

## Built - In - Functions

| Name   | Usage   |
|--|---|
| <code>print()</code>                             | Function prints the message to the screen or any other standard output device.    |
| <code>int()</code>                               | Converts valid data of any type to integer.                                       |
| <code>str()</code>                               | Converts data of any type to a string.  |
| <code>id()</code>                                | To find the id of a object.   |
| <code>round(number,<br/>digits(optional))</code> | Rounds the float value to the given number of decimal digits.                     |
| <code>bool()</code>                              | Converts to boolean data type.  |
| <code>ord(character)</code>                      | Gives unicode value of the character.   |
| <code>chr(unicode)</code>                        | Gives character with the unicode value.   |
| <code>list(sequence)</code>                      | Takes a sequence and converts it into list.                                       |
| <code>tuple(sequence)</code>                     | Takes a sequence and converts it into tuple.                                      |
| <code>set(sequence)</code>                       | Takes any sequence as argument and converts to set, avoiding duplicates.          |
| <code>dict(sequence)</code>                      | Takes any number of key-value pairs and converts to dictionary.                   |
| <code>float()</code>                             | Converts to float data type.  |
| <code>type()</code>                              | Check the datatype of the variable or value using.                                |
| <code>min()</code>                               | Returns the smallest item in a sequence or the smallest of two or more arguments. |
| <code>max()</code>                               | Returns the largest item in a sequence or the largest of two or more arguments.   |
| <code>sum(sequence)</code>                       | Returns the sum of items in a sequence.   |

| Name  | Usage  |
|---|--|
| <code>sorted(sequence)</code>               | Returns a new sequence with all the items in the given sequence ordered in increasing order.   |
| <code>sorted(sequence, reverse=True)</code> | Returns a new sequence with all the items in the given sequence ordered in decreasing order.   |
| <code>len(sequence)</code>                  | Returns the length of the sequence.  |
| <code>map()</code>                          | Applies a given function to each item of a sequence (list, tuple etc.) and returns a sequence of the results.                        |
| <code>filter()</code>                       | Method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.             |
| <code>reduce()</code>                       | Receives two arguments, a function and an iterable. However, it doesn't return another iterable, instead, it returns a single value. |

**Floating Point Approximation:** Float values are stored approximately.

```
print(0.1 + 0.2) # 0.30000000000000004
```

**Floating Point Errors:** Sometimes, floating point approximation gives unexpected results.

```
print((0.1 + 0.2) == 0.3) # False
```

**Different compound assignment operators are:** +=, -=, \*=, /=, %=

```
a = 10
a += 1
print(a) # 11
```

```
a = 10
a -= 2
print(a) # 8
```

```
a = 10
a /= 2
print(a) # 5.0
```

```
a = 10
a %= 2
print(a) # 0
```

**Single And Double Quotes:** A string is a sequence of characters enclosed within quotes.

```
sport = 'Cricket'
sport = "Cricket"
```

**Escape Characters:** Escape Characters are a sequence of characters in a string that is interpreted differently by the computer. We use escape characters to insert characters that are illegal in a string.

```
print("Hello\nWorld")
```

```
# Output is:
Hello
World
```

We got a new line by adding \n escape character.

### Name Usage

|    |              |
|----|--------------|
| \n | New Line     |
| \t | Tab Space    |
| \\ | Backslash    |
| \' | Single Quote |
| \" | Double Quote |

## Set Methods, Operations and Comparisons

### Set Methods:

| Name      | Syntax               | Usage   |
|-----------|----------------------|---|
| add()     | set.add(value)       | Adds the item to the set, if the item is not present already.     |
| update()  | set.update(sequence) | Adds multiple items to the set, and duplicates are avoided.       |
| discard() | set.discard(value)   | Takes a single value and removes if present.                      |
| remove()  | set_a.remove(value)  | Takes a value and removes it if it is present or raises an error. |
| clear()   | set.clear()          | Removes all the items in the set.                                 |

### Set Operations:

**Union:** Union of two sets is a set containing all elements of both sets.

**Syntax:** set\_a | set\_b (or) set\_a.union(sequence)

```
set_a = {4, 2, 8}
set_b = {1, 2}
union = set_a | set_b
print(union) # {1, 2, 4, 8}
```

**Intersection:** The intersection of two sets is a set containing common elements of both sets.

**Syntax:** set\_a & set\_b (or) set\_a.intersection(sequence)

```
set_a = {4, 2, 8}
set_b = {1, 2}
intersection = set_a & set_b
print(intersection) # {2}
```

**Difference:** The difference of two sets is a set containing all the elements in the first set but not the second.

**Syntax:** set\_a - set\_b (or) set\_a.difference(sequence)

```
set_a = {4, 2, 8}
set_b = {1, 2}
diff = set_a - set_b
print(diff) # {8, 4}
```

**Symmetric Difference:** Symmetric difference of two sets is a set containing all elements which are not common to both sets.

