Lecture Roadmap

- Intro to Programming
- Variables
- Functions
  - The stack
  - Scope
- Flow control
  - `if`
  - `while`
  - `for`
- Strings
- Lists
- Tuples
- Dictionaries

Tuples

COMP1730/COMP6730

Reading: Textbook chapter 12 : Alex Downey, *Think Python*, 2nd Edition (2016)

Sections: *Tuples are immutable, Tuple assignment, Tuples as return values*

Lists versus Tuples

- Both a sequences.
- Lists are mutable. Tuples are immutable. Otherwise, they are very similar.
- There are good reasons for using tuples in certain circumstances:
  - Performance – if a list won't change, the python interpreter can make optimisations
  - Hands off – sometimes it is better to not be able to change (or have something else change) the values in your sequence.

Tuples?

- Tuples are immutable. So, think of them like lists that can't be changed.
- A comma-separated sequence of values (with or without parentheses):

```
>>> t = 'a', 'b', 'c', 'd', 'e'
```

- Create with a trailing comma:

```
>>> t1 = 'a',
>>> type(t1)
<class 'tuple'>
```

- Or with the the `tuple()` function:

```
>>> t = tuple('lupins')
>>> t
('l', 'u', 'p', 'i', 'n', 's')
```

Tuples work mostly like lists

- Elements in a tuple can be accessed by indexes:

```
>>> t = ('a', 'b', 'c', 'd', 'e')
>>> t[0]
'a'
```

- And slices can be made from tuples:

```
>>> t[1:3]
('b', 'c')
```

- But they can't be changed:

```
>>> t[0] = 'A'
TypeError: object doesn't support item assignment
```

Why, tuples?

- They make excellent `return` values from a function
- And are good protection from unintended side-effects of functions on your data structures

Exercises

- Only if you want – try a few at the end of *Think Python* Ch. 11

Reading

- *Think Python* Ch 11

Files and IO

COMP1730/COMP6730

Reading: Ch 14 : Alex Downey (2016) *Think Python*, 2nd Ed
(sections: *Persistence, Reading and Writing, Filenames and Paths, Pickling*)
OR
Ch 9: Sweigart (2019) *Automate the boring stuff with python*
(sections: *Files and File Paths, File Reading/Writing Process*)

Persistence (*Think Python*, Ch14)

Australian National University

- When your program is executed, it has no memory of any previous time it may have been run. And nothing in memory will survive after the program exits.
- **Persistence** is the concept of retaining this information or memory *between* program execution instances
- This is commonly done by storing input and output files on disk
- Also in databases (which are the subject of semester-long courses by themselves)
- And with python, can use `pickle` to create dumps of program memory that can be reread at another time

- But, importantly, reading files into your program **provides access to data**

Files and writing programs

Australian National University

- Why would you need to read or write to a file with your program?
  - Files are a very simple kind of **persistent storage**
  - Read in data – write out data after performing some computation
  - Files may contain configuration information
  - Much of data science involves looking at datasets contained in files

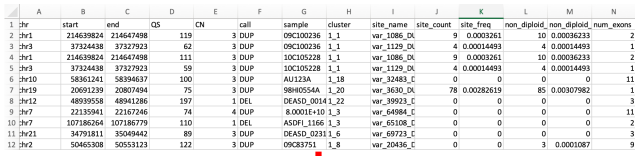
The human genome is routinely stored like this, in FASTA files.
Here is the beginning Chromosome 1:

```
>Chr1
TGCTGTCAAGACTTTAAATAGATACAGACAGAGCATTTTCACTTTTTCCT
ACATCTCTATTATTCTAAAAATGAGAACATTCCAAAAGTCAACCATCCAA
GTTTATTCTAAATAGATGTGTAGAAATAACAGTTGTTTCACAGGAGACTA
ATCGCCCAAGGATATGTGTTTAGAGGTACTGGTTTCTTAAATAAGGTTTT
CTAGTCAGGCAAAAGATTCCCTGGAGCTTATGCATCTGTGGTTGATATTT
TGGGATAAGAATAAAGCTAGAAATGGTGAGGCATATTCAATTTCATTGAA
GATTTCTGCATTCAAAATAAAAACTCTATTGAAGTTACACATACTTTTTT
CATGTATTTGTTTCTACTGCTTTGTAAATTATAACAGCTCAATTAAGAGA
AACCGTACCTATGCTATTTTGTCCTGTGATTCTCCAAGAACCTTCCTAAG
TTATTCTACTTAATTGCTTTATCACTCATATGAATGGGAATTTCTTCTCT
TAATTGCTGCTAATctccccccatcttcaaatactctaccgggcttctgga
acaccacagcttcctggctttttctcctacctcctgggcaagtccttccc
tgtgtcttttgttgagtgttcctcatctgcttaactaccaatcaacctat
tgcccctaatttgatctttggcctgttttcacttagattctatccctacg
tatcacccattcccacagctttaatcaccatctaaacactaggggctctc
```

