CS112
Introduction to Python Programming

Session 09: Input and Output

Shengwei Hou
Ph.D., Assistant Professor
Department of Ocean Science and Engineering
Fall 2022

SUSTech Southern University of Science and Technology

明德求是 日新自强
VIRTUE | TRUTH | ADVANCE

Contents

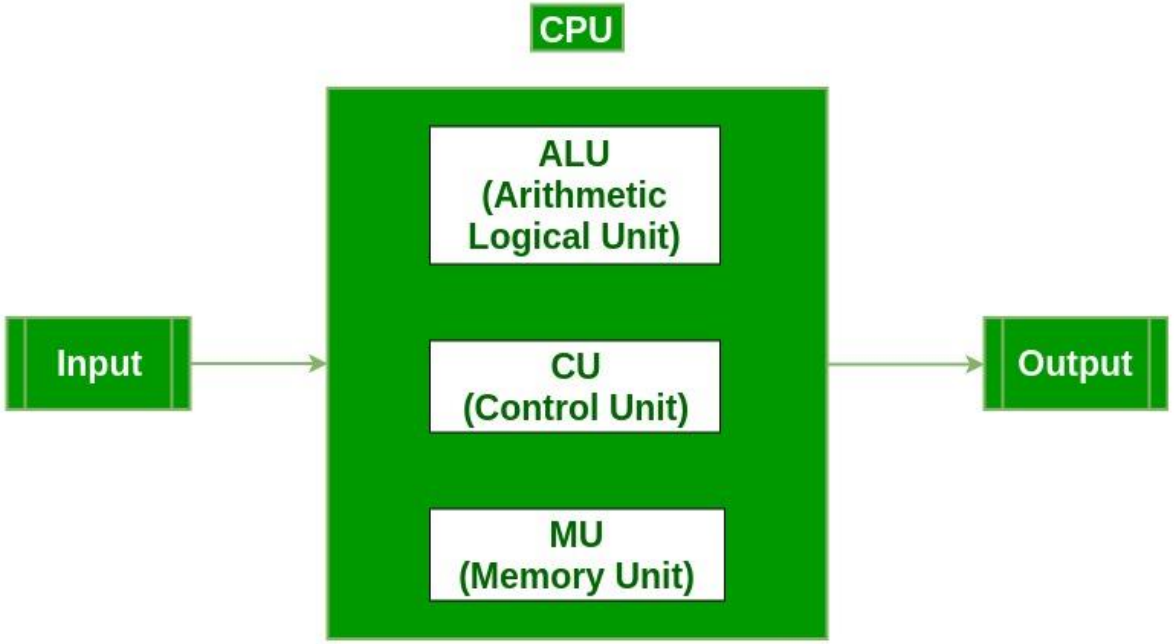
SUSTech Southern University of Science and Technology

- Keyboard input
- Screen output
- Files and file path
- Creating files
- `with` statement
- `read` and `readlines`
- `write` and `writelines`
- Read and write binary files
- Read and write csv files
- Dump and load `pickle` objects
- ~~Load and save text files using NumPy~~

readline read a line

Introduction

- Communicating with computers has two ways: input and output, i.e., I/O
- Two venues for input: the computer ~~keyboard~~ and the input data ~~file~~
- Similarly, two venues for output: the monitor screen and the output data file



Keyboard input

- Python has an `input` function for getting input from the user and assigning it a variable name:

```
strname = input("prompt to user")
```

```
>>> distance = input("Input distance (miles): ")
Input distance (miles): 450
>>> distance
'450'
```

- The result obtained from Python `input` function is always a string, sometimes we need to convert it to a number:

```
>>> distance = eval(distance)
>>> distance
450
>>> distance = int(distance)
>>> distance = float(distance)
```

string → integer.

