VTI Academy
WAY TO ENTERPRISE

Basic Programming

Lesson 05

Outline

1. Module and Package
2. File I/O and Resource management

Module

Importing Modules

```
import x

from x import y

from x import y as z
```

Main block

```python
def main():
    "The main function for the program."
    return 42


# This is the "main block"
if __name__ == '__main__':
    main()
```

Modules

Python's basic tool for organizing code

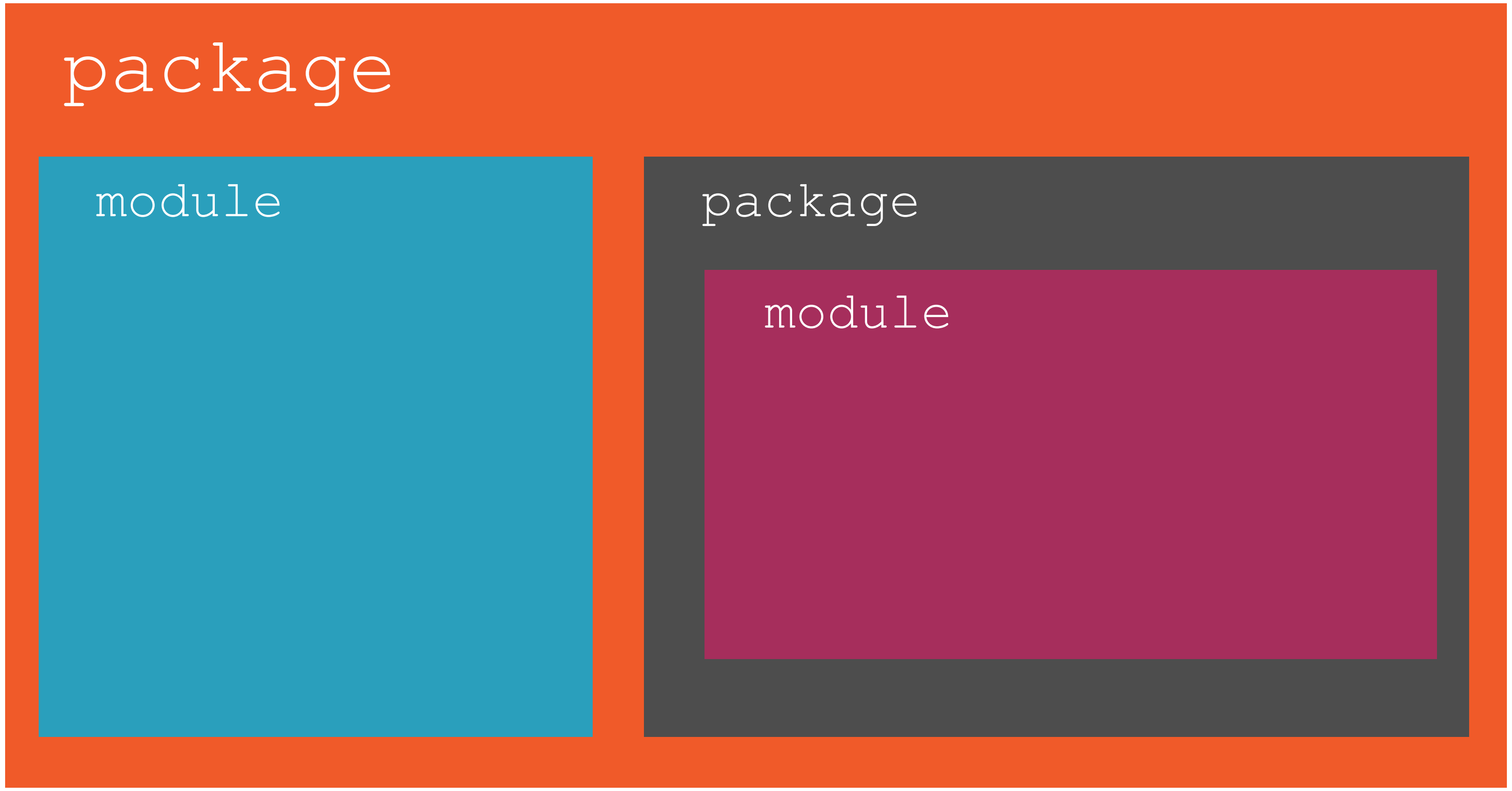
Normally a single Python source file

Load modules with `import`

Represented by `module` objects

Package

Packages is modules that contain other modules



Package vs. Modules

```
package
    module.py
```

Packages are generally directories

```
module.py
```

Modules are generally files

A package is a directory containing `__init__.py`

Project:

`MultiReader`

Read uncompressed text files

Read gzip-compressed files

Read bz2-compressed files

```
import gzip
import sys

opener = gzip.open


if __name__ == '_main_':
    f = opener(
        sys.argv[1],
        mode='wt')
    f.write(' '.join(
        sys.argv[2:]))
    f.close()
```

◀ Alias for `gzip.open`
  – Decompresses during read

◀ "main block"

◀ Use `gzip` to create compressed

◀ file Path to new compressed file

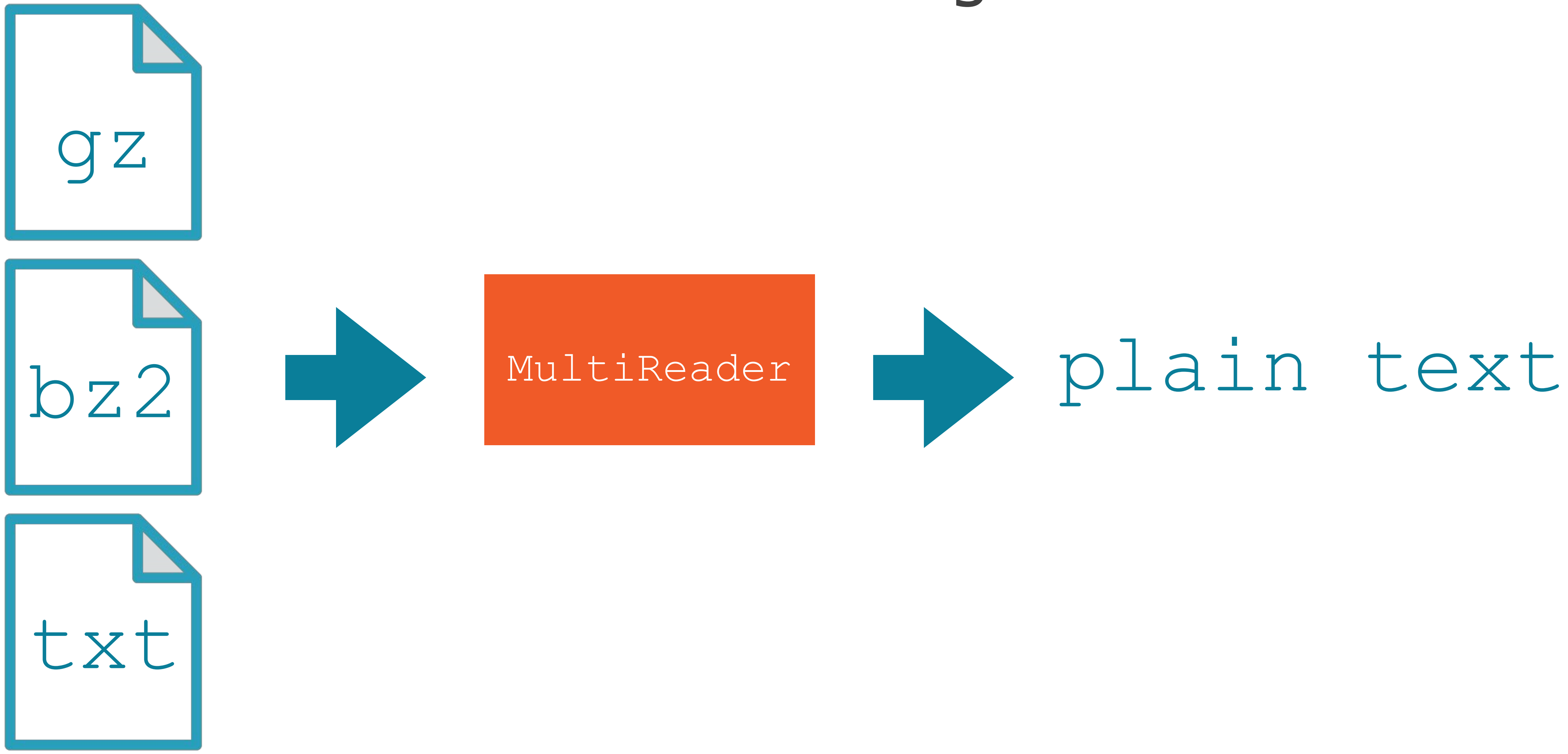
◀ Join to space-separated string

◀ The data to compress

VTI Academy

```
demo_reader
!""  __init__.py
!""  multireader.py
#""  compressed
     !""  __init__.py
     !""  bzipped.py
     #""  gzipped.py
```

MultiReader Program



Key changes to `MultiReader`

Checks for file extension in `extension_map`

If found, specialized opener is used

By default `open()` is used

Relative Imports

Absolute Imports

```python
# Both of these absolute imports mention both ``demo_reader`` and ``compressed``

import demo_reader.compressed.bzipped
from demo_reader.compressed import bzipped
```

Relative Imports Syntax

```
from ..module_name import name
```

Parent of current module

Grandparent of current module

Important Rules for Relative Imports

You can only use the

`from module import name`

form of import

Relative imports can only be used to import modules within the current top-level package

Relative Imports from

`demo_reader/compressed/bzipped.py`

Relative	Absolute
`from . import name`	`from demo_reader.compressed import name`
`from .. import name`	`from demo_reader import name`
`from ..util import name`	`from demo_reader.util import name`

Summary of Relative Imports

Can reduce typing in deeply nested package structures

Promote a certain form of modifiability

In general, prefer absolute imports

`__all__`

Module-level attribute

Controls `from module import *` behavior

If not specified, import all public names

Must be a list of strings

- Each entry is a name to import

While `__all__` can be useful…

…we recommend avoiding `import *` in general

File I/O and Resource management

Resources

Program elements that must be released or closed after use

Python provides special syntax for managing resources

open()

Open a file for reading or writing

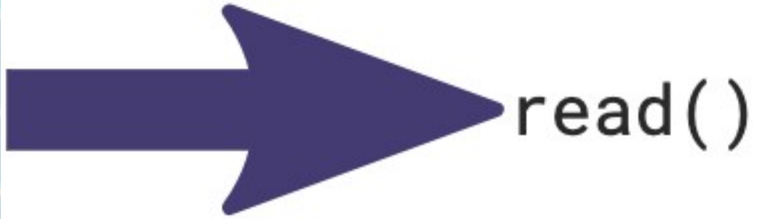
`file`: the path to the file (required)