Comma-separated values (CSV) files

Australian National University

- A very common data file type is the comma-separated-values and tab-separated-values format.
- Think of these as spreadsheet data files, where the columns are separated by either a **comma** or a **tab**:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	chr	start	end	QS	CN	call	sample	cluster	site_name	site_count	site_freq	non_diploid_	non_diploid_	num_exons
2	chr1	214639824	214647498	119	3	DUP	09C100236	1_1	var_1086_DU	9	0.0003261	10	0.00036233	2
3	chr3	37324438	37327923	62	3	DUP	09C100236	1_1	var_1129_DU	4	0.00014493	4	0.00014493	1
4	chr1	214639824	214647498	111	3	DUP	10C105228	1_1	var_1086_DU	9	0.0003261	10	0.00036233	2
5	chr3	37324438	37327923	59	3	DUP	10C105228	1_1	var_1129_DU	4	0.00014493	4	0.00014493	1
6	chr10	58361241	58394637	100	3	DUP	AU123A	1_18	var_32483_D	0	0	0	0	11
7	chr19	20691239	20807494	75	3	DUP	98HI0554A	1_20	var_3630_DU	78	0.00282619	85	0.00307982	1
8	chr12	48939558	48941286	197	1	DEL	DEASD_0014	1_22	var_39923_D	0	0	0	0	3
9	chr7	22135941	22167246	74	4	DUP	8.0001E+10	1_3	var_64984_D	0	0	0	0	11
10	chr7	107186264	107186779	110	1	DEL	ASDFI_1166	1_3	var_65108_D	0	0	0	0	2
11	chr21	34791811	35049442	89	3	DUP	DEASD_0231	1_6	var_69723_D	0	0	0	0	3
12	chr2	50465308	50553123	122	3	DUP	09C83751	1_8	var_20436_D	0	0	3	0.0001087	9

```
chr,start,end,QS,CN,call,sample,cluster,site_name,site_count,site_freq,non_diploid_count,non_diploid_freq,num_exons
chr1,214639824,214647498,119,3,DUP,09C100236,1_1,var_1086_DUP,9,0.000326099,10,0.000362332,2
chr3,37324438,37327923,62,3,DUP,09C100236,1_1,var_1129_DUP,4,0.000144933,4,0.000144933,1
chr1,214639824,214647498,111,3,DUP,10C105228,1_1,var_1086_DUP,9,0.000326099,10,0.000362332,2
chr3,37324438,37327923,59,3,DUP,10C105228,1_1,var_1129_DUP,4,0.000144933,4,0.000144933,1
chr10,58361241,58394637,100,3,DUP,AU123A,1_18,var_32483_DUP,0,0,0,0,11
chr19,20691239,20807494,75,3,DUP,98HI0554A,1_20,var_3630_DUP,78,0.002826189,85,0.003079822,1
chr12,48939558,48941286,197,1,DEL,DEASD_0014_001,1_22,var_39923_DEL,0,0,0,0,3
```

