Dr. D. Y. PATIL INSTITUTE OF TECHNOLOGY, PIMPRI, PUNE-18
Department of First Year Engineering
Programming and Problem Solving
Unit- 4 Notes

Unit IV: Strings

4.1 Strings and Operations

Q 1. What is String? With the help of example explain how we can create string variable in python.

Ans:

- Strings data type is sequence of characters, where characters could be letter, digit, whitespace or any other symbol.

a. Creation of Strings:

- Strings in Python can be created using single quotes or double quotes or even triple quotes.
- Example:

```
string1 = 'Welcome'        # Creating a String with single Quotes
string2 = "Welcome"        # Creating a String with double Quotes
string3 = '''Welcome'''    # Creating a String with Triple Quotes
```

b. Accessing strings:

- In Python, individual characters of a String can be accessed by using the method of Indexing or range slice method [:].
- Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on.

String	**W**	**E**	**L**	**C**	**O**	**M**	**E**
Indexing	0	1	2	3	4	5	6
Negative Index	-7	-6	-5	-4	-3	-2	-1

- **Example:**

```
string = 'Welcome'
print(string[0])          #Accessing string with index
print(string[1])
print(string[2])
print(srting[0:2])        #Accessing string with range slice method
```

Output:

```
w
e
l
wel
```

c. Deleting/Updating from a String:

- In Python, updating or deletion of characters from a String is not allowed as Strings are immutable.
- Although deletion of entire String is possible with the use of a built-in **del keyword**.
- Example:

```
string='welcome'
del string
```

Q 2. Explain Operations on string.

Ans:

Operation	**Description**	**Example**	**Output**
Concatenation(+)	-It joins two strings and returns new list.	x="Good" y="Morning" z=x+y print(z)	Good Morning

Append (+=)	-Append operation adds one string at the end of another string	x="Good" y="Morning" x+=y print(x)	Good Morning
Repetition(*)	-It repeats elements from the strings n number of times	x="Hello" y=x*2 print(y)	HelloHello
Slice []	- It will give you character from a specified index.	x="Hello" print(x[1])	e
Range slice[:]	-It will give you characters from specified range slice.	x="Hello" print(x[0:2])	He

4.2 Strings are immutable

Q 3. Python strings are immutable. Comment on this.

Ans:

- Python Strings are immutable, which means that once it is created it cannot be changed.
- Whenever you try to change/modify an existing string, a new string is created.
- As every object (variable) is stored at some address in computer memory.
- The **id()** function is available in python which returns the address of object(variable) in memory. With the help of memory locations/address we can see that for every modification, string get new address in memory.
- Here is the example to demonstration the address change of string after modification.

```
# prints string1 and its address
string1="Good"
print("String1 value is: ",string1)
print("Address of string1 is: ",id(string1)
```

```
# prints string2 and its address
string2="Morning"
print("String2 value is: ",string2)
print("Address of string2 is: ",id(string2)

#appending string1 to string2
string1+= string2
print("String1 value is: ",string1)
print("Address of string1 is: ",id(string1)
```

Output:

```
String1 value is: Good
Address of String1 is: 1000

String2 value is: Morning
Address of String1 is: 2000

String1 value is: GoodMorning
Address of String1 is: 3000
```

- From the above output you can see string1 has address 1000 before modification. In later output you can see that string1 has new address 3000 after modification.
- It is very clear that, after some operations on a string new string get created and it has new memory location. This is because strings are unchangeable/ immutable in nature. Modifications are not allowed on string but new string can be created at new address by adding/appending new string.

4.3 Strings formatting operator

Q 4. Explain various ways of string formatting with example.

Ans:

- In python, % sign is a string formatting operator.

