PCC-CS393: IT Workshop (Python)

Unit 4 – String Manipulation

Sourav Das (Ph.D. Scholar @ IIITK, M.Tech (CSE), MCA), MACM

Assistant Professor

Department of CSE – AIML and IOT CYS Including BCT

Future Institute of Technology, Kolkata

Contents

- **Introduction,**
- **Accessing Strings,**
- **Basic String Operations,**
- **String Slices,**
- **String Indexing,**
- **String Functions.**

Introduction

- Strings can be used to represent just about anything that can be encoded as text:
  - ₪ Symbols and words,
  - ₪ Contents of text files loaded into memory,
  - ₪ Internet addresses,
  - ₪ Program Statements, and so on.

- They can also be used to hold the absolute binary values of bytes, and multibyte Unicode text used in internationalized programs.

Introduction

- We have already used strings in other languages, too.
- Python's strings serve the same role as character arrays in languages such as C, but they are a somewhat higher-level tool than the arrays.
- Unlike in C, in Python, strings come with a powerful set of processing tools.
- Also unlike languages such as C, Python has no distinct type for individual characters; instead, we just use one-character strings.

Accessing Strings

Operation	Interpretation
`S = ''`	Empty string
`S = "spam's"`	Double quotes, same as single
`S = 's\np\ta\x00m'`	Escape sequences
`S = """..."""`	Triple-quoted block strings
`S = r'\temp\spam'`	Raw strings
`S = b'spam'`	Byte strings in 3.0
`S = u'spam'`	Unicode strings in 2.6 only
`S1 + S2`	Concatenate, repeat
`S * 3`	
`S[i]`	Index, slice, length
`S[i:j]`	
`len(S)`	
`"a %s parrot" % kind`	String formatting expression
`"a {0} parrot".format(kind)`	String formatting method in 2.6 and 3.0
`S.find('pa')`	String method calls: search,
`S.rstrip()`	remove whitespace,
`S.replace('pa', 'xx')`	replacement,
`S.split(',')`	split on delimiter,
`S.isdigit()`	content test,
`S.lower()`	case conversion,
`S.endswith('spam')`	end test,
`'spam'.join(strlist)`	delimiter join,
`S.encode('latin-1')`	Unicode encoding, etc.
`for x in S: print(x)`	Iteration, membership

Accessing Strings

- Python strings are ***immutable***, which means they cannot be changed after they are created (Java strings also use this immutable style).

- Since strings can't be changed, hence we construct ***new strings*** as we go to represent computed values.

- For instance, the expression ***("Hello" + "Python")*** takes in the 2 strings ***"Hello"*** and ***"Python"***, and builds a new string ***"HelloPython"***.

Accessing Strings

- Characters in a string can be accessed using the standard ***[ ]*** syntax, and like Java and C++, Python uses zero-based indexing, so if s is "***hello***", then ***s[1]*** is "***e***".

- If the index is out of bounds for the string, Python raises an error.
-
- The Python style is to halt if it can't tell what to do, rather than just make up a default value.

- The useful ***slice*** syntax also works to extract any substring from a string.

Accessing Strings

- The ***[ ]*** syntax and the ***len( )*** function actually work on any sequence type, like strings, lists, etc.

