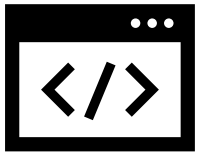
IMPERIAL



BIOE50010 – Programming 2

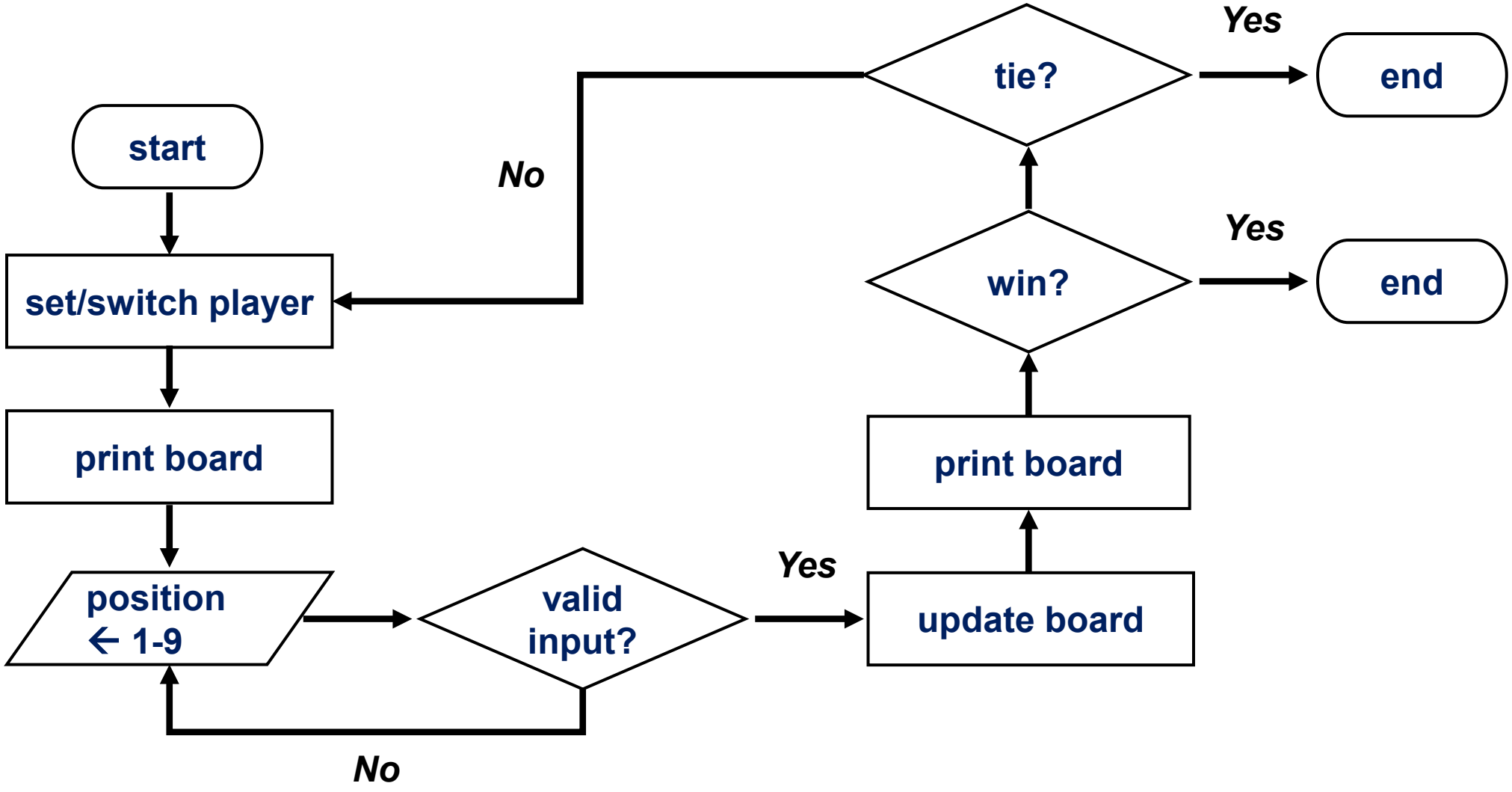
Computer Lab 2

Binghuan Li Department of Chemical Engineering

Maria Portela Department of Bioengineering

Wenhao Ding Department of Bioengineering

13 October, 2024


start
set/switch player
print board
position
← 1-9
valid
input?
No
Yes
update board
print board
win?
Yes
end
tie?
No
Yes
end

displayBoard

Code snippet from `tictactoe_sol1.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!

