

Strings

In Python (and most languages), text is represented in a *string*: a sequence of characters.

October 18, 2010

2

Strings

In Python (and most languages), text is represented in a *string*: a sequence of characters.

A string value is specified by a pair of single or double quotations.

Example.

```
s = 'trinity'
```

October 18, 2010

4

Strings

October 18, 2010

1

Strings

In Python (and most languages), text is represented in a *string*: a sequence of characters.

A string value is specified by a pair of single or double quotations.

October 18, 2010

3

Strings

Each letter of a string is indexed by an integer, starting from 0.

Example.

```
0 1 2 3 4 5 6
t r i n i t y
```

October 18, 2010

6

Strings

To access a string `t`'s `i`th letter, use `t[i]`.

Example.

```
>>> t = 'trinity'
```

October 18, 2010

8

Strings

Each letter of a string is indexed by an integer, starting from 0.

October 18, 2010

5

Strings

To access a string `t`'s `i`th letter, use `t[i]`.

October 18, 2010

7

Strings

To access a string `t`'s `i`th letter, use `t[i]`.

Example.

```
>>> t = 'trinity'
>>> print t[0]
t
```

October 18, 2010

10

Strings

To access a string `t`'s `i`th letter, use `t[i]`.

Example.

```
>>> t = 'trinity'
>>> print t[0]
t
>>> print t[3]
n
```

October 18, 2010

12

Strings

To access a string `t`'s `i`th letter, use `t[i]`.

Example.

```
>>> t = 'trinity'
>>> print t[0]
```

October 18, 2010

9

Strings

To access a string `t`'s `i`th letter, use `t[i]`.

Example.

```
>>> t = 'trinity'
>>> print t[0]
t
>>> print t[3]
```

October 18, 2010

11

Strings

Strings are *immutable*. Unlike variables, letters inside a string cannot be modified.

Example.

```
>>> t = 'trinity'
```

October 18, 2010

14

Strings

Strings are *immutable*. Unlike variables, letters inside a string cannot be modified.

Example.

```
>>> t = 'trinity'
>>> t[0] = 'T'
TypeError: object does not...
```

October 18, 2010

16

Strings

Strings are *immutable*. Unlike variables, letters inside a string cannot be modified.

October 18, 2010

13

Strings

Strings are *immutable*. Unlike variables, letters inside a string cannot be modified.

Example.

```
>>> t = 'trinity'
>>> t[0] = 'T'
```

October 18, 2010

15

len

The function `len` gives the length of (or the number of letters in) a string.

Example.

```
>>> t = 'trinity'
```

October 18, 2010

18

len

The function `len` gives the length of (or the number of letters in) a string.

October 18, 2010

17

len

The function `len` gives the length of (or the number of letters in) a string.

Example.

```
>>> t = 'trinity'
>>> len(t)
7
```

October 18, 2010

20

len

The function `len` gives the length of (or the number of letters in) a string.

Example.

```
>>> t = 'trinity'
>>> len(t)
```

October 18, 2010

19

Slicing strings

Strings can be *sliced* by specifying the first and last indices: `s[i:j]` is the substring of `s` from `i` up to but not including `j`.

October 18, 2010

22

Slicing strings

Strings can be *sliced* by specifying the first and last indices: `s[i:j]` is the substring of `s` from `i` up to but not including `j`.

Example.

```
>>> t = 'trinity'
>>> print t[0:3]
```

October 18, 2010

24

len

The function `len` gives the length of (or the number of letters in) a string.

Example.

```
>>> t = 'trinity'
>>> len(t)
7
```

The last letter is indexed by `len(t) - 1`.

```
>>> t[len(t)-1]
'y'
```

October 18, 2010

21

Slicing strings

Strings can be *sliced* by specifying the first and last indices: `s[i:j]` is the substring of `s` from `i` up to but not including `j`.

