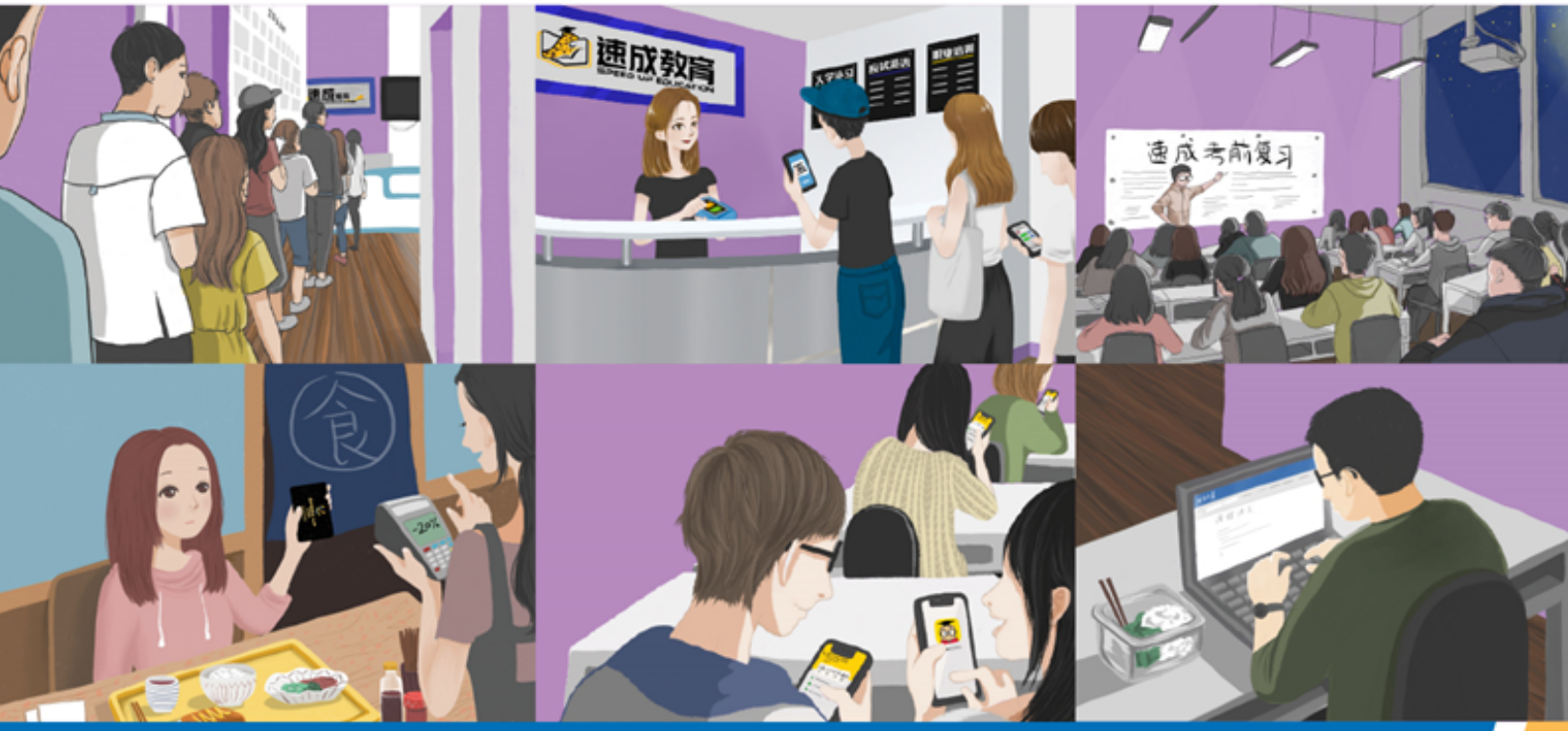
速成教育
SPEED UP EDUCATION



CSCA08

Introduction of Programming I

导师： **Vincent**

UTSC Read & Write File | 2025/3/7

Disclaimer

This study handout is provided by Speed Up Education and its affiliated tutors. The study handout seeks to support your study process and should be used as a complement, not substitute to the university course material, lecture notes, problem, sets, past tests and other available resources.

