Python: Further Topics

Day One

Bruce Beckles

University of Cambridge Computing Service

UNIVERSITY OF CAMBRIDGE

UCS

University of Cambridge Computing Service

`None` ∅

`None` is a special value in Python, with its own data type (**`NoneType`**). It is Python's way of representing "nothing". Its "truth value" is **`False`** (i.e. for the purpose of tests it is equivalent to **`False`**).

It is often used as "placeholder" value, or to mean that there is "no data".

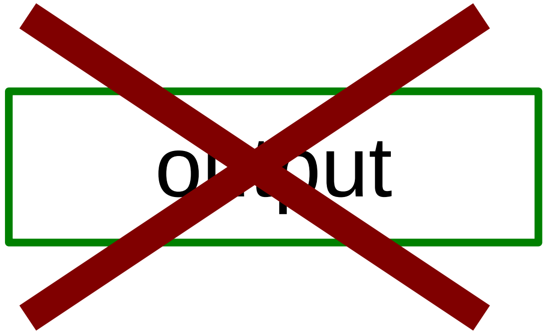
```
>>> None
>>>
```

```
>>> not None
True
```

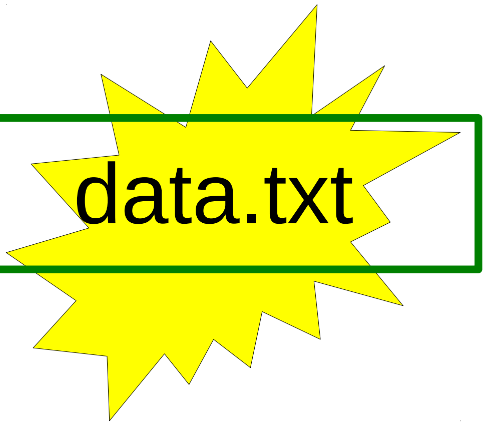
```
>>> type(None)
<type 'NoneType'>
```

What if something goes wrong?

```
>>> data = open('output')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: 'output'
```



```
>>> data = open('data.txt')
>>> data.readlines()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 13] Permission denied
```



“Traceback”: the command's history

“stdin”: “standard input” = the terminal

```
>>>  data = open('output')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: 'output'
```

Only one line of command

Error number

Type of ***exception*** (error)

Error message

Exception handling

try:
        Python commands
except:
        Exception handler

Python exception handling:

`try` some commands

if there's an error…

…execute the ***`except`*** block…

…but if there's no error, don't execute the ***`except`*** block.

(Similar to **`if`**…**`else`** statements)

```
def file2dict(filename):
    import sys
    dict={}
    try:
        data = open(filename)
        for line in data:
            [ key, value ] = line.split()
            dict[key] = value
        data.close()
    except IOError:
        print "Problem with file %s" % filename
        print "Aborting!"
        data.close()
        sys.exit(1)
    return dict
```

utils.py

```
>>> import utils
>>> mydict = utils.file2dict('output')
Problem with file output
Aborting!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "utils.py", line 110, in file2dict
    data.close()
UnboundLocalError: local variable 'data' referenced before assignment
>>>
```

```
def file2dict(filename):
    import sys
    dict={}
    data = None
    try:
        data = open(filename)
        for line in data:
            [ key, value ] = line.split()
            dict[key] = value
        data.close()
    except IOError:
        print "Problem with file %s" % filename
        print "Aborting!"
        if type(data) == file:
        🟩  data.close()
        sys.exit(1)
    return dict
```

utils.py

```
>>> import utils
>>> mydict = utils.file2dict('output')
Problem with file output
Aborting!
$
```

```
def file2dict(filename):
    import sys
    dict={}
    data = None
    try:
        data = open(filename)
        for line in data:
            [ key, value ] = line.split()
            dict[key] = value
        data.close()
    except IOError, error:
        (errno, errdetails) = error
        print "Problem with file %s: %s" % (filename, errdetails)
        print "Aborting!"
        if type(data) == file:
            data.close()
        sys.exit(1)
    return dict
```

utils.py

```
>>> import utils
>>> mydict = utils.file2dict('output')
Problem with file output: No such file or directory
Aborting!
$
```

```
>>> line = "Too many values"
>>> [ key, value ] = line.split()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack
>>> line = "notenough!"
>>> [ key, value ] = line.split()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: need more than 1 value to unpack
>>>
```

Handling multiple exceptions

```
try:
        Python commands
except Exception1:
        Exception handler1
except Exception2:
        Exception handler2
                ...
except:
        Handler for all other
        exceptions
```

try some commands

if there's an error…

…examine the ***except*** blocks…

…if the error is ***Exception1*** use that ***except*** block...

…if it's ***Exception2*** use that ***except*** block…

…and so on…

…if it's not any of the listed exceptions, use the final ***except:*** block if it exists.

