IMPERIAL

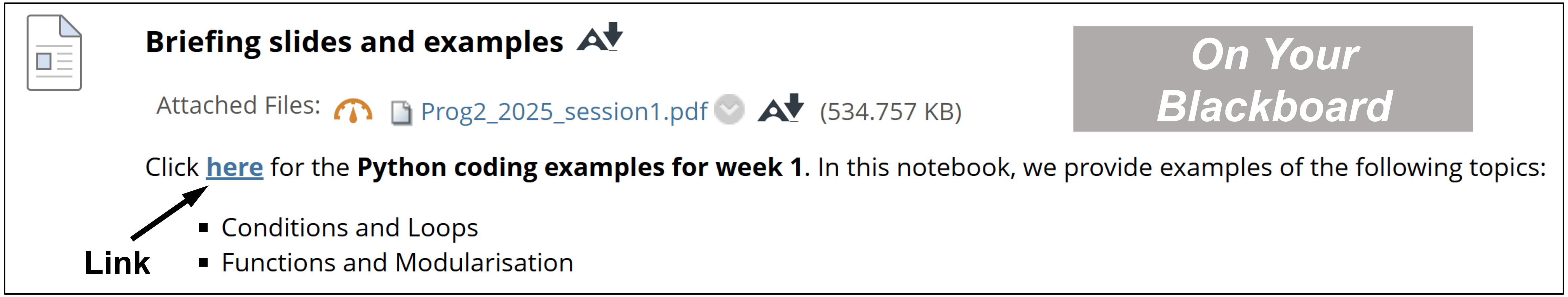
BIOE50010 – Programming 2

Computer Lab 2: Data Structures and File Handling

Binghuan Li, Maria Portela, Gauthier Boeshertz, Samuel George-White, Yilin Sun, Kamrul Hasan, Wenhao Ding, Siyu Mu, Lito Chatzidavari

12 October, 2025

Weekly Example Notebooks



Briefing slides and examples

Attached Files: Prog2_2025_session1.pdf (534.757 KB)

Click here for the **Python coding examples for week 1**. In this notebook, we provide examples of the following topics:

- Conditions and Loops
- Functions and Modularisation

Selected topics for Week 1:

- Conditions and Loops
- Functions and Modularisation

Selected topics for Week 2:

- File I/O
- String Methods
- List Methods

Feedback on Week 1

- Be careful about the Python's **condensed syntax** (commonly seen in AI solutions) .
- **Not recommended** unless you're experienced, as it compensates your code readability and maintainability.

Using "list comprehension"

✗
```
board = [[" " for _ in range(3)] for _ in range(3)]
```

… which is equivalent to

```
board = []
for _ in range(3):
    row = []
    for _ in range(3):
        row.append(" ")
    board.append(row)
```

Create a 2D list structure using nested `for`-loops

Feedback on Week 1 – board printing

Code snippet from `tictactoe.py`

```
board = [“ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”, “ ”]

def displayBoard(board):
     print(board[0] + '|' + board[1] + '|' + board[2])
     print('-----')
     print(board[3] + '|' + board[4] + '|' + board[5])
     print('-----')
     print(board[6] + '|' + board[7] + '|' + board[8])
```

Initialize a 1D list with 9 elements (spaces), also `[“ ”]*9`

All these are just formatting!

