,3,6,6}
S
{1, 2, 3, 4, 6}
```

```
L = [1,2,4,2,3,6,6,"apple", "apple", "orange"]
S = set(L)
print(S)
{1, 2, 3, 4, 'apple', 6, 'orange'}
```

Set (add, update):

```
# add
thisset = {"apple", "banana", "orange"}
print(thisset)
{'banana', 'apple', 'orange'}
thisset.add("mango")
print(thisset)
{'banana', 'apple', 'mango', 'orange'}

# update
x = {"apple", "banana", "cherry"}
y = {"SMIT", "Python", "apple"}
x.update(y)
print(x)
{'apple', 'SMIT', 'banana', 'cherry', 'Python'}
```

Set (difference, remove):

```
# difference
x = {"apple", "banana", "cherry"}
y = {"SMIT", "Python", "apple"}
z = x.difference(y)
print(z)
{'cherry', 'banana'}
z = y.difference(x)
print(z)
{'SMIT', 'Python'}

# remove
fruits = {"apple", "banana", "cherry"}
fruits.remove("cherry")
print(fruits)
{'apple', 'banana'}
fruits.remove("orange")
Traceback (most recent call last):
  File "<pyshell#913>", line 1, in <module>
    fruits.remove("orange")
KeyError: 'orange'
```

Set (discard, pop):

```
# discard
fruits = {"apple", "banana", "cherry"}
fruits.discard("apple")
print(fruits)
{'cherry', 'banana'}
fruits.discard("orange")
print(fruits)
{'cherry', 'banana'}

# pop
fruits = {"apple", "banana", "cherry"}
print(fruits)
{'apple', 'cherry', 'banana'}
fruits.pop()
'apple'
print(fruits)
{'cherry', 'banana'}
```

Set (union, intersection):

```
# union
x = {"apple", "banana", "cherry"}
y = {"SMIT", "Python", "apple"}
z = x.union(y)
print(z)
{'apple', 'SMIT', 'banana', 'cherry', 'Python'}

# intersection
x
{'apple', 'cherry', 'banana'}
y
{'apple', 'SMIT', 'Python'}
z = x.intersection(y)
print(z)
{'apple'}
```

Set (Built-in functions) [summary]:

- *add()* → Adds an element to the set.
- *update()* → Adds elements from another set, list or any iterable into the current set.
- *difference()* → Return elements of one set which is/are not present in other.
- *remove()* → Removes the specified element from the set. Raise error if element does not exist in the set.
- *discard()* → Removes the specified element, no error occurs if item doesn't exist.
- *pop()* → Removes the first element from the set.
- *union()* → Returns a set containing combined elements of sets.
- *intersection()* → Returns a set containing common elements between two sets.
- *symmetric_difference()* → Returns unique elements in both sets.

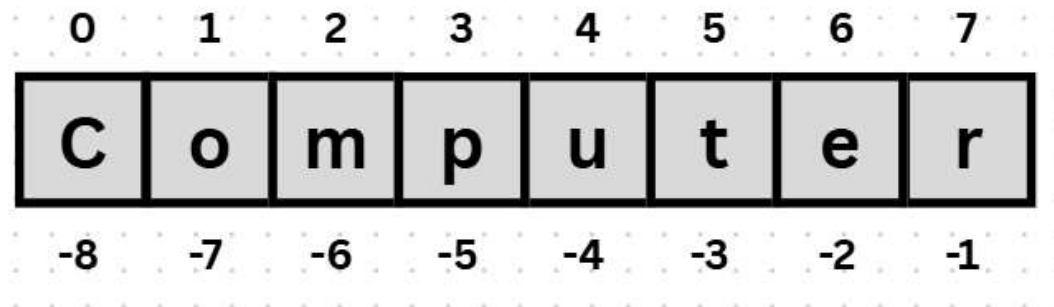
- **Strings slicing:**

Slicing means to extract specific portion (subsequences) from sequence.

Syntax → [start:end:step]

The third parameter defines:

- ❑ Difference between the indexes to be accessed.
- ❑ Direction of access i.e. negative difference defines the access direction from right to left.




