Python

Strings

Strings are sequences of characters

Strings are sequences of characters

No separate character type: just a string of length 1

Strings are sequences of characters

No separate character type: just a string of length 1

Indexed exactly like lists

Strings are sequences of characters

No separate character type: just a string of length 1

Indexed exactly like lists

```
name = 'Darwin'
print name[0], name[-1]
D n
```

for iterates through characters

for iterates through characters

```
name = 'Darwin'
for c in name:
    print c
```

D

a

r

w

i

n

Use either ' or " (as long as they match)

Use either ' or " (as long as they match)

print 'Alan', "Turing"
Alan Turing

Use either ' or " (as long as they match)

print 'Alan', "Turing"

Alan Turing

 Strings are the same no matter how they're created

Use either ' or " (as long as they match)

`print` 'Alan', "Turing"
Alan Turing

 Strings are the same no matter how they're created

`print` 'Alan' == "Alan"
 True

Strings are compared character by character
from left to right

Strings are compared character by character

from left to right

```
print 'a' < 'b'
```
True

Strings are compared character by character

from left to right

```
print 'a' < 'b'
```
True
```
print 'ab' < 'abc'
```
True

Strings are compared character by character

from left to right

```
print 'a' < 'b'
```
True
```
print 'ab' < 'abc'
```
True
```
print '1' < '9'
```
True

Strings are compared character by character

from left to right

```
print 'a' < 'b'
True
print 'ab' < 'abc'
True
print '1' < '9'
True
print '100' < '9'
True
```

Strings are compared character by character

from left to right

print 'a' < 'b'
True
print 'ab' < 'abc'
True
print '1' < '9'
True
print '100' < '9'
True
print 'A' < 'a'
True

Strings are *immutable* : cannot be changed in place

Strings are *immutable* : cannot be changed in place

```
name = 'Darwin'
name[0] = 'C'
```

TypeError: 'str' object does not support item assignment

Strings are *immutable* : cannot be changed in place

```
name = 'Darwin'
name[0] = 'C'
```
TypeError: 'str' object does not support item assignment

Immutability improves performance

Strings are *immutable* : cannot be changed in place

name = 'Darwin'
name[0] = 'C'
TypeError: 'str' object does not support item assignment

Immutability improves performance

See later how immutability improves programmers'

performance

Use + to concatenate strings

Use + to concatenate strings

```
name = 'Charles' + ' ' + 'Darwin'
print name
```
Charles Darwin

Use + to concatenate strings

```
name = 'Charles' + ' ' + 'Darwin'
print name
```
Charles Darwin

Concatenation always produces a new string

Use + to concatenate strings

name = 'Charles' + ' ' + 'Darwin'
print name
Charles Darwin

Concatenation always produces a new string

original = 'Charles'

```
┌──────────┐
│ original │ ⟶ 'Charles'
└──────────┘
```

Use + to concatenate strings

```
name = 'Charles' + ' ' + 'Darwin'
print name
```
Charles Darwin

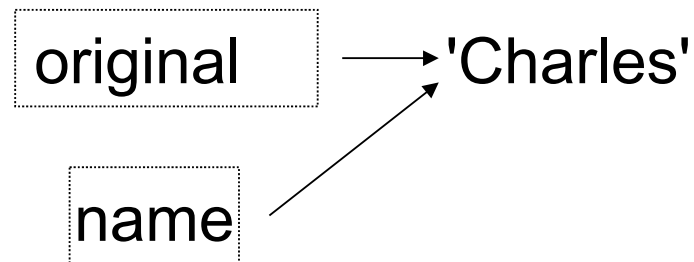# Concatenation always produces a new string

```
original = 'Charles'
name = original
```

original ────→ 'Charles'

name ───────↗

Use + to concatenate strings

```
name = 'Charles' + ' ' + 'Darwin'
print name
```
Charles Darwin

Concatenation always produces a new string

```
original = 'Charles'
name = original
name += ' Darwin'
```

original   ⟶ 'Charles'

name   ⟶ 'Charles Darwin'

Often used to format output

Often used to format output

```
print 'reagant: ' + str(reagant_id) + ' produced ' + \
    str(percentage_yield) + '% yield'
```

Often used to format output

```
print 'reagant: ' + str(reagant_id) + ' produced ' + \
    str(percentage_yield) + '% yield'
```

There's a better way...

Use string % value to format output

Use string % value to format output

```
output = 'reagant: %d' % 123
print output
```

reagant: 123

Use string % value to format output

```
output = 'reagant: %d' % 123
print output
```
reagant: 123

```
percentage_yield = 12.3
print 'yield: %6.2f' % percentage_yield
```
yield: 12.30

And string % (v1, v2, ...) for multiple values

And string % (v1, v2, ...) for multiple values

```
reagant_id = 123
percentage_yield = 12.3
print 'reagant: %d produced %f%% yield' % \
    (reagant_id, percentage_yield)
```

reagant: 123 produced 12.30% yield

And string % (v1, v2, ...) for multiple values

```
reagant_id = 123
percentage_yield = 12.3
print 'reagant: %d produced %f%% yield' % \
    (reagant_id, percentage_yield)
```

reagant: 123 produced 12.30% yield

% operator turns double '%%' into single '%'

Use \n to represent a newline character

Use \n to represent a newline character

Use \' for single quote, \" for double quote

Use \n to represent a newline character

Use \' for single quote, \" for double quote

print 'There isn\'t time\nto do it right.'
There isn't time
to do it right.

Use \n to represent a newline character

Use \' for single quote, \" for double quote

print 'There isn\'t time\nto do it right.'

