Introduction to Python

Strings

Topics

1) Strings Concatenation
2) Indexing and Slicing
3) f-Strings
4) Escape Sequences
5) String Methods
6) For Loops vs Slicing

String

Strings in Python are created with single , double quotes or triple quotes.

```
In [1]: message = ‘what do you like?’
        response = "spam"
        response2 = '''ham'''
        response3 = """spam"""
```

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

```
In [2]: len(response)
Out [2]: 4
```

String Concatenation

```
In [1]: # concatenation with +
        message + response
Out [1]: ‘what do you like?spam’

In [2]: # multiplication is multiple concatenation
        5 * response
Out [2]: ‘spamspamspamspamspam’
```

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.

```
In [1]: message = "what do you like?"

In [2]: message[0] # first character is at index 0
Out [2]: 'w'

In [3]: # negative indices wraps around the end
        message[-1]   # last character
Out [3]: '?'
```

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

```
In [1]: language = "python"
In [2]:  language[0:4] # 0 up to but not including index 4
Out [2]: ‘pyth’

In [3]:  language[0:5:2] # step size of 2
Out [3]: ‘pto’
```

Slicing

```
In [4]: language = ”python"
In [5]: # default start index is 0
          language[:4]
Out [5]: ‘pyth’

In [6]: # default end index is length of string
          language[4:]
Out [6]: ‘on’

In [7]:  language[:]  # default 0 to end of string
Out [7]: ‘python’
```

Slicing

```
In [8]: language = ”python"
In [9]: # negative indices wraps around the end
          language[-1]   # last character
Out [9]: ‘n’

In [10]: # all except the last character
          language[:-1]
Out [10]: ‘pytho’

In [11]: # negative step size traverses backwards
          language[::-1]
Out [11]: ‘nohtyp’
```

Slicing

```
In [12]: language = ”python"

In [13]: language[2:5:-1]
Out [13]: ‘’

In [14]: language[5:2:-1]
Out [14]: ‘noh’
```

f-Strings

f-Strings is the new way to format strings in Python. (v 3.6) An f-string is denoted by preceding the opening quotation mark with the lowercase f character.

```
In [1]: name = “Mike”
In [2]: gpa = 3.2
In [3]: f_str = f“I am {name} with a {gpa} gpa.”
In [4]: 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.

```
In [1]: grade1 = 1.5
In [2]: grade2 = 2.5
In [3]: f_str = f“average is {(grade1+grade2)/2}”
In [4]: f_str
Out [4]: ‘average is 2.0.'
```

f-Strings Precision

```
In [1]: import math
In [2]: x = math.pi
In [3]: print(f“{x}”)
3.141592653589793
In [4]: print(f“{x:.2f}”)
3.14
In [5]: print(f“{x:.3f}”)
3.142
```

str()

The function str() can be construct string objects from integer or float literals.

```
In [1]: y = str(2)     # y will be '2'
        z = str(3.0)   # z will be '3.0'
```

Special Characters

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

```
In [1]: ‘that’s not legal’
  File "<ipython-input-7-2762381d46b7>", line 1
    'that's not legal'
          ^
SyntaxError: invalid syntax
```

Instead, we can use double quotes outside the string.

```
In [2]: print(“It’s legal to do this.”, ‘And he said, “This is ok.”’)
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?

```
In [1]: 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:

```
In [1]: '''three
        lines
        of
        output'''
Out[1]: 'three\nlines \nof\noutput'
```

Notice that the string Python creates contains a \n sequence everywhere our input started a new line. Each newline is a character in the string.

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
strip()	returns a copy of the string with all leading and trailing whitespace and newline('\n) characters removed
replace(old, new)	returns a copy of the string replacing every occurrence of old substring with new substring

String Methods

```
In [1]: s = "hellohihellohihello"

In [2]: s.count("hello")
Out [2]: 3

In [3]: s.find("chao")
Out [3]: -1

In [4]: s.find("hi")
Out [4]: 5
```

String Methods

```
In [1]: s = "hello"

In [2]: s.upper()
Out [2]: 'HELLO'

In [3]: "HELLO".lower()
Out [3]: 'hello'
```

String Methods

```
In [1]: s = "  \n   hello\n     "

In [2]: s.strip()
Out [2]: 'hello'

In [3]: s = "hi, Sarah, I like the name Sarah!"
In [4]: s.replace("Sarah", "John")
Out [4]: 'hi, John, I like the name John!'
```

String Methods

Note that these string methods returns a new string rather than modifying the original string.

```
In [1]: s = "hi, Mike!"

In [2]: s.replace("Mike", "John")
Out [2]: 'hi, John!'

In [3]: s
Out [3]: 'hi, Mike!'
```

For Loops with Strings

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

```
In [1]: s = "hello"
In [2]: for x in s:
            print(x)
h
e
l
l
o
```

Extracting Strings

Extracting characters from a string can be done with a for loop or slicing. For example, consider the problem of exacting every other character from a string.

Using a for loop and concatenation:

```
In [1]: s = "hello!" # 6 characters
In [2]: extract = ""  # initializes to an empty string
In [3]: for i in range(len(s)):  # range(6) = 0,1,..,5
            if i % 2 == 0:
                extract += s[i] # concatenate ith character
In [4]: extract
Out [25]: 'hlo'
```

Extracting Strings

We could also have done the previous problem with slicing. Note this solution is simpler and uses no loops.

Slicing:

```
In [1]: s = "hello!" # 6 characters
In [2]: extract = s[::2]
In [3]: extract
Out [3]: 'hlo'
```

Note: If you're used to using loops in a language Java or C. You'll appreciate, as we will see later, data structures in Python that can do parallel computations without loops(Numpy arrays, Panda's dataframes).

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.