CMPT 165 INTRODUCTION TO THE INTERNET AND THE WORLD WIDE WEB

Unit 7
Python Basics

SFU

Python Basics

To learn Python basics we will use slides from:

[University of Cambridge - Python: Introduction for Absolute Beginners]

and are © the University of Cambridge.

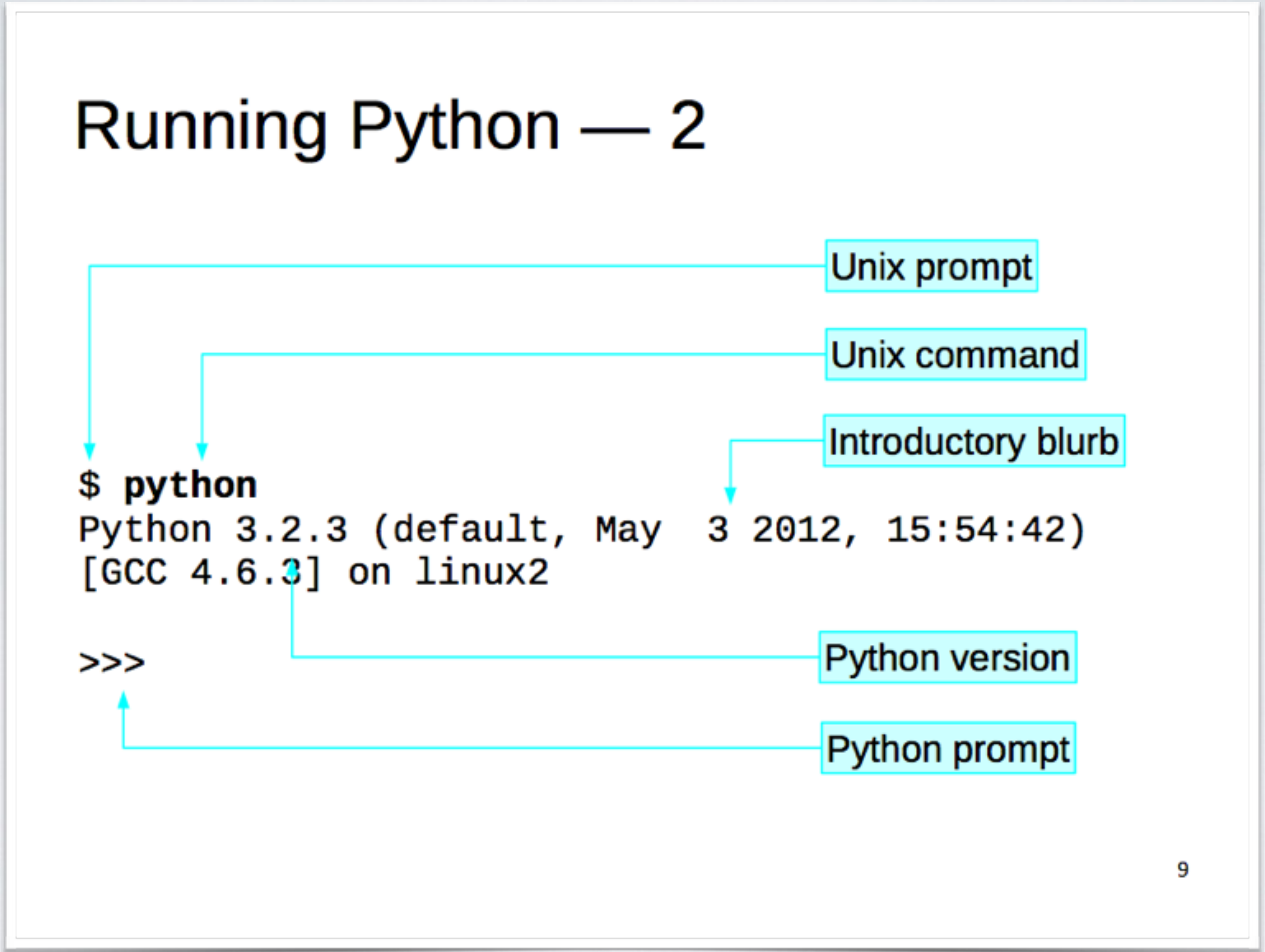
**Some slides have been skipped and some have been modified.*

All example python scripts are at:

http://www.cs.sfu.ca/CourseCentral/165/smakonin/examples/python/

Let's begin…

Python Basics



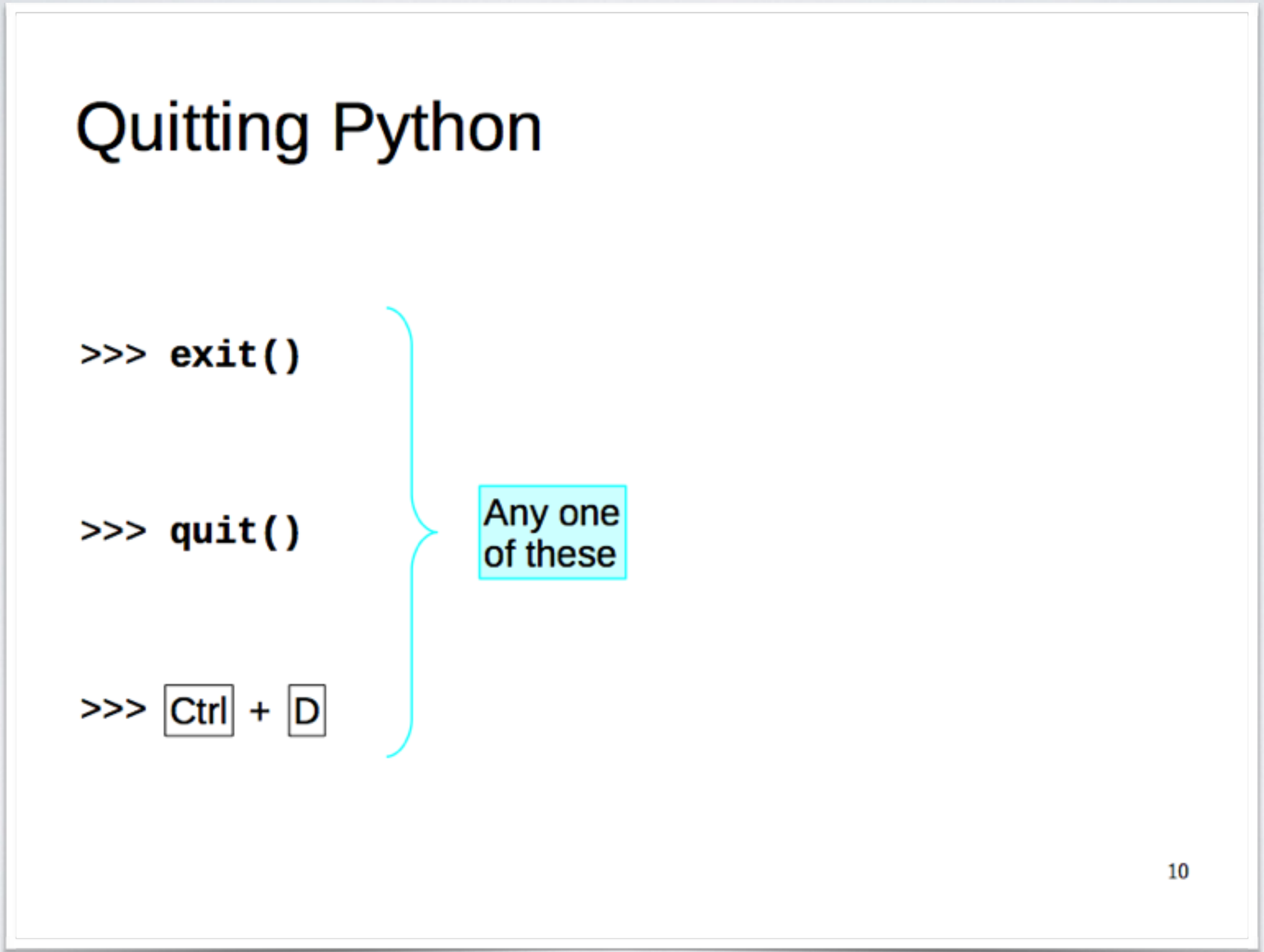
Running Python — 2

```
$ python
Python 3.2.3 (default, May  3 2012, 15:54:42)
[GCC 4.6.3] on linux2

>>>
```

Unix prompt
Unix command
Introductory blurb
Python version
Python prompt
9

Slide taken from University of Cambridge's Python AB Course

Python Basics



Quitting Python

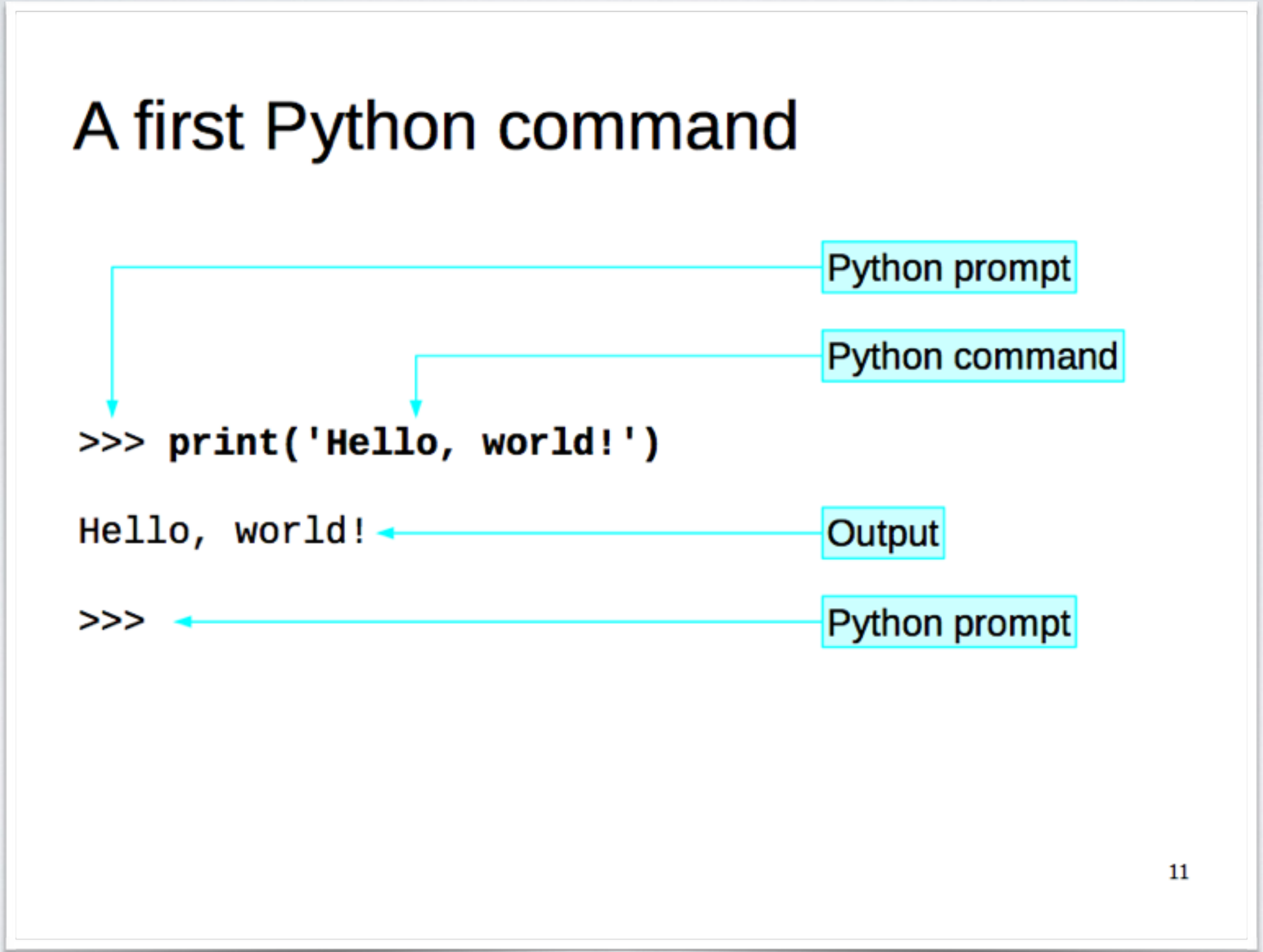
```
>>> exit()

>>> quit()

>>> Ctrl + D
```

$\left.\begin{array}{l}\texttt{>>> exit()}\\ \texttt{>>> quit()}\\ \texttt{>>> Ctrl + D}\end{array}\right\}$ Any one of these

10

Slide taken from University of Cambridge's Python AB Course

Python Basics

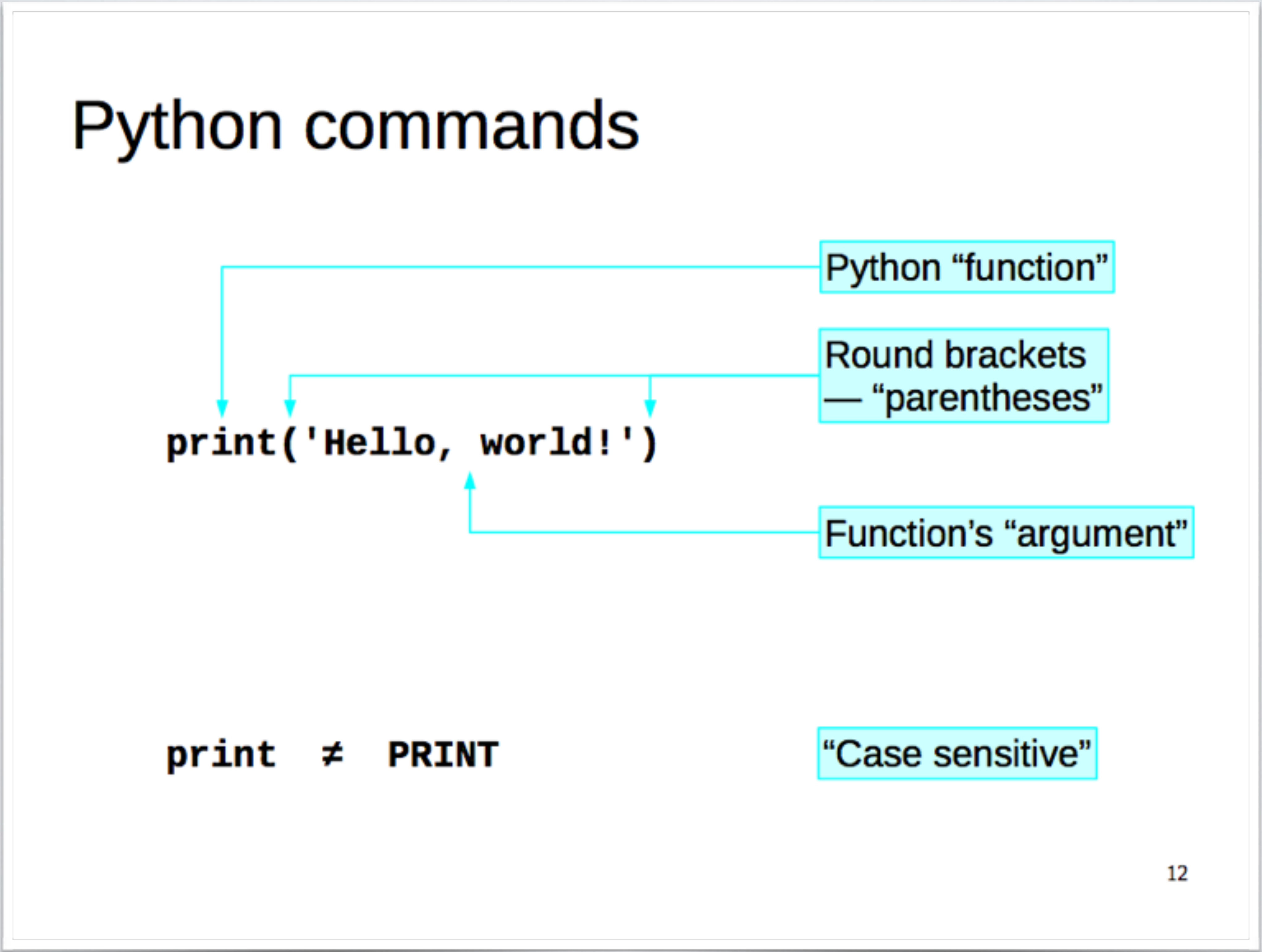


A first Python command

```
>>> print('Hello, world!')
Hello, world!
>>>
```

Slide taken from University of Cambridge's Python AB Course

Python Basics



Python commands

```
print('Hello, world!')
```

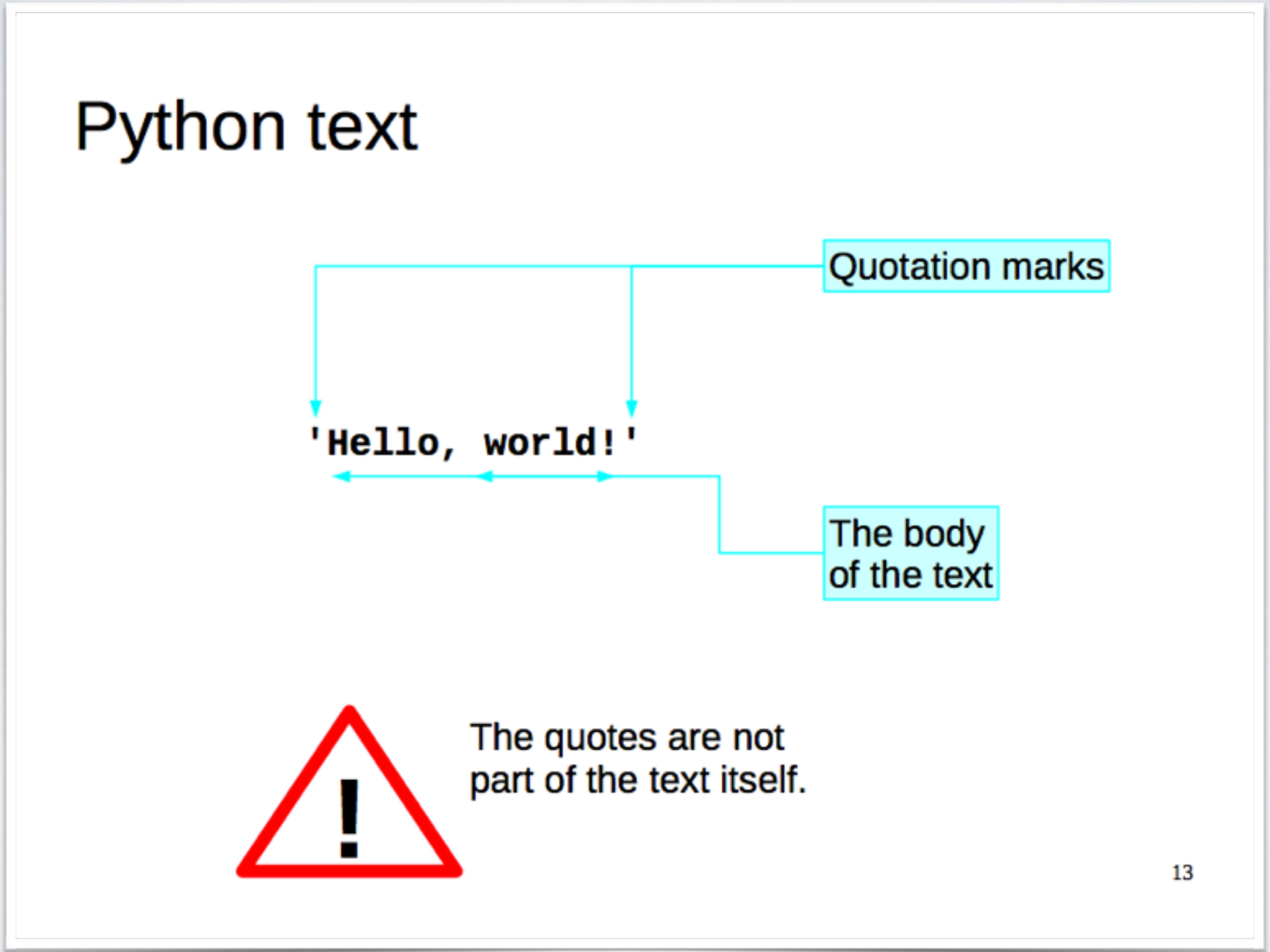
Python "function"
Round brackets — "parentheses"
Function's "argument"

```
print  ≠  PRINT
```

"Case sensitive"

12

Python Basics



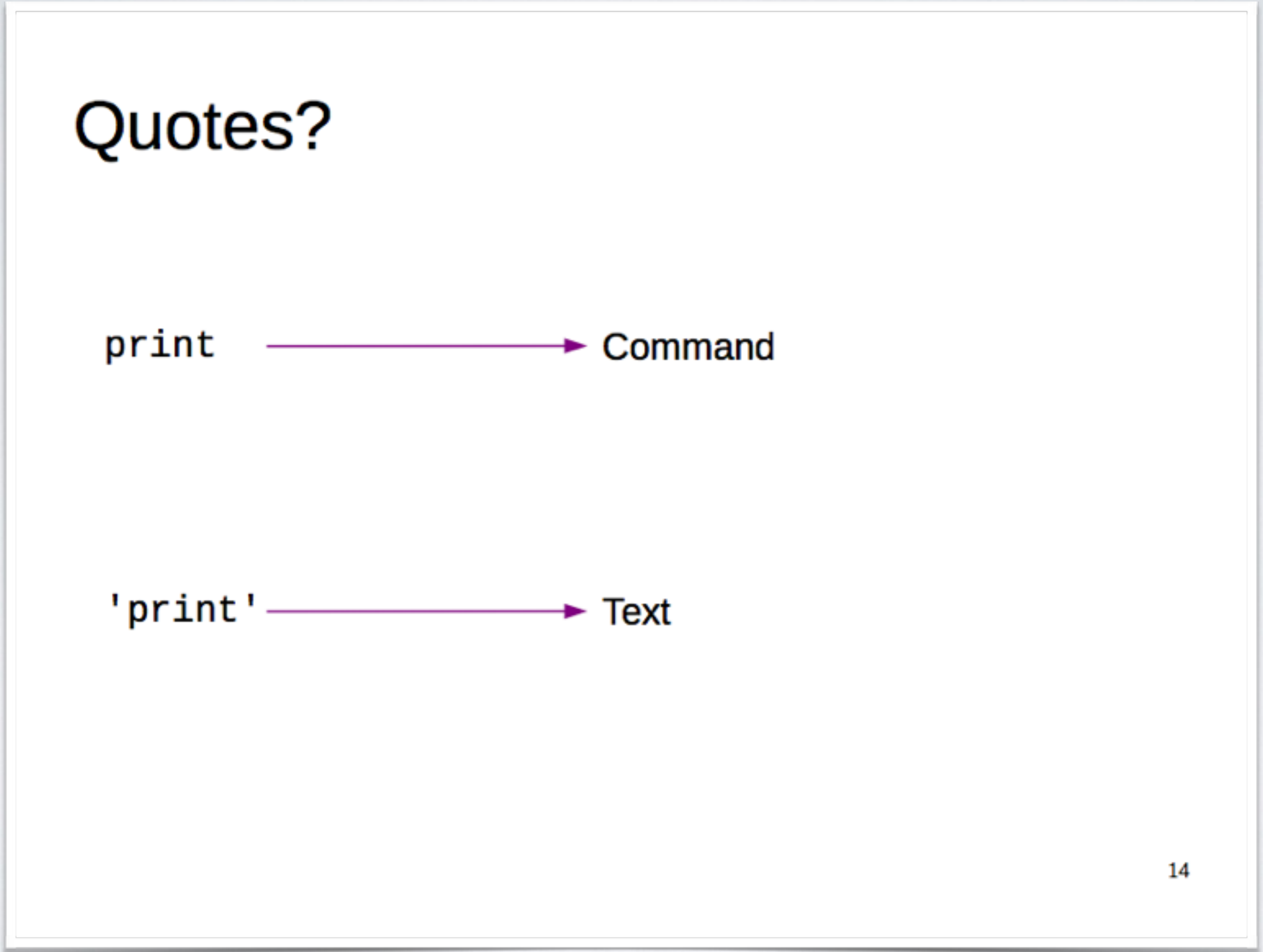
Python text

```
'Hello, world!'
```

Quotation marks
The body of the text

The quotes are not part of the text itself.

Python Basics

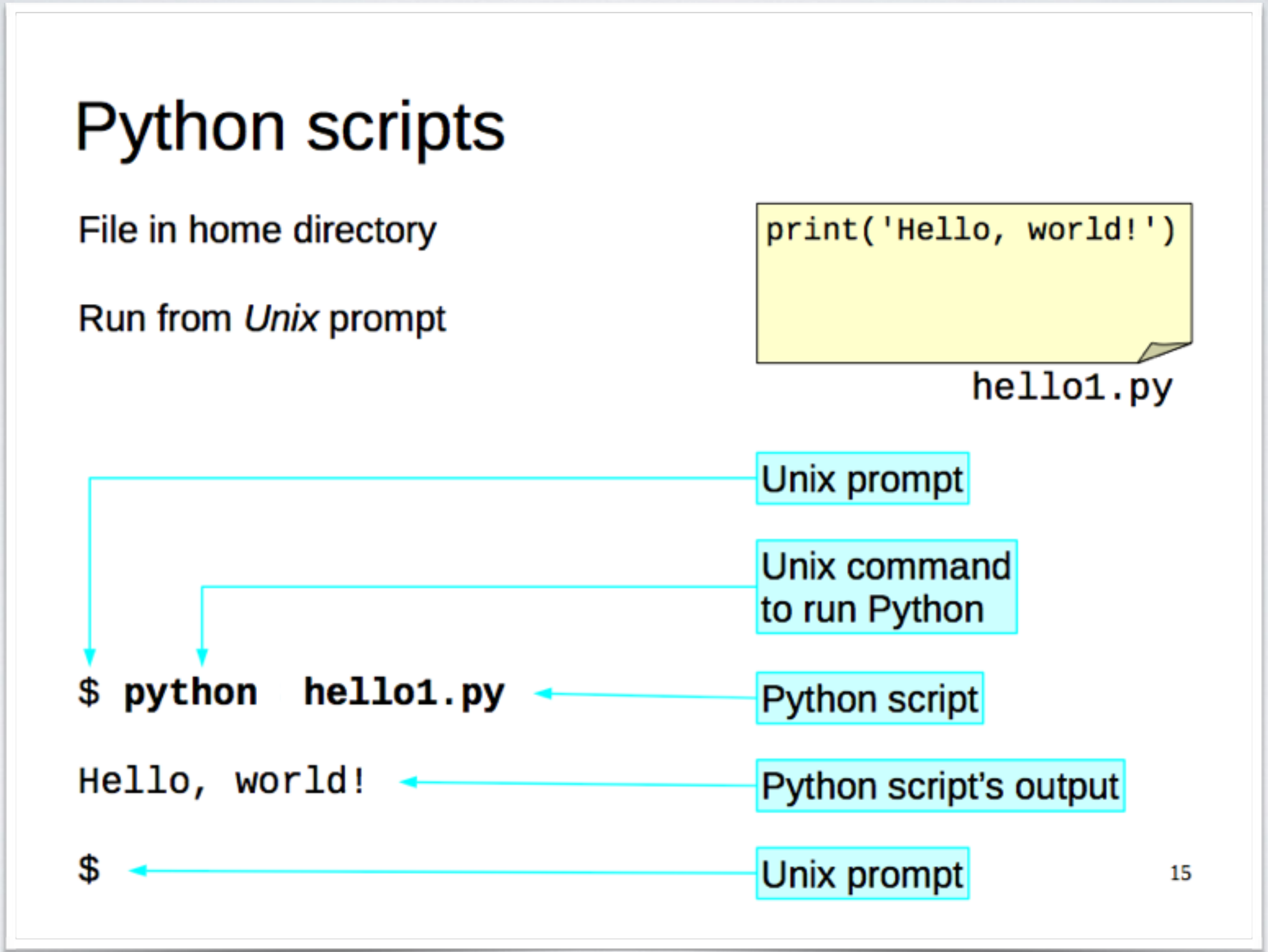

Quotes?

$$\begin{tikzcd}
\texttt{print} \arrow[r] & \text{Command} \\
\texttt{'print'} \arrow[r] & \text{Text}
\end{tikzcd}$$

14

Slide taken from University of Cambridge's Python AB Course

Python Basics



Python scripts

File in home directory

Run from *Unix* prompt

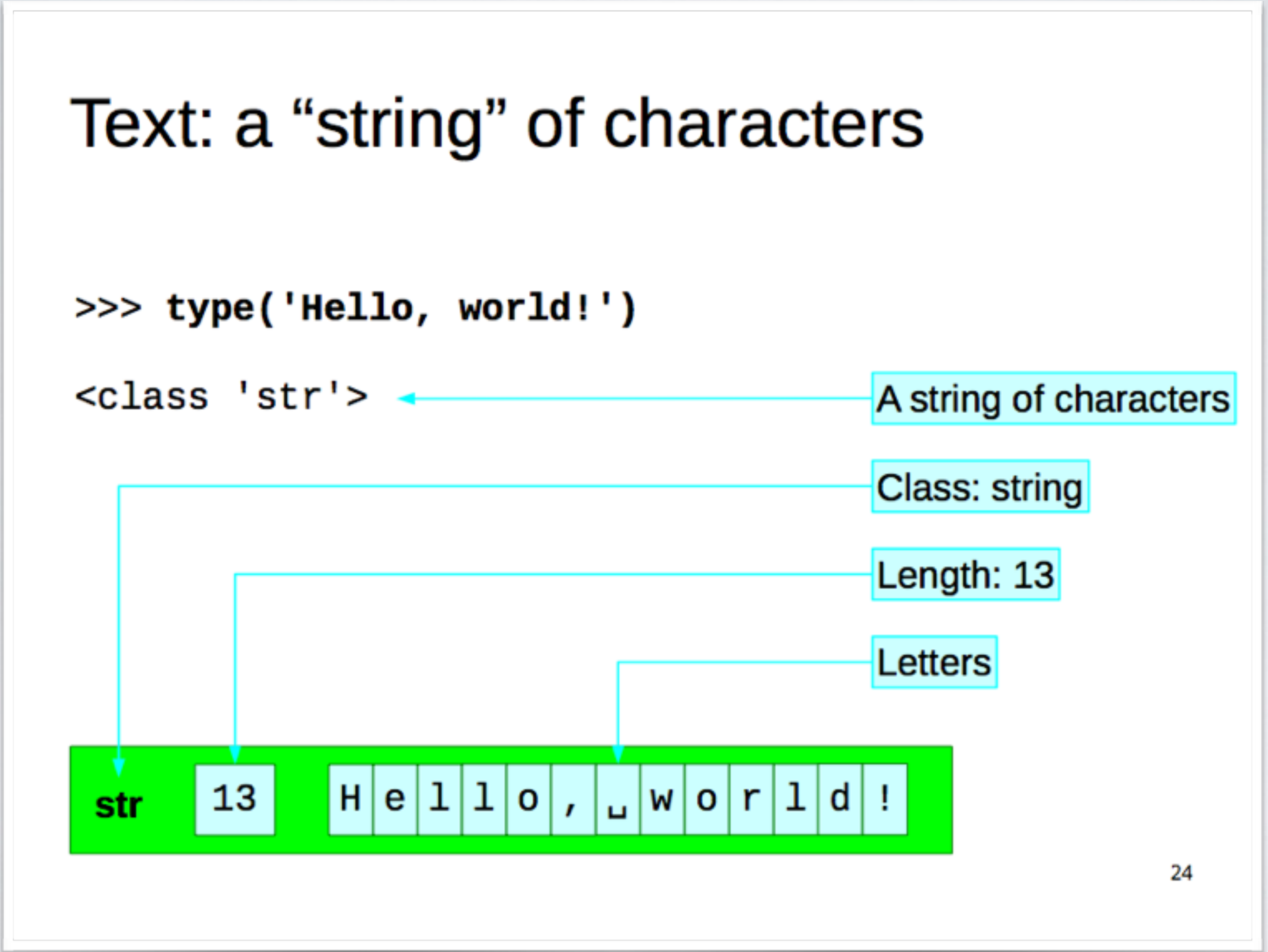
```
print('Hello, world!')
```

hello1.py

```
$ python  hello1.py

Hello, world!

$
```

Unix prompt
Unix command to run Python
Python script
Python script's output
Unix prompt

Python Basics



Text: a "string" of characters

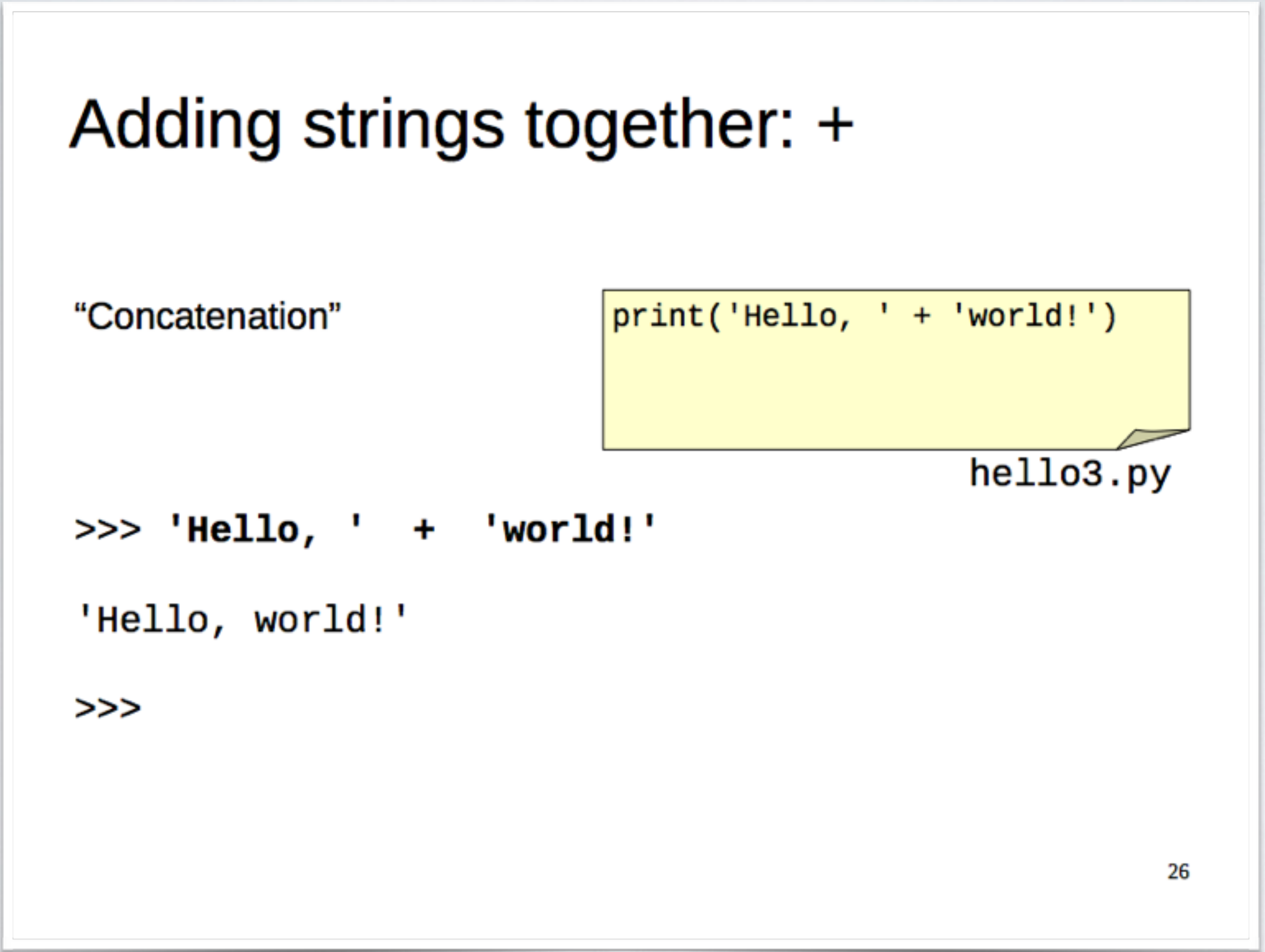
```
>>> type('Hello, world!')
<class 'str'>
```

A string of characters
Class: string
Length: 13
Letters
str
13
H e l l o , ␣ w o r l d !
24

Slide taken from University of Cambridge's Python AB Course

Copyright © 2014 by Stephen Makonin

Python Basics



Adding strings together: +

"Concatenation"

```
print('Hello, ' + 'world!')
```

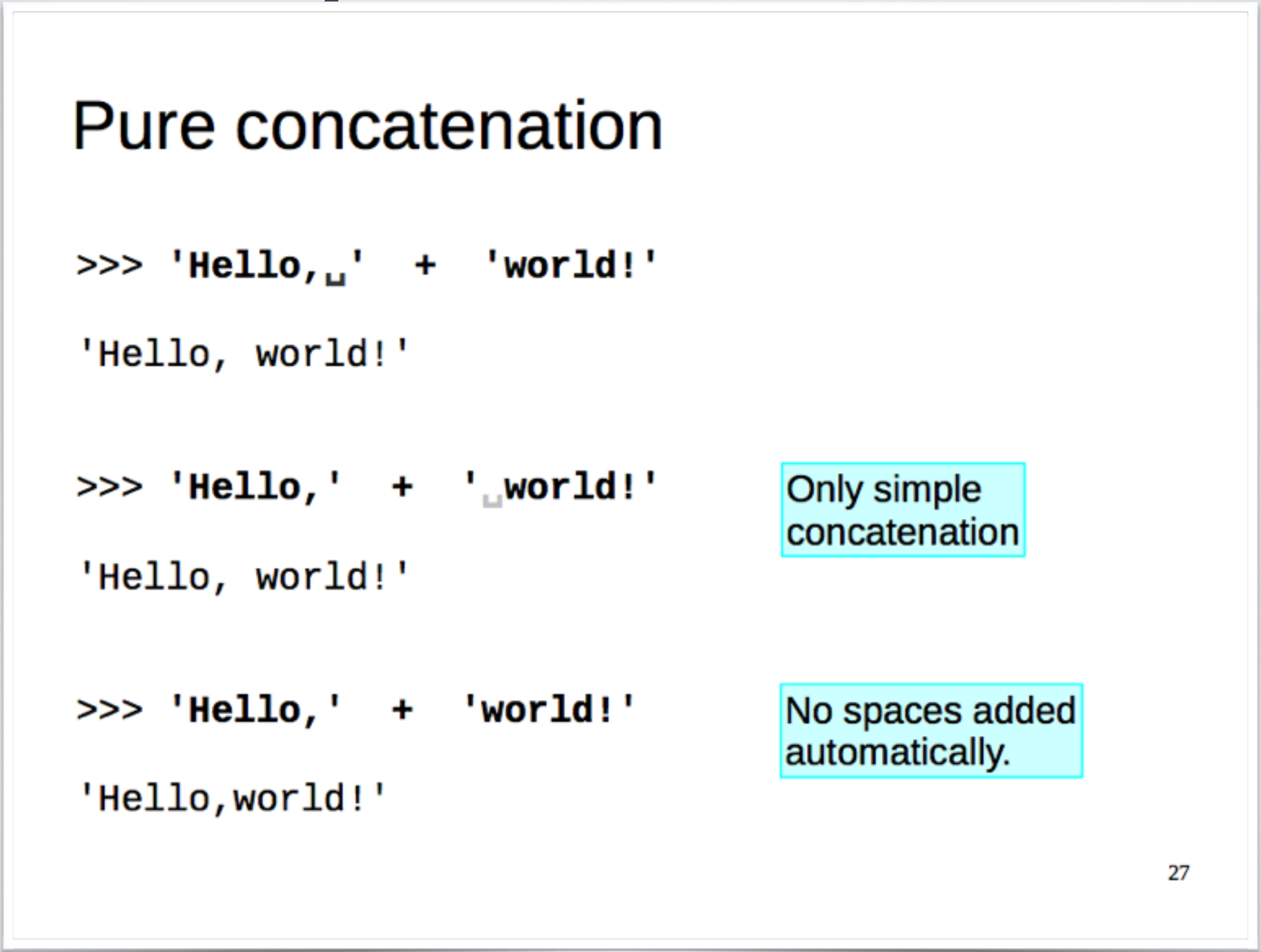
hello3.py

```
>>> 'Hello, '  +  'world!'
'Hello, world!'
>>>
```

26

Slide taken from University of Cambridge's Python AB Course

Python Basics



Pure concatenation

```
>>> 'Hello,␣' + 'world!'
'Hello, world!'

>>> 'Hello,' + '␣world!'
'Hello, world!'

>>> 'Hello,' + 'world!'
'Hello,world!'
```

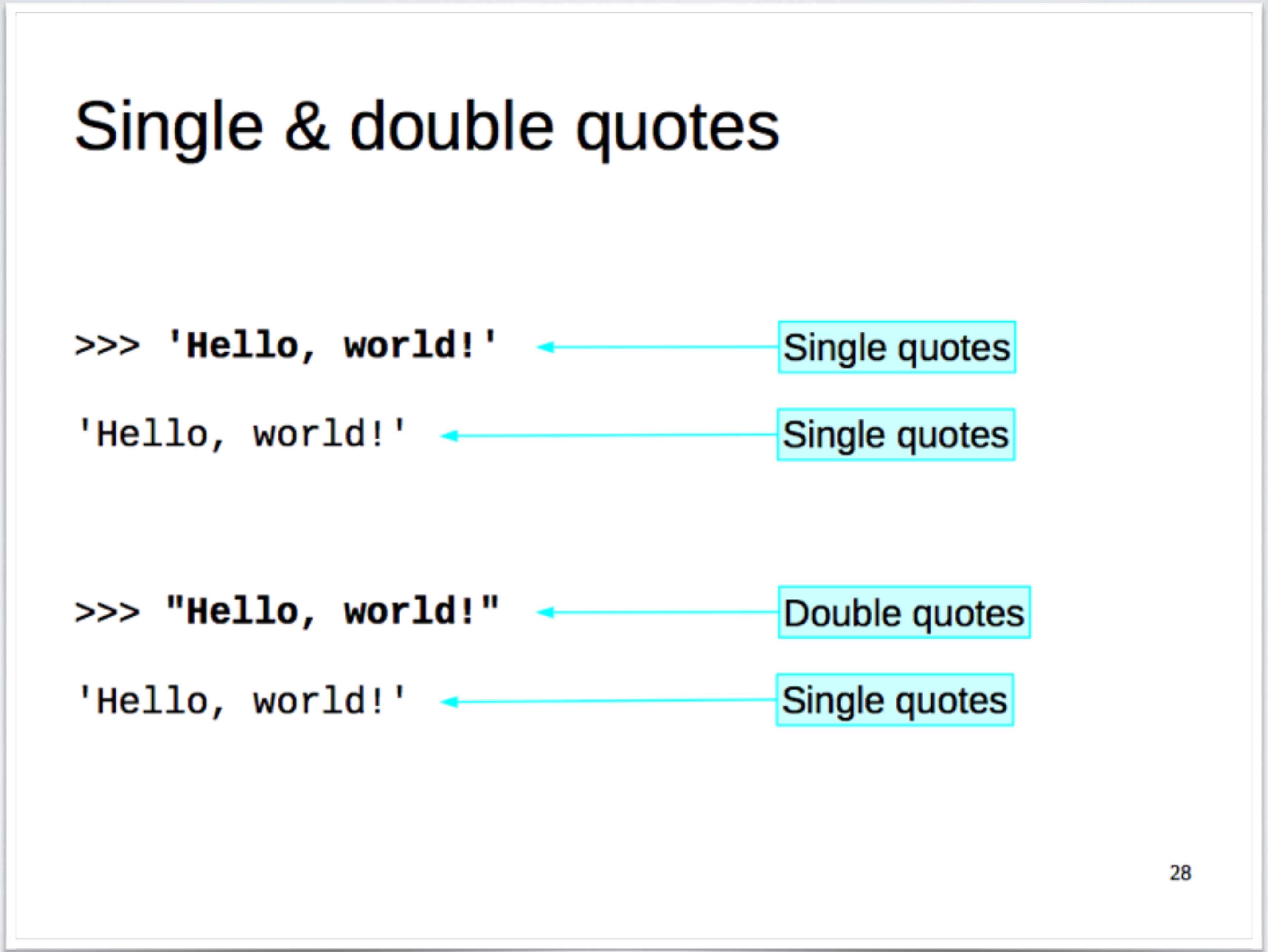
Only simple concatenation

No spaces added automatically.

27

Slide taken from University of Cambridge's Python AB Course

Python Basics



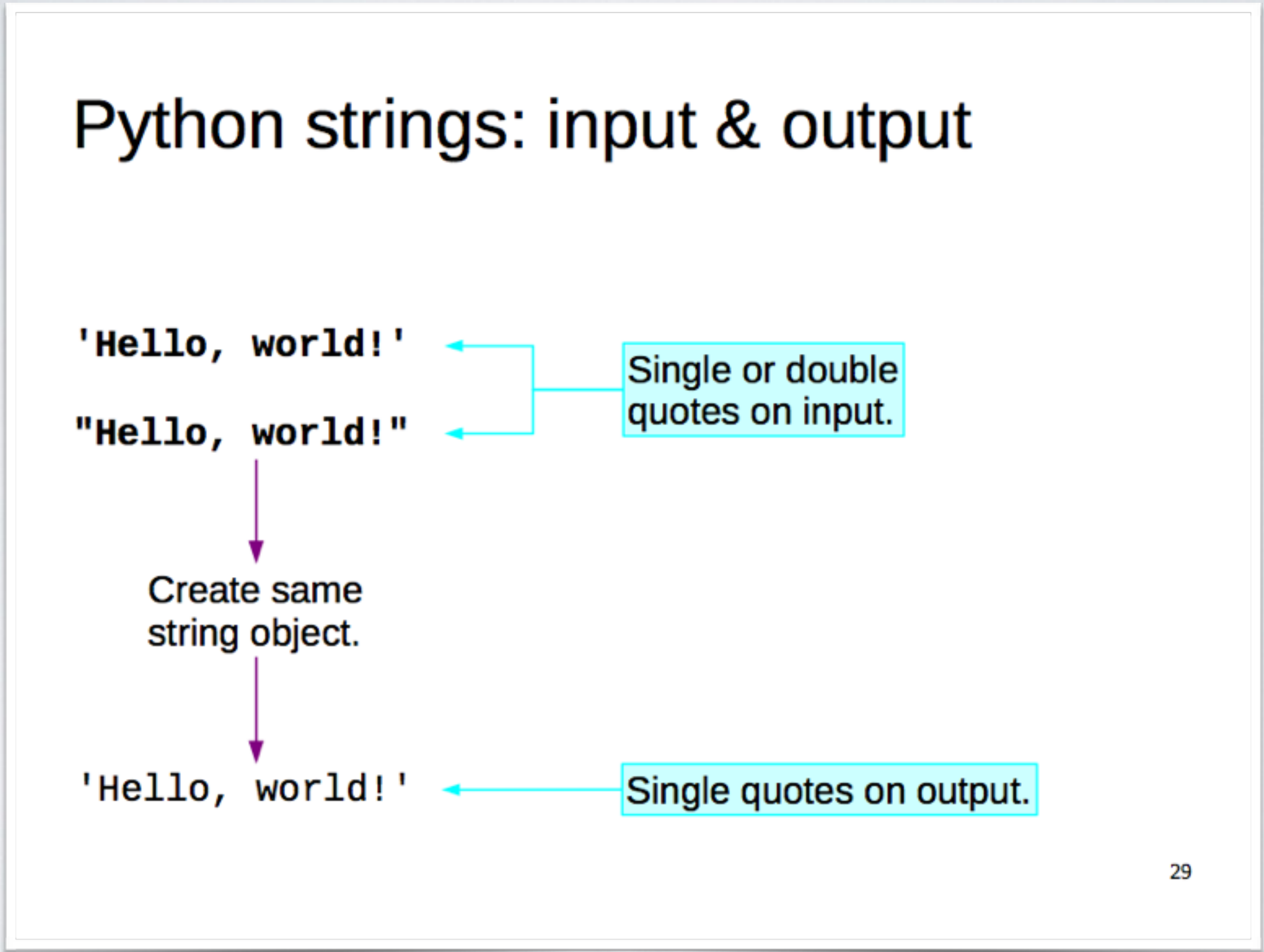
Single & double quotes

```
>>> 'Hello, world!'
'Hello, world!'
```

```
>>> "Hello, world!"
'Hello, world!'
```

Single quotes
Single quotes
Double quotes
Single quotes
28

Python Basics



Python strings: input & output

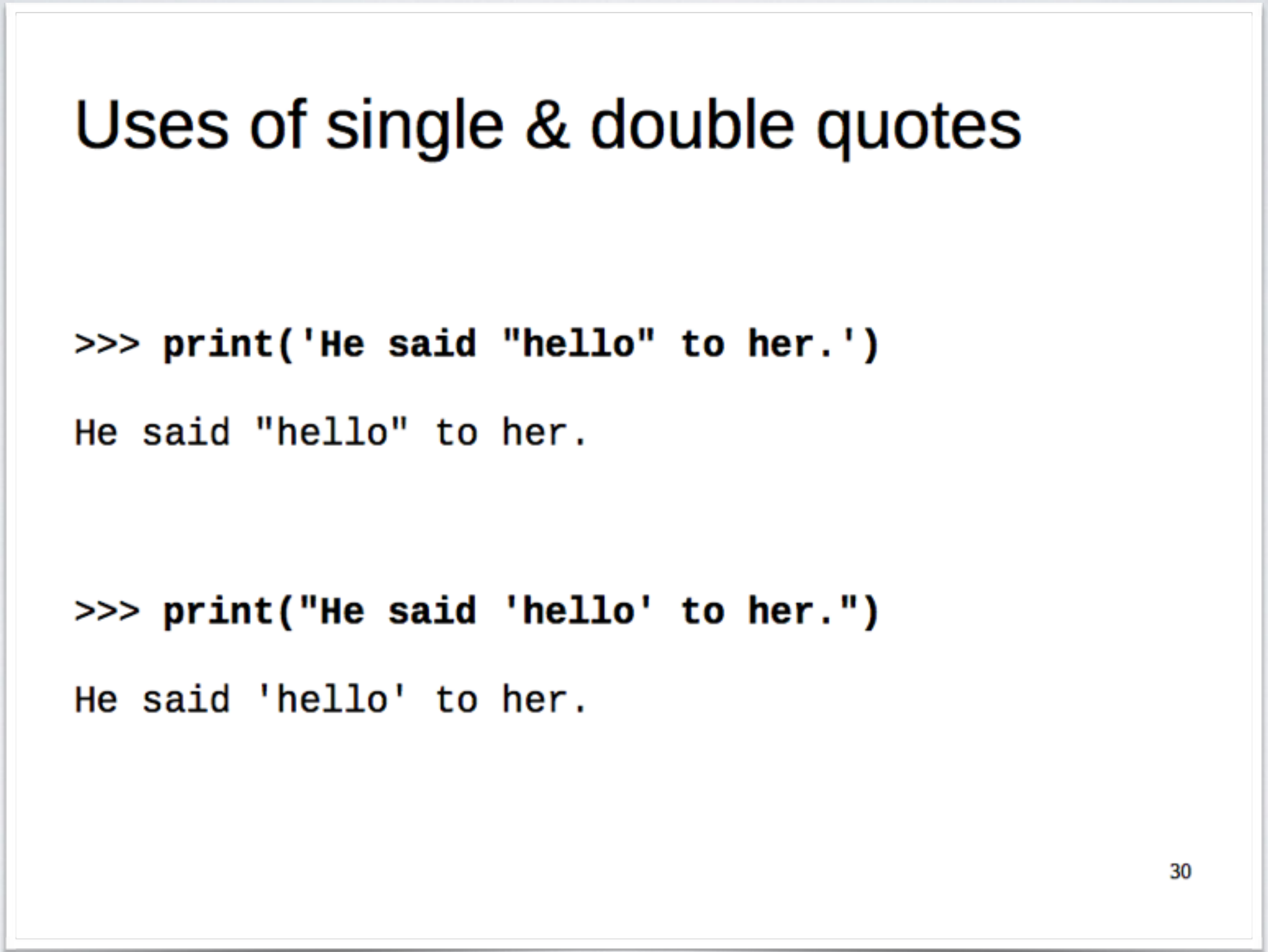
```
'Hello, world!'
"Hello, world!"
```

```
'Hello, world!'
```

29

Slide taken from University of Cambridge's Python AB Course

Python Basics



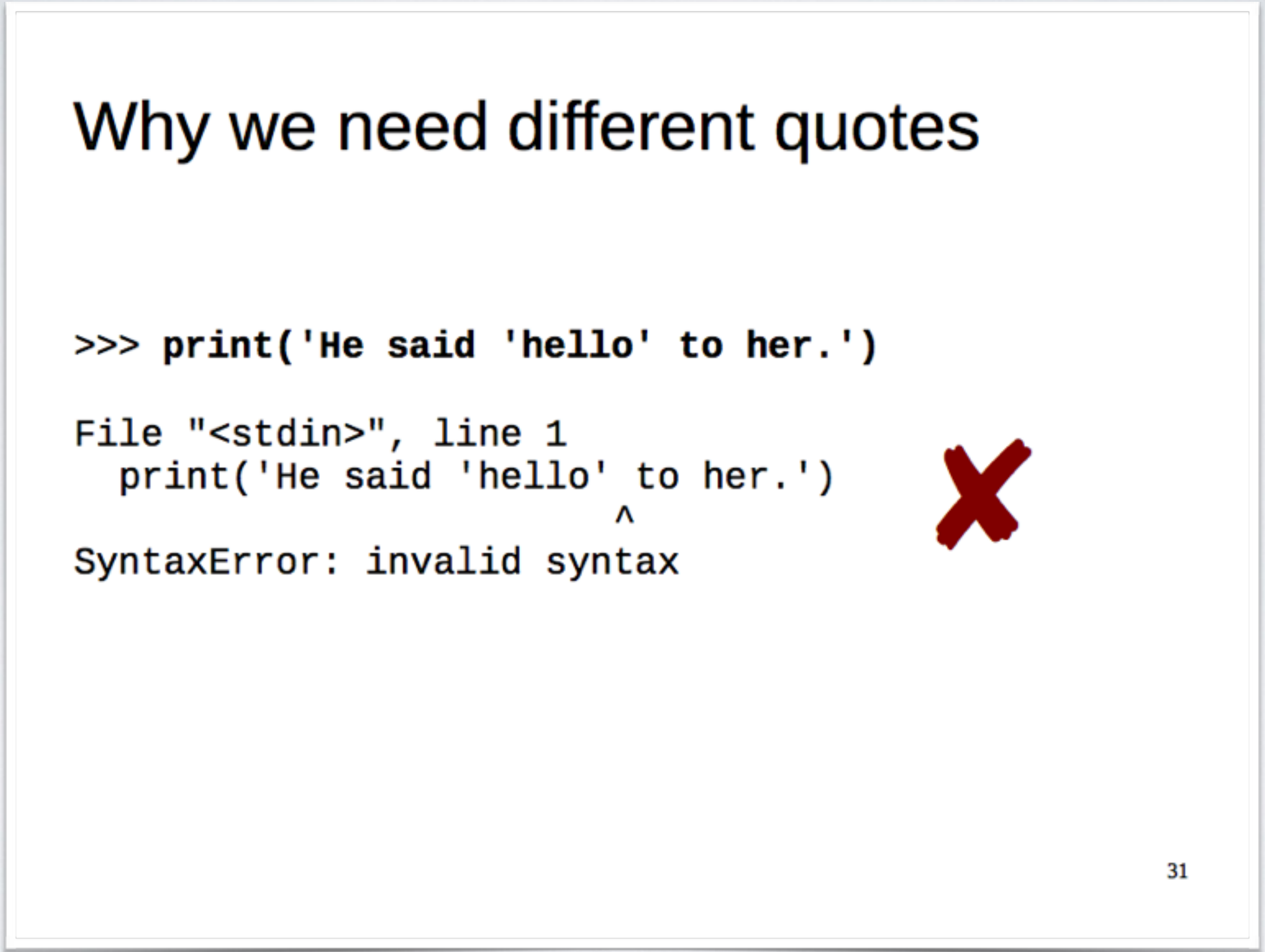
Uses of single & double quotes

```
>>> print('He said "hello" to her.')
He said "hello" to her.

>>> print("He said 'hello' to her.")
He said 'hello' to her.
```

30

Slide taken from University of Cambridge's Python AB Course

Python Basics



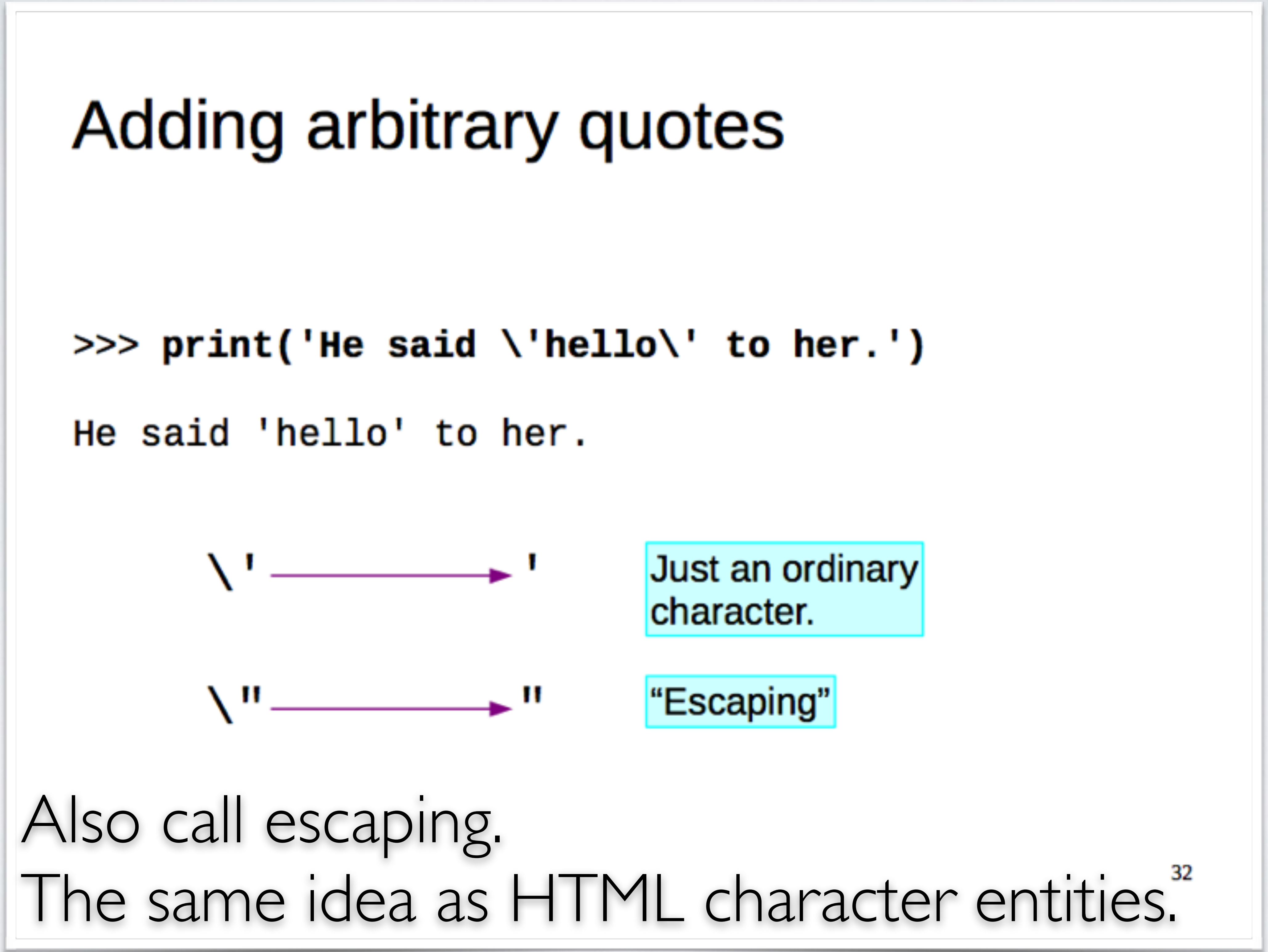
Why we need different quotes

```
>>> print('He said 'hello' to her.')

File "<stdin>", line 1
  print('He said 'hello' to her.')
                        ^
SyntaxError: invalid syntax
```

✗

31

Python Basics



Adding arbitrary quotes

```
>>> print('He said \'hello\' to her.')

He said 'hello' to her.
```

$$\backslash' \longrightarrow '$$

Just an ordinary character.

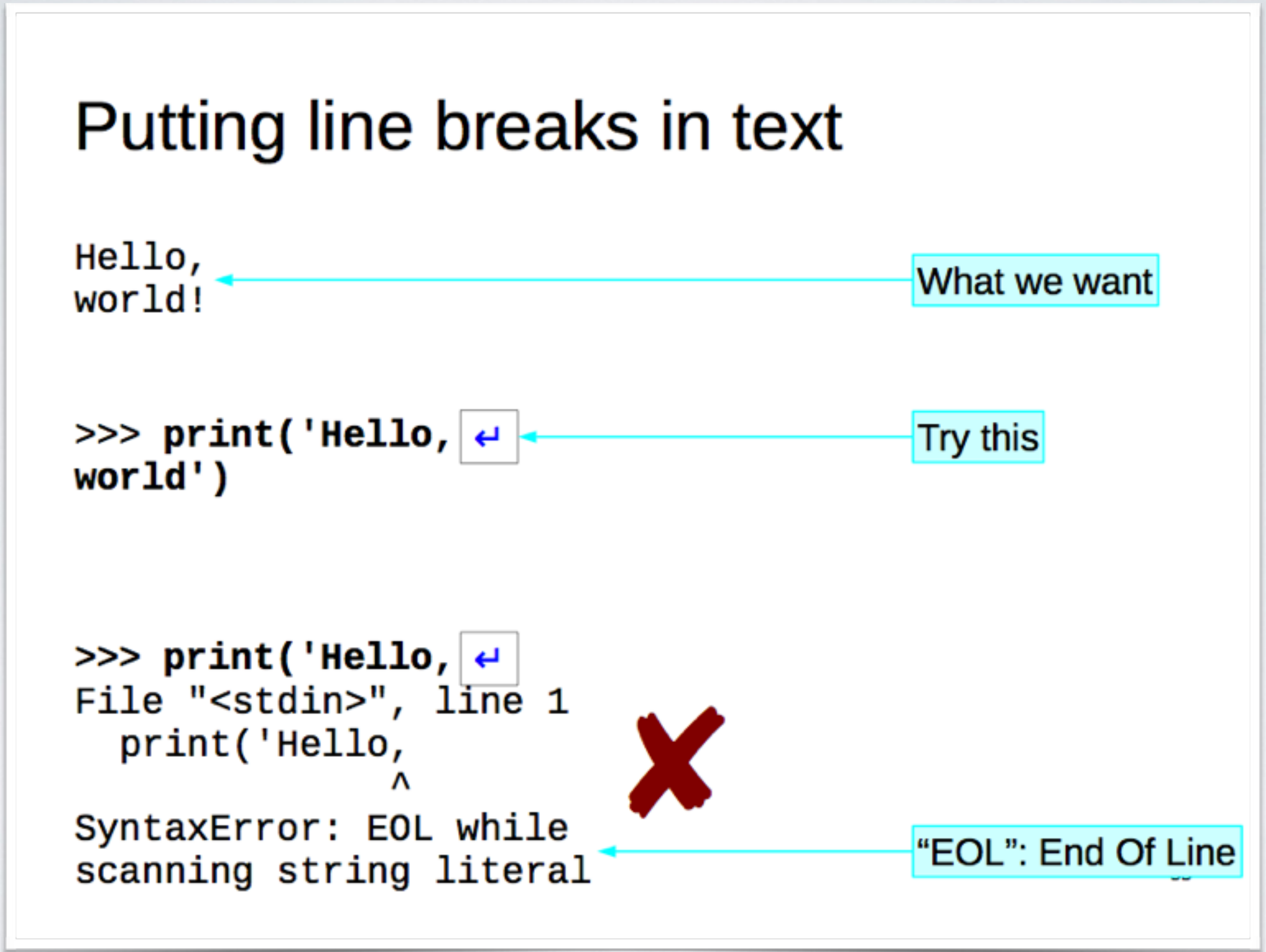
$$\backslash" \longrightarrow "$$

"Escaping"

32

Also call escaping.
The same idea as HTML character entities.

Python Basics



Putting line breaks in text

```
Hello,
world!
```

What we want

```
>>> print('Hello, ↵
world')
```

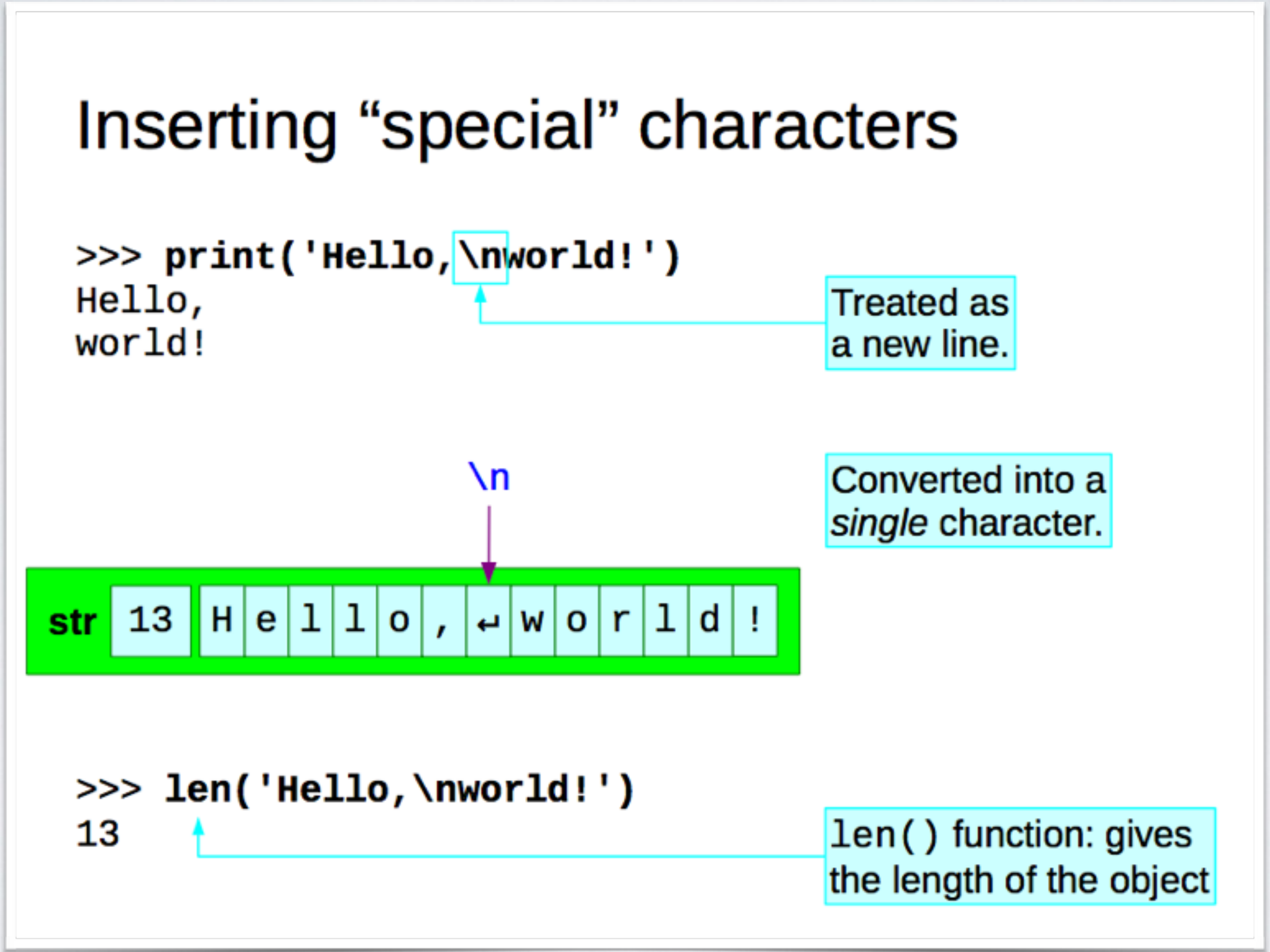
Try this

```
>>> print('Hello, ↵
File "<stdin>", line 1
  print('Hello,
              ^
SyntaxError: EOL while
scanning string literal
```

✗
"EOL": End Of Line

Slide taken from University of Cambridge's Python AB Course

Python Basics



Inserting "special" characters

```
>>> print('Hello,\nworld!')
Hello,
world!
```

Treated as a new line.

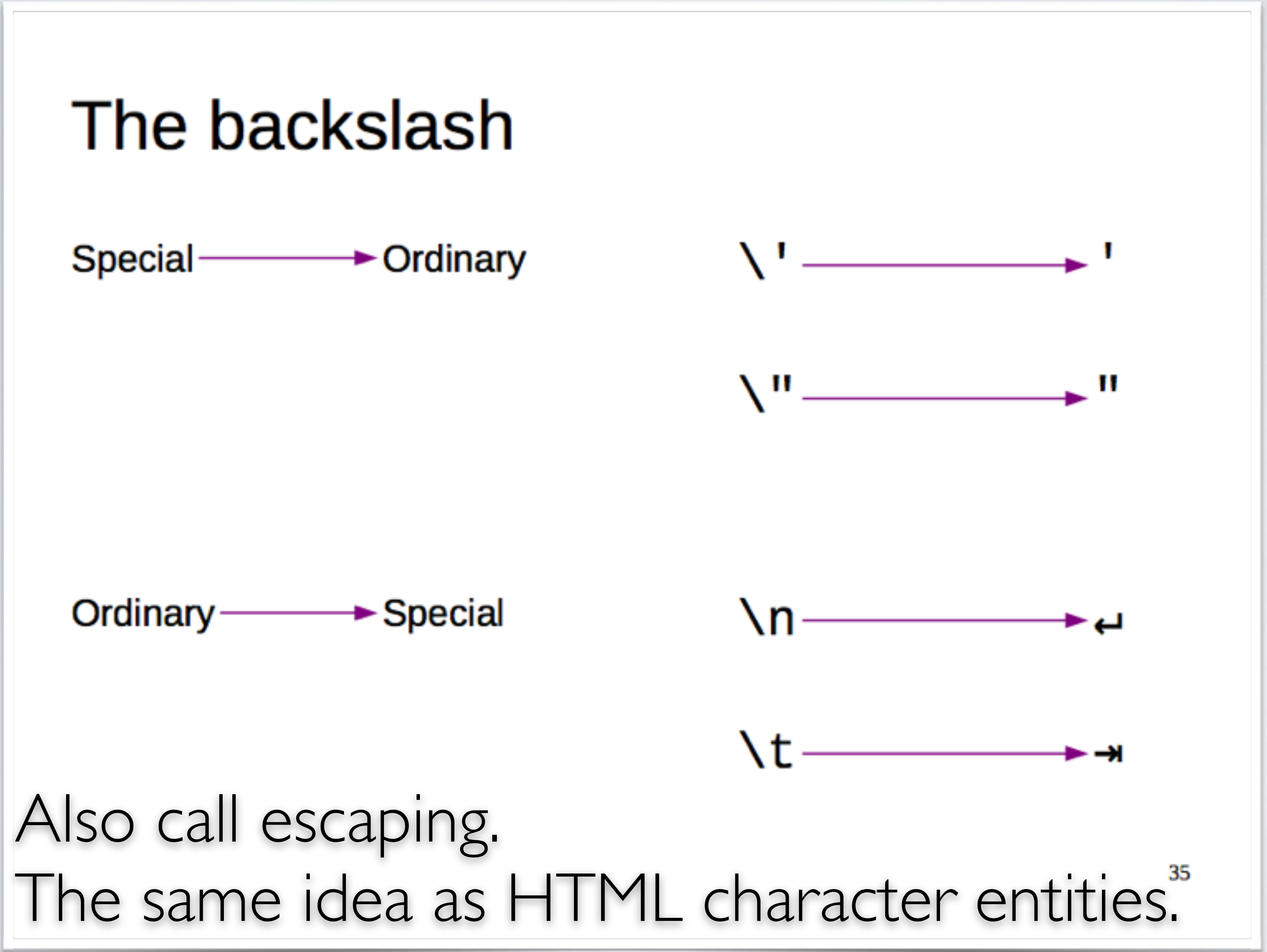
Converted into a *single* character.

\n
str
13
H e l l o , ↵ w o r l d !

```
>>> len('Hello,\nworld!')
13
```

`len()` function: gives the length of the object

Python Basics



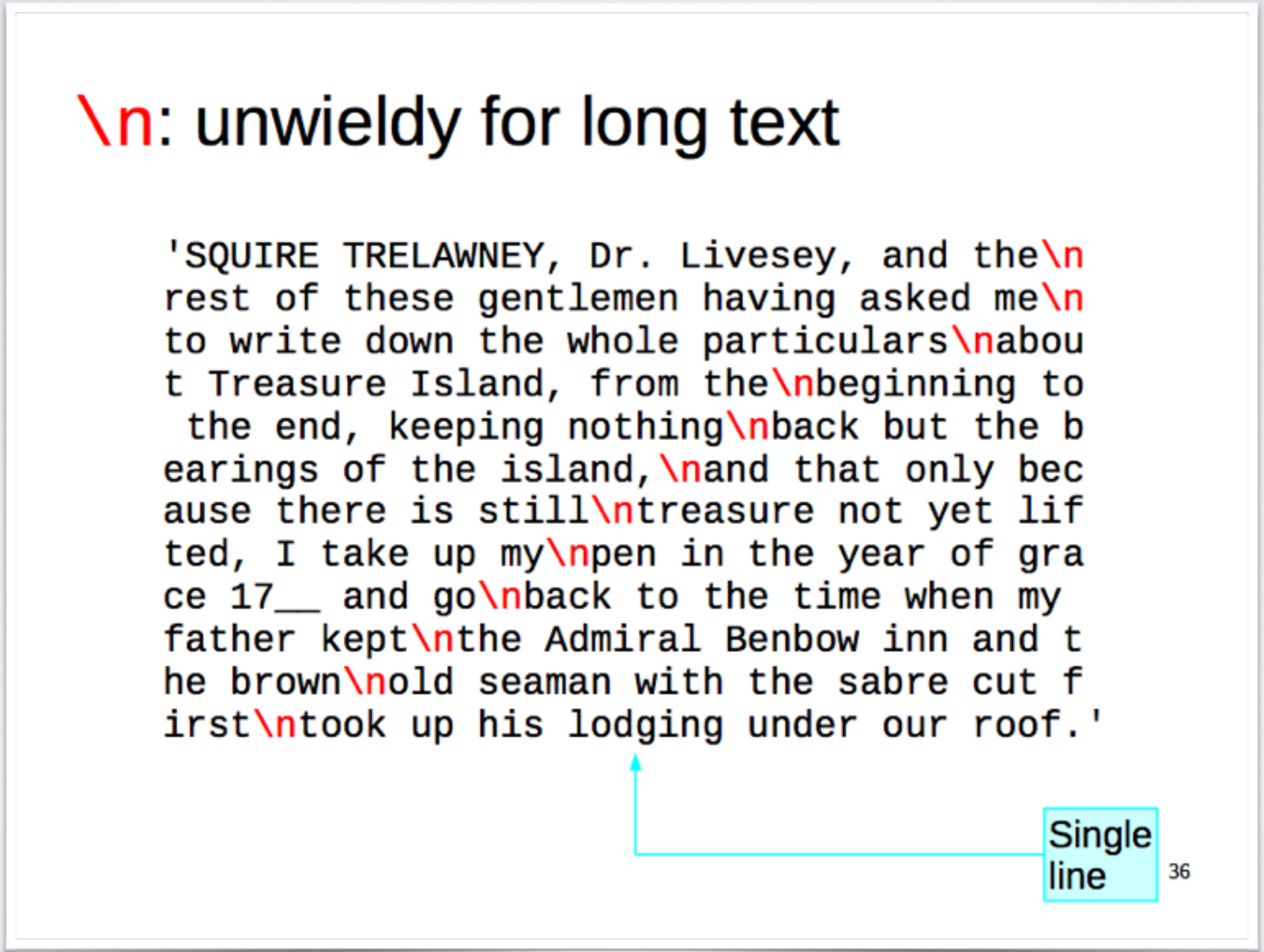
The backslash

Special → Ordinary		`\'` →	`'`
		`\"` →	`"`
Ordinary → Special		`\n` →	↵
		`\t` →	⇥

35

Also call escaping.
The same idea as HTML character entities.

Python Basics



\n: unwieldy for long text

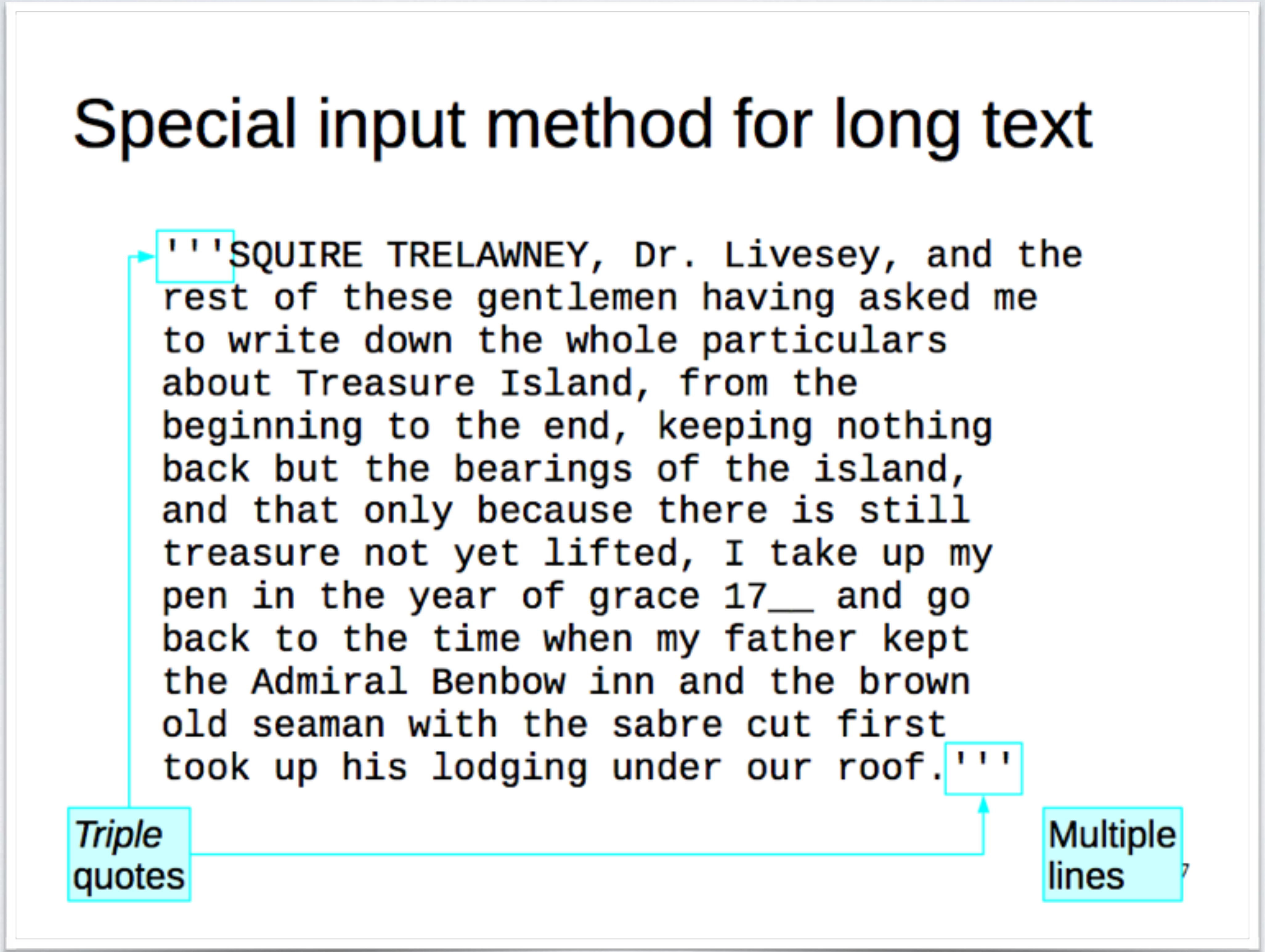
```
'SQUIRE TRELAWNEY, Dr. Livesey, and the\n
rest of these gentlemen having asked me\n
to write down the whole particulars\nabou
t Treasure Island, from the\nbeginning to
 the end, keeping nothing\nback but the b
earings of the island,\nand that only bec
ause there is still\ntreasure not yet lif
ted, I take up my\npen in the year of gra
ce 17__ and go\nback to the time when my
father kept\nthe Admiral Benbow inn and t
he brown\nold seaman with the sabre cut f
irst\ntook up his lodging under our roof.'
```

Single line

36

Slide taken from University of Cambridge's Python AB Course

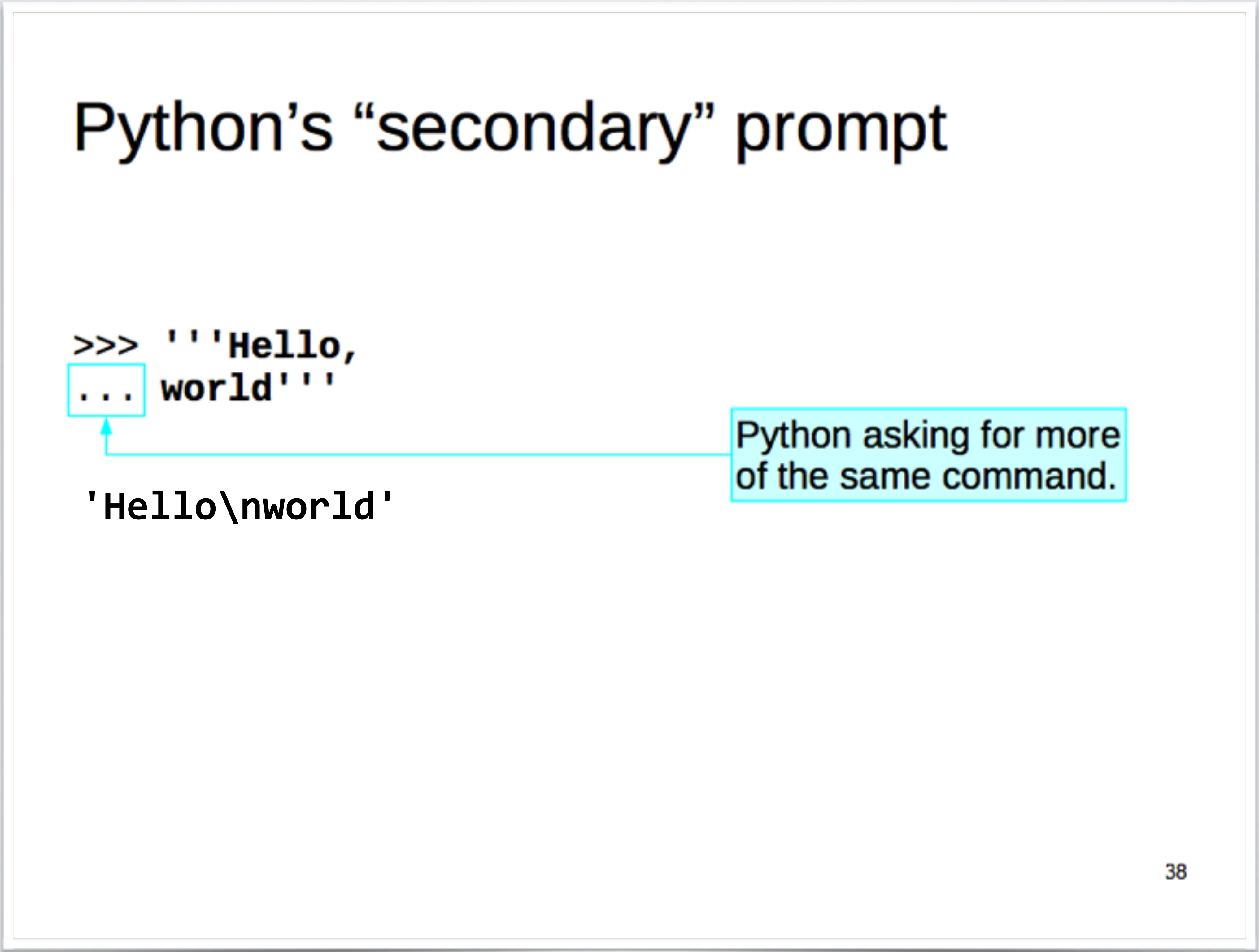
Python Basics



Special input method for long text

```
'''SQUIRE TRELAWNEY, Dr. Livesey, and the
rest of these gentlemen having asked me
to write down the whole particulars
about Treasure Island, from the
beginning to the end, keeping nothing
back but the bearings of the island,
and that only because there is still
treasure not yet lifted, I take up my
pen in the year of grace 17__ and go
back to the time when my father kept
the Admiral Benbow inn and the brown
old seaman with the sabre cut first
took up his lodging under our roof.'''
```

Python Basics



Python's "secondary" prompt

```
>>> '''Hello,
... world'''
```

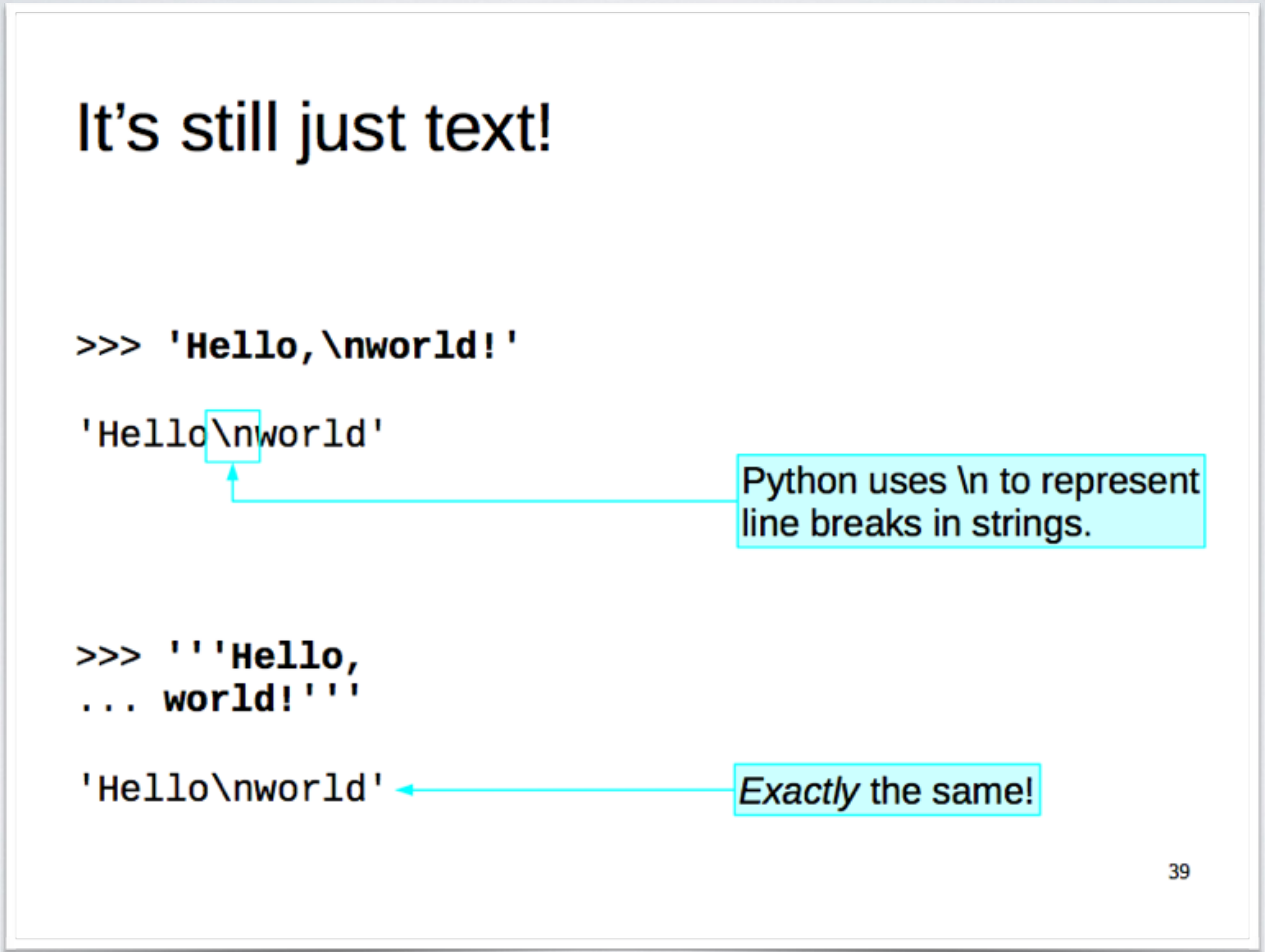
Python asking for more of the same command.

```
'Hello\nworld'
```

38

Slide taken from University of Cambridge's Python AB Course

Python Basics



It's still just text!

```
>>> 'Hello,\nworld!'

'Hello\nworld'
```

Python uses \n to represent line breaks in strings.

