wtech
Teknolojide Kadın Derneği

Introduction to Machine Learning with Python

Strings

Dr. Süha Tuna
İTÜ Informatics Institute

CONTENT

- Basics
- The `in` operatör
- Indexing
- Slices
- Looping
- String methods

Basics

Creating a string A string is created by enclosing text in quotes. You can use either single quotes, `'`, or double quotes, `"`. A triple-quote can be used for multi-line strings. Here are some examples:

```
s = 'Hello'
t = "Hello"
m = """This is a long string that is
spread across two lines."""
```

Input Recall from Chapter 1 that when getting numerical input we use an `eval` statement with the `input` statement, but when getting text, we do not use `eval`. The difference is illustrated below:

```
num = eval(input('Enter a number: '))
string = input('Enter a string: ')
```

Empty string The empty string `''` is the string equivalent of the number 0. It is a string with nothing in it. We have seen it before, in the print statement's optional argument, `sep=''`.

Length To get the length of a string (how many characters it has), use the built-in function `len`. For example, `len('Hello')` is 5.

Concatanation and repetition

Expression	Result
`'AB'+'cd'`	`'ABcd'`
`'A'+'7'+'B'`	`'A7B'`
`'Hi'*4`	`'HiHiHiHi'`

```
s = ''
for i in range(10):
    t = input('Enter a letter: ')
    if t=='a' or t=='e' or t=='i' or t=='o' or t=='u':
        s = s + t
print(s)
```

The `in` operator

The `in` operator is used to tell if a string contains something. For example:

```
if 'a' in string:
    print('Your string contains the letter a.')
```

You can combine `in` with the `not` operator to tell if a string does not contain something:

```
if ';' not in string:
    print('Your string does not contain any semicolons.')
```

Example In the previous section we had the long if condition

```
if t=='a' or t=='e' or t=='i' or t=='o' or t=='u':
```

Using the `in` operator, we can replace that statement with the following:

```
if t in 'aeiou':
```

Indexing

Statement	Result	Description
`s[0]`	`P`	first character of `s`
`s[1]`	`y`	second character of `s`
`s[-1]`	`n`	last character of `s`
`s[-2]`	`o`	second-to-last character of `s`

- The first character of `s` is `s[0]`, not `s[1]`. Remember that in programming, counting usually starts at 0, not 1.
- Negative indices count backwards from the end of the string.

Slices

A *slice* is used to pick out part of a string. It behaves like a combination of indexing and the `range` function. Below we have some examples with the string `s='abcdefghij'`.

```
index:     0 1 2 3 4 5 6 7 8 9
letters:   a b c d e f g h i j
```

Code	Result	Description
`s[2:5]`	`cde`	characters at indices 2, 3, 4
`s[ :5]`	`abcde`	first five characters
`s[5: ]`	`fghij`	characters from index 5 to the end
`s[-2: ]`	`ij`	last two characters
`s[ : ]`	`abcdefghij`	entire string
`s[1:7:2]`	`bdf`	characters from index 1 to 6, by twos
`s[ : :-1]`	`jihgfedcba`	a negative step reverses the string

```
s = s[:4] + 'X' + s[5:]
```

Slices

- The basic structure is

  string name[*starting location* : *ending location+1*]

  Slices have the same quirk as the `range` function in that they do not include the ending location. For instance, in the example above, `s[2:5]` gives the characters in indices 2, 3, and 4, but not the character in index 5.

- We can leave either the starting or ending locations blank. If we leave the starting location blank, it defaults to the start of the string. So `s[:5]` gives the first five characters of `s`. If we leave the ending location blank, it defaults to the end of the string. So `s[5:]` will give all the characters from index 5 to the end. If we use negative indices, we can get the ending characters of the string. For instance, `s[-2:]` gives the last two characters.

- There is an optional third argument, just like in the `range` statement, that can specify the step. For example, `s[1:7:2]` steps through the string by twos, selecting the characters at indices 1, 3, and 5 (but not 7, because of the aforementioned quirk). The most useful step is -1, which steps backwards through the string, reversing the order of the characters.