An example – Variant Call Format

Australian National University

- More complicated example of file storage of data.
- This is a Variant Call Format file – for storing the genetic variation information identified from a personal genome sequence
- Not-quite human readable, but the industry standard.
- Every industry has its' own standards – probably mostly text format, though some more sophisticated

```
##fileformat=VCFv4.3
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=file:///seq/references/1000GenomesPilot-NCBI36.fasta
##contig=
<ID=20,length=62435964,assembly=B36,md5=f126cdf8a6e0c7f379d618ff66beb2da,species
="Homo sapiens",taxonomy=x>
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=A,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS     ID        REF    ALT    QUAL  FILTER  INFO
FORMAT      NA00001        NA00002        NA00003
20     14370    rs6054257  G     A      29    PASS    NS=3;DP=14;AF=0.5;DB;H2
GT:GQ:DP:HQ  0|0:48:1:51,51  1|0:48:8:51,51  1/1:43:5:.,.
20     17330    .          T     A      3     q10     NS=3;DP=11;AF=0.017
GT:GQ:DP:HQ  0|0:49:3:58,50  0|1:3:5:65,3    0/0:41:3
20     1110696  rs6040355  A     G,T    67    PASS    NS=2;DP=10;AF=0.333,0.667;
AA=T;DB GT:GQ:DP:HQ  1|2:21:6:23,27  2|1:2:0:18,2    2/2:35:4
20     1230237  .          T     .      47    PASS    NS=3;DP=13;AA=T
GT:GQ:DP:HQ  0|0:54:7:56,60  0|0:48:4:51,51  0/0:61:2
20     1234567  microsat1  GTC   G,GTCT 50    PASS    NS=3;DP=9;AA=G
GT:GQ:DP     0/1:35:4        0/2:17:2        1/1:40:3
```

What is a file?

Australian National University

- A file is a collection of data on secondary storage (hard drive, USB key, network file server)
- A program can open a file to read/write data
- The data in a file is a sequence of bytes (integer values 0 to 255):
  - A program reading a file must interpret the data (as text, image, sound, etc)
  - Python and the operating system (OS) provide support for interpreting the data as text
- Text vs Binary files:
  - A **text file** contains printable characters (including numbers, spaces, newlines, etc)
  - A **binary file** contains arbitrary data which may not correspond to printable characters. May not be viewed is a simple text editor.

Anatomy of a text file

Australian National University

- Characters are commonly encoded as 'ASCII text':

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

- Lines in a text file commonly end with a **newline** (`\n`) character
- **Non-printing** characters include tabs (`\t`), spaces (`\s`) and other **escape characters**

```
chr,start,end,QS,CN,call,sample,cluster,site_name,site_count,site_freq,non_diploid_count,non_diploid_freq,num_exons
chr1,214639824,214647498,119,3,DUP,09C100236,1_1,var_1086_DUP,9,0.000326099,10,0.000362332,2
chr3,37324438,37327923,62,3,DUP,09C100236,1_1,var_1129_DUP,4,0.000144933,4,0.000144933,1
chr1,214639824,214647498,111,3,DUP,10C105228,1_1,var_1086_DUP,9,0.000326099,10,0.000362332,2
chr3,37324438,37327923,59,3,DUP,10C105228,1_1,var_1129_DUP,4,0.000144933,4,0.000144933,1
chr10,58361241,58394637,100,3,DUP,AU123A,1_18,var_32483_DUP,0,0,0,0,11
chr19,20691239,20807494,75,3,DUP,98HI0554A,1_20,var_3630_DUP,78,0.002826189,85,0.003079822,1
chr12,48939558,48941286,197,1,DEL,DEASD_0014_001,1_22,var_39923_DEL,0,0,0,0,3
```

Invisible newline characters at end of each line

Anatomy of a binary file

Australian National University

- Binary files can contain anything that the developer of the specific binary file designed.
- If you open a binary file as if it were a text file, you might see this:

Figure 9-6: The Windows `calc.exe` program opened in Notepad

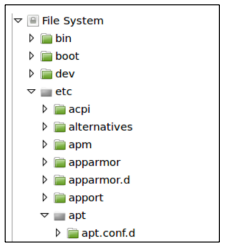
Sweigart (2019) *Automate the boring stuff with Python*, Ch. 9.