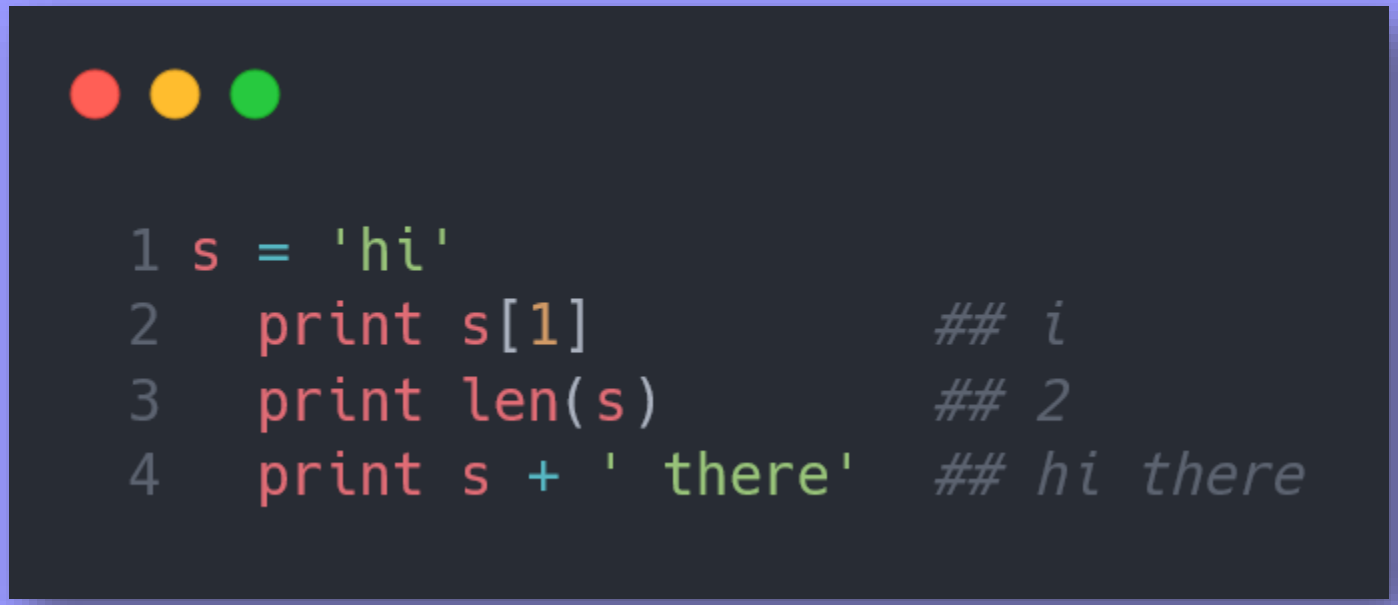
- Python tries to make its operations work consistently across different types.

- In Python, we don't use ***len*** as a variable name to avoid blocking out the ***len( )*** function.

- The '***+***' operator can concatenate two strings. Next, we will see that, variables are not pre-declared, we just assign to them.

Accessing String

❑ **Sample syntax for string declaration and accessing in Python:**



```
1 s = 'hi'
2   print s[1]              ## i
3   print len(s)            ## 2
4   print s + ' there'      ## hi there
```

Accessing Strings

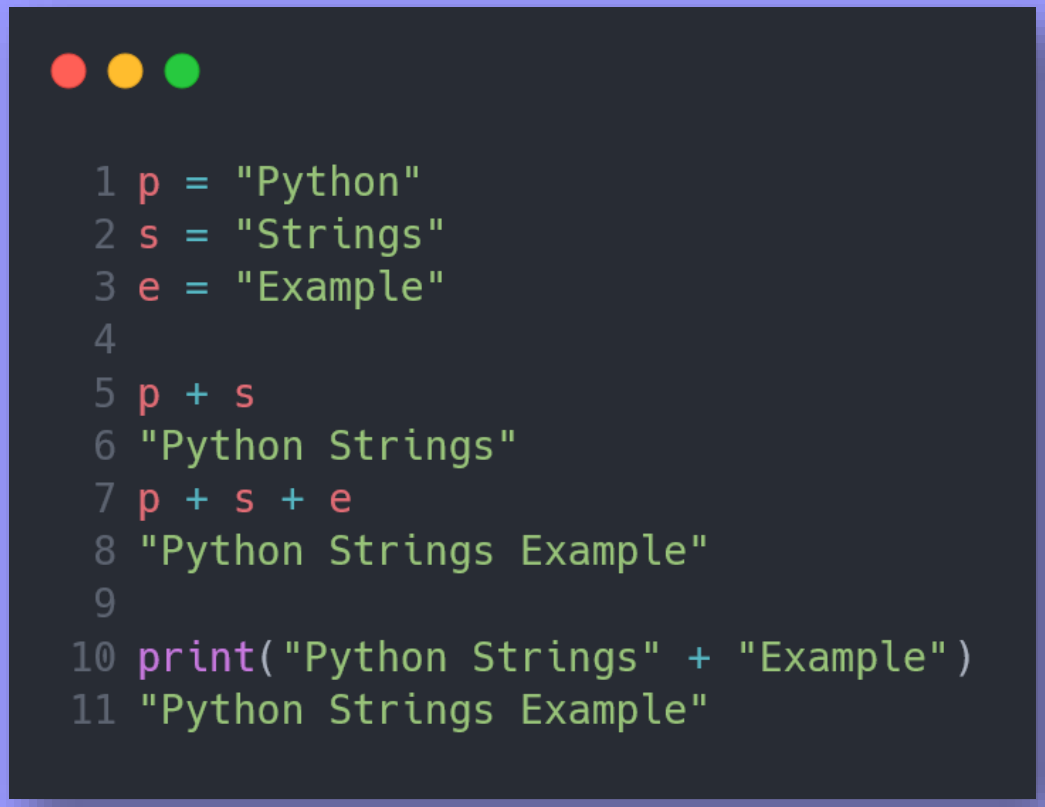
- Unlike Java, the "***+***" does not automatically convert numbers or other types to string form.
- The ***str()*** function converts values to a string form so they can be combined with other strings.

```
1 pi = 3.14
2   text = 'The value of pi is ' + pi      #NO, does not work
3   text = 'The value of pi is '  + str(pi)  #yes
```