`mode`: read, write, or append, plus binary or text

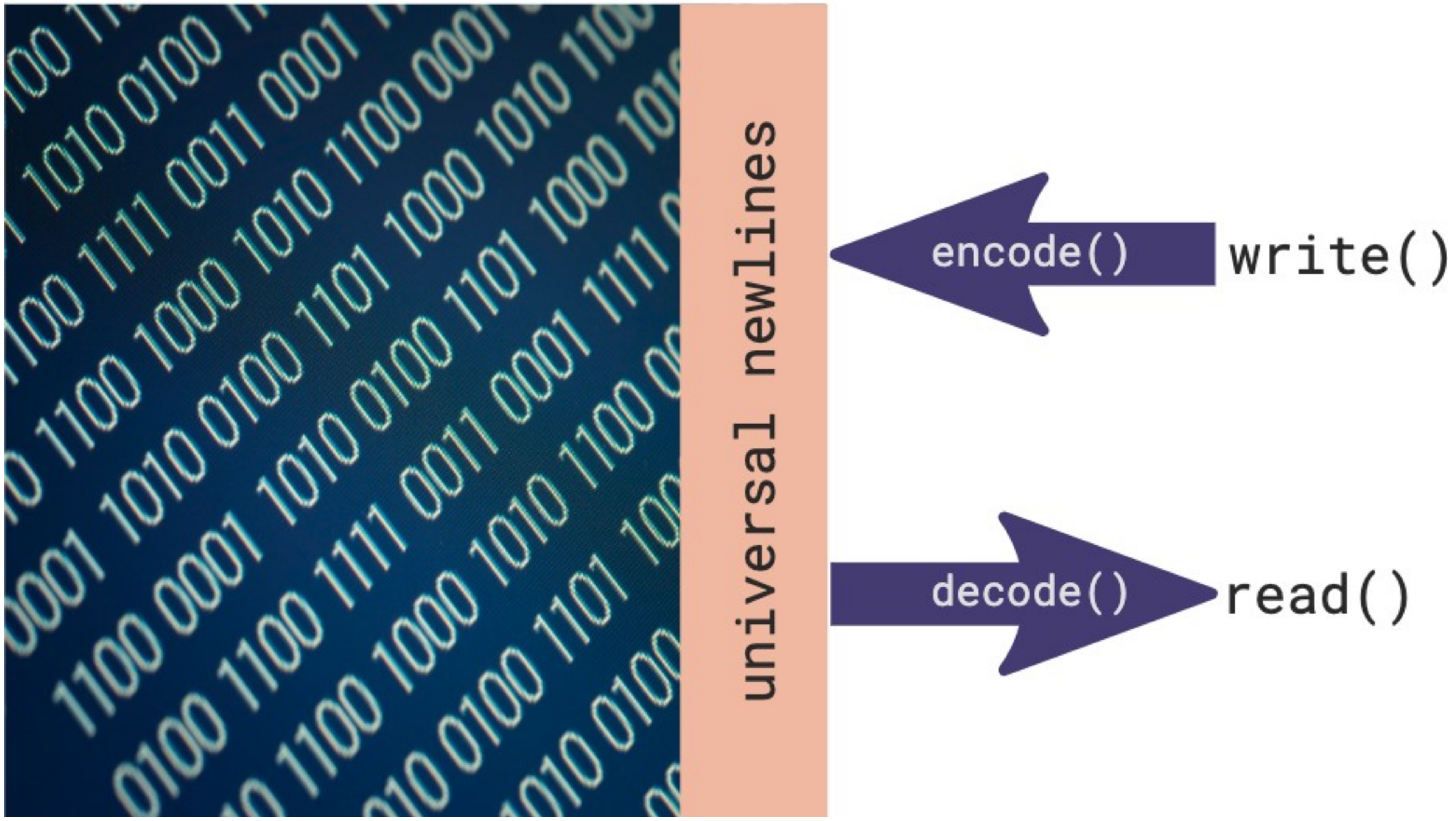
`encoding`: encoding to use in text mode

