Strings

- A string consists of a sequence of characters, which includes letters, numbers, punctuation marks and spaces.
- To represent strings, you can use a single quote, double quotes or triple quotes.

The *str()* Function

- The *str()* function returns a string which is considered an informal or nicely printable representation of the given object. The syntax for *str()* function is,
- ***str(object)***
- It returns a string version of the object. If the object is not provided, then it returns an empty string.

Basic String Operations

>>> str1="ECE"

>>> str2=“BENGALURU"

- >>> TotalStr1=str1+str2
- >>>TotalStr2="ECE "+“BENGALURU“
- >>> Totalstr3=560078+ "PIN“
- >>> Totalstr4=str(560078)+"PIN“
- >>> Totalstr4="560078"+"PIN“
- >>> Totalstr5=“BENGALURU"*4

Basic String Operations

- concatenated using + sign and * operator is used to create a repeated sequence of strings.

```
>>> str1="ECE"
>>> str2=“BENGALRU"
>>> TotalStr1=str1+str2
>>> TotalStr1
>>> TotalStr2="ECE "+“BENGALURU"
>>> TotalStr2
```

- >>> Totalstr3=560078+ "PIN"

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

Totalstr3=560078+ "PIN"

TypeError: unsupported operand type(s) for +: 'int' and 'str'

>>> Totalstr4=str(560078)+"PIN"

>>> Totalstr4

'560078PIN'

>>> Totalstr4="560078"+"PIN"

>>> Totalstr4

'560078PIN‘

>>> Totalstr5=“bangaluru"*4

>>> Totalstr5

in and *not in*

- check for the presence of a string in another string using *in* and *not in* membership operators.
- It returns either a Boolean *True* or *False*.
- >>> colgstrng="Engg college"
- >>> substr="Engg"
- >>> substr in colgstrng
- >>> substr1="Medical"
- >>> substr1 not in colgstrng

String Comparison

- You can use (>, <, <=, >=, ==, !=) to compare two strings resulting in either Boolean *True* or *False* value. Python compares strings using ASCII value of the characters.

- >>> “january"==“jane"

- >>> “january"<“jane"

- >>> “january">“jane"

- >>> “january"!=“jane"

```
>>> Count=len(“bengaluru")
>>> Count1=len(“place mysuru")
>>> max("bengaluru")
>>> min("bengaluru")
```

Characters with highest and lowest ASCII value are calculated

Accessing Characters in String by Index Number

- The index breakdown for the string *"be yourself"* assigned to *word_phrase* string variable is shown below.

The syntax for accessing an individual character in a string is as shown below.
string_name[index]

```
>>> word_phrase[1]
'e‘
```

The last character in the string is referenced by an index value which is the **(size of the string – 1)** or **(len(string) – 1)**

The negative index breakdown for the string “*be yourself*” assigned to *word_phrase* string variable is shown below.

```
>>> word_phrase[-2]
'l'
```

• String Slicing and Joining

The "slice" syntax is a handy way to refer to sub-parts of sequence of characters within an original string.

The syntax for string slicing is,

```
>>> ClgDept="ECE Dept Bengaluru"
>>> ClgDept[4:15:2]
>>> ClgDept[::2]
>>> ClgDept[::3]
>>>ClgDept[-9:-2]
>>>ClgDept[2:]
>>>ClgDept[:5]
>>> ClgDept[:]
>>>ClgDept[5:50]
>>>ClgDept[5:5]
```

- **Write Python Code to Determine Whether the Given String Is a Palindrome or Not, Using Slicing.**

```
Enterstr = input("Enter string: ")
if Enterstr == Enterstr[::-1]:
    print(f" palindrome")
else:
    print(f" Not a palindrome")
#print(Enterstr, Enterstr[::-1])
```

Joining Strings Using *join()* Method

string_name.join(sequence)

- Here sequence can be string or list.
- If the sequence is a string, then *join()* function inserts *string_name* between each character of the string sequence and returns the concatenated string.
- If the sequence is a list, then *join()* function inserts *string_name* between each item of list sequence and returns the concatenated string.
- It should be noted that all the items in the list should be of string type.

```
>>> DOB=["01","02","2020"] but wanted like 01-02-2020
>>> "-".join(DOB)
>>> clgg=[“ECE",“Bengaluru",“Engg"]
>>> ":".join(clgg)
>>> num="20"
>>> chrt="AbC$"
>>> password=num.join(chrt)
>>> password
'A20b20C20$'
```

- **Split Strings Using *split()* Method**
  - The *split()* method returns a list of string items by breaking up the string using the delimiter string.
  - The syntax of *split()* method is, ***string_name.split([separator [, maxsplit]])***

```
>>> datestr="01-02-2020"
>>> datestr.split("-")
['01', '02', '2020']
>>> datestr="01 02 2020"
>>> datestr.split(" ")
```

