PYTHON

STRINGS

Till now, we have discussed numbers as the standard data types in python. In this section of the tutorial, we will discuss the most popular data type in python i.e., string.

In python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

Consider the following example in python to create a string.

```
str = "Hi Python !"
```

Here, if we check the type of the variable str using a python script

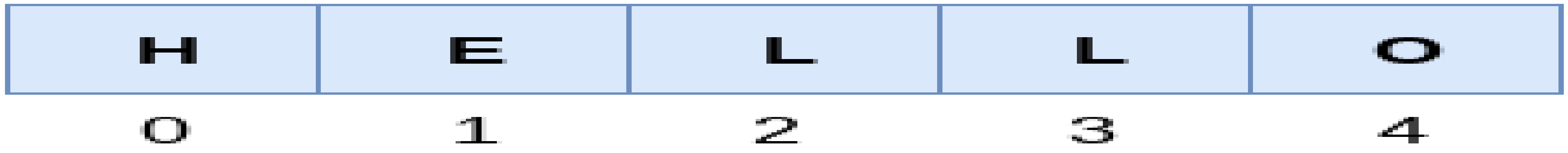
```
print(type(str))
```

In python, strings are treated as the sequence of strings which means that python doesn't support the character data type instead a single character written as 'p' is treated as the string of length 1.

STRINGS INDEXING AND SPLITTING

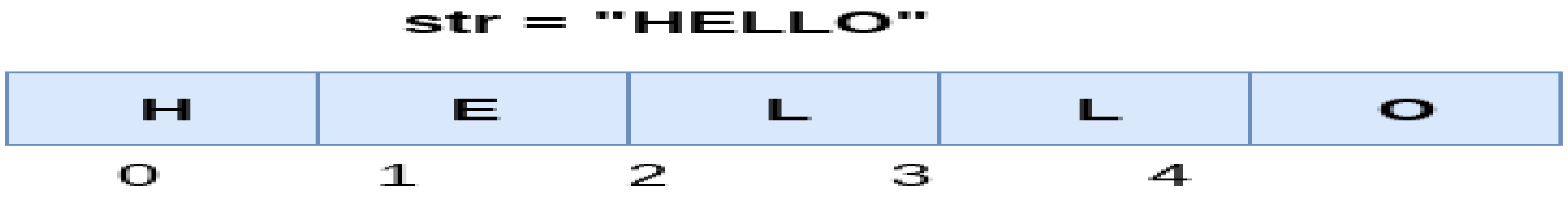
Like other languages, the indexing of the python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.

```
str = "HELLO"
```



```
str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'
```

As shown in python, the slice operator [] is used to access the individual characters of the string. However, we can use the : (colon) operator in python to access the substring. Consider the following example.



```
str[0] = 'H'          str[:] = 'HELLO'

str[1] = 'E'          str[0:] = 'HELLO'

str[2] = 'L'          str[:5] = 'HELLO'

str[3] = 'L'          str[:3] = 'HEL'

str[4] = 'O'          str[0:2] = 'HE'

                      str[1:4] = 'ELL'
```

Here, we must notice that the upper range given in the slice operator is always exclusive i.e., if str = 'python' is given, then str[1:3] will always include str[1] = 'p', str[2] = 'y', str[3] = 't' and nothing else.

REASSIGNING STRINGS

Updating the content of the strings is as easy as assigning it to a new string. The string object doesn't support item assignment i.e., A string can only be replaced with a new string since its content can not be partially replaced. Strings are immutable in python.

```
str = "HELLO"
str[0] = "h"
print(str)
```

However, in example 1, the string str can be completely assigned to a new content as specified in the following example.

```
str = "HELLO"
print(str)
str = "hello"
print(str)
```

STRING OPERATORS

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[:]	It is known as range slice operator. It is used to access the characters from the specified range.
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.

Operator	Description
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python.