Exception handling: exc_info()

import sys

exc_info() returns a *tuple* of three items of information about the current exception: (Exception*Type*, Exception*Details*, *Traceback*)

```
(err_type, err_value) = sys.exc_info()[:2]
```

It is **dangerous** to access the traceback so **don't**: use a slice of the *first two items* in the tuple

Variable for *type* of exception e.g. **`IOError`**

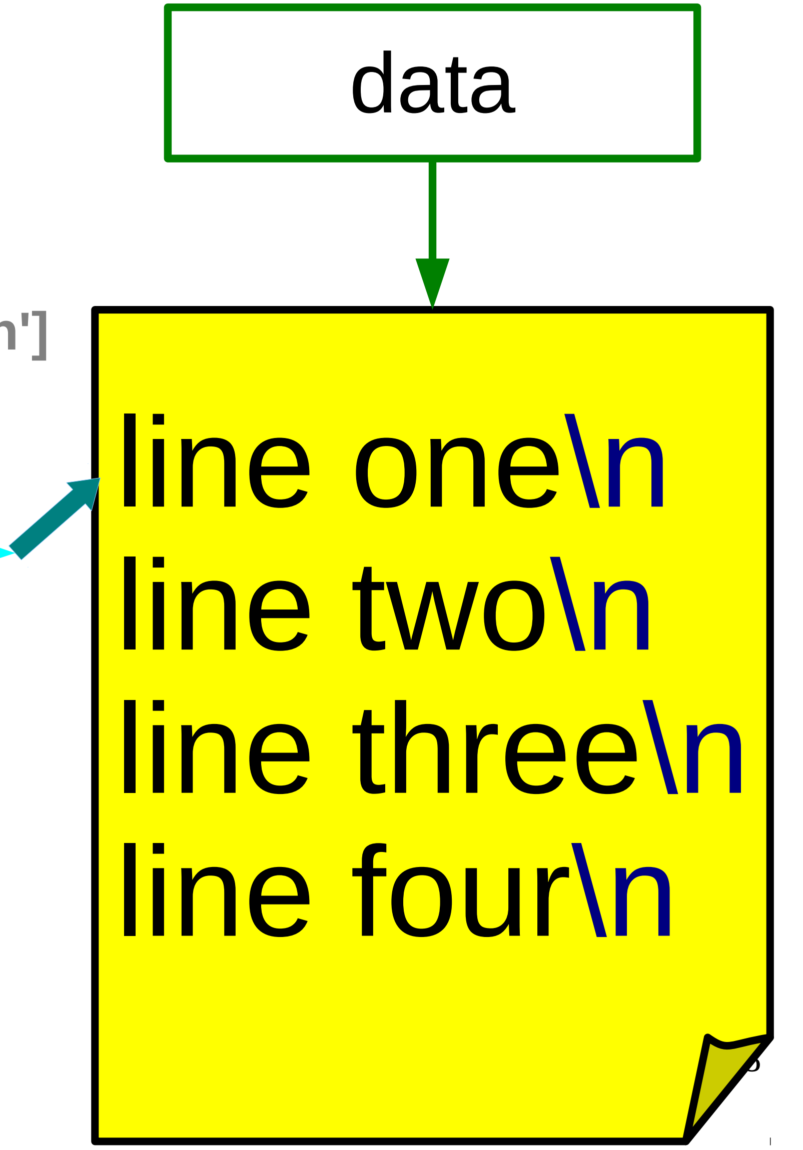
Variable for exception *details* e.g. (2, 'No such file or directory')