Files and directories:

Australian National University

- Files on secondary storage are organized into directories (aka folders)
- This is an abstraction provided by the operating system
- The directory structure is typically tree-like
- File locations can be represented in text form by a file path:

```
/Users/dan/Desktop/Gray_etal_SupplementaryTable_S2_cleaned.csv
```



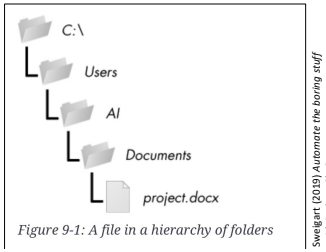


Figure 9-1: A file in a hierarchy of folders

Sweigart (2019) *Automate the boring stuff with Python*, Ch. 9.

Where are you (in the filesystem)?

Australian National University

- In your code, you aren't able to point-and-click your way around.
- Find out the directory 'path':

```
>>> import os
>>> cwd = os.getcwd()
>>> cwd
'/home/dinsdale'
```

- And you can list the files in the directory with:

```
>>> os.listdir(cwd)
['music', 'photos', 'memo.txt']
```

Downey (2015) Think Python, 2nd Ed.

The file 'path':

Australian National University

- A **path** is a string that identifies the location of a file in the directory structure
- Consists of the hierarchical directory names in sequence, with a separator between each (the forward-slash `/`)
- You will see two kinds of paths:
  - **Full** or **absolute** (from the top-level directory)
  - **Relative** (to the current working directory)
- When running a python file (script mode), the current working directory (cwd) is the directory that is was started/executed from
- If the python interpreter was started in interactive mode (iPython or the console), the cwd is the directory that it was started from
- The `os` module has functions to get (and change) the current working directory:

```
>>> import os
>>> os.getcwd()
'/home/patrik/teaching/python'
```

`open()` and `close()` file syntax:

Australian National University

- To open a file, use `open(filename, mode)`
  - The file open modes can be:
    - `r` : read
    - `w` : write
    - `x` : write, but only if the file doesn't already exist
    - `a` : append, by writing after the last line of the existing file
- To close a file, use `close()`

Writing to a file

Australian National University

- To write to a file, first it needs to be opened (in write mode):

```
>>> fout = open('output.txt', 'w')
```

- `fout` is an object that allows you to access this open file
- With the `fout`, you may then write to the file:

```
>>> line1 = "This here's the wattle,\n"
>>> fout.write(line1)
```

```
>>> line2 = "the emblem of our land.\n"
>>> fout.write(line2)
```

- Then, it is a good habit to remember to close the file*:

```
>>> fout.close()
```

Downey (2015) Think Python, 2nd Ed.

Read a file (*Think Python*, Ch 9)

Australian National University

- Use the `open()` command again, but not in write mode:

```
>>>
>>> fin = open('output.txt', 'r')
>>>
>>> fin.readline()
"This here's the wattle,\n"
>>> fin.readline()
'the emblem of our land.\n'
>>>
>>> fin.close()
>>> _
```

- Use `readline()` method to get the next line from the file.
  - Note that each line returned is a string – and has a newline at the end
- Then `close()` the file. You can't read the file once it is closed

File objects

Australian National University

- When we open a file, python creates a file object (or, more abstractly, a **stream** object)
  - The file object is our interface to the file: all reading and writing is done through methods of this object
  - The type of file object (and what we can do with it) depends on the access mode specified when the file was opened (ie. read-only, write-only, append-only)

```
>>>
>>> fin = open('/Users/dan/Downloads/example.csv', 'r')
>>>
>>> type(fin)
<class '_io.TextIOWrapper'>
>>>
```

File Objects

Australian National University

- What are these objects and clas
  interaction with a file?
- This file object is

`<class '_io.TextIOWrapper'>`

- Reach for documentation.