Files Are Stored as Binary

Binary Mode



Text Mode

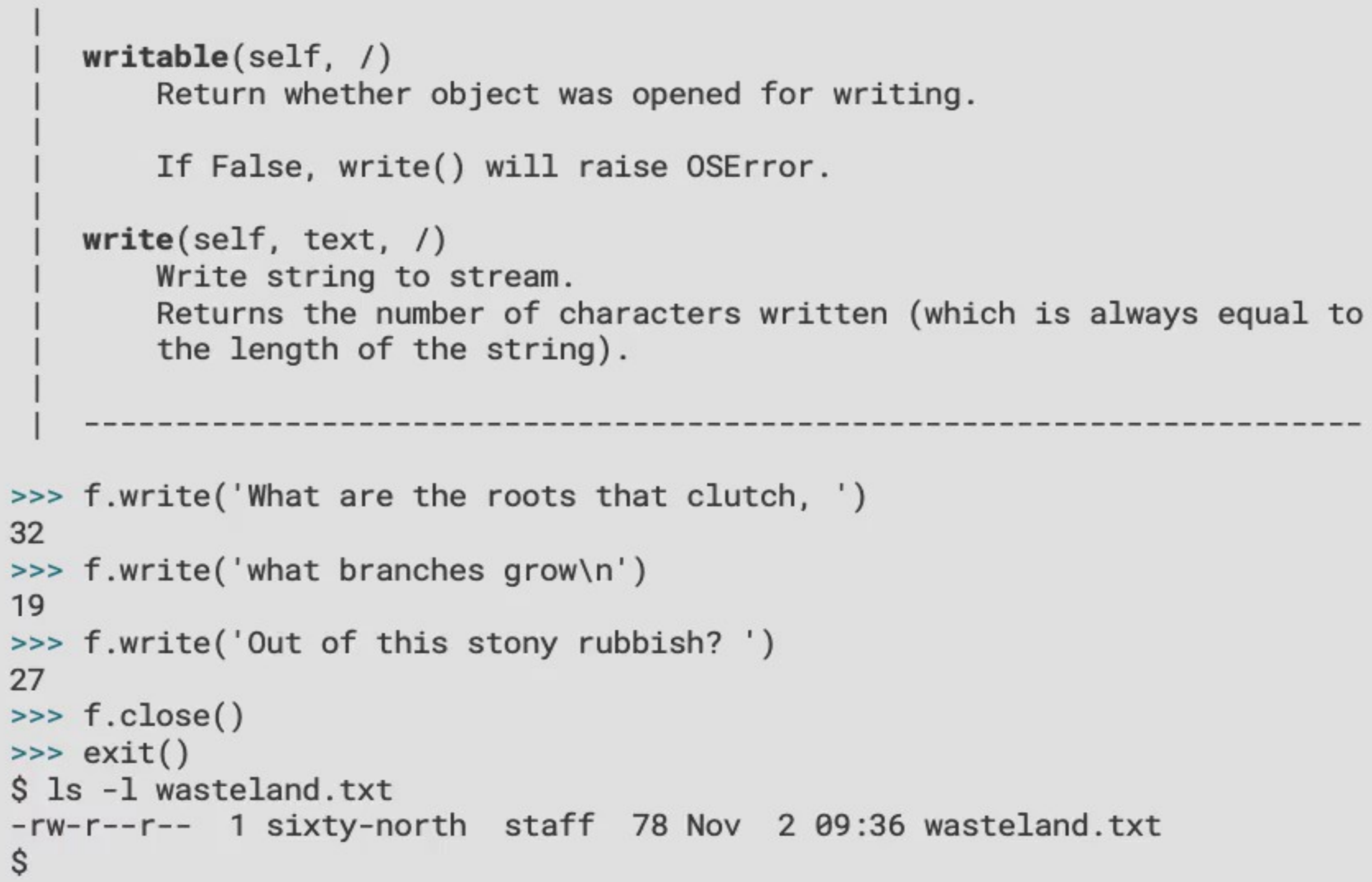


Default Encoding

```
>>> import sys
>>> sys.getdefaultencoding()
'utf-8'
>>>
```

Writing text file

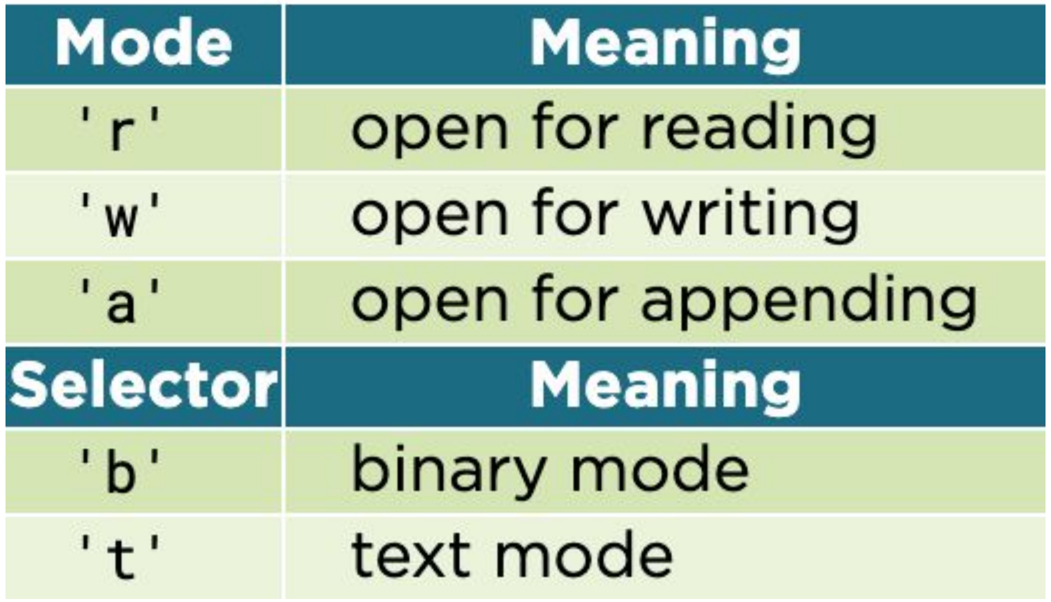
Writing Text



```
 |
 |  writable(self, /)
 |      Return whether object was opened for writing.
 |
 |      If False, write() will raise OSError.
 |
 |  write(self, text, /)
 |      Write string to stream.
 |      Returns the number of characters written (which is always equal to
 |      the length of the string).
 |
 |  ----------------------------------------------------------------------

>>> f.write('What are the roots that clutch, ')
32
>>> f.write('what branches grow\n')
19
>>> f.write('Out of this stony rubbish? ')
27
>>> f.close()
>>> exit()
$ ls -l wasteland.txt
-rw-r--r--  1 sixty-north  staff  78 Nov  2 09:36 wasteland.txt
$
```