Moving to the start of a file

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

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

>>> data.seek( 0 )
```

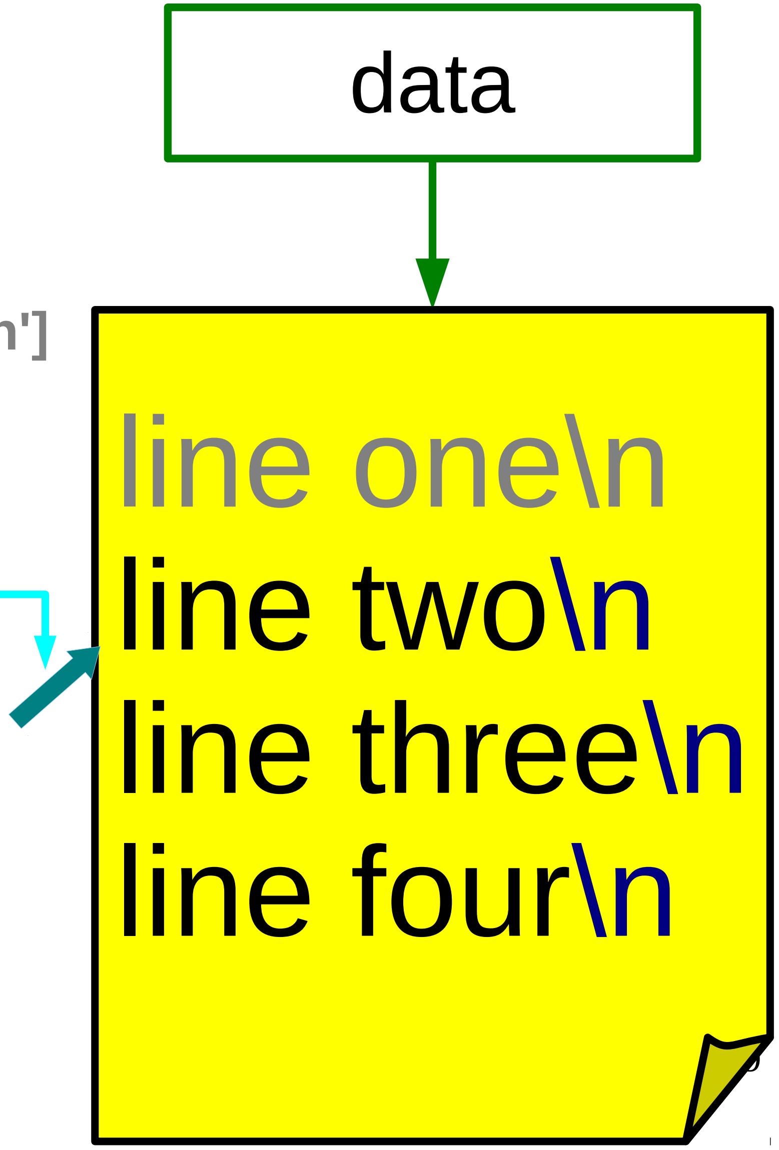
offset in file

position:
start of file



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

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

>>> data.seek(0)

>>> data.readline()
'line one\n'
```

