Because learning changes everything.®

TITLE: Python Programming: Problem Solving, Packages and Libraries

Edition

Lecture PPT Chapter 7: Strings

Learning Objectives

- Create/initialize and access “elements”, that is, “characters” of a string.
- Recognize that a string is a “sequence” and has an “index”, and also understand how “negative index” works in strings.
- Explain “escape sequences”, string “slicing” and “extended slicing”.
- Compare strings and understand that “comparison” generates bool, that is, True/ False.
- Use some common string functions and string methods.
- Appreciate “string format” methods.

Some characteristics of strings in Python

- Strings in Python are built in types.
- Strings are composed of characters. So it is a type, which is composed of smaller pieces. You may, at times, deal with a string as a single entity and at other times may need to access its parts.
- A string is a sequence. A sequence is an “Ordered collection”. If, in a collection, the order does not matter, then it is called an “Unordered collection”.
- A string being an “ordered collection”, therefore, in Python, ‘abc’ is not the same as ‘bac’.
- A string is an object and has its attributes and methods.

Creating strings in Python

- Using single quotes
- Using double quotes
- Using “triple single” or “triple double” quotes
- Using the str(object) builtin function

(The text book provides details along with examples)

String indexing (Some important points)

- In Python, the individual characters of a string can be accessed by their index, which starts from 0 and ends at one less than the length of the string.
- Thus, if the length of a string is n, then its index will be from 0 to n-1. (This is similar to C++).
- However, in Python, you can access the individual characters of a string using negative index also.
- The last character in the string has a negative index of -1 and the first character (for a string of n characters) has an index of –n.
- The indexing applies to string variables as well as string literals.

Positive and negative index

As pointed out earlier, a string can have both positive and negative index.

The following figure (from book) shows how the positive and negative index works for a string:

“Hello World!”

0	1	2	3	4	5	6	7	8	9	10	11	← INDEX
H	e	L	l	o		W	o	r	l	d	!	← String sequence
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	← Negative Index

FIGURE 7.1 The positive and negative index of a string sequence

Some extra points relating to string index

- If a string has n characters, then the index can vary from 0 to n-1 or from -1 to –n.
- If index is greater than n-1 or less than –n, it will throw an error.
- Index can be used for both string variable and for string literal.
- Note that there is no dot (.) between the string variable (Or literal) and the index. (For instance, myStr.[0] or ‘Hello World!’.[0] is incorrect syntax. So to access the first element of a string literal like say ‘cat’ is ‘cat’[0] and not ‘cat’.[0]).

Special character “\” and escape sequences Part 1

- In Python, the backslash, that is, “\” is a “special character”.
- This special character has “special meaning” in a “literal string”. (A literal string in Python is any string surrounded by quotes).
- So the backslash, that is, “\” has “special character” only when it is embedded in a “literal string”.

Special character "\" and escape sequences Part 2

When embedded in a literal string, a special character can do two different things:

- It can convert an ordinary character into a character with special meaning. For instance, the string "t" normally would represent the character t. But if it is preceded by a "\", that is, if you have "\t" then it does not represent the character "t", but rather, it represents the "non-printable" character "tab". There are many other "non-printable" characters, which can be similarly represented. For instance, "\n" represents "new line".
- It can convert characters with "special meaning" into ordinary characters. For instance, a quote (say a single quote → ' or a double quote → ") would represent a beginning or an end of a "literal string". But if a single quote or a double quote is preceded by a backslash, then it does not represent the beginning or end of a string. Rather, it simply represents a quote. Therefore, inside a literal string, the backslash, that is, '\' acts as an "escape sequence". So if a special character, such as a quotation mark is preceded by a backslash, then it "loses its special meaning". This "loss of special meaning" is called "escape sequence".

Some common “escape sequences”.

TABLE 7.1 Some common escape sequences

Escape	What it does	Explanation
\\	Backslash	A backslash would normally have special meaning, but when preceded by another backslash, it acts just like an ordinary backslash. Here, the first backslash escapes the second backslash.
\'	Single quote	A single quote is also a special character. Ordinarily, it would represent the beginning or end of a literal string. But when preceded by a backslash, it acts like an ordinary single quote and does not signal the beginning or the end of a string. Here also, the backslash escapes the quotation mark.
\"	Double quote	(Similar to above explanation)
\n	New line	Character ‘n’ is an ordinary character, but when it is preceded by a backslash, it acquires a special meaning and represents a new line. So here, the backslash does not escape but rather gives special meaning to character n and is used to represent the non-printable character new line.
\t	Tab	Character ‘t’ is an ordinary character, but when it is preceded by a backslash, it acquires a special meaning and represents a tab. So here, the backslash does not escape but rather gives special meaning to character ‘t’ and is used to represent the non-printable character ‘tab’.