String Operations

❑ Addition (+) Operator:

- The *+* operator concatenates strings. It returns a string consisting of the operands joined together.



```
1 p = "Python"
2 s = "Strings"
3 e = "Example"
4
5 p + s
6 "Python Strings"
7 p + s + e
8 "Python Strings Example"
9
10 print("Python Strings" + "Example")
11 "Python Strings Example"
```

String Operations

❑ Multiplication (*) Operator:

- The * operator creates multiple copies of a string. If s is a string and n is an integer, either of the following expressions returns a string consisting of n concatenated copies of ***s***:

s * n

n * s

- The multiplier operand n must be an integer. It can even be zero or negative, in which case the result is an empty string.

String Operations

❑ **Example of String multiplication operations:**

```
1 s = "Python "
2
3 s * 4
4 "Python Python Python Python "
5
6 4 * s
7 "Python Python Python Python "
8
9 "Python " * -5
10 " "
```

String Operations

❑ The in Operator:

- Python also provides a membership operator that can be used with strings. The in operator returns ***True*** if the first operand is contained within the second, and ***False*** otherwise.

- There is also a not in operator, which does exactly the opposite.

String Operations

❑ **in Operator:**

```
1 s = "string"
2
3 s in "This is a string in
  Python"
4
5 True
6
7 s in "This is an array in
  Python"
8
9 False
```

❑ **not in Operator:**

```
1 s = "string"
2
3 s not in "This is a string in
  Python"
4
5 False
6
7 s not in "This is an array in
  Python"
8
9 True
```

String Slicing

- Python also allows a form of indexing syntax that extracts substrings from a string, known as string slicing.

- If **s** is a string, an expression of the form ***s[m:n]*** returns the portion of **s** starting with position ***m***, and up to but not including position ***n***.

❑ **Syntax for string slicing:**

```
s = “Caltech”
s[1:3]
“al”
```

String Slicing

- In Python, String indices are zero-based. The first character in a string has ***index 0***. This applies to both standard indexing and slicing.

- Again, the second index specifies the first character that is not included in the result, the character "***t***" (***s[3]***) in the previous example.

- It may seem slightly unusual, but it produces this result which makes sense: the expression ***s[m:n]*** will return a substring that is ***n - m*** characters in length, in this case, ***3 - 1 = 2***.

- If we omit the first index, the slice starts at the beginning of the string. Hence, ***s[:m]*** and ***s[0:m]*** become equivalent.

String Slicing

- If we omit the second index as in ***s[n:]***, the slice extends from the first index through the end of the string. This is a precise alternative to the more cumbersome ***s[n:len(s)]:***.

```
1 s = "Science"
2
3 s[2:]
4
5 "ience"
6
7 s[2:len(s)]
8
9 "ience"
```

String Slicing

- **ID of a String:**

- ID function in Python accepts a single parameter and is used to return the identity of an object.
- This identity has to be unique and constant for this object during the lifetime.
- Two objects with non-overlapping lifetimes may have the same id() value.
- If we can relate this to C, then they are actually the memory address, here in Python it is the unique id.