position:
after end of first line,
at start of second line

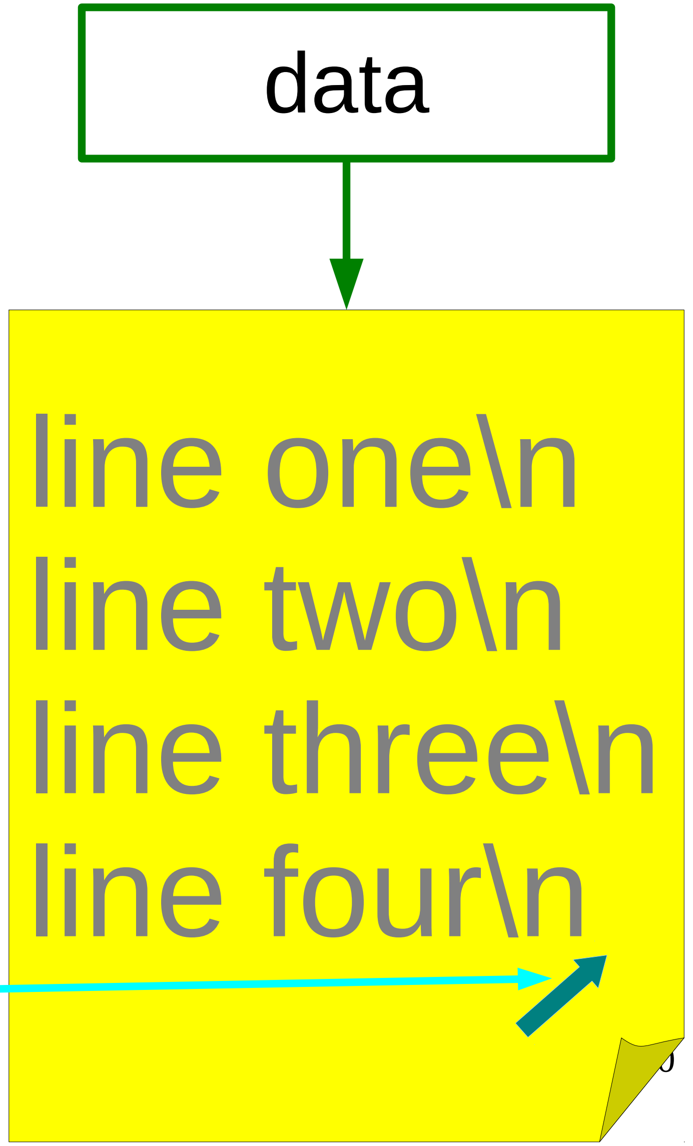


Moving to the end of a file

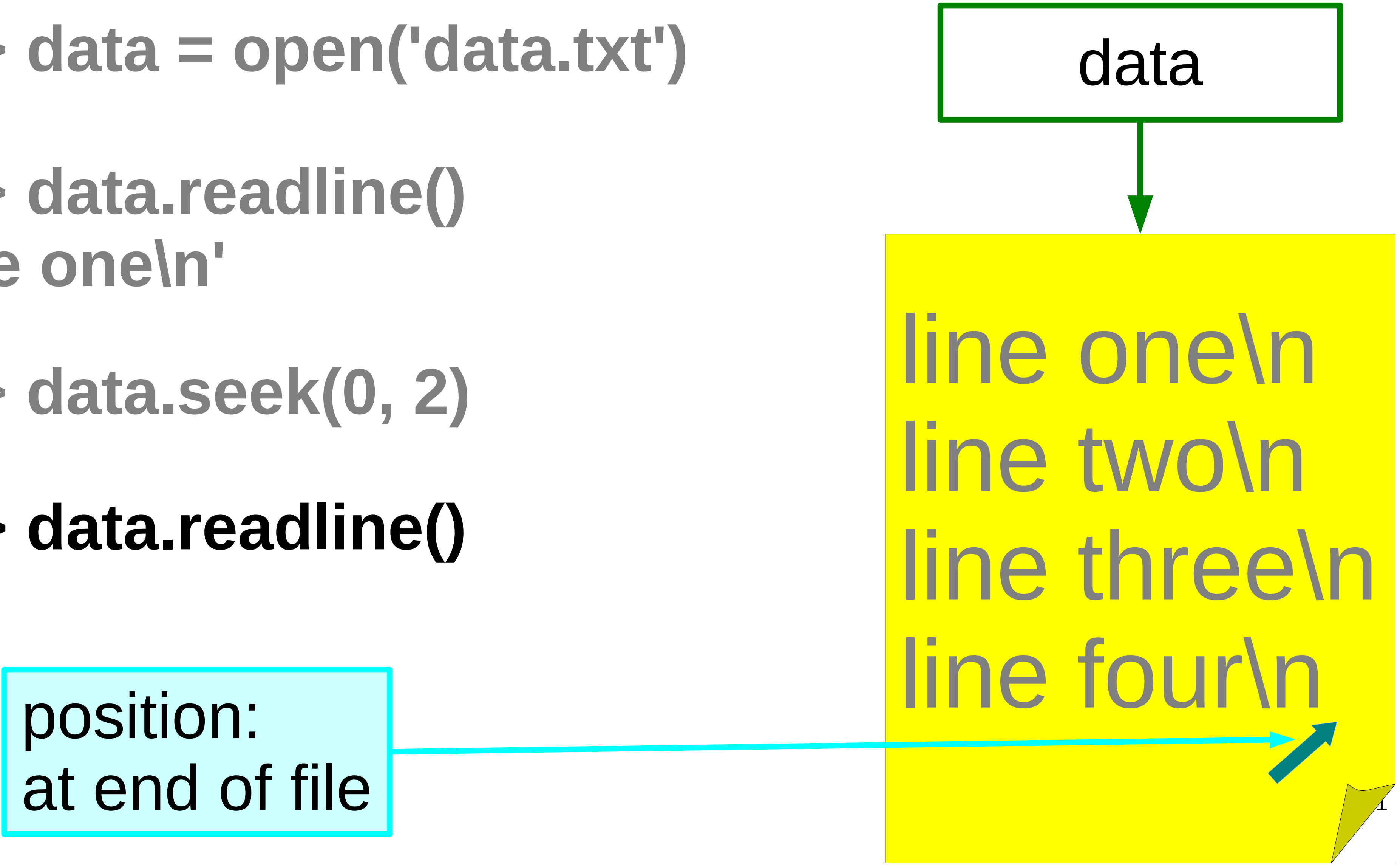
```
>>> data = open('data.txt')

>>> data.readline()
'line one\n'

>>> data.seek(0, 2)
```

specifies that offset is
relative to the end of the file

position:
at end of file



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

>>> data.readline()
'line one\n'

>>> data.seek(0, 2)

>>> data.readline()
''
```

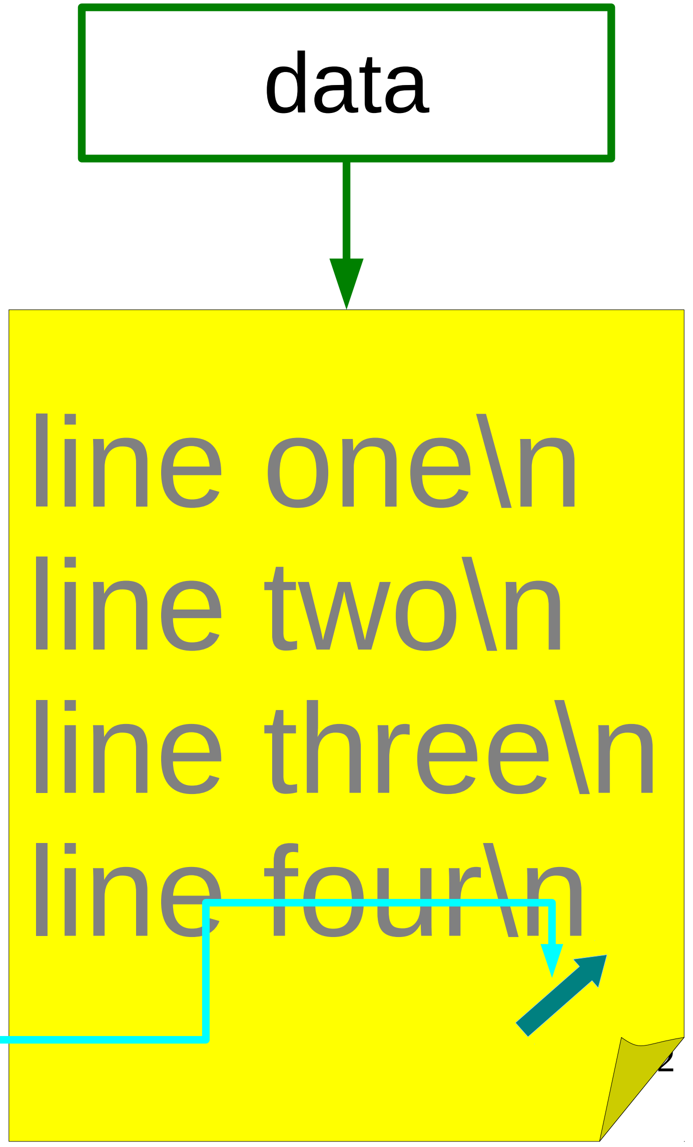


Finding your position in a file

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

>>> data.readline()
'line one\n'

>>> data.seek(0, 2)

>>> data.tell()
39L

>>> data.close()
```