`https://docs.python.org/3/library/io.html`

- Searched for `TextIOWrapper`
- The documentation says that `TextIOWrapper` inherits from `TextIOBase`
- `TextIOBase` is the class with the familiar `readline()` and `write()` methods

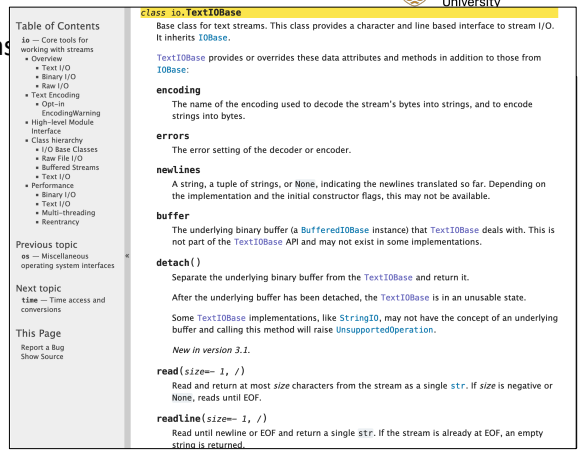


Table of Contents

io — Core tools for working with streams
- Overview
  - Text I/O
  - Binary I/O
  - Raw I/O
- Text Encoding
  - Opt-in EncodingWarning
- High-level Module Interface
- Class hierarchy
  - I/O Base Classes
  - Raw File I/O
  - Buffered Streams
  - Text I/O
- Performance
  - Binary I/O
  - Text I/O
  - Multi-threading
  - Reentrancy

Previous topic

os — Miscellaneous operating system interfaces

Next topic

time — Time access and conversions

This Page

Report a Bug
Show Source

class io.**TextIOBase**

Base class for text streams. This class provides a character and line based interface to stream I/O. It inherits IOBase.

TextIOBase provides or overrides these data attributes and methods in addition to those from IOBase:

encoding

The name of the encoding used to decode the stream's bytes into strings, and to encode strings into bytes.

errors

The error setting of the decoder or encoder.

newlines

A string, a tuple of strings, or None, indicating the newlines translated so far. Depending on the implementation and the initial constructor flags, this may not be available.

buffer

The underlying binary buffer (a BufferedIOBase instance) that TextIOBase deals with. This is not part of the TextIOBase API and may not exist in some implementations.

detach()

Separate the underlying binary buffer from the TextIOBase and return it.

After the underlying buffer has been detached, the TextIOBase is in an unusable state.

Some TextIOBase implementations, like StringIO, may not have the concept of an underlying buffer and calling this method will raise UnsupportedOperation.

New in version 3.1.

read(*size=- 1, /*)

Read and return at most *size* characters from the stream as a single str. If *size* is negative or None, reads until EOF.

readline(*size=- 1, /*)

Read until newline or EOF and return a single str. If the stream is already at EOF, an empty string is returned.

File objects are *iterable*

Australian National University

- **Iterable** objects are those that a `for` loop can work with
- The file stream objects created with `open('filename', 'r')` are iterable
- For example, can list the contents of a file with this:

```
csv_file = '/Users/dan/Desktop/example.csv'

fin = open(csv_file, 'r')

for line in fin:
    print(line, end='')

fin.close()
```

File position

- A file is a sequence of bytes
  - though the file object is not a sequence
- The file object does keep track of where in the file it is reading from or writing to
  - The next read operation (or iteration) starts from the current position
- When a file is open for reading (mode 'r') the starting position is 0 (the beginning of the file)
- The file position does not correspond to the line number

File position with `tell()` and `seek()`

- You can programmatically find the present position in the file with `tell()`. This will return the present position in a the file start):

```
csv_file = '/Users/dan/Desktop/example.csv'

with open(csv_file, 'r') as fin:
    line = fin.readline()
    while line:
        print(fin.tell())
        line = fin.readline()
```

```
117
211
301
395
485
557
651
730
804
879
956
1033
>>>
```

- `seek()` can be used to change the position in the file.
- When a file has been iterated through, the way to go back to the beginning is to use `seek(0)`

File Buffering

- File objects typically have an I/O buffer
  - Constant access to the disc can be slow and buffering this activity makes sense
  - Writing to the file object adds data to the buffer
  - When buffer is full, all data in the buffer is written to the file ('flushing' the buffer)
- Closing the file flushes the buffer
  - If the program stops without closing (with a `close()`), the buffer may not have been flushed and written to file.
  - So you might end up with missing text
  - Always close the file when finished an `open()`

`with`

- The `with` statement can simplify closing files and is recommended in modern python - though it is not mentioned in any of our books(!)
- But is a useful shorthand that you may see in code that you read.
- `with` syntax:

```
with open(filename, mode) as file_obj_name:
    line = file_obj_name.readline()
    print(line)
```

- Note that the absence of the `close()`
- It *just works*

Checking a file `exists()`

Australian National University

- Before trying to open a file, it is always good to check it exists
- You can go:

```
>>>
>>> import os
>>> os.path.exists('output.txt')
True
>>>
```

- This may save you from an error message – and you could gracefully print a message that the file wasn't found.

