11 Text & Files

Key terms: `with open read write close bytes`

Reading: Severance 7

Exercise: Write a program that creates a file with a randomly chosen filename.

A file is information recorded sequentially in some kind of persistent storage. These are organized into file systems using addresses called pathnames, or paths. Absolute pathnames begin at the filesystem root, so in UNIX, they start with a slash: `/proc/version`. Relative pathnames begin at some arbitrary place in the hierarchy, making them inherently ambiguous: `letter.txt`.

A file wrapper object handles reading and writing of files on disk; here we look at reading. Its method `read()` returns the contents of the file as a string:

```
# Store the contents of a file as a single string:
text = open("/proc/version").read()
```

Alternately, the wrapper object is iterable, gracefully handling file content line by line:

```
# Iterate over the lines of a file.
for line in open("/etc/passwd"):
 print(line.split(':')[0])
```

Permissions rules administered by the operating system control access to files; the methods for reading and writing permissions modes are in the `os` module. File access only reads by default, otherwise writing or appending based on a mode parameter. Since several sorts of things can go wrong with file access, it is wise to run robust exception tests to check for various errors:

```
# Try to open the file specified:
import sys
try:
 file = open(sys.argv[1])
except IndexError:
 print ('No filename passed.')
except FileNotFoundError:
 print ('File not found.')
except PermissionError:
 print ('File not readable.')
```

Rarely is it wise to read a file more than once; in the event we want continual access, we might prefer to keep parts of it in memory (for example, as a string).

It is convenient to use the `with` keyword for a "context manager" that closes the file for us.

Python's default encoding is UTF-8: strings are stored with Unicode codepoints, represented in output using the UTF-8 scheme. Python 3 is righteously strict about encodings, which prevents confusion between strings and encoded streams (`bytes`). This change enables Python code to handle all scripts with no fuss. The above examples will raise a `UnicodeDecodeError` if the input file is not valid UTF-8. This behavior is the alternative to the ostrich-style decoding you may have seen in web pages and emails seemingly containing gibberish.

```
# Try to read UTF-8 text from an open file:
try:
 text = file.read()
except UnicodeDecodeError:
 print ('File is not encoded in UTF-8.')
```

Next, we will address functions.