Screen output

- Screen output can be accomplished using Python's `print` function:

```
>>> country = "China"
```

- `str.format()` method

```
>>> print("I live in {0}".format(country))
```

- `f-string` method

```
>>> print(f"I live in {country}")

I live in China
```

Screen output

- Print NumPy arrays with a specified number of decimal points:

```
>>> import numpy as np
>>> a = np.linspace(3, 19, 7)
>>> print(a)
[ 3.          5.66666667  8.33333333 11.         13.66666667
16.33333333 19.        ]
>>> np.set_printoptions(precision=2)
>>> print(a)
[ 3.    5.67  8.33 11.   13.67 16.33 19.  ]
>>> np.set_printoptions(precision=4)
>>> print(a)
[ 3.      5.6667  8.3333 11.     13.6667 16.3333 19.    ]
```

- To return to the default format, type the following:

```
np.set_printoptions(precision=8)
```

Files

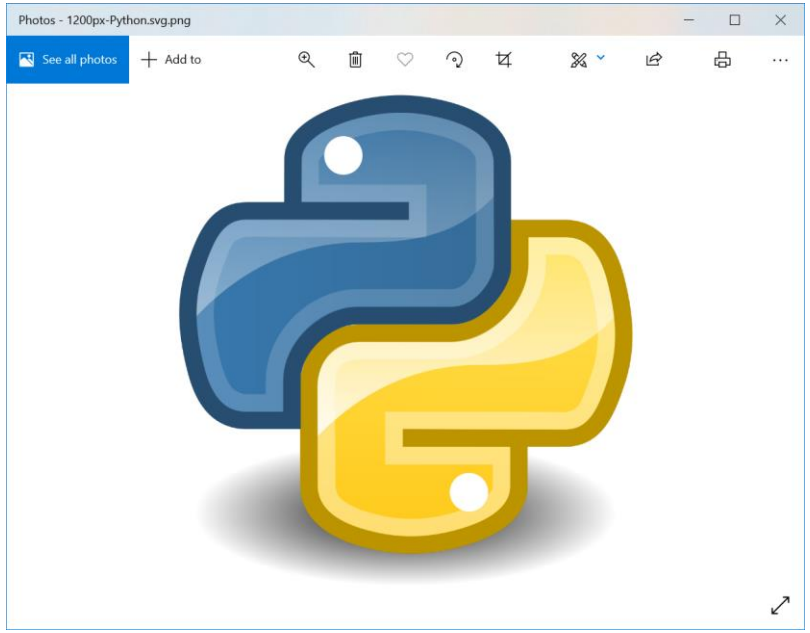
- A file is the common storage unit in a computer, and all programs and data are "written" into a file and "read" from a file
- A file extension is the character or group of characters after the period that makes up an entire file name:

```
assignments.docx
assignments.pdf
data.csv
```

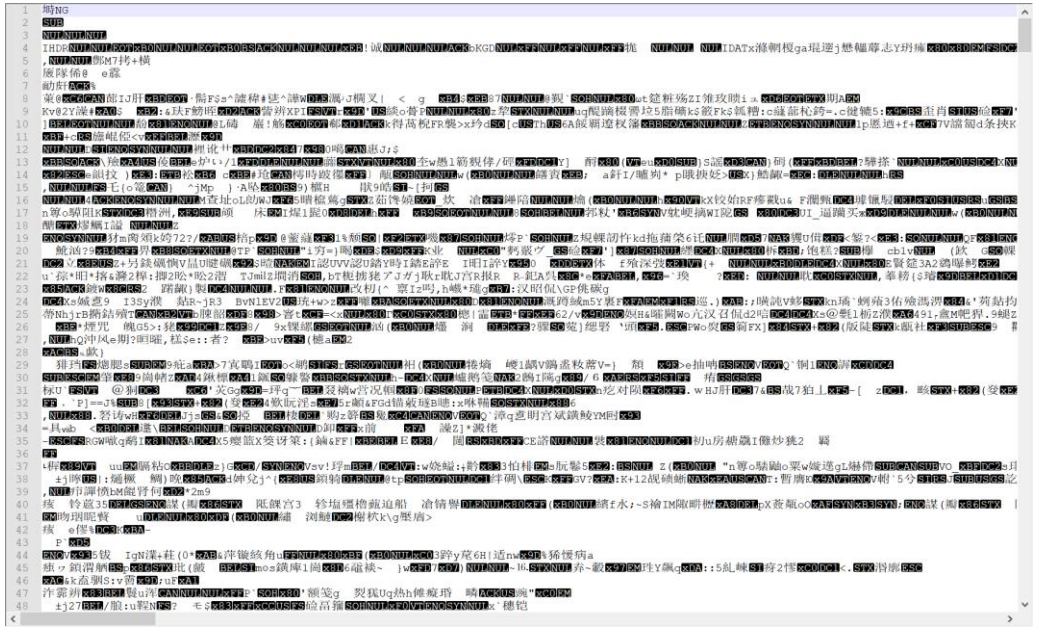
- File extensions often indicate the file type, or file format, of the file but not always. Any file's extensions can be renamed, but that will not convert the file to another format or change anything about the file other than this portion of its name
- All operating systems follow the same general naming conventions for an individual file: a base file name and an optional extension, separated by a period

