File I/O and Resource Management

Austin Bingham
COFOUNDER - SIXTY NORTH

@austin_bingham

Robert Smallshire
COFOUNDER - SIXTY NORTH

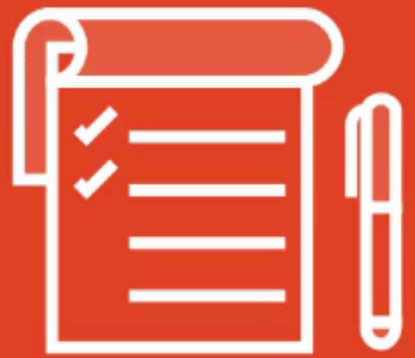
@robsmallshire

Resources

Program elements that must be released or closed after use

Python provides special syntax for managing resources

Overview



Core functions for opening files

Text vs. binary mode

Read and write files

Close files explicitly

Overview

Managing resources with context managers

The `with` keyword for using context managers

With-blocks for running code that uses resources

Using Python to work with binary file formats

The abstraction of file-like objects

Tools for creating context managers

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

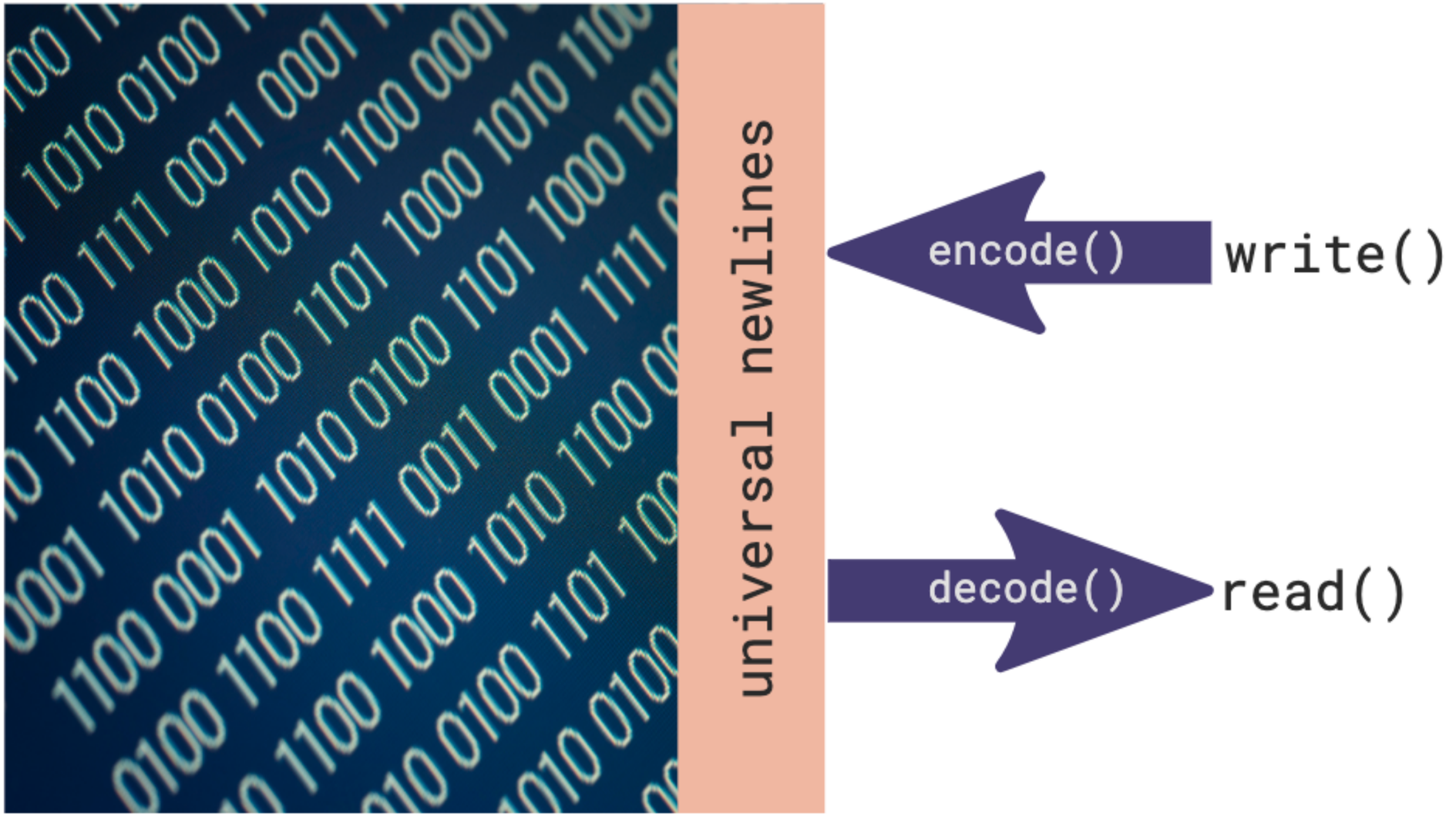
Files Are Stored as Binary

Binary Mode

write()

read()

Text Mode



Default Encoding

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

Writing Text Files

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'`	`'at'`
Open for writing in binary mode	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!

Extra Byte on Windows

```
> dir
 Volume is drive C has no label.
 Volume Serial Number is 36C2-FF83

 Directory of c:\core-python

11/31/2019  20:54                79 wasteland.txt
               1 File(s)             79 bytes
               0 Dir(s)  190,353,698,816 bytes free
```

`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()
>>>
```

Seek cannot move to arbitrary offset. Only 0 and values from `tell()` are allowed. Other values result in undefined behavior.

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
$
```

```
            if c < 0 or c in seen:
                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
$
```

```
"""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, 1685, 846,
