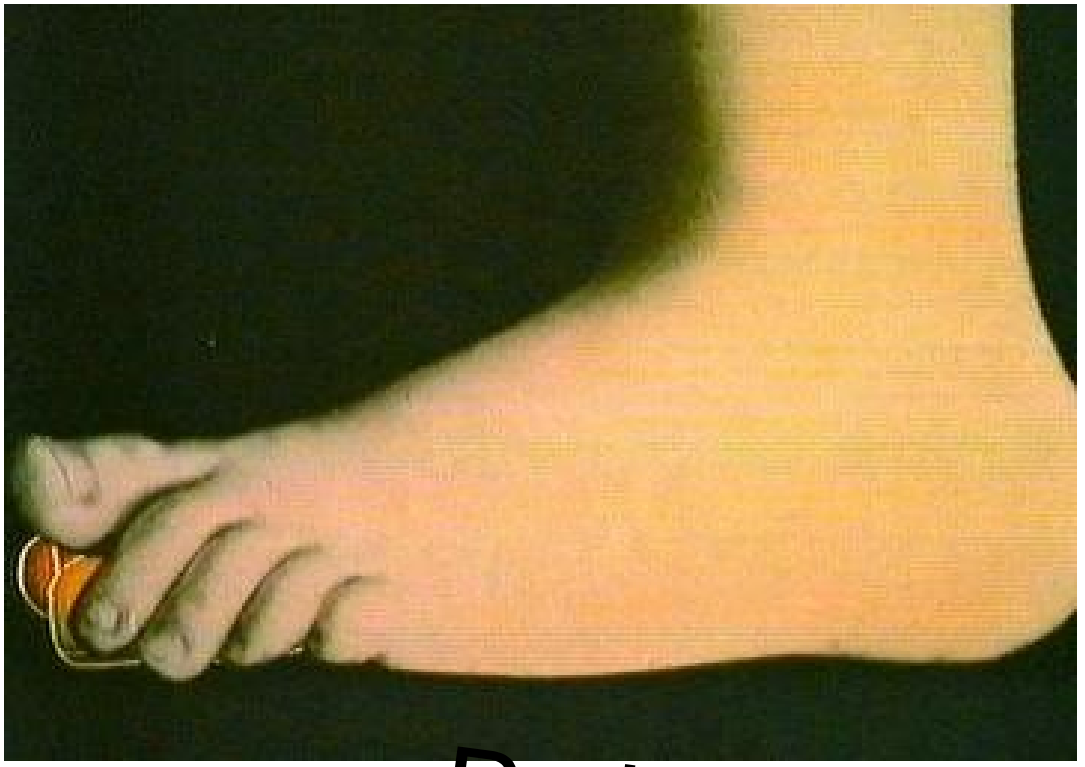
init python(self):
Introduction for Programmers

Rahul Mahale

http://rahulmahale.wordpress.com/

Game Plan

- Lab I
  - Basic Introduction to python
  - Data types and its use.
- Lab II
  - Flow Control in Python
  - Methods(Functions)
- Lab III
  - Modules ,Classes
  - File Operations
  - Puzzles

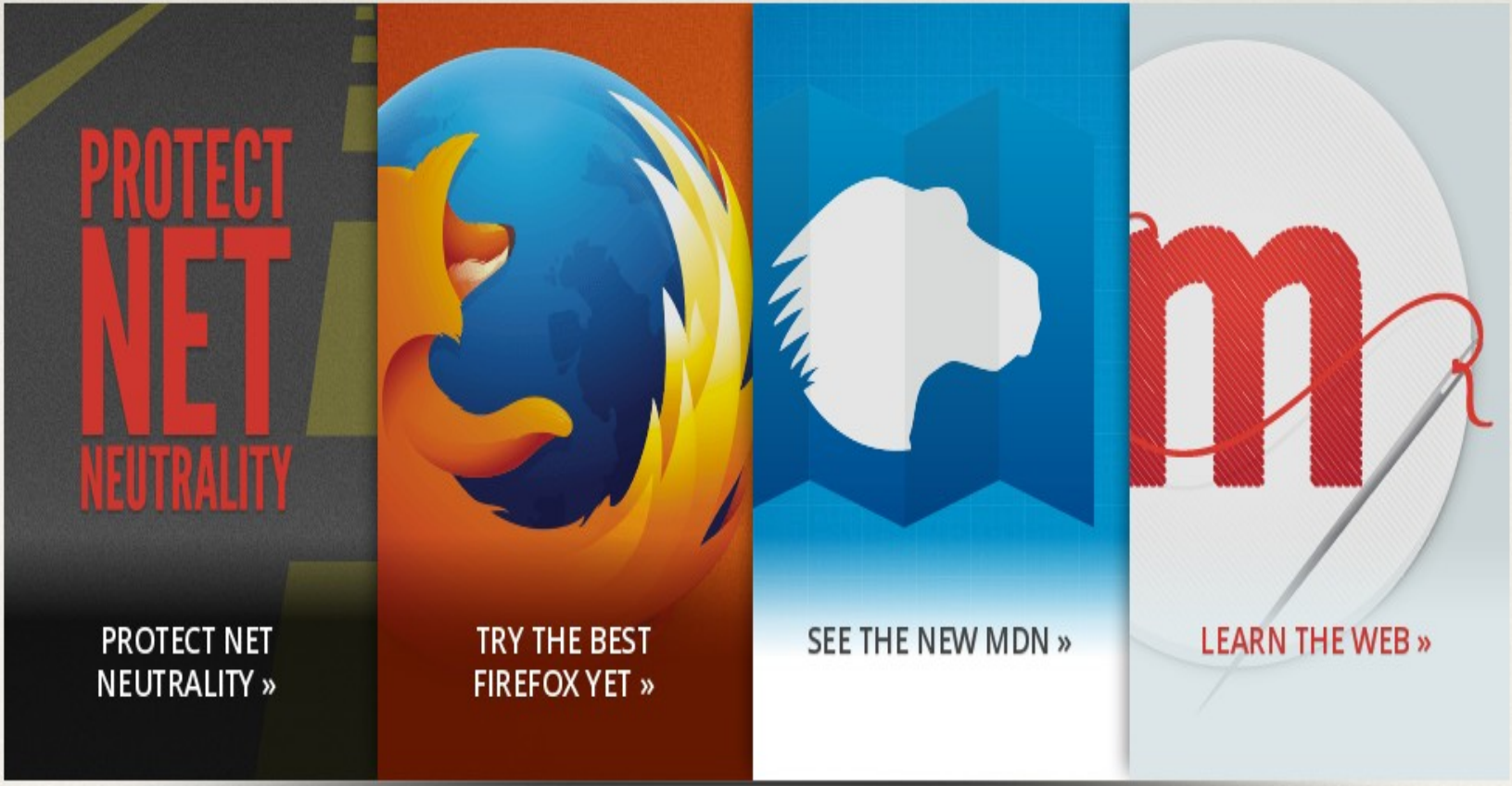


Python: Introduction for Programmers

Python is good for ?

We are mozilla

Doing good is part of our code



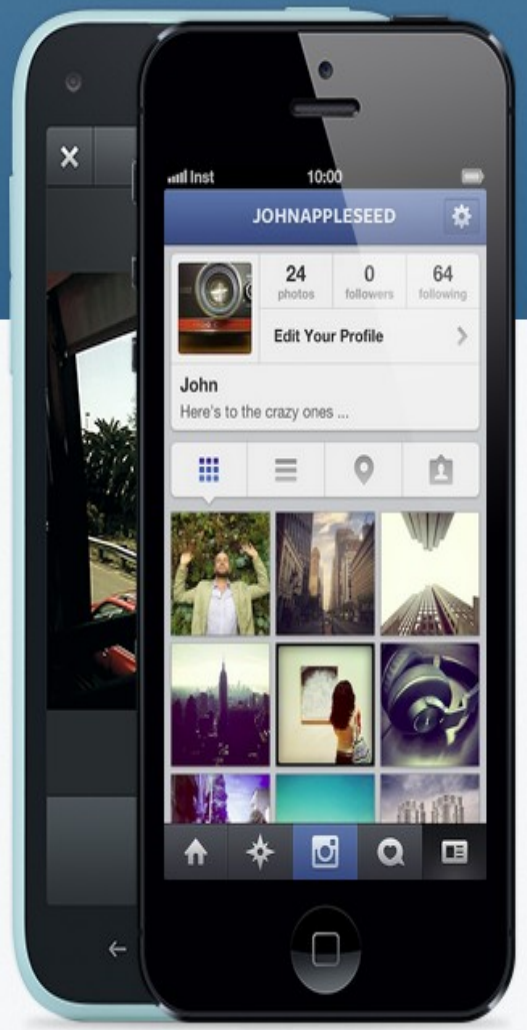
Ac... ... 1 Tue 15 Jul, 15:03 28 °C 9.06KB/s

Instagram – Mozilla Firefox

dding ... | Interac... | Impres... | WordP... | Integra... | How T... | cubew... | Exampl... | WordP... | Tools ‹ ... | Home ... | Instagram ×

instagram.com/#

Google



Instagram

Log in

Capture and Share the World's Moments

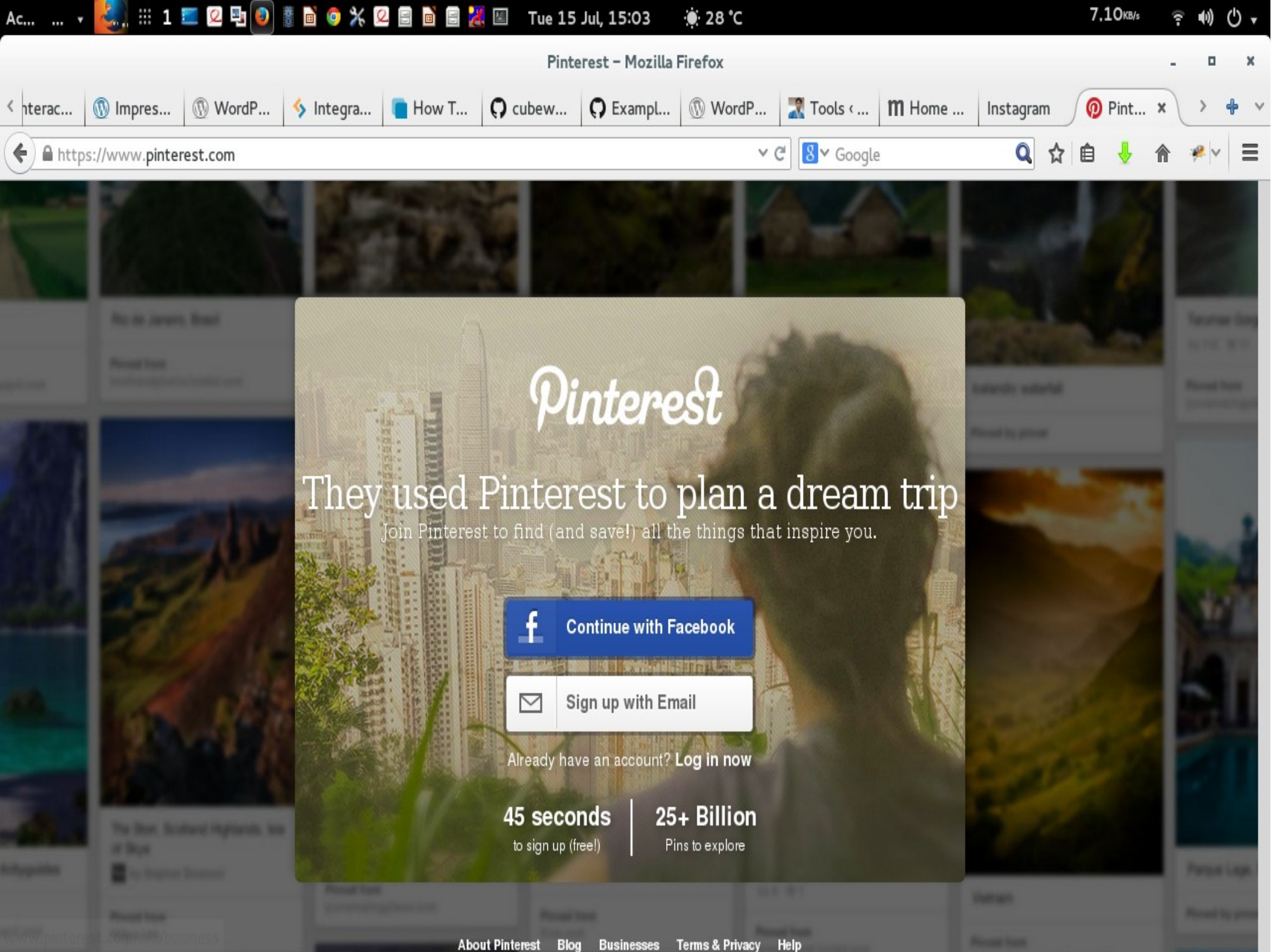
Instagram is a **fast**, **beautiful** and **fun** way to share your life with friends and family.

Take a picture or video, choose a filter to transform its look and feel, then post to Instagram — it's that easy. You can even share to Facebook, Twitter, Tumblr and more. It's a new way to see the world.

Oh yeah, did we mention it's free?

Download on the App Store

GET IT ON Google play


Tue 15 Jul, 15:03
28 °C
Pinterest – Mozilla Firefox
https://www.pinterest.com
Pinterest
They used Pinterest to plan a dream trip
Join Pinterest to find (and save!) all the things that inspire you.
Continue with Facebook
Sign up with Email
Already have an account? Log in now
45 seconds
to sign up (free!)
25+ Billion
Pins to explore
About Pinterest
Blog
Businesses
Terms & Privacy
Help

Science

AstraZeneca

Collaborative drug discovery

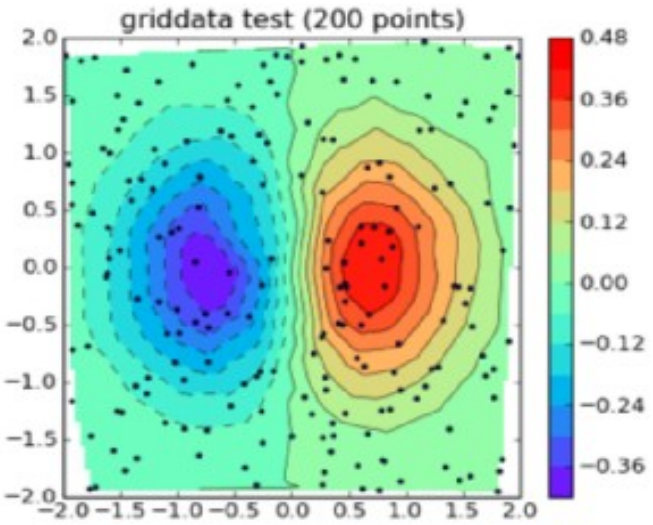
Forecasting

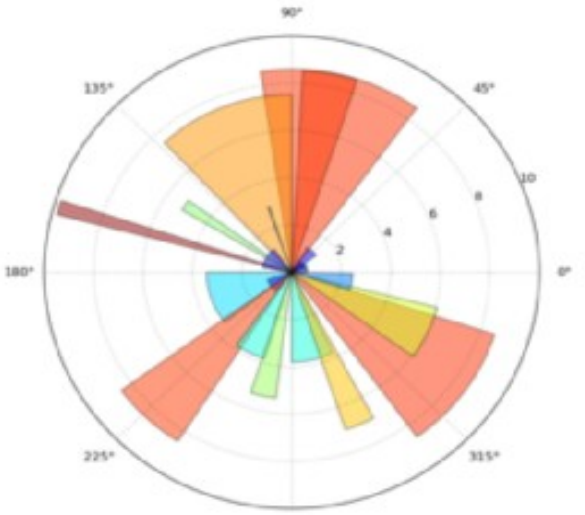
Glue language for many platforms

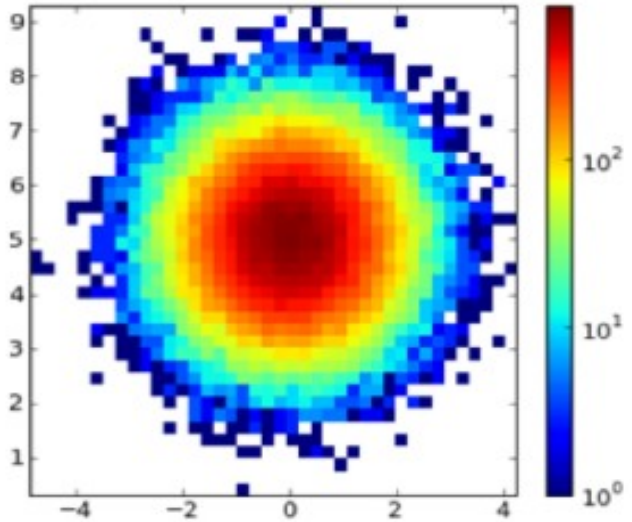
Los Alamos
NATIONAL LABORATORY

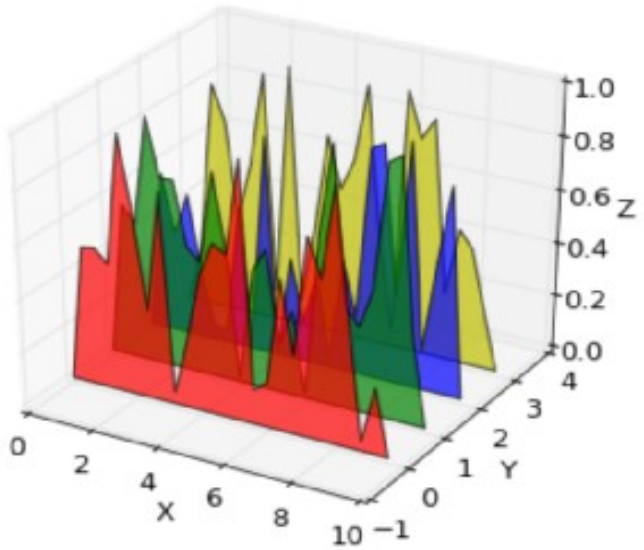
Large-scale physics simulations

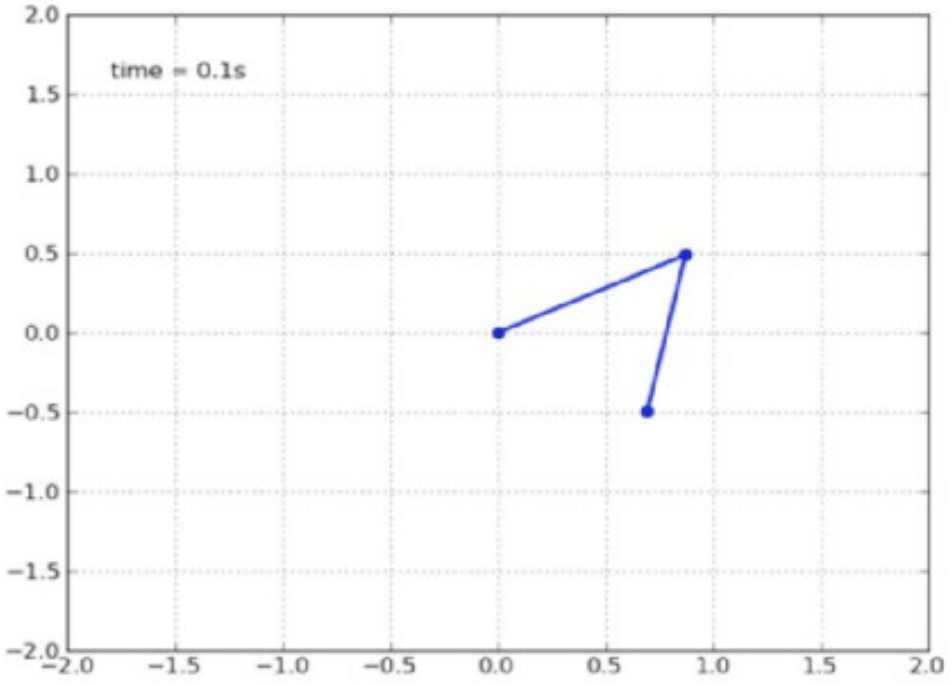
matplotlib







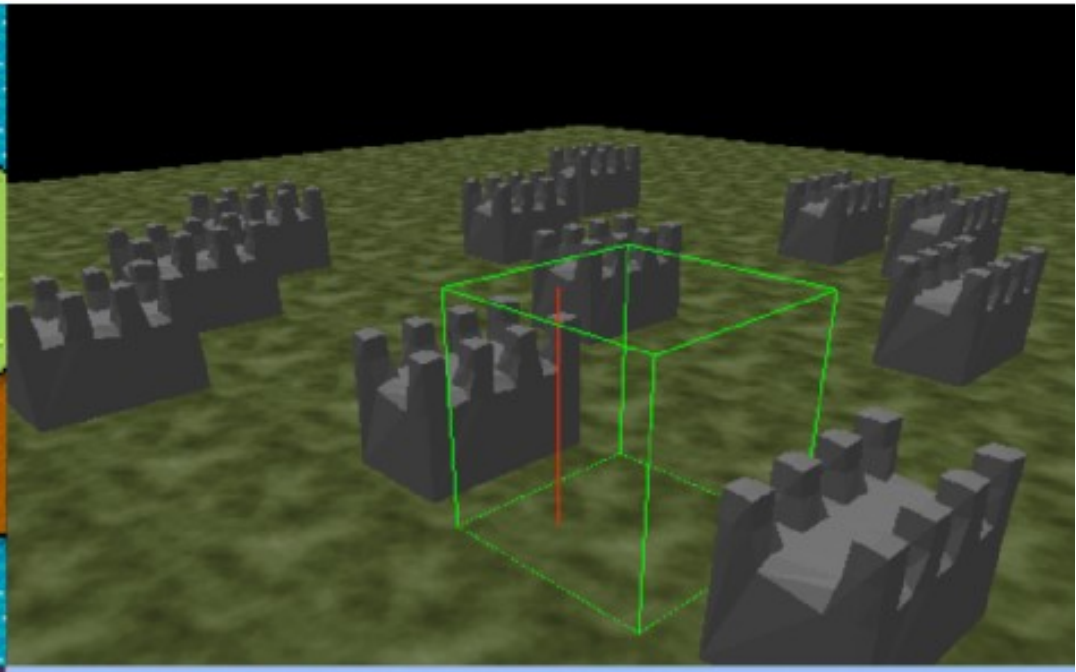




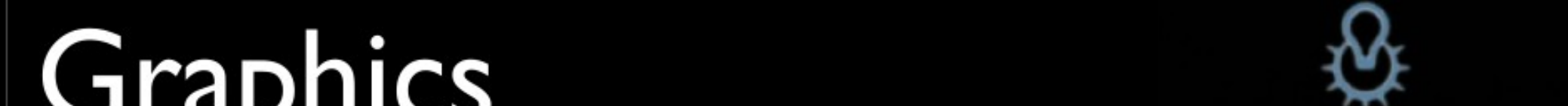
http://matplotlib.org/

Games

http://pygame.org



Graphics



"Python plays a key role in our production pipeline. Without it a project the size of Star Wars: Episode II would have been very difficult to pull off. From crowd rendering to batch processing to compositing, Python binds all things together." - Tommy Burnette, Senior Technical Director

Finance

Python powers the New York Stock Exchange's web based transaction system.

