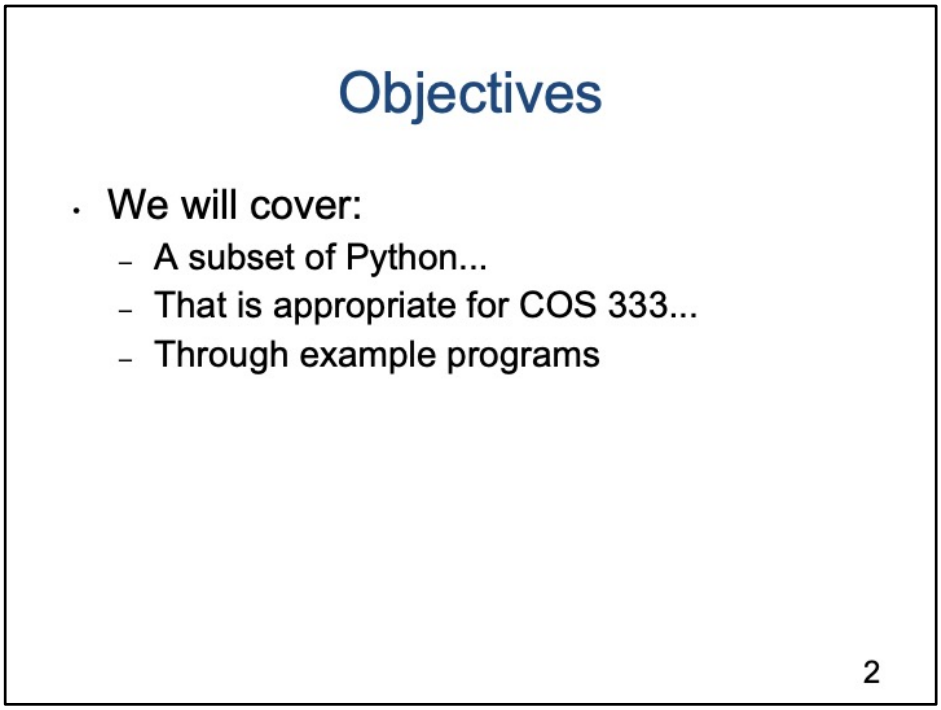
The Python Language (Part 5)

Robert M. Dondero, Ph.D.
Princeton University



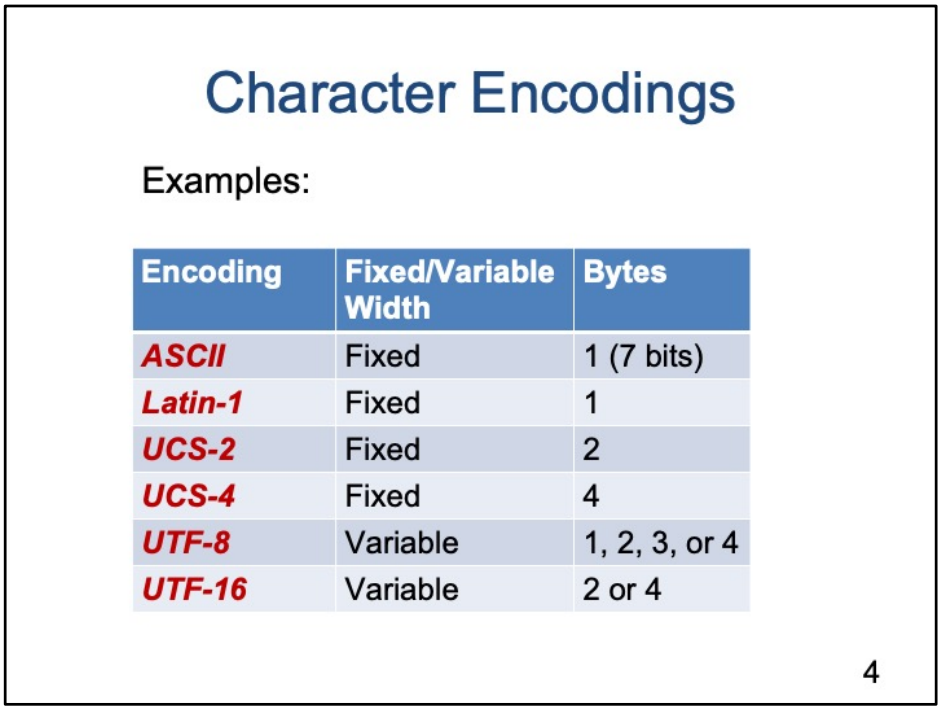
Objectives

Continued from Part 4

[see slide]

Agenda

- **Prelim: character and string encodings**
- Files
- The “with” statement
- Arrays
- Associative arrays
- Iterable classes



Character Encodings

As you know, any modern computer deals with all data as bytes
1 byte = 8 bits
To deal with **characters**, any computer must use a **character encoding**
A character encoding maps each character to the byte(s) which represent it

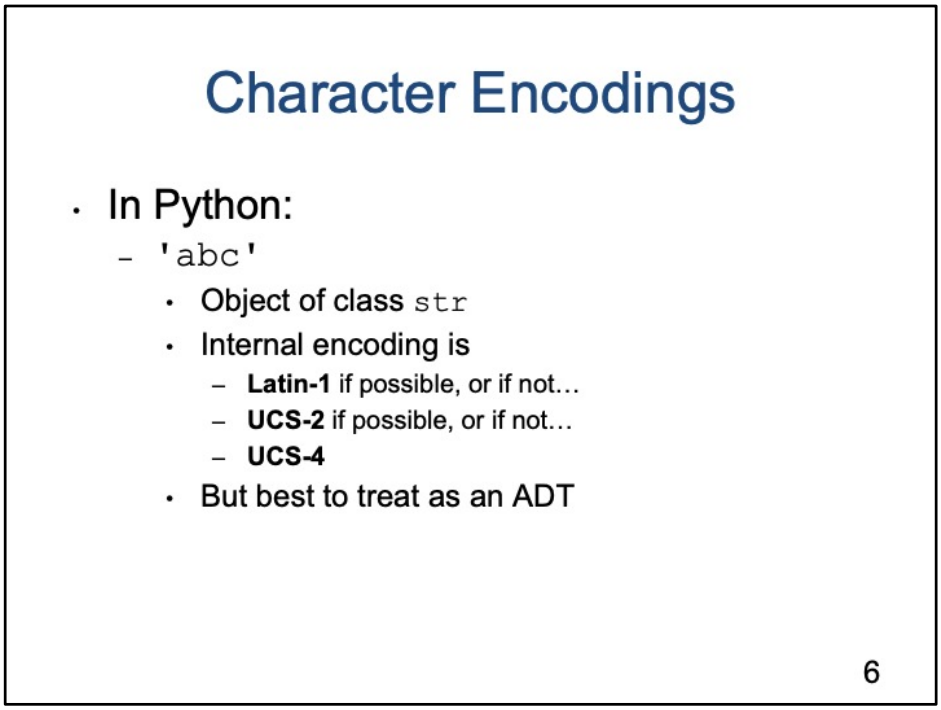
There are many character encodings

[see slide]

ASCII is a fixed-width encoding in which each character is mapped to 1 byte
…

Character Encodings

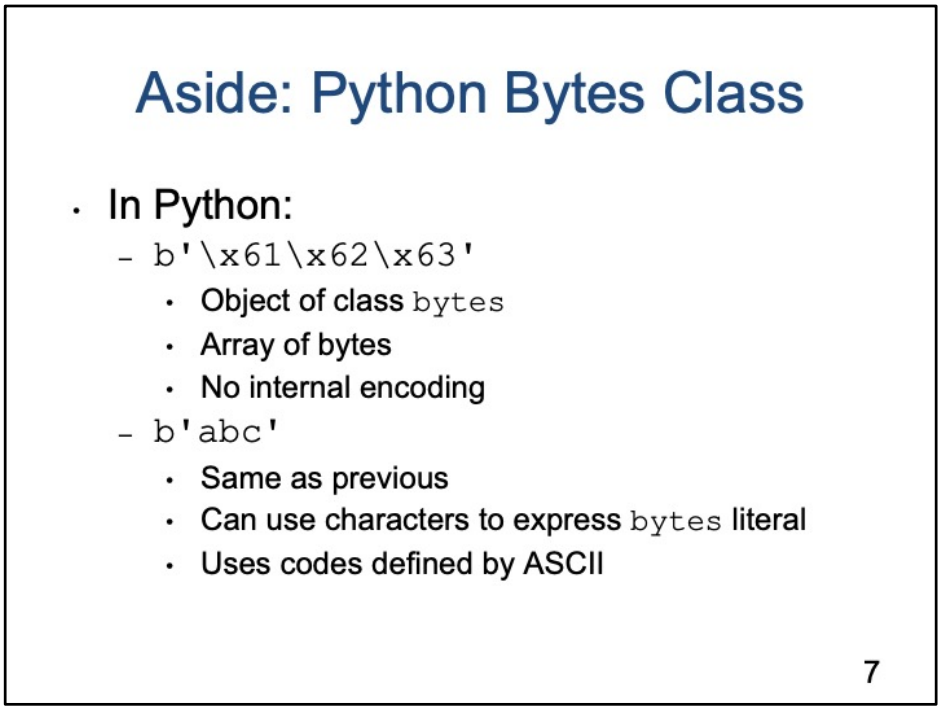
Character	ASCII	Latin-1	UCS-2	UCS-4	UTF-8	UTF-16
space	20	20	0020	00000020	20	0020
!	21	21	0021	00000021	21	0021
0	30	30	0030	00000030	30	0030
A	41	41	0041	00000041	41	0041
a	61	61	0061	00000061	61	0061
a with grave		e0	00e0	000000e0	e0	00e0
Greek small pi			03c0	000003c0	cf80	03c0
Double prime			2033	00002033	e280b3	2033
Aegean number 2000				00010123	f09084a3	d800dd23



Character Encodings

[see slide]

It's best to pretend you don't know how str objects are stored
It's best to treat str as an abstract data type