- The % operator takes a format string on the left and the corresponding values in a tuple on the right.
- The format operator, % allows users to replace parts of string with the data stored in variables.
- The syntax for string formatting operation is:

"<format>" % (<values>)

- The statement begins with a *format string* consisting of a sequence of characters and *conversion specification*.
- Following the format string is a % sign and then a set of values, one per conversion specification, separated by commas and enclosed in parenthesis.
- If there is single value then parenthesis is optional.
- Following is the list of format characters used for printing different types of data:

Format Symbol	Purpose
%c	Character
%d or %i	Signed decimal integer
%s	String
%u	Unsigned decimal integer
%o	Octal integer
%x or %X	Hexadecimal integer
%e or %E	Exponential notation
%f	Floating point number
%g or %G	Short numbers in floating point or exponential notation

Example: Program to use format sequences while printing a string.

```
name="Amar"
age=8
print("Name = %s and Age = %d" %(name,age))
print("Name = %s and Age = %d" %("Ajit",6))
```

Output:

Name = Amar and Age = 8

Name = Ajit and Age = 6

In the output, we can see that %s has been replaced by a string and %d has been replaced by an integer value.

4.4 Built-in String methods and functions

Q 5. List and explain any 5 string methods.

Or

Q. Explain the use of _____ ( ) with the help of an example.

Ans.

Sr. No.	Function	Usage	Example
1	capitalize()	This function is used to capitalize first letter of string.	str="hello" print(str.capitalize()) output: Hello
2	isalnum()	Returns true if string has at least 1 character and every character is either a number or an alphabet and False otherwise.	message="JamesBond007" print(message.isalnum()) output: True
3	isalpha()	Returns true if string has at least 1 character and every character is an alphabet and False otherwise.	message="JamesBond007" print(message.isalpha()) output: False
4	isdigit()	Returns true if string has at least 1 character and every character is a digit and False otherwise.	message="007" print(message.isdigit()) output: True
5	islower()	Returns true if string has at least 1	message="Hello"

		character and every character is a lowercase alphabet and False otherwise.	print(message.islower()) output: False
6	isspace()	Returns true if string contains only white space character and False otherwise.	message=" " print(message.isspace()) output: True
7	isupper()	Returns true if string has at least 1 character and every character is an uppercase alphabet and False otherwise.	message="HELLO" print(message.isupper()) output: True
8	len(string)	Returns length of the string.	str="Hello" print(len(str)) output: 5
9	zfill(width)	Returns string left padded with zeros to a total of width characters. It is used with numbers and also retains its sign (+ or -).	str="1234" print(str.zfill(10)) output: 0000001234
10	lower()	Converts all characters in the string into lowercase.	str="Hello" print(str.lower()) output: hello
11	upper()	Converts all characters in the string into uppercase.	str="Hello" print(str.upper()) output: HELLO
12	lstrip()	Removes all leading white space in string.	str=" Hello" print(str.lstrip()) output: Hello

13	rstrip()	Removes all trailing white space in string.	str=" Hello " print(str.rstrip()) output: Hello
14	strip()	Removes all leading white space and trailing white space in string.	str=" Hello " print(str.strip()) output: Hello
15	max(str)	Returns the highest alphabetical character (having highest ASCII value) from the string str.	str="hello friendz" print(max(str)) output: z
16	min(str)	Returns the lowest alphabetical character (having lowest ASCII value) from the string str.	str="hellofriendz" print(min(str)) output: d
17	replace(old,new[, max])	Replaces all or max (if given) occurrences of old in string with new.	str="hello hello hello" print(str.replace("he","Fo")) output: Follo Follo Follo
18	title()	Returns string in title case.	str="The world is beautiful" print(str.title()) output: The World Is Beautiful
19	swapcase()	Toggles the case of every character (uppercase character becomes lowercase and vice versa).	str="The World Is Beautiful" print(str.swapcase()) output: tHE wORLD iS bEAUTIFUL
20	split(delim)	Returns a list of substrings	str="abc,def, ghi,jkl"

		separated by the specified delimiter. If no delimiter is specified then by default it splits strings on all whitespace characters.	print(str.split(',')) output: ['abc', 'def', ' ghi', 'jkl']
21	join(list	It is just the opposite of split. The function joins a list of strings using delimiter with which the function is invoked.	print('-'.join(['abc', 'def', ' ghi', 'jkl'])) output: abc-def- ghi-jkl
22	isidentifier()	Returns true if the string is a valid identifier.	str="Hello" print(str.isidentifier()) output: True
23	enumerate(str)	Returns an enumerate object that lists the index and value of all the characters in the string as pairs.	str="Hello World" print(list(enumerate(str))) output: [(0, 'H'), (1, 'e'), (2, 'l'), (3, 'l'), (4, 'o'), (5, ' '), (6, 'W'), (7, 'o'), (8, 'r'), (9, 'l'), (10, 'd')]

4.5 Slice operation

Q 6. What is slice operation? Explain with example.

