More Lists, File Input, and Text Processing

01204111 Computers and Programming

Chaiporn Jaikaeo

Department of Computer Engineering
Kasetsart University

Outline

- Reading text files
- Creating lists from other sequences using list comprehensions
- Tabular data and nested lists

Task: *Text File Reader*

- Read lines from a specified text file and display them along with their line numbers
- Suppose there is a file named `data.txt` that contains two lines:

data.txt

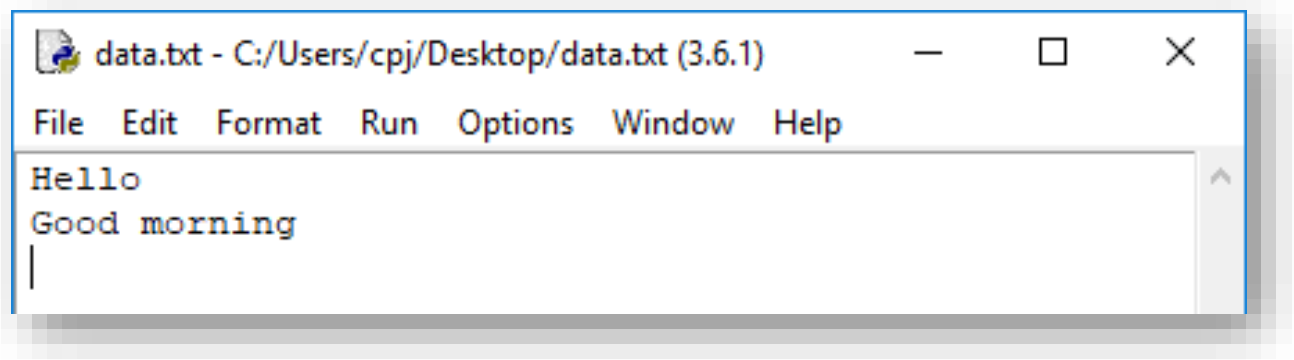
```
Hello
Good morning
```

- Then an example output of the program will be

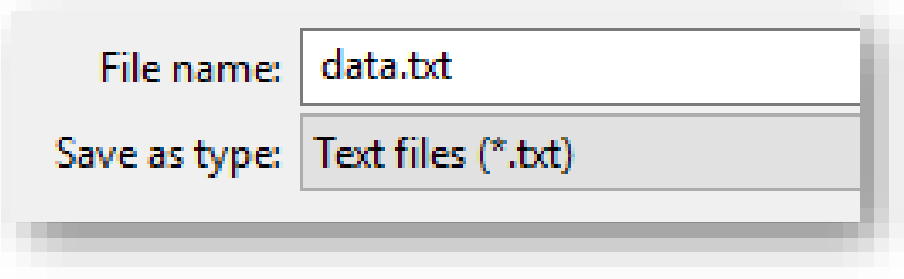
```
Enter file name: data.txt
Line 1: Hello
Line 2: Good morning
```

Creating a Text File

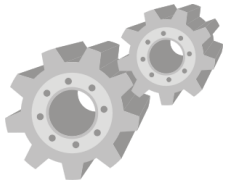
- A text file can be created using any text editor such as Notepad
- IDLE is also a text editor
  - Choose menu File → New File and start writing contents



  - Save a file with **`.txt` extension**, not `.py`



Reading File with Python



- Reading file's contents as a single string by combining `open()` function with `file.read()` method
  - Note that `open()` returns a `file` object, and `file.read()` returns a string

```
open(filename).read()
```

```
>>> s = open("data.txt").read()
>>> s
'Hello\nGood morning\n'
```

- Reading file's contents as a list of strings, one per line
  - Method `str.splitlines()` returns a list

```
open(filename).read().splitlines()
```

```
>>> lines = open("data.txt").read().splitlines()
>>> lines
['Hello', 'Good morning']
```

Trivia: *Functions vs. Methods*

- A **method** is a **function** bound to an object
- Functions are called by just their names (e.g., `len()`, `sum()`)

```
>>> len
<built-in function len>
>>> len("abc")
3
```