Caution – file over-writing

Australian National University

- When using write mode `('w')`:
  - There will be no pop-up message if you are about to overwrite an existing file
  - Inadvertent over-writing or 'clobbering' your file (https://en.wikipedia.org/wiki/Clobbering)
  - The file will be gone
- Can we check if an existing file will be over-written? Yes
  - With `os.path.exists(filepath)`
    - And if it exists, do something else. Like alert the user.
  - Use other file access modes:
    - `w`: write
    - `x`: write if file doesn't already exist
    - `a`: append to file

Trying to open a file that isn't there

Australian National University

- Exceptions occur when you try to open a file that doesn't exist:

```
>>> fin = open('bad_file')
IOError: [Errno 2] No such file or directory: 'bad_file'
```

- (Sneak preview) Handling these exceptions gracefully:

```
try:
    fin = open('bad_file')
except:
    print('Something went wrong.')
```

Downey (2015) Think Python, 2nd Ed.

Putting this all together with a CSV file

Australian National University

Input file path: `/Users/dan/Desktop/example.csv`

```
chr,start,end,QS,CN,call,sample,cluster,site_name,site_count,site_freq,non_diploid_count,non_diploid_freq,num_exons
chr1,214639824,214647498,119,3,DUP,09C100236,1_1,var_1086_DUP,9,0.000326099,10,0.000362332,2
chr3,37324438,37327923,62,3,DUP,09C100236,1_1,var_1129_DUP,4,0.000144933,4,0.000144933,1
chr1,214639824,214647498,111,3,DUP,10C105228,1_1,var_1086_DUP,9,0.000326099,10,0.000362332,2
chr3,37324438,37327923,59,3,DUP,10C105228,1_1,var_1129_DUP,4,0.000144933,4,0.000144933,1
chr10,58361241,58394637,100,3,DUP,AU123A,1_18,var_32483_DUP,0,0,0,0,11
chr19,20691239,20807494,75,3,DUP,98HI0554A,1_20,var_3630_DUP,78,0.002826189,85,0.003079822,1
chr12,48939558,48941286,197,1,DEL,DEASD_0014_001,1_22,var_39923_DEL,0,0,0,0,3
chr7,22135941,22167246,74,4,DUP,80001102141,1_3,var_64984_DUP,0,0,0,0,11
chr7,107186264,107186779,110,1,DEL,ASDFI_1166,1_3,var_65108_DEL,0,0,0,0,2
```

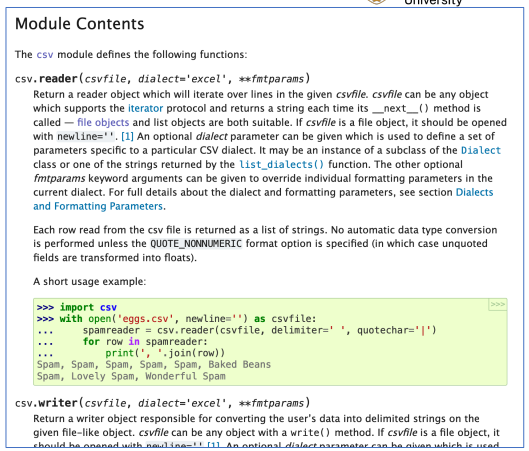
```
import os


csv_file = '/Users/dan/Desktop/example.csv'

if not os.path.exists(csv_file):
    print('File [' + csv_file + '] could not be found. ')
else:
    with open(csv_file, 'r') as input_file:
        for line in input_file:
            line_list = line.split(',')
            chr = line_list[0]
            start = line_list[1]
            end = line_list[2]
            print([chr, start, end])
```

```
['chr', 'start', 'end']
['chr1', '214639824', '214647498']
['chr3', '37324438', '37327923']
['chr1', '214639824', '214647498']
['chr3', '37324438', '37327923']
['chr10', '58361241', '58394637']
['chr19', '20691239', '20807494']
['chr12', '48939558', '48941286']
['chr7', '22135941', '22167246']
['chr7', '107186264', '107186779']
```