Business

rackspace HOSTING

"CORE" CRM and ERP platform

NOKIA

High-level programming environment for Symbian

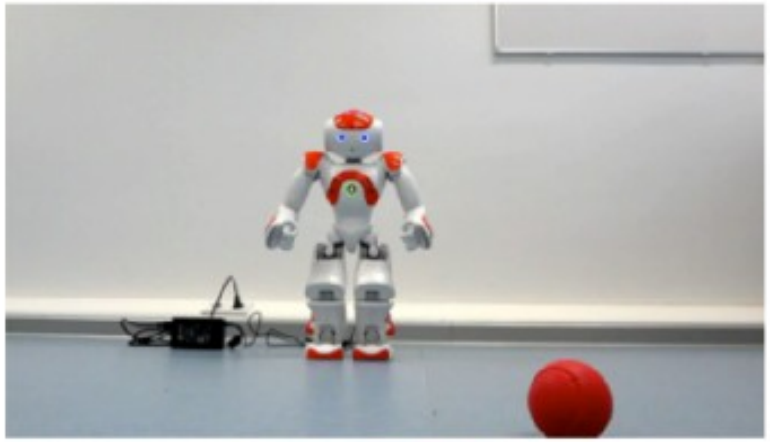
Dropbox

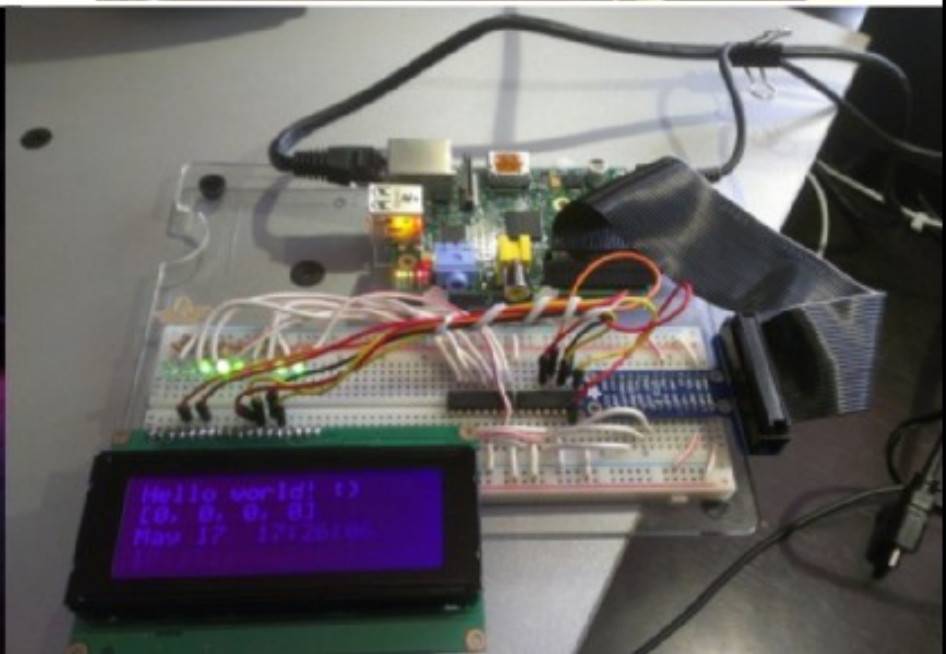
Desktop client

redhat.

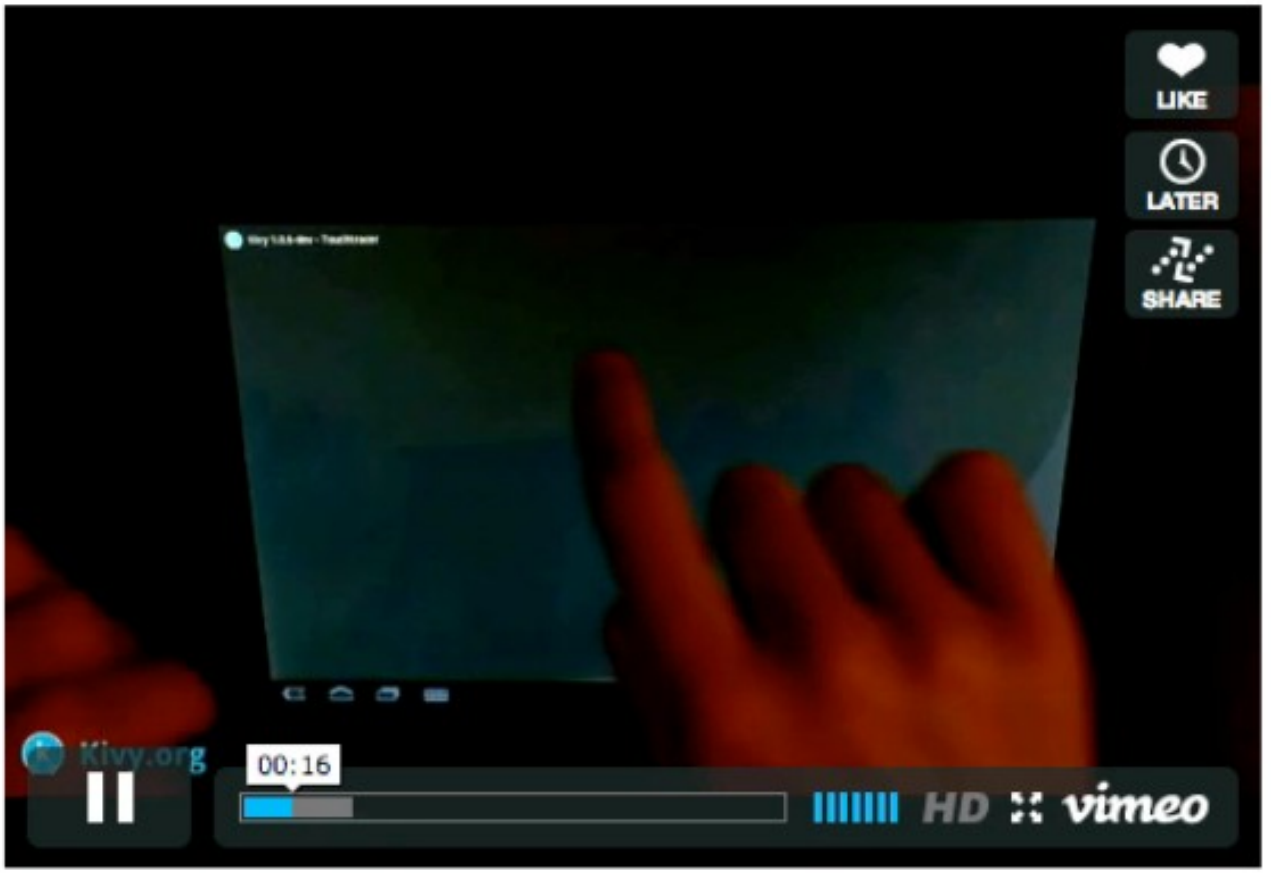
Installer and configuration utilities

Hardware





Even mobile



A multi-touch Android app written in Kivy.

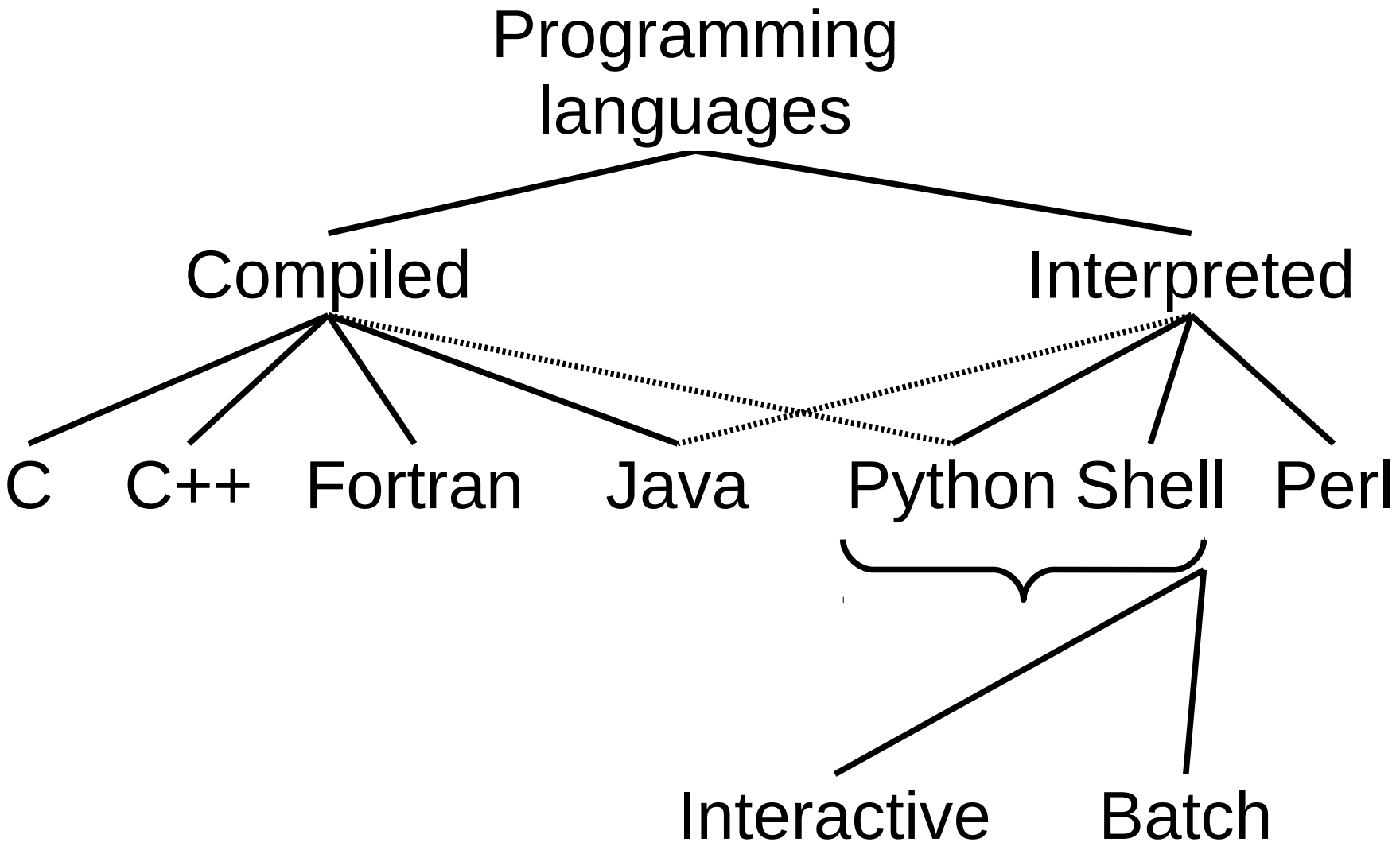
Python is fast enough...

- CPython: the default / reference
  - implementation for the Python bytecode interpreter
- Written in C!

...and the ecosystem makes it even faster

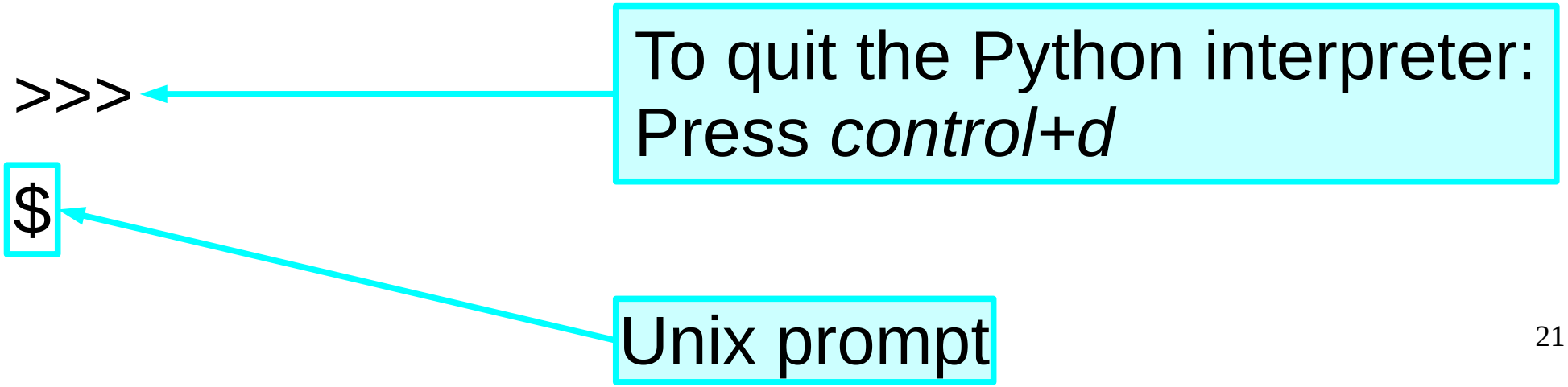
- PyPy: a Python implementation with a Just-In-Time compiler

- C extensions
- Highly-optimized domain libraries


Programming languages
Compiled
Interpreted
C
C++
Fortran
Java
Python
Shell
Perl
Interactive
Batch


Python 2 or Python 3 ?

```
$ python
Python 2.7.5 (default, Nov 12 2013, 16:45:58)
[GCC 4.8.2 20131017 (Red Hat 4.8.2-1)] on linux2
Type "help", "copyright", "credits" or "license"...

>>> print 'Hello, world!'
Hello, world!

>>> 5
3

>>>

$
```



Batch use

$ **python hello.py**

Hello, world!

Data types

- Numbers/Integers
- String
- List
- Dictionary
- Tuple

$\mathbb{Z}$ Integers

$$\{\ \ldots -2, -1, 0, 1, 2, 3, \ldots\ \}$$

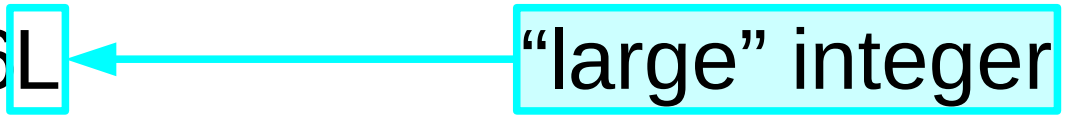
```
>>> 7+3
10
>>> 7-3
4
>>> 7*3
21
>>> 7**3
343
>>> 7/3
2
>>> -7/3
-3
>>> 7%3
1
>>> -7%3
2
```

7^3: use "**" for exponentiation

integer division rounds down

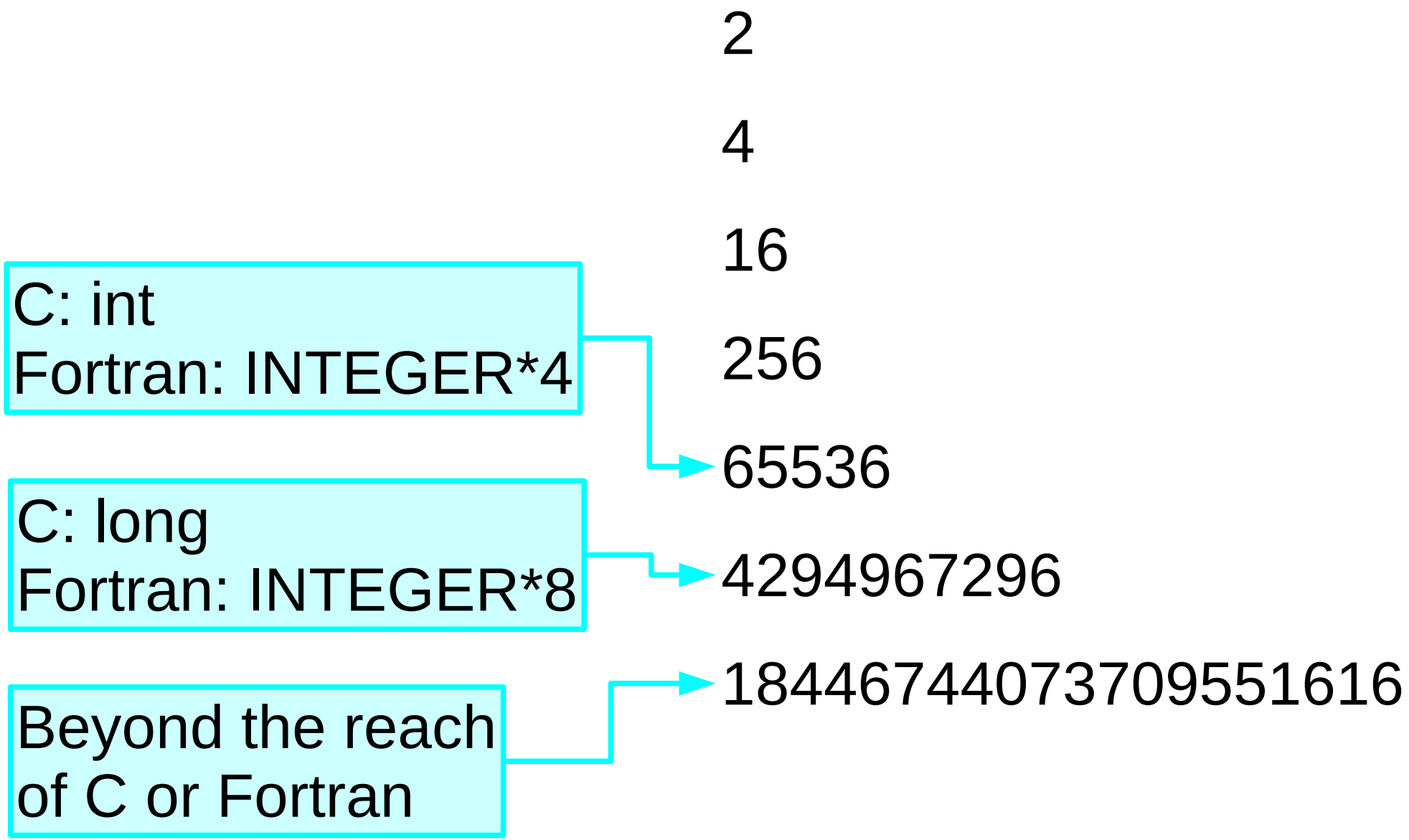
remainder (mod) returns 0 or positive integer

```
>>> 2*2
4

>>> 4*4
16

>>> 16*16
256

>>> 256*256
65536

>>> 65536*65536
4294967296L
```



```
>>> 4294967296*4294967296
18446744073709551616L

>>> 18446744073709551616 *
18446744073709551616
340282366920938463463374607431768211456L

>>> 2**521 - 1
6864797660130609714981900799081393217269
4353001433054093944634591855431833976560
5212255964066145455497729631139148085803
7121987999716643812574028291115057151L
```

No inherent limit to Python's integer arithmetic: can keep going until we run out of memory


2
4
16
256
65536
4294967296
18446744073709551616
C: int
Fortran: INTEGER*4
C: long
Fortran: INTEGER*8
Beyond the reach
of C or Fortran

Floating point numbers

```
>>> 1.0
1.0

>>> 0.5
0.5

>>> 0.25
0.25

>>> 0.1
0.10000000000000001
```

Floating point number

½ is OK

¼ is OK

Powers of two

1/10 is *not*

Usual issues with representation in base 2

```
>>> 2.0*2.0
4.0
>>> 4.0*4.0
16.0

...

>>> 65536.0*65536.0
4294967296.0
>>> 4294967296.0*4294967296.0
1.8446744073709552e+19
```

17 significant figures

>>> **4294967296.0*4294967296.0**
1.8446744073709552e+19

>>> **1.8446744073709552e+19*1.8446744073709552e+19**
3.402823669209384 6e+38

>>> **3.402823669209384 6e+38*3.402823669209384 6e+38**
1.157920892373162e+77

>>> **1.157920892373162e+77*1.157920892373162e+77**
1.340780792994259 7e+154

>>> **1.340780792994259 7e+154*1.340780792994259 7e+154**
inf

overflow

Limit at 2**1023

Machine epsilon

```
>>> 1.0 + 1.0e-16
1.0
>>> 1.0 + 2.0e-16
1.0000000000000002
```

too small to make a difference

large enough

```
>>> 1.0 + 1.1e-16
1.0
>>> 1.0 + 1.9e-16
1.0000000000000002
```

Spend the next few minutes using Python interactively to estimate machine epsilon – we'll write a Python program to do this for us a little later

Mathematical Functions

Function	Returns (description)
abs(x)	The absolute value of x: the (positive) distance between x and zero.
ceil(x)	The ceiling of x: the smallest integer not less than x
cmp(x, y)	-1 if x < y, 0 if x == y, or 1 if x > y
exp(x)	The exponential of x: e^x
fabs(x)	The absolute value of x.
floor(x)	The floor of x: the largest integer not greater than x
log(x)	The natural logarithm of x, for x> 0
log10(x)	The base-10 logarithm of x for x> 0 .
max(x1, x2,...)	The largest of its arguments: the value closest to positive infinity
min(x1, x2,...)	The smallest of its arguments: the value closest to negative infinity
modf(x)	The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
pow(x, y)	The value of x**y.
round(x [,n])	**x** rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
sqrt(x)	The square root of x for x > 0

Multiple assignments in a single line

```
a , b = 1, 2
```

format

```
a=3
b='python'

print ”{} version is {}”.format(b,a)
```

Strings

'Hello, world!'