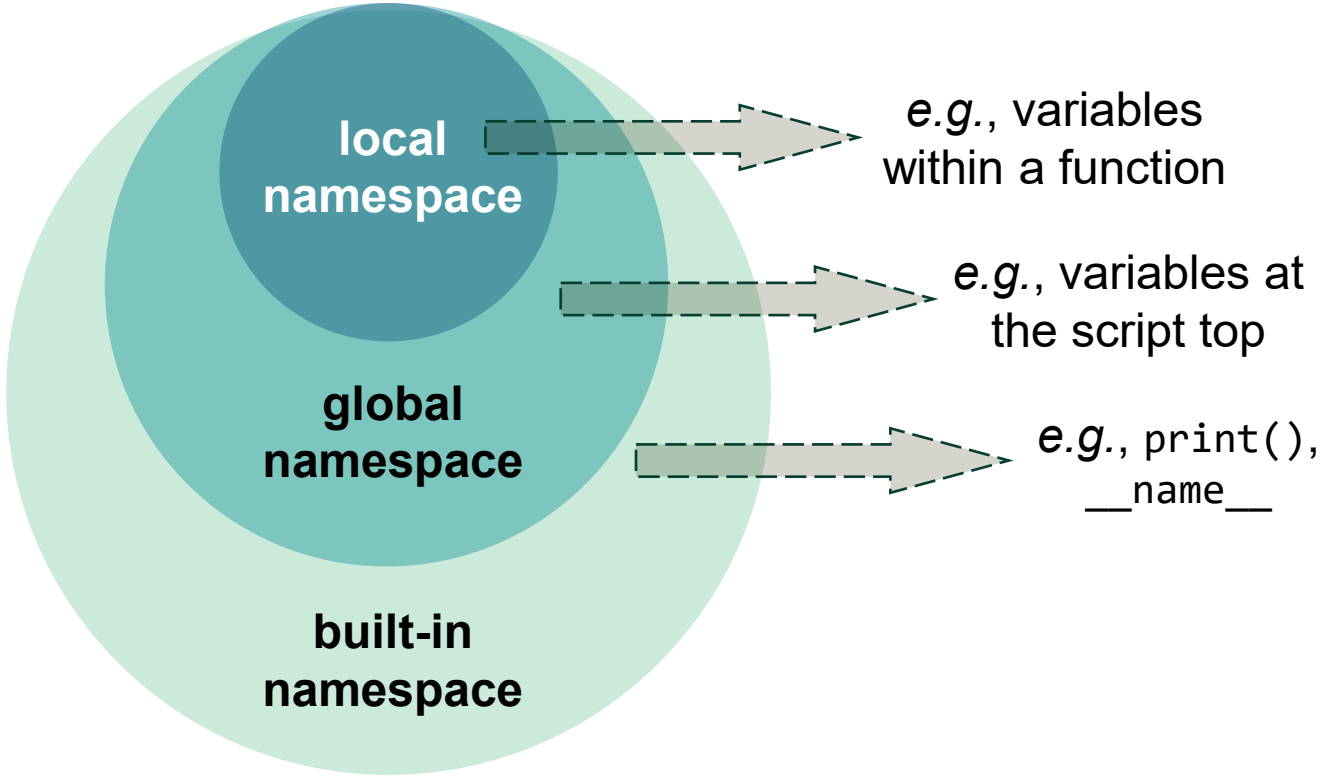
- **Format** your output, rather than directly **printing** out a 2-D list…

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

This won’t format your board… unfortunately

Namespace

- A **namespace** holds a set of variable names or functions (objects) that belongs to a specific context (**scope**) within the program.



Example

```
x = 'global!'

def print_x():
    x = 'local!'
    print(x)

def main():
    print_x()
    print(x)

if __name__ == '__main__':
    main()
```