 1686, 845, 1687, 844, 1688, 2533, 3379, 2532, 3380, 2531, 3381, 2530, 3382, 252
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'
$
```

```
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)

if __name__ == '__main__':
```

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)

if __name__ == '__main__':
```

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()

```
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))
```

Moment of Zen

Beautiful is better than ugly

Sugary syntax
Faultlessness attained
through
Sweet fidelity

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)
```

Binary Files

```
bmp.write(b'\x00\x00\x00\x00') # Unused pixels per meter
bmp.write(b'\x00\x00\x00\x00') # Unused pixels per meter
bmp.write(b'\x00\x00\x00\x00') # Use whole color table
bmp.write(b'\x00\x00\x00\x00') # All colors are important

# Color palette - a linear grayscale
for c in range(256):
    bmp.write(bytes((c, c, c, 0))) # Blue, Green, Red, Zero

# Pixel data
pixel_data_bookmark = bmp.tell()
for row in reversed(pixels):  # BMP files are bottom to top
    row_data = bytes(row)
    bmp.write(row_data)
    padding = b'\x00' * ((4 - (len(row) % 4)) % 4)  # Pad row to multiple of four bytes
    bmp.write(padding)

# End of file
eof_bookmark = bmp.tell()

# Fill in file size placeholder
bmp.seek(size_bookmark)
bmp.write(_int32_to_bytes(eof_bookmark))

# Fill in pixel offset placeholder
bmp.seek(pixel_offset_bookmark)
bmp.write(_int32_to_bytes(pixel_data_bookmark))
```

Bitwise Operators

Bitwise Operators

```
def _int32_to_bytes(i):
    """Convert an integer to four bytes in little-endian format."""
    # &:  Bitwise-and
    # >>: Right-shift
    return bytes((i & 0xff,
                  i >> 8 & 0xff,
                  i >> 16 & 0xff,
                  i >> 24 & 0xff))
```

