



THE UNIVERSITY OF
WESTERN
AUSTRALIA

Lecture 13

File Processing Part I

Objectives

- To understand how to format strings
- To understand the basic text file processing concepts in Python.
- To learn how to read and write text files in Python and string formatting

Revision

```
intlist = []
for i in range(6) :
    if i % 2 == 0 :
        intlist.append(i)
    else:
        intlist[-1] += 1
print(intlist)
```

- What is printed when the code is executed ?

String Formatting

```
>>> amount = 1.50
```

```
>>> print(amount)
```

```
1.5
```

- If the code about is meant to represent an amount in dollars and cents, we conventionally do not use fractional dollars but rather dollars with 2 digits for cents
- Use the format method

```
>>> print("${0:0.2f} change".format(amount))
```

```
$1.50 change
```

- The first part is the string to be printed, called the **template string**. The part between { } is the **format specifier** (where the value is to be inserted and how it should look).

String Formatting

`"${0:0.2f} change".format(amount)`

- The template contains a single specifier slot with the description: `0:0.2f`
- Form of description:
`<index>:<format-specifier>`
- Index tells which parameter to insert into the slot (there can be more than one). In this case, `amount` (numbered from 0)

String Formatting

Looking at 0.2f

- The formatting specifier has the form:
`<width>.<precision><type>`
- `f` means "fixed point" number.
 - *Other specifiers are: `s` for string, `d` for integers*
- `<width>` tells us how many spaces to use to display the value. 0 means to use as much space as necessary.
- `<precision>` is the number of decimal places.

```
>>> "Compare {0} and {0:0.20f}".format(3.14)
'Compare 3.14 and 3.1400000000000000001243'
```

String Formatting Example

```
# Print out a child's multiplication table 0..10
def multiplication_table() :
    for i in range(11) :
        for j in range(11) :
            print("{0:0d} x {1:0d} = {2:0d}".format(i, j, i*j))
        print()
```

```
>>> %Run multiplication_table.py
```

```
0 x 0 = 0
0 x 1 = 0
0 x 2 = 0
0 x 3 = 0
0 x 4 = 0
0 x 5 = 0
0 x 6 = 0
0 x 7 = 0
0 x 8 = 0
0 x 9 = 0
0 x 10 = 0

1 x 0 = 0
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
```

A basic child's multiplication table to illustrate string formatting.

Needs modification to properly resemble table, e.g. numbers along top and down left hand side

F string formatting

- Introduced in Python 3.6+
- Computationally faster
- Easy to understand

```
name = "Muhammad"
age = 62
years = 21
per = 21/(62-21)

print(" {0} spent {1:0d} years or {2:2.2f}% of his life for
the cause.".format(name,years,per*100) )

#Or can write using F string

print(f" {name} spent {years} years or {per*100:2.2f}% of
his life for the cause.")
```

Output:

```
Muhammad spent 21 years or 51.22% of his life for the cause.
```

Multiline Strings

- You sometimes need strings that span more than one line. Two ways to do this:
- Embedded '`\n`' in single string
 - *"Twas brillig, and the slithy toves\nDid gyre and gimble in the wabe\nAll mimsy were the borogoves\nAnd the mome raths outgrabe."*
- Multiline string:
*""Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!""*
- `' '` single quotes are also available
- `"hello\tworld"`
- `'\t'` is the tab character

Files: Multi-line Strings

- A **file** is a sequence of data that is stored in secondary memory (disk drive).
 - *Files don't disappear when program ends*
- Files can contain any data type, but the easiest to work with are text.
- A file usually contains more than one line of text.
- Python uses the standard newline character (`\n`) to mark line breaks.

File Processing

- The process of *opening* a file involves associating a file on disk with variable in memory.
- We can manipulate the file by manipulating this variable.
 - *Read from the file*
 - *Write to the file*

File Processing

- When you've finished working with the file, it needs to be **closed**.
 - *Closing the file causes any outstanding operations and other bookkeeping for the file to be completed.*
- In some cases, not properly closing a file could result in data loss.
 - *Think of safely ejecting your memory stick*

File Processing - Reading

- Reading a file into a program, e.g. word processor
 - *File opened*
 - *Contents read into RAM*
 - *File closed*
 - *Changes to the file are made to the copy stored in memory, not on the disk.*

File Processing - Writing

- Saving a file, i.e. data in RAM onto file
 - *The original file on the disk is reopened in a mode that will allow writing (this actually erases the old contents unless specifically appending)*
 - *File writing operations copy the version of the document in memory to the disk*
 - *The file is closed*

File Processing in Python

- Working with text files in Python
 - *Associate a disk file with a file object using the open function*

```
<filevar> = open (<name>, <mode>)
```
 - *<name> is a string with the actual file name on the disk. The <mode> can be 'r', 'w' or 'a' depending on whether we are reading, writing or appending the file. You can also add '+' in mode to mention both read and write operations.*
 - ```
infile = open("numbers.dat", "r")
```

# File Methods

---

- `<file>.read()` – returns the entire remaining contents of the file as a single (possibly large, multi-line) string. Watch out for final `\n`
- `<file>.readline()` – returns the next line of the file. This is all text up to *and including* the next newline character
- `<file>.readlines()` – returns a list of the remaining lines in the file. Each list item is a single line including the newline characters.



# File Processing

---

```
printfile.py
Prints a file to the screen.

def main():
 fname = input("Enter filename: ")
 infile = open(fname, 'r')
 data = infile.read()
 infile.close()
 print(data)
```

- First, prompt the user for a file name
- Open the file for reading
- The file is read as one string and stored in the variable data

# File Processing

---

- `readline` can be used to read the next line from a file, including the trailing newline character

```
infile = open(someFile, "r")
for i in range(5):
 line = infile.readline()
 print(line[:-1])
```

- This reads the first 5 lines of a file
- Slicing is used to strip out the newline characters at the ends of the lines

# File Processing Loop

---

- Python treats the file itself as a sequence of lines!

```
infile = open(someFile, "r")
for line in infile:
 # process the line here
infile.close()
```

- Most efficient way to read through (and process) file
  - *Multiple calls to readline() is inefficient*

# File Processing

---

- Opening a file for writing prepares the file to receive data
- If you open an existing file for writing, you wipe out the file's contents. If the named file does not exist, a new one is created.

```
outfile = open("mydata.out", "w")
outfile.write(<string>)
```

May use `writelines()` for writing sequence (list) of strings

# Summary

---

- We learned how to format a string for output that is more readable and looks nice
- We learned how to read files
  - *All at once*
  - *Single lines*
  - *All the lines, line-by-line*
- We learned how to write into files