current offset as a long integer

position: at end of file



Renaming a file

```
>>> import os.path
>>> os.path.exists('data1.txt')
True
```

`rename()` renames files.
It lives in the **os** module.

```
>>> import os
>>> os.rename('data1.txt', 'data2.txt')
```

Under Unix/Linux if the new name is a file that already exists, then that file is **deleted**, i.e. **`rename()`** behaves like the Unix **`mv`** command.

```
>>> import os.path
>>> os.path.exists('data1.txt')
False
```

Accessing binary files

input = open('input.dat', **'rb'**)

open for **r**eading
a **b**inary file

input.**read(1)**

read some *bytes* from a file:
bytes are returned as a ***`string`***

maximum number of bytes
to read from file: omit to read
all remaining bytes of file

output = open('output.dat', **'wb'**)

open for **w**riting
in **b**inary mode

Working with modules and functions

```
>>> import utils
>>> reload(utils)
<module 'utils' from 'utils.pyc'>
```

`reload()` reloads an *already loaded* module from the file containing the module.

```
>>> dir(utils)
['__builtins__', '__doc__', '__file__', '__name__', 'dict2file', 'file2dict',
'find_root', 'greet', 'print_and_return', 'print_dict', 'reverse']
```

`dir()` displays all the *names* defined within a module (or indeed in any type of object).

```
>>> callable(utils.file2dict)
True
>>> callable(utils.__doc__)
False
```

`callable()` tells us whether or not we can call something.