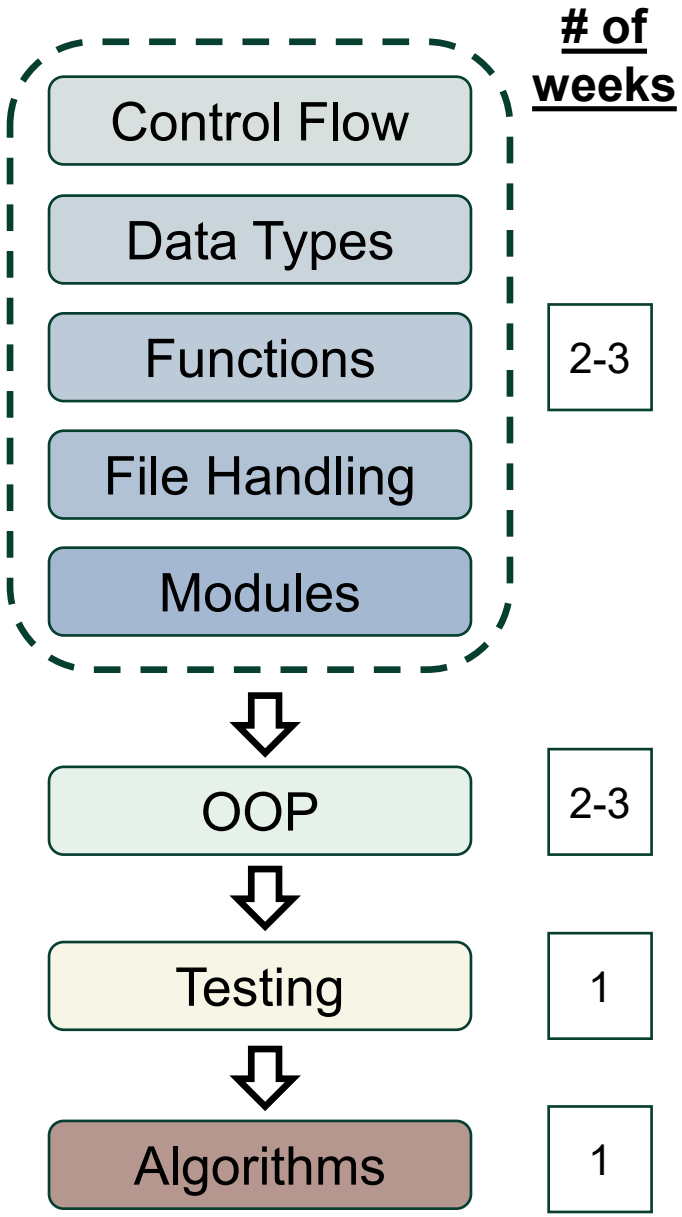
same variable name but hold **different** values

Console

```
>> global!
Local!
```

Progress Check

Week 2:
we are here

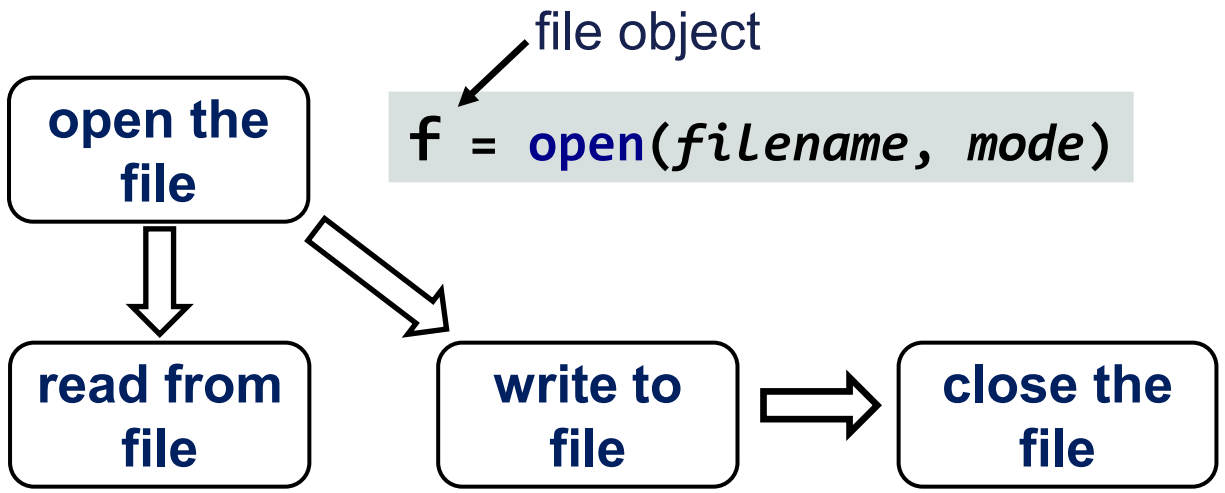


Checklist: you should have mastered…

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

File 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.read()`	`f.write()`	
`f.readline()`	`f.writeline()`	`f.close()`
`f.readlines()`	`f.writelines()`	

`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!

Common Questions

Why do I need to close the file?

- more file handlers = more space used in RAM, hence impact the performance
- 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

- With the absolute path:

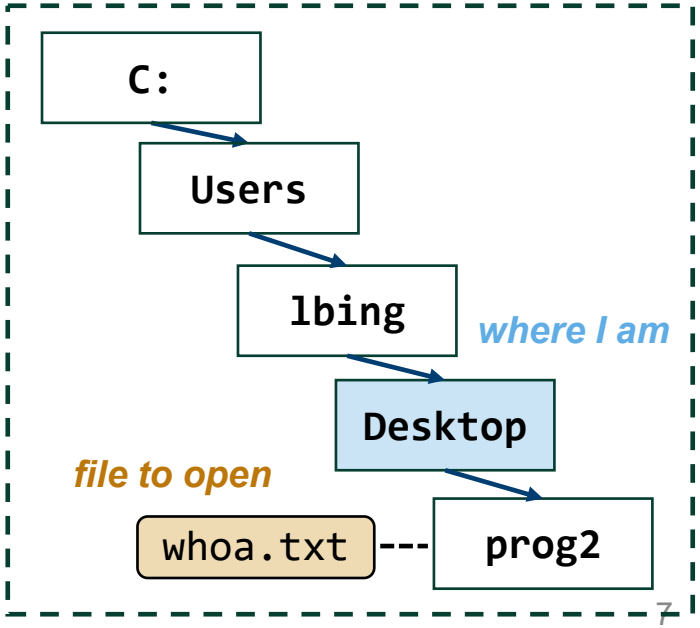
```
f = open('C:/Users/lbing/Desktop/prog2/whoa.txt', 'w')
```