'''Hello, world!'''

"Hello, world!"

"""Hello, world!"""

Single quotes

'Hello, world!'

Single quotes around the string

Double quotes

"Hello, world!"

Double quotes around the string

Exactly equivalent

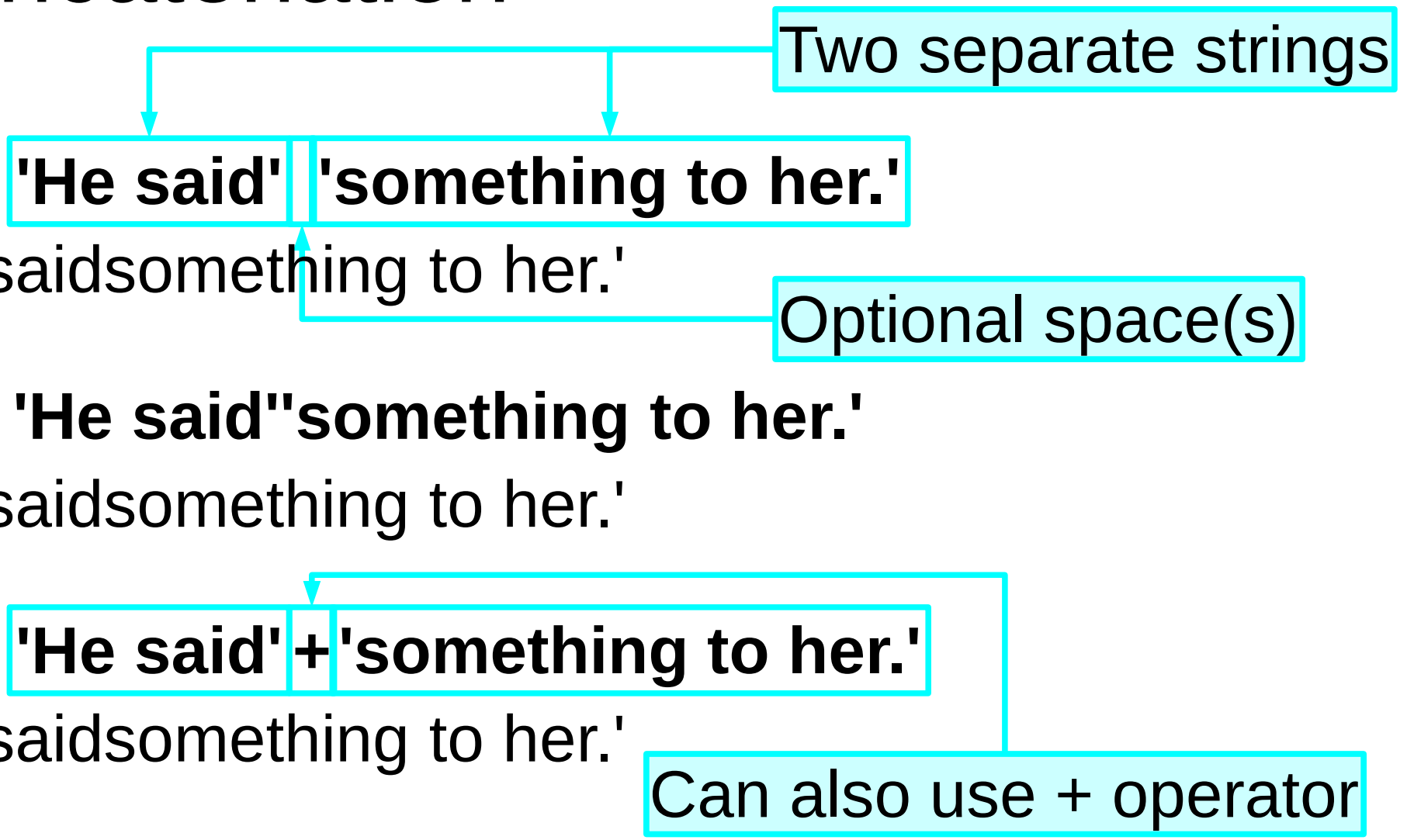
'He said "Python " is awesome'

```
>>> print 'He said "Python!"is awesome'
He said "Python!"is awesome
```

"He said 'Python' Is awesome "

```
>>> print "He said 'Python!' is awesome"
He said 'Python!' is awesome
```

String concatenation



Special characters

\n → ↵

\t → →|

\a → ♫

\\ → \

\' → '

\" → "

```
>>> print 'Hello,\n world!'
Hello,
world!
```

"\n" converted to "new line"

Long strings

Triple double quotes

"""Long pieces of
text are easier to
handle if literal new
lines can be
embedded in them."""

Long strings

Triple single quotes

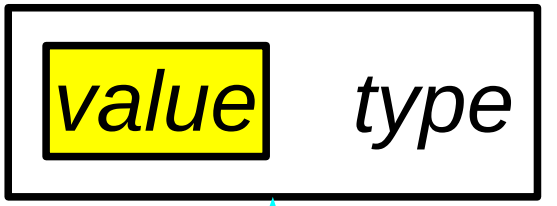
'''Long pieces of
text are easier to
handle if literal new
lines can be
embedded in them.'''

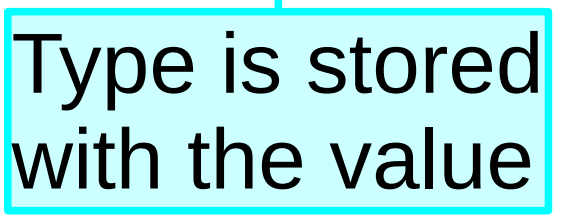
String Methods Available

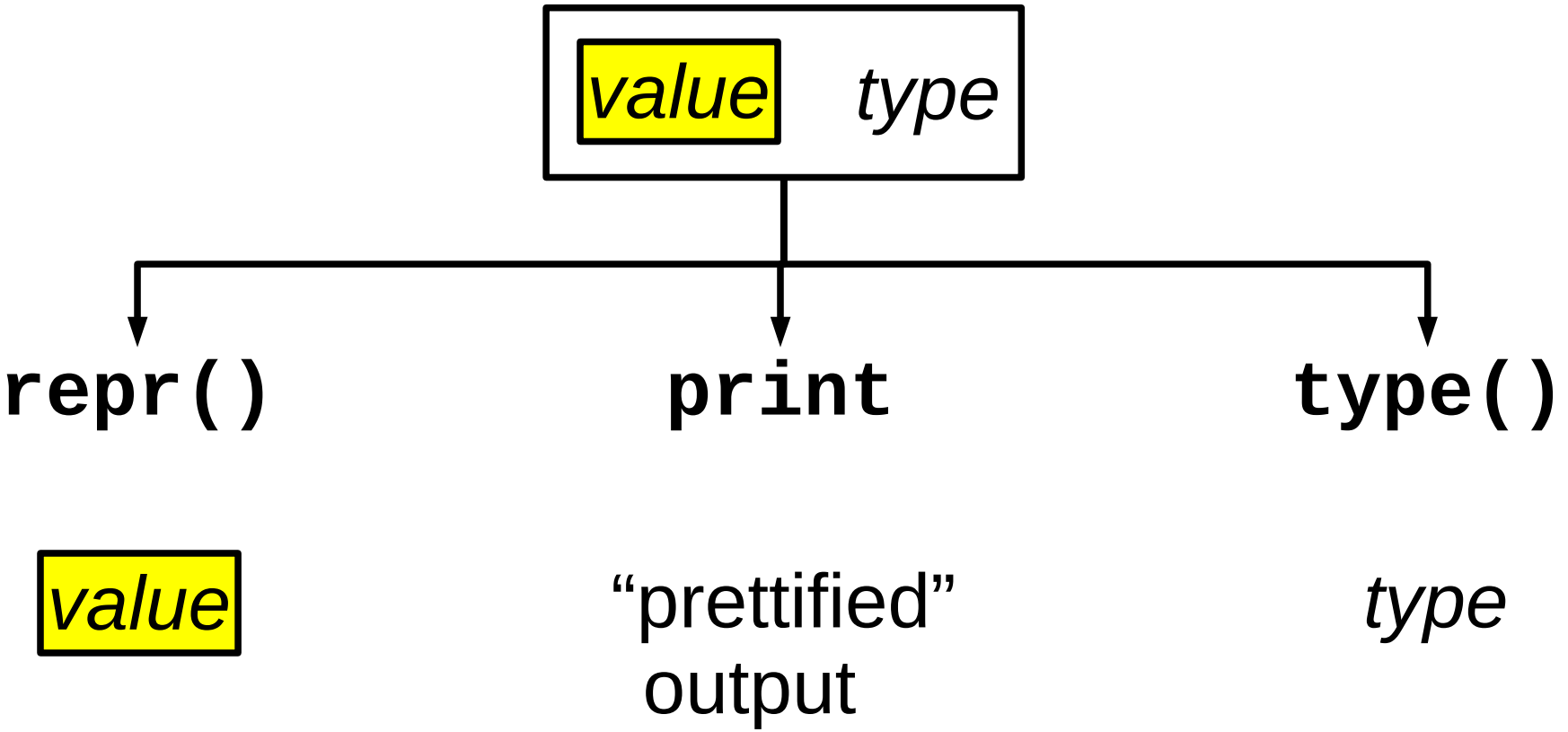
title()
upper()
lower()
swapcase()
isalpha()
isdigit()
islower()
isupper()
istitile()
split()

strip()
lstrip()
rstrip()
find()
startswith()
endswith()
replace()

How Python stores values







```
>>> print 1.2345678901234567
1.23456789012

>>> type( 1.2345678901234567 )
<type 'float'>

>>> repr( 1.2345678901234567 )
'1.2345678901234567'
```

Two other useful types

Complex

```
>>> (1.0 + 2.0j) * (1.5 + 2.5j)
(-3.5+5.5j)
```

Boolean

```
>>> True and False
False

>>> 123 == 234
False
```

Comparisons

```
>>> 1 == 2
False

>>> 1 < 2
True

>>> 1 >= 2
False

>>> 1 == 1.0
True
```

```
>>> 'abc' == 'ABC'
False

>>> 'abc' < 'ABC'
False

>>> 'abc' >= 'ABC'
True
```

… not equal to …

```
>>> not 1 == 2
True
```

```
>>> not 'abc' == 'ABC'
True
```

```
>>> 1 != 2
True
```

```
>>> 'abc' != 'ABC'
True
```

Conjunctions

```
>>>  1 == 2 and 3 == 3
False

>>>  1 == 2 or 3 == 3
True
```

Evaluate the following Python expressions in your head:

```
>>> 2 - 2 == 1 / 2

>>> True and False or True

>>> 1 + 1.0e-16 > 1

>>> 5 == 6 or 2 * 8 == 16

>>> 7 == 7 / 2 * 2

>>> 'AbC' > 'ABC'
```

Now try them interactively in Python and see if you were correct.

Precedence

First

x**y

-6, +6

x/y, x*y, x%y

x+y, x-y

x<y, x<=y, …

x **in** y, x **not in** y

not x

x **and** y

Last x **or** y

Arithmetic operations

Logical operations

Flow control in Python: `if`

```
if x > 0.0 :
    print 'Positive'
```

compulsory

indentation

```
elif x < 0.0 :
    print 'Negative'
    x = -1.0 * x
```

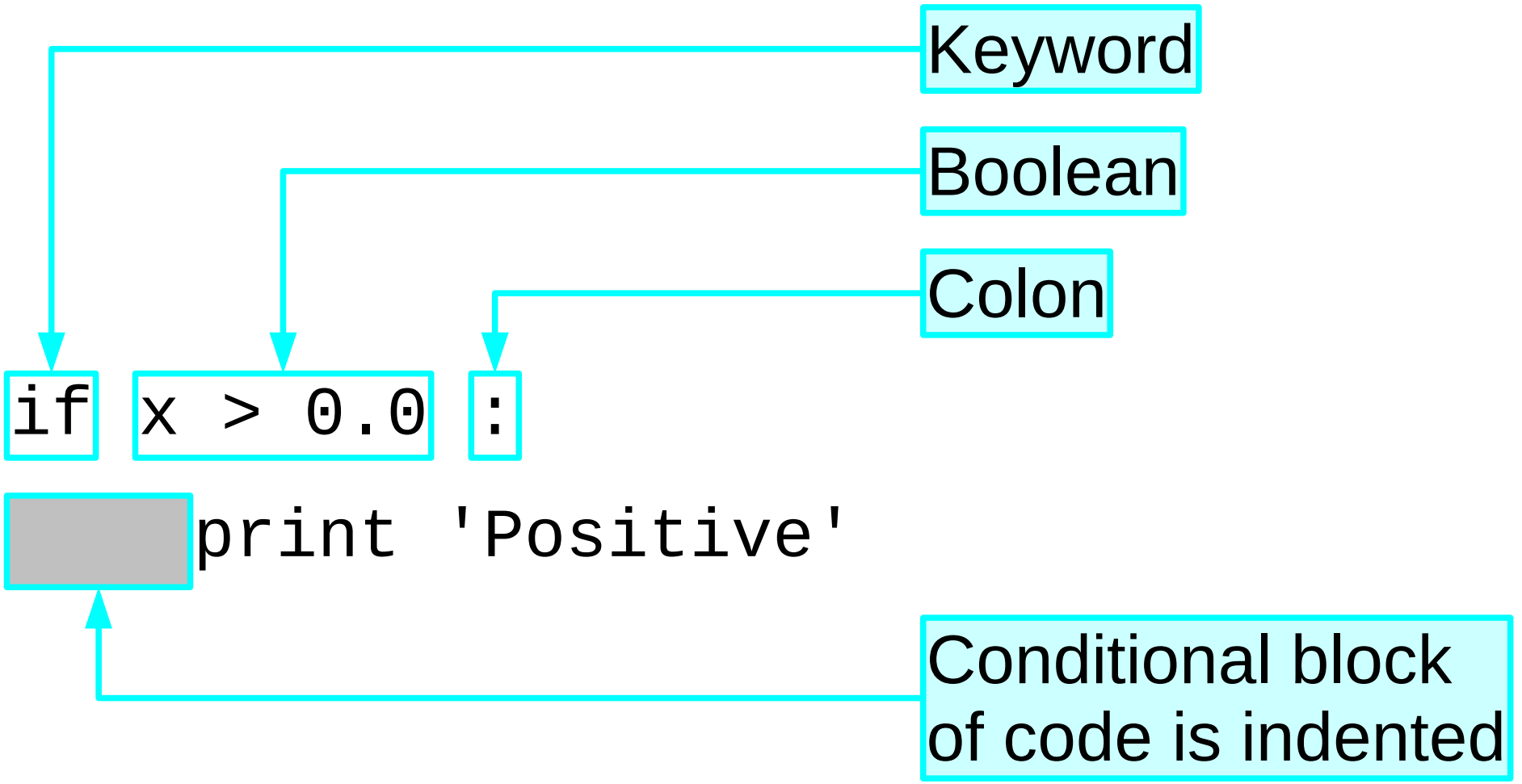
optional, repeatable

multiple lines indented

```
else :
    print 'Zero'
```

optional

Flow control in Python: `if`



Nested indentation

```
if x > 0.0 :
    print 'Positive'
else :
    if x < 0.0 :
        print 'Negative'
        x = -1.0 * x
    else :
        print 'Zero'
```

Flow control in Python: **`while`**

```
while x % 2 == 0 :
    print x, 'still even'
    x = x/2


else :
    print x, 'is odd'
```

compulsory

optional

for loop

```
#!/usr/bin/python
for i in range(1, 5):
  print i
else:
  print 'The for loop is over'
```

range()

range() is a built in class

```
range(stop)
rang(start,stop)
range(start, stop[, step])
```

Remember Fibonacci series ?

1,1,2,3,5,8,13,21.....

break *statement*

continue *statement*

Converting from one type to another

In and out of strings

```
>>> float('0.25')
0.25
```

↔

```
>>> str(0.25)
'0.25'
```

```
>>> int('123')
123
```

↔

```
>>> str(123)
'123'
```

Converting from one type to another

Between numeric types

```
>>> int(12.3)
12
```

loss of precision

```
>>> float(12)
12.0
```

Converting from one type to another

If you treat it like a list…

```
>>> list('abcd')
['a', 'b', 'c', 'd']
>>> list(data)
['line one\n', 'line two\n', 'line three\n', 'line four\n']
>>> list({'H':'hydrogen', 'He':'helium'})
['H', 'He']
```

Time for a break…

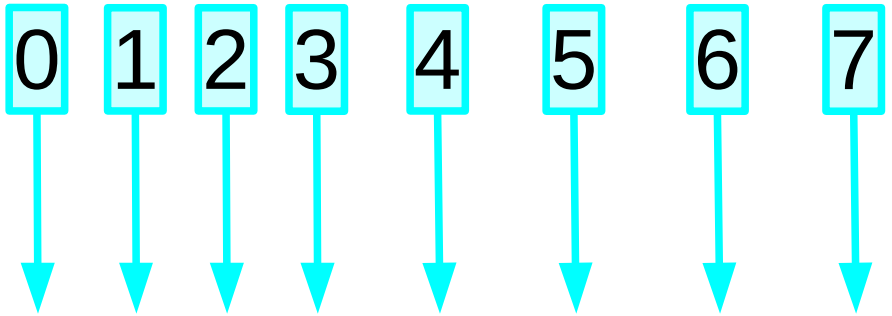
January February March April May June July
August September October November December

