

Module 8

Strings in Python

- A String is a sequence of characters
- characters can be
 - Alphabets (upper case and lower case)
 - Digits (0-9)
 - whitespace (space)
 - Special characters (@, #, \$, ...)
- They can be defined using single, double or triple quotes

```
s = "Uvarej"
```

```
type(s)
```

```
str
```

```
s1 = 'Uvarej'
```

```
type(s1)
```

```
str
```

```
s2 = """
```

```
This is a
```

```
• Multiline String
```

```
"""
```

```
type(s2)
```

```
str
```

ASCII vs Unicode (character encoding system)

ASCII and Unicode are both standards for representing characters on computers and other devices. But they differ in number of ways.

- ① ASCII was developed before Unicode
- ② ASCII supports upto 256 characters, while Unicode supports over 149,000. ASCII is limited to English characters, while Unicode can represent characters from many languages.
- ③ ASCII uses a fixed-length encoding of 8-bits to represent each character, while Unicode uses variable-length encoding of 7, 8, 16 or 32 bits.
- ④ Unicode is backward compatible with ASCII, as the first seven bits of code points in Unicode are identical to ASCII.

Python ord(), chr() functions

These are built-in functions, which are used to convert a character into an int, and vice versa. Python ord() and chr() functions are exactly opposite of each other.

Python ord() function takes string argument of a single Unicode character and returns its integer Unicode code point value.

ord("a")	ord("铁") ← Chinese
97	35830
ord("z")	ord("क") ← Hindi
122	2325
ord("A")	ord("😊") ← Emoji
65	128512
ord("Z")	
90	

Python chr() function takes integer argument and returns the string representing a character at that code point.

chr(97)

'a'

chr(2325)

'क'

Indexing in a String

String indexing in Python allows us to access individual characters in a string using their positions, called indices.

(i) Zero-Based Indexing

Here, the first character in a string is at index 0, the second character is at index 1, and so on.

my_string = "Hello"

my_string[0]

'H'

my_string[1]

'e'

len(my_string)

5

len("Uvaraj Thillai")

14

(ii) Negative Indexing

Negative indices count from the end of the string. The last character is at index -1, the second last is -2, and so on.

my-string [-1]

o

my-string [-2]

L

len(my-string)

5

print(my-string[-5])

H

(iii) Slicing

We can extract a substring using the slicing syntax.

my-string[start : end : step]

↑ optional : Default 1
↑ excluded
↑ included : Default 0

name = "Uvaraj Thillai"

size = len(name)

name[0 : size]

'Uvaraj Thillai'

name[2 : size]

'araj Thillai'

name[:]

'Uvaraj Thillai'

name[:2]

'Ua hla'

name[::]

'Uvaraj Thillai'

name[::-1]

'Uvaraj Thilla'

name[-2]

'a'

This will give the whole string.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
U	v	a	n	a	j		T	h	i	l	l	a	i

-15

name [-1 : 0 : -1]

name [-1 : 0]

'ialliht janar'

name [1 : -2]

'varaj Thill'

name [1 : -2 : -1]

' '

name [: : -1]

'ialliht jararU'

This gives whole string in Reverse.

name [-1 : -15 : -1]

'ialliht jararU'

String Methods

- Capitalize
- title
- upper
- lower
- find

- count
- index
- replace
- split

- islower
- isupper
- isnumeric
- isalpha

name = "Uvaraj Thillai"

name.capitalize()

"Uvaraj thillai"

name.title()

"Uvaraj Thillai"

name.upper()

"UVARAJ THILLAI"

name.lower()

"uvaraj thillai"

name.find ?

Docstring:

S.find(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. optional arguments start and end are interpreted as in slice notation. Return -1 on failure.

name.find("a")

2

name.count ?

Docstring:

S.count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

name.count("a")

3

name.count("z")

0

name.index("t")

7

← Returns the lowest index in string where substring is found.

name

'uvArAj thillai'

name.replace("a", "A") ← Replaces "a" with "A"

'uVArAj thillAi'

name.split()

['uvArAj', 'thillai']

name.split("a")

['uv', 'r', 'j thill', 'i']

name.islower()

True

"n".isupper()

False

"Uvaraj".islower()

True

"23".isnumeric()

True

"23a".isnumeric()

False

name.isalpha() ← As there is a space in the name,
it is not alphabetic.

False

"Uvaraj".isalpha()

True

String Formatting

name = input()

age = input()

print("My name is ", name, ". And my age is ", age)

Uvaraj

36

My name is Uvaraj. And my age is 36

name = input()

age = input()

print("My name is {}. My age is {}".format(name, age))

Uvaraj

36

My name is Uvaraj. My age is 36

name = "Uvaraj"

age = 36

print("My name is {name}. My age is {age}")

My name is {name}. My age is {age}

print(f"My name is {name}. My age is {age}")

My name is Uvaraj. My age is 36

} Available
from
Python 3.6
onwards

String Concatenation

first = "Uvaraj"

last = "Thillai"

print(first + last)

UvarajThillai

"2" + "a"

'2a'

"a" + "a" + "a"

'aaa'

"a" * 3

'aaa'

"2" + .2

Type Error: Can only concatenate str (not "int") to str.

"2" + "2"

'22'

2 + 2

4

Challenges

① Print all vowels of a given string.

```
text = "The quick brown fox jumps over the lazy dog"
```

```
for i in text:
```

```
    if i == "a" or i == "e" or i == "i" or i == "o" or i == "u":
```

```
        print(i, end=" ")
```

e u i o o u o e e a o

② Find if a string is Palindrome or not.

A string is said to be palindrome, if the reverse of the string is the same as the string. Eg. RADAR

```
s = input()
```

```
if s == s[::-1]:
```

```
    print("Palindrome")
```

```
else:
```

```
    print("Not Palindrome")
```

radar

Palindrome

radix

Not Palindrome

③ Length of unique words

Given two sentences, write a program to return the sum of the total number of unique words from each sentence.

Input format:

The first line indicates the No. of test cases. There will be two lines of inputs for each Test cases as following. The two lines consists of two sentences in String format.

Output format:

The no. of unique words from the sentences in integer format.

Note: If a word is present in both sentences, it should be counted separately for both sentences.

Sample input :

1
in data analysis we use data and process it further to create better interpreted data,
more and more data will be passively collected

Sample output :

20

def set_operation (sent1, sent2)

split sentences into words

words1 = set (sent1.split())

words2 = set (sent2.split())

calculate the lengths of unique words

unique_count1 = len (words1)

unique_count2 = len (words2)

Return the sum of unique word counts

return unique_count1 + unique_count2

set_operation (sentence1, sentence2)

20

④ What is the output of the following program ?

name = "Uvraj"

print (name[::-1])

'jararU'