File IO

Introduction to File Input and Output

- For programs to retain data between runs, data must be saved
  - Data is saved to a “file”
  - Saved data can be retrieved and used at a later time

Types of Files and File Access Methods

- In general, two types of files
  - Text file: contains data that has been encoded as text
  - Binary file: contains arbitrary binary data
- Two ways to access data stored in file
  - Sequential access: file read sequentially from beginning to end, can't skip ahead
    - Most input files are this type
  - Direct access: can jump directly to any piece of data in the file

Filenames and File Objects

- File Paths: A string that specifies a specific file on (a) disk
  - Absolute: Gives a specific path from the "root" of the file system
  - Relative: begins relative to a specific directory

- Ex:
  - /home/bill/apps/mydata.txt - Absolute
  - mydata.txt - Relative
  - ./mydata.txt - Relative

Filenames and File Objects

- Filename: A sequence of characters that give the file a unique name within a directory

- File extensions: short sequences of characters that appear at the end of a filename preceded by a period
  - Extension provides a hint of what type of data is in the file
    - .txt
    - .docx
    - .py

Extensions

- Extension provides a ***hint*** of what type of data is in the file

Extensions

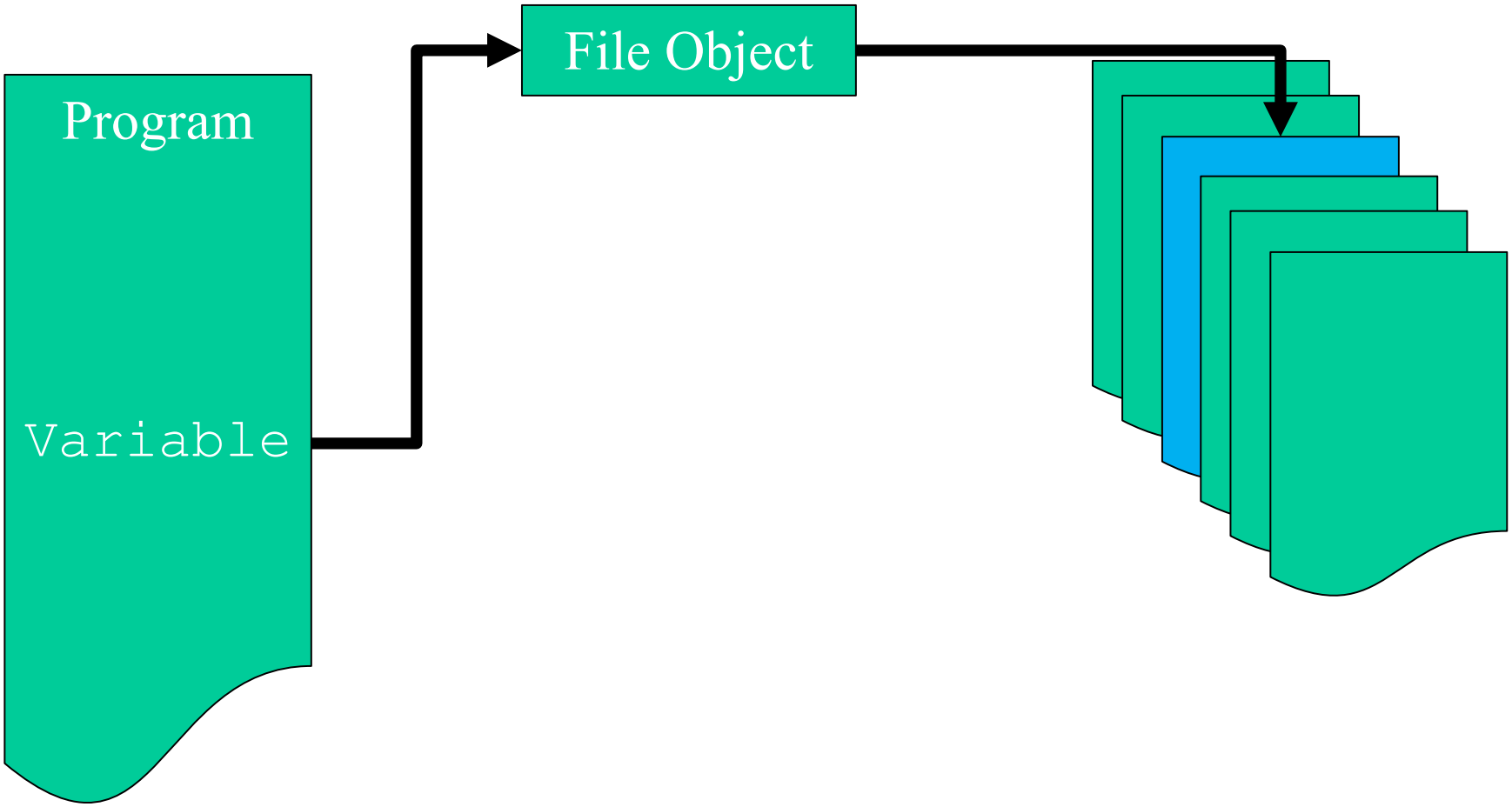
- Extension provides a ***hint*** of what type of data is in the file