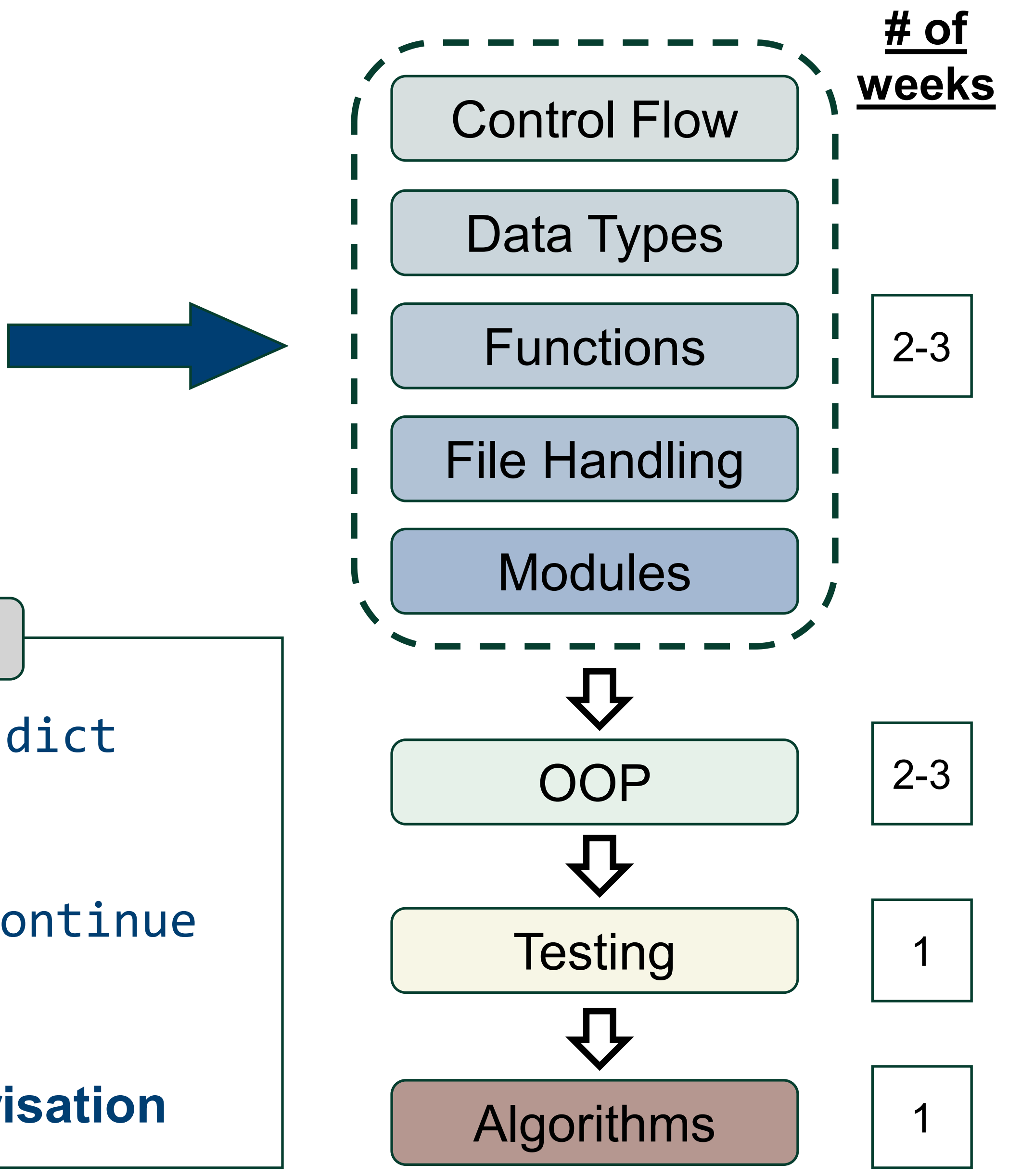
- **Format** your output, rather than directly **printing** out a 2D list…

```
board = [[1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]]
print(board)
```