  - ['01', '02', '2020']

```
>>> datestr.split()
```

  - ['01', '02', '2020']

- **Strings Are Immutable**
- As strings are immutable, it cannot be modified. The characters in a string cannot be changed once a string value is assigned to string variable. However, you can assign different string values to the same string variable.

```
>>> clg1=“bengaluru"
>>> id(clg1)
18074304
>>> clg1=“engineering"
>>> id(clg1)
18080224
```

String Traversing

- Since the string is a sequence of characters, each of these characters can be traversed using the *for* loop.
- **Program to Demonstrate String Traversing Using the *for* Loop**

```
Example = "google"
index = 0
print(f"In the string '{Example}'")
for i in Example:
    print(f"Character '{i}' has an index value of {index}")
    index += 1
```

- **Program to Print the Characters Which Are Common in Two Strings**

```
string_1="Jasmine"
string_2="Jasan"
for letter in string_1:
    if letter in string_2:
        print(f"Character '{letter}' is found in both the strings")
```

- **Write Python Program to Count the Total Number of Vowels, Consonants and Blanks in a String**

```
user_string = input("Enter a string: ")
vowels,consonants,blanks = 0,0,0
for each_ch in user_string:
    if(each_ch=='a' or each_ch=='e' or each_ch== 'i' or each_ch== 'o' or
       each_ch == 'u'):
        vowels += 1
    elif "a" < each_ch < "z":
        consonants += 1
    elif each_ch == " ":
        blanks += 1
print(f"Total Vowels in user entered string is {vowels}")
print(f"Total Consonants in user entered string is {consonants}")
print(f"Total Blanks in user entered string is {blanks}")
```

- **Write Python Program to Calculate the Length of a String Without Using Built-In *len()* Function**

```
user_string = input("Enter a string: ")
i= 0
for i in user_string:
    count_ch += 1
print(f"The length of user entered string is {count_ch} ")
#using function
print("The length of user entered string using function",len(user_string))
```

- String Methods

>>> dir(str)

- ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',
- '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__', '__gt__',
- '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__
- mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
- '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize',
- 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format',
- 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower',
- 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
- 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
- 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'transla
- te', 'upper', 'zfill']

- String methods like *capitalize()*, *lower()*, *upper()*, *swapcase()*, *title()* and *count()* are used for conversion purpose.
- String methods like *islower()*, *isupper()*, *isdecimal()*, *isdigit()*, *isnumeric()*, *isalpha()* and *isalnum()* are used for comparing strings.
- Some of the string methods used for padding are *rjust()*, *ljust()*, *zfill()* and *center()*.
- The string method *find()* is used to find substring in an existing string. You can use string methods like *replace()*, *join()*, *split()* and *splitlines()* to replace a string in Python.

```
>>> "2020".isdigit()
>>>“bengaluru".isalpha()
>>>str3="2021“
>>> str3.isalnum()
>>> “bengaluru".islower()
>>> “BEngaluru".isupper()
>>> “BENGALURU".isupper()
>>> str1="This place is Bengaluru 560078"
>>> str1.endswith("560078")
>>> str1.startswith("T")
>>> str1.startswith("t")
>>>"cucumber".find("mber")
```

- 1. >>> fact = "Abraham Lincoln was also a champion wrestler"
- 2. >>> fact.isalnum()
- False
- 3. >>> "sailors".isalpha()
- True
- 4. >>> "2018".isdigit()
- True
- 5. >>> fact.islower()
- False
- 6. >>> "TSAR BOMBA".isupper()
- True
- 7. >>> "columbus".islower()
- True
- 8. >>> warriors = "ancient gladiators were vegetarians"
- 9. >>> warriors.endswith("vegetarians")
- True
- 10. >>> warriors.startswith("ancient")
- True

- 11. >>> warriors.startswith("A")
- False
- 12. >>> warriors.startswith("a")
- True
- 13. >>> "cucumber".find("cu")
- 0
- 14. >>> "cucumber".find("um")
- 3
- 15. >>> "cucumber".find("xyz")
- -1
- 16. >>> warriors.count("a")
- 5
- 17. >>> species = "charles darwin discovered galapagos tortoises"
- 18. >>> species.capitalize()
- 'Charles darwin discovered galapagos tortoises'
- 19. >>> species.title()
- 'Charles Darwin Discovered Galapagos Tortoises'
- 20. >>> "Tortoises".lower()
- 'tortoises'