- Extension ≠ file contents

Filenames and File Objects

- File object: object associated with a specific file
  - Provides a way for a program to work with the file
  - Stores metadata about the file
    - File path
    - Position
    - Mode
    - Etc.

Filenames and File Objects (cont'd.)



Opening a File

- `open` function: used to open a file
  - Creates a file object and associates it with a file on the disk
  - General format:

    file_object = open(*filename*, *mode*)
- Mode: string specifying how the file will be opened
  - Example: reading only (`'r'`), writing (`'w'`), and appending (`'a'`)

More About Mode

```
file_object = open(filename, mode)
```

- `r- Reading`
  - If the file exists, point to the start of the file
- `w- Writing`
  - If the file exists, delete it and create an empty file of the same name
  - Else, create the file
- `a- Appending`
  - If the file exists, point to the end of the file
  - Else, create it

Specifying the Location of a File

- If `open` function receives a filename that does not contain a path, assumes that file is in same directory as program
- If a file is created without a path, it is created in the same directory as the program

- Can specify alternative path and file name in the `open` function argument

Writing Data to a File

- `write`: file object method for writing data to a file
  - Format: *`file_variable`*`.write(`*`string`*`)`
  - Only accepts strings
    - Other datatypes must be casted into strings
  - Will only write the data given to it
    - Will not add spaces, newlines, or other characters automatically
- File should be closed using file object `close` method
  - Format: *`file_variable`*`.close()`