Example.

```
>>> t = 'trinity'
```

October 18, 2010

23

Slicing strings

Strings can be *sliced* by specifying the first and last indices: `s[i:j]` is the substring of `s` starting from `i` through `j`.

Example.

```
>>> t = 'trinity'
>>> print t[0:3]
trin
>>> print t[2:5]
```

October 18, 2010

26

Iteration with For

Example.

```
str = 'trinity'
for c in str:
    print c,
```

Output: t r i n i t y

October 18, 2010

28

Slicing strings

Strings can be *sliced* by specifying the first and last indices: `s[i:j]` is the substring of `s` from `i` up to but not including `j`.

Example.

```
>>> t = 'trinity'
>>> print t[0:3]
tri
```

October 18, 2010

25

Slicing strings

Strings can be *sliced* by specifying the first and last indices: `s[i:j]` is the substring of `s` starting from `i` through `j`.

Example.

```
>>> t = 'trinity'
>>> print t[0:3]
trin
>>> print t[2:5]
ini
```

October 18, 2010

27

Iteration

```
for c in str:  
    print c,  
  
is equivalent to
```

October 18, 2010

30

Iteration

```
for c in str:  
    print c,
```

October 18, 2010

29

For vs. While Loop

```
for c in str:  
    <statements>
```

October 18, 2010

32

Iteration

```
for c in str:  
    print c,  
  
is equivalent to
```

```
i = 0  
while i < len(str):  
    print str[i],  
    i = i + 1
```

October 18, 2010

31

For vs. While Loop

```
for c in str:  
<statements>
```

is equivalent to

```
i = 0  
while i < len(str):  
<statements>  
i = i + 1
```

October 18, 2010

34

Iteration

```
for <elements> in <object>:  
<statements>
```

- Particularly useful when you wish to visit every element of <object>.

October 18, 2010

36

For vs. While Loop

```
for c in str:  
<statements>
```

is equivalent to

October 18, 2010

33

Iteration

```
for <elements> in <object>:  
<statements>
```

October 18, 2010

35

Function: test for a letter

Example.

```
def has_ch(word, ch):
    for letter in word:
        if letter == ch:
            return True
    return False
```

October 18, 2010

38

Function: Test for a letter

Example.

```
def has_ch(word, ch):
    for letter in word:
        if letter == ch:
            return True
    return False
```

```
>>> has_ch('trinity', 'a')
```

```
False
```

October 18, 2010

40

Iteration

```
for <element> in <object>:
```

```
<statements>
```

- Particularly useful when you wish to visit every element of <object>.
- With strings, you can visit every character without dealing with indices.

October 18, 2010

37

Function: Test for a letter

Example.

```
def has_ch(word, ch):
    for letter in word:
        if letter == ch:
            return True
    return False
```

```
>>> has_ch('trinity', 'a')
```

October 18, 2010

39

Function: Count a letter

Example.

```
def count_ch(word, ch):
    counter = 0
    for letter in word:
        if letter == ch:
            counter = counter + 1
    return counter
```

This function counts the number of ch characters in word.

October 18, 2010

42

Function: Count a letter

Example.

```
def count_ch(word, ch):
    counter = 0
    for letter in word:
        if letter == ch:
            counter = counter + 1
    return counter
```

```
>>> count_ch('trinity', 't')
```

```
2
```

October 18, 2010

44

Function: Count a letter

Example.

```
def count_ch(word, ch):
    counter = 0
    for letter in word:
        if letter == ch:
            counter = counter + 1
    return counter
```

October 18, 2010

41

Function: Count a letter

Example.

```
def count_ch(word, ch):
    counter = 0
    for letter in word:
        if letter == ch:
            counter = counter + 1
    return counter
```

```
>>> count_ch('trinity', 't')
```

October 18, 2010

43

In-class Exercises

- Write an iterative function that will print the letters of any string in reverse order.
- Write a recursive function that will print the letters of any string in reverse order.
- Write an iterative function that will return the reverse of the string.
- Write a recursive function that will return the reverse of a string.