✗ This won’t format your board… unfortunately

Progress Check

Week 2:
we are here

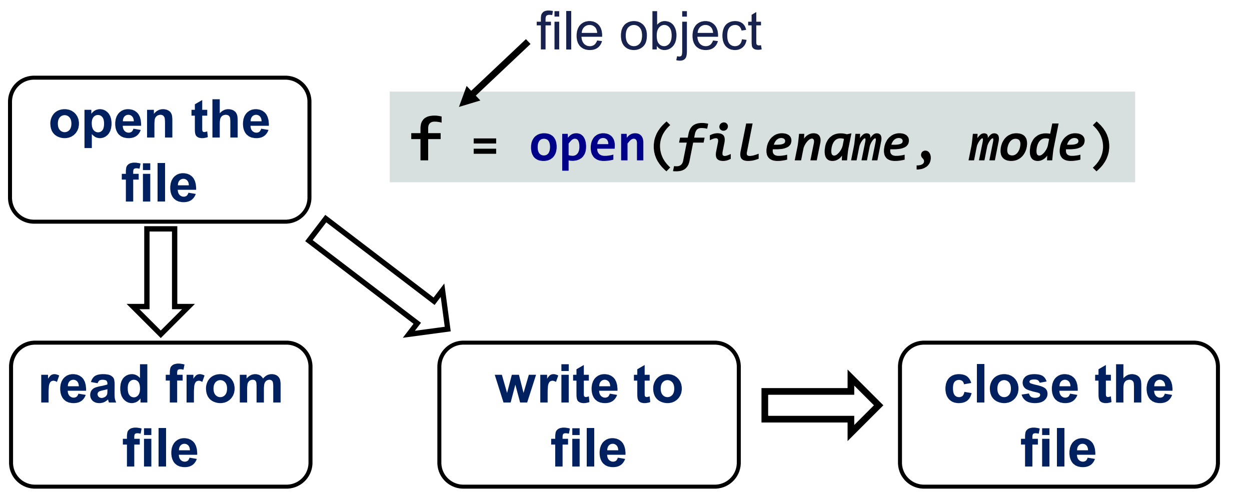


Revision Points (from week 1)

- **Data structures**: `int`, `float`, `str`, `list`, `dict`
- **Typecasting**: *e.g.*, `str` "2" to `int 2`
- **Control flow**: conditions, loops, `break`, `continue`
- **Input** and **print**
- **Functions** and concept of code **modularisation**

File Input/Output (I/O)

- A program saves into a file for later use, it **writes** data into a file; the data can be **read** into the program from the file in future.



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

```
f.read()
f.readline()
f.readlines()
```

```
f.write()
f.writeline()
f.writelines()
```

```
f.close()
```

`f = open(…)` returns a **file object**. Possible things you can do with the file (***mode***):

- ‘r’ read (default)
- ‘w’ write
- ‘a’ append
- ‘x’ create
- ‘b’ binary

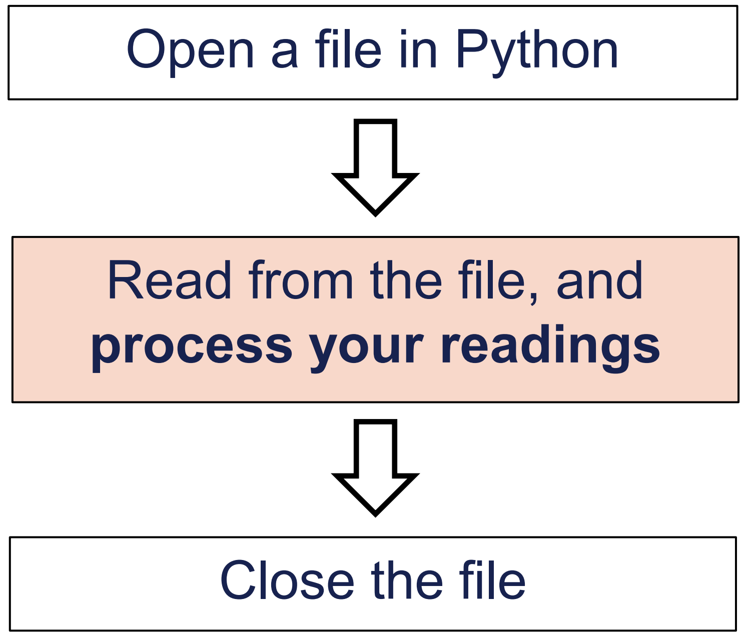
⇩

You process the file: read or write

⇩

Close the file once finished all operations!

String and List Methods



- After reading data from a file, the contents are saved in a structure such as a **string** or a **list**.
- It is your task to **process the raw readings** before using them for further analysis: clean, transform, sort, organise…
- These operations can be done with **string and list methods**, *e.g.*,
  - Use `.split()` or `.strip()` to process text strings.
  - Use `.append()` or `.sort()` to manage lists.

Example of Using String Methods

`.split(`*`delimiters`*`):` split a string by specified **delimiters**

```
text =   The quick brown fox , jumps over * the lazy dog.

                     ⬇ text.split(',')

         The quick brown fox  |  jumps over * the lazy dog.   = text_2

                                   ⬇ text_2.split('*')

         The quick brown fox  |  jumps over  |  the lazy dog.
```

`.strip():` remove white spaces (or specified strings) at both ends of the string

```
text =   “␣␣The quick brown fox jumps over the lazy dog␣␣”

                     ⬇ text.strip()

         “The quick brown fox jumps over the lazy dog”
```

Your tasks today

1. Familiarize yourself with **OS commands** in Windows Command Prompt or Mac Terminal (these are NOT the Python tasks!).
2. Three mini tasks in Python on **file I/O**, with **use of string/list methods**:
   - Task 1: read a poem from a `.txt` file
   - Task 2: read and format the DNA to protein data from a `.csv` file
   - Task 3: read and process nucleotide sequences

To start…

1. Syntax learning: **lecture slides** and **weekly example notebook**.
2. Coding requirements: your output *must* match the given console output.
3. Use the **appendices** when necessary.