Looping in strings

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

```
for c in s:
    print(c)
```

String methods

Method	Description
`lower()`	returns a string with every letter of the original in lowercase
`upper()`	returns a string with every letter of the original in uppercase
`replace(x,y)`	returns a string with every occurrence of `x` replaced by `y`
`count(x)`	counts the number of occurrences of `x` in the string
`index(x)`	returns the location of the first occurrence of `x`
`isalpha()`	returns `True` if every character of the string is a letter

Statement	Description
`print(s.count(' '))`	prints the number of spaces in the string
`s = s.upper()`	changes the string to all caps
`s = s.replace('Hi','Hello')`	replaces each `'Hi'` in `s` with `'Hello'`
`print(s.index('a'))`	prints location of the first `'a'` in `s`

Escape characters

- `\n` the *newline character*. It is used to advance to the next line. Here is an example:

```
print('Hi\n\nthere!')
```

```
Hi

There!
```

- `\'` for inserting apostrophes into strings. Say you have the following string:

```
s = 'I can't go'
```

This will produce an error because the apostrophe will actually end the string. You can use `\'` to get around this:

```
s = 'I can\'t go'
```

Another option is to use double quotes for the string:

```
"s = I can't go"
```

- `\"` analogous to `\'`.
- `\\` This is used to get the backslash itself. For example:

```
filename = 'c:\\programs\\file.py'
```

- `\t` the tab character

Examples

```
s = input('Enter some text: ')
for i in range(len(s)):
    if s[i]=='a':
        print(i)
```

```
s = input('Enter some text: ')
doubled_s = ''
for c in s:
    doubled_s = doubled_s + c*2
```

```
name = input('Enter your name: ')
for i in range(len(name)):
    print(name[:i+1], end=' ')
```

```
s = s.lower()
for c in ',.;:-?!()\'"':
    s = s.replace(c, '')
```

```
s = input('Enter your decimal number: ')
print(s[s.index('.'):])
```

```
from math import floor
num = eval(input('Enter your decimal number: ')
print(num - floor(num))
```

Exercises

1. Write a program that asks the user to enter a string. The program should then print the following:
   (a) The total number of characters in the string
   (b) The string repeated 10 times
   (c) The first character of the string (remember that string indices start at 0)
   (d) The first three characters of the string
   (e) The last three characters of the string
   (f) The string backwards
   (g) The seventh character of the string if the string is long enough and a message otherwise
   (h) The string with its first and last characters removed
   (i) The string in all caps
   (j) The string with every *a* replaced with an *e*

2. A simple way to estimate the number of words in a string is to count the number of spaces in the string. Write a program that asks the user for a string and returns an estimate of how many words are in the string.

3. People often forget closing parentheses when entering formulas. Write a program that asks the user to enter a formula and prints out whether the formula has the same number of opening and closing parentheses.

Exercises

4. Write a program that asks the user to enter a word and prints out whether that word contains any vowels.

5. Write a program that asks the user to enter a string. The program should create a new string called `new_string` from the user's string such that the second character is changed to an asterisk and three exclamation points are attached to the end of the string. Finally, print `new_string`. Typical output is shown below:

```
Enter your string: Qbert
Q*ert!!!
```

6. Write a program that asks the user to enter a string `s` and then converts `s` to lowercase, removes all the periods and commas from `s`, and prints the resulting string.

7. Write a program that asks the user to enter a word and determines whether the word is a palindrome or not. A palindrome is a word that reads the same backwards as forwards.

8. At a certain school, student email addresses end with `@student.college.edu`, while professor email addresses end with `@prof.college.edu`. Write a program that first asks the user how many email addresses they will be entering, and then has the user enter those addresses. After all the email addresses are entered, the program should print out a message indicating either that all the addresses are student addresses or that there were some professor addresses entered.

9. Ask the user for a number and then print the following, where the pattern ends at the number that the user enters.

```
1
 2
  3
   4
```