CS106A Summer 2025
Neel Kishnani

Final Exam

1. Snake Case	15
2. Merge Index	16
3. Most Popular Document	18
4. Sorting	8
5. Air Travel	20
6. Timing	16
	93

_____

Print name (**legibly**)

_____

Your `@stanford.edu` email

Exam instructions: There are 6 questions. Write all answers directly on the exam. This printed exam is **closed-book and closed-device**; you may refer only to your one letter-sized page of prepared notes and the provided reference sheet.

Python coding guidelines: Unless otherwise restricted in the instructions for a specific problem, you are free to use any of the libraries, functions, and data types we have learned in class. You don't need `import` statements in your solutions, just assume the required library files are available to you. You are free to create helper functions unless the problem states otherwise. Helper functions can be defined before or after they are used. Comments are not required, but when your code is incorrect, comments could clarify your intentions and help earn partial credit.

THE STANFORD UNIVERSITY HONOR CODE

A. The Honor Code is an undertaking of the students, individually and collectively:

(1) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;

(2) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

B. The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid as far as practicable, academic procedures that create temptations to violate the Honor Code.

C. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

_____(signature)

I accept and acknowledge the honor code.

Problem 1: Snake Case (15 points)

So far in this class, we have followed the convention that variables and function names have all lowercase characters and have underscores separating individual words. This convention is called `snake_case` and `can_be_used_in_normal_sentences` too :)

Another common naming convention (popular in the C++ language) is called camel case. In camel case, a sequence of words is structured as: the first word starts with a lowercase character and each subsequent word starts with an uppercase character. For example, the snake case string `hello_there_everyone` would be `helloThereEveryone` in camel case.

Your task is to write a function `make_snake_case` function which accepts a valid camel case string and returns its equivalent snake case form.

Here are some sample outputs of this function, expressed as Python doctests:

```
>>> make_snake_case("helloThere")
'hello_there'

"""
Notice that the snake case equivalent of a word
that has no uppercase characters is simply the
word itself.
"""
>>> make_snake_case("cs106a")
'cs106a'

>>> make_snake_case("cs106aIsAwesome")
'cs106a_is_awesome'
```

Some notes on this problem:

- To check if a letter is upper case, you can use `ch.isupper()`. For example, `"A".isupper()` returns `True` while `"v".isupper()` returns `False`.
- `isupper()` returns `False` for non-alphabetical characters.
- To convert a letter to lower case you can use `ch.lower()`. For example `"E".lower()` returns `"e"` and `"L".lower()` returns `"l"`.

_____ (Your SUID, e.g. 006892456, required on every page)

Please write your solution for Problem 1 here.

```
def make_snake_case(camel_case):
```

_____ (Your SUID, e.g. 006892456, required on every page)

Use this page as additional space for Problem 1, if needed.

This page intentionally left blank.

Problem 2: Merge Index (16 points)

In Assignment 5 you built a search engine using two core data structures: a dictionary (index) associating terms to a list of documents where the term appears, and a dictionary associating document name to title. These two structures could look like:

```
index = {
      "term1": ["doc1.txt", "doc2.txt"],
      "term2": ["doc3.txt", "doc4.txt"],
      "term3": ["doc2.txt", "doc4.txt"]
}

file_titles = {
      "doc1.txt": "title1",
      "doc2.txt": "title2",
      "doc3.txt": "title3",
      "doc4.txt": "title4",
}
```

Write a function **`merge_index`** that accepts these two dictionaries **`index`** and **`file_titles`** as parameters and returns a single dictionary that combines the two. This "merged" structure will associate terms in **`index`** to **another dictionary,** where this "inner dictionary" will associate document names with their title for each document in which the word appears.

Using the **`index`** and **`file_titles`** dictionaries above, **`merge_index(index, file_titles)`** would return:

```
{
      "term1": {
            "doc1.txt": "title1",
            "doc2.txt": "title2"
      },
      "term2": {
            "doc3.txt": "title3",
            "doc4.txt": "title4"
      }
      "term3": {
            "doc2.txt": "title2",
            "doc4.txt": "title4"
      }
}
```

You may assume that all of the documents references in **`index`** are valid keys within the **`file_titles`** dictionary.