String Slicing

- Omitting both indices returns the original string, in its entirety. Literally. It's not a copy, it's a reference to the original string.

```
s = "Python"

t = s[:]

id(s)

59569395

id(t)

59569395

s is t

True
```

String Slicing

- Negative indices can be used with slicing as well. ***-1*** refers to the last character, ***-2*** the second-to-last, and so on, just as with simple indexing. The example shows how to slice the substring ***"yth"*** from the string ***"Python"*** using both positive and negative indices.

```
1 s = "Python"
2
3 s[-5:-2]
4
5 "yth"
6
7 s[1:4]
8
9 "yth"
10
11 s[-5:-2] == s[1:4]
12
13 True
```

String Slicing

- There is one more variant of the slicing syntax. Adding an additional ***:*** and a third index designates a stride (also called a step), which indicates how many characters to jump after retrieving each character in the slice.

- For example, for the string ***"Python"***, the slice ***0:6:2*** starts with the first character and ends with the last character (the whole string), and every second character is skipped.

String Slicing

- Similarly, ***1:6:2*** specifies a slice starting with the second character (***index 1***) and ending with the last character, and again the stride value 2 causes every other character to be skipped.

String Slicing

- We can specify a negative stride value as well, in which case Python steps backward through the string. In that case, the starting/first index should be greater than the ending/second index.

```
1 s = "Python is a highly advanced and useful programming language."
2
3 print(s[::-1])
4
5 egaugnal gnimmargorp lufesu dna decnavda ylhgih a si nohtyP
```

String Indexing

- In Python, strings are ordered sequences of character data, and thus can be indexed in this way. Individual characters in a string can be accessed by specifying the string name followed by a number in square brackets (***[ ]***).

- String indexing in Python is zero-based: the first character in the string has index ***0***, the next has index ***1***, and so on. The index of the last character will be the length of the string minus one.

String Indexing

❑ **The individual characters can be accessed by index as follows:**

```
1 s = "Python"
2
3 s[0]
4
5 "P"
6
7 s[1]
8
9 "y"
10
11 len(s)
12
13 6
14
15 s[len(s)-1]
16
17 "n"
```

String Indexing

- String indices can also be specified with negative numbers, in which case indexing occurs from the end of the string backward: ***-1*** refers to the last character, ***-2*** the second-to-last character, and so on.

```
1  s = "Python"
2
3  s[-1]
4
5  "n"
6
7  s[-2]
8
9  "o"
10
11 len(s)
12
13 6
14
15 s[-len(s)]
16
17 "P"
```

String Functions

❑ **Built in String Methods:**

- **Case Conversion:**

- Methods in this group perform case conversion on the target string.

s.capitalize()

- ***s.capitalize()*** returns a copy of s with the first character converted to uppercase and all other characters converted to lowercase.

String Functions

❑ **s.lower():**

- Converts alphabetic characters to lowercase.

- ***s.lower()*** returns a copy of s with all alphabetic characters converted to lowercase.

String Functions

❑ **s.swapcase():**

- Swaps case of alphabetic characters.

- ***s.swapcase()*** returns a copy of s with uppercase alphabetic characters converted to lowercase and vice versa.

String Functions

❑ **s.title():**

- Converts the target string to “title case.”
- ***s.title()*** returns a copy of s in which the first letter of each word is converted to uppercase and remaining letters are lowercase.
- A major limitation of this function is that it does not attempt to distinguish between important and unimportant words, and it does not handle apostrophes, possessives, or acronyms semantically.

String Functions

❑ **s.upper():**

- Converts alphabetic characters to uppercase.

- ***s.upper()*** returns a copy of s with all alphabetic characters converted to uppercase.

String Functions

❑ **Finding and Replacing Contents in Between Strings:**

- These methods in this group supports optional ***<start>*** and ***<end>*** arguments.
- These are interpreted as for string slicing.
- The action of the method is restricted to the portion of the target string starting at character position ***<start>*** and proceeding up to but not including character position ***<end>***.
- If ***<start>*** is specified but ***<end>*** is not, the method applies to the portion of the target string from ***<start>*** through the end of the string.

String Functions

❑ **s.count(<sub>[, <start>[, <end>]]):**

- Counts occurrences of a substring in the target string.

- ***s.count(<sub>)*** returns the number of non-overlapping occurrences of substring <sub> in s

String Functions

❑ **s.endswith(<suffix>[, <start>[, <end>]]):**

- Determines whether the target string ends with a given substring.