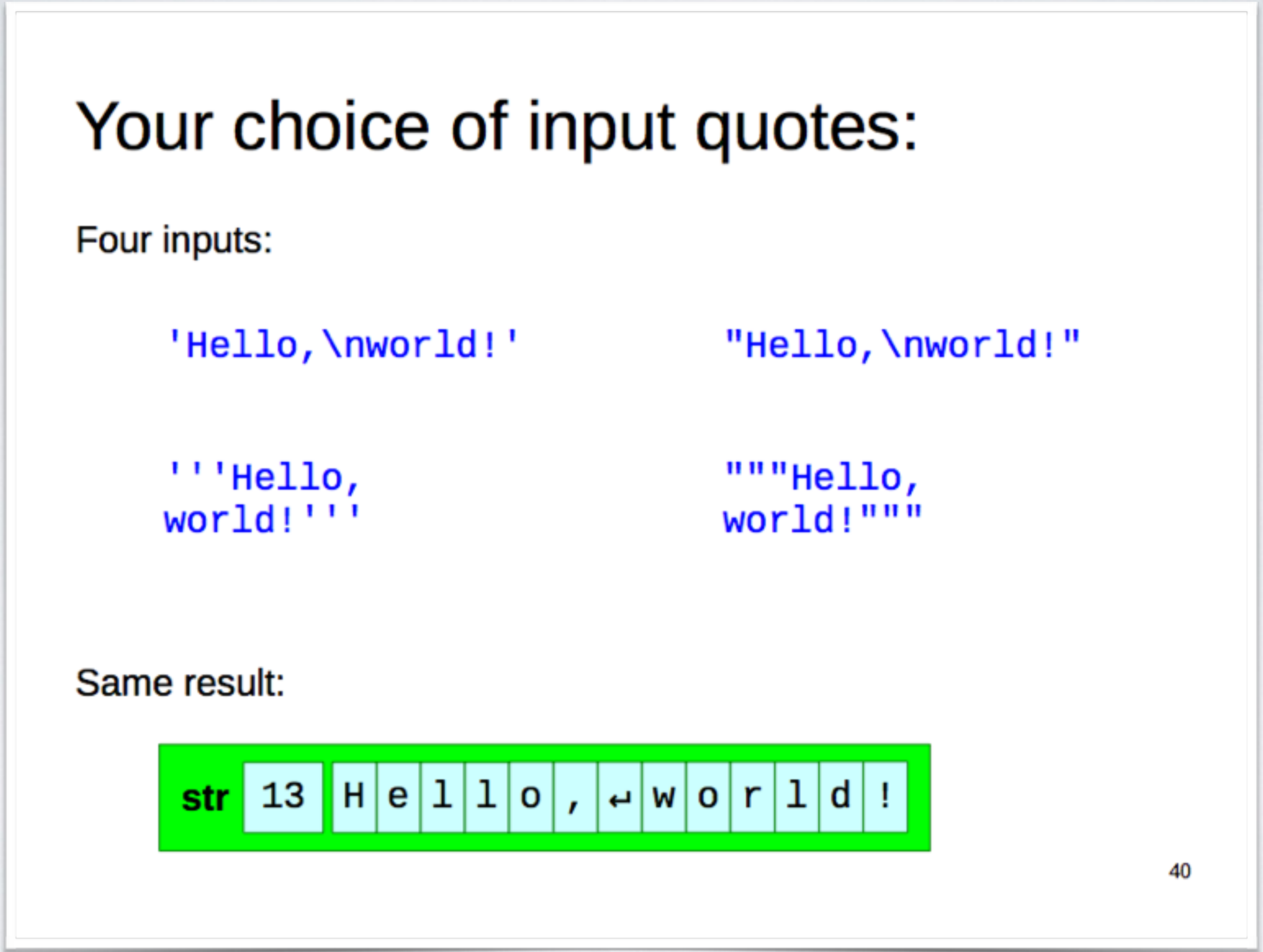
```
>>> '''Hello,
... world!'''

'Hello\nworld'
```

Exactly the same!

39

Slide taken from University of Cambridge's Python AB Course

Python Basics



Your choice of input quotes:

Four inputs:

```
'Hello,\nworld!'                "Hello,\nworld!"

'''Hello,                       """Hello,
world!'''                       world!"""
```

Same result:

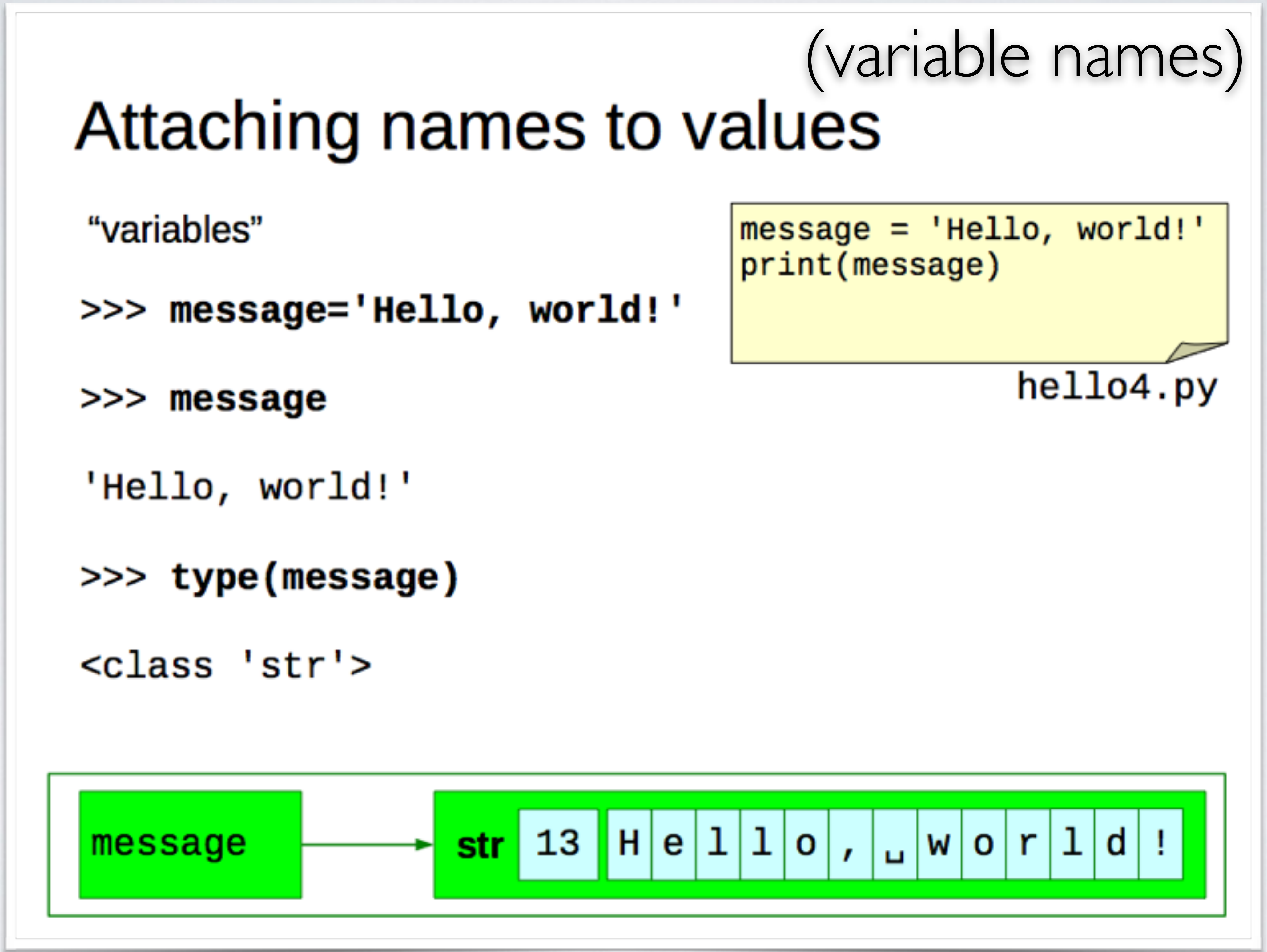
```
str | 13 | H | e | l | l | o | , | ↵ | w | o | r | l | d | !
```

40

Slide taken from University of Cambridge's Python AB Course

Python Basics

(variable names)



Attaching names to values

“variables”

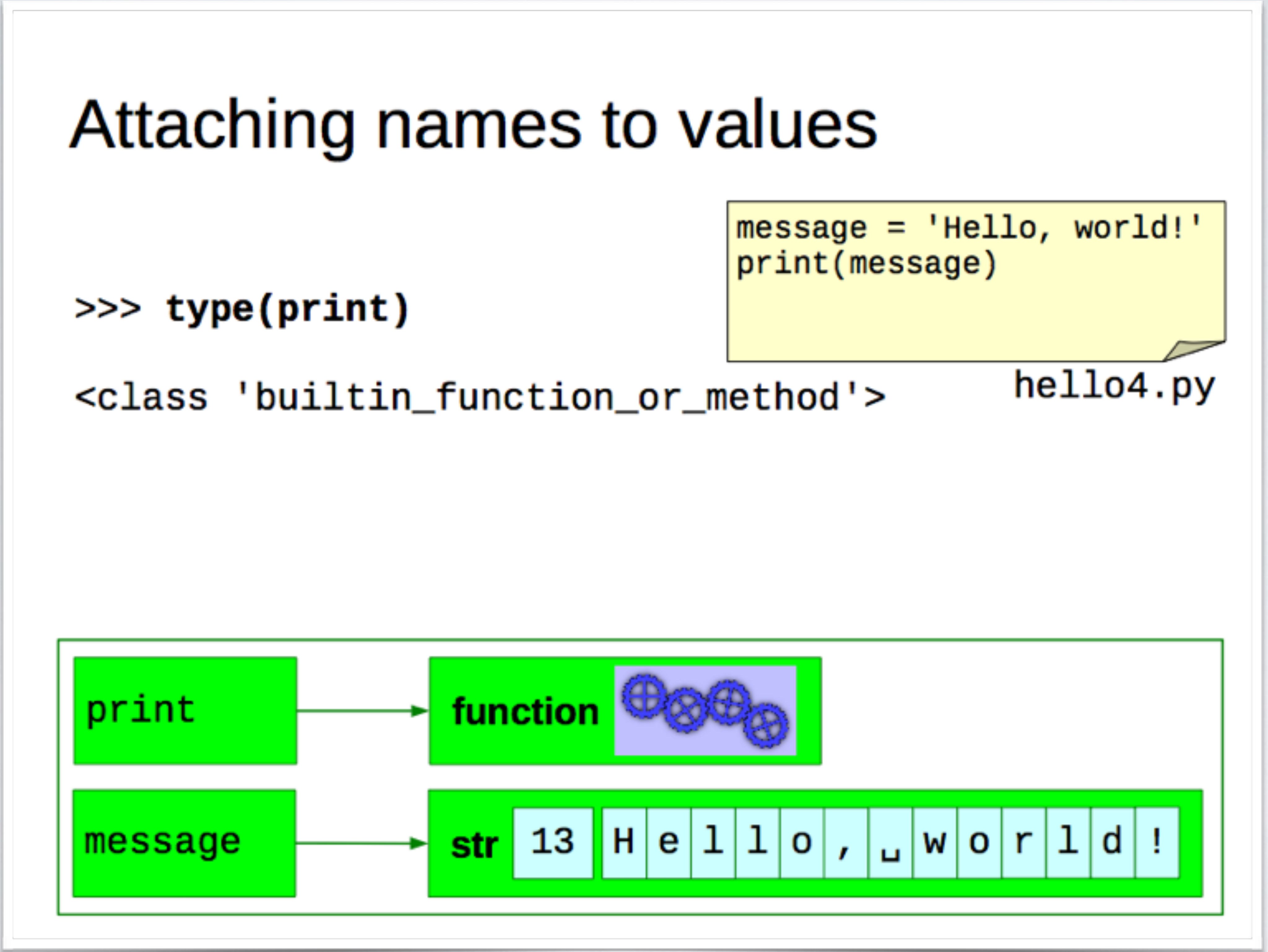
```
>>> message='Hello, world!'

>>> message

'Hello, world!'

>>> type(message)

<class 'str'>
```

```
message = 'Hello, world!'
print(message)
```

hello4.py

message
str
13
H e l l o , ␣ w o r l d !

Python Basics



Attaching names to values

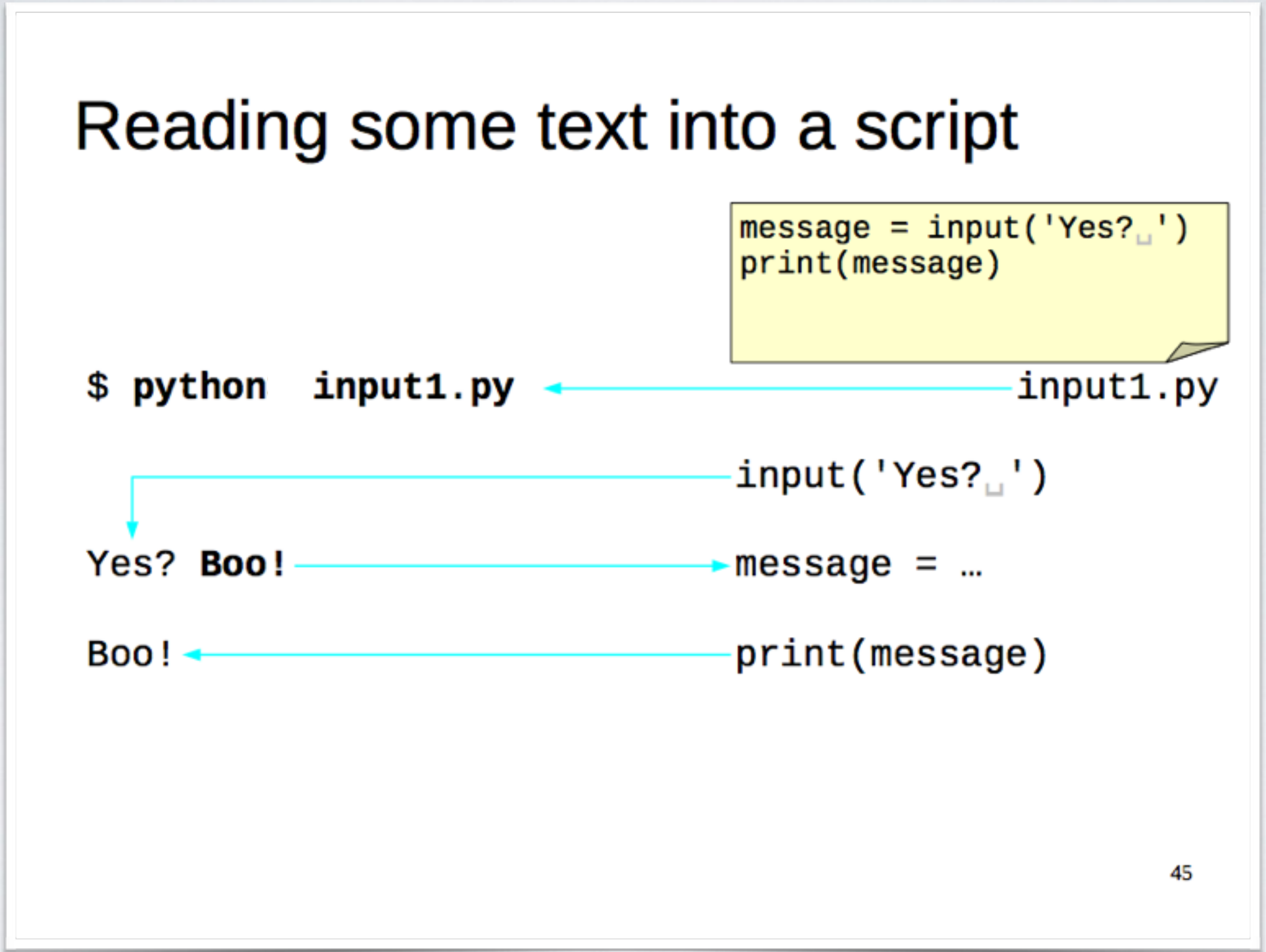
```
message = 'Hello, world!'
print(message)
```

hello4.py

```
>>> type(print)
<class 'builtin_function_or_method'>
```

Slide taken from University of Cambridge's Python AB Course

Python Basics

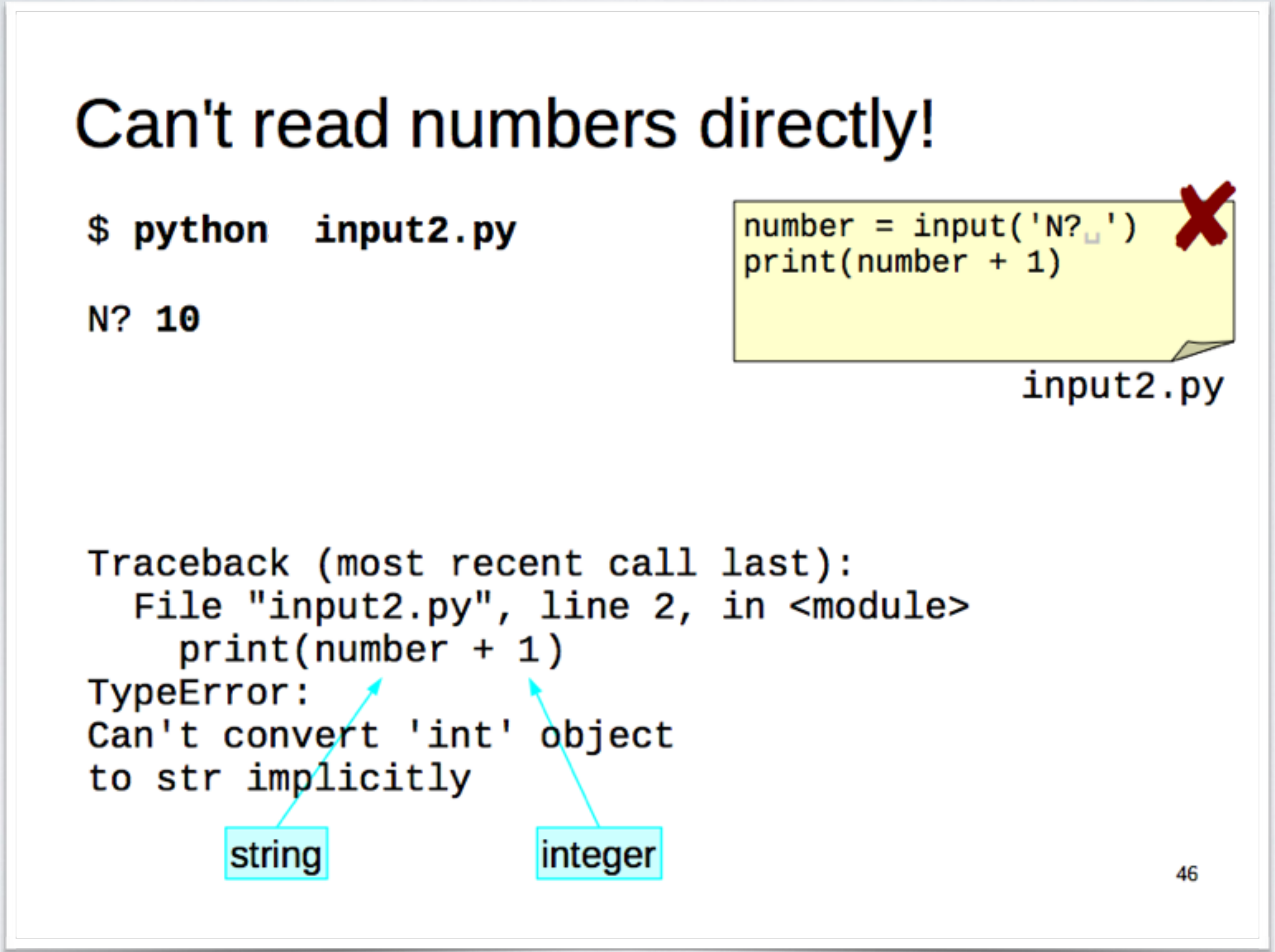


Reading some text into a script

```
message = input('Yes?␣')
print(message)
```

Slide taken from University of Cambridge's Python AB Course

Python Basics



Can't read numbers directly!

```
$ python  input2.py

N? 10
```

```
number = input('N?␣')
print(number + 1)
```

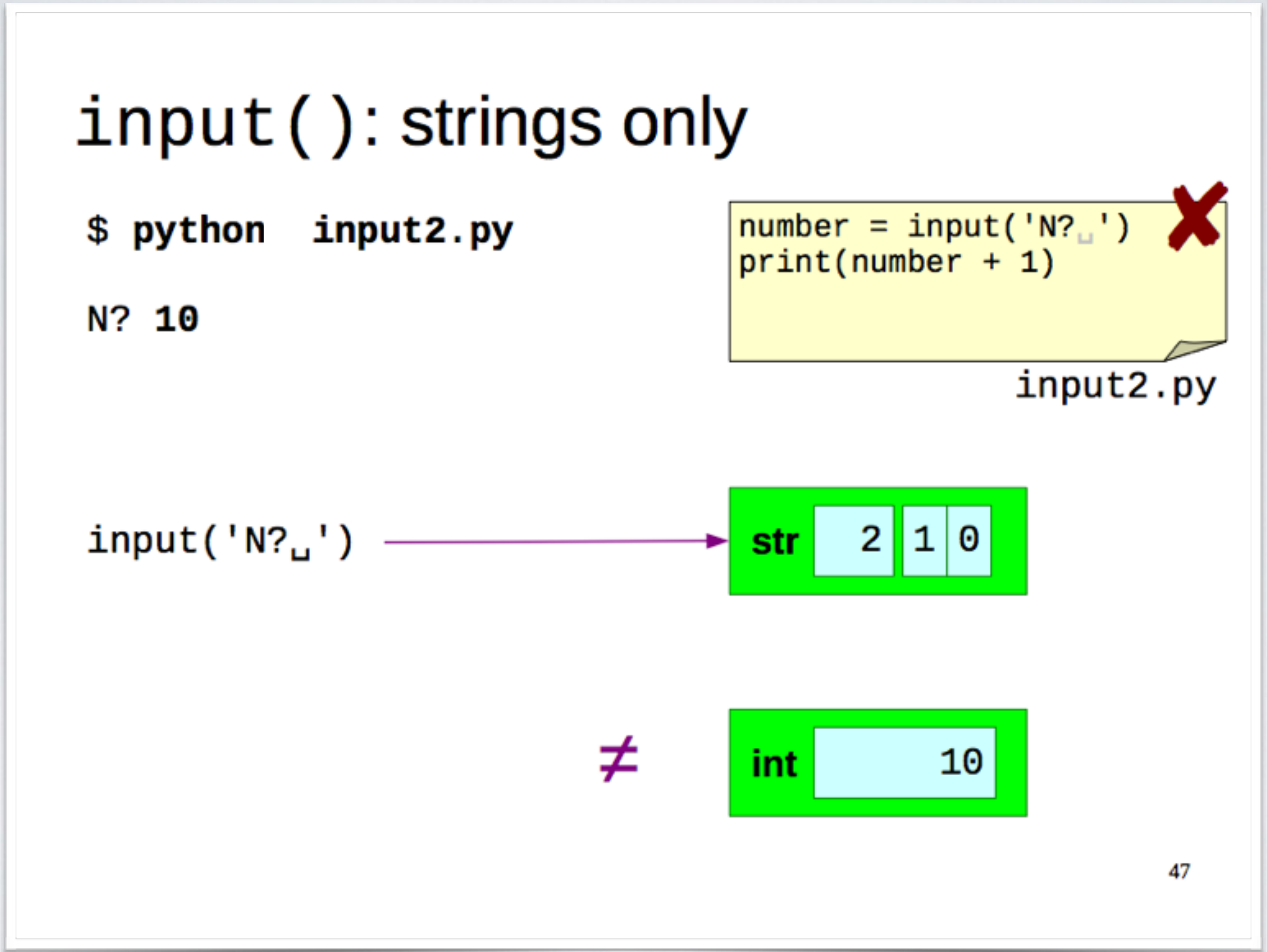
input2.py

```
Traceback (most recent call last):
  File "input2.py", line 2, in <module>
    print(number + 1)
TypeError:
Can't convert 'int' object
to str implicitly
```

string
integer
46

Slide taken from University of Cambridge's Python AB Course

Python Basics



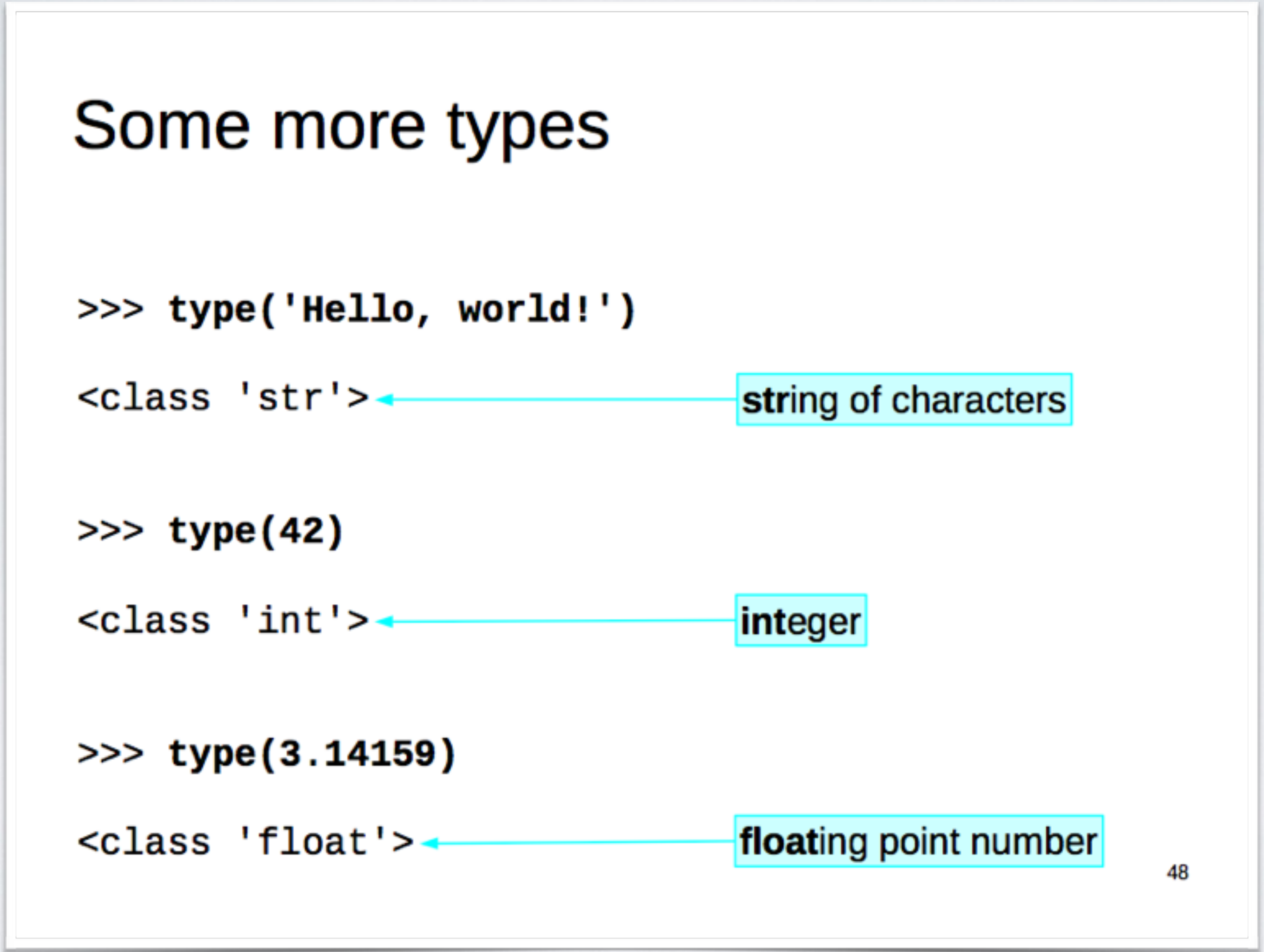
input(): strings only

```
$ python  input2.py

N? 10
```

```
number = input('N?␣')
print(number + 1)
```

Slide taken from University of Cambridge's Python AB Course

Python Basics



Some more types

```
>>> type('Hello, world!')
<class 'str'>
```

string of characters

```
>>> type(42)
<class 'int'>
```

integer

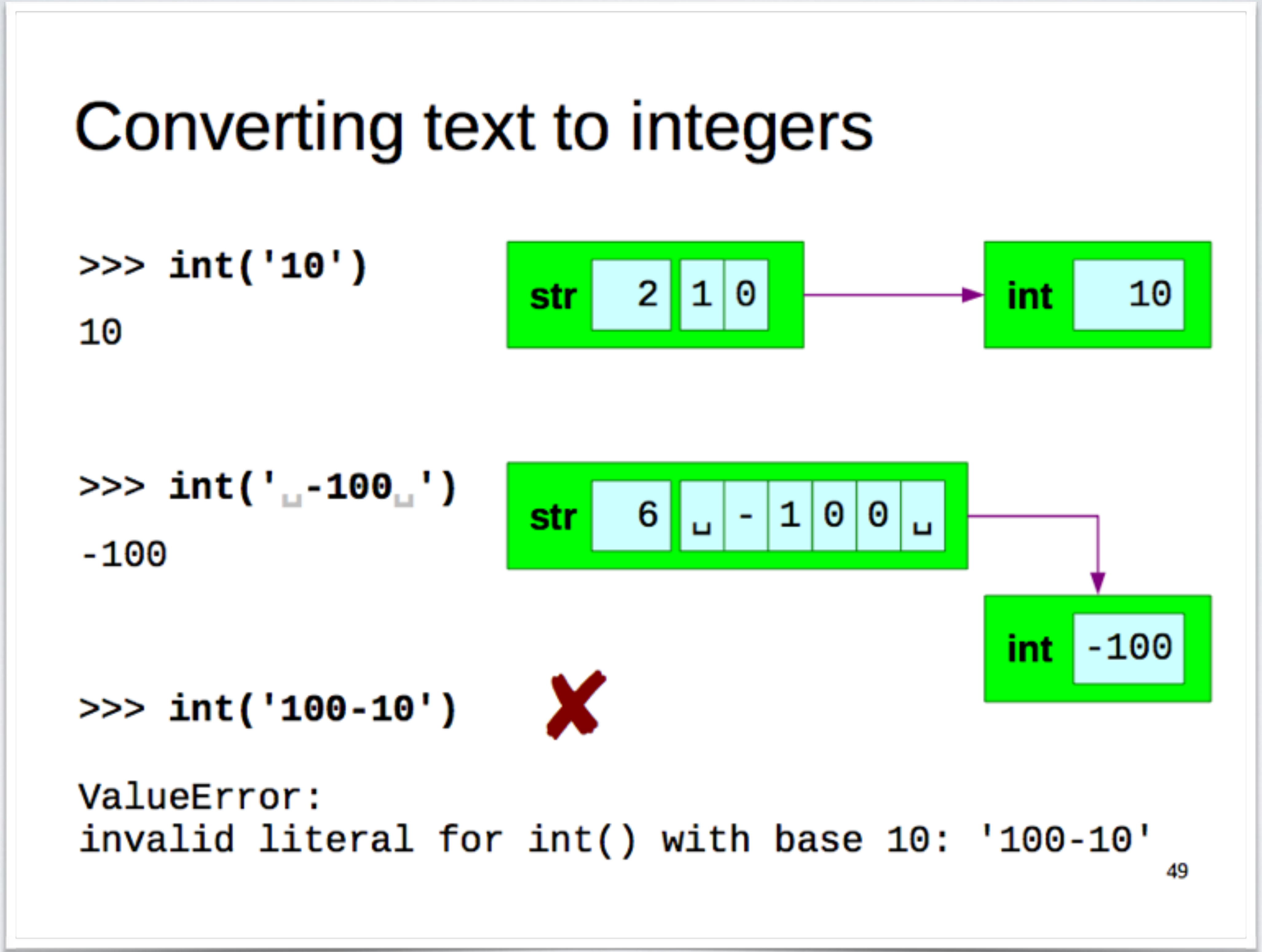
```
>>> type(3.14159)
<class 'float'>
```

floating point number

48

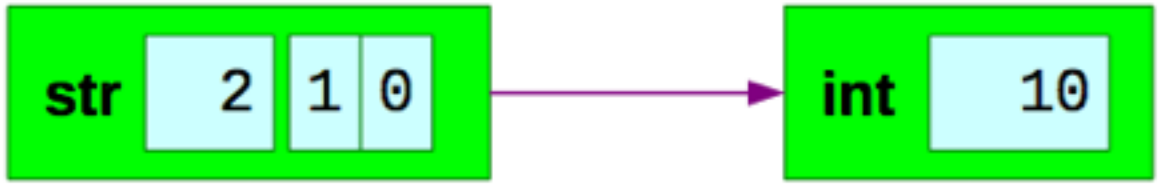
Slide taken from University of Cambridge's Python AB Course

Python Basics

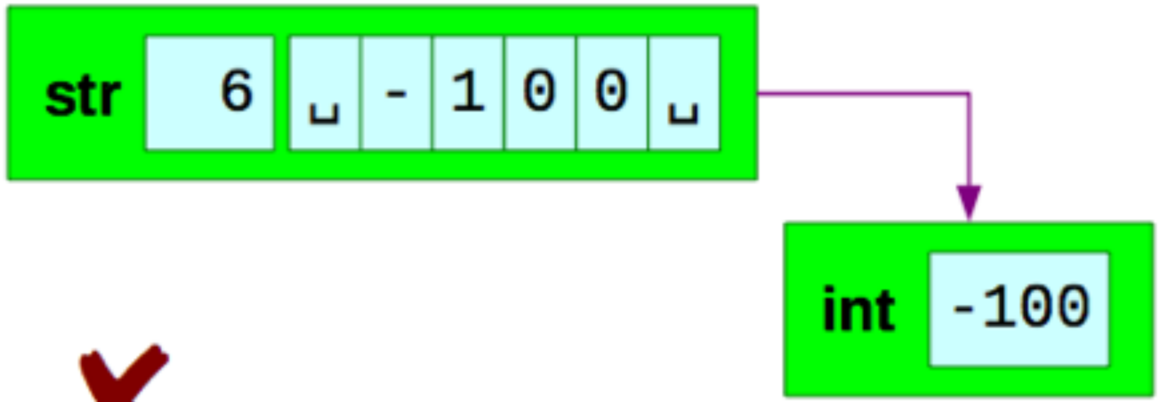


Converting text to integers

```
>>> int('10')
10
```



```
>>> int('␣-100␣')
-100
```



```
>>> int('100-10')
```

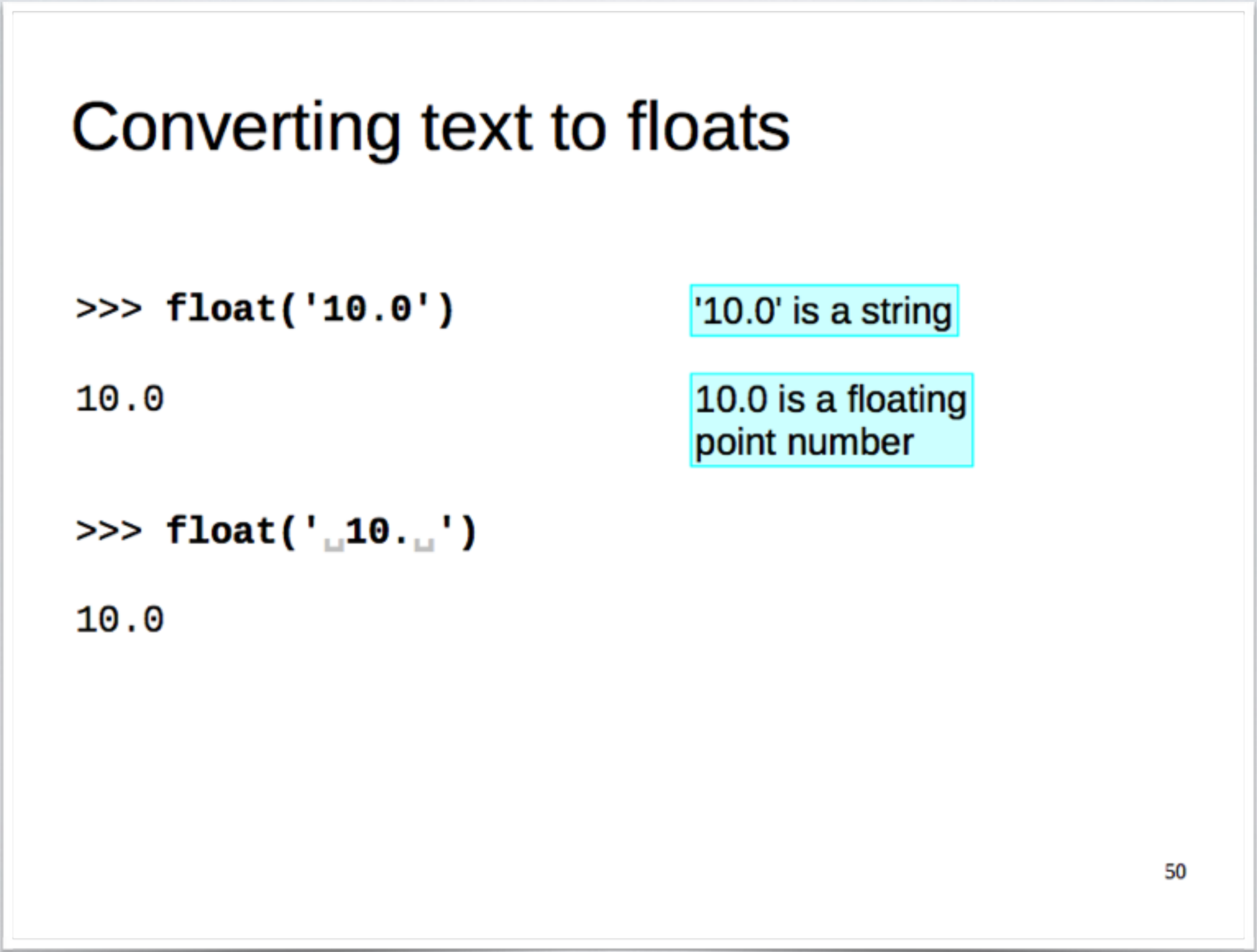
✘

```
ValueError:
invalid literal for int() with base 10: '100-10'
```

49

Slide taken from University of Cambridge's Python AB Course

Python Basics



Converting text to floats

```
>>> float('10.0')

10.0


>>> float('␣10.␣')

10.0
```

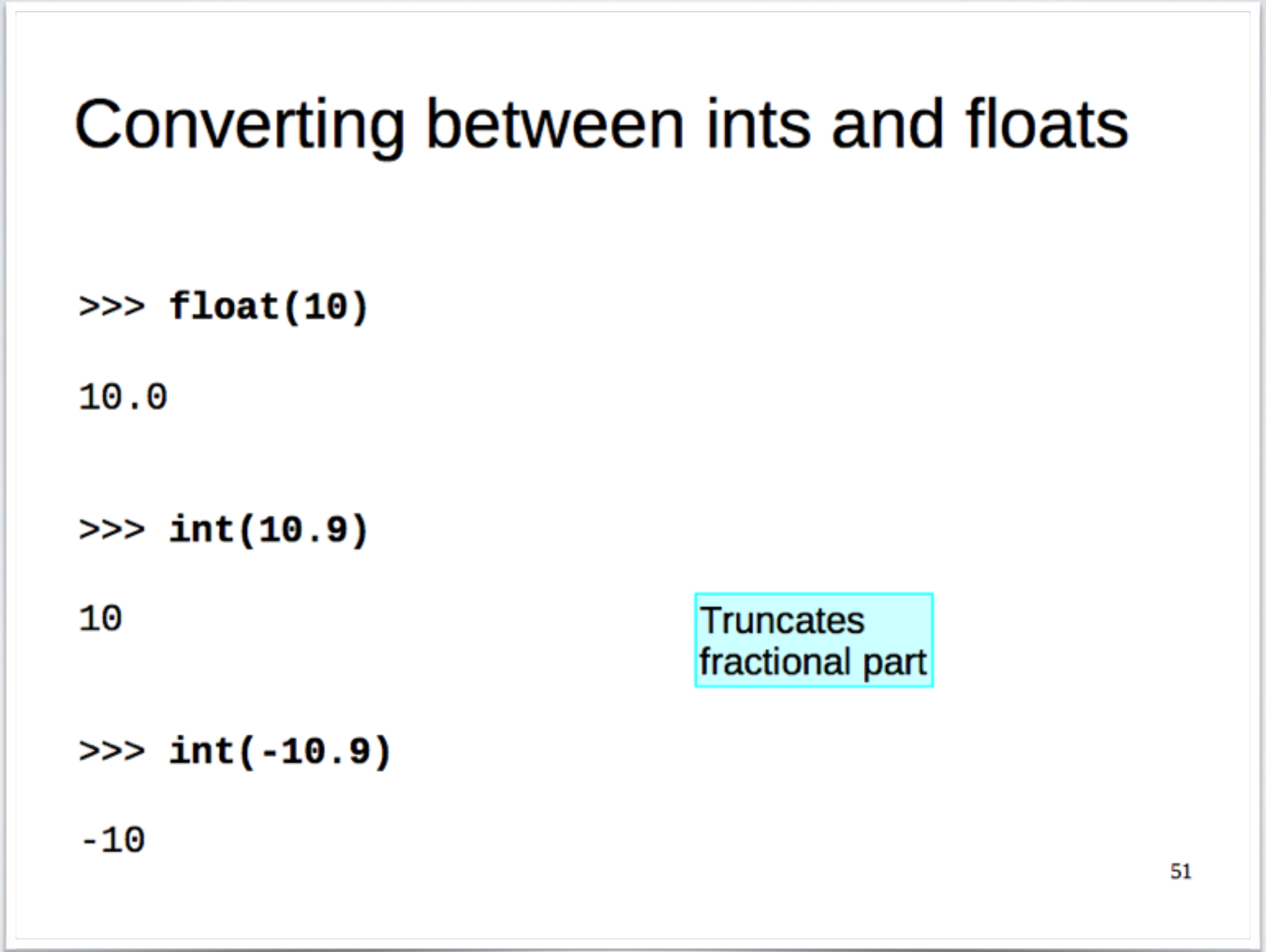
'10.0' is a string

10.0 is a floating point number

50

Slide taken from University of Cambridge's Python AB Course

Python Basics



Converting between ints and floats

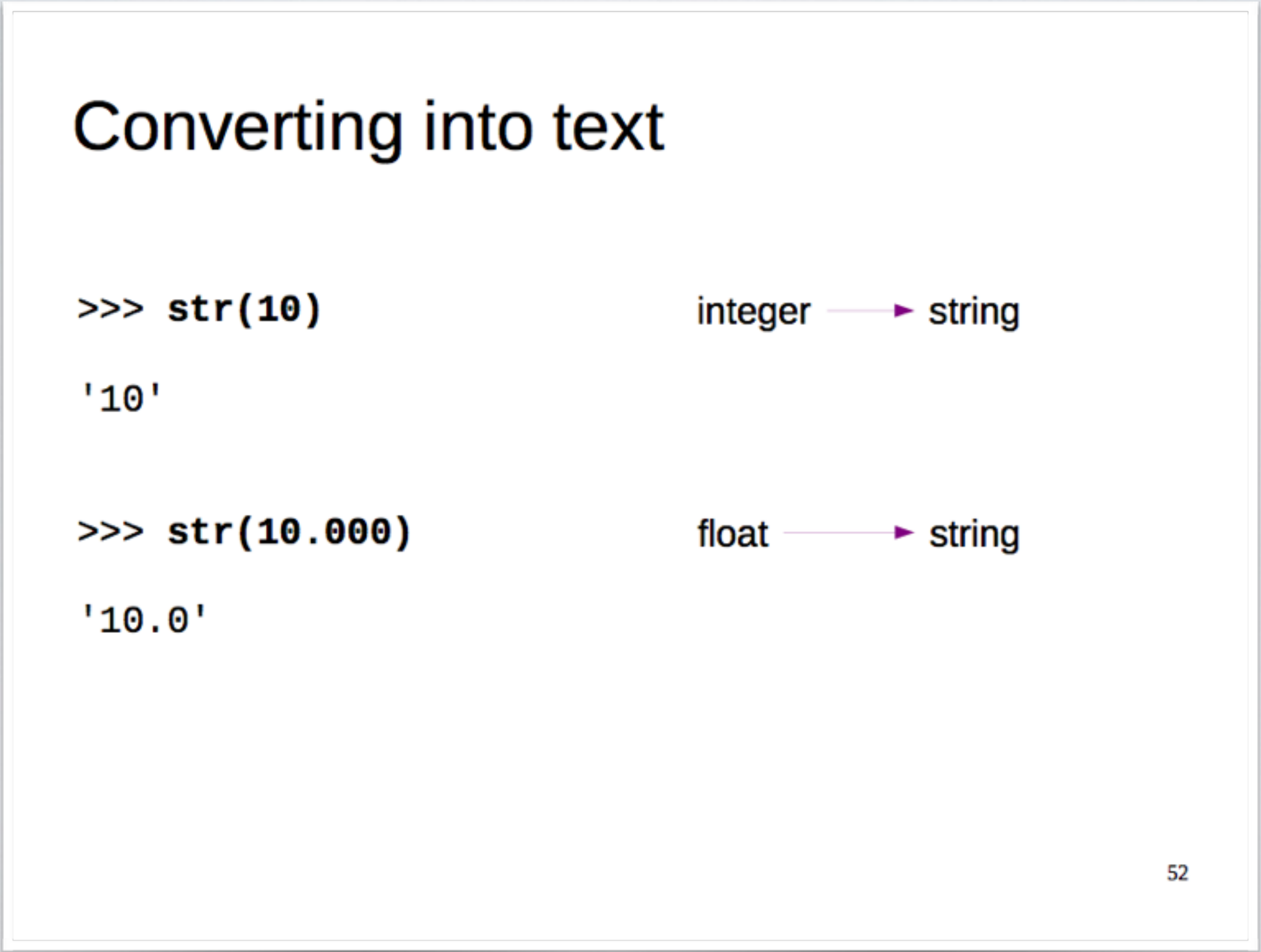
```
>>> float(10)
10.0

>>> int(10.9)
10

>>> int(-10.9)
-10
```

Truncates fractional part

51

Slide taken from University of Cambridge's Python AB Course

Python Basics



Converting into text

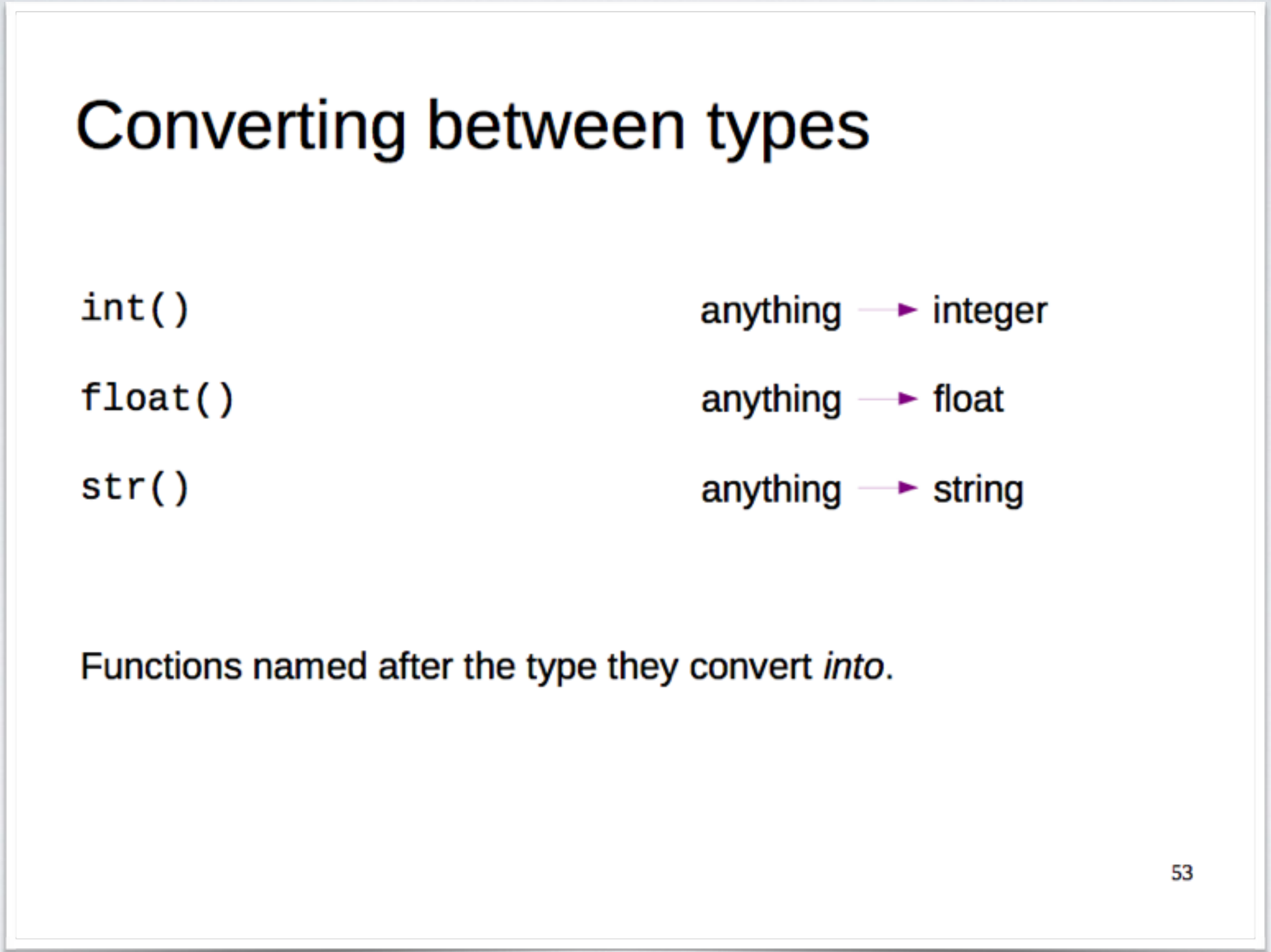
```
>>> str(10)
'10'
```

$\text{integer} \longrightarrow \text{string}$

```
>>> str(10.000)
'10.0'
```

$\text{float} \longrightarrow \text{string}$

Python Basics



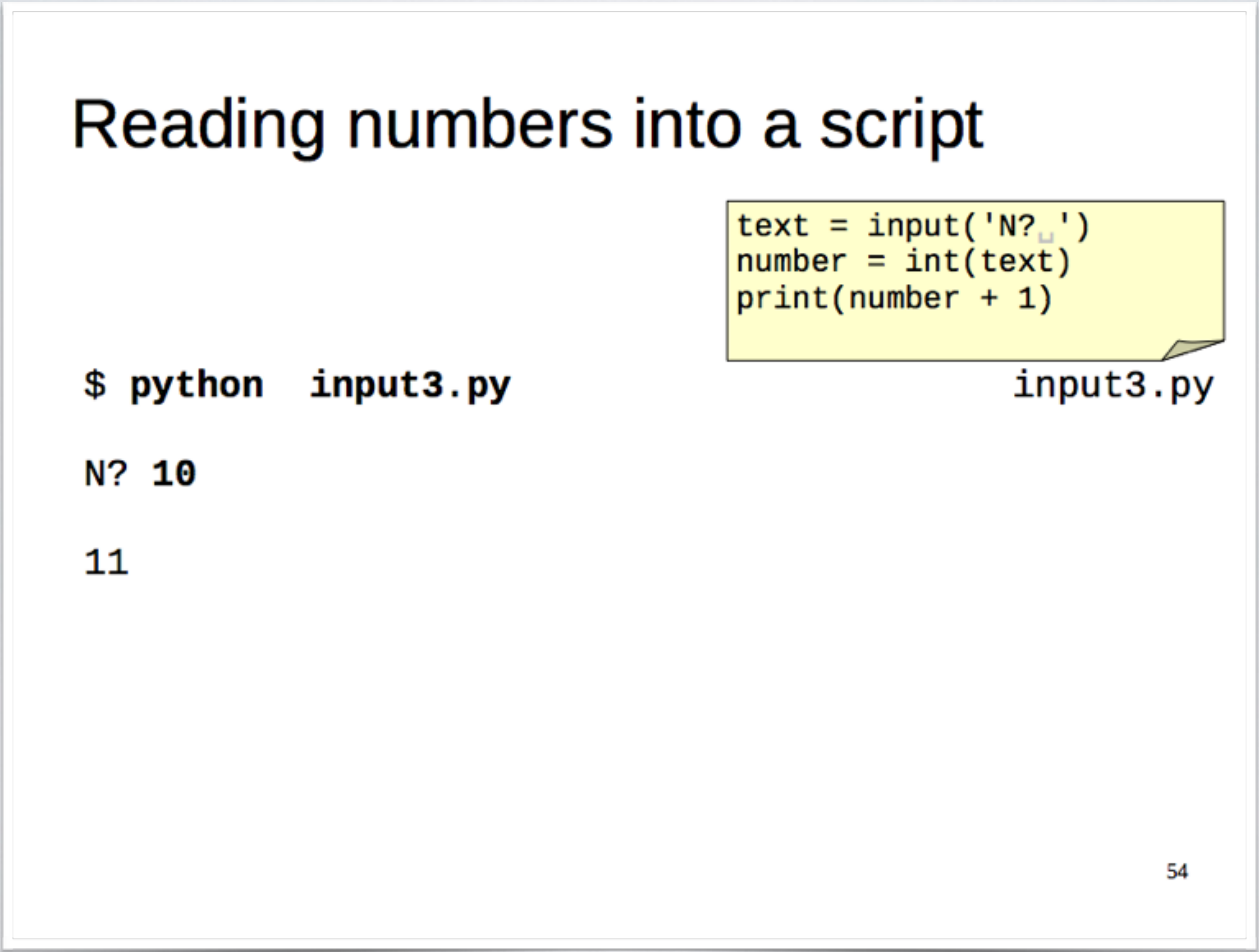
Converting between types

`int()`	anything $\longrightarrow$ integer
`float()`	anything $\longrightarrow$ float
`str()`	anything $\longrightarrow$ string

Functions named after the type they convert *into*.

53

Python Basics



Reading numbers into a script

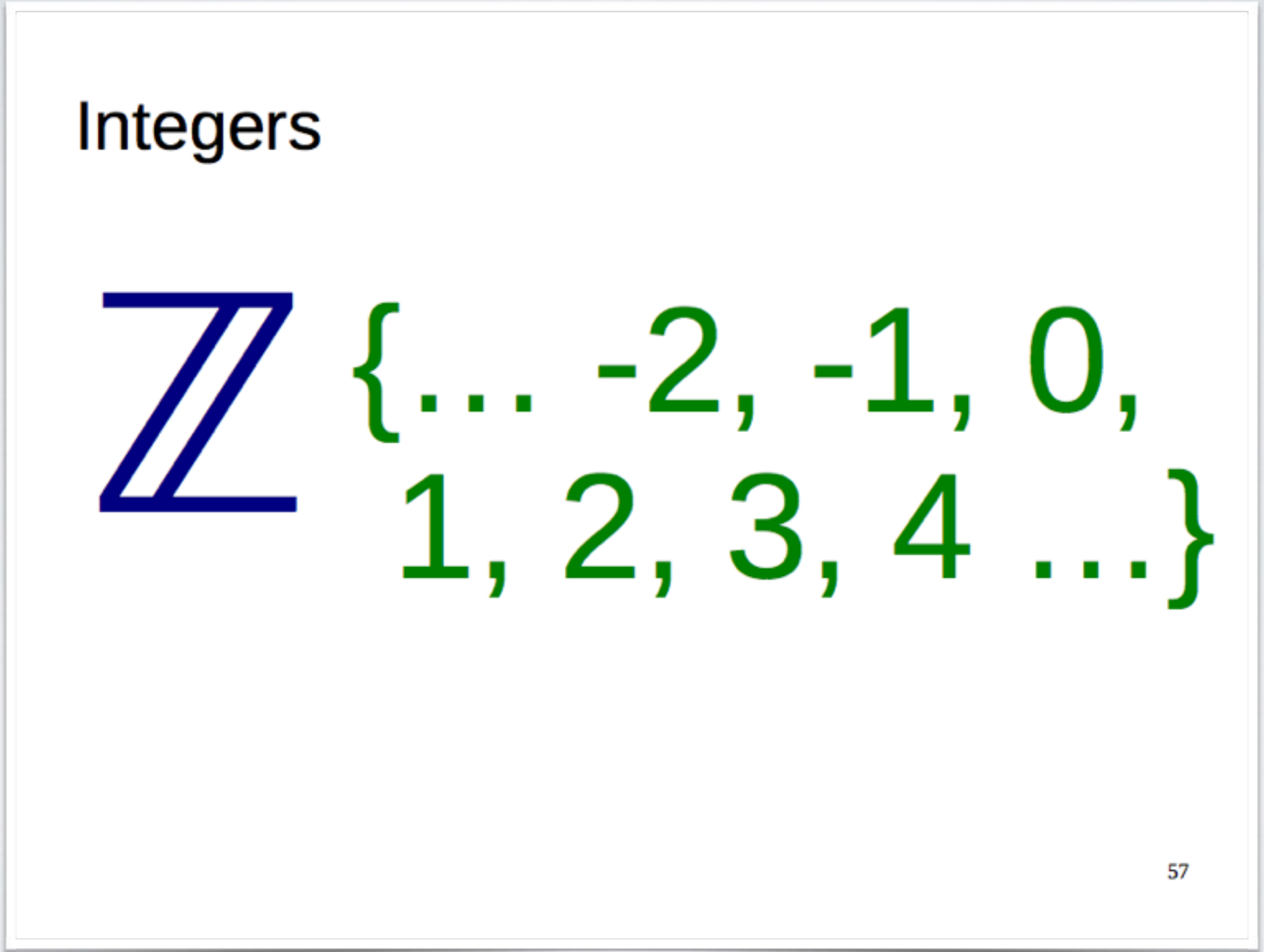
```
text = input('N?␣')
number = int(text)
print(number + 1)
```

input3.py

```
$ python  input3.py

N? 10

11
```

54

Python Basics



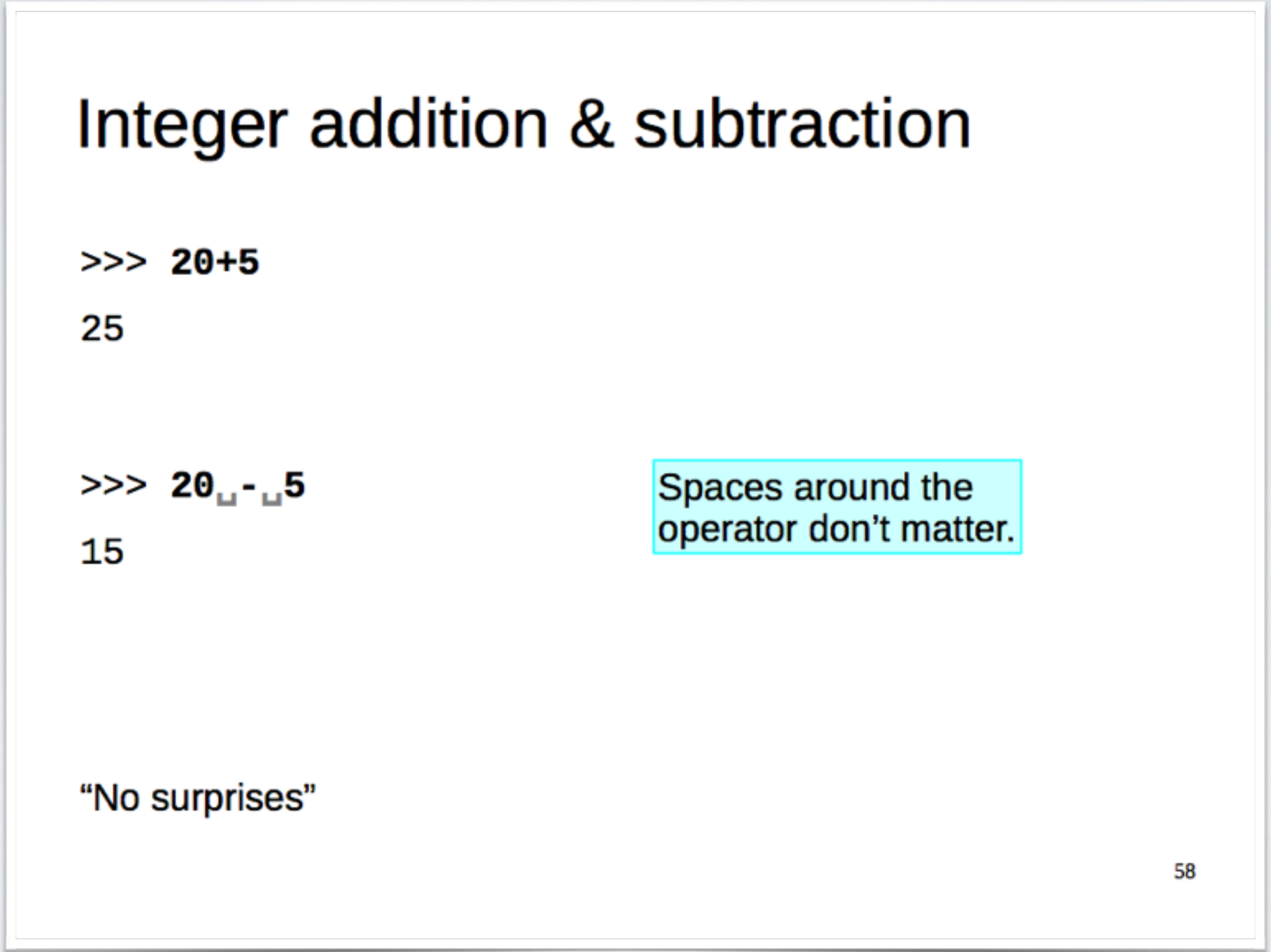
Integers

$\mathbb{Z}$ {… -2, -1, 0, 1, 2, 3, 4 …}

57

Slide taken from University of Cambridge's Python AB Course

Python Basics



Integer addition & subtraction

```
>>> 20+5
25

>>> 20␣-␣5
15
```

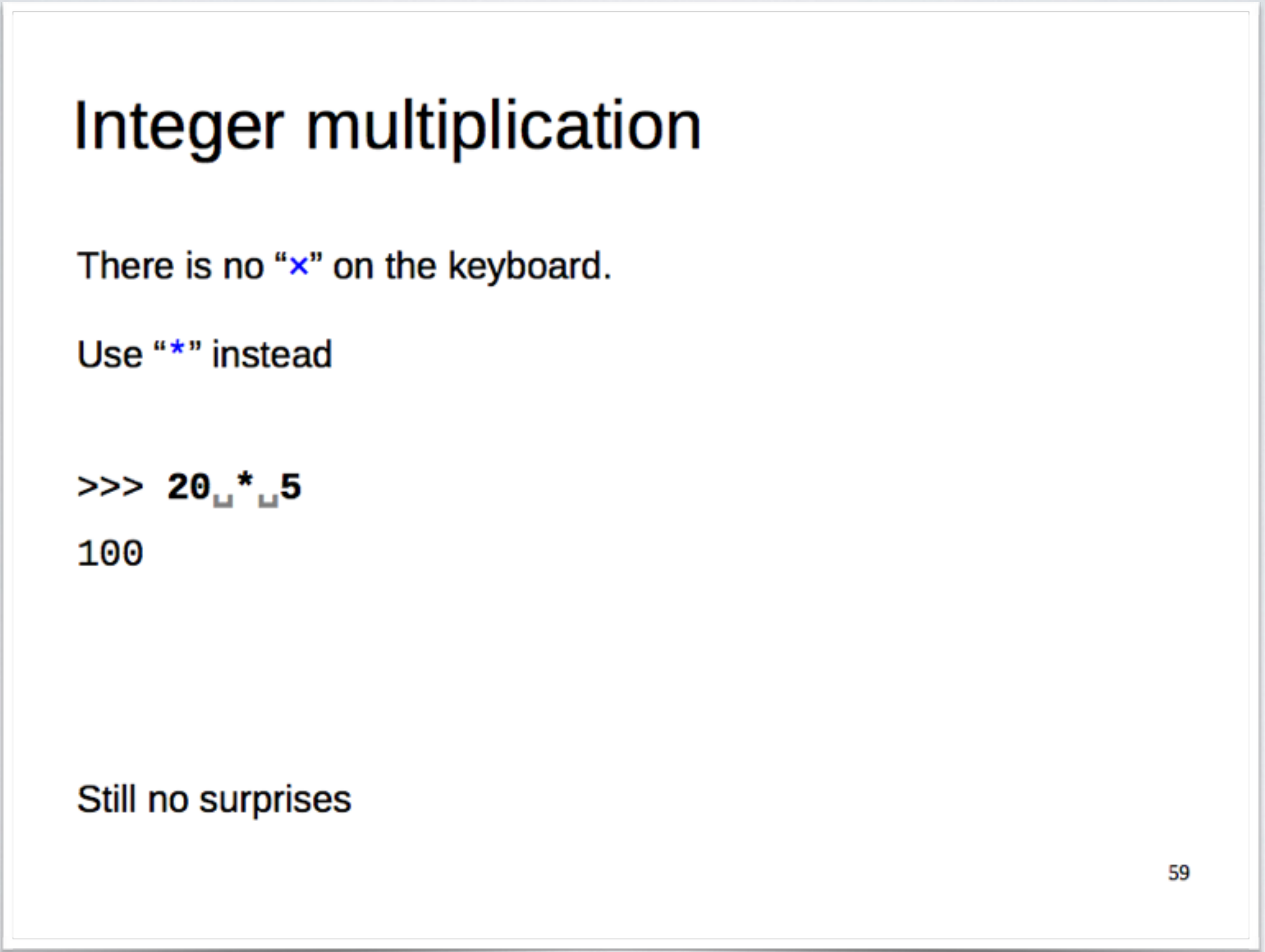
Spaces around the operator don’t matter.

“No surprises”

58

Slide taken from University of Cambridge’s Python AB Course

Python Basics



Integer multiplication

There is no “×” on the keyboard.

Use “*” instead

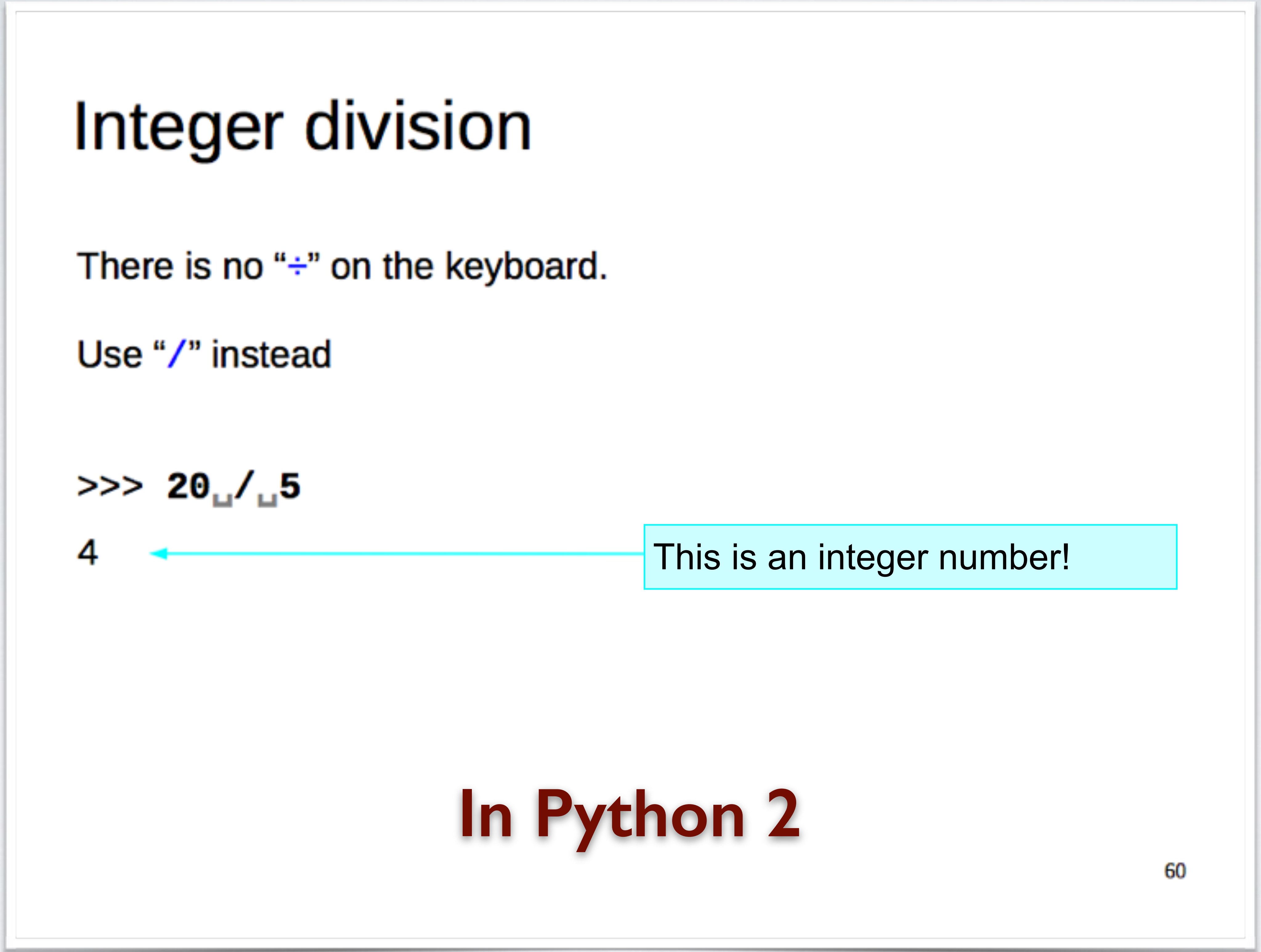
```
>>> 20␣*␣5
100
```

Still no surprises

59

Slide taken from University of Cambridge’s Python AB Course

Python Basics



Integer division

There is no “÷” on the keyboard.

Use “/” instead

```
>>> 20␣/␣5
4
```

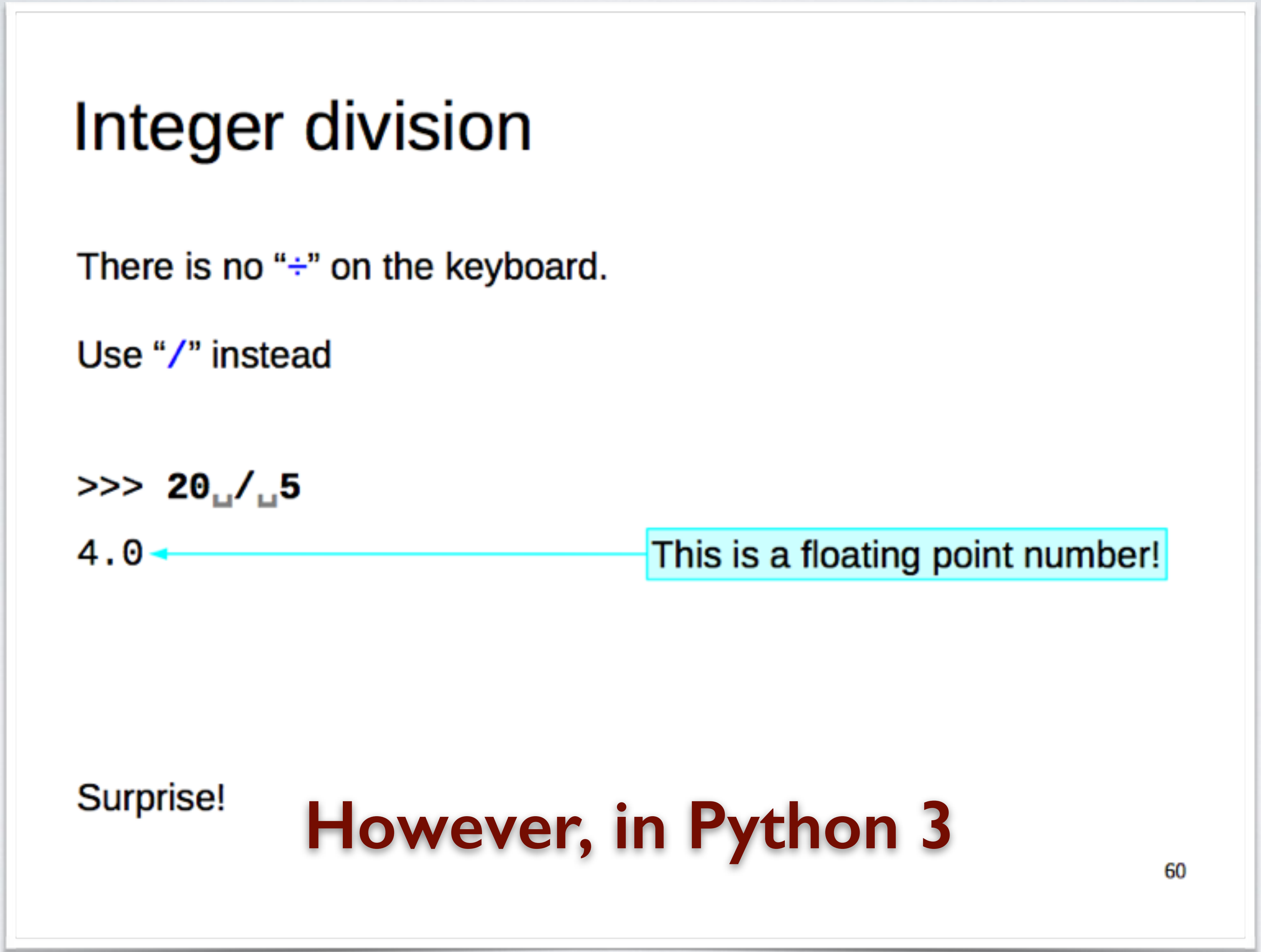
This is an integer number!

In Python 2

60

Slide taken from University of Cambridge’s Python AB Course

Python Basics



Integer division

There is no "÷" on the keyboard.

Use "/" instead

```
>>> 20␣/␣5
4.0
```

This is a floating point number!

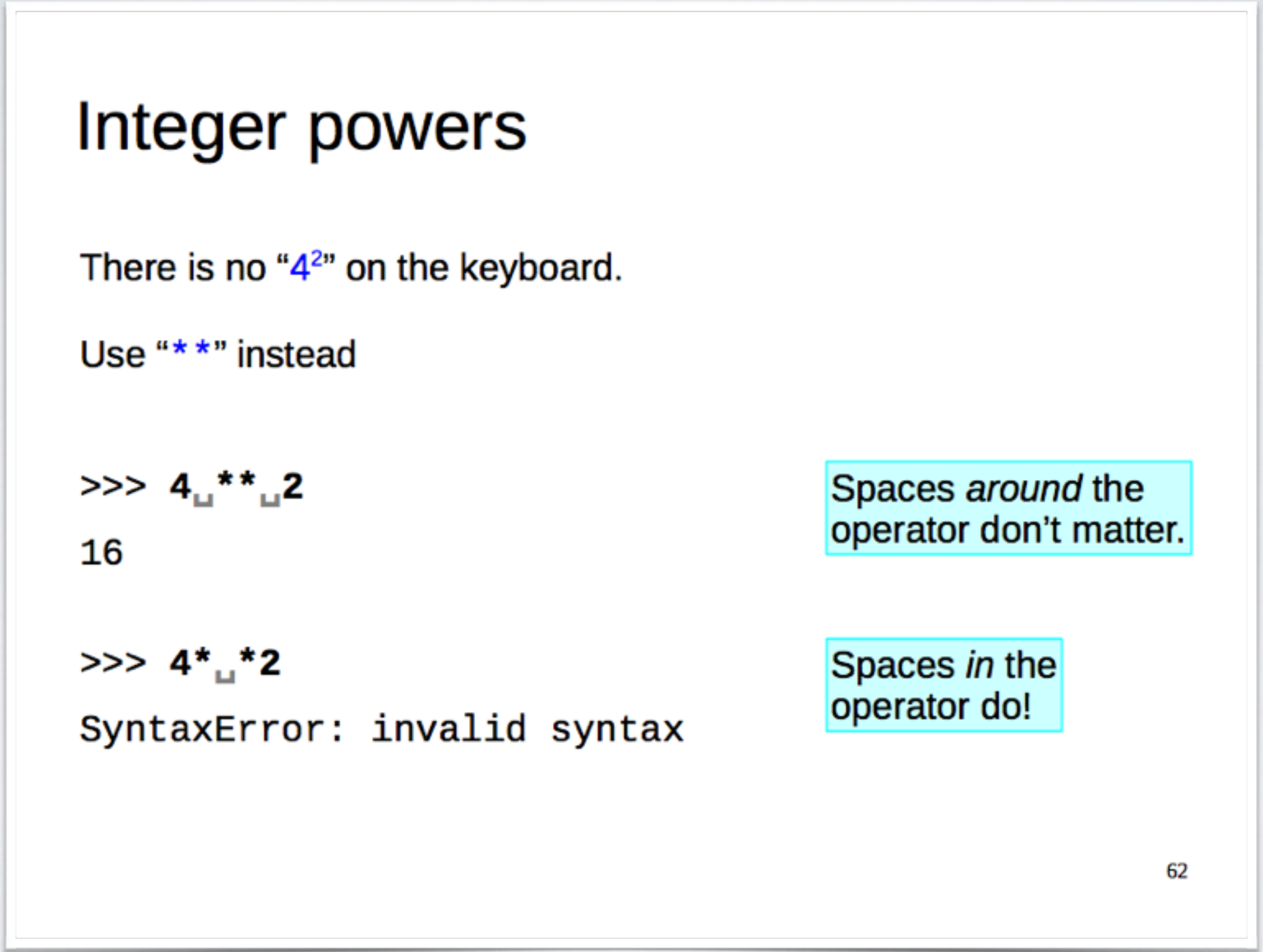
Surprise!

However, in Python 3

60

Slide taken from University of Cambridge's Python AB Course

Python Basics



Integer powers

There is no “4^2” on the keyboard.

Use “**” instead

```
>>> 4␣**␣2
16
```

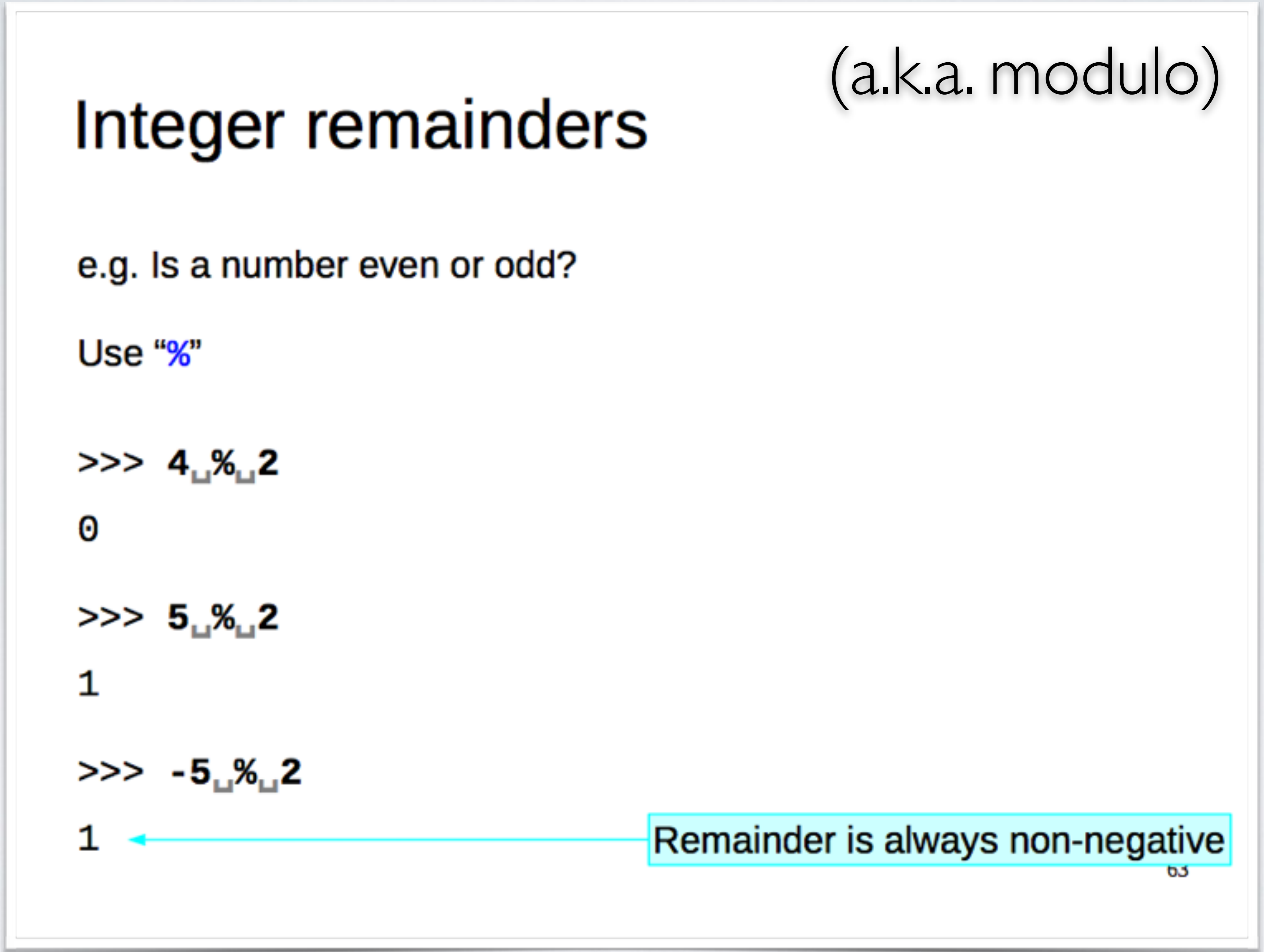
Spaces *around* the operator don’t matter.

```
>>> 4*␣*2
SyntaxError: invalid syntax
```

Spaces *in* the operator do!

62

Python Basics



(a.k.a. modulo)

Integer remainders

e.g. Is a number even or odd?