full path starting from `C:` disk

- With the relative path:

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

the current directory (…\Desktop)



Your tasks today

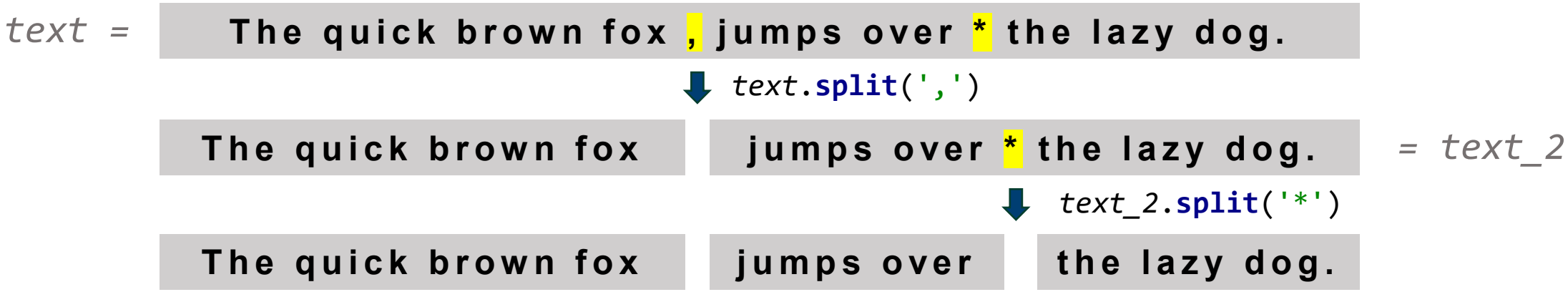
1. Familiar yourself with **Navigating Folders** in Command Prompt / Terminal (not in Python)
2. Three mini tasks on **file I/O** with **use of Python string / list methods**
   - Task 1: read a poem from a `.txt` file
   - Task 2: read and format the DNA to protein data from a `.csv` file
   - Task 3: read and process nucleotide sequences

To start…

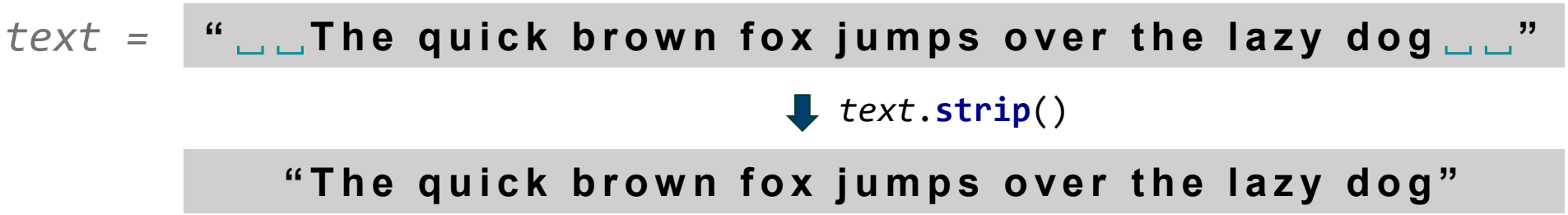
- Study the Python scripts from your Friday lecture
- Read all information and the sample output provided in the lab carefully
- Consult the help pages attached to the end of the document when necessary

Hints: Two Useful String Methods

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



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



Questions?

That's it for now.

You can now proceed to the Lab 2 exercises.

OS Commands

- **Command Prompt** in MS Windows, **Terminal** in MacOS / some Linux distributions

Tasks	Windows Command	Unix-like OS Command (e.g. MacOS, Linux)
change directory	`cd`	`cd`
directory listing	`dir`	`ls -l`
copying a file	`copy`	`cp`
moving a file	`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`

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.

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.

Need More Help?

Coding examples

- **Working with Lists and Files**:
  - pp.395-397 program 7-13 (writelines.py), program 7-14 (write_list.py), and program 7-15 (read_list.py)
  - also read pp. 328 figure 6-17 for general logic for detecting the end of a file