Files

- Python supports two types of files – text files and binary files, and they encode data differently
- Both binary and text files contain data stored as a series of bits (binary values of 1s and 0s), the bits in text files represent characters, while the bits in binary files represent custom data



python.png



Files

- Binary file formats may include multiple types of data in the same file
- Binary files often contain headers, which are bytes of data at the beginning of a file that identifies the file's contents
- If a binary file has an invalid header, a software program may not open it or may report that the file is corrupted
- Text files are more restrictive than binary files since they can only contain textual data, and are less likely to become corrupted
- A typical plain text file contains several lines of text that are each followed by an End-of-Line (EOL) character. An End-of-File (EOF) marker is placed after the final character, which signals the end of the file
- Binary files are often of smaller size than text files, and are faster than text files in I/O

File path

- A path is needed to make use of files, which is a route so that the user or the program knows where the file is located
- The path to a specified file consists of one or more components, separated by a special character: a backslash (\) for Windows and forward slash for Linux (/)
- In Windows, the maximum length for a path is 260 characters and in Linux the maximum path length is of 4096 characters
- File and Directory names in Windows are not case sensitive while in Linux it is case sensitive

```
"D:\" and "d:\" refer to the same drive
```

absolute path
Win: C:\\A\x.py
Linux: /home/username/A/x.py

relative path
$ python ../A/x.py

A
- x.py
B
- y.py

File path

- A file path can be absolute path or relative path
- A path is an absolute path if it points to the file location, which always contains the root and the complete directory list
- The root directory contains all other directories in the drive. The main root directory of the Windows system is `C:\` and the root directory in Linux system is `/`(forward slash)
- A path is a relative path if it contains “double-dots” or “single-dot” :

  `"..\langur.txt"` specifies a file named `"langur.txt"` located in the parent of the current directory

  `".\bison.txt"` specifies a file named `"bison.txt"` located in a current directory

  `"..\..\langur.txt"` specifies a file that is two directories above the current directory

Creating files

- All files must be opened first before they can be read from or written to using the Python's built-in `open()` function:

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

  - `filename` is a **string** containing the file path to be opened
  - `mode` (optional) is a **string** describing the way the file is used

Mode	Description
"r"	open the file in read only mode (default)
"w"	open the file for writing. if exists, it'll be overwritten; else, creates a new file
"a"	open the file for appending data at the end of the file automatically
"r+"	open the file for both reading and writing → not recommended
"w+"	open the file for reading and writing. if exists, it'll be overwritten; else, creates a new file
"a+"	open the file for reading and appending. if exists, the data will be appended; else, creates a new file
"rb"	open the binary file in read-only mode
"wb"	open the file for writing the data in binary format

Creating files

- The `open()` method returns a file handler object that can be used to read or modify the file:

```
>>> file_handler = open("moon.txt","r")
>>> file_handler = open("C:\\langur.txt","r")
>>> file_handler = open("C:\prog\example.txt","r")
>>> file_handler = open("..\\bison.txt","r")
```

- The `close()` method close the file once the processing is completed:

```
>>> file_handler = open("moon.txt","r")
>>> file_handler.close()
```

- If the file is not closed explicitly, Python's garbage collector will eventually destroy the object and close the opened file, but the file may have stayed open for a while

Example


```
def read_file():
    file_handler = open("file_test.txt")
    print("Printing each line in the text file")
    for each_line in file_handler:
        print(each_line, end="")
    file_handler.close()
def main():
    read_file()
if __name__ == "__main__":
    main()
```