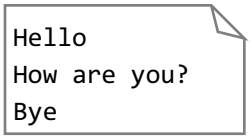
The study handout is distributed for free to the students participating in Speed Up Education classes and is not for sale or other commercial uses whatsoever. We kindly ask you to refrain from copying or selling in part or in whole any information provided in the handout.

Thanks for choosing Speed Up Education. We wish you the best of luck in your exam.

File (TextIO)

Reading Files

Lets say we have a file called `SpeedUp.txt` and we want to read it and print it.



SpeedUp.txt

```
>>> Hello
    How are you?
    Bye
```

Method 1 – `read()` 從 f 指到的地方 讀完整個文件 as a string

```
f = open("SpeedUp.txt", "r")
lines = f.read()       -> "Hello\nHow are you?\nBye"
print(lines)
f.close()
```

Method 2 – `readlines()` 從 f 指到的地方 讀完整個文件 as a list

```
f = open("SpeedUp.txt", "r")
lines = f.readlines()  -> ["Hello\n", "How are you?\n", "Bye"]
for line in lines:
    print(lines.strip("\n"))
f.close()
```

Method 3 – `readline()` 讀 f 指到的那一行 讀完, f 指到下一行

```
f = open("SpeedUp.txt", "r")
line = f.readline()    -> "Hello\n"
while line != "":
    print(line.strip("\n"))
    line = f.readline()-> "How are you?\n" -> "Bye" -> ""
f.close()
```

Method 4 – `for` loop 讀 f 指到的那一行開始, 一次讀一行

```
f = open("SpeedUp.txt", "r")
for line in f:         -> "Hello\n" -> "How are you?\n" -> "Bye"
    print(line.strip("\n"))
f.close()
```

Example – what is the output of the following code?

```
def print_file(f):
    for line in f:
        print(line.strip() + "!")

if __name__ == "__main__":
    f = open("SpeedUp.txt", "r")
    f.readline()
    print_file(f)
    f.close()
```

```
>>>
```

Writing files

```
f = open("SpeedUp.txt", "w")
f.write("I")
f.write("hate")
f.write("csc108")
f.close()
```

```
f = open("SpeedUp.txt", "w")
f.write("I\n")
f.write("hate\n")
f.write("csc108\n")
f.close()
```

```
f = open("SpeedUp.txt", "w")
f.write("I\n")
f.write("hate\n")
f.write("csc108\n")
f.close()
f = open("SpeedUp.txt", "w")
f.write("Hello")
f.close()
```

```
f = open("SpeedUp.txt", "w")
f.write("Hello\n")
f.close()
f = open("SpeedUp.txt", "a")
f.write("Bye")
f.close()
```

Example

```
Roses are red,
Violets are blue.
I love CSC108!
How about you?
```

```
mario = open('my_file.txt', 'r')
jennifer = mario.readline().strip()
joseph = mario.readlines()[0]
jonathan = mario.read()
```

What does the variable `jennifer` refer to?

(A) The empty string: `''`
(B) The string: `'Roses are red,\n'`
(C) The string: `'Roses are red,'`
(D) The string: `'Violets are blue.'`

What does the variable `joseph` refer to?

(A) The string: `'Roses are red,\n'`
(B) The string: `'Violets are blue.\n'`
(C) The string: `'How about you?\n'`
(D) The list: `['Violets are blue.\n', 'I love CSC108!\n', 'How about you?\n']`
(E) The list: `['Roses are red,\n', 'Violets are blue.\n', 'I love CSC108!\n', 'How about you?\n']`

What does the variable `jonathan` refer to?

(A) An empty string: `''`
(B) The string: `'\n'`
(C) The string: `'How about you?\n'`
(D) An empty list: `[]`

Example

```
def func(filename: str) -> None:
    f = open(filename, 'r')
    for line in f:
        line = f.readline().strip()
        print(line)
        f.readline()
    f.close()
```

It is given that the function call **func('example.txt')** prints 10,000 lines.
How many lines are there in **example.txt**?

