Introduction to PYTHON

String

Create a String in Python

String Representation

Concatenation of Strings

Repetition of Strings

Updating Strings

Raw String

Strings

- A string is a sequence of characters.
- A character is simply a symbol.
- Computers do not deal with characters, they deal with numbers (binary).
- A character is internally stored and manipulated as a combination of 0's and 1's.
- This conversion of character to a number is called **encoding**, and the reverse process is **decoding.**

- **ASCII** and **Unicode** are some of the popular encoding used.
- Unicode was introduced to include every character in all languages and bring uniformity in encoding.

Python Strings

- Python does not support a character type
  - These are treated as strings of length one.
- Strings are **immutable**
- In Python, **string is a sequence of Unicode character**.
- Strings are ordered sequence of strings of length one.

Create a string in Python

- Strings can be created by enclosing characters inside a
  - Single quote
  - Double quotes
  - Even triple quotes
    - represent multiline strings and
    - docstrings.

String Representation

```
>>>'Hello'
'Hello'

>>>"Hello"
'Hello'

>>>"""Hello"""
'Hello'

>>>'''Hello'''
'Hello'
```

```
# triple quotes string can extend multiple lines
>>>my_string = """Hello, welcome to
    the world of Python""“

>>>print(my_string)
Hello, welcome to
the world of Python
```

Concatenation of Strings

```
>>>"Hello""World"
'HelloWorld'

>>>"Hello"+"World"
'HelloWorld’
```

Repetition of Strings

```
>>> "Hello"*2
'HelloHello‘
```

Updating Strings

- A string can’t be updated once it has been created.
- But, we can "update" an existing string by (re)assigning a variable to another string.
- The new value can be related to its previous value or to a completely different string altogether.

```
>>>var1 = 'Hello World!'

>>>print ("Updated String :- ", var1[:6] + 'Python')

Updated String :-  Hello Python
```

Python String Formatting

>>> print(“ *He said, "What's there?“* “)

... SyntaxError: invalid syntax

>>> print(‘ *He said, "What's there?“* ‘)

... SyntaxError: invalid syntax

Solution

- Use triple quotes

or

- Use backslash.

- The backslash (\) character is used to escape characters that otherwise have a special meaning, such as newline, backslash itself, or the quote character.

```
# using triple quotes
print(“‘ He said, "What's there?“  “’’)

# escaping single quotes
print('He said, "What\'s there?"')

# escaping double quotes
print("He said, \"What's there?\"")
```

```
>>>print("d:\new folder\team 1")
d:
ew folder       eam 1

>>>print("d:\\new folder\\team 1")
d:\new folder\team 1
```

```
>>> print("This is \x48\x45\x58 representation")
This is HEX representation
```

Raw String to ignore escape sequence

```
>>>print("Welcome to \new delhi")
Welcome to
ew delhi
```

```
>>>print(r"Welcome to \new delhi")

Welcome to \new delhi
```

Iterating Through String

```
count = 0
for letter in 'Hello World':
        if(letter == ‘o'):
                count += 1
print(count,'letters found')
```

String Membership Test

- We can test if a sub string exists within a string or not, using the keyword in.

```
>>> 'a' in 'program'
    True
>>> 'at' not in 'battle'
    False
```

Introduction to PYTHON

String Formatting

String Formatting

Using %

Using format()

Using f"String"

Old style formatting

String Formatting Using %

- This operator is unique to strings.

Example –

```
print ("My name is %s and roll number is %d " % (‘ABC', 210))
```

```
My name is ABC and roll number is 210
```

```
>>> x = 12.3456789

>>> print('The value of x is %.2f' %x)
    The value of x is 12.35

>>> print('The value of x is %.4f' %x)
    The value of x is 12.3457
```

Format Symbol	Conversion
%c	Character
%s	String
%d	Integer
%o	Octal
%x	Hexadecimal in Lower Case
%X	Hexadecimal in Upper Case
%f	Floating Point Number

String Formatting using format()

- Format strings contains curly braces {} as **placeholders or replacement fields** which gets replaced.
- We can use **positional arguments or keyword arguments** to specify the order.

```
# default(implicit) order

print("{}, {} and {}".format(‘AA','BB',‘CC'))
```

Output:

AA BB and CC

```
# order using positional argument

print("{1}, {0} and {2}".format(‘AA','BB',‘CC'))

Output:

BB AA and CC
```

```
# order using keyword argument

print("{c}, {b} and {a}".format(a=‘AA’, b=‘BB', c=‘CC'))
```

Output:

```
CC BB and AA
```

- The **format()** method can have optional format specifications.
- They are separated from field name using **colon**.

For example,

we can left-justify <

right-justify >

center ^

a string in the given space.