Character Encodings

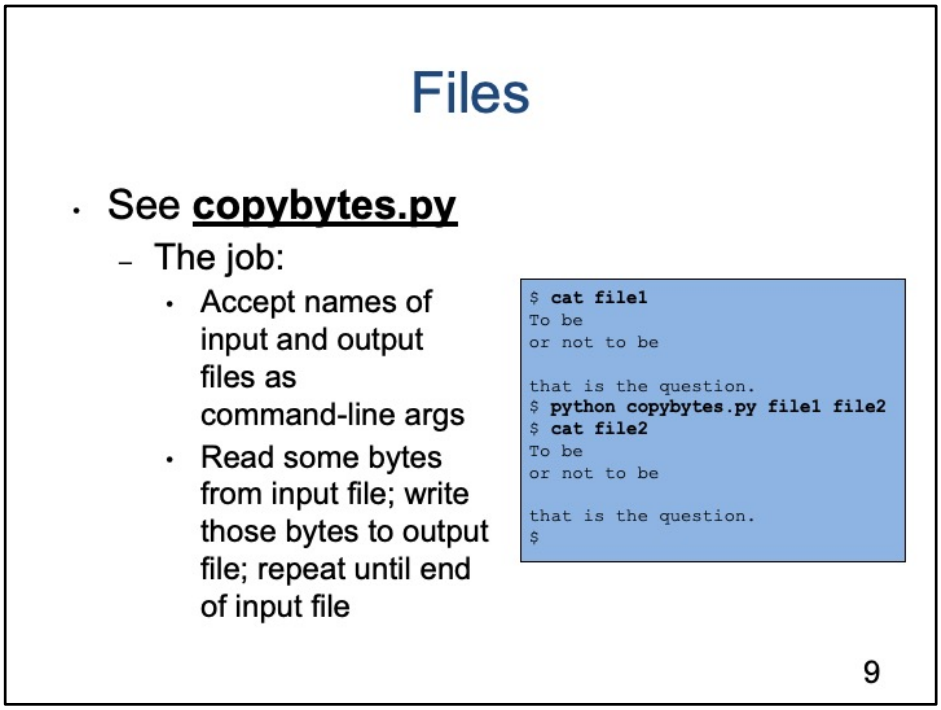
Python also has a related data type; bytes

[see slide]

We'll use bytes objects when doing network programming (soon)

Agenda

- Prelim: character and string encodings
- **Files**
- The “with” statement
- Arrays
- Associative arrays
- Iterable classes



Files

[see slide]

Notes:

Command-line arguments

sys.argv is a list

sys.argv[0] is the name of the program
sys.argv[1] is the first command-line arg
sys.argv[2] is the second command-line arg
…

len() function is defined in __builtin__ module

len(somelist) => somelist.__len__() => number of elements in somelist

File I/O

in_file = open(in_file_name, mode='rb')

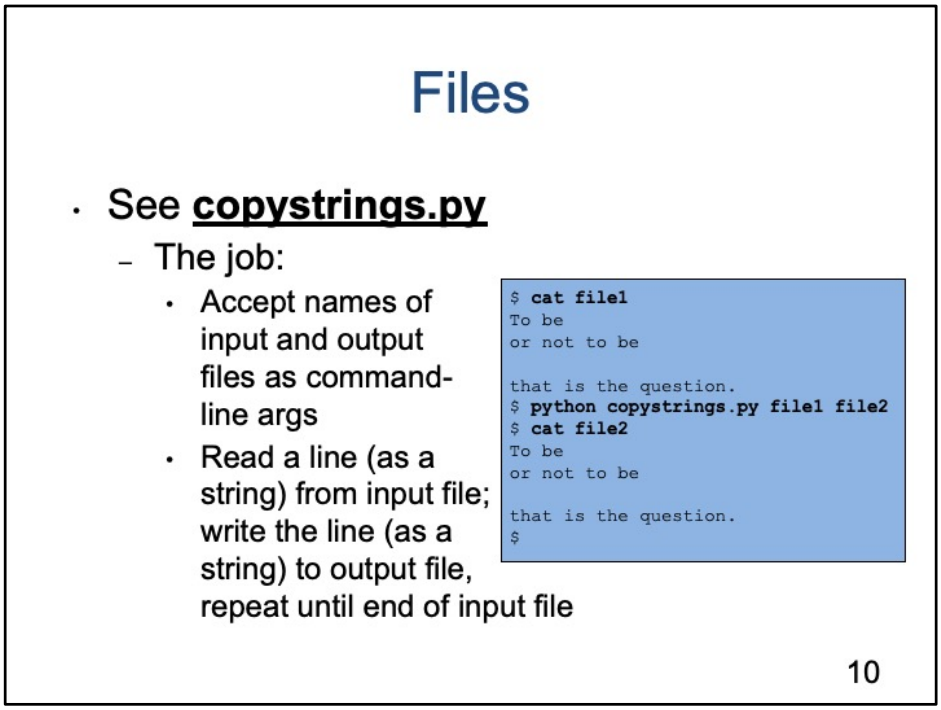
in_file.read(MAX_BYTE_COUNT) reads up to MAX_BYTE_COUNT bytes from in_file, returns bytes object

out_file = open(out_file_name, mode='wb')

out_file.write(byte_array) writes bytes from bytes object b to out_file

```
in_file.close()
```

This program: to be polite; other programs: necessary



Files

[see slide]

Notes

File I/O

in_file = open(inFileName, mode='r', encoding='utf-8')

in_file contains strings encoded as UTF-8

in_file .readline() reads bytes comprising 1 line from in_file , *decodes* to str object, returns str object

out_file = open(outFileName, mode='w', encoding='utf-8')

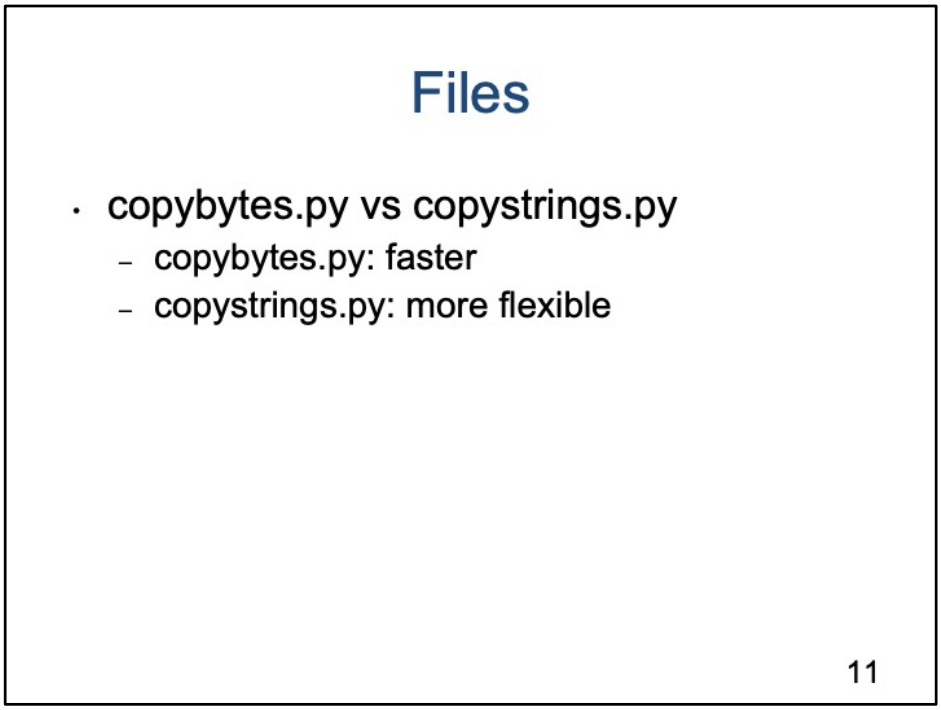
out_file should contain strings encoded as UTF-8

out_file.write(line) *encodes* line to bytes and writes to out_file

Implicit call of readline() via for statement!

for statement can iterate over any Iterable object – even a file!

More on that later



Files

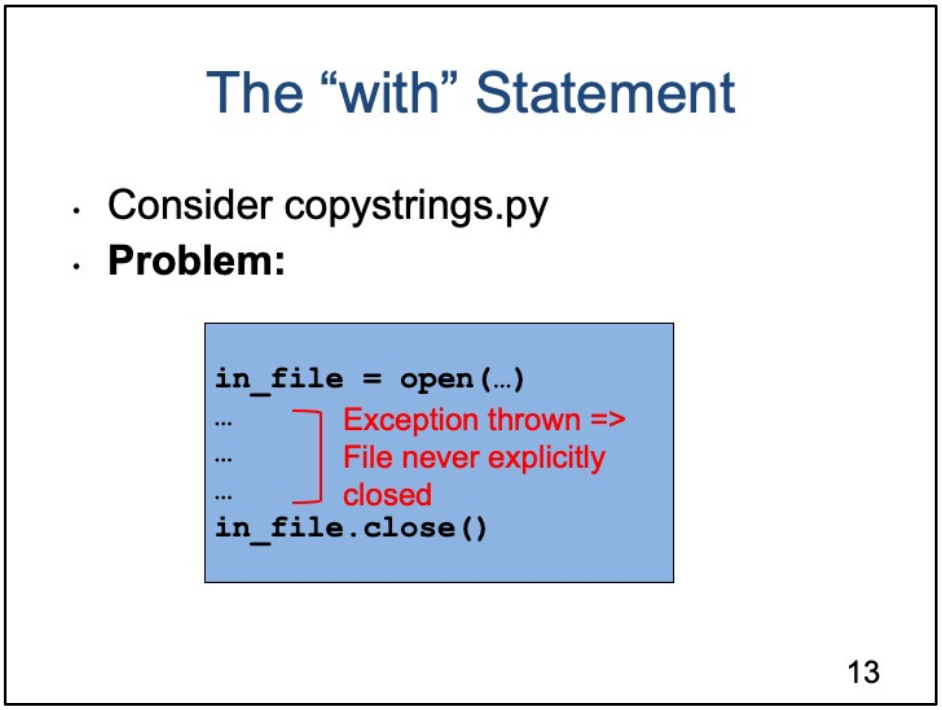
copybytes.py vs. copystrings.py

copybytes.py: faster
No decoding or encoding

copystrings.py: more flexible
Data can be manipulated as Strings
Sorted, concatenated, uppercase, …

Agenda

- Prelim: character and string encodings
- Files
- **The "with" statement**
- Arrays
- Associative arrays
- Iterable classes