- Methods are called with their names and objects they are bound to (e.g., *`str`*`.split()`, where *`str`* is replaced by a string)

```
>>> s = "Hello, World"
>>> s.split
<built-in method split of str object at 0x109665e70>
>>> s.split(",")
['Hello', ' World']
```

Text File Reader – Program

- Our program reads a file as a list of strings, then traverse the list to print out each line

```
filename = input("Enter file name: ")
lines = open(filename).read().splitlines()
for i in range(len(lines)):
    print(f"Line {i+1}: {lines[i]}")
```

```
Enter file name: data.txt
Line 1: Hello
Line 2: Good morning
```

data.txt

```
Hello
Good morning
```

Trivia – *File Location Matters*

- If the text file is located in **the same folder** as the program
  - Just type the file name, i.e., `data.txt`
- If not, the entire path name of the file must be used, e.g., `C:\Users\user\Desktop\data.txt`
  - Windows:
    - Click a file icon in Explorer
    - Press **Ctrl-C**
    - Back to IDLE and press **Ctrl-V**
  - macOS:
    - Click a file icon in Finder
    - Press **Alt-Command-C**
    - Back to IDLE and press **Command-V**

```
Enter file name: data.txt
Line 1: Hello
Line 2: Good morning
```

Trivia – *Files should be closed*

- Opened files should be properly closed
  - Files in the examples are closed automatically in most Python environments
  - In real applications, you should explicitly close a file
- Two common methods: using the `with` statement or the `close()` method

```
with open("file.txt") as f:
    for line in f.readlines():
        # process the lines
```

file is closed automatically when exiting the `with` block

```
f = open("file.txt")
for line in f.readlines():
    # process the lines
f.close()
```

file is closed manually

We won't use them in this course ☺

Task: *Score Ranking*

- Read a file containing a list of *scores*
- Then sort the scores from highest to lowest and print out the ranking

```
Enter score file: scores.txt
Rank #1: 97.5
Rank #2: 87.3
Rank #3: 75.6
Rank #4: 63.0
Rank #5: 37.6
```

scores.txt

```
87.3
75.6
63.0
97.5
37.6
```

Score Ranking – Ideas

- Scores must be read as *a list of numbers*, not strings
- Each string member must get converted into a number

lines	"87.3"	"75.6"	"63.0"	"97.5"	"37.6"

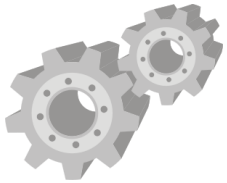
float()

scores	87.3	75.6	63.0	97.5	37.6

- Straightforward code with a `for` loop:

```
:
lines = open(filename).read().splitlines()
scores = []
for x in lines:
    scores.append(float(x))
:
```

List Comprehensions



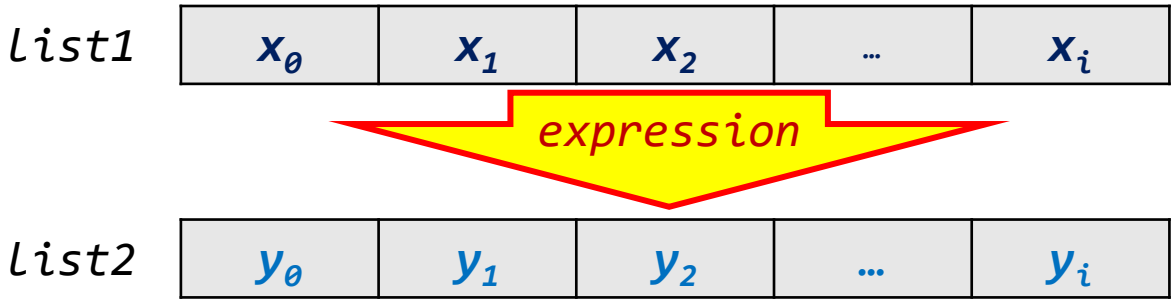
- **List comprehensions** are a powerful and concise way to create new lists from other sequences

```
list2 = [ expression for item in list1 ]
```

Similar to a set notation in mathematics, e.g., $S = \{2x \mid x = 1,2,3\}$

- It behaves exactly like

```
list2 = []
for item in list1:
    list2.append(expression)
```