EXAMPLE

```
str = "Hello"
str1 = " world"
print(str*3)
print(str+str1print(str[4])
print(str[2:4]);
print('w' in str)
print('wo' not in str1)
print(r'C://python37')
print("The string str : %s"%(str))
```

PYTHON FORMATTING OPERATOR

Python allows us to use the format specifiers used in C's printf statement. The format specifiers in python are treated in the same way as they are treated in C. However, Python provides an additional operator % which is used as an interface between the format specifiers and their values. In other words, we can say that it binds the format specifiers to the values.

```
Integer = 10;

Float = 1.290

String = “Rajkot"

print("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\nHi I am stri
ng ... My value is %s"%(Integer,Float,String));
```

PYTHON STRING FORMATTING

ESCAPE SEQUENCE

Let's suppose we need to write the text as - They said, "Hello what's going on?"- the given statement can be written in single quotes or double quotes but it will raise the **SyntaxError** as it contains both single and double-quotes.

Consider the following example to understand the real use of Python operators.

```
str = "They said, "Hello what's going on?""
print(str)
```

We can use the triple quotes to accomplish this problem but Python provides the escape sequence.

The backslash(/) symbol denotes the escape sequence. The backslash can be followed by a special character and it interpreted differently. The single quotes inside the string must be escaped. We can apply the same as in the double quotes.

```
# using triple quotes
print("""They said, "What's there?"""")

# escaping single quotes
print('They said, "What\'s going on?"')

# escaping double quotes
print("They said, \"What's going on?\"")
```

Sr.	Escape Sequence	Description
1.	\newline	It ignores the new line.
2.	\\	Backslash
3.	\'	Single Quotes
4.	\\"	Double Quotes
5.	\a	ASCII Bell
6.	\b	ASCII Backspace(BS)
7.	\f	ASCII Formfeed
8.	\n	ASCII Linefeed
9.	\r	ASCII Carriege Return(CR)
10.	\t	ASCII Horizontal Tab
11.	\ooo	Character with octal value
12.	\xHH	Character with hex value.

THE FORMAT() METHOD

The **format()** method is the most flexible and useful method in formatting strings. The curly braces {} are used as the placeholder in the string and replaced by the **format()** method argument. Let's have a look at the given an example:

```
# Using Curly braces
print("{} and {} both are the best friend".format("Devansh","Abhishek"))

#Positional Argument
print("{1} and {0} best players ".format("Virat","Rohit"))

#Keyword Argument
print("{a},{b},{c}".format(a = "James", b = "Peter", c = "Ricky"))
```

FUNCTIONS

CAPITALIZE()

Python **capitalize()** method converts first character of the string into uppercase without altering the whole string. It changes the first character only and skips rest of the string unchanged.

CENTER() METHOD

Python **center()** method alligns string to the center by filling paddings left and right of the string. This method takes two parameters, first is a width and second is a fillchar which is optional. The fillchar is a character which is used to fill left and right padding of the string.

```
# Python center() function example
# Variable declaration
str = "Hello Javatpoint"
# Calling function
str2 = str.center(20)
# Displaying result
print("Old value:", str)
print("New value:", str2)
```

COUNT()

It returns the number of occurences of substring in the specified range. It takes three parameters, first is a substring, second a start index and third is last index of the range. Start and end both are optional whereas substring is required.

```
# Python count() function example
# Variable declaration
str = "Hello Javatpoint"
str2 = str.count('t')
# Displaying result
print("occurences:", str2)
```

ENDSWITH()

Python **endswith()** method returns true of the string ends with the specified substring, otherwise returns false.

```
# Python endswith() function example

# Variable declaration

str = "Hello this is javatpoint."

isends = str.endswith(".")

# Displaying result

print(isends)
```

EXPANDTABS()

Python **expandstabs()** method replaces all the characters by sepecified spaces. By default a single tab expands to 8 spaces which can be overridden according to the requirement.

We can pass 1, 2, 4 and more to the method which will replace tab by the these number of space characters.

```
# Python endswith() function example
# Variable declaration
str = "Welcome \t to \t the \t Javatpoint."
# Calling function
str2 = str.expandtabs()
# Displaying result
print(str2)
```

FIND()

Python **find()** method finds substring in the whole string and returns index of the first match. It returns -1 if substring does not match.

```
# Python find() function example
# Variable declaration
str = "Welcome to the Javatpoint."
# Calling function
str2 = str.find("the")
# Displaying result
print(str2)
```

INDEX()

Python **index()** method is same as the find() method except it returns error on failure. This method returns index of first occurred substring and an error if there is no match found.

```
# Python index() function example
# Variable declaration
str = "Welcome to the Javatpoint."
# Calling function
str2 = str.index("at")
# Displaying result
print(str2)
```

ISALNUM()