Lists

H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

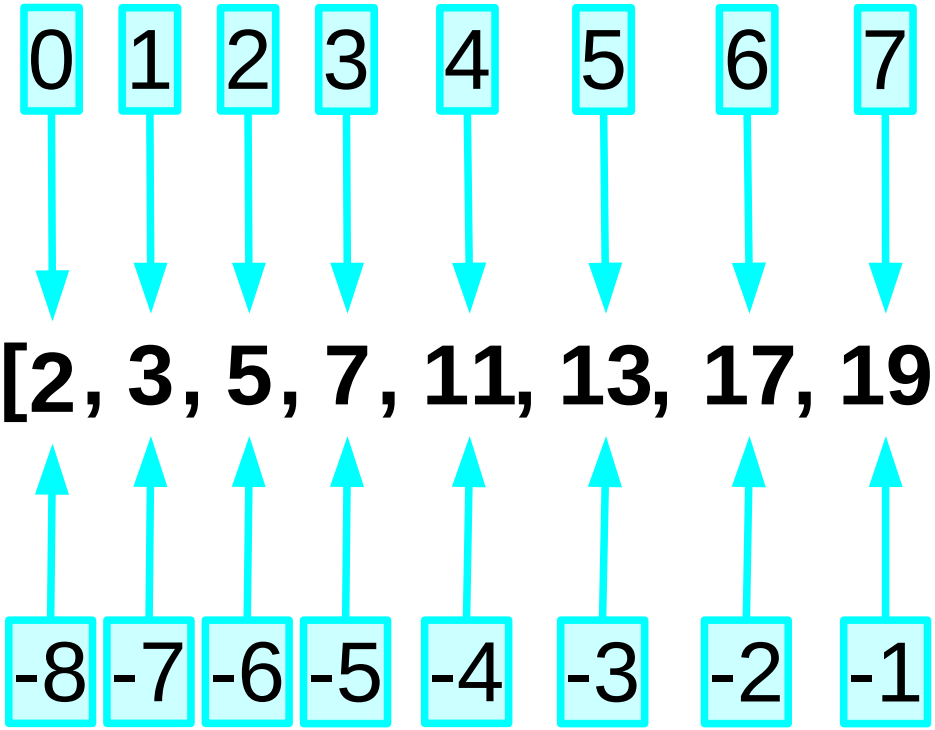
Red Orange Yellow Blue Indigo Violet

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

>>> type([ 2, 3, 5, 7, 11, 13, 17, 19])
<type 'list'>

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



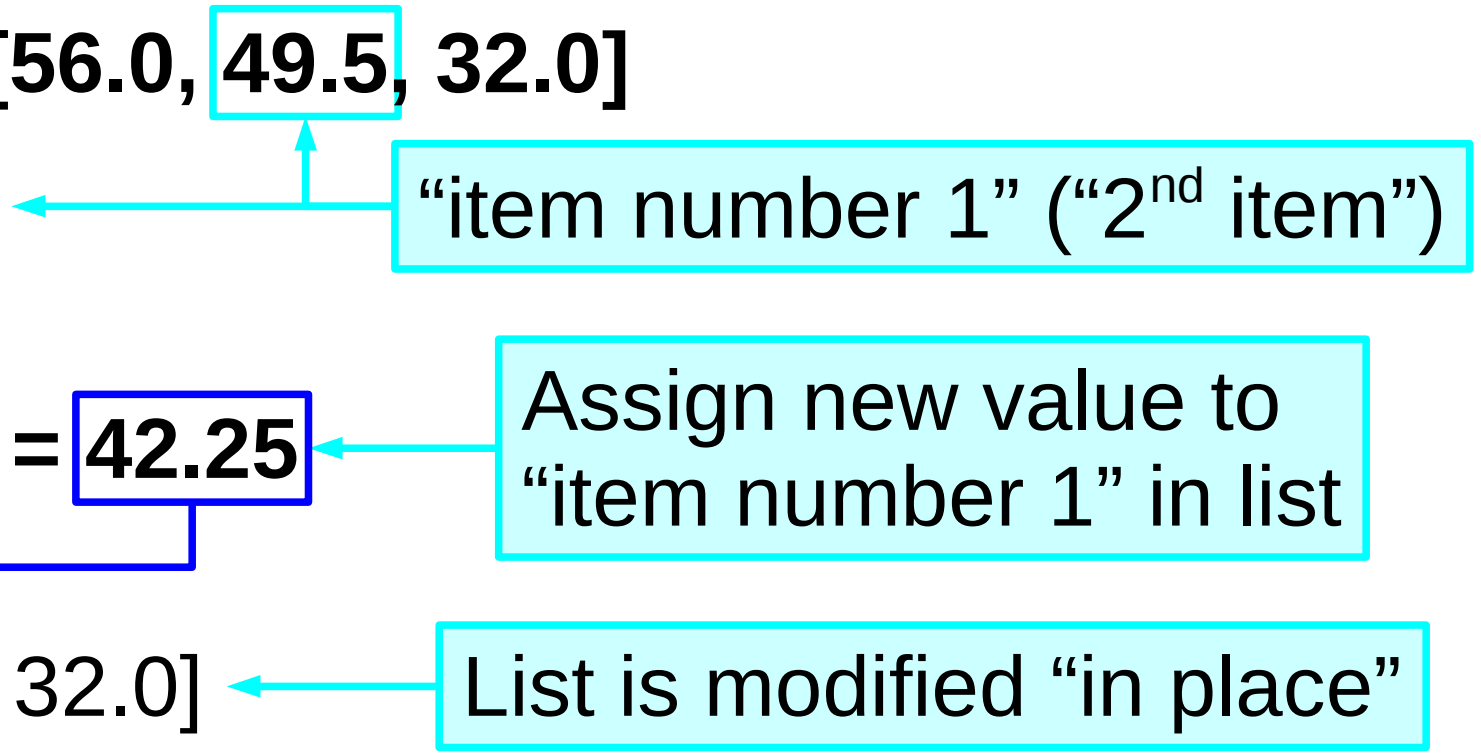

```
>>> primes = [ 2, 3, 5, 7, 11, 13, 17, 19]

>>> primes[2]
5
```

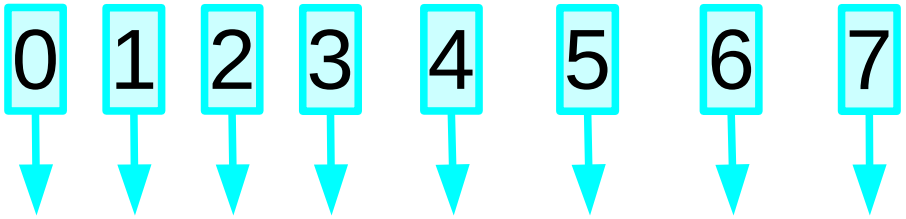
Indexing starts at 0


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

Changing an item in a list

```
>>> data = [56.0, 49.5, 32.0]
>>> data[1]
49.5
>>> data[1] = 42.25
>>> data
[56.0, 42.25, 32.0]
```






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

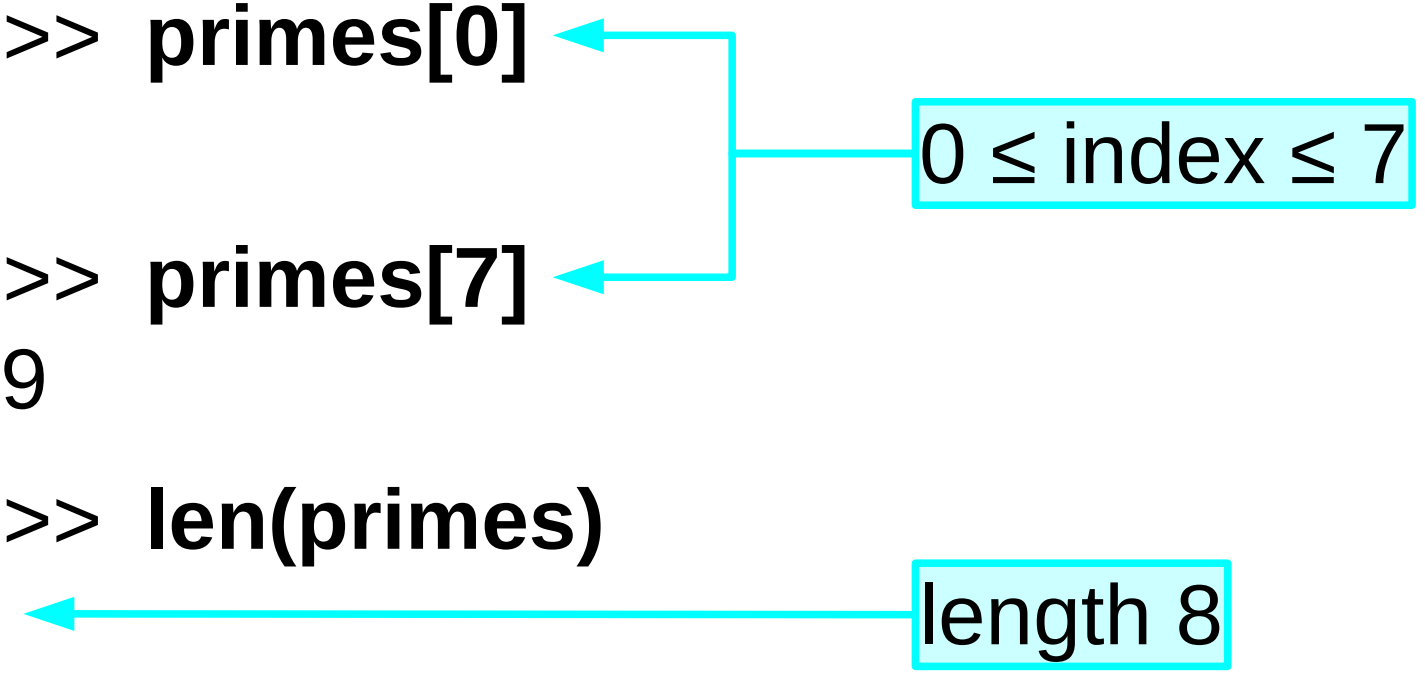
```
>>> primes[8]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Where the error was.

The error message.

Counting from zero and the `len()` function

```
>>> primes = [2, 3, 5, 7, 11, 13, 17, 19]
>>> primes[0]
2
>>> primes[7]
19
>>> len(primes)
8
```



Empty lists

```
>>> empty = [ ]

>>> len(empty)
0

>>> len([ ])
0
```

Single item lists

A list with one item is not the same as the item itself!

```
>>> [1234] == 1234
False
```

```
>>> type([1234])
<type 'list'>

>>> type(1234)
<type 'int'>
```

Lists of anything

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

List of integers

```
names = [ 'Alice', 'Bob', 'Cathy', 'Dave' ]
```

List of strings

```
roots = [ 0.0,  1.57079632679,   3.14159265359 ]
```

List of floats

```
lists = [ [ 1, 2, 3 ], [5], [9, 1] ]
```

List of *lists*

Mixed lists

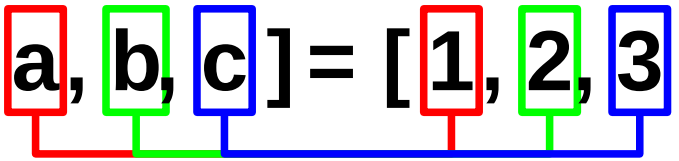
```
stuff = [ 2, 'Bob', 3.14159265359, 'Dave' ]
```

Legal, but not a good idea.

See “tuples” later.

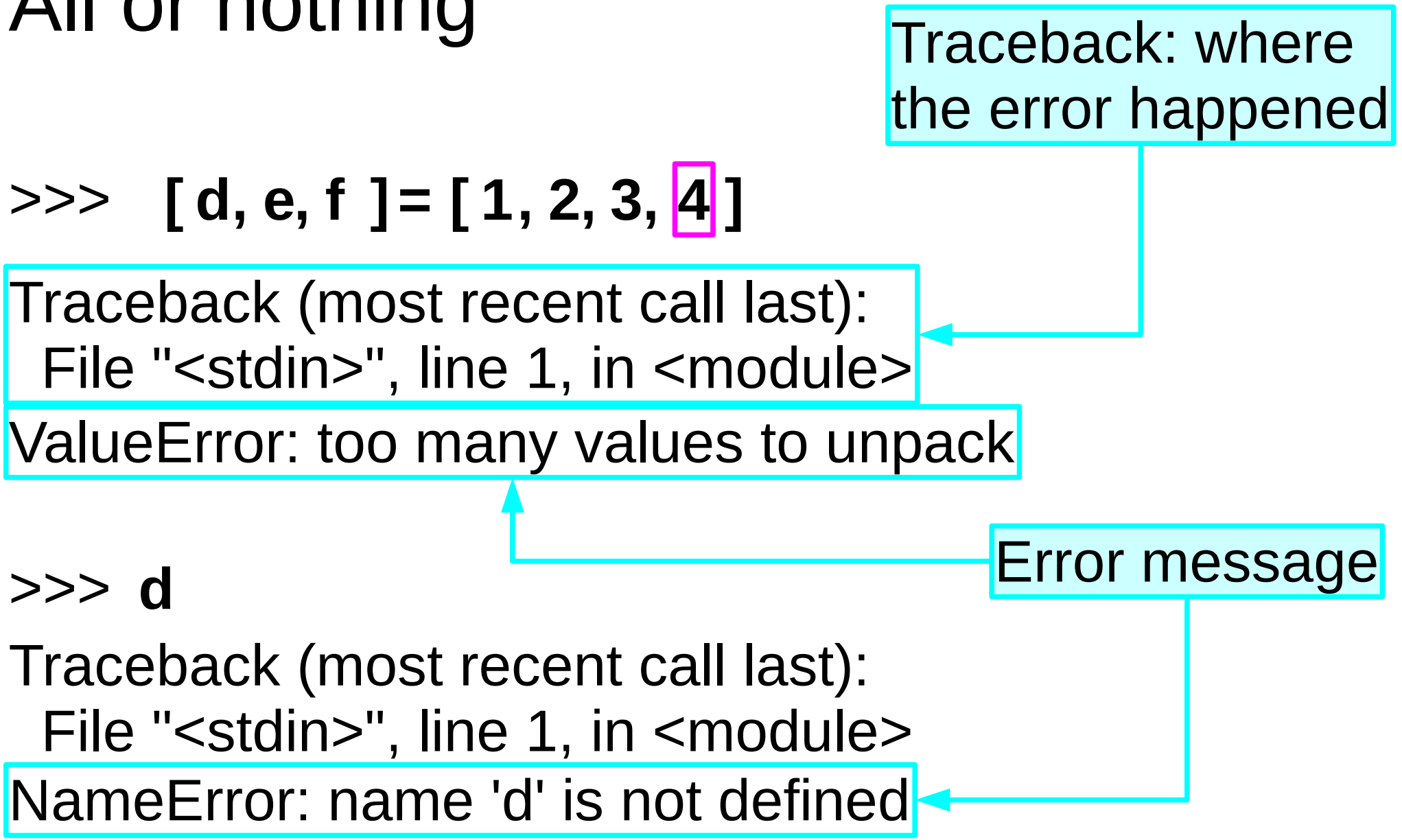
Lists of variables

```
>>>  [a, b, c] = [1, 2, 3]
```



```
>>> a
1
>>> b
2
>>> c
3
```

All or nothing

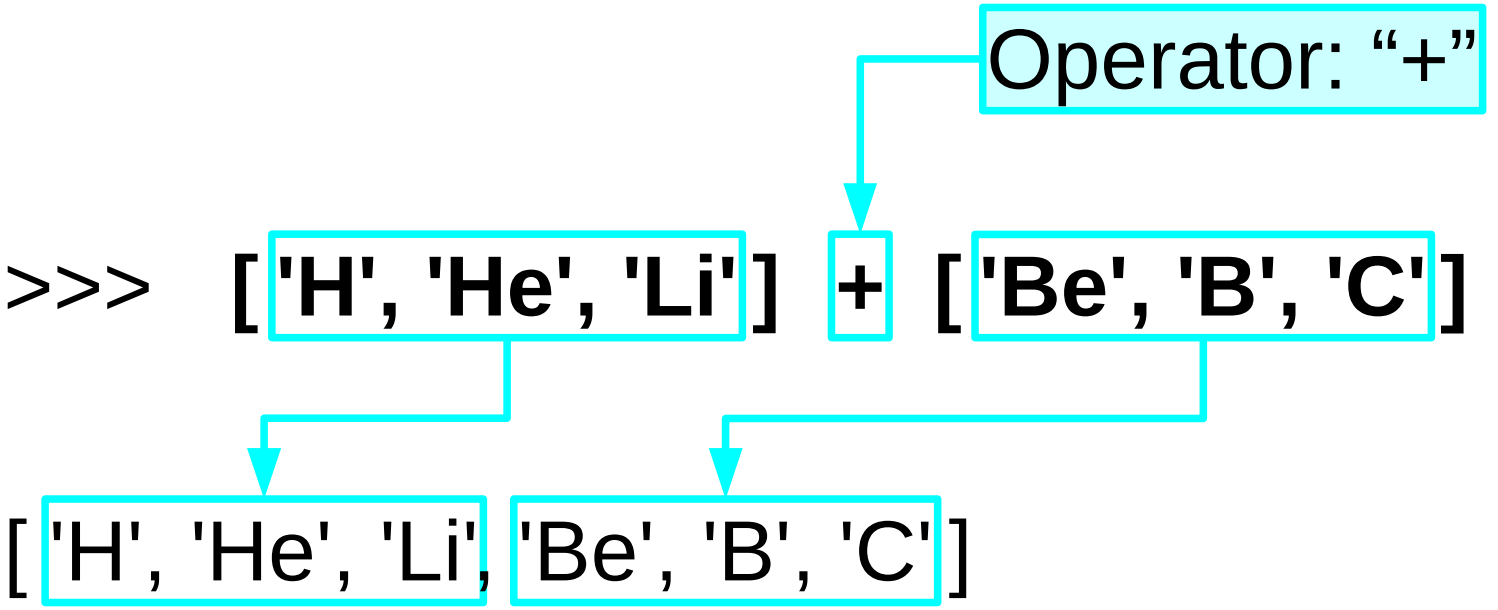


All or nothing

```
>>>  [ g, h, i , j ] = [ 1, 2, 3 ]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: need more than 3 values to unpack

>>> g
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'g' is not defined
```

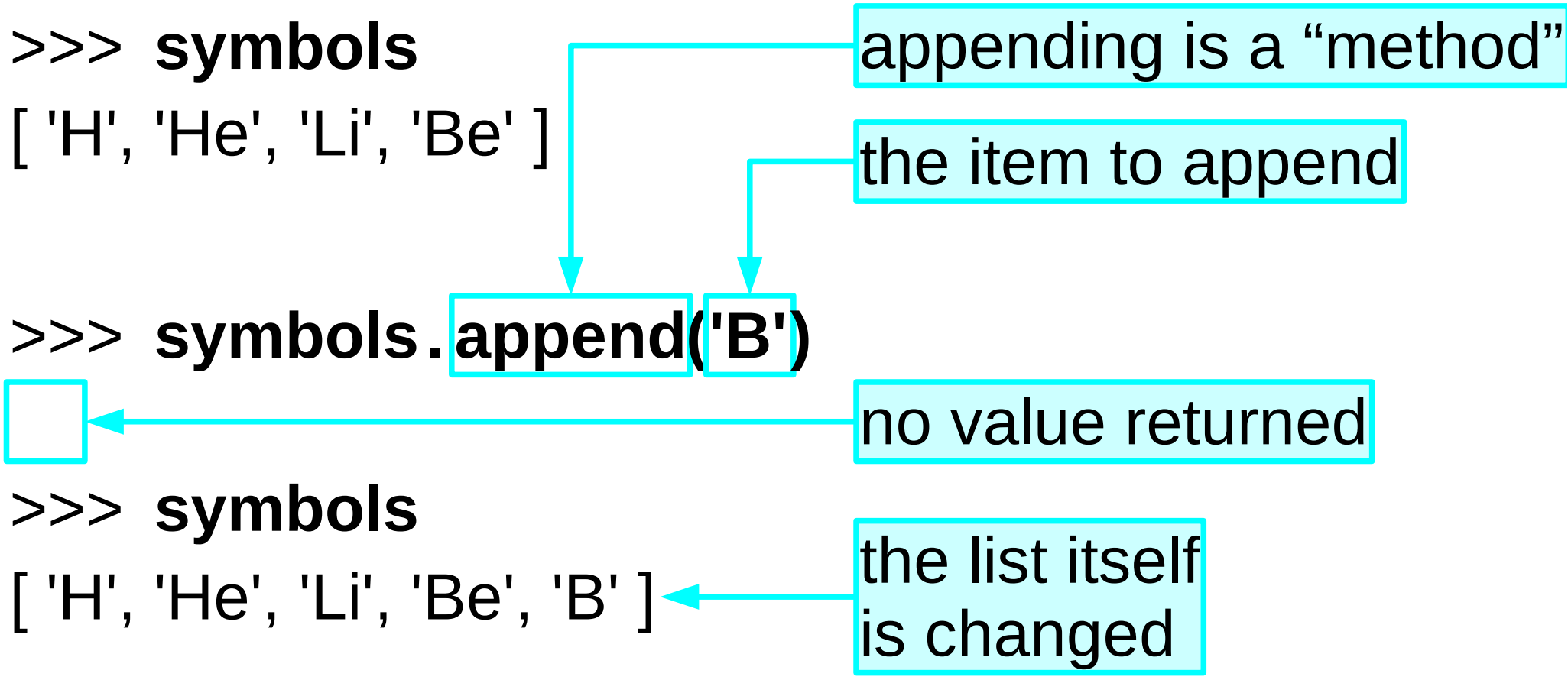
Error message

Concatenating lists

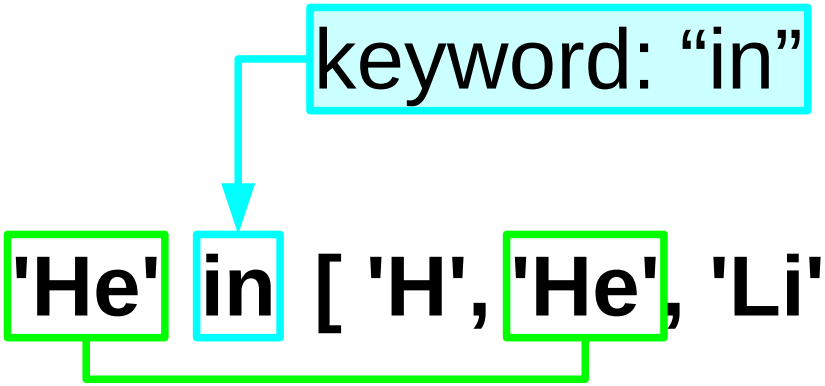


Appending an item: **append( )**