Another way: the CSV library

Australian National University

- Reading a CSV formatted file is a common task
- Could use the csv built-in library https://docs.python.org/3/library/csv.html
- The csv library has useful methods
  - `csv.reader()`
  - `csv.writer()`



Module Contents

The csv module defines the following functions:

csv.**reader**(*csvfile*, *dialect='excel'*, ***fmtparams*)

Return a reader object which will iterate over lines in the given *csvfile*. *csvfile* can be any object which supports the iterator protocol and returns a string each time its __next__() method is called — file objects and list objects are both suitable. If *csvfile* is a file object, it should be opened with `newline=''`. [1] An optional *dialect* parameter can be given which is used to define a set of parameters specific to a particular CSV dialect. It may be an instance of a subclass of the Dialect class or one of the strings returned by the list_dialects() function. The other optional *fmtparams* keyword arguments can be given to override individual formatting parameters in the current dialect. For full details about the dialect and formatting parameters, see section Dialects and Formatting Parameters.

Each row read from the csv file is returned as a list of strings. No automatic data type conversion is performed unless the `QUOTE_NONNUMERIC` format option is specified (in which case unquoted fields are transformed into floats).

A short usage example:

```
>>> import csv
>>> with open('eggs.csv', newline='') as csvfile:
...     spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
...     for row in spamreader:
...         print(', '.join(row))
Spam, Spam, Spam, Spam, Spam, Baked Beans
Spam, Lovely Spam, Wonderful Spam
```

csv.**writer**(*csvfile*, *dialect='excel'*, ***fmtparams*)

Return a writer object responsible for converting the user's data into delimited strings on the given file-like object. *csvfile* can be any object with a `write()` method. If *csvfile* is a file object, it

Another way: the CSV library

Australian National University

- Example with the csv built-in library:

```
import csv

csv_file = '/Users/dan/Desktop/example.csv'

with open(csv_file, 'r') as input_file:
    csv_in = csv.reader(input_file)
    for row in csv_in:
        chr = row[0]
        start = row[1]
        end = row[2]
        print([chr, start, end])
```

```
['chr', 'start', 'end']
['chr1', '214639824', '214647498']
['chr3', '37324438', '37327923']
['chr1', '214639824', '214647498']
['chr3', '37324438', '37327923']
['chr10', '58361241', '58394637']
['chr19', '20691239', '20807494']
['chr12', '48939558', '48941286']
['chr7', '22135941', '22167246']
['chr7', '107186264', '107186779']
['chr21', '34791811', '35049442']
['chr2', '50465308', '50553123']
```

Another way: open files with PANDAS

Australian National University

- What is PANDAS?
  - Like the built-in libraries, PANDAS is also a python library (but is not built-in)
  - Adds support to python for data manipulation, analysis and has data structures for manipulating numerical tables (https://pandas.pydata.org/docs/)
  - All sorts of other useful functions:
    - `read_csv()`
    - `read_json()`
    - `read_html`
    - `read_parquet()`
    - `read_excel()`

pandas

Getting started User Guide API reference Development Release notes

Input/output

pandas.read_pickle
pandas.DataFrame.to_pickle
pandas.read_table
pandas.read_csv
pandas.DataFrame.to_csv
pandas.read_fwf
pandas.read_clipboard
pandas.DataFrame.to_clipboard
pandas.read_excel
pandas.DataFrame.to_excel
pandas.ExcelFile.parse
pandas.io.formats.style.Styler.to_excel
pandas.ExcelWriter
pandas.read_json
pandas.json_normalize
pandas.DataFrame.to_json
pandas.io.json.build_table_schema
pandas.read_html
pandas.DataFrame.to_html
pandas.io.formats.style.Styler.to_html
pandas.read_xml