The “with” Statement

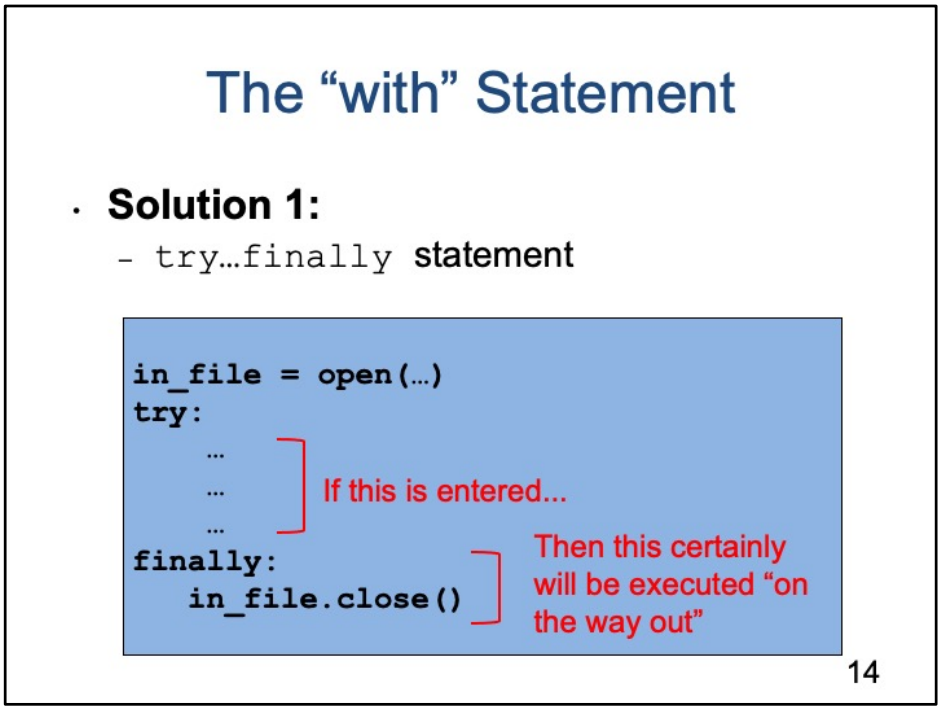
[see slide]

If an exception is thrown in the area between open() and close(), then close() is never called

In copystrings.py: no problem
The process will exit soon anyway

In other programs: maybe a problem

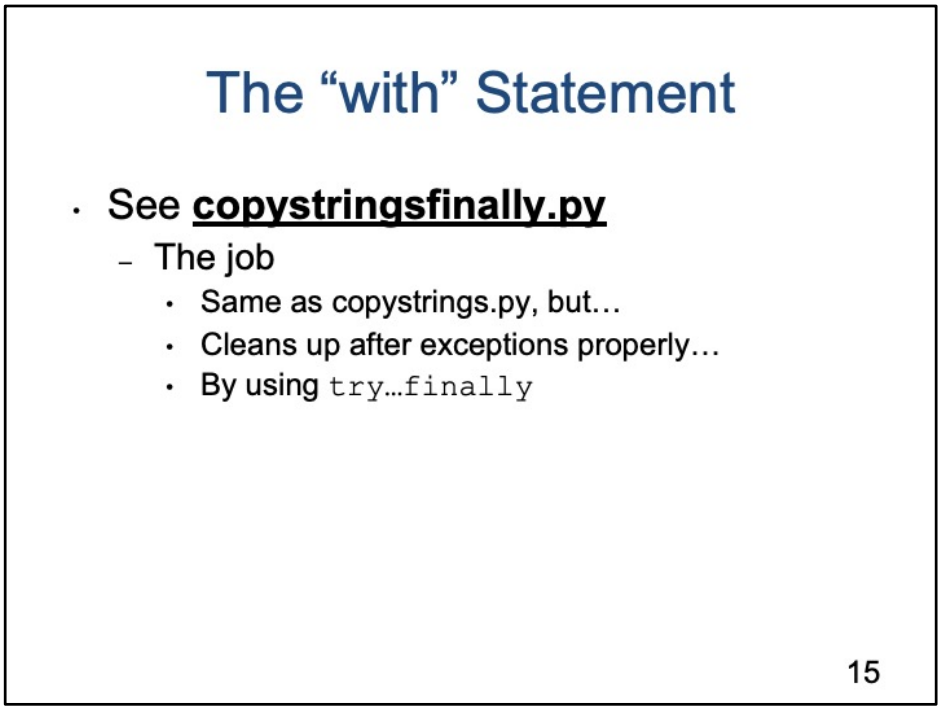
In general, whenever a process opens a file it should close the file, no matter what



The “with” Statement

[see slide]

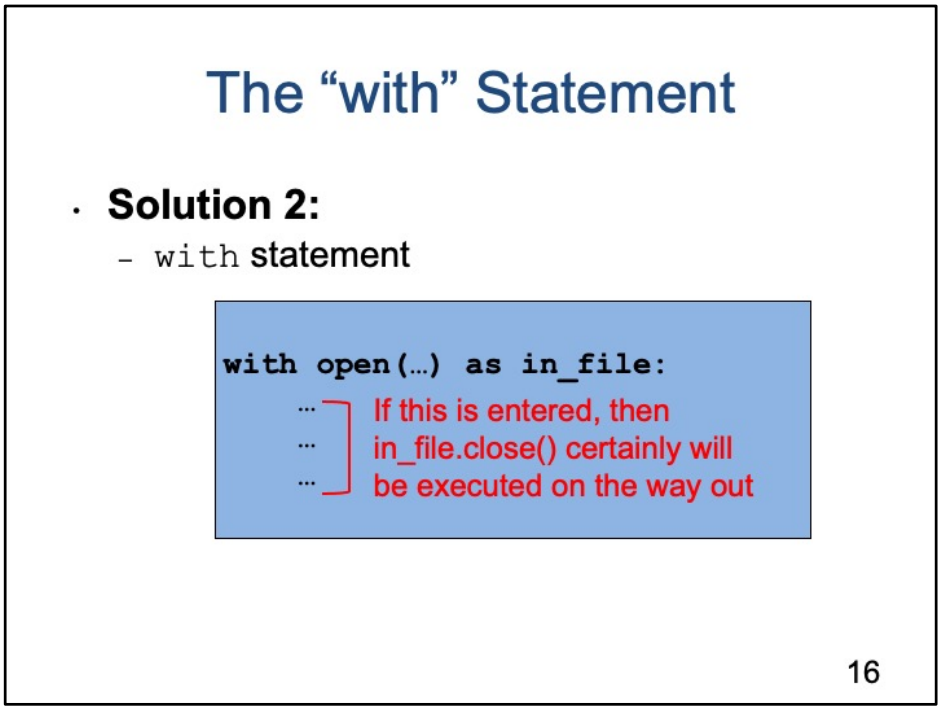
If the try statement is entered, then
in_file.close() certainly will be executed on the way out
whether or not the statements within the try statement throw an exception



The “with” Statement

[see slide]

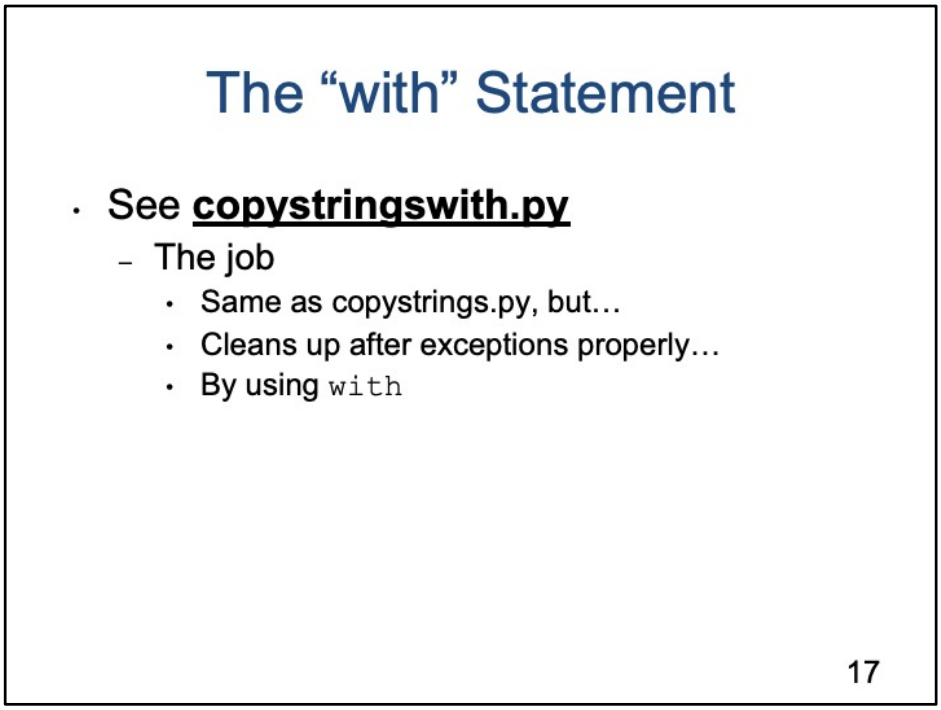
Awkward; error prone; hard to read
Common!



The “with” Statement

[see slide]

New to Python 2.6 and 3.0
If the with statement is entered, then
f.close() certainly will be executed automatically on the way out
whether or not the statements within the with statement throw an exception



The “with” Statement

[see slide]

Less awkward; less error prone, easier to read

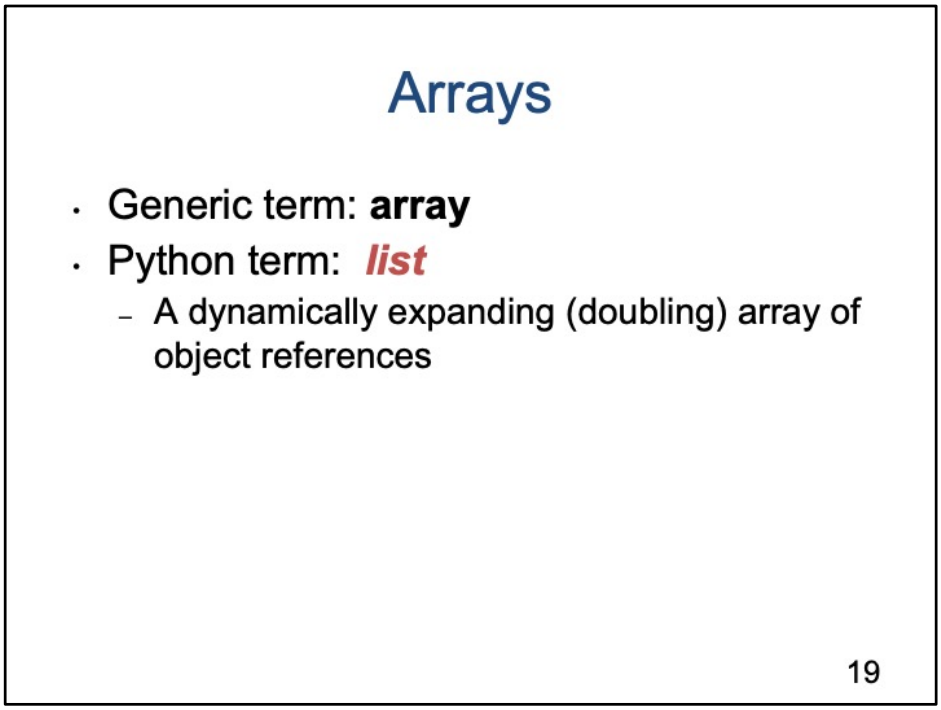
Generalizing:

When you open a file, you should make sure you close it
As we’ll see…
When you open a network socket, you should make sure you close it
When you open a DB connection, you should make sure you close it
When you open a DB cursor, you should make sure you close it
When you acquire a lock on an object, you should make sure you release it

In those situations, you should use the with statement

Agenda

- Prelim: character and string encodings
- Files
- The “with” statement
- **Arrays**
- Associative arrays
- Iterable classes



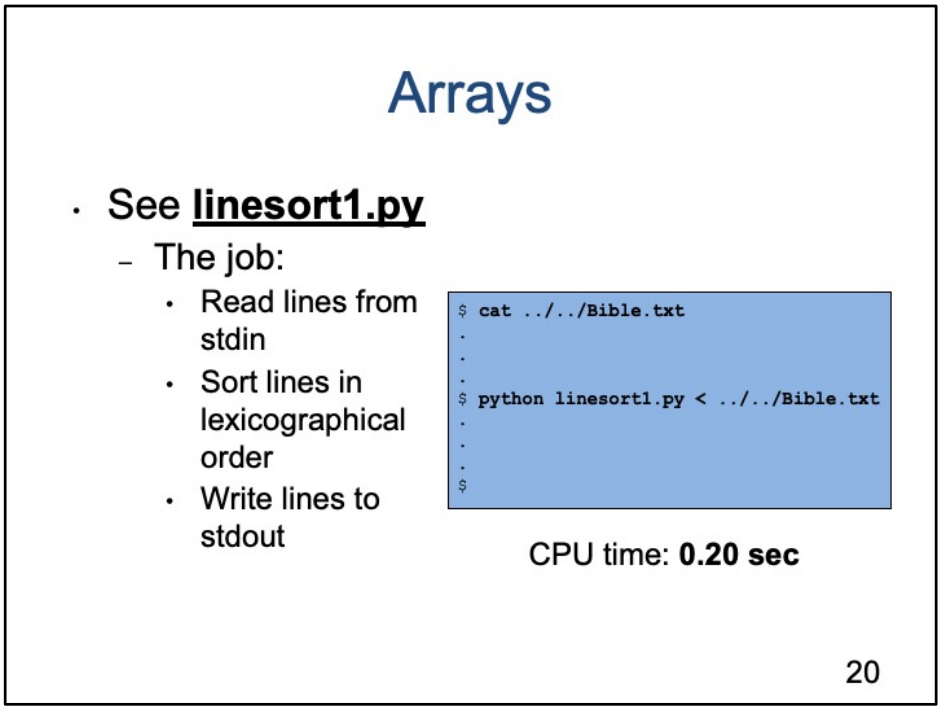
Arrays

Generic term: **array**

Wikipedia: "A data structure consisting of a collection of elements…, each identified by at least one array index or key. An array is stored such that the position of each element can be computed from its index tuple by a mathematical formula."

Python term: **list**

Python offers a list class
Not a linked list!
A dynamically expanding (doubling) array of object references



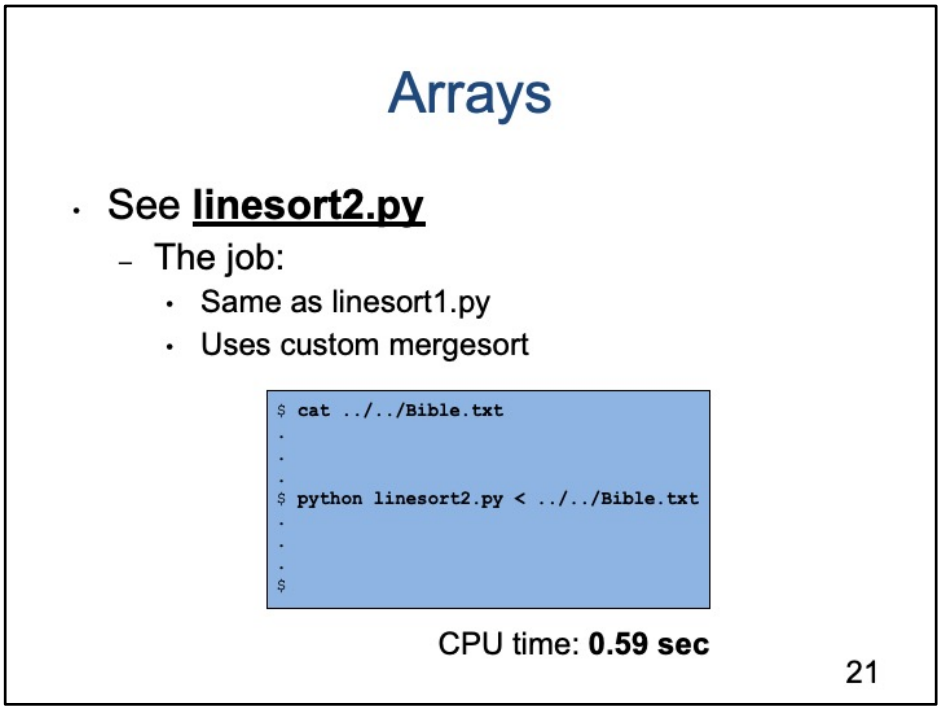
Arrays

[see slide]

The sort() method in the list class uses…

timsort

A hybrid of merge sort and insertion sort
Guaranteed to be stable



Arrays

[see slide]

Notes:

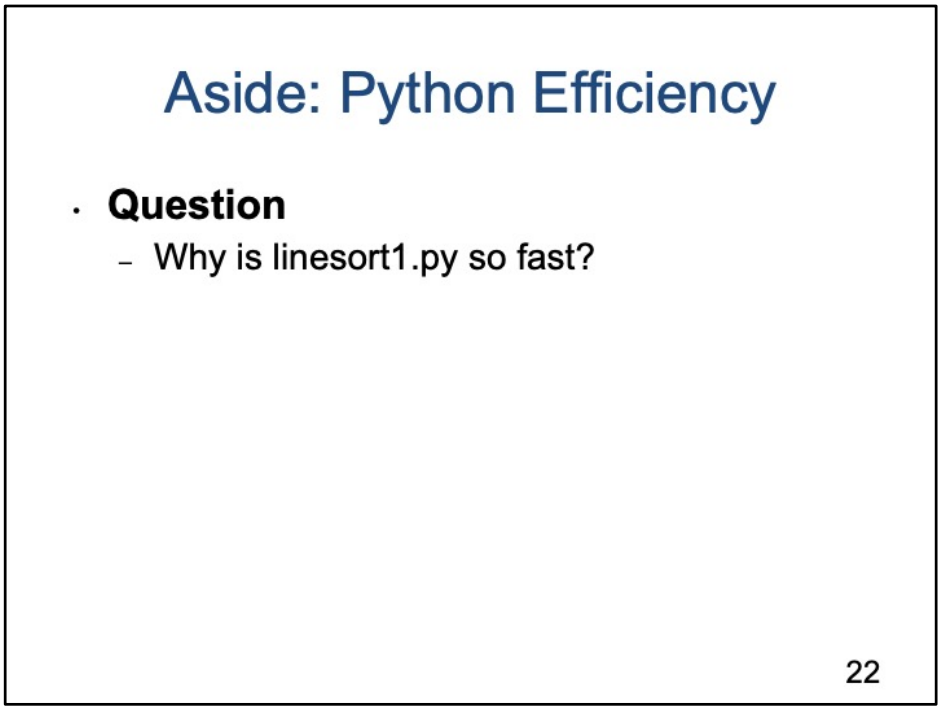
Uses custom mergesort

Performance:

CPU time to sort Bible.txt: 0.59 sec

Much slower than linesort1.py

Slower than Java version



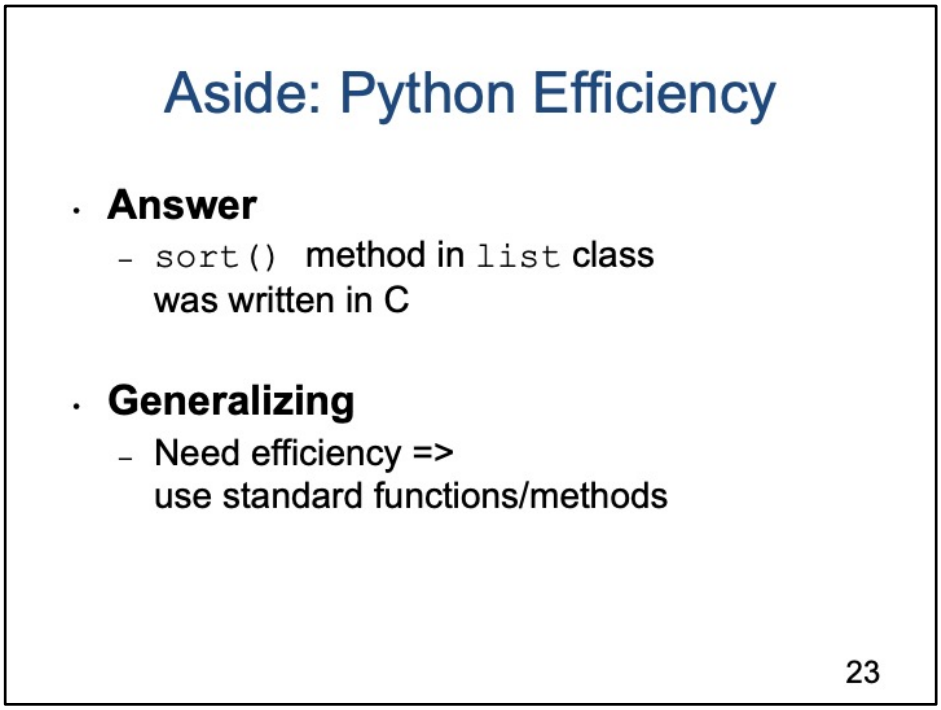
Aside: Python Efficiency

Question

Why is linesort1.py so fast?