- 21. >>> "galapagos".upper()
- 'GALAPAGOS'
- 22. >>> "Centennial Light".swapcase()
- 'cENTENNIAL lIGHT'
- 23. >>> "history does repeat".replace("does", "will")
- 'history will repeat'
- 24. >>> quote = " Never Stop Dreaming "
- 25. >>> quote.rstrip()
- ' Never Stop Dreaming'
- 26. >>> quote.lstrip()
- 'Never Stop Dreaming '
- 27. >>> quote.strip()
- 'Never Stop Dreaming‘
- 28. >>> 'ab c\n\nde fg\rkl\r\n'.splitlines()
- ['ab c', '', 'de fg', 'kl']
- 29. >>> "scandinavian countries are rich".center(40)
- 'scandinavian countries are rich'

Program using Sort()

```
# List of Integers
numbers = [1, 3, 4, 2,5,9,6,0]
numbers.sort()
print(numbers)

# List of Floating point numbers
decimalnumber = [2.01, 2.00, 3.67, 3.28, 1.68]
decimalnumber.sort()
print(decimalnumber)

# List of strings
words = ["ECE", "DSCE", "Bengaluru"]
words.sort()
print(words)
```

```
#Program 5.7: Write Python Program to Convert Uppercase Letters to
#Lowercase and Vice Versa
def case_conversion(user_string):
    convert_case = str()
    for each_char in user_string:
        if each_char.isupper():
            convert_case += each_char.lower()
        else:
            convert_case += each_char.upper()
    print(f"The modified string is {convert_case}")

input_string = input("Enter a string ")
case_conversion(input_string)
```

```
#Program 5.9: Write Python Program to Count the Occurrence of User-Entered
#Words in a Sentence
def count_word(word_occurrence, user_string):
    word_count = 0
    for each_word in user_string.split():
        if each_word == word_occurrence:
            word_count += 1
    print(f"The word '{word_occurrence}' has occurred {word_count} times")

input_string = input("Enter a string ")
user_word = input("Enter a word to count its occurrence ")
count_word(user_word, input_string)
```

```
#Program 5.6: Write Python Program That Accepts a Sentence and Calculate
#the Number of Words, Digits, Uppercase Letters and Lowercase Letters
def string_processing(user_string):
    word_count = 0
    digit_count = 0
    upper_case_count = 0
    lower_case_count = 0
    for each_char in user_string:
        if each_char.isdigit():
            digit_count += 1
        elif each_char.isspace():
            word_count += 1
        elif each_char.isupper():
            upper_case_count += 1
        elif each_char.islower():
            lower_case_count += 1
        else:
            pass
    print(f"Number of digits in sentence is {digit_count}")
    print(f"Number of words in sentence is {word_count + 1}")
    print(f"Number of upper case letters in sentence is {upper_case_count}")
    print(f"Number of lower case letters in sentence is {lower_case_count}")

user_input = input("Enter a sentence ")
string_processing(user_input)
```

- **Formatting Strings**
- *%-formatting* - **"f-strings"**
- *str.format()*.

- **Escape Sequences**
- Escape Sequences are a combination of a backslash (\) followed by either a letter or a combination of letters and digits. Escape sequences are also called as control sequences.
- The backslash (\) character is used to escape the meaning of characters that follow it by substituting their special meaning with an alternate interpretation. So, all escape sequences consist of two or more characters.

```
1. >>> print("You can break \
… single line to \
… multiple lines")
You can break single line to multiple lines
2. >>> print('print backslash \\ inside a string ')
print backslash \ inside a string
3. >>> print('print single quote \' within a string')
print single quote ' within a string
4. >>> print("print double quote \" within a string")
print double quote " within a string
5. >>> print("First line \nSecond line")
First line
Second line
6. >>> print("tab\tspacing")
tab spacing
```

7. >>> print("same\rlike")

like

8. >>> print("He\bi")

Hi

9. >>> print("\u20B9")

10. >>> print("\046")

&

11. >>> print("\x24")

$

- **Raw Strings**

- A raw string is created by prefixing the character ***r*** to the string. In Python, a raw string ignores all types of formatting within a string including the escape characters.
- 1. >>> print(r"Bible Says, \"Taste and see that the LORD is good; blessed is the man who takes refuge in him.\"")

Bible Says, \"Taste and see that the LORD is good; blessed is the man who takes refuge in him.\"

Unicodes

1. >>> unicode_string = u'A unicode \u018e string \xf1'
2. >>> unicode_string

'A unicode string ñ‘

Summary

- A string is a sequence of characters.
- To access values through slicing, square brackets are used along with the index.
- Various string operations include conversion, comparing strings, padding, finding a substring in an existing string and replace a string in Python.
- Python strings are immutable which means that once created they cannot be changed.