Python **isalnum()** method checks whether the all characters of the string is alphanumeric or not. A character which is either a letter or a number is known as alphanumeric. It does not allow special chars even spaces.

```
# Python isalnum() function example
# Variable declaration
str = "Welcome"
# Calling function
str2 = str.isalnum()
# Displaying result
print(str2)
```

ISALPHA()

Python **isalpha()** method returns true if all characters in the string are alphabetic. It returns False if the characters are not alphabetic. It returns either True or False.

```
# Python isalpha() method example
# Variable declaration
str = "Javatpoint"
# Calling function
str2 = str.isalpha()
# Displaying result
print(str2)
```

ISDECIMAL()

Python **isdecimal()** method checks whether all the characters in the string are decimal or not. Decimal characters are those have base 10.

This method returns boolean either true or false.

```
# Python isdecimal() method example
# Variable declaration
str = "Javatpoint"
# Calling function
str2 = str.isdecimal()
# Displaying result
print(str2)
```

ISDIGIT()

Python **isdigit()** method returns True if all the characters in the string are digits. It returns False if no character is digit in the string.

```
# Python isdigit() method example

# Variable declaration

str = '12345'

# Calling function

str2 = str.isdigit()

# Displaying result

print(str2)
```

ISIDENTIFIER()

Python **isidentifier()** method is used to check whether a string is a valid identifier or not. It returns True if the string is a valid identifier otherwise returns False.

Python language has own identifier definition which is used by this method.

```
# Python isidentifier() method example

# Variable declaration

str = "abcdef"

# Calling function

str2 = str.isidentifier()

# Displaying result

print(str2)
```

ISLOWER()

Python string **islower()** method returns True if all characters in the string are in lowercase. It returns False if not in lowercase.

```
# Python islower() method example
# Variable declaration
str = "javatpoint"
# Calling function
str2 = str.islower()
# Displaying result
print(str2)
```

ISNUMERIC()

Python **isnumeric()** method checks whether all the characters of the string are numeric characters or not. It returns True if all the characters are true, otherwise returns False.

Numeric characters include digit characters and all the characters which have the Unicode numeric value property.

```
# Python isnumeric() method example
# Variable declaration
str = "12345"
# Calling function
str2 = str.isnumeric()
# Displaying result
print(str2)
```

ISPRINTABLE()

Python **isprintable()** method returns True if all characters in the string are printable or the string is empty. It returns False if any character in the string is Nonprintable.

```
# Python isprintable() method example
# Variable declaration
str = "Hello, Javatpoint"
# Calling function
str2 = str.isprintable()
# Displaying result
print(str2)
```

ISUPPER()

Python **isupper()** method returns True if all characters in the string are in uppercase. It returns False if characters are not in uppercase.

ISSPACE()

Python **isspace()** method is used to check space in the string. It returna true if there are only whitespace characters in the string. Otherwise it returns false. Space, newline, and tabs etc are known as whitespace characters and are defined in the Unicode character database as **Other or Separator.**

ISTITLE()

Python **istitle()** method returns True if the string is a titlecased string. Otherwise returns False.

JOIN()

Python **join()** method is used to concat a string with iterable object. It returns a new string which is the concatenation of the strings in iterable. It throws an exception TypeError if iterable contains any non-string value.

It allows various iterables like: List, Tuple, String etc.

```
# Python join() method example

# Variable declaration

str = ":"   # string

list = ['1','2','3']    # iterable

# Calling function

str2 = str.join(list)

# Displaying result

print(str2)
```

LJUST()

Python **ljust()** method left justify the string and fill the remaining spaces with fillchars. This method returns a new string justified left and filled with fillchars.

```
# Python ljust() method example
# Variable declaration
str = 'Javatpoint'
# Calling function
str = str.ljust(20)
# Displaying result
print(str)
```

LOWER()

Python **lower()** method returns a copy of the string after converting all the characters into lowercase.

LSTRIP()

Python **lstrip()** method is used to remove all leading characters from the string. It takes a char type parameter which is optional. If parameter is not provided, it removes all the leading spaces from the string.