pandas.read_csv

pandas.read_csv(*filepath_or_buffer*, *+*, *sep=_NoDefault.no_default*, *delimiter=None*, *header='infer'*, *names=_NoDefault.no_default*, *index_col=None*, *usecols=None*, *squeeze=None*, *prefix=_NoDefault.no_default*, *mangle_dupe_cols=True*, *dtype=None*, *engine=None*, *converters=None*, *true_values=None*, *false_values=None*, *skipinitialspace=False*, *skiprows=None*, *skipfooter=0*, *nrows=None*, *na_values=None*, *keep_default_na=True*, *na_filter=True*, *verbose=False*, *skip_blank_lines=True*, *parse_dates=None*, *infer_datetime_format=False*, *keep_date_col=False*, *date_parser=None*, *dayfirst=False*, *cache_dates=True*, *iterator=False*, *chunksize=None*, *compression='infer'*, *thousands=None*, *decimal='.'*, *lineterminator=None*, *quotechar='"'*, *quoting=0*, *doublequote=True*, *escapechar=None*, *comment=None*, *encoding=None*, *encoding_errors='strict'*, *dialect=None*, *error_bad_lines=None*, *warn_bad_lines=None*, *on_bad_lines=None*, *delim_whitespace=False*, *low_memory=True*, *memory_map=False*, *float_precision=None*, *storage_options=None*) [source]

Read a comma-separated values (csv) file into DataFrame.

Also supports optionally iterating or breaking of the file into chunks.

Additional help can be found in the online docs for IO Tools.

Parameters: **filepath_or_buffer** : *str, path object or file-like object*

Any valid string path is acceptable. The string could be a URL. Valid URL

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Open CSV file with `pandas.read_csv()`

Australian National University

- Opening our file and printing what we need is much simpler:

```
import pandas as pd

csv_file = '/Users/dan/Desktop/example.csv'

csv_data = pd.read_csv(csv_file)
print(csv_data[['chr','start','end']])
```

```
>>>
>>> print(csv_data[['chr', 'start', 'end']])
      chr      start        end
0    chr1  214639824  214647498
1    chr3   37324438   37327923
2    chr1  214639824  214647498
3    chr3   37324438   37327923
4   chr10   58361241   58394637
5   chr19   20691239   20807494
6   chr12   48939558   48941286
7    chr7   22135941   22167246
8    chr7  107186264  107186779
9   chr21   34791811   35049442
10   chr2   50465308   50553123
>>>
```

- But what is the object returned by `read_csv()`?

```
>>>
>>> type(csv_data)
<class 'pandas.core.frame.DataFrame'>
>>>
```

- What can we do with a `pandas.core.frame.DataFrame`?

PANDAS can help read binary files

- Includes support for many binary file format:
  - `read_hdf5()`
  - `read_parquet()`
  - `read_excel()`
  - `read_pickle()`

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	LaTeX		Styler.to_latex
text	XML	read_xml	to_xml
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	to_orc
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Pickle files

- Sometimes it is desirable to store the state of a variable or an object to re-load in a later program run
  - Python does this with Pickle (or with Shelve)
- Pickle creates a string representation of an object, which can stored in a file or database – and later turned back into the original object:

```
>>> import pickle
>>> t = [1, 2, 3]
>>> pickle.dumps(t)
b'\x80\x03]q\x00(K\x01K\x02K\x03e.'
```

- `pickle.dump()` strings can be re-loaded with `pickle.loads()`:

```
>>> t1 = [1, 2, 3]
>>> s = pickle.dumps(t1)
>>> t2 = pickle.loads(s)
>>> t2
[1, 2, 3]
```

Pickle example

- Writing data objects to a file and reloading later:

```
import pickle
import os

signup_names = list()

if os.path.exists('names.txt'):
    names_in = open('names.txt', 'rb')
    signup_names = pickle.load(names_in)
    names_in.close()

.
.
.
```

Exercises

- Exercises in *Think Python* are very time consuming in this chapter (Ch 14). Focus on your homework instead.

Reading

- *Think Python* Ch 14 (sections: *Persistence, Reading and Writing, Filenames and Paths, Pickling*)

OR

- *Automate the boring stuff with python* Ch 9 (sections: *Files and File Paths, File Reading/Writing Process*)