Examples: *List Comprehensions*

- Create a new list with all values doubled from another list

```
>>> L1 = [5,1,2,8,9,12,16]
>>> L2 = [2*x for x in L1]
>>> L2
[10, 2, 4, 16, 18, 24, 32]
```

- Create a list of squares of *n*, where *n* = 1,2,…,10
  - A *range* object can be used directly inside a list comprehension

```
>>> [i**2 for i in range(1,11)]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- Compute the sum of squares of *n*, where *n* = 1,2,…,10

```
>>> sum([i**2 for i in range(1,11)])
385
```

Score Ranking – Ideas

- With a list comprehension, the code

```
scores = []
for x in lines:
    scores.append(float(x))
```

can be replaced by a much more concise statement:

```
scores = [float(x) for x in lines]
```

Score Ranking – Program

```
filename = input("Enter score file: ")
lines = open(filename).read().splitlines()
scores = [float(x) for x in lines]
scores.sort(reverse=True)
for i in range(len(scores)):
    print(f"Rank #{i+1}: {scores[i]}")
```

Sort the scores from highest to lowest

```
Enter score file: scores.txt
Rank #1: 97.5
Rank #2: 87.3
Rank #3: 75.6
Rank #4: 63.0
Rank #5: 37.6
```

scores.txt

```
87.3
75.6
63.0
97.5
37.6
```

Caveats – *Empty Lines in File*

- Empty lines in the input file will break the program

```
Enter score file: scores.txt
Traceback (most recent call last):
  File "score-rank.py", line 3, in <module>
    scores = [float(x) for x in lines]
  File "score-rank.py", line 3, in <listcomp>
    scores = [float(x) for x in lines]
ValueError: could not convert string to float:
```

scores.txt

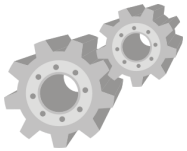
```
87.3
75.6

63.0
97.5
37.6
```

empty line

empty line

- We must <u>filter out</u> those empty lines before converting them to floats

Conditional List Comprehensions



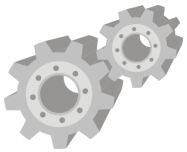
- Only certain members in the original list are selected to be included in the new list using `if` keyword

```
list2 = [ expression for item in list1 if condition]
```

- The above is similar to

```
list2 = []
for item in list1:
    if condition:
        list2.append(expression)
```

Examples:
Conditional List Comprehensions



- Split numbers into odd and even sets of numbers

```
>>> L = [5,1,2,8,9,12,16]
>>> odd = [x for x in L if x%2 == 1]
>>> even = [x for x in L if x%2 == 0]
>>> odd
[5, 1, 9]
>>> even
[2, 8, 12, 16]
```

- Create a list of positive integers less than 100 that are divisible by 8 but not divisible by 6

```
>>> [x for x in range(1,100) if x%8 == 0 and x%6 != 0]
[8, 16, 32, 40, 56, 64, 80, 88]
```

Score Ranking – Revised Program

- This version skips empty lines in the input file

```
filename = input("Enter score file: ")
lines = open(filename).read().splitlines()
scores = [float(x) for x in lines if x != ""]
scores.sort(reverse=True)
for i in range(len(scores)):
    print(f"Rank #{i+1}: {scores[i]}")
```

This condition helps skip empty lines

```
Enter score file: scores.txt
Rank #1: 97.5
Rank #2: 87.3
Rank #3: 75.6
Rank #4: 63.0
Rank #5: 37.6
```

scores.txt

```
87.3
75.6

63.0
97.5
37.6
```

Challenge – *Top-Three Ranking*

- Modify the program so that it always outputs only the top three ranks

```
Enter score file: scores.txt
Rank #1: 97.5
Rank #2: 87.3
Rank #3: 75.6
```

scores.txt