Why is linesort1.py much faster than linesort2.py?

Why is linesort1.py faster than Java equivalent?



Aside: Python Efficiency

Answer

sort() method in list class was written in C, not Python

And is present at run-time in machine language!

All of the Python standard functions and classes were written in C

That's why the standard functions and methods are so fast

Generalizing

Don't count on Python handling **your** functions/methods so efficiently!

If you want/need speed, try to avoid doing work using your own functions/methods

Constrain yourself to using the standard functions/methods as much as you can

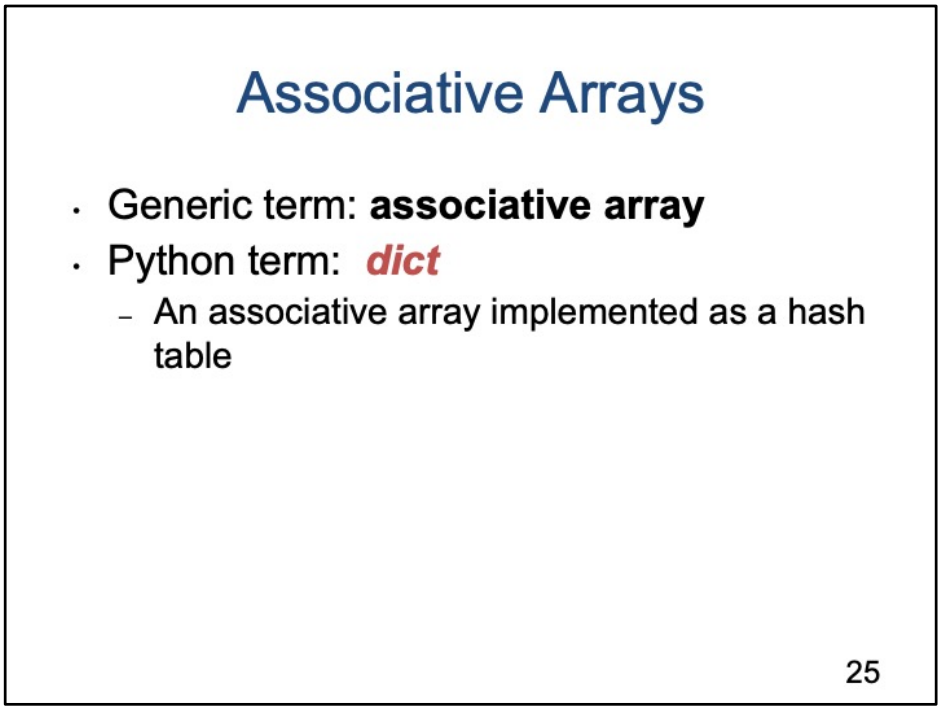
Commentary

Python is awkward for a CS1 + CS2 course sequence

Will Python ever replace Java as the most popular language for CS1 courses?

Agenda

- Prelim: character and string encodings
- Files
- The “with” statement
- Arrays
- **Associative arrays**
- Iterable classes



Associative Arrays

Generic term: **associative array**

Wikipedia: "An abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection"
Typical implementation: red-black tree, hash table, …

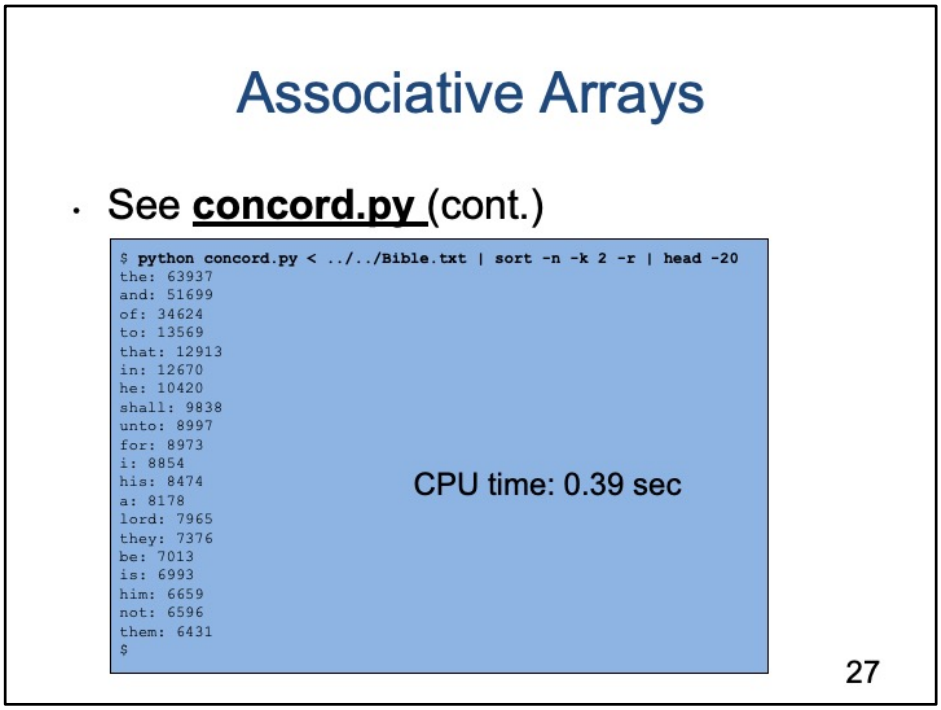
Python term: **dict**

Python offers a dict class
A dict object is a "dictionary"
An associative array implemented as a hash table

Associative Arrays

- See **concord.py**
  - The job:
    - Read words from stdin
    - Write a concordance to stdout
      - Each word and its occurrence count

```
$ cat ../../Bible.txt
.
.
.
$ python concord.py < ../../Bible.txt
welcome: 1
to: 13569
you: 2621
have: 3905
.
.
.
chrysolyte: 1
chrysoprasus: 1
transparent: 1
proceeding: 1
$
```



Associative Arrays

[see slide]

Notes:

- `dict` object
  - An associative array
  - `in` operator
  - Indexing using `[]` operator
  - Iteration using `for` statement
- Regular expressions
  - `[a-z]`
    - Used to find a lower-case letter
  - `[a-z]+`
    - Used to find a sequence of 1 or more lower-case letters
- Raw strings
  - `r'[a-z]+'`
  - Backslash is not interpreted as an escape character
- Performance
  - Python: Time to handle Bible.txt: 0.39 sec
  - Java: Time to handle Bible.txt: 0.47sec

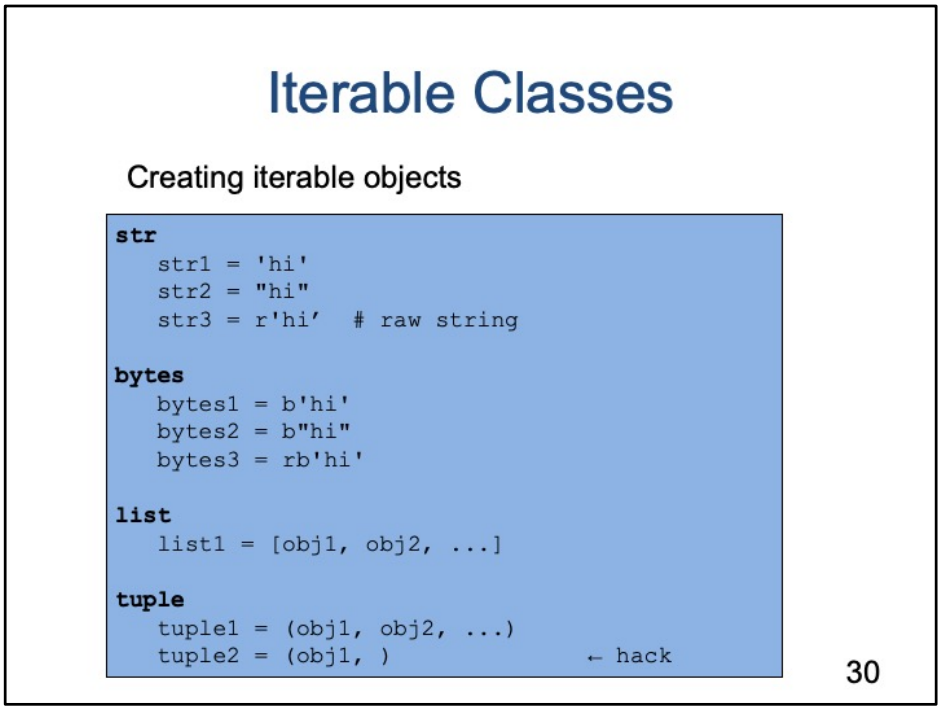
Generalizing...

Agenda

- Prelim: character and string encodings
- Files
- The “with” statement
- Arrays
- Associative arrays
- **Iterable classes**
- Variadic functions/methods

Iterable Classes

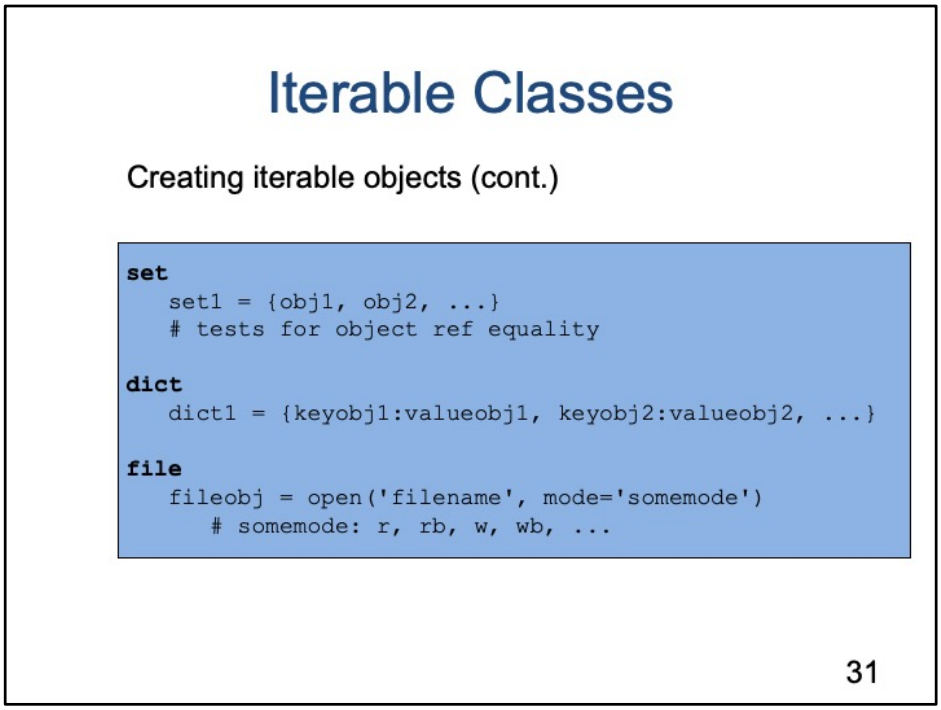
- Python iterable classes
  - `str`
    - An immutable sequence of characters
  - `bytes`
    - An immutable sequence of bytes
  - `list`
    - A sequence of object references
  - `tuple`
    - An immutable sequence of object references
  - `set`
    - A group of object references that contains no duplicates
  - `dict`
    - An associative array implemented as a hash table
  - file
    - A persistent sequence of bytes