**Syntax:** set\_a ^ set\_b (or) set\_a.symmetric\_difference(sequence)

```
set_a = {4, 2, 8}
set_b = {1, 2}
symmetric_diff = set_a ^ set_b
print(symmetric_diff) # {8, 1, 4}
```

**Set Comparisons:** Set comparisons are used to validate whether one set fully exists within another.

**issubset():** set2.issubset(set1) Returns True if all elements of the second set are in the first set. Else, False.

**issuperset():** set1.issuperset(set2) Returns True if all elements of second set are in first set. Else, False.

**isdisjoint():** set1.isdisjoint(set2) Returns True when they have no common elements. Else, False.

## Tuples

**Tuple:** Holds an ordered sequence of items. Tuple is an immutable object, whereas a list is a mutable object.

```
tuple_a = (5, "Six", 2, 8.2)
```

**Accessing Tuple Elements:** Accessing Tuple elements is also similar to string and list accessing and slicing.

```
tuple_a = (5, "Six", 2, 8.2)
print(tuple_a[1]) # Six
```

**Tuple Slicing:** The slice operator allows you to specify where to begin slicing, where to stop slicing, and what step to take. Tuple slicing creates a new tuple from an old one.

```
tuple= ('a','b','c','d','e','f','g','h','i','j')
print(tuple[0:2]) # ('a', 'b')
print(tuple[-1:-3:-2]) # ('j',)
print(tuple[1:7:2]) # ('b', 'd', 'f')
```

**Membership Check:** Check if the given data element is part of a sequence or not. Membership Operators `in` and `not in`.

```
tuple_a = (1, 2, 3, 4)
is_part = 5 in tuple_a
print(is_part) # False
```

```
tuple_a = (1, 2, 3, 4)
is_part = 5 not in tuple_a
print(is_part) # True
```

**Tuple Packing:** `()` brackets are optional while creating tuples. In Tuple Packing, Values separated by commas will be packed into a tuple.

```
a = 1, 2, 3
print(type(a))
print(a)

# Output is:
<class 'tuple'>
(1, 2, 3)
```

**Unpacking:** Values of any sequence can be directly assigned to variables. Number of variables in the left should match the length of the sequence.

```
tuple_a = ('R', 'e', 'd')
(s_1, s_2, s_3) = tuple_a
```

```
print(s_1, s_2, s_3) # R e d
```

## Dictionaries

**Dictionaries:** Unordered collection of items. Every dictionary item is a Key-value pair.

**Creating a Dictionary:** Created by enclosing items within {curly} brackets. Each item in the dictionary has a key-value pair separated by a comma.

```
dict_a = {  
    "name": "Teja",  
    "age": 15  
}
```

**Immutable Keys:** Keys must be of an immutable type and must be unique. Values can be of any data type and can repeat.

**Accessing Items:** To access the items in dictionary, we use square bracket [ ] along with the key to obtain its value.

```
dict_a = {  
    'name': 'Teja',  
    'age': 15  
}  
print(dict_a['name']) # Teja
```

**Accessing Items - Get:** The get() method returns None if the key is not found.

```
dict_a = {  
    'name': 'Teja',  
    'age': 15  
}  
print(dict_a.get('name')) # Teja  
print(dict_a.get('city')) # None
```

**Membership Check:** Checks if the given key exists.

```
dict_a = {  
    'name': 'Teja',  
    'age': 15  
}
```

```
result = 'name' in dict_a
print(result) # True
```

### Adding a Key-Value Pair:

```
dict_a = {'name': 'Teja', 'age': 15 }
dict_a['city'] = 'Goa'
print(dict_a) # {'name': 'Teja', 'age': 15, 'city': 'Goa'}
```

**Modifying an Existing Item:** As dictionaries are mutable, we can modify the values of the keys.

```
dict_a = {
    'name': 'Teja',
    'age': 15
}
dict_a['age'] = 24
print(dict_a) # {'name': 'Teja', 'age': 24}
```

**Deleting an Existing Item:** We can also use the del keyword to remove individual items or the entire dictionary itself.

```
dict_a = {
    'name': 'Teja',
    'age': 15
}
del dict_a['age']
print(dict_a) # {'name': 'Teja'}
```

## Sets

**Sets:** Unordered collection of items. Every set element is Unique (no duplicates) and Must be immutable.

**No Duplicate Items:** Sets contain unique elements

```
set_a = {"a", "b", "c", "a"}
print(set_a) # {'b', 'a', 'c'}
```

**Immutable Items:** Set items must be immutable. As List is mutable, Set cannot have list as an item.

```
set_a = {"a", ["c", "a"]}
print(set_a) # TypeError: unhashable type: 'list'
```

# Dictionary Views & Methods

## Dictionary Views:

| View   | Syntax                     | Usage                                      |
|--------|----------------------------|--|
| keys   | <code>dict.keys()</code>   | Returns dictionary Keys.                   |
| Values | <code>dict.values()</code> | Returns dictionary Values.                 |
| items  | <code>dict.items()</code>  | Returns dictionary items(key-value) pairs. |

## Dictionary Methods:

| Name   | Syntax                             | Usage  |
|--------|------------------------------------|--|
| copy   | <code>dict.copy()</code>           | Returns copy of a dictionary.                  |
| update | <code>dict.update(iterable)</code> | Inserts the specified items to the dictionary. |
| clear  | <code>dict.clear()</code>          | Removes all the elements from a dictionary.    |