Questions?

That's it for now.

You can now proceed to the Lab 2 exercises.

Appendix 1: **Summary of OS Commands**

- **These are NOT Python commands!**
- They are used to perform file management tasks (*e.g.*, copy and paste files) in operating systems (OS) without relying on the graphical user interfaces.

Tasks	**Windows Command** (to be used in Command Prompt)	**Unix-like OS Command** (to be used in Terminal on MacOS)
change directory (folder)	`cd`	`cd`
directory listing	`dir`	`ls -l`
copy a file	`copy`	`cp`
move a file ("cut")	`move`	`mv`
delete a file	`del`	`rm`
clear screen	`cls`	`clear`
display current directory location	`chdir`	`pwd`
create a new directory	`md`	`mkdir`
delete a directory	`rmdir`	`rm -rf/rmdir`

Appendix 2: **More on File Input/Output (I/O)**

Why do I need to close the file?

- more file handlers = more space used in RAM → performance compensation
- many changes to files in python do not go into effect until the file is closed
- likelihood for data corruption
- theoretically, the number of file handlers has a limit

Relative path and absolute path

- Using **absolute path**: starting from the *root directory*

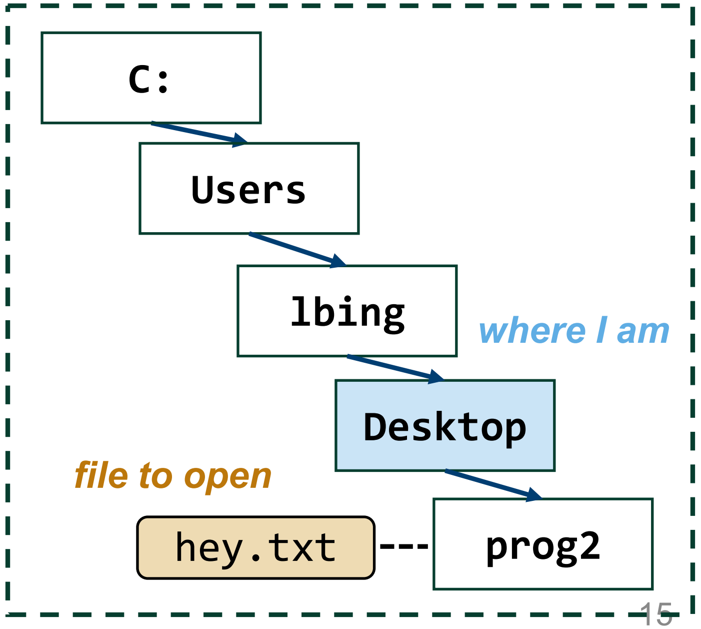
```
f = open('C:/Users/lbing/Desktop/prog2/hey.txt', 'w')
```

full path starting from `C:` disk

- Using **relative path**: with respect to the *current directory*

```
f = open('./prog2/hey.txt', 'w')
```

the current directory (…\Desktop)