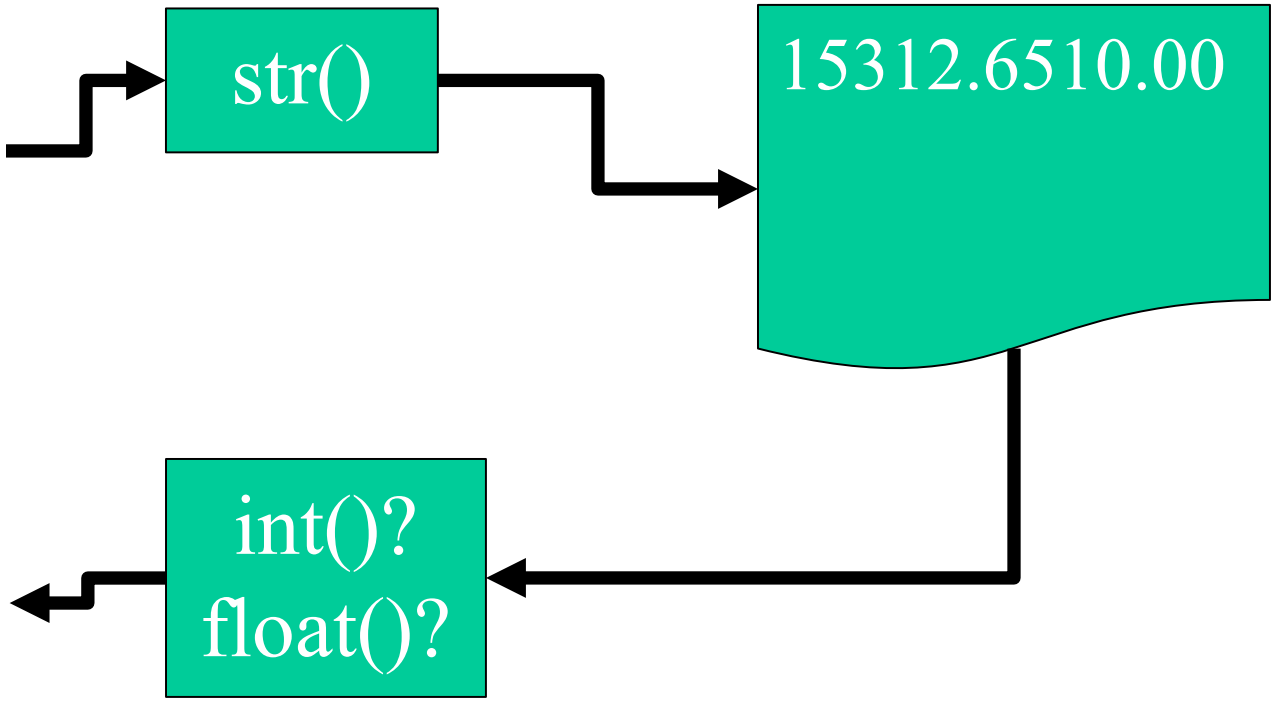
Reading Data From a File

- `read`: file object method that reads entire file contents into memory
  - Only works if file has been opened for reading
  - Contents returned as a string
- `readline`: file object method that reads a line from the file
  - Line returned as a string, including `'\n'`
- Read position: marks the location of the next item to be read from a file
  - Handled internally by the file object

Exercise 10A

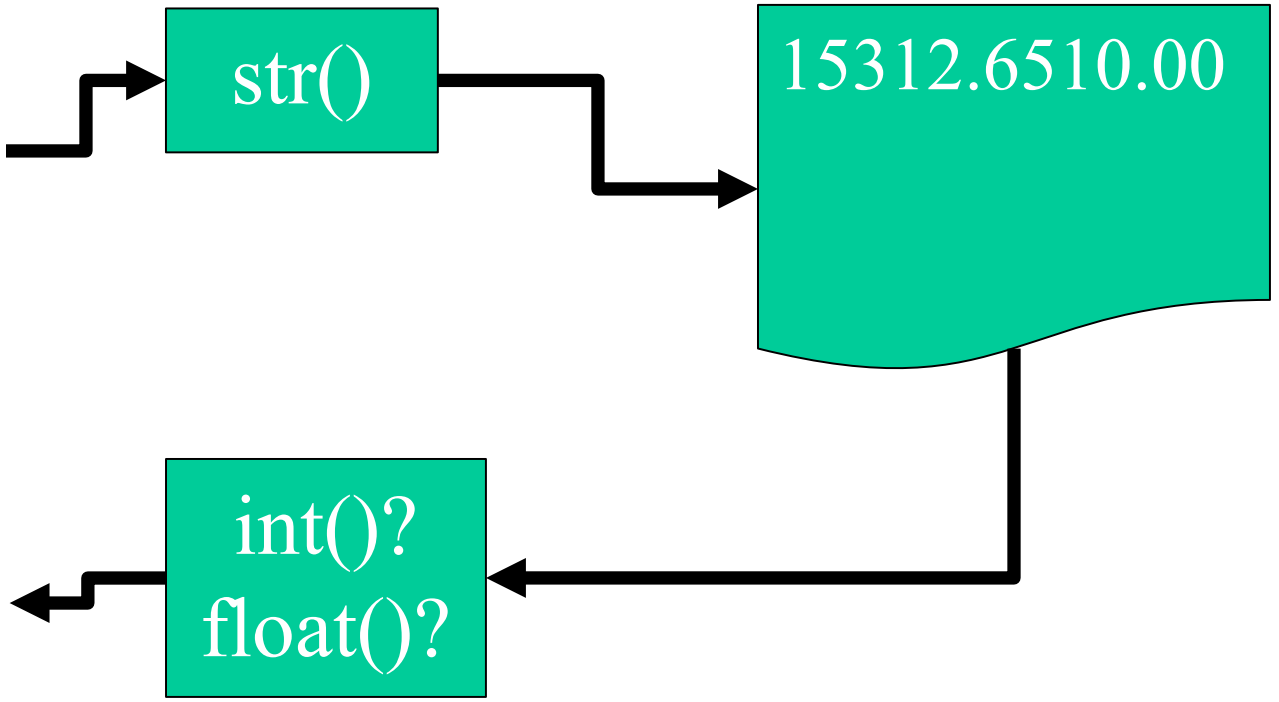
- Write two programs to operate on variables a, b, and c. The first program will write all three variables to a file, the second will read the file and place the data back into the three respective variables.

```
a = 153
b = 12.65
c = 10.00
```