Example

```
def func2(s: str) -> None:
    """ String s consists of ONLY a's and/or w's """
    for c in s:
        f = open('file.txt', c)
        f.write(s)
        f.close()
```

Assume ***file.txt*** is initially empty. which of the following CANNOT be the content of ***file.txt*** after the function is called once?

a) wawawawa b) aaaa c) wawa d) wwwww e) awwwwa

Example

In a play, the lines each character speaks have the following format:

```
[CHARACTER_NAME] first line of dialog
    additional lines of dialog (0 or more)
```

A file representing a play will contain character dialog formatted as above.
Here is an example from Shakespeare's *Antony and Cleopatra*:

```
[CLEOPATRA] I am sick and sullen.
[MARK ANTONY] I am sorry
    to give breathing
    to my purpose,--
[CLEOPATRA] Help me away,
    dear Charmian; I shall fall:
    It cannot be thus long,
    the sides of nature
    Will not sustain it.
[MARK ANTONY] Now,
    my dearest queen,--
[CLEOPATRA] Pray you,
    stand further from me.
[MARK ANTONY] What's the matter?
```

In this question, you will write a function that reads a file containing lines from a play in this format. Some things you can assume:

- Square brackets are never used except to mark a `CHARACTER_NAME`.
- `CHARACTER_NAME`s may be any case, but will have consistent case throughout the file.
- There are no blank lines.
- There will always be a non-empty first line of dialog.

On the following page, complete the body of the function according to its docstring. Your solution should be general, and work for any play file that has the format described above. You must use the existing starter code in your solution.

Assume the `play.txt` file referred to in the docstring example is the example file above.

```
def read_lines(play: TextIO, character: str) -> list[str]:
    """ Return the list of dialogs (with all newlines removed)
    made by character in play.

    >>> file = open('play.txt')
    >>> actual = read_lines(file, 'MARK ANTONY')
    >>> expected = ['I am sorry to give breathing to my purpose,--',
    ...             'Now, my dearest queen,--',
    ...             'What's the matter?']
    >>> actual == expected
    True
    >>> file.close()
    """
```

```
def count_odds_from_file(number_file: TextIO) -> List[int]:
    """Return a list of counts of odd numbers in each section of the file
    number_file. Each section in number_file begins with a line that contains
    only START and ends with a line that contains only END.

    Preconditions: each line in number_file is either START, END, or a string
    of digits that can be converted to an int.
    number_file is properly formatted to contain 0 or more sections structured:
    START
    <0 or more lines of numbers>
    END

    >>> f = open('sample_numbers.txt')
    >>> count_odds_from_file(f)
    [1, 0, 2]
    >>> f.close()
    """
```

```
START
1
2
END
START
4
END
START
5
6
7
END
```

Example

Vincent built some tools to work with grade files. The files consist of a name, a course and a grade separated by commas, one grade per line. After the grade data is a line starting with `---` and then other data. A sample file might look something like the following:

```
Alice,CSCA08,99
Bob,CSCA08,70
Alice,MATA31,95
Alice,CSCA48,85
Carol,ABCA01,60
Bob,CSCA48,50
---
This file is private and confidential...
```

Vincent wrote a function called `build_marks_dict` that reads a grade file and turns it into a dictionary that maps student names to dictionaries mapping courses to grades. A sample dictionary of that type might look something like:

```
{'Alice': {'CSCA08': 99.0, 'MATA31': 95.0, 'CSCA48': 85.0},
 'Bob': {'CSCA08': 70.0, 'CSCA48': 50.0},
 'Carol': {'ABCA01': 60.0}
}
```

He then wrote another function called `calculate_averages` that takes a dictionary formatted like the one above, and produces a dictionary mapping course codes to averages. A sample dictionary of that type might look something like:

```
{'CSCA08': 84.5, 'MATA31': 95.0, 'CSCA48': 67.5, 'ABCA01': 60.0}
```

Vinceent also wrote some global code to test the functions.

Vincent had the functions completed and tested. But then... disaster struck! Alice get holds onto his computer and delete them all. Please help Vincent to re-implement the codes ☺

```
def build_marks_dict(input_file):
```

```
def calculate_averages(student_to_marks):
```

Example

Design and Implementation

In this question we will write a program that reads a transcript of a chat session, collects the data in a dictionary, and processes this data.

The question will guide you through the development of several functions that, together, will accomplish this task. You can solve each part in this question independently of all other parts. For example, you can solve Part(b) even if you have not solved Part(a), etc.

The file format is guaranteed to be as follows:

The first line (if the file is not empty) contains the special text `[user_messages]`. A line that follows the line with `[user_messages]` always contains a username (and nothing else). The next one or more lines are messages from this user. Each message appears on its own line. A message is never empty. There is always at least one message line following a line with a username. After the last message in the current series, there is a special text `[\user_messages],` on its own line.