Iterable Classes

[see slide]

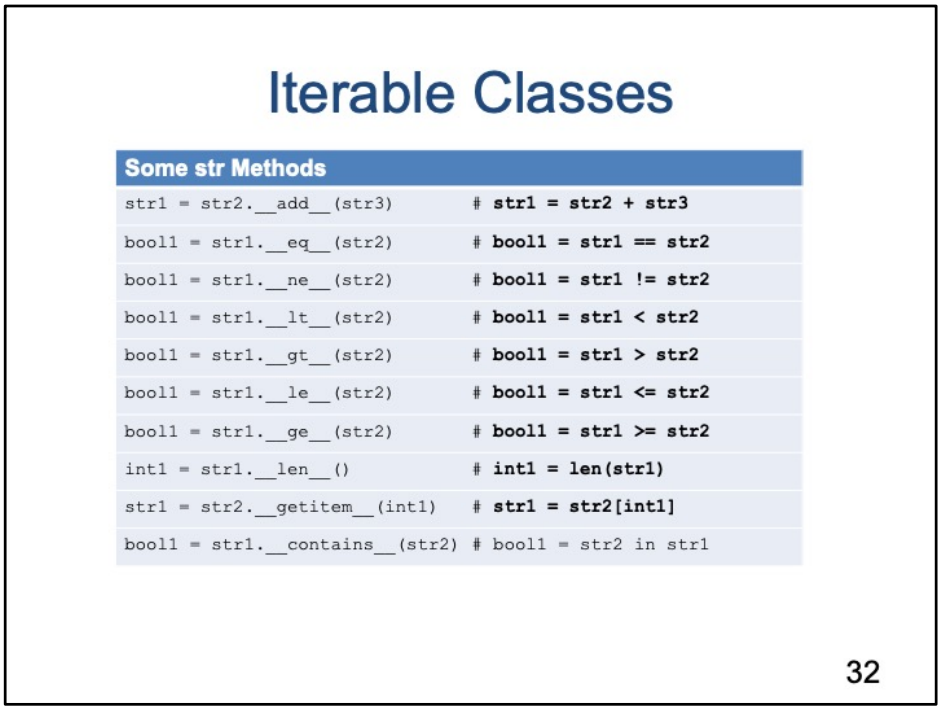
Shows the syntax for creating iterable objects



Iterable Classes

[see slide]

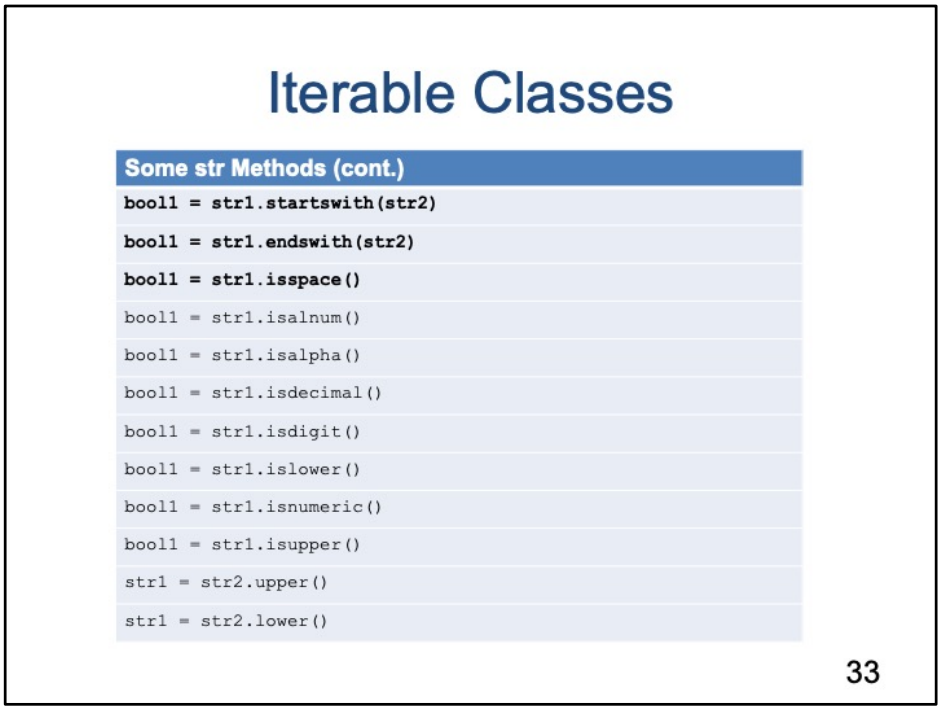
Shows the syntax for creating iterable objects



Iterable Classes

For reference

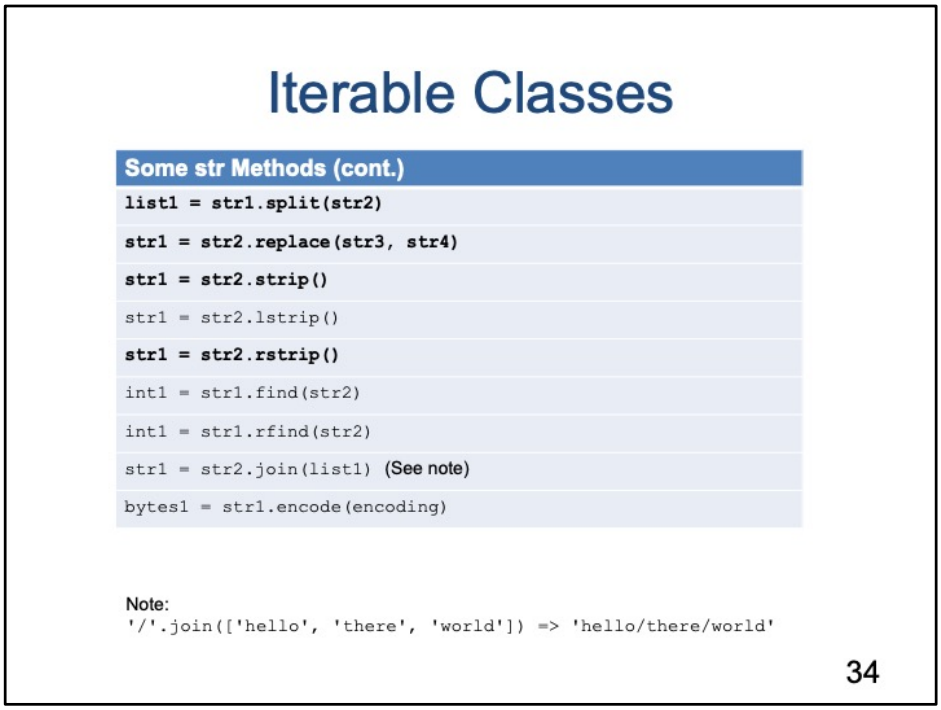
[see slide]



Iterable Classes

For reference

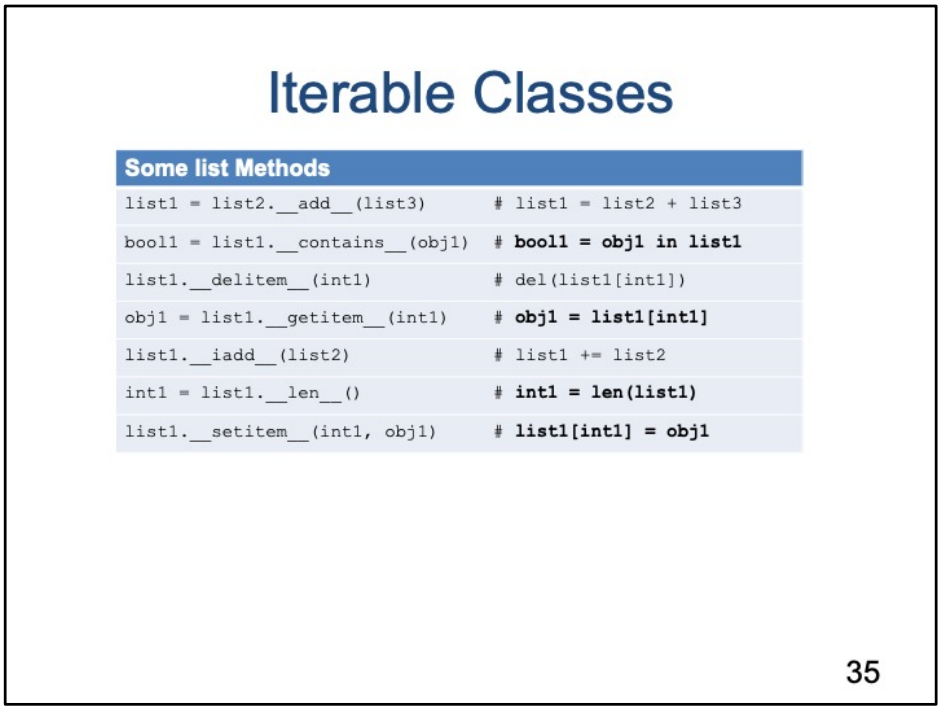
[see slide]