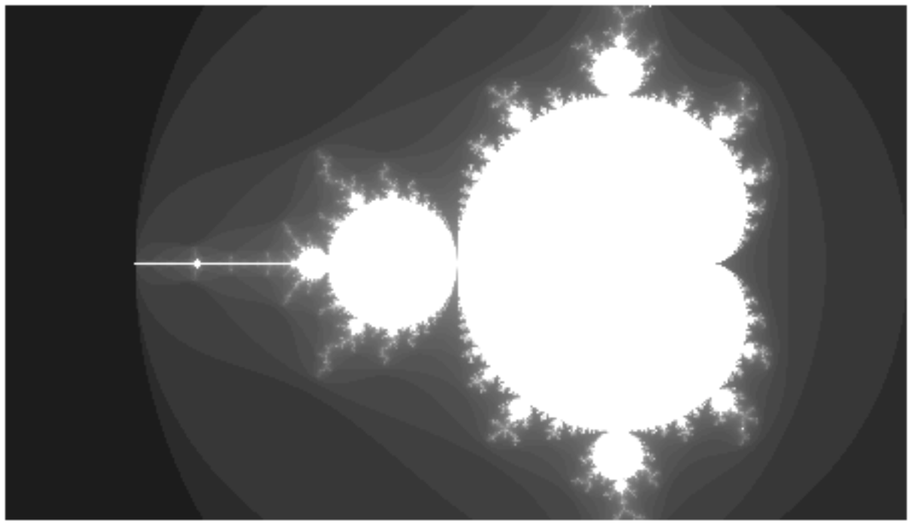
Pixel Data

```
    Args:
        real: The real coordinate
        imag: The imaginary coordinate

    Returns:
        An integer in the range 1-255.
    """
    x = 0
    y = 0
    for i in range(1, 257):
        if x*x + y*y > 4.0:
            break
        xt = real + x*x - y*y
        y = imag + 2.0 * x * y
        x = xt
    return int(math.log(i) * 256 / math.log(256)) - 1
```

```
>>> import fractal
>>> pixels = fractal.mandelbrot(448, 256)
>>> import reprlib
>>> reprlib.repr(pixels)
'[[31, 31, 31, 31, 31, 31, ...], [31, 31, 31, 31, 31, 31, ...], [31, 31, 31, 31,
 31, 31, ...], [31, 31, 31, 31, 31, 31, ...], [31, 31, 31, 31, 31, 31, ...], [31
, 31, 31, 31, 31, 31, ...], ...]'
>>> import bmp
>>> bmp.write_grayscale("mandel.bmp", pixels)
>>>
```

The Mandelbrot Set



Reading Binary Data

```
        ValueError: If the file was not a BMP file.
        OSError: If there was a problem reading the file.
    """

    with open(filename, 'rb') as f:
        magic = f.read(2)
        if magic != b'BM':
            raise ValueError("{} is not a BMP file".format(filename))

        f.seek(18)
        width_bytes = f.read(4)
        height_bytes = f.read(4)

        return (_bytes_to_int32(width_bytes),
                _bytes_to_int32(height_bytes))


def _bytes_to_int32(b):
    "Convert a bytes object containing four bytes into an integer."
    return b[0] | (b[1] << 8) | (b[2] << 16) | (b[3] << 24)
```

```
>>> import bmp
>>> bmp.dimensions("mandel.bmp")
(448, 256)
>>>
```