Use "%"

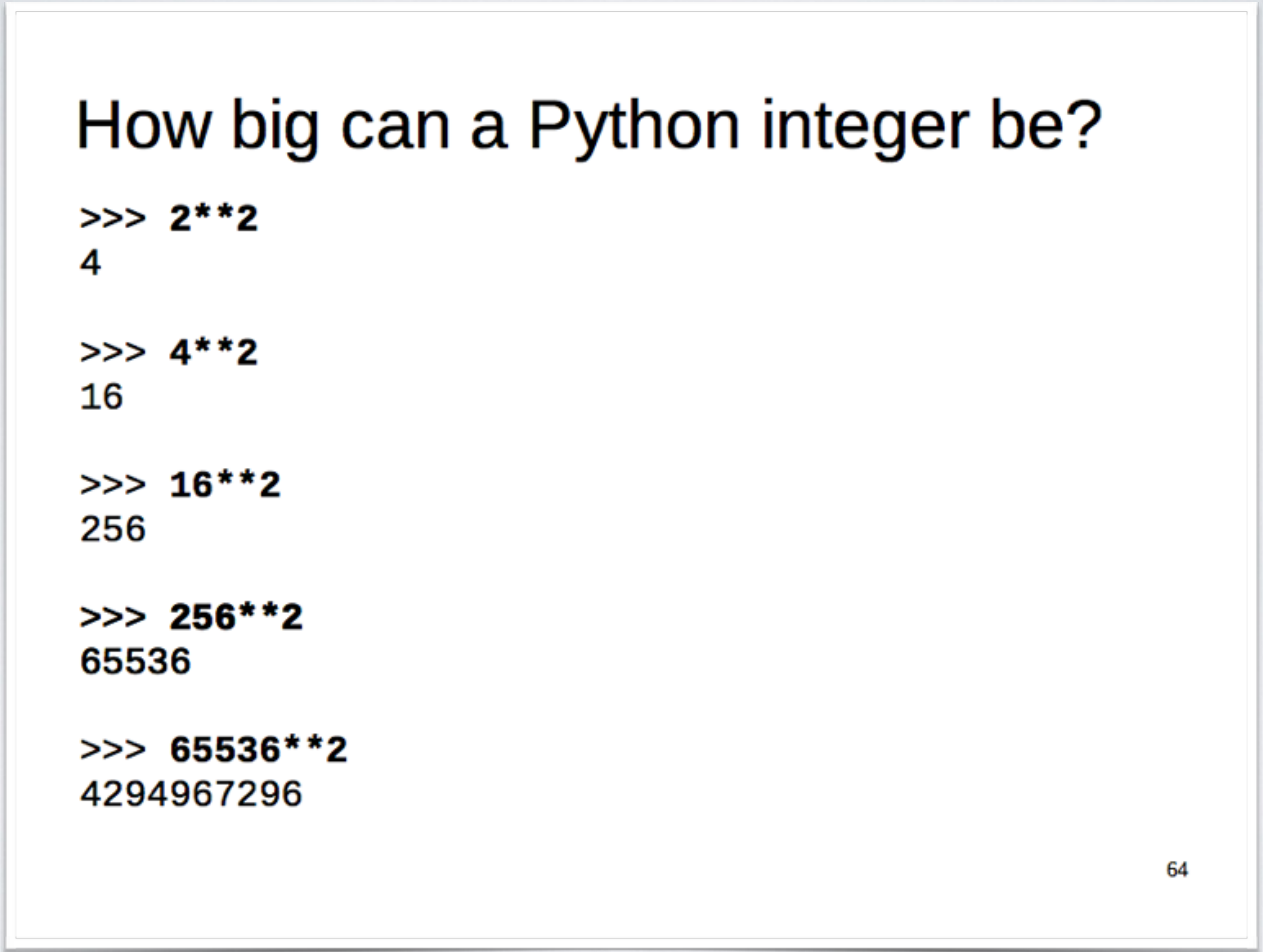
```
>>> 4␣%␣2
0
>>> 5␣%␣2
1
>>> -5␣%␣2
1
```

Remainder is always non-negative

63

Slide taken from University of Cambridge's Python AB Course

Python Basics



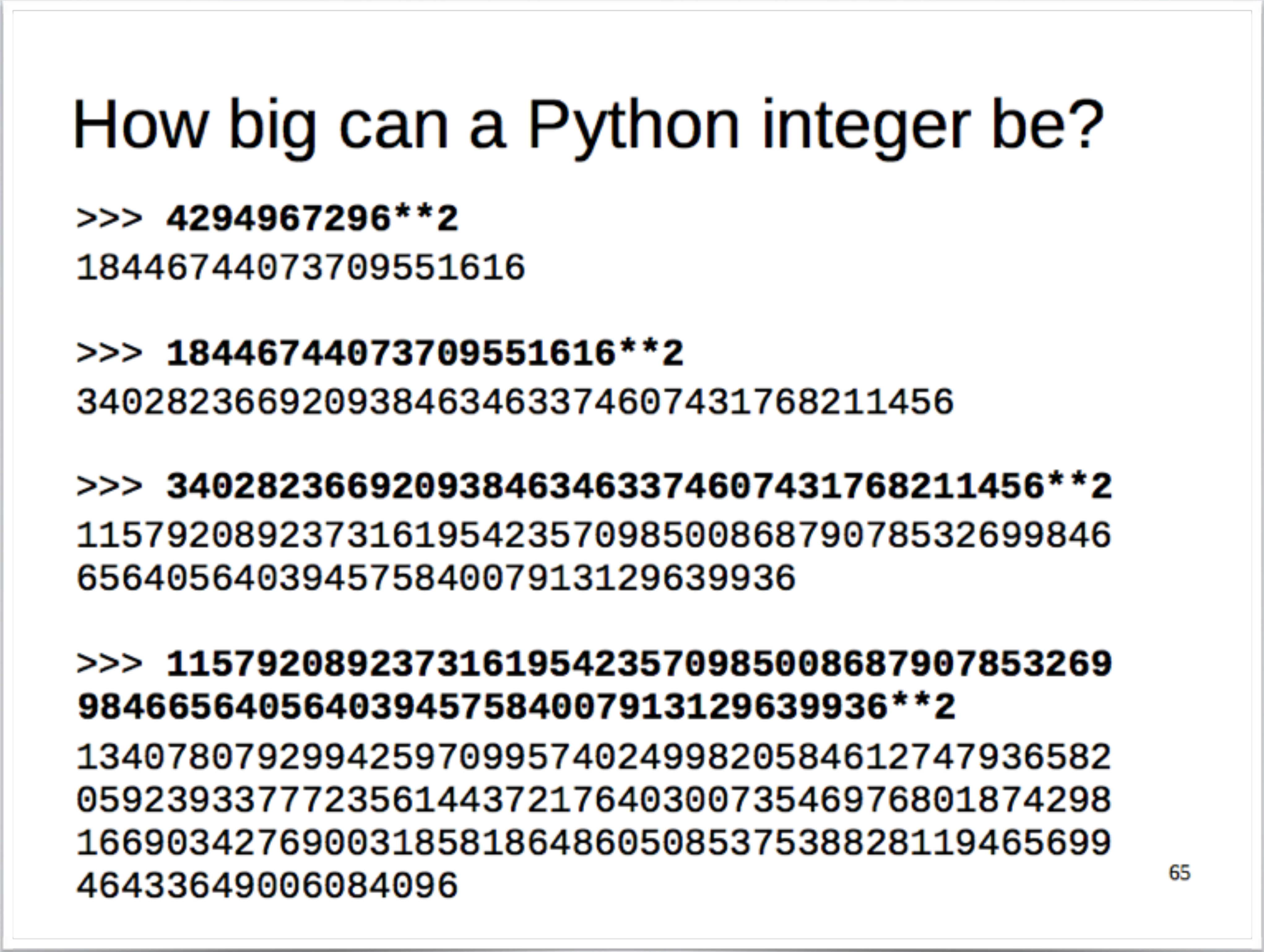
How big can a Python integer be?

```
>>> 2**2
4

>>> 4**2
16

>>> 16**2
256

>>> 256**2
65536

>>> 65536**2
4294967296
```

64

Slide taken from University of Cambridge's Python AB Course

Python Basics



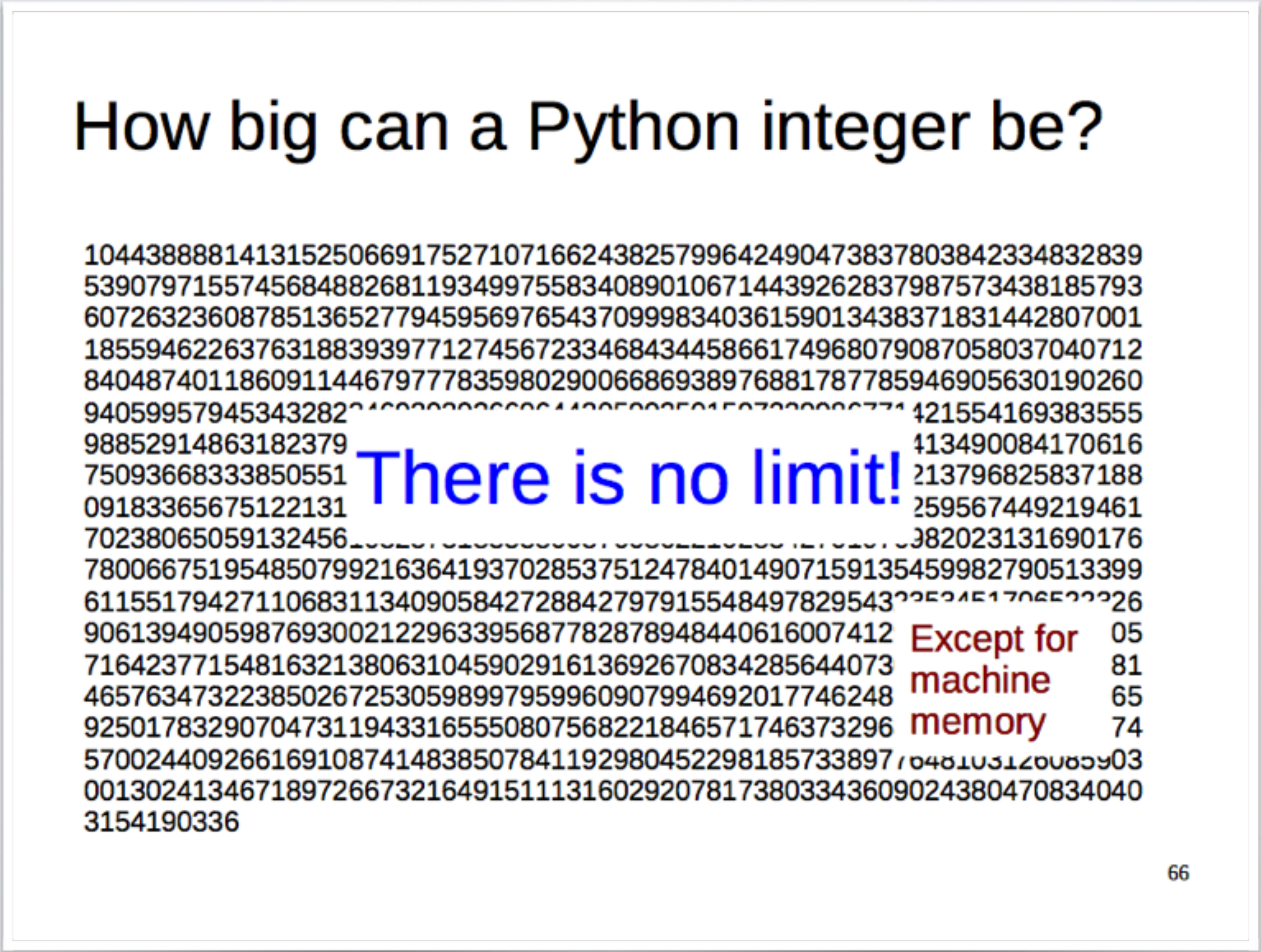
How big can a Python integer be?

```
>>> 4294967296**2
18446744073709551616

>>> 18446744073709551616**2
340282366920938463463374607431768211456

>>> 340282366920938463463374607431768211456**2
1157920892373161954235709850086879078532699846
65640564039457584007913129639936

>>> 1157920892373161954235709850086879078532699
84665640564039457584007913129639936**2
1340780792994259709957402499820584612747936582
0592393377723561443721764030073546976801874298
1669034276900318581864860508537538828119465699
46433649006084096
```

65

Slide taken from University of Cambridge's Python AB Course

Python Basics



How big can a Python integer be?

```
10443888814131525066917527107166243825799642490473837803842334832839
53907971557456848826811934997558340890106714439262837987573438185793
60726323608785136527794595697654370999834036159013438371831442807001
18559462263763188393977127456723346843445866174968079087058037040712
84048740118609114467977783598029006686938976881787785946905630190260
9405995794534328[illegible]421554169383555
98852914863182379 [illegible] 413490084170616
75093668333850551 [illegible] 213796825837188
09183365675122131 [illegible] 259567449219461
70238065059132456[illegible]982023131690176
78006675195485079921636419370285375124784014907159135459982790513399
6115517942711068311340905842728842797915548497829543[illegible]26
9061394905987693002122963395687782878948440616007412 [illegible] 05
7164237715481632138063104590291613692670834285644073 [illegible] 81
4657634732238502672530598997959960907994692017746248 [illegible] 65
9250178329070473119433165550807568221846571746373296 [illegible] 74
5700244092661691087414838507841192980452298185733897[illegible]03
00130241346718972667321649151113160292078173803343609024380470834040
3154190336
```

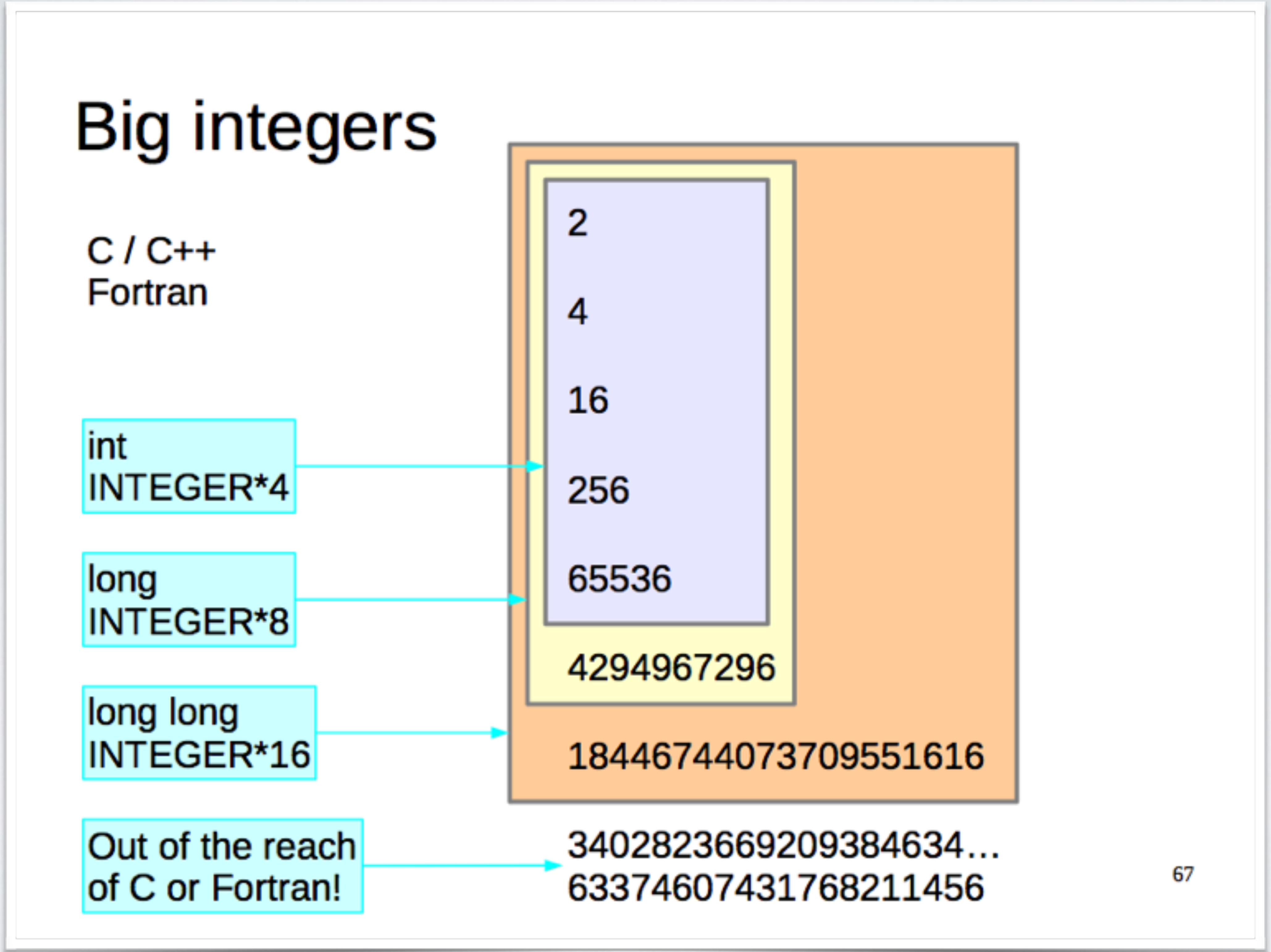
There is no limit!

Except for machine memory

66

Slide taken from University of Cambridge's Python AB Course

Python Basics

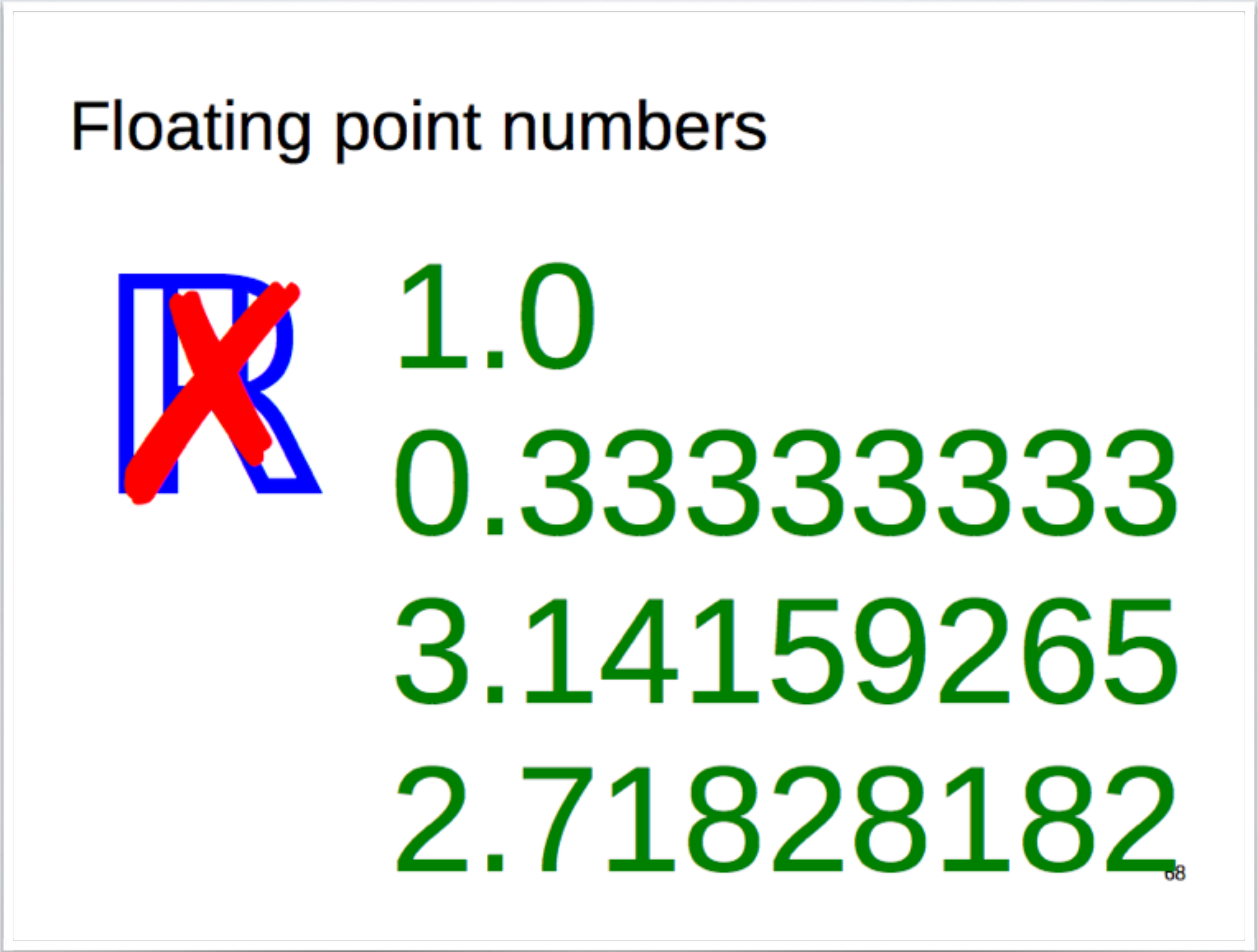


Big integers

C / C++
Fortran

Slide taken from University of Cambridge's Python AB Course

Python Basics



Floating point numbers

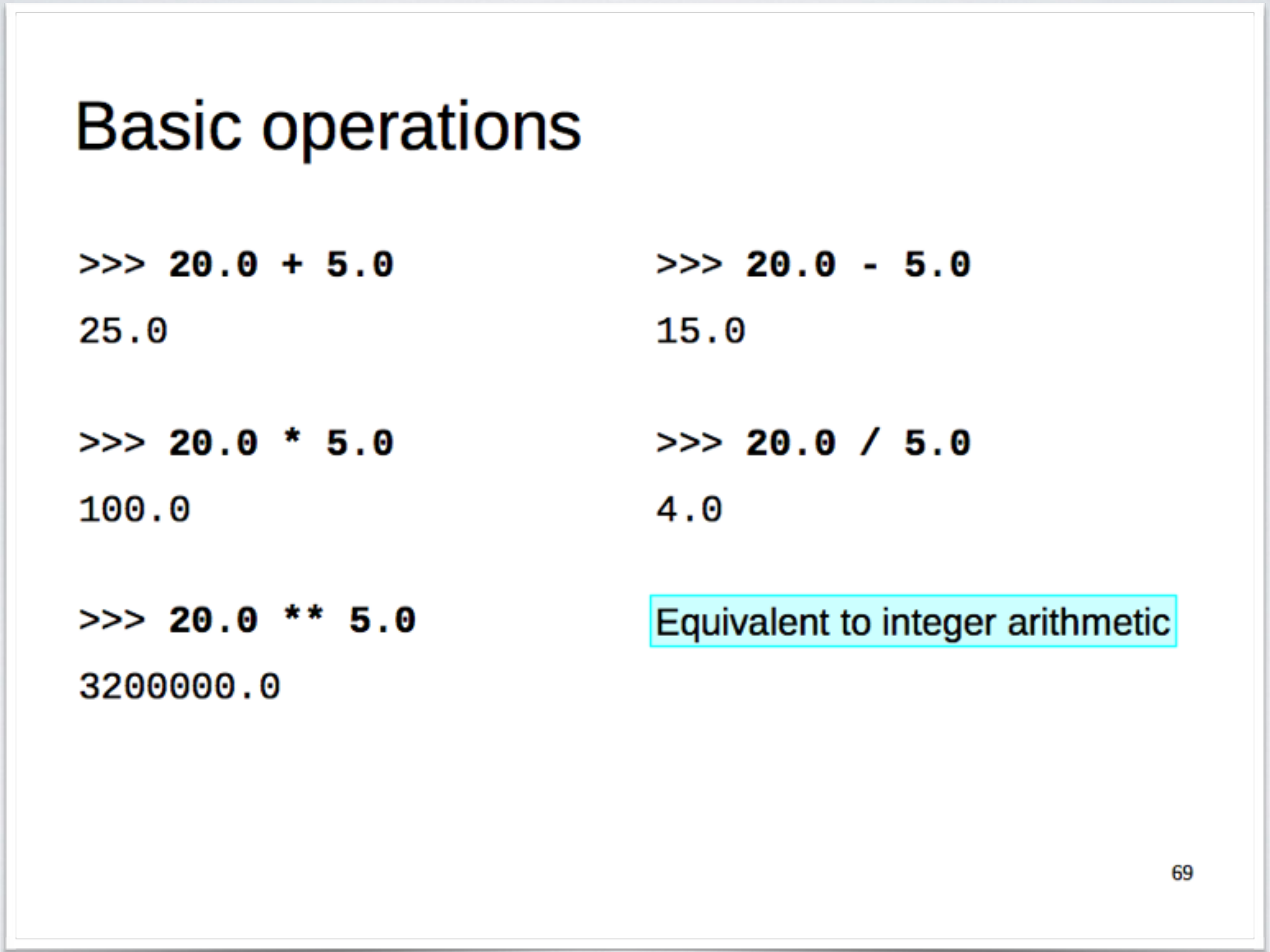
$\cancel{\mathbb{R}}$

1.0
0.33333333
3.14159265
2.71828182

68

Slide taken from University of Cambridge's Python AB Course

Python Basics



Basic operations

```
>>> 20.0 + 5.0
25.0
```

```
>>> 20.0 - 5.0
15.0
```

```
>>> 20.0 * 5.0
100.0
```

```
>>> 20.0 / 5.0
4.0
```

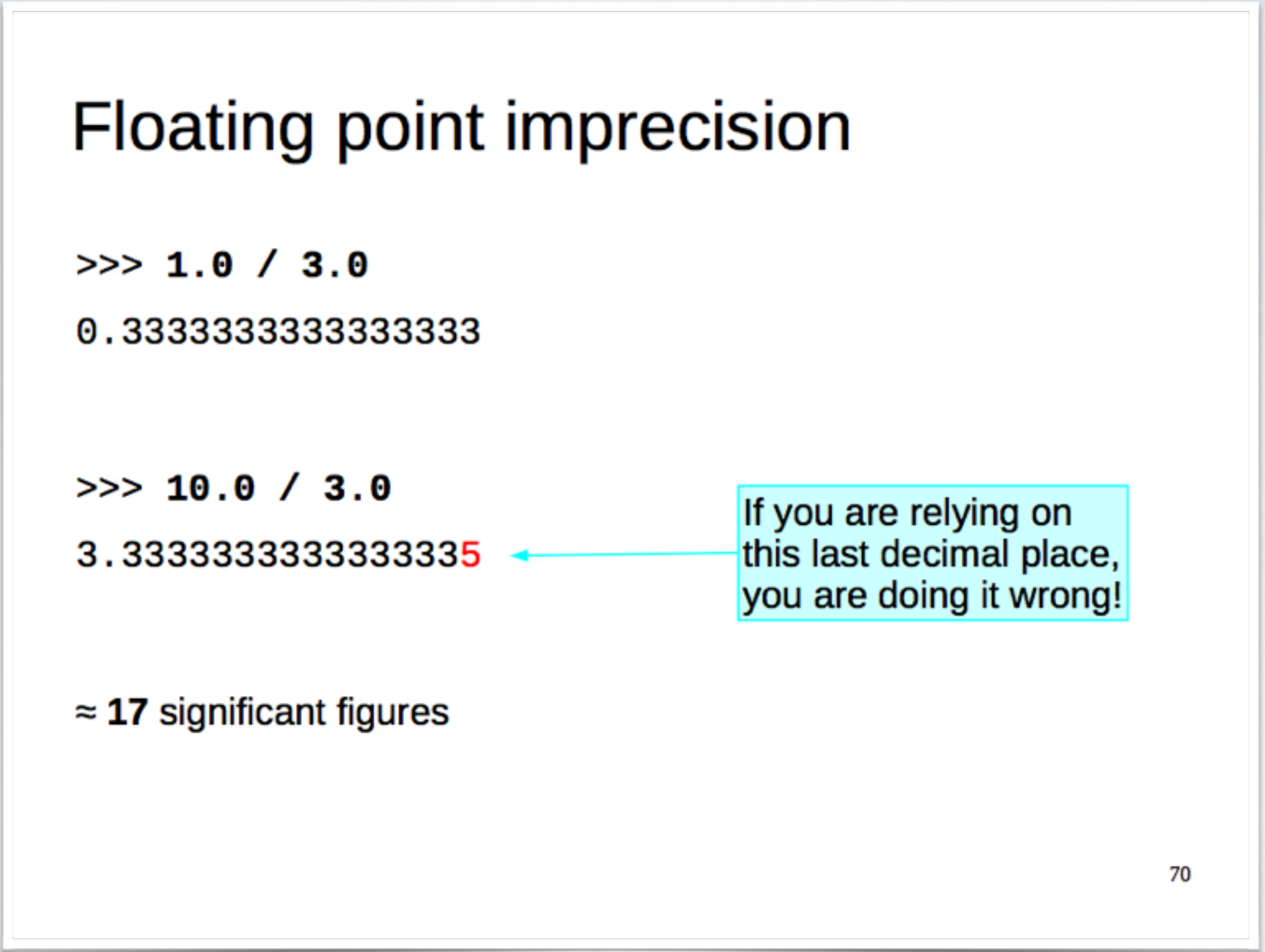
```
>>> 20.0 ** 5.0
3200000.0
```

Equivalent to integer arithmetic

69

Slide taken from University of Cambridge's Python AB Course

Python Basics



Floating point imprecision

```
>>> 1.0 / 3.0
0.3333333333333333


>>> 10.0 / 3.0
3.3333333333333335
```

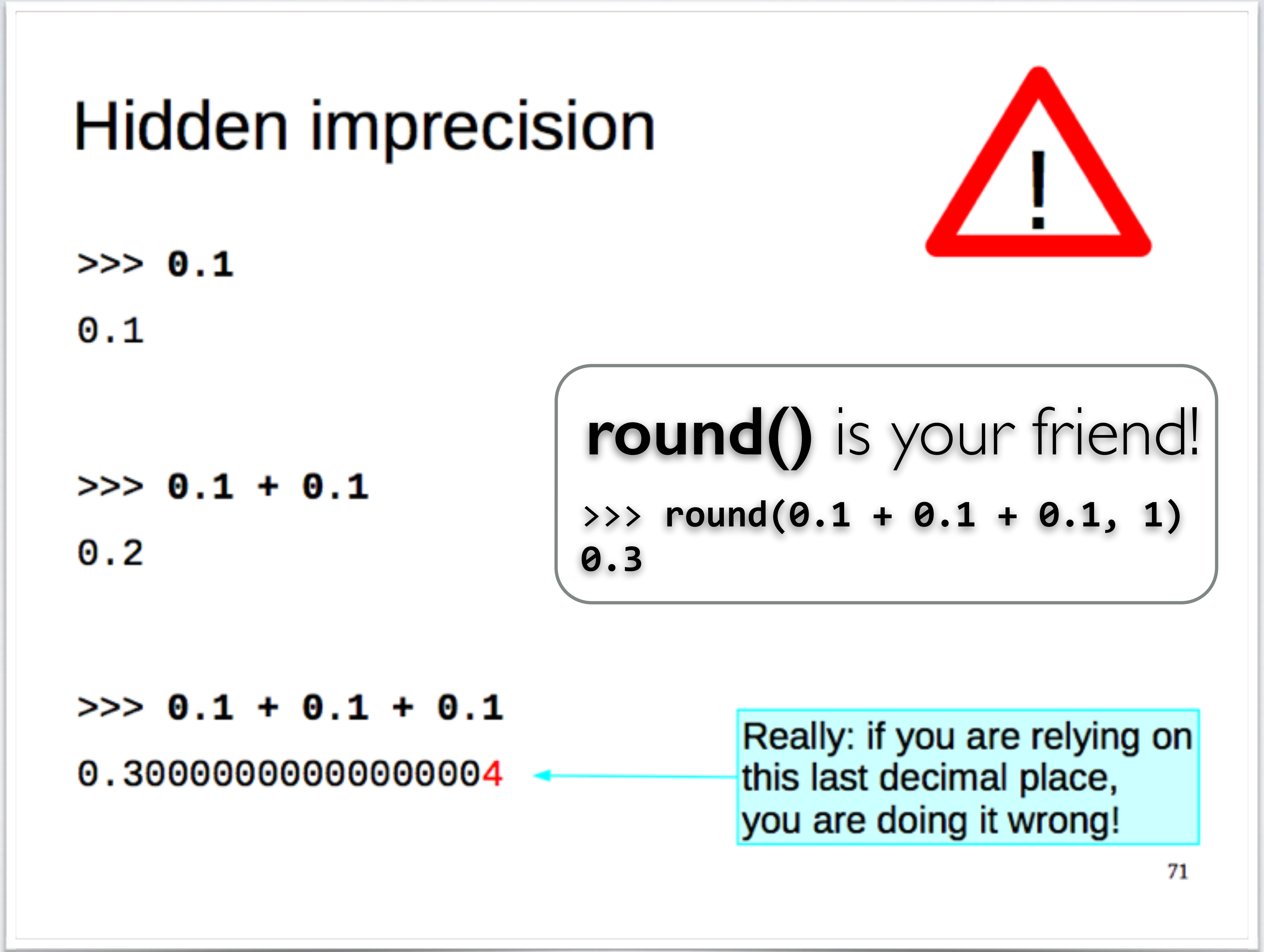
If you are relying on this last decimal place, you are doing it wrong!

≈ **17** significant figures

70

Slide taken from University of Cambridge's Python AB Course

Python Basics



Hidden imprecision

```
>>> 0.1
0.1

>>> 0.1 + 0.1
0.2

>>> 0.1 + 0.1 + 0.1
0.30000000000000004
```

Really: if you are relying on this last decimal place, you are doing it wrong!

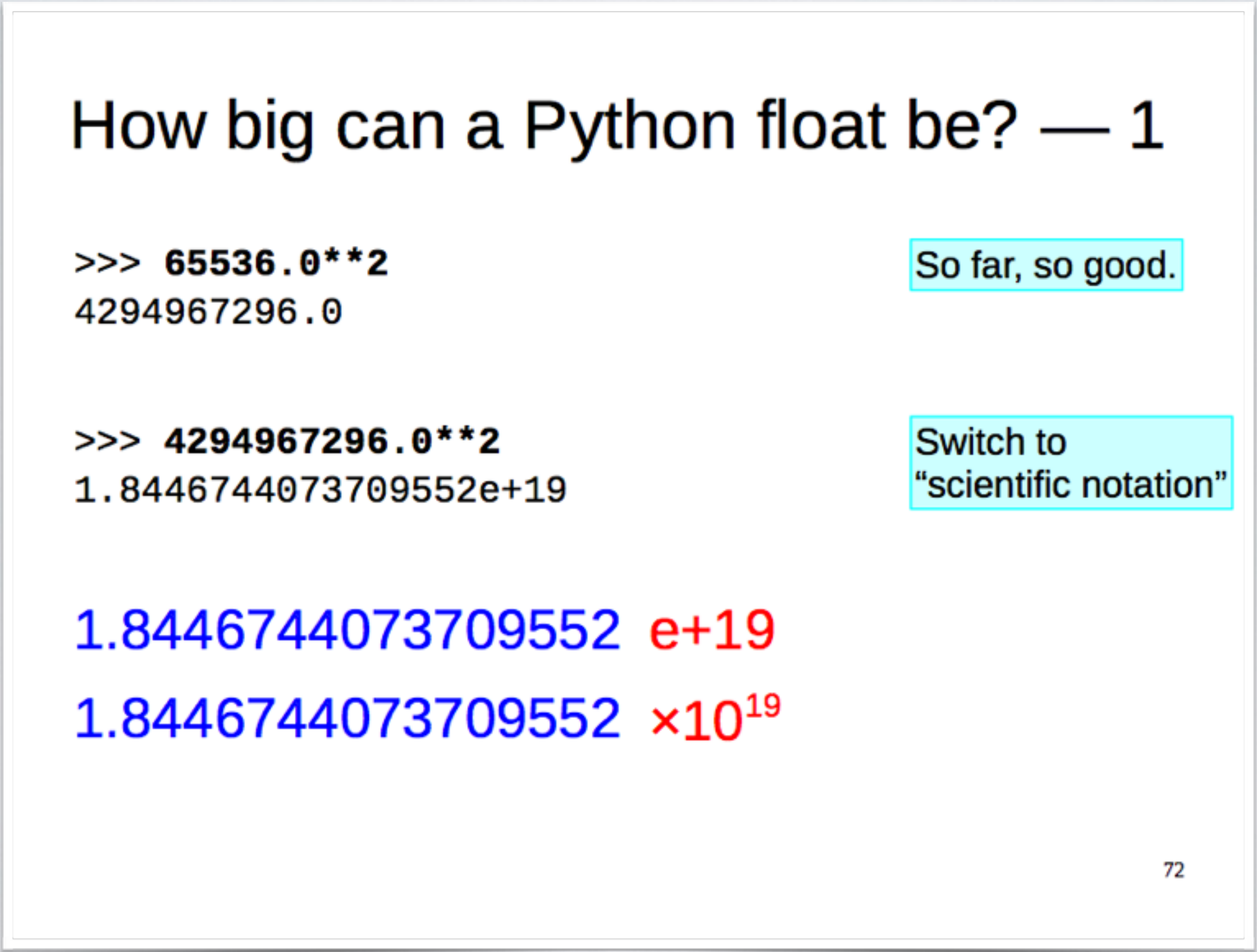
71

round() is your friend!

```
>>> round(0.1 + 0.1 + 0.1, 1)
0.3
```

Slide taken from University of Cambridge's Python AB Course

Python Basics



How big can a Python float be? — 1

```
>>> 65536.0**2
4294967296.0
```

So far, so good.

```
>>> 4294967296.0**2
1.8446744073709552e+19
```

Switch to "scientific notation"

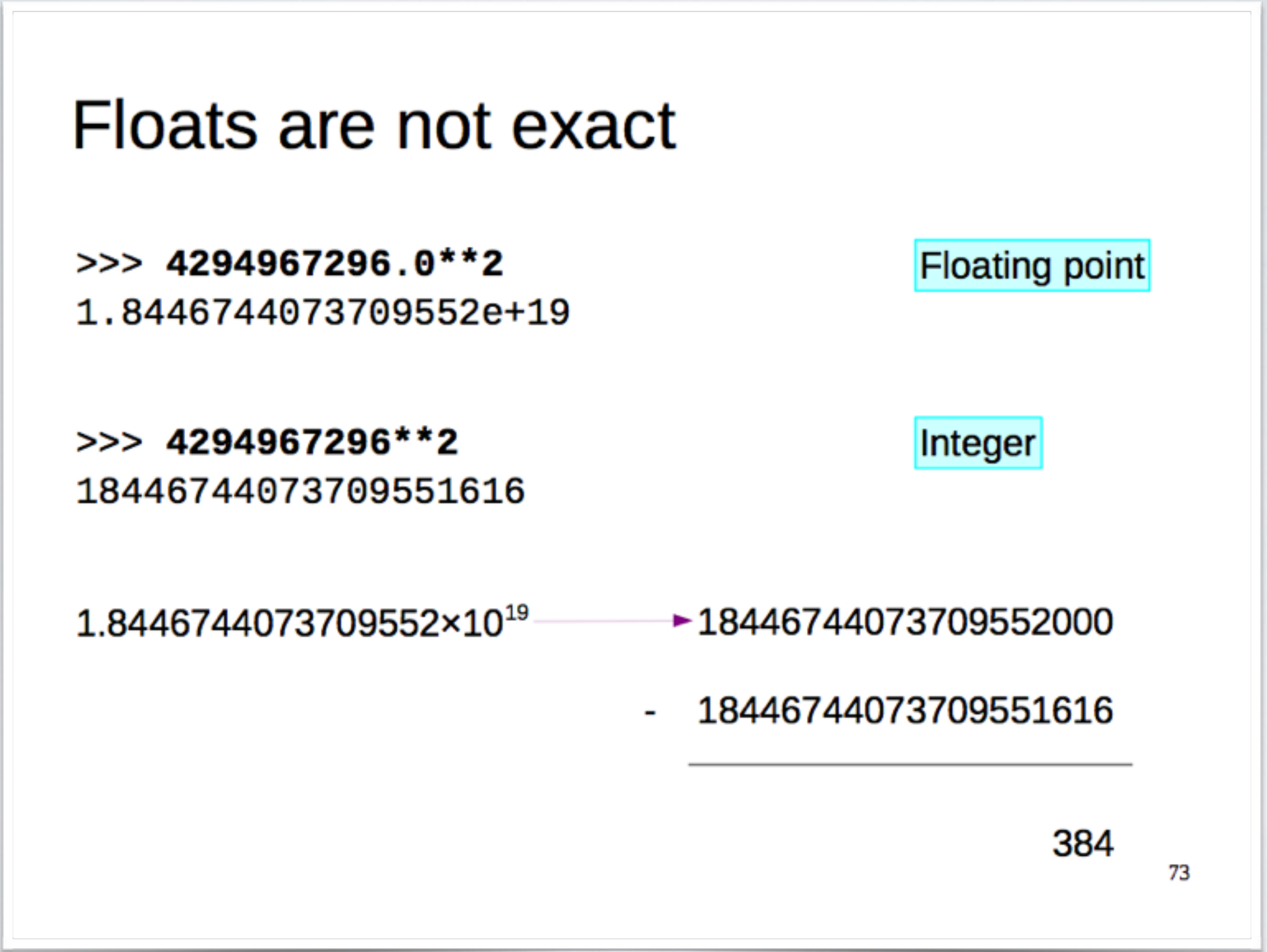
1.8446744073709552 e+19

1.8446744073709552 $\times 10^{19}$

72

Slide taken from University of Cambridge's Python AB Course

Python Basics



Floats are not exact

```
>>> 4294967296.0**2
1.8446744073709552e+19
```

Floating point

```
>>> 4294967296**2
18446744073709551616
```

Integer

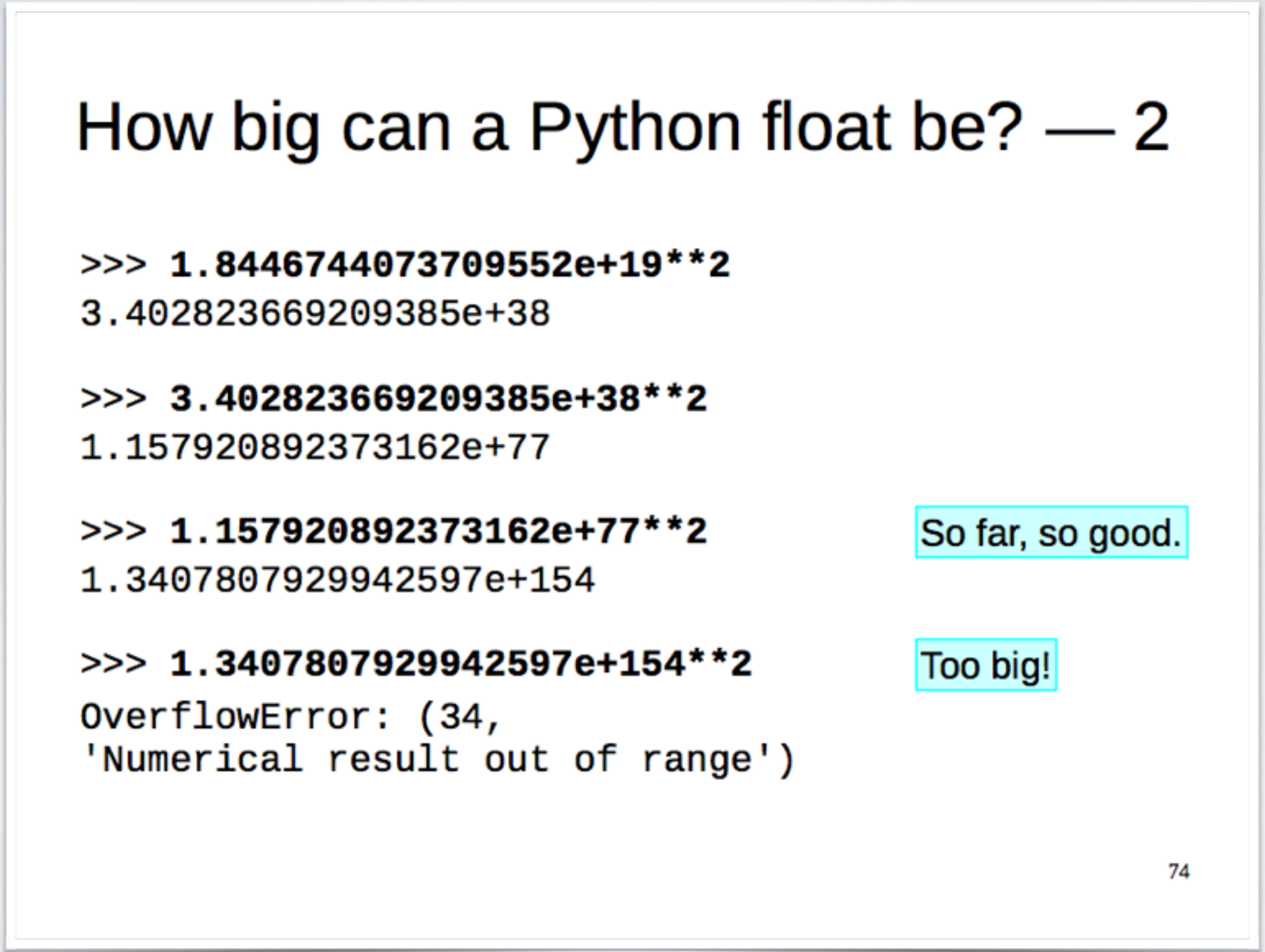
$1.8446744073709552 \times 10^{19} \rightarrow 18446744073709552000$

$$\begin{array}{rr} & 18446744073709552000 \\ - & 18446744073709551616 \\ \hline & 384 \end{array}$$

73

Slide taken from University of Cambridge's Python AB Course

Python Basics



How big can a Python float be? — 2

```
>>> 1.8446744073709552e+19**2
3.402823669209385e+38

>>> 3.402823669209385e+38**2
1.157920892373162e+77

>>> 1.157920892373162e+77**2
1.3407807929942597e+154

>>> 1.3407807929942597e+154**2
OverflowError: (34,
'Numerical result out of range')
```

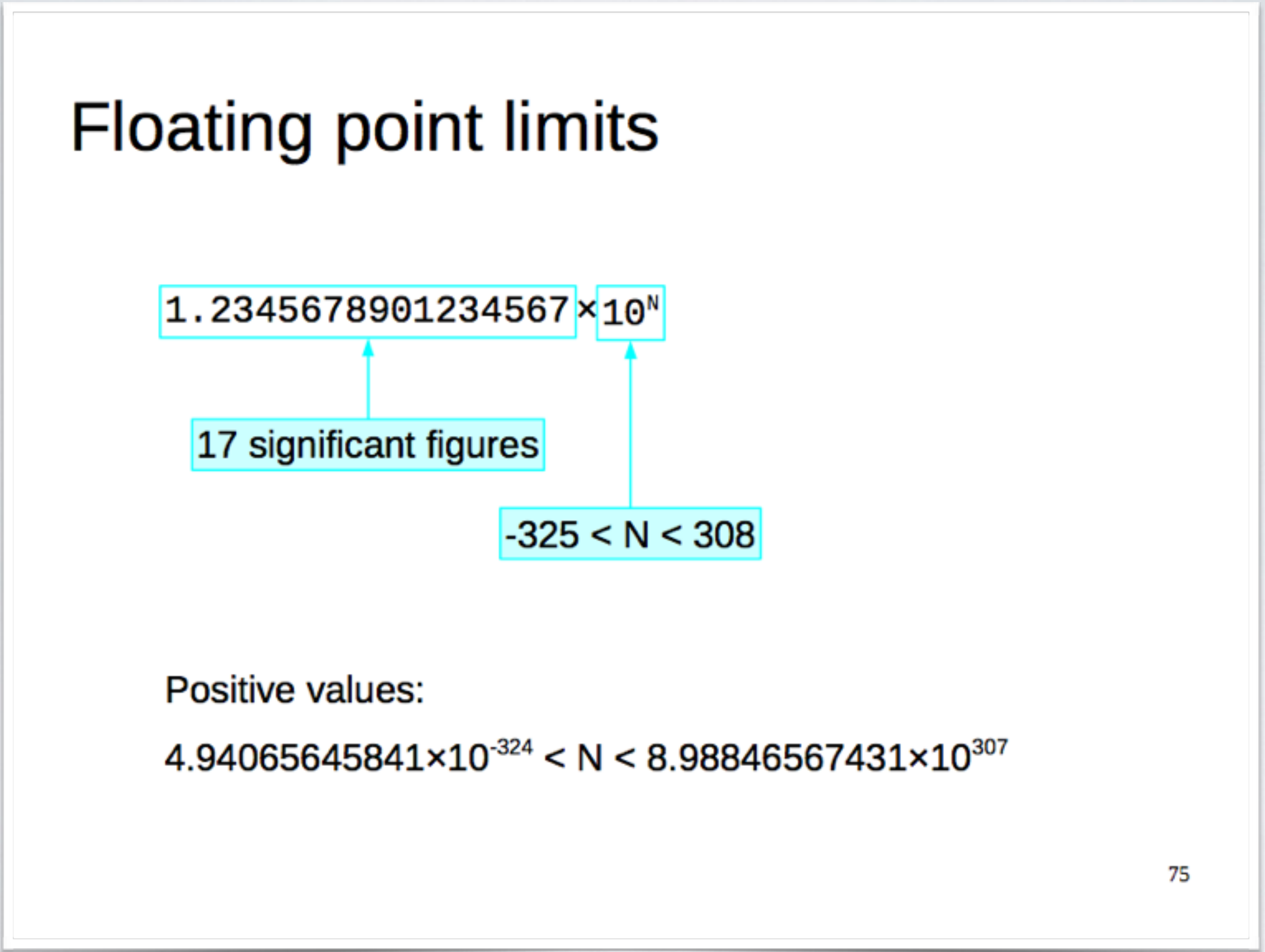
So far, so good.

Too big!

74

Slide taken from University of Cambridge's Python AB Course

Python Basics



Floating point limits

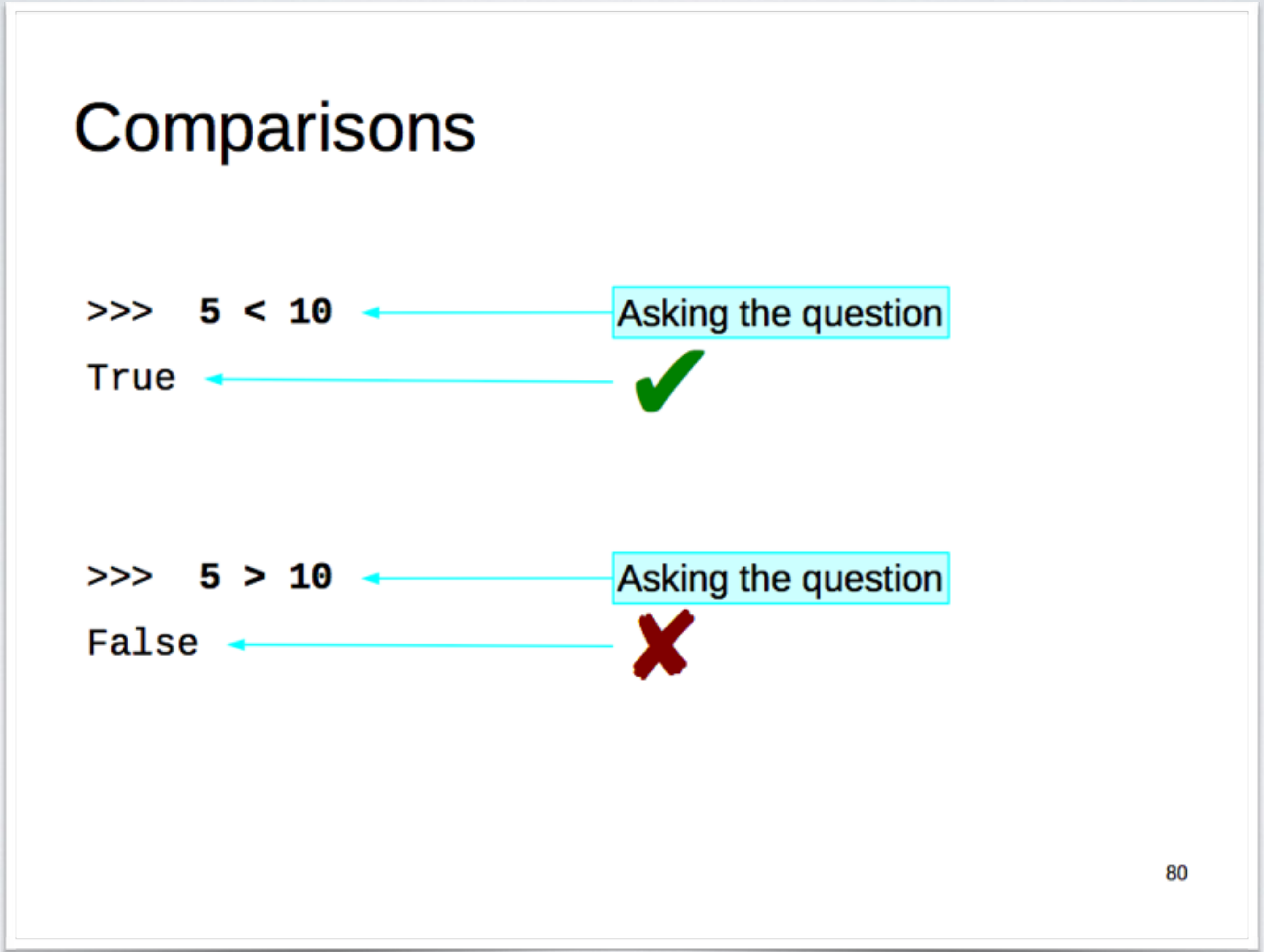
Positive values:

$4.94065645841 \times 10^{-324} < N < 8.9884656743 \times 10^{307}$

75

Slide taken from University of Cambridge's Python AB Course

Python Basics



Comparisons

```
>>>  5 < 10
True
```

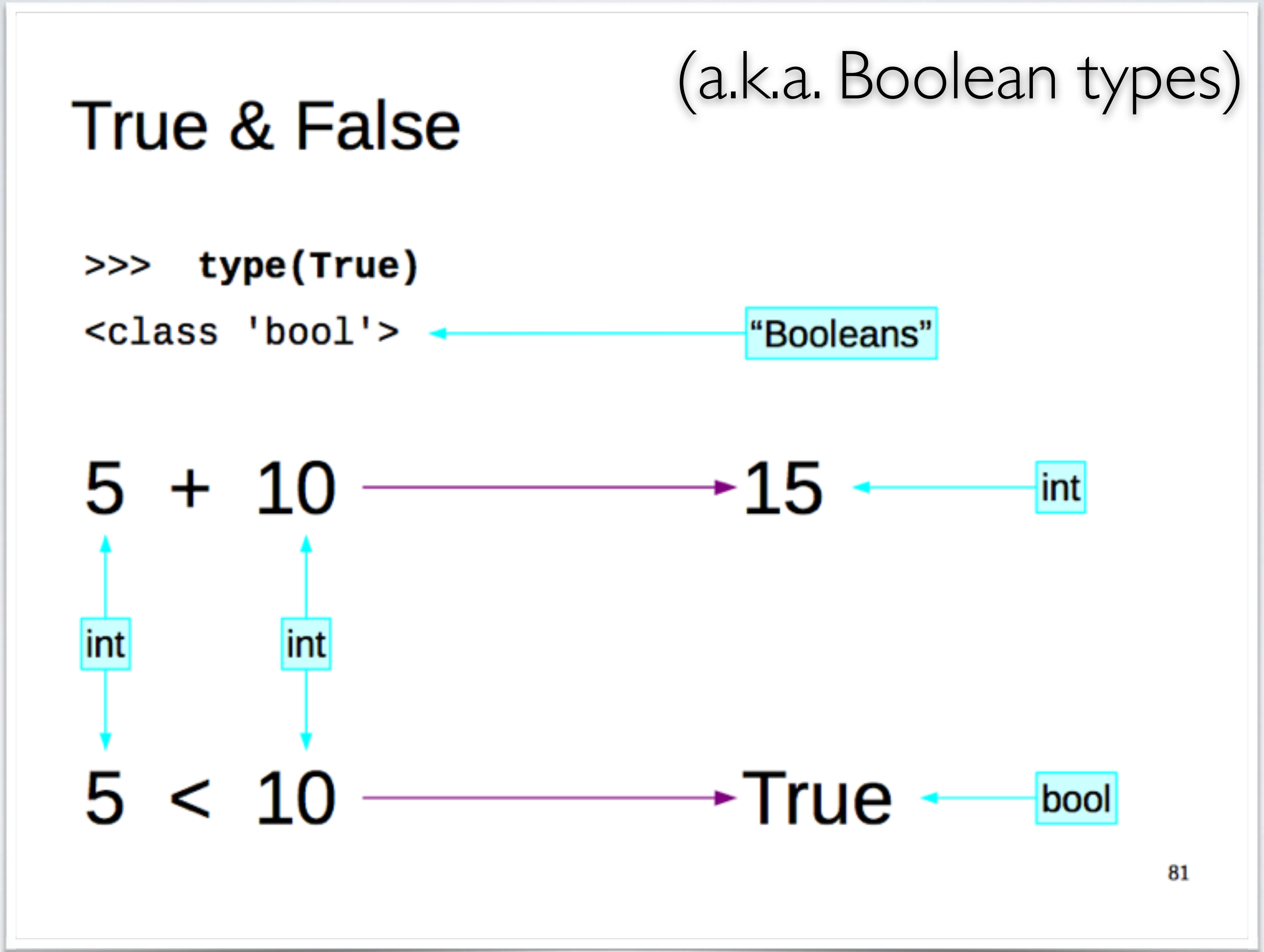
Asking the question
✓

```
>>>  5 > 10
False
```

Asking the question
✗

80

Python Basics



(a.k.a. Boolean types)

True & False

```
>>>  type(True)
<class 'bool'>
```

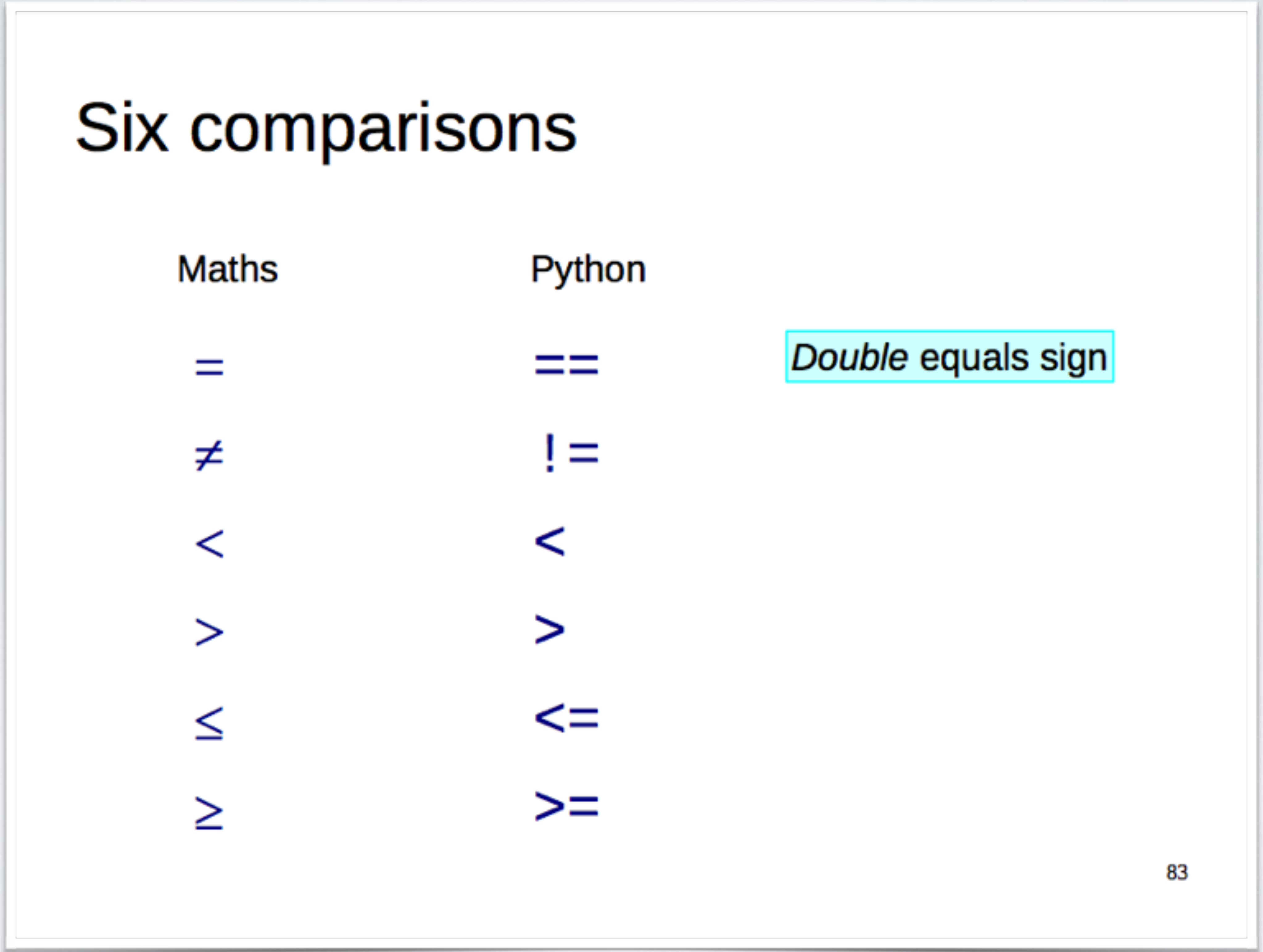
5 + 10 → 15

5 < 10 → True

81

Slide taken from University of Cambridge’s Python AB Course

Python Basics



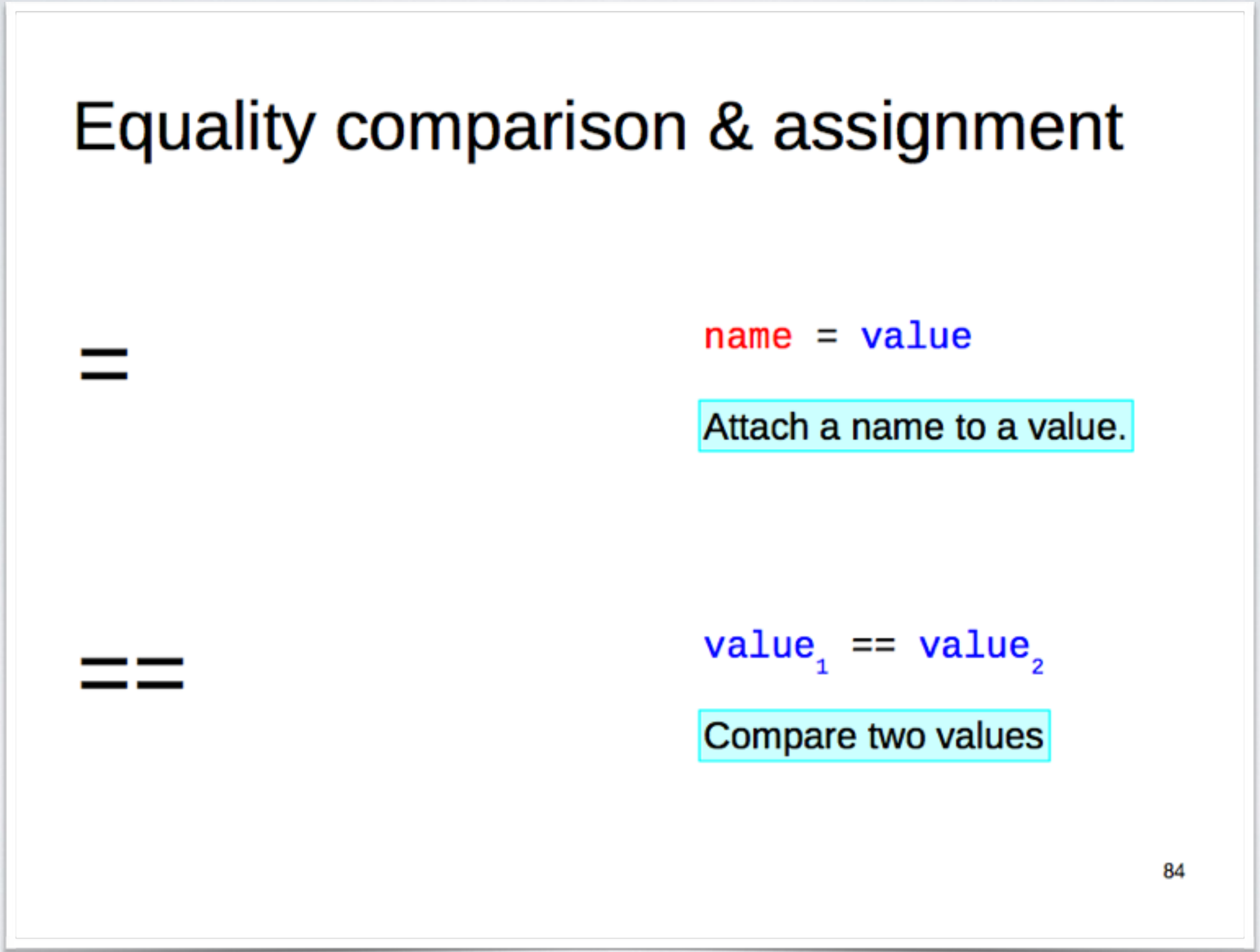
Six comparisons

Maths	Python	
=	==	*Double* equals sign
≠	!=	
<	<	
>	>	
≤	<=	
≥	>=	

83

Slide taken from University of Cambridge's Python AB Course

Python Basics



Equality comparison & assignment

=

```
name = value
```

Attach a name to a value.

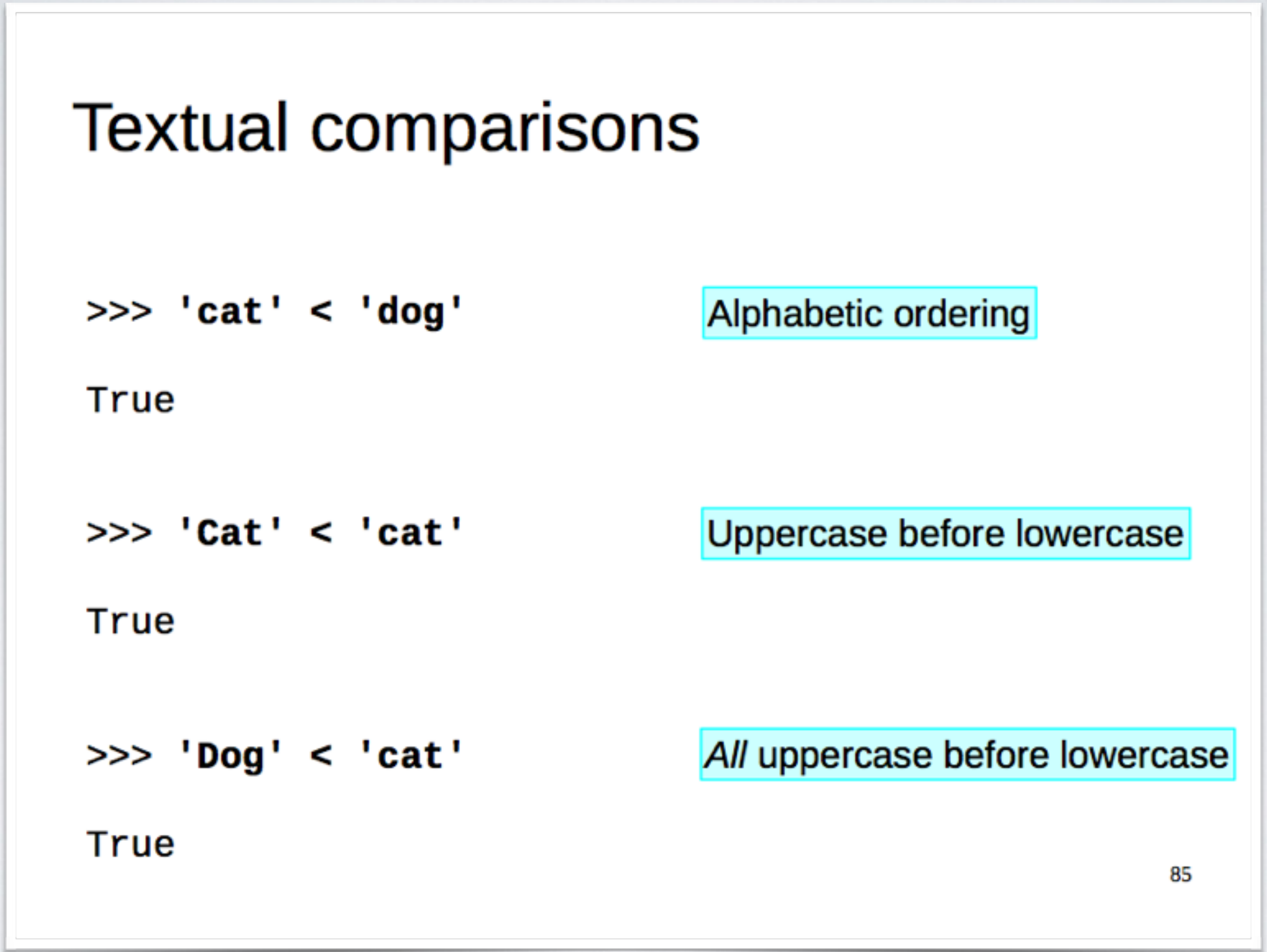
==

$\texttt{value}_1$ == $\texttt{value}_2$

Compare two values

84

Python Basics



Textual comparisons

```
>>> 'cat' < 'dog'
True
```

Alphabetic ordering

```
>>> 'Cat' < 'cat'
True
```

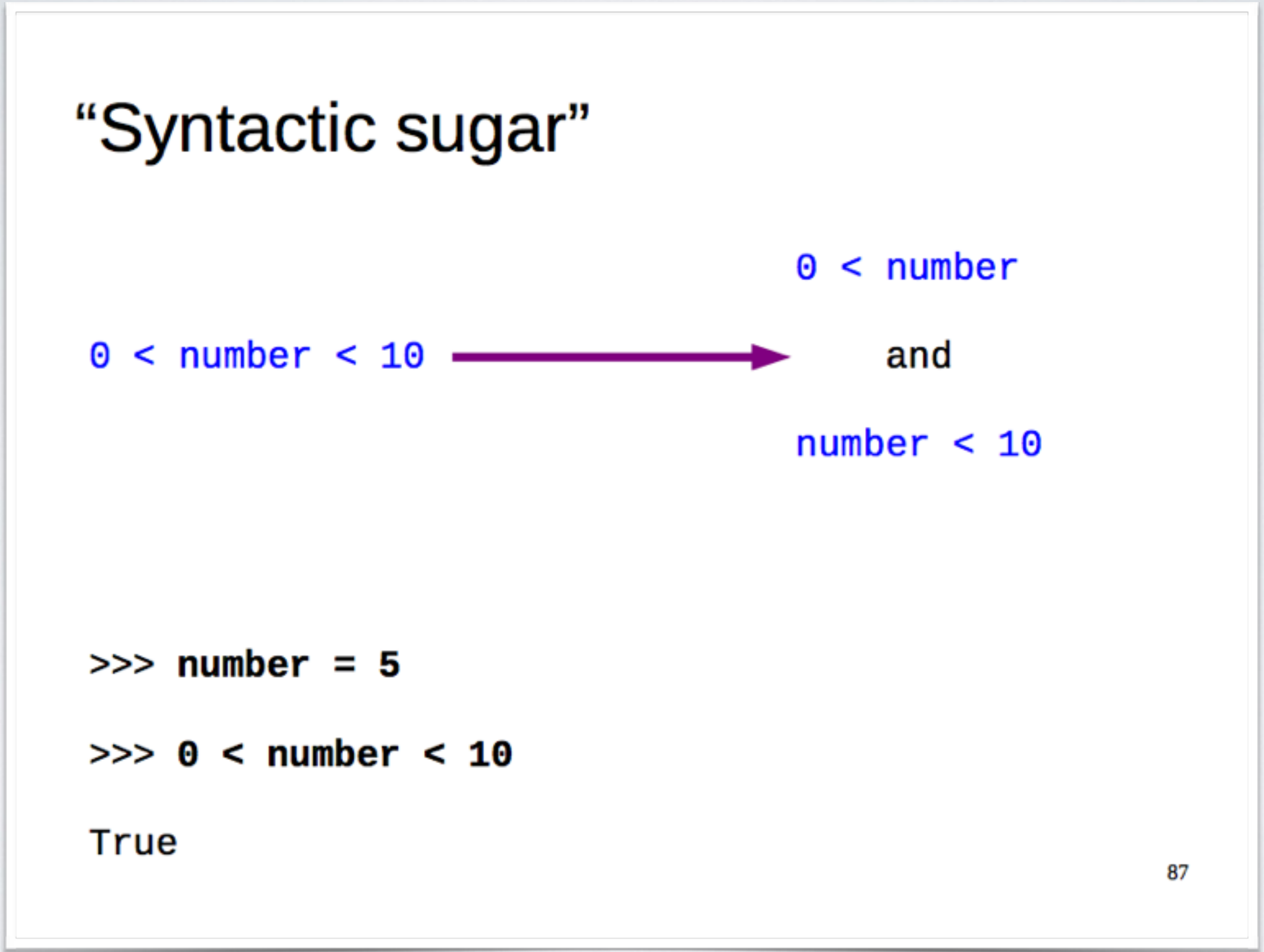
Uppercase before lowercase

```
>>> 'Dog' < 'cat'
True
```

All uppercase before lowercase

85

Python Basics



"Syntactic sugar"

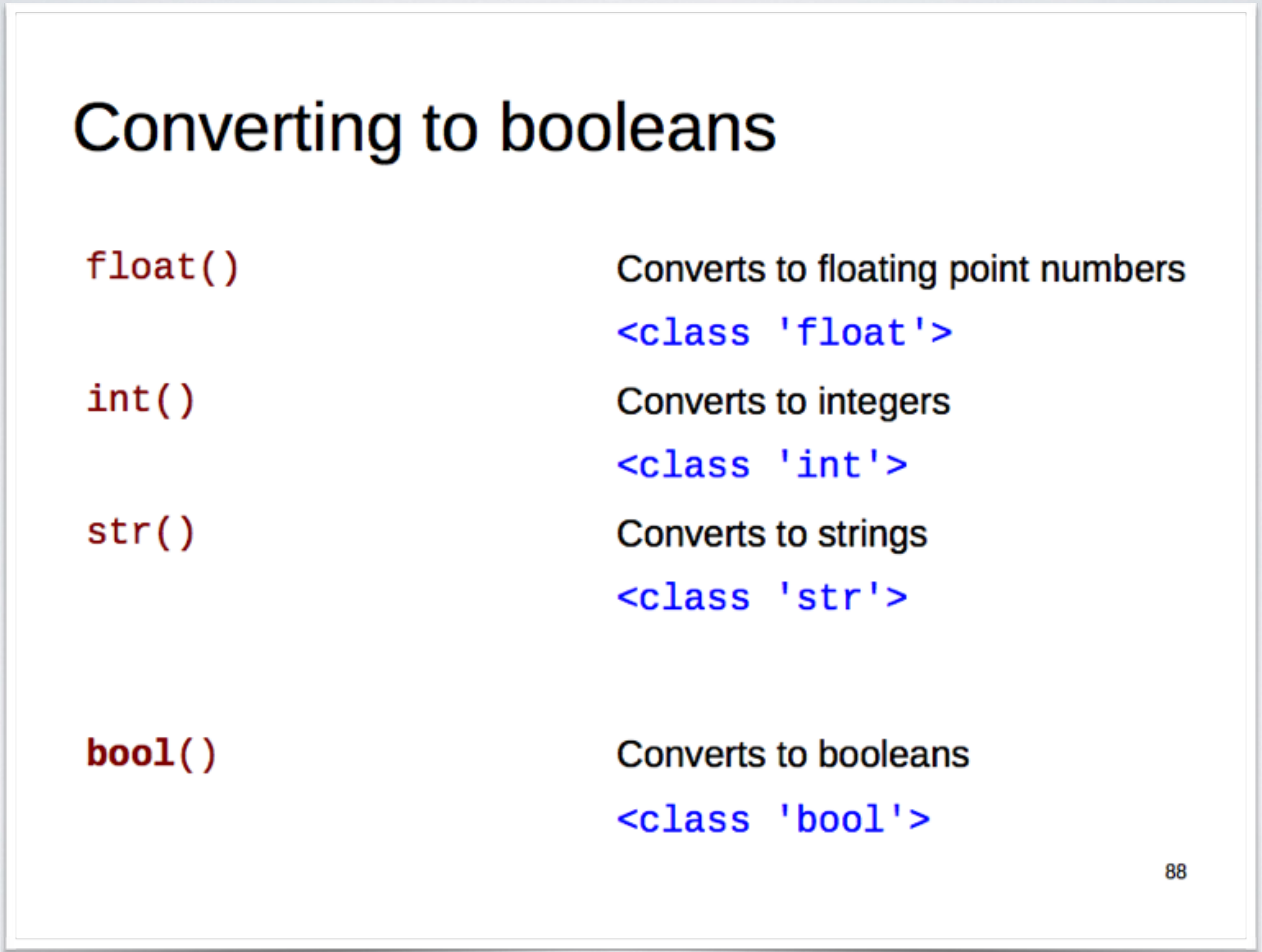
$$0 < number < 10 \xrightarrow{} \begin{matrix} 0 < number \\ \text{and} \\ number < 10 \end{matrix}$$

```
>>> number = 5
>>> 0 < number < 10
True
```

87

Slide taken from University of Cambridge's Python AB Course

Python Basics

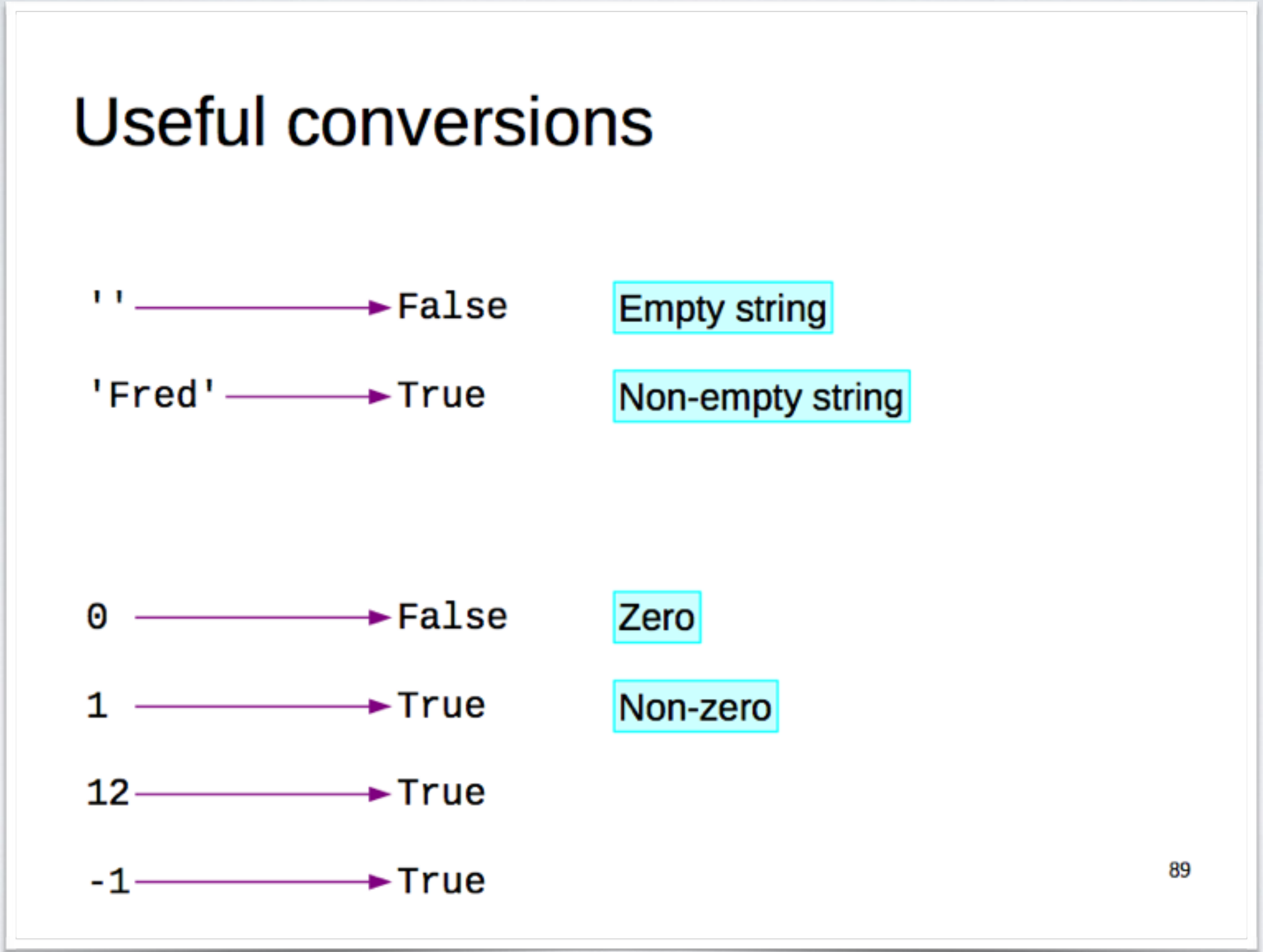


Converting to booleans

`float()`	Converts to floating point numbers `<class 'float'>`
`int()`	Converts to integers `<class 'int'>`
`str()`	Converts to strings `<class 'str'>`
`bool()`	Converts to booleans `<class 'bool'>`

88

Python Basics

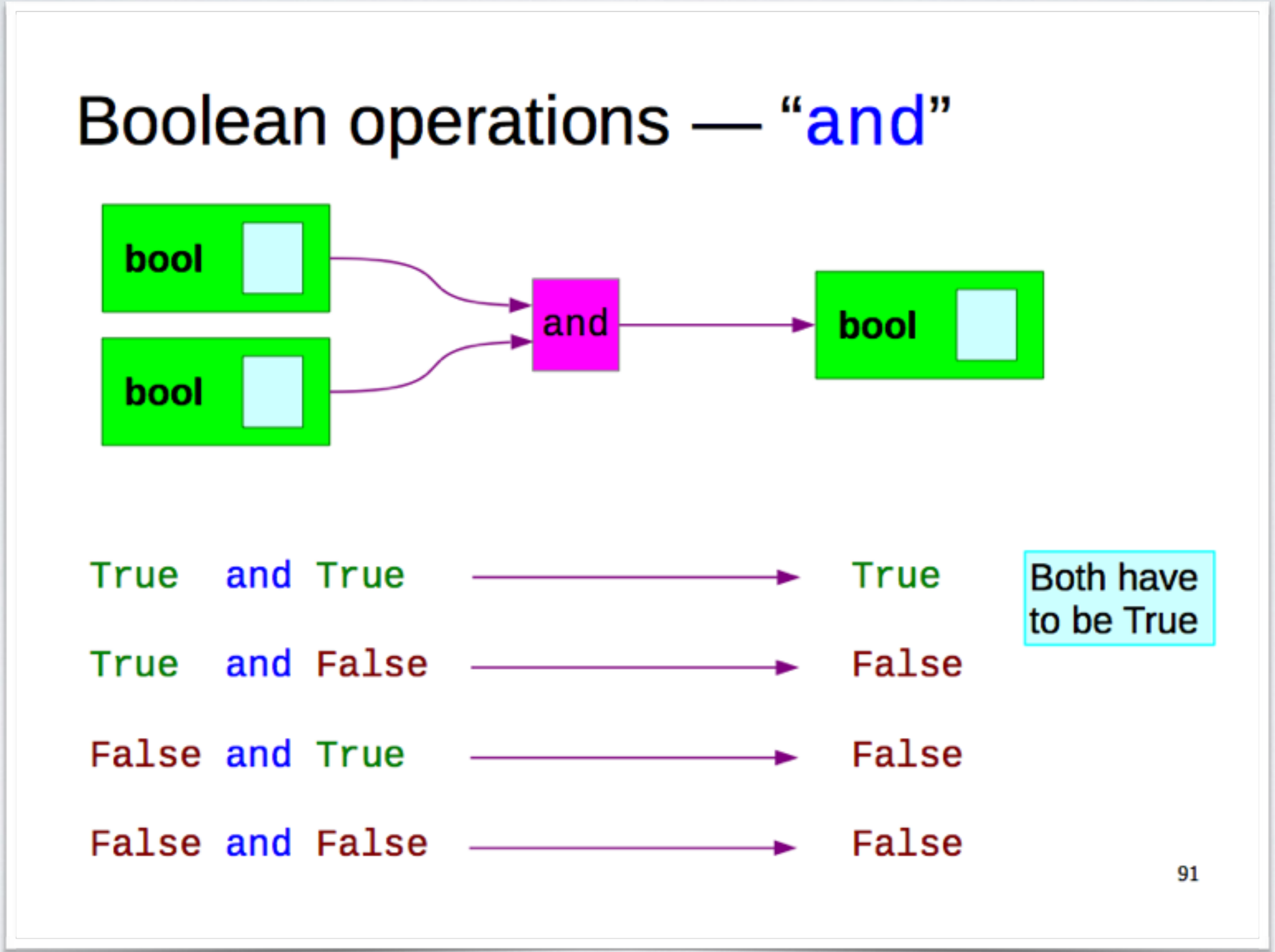


Useful conversions

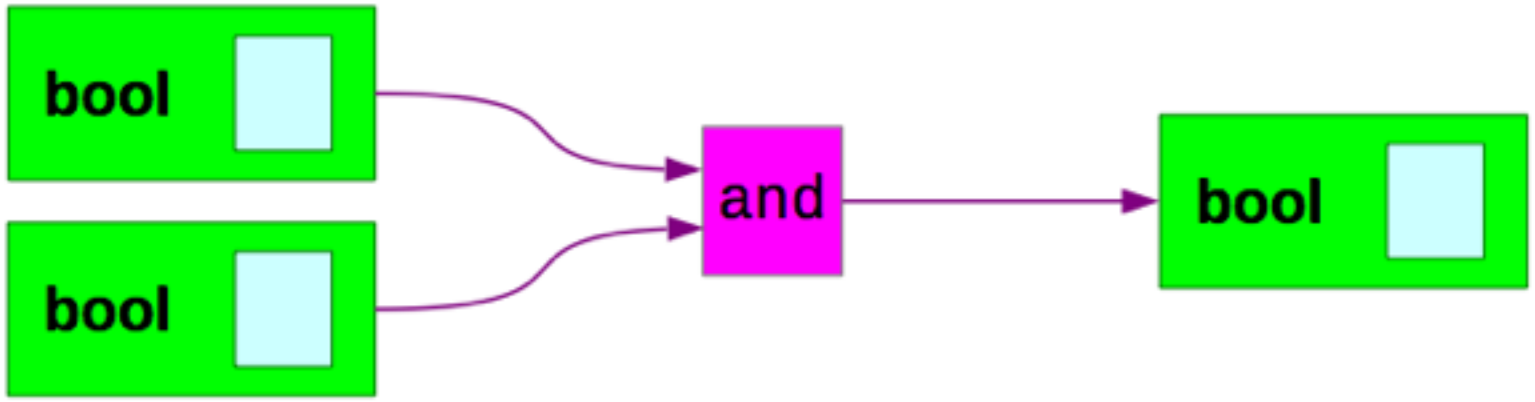
$$\begin{tikzcd}
\texttt{''} \arrow[r] & \texttt{False} & \boxed{\text{Empty string}} \\
\texttt{'Fred'} \arrow[r] & \texttt{True} & \boxed{\text{Non-empty string}} \\
\texttt{0} \arrow[r] & \texttt{False} & \boxed{\text{Zero}} \\
\texttt{1} \arrow[r] & \texttt{True} & \boxed{\text{Non-zero}} \\
\texttt{12} \arrow[r] & \texttt{True} & \\
\texttt{-1} \arrow[r] & \texttt{True} &
\end{tikzcd}$$