(mode is read by default)

"for" statement can be applied to a sequence

if no "end=''" it prints an additional null row.

file_test.txt

```
The Python's core philosophy:
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Readability counts.
```

with statement

- The with statement automatically closes the file after executing its block of statements:

```
with open (file, mode) as file_handler:
    Statement_1
    Statement_2
```

- Example:

```
def read_file():
    print("Printing each line in the text file")
    with open("file_test.txt") as file_handler:
        for each_line in file_handler:
            print(each_line, end="")
def main():
    read_file()
if __name__ == "__main__":
    main()
```

File object attributes

- The file handler has various attributes:

```
>>> fh = open("computer.txt", "w")
>>> fh.name
'computer.txt'
>>> fh.closed
False
>>> fh.mode
'w'
```

Attribute	Description
`fh.closed`	return a Boolean `True` if the file is closed or `False` otherwise
`fh.mode`	return the access mode with which the file was opened
`fh.name`	return the name of the file

File methods to read and write data

- List of methods associated with the file object:

Method	Description
`fh.read([size])`	read the contents of a file up to a size and return as a string. The argument `size` is optional, and, if not specified, the entire contents of the file will be read and returned
`fh.readline()`	read a single line in file
`fh.readlines()`	read all the lines of a file as ~~list~~ items
`fh.write()`	write the contents of the string to the file, returning the number of characters written
`fh.writelines()`	write a sequence of strings to the file
`fh.tell()`	return an integer giving the file handler's current position within the file, measured in bytes from the beginning of the file
`fh.seek(offset, from_what)`	change the file handler's position. The position is computed from adding offset to a reference point. The reference point is selected by the `from_what` argument. A `from_what` value of 0 measures from the beginning of the file (default), 1 uses the current file position, and 2 uses the end of the file as the reference point

Examples: read


```
>>> fh = open("example.txt", "w")
>>> fh.write("abcdefgh")
8
>>> fh.close()
>>> fh = open("example.txt")
>>> fh.read(2)
'ab'
>>> fh.read(2)
'cd'
>>> fh.read()
'efgh'
```

file_test.txt

```
1  The Python's core philosophy:
2  Beautiful is better than ugly.
3  Explicit is better than implicit.
4  Simple is better than complex.
5  Complex is better than complicated.
6  Readability counts.
```

```
print("Printing entire file contents:")
with open("file_test.txt") as fh:
    print(fh.read(), end="")
```

Examples: `readline` and `readlines`


```
with open("file_test.txt") as fh:
    print("Printing a single line from the file:")
    print(fh.readline(), end="")
    print("Printing a single line from the file:")
    print(fh.readline(), end="")
```

```
Printing a single line from the file:
The Python's core philosophy:
Printing a single line from the file:
Beautiful is better than ugly.
```

```
with open("file_test.txt") as fh:
    print("Printing file content as a list:")
    print(fh.readlines())
```

rows → elements of the list

```
Printing file content as a list:
["The Python's core philosophy:\n", 'Beautiful is better than
ugly.\n', 'Explicit is better than implicit.\n', 'Simple is better
than complex.\n', 'Complex is better than complicated.\n',
'Readability counts.']
```

Examples: `write` and `writelines`


```
>>> fh = open("moon.txt","w")
>>> fh.write("Moon is a natural satellite")
27
>>> fh.close()
>>> fh = open("moon.txt", "a+")
>>> fh.write("of the earth")
12
>>> fh.close()
>>> fh = open("moon.txt")
>>> fh.read()
'Moon is a natural satelliteof the earth'
>>> fh.close()
>>> fh = open("moon.txt", "w")
>>> fh.writelines(["Moon is a natural satellite", " ", "of the earth"])
>>> fh.close()
>>> fh = open("moon.txt")
>>> fh.read()
'Moon is a natural satellite of the earth'
>>> fh.close()
```