- We can also format integers as binary, hexadecimal etc.
- Floats can be rounded or displayed in the exponent format.

```
print("|{:<10}|{:^10}| {:>10}|".format("bread", "butter", "ham"))
```

```
|bread     |  butter  |        ham|
>>> "Binary representation of {0} is {0:b}".format(12)

'Binary representation of 12 is 1100'

>>>print("{0:b}, {0:o}, {0:x}".format(16))
10000,20,10
```

```
>>> # formatting floats
>>> "Exponent representation: {0:e}".format(1566.345)

'Exponent representation: 1.566345e+03'
```

```
>>> # round off
>>> print("One third is: {0:.3f}".format(1/3))

'One third is: 0.333‘
```

String formatting using f”String”

- Supported by Python Version 3.6 onwards
- **f-string** is a literal string,
- Prefixed with 'f',
- Contains expressions inside braces.
- The **expressions** are replaced with their **values**.
- **f-string** is really an expression evaluated at run time, not a constant value

```
>>>f"{2+3}"
'5'

>>>name = "James"
>>>last_name = "Bond"
>>>code = "007"

>>>f"Hi agent {name} {last_name} your code is {code}"
'Hi agent James Bond your code is 007'
```

Introduction to PYTHON

String's Functions

lower() upper() split() join() find() index()
capitalize() replace() center() count()
endswith() isalpha() isalnum() isdigit()
isnumeric() isspace() ljust() rjust() center()
lstrip() rstrip() strip() swapcase() title()

lower() & upper()

```
>>> "PrOgRaMmInG".lower()
    'programming'

>>> "PrOgRaMmIng".upper()
    'PROGRAMMING'
```

split(str=“ ", num=string.count(str))

Splits string according to delimiter str (space if not provided) and returns list of substrings;
split into at most num substrings if given.

```
>>> "This will split all words into a list".split()

['This', 'will', 'split', 'all', 'words', 'into', 'a', 'list']
```

```
>>>a="12304560789“
>>>a.split("0")
['123', '456', '789']
```

join()

```
>>> ' '.join(['This', 'will', 'join', 'all', 'words', 'into', 'a', 'string'])

'This will join all words into a string'
```

find()

```
>>> 'Happy New Year'.find('ew')
        7
```

index()

>>>s.index(substr, beg=0, end=len(string)**)**

Same as find(), but raises an exception if str not found.

replace(old, new [, max])

Replaces all occurrences of old in string with new or at most max occurrences if max given.

```
>>> 'Happy New Year'.replace('Happy','Brilliant')
‘Brilliant New Year‘

>>>a="12304560789“
>>>a.replace(“z","0")
'12304560789‘

>>>a.replace("0","11")
'1231145611789'
```

capitalize()

- Capitalizes first letter of string

```
>>>A='hello world'
>>>A.capitalize()
Hello world
```

center(width, fillchar)

- Returns a string padded with fillchar with the original string centered to a total of width columns.

```
>>>a="hello“
>>>a.center (10,'*')

'**hello***'
```

count(substr, beg,end**)**

- Counts how many times substr occurs in string or
- in a substring of string if starting index beg and ending index end are given.

```
a="hello world“
a.count('l',0,len(a))
```

Output:

```
 3
```

endswith(suffix, beg, end**)**

- Determines if string or
- a substring of string (if starting index and ending index are given)
  - ends with suffix; returns true if so and false otherwise.

```
>>>a="hello world“
>>>a.endswith('d',0,len(a))
True
>>>"Hello world".endswith("o",0,5)
True
```

isalnum()

isalnum()

- Returns true if string has at least 1 character and **all characters are alphanumeric** and false otherwise.

```
>>>"!!!".isalnum()
False
```

isalpha()

- Returns true if string has at least 1 character and **all characters are alphabetic** and false otherwise.

```
>>>a='123‘
>>>a.isalpha()
False

>>>a='aanbvnv'
>>>a.isalpha()
True
```

isdigit()   or isnumeric()

>>>isdigit()   or isnumeric()

- Returns true if the string contains only digits and false otherwise.

islower() & isupper()

islower()

- Returns true if string has all cased characters in lowercase and false otherwise.

isupper()

- Returns true if string has all cased characters in uppercase and false otherwise.

isspace()

isspace()

- Returns true if string contains only whitespace characters and false otherwise.

>>>"\n\t".isspace()

- True

>>>"   ".isspace()

- True

ljust() & rjust()

>>>ljust(width[, fillchar])

- Returns a space-padded string with the original string left-justified to a total of width columns.

>>>rjust(width,[, fillchar])

- Returns a space-padded string with the original string right-justified to a total of width columns.

```
>>>a='hh‘
>>>a.ljust(10,'*')
'hh********'
```

lstrip() & rstrip()

lstrip()

Removes all leading whitespace in string.

```
>>>a="   dffgfg“
```

```
>>>a.lstrip()
'dffgfg‘
```

rstrip()

- Removes all trailing whitespace of string.

strip([chars])

Performs both lstrip() and rstrip() on string

```
>>>"$Hello$World$".strip("$")
'Hello$World‘

>>>" hello world ".strip()
'hello world'
```

swapcase()

- Inverts case for all letters in string.

```
>>>"Hello World".swapcase()
'hELLO wORLD'
```

title()

```
>>>"hello world".title()
'Hello World'
```