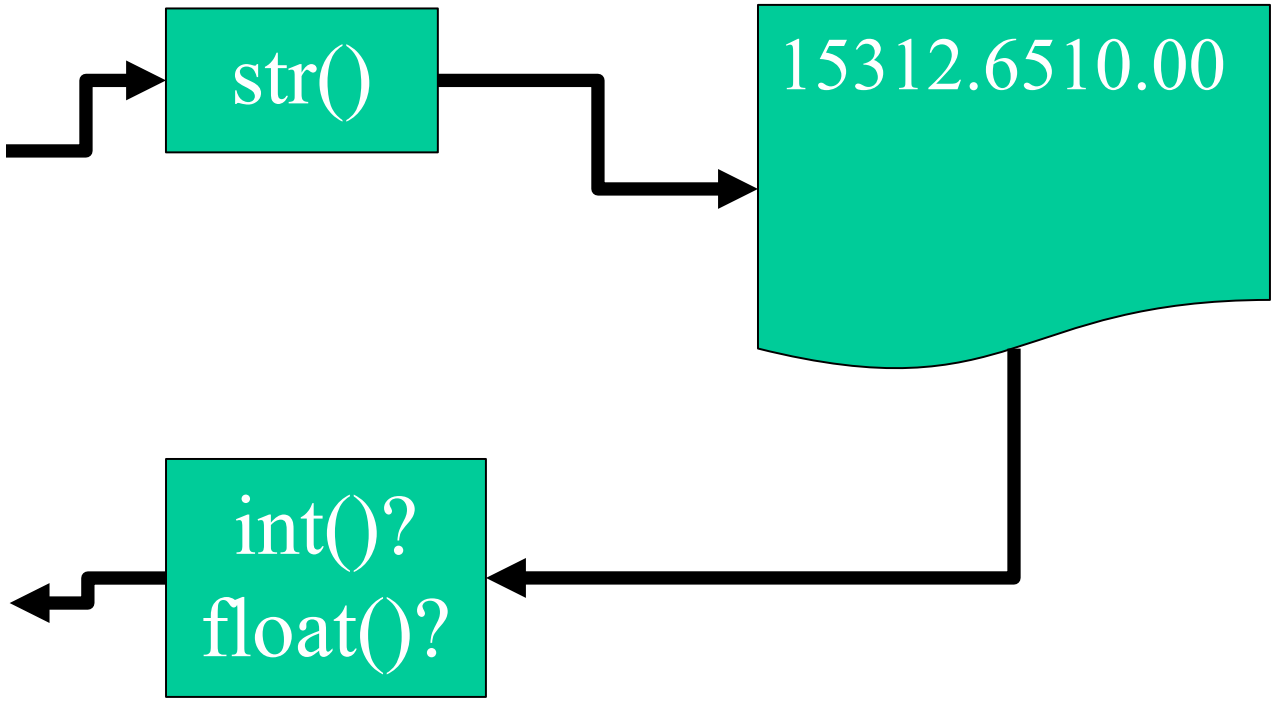

```
a = ???
b =
c =
```

```
a = 153
b = 12.65
c = 10.00
```