89

Python Basics



Boolean operations — "and"

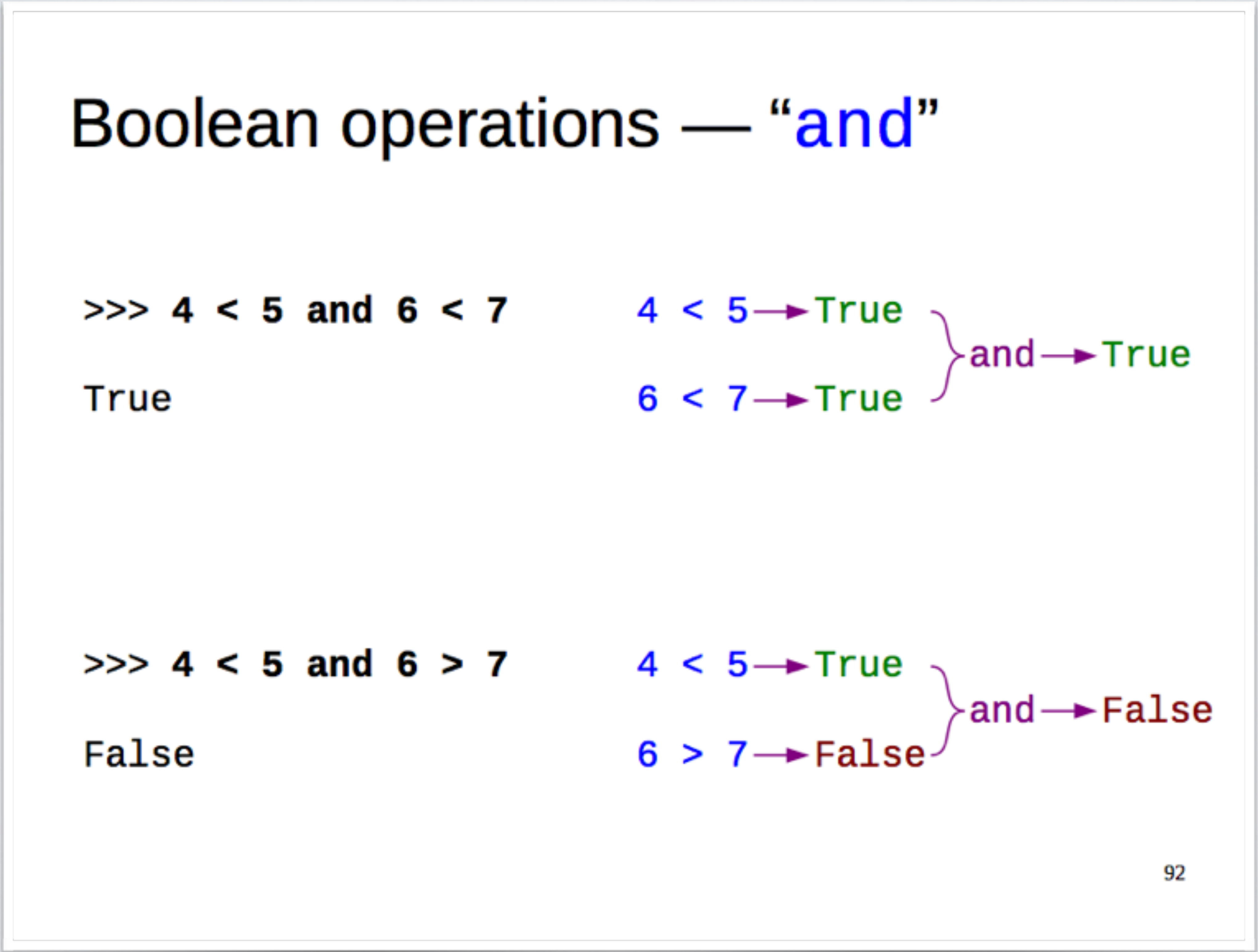


```
True  and True    ———————→    True
True  and False   ———————→    False
False and True    ———————→    False
False and False   ———————→    False
```

Both have to be True

91

Python Basics



Boolean operations — “and”

```
>>> 4 < 5 and 6 < 7
True
```

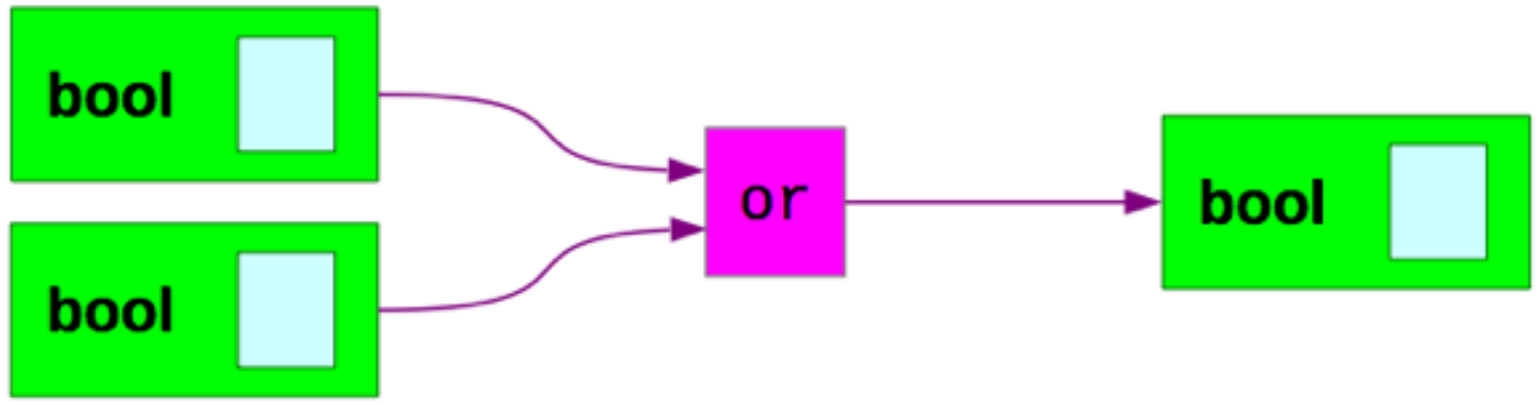
$$\left.\begin{array}{l} 4 < 5 \rightarrow \texttt{True} \\ 6 < 7 \rightarrow \texttt{True} \end{array}\right\} \texttt{and} \rightarrow \texttt{True}$$

```
>>> 4 < 5 and 6 > 7
False
```

$$\left.\begin{array}{l} 4 < 5 \rightarrow \texttt{True} \\ 6 > 7 \rightarrow \texttt{False} \end{array}\right\} \texttt{and} \rightarrow \texttt{False}$$

Python Basics

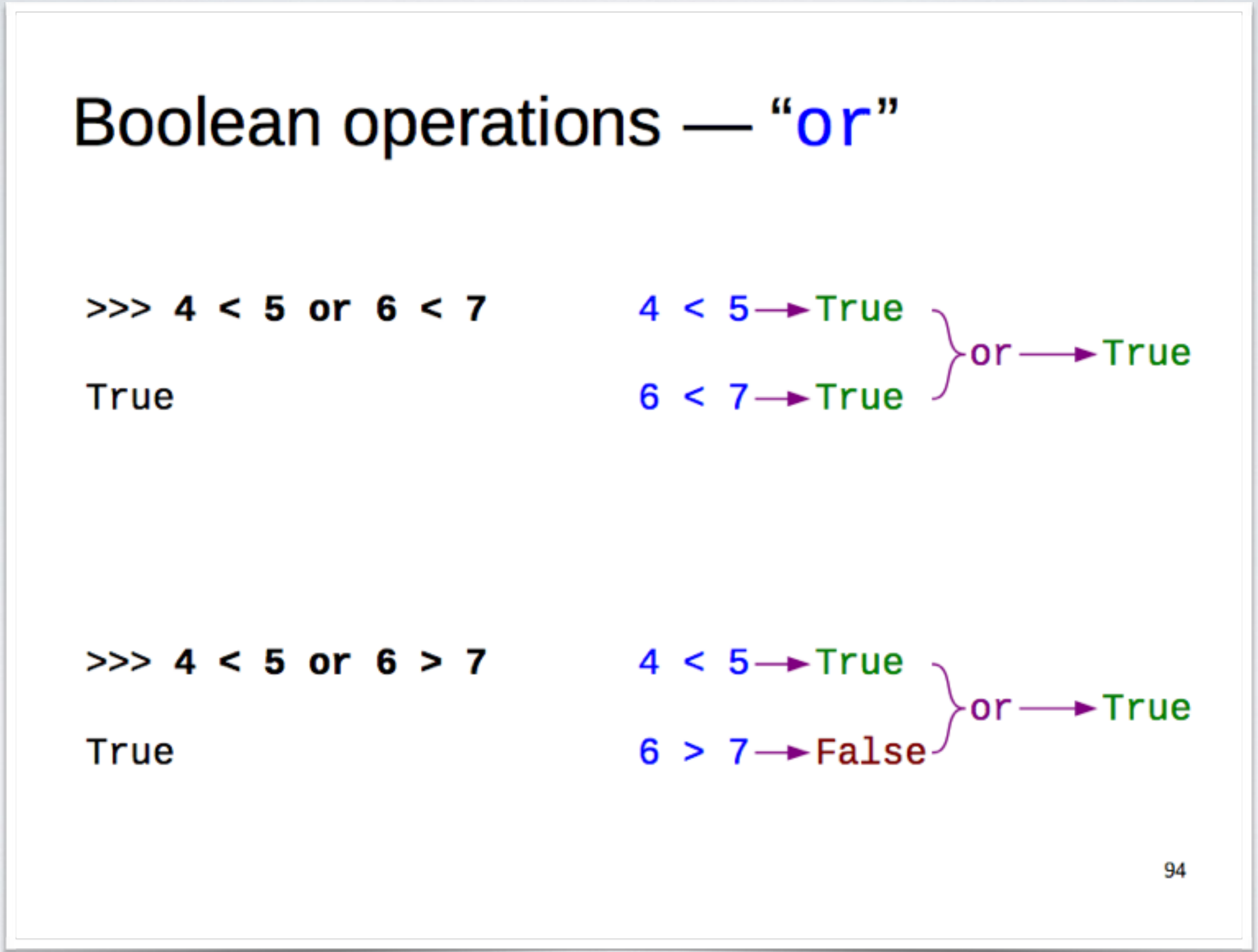
Boolean operations — “or”



```
True  or True   →   True
True  or False  →   True
False or True   →   True
False or False  →   False
```

At least one has to be True

Python Basics



Boolean operations — “or”

```
>>> 4 < 5 or 6 < 7
True
```

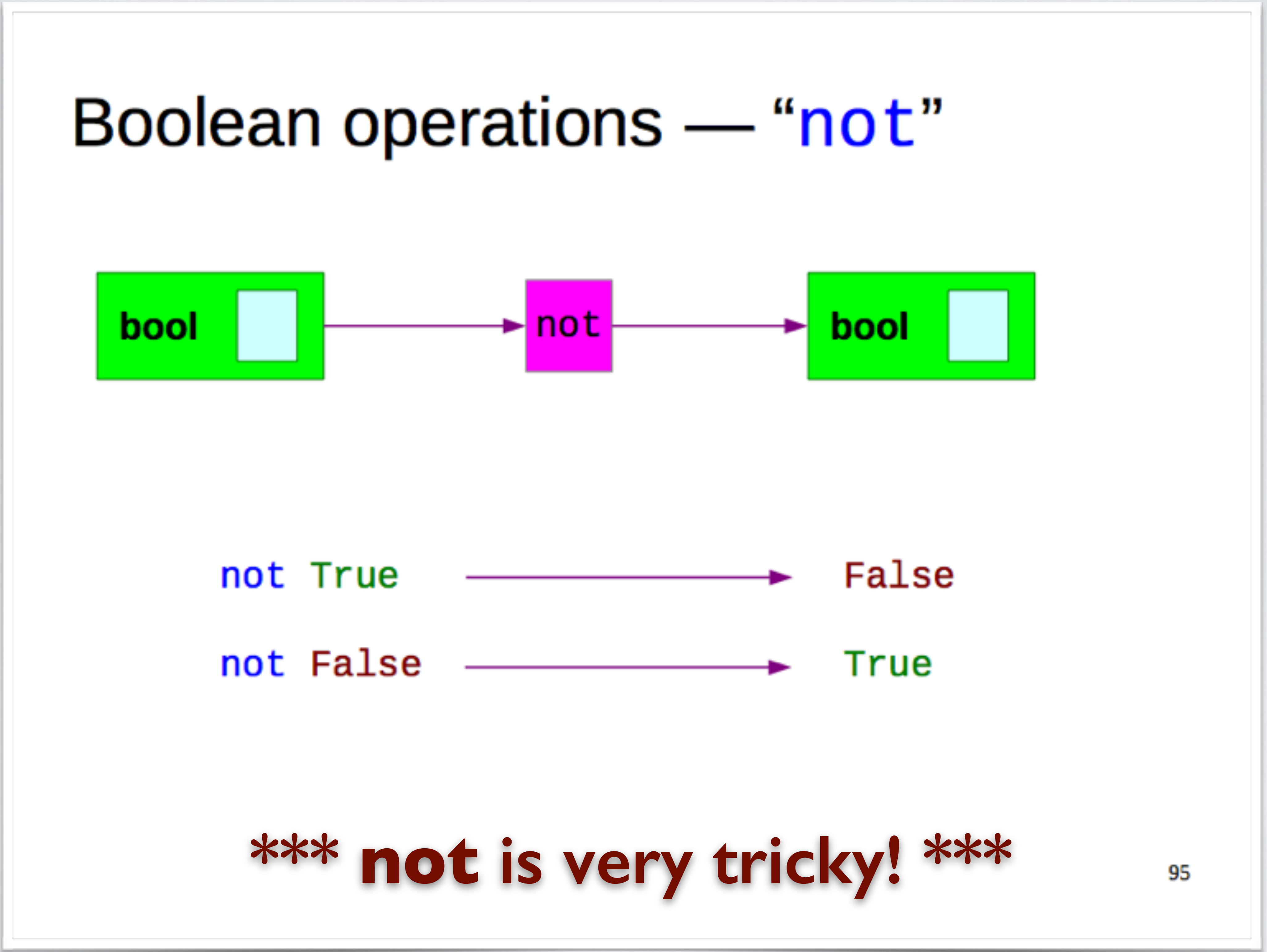
$$\begin{tikzcd} 4<5 \arrow[r] & \text{True} \arrow[dr, dash] & & \\ & & \text{or} \arrow[r] & \text{True} \\ 6<7 \arrow[r] & \text{True} \arrow[ur, dash] & & \end{tikzcd}$$

```
>>> 4 < 5 or 6 > 7
True
```

$$\begin{tikzcd} 4<5 \arrow[r] & \text{True} \arrow[dr, dash] & & \\ & & \text{or} \arrow[r] & \text{True} \\ 6>7 \arrow[r] & \text{False} \arrow[ur, dash] & & \end{tikzcd}$$

94

Python Basics



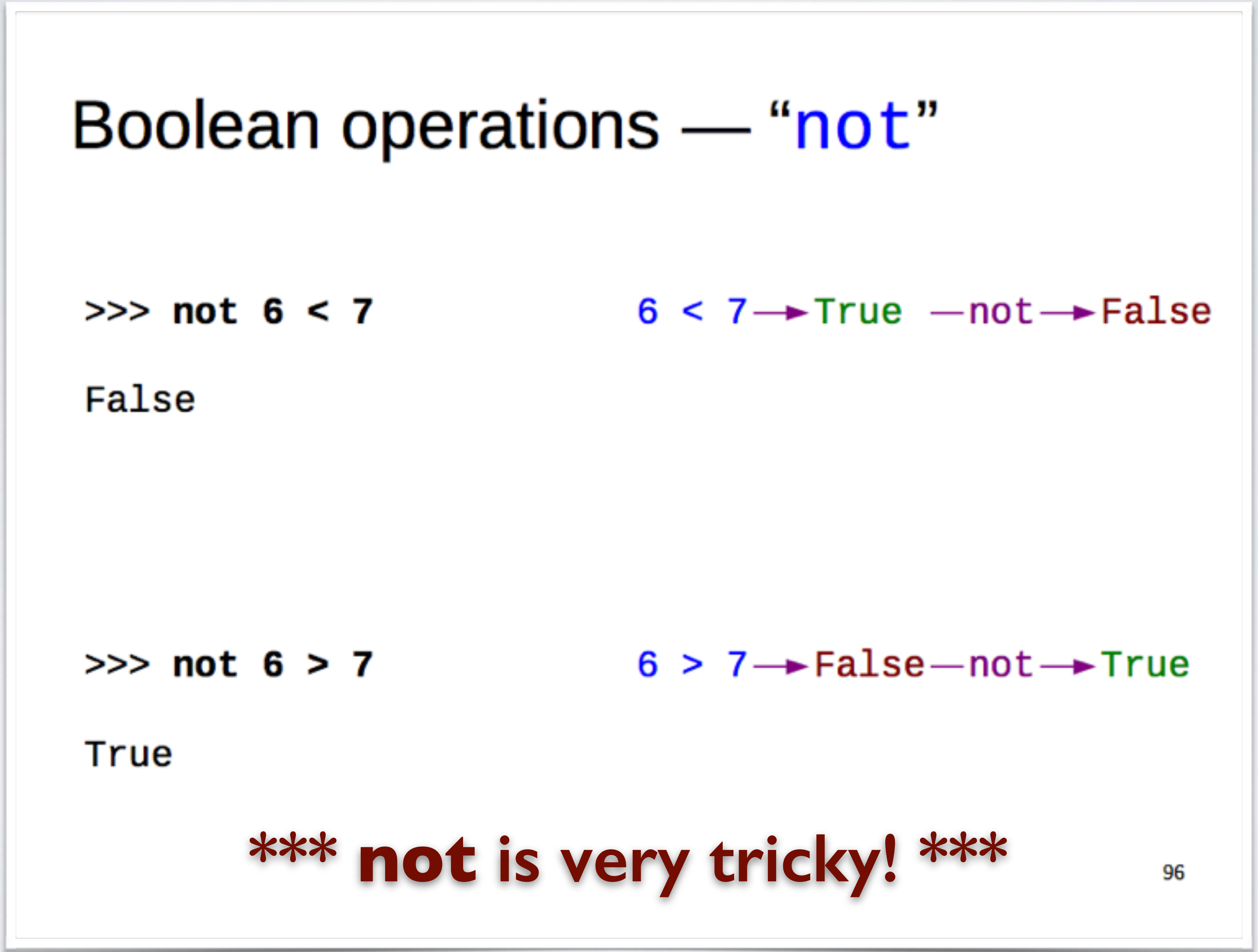
Boolean operations — "not"

bool
not
bool

```
not True            False
not False           True
```

*** **not** is very tricky! ***

Python Basics



Boolean operations — “not”

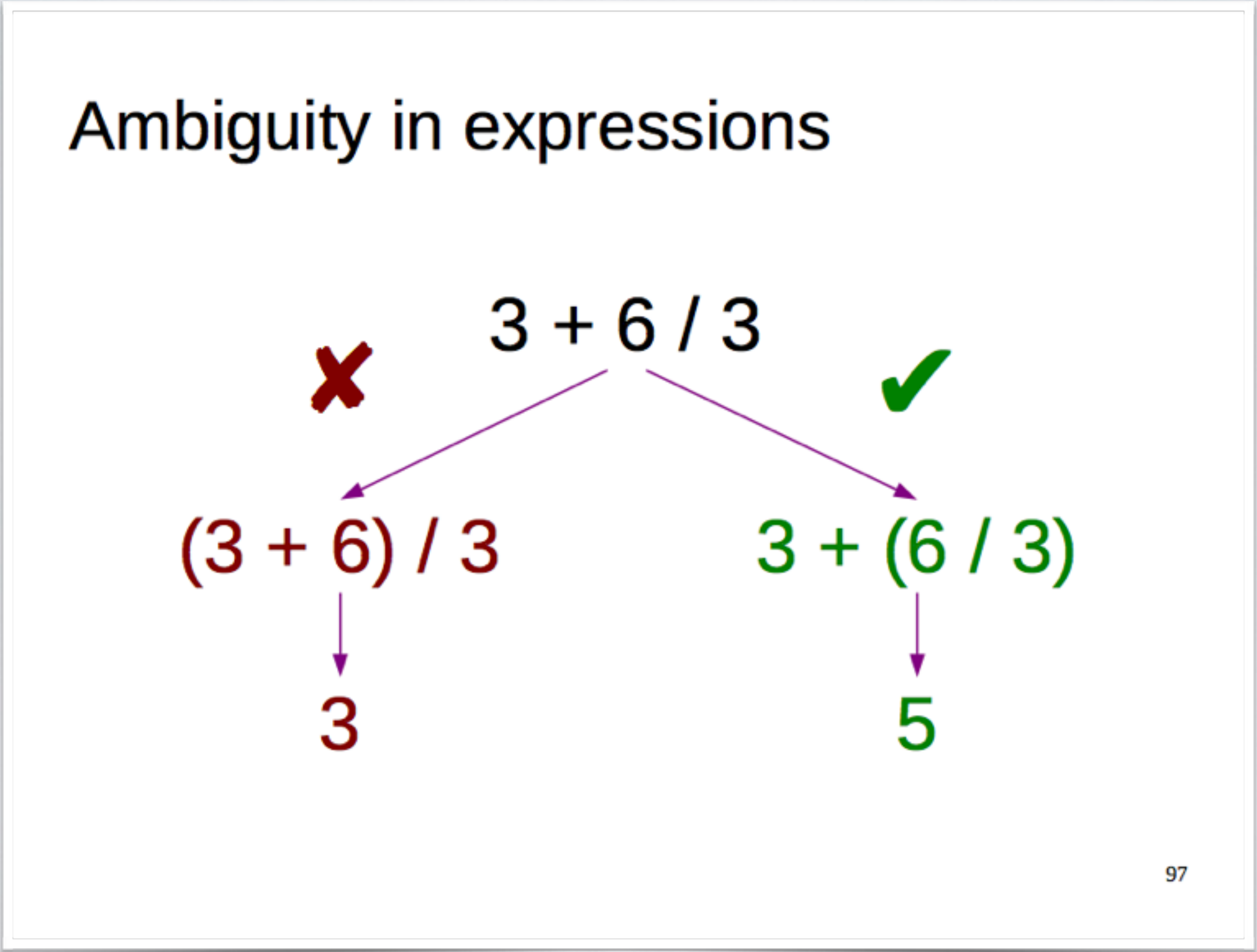
```
>>> not 6 < 7                 6 < 7→True —not→False

False



>>> not 6 > 7                 6 > 7→False—not→True

True
```

*** **not** is very tricky! ***

96

Slide taken from University of Cambridge’s Python AB Course

Python Basics

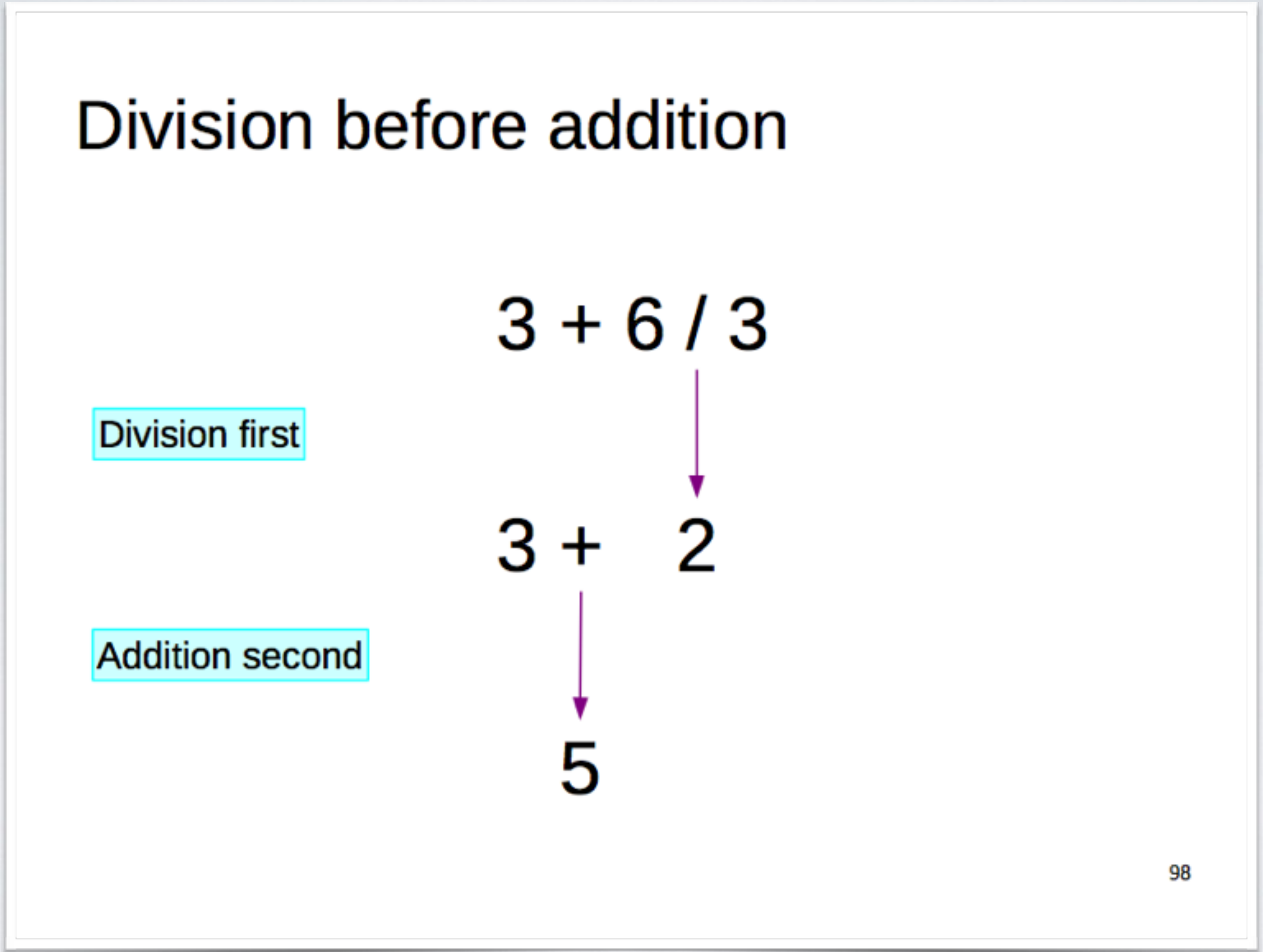


Ambiguity in expressions

$$\begin{tikzcd}
 & 3 + 6 / 3 \arrow[dl, "✘"'] \arrow[dr, "✔"] & \\
(3 + 6) / 3 \arrow[d] & & 3 + (6 / 3) \arrow[d] \\
3 & & 5
\end{tikzcd}$$

97

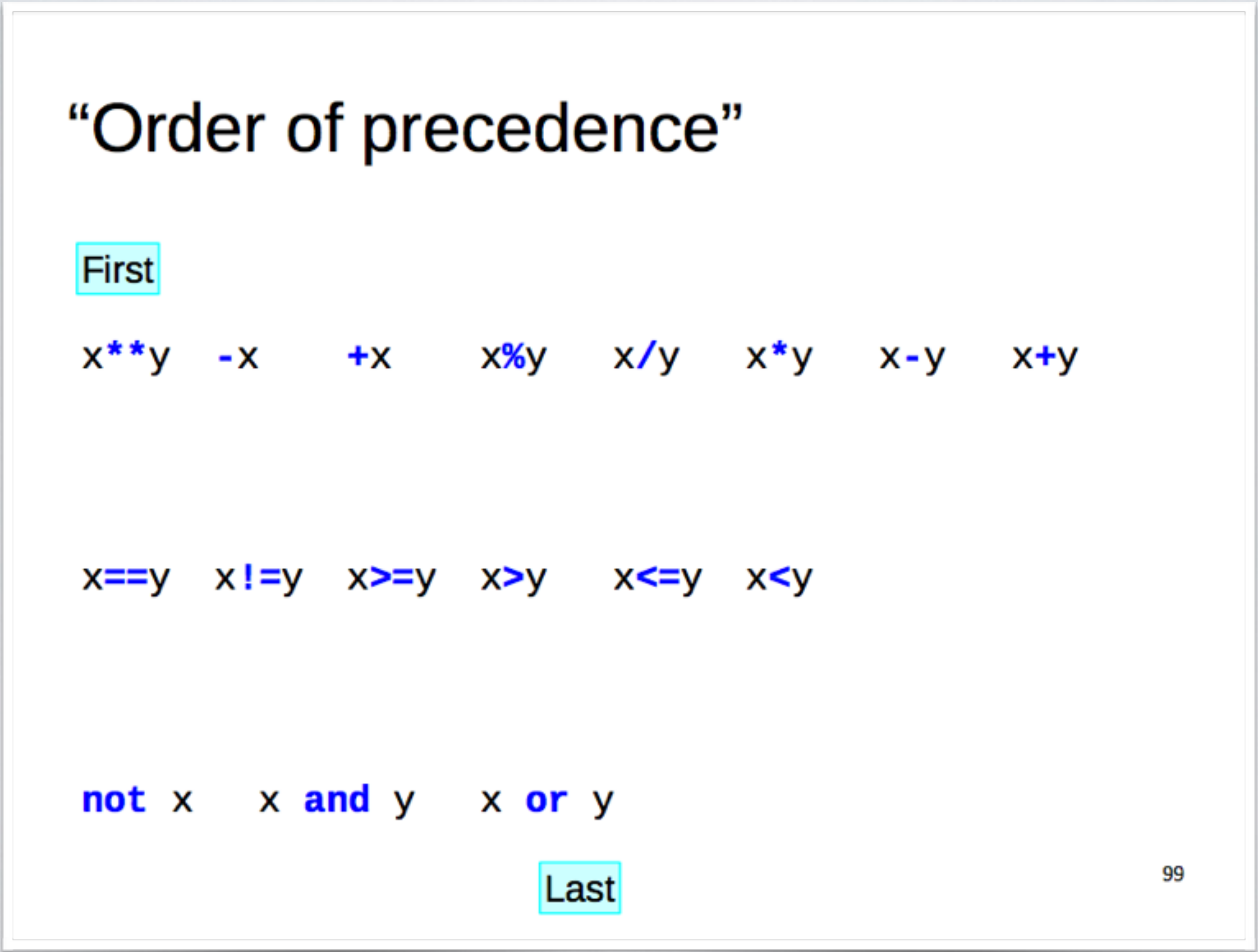
Python Basics



Division before addition

$$
\begin{array}{c}
3 + 6 / 3 \\
\downarrow \\
3 + \quad 2 \\
\downarrow \\
5
\end{array}
$$

Python Basics



“Order of precedence”

First

```
x**y   -x    +x    x%y   x/y   x*y   x-y   x+y
```

```
x==y  x!=y  x>=y  x>y   x<=y  x<y
```

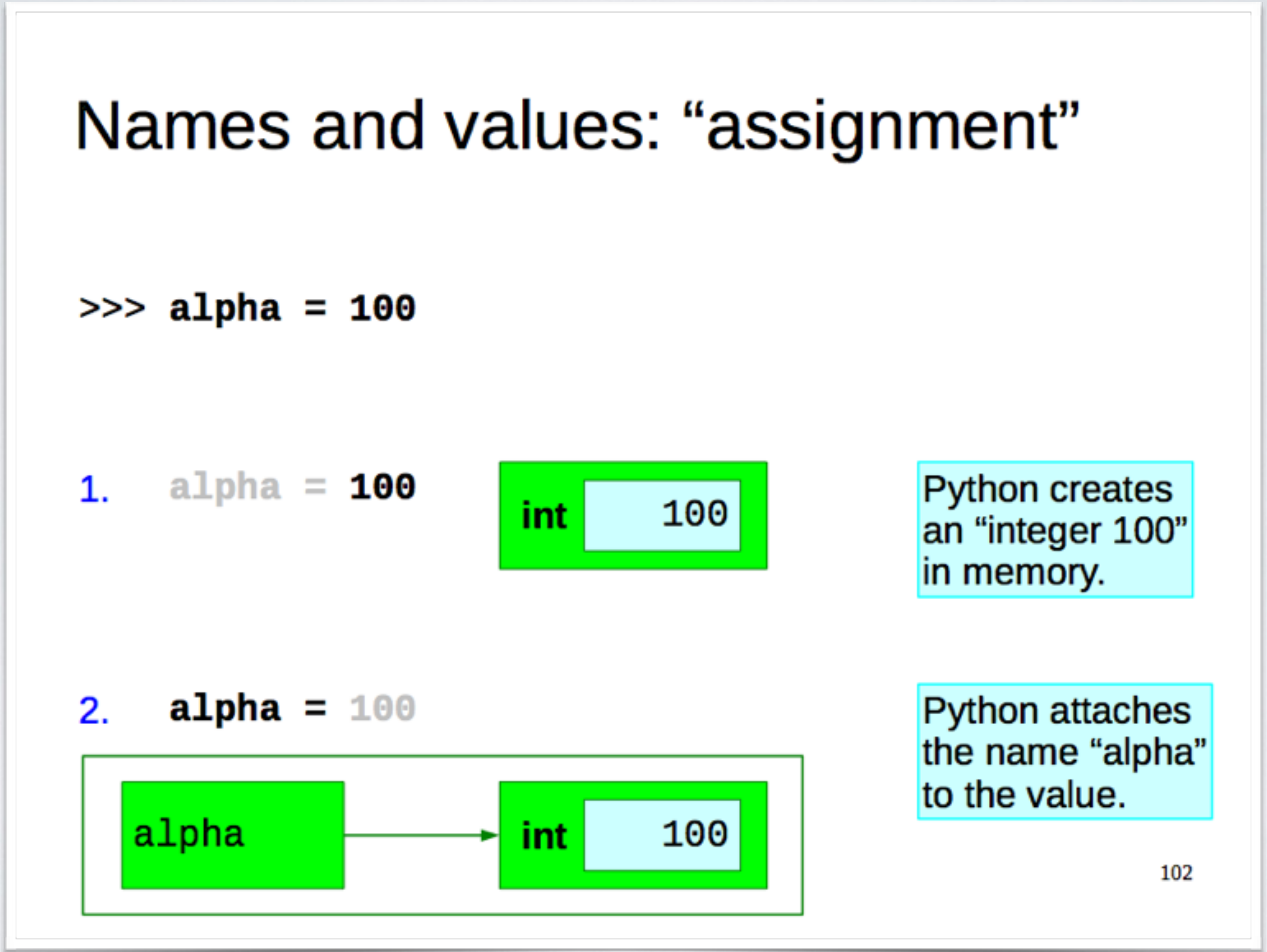
```
not x   x and y   x or y
```

Last

99

Slide taken from University of Cambridge’s Python AB Course

Python Basics



Names and values: “assignment”

```
>>> alpha = 100
```

1. `alpha = 100`

Python creates an “integer 100” in memory.

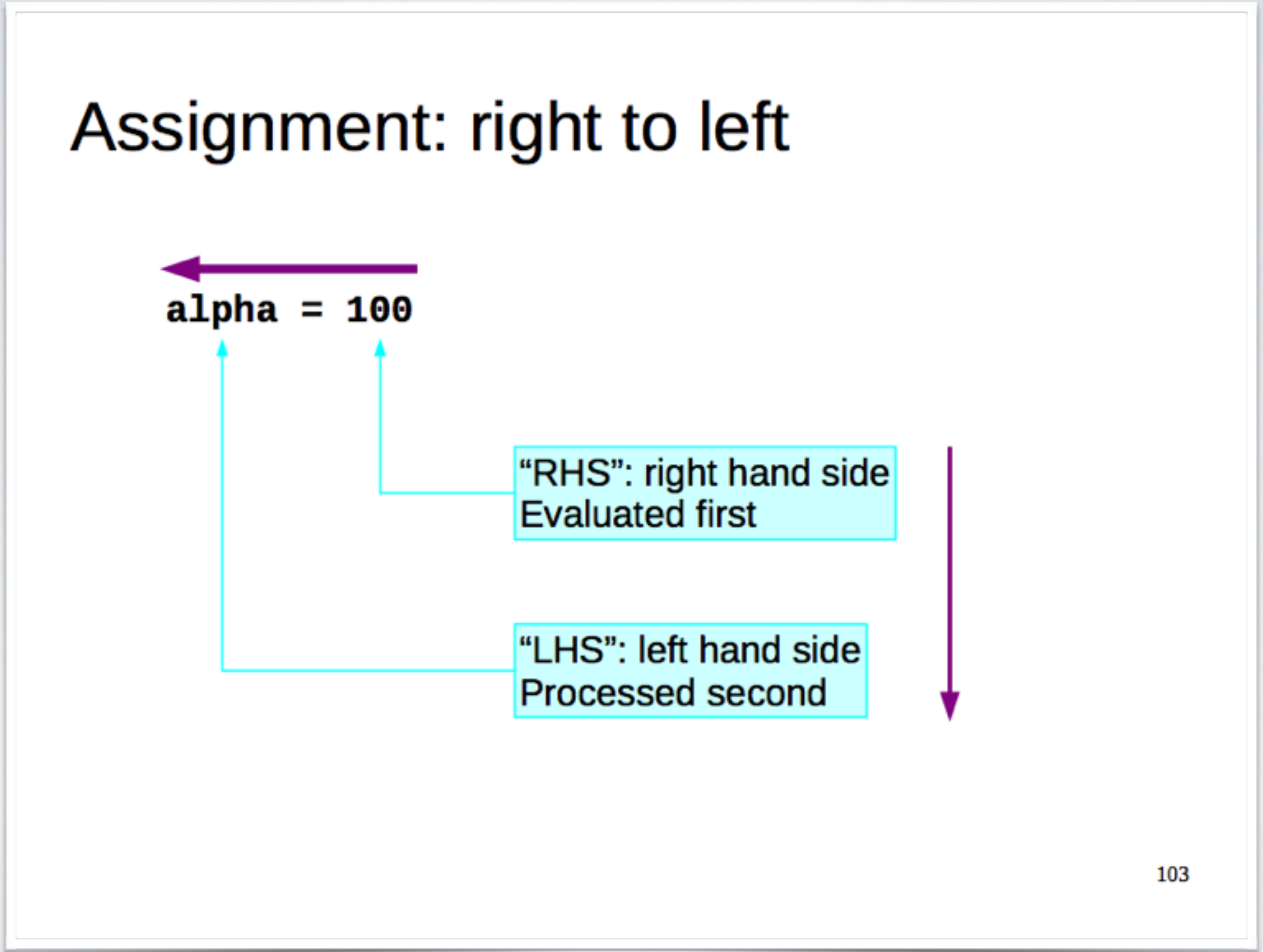
2. `alpha = 100`

Python attaches the name “alpha” to the value.

int
100
alpha
int
100
102

Slide taken from University of Cambridge’s Python AB Course

Python Basics

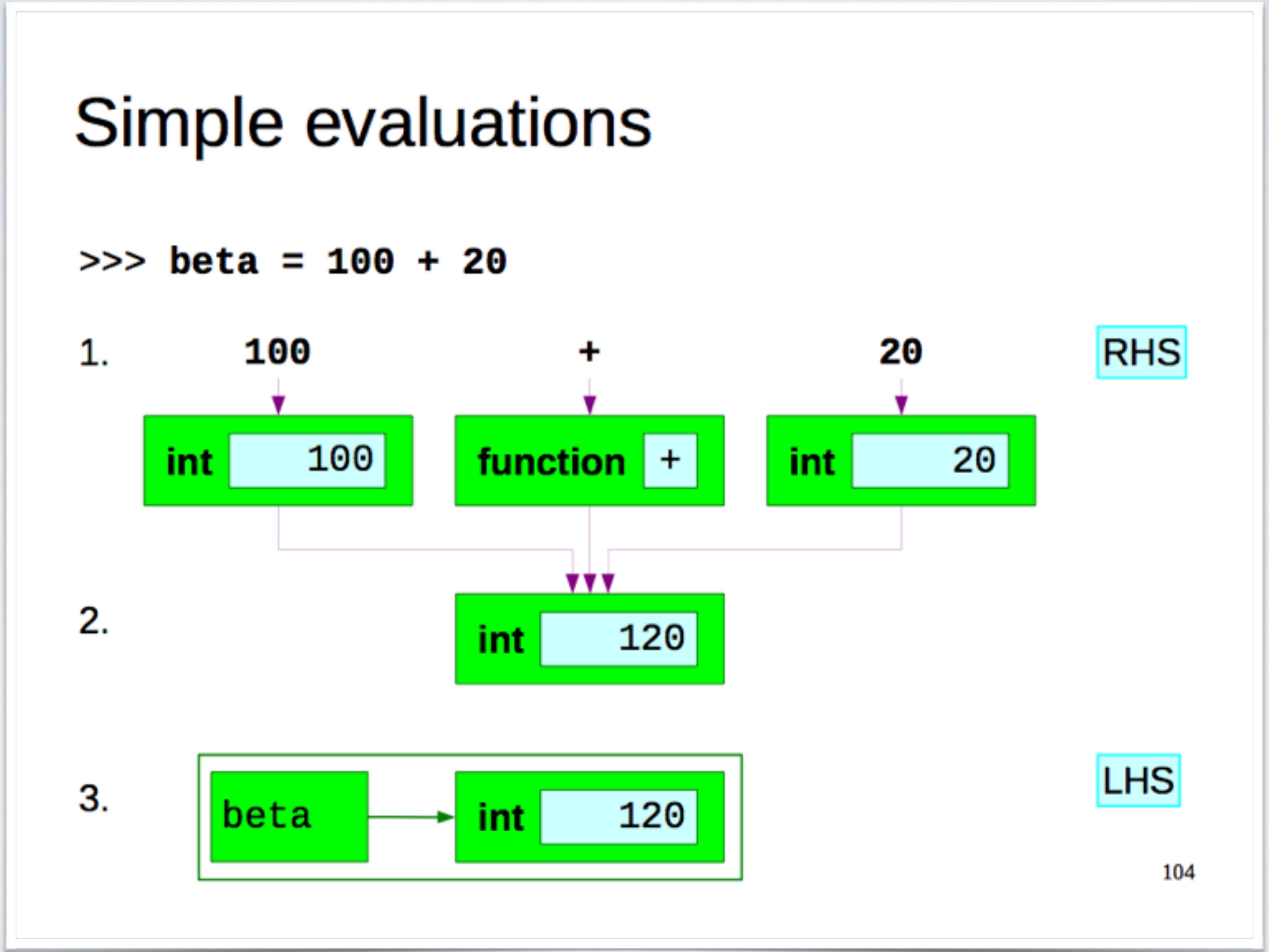


Assignment: right to left

```
alpha = 100
```

Slide taken from University of Cambridge's Python AB Course

Python Basics

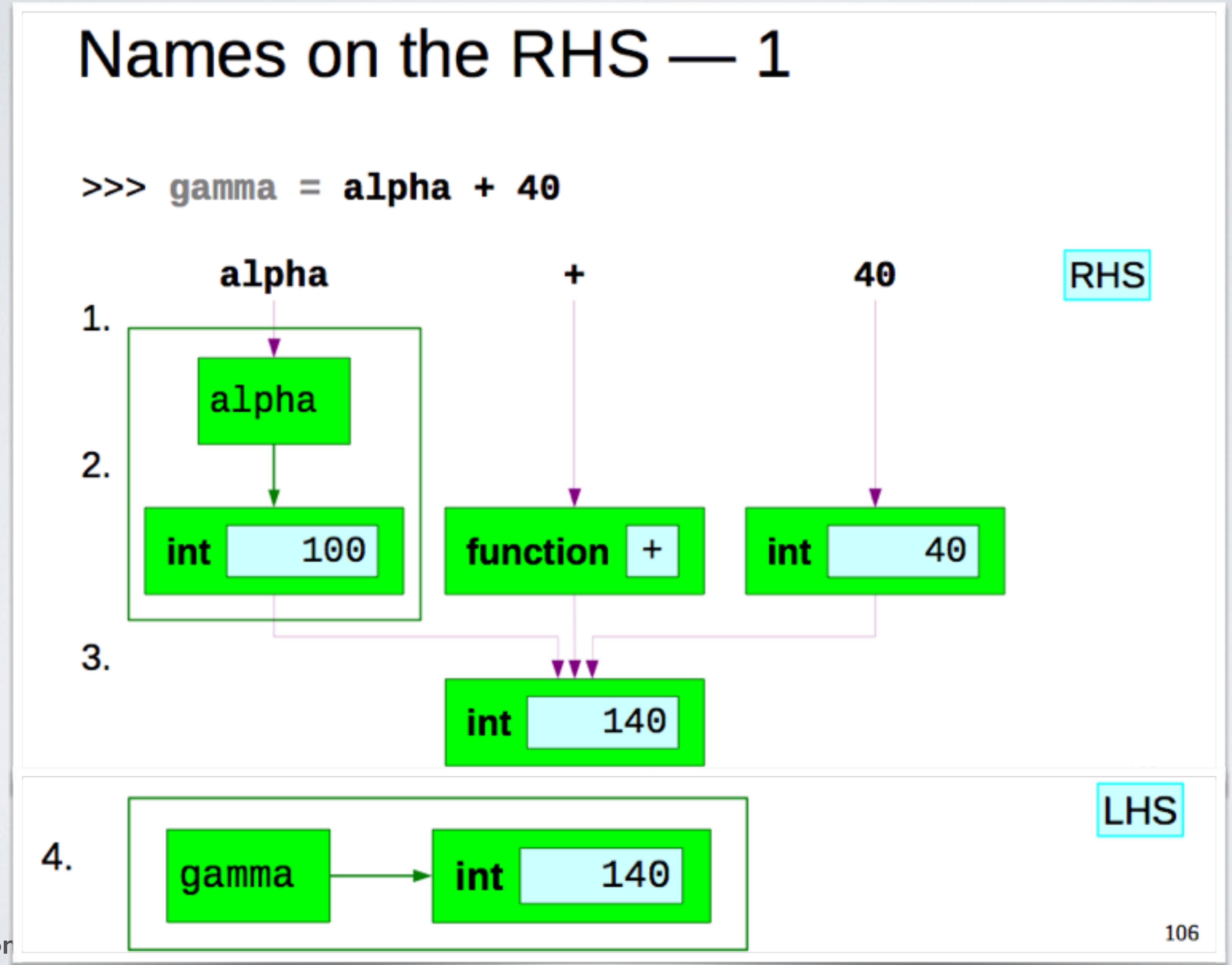


Simple evaluations

```
>>> beta = 100 + 20
```

Slide taken from University of Cambridge's Python AB Course

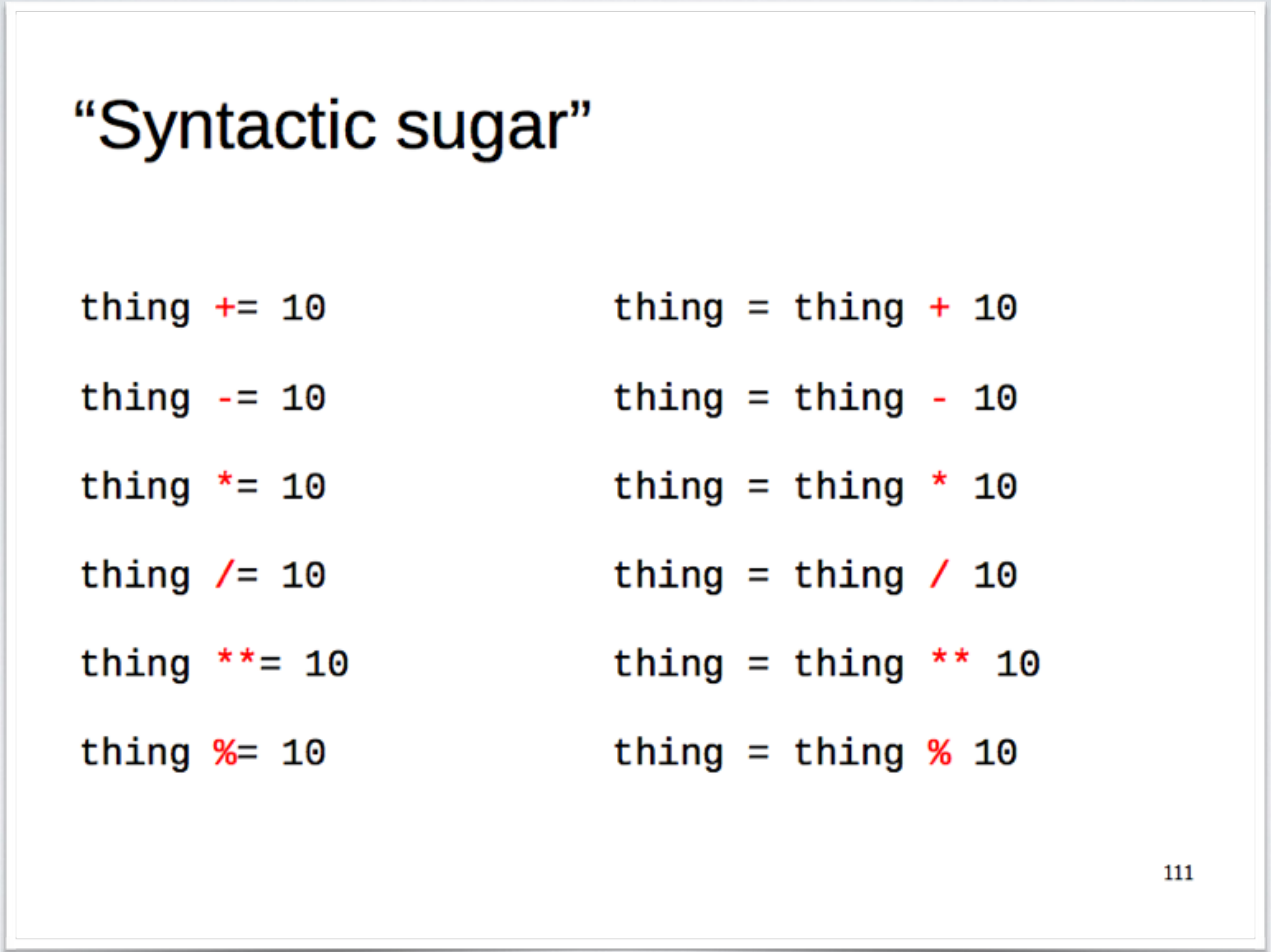
Python Basics



Names on the RHS — 1

```
>>> gamma = alpha + 40
```

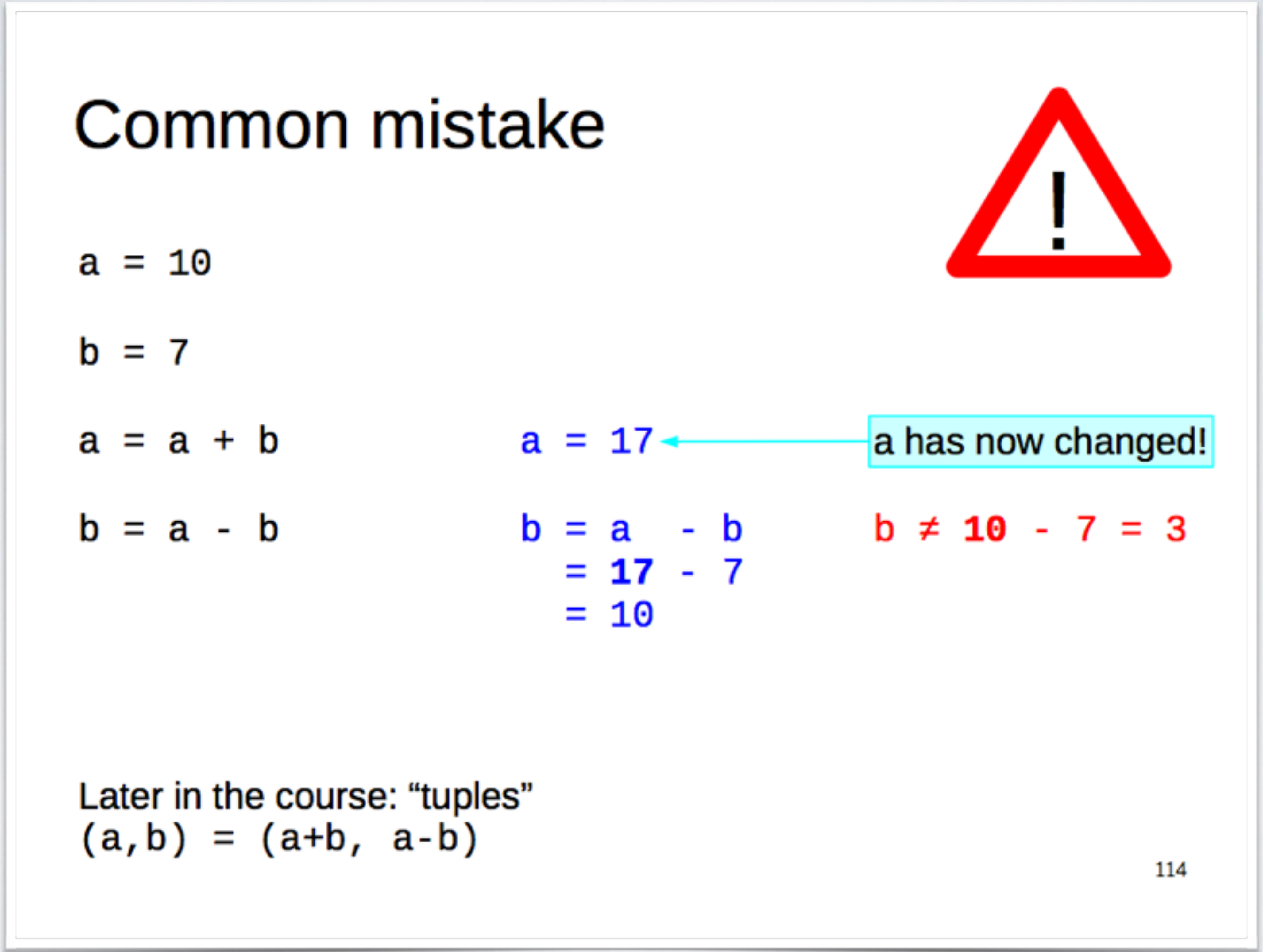
Python Basics


“Syntactic sugar”

```
thing += 10          thing = thing + 10

thing -= 10          thing = thing - 10

thing *= 10          thing = thing * 10

thing /= 10          thing = thing / 10

thing **= 10         thing = thing ** 10

thing %= 10          thing = thing % 10
```

111

Python Basics



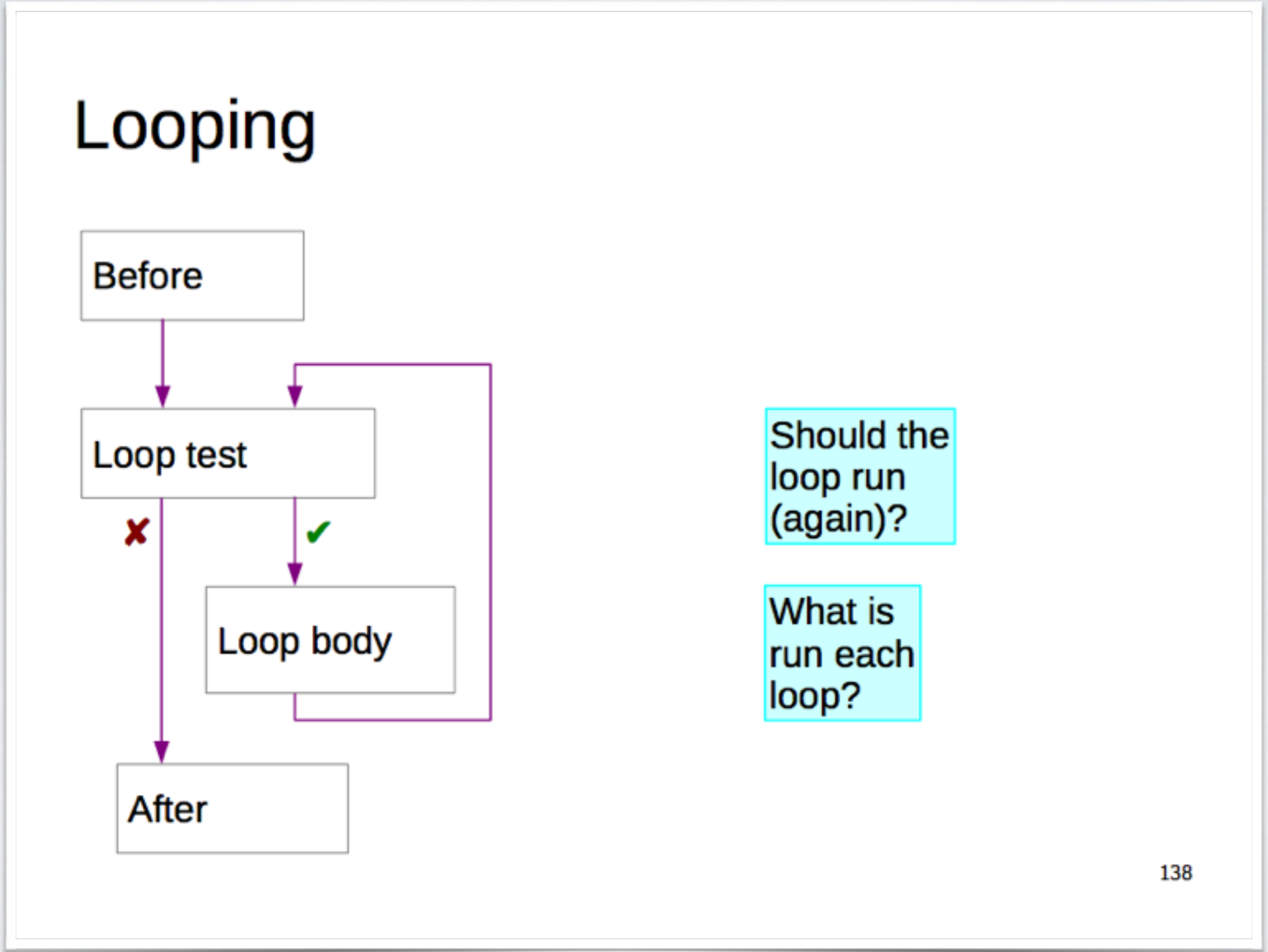
Common mistake

```
a = 10

b = 7

a = a + b           a = 17 ◄────── a has now changed!

b = a - b           b = a  - b      b ≠ 10 - 7 = 3
                      = 17 - 7
                      = 10
```

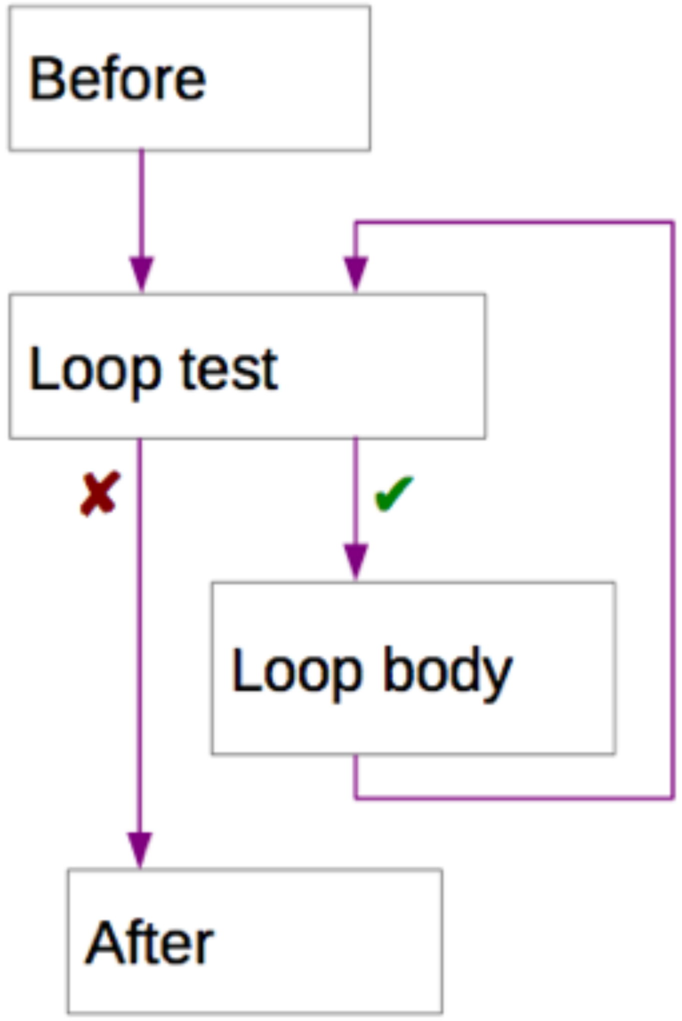
Later in the course: “tuples”
`(a,b) = (a+b, a-b)`

114

Python Basics



Looping



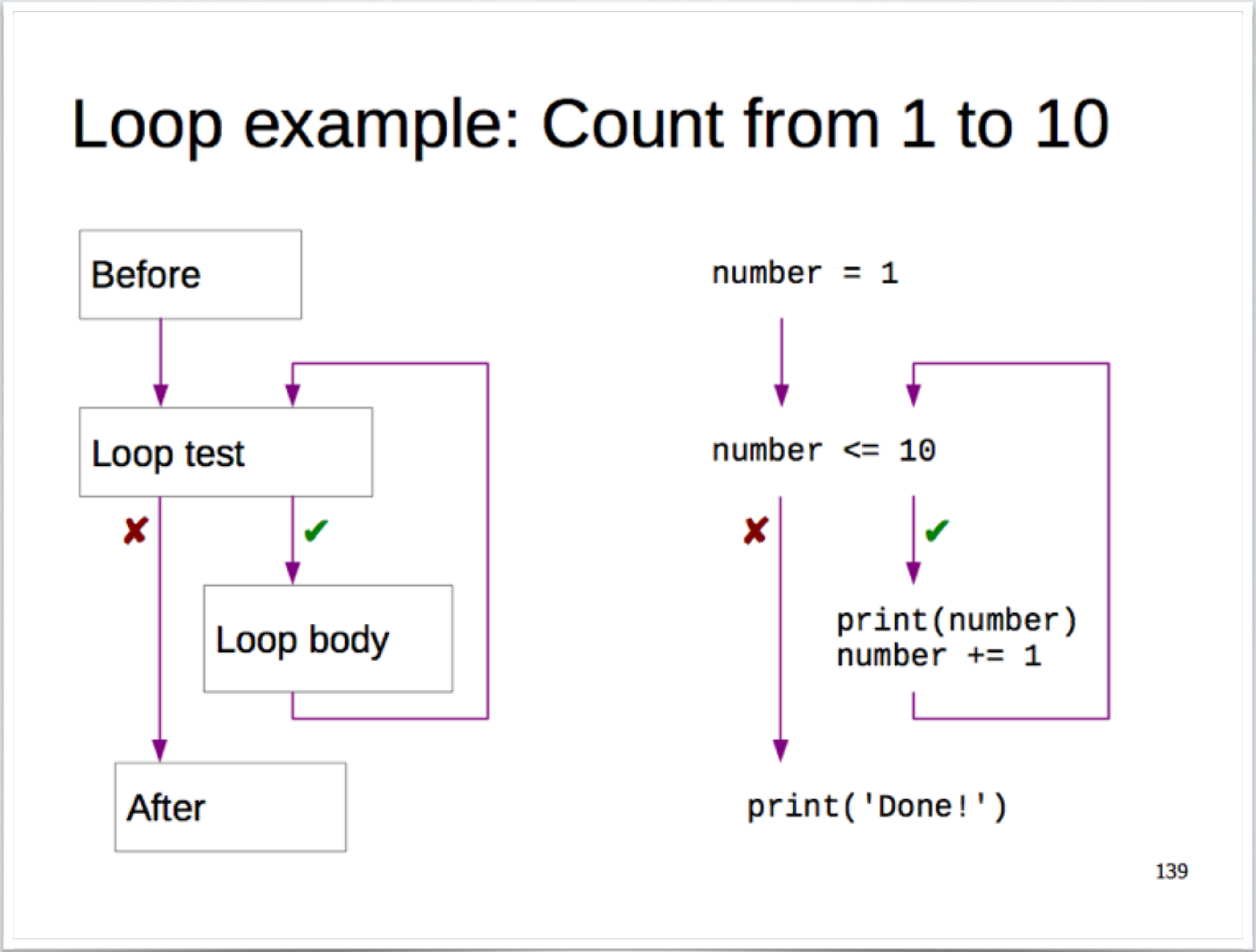
Should the loop run (again)?

What is run each loop?

138

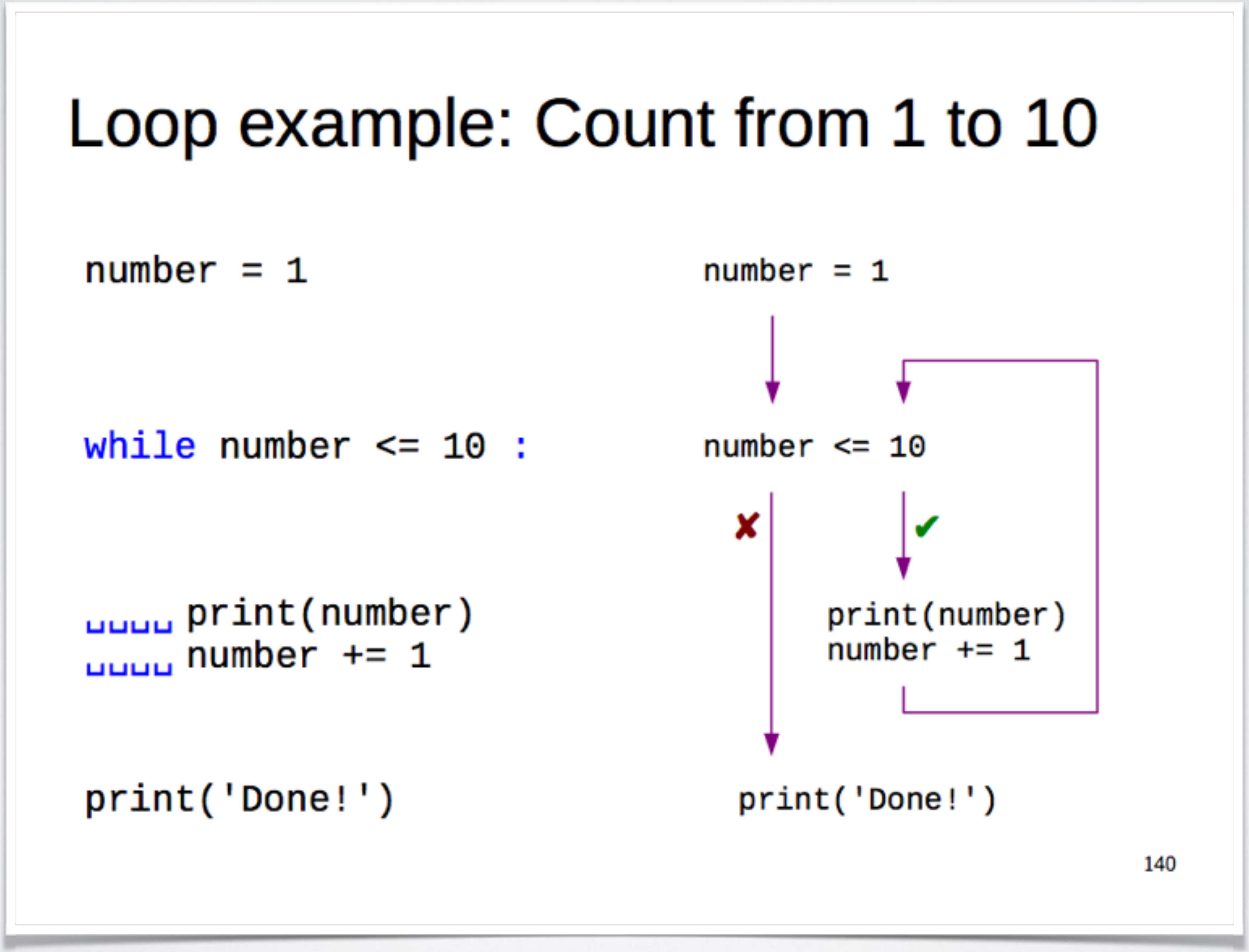
Slide taken from University of Cambridge's Python AB Course

Python Basics



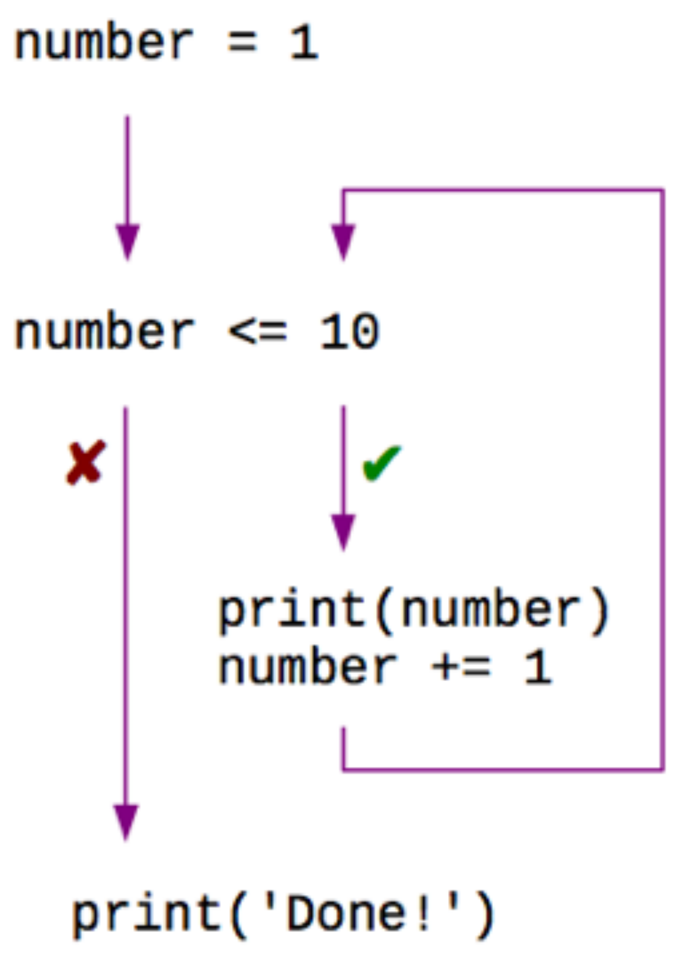
Loop example: Count from 1 to 10

Python Basics



Loop example: Count from 1 to 10

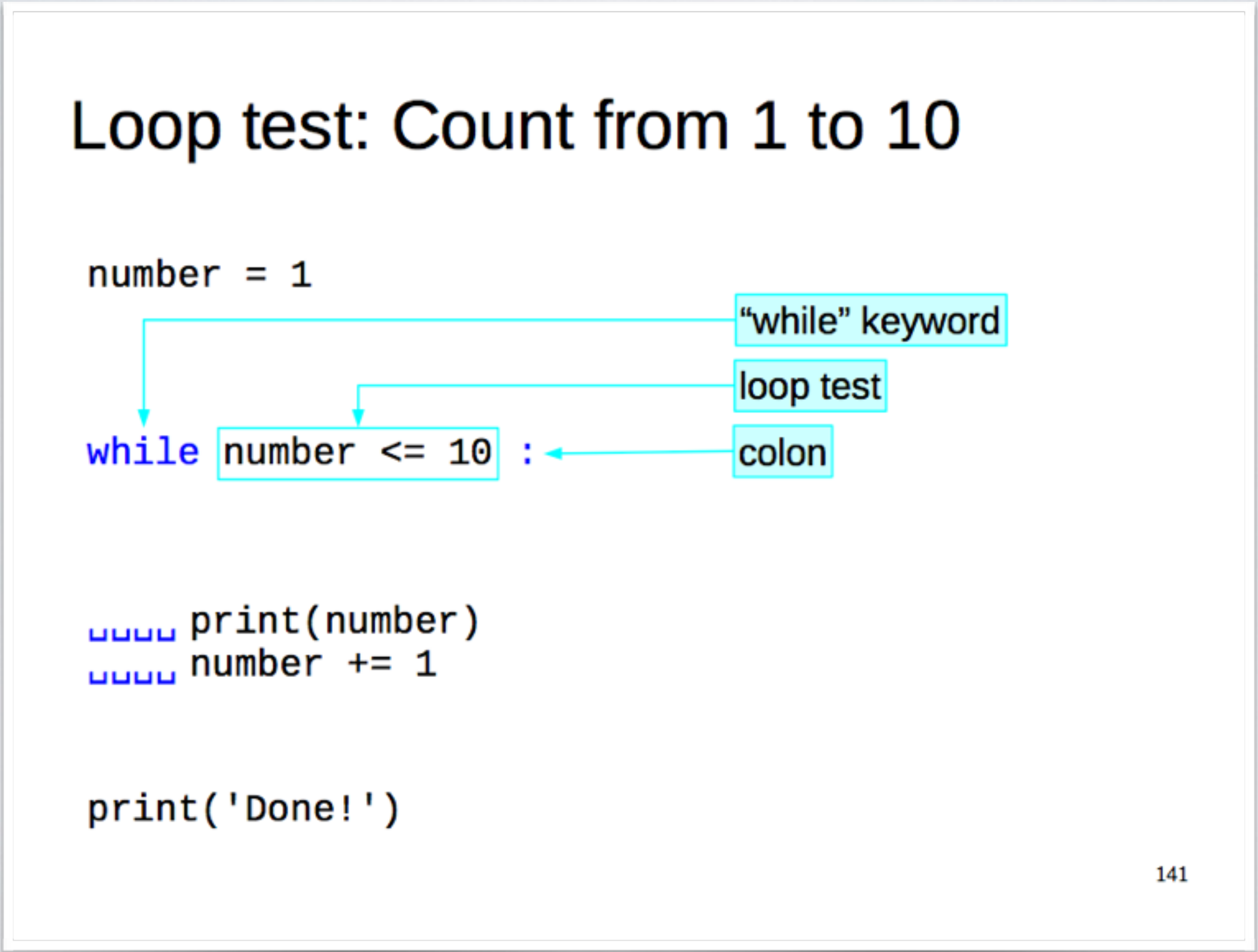
```
number = 1

while number <= 10 :

␣␣␣␣print(number)
␣␣␣␣number += 1

print('Done!')
```



140

Slide taken from University of Cambridge's Python AB Course
Copyright © 2014 by Stephen Makonin

Python Basics

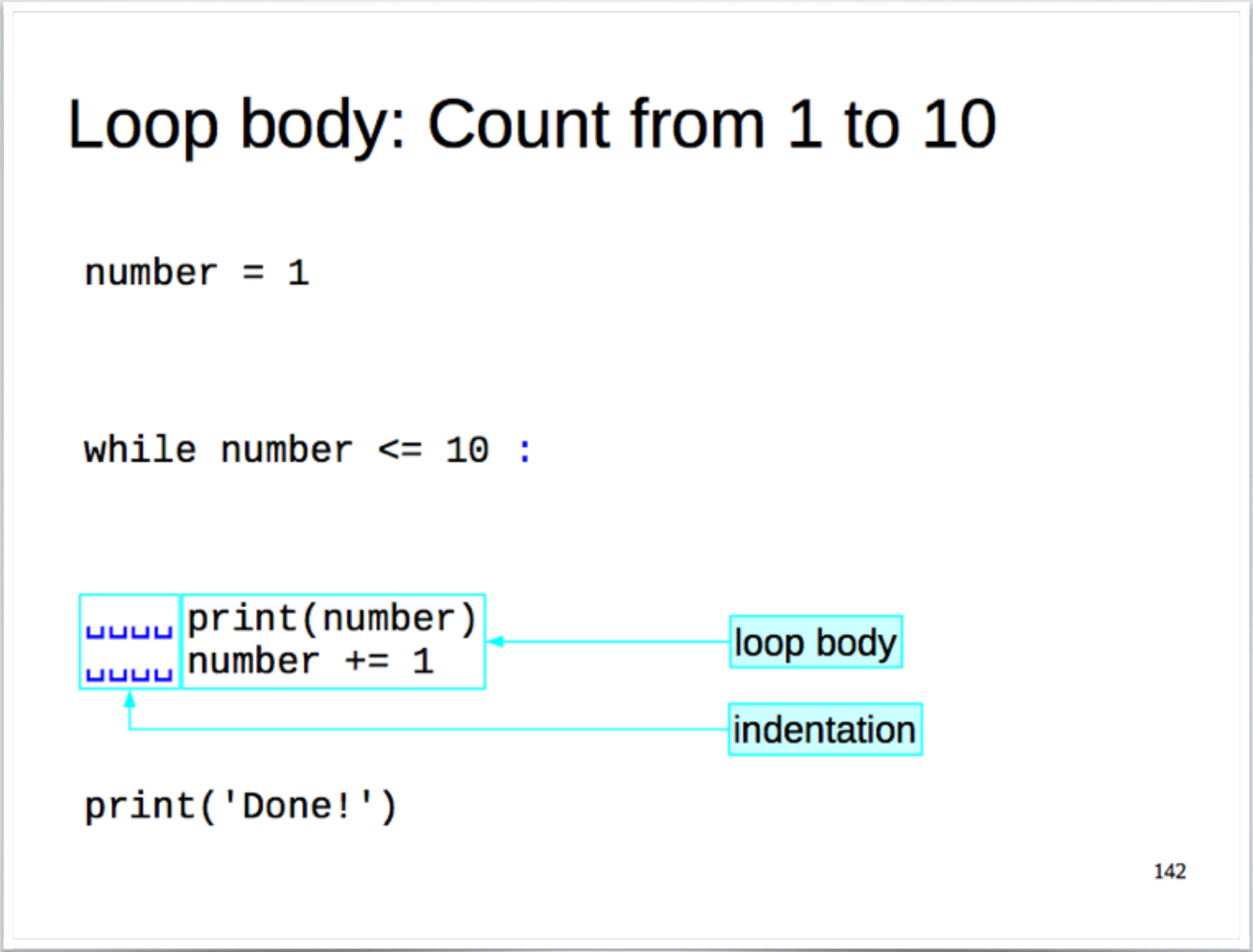


Loop test: Count from 1 to 10

```
number = 1
while number <= 10 :
␣␣␣␣print(number)
␣␣␣␣number += 1
print('Done!')
```

“while” keyword
loop test
colon
141

Python Basics



Loop body: Count from 1 to 10

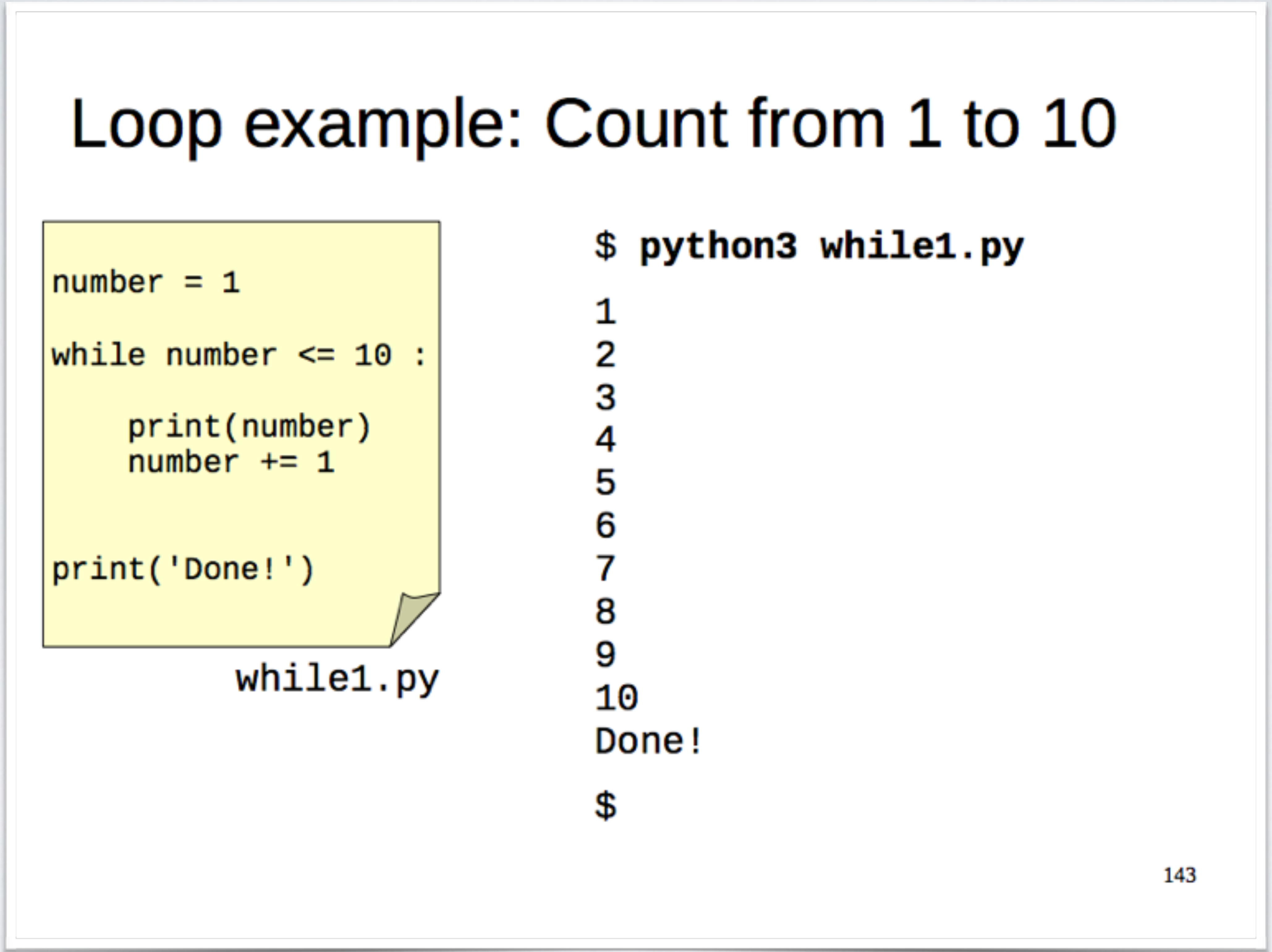
```
number = 1

while number <= 10 :

␣␣␣␣print(number)
␣␣␣␣number += 1

print('Done!')
```

loop body
indentation

142

Slide taken from University of Cambridge's Python AB Course

Python Basics



Loop example: Count from 1 to 10

```
number = 1

while number <= 10 :

    print(number)
    number += 1


print('Done!')
```