```
87.3
75.6

63.0
97.5
37.6
```

Tabular Data

- Most real-world data are often available in tabular form
  - For example, this is a snapshot of *household income statistics by year* available at http://data.go.th

										บาท Baht
2541	2543	2545	2547	2549	2550	2552	2554	2556	2558	Region and province
(1998)	(2000)	(2002)	(2004)	(2006)	(2007)	(2009)	(2011)	(2013)	(2015)	
12,492	12,150	13,736	14,963	17,787	18,660	20,904	23,236	25,194	26,915	Whole Kingdom
24,929	25,242	28,239	28,135	33,088	35,007	37,732	41,631	43,058	41,002	Greater Bangkok
26,054	26,909	29,589	29,843	36,658	39,020	42,380	48,951	49,191	45,572	Bangkok
18,100	15,745	19,680	19,946	20,382	21,302	23,359	23,798	29,575	25,457	Samut Prakan
24,211	24,566	29,119	26,658	31,152	32,743	34,626	35,120	30,664	36,884	Nonthaburi
21,793	19,282	22,838	21,530	25,143	26,107	26,686	21,616	33,461	41,057	Pathum Thani
12,643	13,012	14,128	16,355	19,279	18,932	20,960	20,822	26,114	26,601	Central Region
12,918	14,904	13,319	14,980	19,676	21,676	25,820	22,302	26,482	28,379	Phra Nakhon Si Ayutthaya
10,878	12,544	11,653	12,855	18,300	17,704	25,506	21,140	28,641	23,351	Ang Thong
10,587	10,649	11,010	15,003	19,935	16,852	22,405	17,178	23,426	22,955	Lop Buri

CSV Files

- **C**omma-**S**eparated **V**alues
- Commonly used to store tabular data as a text file
  - Each line is a row
  - Columns in each line (row) are separated by commas

rows

Subject	Credits	Grade
01175112	1	B+
01204111	3	A
01417167	3	B

columns

grades.txt

```
01175112,1,B+
01204111,3,A
01417167,3,B
```

- CSV files can be opened directly in Microsoft Excel

Task: *GPA Calculator*

- Read a CSV file containing a list of *subject codes*, their *credits*, and the *grades* received
- Then display *grade summary*, *total credits*, and *GPA*

```
Enter grade data file: grades.txt
-----------------------------------
  Subject    Credits  Grade  Point
-----------------------------------
  01175112     1       B+     3.5
  01204111     3       A      4.0
  01355112     3       C+     2.5
  01417167     3       B      3.0
-----------------------------------
Total credits = 10
GPA = 3.20
```

grades.txt

```
01175112,1,B+
01204111,3,A
01355112,3,C+
01417167,3,B
```

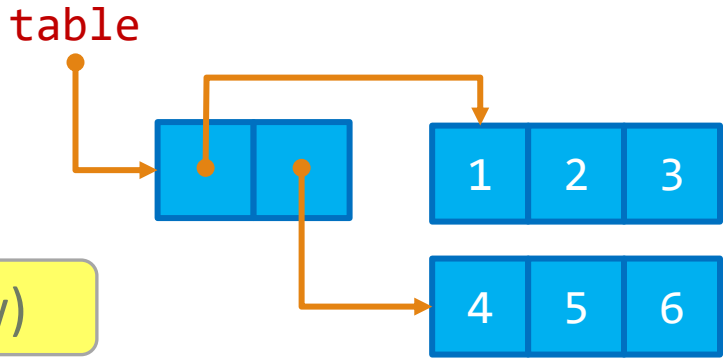
GPA Calculator – Ideas

- How to store tabular data in Python?
  - A table is a <u>list</u> of rows; each row is a <u>list</u> of columns
- We need a ***list of lists***
  - also known as a ***nested list***

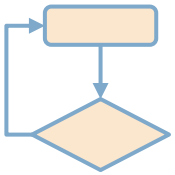
```
>>> table = [[1,2,3],[4,5,6]]
>>> len(table)
2
>>> table[1]
[4, 5, 6]
>>> table[1][2]
6
```

Access row#1 (2nd row)

Access column#2 (3rd column) in row#1 (2nd row)

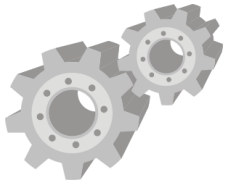


GPA Calculator – Steps



- Divide the whole task into three major steps
  - **Step 1:** read grade table data from file as a nested list
  - **Step 2:** display the grade table
  - **Step 3:** calculate total credits and GPA

Breaking Lines into Columns



- Python provides `str.split()` method

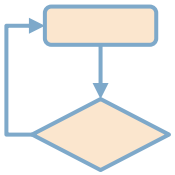
```
>>> line = "01204111,3,A"
>>> line.split(",")
['01204111', '3', 'A']
```

- Let us try using it inside a list comprehension