Iterable Classes

For reference

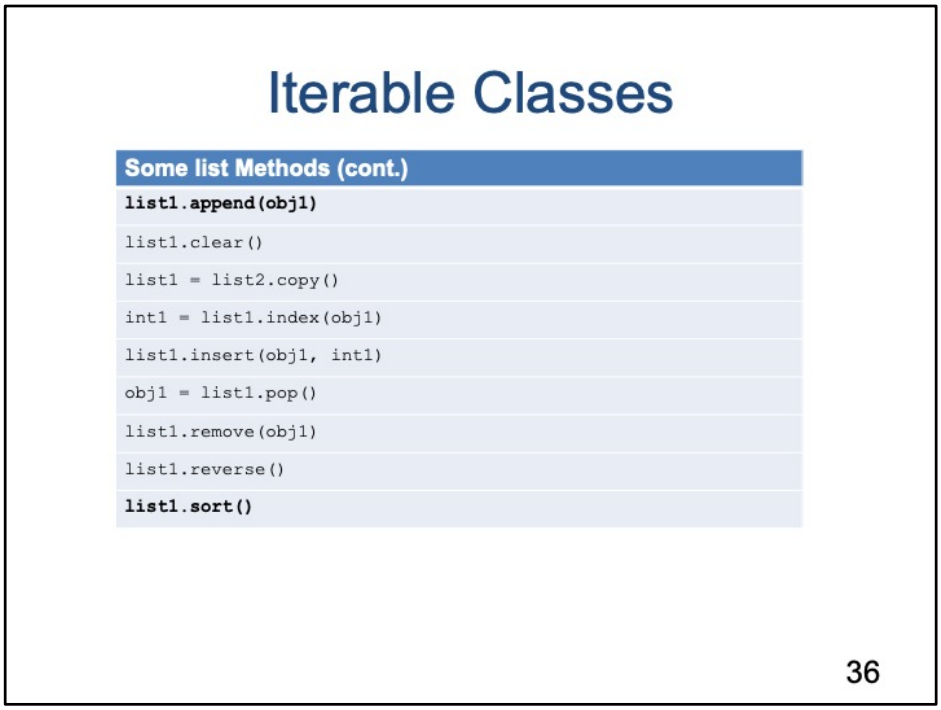
[see slide]



Iterable Classes

For reference

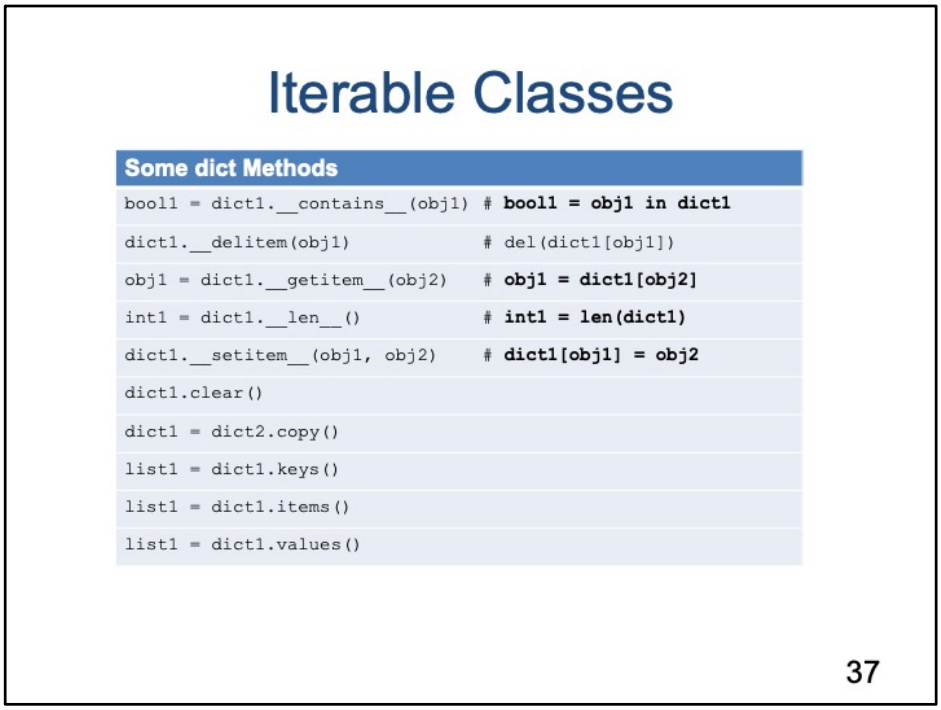
[see slide]



Iterable Classes

For reference

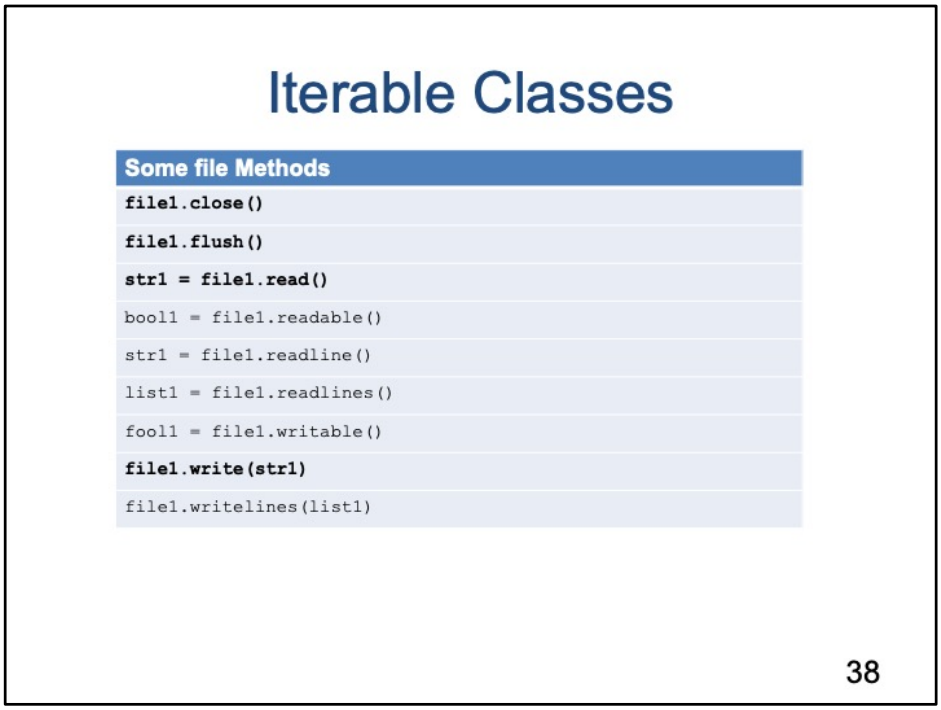
[see slide]



Iterable Classes

For reference

[see slide]



Iterable Classes

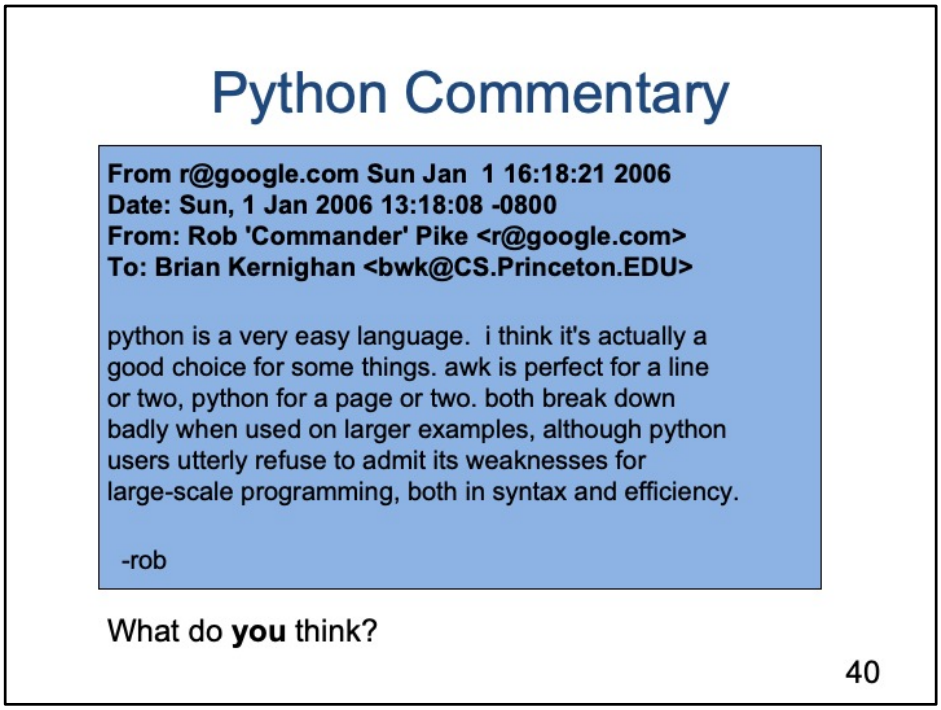
For reference

[see slide]

Iterable Classes

For more information:

```
$ python
>>> help(str)
>>> help(bytes)
>>> help(tuple)
>>> help(list)
>>> help(set)
>>> help(dict)
>>> quit()
$
```



Python Commentary

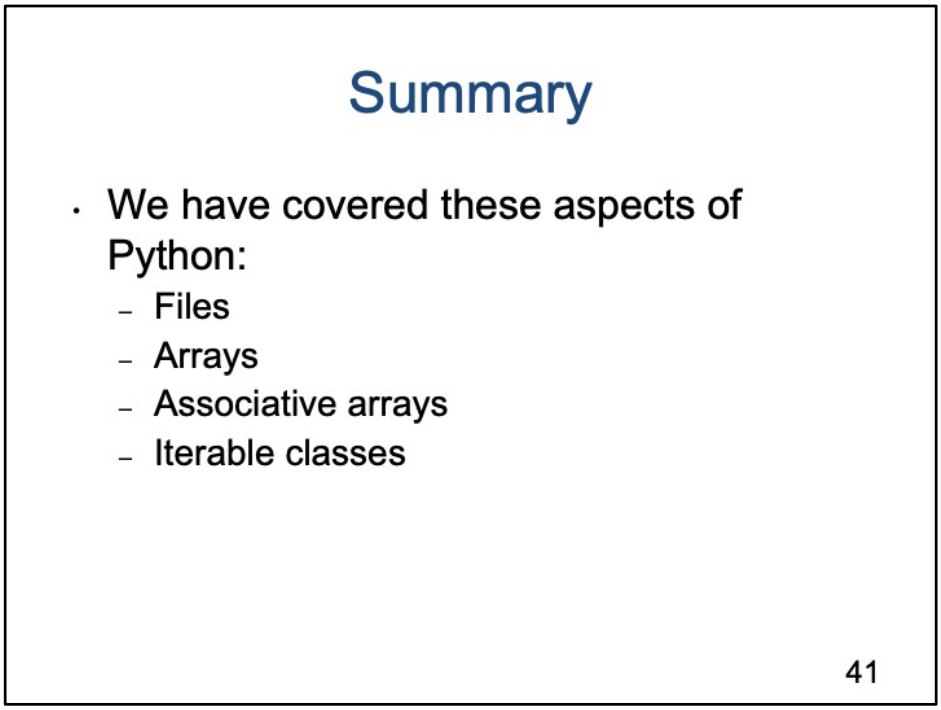
[see slide]

Could sell this on EBay!