```
>>> symbols = [ 'H', 'He', 'Li', 'Be' ]
>>> symbols
[ 'H', 'He', 'Li', 'Be' ]
>>> symbols.append('B')
>>> symbols
[ 'H', 'He', 'Li', 'Be', 'B' ]
```



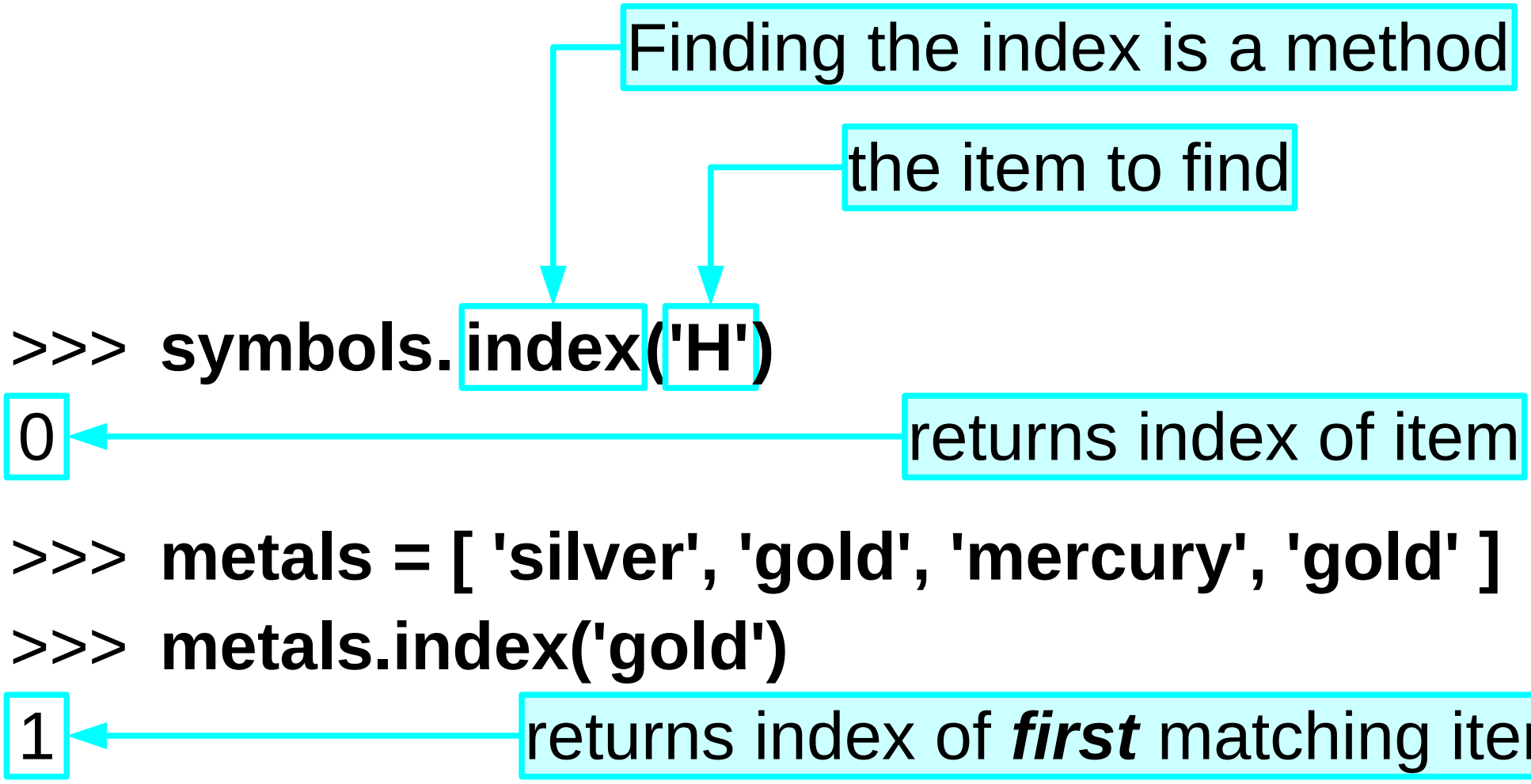
Membership of lists



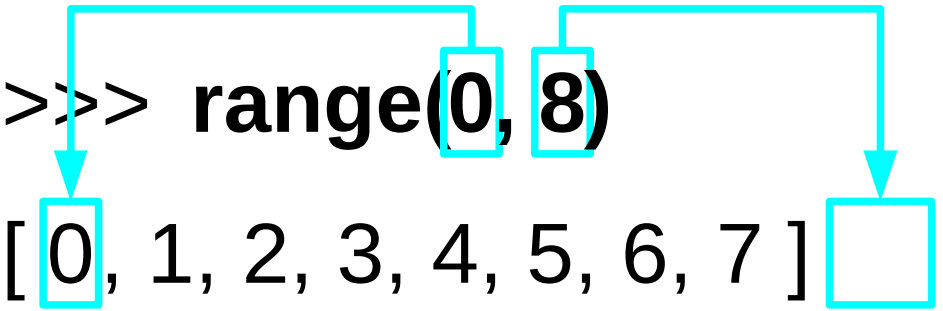

```
>>>  'He' in [ 'H', 'He', 'Li' ]
True

>>>  'He' in [ 'Be', 'B', 'C' ]
False
```

Finding the index of an item

```
>>> symbols =  [ 'H', 'He', 'Li', 'Be' ]
```



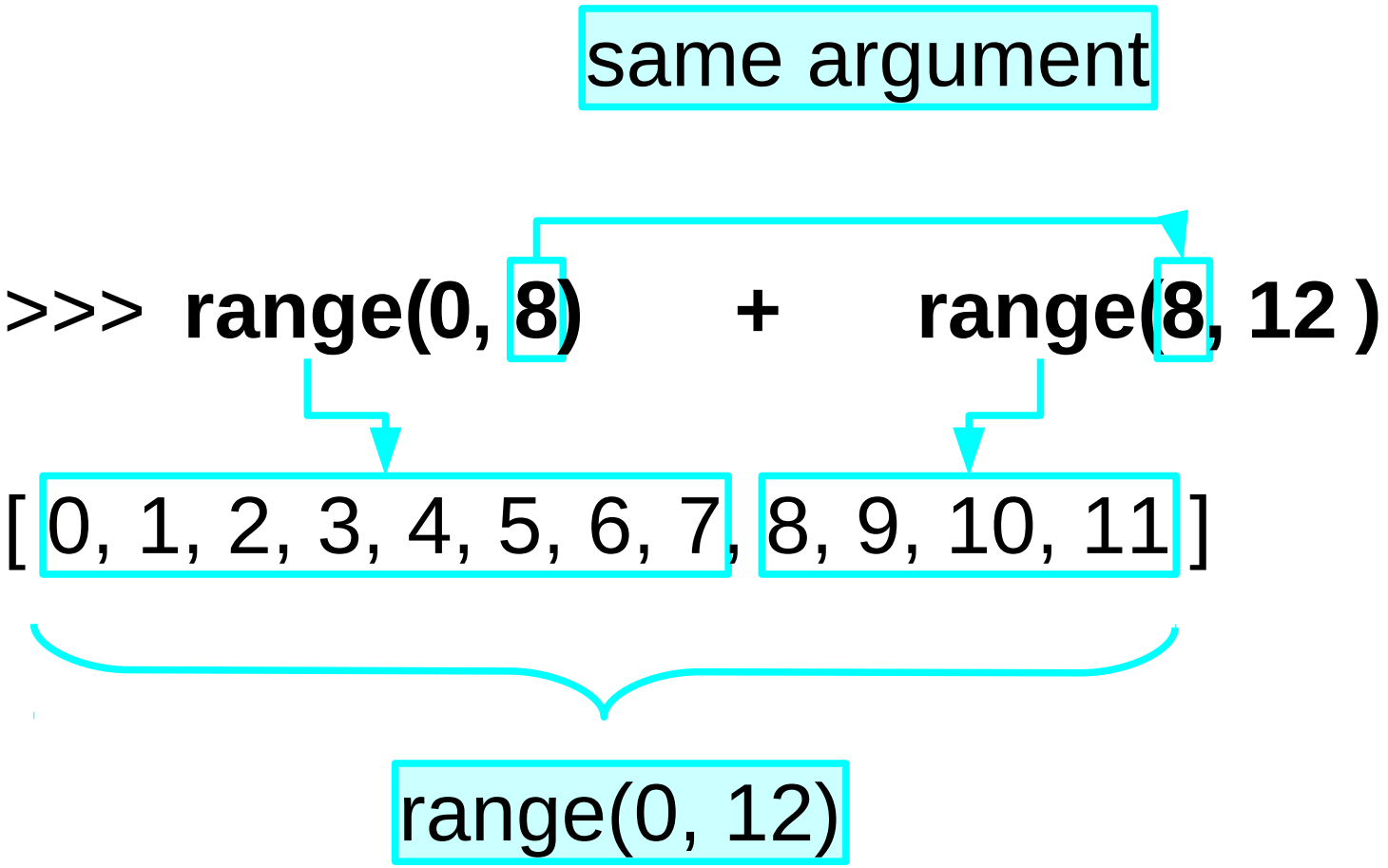
Functions that give lists: **`range()`**



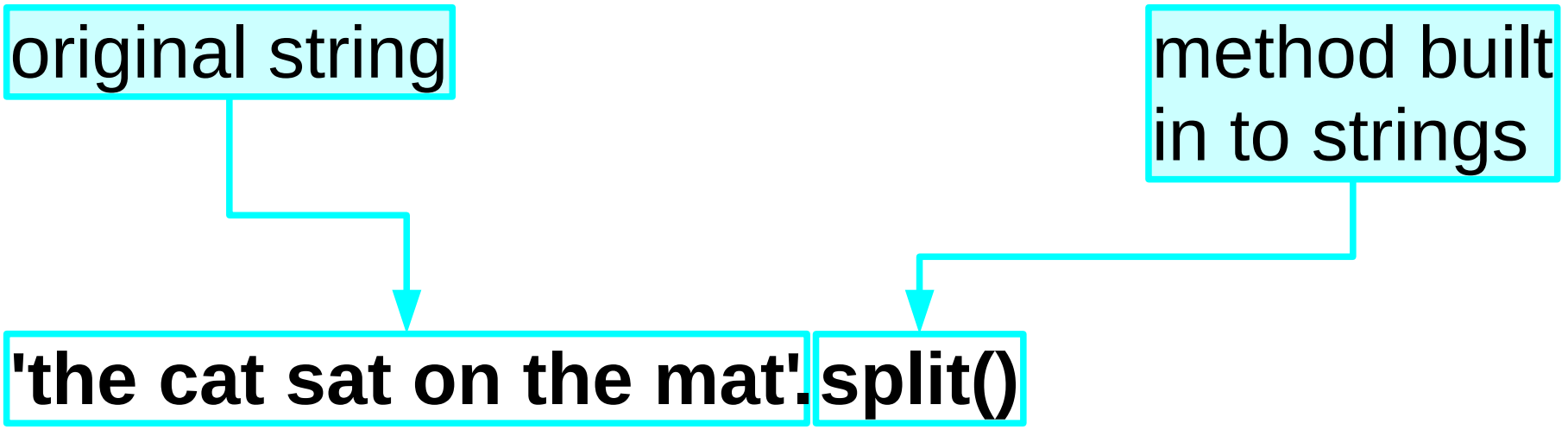
First integer in list

One beyond last integer in list

`range()`: Why miss the last number?



Functions that give lists: **`split()`**



['the', 'cat', 'sat', 'on', 'the', 'mat']

Split on white space

Spaces discarded

`split()`: Only good for trivial splitting

```
>>> 'the cat sat on the mat'.split()
[ 'the', 'cat', 'sat', 'on', 'the', 'mat' ]
```

Split on white space

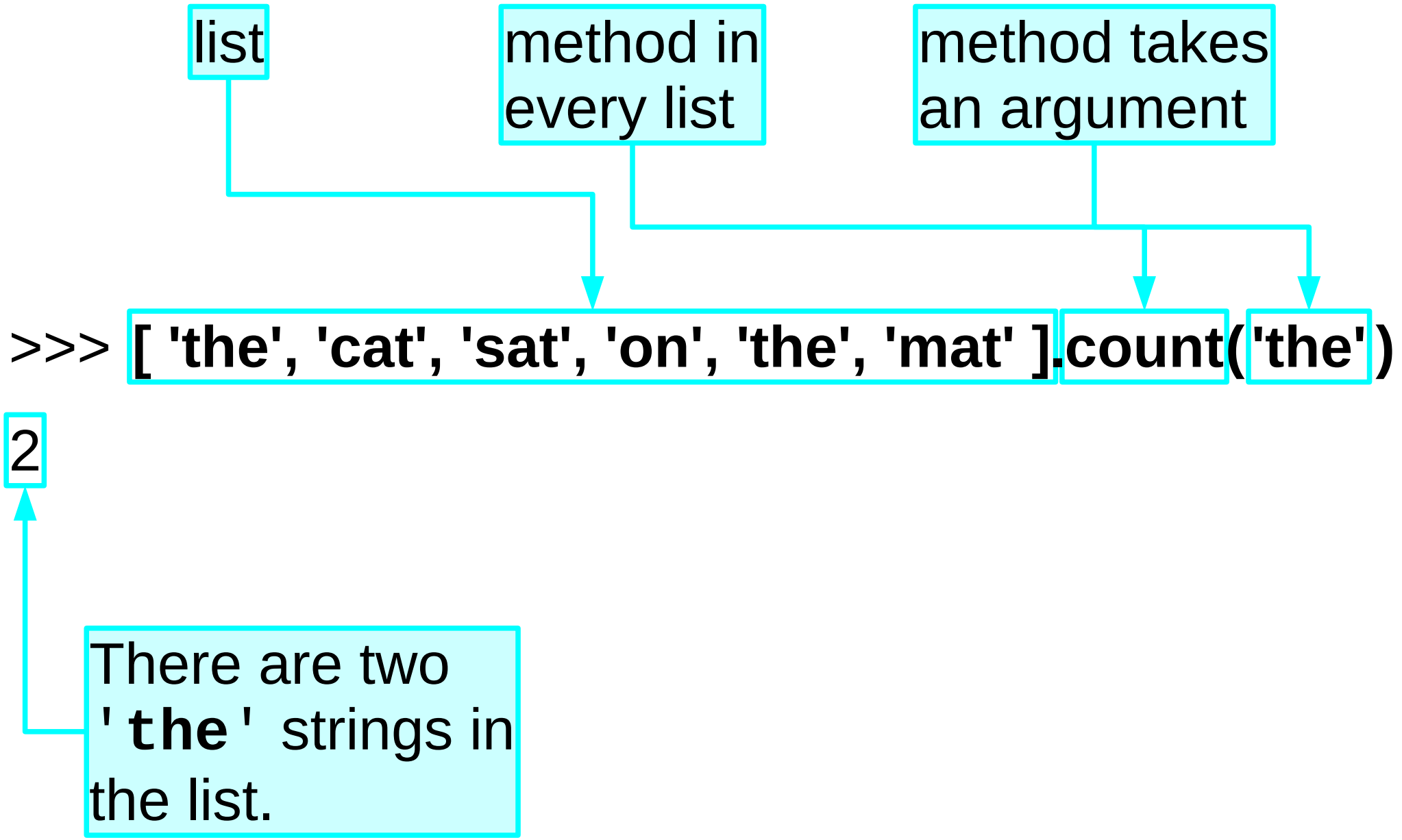
Spaces discarded

Trivial operation

Regular expressions

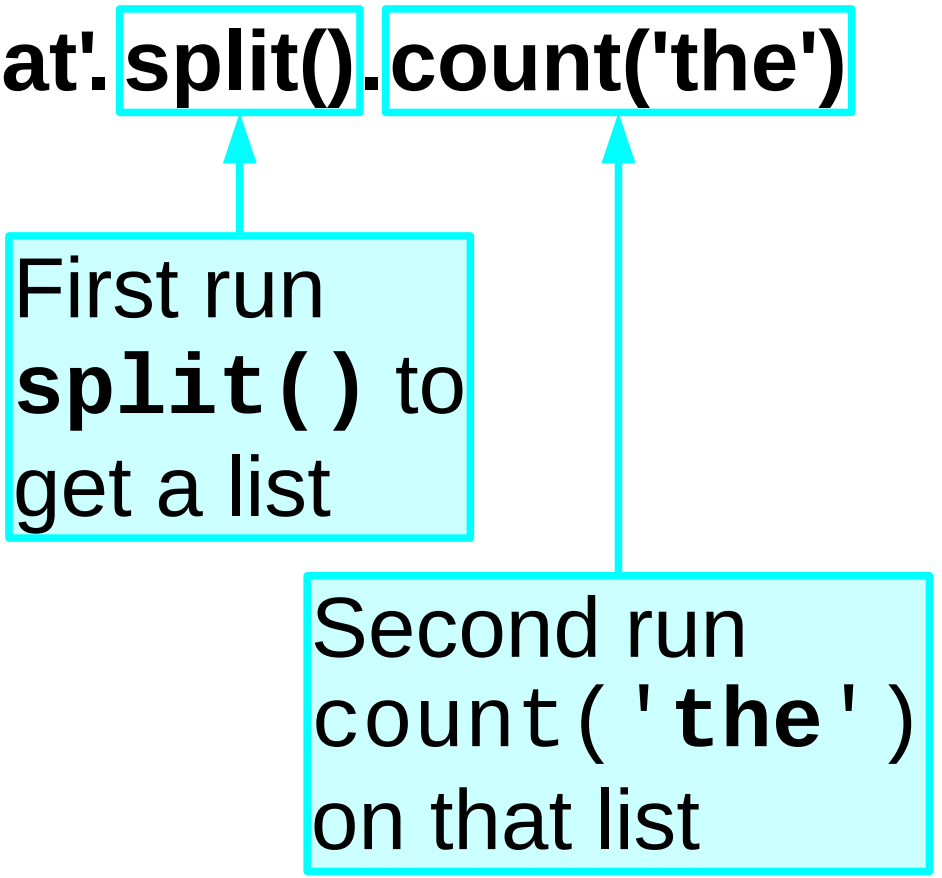
Comma separated values

Use the specialist Python support for these.

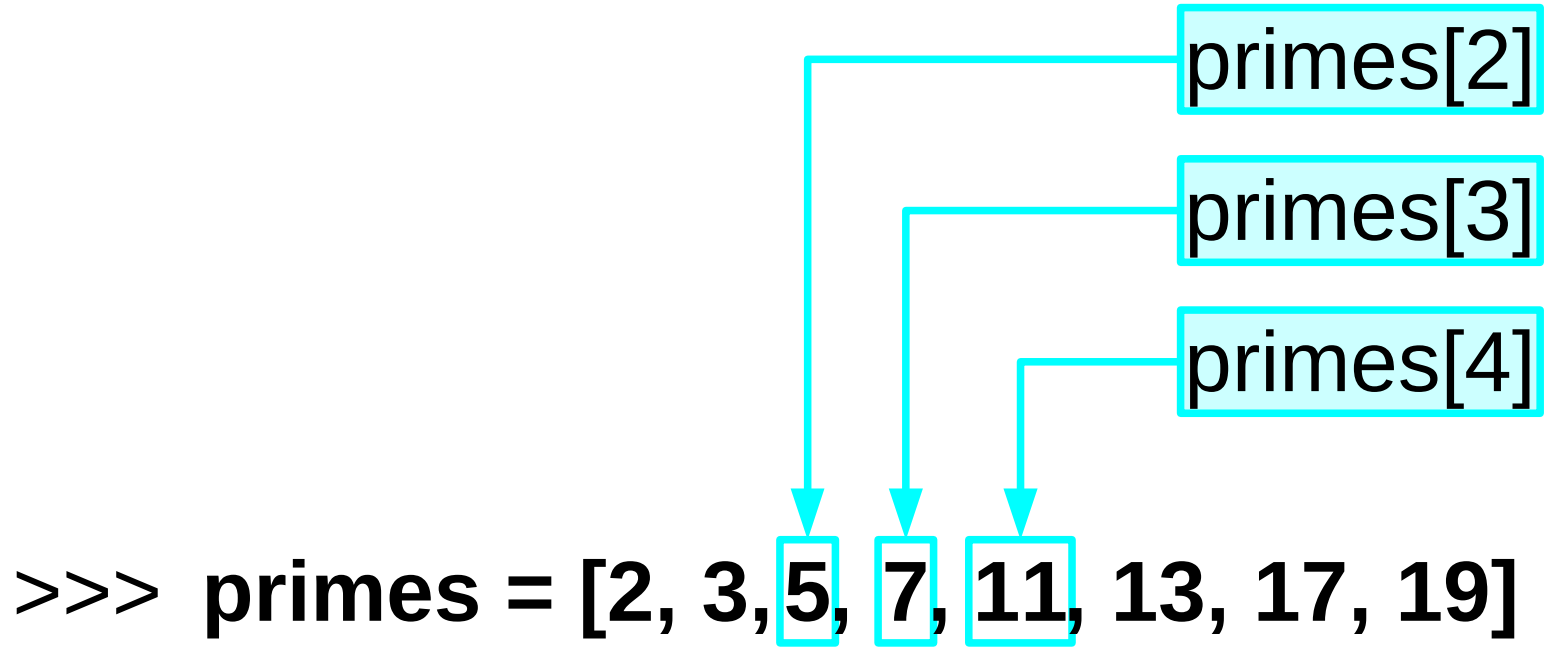

list
method in every list
method takes an argument
>>> ['the', 'cat', 'sat', 'on', 'the', 'mat'].count('the')
2
There are two 'the' strings in the list.

Combining methods

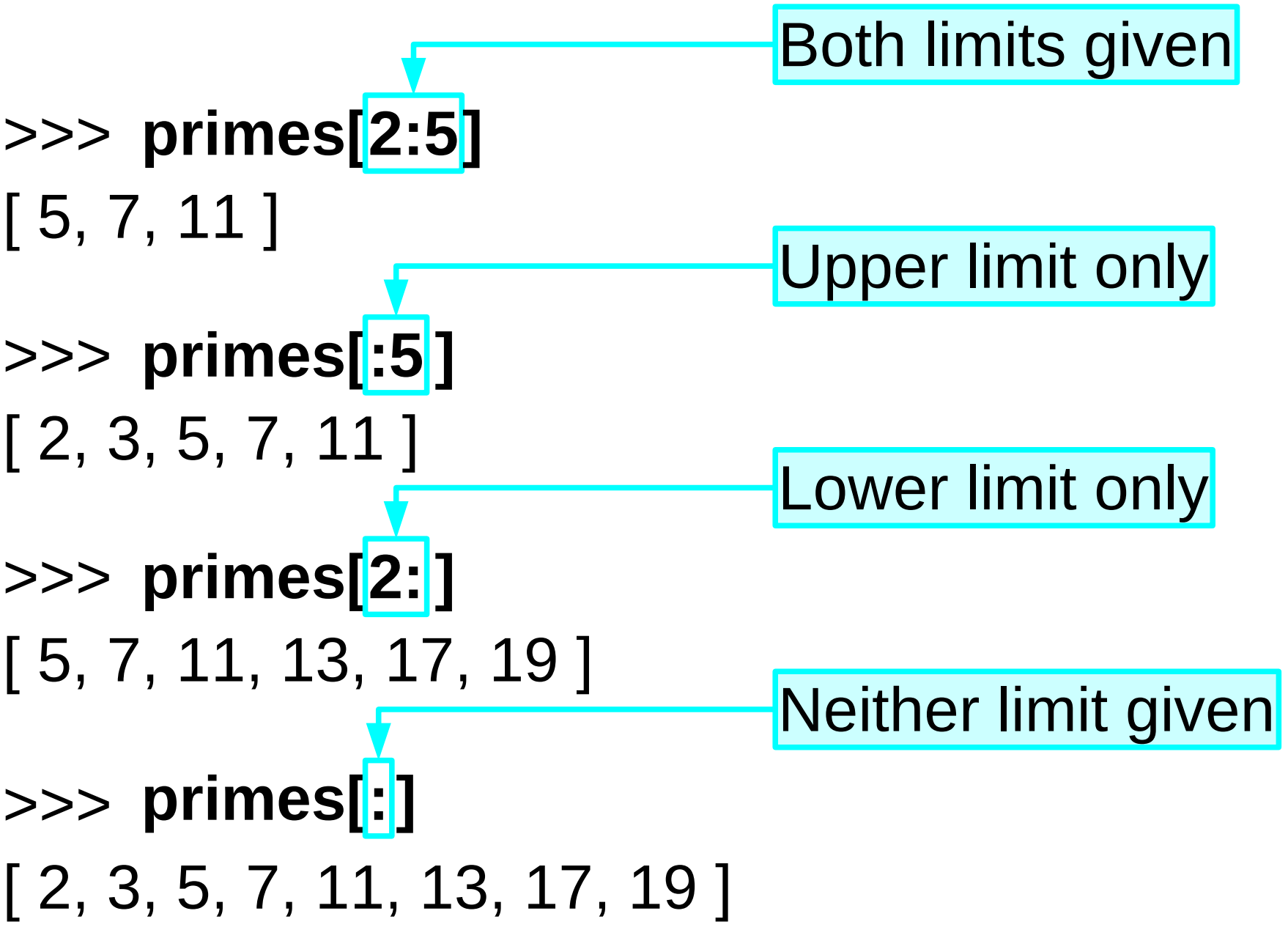
```
>>> 'the cat sat on the mat'.split().count('the')
2
```



Extracts from lists: “slices”



[5, 7, 11]


Both limits given
>>> primes[2:5]
[5, 7, 11]
Upper limit only
>>> primes[:5]
[2, 3, 5, 7, 11]
Lower limit only
>>> primes[2:]
[5, 7, 11, 13, 17, 19]
Neither limit given
>>> primes[:]
[2, 3, 5, 7, 11, 13, 17, 19]