```
>>> lines = open("grades.txt").read().splitlines()
>>> lines
['01175112,1,B+', '01204111,3,A', '01355112,3,C+', '01417167,3,B']
>>> table = [x.split(",") for x in lines]
>>> table
[['01175112', '1', 'B+'], ['01204111', '3', 'A'], ['01355112',
'3', 'C+'], ['01417167', '3', 'B']]
```

We now got a nested list!

GPA Calculator – Steps



Step 1 - read grade table from file as a nested list

- We will define `read_table()` function as follows

```
def read_table(filename):
    lines = open(filename).read().splitlines()
    table = [x.split(",") for x in lines if x != ""]
    return table
```

grades.txt

```
01175112,1,B+
01204111,3,A
01355112,3,C+
01417167,3,B
```

- Let's test it

```
>>> read_table("grades.txt")
[['01175112', '1', 'B+'], ['01204111', '3', 'A'], ['01355112',
'3', 'C+'], ['01417167', '3', 'B']]
```

GPA Calculator – Steps

- The resulting table is not complete

```
>>> read_table("grades.txt")
[['01175112', '1', 'B+'], ['01204111', '3', 'A'],
['01355112', '3', 'C+'], ['01417167', '3', 'B']]
```

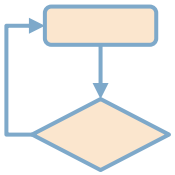
grades.txt

```
01175112,1,B+
01204111,3,A
01355112,3,C+
01417167,3,B
```

- Output on the right is what we expect to get in the end
  - The *credits* column should store integers, not strings, for later calculation
  - The *point* column is still missing

```
Enter grade data file: grades.txt
-----------------------------------------
  Subject    Credits   Grade   Point
-----------------------------------------
  01175112     1        B+     3.5
  01204111     3        A      4.0
  01355112     3        C+     2.5
  01417167     3        B      3.0
-----------------------------------------
Total credits = 10
GPA = 3.20
```

GPA Calculator – Steps



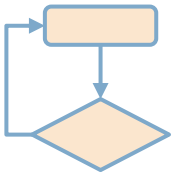
- We will traverse the `table` list to perform adjustment on each row
  - We also define `grade_point()` function to map a grade to a point

```
def read_table(filename):
    lines = open(filename).read().splitlines()
    table = [x.split(",") for x in lines if x != ""]
    for row in table:
        # convert credits to integers
        row[1] = int(row[1])
        # add a new column for grade point
        row.append(grade_point(row[2]))
    return table
```

```
>>> table = read_table("grades.txt")
>>> table
[['01175112', 1, 'B+', 3.5], ['01204111', 3,
'A', 4.0], ['01355112', 3, 'C+', 2.5],
['01417167', 3, 'B', 3.0]]
```

```
def grade_point(grade):
    if grade == "A":
        return 4.0
    elif grade == "B+":
        return 3.5
    elif grade == "B":
        return 3.0
    elif grade == "C+":
        return 2.5
    elif grade == "C":
        return 2.0
    elif grade == "D+":
        return 1.5
    elif grade == "D":
        return 1.0
    elif grade == "F":
        return 0.0
```

GPA Calculator – Steps



Step 2 - display the grade table

- Traverse the table list and print out each row

```
def print_table(table):
    print("-----------------------------------")
    print("  Subject   Credits  Grade  Point")
    print("-----------------------------------")
    for row in table:
        print(f"  {row[0]:8} {row[1]:5}       {row[2]:<5} {row[3]:.1f}")
    print("-----------------------------------")
```

