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.

Example.

```
s = 'trinity'
```


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


Strings

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

Example.

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


`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`.

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-1`.

Example.

```
>>> t = 'trinity'
>>> print t[0:4]
trin
>>> print t[2:6]
init
```


Iteration

Recall the *for* statement for simple iteration:

```
for i in range(n):
  <statements>
```

This is actually a special case of

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


Iteration

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

- `<object>` is a collection of elements.
- `<element>` is a variable that refers to an element in `<object>`.
- `<statements>` can be any sequence of statements.

Iteration

Example.

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

- The string `s` is a collection of characters.
- `c` is a variable that refers to a character in the string `s`.

Iteration

Example.

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

Output: `t r i n i t y`

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.

Iteration

Example.

```
def has_a(word):
  for letter in word:
    if letter == 'a':
      return True
  return False
```

This function checks if `word` has any `'a'`.

Iteration

Example.

```
def has_a(word):
  for letter in word:
    if letter == 'a':
      return True
  return False
```

```
>>> has_a('trinity')
False
```


Iteration

Example.

```
def has_a(word):
  for letter in word:
    if letter == 'a':
      return True
  return False
```

```
>>> has_a('hartford')
True
```

Iteration

Example.

```
def count_t(word):
  counter = 0
  for letter in word:
    if letter == 't':
      counter = counter + 1
  return counter
```

This function counts the number of `'t's` in word.

Iteration

Example.

```
def count_t(word):
  counter = 0
  for letter in word:
    if letter == 't':
      counter = counter + 1
  return counter
```

```
>>> count_t('trinity')
2
```


Iteration

Example.

```
def count_t(word):
  counter = 0
  for letter in word:
    if letter == 't':
      counter = counter + 1
  return counter
```

```
>>> count_t('hartford')
1
```

Iteration

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

- A string is just one example of `<object>`.
- Integers can be `<object>` as well.
- `range(m, n)` gives a sequence of integers from `m` to `n-1`.
- `range(n)` gives a sequence of integers from `0` to `n-1`.

Iteration

Example.

```
s = 'trinity'
for i in range(2, 6):
  print s[i],
```

Output: `i n i t`

Iteration

Example.

```
s = 'trinity'
for i in range(4):
  print s[i],
```

Output: `t r i n`