- ***s.endswith(<suffix>)*** returns True if s ends with the specified <suffix> and False otherwise

String Functions

❑ **s.find(<sub>[, <start>[, <end>]]):**

- Searches the target string for a given substring.
- We can use ***s.find()*** to see if a Python string contains a particular substring.
- ***s.find(<sub>)*** returns the lowest index in s where substring ***<sub>*** is found.
- This function returns ***-1*** if the specified substring is not found.

String Functions

❑ **s.index(<sub>[, <start>[, <end>]]):**

- Searches the target string for a given substring.

- This function is identical to ***s.find()***, except that it raises an exception if ***<sub>*** is not found rather than returning ***-1***.

String Functions

❑ **s.rfind(<sub>[, <start>[, <end>]]):**

- Searches the target string for a given substring starting at the end.
- ***s.rfind(<sub>)*** returns the highest index in s where substring <sub> is found.
- As also with ***s.find()***, if the substring is not found, ***-1*** is returned.

String Functions

❑ **s.startswith(<prefix>[, <start>[, <end>]]):**

- Determines whether the target string starts with a given substring.

- When we use the ***.startswith()*** function in Python , ***s.startswith(<suffix>)*** returns True if ***s*** starts with the specified ***<suffix>*** and False otherwise.

String Functions

❑ **Character Classification:**

- Methods in this group classify a string based on the characters it contains.

s.isalnum()

- Determines whether the target string consists of alphanumeric characters.

- ***s.isalnum()*** returns True if s is nonempty and all its characters are alphanumeric (either a letter or a number), and False otherwise.

String Functions

❑ **s.isalpha():**

- Determines whether the target string consists of alphabetic characters.

- ***s.isalpha()*** returns True if s is nonempty and all its characters are alphabetic, and False otherwise.

String Functions

❑ **s.isdigit():**

- Determines whether the target string consists of digit characters.
- We can use the ***s.isdigit()*** function in Python to check if any string is made of only digits.
- ***s.isdigit()*** returns True if s is nonempty and all its characters are numeric digits, and False otherwise.

String Functions

❑ **s.isidentifier():**

- Determines whether the target string is a valid Python identifier.

- ***s.isidentifier()*** returns True if s is a valid Python identifier according to the language definition, and False otherwise.

String Functions

❑ **s.isupper():**

- Determines whether the target string's alphabetic characters are uppercase.
- ***s.isupper()*** returns True if s is nonempty and all the alphabetic characters it contains are uppercase, and False otherwise.
- Non-alphabetic characters are ignored.

String Functions

❑ **s.islower():**

- Determines whether the target string's alphabetic characters are lowercase.
- ***s.islower()*** returns True if s is nonempty and all the alphabetic characters it contains are lowercase, and False otherwise.
- Non-alphabetic characters are ignored.

String Functions

❑ **s.isprintable():**

- Determines whether the target string consists entirely of printable characters.
- ***s.isprintable()*** returns True if s is empty or all the alphabetic characters it contains are printable.
- This function returns False if s contains at least one non-printable character. Non-alphabetic characters are ignored.

String Functions

❑ **s.isspace():**

- Determines whether the target string consists of whitespace characters.

- ***s.isspace()*** returns True if ***s*** is nonempty and all characters are whitespace characters, and False otherwise.

- The most commonly encountered whitespace characters are space ' ', tab ***'\t'***, and newline ***'\n'***.

String Functions

❑ **String Formatting:**

- Methods in this group modify or enhance the format of a string.

s.center(<width>[, <fill>])

- Centers a string in a field.

- ***s.center(<width>)*** returns a string consisting of s centered in a field of width ***<width>***. By default, padding consists of the ASCII space character.

String Functions

❑ **s.lstrip([<chars>]):**

- Trims leading characters from a string.

- ***s.lstrip()*** returns a copy of s with any whitespace characters removed from the left end.

String Functions

❑ **s.replace(<old>, <new>[, <count>]):**

- Replaces occurrences of a substring within a string.
- In Python, to remove a character from a string, you can use the Python string ***.replace() function***.
- ***s.replace(<old>, <new>)*** returns a copy of s with all occurrences of substring <old> replaced by <new>.