```
#!/usr/bin/python

# This is a list of some metallic
#  elements.
metals = [ 'silver', 'gold', … ]

# Make a new list that is almost
#  identical to the metals list: the new
#  contains the same items, in the same
#  order, except that it does *NOT*
#  contain the item 'copper'.

    What goes here?

# Print the new list.
```

metals.py

```
#!/usr/bin/python

# This is a list of some data values.
data = [ 5.75, 8.25, … ]

# Make two new lists from this list.
#  The first new list should contain
#  the first half of data, in the same
#  order, whilst the second list should
#  contain the second half, so:
#    data = first_half + second_half
#  If there are an odd number of items,
#  make the first new list the larger
#  list.


# Print the new lists.
```

What goes here?

data.py

An answer

```
#!/usr/bin/python

# This is a list of some metallic
#  elements.
metals = [ 'silver', 'gold', … ]

# Make a new list that is almost
#  identical to the metals list: the new
#  contains the same items, in the same
#  order, except that it does *NOT*
#  contain the item 'copper'.
new_metals = []
for metal in metals:
    if metal != 'copper':
      new_metals.append(metal)

# Print the new list.
print new_metals
```

metals.py

Answer

```
#!/usr/bin/python
# This is a list of some data values.
data = [ 5.75, 8.25, … ]
# Make two new lists from this list.
#  The first new list should contain
#  the first half of data, in the same
#  order, whilst the second list should
#  contain the second half, so:
#    data = first_half + second_half
#  If there are an odd number of items,
#  make the first new list the larger
#  list.
if len(data) % 2 == 0:
      index = len(data) / 2
else:
      index = (len(data) + 1) / 2

first_half = data[:index]
second_half = data[index:]

# Print the new lists.
print first_half
print second_half
```

data.py

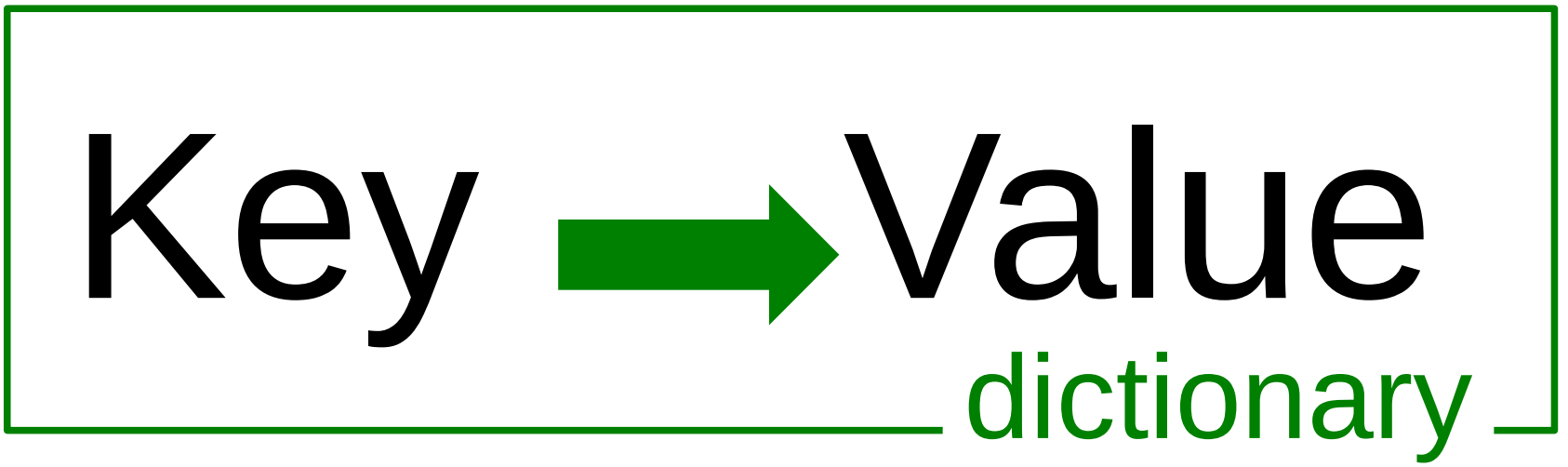
Lists : More Methods

insert()
count()
remove()
reverse()
sort()
del a[-1]

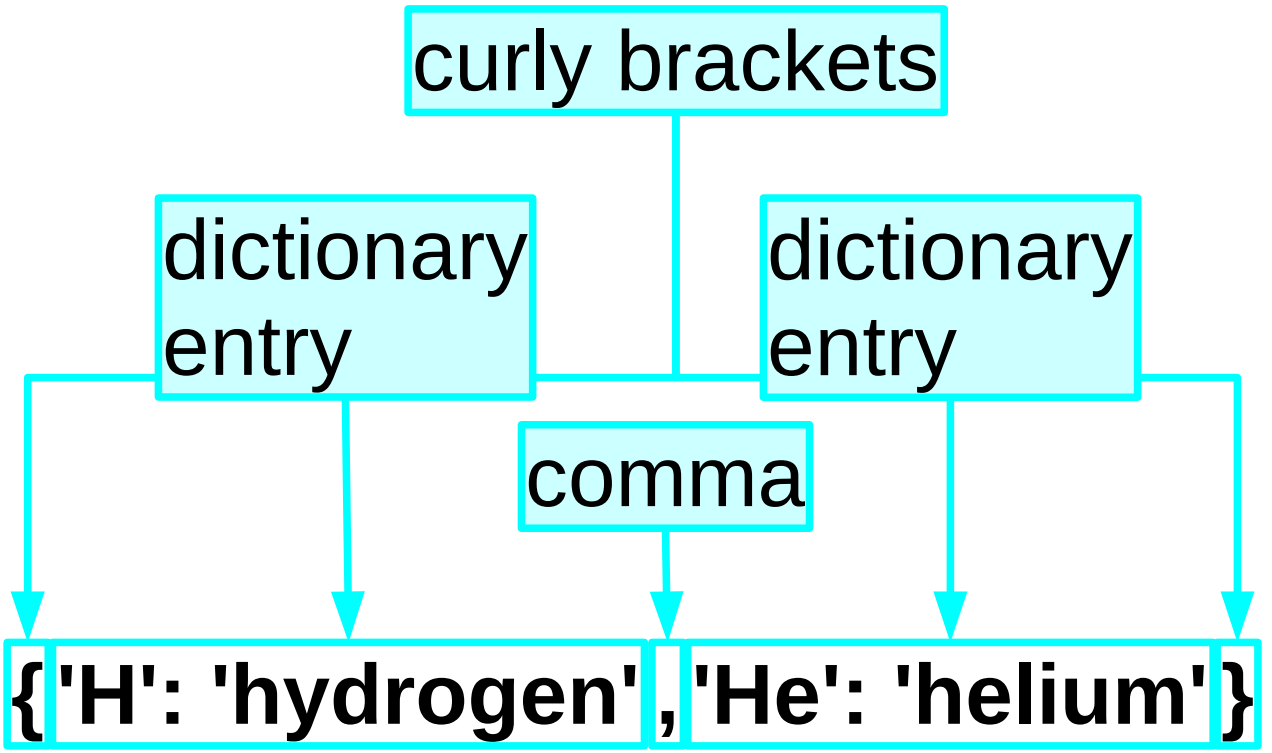
pop()
append()

Using it as stack and queue

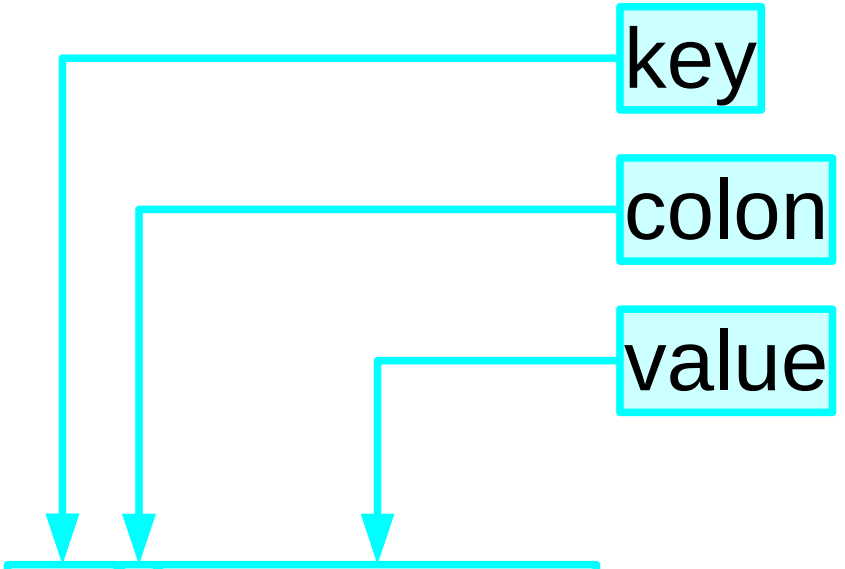
Dictionaries



Creating a dictionary — 1

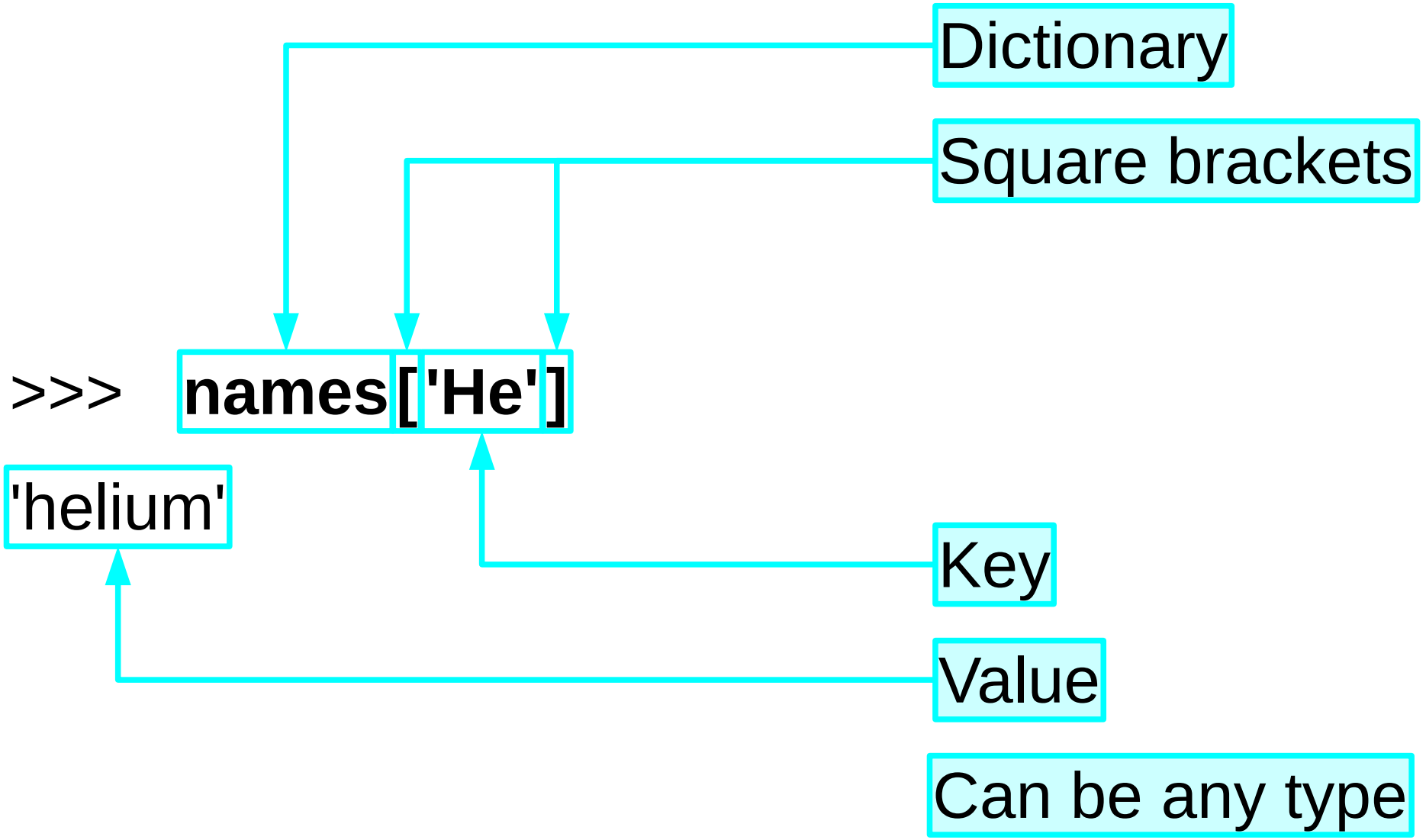


Creating a dictionary — 2




```
>>> names = {'H':'hydrogen', 'He': 'helium'}
```

Accessing a dictionary

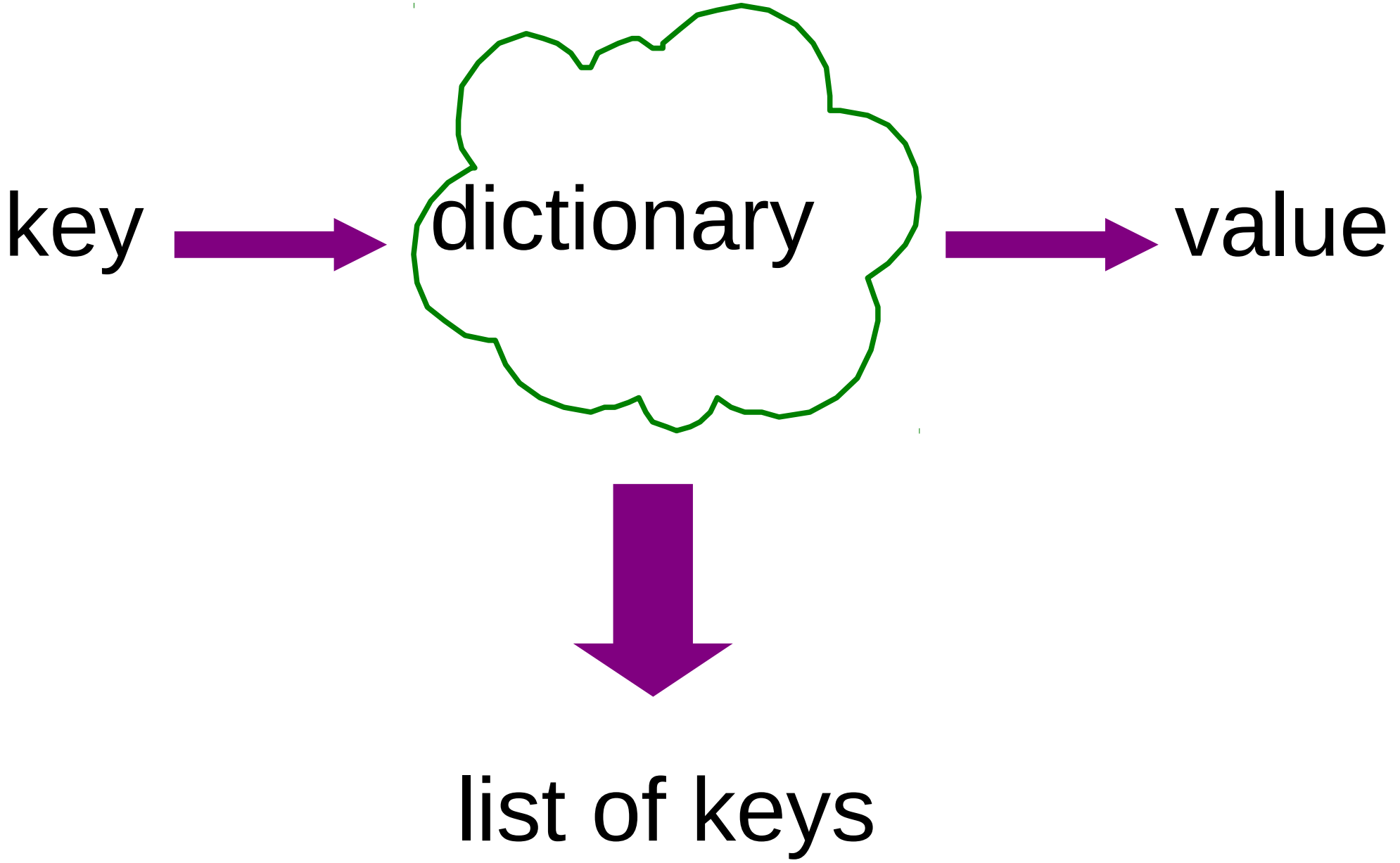


Creating a dictionary — 3

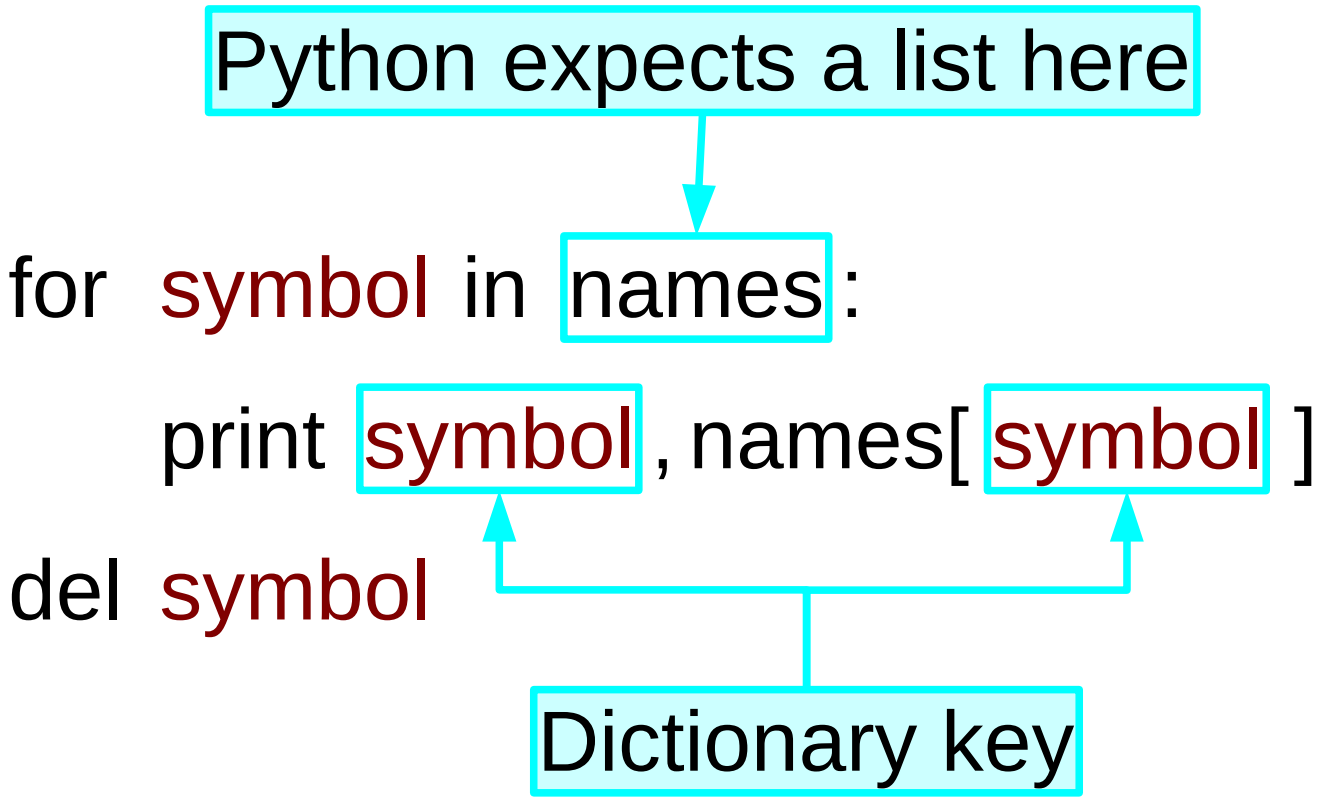
```
>>> names = {}
>>> names[ 'H' ]  = 'hydrogen'
>>> names[ 'He' ] = 'helium'
>>> names[ 'Li' ] = 'lithium'
>>> names[ 'Be' ] = 'beryllium'
```

Start with an empty dictionary

Add entries


key
dictionary
value
list of keys

Treat a dictionary like a list…



…and it behaves like a list of keys

Example

