Literals

- Python Literals can be defined as data that is given in a variable or constant.
- A literal in Python is a syntax that is used to completely express a fixed value of a specific data type. Literals are constants that are self-explanatory and don't need to be computed or evaluated.
- They are used to provide variable values or to directly utilize them in expressions.
- Generally, literals are a notation for representing a fixed value in source code.

❖ Types of Literals in Python

- **String literals**
- **Character literal**
- **Numeric literals**
- **Boolean literals**
- **Literal Collections**
- **Special literals**

❖ String Literals

- A string is literal and can be created by writing a text(a group of Characters) surrounded by a single(”), double(“), or triple quotes.
- We can write multi-line strings or display them in the desired way by using triple quotes.

Example:

```
# in single quote
s = 'Topper'
# in double quotes
t = "World"
print(s)
print(t)
```

Output:

```
Topper
World
```

❖ Character literal

It is also a type of Python string literal where a single character is surrounded by single or double quotes.

Example:

```
# character literal in single quote
v = 'n'
# character literal in double quotes
w = "a"
print(v)
print(w)
```

Output:

```
n
a
```

❖ Numeric literal

They are immutable and there are three types of numeric literal:

- Integer
- Float
- Complex

➢ Integer

Both positive and negative numbers including 0. There should not be any fractional part.

Example:

```
# integer literal

# Binary Literals

a = 0b10100

# Decimal Literal

b = 50

# Octal Literal

c = 0o320

# Hexadecimal Literal

d = 0x12b


print(a, b, c, d)
```

Output:

```
20   50   208   299
```

➢ Float

These are real numbers having both integer and fractional parts.

Example:

```
# Float Literal
e = 24.8
f = 45.0
print(e, f)
```

Output:

```
24.8   45.0
```

➢ Complex

The numerals will be in the form of **a + bj**, where ‘**a’** is the real part and ‘**b**‘ is the complex part. **Numeric literal [Complex]**

Example:

```
z = 7 + 5j


# real part is 0 here.
k = 7j
print(z, k)
```

Output:

```
(7+5j) 7j
```

❖ Boolean literal

There are only two Boolean literals in Python. They are **true** and **false**. In Python, **True** represents the value as **1,** and **False** represents the value as **0**.

Example:

```
a = (1 == True)
b = (1 == False)
c = True + 3
d = False + 7


print("a is", a)
print("b is", b)
print("c:", c)
print("d:", d)
```

Output:

```
a is True
b is False
c: 4
d: 7
```

❖ Literal collections

Python provides four different types of literal collections:

1. **List literals**
2. **Tuple literals**
3. **Dict literals**
4. **Set literals**

➢ List literal

The list contains items of different data types. The values stored in the List are separated by a comma (,) and enclosed within square brackets([]). We can store different types of data in a List. Lists are mutable.

Example:

```
number = [1, 2, 3, 4, 5]

name = ['Amit', 'kabir', 'bhaskar', 2]

print(number)

print(name)
```

Output:

```
[1, 2, 3, 4, 5]
['Amit', 'kabir', 'bhaskar', 2]
```

➢ Tuple literal

A tuple is a collection of different data-type. It is enclosed by the parentheses '**()**' and each element is separated by the comma(,). It is immutable.

Example:

```
even_number = (2, 4, 6, 8)

odd_number = (1, 3, 5, 7)


print(even_number)

print(odd_number)
```

Output:

```
(2, 4, 6, 8)
(1, 3, 5, 7)
```

➢ Dictionary literal

The dictionary stores the data in the key-value pair. It is enclosed by curly braces '**{}**' and each pair is separated by the commas(,). We can store different types of data in a dictionary. Dictionaries are mutable.

➢ Set literal

Set is the collection of the unordered data set. It is enclosed by the {} and each element is separated by the comma(,).

Example:

```
vowels = {'a', 'e', 'i', 'o', 'u'}

fruits = {"apple", "banana", "cherry"}


print(vowels)

print(fruits)
```

Output:

```
{'o', 'e', 'a', 'u', 'i'}
{'apple', 'banana', 'cherry'}
```

❖ Special literal

Python contains one special literal (None). **'None'** is used to define a null variable. If **'None'** is compared with anything else other than a **'None'**, it will return **false**.

Example:

```
water_remain = None

print(water_remain)
```

Output:

```
None
```