Examples: seek and tell


```
>>> f = open('workfile', 'w')
>>> f.write('0123456789abcdef')
16
>>> f.close()
>>> f = open('workfile', 'r')
>>> f.seek(5)
5
>>> f.read(2)
'56'
>>> f.tell()
7
>>> f.read()
'789abcdef'
>>> f.tell()
16
>>> f.read()
''
```

file_test.txt

```
1  The Python's core philosophy:
2  Beautiful is better than ugly.
3  Explicit is better than implicit.
4  Simple is better than complex.
5  Complex is better than complicated.
6  Readability counts.
```

```
with open("file_test.txt", 'r') as ih:
    print(ih.read(2))
    print(ih.tell())
    print(ih.read(2))
    print(ih.tell())
    print(ih.readline())
    ih.seek(4)
    print(ih.readline())
```

```
Th
2
e
4
Python's core philosophy:

Python's core philosophy:
```

`fh.tell()`	**return an integer giving the file handler's current position within the file, measured in bytes from the beginning of the file**
`fh.seek(offset, from_what)`	change the file handler's position. The position is computed from adding offset to a reference point. The reference point is selected by the `from_what` argument. A `from_what` value of 0 measures from the beginning of the file (default), 1 uses the current file position, and 2 uses the end of the file as the reference point

Read and write binary files

- Files opened in binary mode (including 'b' in the mode argument) return contents as bytes objects without any decoding
- A Python program to create a new image from an existing image:

```
def main():
    with open("python.png", "rb") as exist_image,
open("python_new.png", "wb") as new_image:
        for each_line_bytes in exist_image:
            new_image.write(each_line_bytes)
if __name__ == "__main__":
    main()
```

Example


```
def main():
    with open("workfile.bin", "wb+") as f:
        f.write(b'abcdef')
    with open("workfile.bin", "rb") as f:
        byte = f.read(1)
        print("Print each byte in the file")
        while byte:
            print(byte)
            byte = f.read(1)
if __name__ == "__main__":
    main()
```

```
Print each byte in the file
b'a'
b'b'
b'c'
b'd'
b'e'
b'f'
```

The `pickle` module

- The `read()` method only returns strings. However, things get a lot more complicated when you want to save more complex data types like lists, dictionaries, or class instances
- The `pickle` module can take almost any Python object and convert it to a string – pickling:

```
pickle.dump(obj, file_handler)
```

- Reconstructing the object from the string representation is called unpickling:

```
obj = pickle.load(file_handler)
```

Example

- A program to save a dictionary in python pickle:

```
import pickle
def main():
    bbt = {'cooper': 'sheldon'}
    with open('filename.pickle', 'wb') as handle:
        pickle.dump(bbt, handle)
    with open('filename.pickle', 'rb') as handle:
        bbt = pickle.load(handle)
        print(f"Unpickling {bbt}")
if __name__ == "__main__":
    main()
```

```
Unpickling {'cooper': 'sheldon'}
```

Read and write CSV files

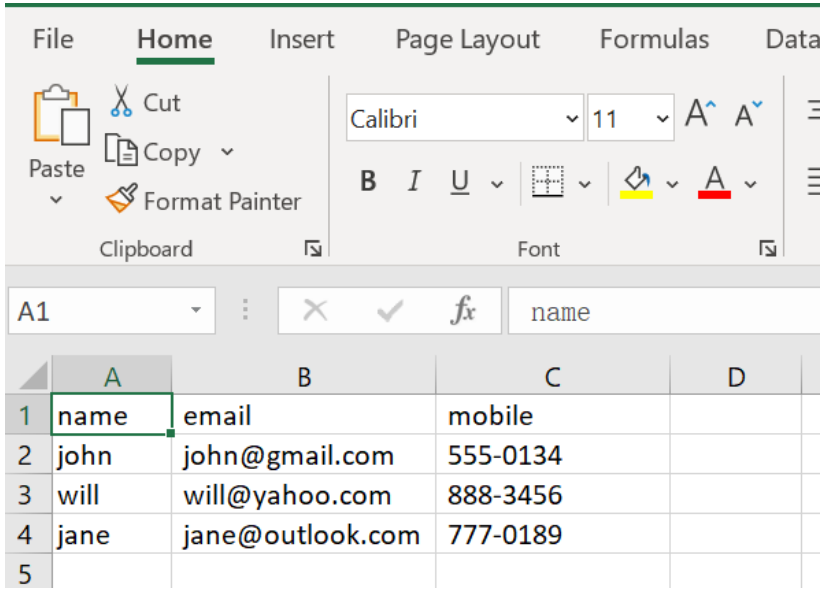
- CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases
- Columns are separated with commas, and rows are separated by line breaks or the invisible "\n" character. However, the last value is not followed by a comma