```
#!/usr/bin/python

names = {
  'H': 'hydrogen',
  'He': 'helium',
  …
  'U':  'uranium',
  }

for symbol in names:
  print names[symbol]
del symbol
```

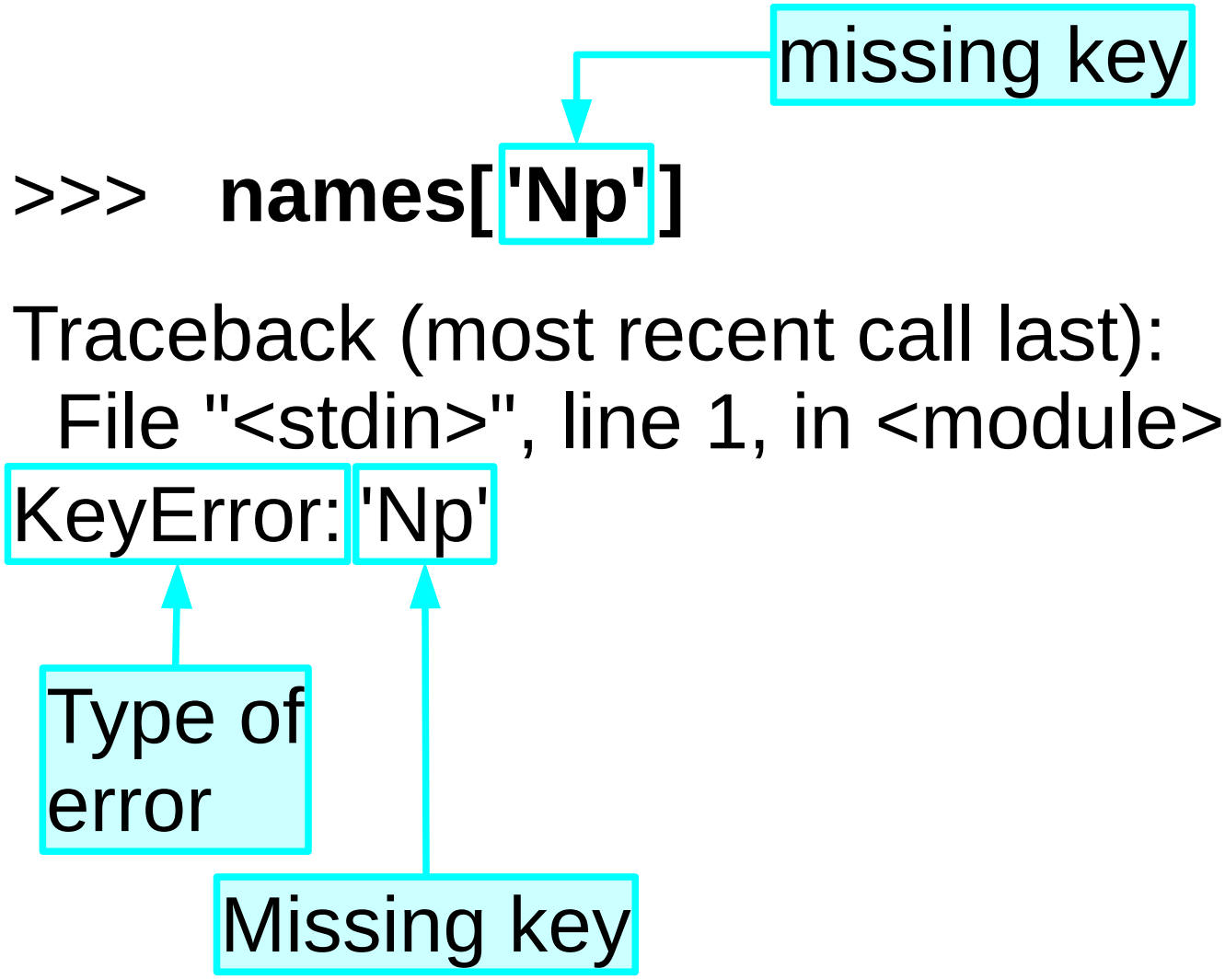
chemicals.py

$ python chemicals.py

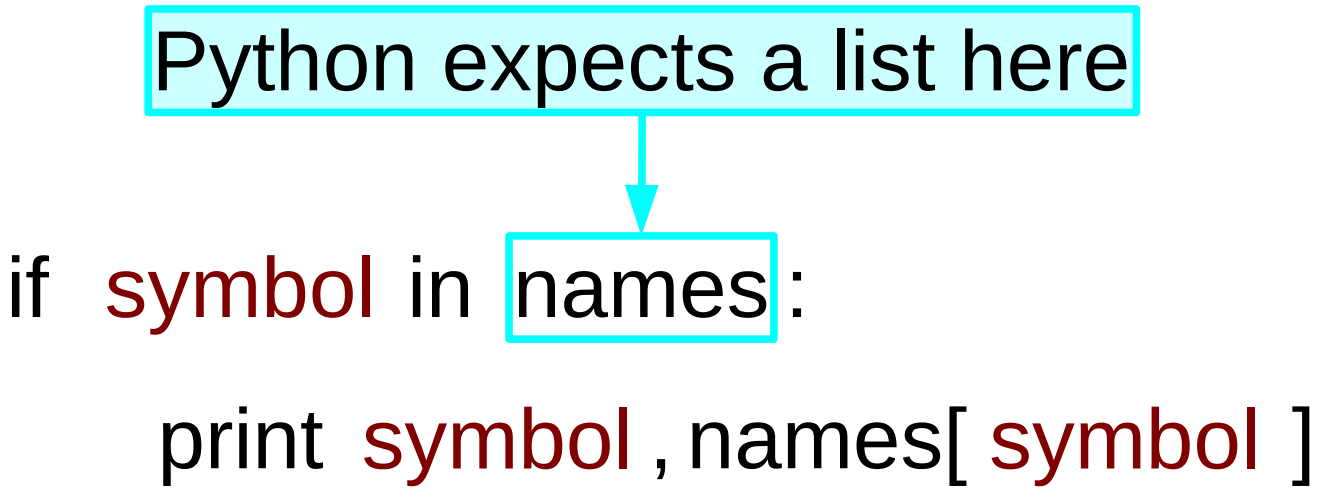
```
ruthenium
rhenium
…
astatine
indium
```

No relation between order in file and output!

Missing keys



Treat a dictionary like a list…



…and it behaves like a list of keys

Missing keys

```
>>>   names['Np']

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Np'
```

```
>>> 'Np' in names
False
```

Test for membership of a *list*

'Np' is not a key in the dictionary

Tuples

Singles
Doubles
Triples
Quadruples
Quintets

(42 , 1.95 , 'Bob')

(-1 , +1)

('Intro. to Python', 25, 'TTR1')

“not the same as lists”

Tuples are not the same as lists

(minimum, maximum)

(age, name, height)

(age, height, name)

(age, height, name, weight)

Independent, grouped items

Related, sequential items

[2, 3, 5, 7]

[2, 3, 5, 7, 11]

[2, 3, 5, 7, 11, 13]

Access to components

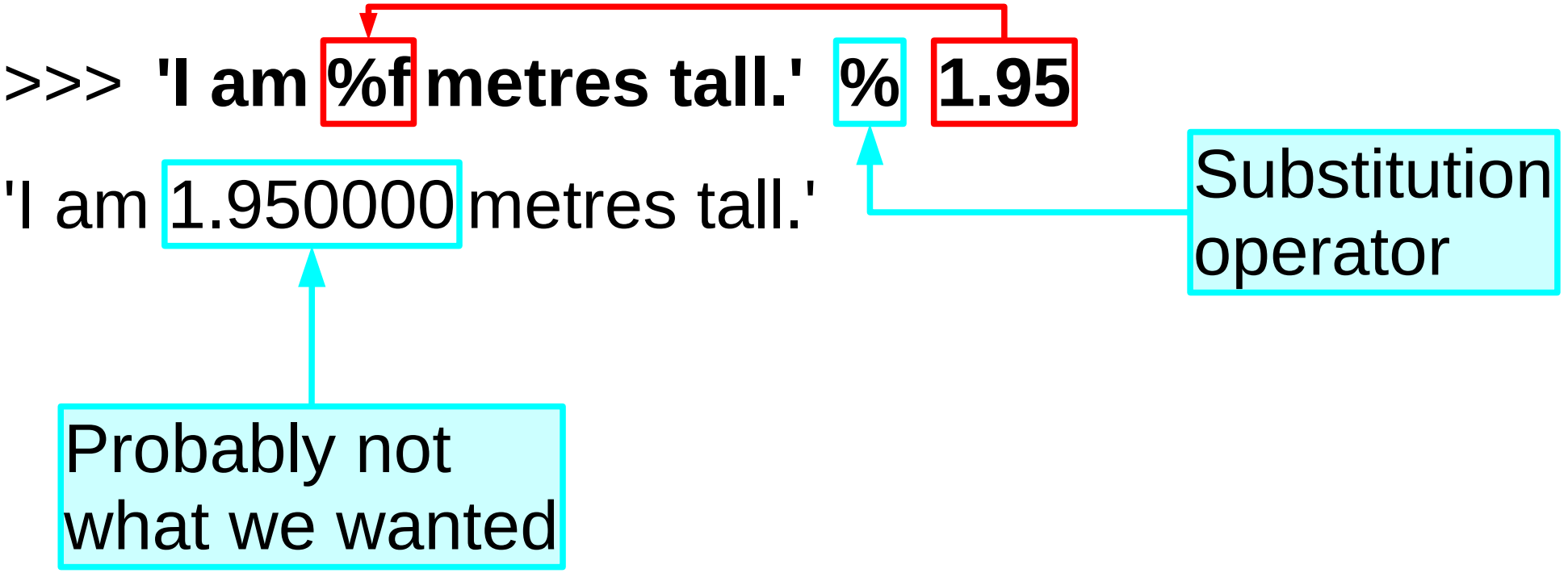
Same access syntax as for lists:

```
>>> ('Bob', 42, 1.95)[0]
'Bob'
```

But tuples are *immutable*:

```
>>> ('Bob', 42, 1.95)[1]  =  43
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support
item assignment
```

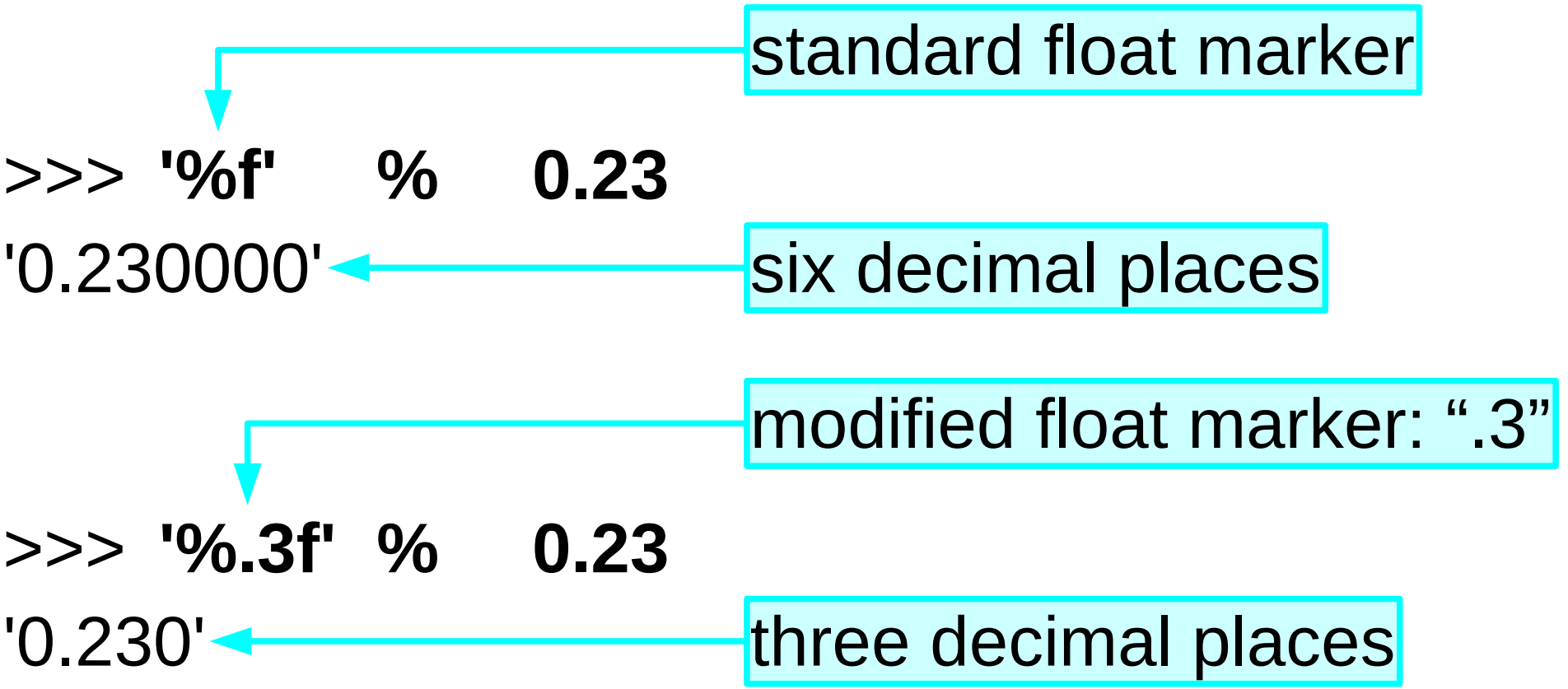
String substitution



Substituting multiple values

```
>>> 'I am %f metres tall and my name is %s.'
    % (1.95, 'Bob')
'I am 1.950000 metres tall and my name is Bob.'
```

Formatted substitution



More complex formatting possible

```
'23'    '23.4567'    '   23.46'
'   23' '23.456700' '23.46   '
'0023' '23.46'       '  +23.46'
'  +23' '+23.4567' '+23.46 '
'+023' '+23.456700'
'23  ' '+23.46'     'Bob'
'+23 ' '0023.46'    'Bob  '
       '+023.46'    '   Bob'
```

Uses of tuples

1. Functions
2. Related data
3. String substitution

And now for something completely…

Defining functions

```
def functionname(params):
    statement1
    statement2
```

Local and global variables

Default argument values

```
def functionname(param1, param2 = value):
    statement1
    statement2
```

Keyword arguments

```
def functionname(param1, param2 = value):
    statement1
    statement2
```

return *statement*

```
def functionname(param1, param2 = value):
    statement1
    statement2
    return “values are passsed”
```

Classes

class *classname:*
    body

self

__init__

- runs as soon as an object of a class is instantiated
- useful to do any initialization

Inheritance

- Reusing the code

```
class A:

class B(A):

class C(A)
```

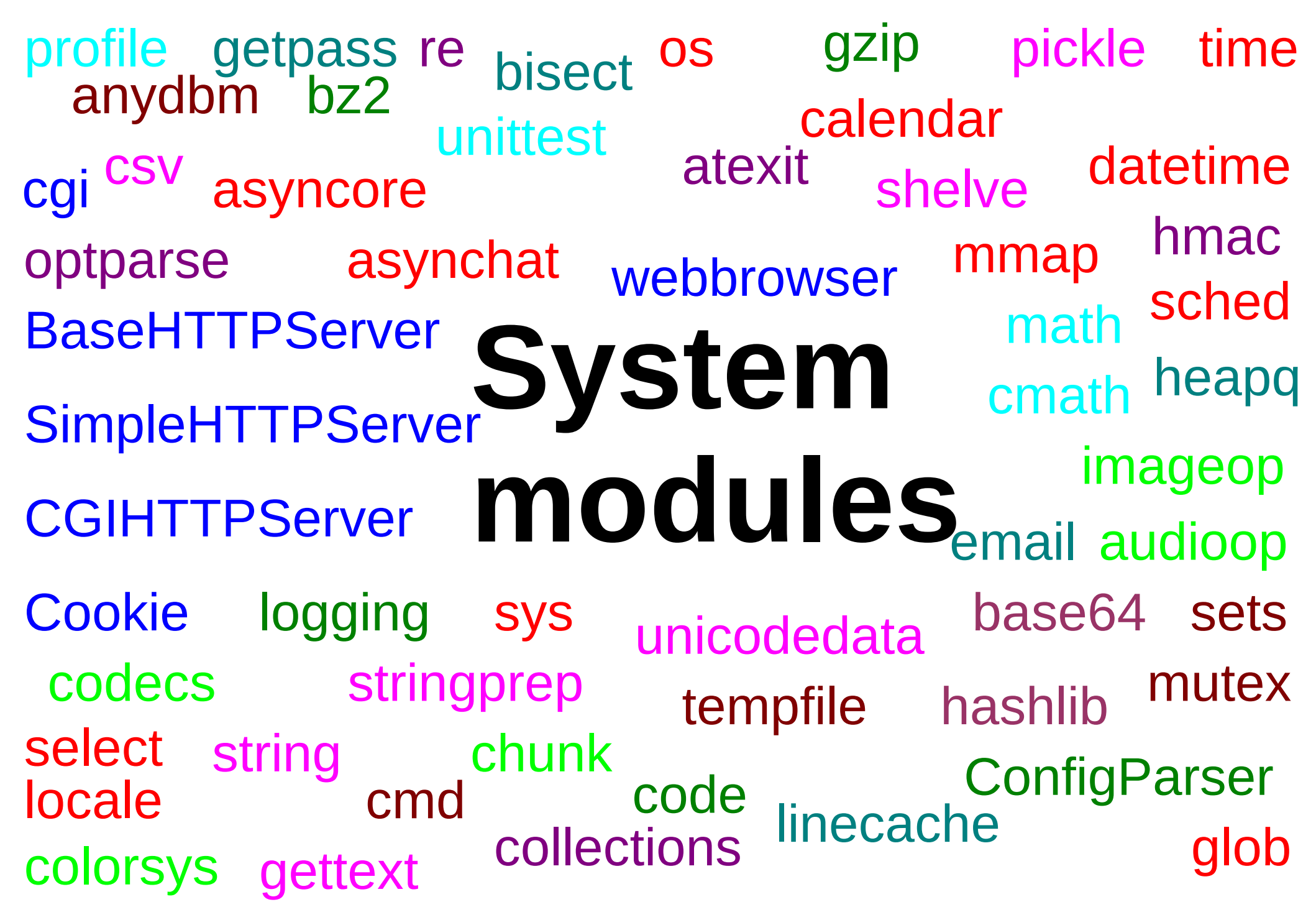
Inheritance

- Reusing the code