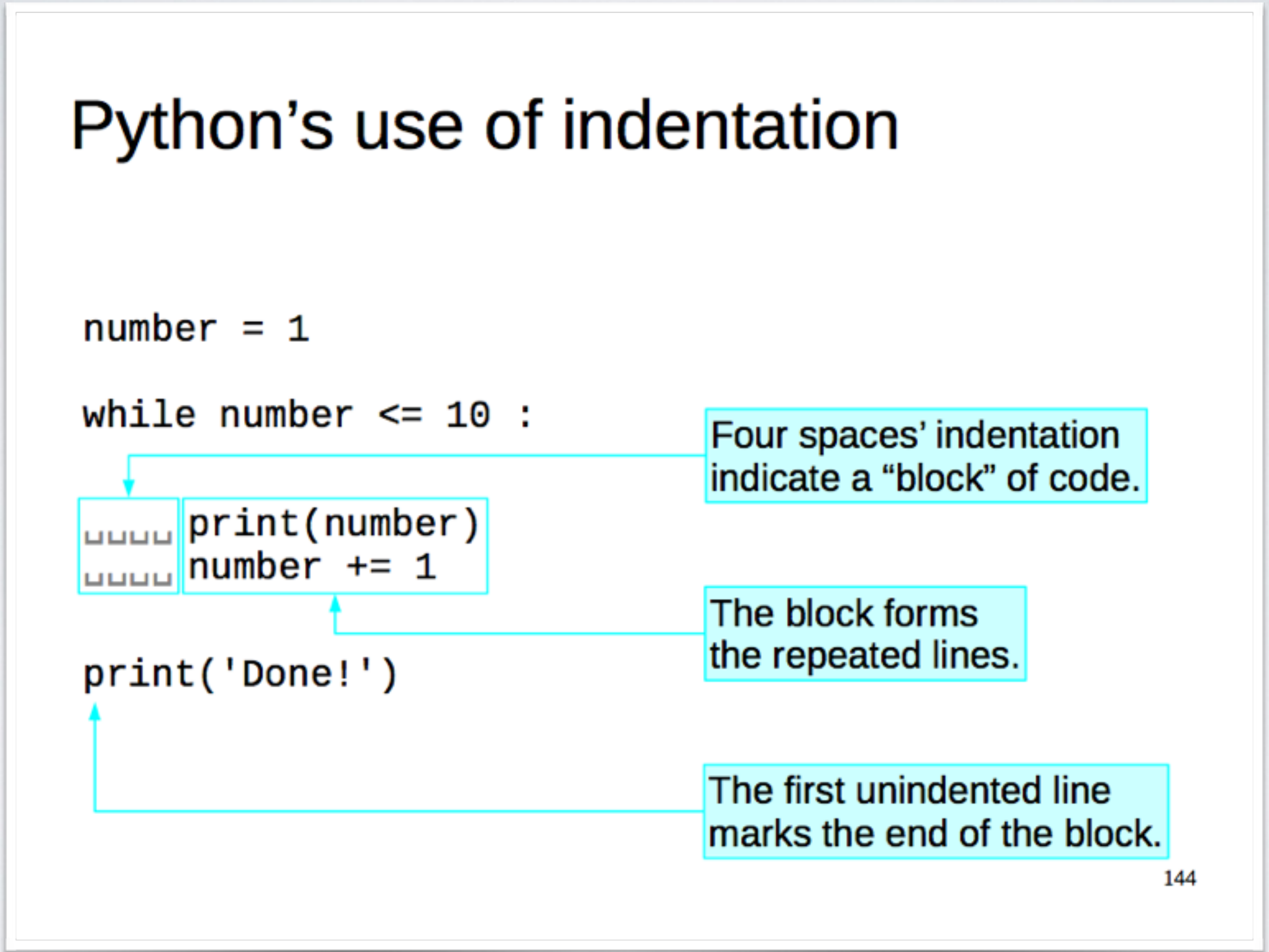
while1.py

```
$ python3 while1.py
1
2
3
4
5
6
7
8
9
10
Done!
$
```

143

Slide taken from University of Cambridge's Python AB Course

Python Basics



Python's use of indentation

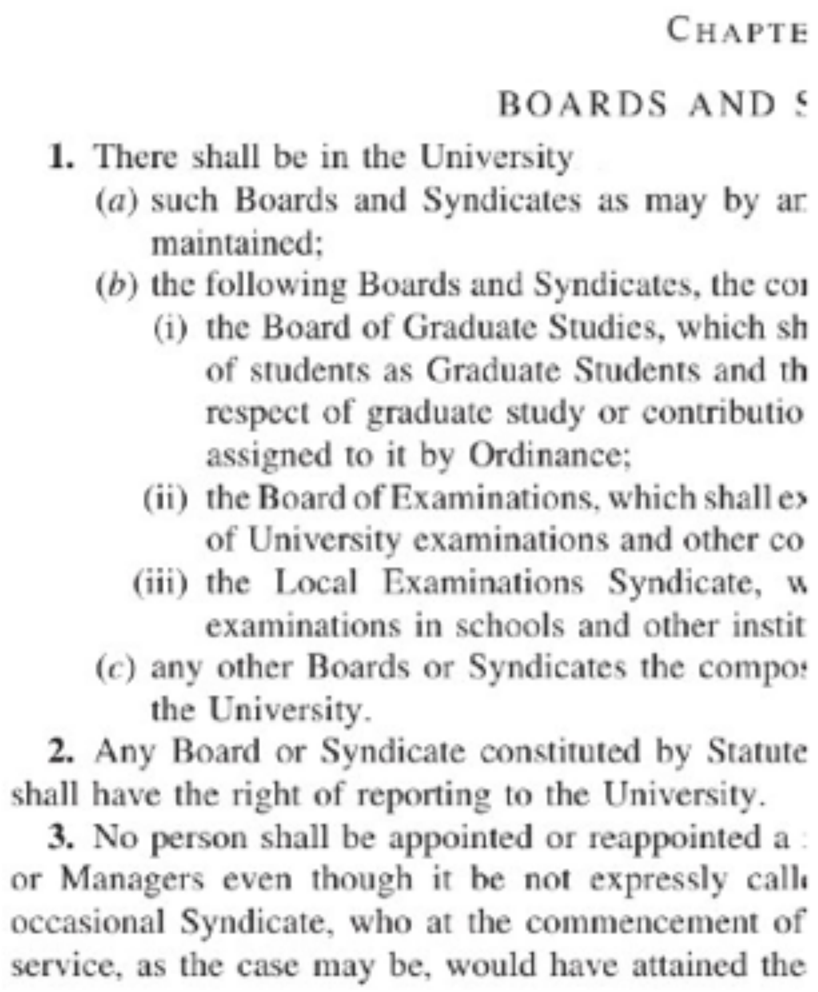
```
number = 1

while number <= 10 :
␣␣␣␣print(number)
␣␣␣␣number += 1

print('Done!')
```

Four spaces' indentation indicate a "block" of code.
The block forms the repeated lines.
The first unindented line marks the end of the block.
144

Slide taken from University of Cambridge's Python AB Course

Python Basics

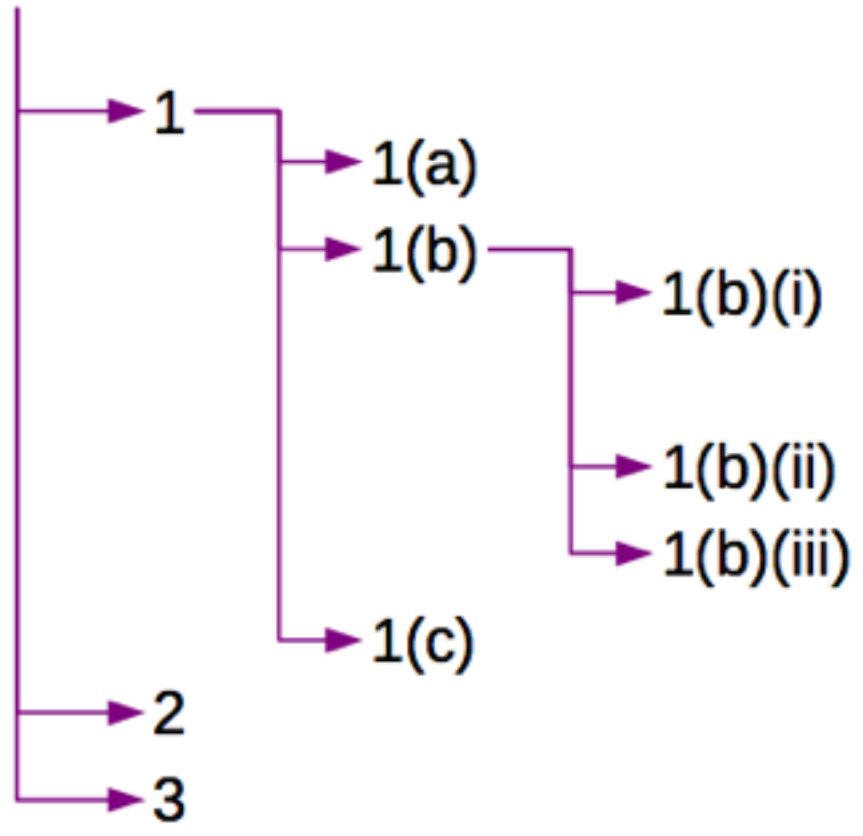
c.f. “legalese”



CHAPTE

BOARDS AND S

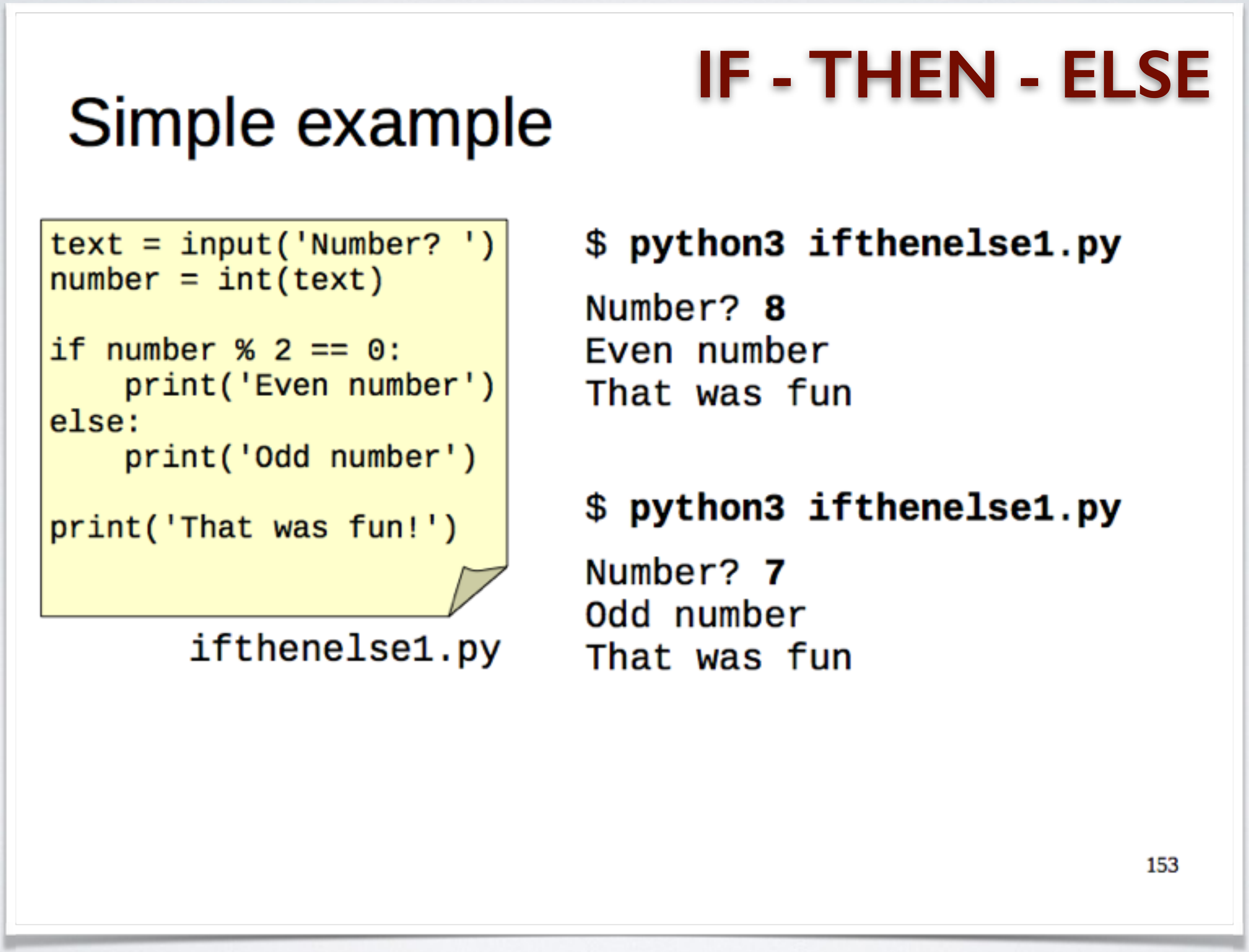
1. There shall be in the University
(*a*) such Boards and Syndicates as may by ar
maintained;
(*b*) the following Boards and Syndicates, the coı
(i) the Board of Graduate Studies, which sh
of students as Graduate Students and th
respect of graduate study or contributio
assigned to it by Ordinance;
(ii) the Board of Examinations, which shall e›
of University examinations and other co
(iii) the Local Examinations Syndicate, w
examinations in schools and other instit
(*c*) any other Boards or Syndicates the compo:
the University.
2. Any Board or Syndicate constituted by Statute
shall have the right of reporting to the University.
3. No person shall be appointed or reappointed a :
or Managers even though it be not expressly call‹
occasional Syndicate, who at the commencement of
service, as the case may be, would have attained the



145

Slide taken from University of Cambridge’s Python AB Course
Copyright © 2014 by Stephen Makonin

Python Basics



IF - THEN - ELSE

Simple example

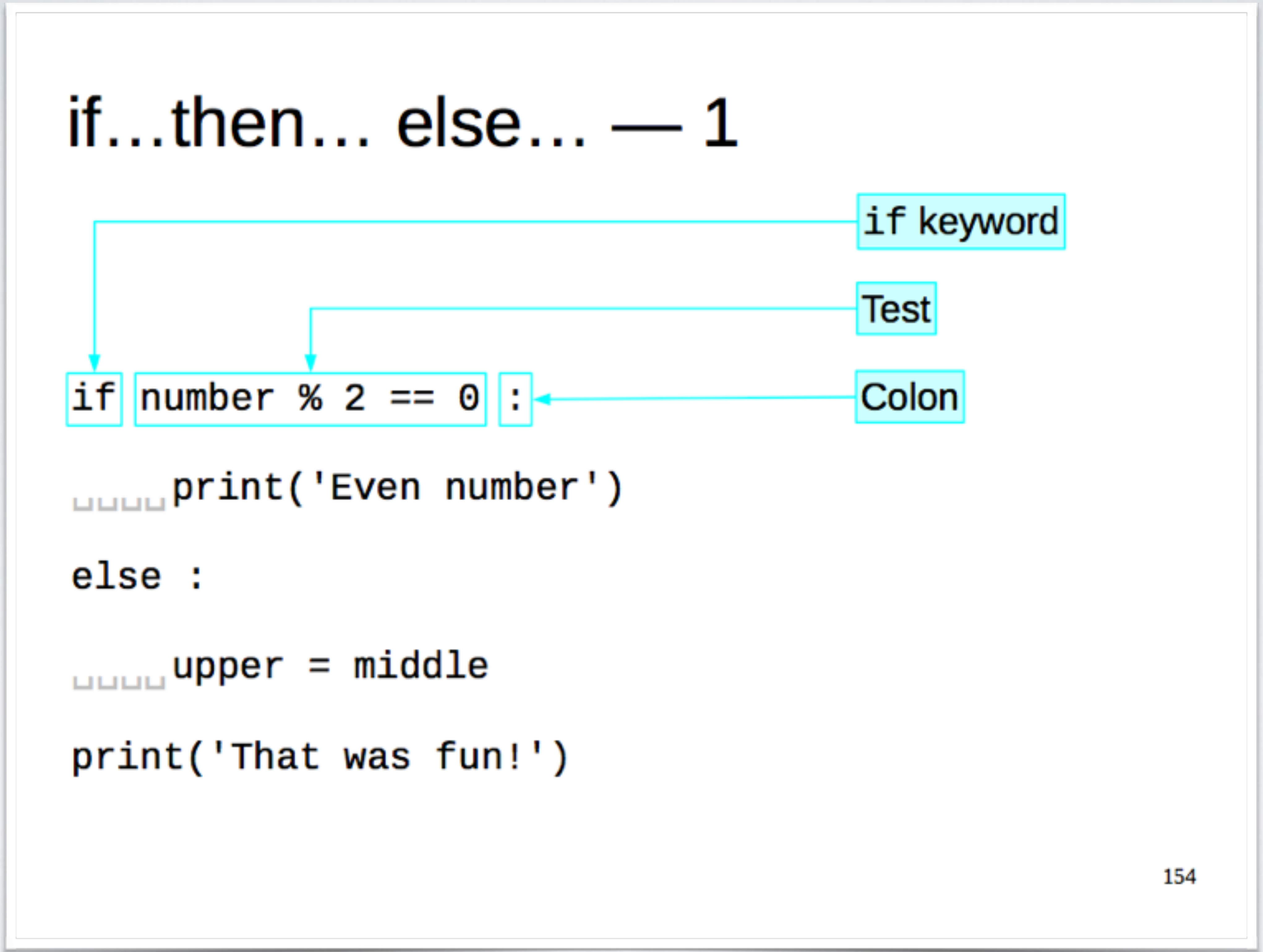
```
text = input('Number? ')
number = int(text)

if number % 2 == 0:
    print('Even number')
else:
    print('Odd number')

print('That was fun!')
```

ifthenelse1.py

```
$ python3 ifthenelse1.py

Number? 8
Even number
That was fun


$ python3 ifthenelse1.py

Number? 7
Odd number
That was fun
```

153

Python Basics

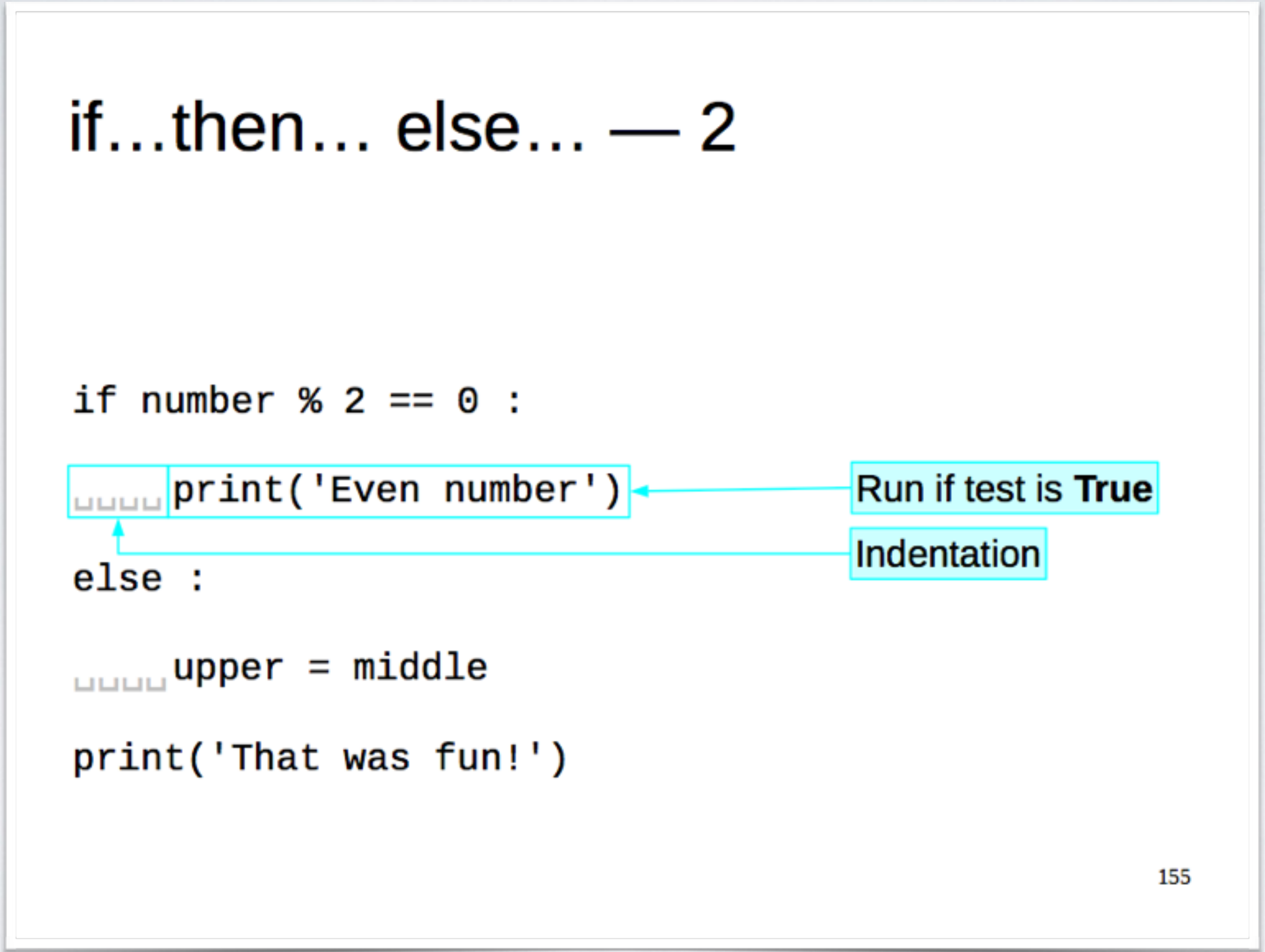


if…then… else… — 1

if keyword
Test
Colon

```
if number % 2 == 0 :
␣␣␣␣print('Even number')
else :
␣␣␣␣upper = middle
print('That was fun!')
```

Python Basics



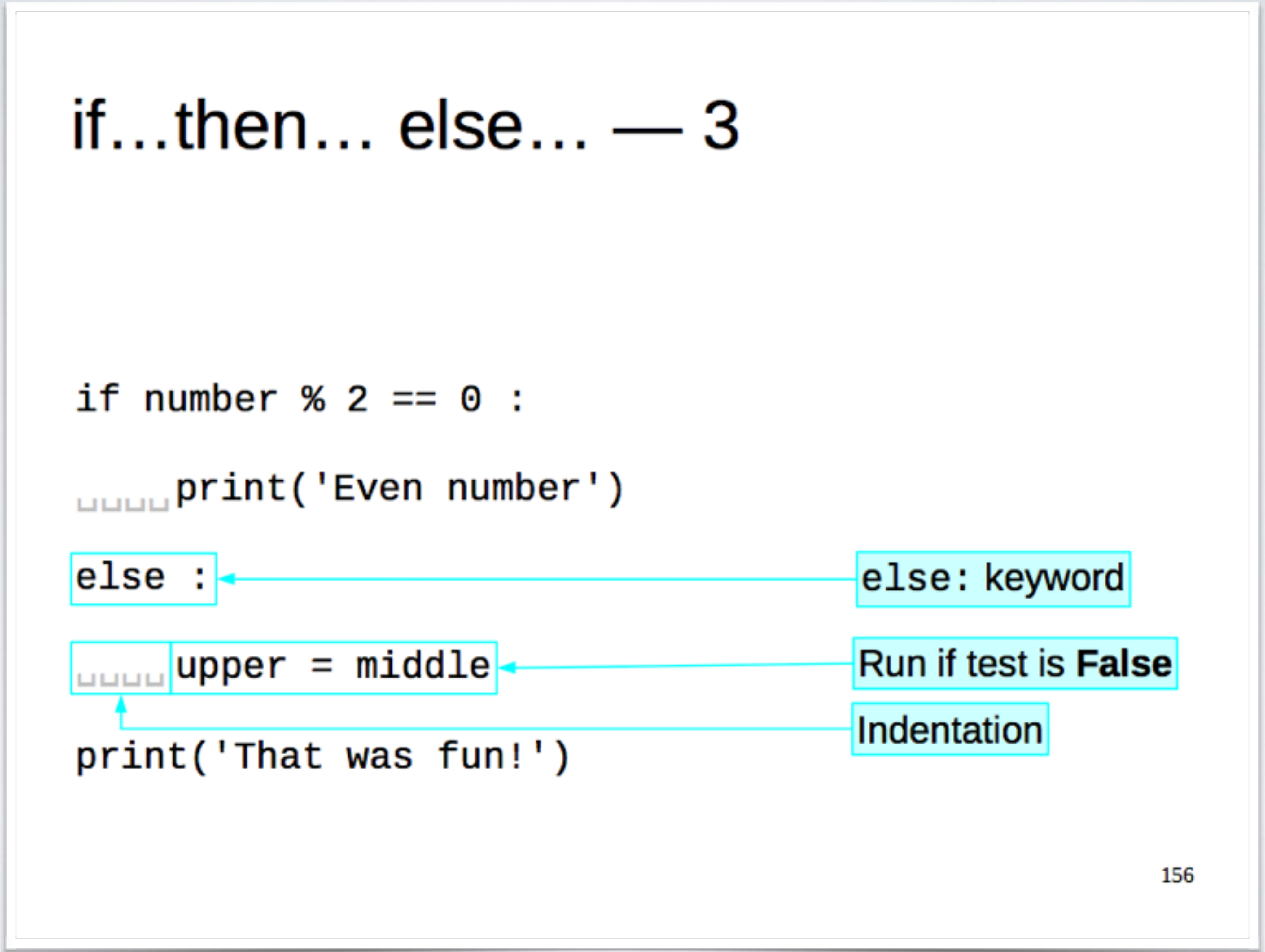
if…then… else… — 2

```
if number % 2 == 0 :
␣␣␣␣print('Even number')
else :
␣␣␣␣upper = middle
print('That was fun!')
```

Run if test is **True**
Indentation
155

Slide taken from University of Cambridge's Python AB Course

Python Basics



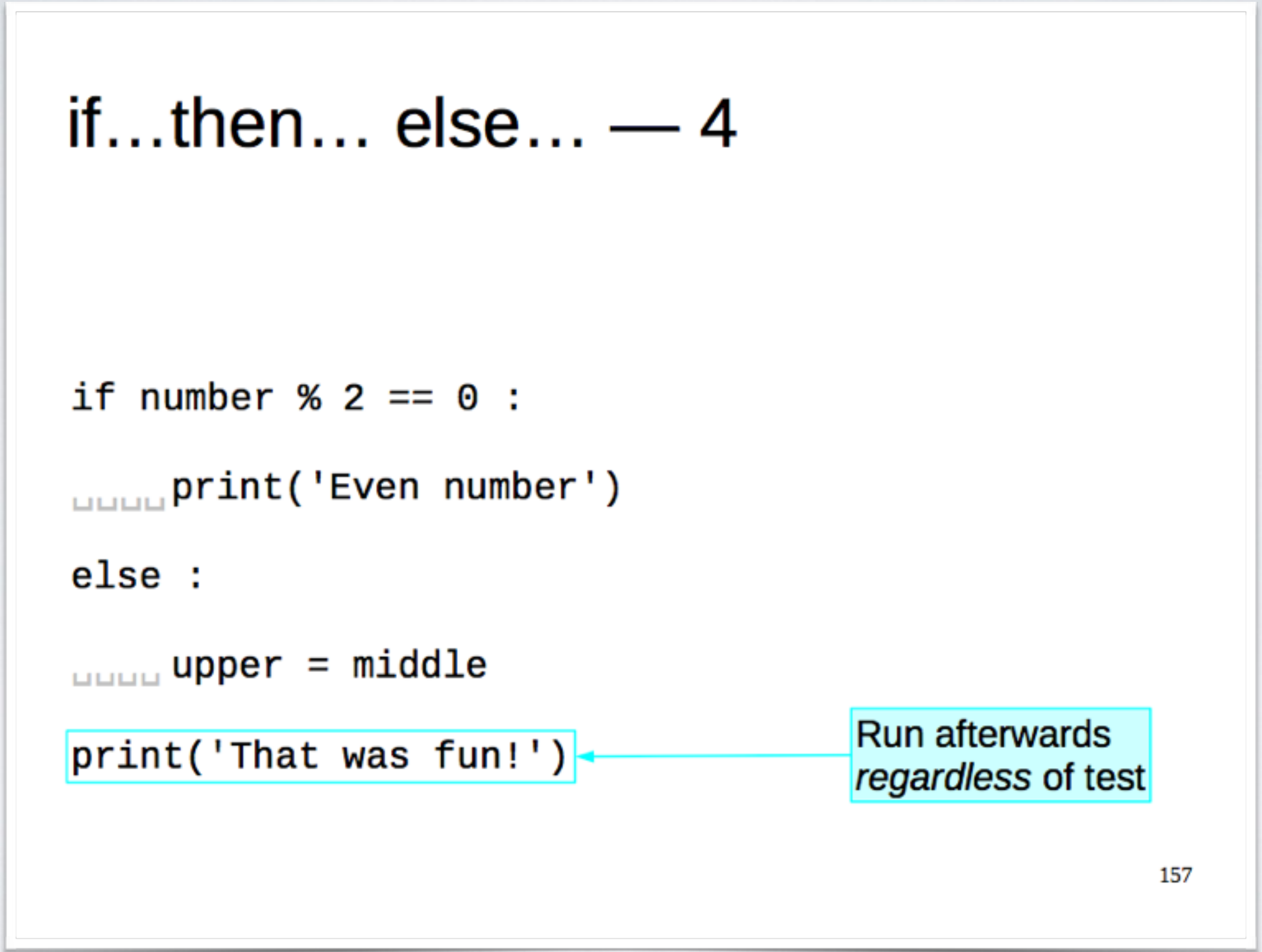
if…then… else… — 3

```
if number % 2 == 0 :
␣␣␣␣print('Even number')
else :
␣␣␣␣upper = middle
print('That was fun!')
```

156

Slide taken from University of Cambridge's Python AB Course

Python Basics



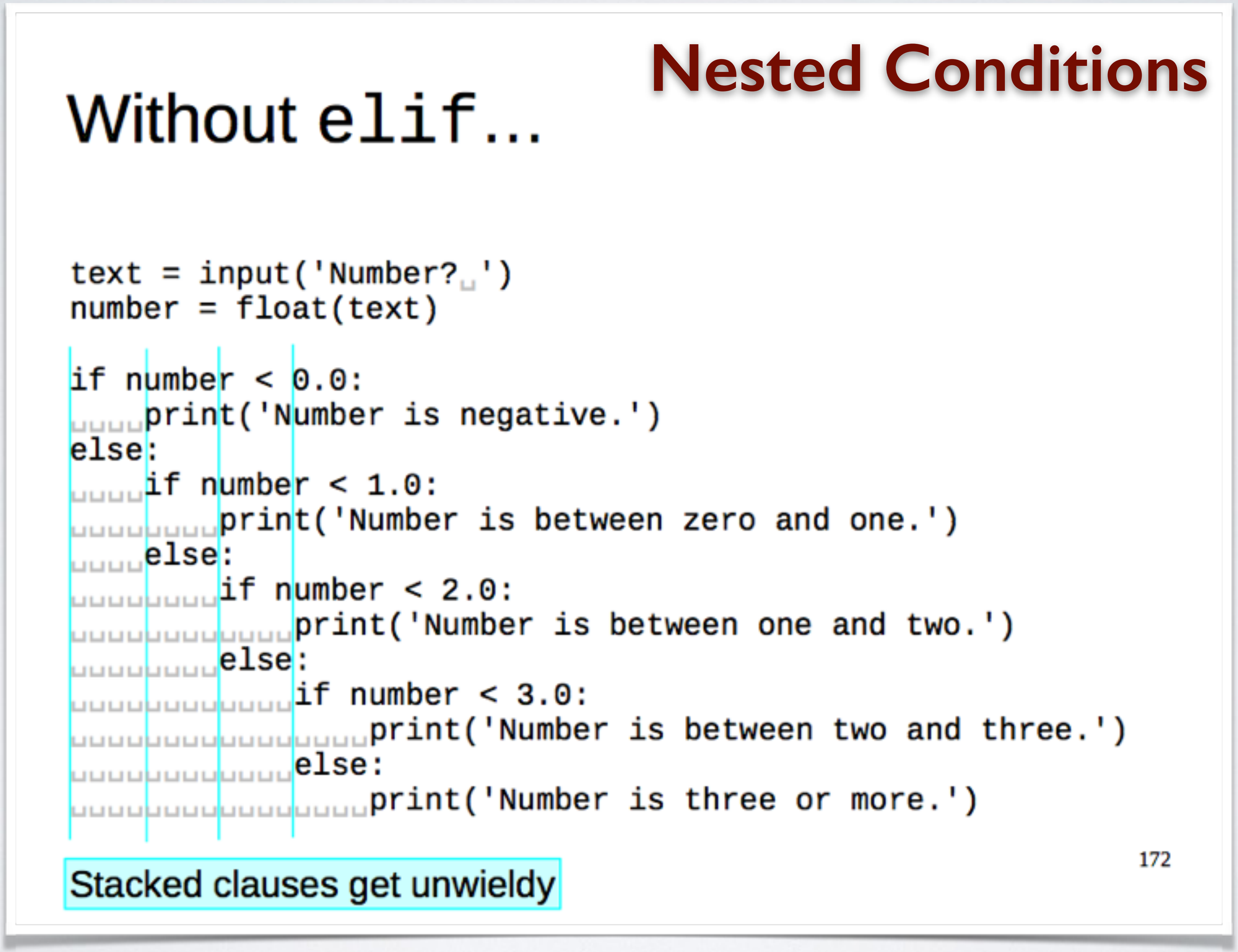
if…then… else… — 4

```
if number % 2 == 0 :
␣␣␣␣print('Even number')
else :
␣␣␣␣upper = middle
print('That was fun!')
```

Run afterwards *regardless* of test

157

Python Basics



Nested Conditions

Without elif...

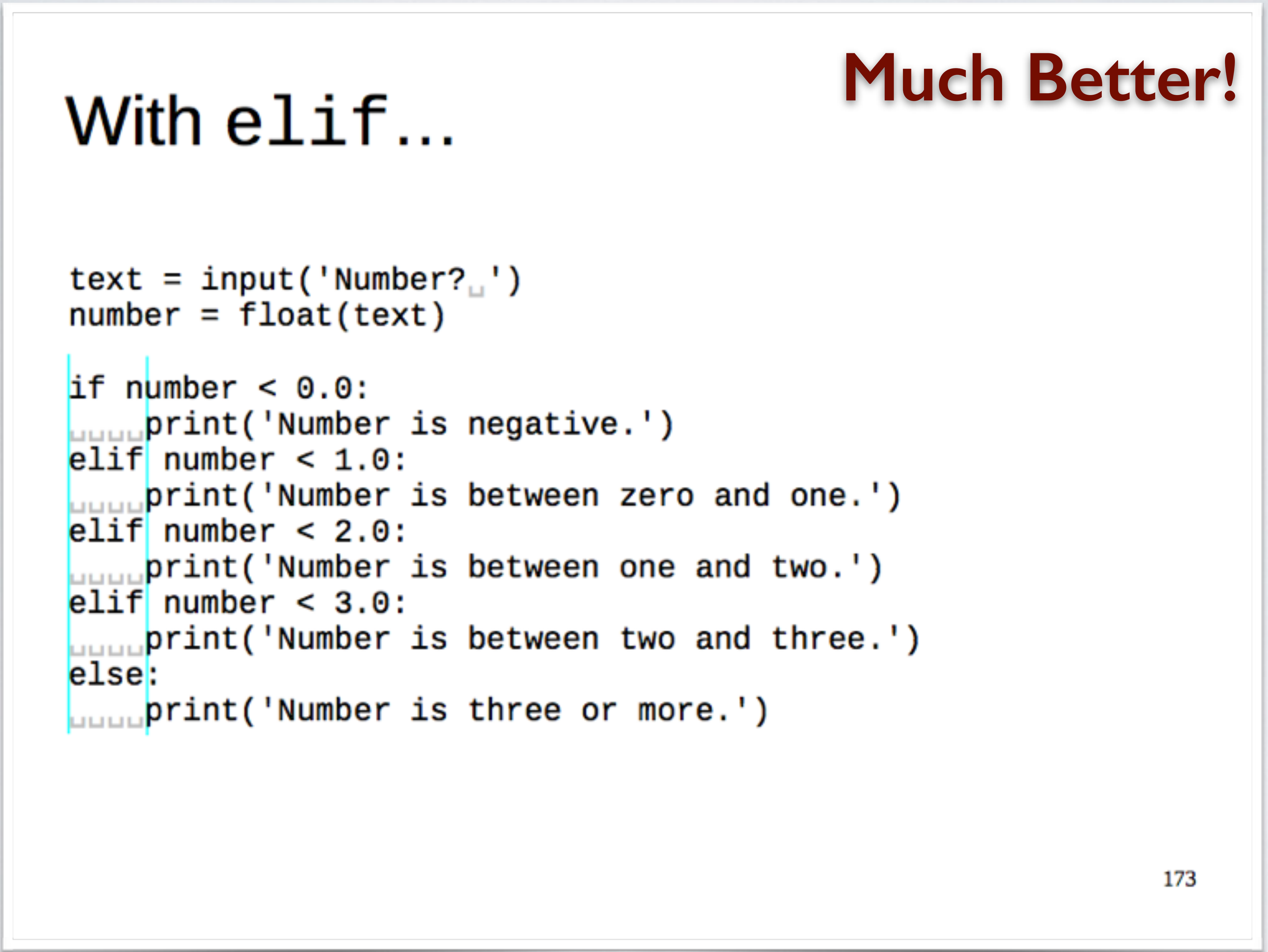
```
text = input('Number?␣')
number = float(text)

if number < 0.0:
␣␣␣␣print('Number is negative.')
else:
␣␣␣␣if number < 1.0:
␣␣␣␣␣␣␣␣print('Number is between zero and one.')
␣␣␣␣else:
␣␣␣␣␣␣␣␣if number < 2.0:
␣␣␣␣␣␣␣␣␣␣␣␣print('Number is between one and two.')
␣␣␣␣␣␣␣␣else:
␣␣␣␣␣␣␣␣␣␣␣␣if number < 3.0:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣print('Number is between two and three.')
␣␣␣␣␣␣␣␣␣␣␣␣else:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣print('Number is three or more.')
```

Stacked clauses get unwieldy

172

Slide taken from University of Cambridge's Python AB Course

Python Basics



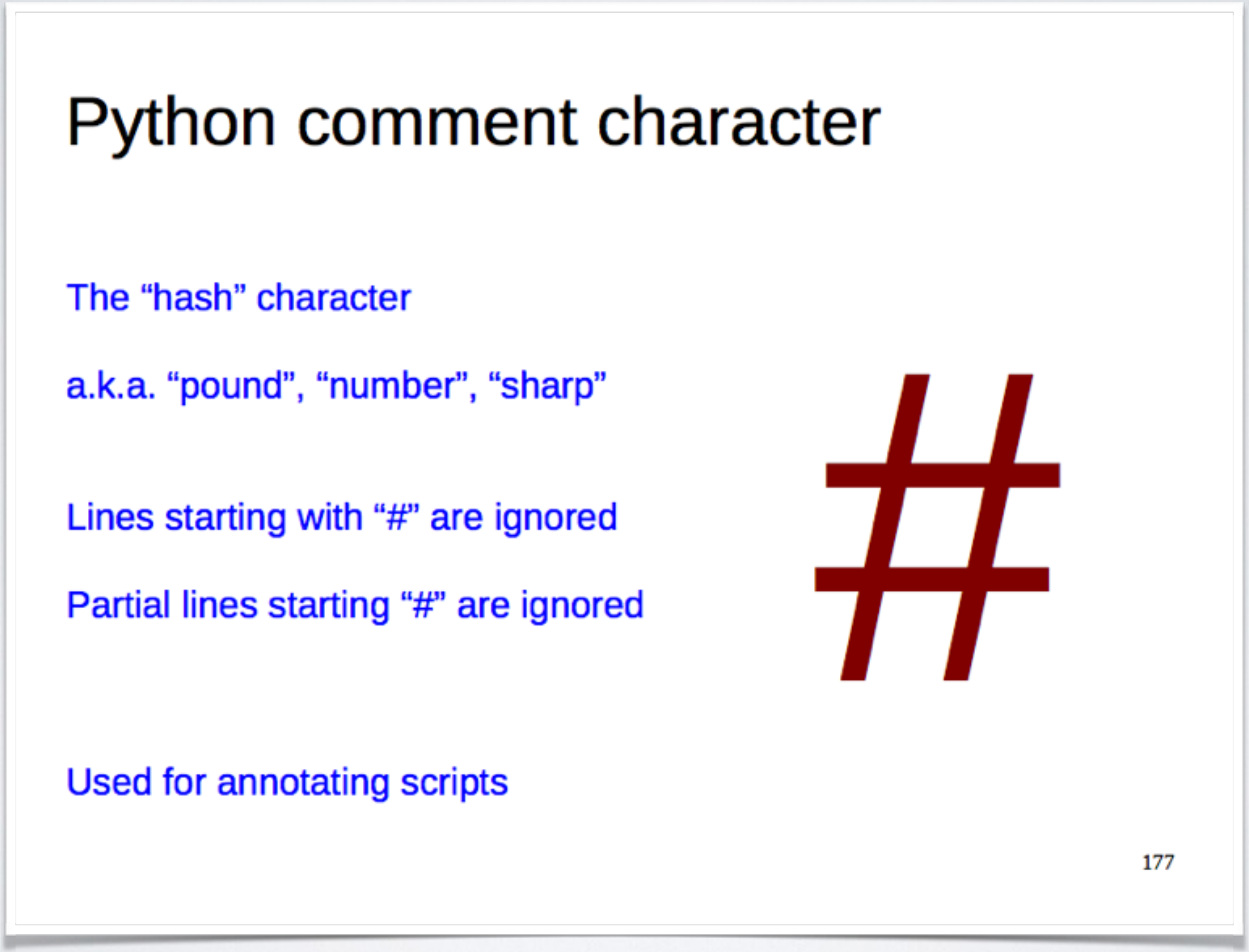
Much Better!

With elif…

```
text = input('Number?␣')
number = float(text)

if number < 0.0:
␣␣␣␣print('Number is negative.')
elif number < 1.0:
␣␣␣␣print('Number is between zero and one.')
elif number < 2.0:
␣␣␣␣print('Number is between one and two.')
elif number < 3.0:
␣␣␣␣print('Number is between two and three.')
else:
␣␣␣␣print('Number is three or more.')
```

173

Python Basics



Python comment character

The “hash” character

a.k.a. “pound”, “number”, “sharp”

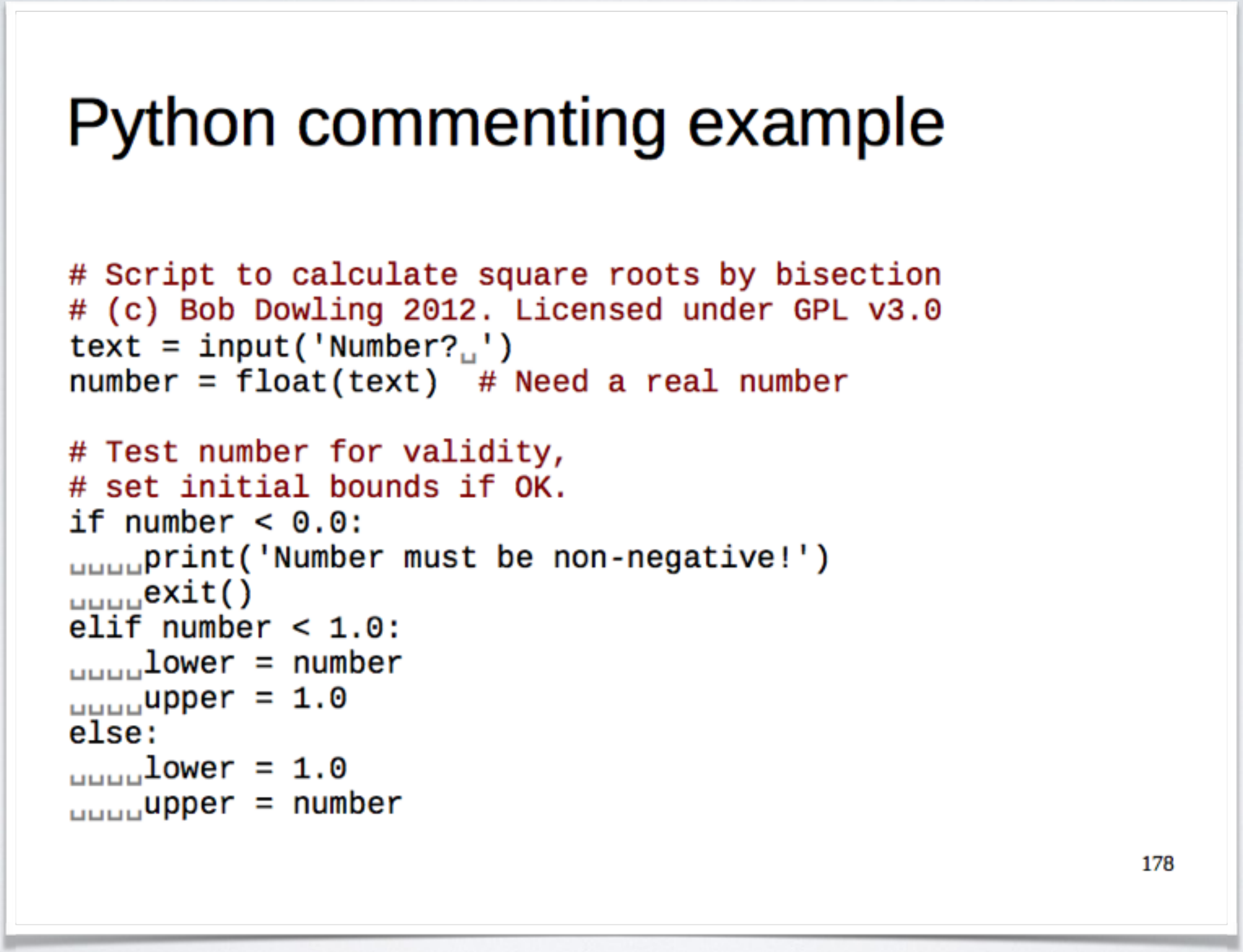
Lines starting with “#” are ignored

Partial lines starting “#” are ignored

Used for annotating scripts

177

Python Basics



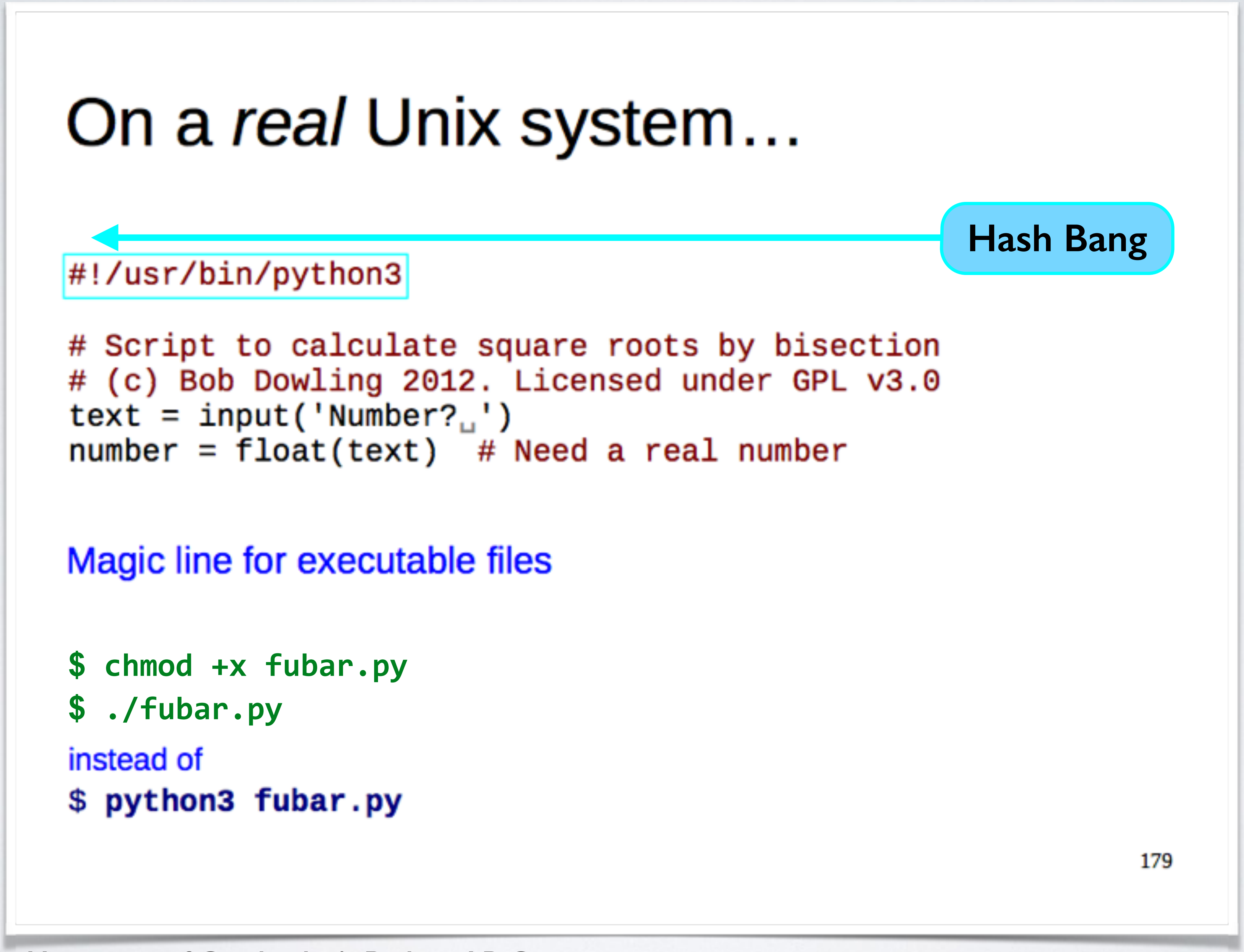
Python commenting example

```
# Script to calculate square roots by bisection
# (c) Bob Dowling 2012. Licensed under GPL v3.0
text = input('Number?␣')
number = float(text)  # Need a real number

# Test number for validity,
# set initial bounds if OK.
if number < 0.0:
␣␣␣␣print('Number must be non-negative!')
␣␣␣␣exit()
elif number < 1.0:
␣␣␣␣lower = number
␣␣␣␣upper = 1.0
else:
␣␣␣␣lower = 1.0
␣␣␣␣upper = number
```

178

Slide taken from University of Cambridge’s Python AB Course

Python Basics



On a *real* Unix system…

```
#!/usr/bin/python3

# Script to calculate square roots by bisection
# (c) Bob Dowling 2012. Licensed under GPL v3.0
text = input('Number?␣')
number = float(text)  # Need a real number
```

Magic line for executable files

```
$ chmod +x fubar.py
$ ./fubar.py
```

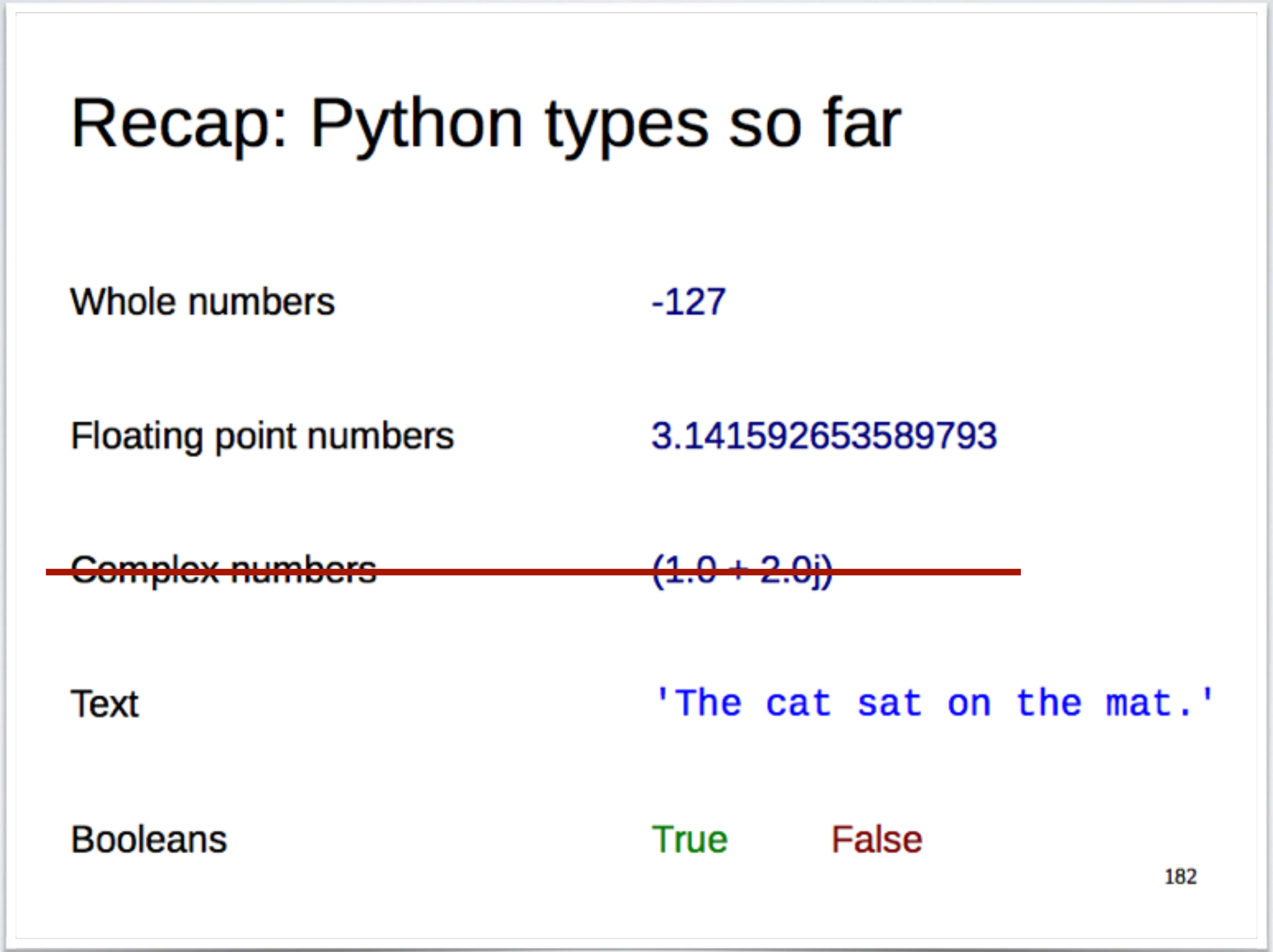
instead of

```
$ python3 fubar.py
```

179

Slide taken from University of Cambridge's Python AB Course

Python Basics

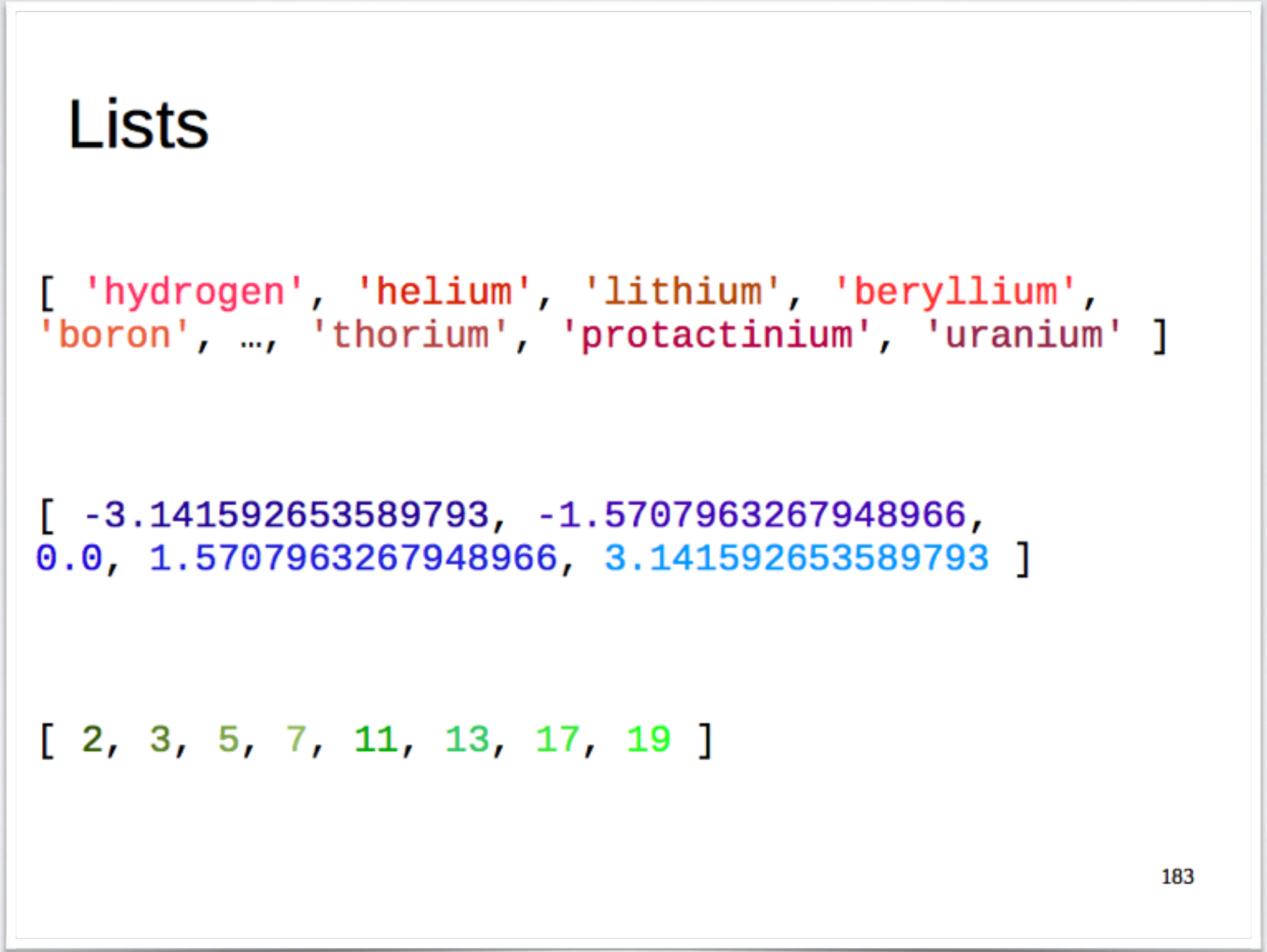


Recap: Python types so far

Whole numbers	-127	
Floating point numbers	3.141592653589793	
~~Complex numbers~~	~~(1.0 + 2.0j)~~	
Text	`'The cat sat on the mat.'`	
Booleans	True	False

182

Python Basics



Lists

```
[ 'hydrogen', 'helium', 'lithium', 'beryllium',
'boron', …, 'thorium', 'protactinium', 'uranium' ]
```

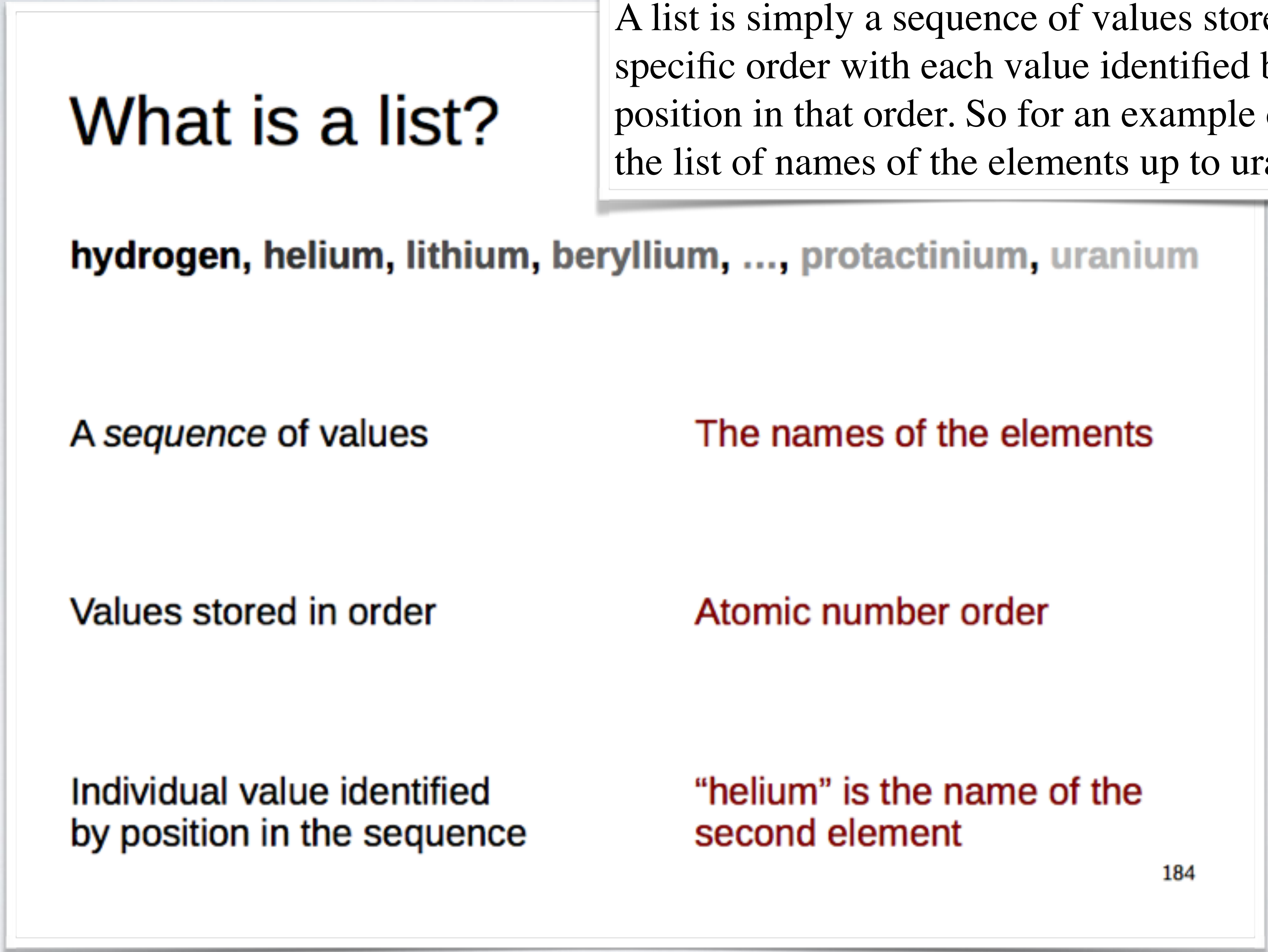
```
[ -3.141592653589793, -1.5707963267948966,
0.0, 1.5707963267948966, 3.141592653589793 ]
```

```
[ 2, 3, 5, 7, 11, 13, 17, 19 ]
```

183

Python Basics

A list is simply a sequence of values stored in a specific order with each value identified by its position in that order. So for an example consider the list of names of the elements up to uranium.



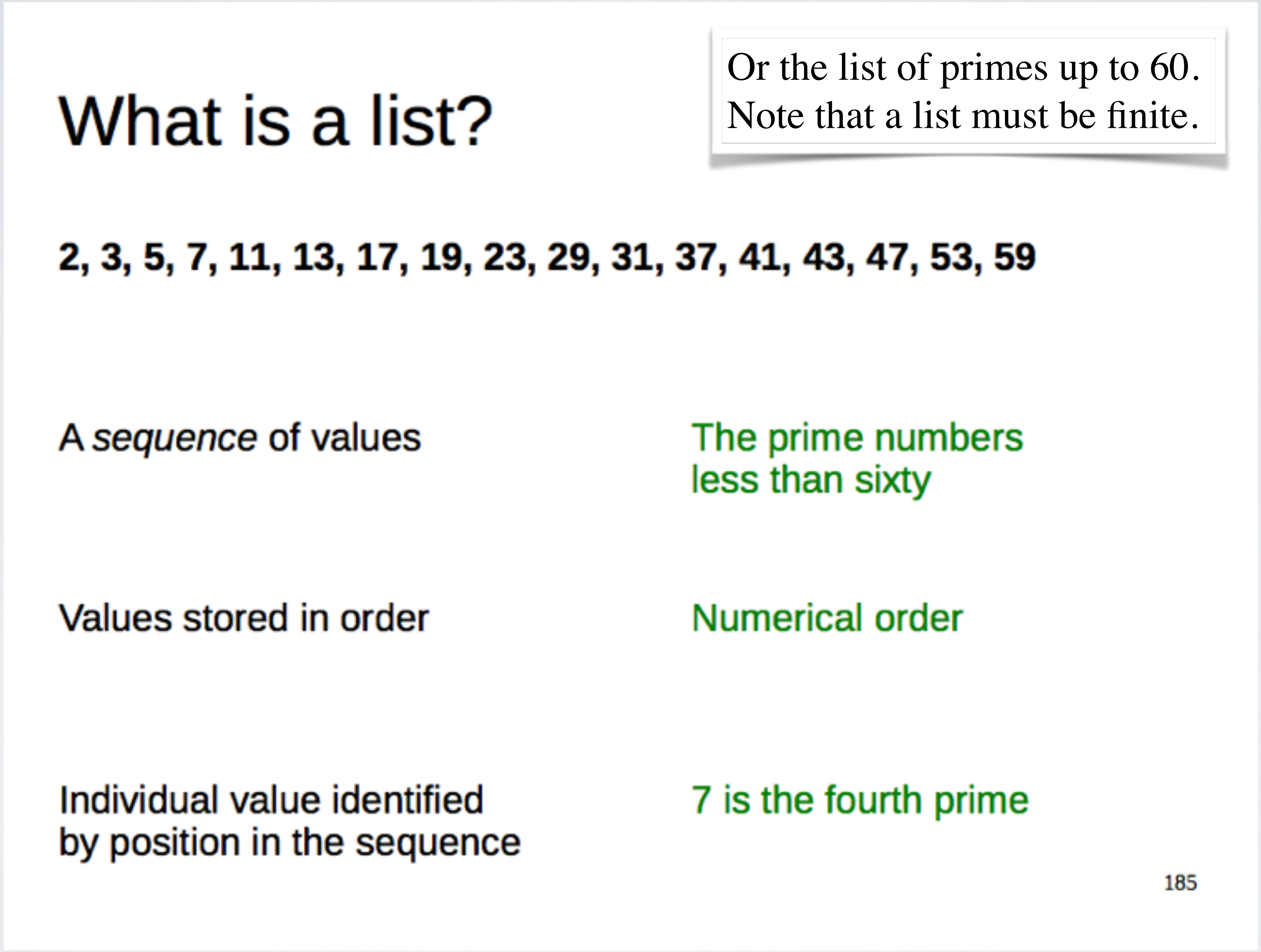
What is a list?

hydrogen, helium, lithium, beryllium, …, protactinium, uranium

A *sequence* of values	The names of the elements
Values stored in order	Atomic number order
Individual value identified by position in the sequence	"helium" is the name of the second element

184

Python Basics



Or the list of primes up to 60.
Note that a list must be finite.

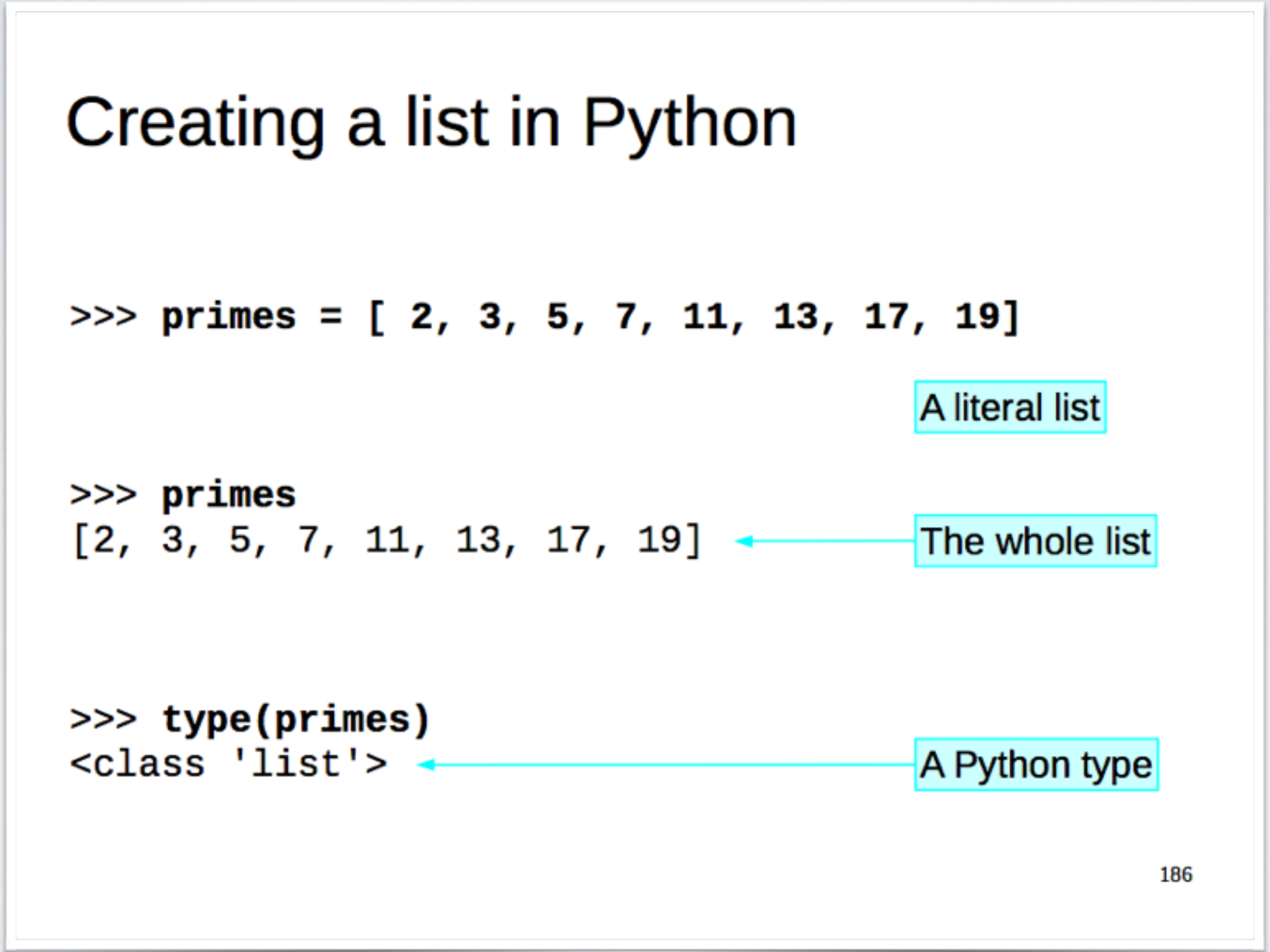
What is a list?

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59

A *sequence* of values	The prime numbers less than sixty
Values stored in order	Numerical order
Individual value identified by position in the sequence	7 is the fourth prime

185

Python Basics



Creating a list in Python

```
>>> primes = [ 2, 3, 5, 7, 11, 13, 17, 19]
```

A literal list

```
>>> primes
[2, 3, 5, 7, 11, 13, 17, 19]
```

The whole list

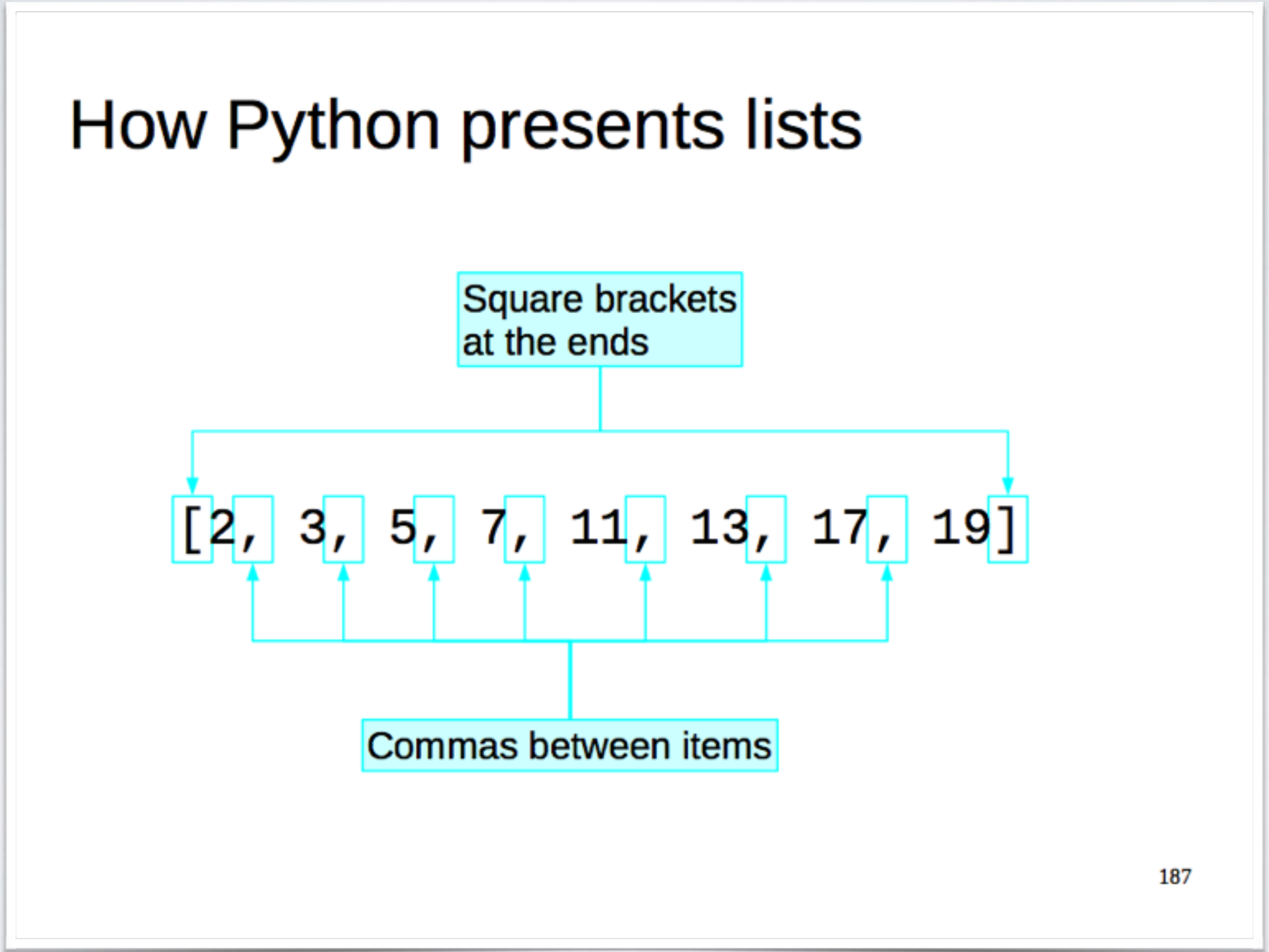
```
>>> type(primes)
<class 'list'>
```

A Python type

186

Slide taken from University of Cambridge's Python AB Course

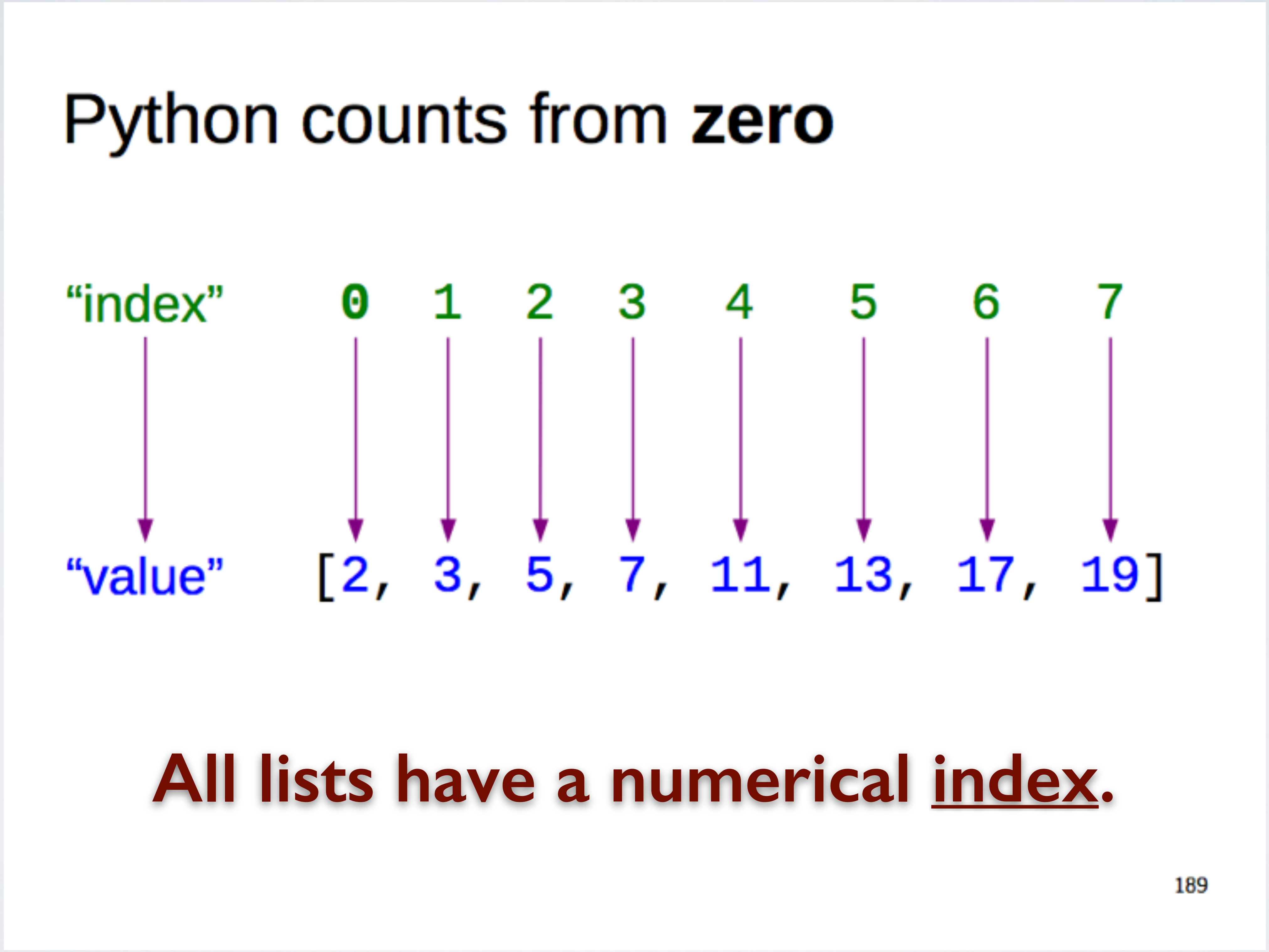
Python Basics



How Python presents lists

Slide taken from University of Cambridge's Python AB Course

Python Basics

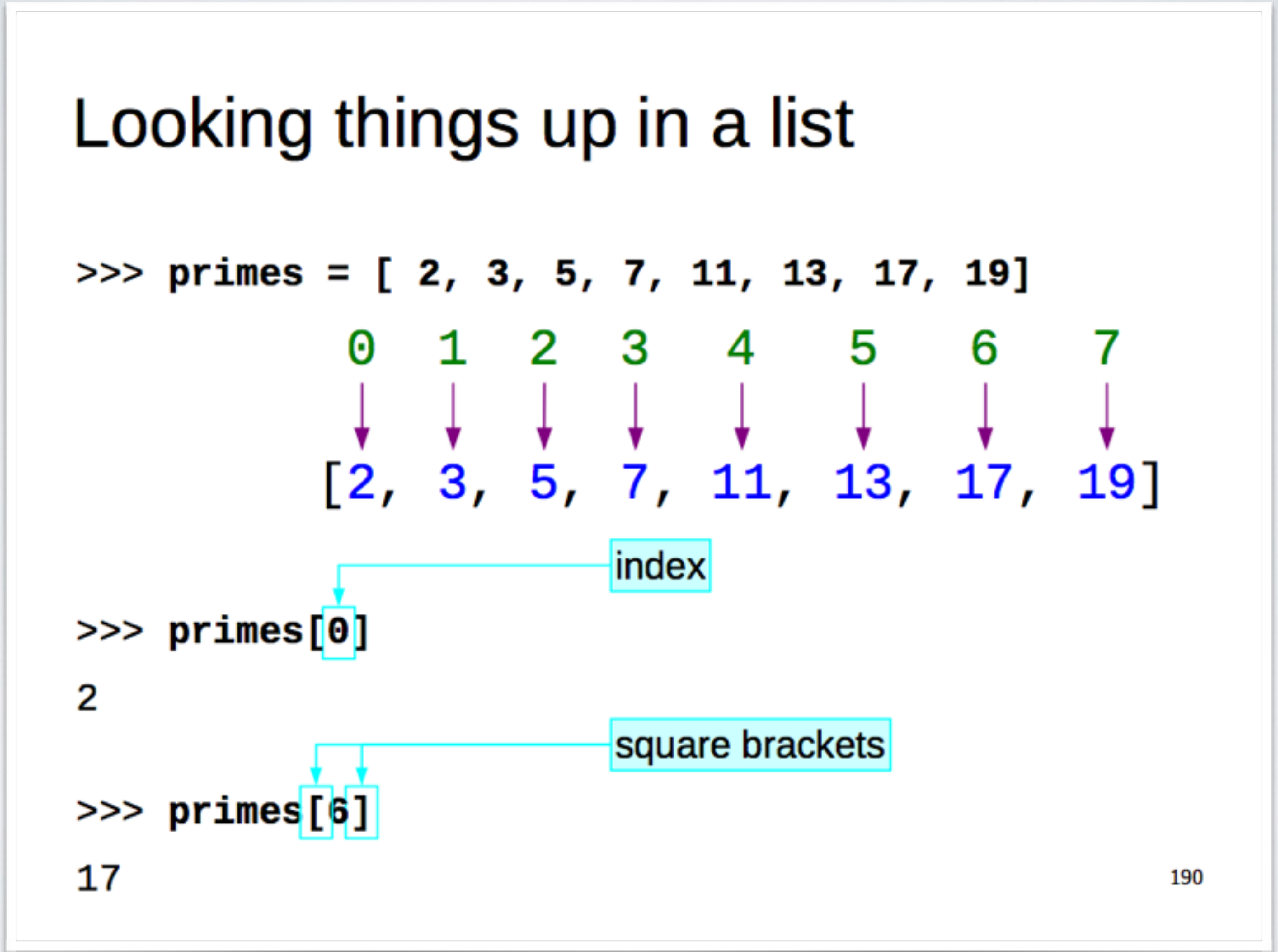


Python counts from **zero**

All lists have a numerical index.