There isn't time
to do it right.

print "But you said,\n\"There is time to do it over.\""

But you said,
"There is time to do it over."

Use \\ for a literal \ character

Use \\ for a literal \ character

print 'Most mathematicians write a\\b instead of a%b.'
Most mathematicians write a\b instead of a%b.

Use \\ for a literal \ character

print 'Most mathematicians write a\\b instead of a%b.'
Most mathematicians write a\b instead of a%b.

Common pattern with *escape sequences*

Use \\ for a literal \ character

print 'Most mathematicians write a\\b instead of a%b.'
Most mathematicians write a\b instead of a%b.

Common pattern with *escape sequences*

– Use a character to mean "what follows is special"

Use \\ for a literal \ character

print 'Most mathematicians write a\\b instead of a%b.'
Most mathematicians write a\b instead of a%b.

Common pattern with *escape sequences*

– Use a character to mean "what follows is special"
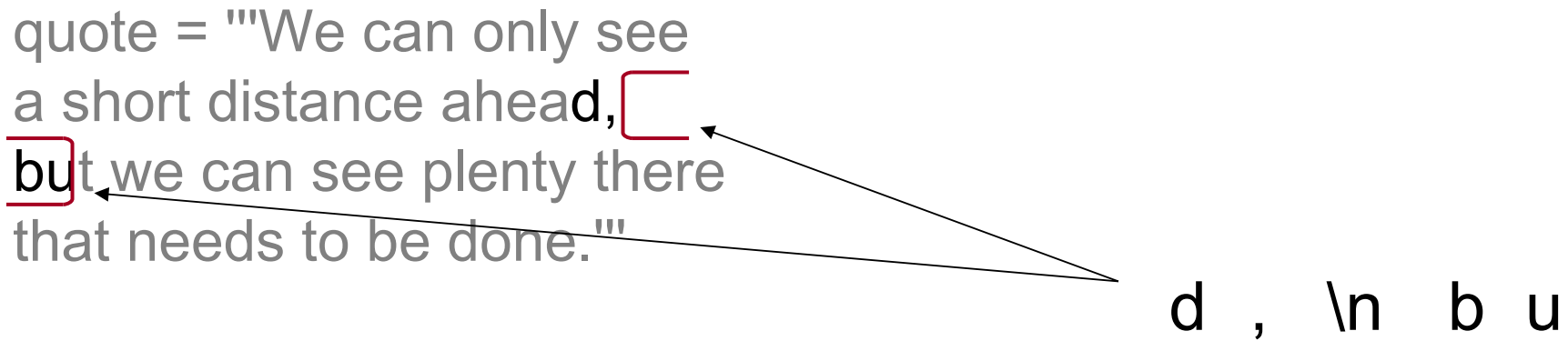
– Double it up to mean "that character itself"

Use triple quotes (either kind) for multi-line strings

Use triple quotes (either kind) for multi-line strings

```
quote = '''We can only see
a short distance ahead,
but we can see plenty there
that needs to be done.'''
```

Use triple quotes (either kind) for multi-line strings

quote = '''We can only see
a short distance ahead,
but we can see plenty there
that needs to be done.'''

d , \n b u

Use triple quotes (either kind) for multi-line strings

```
quote = '''We can only see
a short distance ahead,
but we can see plenty there
that needs to be done.'''

quote = 'We can only see\na short distance ahead\n' + \
    'but we can see plenty there\nthat needs to be done.'
```

Strings have methods

Strings have methods

```
name = 'newTON'
print name.capitalize(), name.upper(), name.lower(), name
```
Newton NEWTON newton newTON

Strings have methods

```
name = 'newTON'
print name.capitalize(), name.upper(), name.lower(), name
```
Newton NEWTON newton newTON
```
dna = 'acggtggtcac'
print dna.count('g'), dna.count('x')
```
4 0

Strings have methods

```
name = 'newTON'
print name.capitalize(), name.upper(), name.lower(), name
```
Newton NEWTON newton newTON
```
dna = 'acggtggtcac'
print dna.count('g'), dna.count('x')
```
4 0
```
print dna.find('t'), dna.find('t', 5), dna.find('x')
```
4 7 -1

Strings have methods

```
name = 'newTON'
print name.capitalize(), name.upper(), name.lower(), name
```
Newton NEWTON newton newTON
```
dna = 'acggtggtcac'
print dna.count('g'), dna.count('x')
```
4 0
```
print dna.find('t'), dna.find('t', 5), dna.find('x')
```
4 7 -1
```
print dna.replace('t', 'x'), dna
```
acggxggxcac acggtggtcac

Strings have methods

```
name = 'newTON'
print name.capitalize(), name.upper(), name.lower(), name
```
Newton NEWTON newton newTON
```
dna = 'acggtggtcac'
print dna.count('g'), dna.count('x')
```
4 0
```
print dna.find('t'), dna.find('t', 5), dna.find('x')
```
4 7 -1
```
print dna.replace('t', 'x')
```
acggxggxcac acggtggtcac
```
print dna.replace('gt', '')
```
acggcac

Can chain method calls together

Can chain method calls together

```
element = 'cesium'
print element.upper().center(10, '.')
```

Can chain method calls together

```
element = 'cesium'
print element.upper().center(10, '.')
```

convert to upper case

Python

Strings

Can chain method calls together

```
element = 'cesium'
print element.upper().center(10, '.')
```

center in a field

10 characters wide

Can chain method calls together

```
element = 'cesium'
print element.upper().center(10, '.')
```
..CESIUM..

narrated by

Dominique Vuvan

October 2010