Datatypes

* → String in python can have single quote ' ' or double quote " " also """ """ Triple quotes. Triple quote is usually used for multi line strings or doc strings.

Suppose your string has single quote or double quote in between, then start-stop the string with double or single quote depend upon which quote the original string has.

Ex: print → Welcome to Python's world.

code → "Welcome to Python's World".

or 'Welcome to Python"s World".

* \n → New Line, \t → tab } → special character

r → raw string or regular expression.

→ By using r, we indicate to python to consider the character as regular & not as special character like newline(\n).

* To know the datatype of Variable in python we use type() function — its a built-in function.

Ex① word

'Welcome to Python's world'

dp type(word)

↳ <class 'str'>

word is an instance of class string.

str — is a built-in class in python.

② a = (1, 2, 3, 4)

dp type(a)

↳ <class 'list'>

* By using dir() function, using it we can find methods of a particular class.

Ex dir(word)

o/p → All method of str class will be displayed.

→ There is a method upper() in str class, if we use this, it will give a copy of strings but in upper case

Ex word
'hello world'

word.upper()
HELLO WORLD

} strings is imutable

→ strip() → Removes blank spaces
~~function~~ attribute.

→ split() → splits a sentence or string in to list

Ex Sentence = welcome to python

Sentence.split( )

o/p → ("welcome", 'to', "python")

→ len() → Built-in function, used to know the length of strings or any collection

→ word = "hello"

word.lower()

o/p → hello

word.count('l')

o/p → 2

⇒ Sentence

"hello world hello hello ~~there~~ there"

→ Sentence.find('l')
2 ⟶ index of 1st occurance

→ Sentence.find("world")
6 ⟶ gives index of 1st character of the word occurance

⟹ Sentence = "today is beautiful day"

→ Sentence.find("day")
o/p → 2

Sentence.rfind("day")
o/p ⟶ 19

* if a particular character is not present, it will return negative value.

→ sentence = "hello world"
Sentence.replace("world", 'Universe')
hello Universe.

} replace() will replace the original value with given value but it will not modify the original string & print.

⟹
word = "hello world"
word.startswith("hello")
o/p → True
word.endswith("hello")
o/p ⟶ False

} Returns True if the string has given string/word. else False

⟹ word = "***hello world***"
word.strip('*')
o/p 'hello world'

} strip method removes the specified characters & gives o/p.

⟶ word = " hello world "

word.strip()
hello world

} removes leading and trailing spaces in the string.

⟹

word = "hello"

'-'.join word

o/p h-e-l-l-o

} "-" will be added before or between every character

⟶

in ⟶ is called Membership operator, if it finds the item in collection it returns true else false.

Ex

sentence = "hello world welcome to python"

"java" in sentence

o/p → False

⟹ swapcase() ⟶ gives the o/p with swapped case
upper to lower +
lower to upper

string slicing

Slicing :-

-11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 ⟶ Negative indexing

⟹ h e l l o w o r l d

0 1 2 3 4 5 6 7 8 9 10 ⟶ Positive indexing

word = "hello world"

→ word(0)
o/p 'h'

} goes to index of string & give o/p of character at that index.

→ word(10)
o/p 'd'

→ word[-1]
'd'

} no matter the length of string,
To get 1st character use index 0
To get last character use index -1

word[2:9]
o/p → 'llo wor'

word[0:5]
o/p 'hello'

} end index, will go up to end index but not include it. + will print upto a character before it.

word[:9]
o/p 'hello wor'

} if starting index is not given then it starts from 0. + if end index is not given, it will go up to last index.

word[-4:]
o/p 'orld'

} start index is -4 + end index will be higher negative value (right side)

word[:-6]
o/p 'hello'

} it will go up to -6 + not include it.

word[-9:9]
o/p 'llo wor'

word[:]
o/p 'hello world'

word[3:-2]
o/p 'lo wor'

word[-10:-2]
o/p 'ello wor'

word[4:-10] } gives Empty string. because we are traversing in left.
''

word[8:-10] } Empty due to left side traversing.
''

→ word[::2] } step size is 2, will skip every 2nd character
'hlowrld'

>>> word[::1]
o/p 'hello world'
>>> word[::] } Default step size always +1.
o/p 'hello world

>>> word[::-1] } ~~left~~ to Right to left traversing
o/p 'dlrow oellh'

>>> word[-3:] } last 3 character, it will go from -3 to last of string.
o/p 'rld'

>>> file = "apple.pdf"
file[-3:]
o/p pdf

>>> url
'http://www.google.com
url[:4]
o/p: http

→ We can concat strings using + operator.

```
>>> year = 2022
>>> name = stere
>>> word = name + year
  o/p  Type Error
```

} We can concat only strings with another strings + not integer.