189

Python Basics



Looking things up in a list

```
>>> primes = [ 2, 3, 5, 7, 11, 13, 17, 19]
```

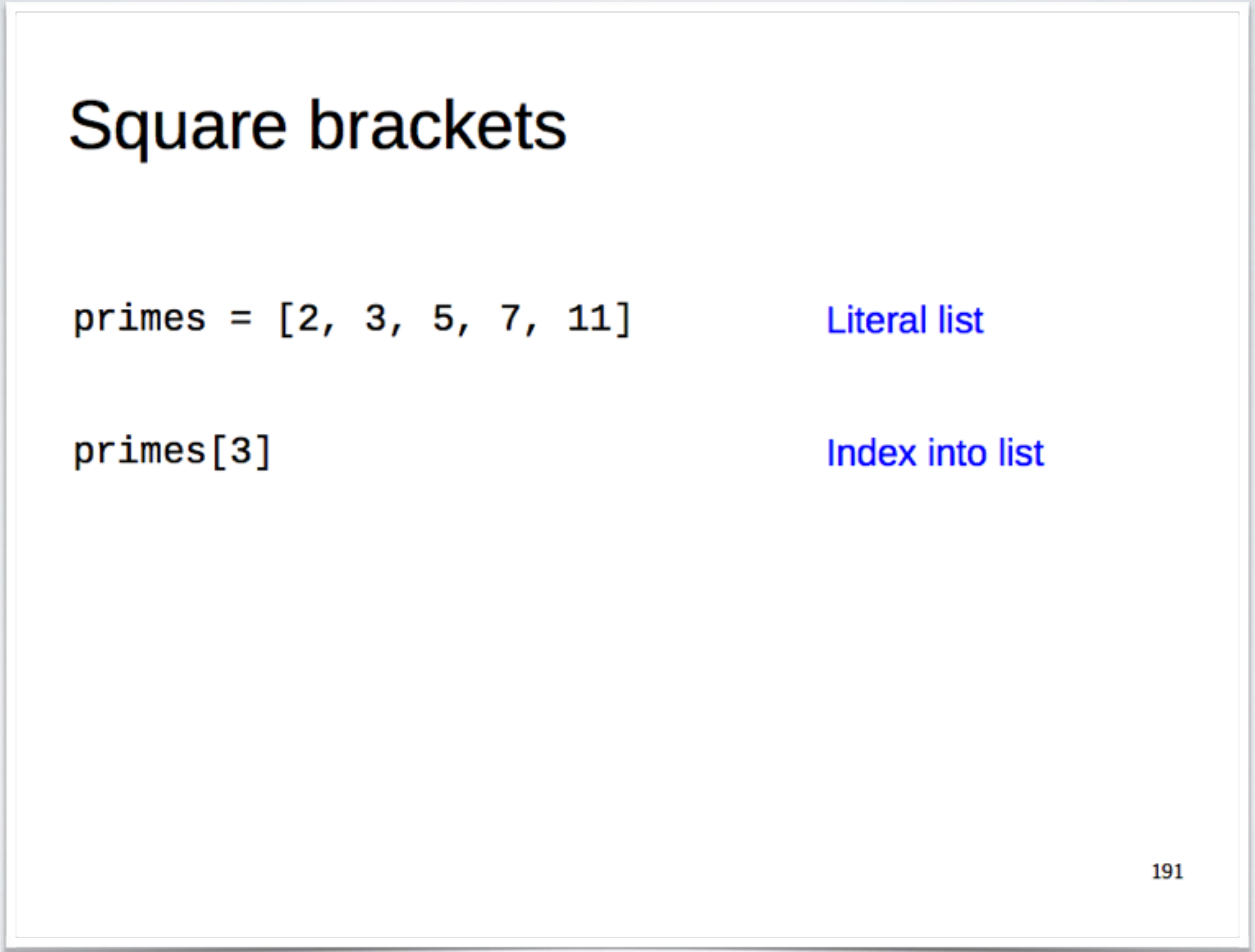
0 1 2 3 4 5 6 7
[2, 3, 5, 7, 11, 13, 17, 19]
index
square brackets

```
>>> primes[0]
2
>>> primes[6]
17
```

190

Slide taken from University of Cambridge's Python AB Course

Python Basics



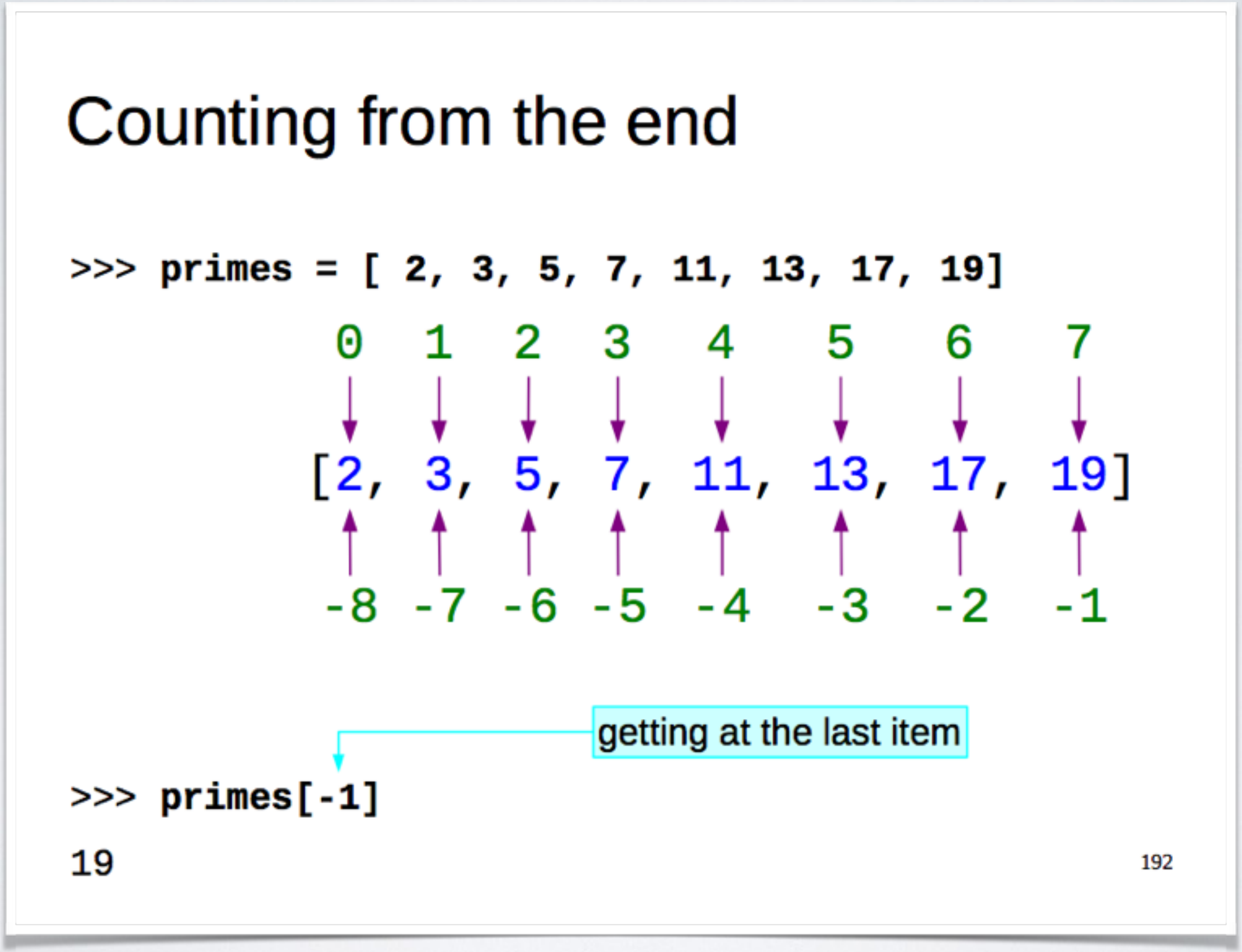
Square brackets

```
primes = [2, 3, 5, 7, 11]
```
Literal list

```
primes[3]
```
Index into list

191

Python Basics



Counting from the end

```
>>> primes = [ 2, 3, 5, 7, 11, 13, 17, 19]
```

```
 0   1   2   3    4    5    6    7
[2,  3,  5,  7,  11,  13,  17,  19]
-8  -7  -6  -5   -4   -3   -2   -1
```

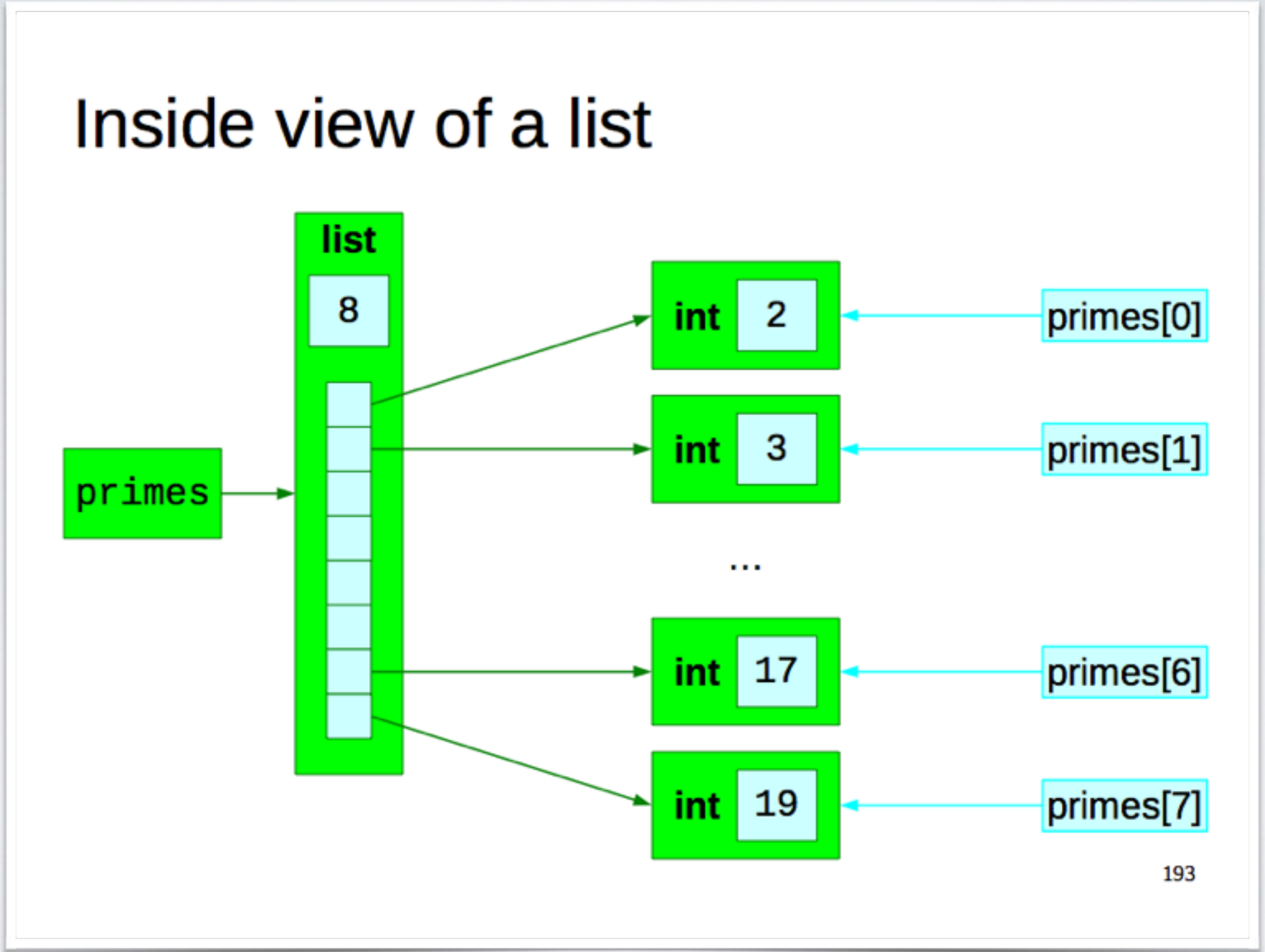
getting at the last item

```
>>> primes[-1]
19
```

192

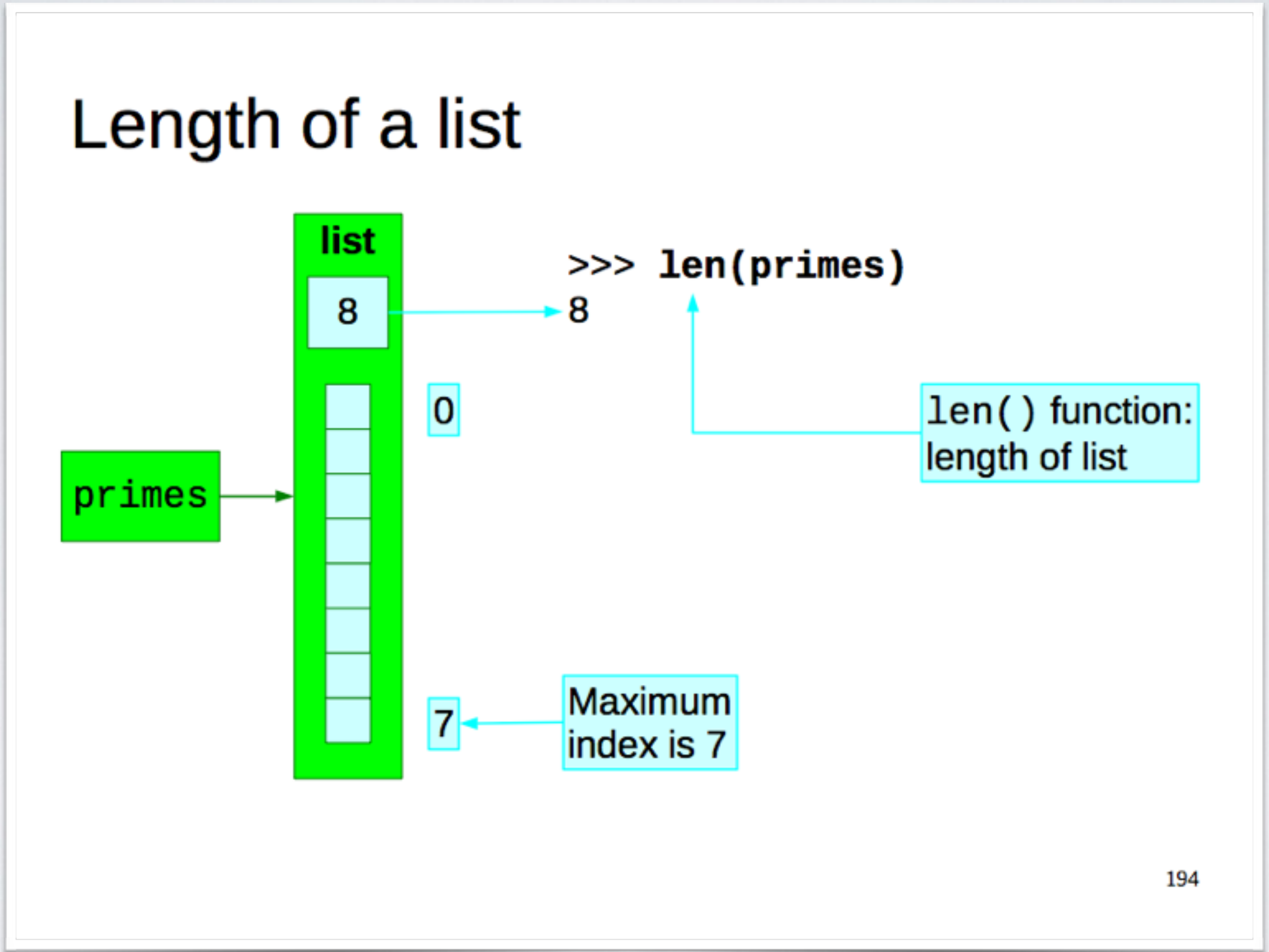
Slide taken from University of Cambridge's Python AB Course

Python Basics



Inside view of a list

Python Basics

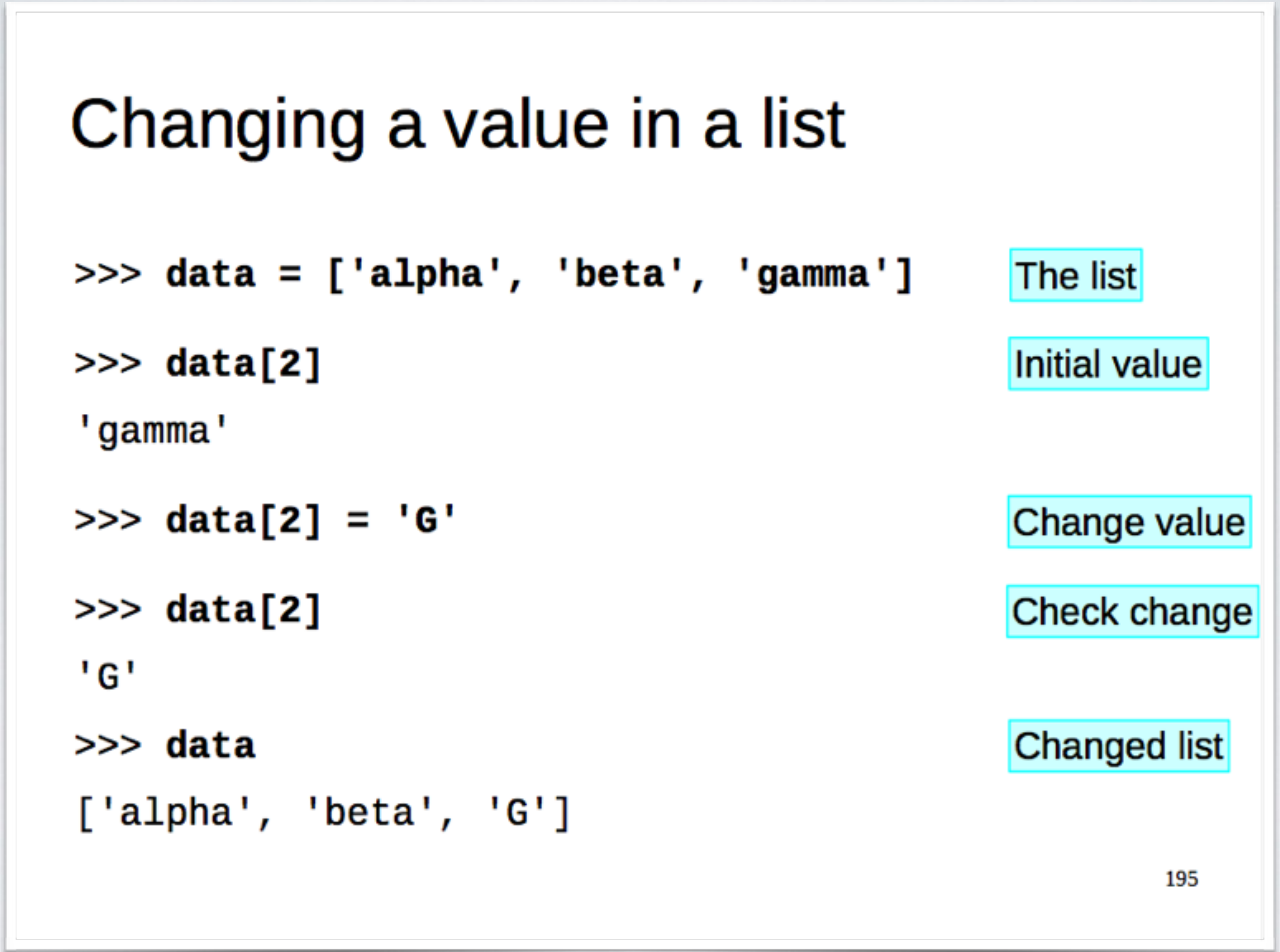


Length of a list

```
>>> len(primes)
8
```

Slide taken from University of Cambridge's Python AB Course

Python Basics



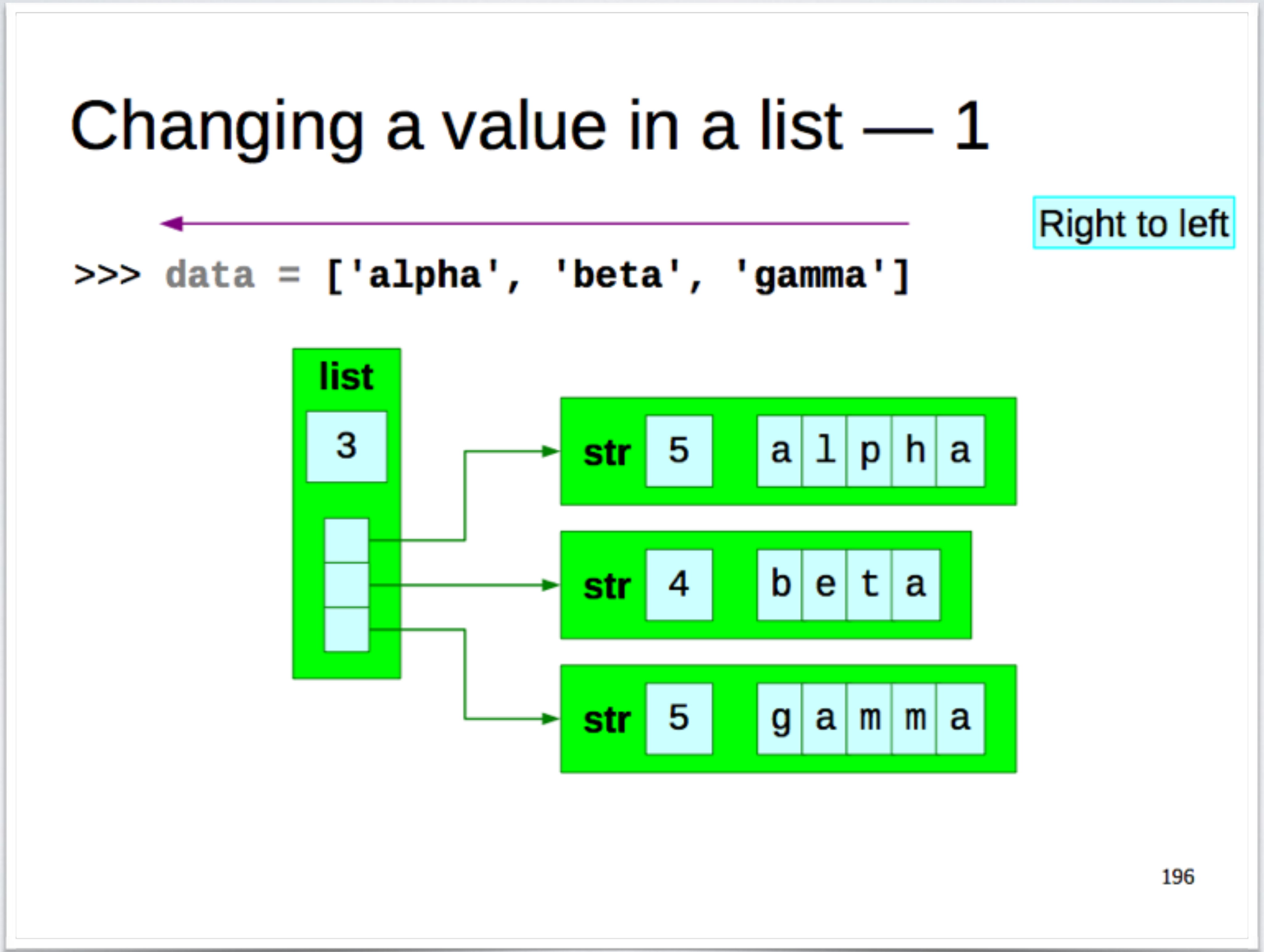
Changing a value in a list

```
>>> data = ['alpha', 'beta', 'gamma']
>>> data[2]
'gamma'
>>> data[2] = 'G'
>>> data[2]
'G'
>>> data
['alpha', 'beta', 'G']
```

The list
Initial value
Change value
Check change
Changed list

195

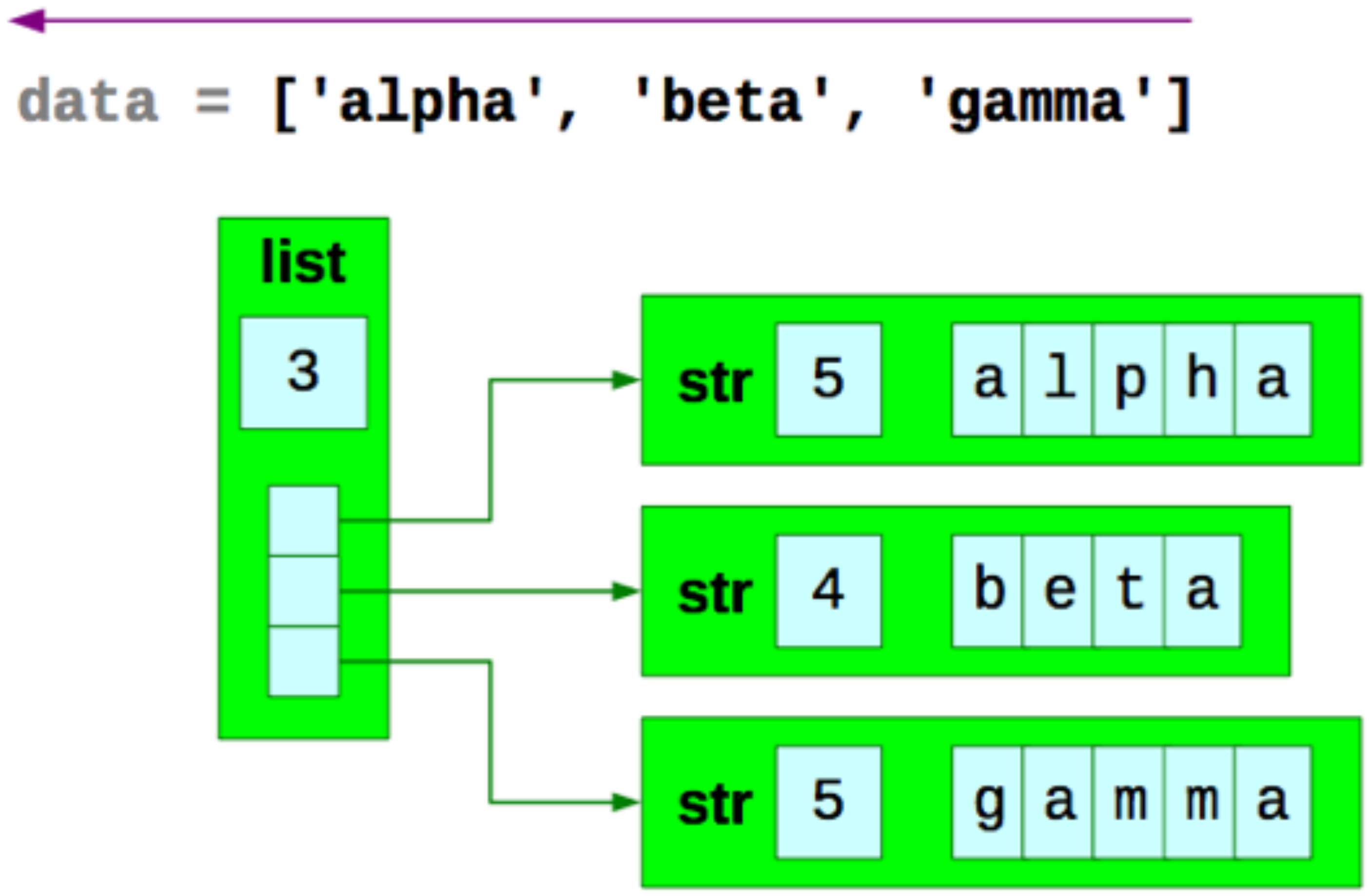
Python Basics



Changing a value in a list — 1

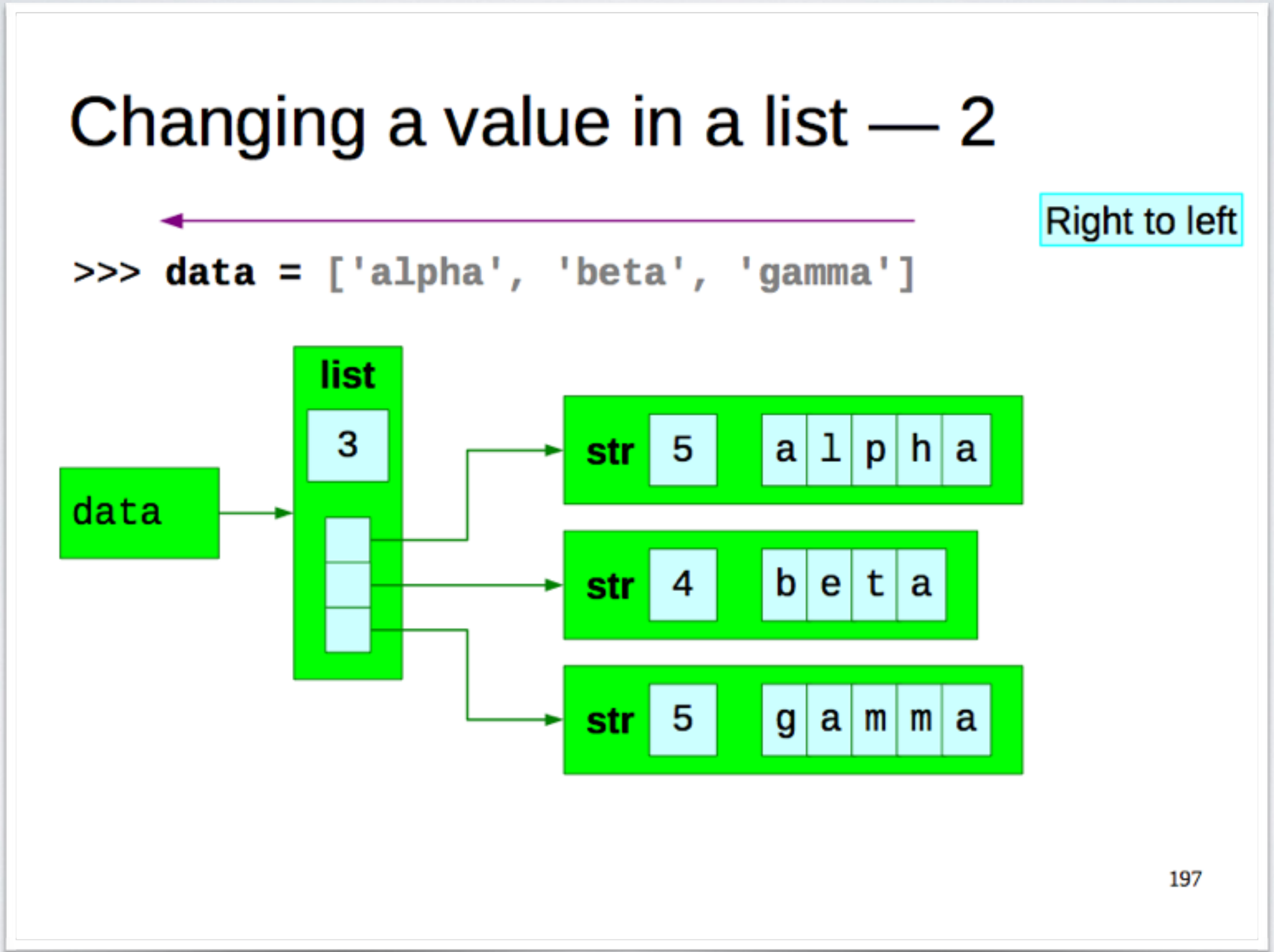
Right to left

```
>>> data = ['alpha', 'beta', 'gamma']
```



196

Python Basics



Changing a value in a list — 2

Right to left

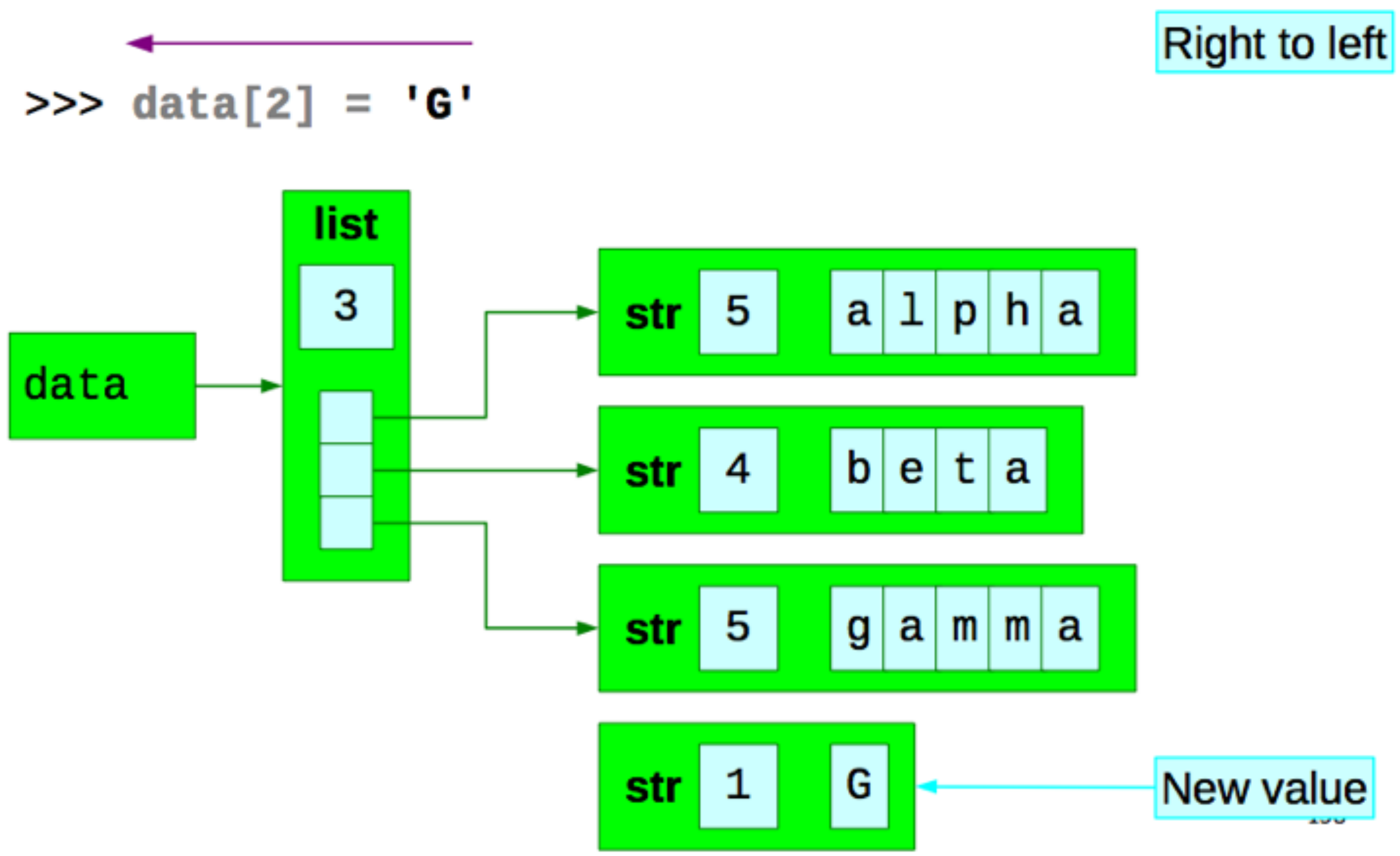
```
>>> data = ['alpha', 'beta', 'gamma']
```

197

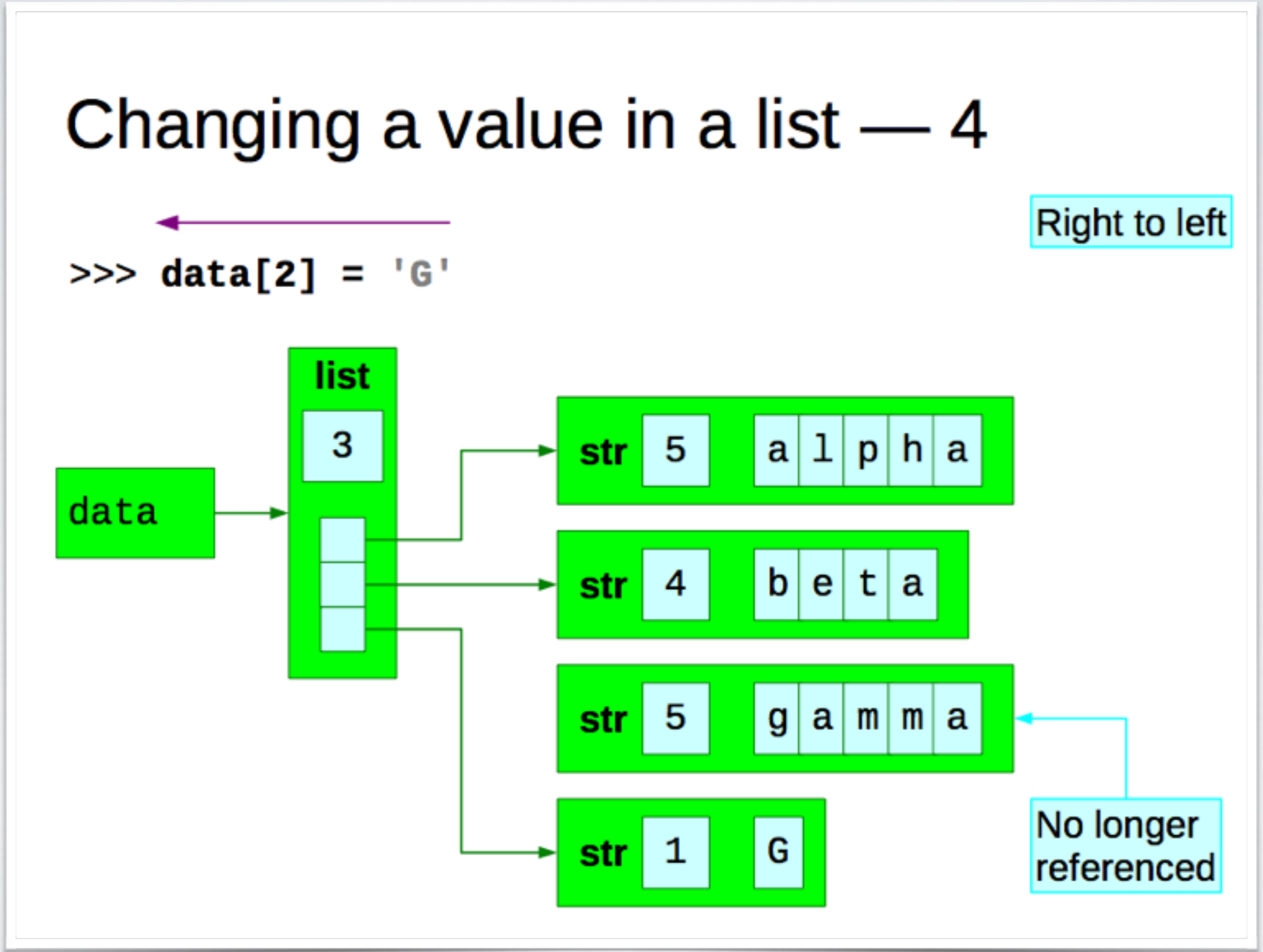
Python Basics

Changing a value in a list — 3

```
>>> data[2] = 'G'
```



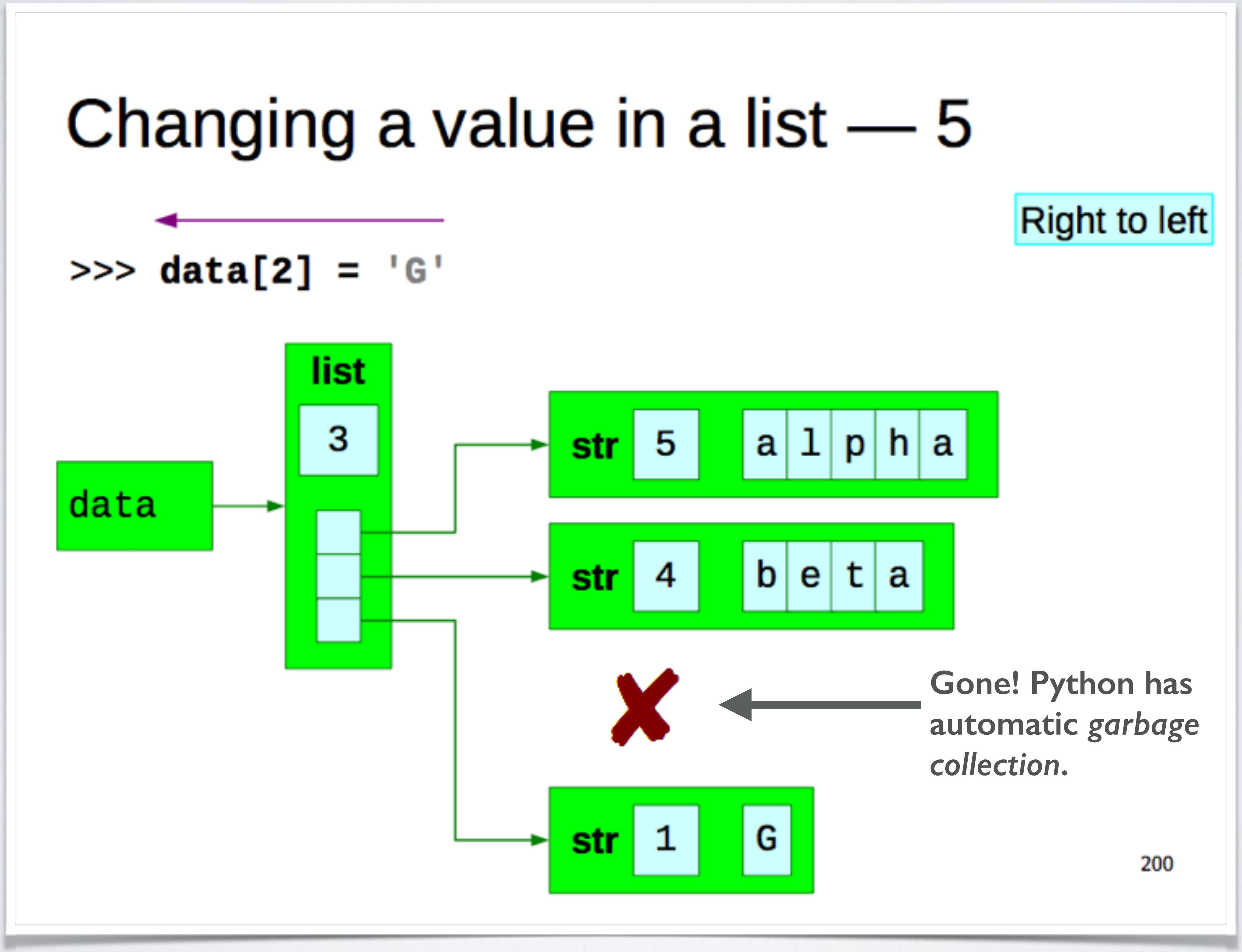
Python Basics



Changing a value in a list — 4

```
>>> data[2] = 'G'
```

Python Basics



Changing a value in a list — 5

Right to left

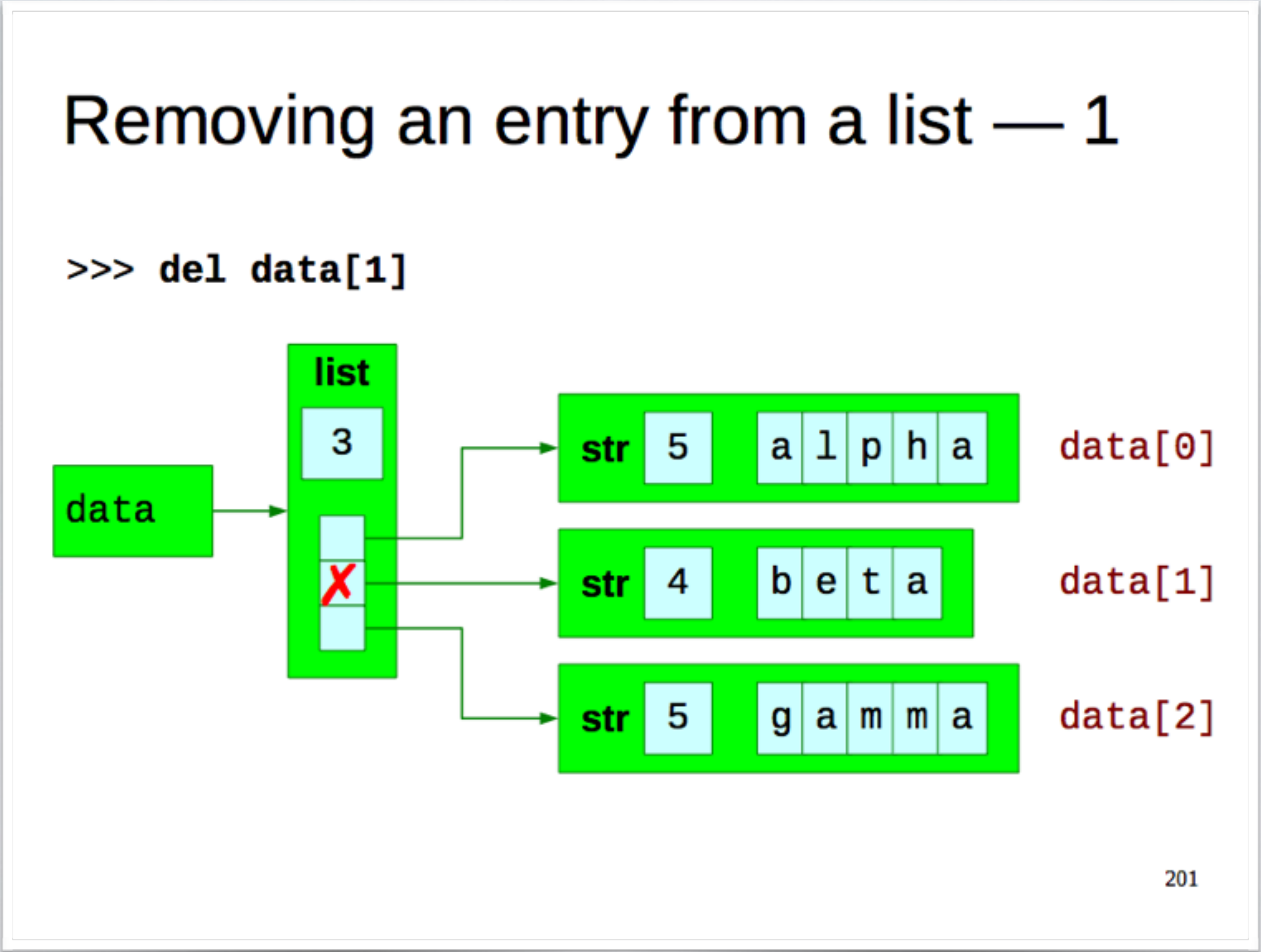
```
>>> data[2] = 'G'
```

Gone! Python has automatic *garbage collection*.

200

Slide taken from University of Cambridge's Python AB Course

Python Basics

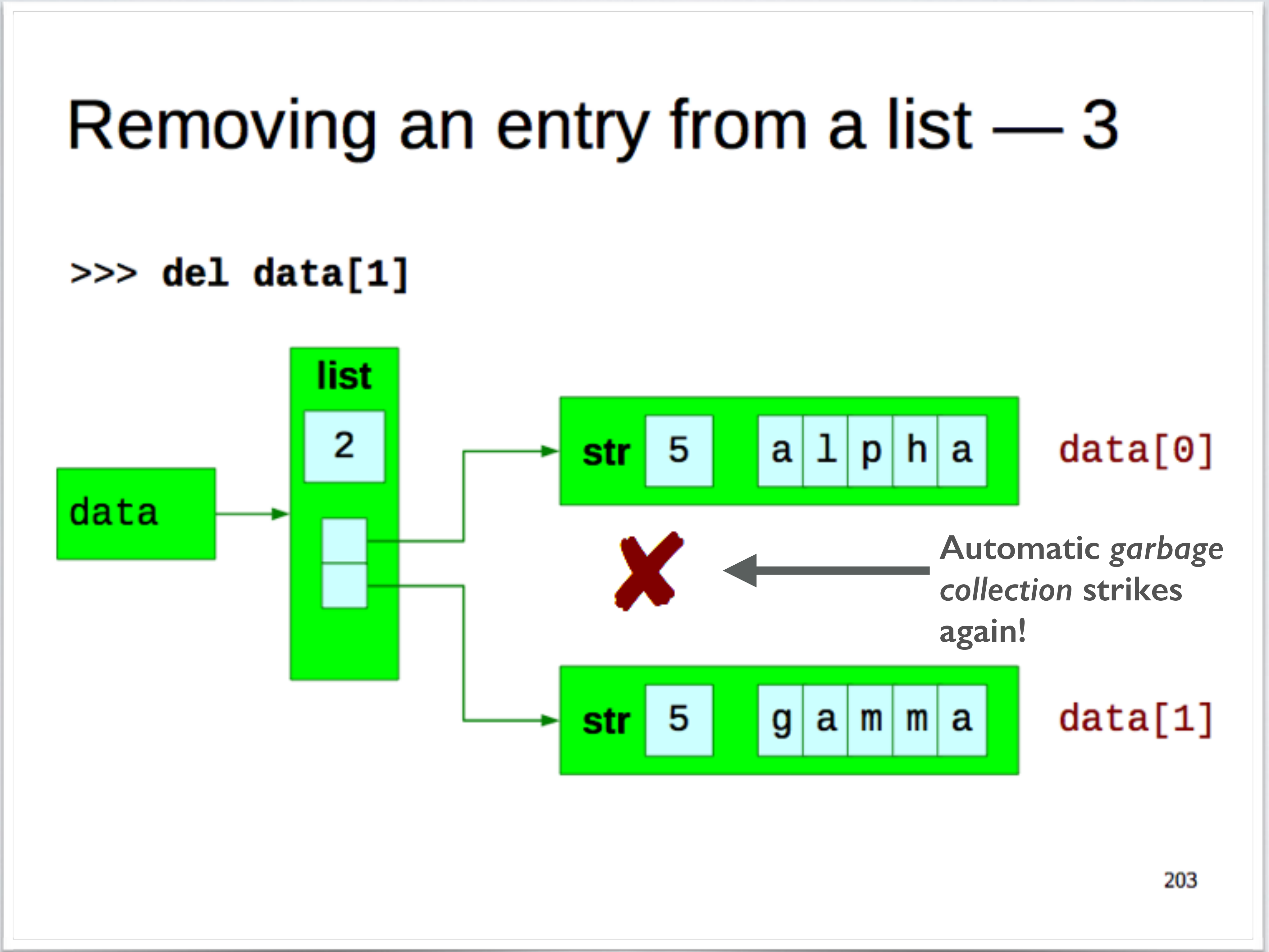


Removing an entry from a list — 1

```
>>> del data[1]
```

201

Python Basics

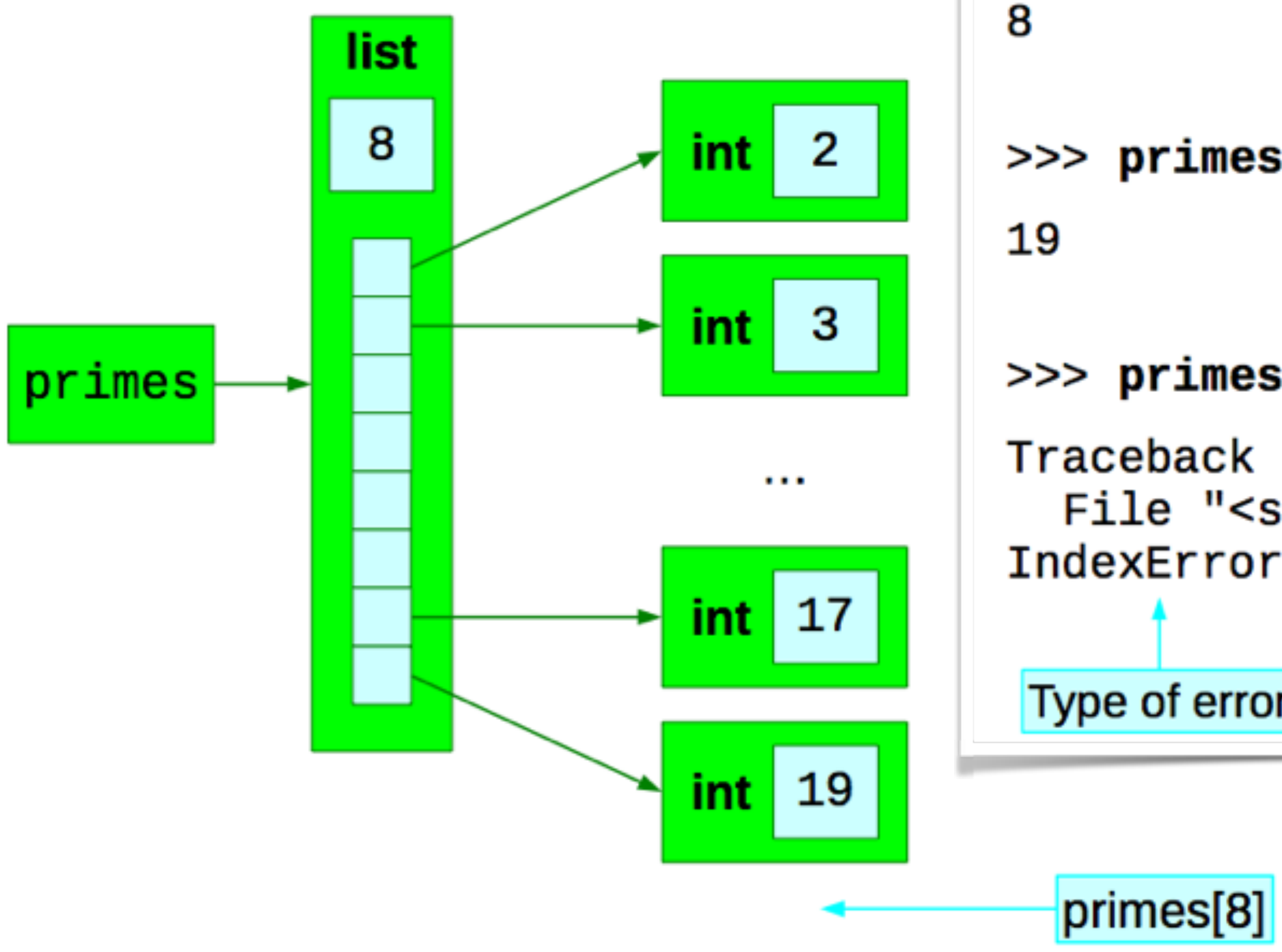


Removing an entry from a list — 3

```
>>> del data[1]
```

Python Basics

Running off the end



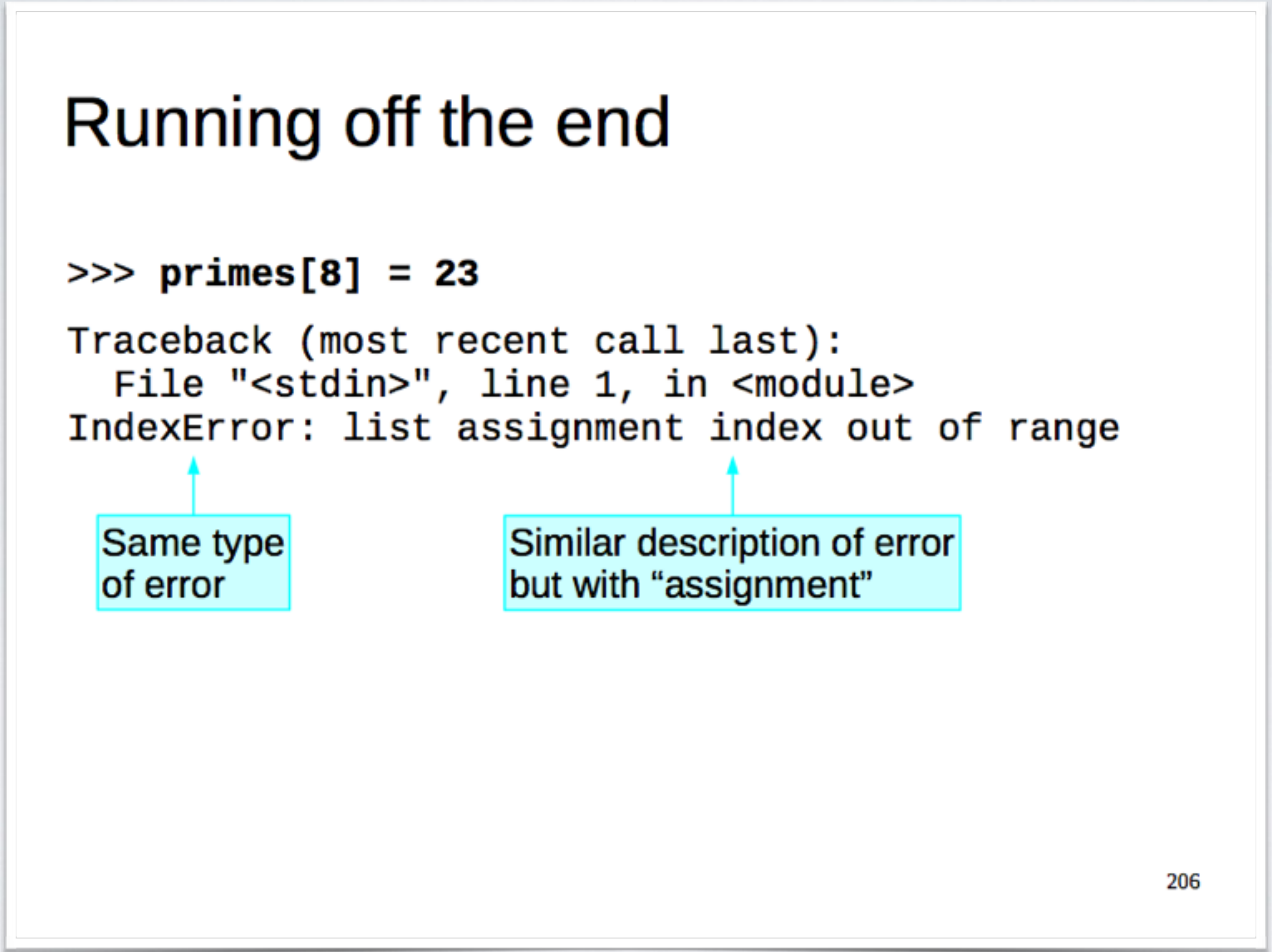
```
>>> len(primes)
8

>>> primes[7]
19

>>> primes[8]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Type of error

Description of error

204

Python Basics



Running off the end

```
>>> primes[8] = 23

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

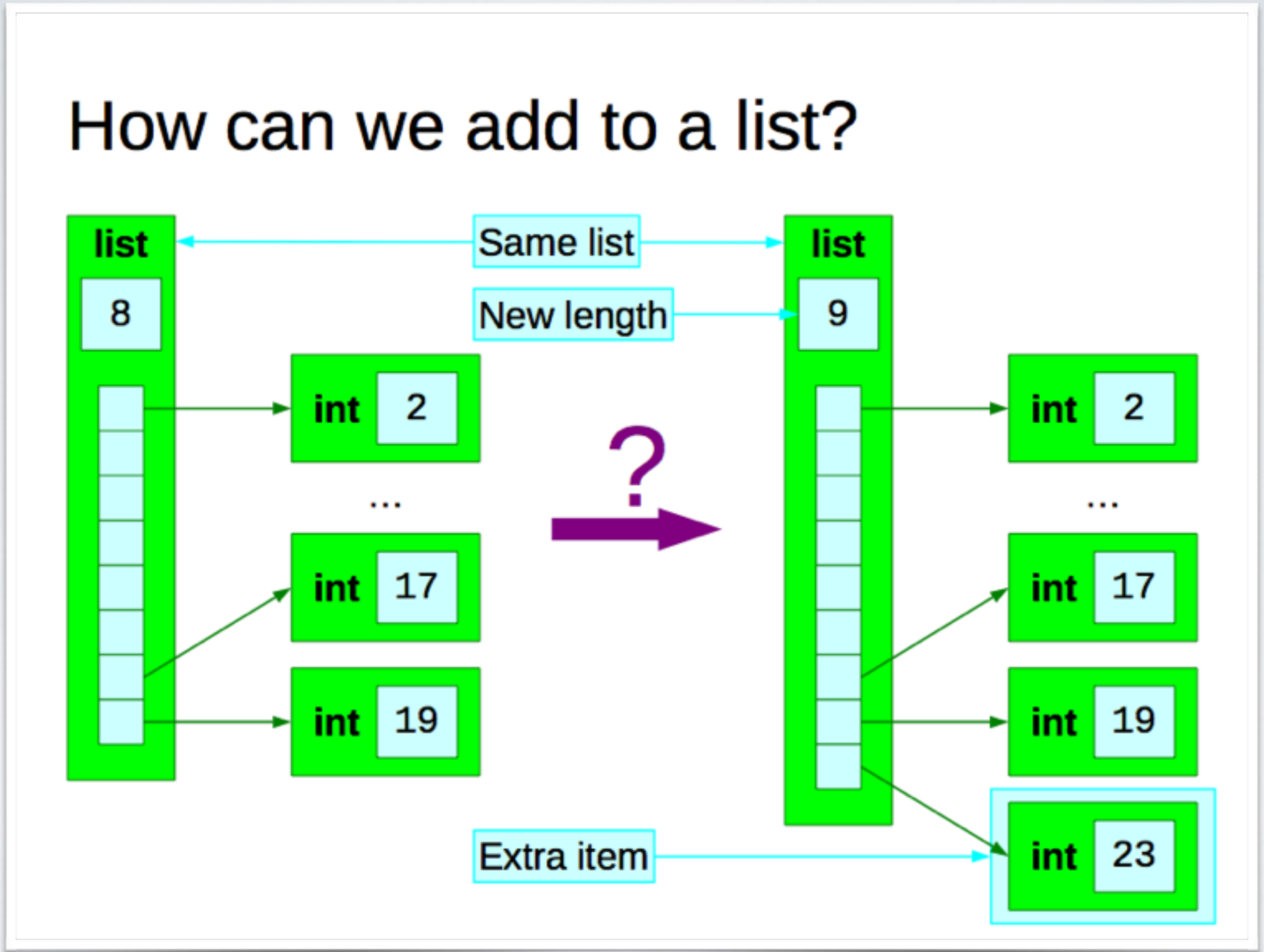
Same type of error

Similar description of error but with “assignment”

206

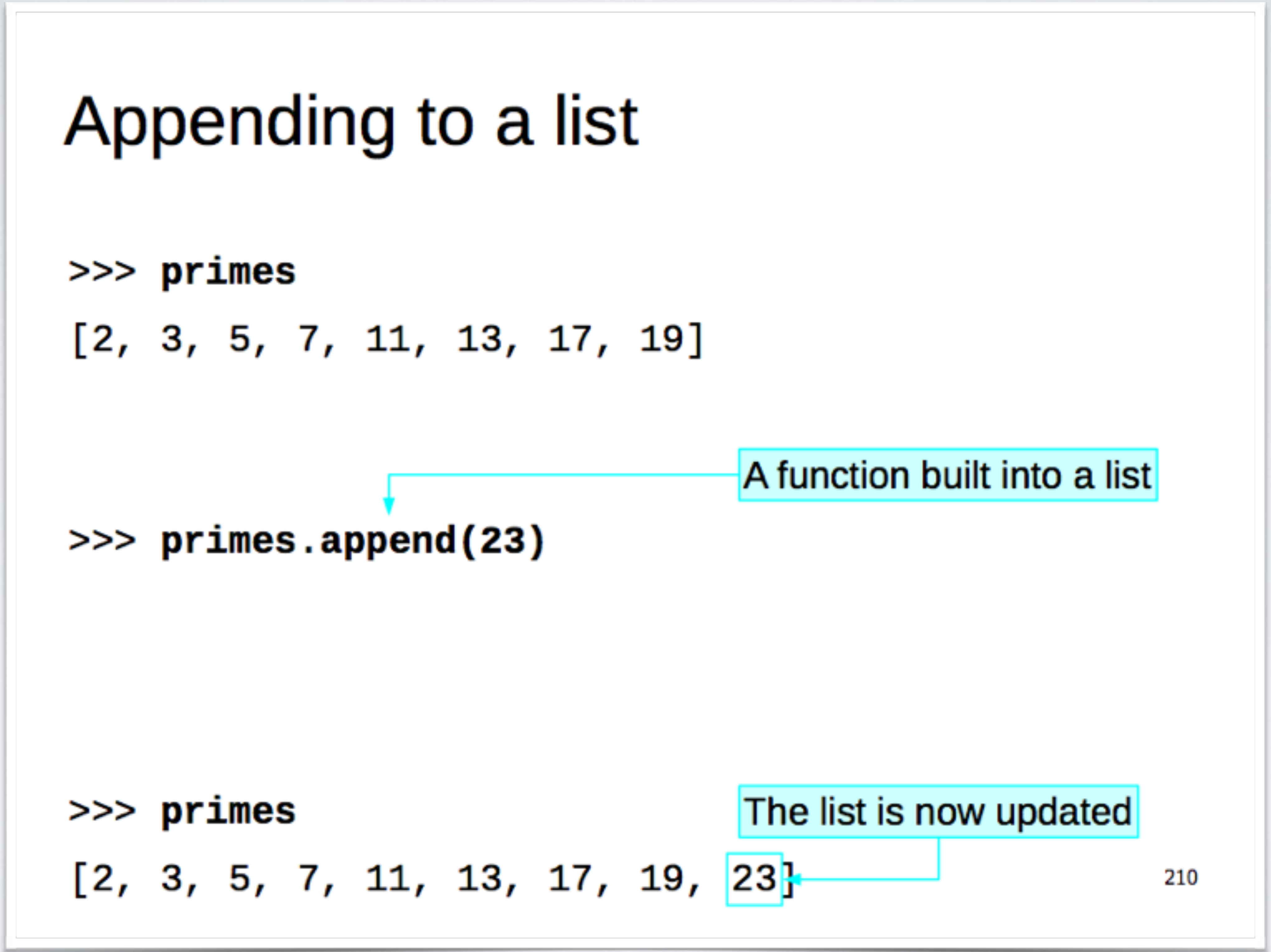
Slide taken from University of Cambridge’s Python AB Course

Python Basics



How can we add to a list?

Python Basics

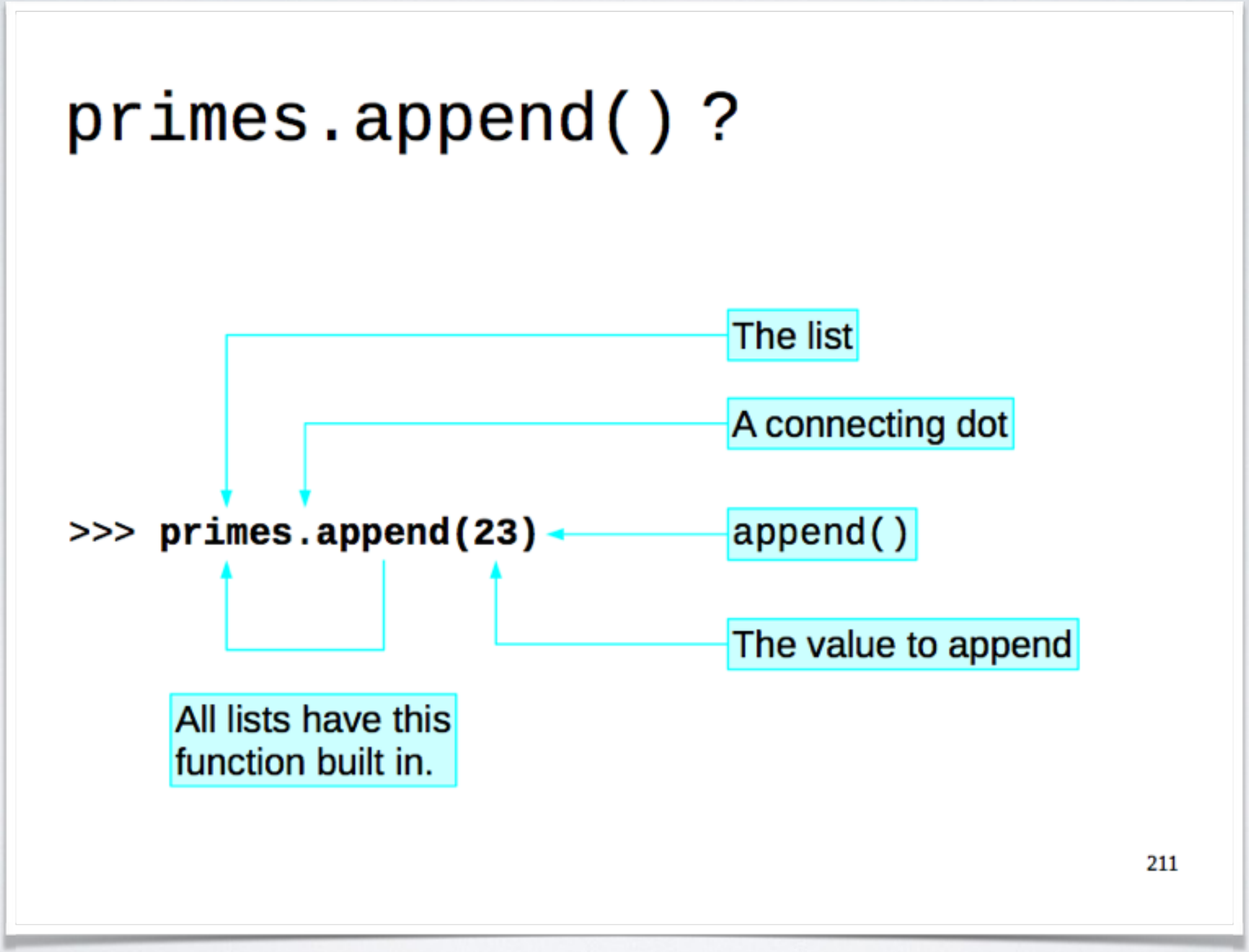


Appending to a list

```
>>> primes
[2, 3, 5, 7, 11, 13, 17, 19]

>>> primes.append(23)

>>> primes
[2, 3, 5, 7, 11, 13, 17, 19, 23]
```

A function built into a list
The list is now updated
210

Python Basics

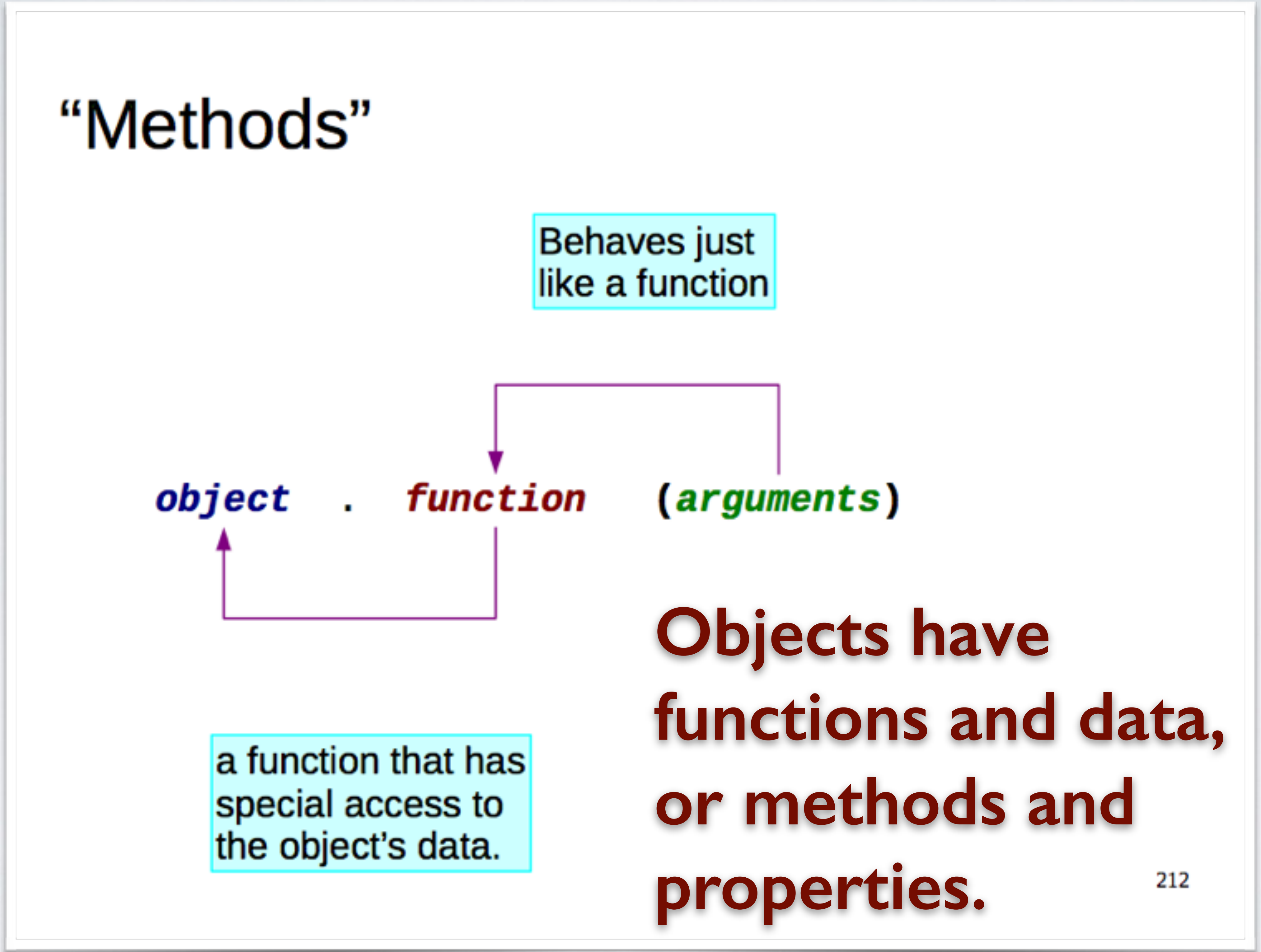


primes.append() ?

```
>>> primes.append(23)
```

Slide taken from University of Cambridge's Python AB Course

Python Basics



“Methods”

Behaves just like a function

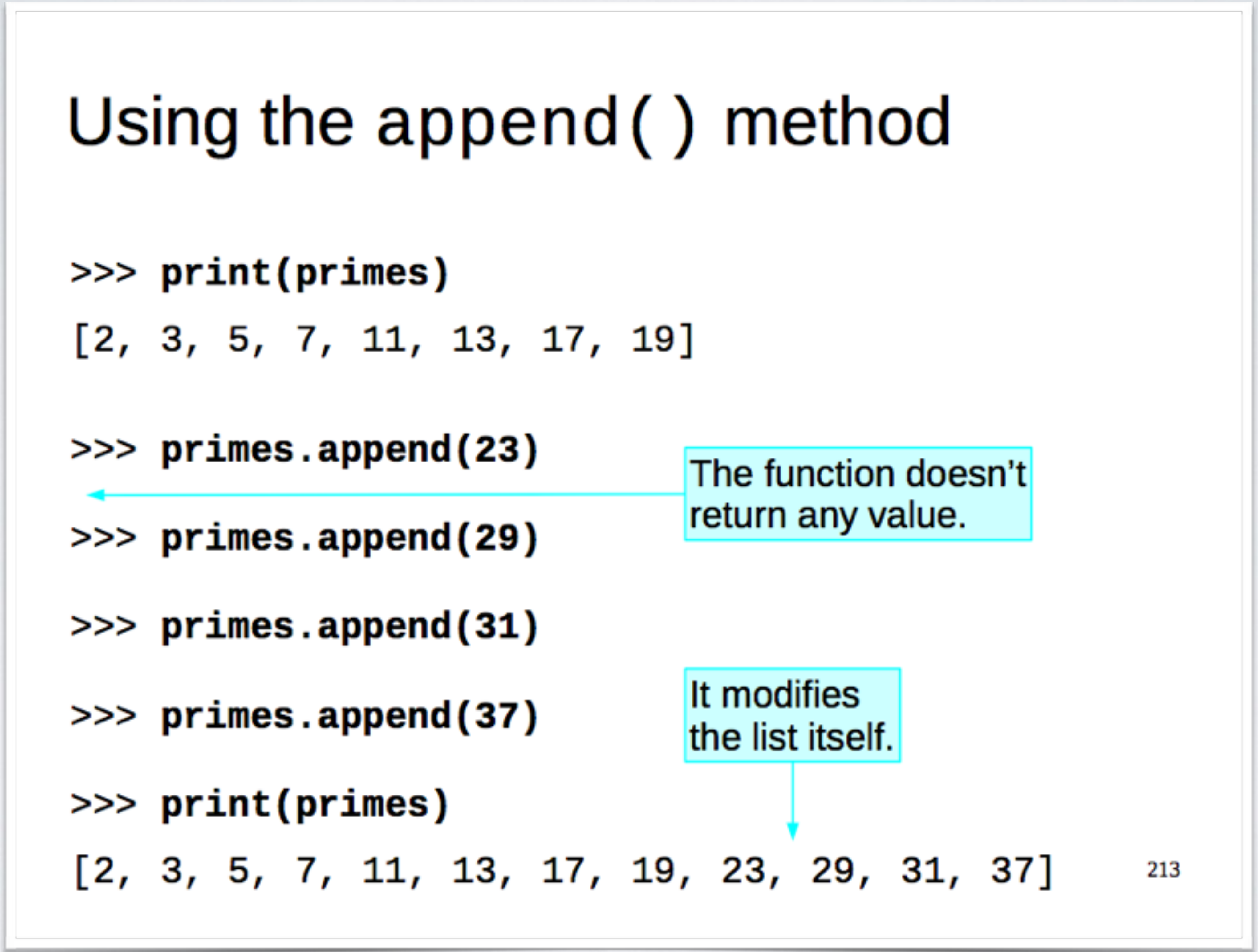
```
object . function (arguments)
```

a function that has special access to the object’s data.

Objects have functions and data, or methods and properties.