Augmented assignment

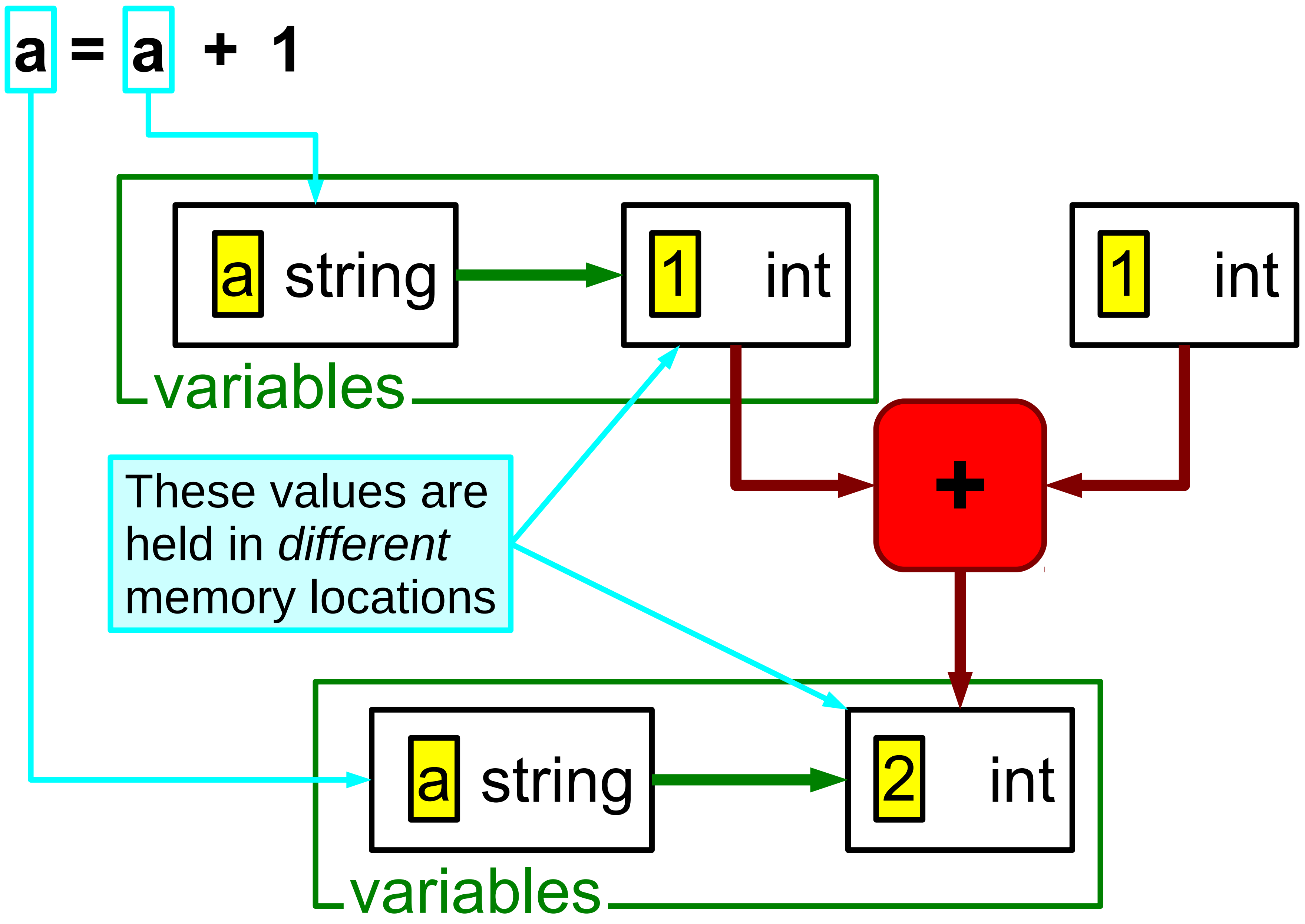
```
>>> a = 1
>>> a += 1
>>> a
2

>>> a -= 1
>>> a
1

>>> a *= 4
>>> a
4
```

```
>>> a = 1
>>> a = a + 1
>>> a
2

>>> a = a - 1
>>> a
1

>>> a = a * 4
>>> a
4
```

Similarly, we can also use the following for…

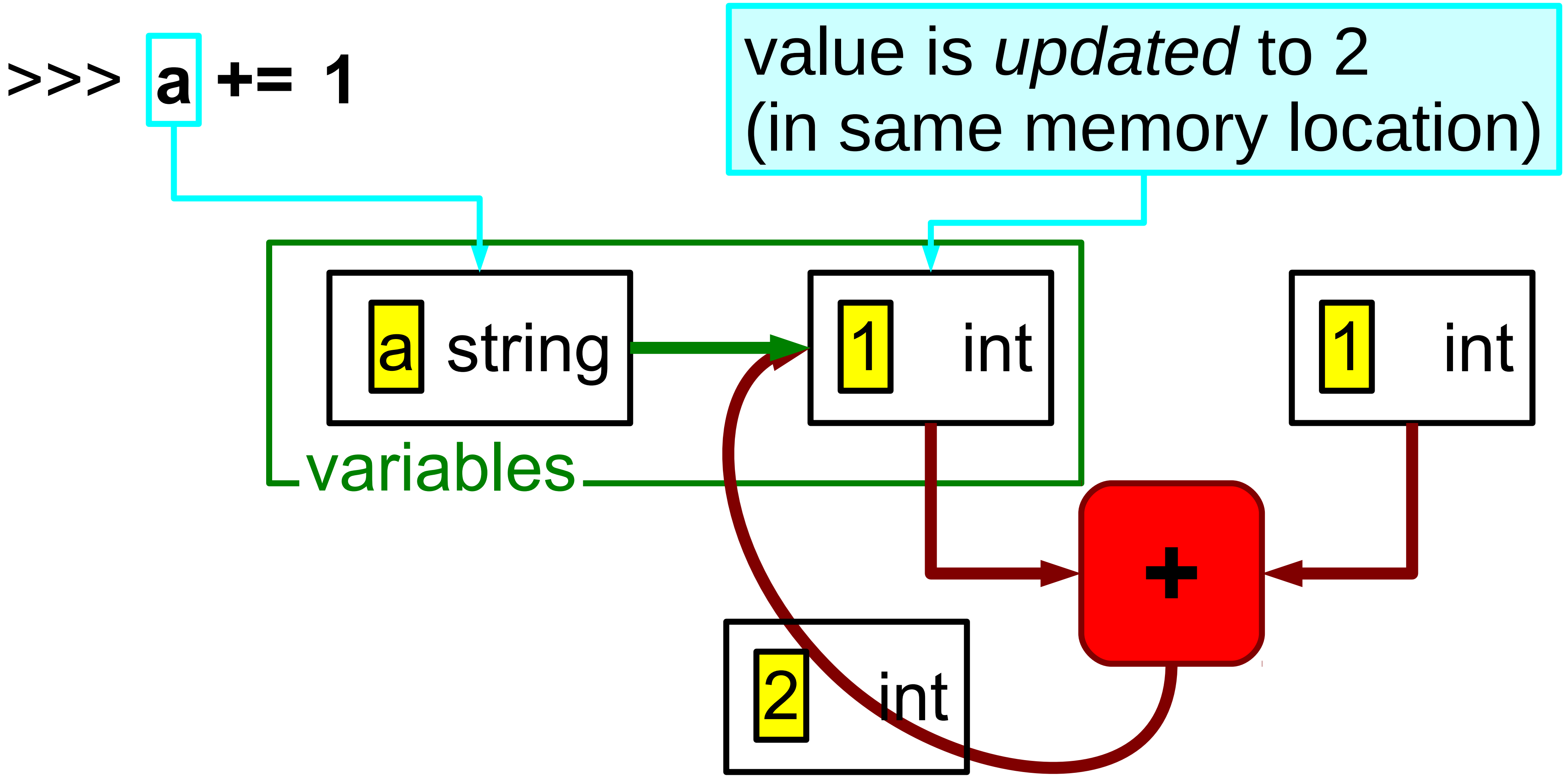
division:	/=
exponentiation:	**=
remainder:	%=

```
>>> a = a + 1
```