open( ) Modes



Mode	Meaning
'r'	open for reading
'w'	open for writing
'a'	open for appending
Selector	**Meaning**
'b'	binary mode
't'	text mode

Open Mode Examples

`'wb'`

Open for writing in binary mode

`'at'`

Open for appending in text mode

`open()` returns a file-like object.

`help()` works on modules, methods, and types.

And it works on instances, too!

`write()` returns the number of codepoints written. Don't sum these values to determine file length.

Reading Text

```
>>> g = open('wasteland.txt', mode='rt', encoding='utf-8')
>>> g.read(32)
'What are the roots that clutch, '
>>> g.read()
'what branches grow\nOut of this stony rubbish? '
>>> g.read()
''
>>> g.seek(0)
0
>>> g.readline()
'What are the roots that clutch, what branches grow\n'
>>> g.readline()
'Out of this stony rubbish? '
>>> g.readline()
''
>>> g.seek(0)
0
>>> g.readlines()
['What are the roots that clutch, what branches grow\n', 'Out of this stony rubb
ish? ']
>>> g.close()
>>>
```

Appending to a file

Appending Text

```
>>> h = open('wasteland.txt', mode='at', encoding='utf-8')
>>> h.writelines(
...     ['Son of man,\n',
...      'You cannot say, or guess, ',
...      'for you know only,\n',
...      'A heap of broken images, ',
...       'where the sun beats\n'])
>>> h.close()
>>>
```

File iteration

```python
# files.py
import sys


f = open(sys.argv[1], mode='rt', encoding='utf-8')
for line in f:
    print(line)
f.close()
```

```
$ python files.py wasteland.txt
What are the roots that clutch, what branches grow

Out of this stony rubbish? Son of man,

You cannot say, or guess, for you know only,

A heap of broken images, where the sun beats

$
```