212

Python Basics



Using the append( ) method

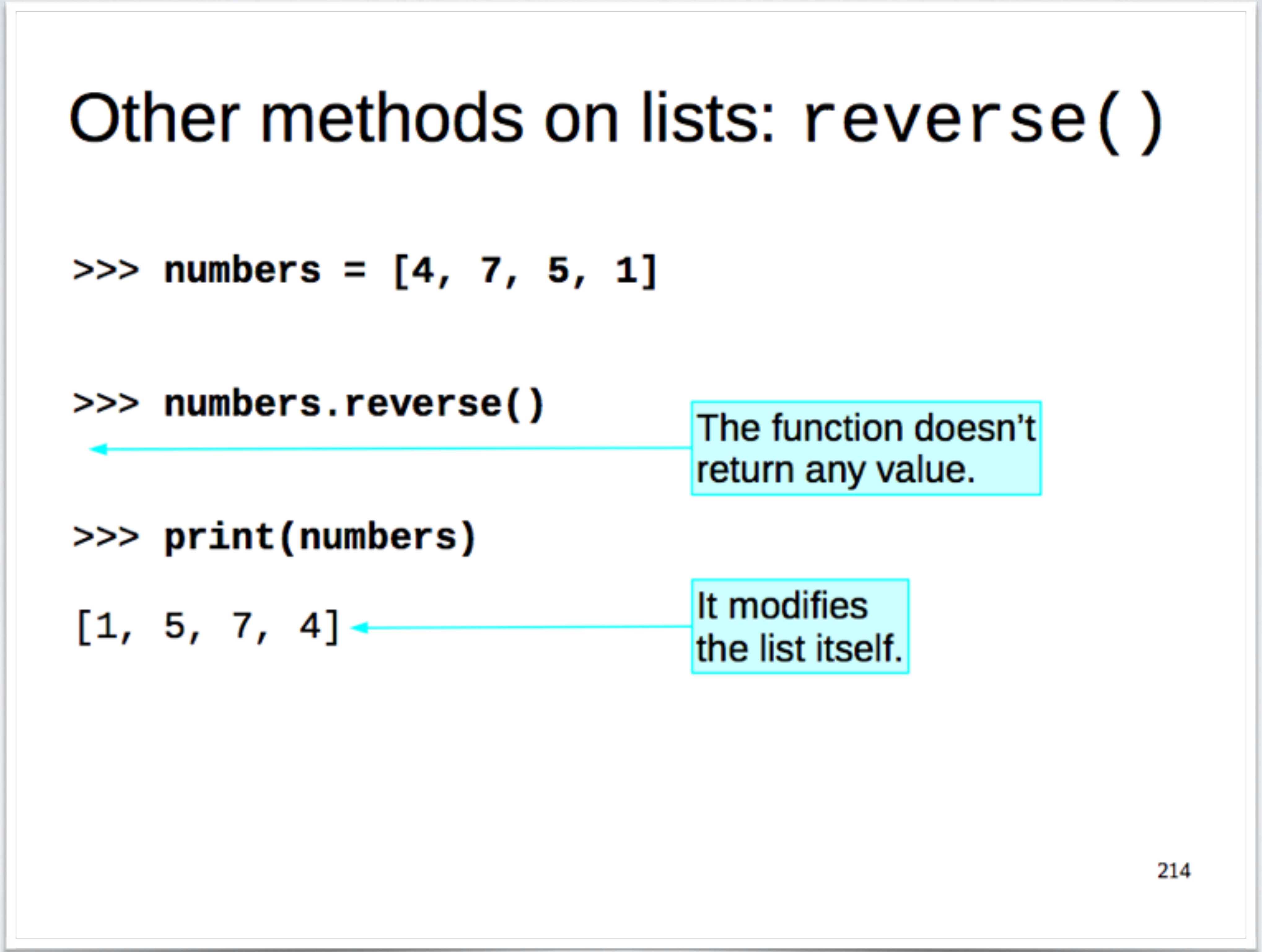
```
>>> print(primes)
[2, 3, 5, 7, 11, 13, 17, 19]

>>> primes.append(23)

>>> primes.append(29)

>>> primes.append(31)

>>> primes.append(37)

>>> print(primes)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```

The function doesn't return any value.

It modifies the list itself.

213

Python Basics



Other methods on lists: `reverse()`

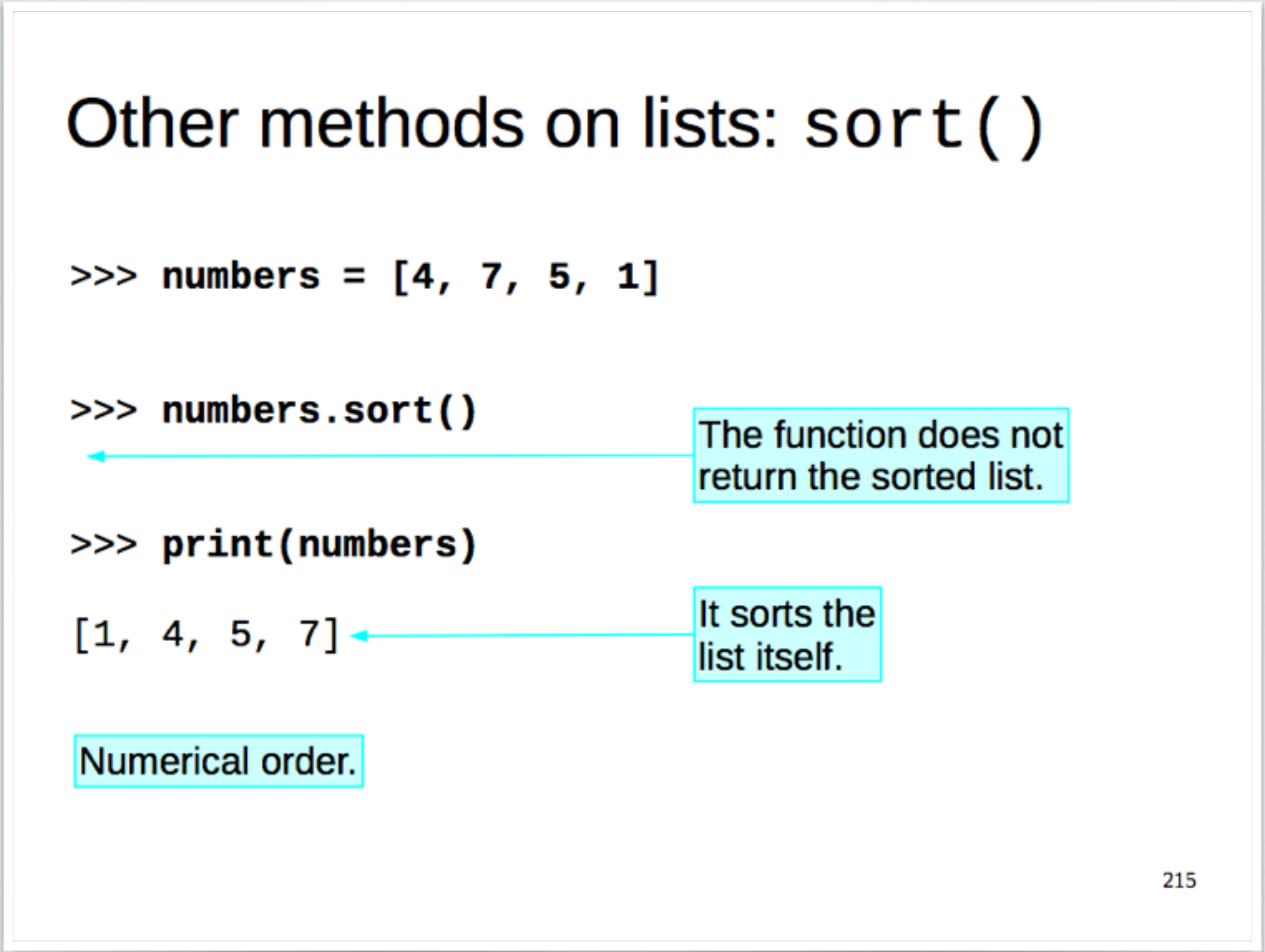
```
>>> numbers = [4, 7, 5, 1]

>>> numbers.reverse()

>>> print(numbers)
[1, 5, 7, 4]
```

The function doesn't return any value.

It modifies the list itself.

214

Python Basics



Other methods on lists: sort()

```
>>> numbers = [4, 7, 5, 1]

>>> numbers.sort()

>>> print(numbers)
[1, 4, 5, 7]
```

The function does not return the sorted list.

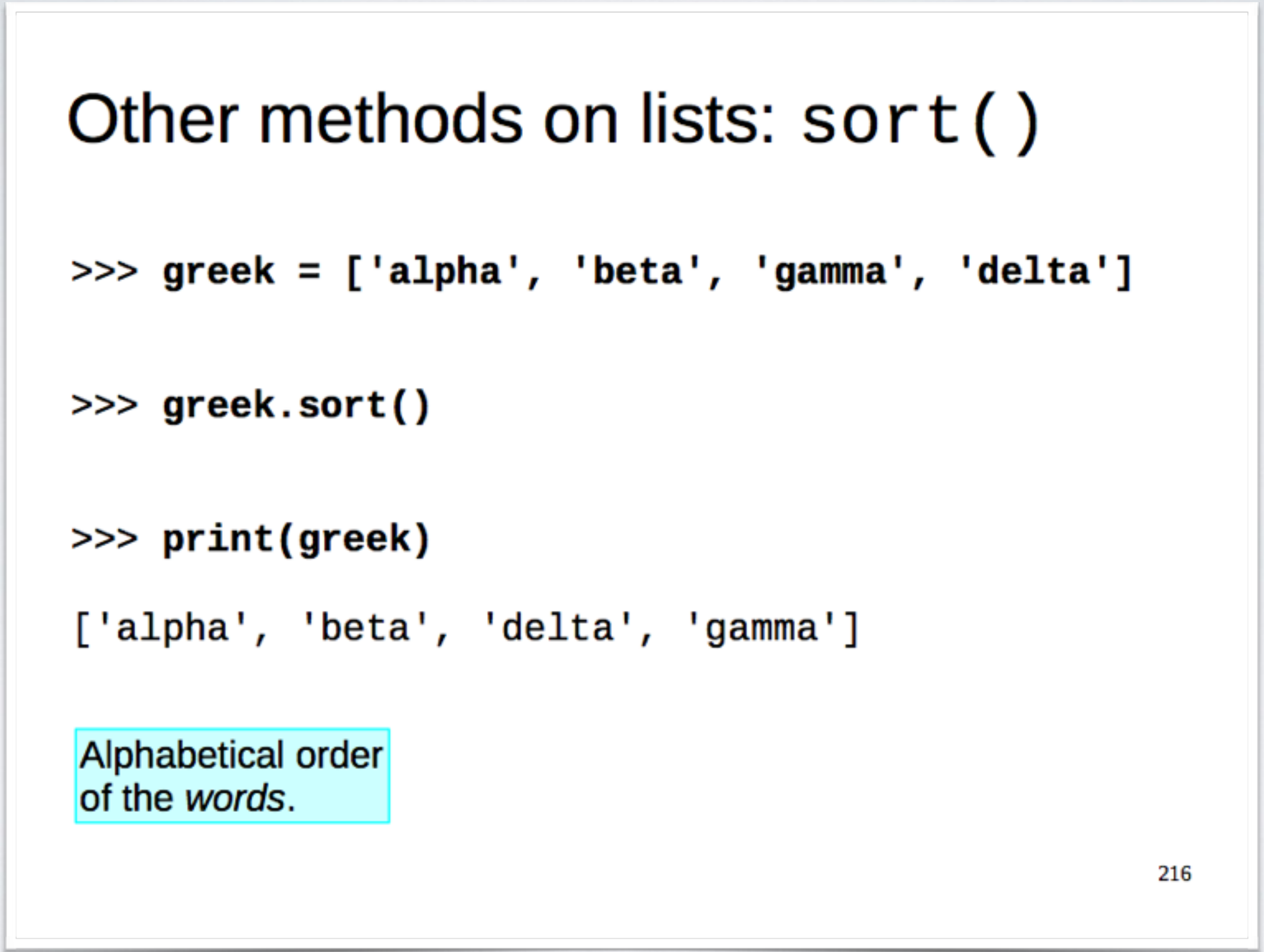
It sorts the list itself.

Numerical order.

215

Slide taken from University of Cambridge's Python AB Course

Python Basics



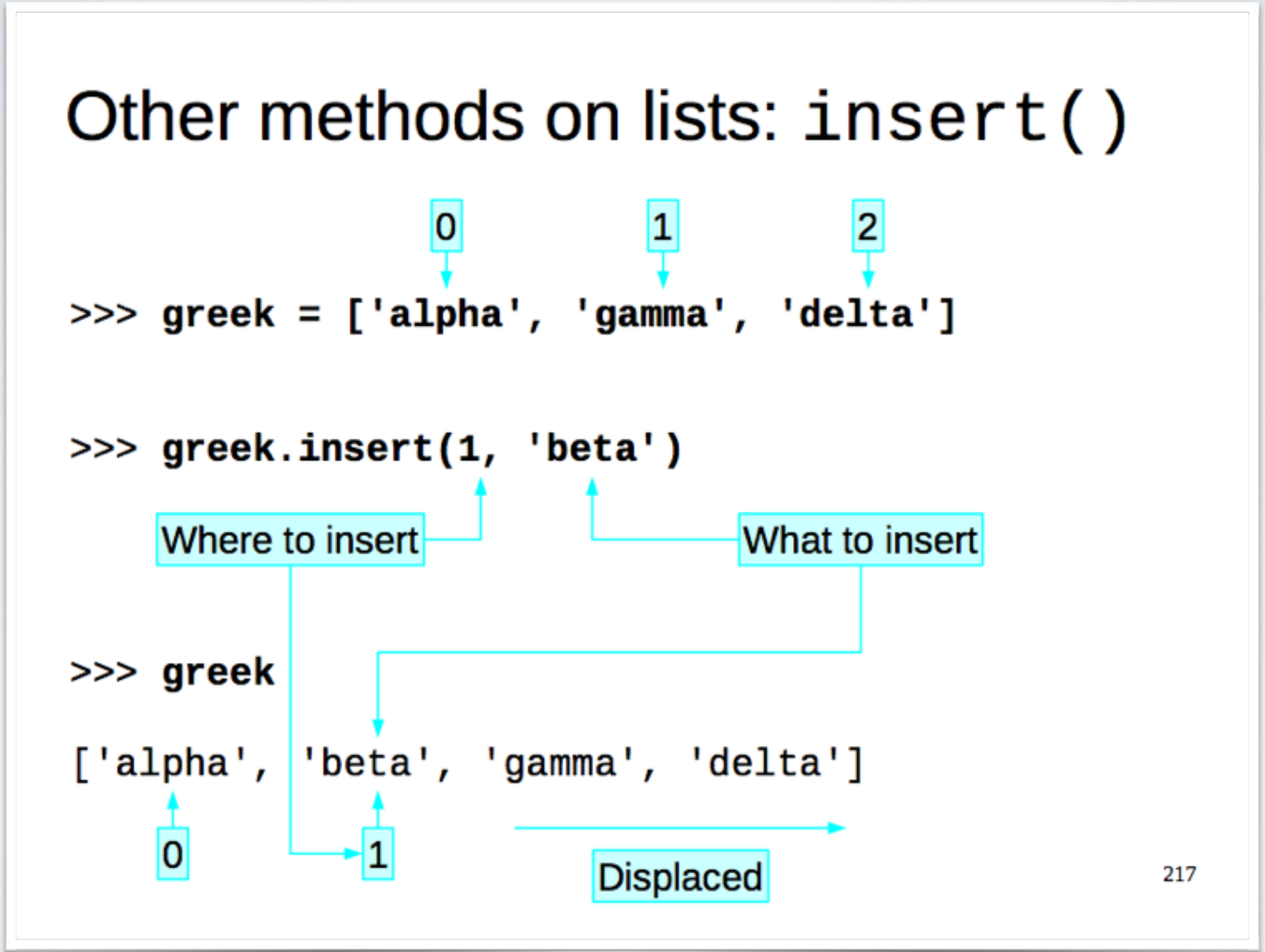
Other methods on lists: `sort()`

```
>>> greek = ['alpha', 'beta', 'gamma', 'delta']

>>> greek.sort()

>>> print(greek)
['alpha', 'beta', 'delta', 'gamma']
```

Alphabetical order
of the *words*.

216

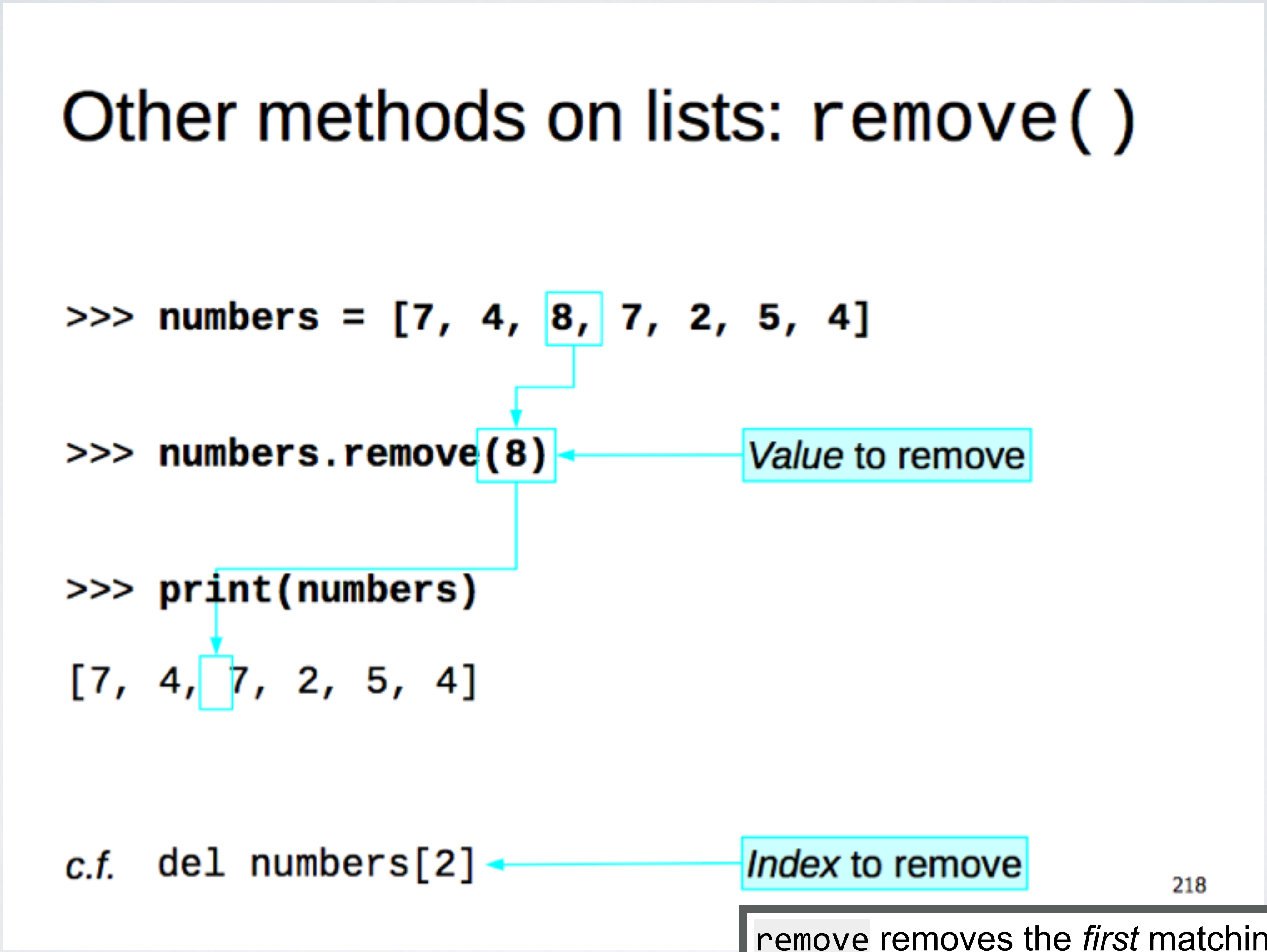
Python Basics



Other methods on lists: `insert()`

```
>>> greek = ['alpha', 'gamma', 'delta']

>>> greek.insert(1, 'beta')

>>> greek

['alpha', 'beta', 'gamma', 'delta']
```

Python Basics



Other methods on lists: `remove()`

```
>>> numbers = [7, 4, 8, 7, 2, 5, 4]

>>> numbers.remove(8)

>>> print(numbers)
[7, 4, 7, 2, 5, 4]
```

Value to remove

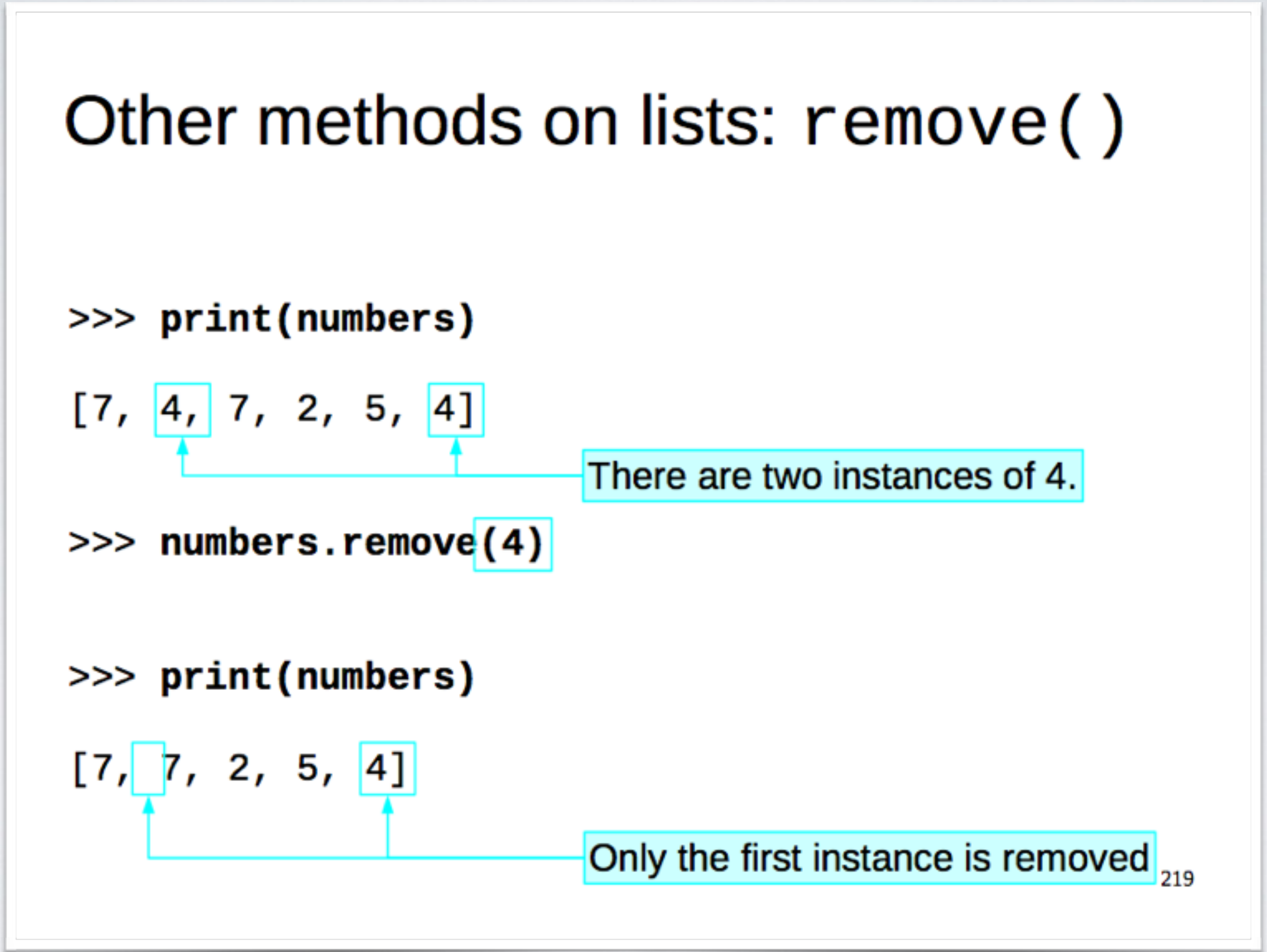
c.f. `del numbers[2]`

Index to remove

218

`remove` removes the *first* matching *value*, not a specific index like `del`

Python Basics



Other methods on lists: remove()

```
>>> print(numbers)
[7, 4, 7, 2, 5, 4]
```

There are two instances of 4.

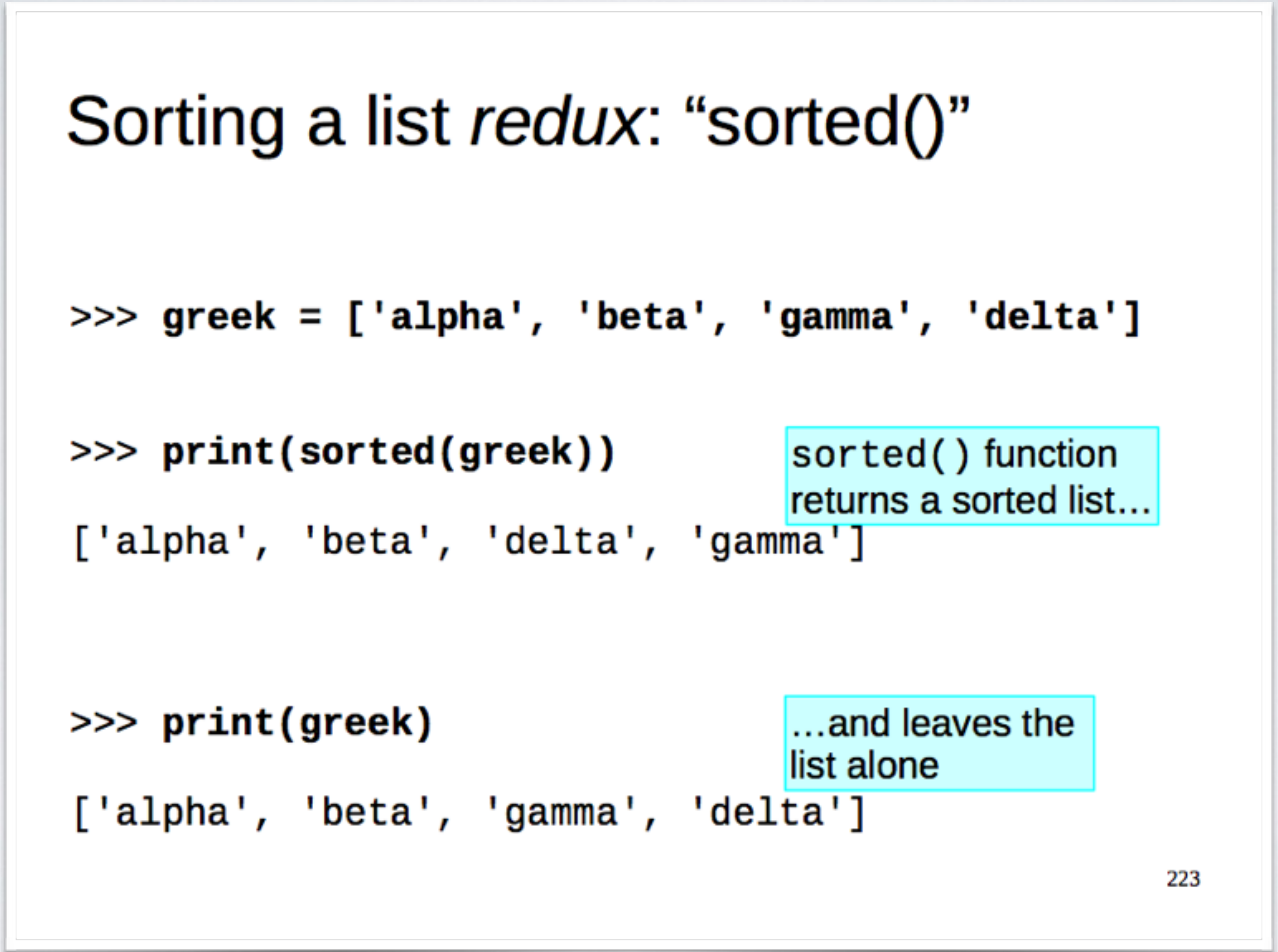
```
>>> numbers.remove(4)

>>> print(numbers)
[7, 7, 2, 5, 4]
```

Only the first instance is removed

219

Slide taken from University of Cambridge's Python AB Course

Python Basics



Sorting a list *redux*: “sorted()”

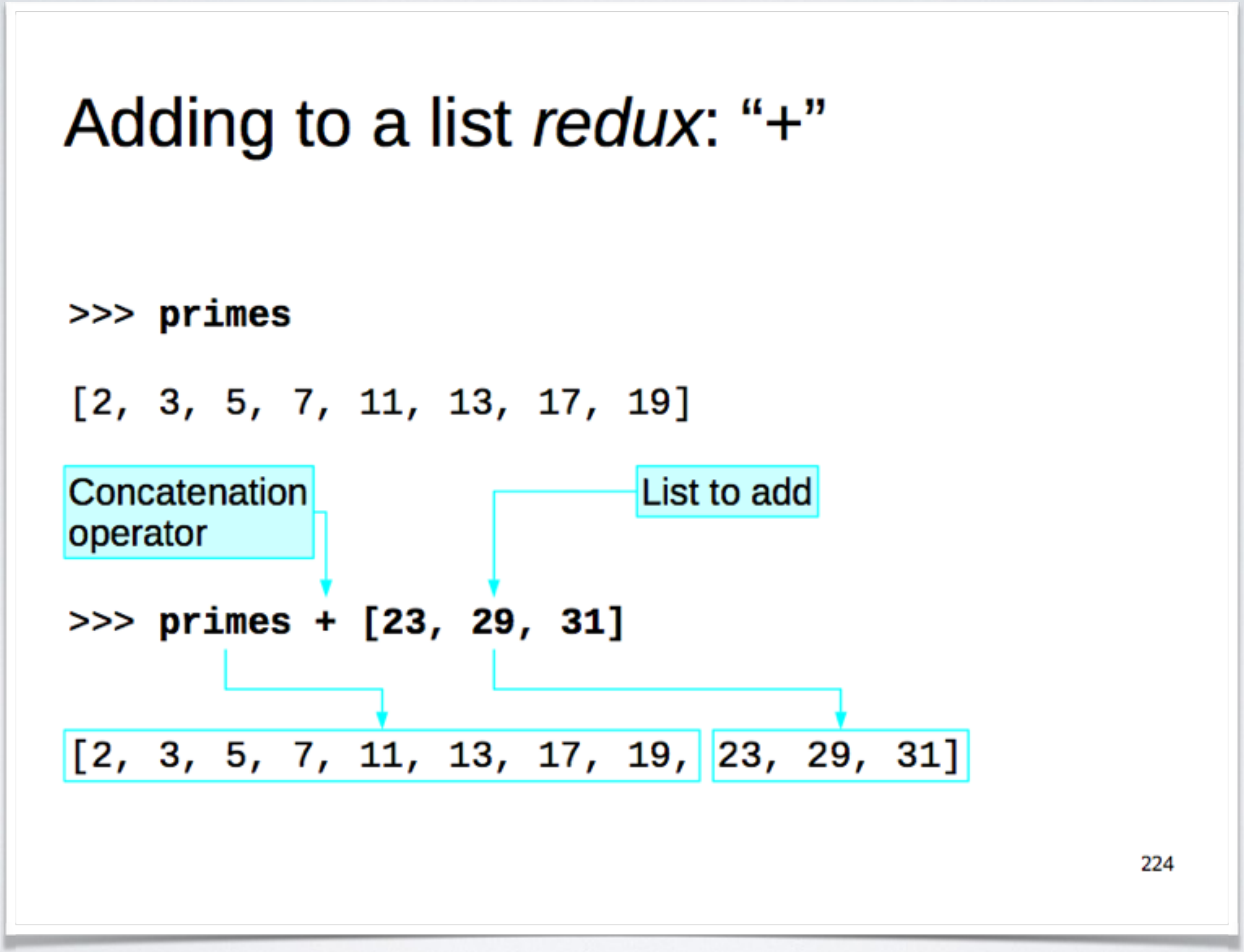
```
>>> greek = ['alpha', 'beta', 'gamma', 'delta']

>>> print(sorted(greek))

['alpha', 'beta', 'delta', 'gamma']


>>> print(greek)

['alpha', 'beta', 'gamma', 'delta']
```

`sorted()` function returns a sorted list…

…and leaves the list alone

223

Python Basics



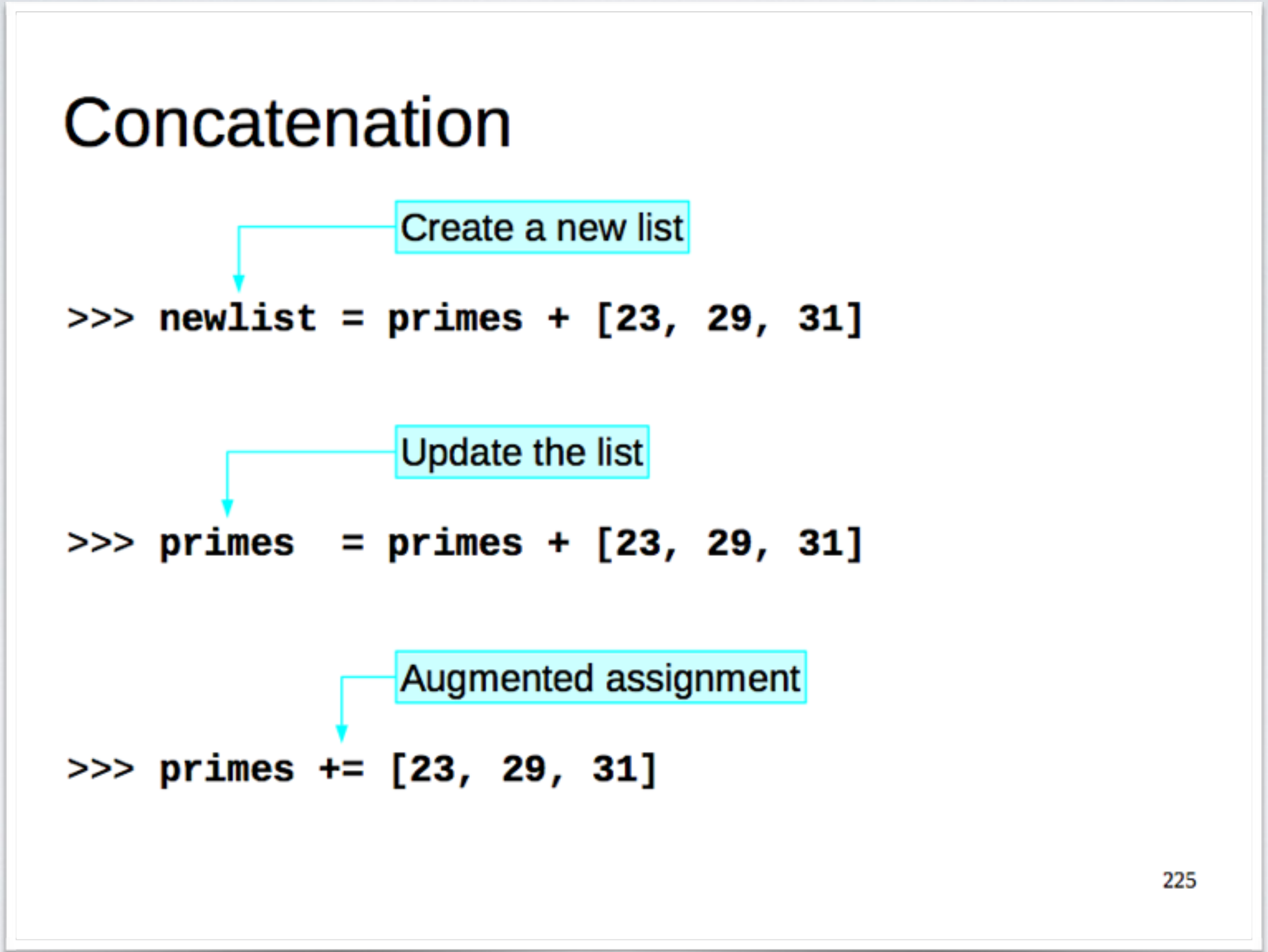
Adding to a list *redux*: "+"

```
>>> primes
[2, 3, 5, 7, 11, 13, 17, 19]
>>> primes + [23, 29, 31]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
```

Concatenation operator
List to add

224

Python Basics



Concatenation

Create a new list

```
>>> newlist = primes + [23, 29, 31]
```

Update the list

```
>>> primes  = primes + [23, 29, 31]
```

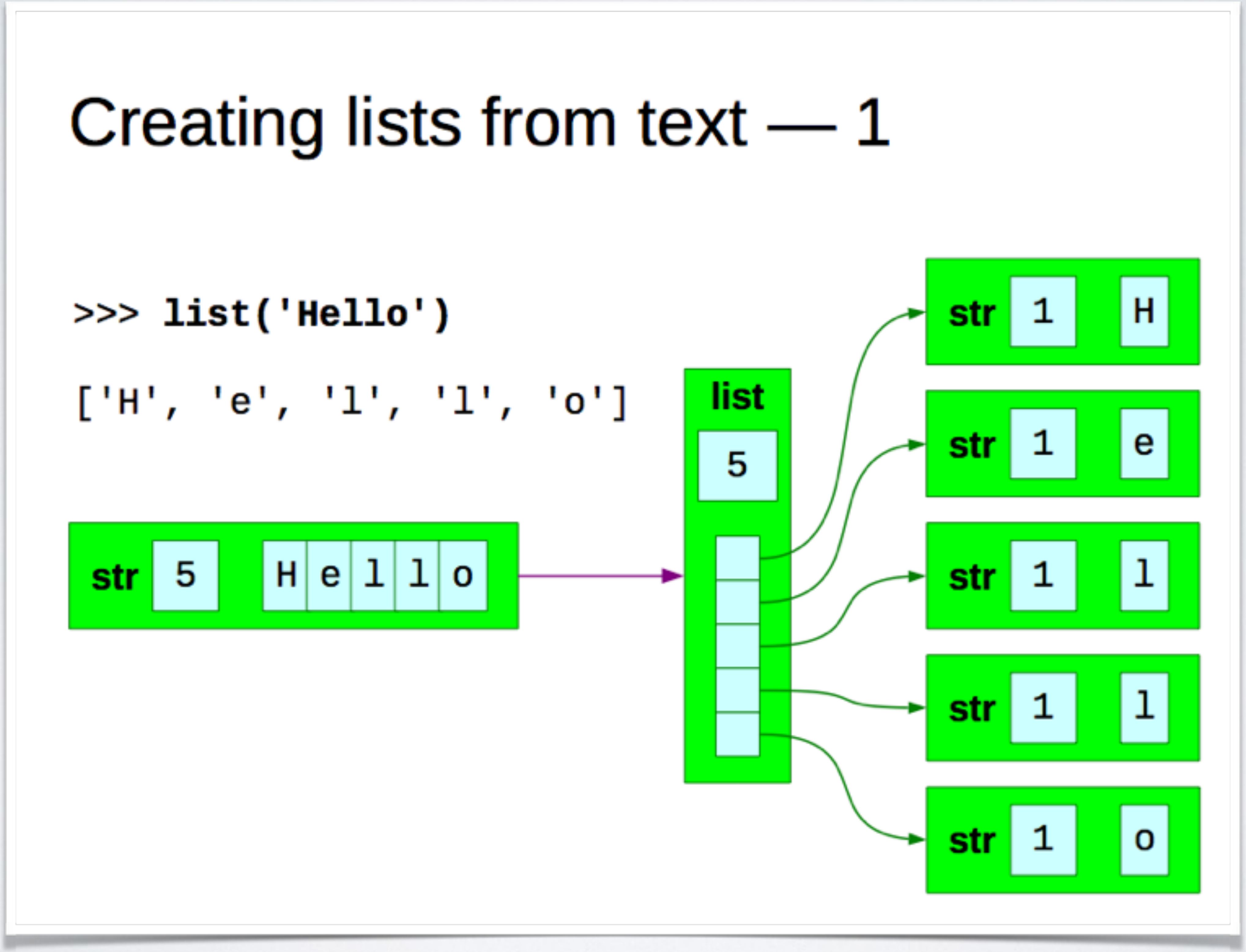
Augmented assignment

```
>>> primes += [23, 29, 31]
```

225

Slide taken from University of Cambridge's Python AB Course

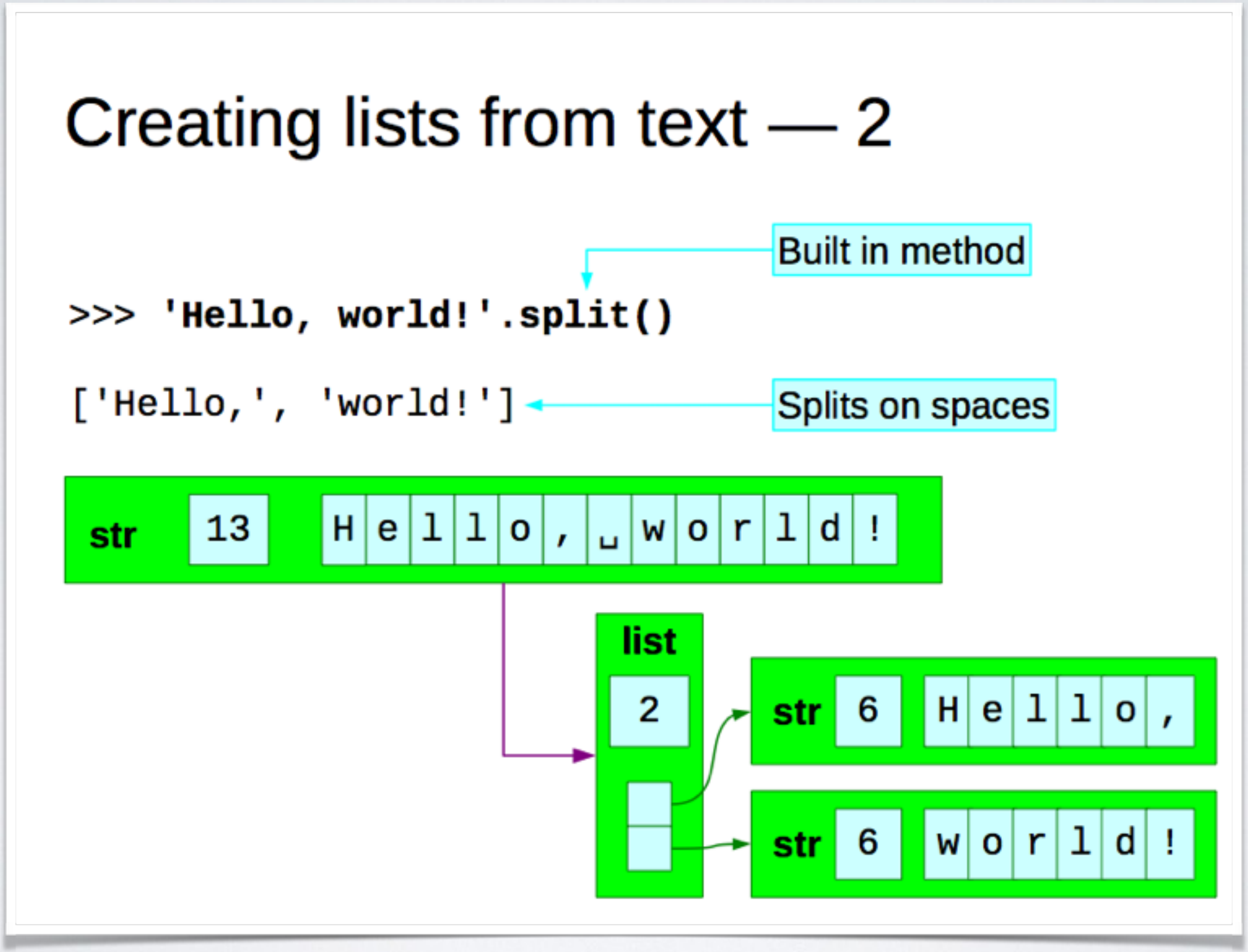
Python Basics



Creating lists from text — 1

```
>>> list('Hello')
['H', 'e', 'l', 'l', 'o']
```

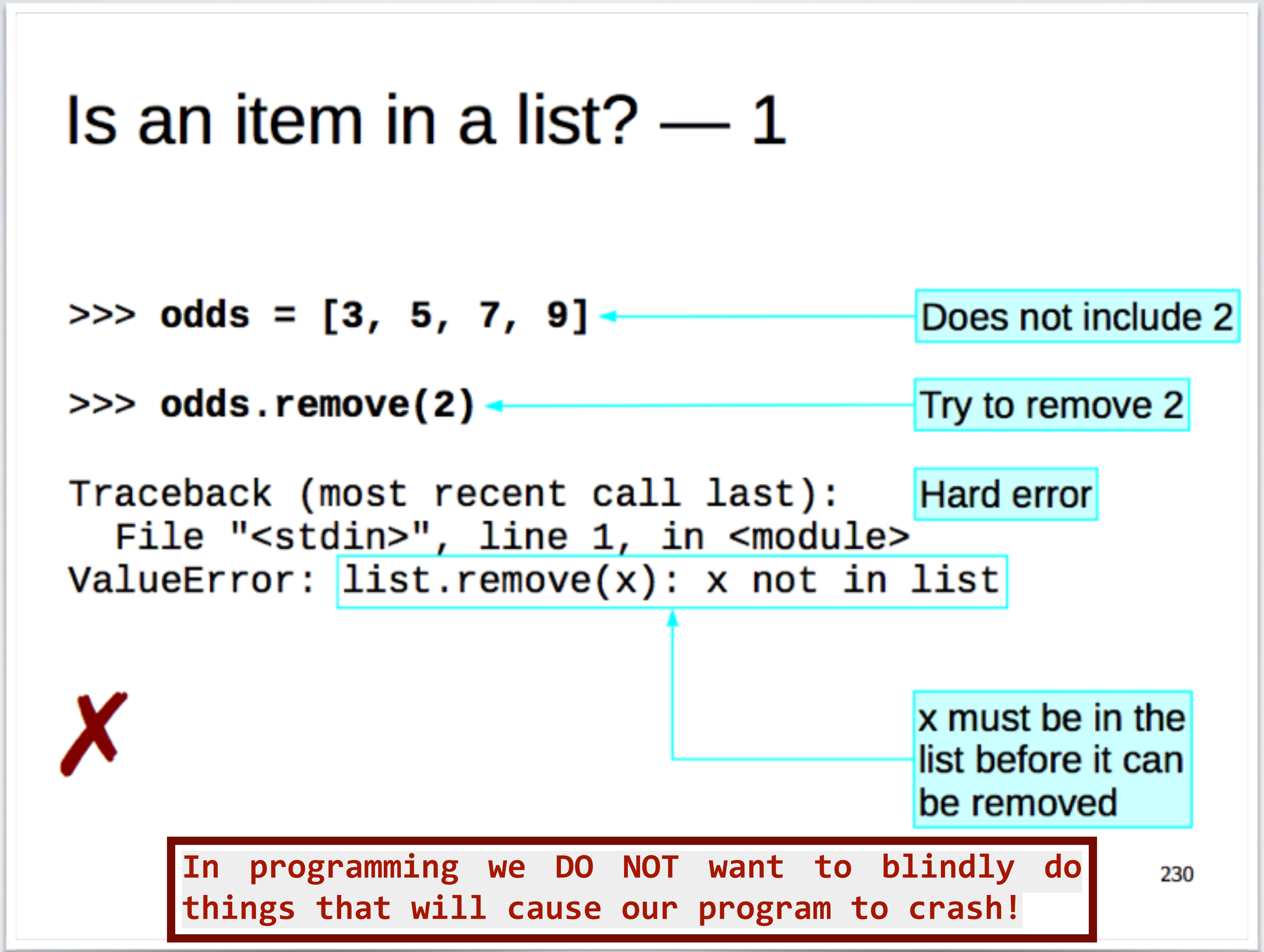
Python Basics



Creating lists from text — 2

```
>>> 'Hello, world!'.split()
['Hello,', 'world!']
```

Python Basics



Is an item in a list? — 1

```
>>> odds = [3, 5, 7, 9]

>>> odds.remove(2)

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

Does not include 2

Try to remove 2

Hard error

x must be in the list before it can be removed

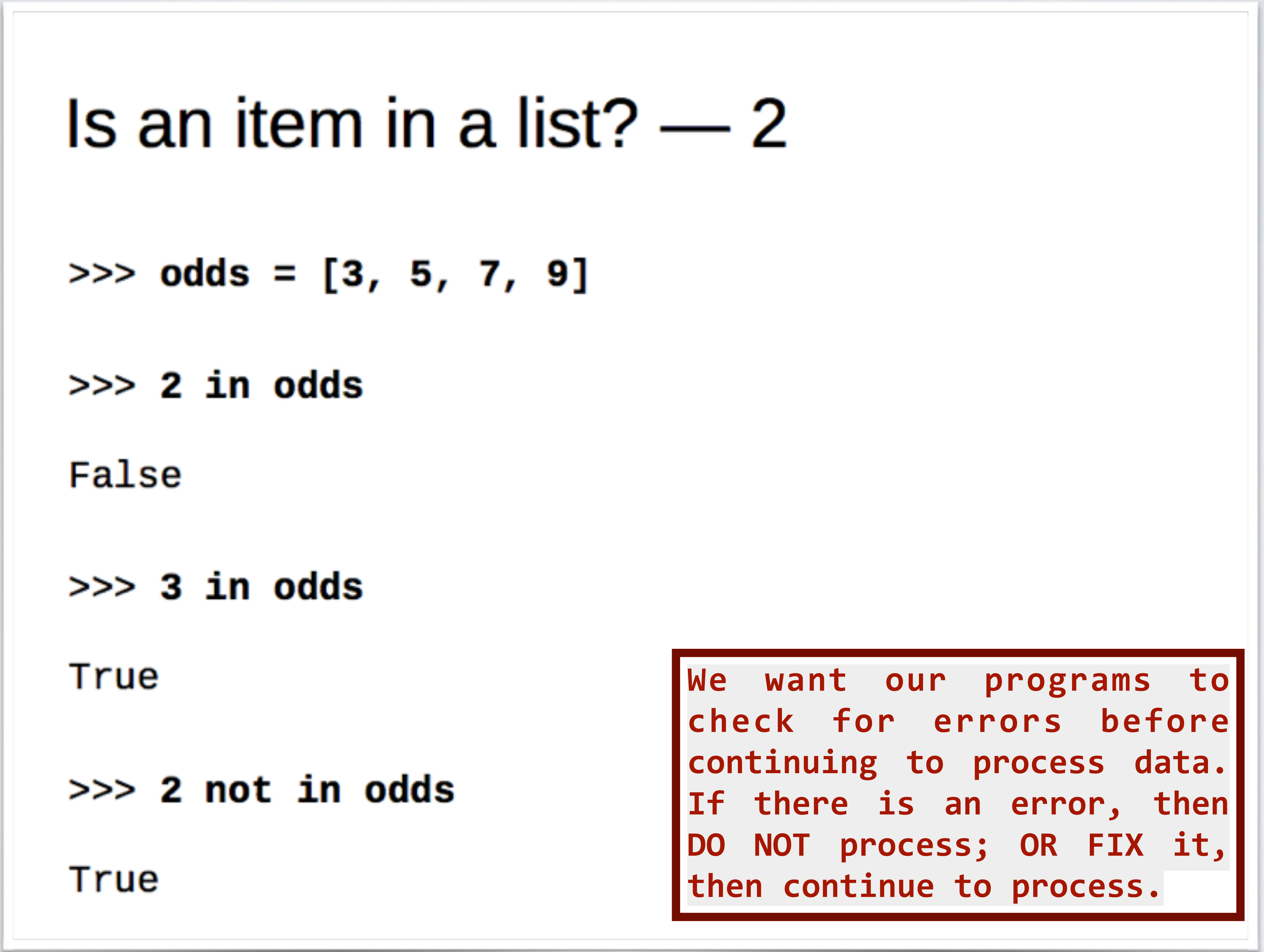
✗

In programming we DO NOT want to blindly do things that will cause our program to crash!

230

Slide taken from University of Cambridge's Python AB Course

Python Basics

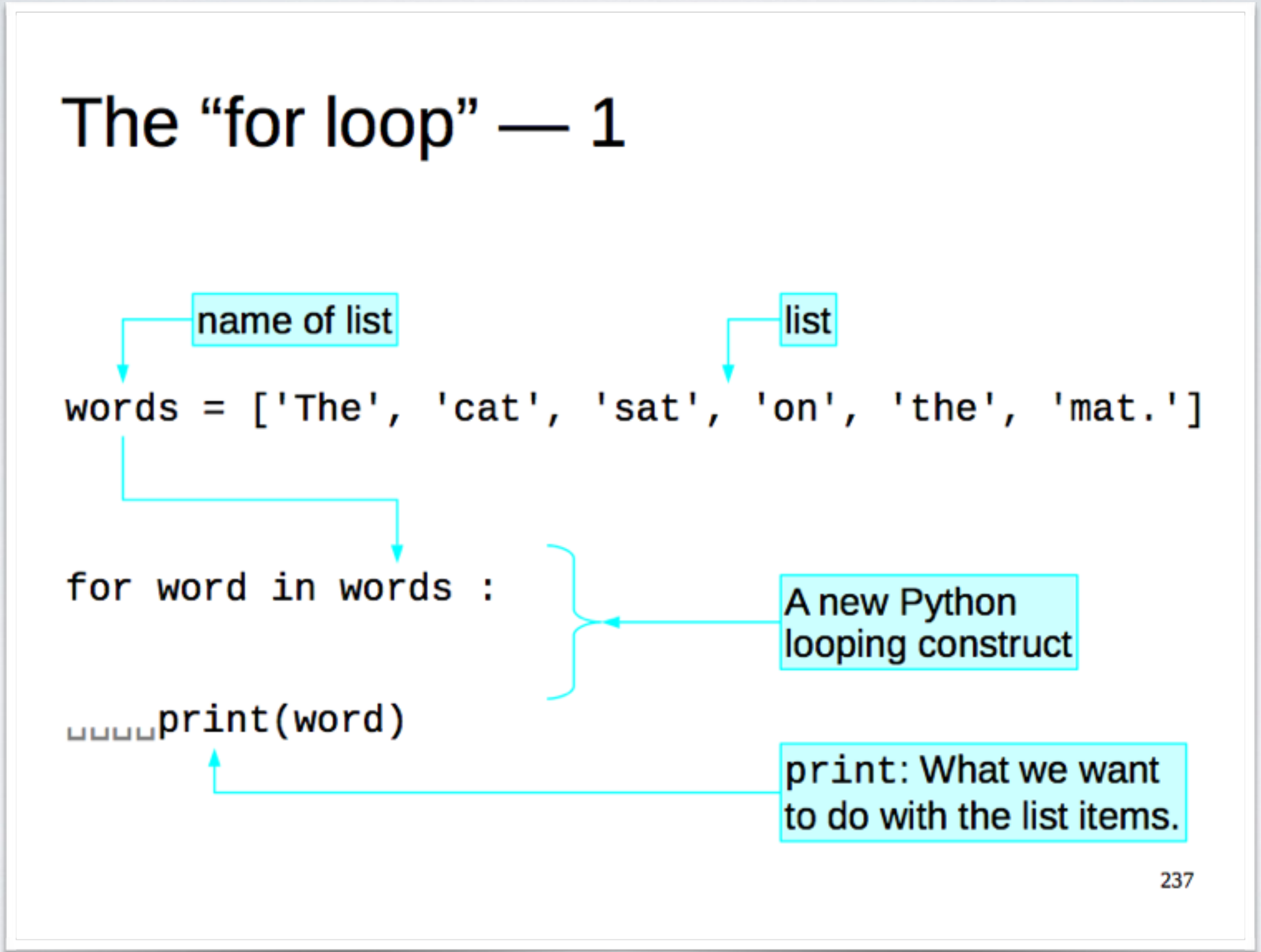


Is an item in a list? — 2

```
>>> odds = [3, 5, 7, 9]

>>> 2 in odds

False

>>> 3 in odds

True

>>> 2 not in odds

True
```

We want our programs to check for errors before continuing to process data. If there is an error, then DO NOT process; OR FIX it, then continue to process.

Python Basics

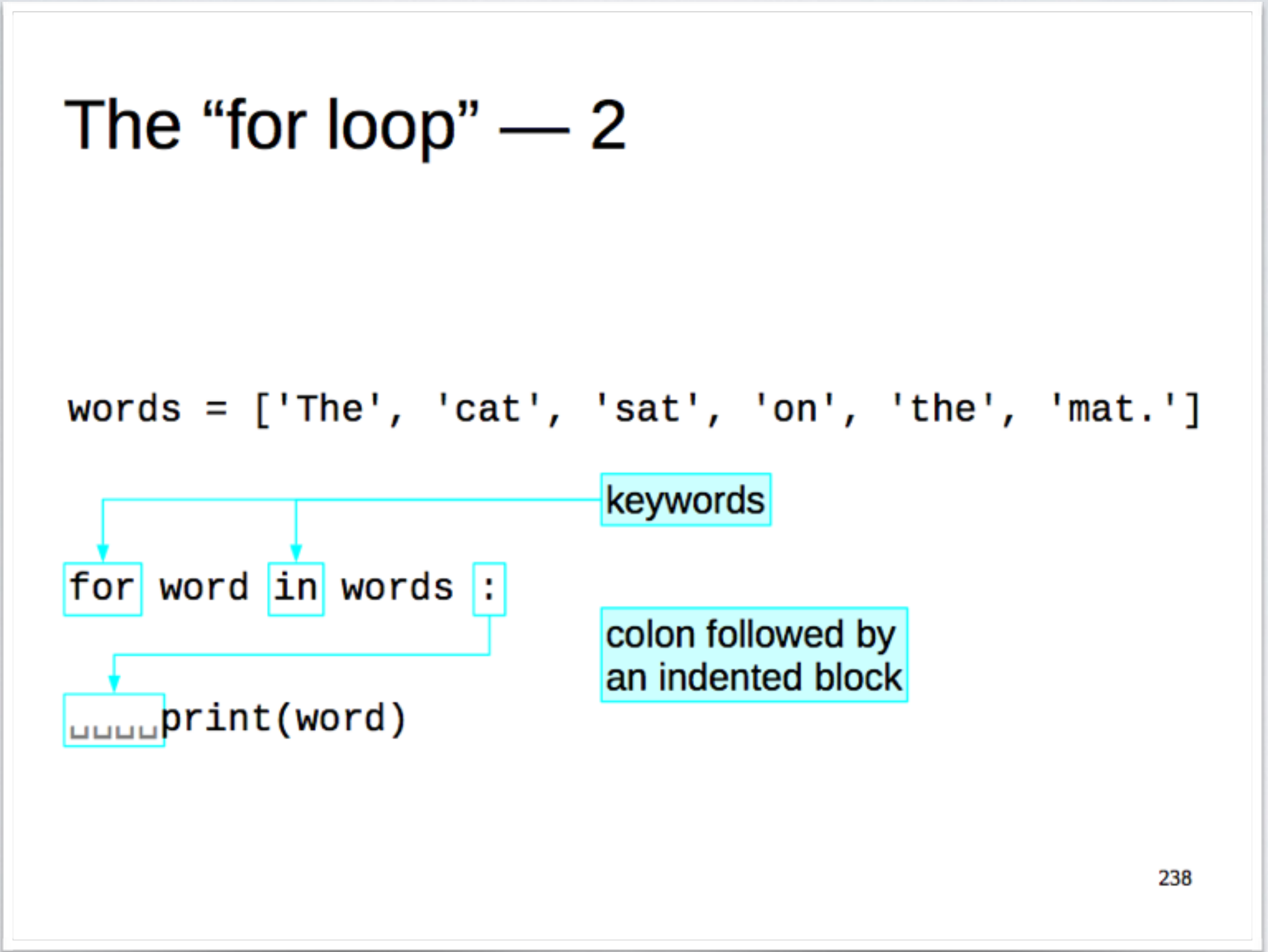


The “for loop” — 1

```
words = ['The', 'cat', 'sat', 'on', 'the', 'mat.']

for word in words :

␣␣␣␣print(word)
```

name of list
list
A new Python looping construct
print: What we want to do with the list items.
237

Python Basics

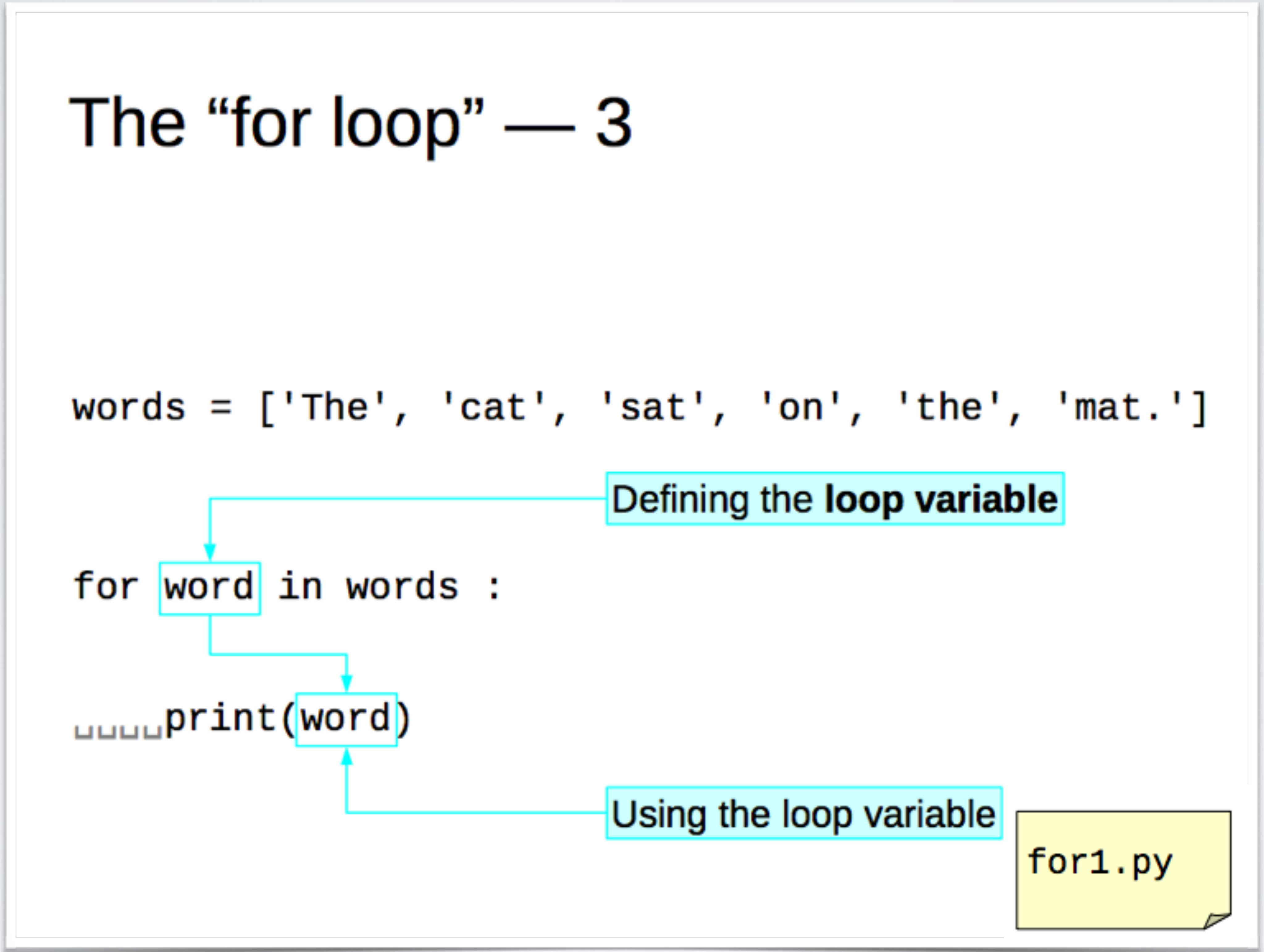


The “for loop” — 2

```
words = ['The', 'cat', 'sat', 'on', 'the', 'mat.']

for word in words :
␣␣␣␣print(word)
```

keywords
colon followed by an indented block
238

Python Basics

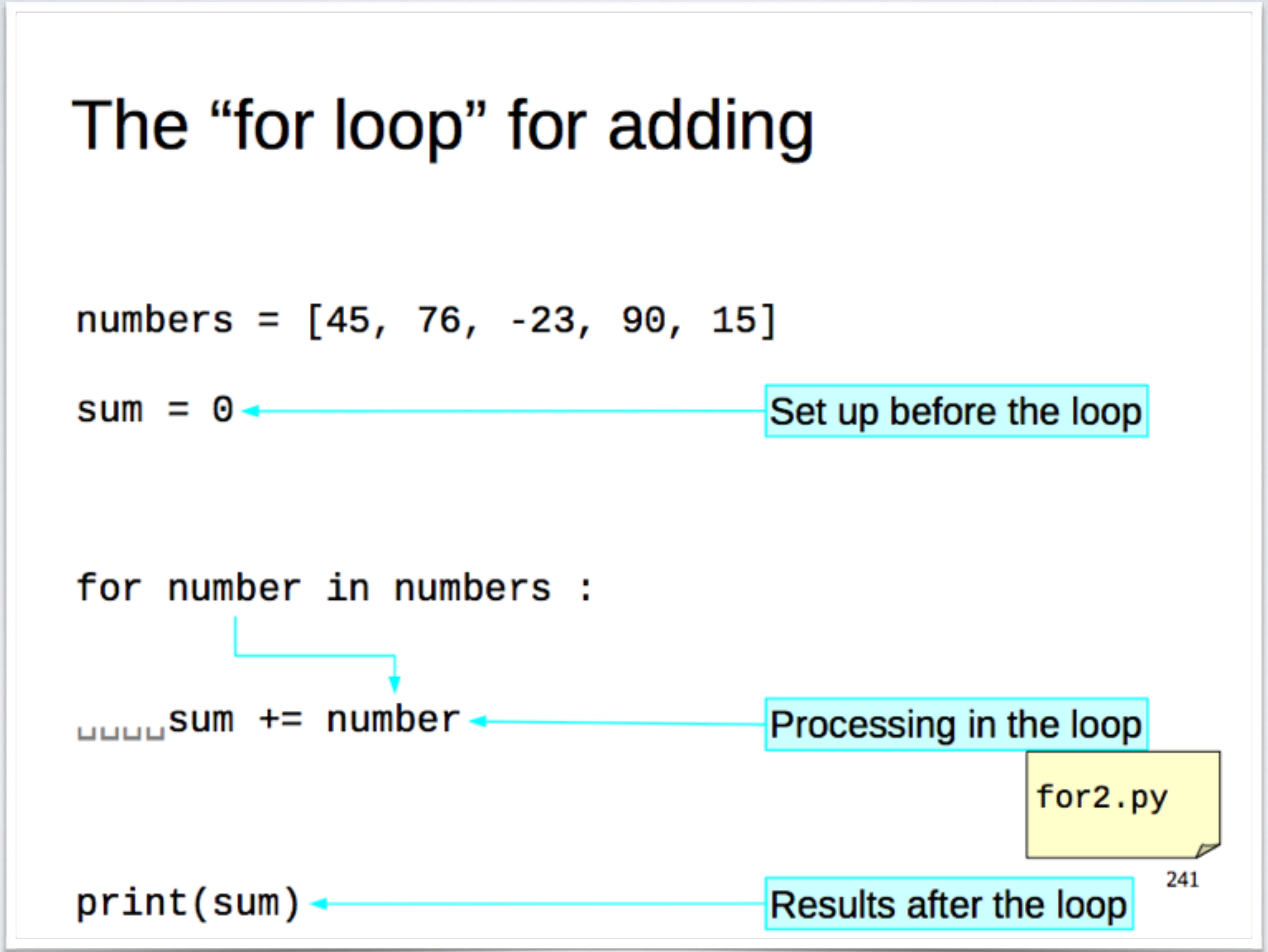


The “for loop” — 3

```
words = ['The', 'cat', 'sat', 'on', 'the', 'mat.']

for word in words :
␣␣␣␣print(word)
```

Defining the **loop variable**
Using the loop variable
for1.py

Python Basics

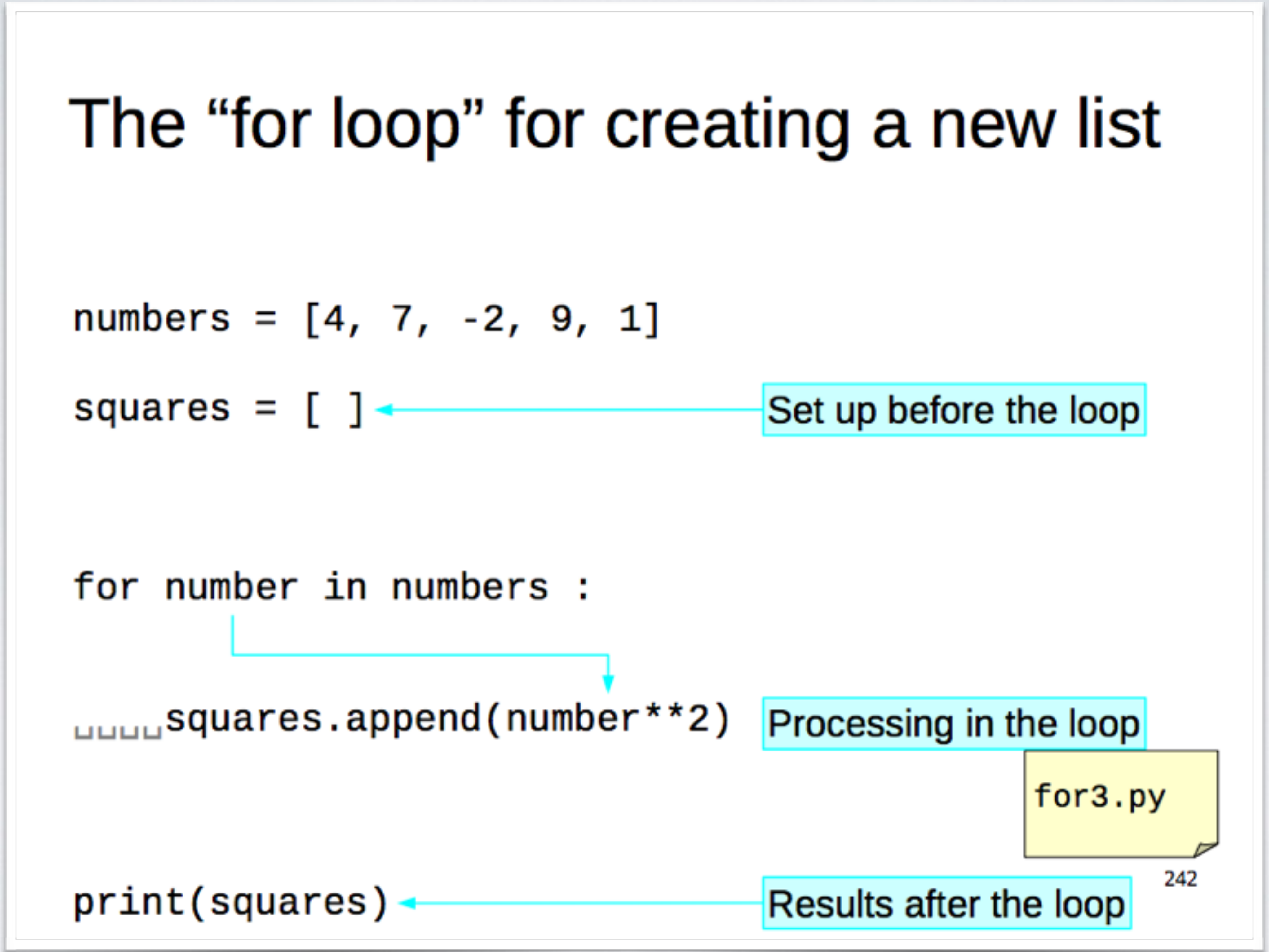


The “for loop” for adding

```
numbers = [45, 76, -23, 90, 15]

sum = 0

for number in numbers :

␣␣␣␣sum += number

print(sum)
```

Set up before the loop
Processing in the loop
for2.py
241
Results after the loop

Python Basics

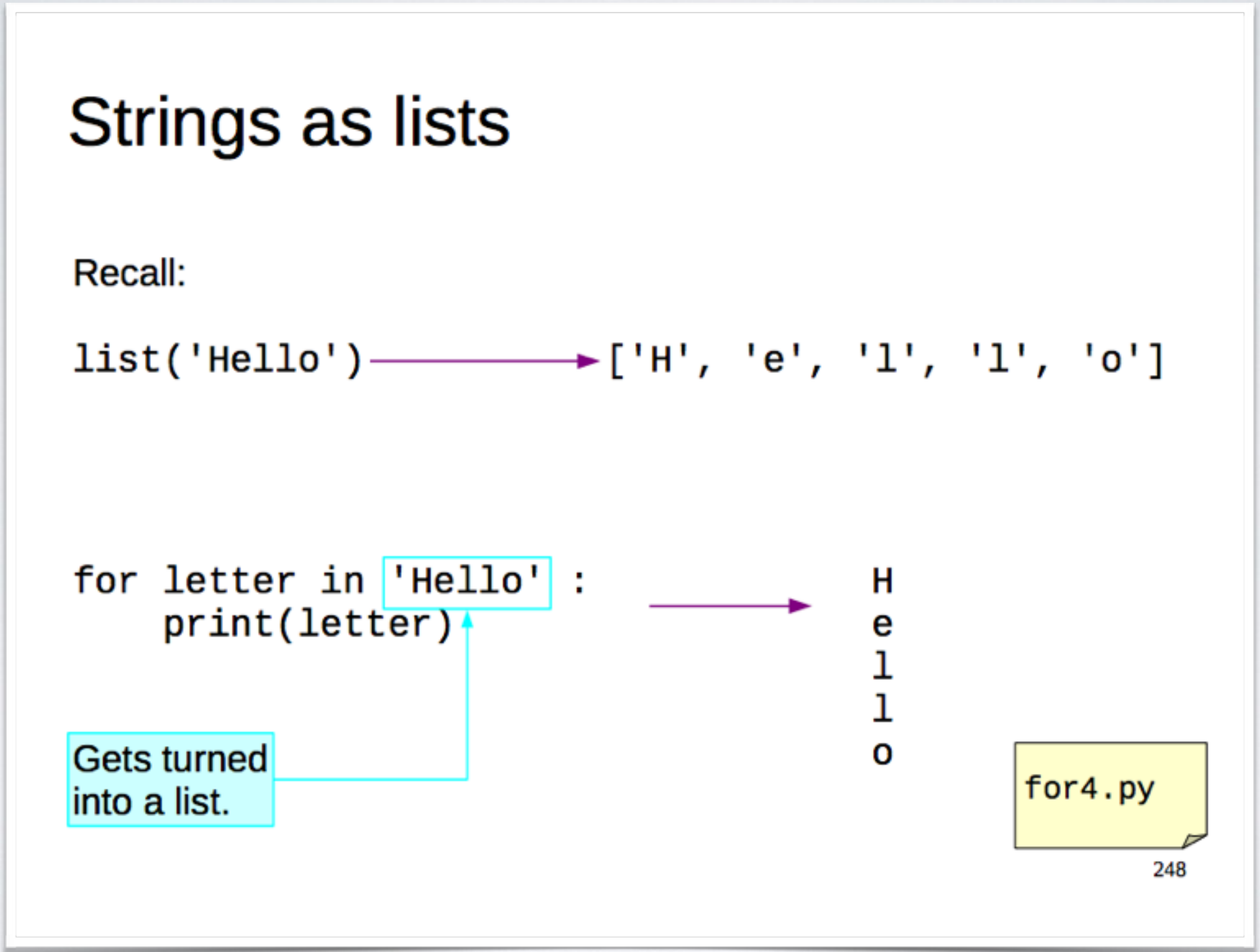


The “for loop” for creating a new list

```
numbers = [4, 7, -2, 9, 1]

squares = [ ]

for number in numbers :

␣␣␣␣squares.append(number**2)

print(squares)
```

Set up before the loop
Processing in the loop
for3.py
242
Results after the loop

Python Basics



Strings as lists

Recall:

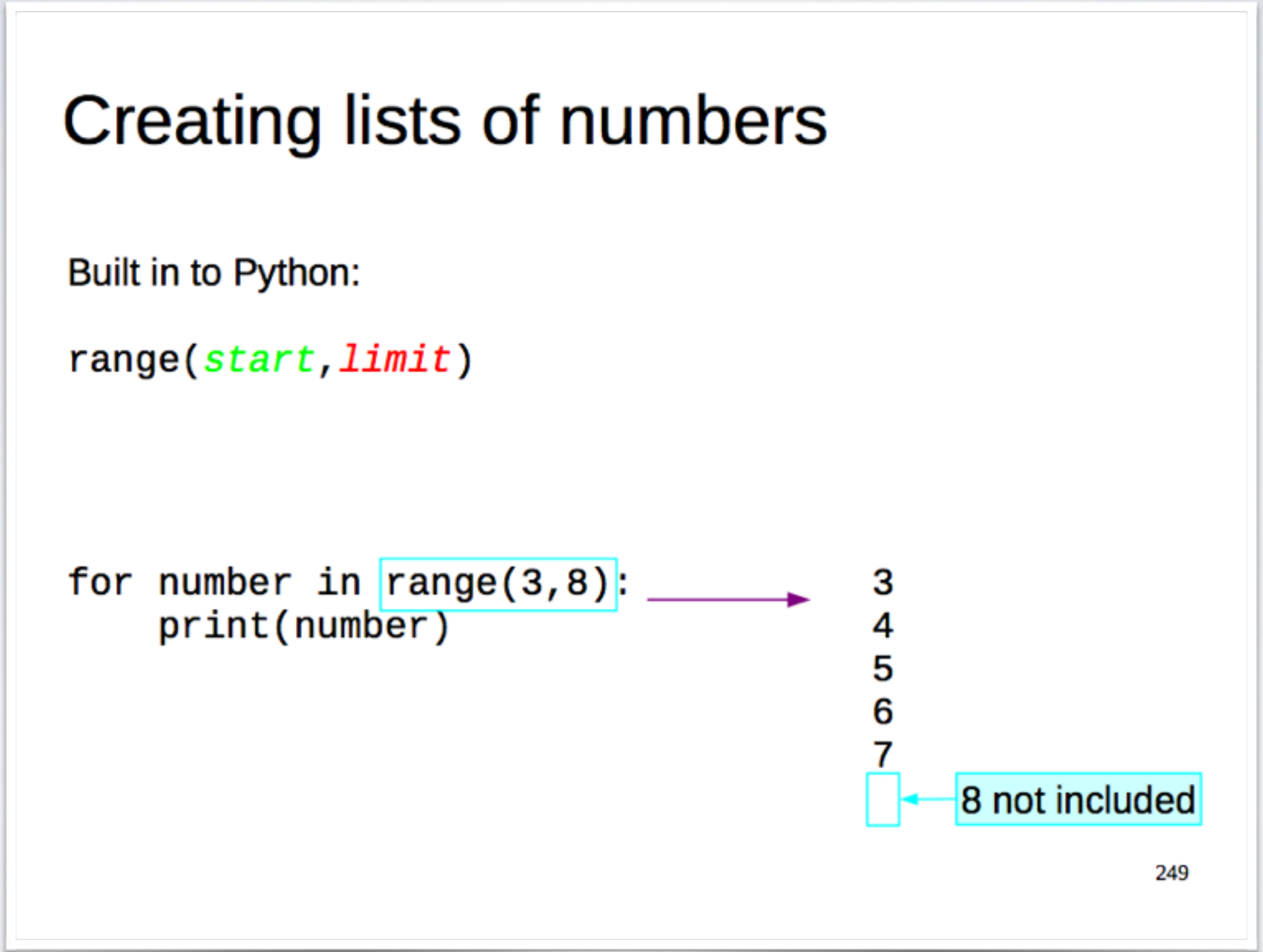
```
list('Hello') → ['H', 'e', 'l', 'l', 'o']
```

```
for letter in 'Hello' :
    print(letter)
```

```
H
e
l
l
o
```

Gets turned into a list.
for4.py
248

Python Basics



Creating lists of numbers

Built in to Python:

```
range(start,limit)
```

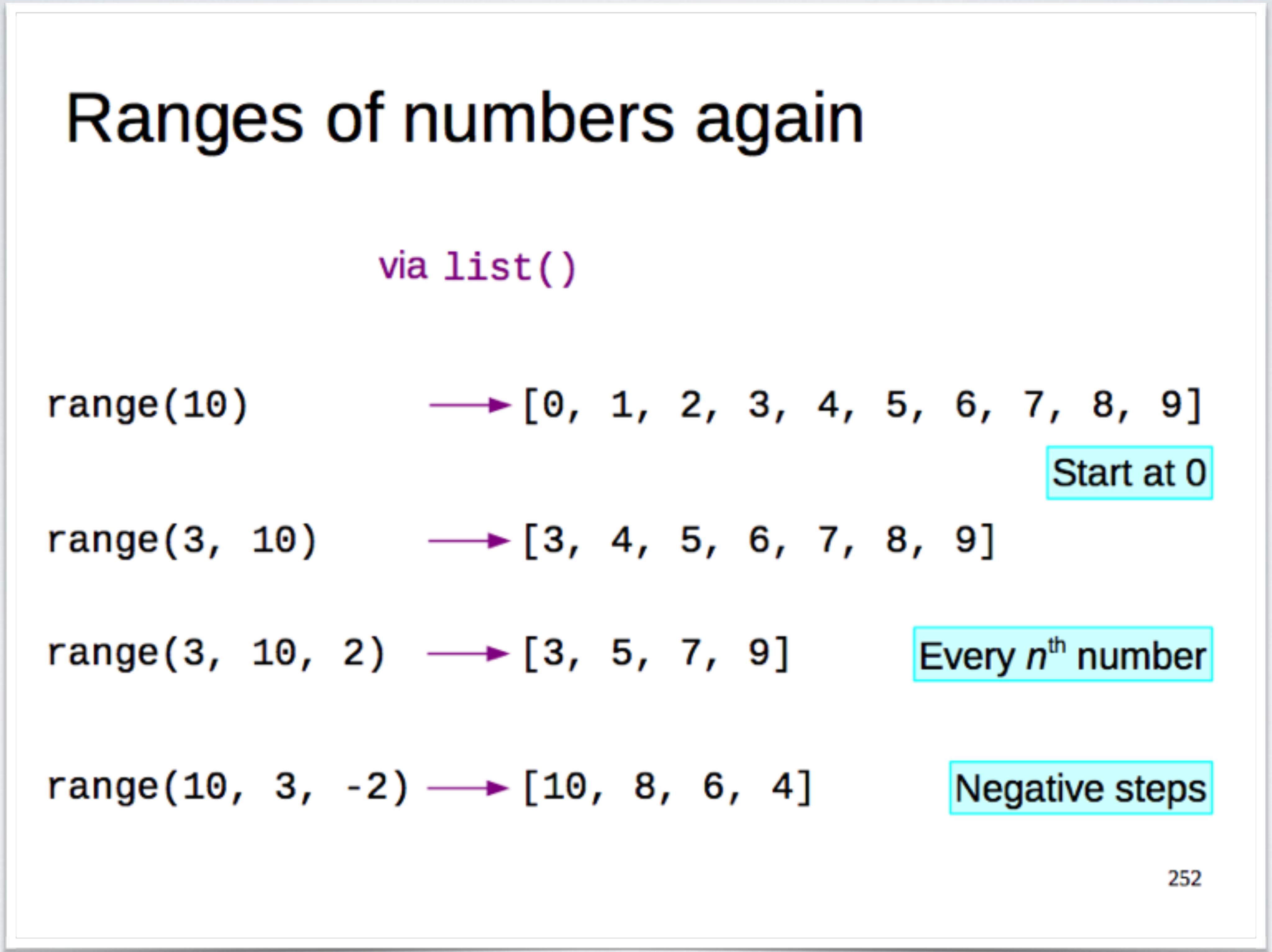
```
for number in range(3,8):
    print(number)
```

```
3
4
5
6
7
```

8 not included

249

Python Basics



Ranges of numbers again

via `list()`

```
range(10)          → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Start at 0

```
range(3, 10)       → [3, 4, 5, 6, 7, 8, 9]

range(3, 10, 2)    → [3, 5, 7, 9]
```

Every *n*th number

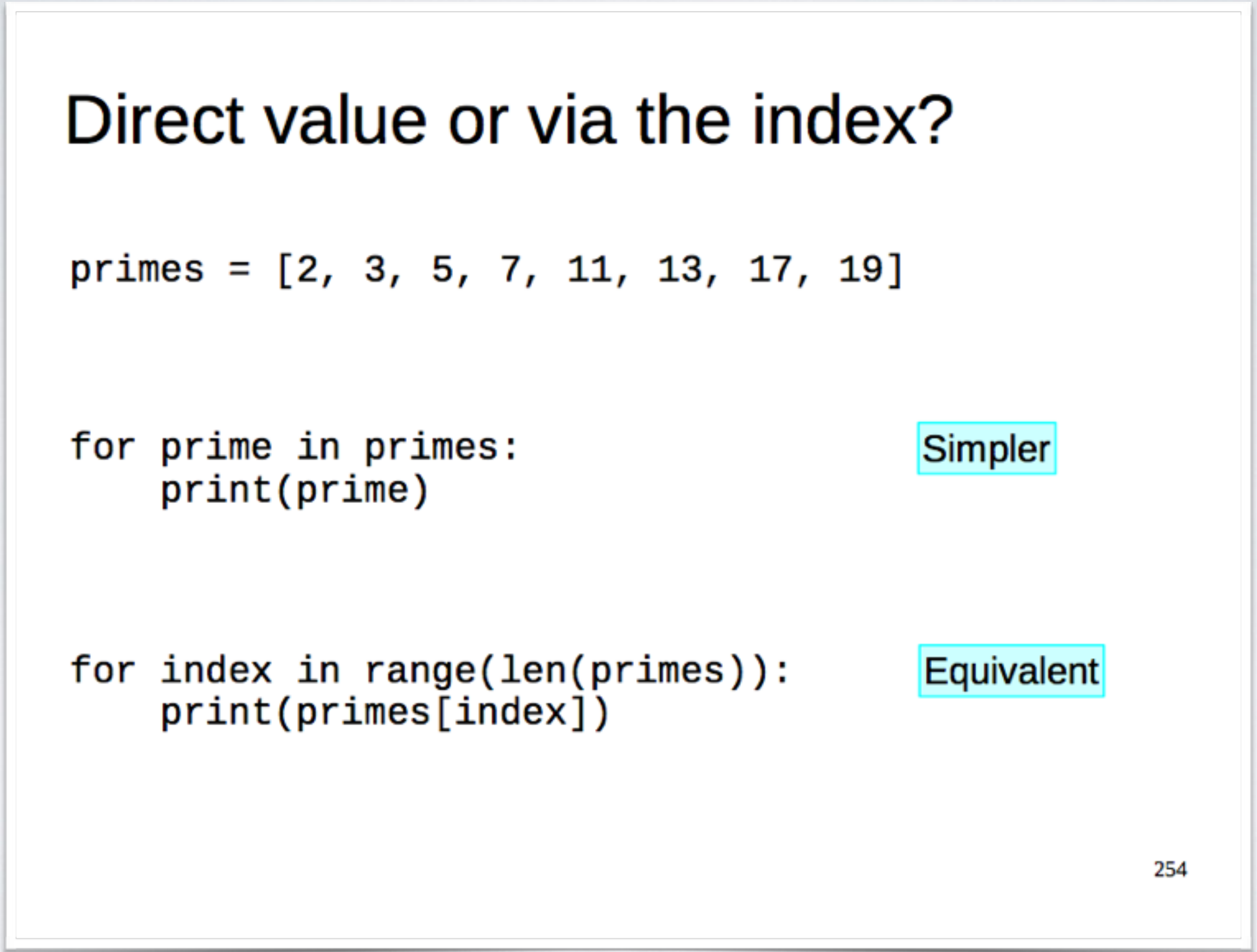
```
range(10, 3, -2)   → [10, 8, 6, 4]
```

Negative steps

252

Slide taken from University of Cambridge's Python AB Course

Python Basics



Direct value or via the index?

```
primes = [2, 3, 5, 7, 11, 13, 17, 19]
```

```
for prime in primes:
    print(prime)
```

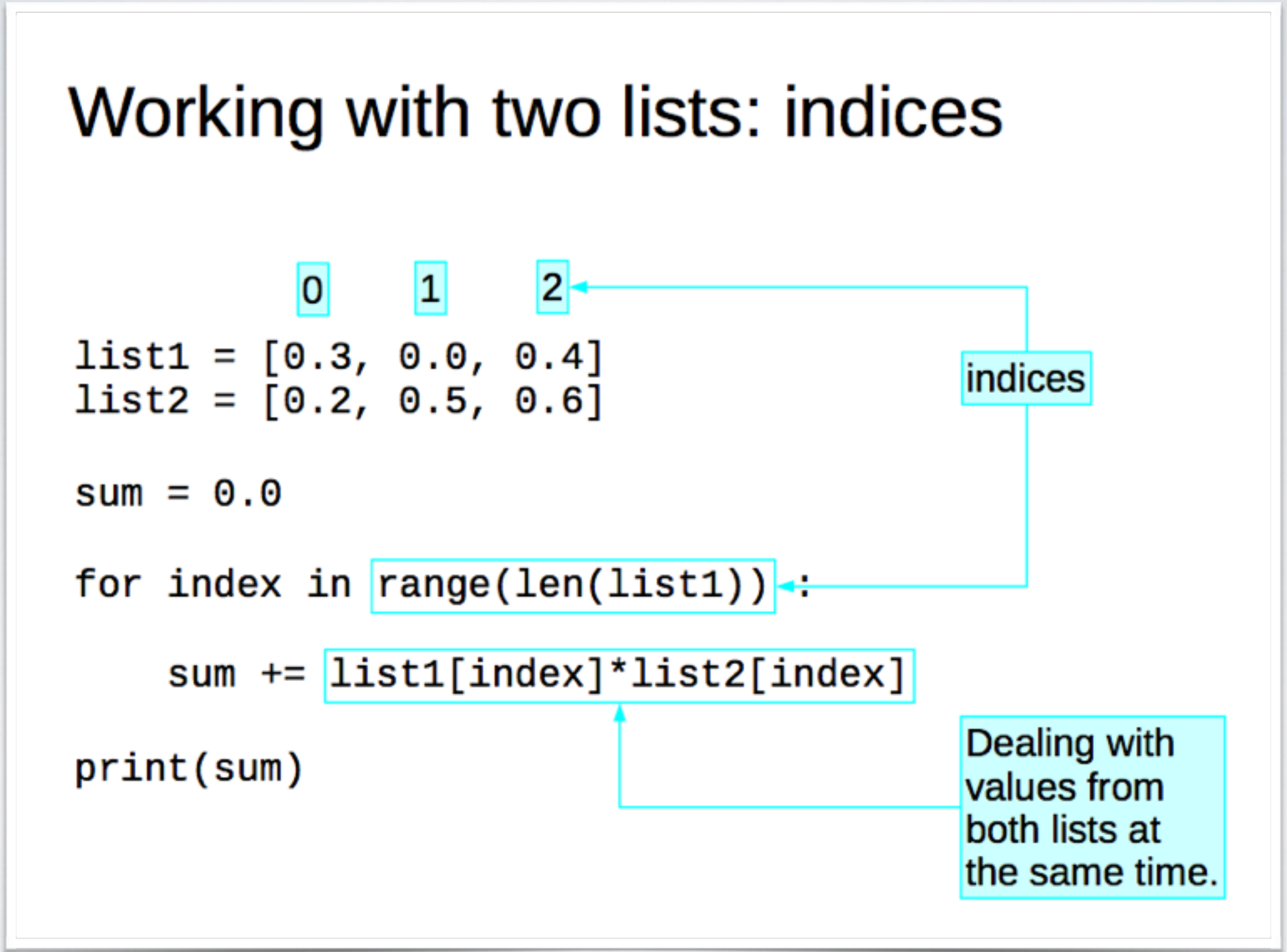
Simpler

```
for index in range(len(primes)):
    print(primes[index])
```

Equivalent

254

Python Basics



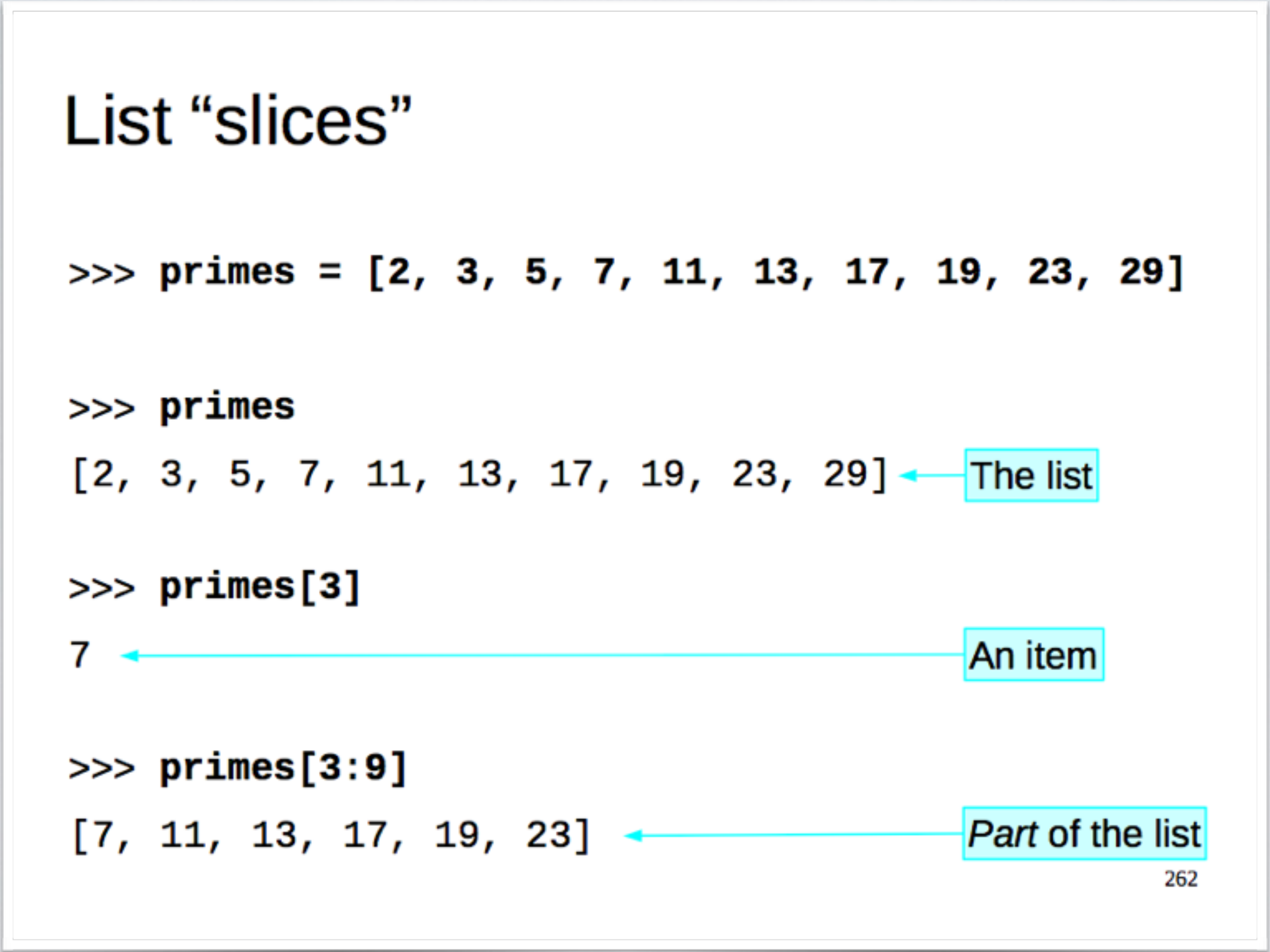
Working with two lists: indices

0
1
2
indices
Dealing with values from both lists at the same time.