Variable is re-assigned to “point” at the answer, which is in a different part of memory

```
>>> a += 1
```

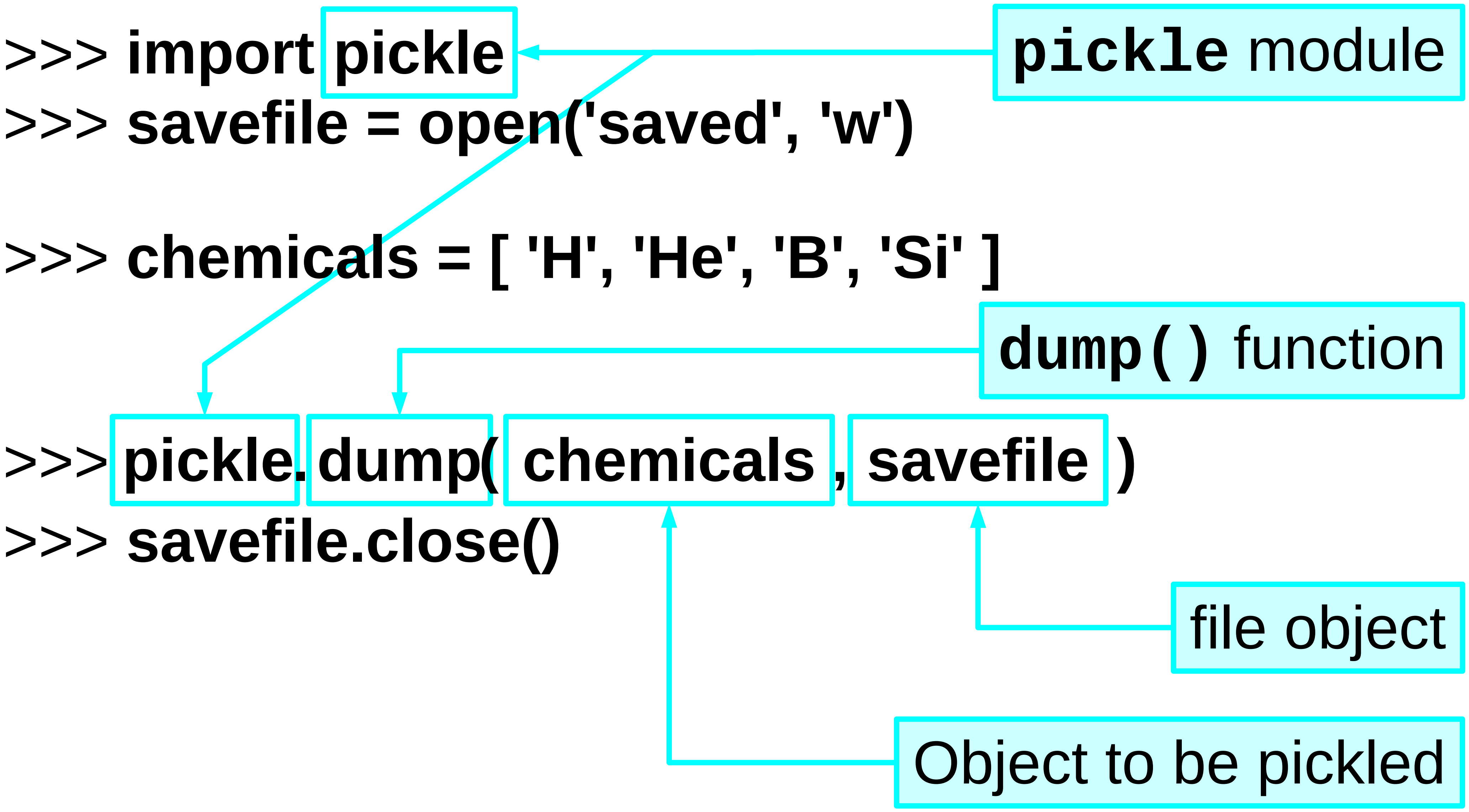


value is *updated* to 2
(in same memory location)
a
string
1
int
1
int
variables
+
2
int

