Introduction to Python

Strings

Topics

String

In Python, text is represented as a **string**, which is a sequence of *characters* (letters, digits, and symbols). Strings in Python are created with single , double quotes or triple quotes.

```
message = ‘what do you like?’
response = ”spam”
response2 = '''ham'''
response3 = ”””spam”””
```

The built-in len method can compute the length of a string.

```
print(len(response))      # 4
```

String Concatenation

```
# concatenation with +

message = ‘what do you like?’
response = ”spam”
message2 = message + response
print(message2)              # what do you like?spam
```

String Indexing

Python allows you to retrieve individual members of a string by specifying the *index* of that member, which is the integer that uniquely identifies that member's position in the string. This is exactly the same as indexing into a list!

```
message = ”hello”
print(message[0])          # h
print(message[1])          # e
print(message[-1])         # o
print(message[4])          # o
print(message[5])          # error! out of range!
```

Slicing

We can also “slice” a string, specifying a start-index and stop-index, and return a subsequence of the items contained within the slice.

Slicing is a very important indexing scheme that we will see many times in other data structures(lists, tuples, strings, Numpy's arrays, Panda's data frames, etc..). Slicing can be done using the syntax:

```
some_string[start:stop:step]
```

where

start: index of beginning of the slice(included), default is 0

stop: index of the end of the slice(excluded), default is length of string

step: increment size at each step, default is 1.

Slicing

```
language = "python"
print(language[0:4])  # pyth
                      # 0 up to but not including index 4
print(language[:4])   # pyth, default start index at 0
print(language[4:])   # on, default end index is length of string
print(language[:])    # python, 0 to end of string
print(language[:-1])  # pytho, all except the last character
print(language[0:5:2]) # pto, step size of 2
print(language[::-1])  # negative step size traverses backwards
                       # nohtyp
print(language[language[2:5:-1] ])  # empty string
print(language[language[5:2:-1] ])  # noh, from 5 to 2 backwards
```

Slicing

When step size is negative, the default starting index is the last element and the default end is the first element inclusive.

```
language = ”python"
```

```
The default start index is the last element.
print(language[:-3:-1]) # no, last two characters reversed
```

```
The default stop index is the first element inclusive.
print(language[1::-1]) # yp, first two characters reversed
```

List Slicing

List slicing works the same way!

```
L = [1, 2, 3, 4, 5]
print(L[1:3])          # [2, 3]
print(L[0:4:2])        # [1, 3]
print(L[:])            # [1, 2, 3, 4, 5]
print(L[::-1])         # [5, 4, 3, 2, 1]
```

f-Strings

f-Strings is the new way to format strings in Python. (v 3.6)

Also called "formatted string literals," f-strings are string literals that have an f at the beginning and curly braces containing expressions that will be replaced with their values.

```
name = “Mike”
gpa = 3.2
f_str = f“I am {name} with a {gpa} gpa.”
print(f_str)
I am Mike with a 3.2 gpa.
```

f-Strings

An f-string is special because it permits us to write Python code *within* a string; any expression within curly brackets, {}, will be executed as Python code, and the resulting value will be converted to a string and inserted into the f-string at that position.

```
grade1 = 1.5
grade2 = 2.5
ave = f“average is {(grade1+grade2)/2}”
print(ave)     # average is 2.0
```

```
This is equivalent but it is preferable to use an f-string.
average = “average is ” + str((grade1+grade2)/2)
```

f-Strings Precision(optional)

```
import math
x = math.pi
print(f“{x}”)   # 3.141592653589793
print(f“{x:.2f}”) # 3.14
print(f“{x:.3f}”) # 3.142
```

Special Characters

It is not valid syntax to have a single quote inside of a single quoted string.

```
a = ‘that’s not legal’  # SyntaxError: invalid syntax
```

Instead, we can use double quotes outside the string. Or alternatively, use double quotes inside and single quotes outside.

```
print(“It’s legal to do this.”, ‘And he said, “This is ok.”’)
```

Escape Sequence

Escape sequence is a special sequence of characters used to represent certain special characters in a string.

`\n`	new line character
`\"`	double quote
`\'`	single quote
`\\`	backslash character
`\t`	tab

Escape Sequence

What is the output?

```
print(“How \tmany \‘lines\’\n are\n shown\n \“here\”?”)
```

```
How	many ‘lines’
are
shown
“here”?
```

Multiline String

To span multiple lines, put three single quotes or three double quotes around the string instead of one. The string can then span as many lines as you want:

```
a = '''three
  lines
     of
output'''
print(a)
```

Output:

```
three
  lines
     of
output
```

Note that the spacing is preserved.

Multiline strings are often used in documentation strings as we will see later.

String Methods

The following is a short list of useful string methods. These methods can be accessed through the dot notation applied to a string variable or literal.

count(value)	returns the number of times value appears in the string.
find(value)	returns the lowest index of a substring value in a string. If substring is not found, returns -1.
upper() and lower()	returns a copy of the string capitalizing(or lower casing) all characters in the string
split()	splits a string into a list. A separator can be specified. The default separator is any whitespace.

String Methods

```
s = "hellobyehihellohihello"
a = s.count("hello")
print(a)                # 3
index = s.find("bye")
print(index)            # 5
print(s.find("chao"))   # -1, not found

b = "python"
print(b.upper())        # PYTHON
print("JAVA".lower())   # java
```

For Loops with Strings

A for loop can be used to loop through each character of a string.

```
s = "hello"
for x in s:
     print(x)
```

```
h
e
l
l
o
```

Equivalently:

```
s = "hello"
for i in range(len(s)):
   print(s[i])
```

split()

split() method splits a string into a list. A separator can be specified. The default separator is any whitespace.

```
fruits = "apple mango banana grape"
fruit_lst = fruits.split()
print(fruit_lst)    #  [‘apple’, ‘mango’, ‘banana’, ‘grape’]

greeting = “hi, I am Mike, I just graduate.”
greet_lst = greeting.split(", ")
print(greet_lst)    # [‘hi’, ‘I am Mike’, ‘I just graduate.’]
```

Find a word from a list of words

Write a function that accepts two parameters: a list of words and a target word. The function returns the index of the target word in the list. If the target word is not in the list, returns -1.

```
def search(lst_words, target):
    for i in range(len(lst_words)):
        if lst_words[i] == target:
            return i
    return -1
words = ["hi", "hello", "goodbye", "hola", "bonjour"]
print(search(words, "goodbye"))    # 2
print(search(words, "bye"))        # -1
index = search(words, "bonjour")
print(index)                       # 4
```

References

1) Vanderplas, Jake, A Whirlwind Tour of Python, O'reilly Media.

This book is completely free and can be downloaded online at O'reilly's site.