```

>>> S = 'Computer'
>>> S[1:5:1]          # Same as S[1:5]
'ompu'
>>> S[1:5:2]
'op'
>>> S[1:3]           # Slice of S from offset 1 through 2 (not 3)
'om'
>>> S[1:]            # Slice of S from offset 1 to the end (1:len(S))
'omputer'
>>> S[:3]
'Com'
>>> S[5:1:-1]
'tupm'
>>> S[5:1:-2]
'tp'
>>> S[::-1]
'retupmoC'
>>> S[::-1]          # Same as S[::-1]
'Computer'
>>> S[-4:]           # access last 4 character of string same as S[4:]
'uter'
>>> S[:-4]           # access all character except the last 4 same as S[:4]
'Comp'

```

- **String concatenation and repetition:**

Joining strings using + operator and repeating strings using * operators.

```
string1 = "Hello"
string2 = "Python"
combined_string = string1 + " " + string2
repeated_string = string1 * 3

print(combined_string)
Hello Python
print(repeated_string)
HelloHelloHello
|
```

- **String case-conversion:**

Methods like *lower()*, *upper()*, *capitalize()*, *title()* and *swapcase()* to change the case of characters.

```
string = "PyThOn"
lowercase_text = string.lower()
uppercase_text = string.upper()
capitalize_text = string.capitalize()
title_text = string.title()
swapcase_text = string.swapcase()

print(lowercase_text)
python
print(uppercase_text)
PYTHON
print(capitalize_text)
Python
print(title_text)
Python
print(swapcase_text)
pYtHoN

string = "this is a sample string for title case"
print(string.title())
This Is A Sample String For Title Case
```

- **String searching and replacing:**

Methods like *find()*, *index()*, *replace()* is used to locate and modify substrings

```
sentence = "Python is powerful."
position = sentence.find("powerful")
print(position)
10
new_string = sentence.replace("powerful", "amazing")
print(new_string)
Python is amazing.
print(sentence)
Python is powerful.
```

```
sentence = "Python is powerful."
print(sentence.find("are"))
-1
print(sentence.index("are"))
Traceback (most recent call last):
  File "<pyshell#371>", line 1, in <module>
    print(sentence.index("are"))
ValueError: substring not found
```

```
sentence = "Hello world, Hello Python"
# replace all occurrences
new_string = sentence.replace("Hello", "Hi")
print(new_string)
Hi world, Hi Python
# replace a specific number of occurrences
new_string = sentence.replace("Hello", "Hi", 1)
print(new_string)
Hi world, Hello Python
```

- **String splitting and joining:**

The *split()* method is used to break a string into a list of substring based on a delimiter, and *join()* method to concatenate elements of an iterable into a single string.

```
data = "apple,banana,orange"
fruits = data.split(",")
print(fruits)
['apple', 'banana', 'orange']
joined_string = "-".join(fruits)
print(joined_string)
apple-banana-orange
```

- **String stripping:**

Methods like *strip()*, *lstrip()*, *rstrip()* is used to remove leading/trailing whitespaces or specified character.

```
padded_string = " hello "  
len(padded_string)  
7  
trimmed_string = padded_string.strip()  
print(trimmed_string)  
hello  
len(trimmed_string)  
5  
trimmed_string_left = padded_string.lstrip()  
print(trimmed_string_left)  
hello  
len(trimmed_string_left)  
6  
trimmed_string_right = padded_string.rstrip()  
print(trimmed_string_right)  
hello  
len(trimmed_string_right)  
6  
  
specific_character_text = "---Python---"  
stripped_specific = specific_character_text.strip("-")  
print(stripped_specific)  
Python
```

- **String count:**

The *count()* method is used to determine the number occurrences of a specified substring within a given string.

```
# counting occurrence of a string character
text = "hello world"
count_o = text.count("o")
print(count_o)
2

# counting occurrence of a substring
sentence = "the quick brown fox jumps over the lazy dog"
count_the = sentence.count("the")
print(count_the)
2

# counting within a specified range
string = "banana republic"
count_a_in_range = string.count("a", 3, 10)
print(count_a_in_range)
2
```