```
list1 = [0.3, 0.0, 0.4]
list2 = [0.2, 0.5, 0.6]

sum = 0.0

for index in range(len(list1)) :

    sum += list1[index]*list2[index]

print(sum)
```

Slide taken from University of Cambridge's Python AB Course

Python Basics

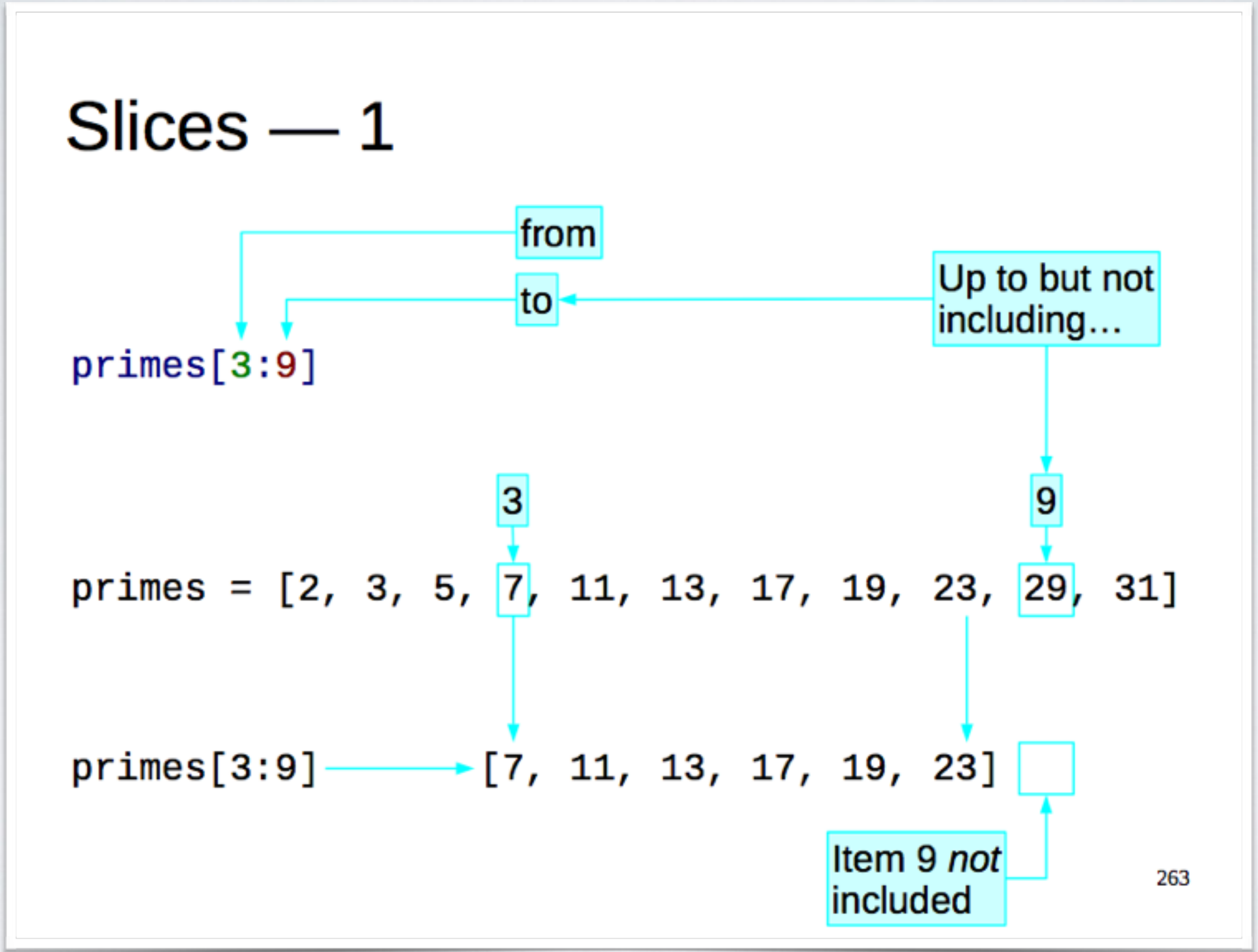


List "slices"

```
>>> primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

>>> primes
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

>>> primes[3]
7

>>> primes[3:9]
[7, 11, 13, 17, 19, 23]
```

The list
An item
Part of the list
262

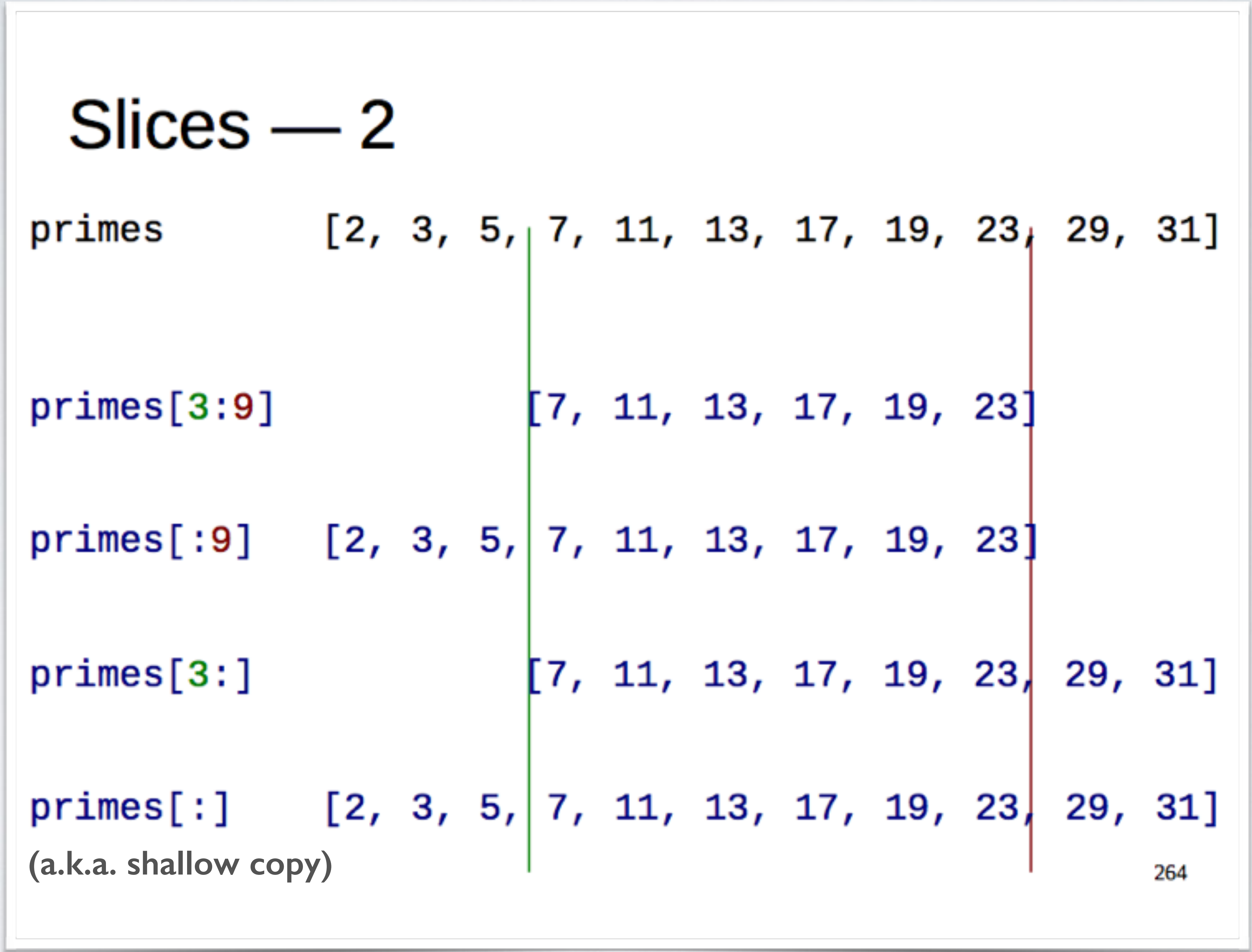
Python Basics



Slices — 1

```
primes[3:9]

primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]

primes[3:9] → [7, 11, 13, 17, 19, 23]
```

Python Basics



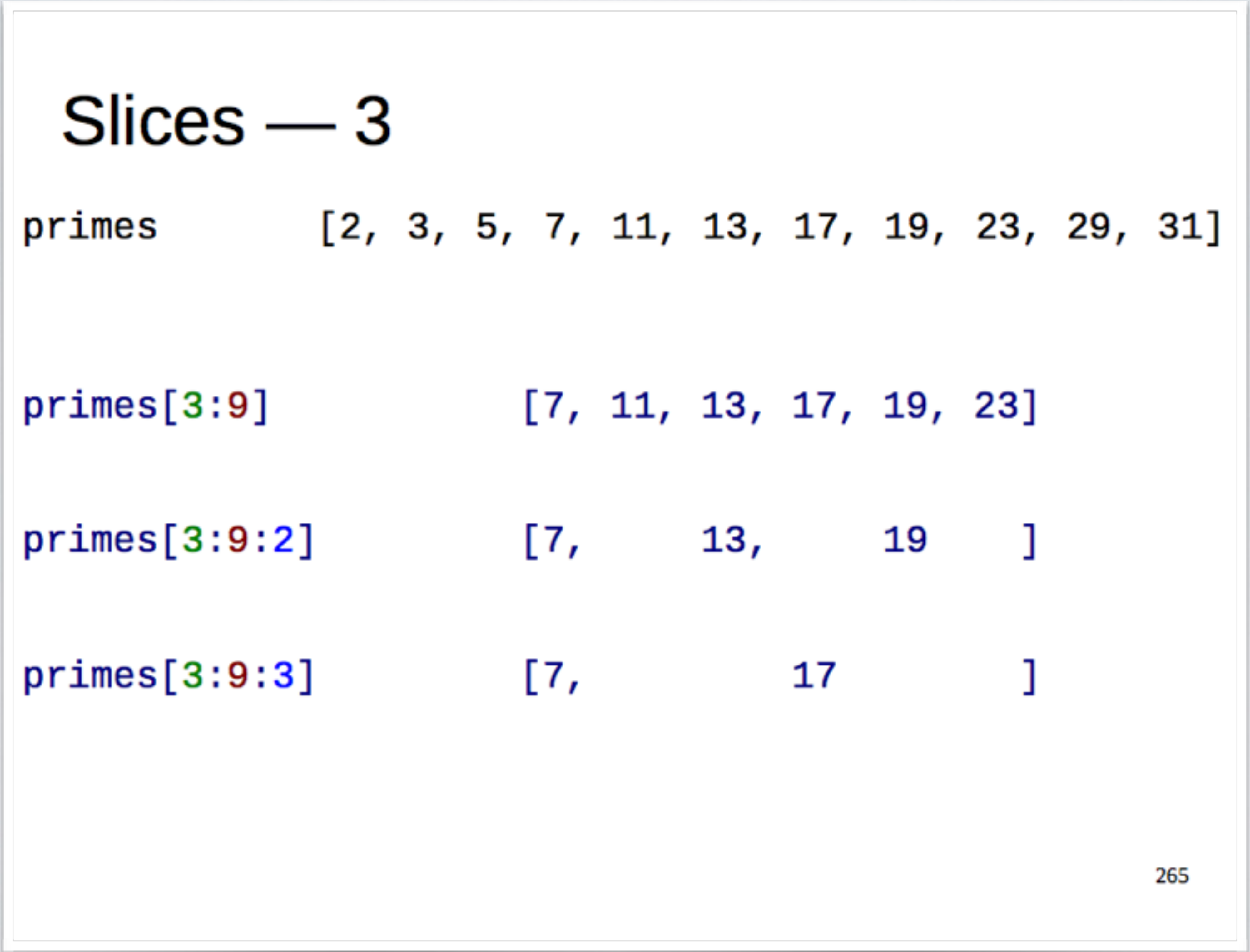
Slices — 2

```
primes          [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]


primes[3:9]              [7, 11, 13, 17, 19, 23]

primes[:9]      [2, 3, 5, 7, 11, 13, 17, 19, 23]

primes[3:]               [7, 11, 13, 17, 19, 23, 29, 31]

primes[:]       [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
```

(a.k.a. shallow copy)

264

Python Basics



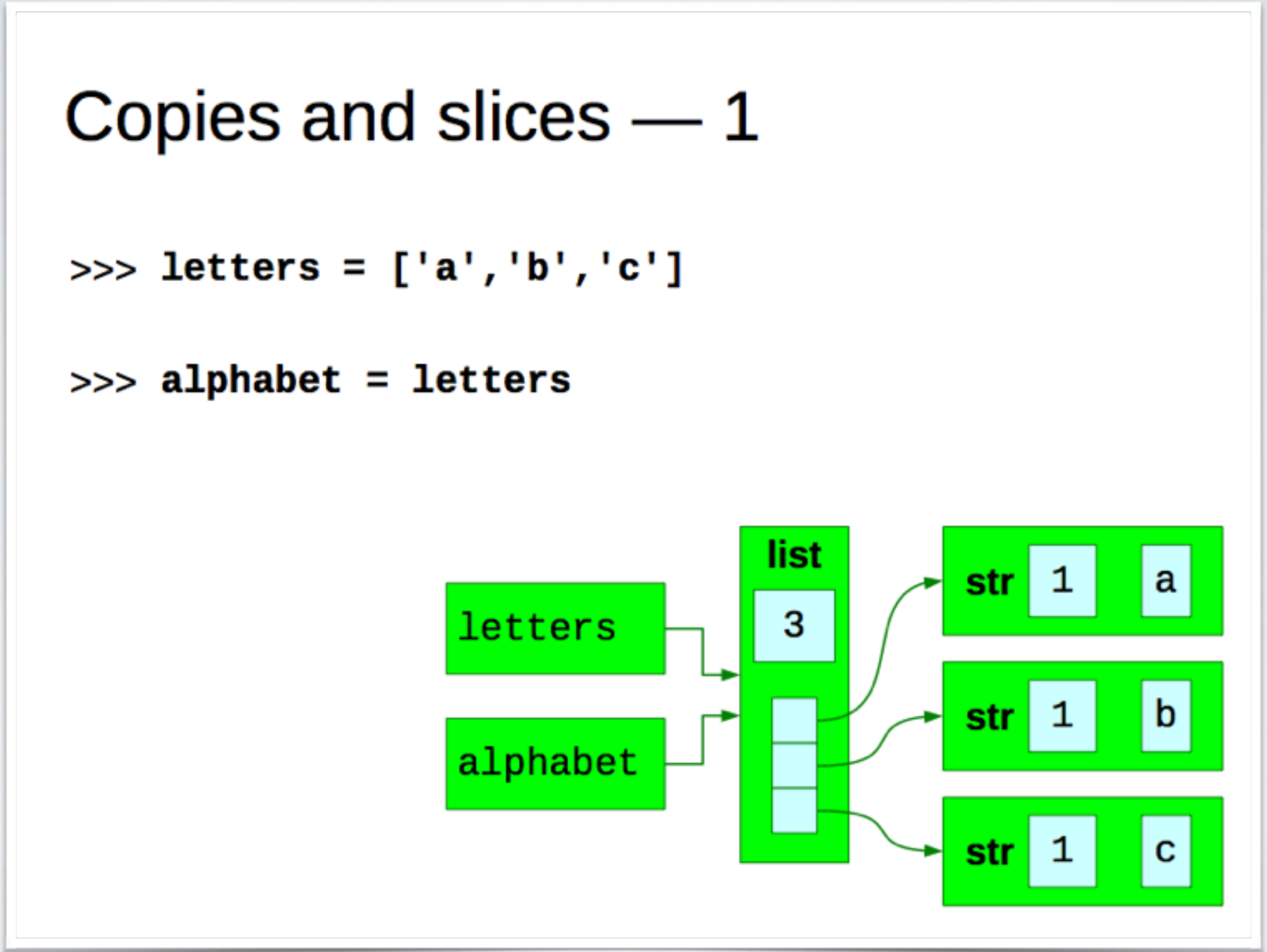
Slices — 3

```
primes        [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]


primes[3:9]             [7, 11, 13, 17, 19, 23]

primes[3:9:2]           [7,     13,     19    ]

primes[3:9:3]           [7,         17        ]
```

265

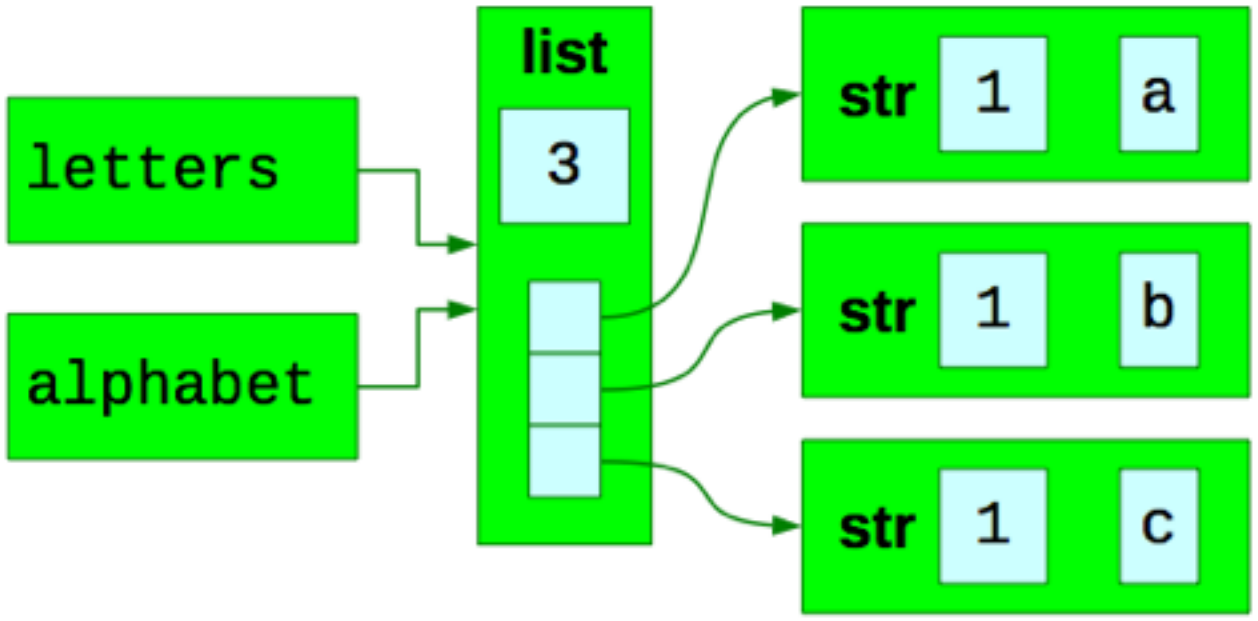
Slide taken from University of Cambridge's Python AB Course

Python Basics



Copies and slices — 1

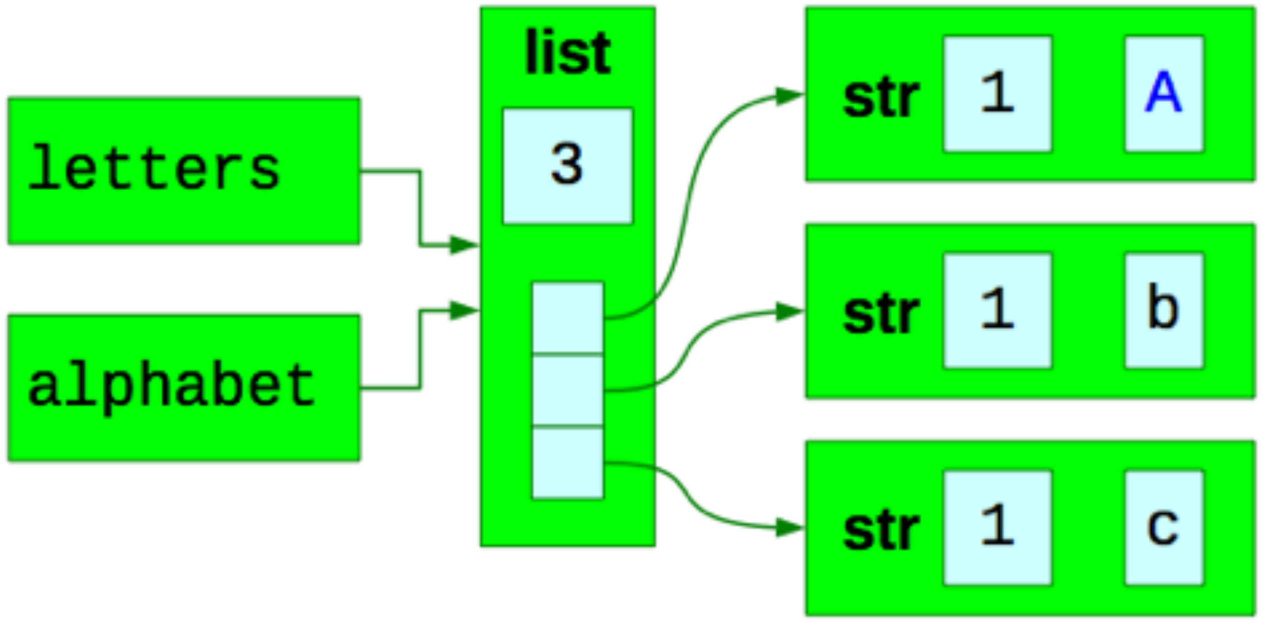
```
>>> letters = ['a','b','c']

>>> alphabet = letters
```



Python Basics

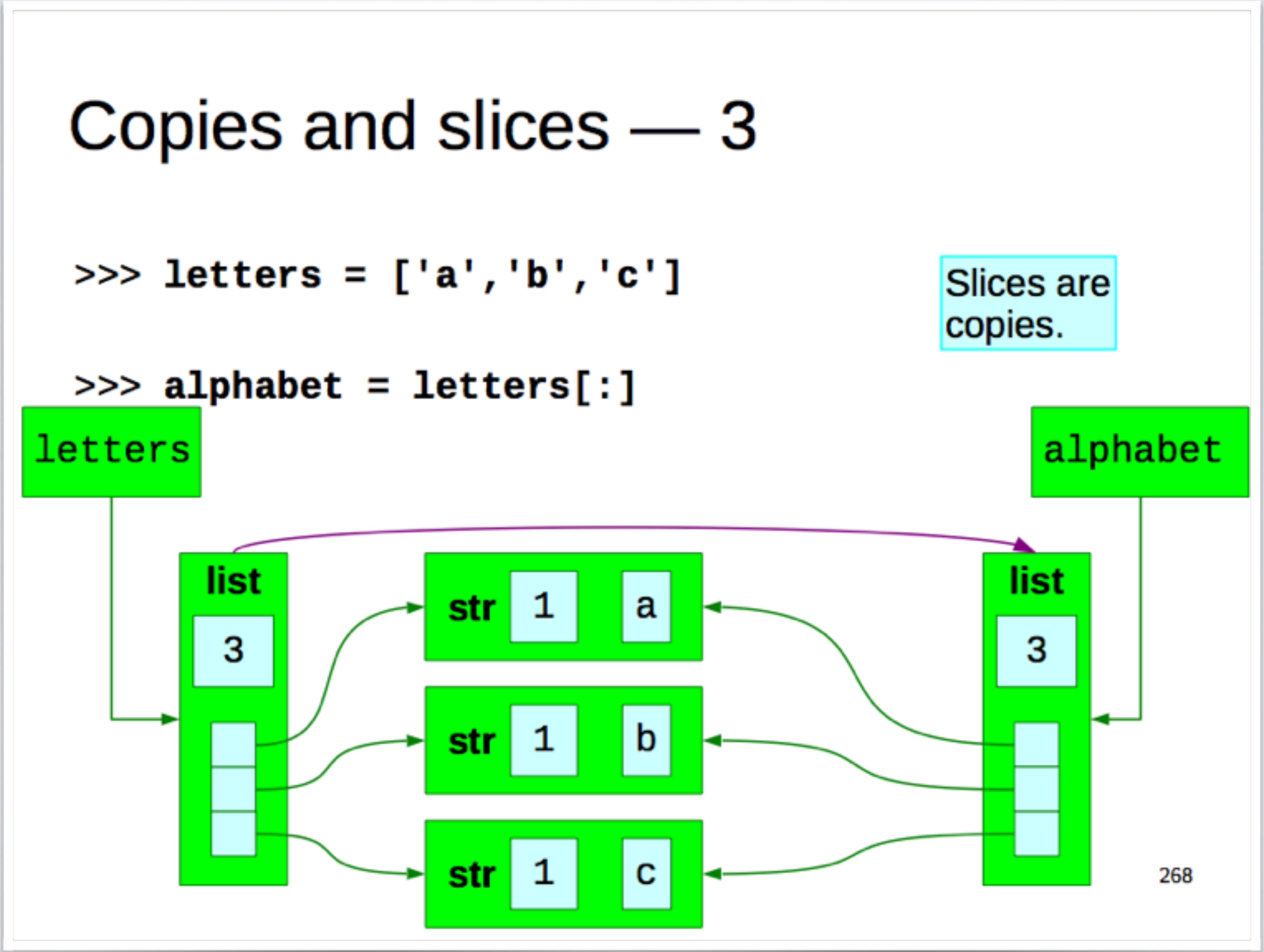
Copies and slices — 2

```
>>> letters[0] = 'A'

>>> print(alphabet)

['A', 'b', 'c']
```



Slide taken from University of Cambridge's Python AB Course

Python Basics



Copies and slices — 3

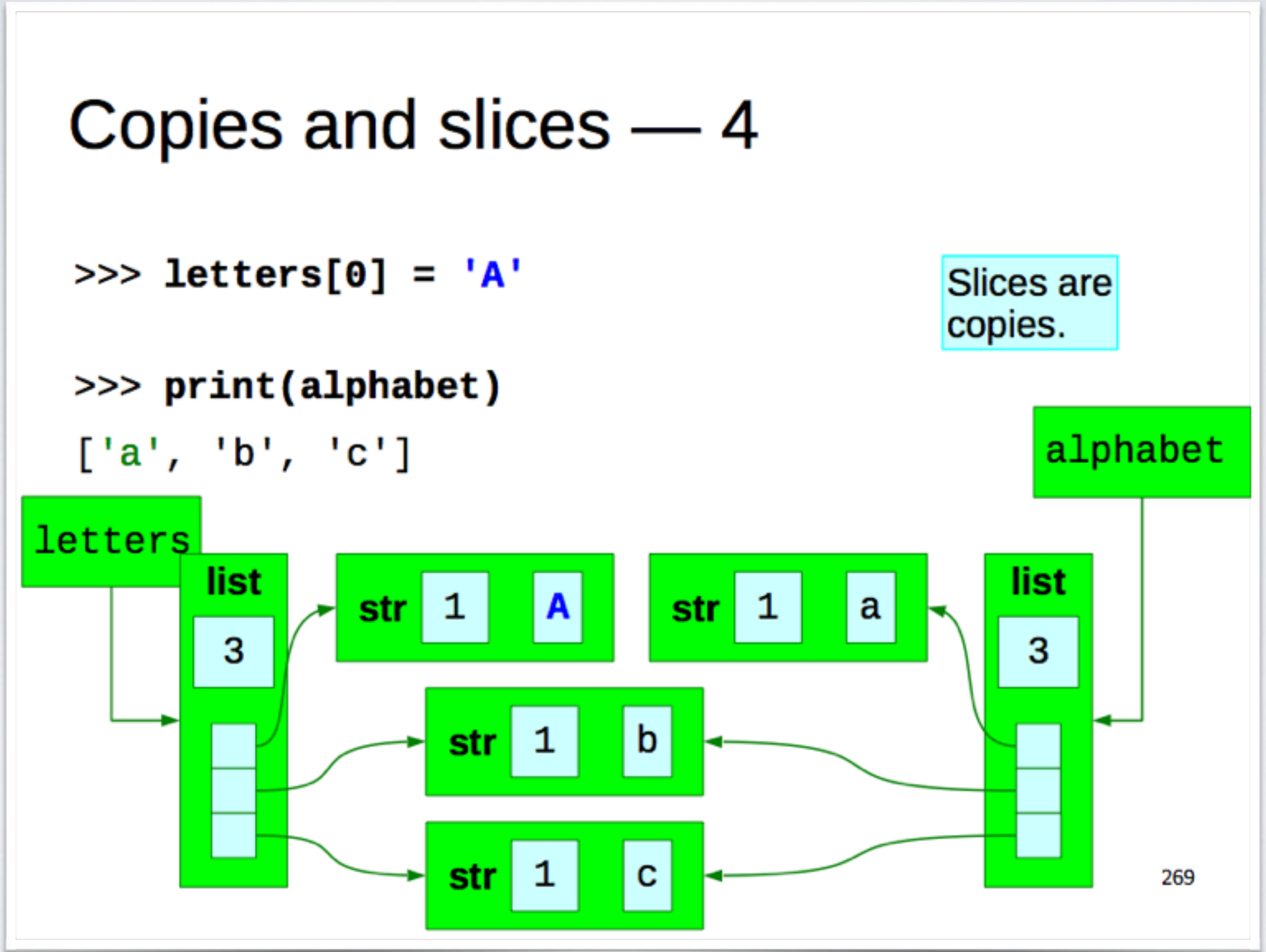
```
>>> letters = ['a','b','c']

>>> alphabet = letters[:]
```

Slices are copies.

letters
alphabet
list
3
str 1 a
str 1 b
str 1 c
list
3

268

Python Basics



Copies and slices — 4

```
>>> letters[0] = 'A'

>>> print(alphabet)
['a', 'b', 'c']
```

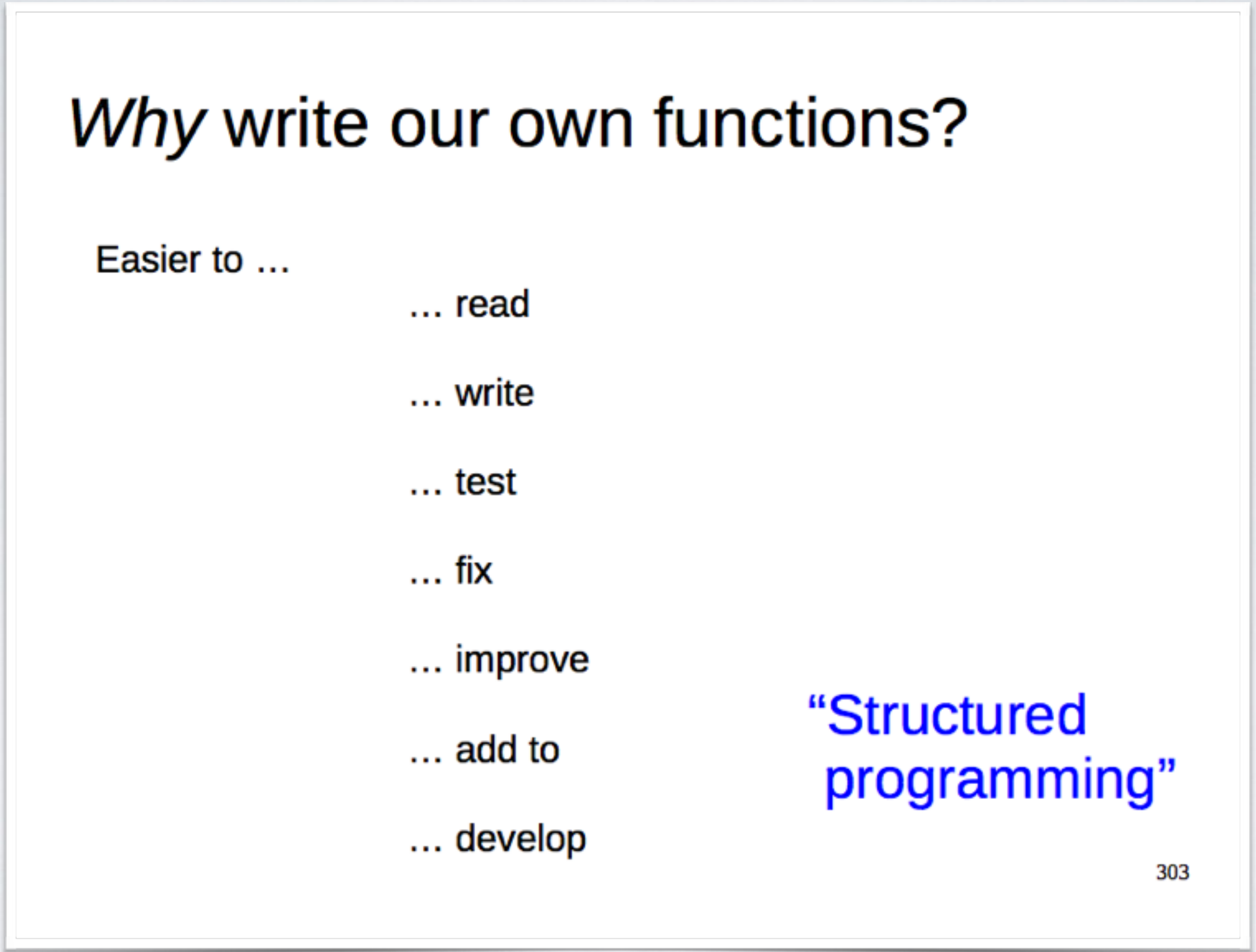
Slices are copies.

alphabet
letters
list
3
str 1 A
str 1 a
str 1 b
str 1 c
list
3

269

Slide taken from University of Cambridge's Python AB Course

Python Basics



Why write our own functions?

Easier to …

… read

… write

… test

… fix

… improve

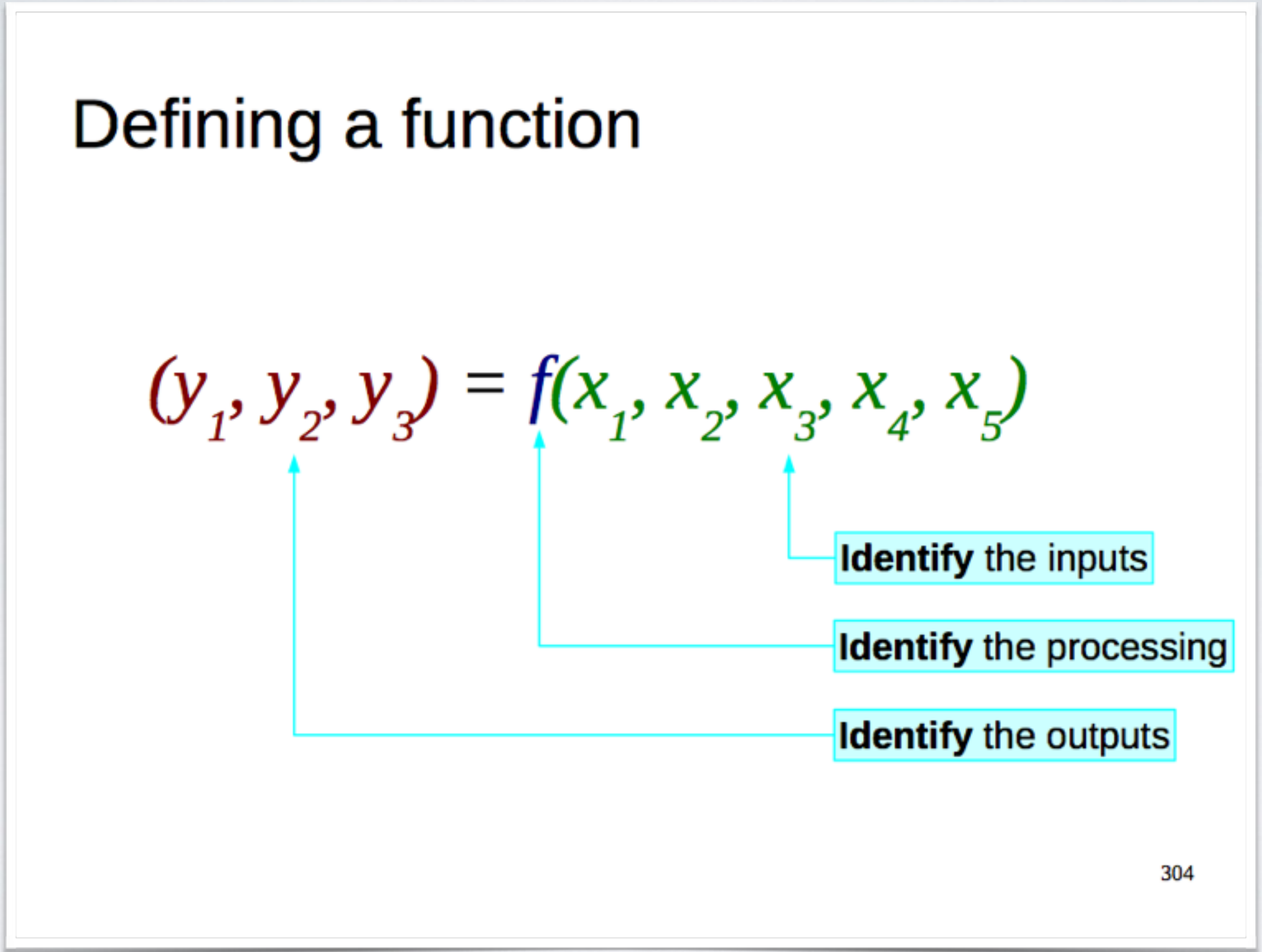
… add to

… develop

"Structured programming"

303

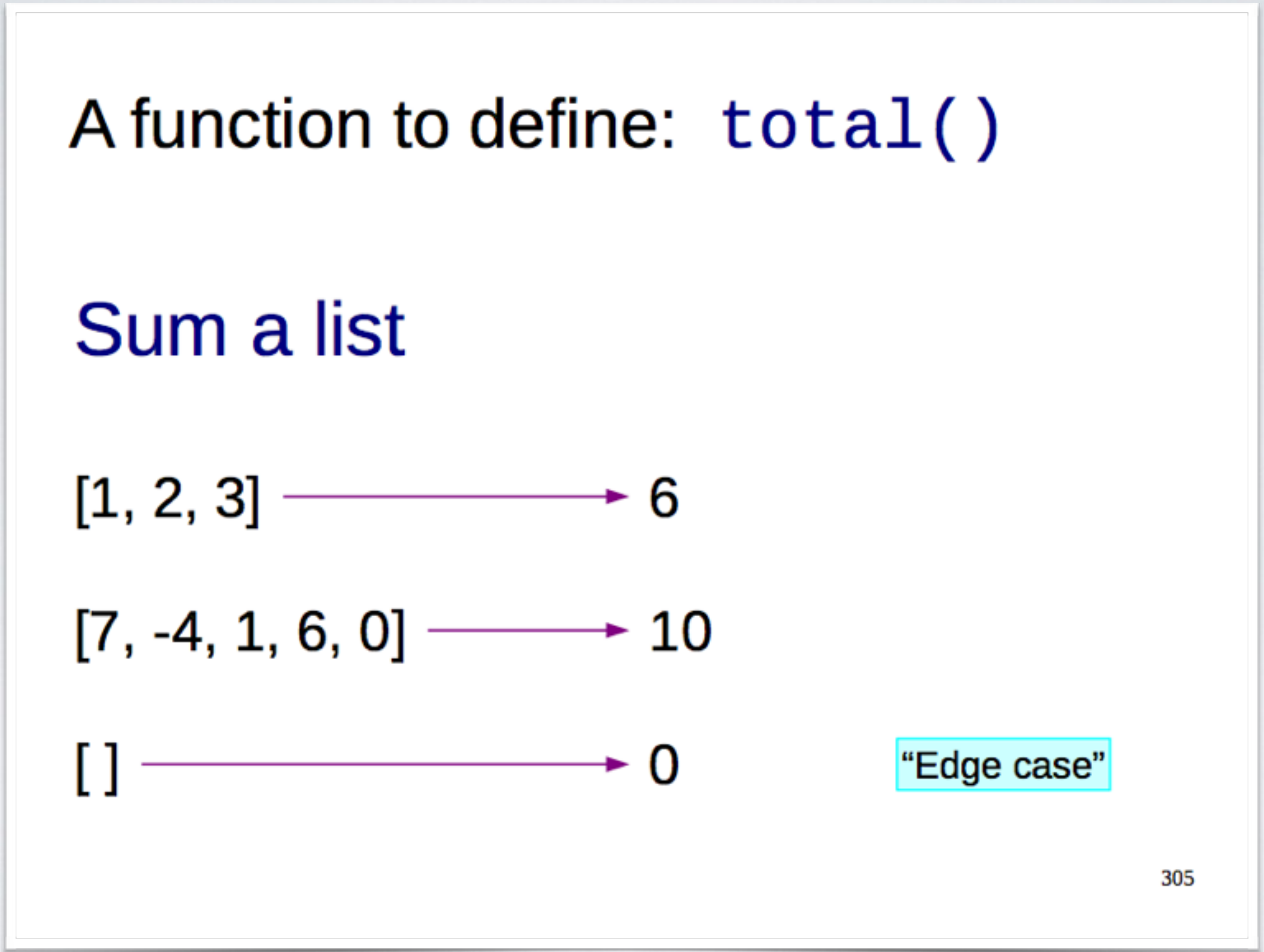
Python Basics



Defining a function

$$(y_1, y_2, y_3) = f(x_1, x_2, x_3, x_4, x_5)$$

Python Basics



A function to define: `total()`

Sum a list

$$[1, 2, 3] \longrightarrow 6$$

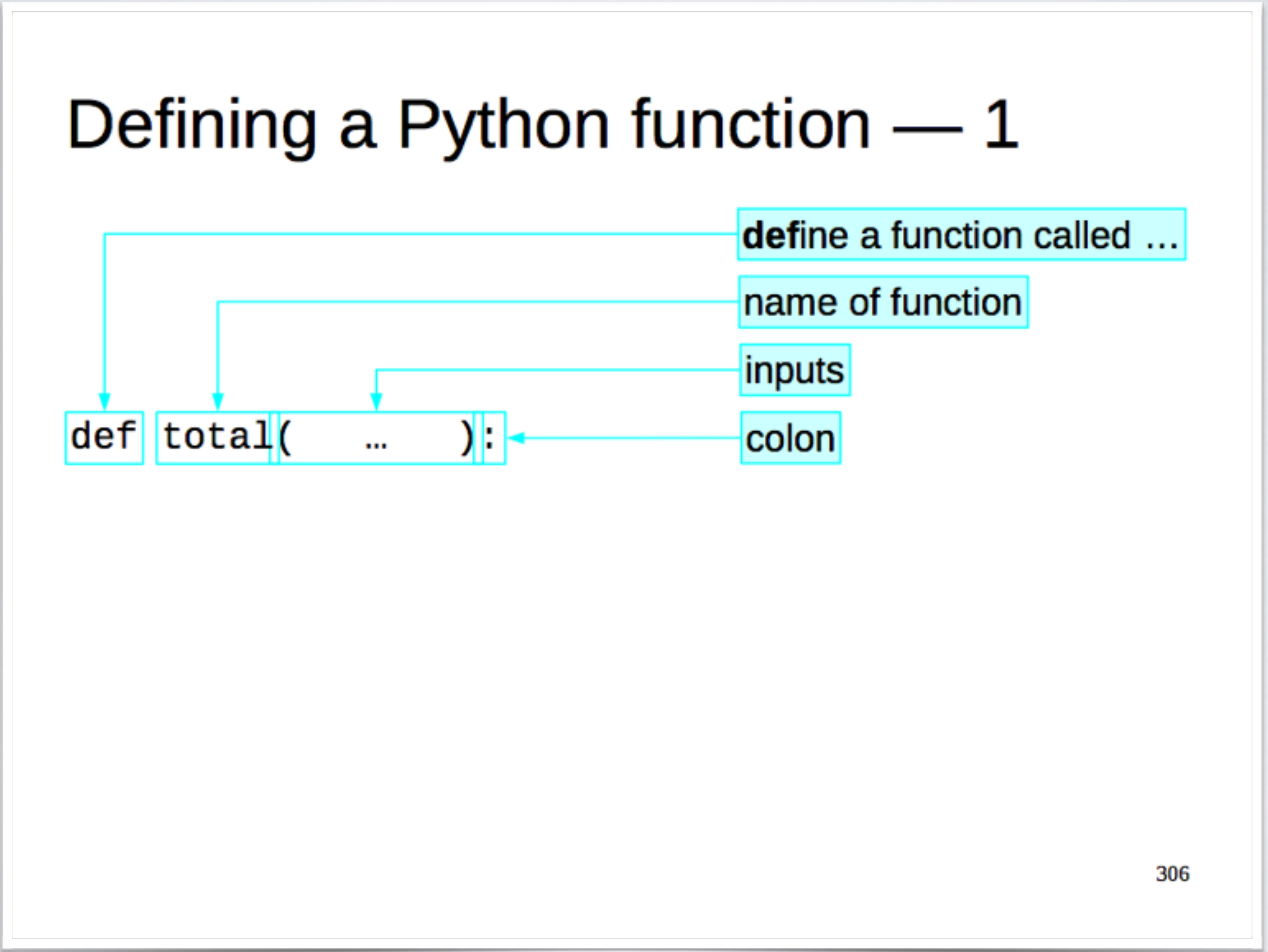
$$[7, -4, 1, 6, 0] \longrightarrow 10$$

$$[\,] \longrightarrow 0$$

"Edge case"

305

Python Basics

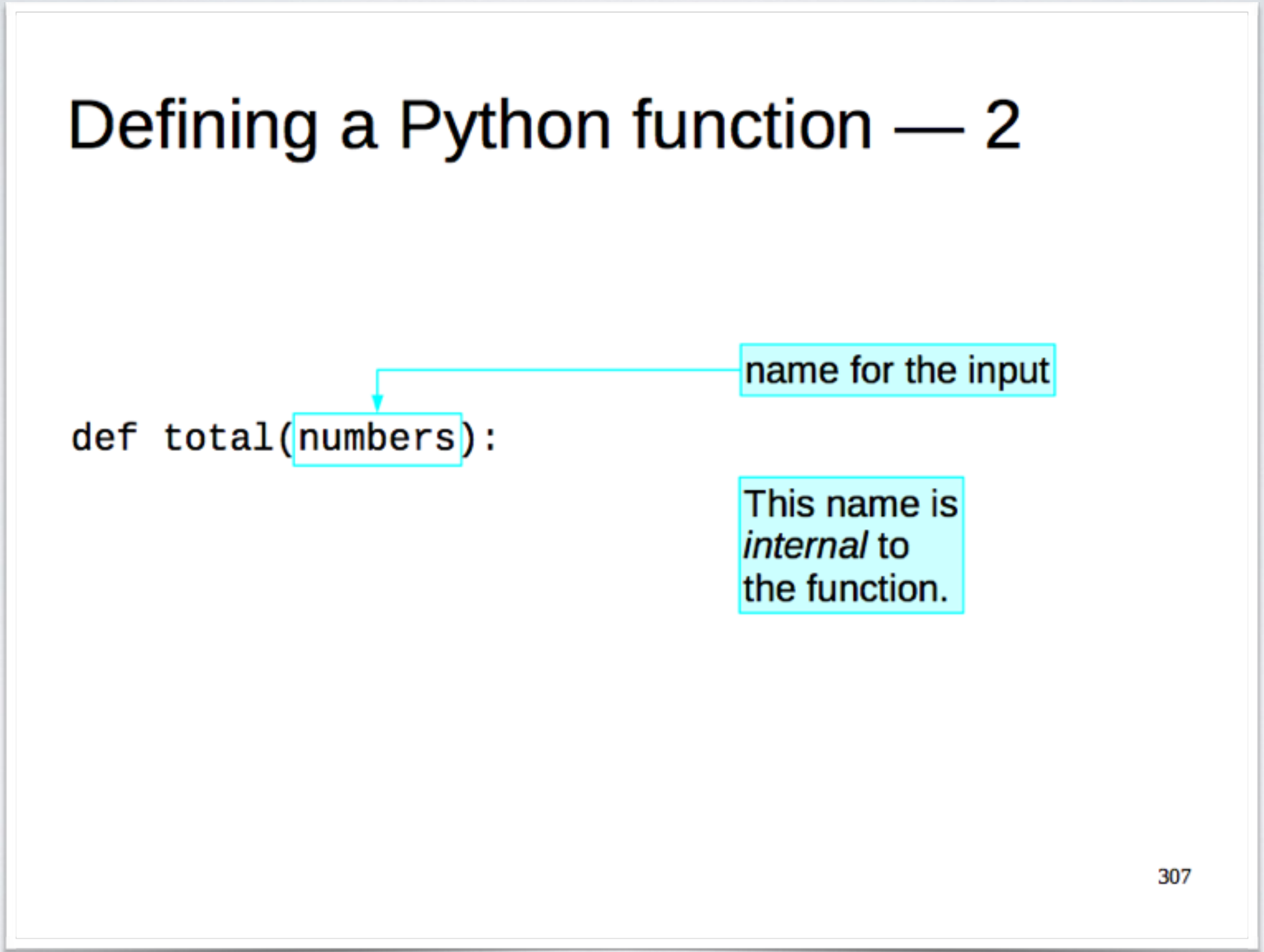


Defining a Python function — 1

306

Slide taken from University of Cambridge's Python AB Course

Python Basics



Defining a Python function — 2

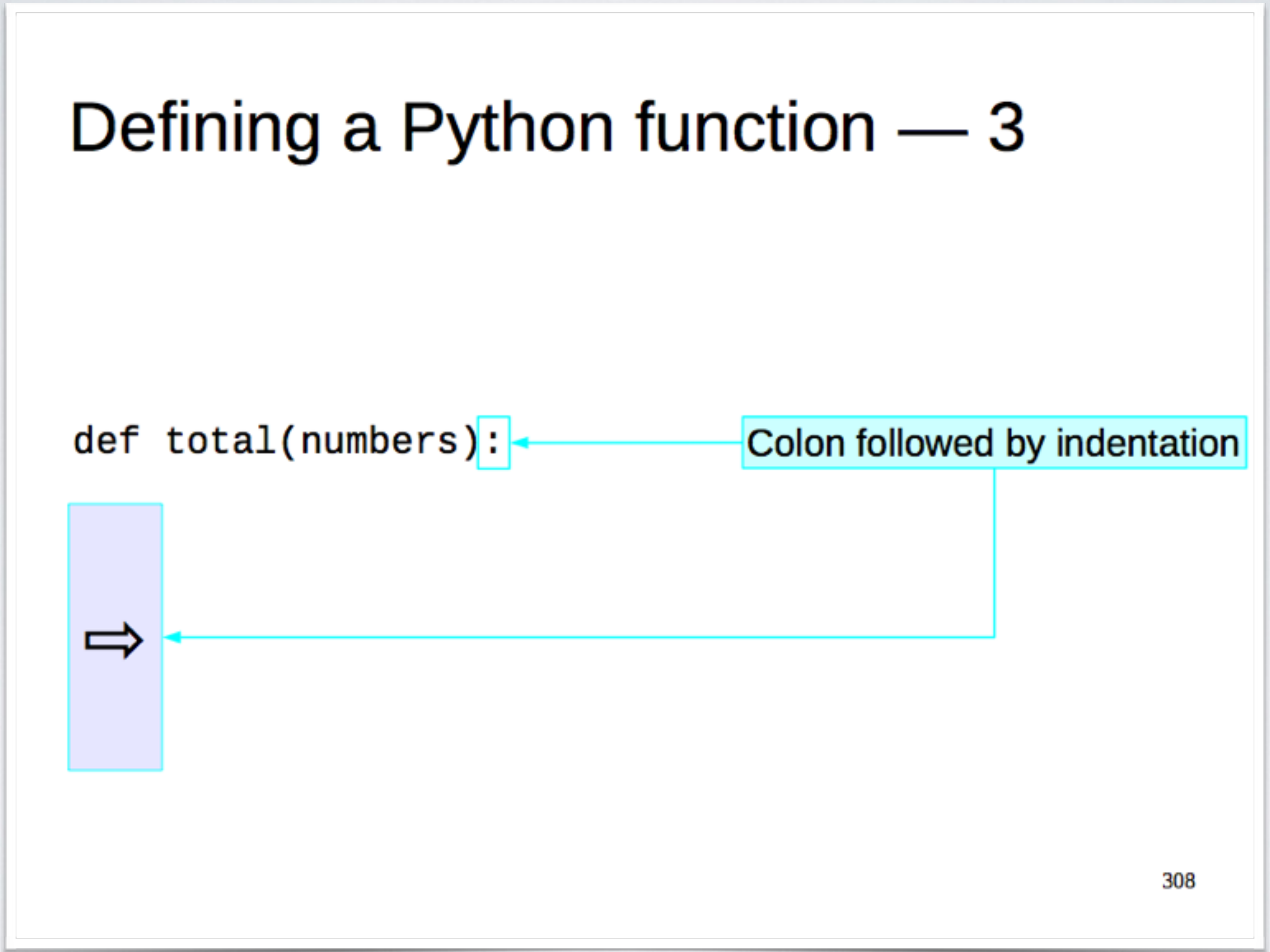
```
def total(numbers):
```

name for the input
This name is *internal* to the function.

307

Slide taken from University of Cambridge’s Python AB Course

Python Basics



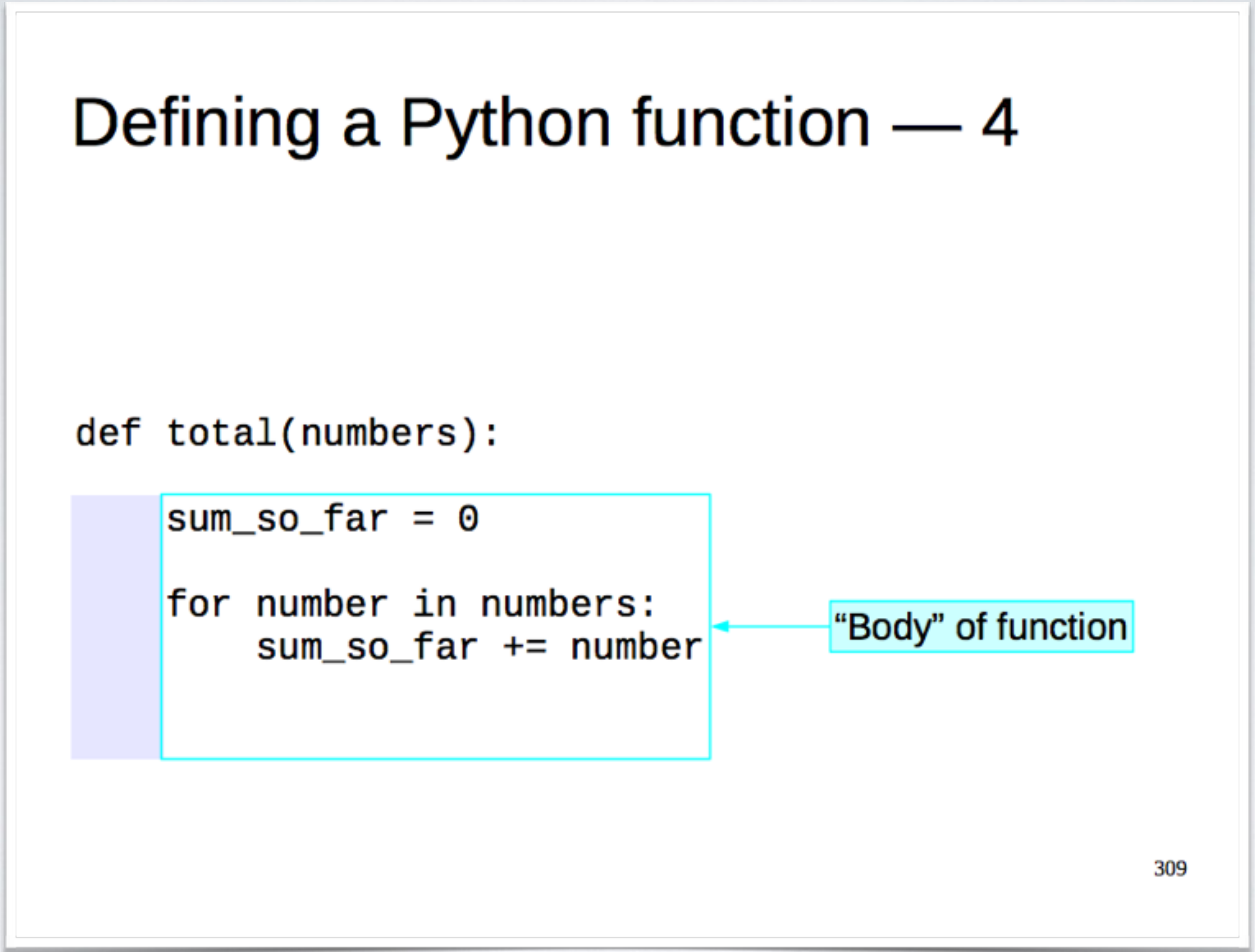
Defining a Python function — 3

```
def total(numbers):
```

Colon followed by indentation
⇨
308

Slide taken from University of Cambridge's Python AB Course

Python Basics



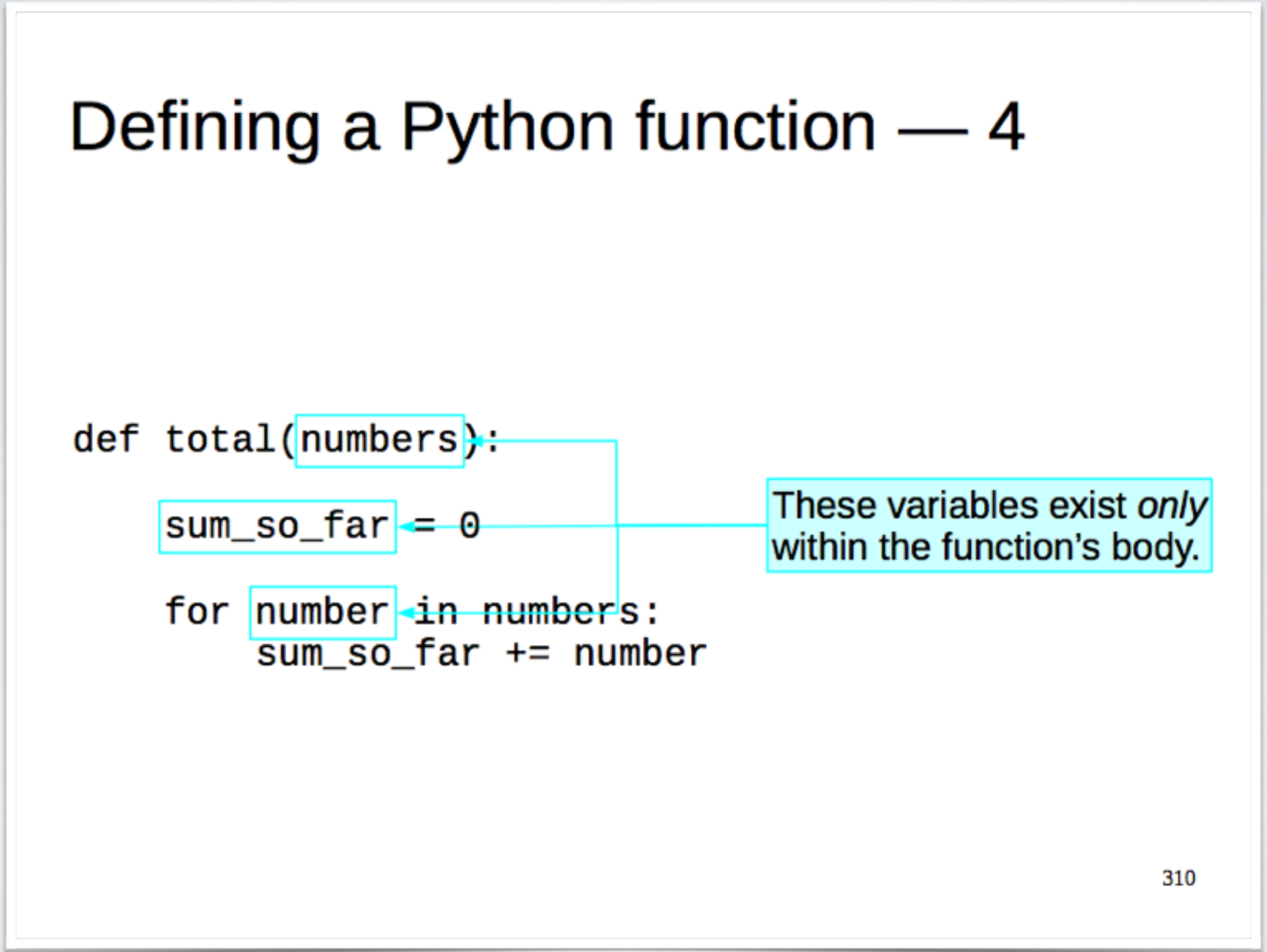
Defining a Python function — 4

```
def total(numbers):

    sum_so_far = 0

    for number in numbers:
        sum_so_far += number
```

“Body” of function

309

Python Basics

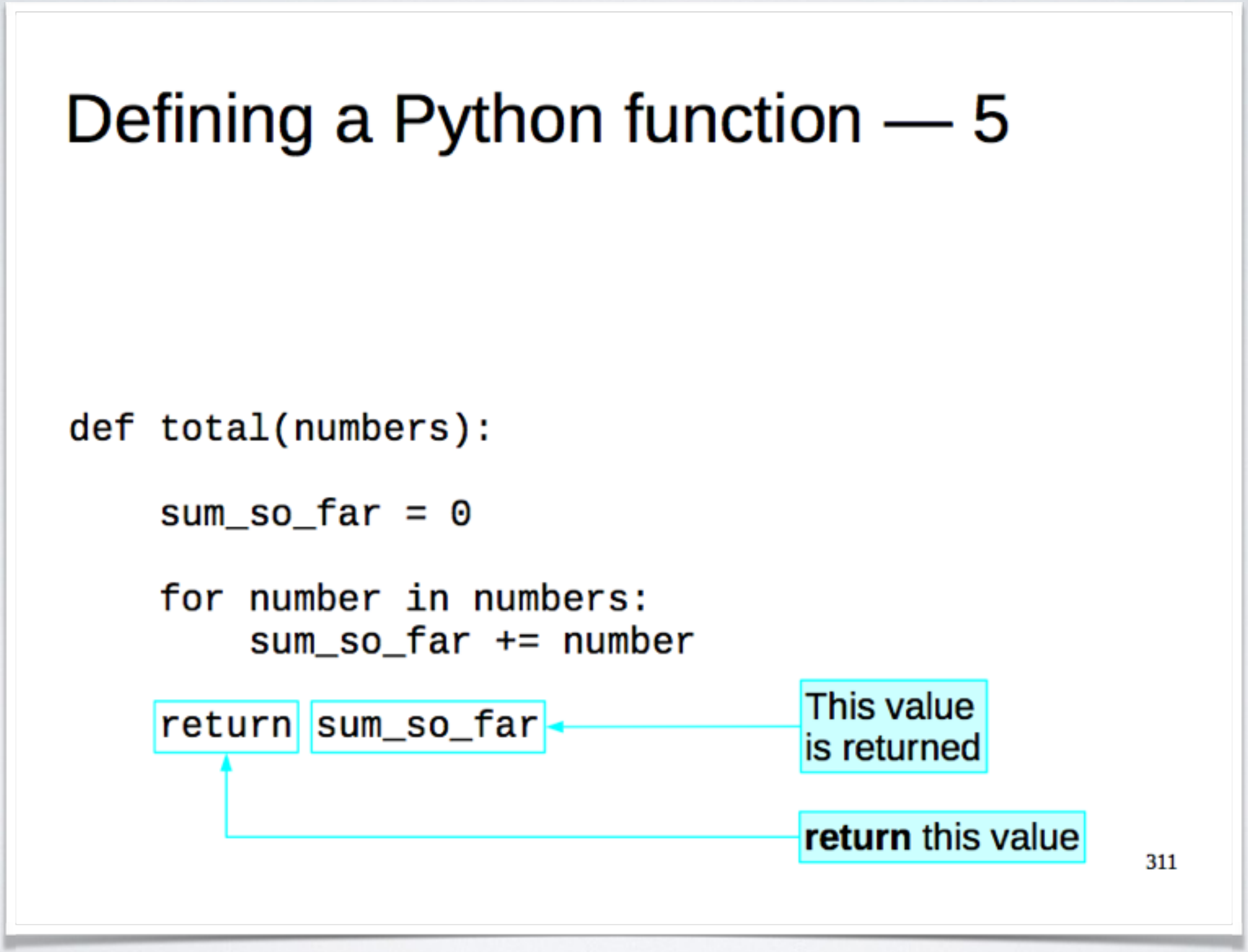


Defining a Python function — 4

```
def total(numbers):

    sum_so_far = 0

    for number in numbers:
        sum_so_far += number
```

These variables exist *only* within the function's body.

Python Basics



Defining a Python function — 5

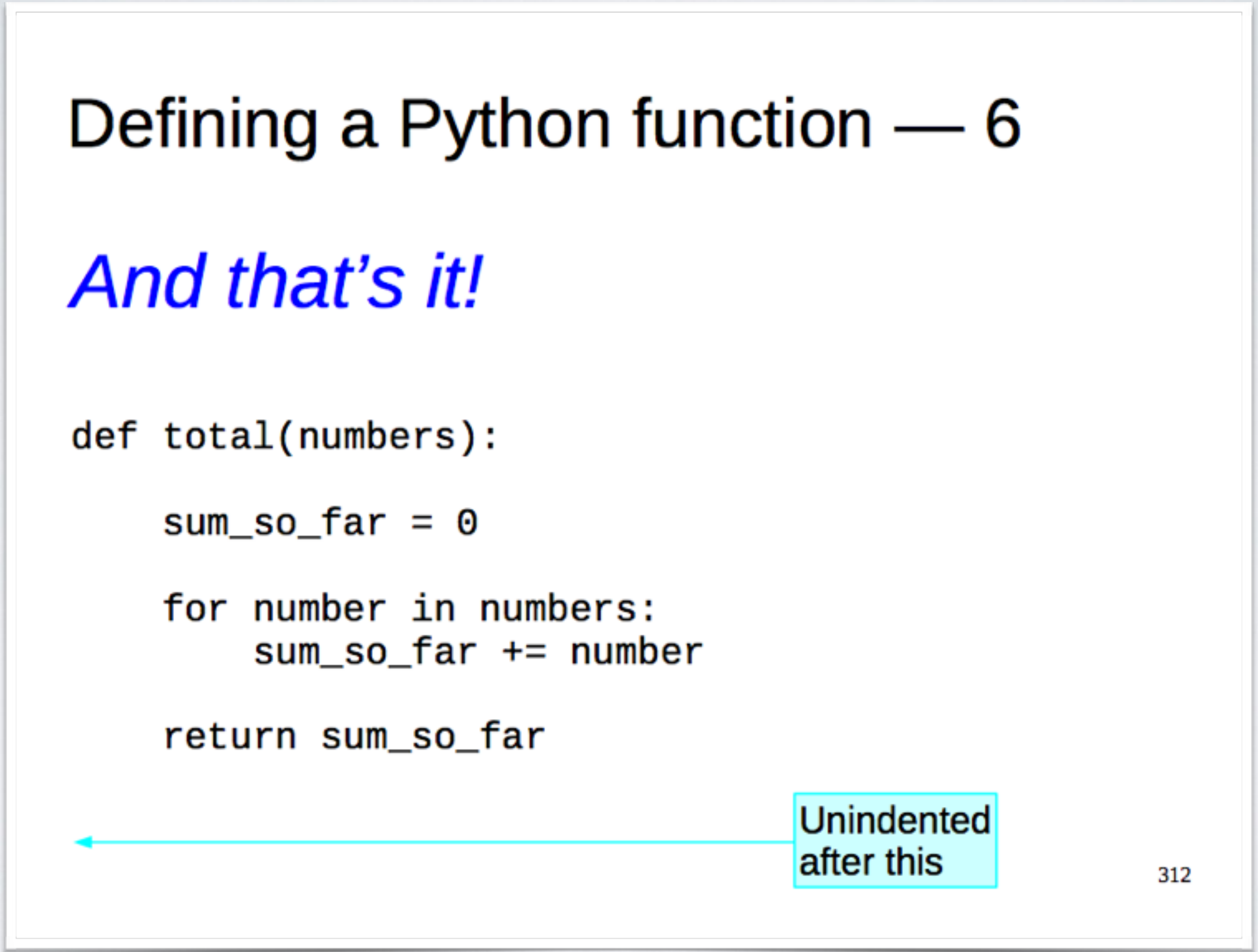
```
def total(numbers):

    sum_so_far = 0

    for number in numbers:
        sum_so_far += number

    return sum_so_far
```

This value is returned
return this value

311

Slide taken from University of Cambridge's Python AB Course

Python Basics



Defining a Python function — 6

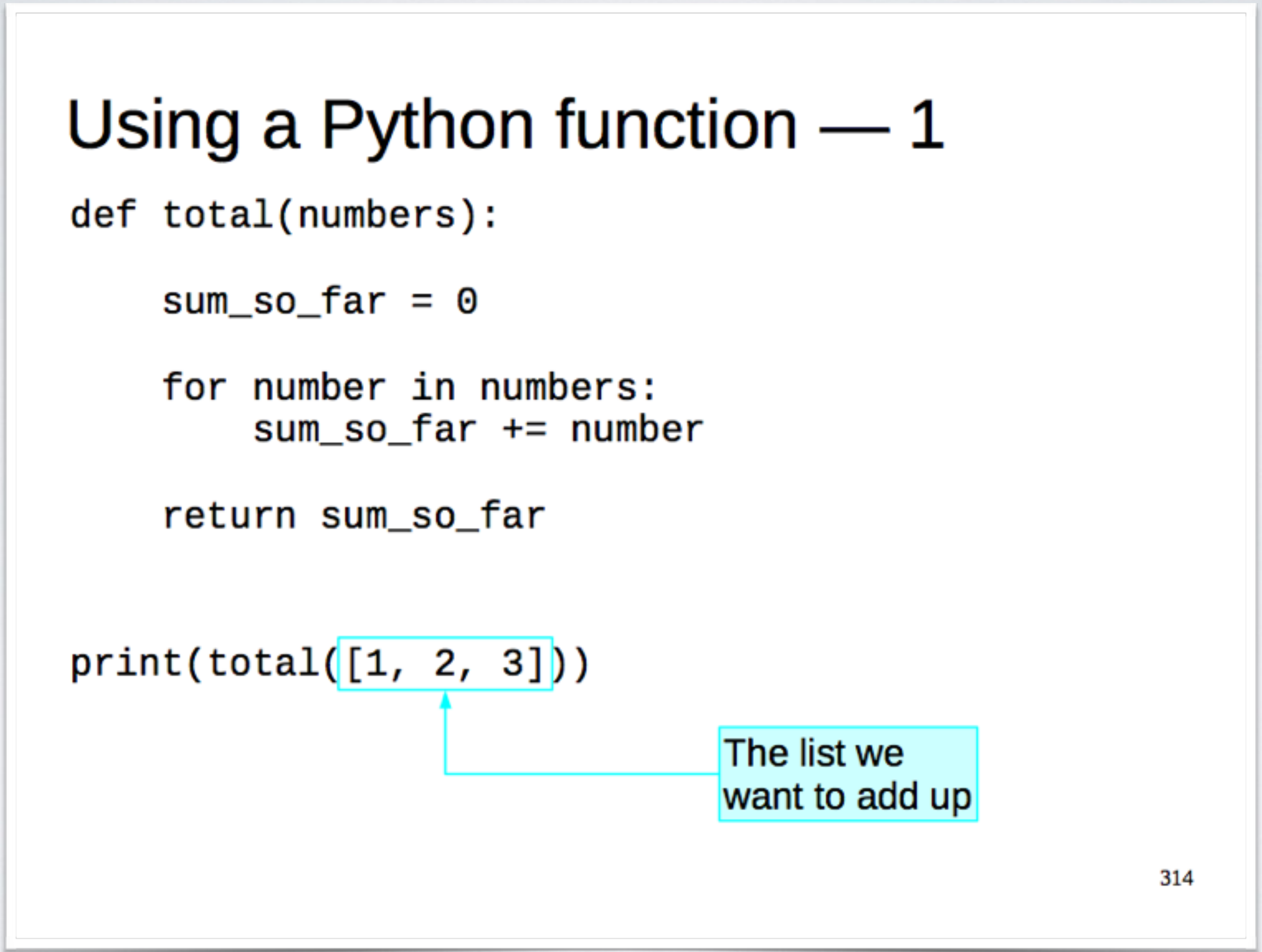
And that's it!

```
def total(numbers):

    sum_so_far = 0

    for number in numbers:
        sum_so_far += number

    return sum_so_far
```

Unindented after this

312

Python Basics



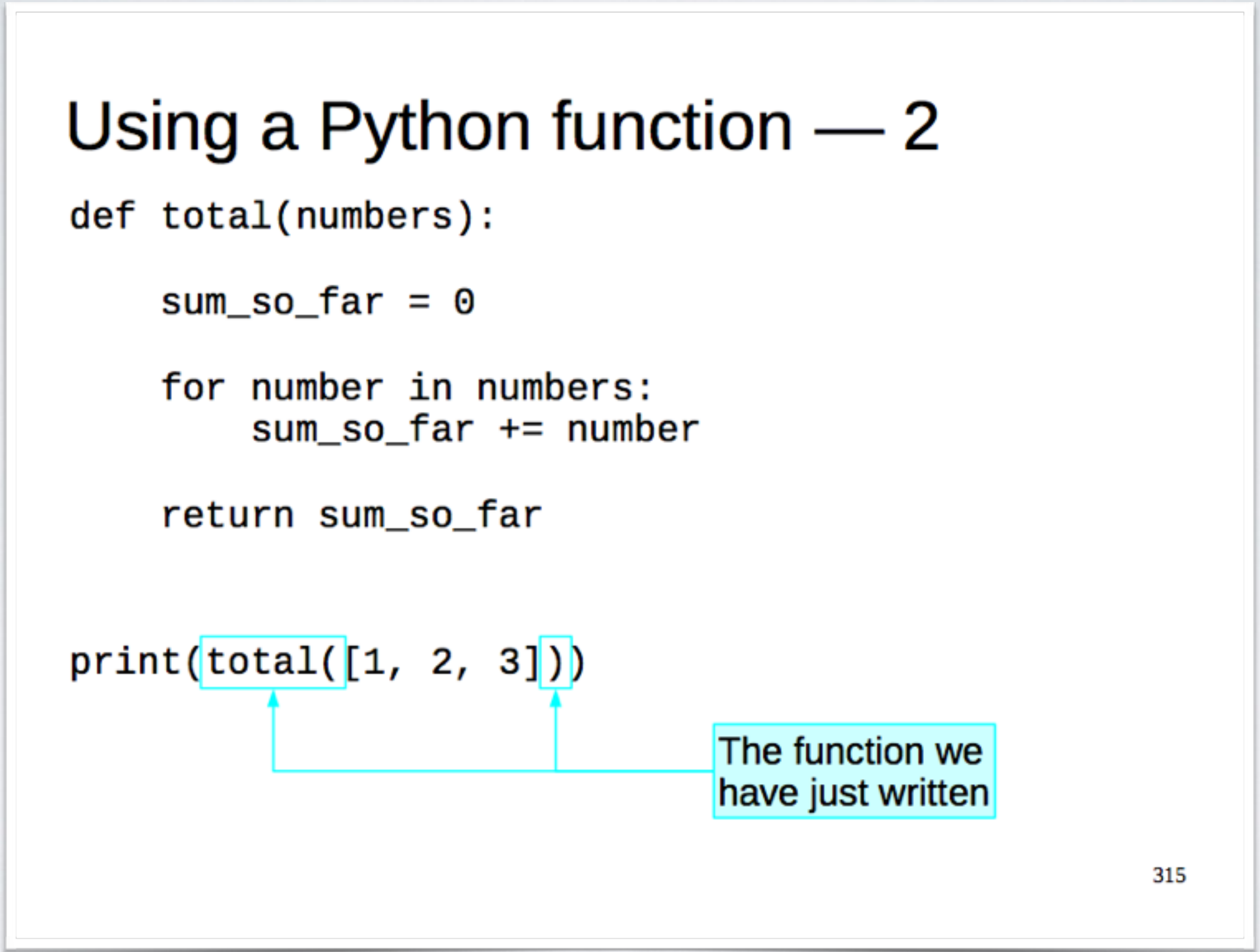
Using a Python function — 1

```
def total(numbers):

    sum_so_far = 0

    for number in numbers:
        sum_so_far += number

    return sum_so_far


print(total([1, 2, 3]))
```

The list we want to add up

314

Python Basics

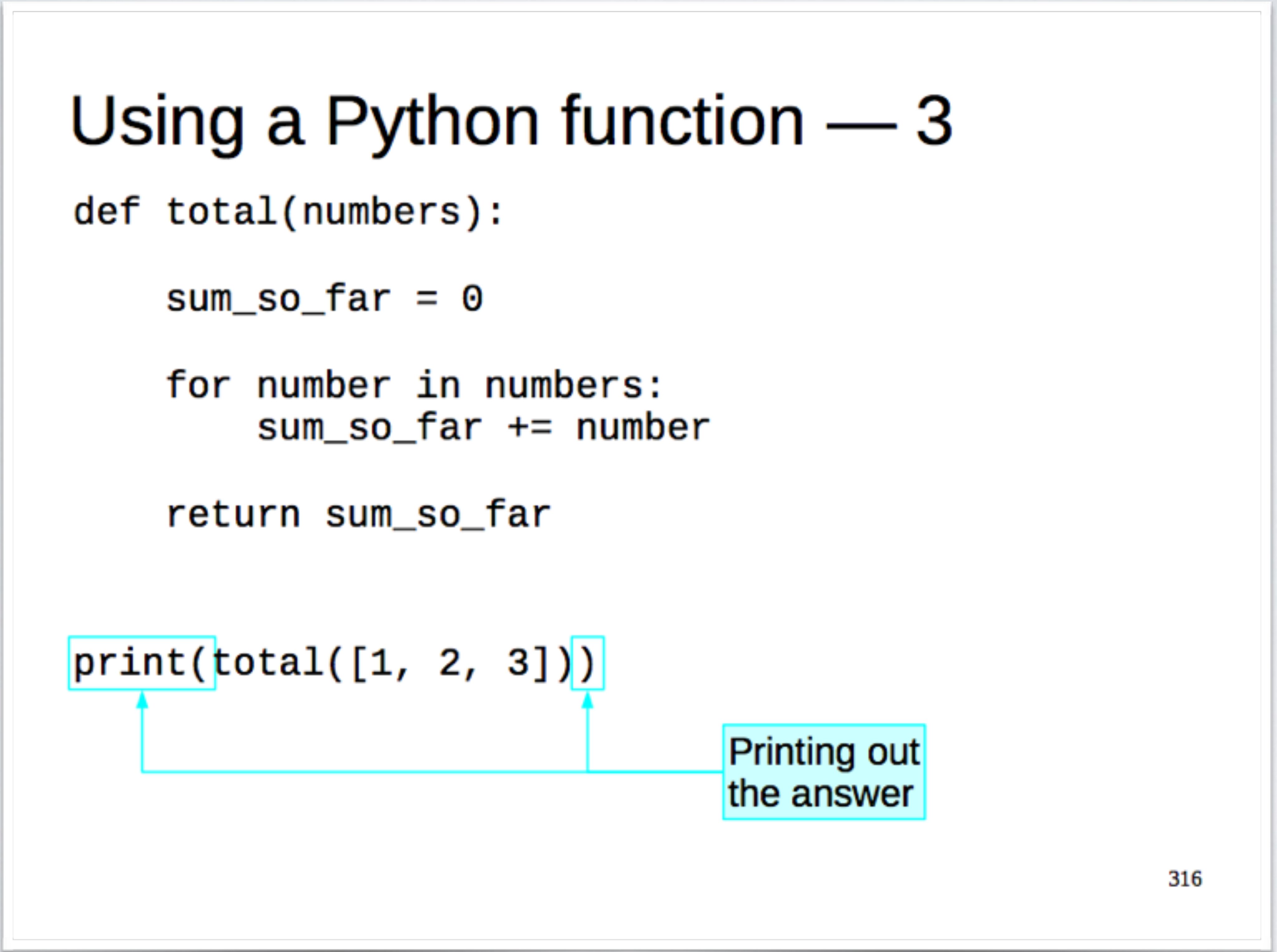


Using a Python function — 2

```
def total(numbers):

    sum_so_far = 0

    for number in numbers:
        sum_so_far += number

    return sum_so_far


print(total([1, 2, 3]))
```

315

Python Basics



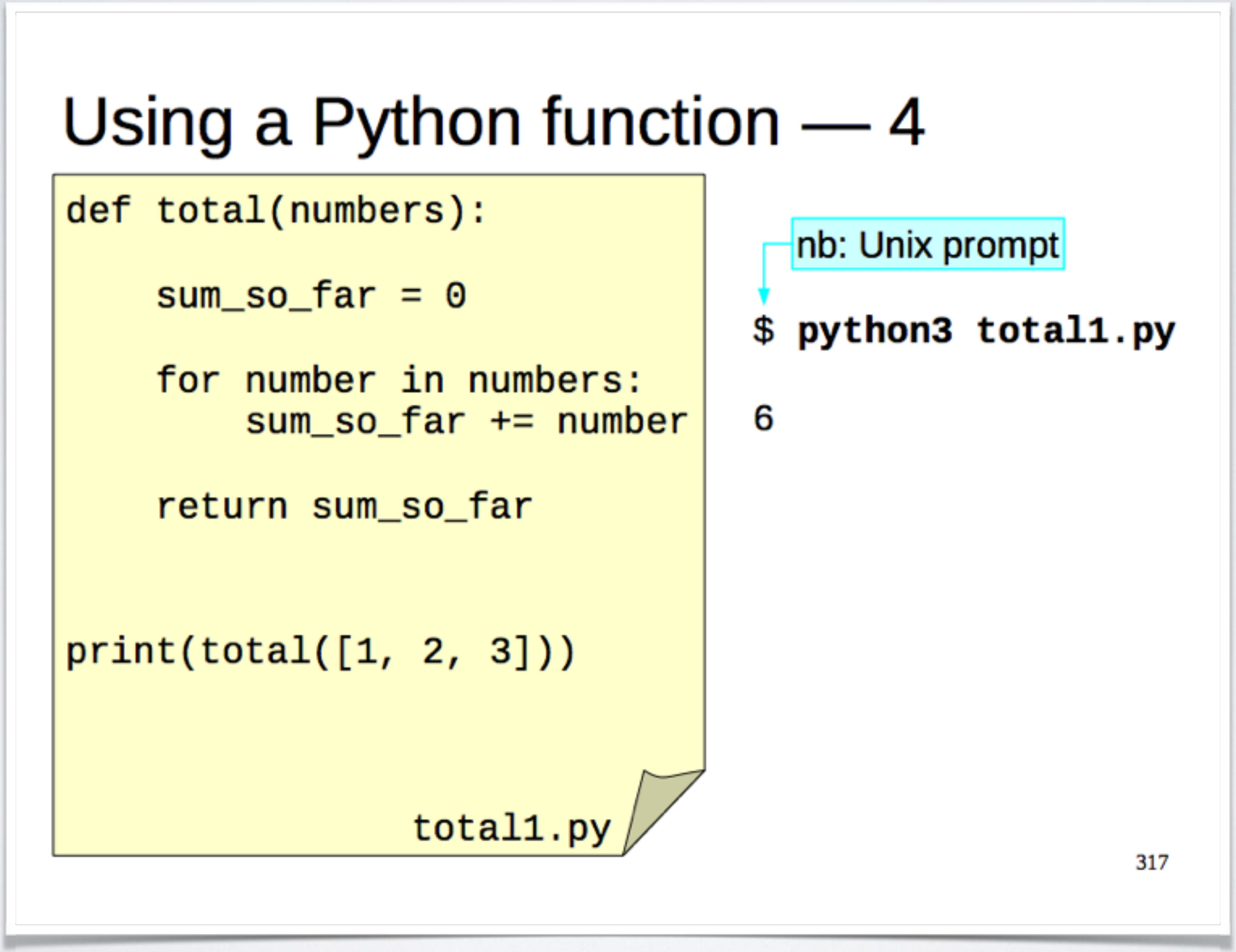
Using a Python function — 3

```
def total(numbers):

    sum_so_far = 0

    for number in numbers:
        sum_so_far += number

    return sum_so_far


print(total([1, 2, 3]))
```

Printing out the answer

316

Python Basics



Using a Python function — 4

```
def total(numbers):

    sum_so_far = 0

    for number in numbers:
        sum_so_far += number

    return sum_so_far


print(total([1, 2, 3]))
```

total1.py

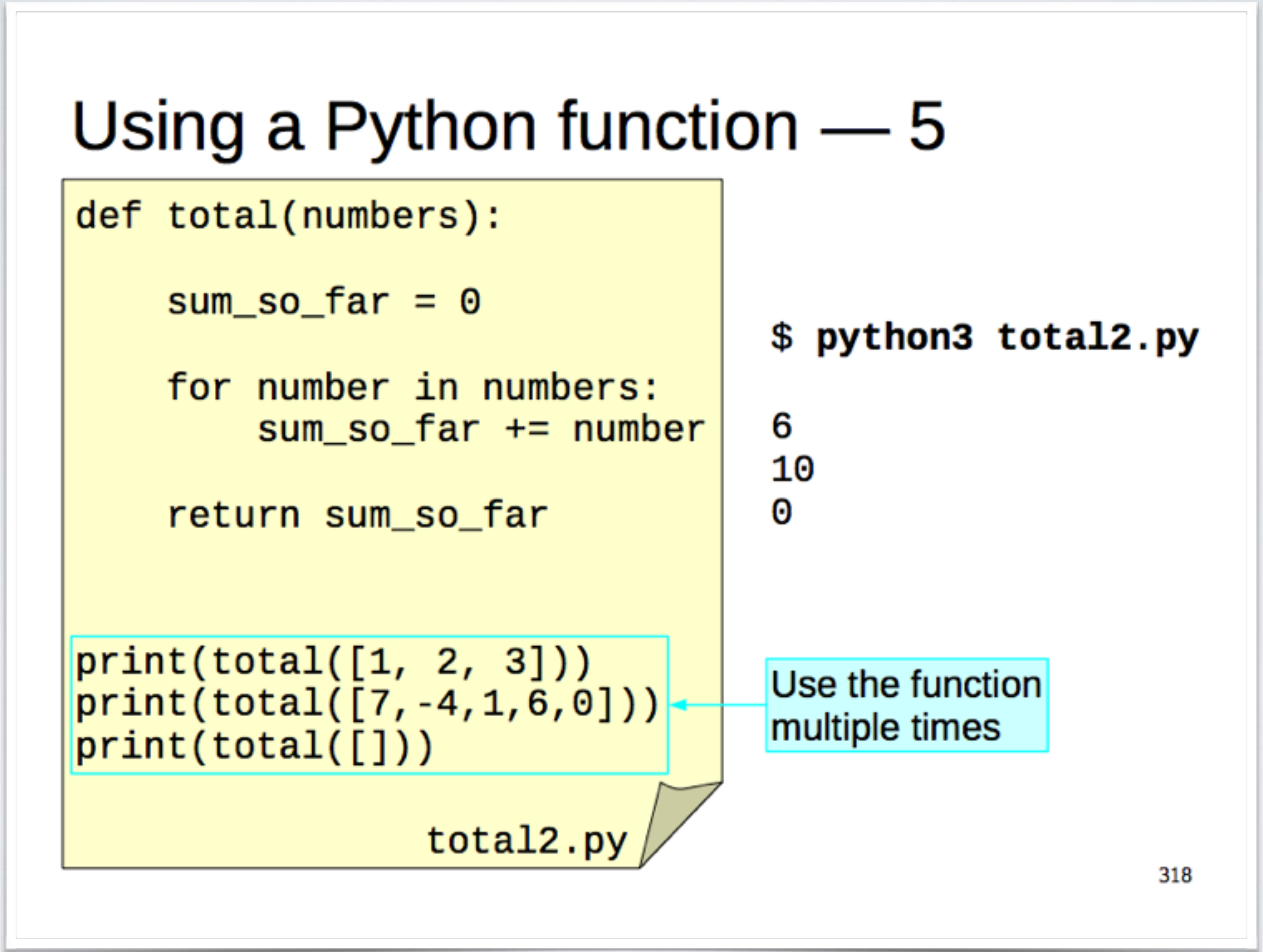
nb: Unix prompt

```
$ python3 total1.py

6
```

317

Slide taken from University of Cambridge's Python AB Course

Python Basics



Using a Python function — 5

```
def total(numbers):

    sum_so_far = 0

    for number in numbers:
        sum_so_far += number

    return sum_so_far


print(total([1, 2, 3]))
print(total([7,-4,1,6,0]))
print(total([]))
```

total2.py

```
$ python3 total2.py

6
10
0
```

Use the function multiple times

318

Slide taken from University of Cambridge's Python AB Course