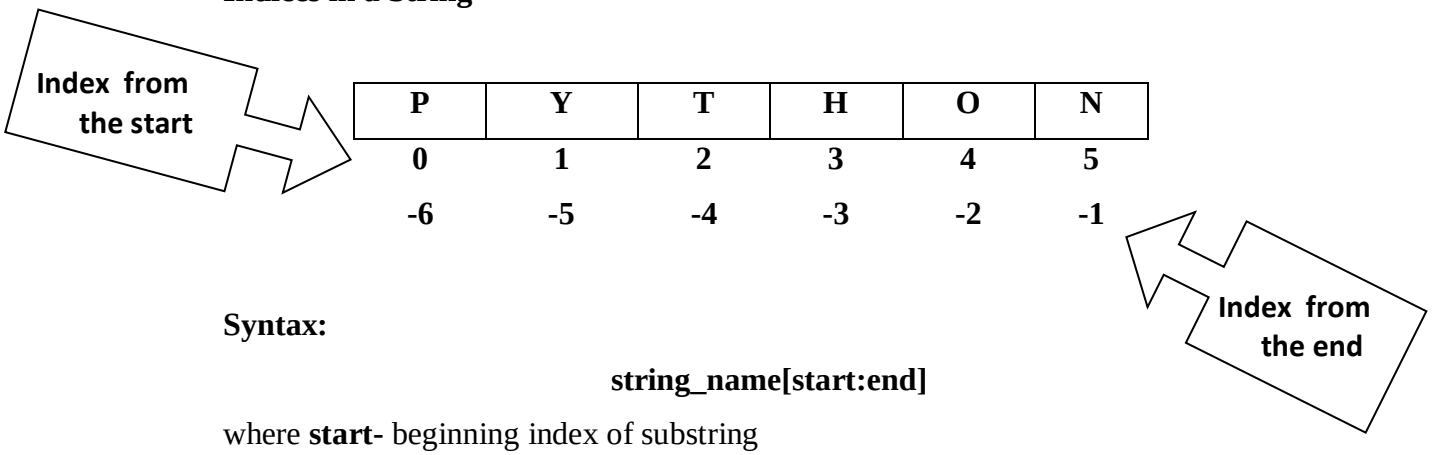
Ans.

Slice: A substring of a string is called a slice.

A slice operation is used to refer to sub-parts of sequences and strings.

Slicing Operator: A subset of a string from the original string by using [] operator known as Slicing Operator.

Indices in a String



Syntax:

string_name[start:end]

where **start-** beginning index of substring

end -1 is the index of last character

Program to demonstrate slice operation on string objects

```
str="PYTHON"
print("str[1:5]= ", str[1:5])     #characters start at index 1 and extending upto  index 4
                                  # but not including index 5
print("str[ :6]= ", str[ :6])     # By defaults indices start  at index 0
print("str[1: ]= ", str[1: ])     # By defaults indices ends  upto last index
print("str[ : ]= ", str[ : ])     # By defaults indices start at index 0 and end upto last
                                  #character in the string
#negative index
print("str[-1]= ", str[ -1])      # -1 indicates last character
print("str[ :-2 ]= ", str[ : -2]) #all characters upto -3
print("str[ -2: ]= ", str[ -2: ]) #characters from index -2
print("str[-5 :-2 ]= ", str[ -5: -2])   # characters from index -5 upto character index -3
```

OUTPUT

```
str[1:5]= YTHO
str[ :6]= PYTHON
str[1: ]= YTHON
str[ : ]= PYTHON
```

str[-1]= N

str[:-2]= PYTH

str[-2:]= ON

str[-5 :-2]= YTH

Specifying Stride while Slicing Strings

- In the slice operation, you can specify a third argument as the stride, which refers to the number of characters to move forward after the first character is retrieved from the string.
- The default value of stride is 1, i.e. where value of stride is not specified, its default value of 1 is used which means that every character between two index number is retrieved.