The next sequence of messages, again, starts with `[user_messages]` line followed by a username line. This username can be new to the chat, or could have already messaged before. And so on.

We will write a function `make_user_to_msgs` that reads a file in the above format and creates a dictionary that maps usernames to list of their messages. For example, a file on the left will produce the dictionary on the right below:

```
[user_messages]
Anya
Hello, everyone!
[/user_messages]
[user_messages]
Jason
Hi, Anya!
[/user_messages]
[user_messages]
Anya
CSCA08 is my favourite course.
What is your favourite course?
[/user_messages]
[user_messages]
Kaveh
It is certainly my favourite course!
[/user_messages]
[user_messages]
Jason
I like both my courses equally.
CSCA08 and MATA31.
[/user_messages]
```

```
{'anya': ['Hello, everyone!',
    'CSCA08 is my favourite course.',
    'What is your favourite course?'],
'Jason': ['Hi, Anya!',
    'I like both my courses equally.',
    'CSCA08 and MATA31.'],
'kaveh': ['It is certainly my favourite course!']}
```

(In the docstrings for our functions, we will assume that this example dictionary is referred to by the constant DATA.)

We will begin by defining two functions that will become our helper functions for the main tasks. Complete the following implementations.

Part(a)

```
def make_word(text: str) -> str:
    '''Return text, converted to lowercase, and with all leading and
    trailing non-alphanumeric characters removed.

    >>> make_word(’Anya123')
    'anya123'
    >>> make_word(’$100.50?!')
    '100.50'
    >>> make_word('@?!...')
    ''
    '''
```

Part(b) Reading from Files

```
SOM = '[user_messages]'
EOM = '[/user_messages]'

def get_user_messages(msg_file: TextIO) -> List[str]:
    '''Read msg_file from the current position, and return a list of
    messages of the "current" user. Stop reading at the end of the file or
    after reading the next occurrence of EOM, whichever occurs first.

    Precondition: the current position is a first message line inside some
    user_messages block in msg_file.

    For example, if the current position in msg_file is the beginning
    of the line "CSCA08 is my favourite course.", then the return
    value is
    ['CSCA08 is my favourite course.', 'What is your favourite course?']

    '''

    messages = []

    line = msg_file.readline()

    while
```

Part(c) Reading from Files, Building Dictionaries, Using Helper Functions

We are now ready to implement the function make_user_to_msgs. Hint: use the function get_user_messages defined above (even if you have not implemented it)!

```
def make_user_to_msgs(msg_file: TextIO) -> Dict[str, List[str]]:
    '''Read msg_file, collect its contents into a dictionary that maps
    usernames (converted to lowercase) to lists of their messages, and
    return this dictionary.

    '''

    user_to_msgs = {}

    line = msg_file.readline()

    while
```

We will now implement two functions that process the data collected by make_user_to_msgs. Complete the following implementations.

Part(d) Working with Dictionaries: Basic

```
def get_msg_counts(data: Dict[str, List[str]] -> Dict[str, int]:
    '''Return a dictionary that maps each username to a total number of
     messages posted by that user, based on the information in data.

    >>> get_msg_counts(DATA) == {'anya': 3, 'jason': 3, 'kaveh': 1}
    True
    '''
```

Part(e) Working with Dictionaries and Lists, Using Helper Functions

Hints: use the function make_word defined above (even if you have not implemented it)!

```
def get_unique_words(data: Dict[str, List[str]]) -> Dict[str, List[str]]:
    '''Return a dictionary that maps each username to a list of all unique
    words from the messages posted by that user, in order, based on
    the information in data. Convert all words to lowercase, and strip
    all leading and trailing non_alphanumeric characters.

    >>> usr_to_words = {'anya': [  'hello', 'everyone', 'csca08', 'is', 'my',
    ...                            'favourite', 'course', 'which', 'your'],
    ...                'jason': [  'hi', 'anya', 'i', 'like', 'both', 'my',
    ...                            'courses', 'equally', 'csca08', 'and',
    ...                            'mata31'],
    ...                'kaveh': [  'it', 'is', 'certainly', 'my', 'favourite',
    ...                            'course']}
    >>> get_unique_words(DATA) == usr_to_words
    True
    '''
```