```
class A:

class B(A):

class C(A):
```

Modules

Import *modulename*
from *modulename* import *

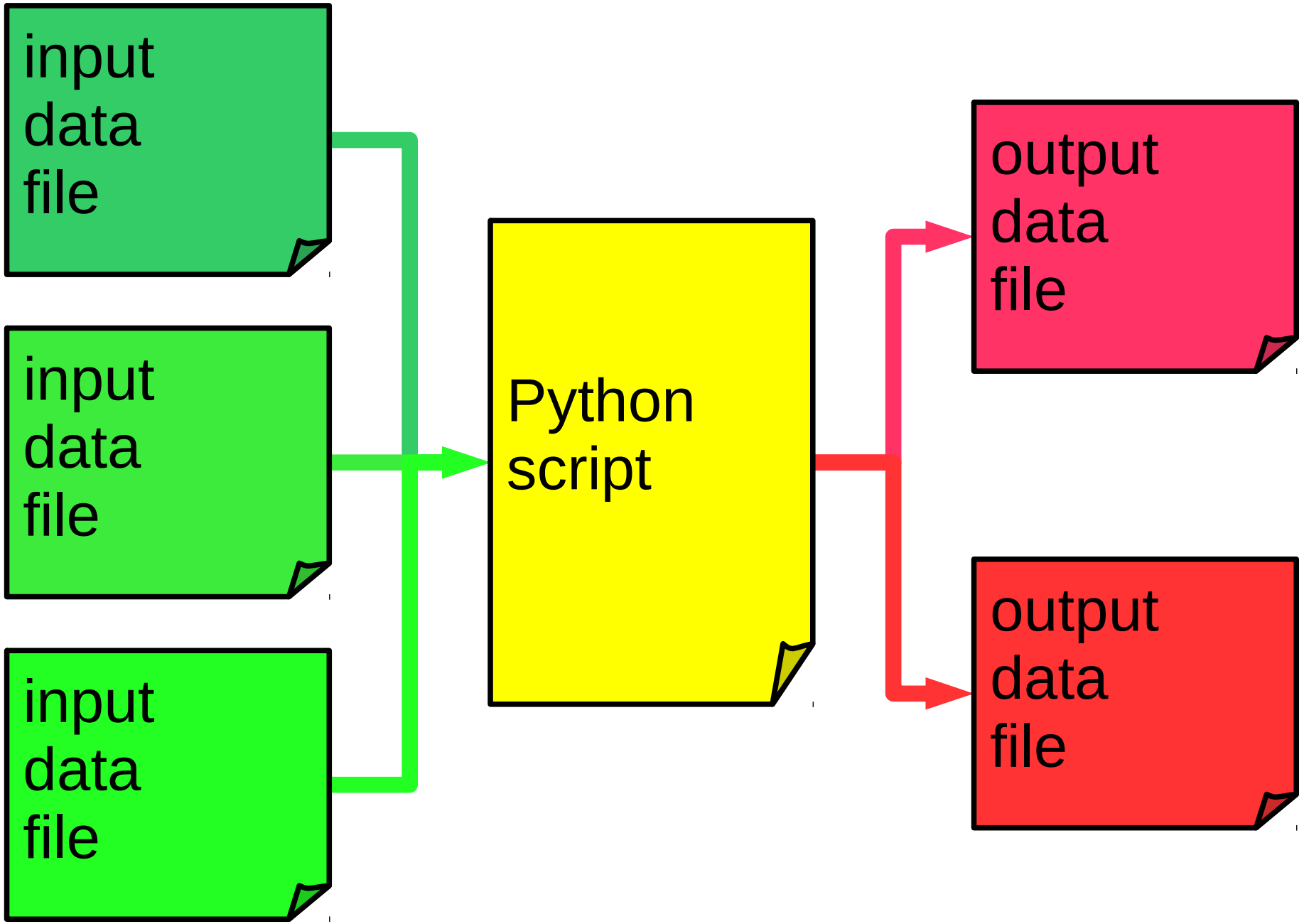

profile
getpass
re
bisect
os
gzip
pickle
time
anydbm
bz2
calendar
unittest
csv
atexit
datetime
cgi
asyncore
shelve
hmac
optparse
asynchat
webbrowser
mmap
sched
BaseHTTPServer
math
System
modules
heapq
cmath
SimpleHTTPServer
imageop
CGIHTTPServer
email
audioop
Cookie
logging
sys
unicodedata
base64
sets
codecs
stringprep
mutex
tempfile
hashlib
select
string
chunk
ConfigParser
locale
cmd
code
linecache
collections
glob
colorsys
gettext

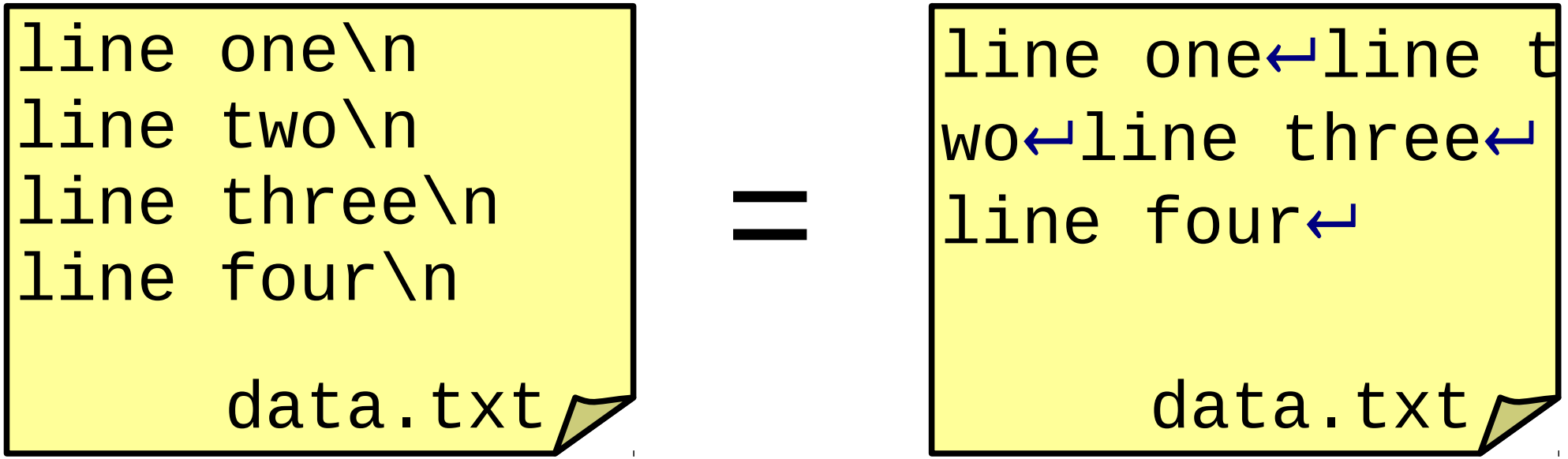
Let’s break for an exercise…

Accessing the system

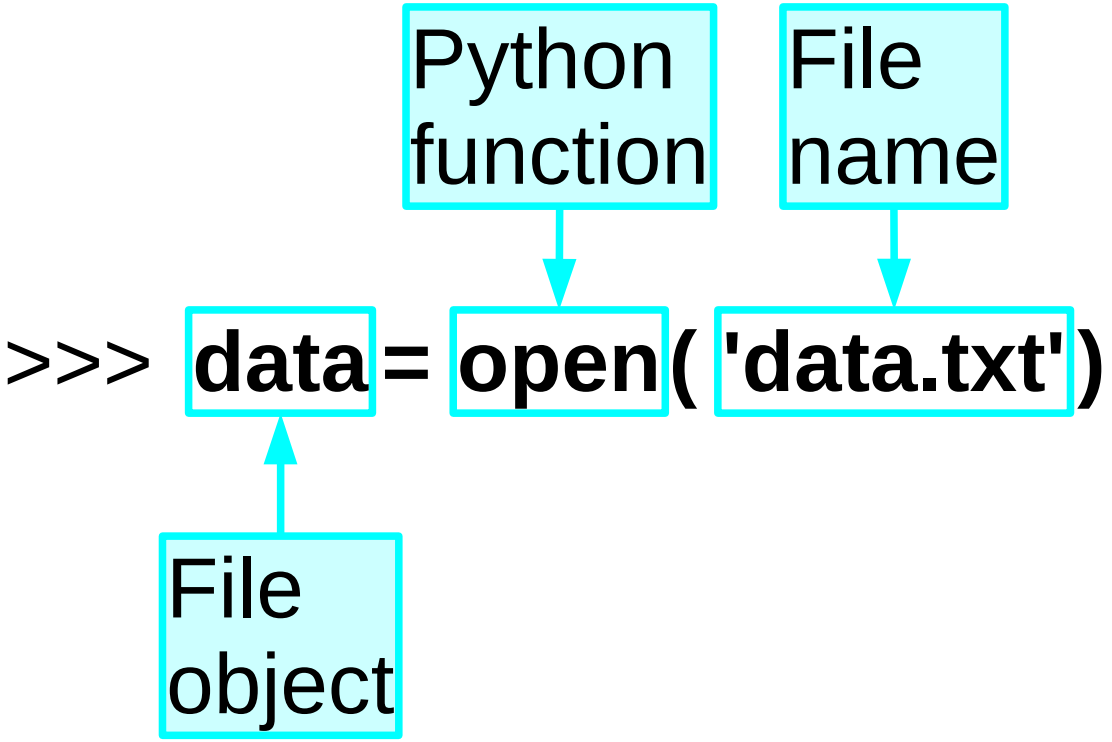
1. Files
2. Standard input & output
3. The command line

May want to access many files



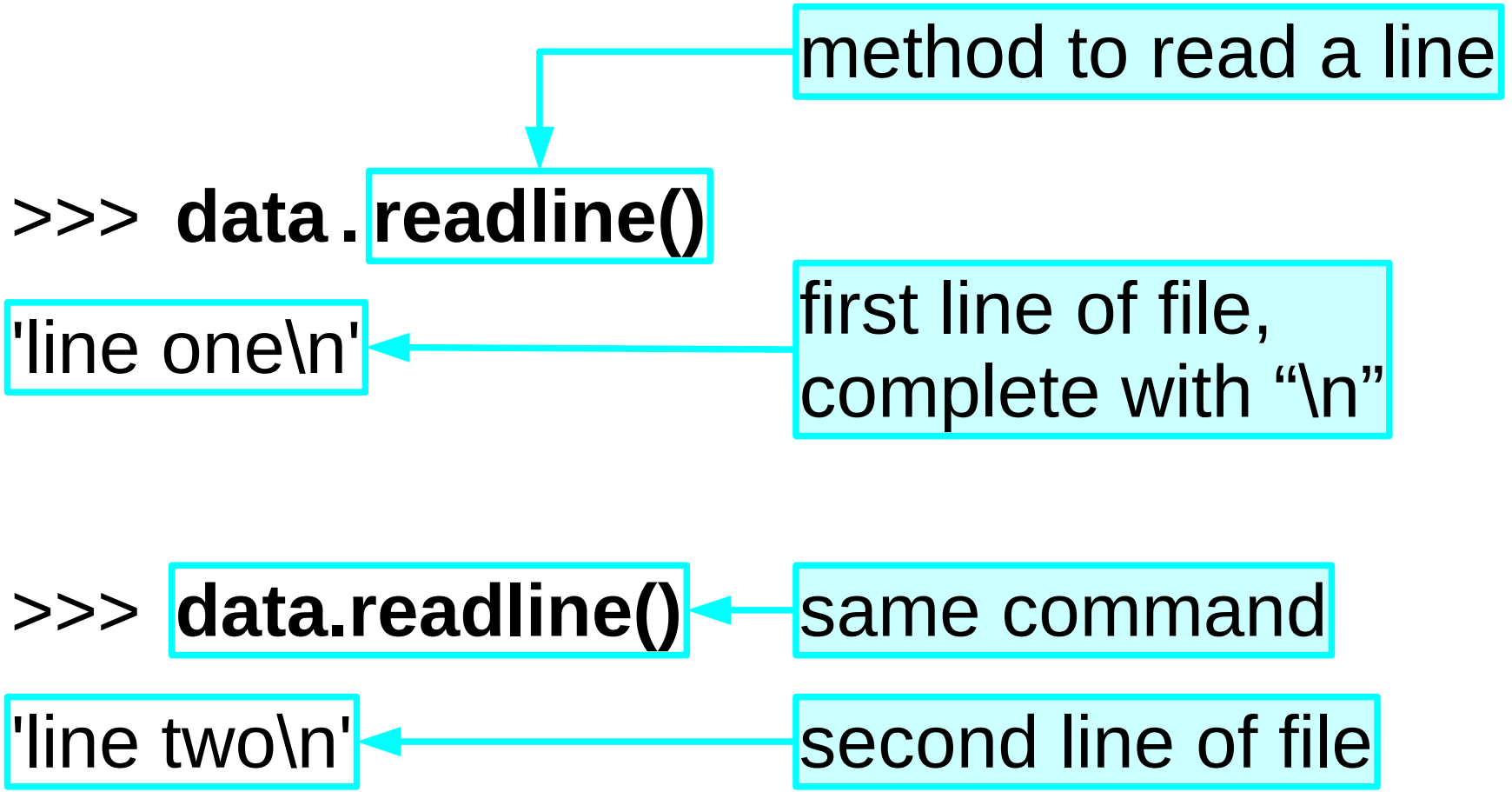

line one\n
line two\n
line three\n
line four\n
data.txt
=
line one↵line t
wo↵line three↵
line four↵
data.txt

\n ↵



All access to the file is via the file object

```
>>> data = open( 'data.txt' )
```



```
>>> data = open( 'data.txt' )
>>> data.readline()
'line one\n'
>>> data.readline()
'line two\n'

>>> data.readlines()
[ 'line three\n', 'line four\n' ]
```

remaining lines

```
>>> data = open( 'data.txt' )
>>> data.readline()
'line one\n'
>>> data.readline()
'line two\n'
>>> data.readlines()
[ 'line three\n', 'line four\n' ]


>>> data.close()
>>> del data
```

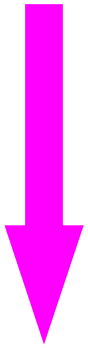
disconnect

delete the variable

Treating file objects like lists:

```
for line in data.readlines():
    do stuff
```

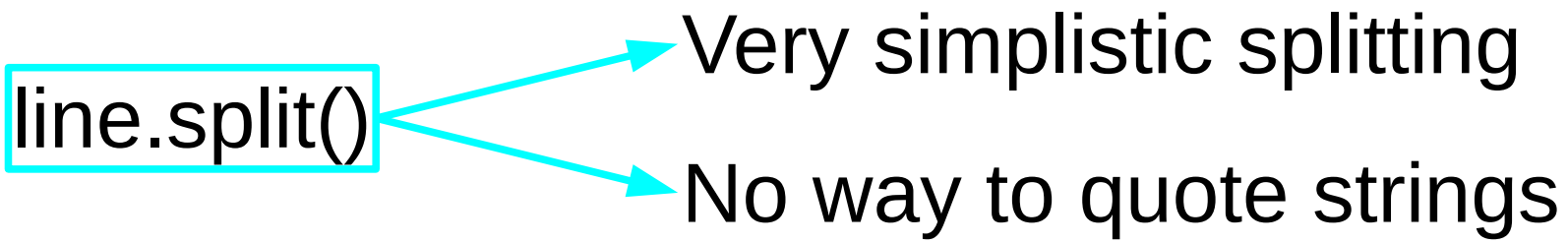
reads the lines all at once



```
for line in data:
    do stuff
```

reads the lines as needed

Very primitive input

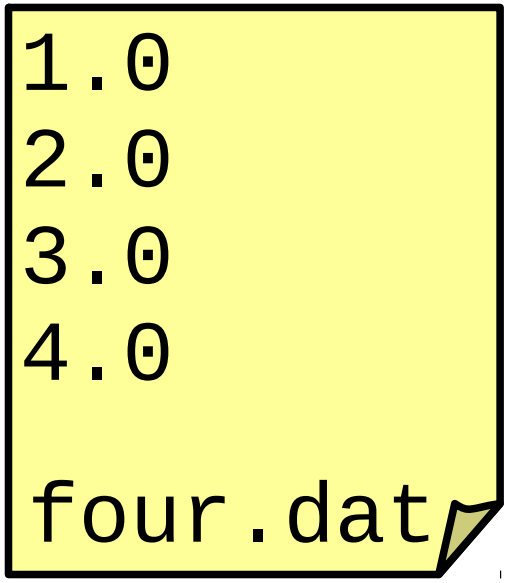


Comma separated values: **csv** module
Regular expressions: **re** module

“Python: Further Topics” course

“Python: Regular Expressions” course

Reading data gets you strings



readlines() → ['1.0\n', '2.0\n', …]

strings

not floats

```
>>> '1.0\n'.strip()
'1.0'
```

Method to clear trailing white space

Still need to convert string to other types

```
#!/usr/bin/python

# This script reads in some
#  numbers from the file 'numbers.txt'.
# It then prints out the smallest
#  number, the arithmetic mean of
#  the numbers, and the largest
#  number.
```

What goes here? (Use the function you wrote in an earlier exercise.)

Output to files

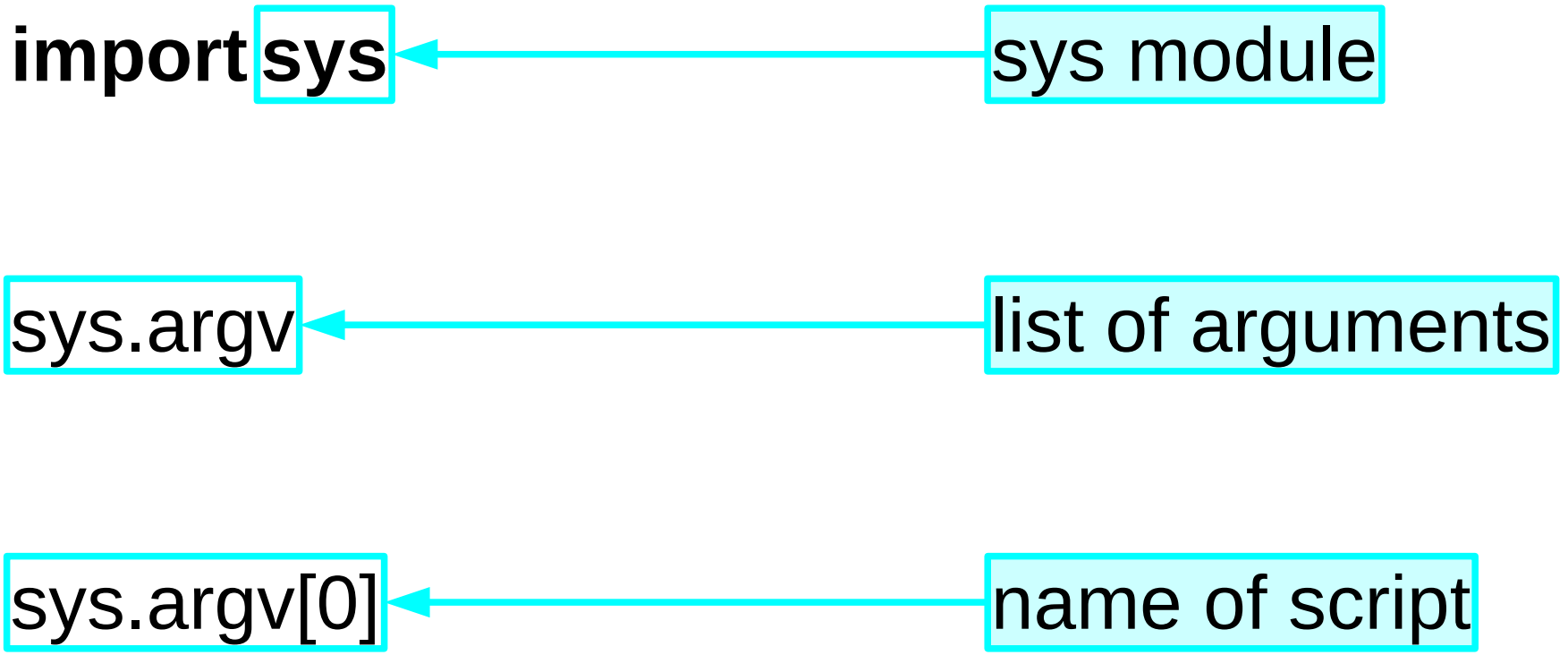
```
>>> output = open('output.dat', 'w')
>>> output.write('alpha\n')
>>> output.write('bet')
>>> output.write('a\n')
>>> output.writelines([ 'gamma\n', 'delta\n'])
>>> output.close()
```

explicit “\n”

write(): writes *lumps* of data

Flushes to file system

Command line



```
#!/usr/bin/python

print sys.argv[0]

print sys.argv
```

args.py

$ **python args.py 0.25 10**

args.py

['args.py', '0.25', '10']

NB: list of *strings*

Questions


Thank You!