Use `sys.stdout.write()` instead of print.
This won't add newlines like `print()`.

```python
# files.py
import sys


f = open(sys.argv[1], mode='rt', encoding='utf-8')
for line in f:
    sys.stdout.write(line)
f.close()
```

```
$ python files.py wasteland.txt
What are the roots that clutch, what branches grow
Out of this stony rubbish? Son of man,
You cannot say, or guess, for you know only,
A heap of broken images, where the sun beats
$
```

```python
            c = a + n
        a = c


def write_sequence(filename, num):
    """Write Recaman's sequence to a text file."""
    f = open(filename, mode='wt', encoding='utf-8')
    f.writelines(f"{r}\n"
                 for r in islice(sequence(), num + 1))
    f.close()


if __name__ == '__main__':
    write_sequence(filename=sys.argv[1],
                   num=int(sys.argv[2]))
```

```
$ python recaman.py recaman.dat 1000
$
```

```python
"""Read and print an integer series."""
import sys

def read_series(filename):
    f = open(filename, mode='rt', encoding='utf-8')
    series = []
    for line in f:
        a = int(line.strip())
        series.append(a)
    f.close()
    return series

def main(filename):
    series = read_series(filename)
    print(series)
```

```
53, 1679, 852, 1680, 851, 1681, 850, 1682, 849, 1683, 848, 1684, 847, 1
 1686, 845, 1687, 844, 1688, 2533, 3379, 2532, 3380, 2531, 3381, 2530,
9, 3383, 2528, 3384, 2527, 3385, 2526, 3386, 2525, 3387, 2524, 3388, 2523, 3389,
 2522, 3390, 2521, 1651, 780, 1652, 779, 1653, 778, 1654, 777, 1655, 776, 1656,
775, 1657, 774, 1658, 773, 1659, 772, 1660, 771, 1661, 770, 1662, 769, 1663, 768
, 1664, 767, 1665, 766, 1666, 765, 1667, 764, 1668, 763, 1669, 762, 1670, 761, 1
671, 760, 1672, 759, 1673, 758, 1674, 757, 1675, 756, 1676, 755, 1677, 754, 1678
, 753, 1679, 752, 1680, 751, 1681, 750, 1682, 749, 1683, 748, 1684, 747, 1685, 7
46, 1686, 745, 1687, 744, 1688, 743, 1689, 742, 1690, 741, 1691, 740, 1692, 739,
 1693, 738, 1694, 737, 1695, 736, 1696, 735, 1697, 734, 1698, 733, 1699, 732, 17
00, 731, 1701, 730, 1702, 729, 1703, 728, 1704, 727, 1705, 726, 1706, 725, 1707,
 724, 1708, 2693, 3679, 2692, 3680, 2691, 3681, 2690, 3682, 2689, 3683, 2688, 36
84, 2687, 3685, 2686, 3686]
$ echo "oops" >> recaman.dat
$ python series.py recaman.dat
Traceback (most recent call last):
  File "series.py", line 18, in <module>
    main(sys.argv[1])
  File "series.py", line 14, in main
    series = read_series(filename)
  File "series.py", line 8, in read_series
    a = int(line.strip())
ValueError: invalid literal for int() with base 10: 'oops'
$
```

```python
def read_series(filename):
    try:
        f = open(filename, mode='rt', encoding='utf-8')
        series = []
        for line in f:
            a = int(line.strip())
            series.append(a)
    finally:
        f.close()
    return series

def main(filename):
    series = read_series(filename)
    print(series)
```

Sequence Reader

```
"""Read and print an integer series."""
import sys

def read_series(filename):
    try:
        f = open(filename, mode='rt', encoding='utf-8')
        return [int(line.strip()) for line in f]
    finally:
        f.close()

def main(filename):
    series = read_series(filename)
    print(series)
```

File Usage Pattern

```
f = open(...)
# work with file
f.close()
```

If you don't close, you can lose data!

We want a mechanism to pair `open()` and `close()` automatically.

with-block

Control flow structure for managing resources

Can be used with any objects - such as files - which support the context-manager protocol

Using `with` in `read_series()`

```python
def read_series(filename):
    with open(filename, mode='rt', encoding='utf-8') as f:
        return [int(line.strip()) for line in f]
```

Using `with` in `write_sequence()`

```
def write_sequence(filename, num):
    """Write Recaman's sequence to a text file."""
    with open(filename, mode='wt', encoding='utf-8') as f:
        f.writelines(f"{r}\n"
                     for r in islice(sequence(), num + 1))
```

Expansion of the with-block

```
with EXPR as VAR:
    BLOCK
```

```
mgr = (EXPR)
    exit = type(mgr).__exit__
    value = type(mgr).__enter__(mgr)
    exc = True
    try:
        try:
            VAR = value
            BLOCK
        except:
            exc = False
            if not exit(mgr, *sys.exc_info()):
                raise
    finally:
        if exc:
            exit(mgr, None, None, None)
```

I/O JSON file

JSON file

Parse content to JSON using json library:

```
import json
with open('data.json', 'r', encoding='utf-8') as f:
    obj = json.load(f)
```

Or you can use double quotation marks, as shown below:

```
import json
with open('data.json', 'w', encoding='utf-8') as f:
    json.dump(obj, f, ensure_ascii=False, indent=4)
```