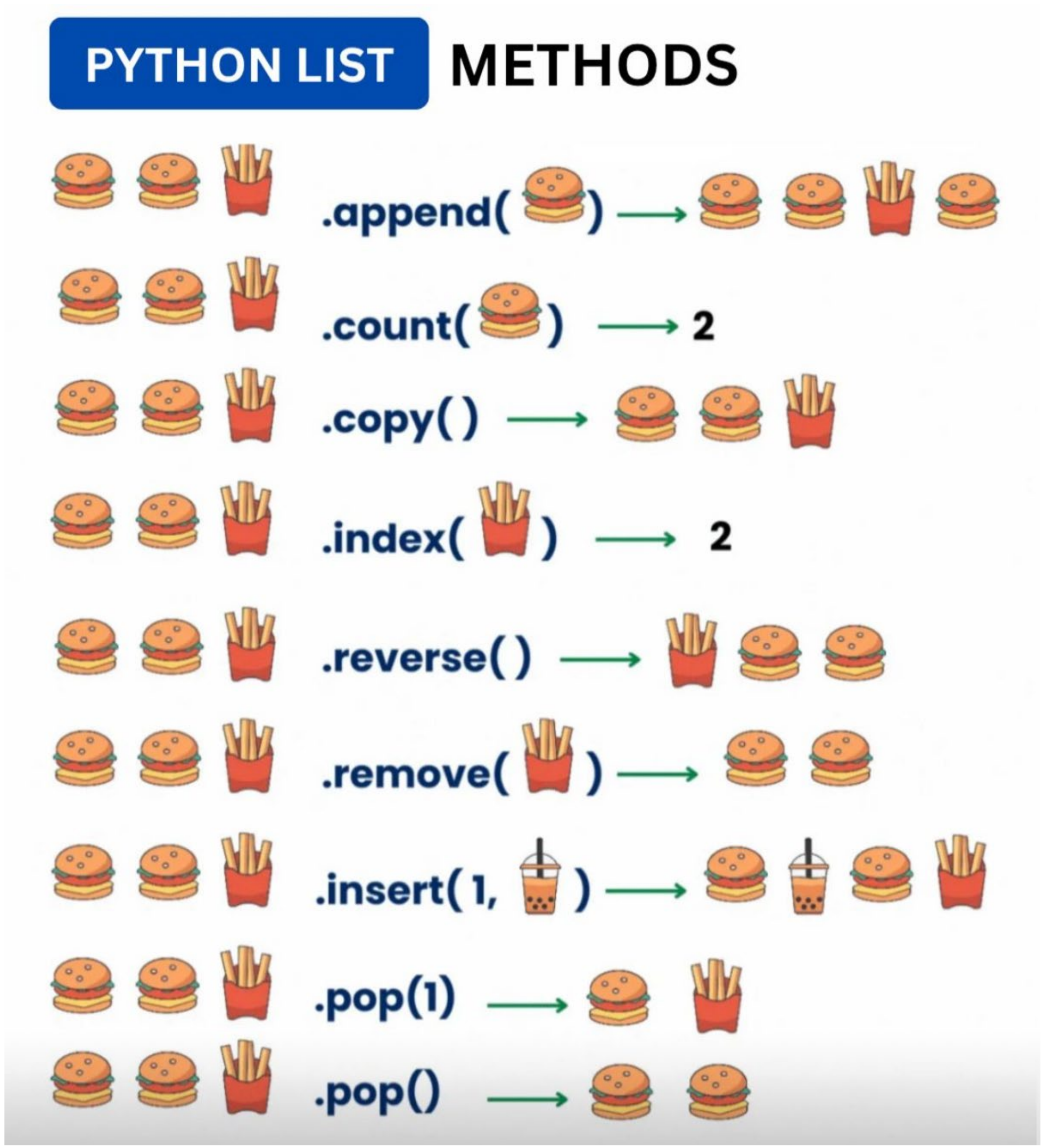


Appendix 3: **Potentially Useful String Methods**

Method	**Description**
`startswith(substring)`	The method returns **`true`** if the string starts with *substring*.
`endswith(substring)`	The method returns **`true`** if the string ends with *substring*.
`find(substring)`	The method returns the lowest index in the string where *substring* is found. If *substring* is not found, the returns -1.
`replace(old, new)`	The method returns the string with all instances of *old* replaced by *new*.
`lstrip(char)`	The method returns a copy of the string with the specified character (*char*) that appear at the beginning (left) of the string removed.
`rstrip(char)`	The method returns a copy of the string with the specified character (*char*) that appear at the end (right) of the string removed.
`split(delimiter)`	The method returns a list containing the words in the string separated by the specified *delimiter*, by default the *delimiter* is a whitespace.

Appendix 4: **Potentially Useful List Methods**

Method	**Description**
`append(item)`	Adds *item* to the end of the list.
`index(item)`	Returns the index of the first element whose value is equal to *item*. A `ValueError` exception is raised if item is not found in the list.
`insert(index, item)`	Inserts *item* into the list at the specified *index*.
`sort()`	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
`remove(item)`	Removes the first occurrence of *item* from the list. A `ValueError` exception is raised if *item* is not found in the list.
`reverse()`	Reverses the order of the items in the list.



By Brij kishore Pandey