CSV file "contact.csv":

```
name, email, mobile
john, john@gmail.com, 555-0134
will, will@yahoo.com, 888-3456
jane, jane@outlook.com, 777-0189
```

contact.csv

```
1  name, email, mobile
2  john, john@gmail.com, 555-0134
3  will, will@yahoo.com, 888-3456
4  jane, jane@outlook.com, 777-0189
```



	A	B	C	D
1	name	email	mobile	
2	john	john@gmail.com	555-0134	
3	will	will@yahoo.com	888-3456	
4	jane	jane@outlook.com	777-0189	
5				

Spaces after "," are a part of the field

- *Human-readable format and easy to edit manually*
- *Simple to generate, parse and handle*
- *It is a standard format and is supported by many applications*

```
['name', ' email', ' mobile']
['john', ' john@gmail.com', ' 555-0134']
['will', ' will@yahoo.com', ' 888-3456']
['jane', ' jane@outlook.com', ' 777-0189']
```

Read and write CSV files

- `csv` module can be used to work with CSV files:

```
import csv
```

- To read from a CSV file use `csv.reader()` method:

```
csv.reader(csvfile)
```

- To write to a CSV file, use the `csv.writer()` method:

```
csvwriter = csv.writer(csvfile)
csvwriter.writerow(row)
csvwriter.writerows(rows)
```

Example

- Write a program to read and display each row in "*biostats.csv*" CSV file:

biostats.csv:

```
Name,"Sex","Age","Height(in)","Weight(lbs)"
Alex,"M",41,74,170
Bert,"M",42,68,166
Elly,"F",30,66,124
Fran,"F",33,66,115
```

```
import csv
with open('biostats.csv') as csvfile:
    csv_reader = csv.reader(csvfile)
    print("Print each row in CSV file: ")
    for each_row in csv_reader:
        print(",".join(each_row))
```

Example

- Write the following contents into a CSV file using `writerow` and `writerows`:

```
csv_header = ['Name', 'Sex', 'Age', 'Height(in)', 'Weight(lbs)']
each_row = [['Alex', 'M', '41', '74', '170'],
            ['Bert', 'M', '42', '68', '166'],
            ['Elly', 'F', '30', '66', '124'],
            ['Fran', 'F', '33', '66', '115']]
import csv
with open('biostats.csv', 'w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(csv_header)
    csv_writer.writerows(each_row)
```

Example

- Read data from '*biostats.csv*' CSV file using `DictReader`:

```
import csv
with open('biostats.csv') as csvfile:
    csv_reader = csv.DictReader(csvfile)
    for row in csv_reader:
        print(f"{row['Name']}, {row['Age']}")
```

```
Alex, 41
Bert, 42
Elly, 30
Fran, 33
```

Example

- Write data to a CSV file using `DictWriter`:

```
with open('names.csv', 'w', newline='') as csvfile:
    field_names = ['first_name', 'last_name']
    writer = csv.DictWriter(csvfile, fieldnames=field_names)
    writer.writeheader()
    writer.writerow({'first_name': 'Baked', 'last_name': 'Beans'})
    writer.writerow({'first_name': 'Lovely', 'last_name': 'Spam'})
    writer.writerow({'first_name': 'Wonderful', 'last_name': 'Spam'})
```

Read data from text file using NumPy

- Use **NumPy**'s `loadtxt` to load data from a plain text file having the same number of values in each row:

mydata.txt

```
Data for falling mass experiment
Date: 16-Aug-2016
Data taken by Lauren and John
data point time (sec) height (mm) uncertainty (mm)
0 0.0 180 3.5
1 0.5 182 4.5
2 1.0 178 4.0
3 1.5 165 5.5
4 2.0 160 2.5
5 2.5 148 3.0
6 3.0 136 2.5
```

```
import numpy as np
mydata = np.loadtxt("mydata.txt", skiprows=4)

dataPt, time, height, error = np.loadtxt("mydata.txt", skiprows=4, unpack=True)
dataPt, height = np.loadtxt("mydata.txt", skiprows=4, usecols=(0,2), unpack=True)
```