“Traversing a string”

“Traversing a string” literally means going over the characters of a string “one by one”. Here, one can use the “index” of each character in the string. Some common ways for doing this are:

- **Traversing a string using ‘for’ loop**
- **Traversing a string using while loop**
- **Traversing a string using ‘for’ loop with the range() function**

String operations

Some common operators which can be used on strings are:

- **“+”,**
- **“*”,**
- **“in”,**
- **“not in”**
- **“slice[n:m]”**

Similarly “relational operators” can also be used to “compare” strings.

Summary of string slicing

The basics of slicing can be summarized as follows:

- A string (which is a sequence object), can be sliced using a pair of indices in square brackets and separated by a colon.
- The index on the left is lower bound and is included in the slice.
- The index on the right is the upper bound and is excluded from the slice.
- Slicing creates a new object, that is, a new string.
- Further, you can exclude (writing in square brackets) either the upper bound or the lower bound or both.
  - If lower bound is excluded, it defaults to 0. (For example myStr[:6] is same as myStr[0:6])
  - If upper bound is excluded it defaults to the index of the last character of the string (Which is also -1). For example myStr[1:] is same as myStr[1:-1]
- If both lower and upper bound are excluded, it copies the entire string and a new string which is the same as the original, is created.

String “extended slicing”

- Extended slice is of format myStr[x: y: z].
- x is start index, y is stop index and z is the step or stride.
- Any of the three can be excluded.
- If z is excluded it defaults to +1.
- As before, if x is excluded, it defaults to 0 and if y is excluded, it defaults to index of the last character of the string.
- Note that as before, the character at index x is always included and the character at index y is never included.

Example of etended string slicing

Given below is an example script and a figure showing how extended string slicing works on a string:

```
myLetters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
>>> myLetters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> myLetters[1:25:2]
'BDFHJLNPRTVX'
```

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
myLetters[1:25:2]																									
	B		D		F		H		J		L		N		P		R		T		V		X		

FIGURE 7.4 How “”String extended slicing works

Some common string methods

str.capitalize() where str is a string

str.count(sub[, start[, stop]])

str.find(pat[, start[, stop]])

str.isalnum()

str.isalpha()

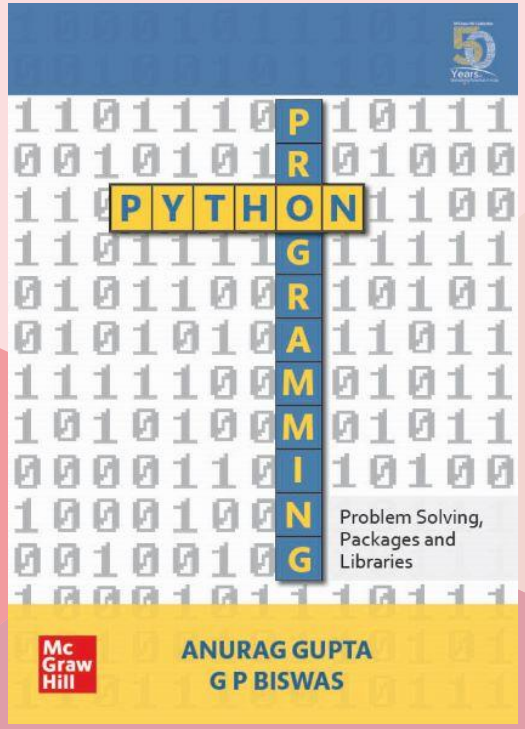
str.isdigit()

A short note on string module

- **Note that the function str() is different from the string module in Python.**
- **In Python, the string is a module, which needs to be imported using the import statement.**
- **In fact, with Python 3.x you don't need to import the string module since most of the functionalities provided by this module are already available in the string type.**
- **However, the string module has a number of built-in constants, which can be used.**

Example code shows how the string module may be used

```
>>>import string
>>> string.ascii_uppercase      # Gives uppercase letters
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.ascii_lowercase      # Gives lowercase letters
'abcdefghijklmnopqrstuvwxyz'
>>> string.ascii_letters # Gives both upper and lower
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.digits               # Gives all digits
'0123456789'
>>> string.hexdigits      # Gives all hex digits ie 0-9 and A-F
'0123456789abcdefABCDEF'
>>> string.octdigits            # Gives all oct digits ie 0-7
'01234567'
>>> string.punctuation          # Gives all punctuation marks
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
>>> string.whitespace           # Gives all white spaces
including tabs and newline
' \t\n\r\x0b\x0c'
>>>
```


PYTHON
PROGRAMMING
Problem Solving, Packages and Libraries
Mc Graw Hill
ANURAG GUPTA
G P BISWAS

Because learning changes everything.®

Thank You!

For any queries or feedback contact us at:

support.india@mheducation.com

1800-103-5875

www.mheducation.co.in