PARTITION()

Python **partition()** method splits the string from the string specified in parameter. It splits the string from at the first occurrence of *parameter* and returns a tuple. The tuple contains the three parts before the separator, the separator itself, and the part after the separator.

```
# Python partition() method example
str = "Java is a programming language"
str2 = str.partition("is")  # Calling function
print(str2)
str2 = str.partition("Java")  # when seperate from the start
print(str2)
str2 = str.partition("language")   # when seperate is the end
print(str2)
str2 = str.partition("av")   # when seperater is a substring
print(str2)
```

REPLACE()

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

```
# Python replace() method example
# Variable declaration
str = "Java is a programming language"
# Calling function
str2 = str.replace("Java","C")
# Displaying result
print("Old String: \n",str)
print("New String: \n",str2)
```

RFIND()

Python **rfind()** method finds a substring in in the string and returns the highest index. It means it returns the index of most righmost matched subtring of the string. It returns -1 if substring not found.

RINDEX()

Python **rindex()** method works same as the rfind() method except it throws error ValueError. This method throws an exception **ValueError** if the substring is not found. The syntax is given below.

RJUST()

Python **rjust()** method right justify the string and fill the remaining spaces with fillchars. This method returns a new string justified right and filled with fillchars.

```
# Python rjust() method example
# Variable declaration
str = 'Javatpoint'
# Calling function
str = str.rjust(20)
# Displaying result
print(str)
```

RSTRIP()

Python **rstrip()** method removes all the trailing characters from the string. It means it removes all the specified characters from right side of the string. If we don't specify the parameter, It removes all the whitespaces from the string. This method returns a string value.

```
# Python rstrip() method example
# Variable declaration
str = "Java and C#"
# Calling function
str2 = str.rstrip('#')
# Displaying result
print("Old string: ",str)
print("New String: ",str2)
```

```
# Python rstrip() method example
# Variable declaration
str = "Java and C# "
# Calling function
str2 = str.rstrip()
# Displaying result
print("Old string: ",str)
print("New String: ",str2)
```

SPLIT()

Python **split()** method splits the string into a comma separated list. It separates string based on the separator delimiter. This method takes two parameters and both are optional. It is described below.

```
# Python split() method example
# Variable declaration
str = "Java is a programming language"
# Calling function
str2 = str.split()
# Displaying result
print(str)
print(str2)
```

SPLITLINES()

Python **splitlines()** method splits the string based on the lines. It breaks the string at line boundaries and returns a list of splitted strings. Line breakers can be a new line (\n), carriage return (\r) etc. A table of line breakers are given below which split the string.

```
# Python splitlines() method example
# Variable declaration
str = "Java is a programming language"
# Calling function
str2 = str.splitlines() # returns a list having single element
print(str)
print(str2)
str = "Java \n is a programming \r language"
str2 = str.splitlines() # returns a list having splitted elements
print(str2)
```

```
# Python splitlines() method example
# Variable declaration
str = "Java \n is a programming \r language"
# Calling function
str2 = str.splitlines(True) # returns a list having splitted elements
print(str2)
```

```
# Python splitlines() method example
# Variable declaration
str = "Java \n is a programming \r language for \r\n  software development"
# Calling function
str2 = str.splitlines() # returns a list having splitted elements
# Displaying result
print(str2)
# getting back list to string
print("".join(str2)) # now it does not contain any line breaker character
```

STARTSWITH()

Python startswith() method returns either True or False. It returns True if the string starts with the prefix, otherwise False. It takes two parameters start and end. Start is a starting index from where searching starts and end index is where searching stops.

SWAPCASE()

Python swapcase() method converts case of the string characters from uppercase to lowercase and vice versa. It does not require any parameter and returns a string after case conversion.

TRANSLATE()

Python translate() method a string in which each character has been mapped through the given translation table. We can use maketrans() method to create a translation map from character-to-character mappings in different formats.

```
# Python translate() method
# Declaring table and variables
table = { 97 : 103, 111 : 112, 117 : None }
str = "Hello javatpoint"
# Calling function
str2 = str.translate(table)
# Displaying result
print(str2)
```

UPPER()

Python upper() method converts all the character to uppercase and returns a uppercase string.

ZFILL()

Python zfill() method fills the string at left with 0 digit to make a string of length width. It returns a string contains a sign prefix + or - before the 0 digit.

```
# Python zfill(width) method
# Declaring variables
text = "Zfill Example"
# Calling Function
str2 = text.zfill(20)
# Displaying result
print(str2)
```

CONTINUE IN NEXT UNIT .....