Read data from CSV file using NumPy

- Use NumPy's `loadtxt` to load data from a plain text file having the same number of values in each row:

biostats.csv:

```
Name,"Sex","Age","Height(in)","Weight(lbs)"
Alex,"M",41,74,170
Bert,"M",42,68,166
Elly,"F",30,66,124
Fran,"F",33,66,115
```

```
import numpy as np
mydata = np.loadtxt("biostats.csv", skiprows=1, usecols=(2,3,4), delimiter=',')
Age, Height, Weight = np.loadtxt("biostats.csv", skiprows=1, usecols=(2,3,4), delimiter=',', unpack=True)
```

Write data to text or CSV file using NumPy

- Use NumPy's `savetxt` to save data to a plain text or CSV file:

```
import numpy as np
data = np.random.rand(3, 5)
data[:,2] = data[:,2] * 100
data[2,:] = data[2,:] * 10
comments = "This is a group of random data\n"
comments += "Created on Thu Apr 22 2021"
np.savetxt("random.txt", data, header=comments, fmt="%12.5e")
np.savetxt("random.csv", data, header=comments, fmt="%12.5e",
delimiter=',')
```

Python `os` and `os.path` modules

- Python `os` module provides a portable way of using operating system dependent functionality:

```
import os
```

Method	Description
`os.chdir(path)`	change the current working directory to `path`
`os.getcwd()`	return a string representing the current working directory
`os.mkdir(path)`	create the directory named `path`
`os.remove(path)`	remove (deletes) the file `path`
`os.rmdir(path)`	remove (deletes) the directory `path`
`os.rename(old_name, new_name)`	rename the file from `old_name` to `new_name`
`os.listdir(path='.')`	return a list containing the names of the entries in the directory given by path

Python os and os.path modules


```
import os.path
```

Method	Description
os.path.join(path, *paths)	join one or more path components
os.path.exists(path)	return True if path refers to an existing path
os.path.isfile(path)	return True if path is an existing regular file
os.path.isdir(path)	return True if path is an existing directory
os.path.getmtime(path)	return the time of last modification of path
os.path.abspath(path)	return a normalized absolutized version of the pathname path
os.path.isabs(path)	return True if path is an absolute pathname
os.path.relpath(path, start=os.curdir)	returns a relative filepath to path either from current directory or an optional start directory
os.path.split(path)	split path into a pair, (head, tail) where the tail is the last pathname component and the head is everything leading up to that
os.path.getsize(path)	return the size, in bytes, of path

Examples


```
>>> import os
>>> os.getcwd()
'C:\\Users\\kegao'
>>> os.chdir('Desktop')
>>> os.getcwd()
'C:\\Users\\kegao\\Desktop'
>>> os.mkdir('Data_Science')
>>> os.chdir('Data_science')
>>> os.getcwd()
'C:\\Users\\kegao\\Desktop\\Data_science'
>>> os.mkdir("Machine_Learning")
>>> os.listdir('.')
['Machine_Learning']
>>> os.rmdir("Machine_Learning")
>>> os.listdir('.')
[]
```

