Science & Technology Facilities Council | UK Research and Innovation | NERC SCIENCE OF THE ENVIRONMENT

Python

Strings

Centre for Environmental Data Analysis
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

software carpentry

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

National Centre for Earth Observation
NATURAL ENVIRONMENT RESEARCH COUNCIL

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 (based on their Unicode value)

Strings are compared character by character

from left to right (based on their Unicode value)

```
print('a' < 'b')
True
```

Strings are compared character by character

from left to right (based on their Unicode value)

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

Strings are compared character by character

from left to right (based on their Unicode value)

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

Strings are compared character by character

from left to right (based on their Unicode value)

```
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 (based on their Unicode value)

```
print('a' < 'b')
True
print('ab' < 'abc')
True
print('1' < '9')
True
print('100' < '9')
True
print('A' < 'a’)  # equates to(U+0041 < U+0061)
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

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
```

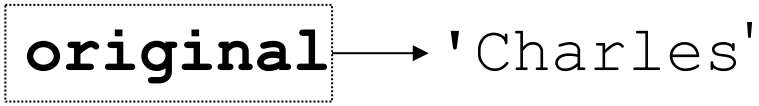
Concatenation always produces a new string

Use + to concatenate strings

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

Concatenation always produces a new string

```
original = 'Charles'
```

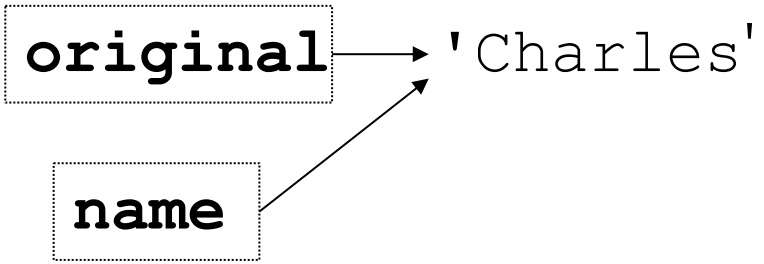


Use + to concatenate strings

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

Concatenation always produces a new string

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

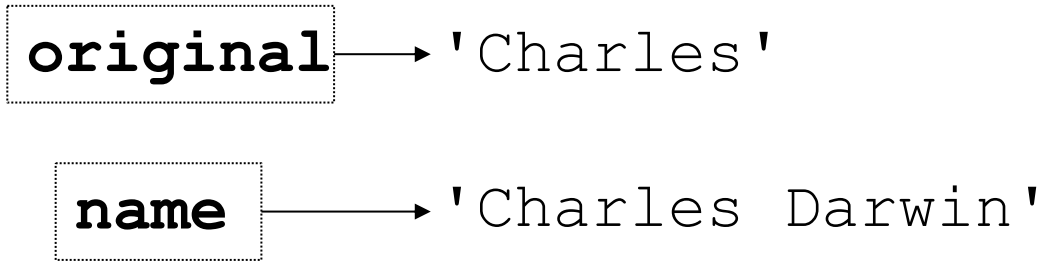


Use + to concatenate strings

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

Concatenation always produces a new string

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



Strings can be formatted with +…

Strings can be formatted with +…

```
print('reagent: ' + str(reagent_id) + ' produced ' + \
      str(percentage_yield) + '% yield')
```

Strings can be formatted with +…

```
print('reagent: ' + str(reagent_id) + ' produced ' + \
      str(percentage_yield) + '% yield')
```

There's a better way...

f-strings!!! (>python 3.6)

f-strings!!! (>python 3.6)

The example on the last slide:

```
print('reagent: ' + str(reagent_id) + ' produced ' + \
      str(percentage_yield) + '% yield')
```

f-strings!!! (>python 3.6)

The example on the last slide:

```
print('reagent: ' + str(reagent_id) + ' produced ' + \
      str(percentage_yield) + '% yield')
```

Using f-strings:

```
print(f'reagent: {reagent_id} produced ' \
      f'{percentage_yield} % yield')
```

f-strings!!! (>python 3.6)

The example on the last slide:

```
print('reagent: ' + str(reagent_id) + ' produced ' + \
      str(percentage_yield) + '% yield')
```

Using f-strings:

```
print(f'reagent: {reagent_id} produced ' \
      f'{percentage_yield} % yield')
```

Use {{ to get a { in the output:

```
print(f'reagent: {{reagent_id}} ')
reagent: {123}
```

F-strings can also format the output:

```
yield = 0.34
```

Other format options:
https://docs.python.org/3/library/string.html#format-specification-mini-language

F-strings can also format the output:

```
yield = 0.34
```

As a percentage:

```
print(f'yield: {yield:.2%}')
'yield: 34.00%'
```

Change precision:

```
print(f'yield: {yield:.1f}')
'yield: 0.3'
```

Add padding:

```
print(f'yield: {yield:07.2%}')
'yield: 034.00%'
```

Other format options:
https://docs.python.org/3/library/string.html#format-specification-mini-language

In older code you may come across two different styles

In older code you may come across two different styles

.format()

```
'{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'

'Coordinates: {latitude}, {longitude}'.format(
        latitude='37.24N', longitude='-115.81W')

'Coordinates: 37.24N, -115.81W'
```

In older code you may come across two different styles

.format()

```
'{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'

'Coordinates: {latitude}, {longitude}'.format(
        latitude='37.24N', longitude='-115.81W')

'Coordinates: 37.24N, -115.81W'
```

You could, but f-strings are faster and clearer

% format

```
print('reagent: %d' % 123)
reagant: 123

print('Name: %s; weight: %.2fkg' % ('Bert', 122))
Name: Bert; weight: 122.00kg
```

% format

```
print('reagent: %d' % 123)
reagant: 123

print('Name: %s; weight: %.2fkg' % ('Bert', 122))
Name: Bert; weight: 122.00kg
```

This is a hang-over from python 2.7

This approach is now discouraged.

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"

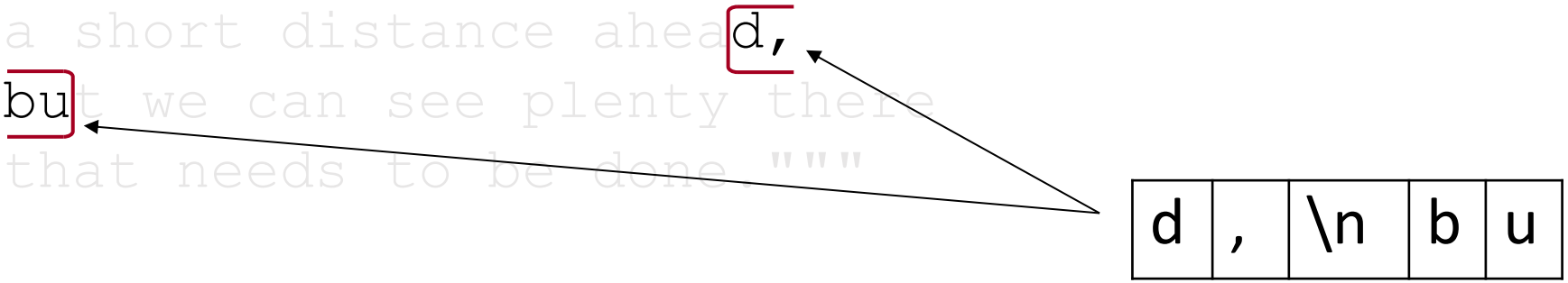
Use triple quotes (either kind) for multi-line strings

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."""
```



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,\nbut 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())
```

```
Newton NEWTON newton
```

Strings have methods

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

Strings have methods

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

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..
```

The power of **regular expressions**

When programming in any language you will want to know about *regular expressions* – for advanced string/text processing. In Python use the "re" library. Example uses are:

`/<([A-Z][A-Z0-9]*)\b[^>]*>(.*?)</\1>/` Matches the opening and closing pair of any HTML tag; captures tag name and content.

`/b[aeiou]+t/` Matches "bat" and "bit" etc, but also "boot" and "boat".

`/(\[0-9]{1,3})\.(\[0-9]{1,3})\.(\[0-9]{1,3})\.(\[0-9]{1,3})/`
Matches any IP address ((e.g. 66.70.7.154) and captures the each number for re-use.

See: https://docs.python.org/3.7/howto/regex.html