→ We can type cast an integer to string + then concat it.

→ we cannot concat 2 different objects.

```
>>> name
    'stere'
>>> year
    '2022'
>>> print(name, year)
    stere 2022
```

→ In Python we can concat 2 diff objects using coma [,] or + operator. [But for [+] we can concat only objects of same type]

```
fname = 'stere'
lname = 'jobs'
age = 26
sentence = f"hello {fname} you are {age} years of age
```

→ f stands for formatted string, we should use f and then type the sentence.

o/p = "Hello stere you are 26 years of age.

→ so we get the concated string with using '+' operator, just by using f letter followed by string, value should be in flower braces.

```
>>> print (f'{fname} {lname} {age}')
steve jobs 28
>>> print (f"{fname:>10} {name:>10} {age:>10}")
     steve       jobs         28
>>> print (f"{fname:<10} {name:<10} {age:<10}")
steve      jobs       28
```

Right justification ⟶

Left justification ⟶

centre justification ⟶ ^

List

⟶ List can hold duplicate.

⟶ We can add item by using append() method.

syntax
```
listname.append('itemname')
```

ex
```
name.append('apple')
```

⟶ `names.insert(0, 'apple')` — it will add apple at '0' — zero index.

⟶ Lists are mutable objects.

* if value of an object is changeable then its called mutable object.

>>> names.pop() ⟶ By default pop removes the last item & returns it.
Basically it modifies List.

a = names.pop(5) ⟶ pops item at index 5.

if value of ~~val~~ index is more then items in list. it gives index Out of ~~Bound~~ range.

⟹ reverse() method.

reverse() ⟶ it reverses the list items.

sort() ⟶ sorts list in ascending order & to descend to ~~reverse~~ order, we have to reverse & then sort, it will give o/p in descending order.

→ id() function gives memory address of an list.

→ We use Hex() function to get hexadecimal value of an integer.

→ extend() function. using it we can extend list

→ append() function ⟶ we can add item at the end of list

insert() function ⟶ we can add item at particular index. of strings.

Looping statement

=> for loop.

```
for i in range(0,10)
    print(i)
```

range() is a built in function

Syntax

range(start index, end index, step size)

Eg

```
for i in range(0,10,2)
    print(i)
```

o/p 0, 2, 4, 6, 8, 10

→ if we dont mention start index it will take default 0 (zero).

Eg

```
for i in range (10,-1,-1)
    print (i)
```

o/p

10	4
9	3
8	2
7	1
6	0
5	

word = "hello world"

Ex for i = range(0, len(word))
    print(word[i])

o/p
h
e
l
l
o
w
o
r
l
d

Ex for i in range(len(word)-1, -1, -1):
    print(word[i])

Ex for i in range(0, len(word))
    print(f'The character {word[i]} is at index {i}')

Ex for _ in range(0,5):
    print("Hello world")

'_' Underscore is called throw away Variable in python.

Ex word = "hello world"

~~for letter in word~~
~~print~~

for letter in word:
    print(letter)

index letter
for ~~letter, index~~ in enumerate(word)
    print(~~letter, index~~) (index, letter)

→ Built in function

o/p
0 h
1 e
2 l
3 l
4 o
5 w
6 o
7 r
8 l
9 d

→ When we need both item + index then we should make use of enumerate function.

Ex:- names = ["apple", "google", 'yahoo', "amazon", "flipkart"]

```
for name in names:
    print(name)
```

Ex:-
```
for name in reversed(names):
    print(name)
```

→ reversed is an built in function which is used to iterate

Ex:-
```
for i in range(0,10,0.5):
    print(i)
```

o/p → Type Error → we can not keep step size in floating no.

Ex:-
```
for name in names[::2]:
    print(name)
```

Ex:- Suppose we have 2 list of same length

```
for city, pop in zip(cities, population):
    print(city, pop)
```

→ zip is used to iterate through multiple iterables parallely

→ Suppose we have 3 list

```
a = [1, 2, 3]
b = ['x', 'y', 'z']
c = ['alpha', 'beta', 'gama']

for item1, item2, item3 in zip(a, b, c):
    print(item1, item2, item3)
```

o/p

```
1 x alpha
2 y beta
3 z gama
```

→ if any one of list have an item more i.e
1. list is longer than other, then it will ignore the extra item