  Program to use slice operation with stride

  str=" Welcome to the world of Python"

  print("str[2: 10]= ", str[2:10]) #default stride is 1

  print("str[2:10:1]= ", str[2:10:1]) #same as stride=1

  print("str[2:10:2]= ", str[2:10:2]) #skips every alternate character

  print("str[2:10:4]= ", str[2:10:4]) #skips every fourth character

  OUTPUT

  str[2: 10]=lcome to

  str[2: 10]= lcome to

  str[2:10:2]=loet

  str[2:10:4]=l

- Whitespace characters are skipped as they are also part of the string.
-

4.6 ord() and chr() functions

Q 7.Write a short note on ord() and chr() functions

Ans.

- The ord() function return the ASCII code of the character

- The chr() function returns character represented by a ASCII number.

ch='R' print(ord(ch)) **OUTPUT** 82	print(chr(82)) **OUTPUT** R	print(chr(112)) **OUTPUT** p	print(ord('p')) **OUTPUT** 112

4.7 in and not in operators

Q 8.Write a short note on in and not in operators

OR

Q.With the help of example, explain significance of membership operators.

Ans.

- *in* and *not in* operators can be used with strings to determine whether a string is present in another string. Therefore the *in* and *not in* operator is known as membership operators.

- For example:

str1=" Welcome to the world of Python!!!"
str2="the"
if str2 in str1:
print("found")
else:
print("Not found")
 	
OUTPUT
Found
	str1=" This is very good book"
	str2="best"
	if str2 in str1:
	print("found")
	else:
	print("Not found")
	
	OUTPUT
	Not found

- You can also use *in* and *not in* operators to check whether a character is present in a word.
- For example:

'u' in "starts"	'v' not in "success"

OUTPUT	OUTPUT
False	True

4.8 Comparing strings

Q 9. Explain string comparison operator with example?

Ans.

- Python allows us to combine strings using relational (or comparison) operators such as >, <, <=,>=, etc.
- Some of these operators along with their description and usage are given as follows:

Operator	Description	Example
==	If two strings are equal, it returns True.	>>>"AbC"=="AbC" True
!= or <>	If two strings are not equal, it returns True.	>>>"AbC"!="Abc" True
>	If the first string is greater than the second, it returns True.	>>>"abc">"Abc" True
<	If the second string is greater than the first, it returns True.	>>>"abC"<"abc" True
>=	If the first string is greater than or equal to the second, it returns True.	>>>"aBC">=""ABC" True
<=	If the second string is greater than or equal to the first, it returns True.	>>>"ABc"<="ABc" True

- These operators compare the strings by using ASCII value of the characters.
- The ASCII values of A-Z are 65-90 and ASCII code for a-z is 97-122.
- For example, book is greater than Book because the ASCII value of 'b' is 98 and 'B' is 66.

String Comparison Programming Examples: ***(Any one)***

- There are different ways of comparing two strings in Python programs:

- Using the **==(equal to)** operator for comparing two strings:
  - If we simply require comparing the values of two variables then you may use the '==' operator.
  - If strings are same, it evaluates to True, otherwise False.