File-like objects

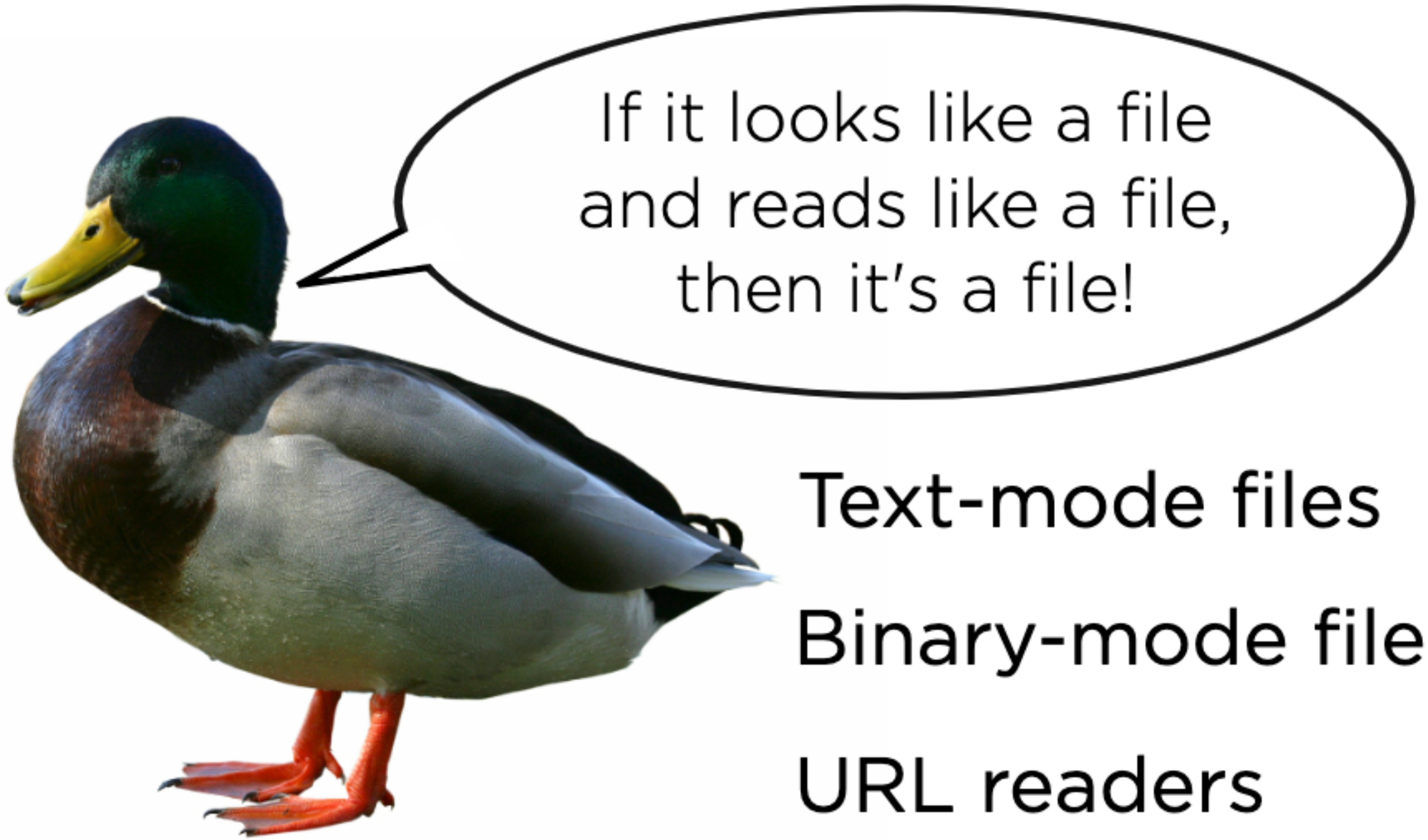
Objects that behave like files

A semi-formal protocol

File behaviors are too varied for a fully-specified protocol

Use an EAFP approach with file-like objects when necessary

Duck-typing and Files



Text-mode files

Binary-mode files

URL readers

```
>>> def words_per_line(flo):
...     return [len(line.split()) for line in flo.readlines()]
...
>>> with open("wasteland.txt", mode='rt', encoding='utf-8') as real_file:
...     wpl = words_per_line(real_file)
...
>>> wpl
[9, 8, 9, 9]
>>> type(real_file)
<class '_io.TextIOWrapper'>
>>> from urllib.request import urlopen
>>> with urlopen("http://sixty-north.com/c/t.txt") as web_file:
...     wpl = words_per_line(web_file)
...
>>> wpl
[6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 7, 8, 14, 12, 8]
>>> type(web_file)
<class 'http.client.HTTPResponse'>
>>>
```