```
>>> print_table(table)   # table from previous step
-----------------------------------
  Subject   Credits  Grade  Point
-----------------------------------
  01175112     1       B+    3.5
  01204111     3       A     4.0
  01355112     3       C+    2.5
  01417167     3       B     3.0
-----------------------------------
```

Not so difficult, but a lot of tweaking to get a nice-looking table

GPA Calculator – Steps

Step 3 - calculate total credits and GPA

- Total of credits is computed from the summation of column#1 in all rows

```
total_credits = sum([row[1] for row in table])
```

```
>>> [row[1] for row in table]
[1, 3, 3, 3]
```

```
>>> table
[['01175112', 1, 'B+', 3.5],
['01204111', 3, 'A', 4.0],
['01355112', 3, 'C+', 2.5],
['01417167', 3, 'B', 3.0]]
```

GPA Calculator – Steps

Step 3 - calculate total credits and GPA

- GPA is computed from the summation of credits*point of all subjects
  - credits → column#1, point → column#3

```
>>> table
[['01175112', 1, 'B+', 3.5],
['01204111', 3, 'A', 4.0],
['01355112', 3, 'C+', 2.5],
['01417167', 3, 'B', 3.0]]
```

```
>>> [row[1]*row[3] for row in table]
[3.5, 12.0, 7.5, 9.0]
```

```
sum_credits_point = sum([row[1]*row[3] for row in table])
gpa = sum_credits_point/total_credits
```

GPA Calculator – Main Program

- `read_table()` and `print_table()` are not shown

```
filename = input("Enter grade data file: ")
table = read_table(filename)
print_table(table)
total_credits = sum([row[1] for row in table])
sum_credits_point = sum([row[1]*row[3] for row in table])
gpa = sum_credits_point/total_credits
print(f"Total credits = {total_credits}")
print(f"GPA = {gpa:.2f}")
```

grades.txt

```
01175112,1,B+
01204111,3,A
01355112,3,C+
01417167,3,B
```

```
Enter grade data file: grades.txt
-----------------------------------
  Subject   Credits  Grade  Point
-----------------------------------
  01175112     1       B+    3.5
  01204111     3       A     4.0
  01355112     3       C+    2.5
  01417167     3       B     3.0
-----------------------------------
Total credits = 10
GPA = 3.20
```

Notes: *Why Subroutines?*

- Most examples in this course could be written without using subroutines at all
  - That would also result in a bit shorter programs
- However, breaking a task into subroutines
  - helps focus on smaller, more manageable problems (i.e., separation of concerns),
  - makes programs easier to read, test, and find bugs, and
  - makes it easier to divide tasks among team members

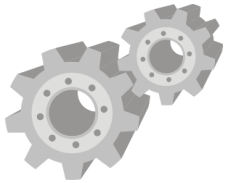
Conclusion

- Data can be read into a program from a **text file** instead of being entered by hand
  - Saves time and reduces user error
- **List comprehensions** help create new lists in an expressive and concise way
- **Tabular data** can be represented in Python as a **nested list**

References

- **Python Language for Grades 10-12 (in Thai).** The Institute for the Promotion of Teaching Science and Technology (ISPT).
- List comprehensions
  - https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions
- How to read a file with Python
  - https://www.webucator.com/how-to/how-read-file-with-python.cfm

Syntax Summary (1)



- Open a file and read its contents as a single string

```
open(filename).read()
```

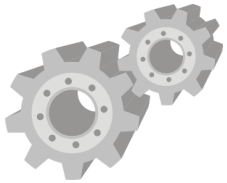
- Open a file and read its contents as a list of strings, one string per line

```
open(filename).read().splitlines()
```

- Split a string *s* into a list of strings using the specified *delimiter*

```
s.split(delimiter)
```

Syntax Summary (2)



- Create a list using a list comprehension

```
[expression for item in list]
```

- Create a list using a conditional list comprehension

```
[expression for item in list if condition]
```

Revision History

- September 2016 – Intiraporn Mulasatra (int@ku.ac.th)
  - Prepared slides for files and sorting in C#
- October 2017 – Chaiporn Jaikaeo (chaiporn.j@ku.ac.th)
  - Revised for Python