  - Example1:

```
first_str='Kunal works at Phoenix'
second_str='Kunal works at Phoenix'
print("First String:", first_str)
print("Second String:", second_str)
#comparing by ==
if first_str==second_str:
        print("Both Strings are Same")
else:
        print("Both Strings are Different")
```

Output:

```
First String: Kunal works at Phoenix
Second String: Kunal works at Phoenix
Both Strings are Same
```

  - Example2(Checking Case Sensitivity):

```
first_str='Kunal works at PHOENIX'
second_str='Kunal works at Phoenix'
print("First String:", first_str)
print("Second String:", second_str)
#comparing by ==
if first_str==second_str:
        print("Both Strings are Same")
else:
        print("Both Strings are Different")
```

Output:

First String: Kunal works at PHOENIX

Second String: Kunal works at Phoenix

Both Strings are Different

- Using the **!=(not equal to)** operator for comparing two strings:
  - The != operator works exactly opposite to ==, that is it returns true is both the strings are not equal.
  - Example:

```
first_str='Kunal works at Phoenix'
second_str='Kunal works at Phoenix'
print("First String:", first_str)
print("Second String:", second_str)
#comparing by !=
if first_str!=second_str:
        print("Both Strings are Different")
else:
        print("Both Strings are Same")
```

output:

First String: Kunal works at Phoenix

Second String: Kunal works at Phoenix

Both Strings are Same

- Using the **is** operator for comparing two strings:
  - The **is** operator compares two variables based on the object id and returns True if the two variables refer to the same object.
  - Example:

```
name1=”Kunal”
name2=”Shreya”
print(“name1:”,name1)
```

```
print("name2:",name2)
print("Both are same",name1 is name2)
name2="Kunal"
print("name1:",name1)
print("name2:",name2)
print("Both are same",name1 is name2)
```

- Output:

```
name1=Kunal
name2=Shreya
Both are same False
name1=Kunal
name2=Kunal
Both are same True
```

- In the above example, name2 gets the value of Kunal and subsequently name1 and name2 refer to the same object.
-

4.9 Iterating strings

Q. No.10 How to iterate a string using:

Ans.

i) for loop with example

ii) while loop with example

Ans.

- String is a sequence type (sequence of characters).
- We can iterate through the string using:

i) for loop:

- for loop executes for every character in str.
- The loop starts with the first character and automatically ends when the last character is accessed.
- Example-

```
str="Welcome to python"
for i in str:
print(i,end=' ')
```

Output-

W e l c o m e t o P y t h o n

ii) while loop:

- We can also iterate through the string using while loop by writing the following code.
- Example-

```
message=" Welcome to python"
index=0
while index < len(message):
        letter=message[index]
        print(letter,end=' ')
        index=index+1
```

Output-

W e l c o m e t o P y t h o n

- In the above program the loop traverses the string and displays each letter.
- The loop condition is index < len(message), so the moment index becomes equal to the length of the string, the condition evaluates to False, and the body of the loop is not executed.
- Index of the last character is len(message)-1.

4.10 The string module

Q. No. 11 Write a note on string module?

- The string module consists of a number of useful constants, classes and functions.
- These functions are used to manipulate strings.

- String Constants: Some constants defined in the string module are:
  - string.ascii_letters: Combination of ascii_lowecase and ascii_uppercase constants.
  - string.ascii_lowercase: Refers to all lowercase letters from a-z.
  - string.ascii_uppercase: Refers to all uppercase letters from A-Z.
  - string.lowercase: A string that has all the characters that are considered lowercase letters.
  - string.uppercase: A string that has all the characters that are considered uppercase letters.
  - string.digits:Refers to digits from 0-9.
  - string.hexdigits: Refers to hexadecimal digits,0-9,a-f, and A-F.
  - string.octdigits: Refers to octal digits from 0-7.
  - string.punctuation: String of ASCII characters that are considered to be punctuation characters.
  - string.printable: String of printable characters which includes digits, letters, punctuation, and whitespaces.
  - string.whitespace: A string that has all characters that are considered whitespaces like space, tab, return, and vertical tab.
- Example: (Program that uses different methods such as upper, lower, split, join, count, replace, and find on string object)

```
str="Welcome to the world of Python"
print("Uppercase-", str.upper())
print("Lowercase-", str.lower())
print("Split-", str.split())
print("Join-", '-'.join(str.split()))
print("Replace-",str.replace("Python","Java"))
print("Count of o-", str.count('o'))
print("Find of-",str.find("of"))
```