Saving complex objects to a file

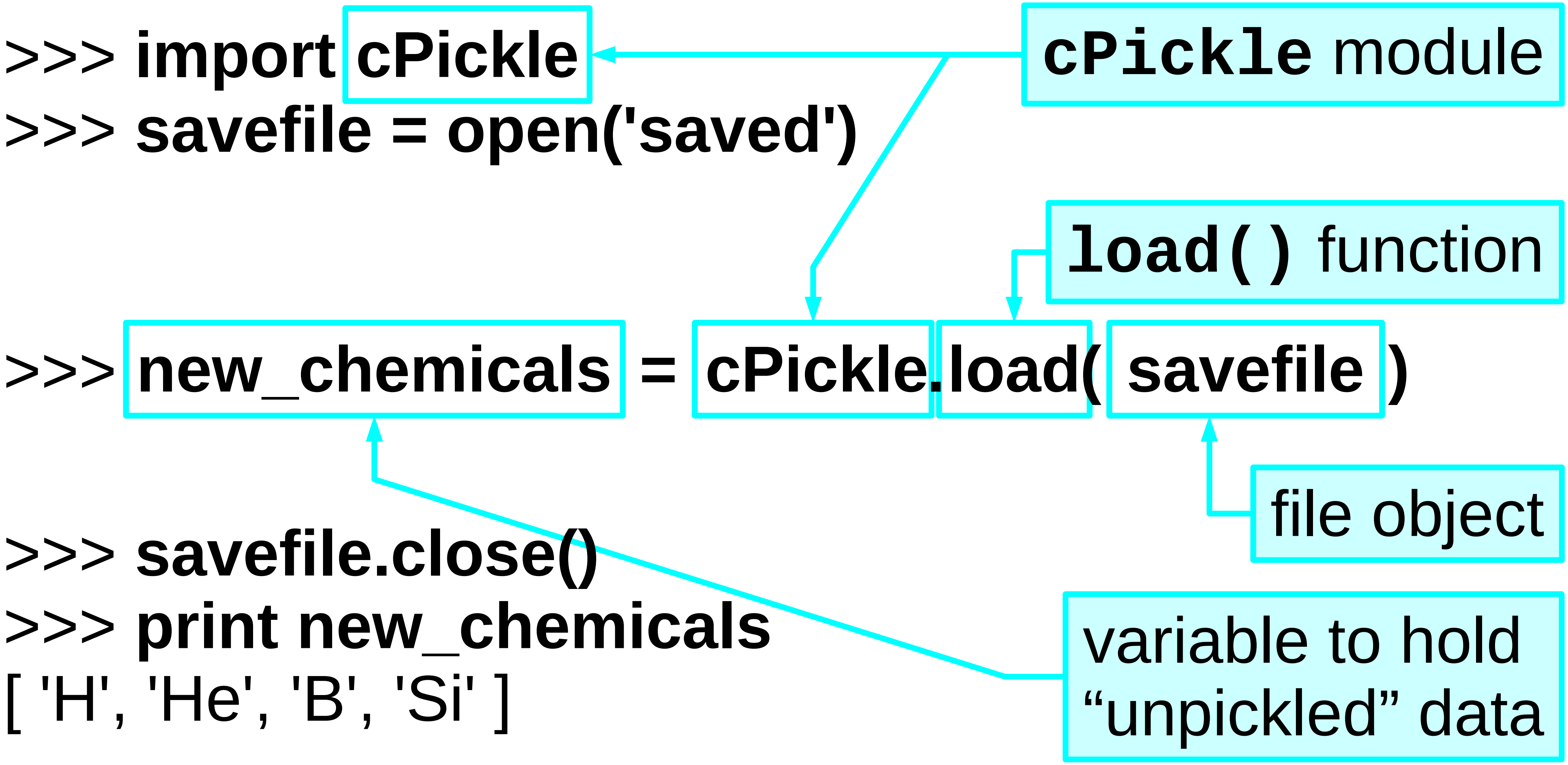
Object serializiation:
`pickle` and **`cPickle`** modules

Pickling data to a file



```
>>> import pickle
>>> savefile = open('saved', 'w')

>>> chemicals = [ 'H', 'He', 'B', 'Si' ]

>>> pickle.dump( chemicals , savefile )
>>> savefile.close()
```

Restoring pickled data



```
>>> import cPickle
>>> savefile = open('saved')

>>> new_chemicals = cPickle.load( savefile )

>>> savefile.close()
>>> print new_chemicals
[ 'H', 'He', 'B', 'Si' ]
```