Creating a Simple Context Manager

```
"""Demonstrate raiding a refrigerator."""

class RefrigeratorRaider:
    """Raid a refrigerator"""

    def open(self):
        print("Open fridge door.")

    def take(self, food):
        print(f"Finding {food}...")
        if food == 'deep fried pizza':
            raise RuntimeError("Health warning!")
        print(f"Taking {food}")

    def close(self):
        print("Close fridge door.")

def raid(food):
    r = RefrigeratorRaider()
    r.open()
    r.take(food)
    r.close()
```

- A class for raiding the fridge
- Open the refrigerator door
- Take some food
- Close the refrigerator door
- Raid for some food!

```
>>> from fridge import raid
>>> raid('bacon')
Open fridge door.
Finding bacon...
Taking bacon
Close fridge door.
>>> raid('deep fried pizza')
Open fridge door.
Finding deep fried pizza...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/private/var/folders/0k/58g36_tx22xcxqd9mwqzg_h00000gp/T/tmpm3s632iq/slide_spec/raid-1/fridge.py", line 21, in raid
    r.take(food)
  File "/private/var/folders/0k/58g36_tx22xcxqd9mwqzg_h00000gp/T/tmpm3s632iq/slide_spec/raid-1/fridge.py", line 12, in take
    raise RuntimeError("Health warning!")
RuntimeError: Health warning!
>>>
```

```
        print(f"Finding {food}...")
        if food == 'deep fried pizza':
            raise RuntimeError("Health warning!")
        print(f"Taking {food}")

    def close(self):
        print("Close fridge door.")


def raid(food):
    with closing(RefrigeratorRaider()) as r:
        r.open()
        r.take(food)
        r.close()
```

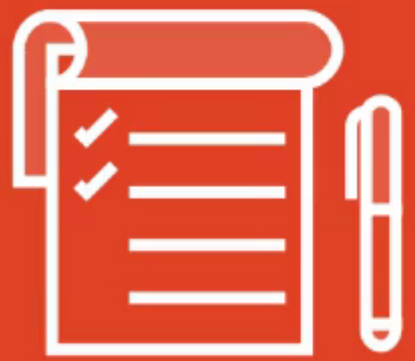
```
>>> from fridge import raid
>>> raid('spam')
Open fridge door.
Finding spam...
Taking spam
Close fridge door.
Close fridge door.
>>>
```

Remove Duplicate `close`

```
def raid(food):
    with closing(RefrigeratorRaider()) as r:
        r.open()
        r.take(food)
        # removed explicit call to close()
```

```
>>> from fridge import raid
>>> raid('deep fried pizza')
Open fridge door.
Finding deep fried pizza...
Close fridge door.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/private/var/folders/0k/58g36_tx22xcxqd9mwqzg_h00000gp/T/tmpqlt_362o/slide_spec/raid-3/fridge.py", line 25, in raid
    r.take(food)
  File "/private/var/folders/0k/58g36_tx22xcxqd9mwqzg_h00000gp/T/tmpqlt_362o/slide_spec/raid-3/fridge.py", line 15, in take
    raise RuntimeError("Health warning!")
RuntimeError: Health warning!
>>>
```

Summary



Files are opened with `open()`

Specify encodings for text-mode files

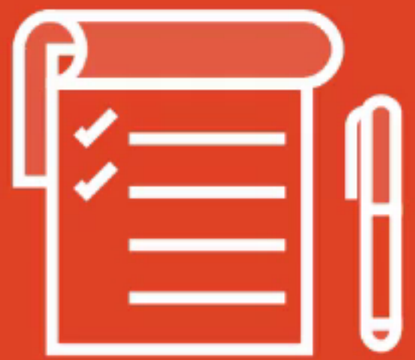
Text-mode works with `str` and does universal newline translation and encoding

Binary-mode works with `bytes` objects

Callers are responsible for providing newlines

Files should always be closed after use

Summary



Files have line-reading support and yield lines on iteration

Files are context managers which close files on exit

The notion of file-like object is loosely defined by very useful

Context managers aren't restricted to file-like objects

`help()` can be called on objects, not just types

Python supports the bitwise operators &, |, <<, and >>

Congratulations!

Look for more Core Python courses

Companion Book Series

leanpub.com/python-apprentice/

Congratulations!