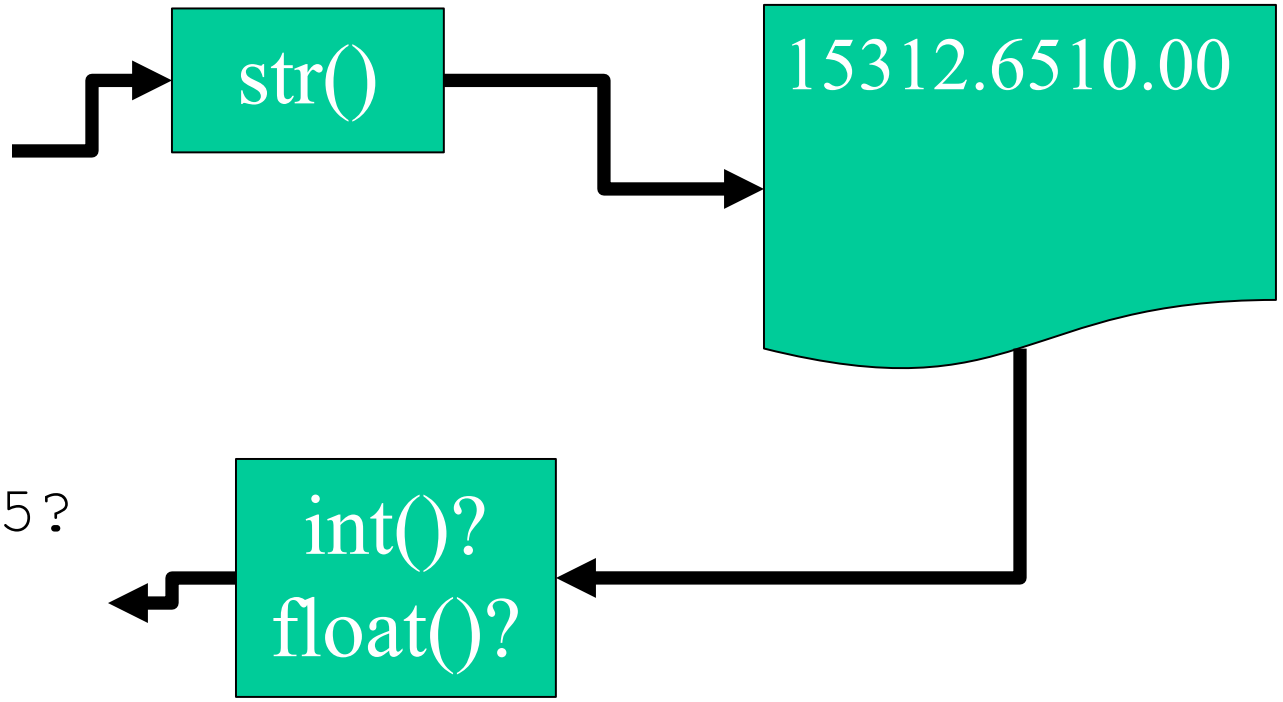

```
a = 1?
b =
c =
```

```
a = 153
b = 12.65
c = 10.00
```




```
a = 15312?
b =
c =
```

```
a = 153
b = 12.65
c = 10.00
```




```
a = 15312.65?
b =
c =
```

Parsing Data

- In most cases, data items need to be separated in a clean, obvious way.
- Also, aggregate data needs to be separated
  - Consider a “person” (ID, Name, Age)

- Record: A collection of data items that describes an entity
- Field: An individual data item within a record

- Records must be delimited
  - Fields must *also* be delimited

Fixed Width Fields

- Some formats give each field a fixed number of characters
  - Unused characters are ignored

- Ex.
  - ID: 4 chars
  - Name 10 chars
  - Age 3 chars

```
IIIINNNNNNNNNNAAA
0001WILLIAM---028
```

Variable Width Fields

- Delimiter: a sequence or pattern of characters that separates two distinct values
- Allows a field to be any size
  - Instead of counting, the parser simply watches for the delimiter

- Ex.
  - Record Delimiter: \n
  - Field Delimiter: ,

```
1,William,28
11432343,Ravenscroft,26
```

Which Delimiter to Use?

- Delimiters can be anything.
  - Literally
  - Any character, sequence of characters
  - Up to the programmer

  - Just beware, what happens if your delimiter appears within the data you are trying to store?

Conventional Delimiters

- Records
  - `\n` – Escape code for a new line
- Fields
  - `\t` – Escape code for tab
  - `,` – Used in Comma Separated Value (CSV) files