Examples


```
>>> os.path.join("C:\\Users\\kegao\\Desktop", "Data_Science")
'C:\\Users\\kegao\\Desktop\\Data_Science'
>>> os.listdir('.')
['Big_Data.docx', 'Deep_Learning.txt']
>>> os.path.abspath("Big_Data.docx")
'C:\\Users\\kegao\\Desktop\\Data_science\\Big_Data.docx'
>>> os.path.getsize("Big_Data.docx")
12547
>>> os.path.split("C:\\Users\\kegao\\Desktop\\Data_science\\Big_Data.docx")
('C:\\Users\\kegao\\Desktop\\Data_science', 'Big_Data.docx')
>>> os.path.splittext("C:\\Users\\kegao\\Desktop\\Data_science\\Big_Data.docx")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'ntpath' has no attribute 'splittext'
>>> os.path.splitext("C:\\Users\\kegao\\Desktop\\Data_science\\Big_Data.docx")
('C:\\Users\\kegao\\Desktop\\Data_science\\Big_Data', '.docx')
>>> os.path.basename("C:\\Users\\kegao\\Desktop\\Data_science\\Big_Data.docx")
'Big_Data.docx'
>>> os.path.basename("C:\\Users\\kegao\\Desktop\\Data_science")
'Data_science'
>>> os.path.dirname("C:\\Users\\kegao\\Desktop\\Data_science\\Big_Data.docx")
'C:\\Users\\kegao\\Desktop\\Data_science'
>>> os.path.relpath("C:\\Users\\kegao\\Desktop\\Data_science\\Big_Data.docx")
'Big_Data.docx'
```

Exercises

Exercise 1

- Write a program to remove the comment character in the Python file (*simple_code.py*) containing the following contents:

simple_code.py:

```
print("This is a sample program")
#print("Python is a very versatile language")
```

Hint: you may use the `.replace()` method for string

Exercise 2

- Write a program to reverse alphabets in each word in the following text file (*some_text.txt*), and print each line:

some_text.txt:

```
Explicit is better than implicit
Simple is better than complex
```

Hint: you may use the `.rstrip()` method to remove ending `"\n"`

Exercise 3

- Write a program to find the longest word in a file (*some_text.txt*). Get the file name from user:

Exercise 4

- FASTA format: A sequence record in a FASTA format consists of a single-line description (sequence name), followed by line(s) of sequence data. The first character of the description line is a greater-than (">") symbol. A FASTA format can hold multiple sequence records. Here is a FASTA format example for protein sequences:

```
>seq0
FQTWEEFSRAAEKLYLADPMKVRVVLKYRHVDGNLCIKVTDDLVCLVYRTDQAQDVKKIEKF
>seq1
KYRTWEEFTRAAEKLYQADPMKVRVVLKYRHCDGNLCIKVTDDVVCLLYRTDQAQDVKKIEKFHSQLMRLME LKVTDNKECLKFKTDQAQEAKKMEKLNNIFFTLM
>seq2
EEYQTWEEFARAAEKLYLTDPMKVRVVLKYRHCDGNLCMKVTDDAVCLQYKTDQAQDVKKVEKLHGK
>seq3
MYQVWEEFSRAVEKLYLTDPMKVRVVLKYRHCDGNLCIKVTDNSVCLQYKTDQAQDVK
>seq4
EEFSRAVEKLYLTDPMKVRVVLKYRHCDGNLCIKVTDNSVVSYEMRLFGVQKDNFALEHSLL
>seq5
SWEEFAKAAEVLYLEDPMKCRMCTKYRHVDHKLVVKLTDNHTVLKYVTDMAQDVKKIEKLTTLLMR
>seq6
FTNWEEFAKAAERLHSANPEKCRFVTKYNHTKGELVLKLTDDVVCLQYSTNQLQDVKKLEKLSSTLLRSI
>seq7
SWEEFVERSVQLFRGDPNATRYVMKYRHCEGKLVLKVTDDRECLKFKTDQAQDAKKMEKLNNIFF
>seq8
SWDEFVDRSVQLFRADPESTRYVMKYRHCDGKLVLKVTDNKECLKFKTDQAQEAKKMEKLNNIFFTLM
>seq9
KNWEDFEIAAENMYMANPQNCRYTMKYVHSKGHILLKMSDNVKCVQYRAENMPDLKK
>seq10
FDSWDEFVSKSVELFRNHPDTTRYVVKYRHCEGKLVLKVTDNHECLKFKTDQAQDAKKMEK
```

- Write a program to calculate the mean sequence length of a FASTA file.

Exercise 5

- Write a program to count how many unique words in the following text file:

zen.txt

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Readability counts.
```

Hint: you may use the `.rstrip()` method to remove ending "`\n`", and use the `.replace()` method to replace the "`.`"character following the last word of each sentence.