I'm a big fan of C, Python, and Java
But I like them for different jobs!

- C: systems programming
- Python: application programming, small applications (5 programmers, a few months)
- Java: application programming, big applications (25 programmers, a few years)

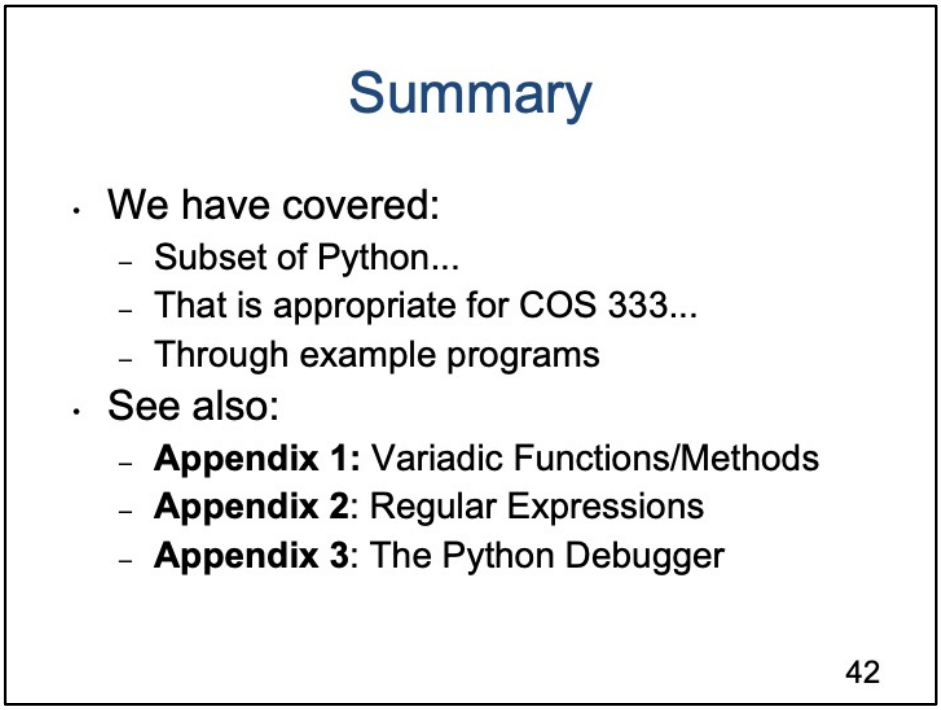
Rhetorically: What do you think?



Summary

Summary of this lecture

[see slide]



Summary

Summary of the Python sequence of lectures

[see slide]

Appendix 1:
Variadic Functions/Methods

Variadic Functions/Methods

- ***Variadic*** function/method:
  - A function/method that can be called with a variable number of arguments
  - Example: `printf()` in C
    - `printf("hello");`
    - `printf("The answer is %d", 5);`
    - `printf("The answers are %d and %d", 5, 10);`

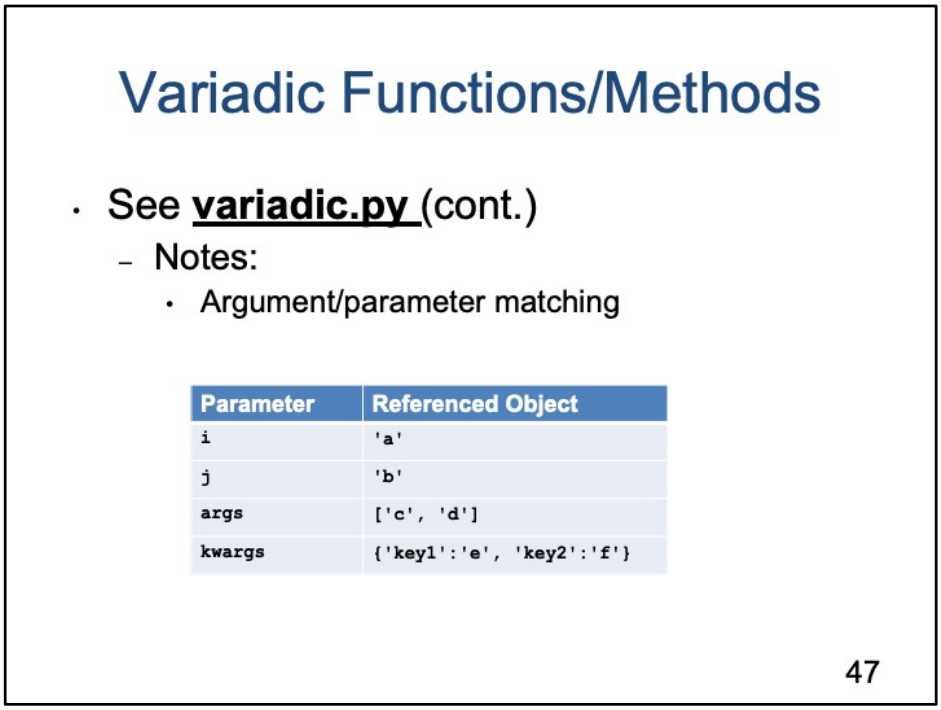
Variadic Functions/Methods

- **Question**
  - How to define variadic functions/methods in Python?
- **Answer 1**
  - Default parameter values
    - Already described
- **Answer 2**
  - `*args` and `**kwargs`...

Variadic Functions/Methods

- See **variadic.py**
  - The job:
    - Nonsensical
    - Uses `*args` and `**kwargs`

```
$ python variadic.py
a
b
c
d
key1 e
key2 f
$
```



Variadic Functions/Methods

[see slide]

The body of main() simply prints the parameter values to demonstrate that argument/parameter matching the occurred

Easy alternative: pass a list and/or a dict

But this mechanism does make the code more concise
And is used often in standard functions/methods
So it's worth knowing

Appendix 2:
Regular Expressions

Regular Expressions

- Used widely
  - Java (string manipulation)
  - Python (string manipulation)
  - Unix `grep` command (file searching)
  - Bash shell (filename wildcards)
  - SQL `like` clauses (querying databases)
    - See upcoming *Database Programming* lectures
  - …

Regular Expressions

RE	Matches
thing	thing anywhere in string
^thing	thing at beginning of string
thing$	thing at end of string
^thing$	string that contains only thing
^	any string, even empty
^$	empty string
.	non-empty, i.e. the first char in string
thing.$	thing plus any char at end of string
thing\.$	thing. at end of string
\\thing\\	\thing\ anywhere in string
[tT]hing	thing or Thing anywhere in string
thing[0-9]	thing followed by one digit
thing[^0-9]	thing followed by a non-digit
thing[0-9][^0-9]	thing followed by digit, then non-digit
thing1.*thing2	thing1 then any (or no) text then thing2
^thing1.*thing2$	thing1 at beginning and thing2 at end

Thanks to Prof. Brian Kernighan

Regular Expressions

- What do these match?
  - `a.*e.*i.*o.*u`
    - Try with `grep` command and /usr/share/dict/words file
  - `^[^aeiou]*a[^aeiou]*e[^aeiou]*i[^aeiou]*o[^aeiou]*u[^aeiou]*$`
    - Try with `grep` command and /usr/share/dict/words file

Thanks to Prof. Brian Kernighan

Regular Expressions

- Implementations vary
  - See *Mastering Regular Expressions* (Jeffrey Friedl) book
    - ~500 pages!
- In Python…

Regular Expressions

RE	Matches
X	the character X, except for metacharacters
\X	the character X, where X is a metacharacter
.	any character except \n (use DOTALL as argument to compile() to match \n too)
^	start of string
$	end of string
XY	X followed by Y
X*	zero or more cases of X (X*? is the same, but non-greedy)
X+	one or more cases of X (X+? is the same, but non-greedy)
X?	zero or one case of X (X?? is the same, but non-greedy)
[...]	any one of ...
[^...]	any character other than ...
[X-Y]	any character in the range X through Y
X|Y	X or Y
(...)	..., and indicates a group

Precedence: * + ? higher than concatenation, which is higher than |

Regular Expressions

RE	Matches
\t	tab
\v	vertical tab
\n	newline
\r	return
\f	form feed
\a	alert
\e	escape
\\	backslash
\A	empty string at start of given string
\b	empty string, but only at start or end of a word
\B	empty string, but not at start or end of a word
\d	a digit
\D	a non-digit
\s	a white space character, that is, [\t\n\r\f\v]
\S	a non-white space character
\w	an alphanumeric character, that is, [a-zA-Z0-9_]
\W	a non-alphanumeric character
\Z	the empty string at the end of the given string

Regular Expressions

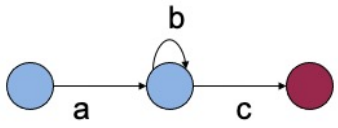
- What kinds of strings do these regular expressions match?
  - `[-+]?([0-9]+\.?[0-9]*|\.[0-9]+)([Ee][-+]?[0-9]+)?`
  - `/\*.*?\*/` (use with `DOTALL`)
    - Why the question mark?
    - Why `DOTALL`?
- Commentary: Regular expressions are write-only!!!

Regular Expressions

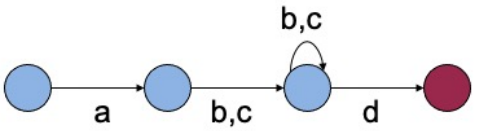
- Some theory:
  - Regular expressions have the same power as ***deterministic finite state automata*** (DFAs)
  - A regular expression defines a ***regular language***
  - A DFA also defines a regular language

Regular Expressions

ab*c



a[bc]+d



Appendix 3:
The Python Debugger

The Python Debugger

- `pdb` debugger is bundled with Python
- To use `pdb`:

```
$ python -m pdb somefile.py
```

The Python Debugger

Some `pdb` Commands

- `help`
- `break` *`functionOrMethod`*
- `break` *`filename:linenum`*
- `run`
- `list`
- `next`
- `step`
- `continue`
- `print` *`expr`*
- `where`
- `quit`

Note similarities with `gdb`

The Python Debugger

- Common commands have abbreviations: h, b, r, l, n, s, c, p, w, q
- Blank line means repeat the same command
- Beware: Cannot easily read from stdin