Strings

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

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'
```


Strings

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.

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
```


Strings

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

Example.

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


Strings

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

Strings

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

Strings

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

Example.

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


Strings

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

Example.

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


Strings

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

Example.

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


Strings

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

Example.

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

Strings

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

Example.

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


Strings

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

Example.

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


Strings

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

Strings

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

Example.

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

len

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

Example.

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


len

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

Example.

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


len

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

Example.

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


len

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

Example.

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

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'
```


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`.

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'
```


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]
```

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]
```


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
```


Iteration with For

Example.

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

Output: `t r i n i t y`

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
```

Iteration

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

is equivalent to

Iteration

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


For vs. While Loop

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


Iteration

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

is equivalent to

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

For vs. While Loop

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

is equivalent to

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


For vs. While Loop

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

is equivalent to

Iteration

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

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

Iteration

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

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.

Function: test for a letter

Example.

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


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
```


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' )
```

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`.

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

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')
```


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

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.