_____ (Your SUID, e.g. 006892456, required on every page)

Please write your solution for Problem 2 here.

```
def merge_index(index, file_titles):
```

\

_____ (Your SUID, e.g. 006892456, required on every page)

Use this page as additional space for Problem 2, if needed.

This page intentionally left blank.

Problem 3: Most Popular Document (18 points)

Using the same index structure from Assignment 5 (and from Problem 2), find the most popular document in the index. We define the most popular document as the one that contains the most terms within the index.

Write a function **`most_popular_document(index)`** that accepts the dictionary **`index`** as a parameter and returns a **`string`** which is the name of the document that contains the most terms in **`index`**.

For example, if the **`index`** is:

```
{
	"term1": [
		"doc1.txt",
		"doc2.txt”
	],
	"term2": [
		"doc2.txt",
		"doc3.txt"
	]
}
```

`most_popular_document(index)` would return the string **`"doc2.txt"`**. This is because **`doc1.txt`** is only associated with one term: **`term1`**. **`doc3.txt`** is only associated with one term: **`term2`**. **`doc2.txt`** is associated with both **`term1`** and **`term2`**.

If there are multiple documents that would be the most popular, you may return any of them.

_____ (Your SUID, e.g. 006892456, required on every page)

Please write your solution to Problem 3 here:

```
def most_popular_document(index):
```

_____ (Your SUID, e.g. 006892456, required on every page)

Use this page as additional space for Problem 3 if needed.

This page intentionally left blank.

Problem 4: Sorting (8 points)

Given the following lists, provide the output of each code snippet.

```
foo = [-1, 5, 3, 9, 2, 0]
bar = sorted(foo)
baz = sorted(foo, reverse=True)
```

a. `print(bar[2:])`

b. `print(bar[1: len(bar) - 2])`

c. `print(baz[len(baz) - 3:])`

d. `print(baz[4:])`

Problem 5: Air Travel (20 points)

You are planning a fun trip after finishing CS106A! While planning your trip, you need to book your air strategically to not have too many stops. Your goal is to reach your destination with **at most 1 stop** during your travel.

Consider the following dictionary representing flights on the day you're traveling:

```
{
      "SFO": {
            "airport_city": "San Francisco",
            "airport_country": "USA",
            "flights": [
                  { "airport_code": "JFK", "departure_time": "14:35" },
                  { "airport_code": "LAX", "departure_time": "03:14" }
            ]
      },
      "JFK": {
            "airport_city": "New York",
            "airport_country": "USA",
            "flights": [
                  { "airport_code": "PNQ", "departure_time": "14:35" }
            ]
      },
      "PNQ": {
            "airport_city": "Pune",
            "airport_country": "India",
            "flights": [
                  { "airport_code": "NRT", "departure_time": "08:39" },
                  { "airport_code": "PVG", "departure_time": "23:11" }
            ]
      },
      "PVG": {
            "airport_city": "Shanghai",
            "airport_country": "China",
            "flights": [
                  { "airport_code": "IST", "departure_time": "10:02" }
            ]
      }
}
```

Notice that the keys in this dictionary are airport codes: SFO, JFK, etc. Each *inner dictionary* has the following keys: **`"airport_city"`**, **`"airport_country"`**, and **`"flights"`**. The **`"flights"`** key is associated with a list of dictionaries representing the flights one can take from this specific airport. These flight dictionaries have two keys **`"airport_code"`** and **`"departure_time"`**. For this problem, flight data will look like the dictionary above and will be written to a JSON file that you will use in your solution.

Write a function **`can_travel(filename, start, end)`** that returns either **`True`** if you can get from airport **`start`** to airport **`end`** with at most 1 stop or **`False`** if not. You do not have to consider if the **`"departure_time"`** for each flight when making this determination.

- **`filename`** is the name of the JSON file to read in with air travel data in the format described above.
- **`start`** is the airport code representing the beginning of your journey.
- **`end`** is the airport code representing the destination of your journey.

For example, if the data above were written to a file **`travel.json`**, calling **`can_travel("travel.json", "SFO", "PNQ")`** would return **`True`** since you can travel from **`SFO`** to **`JFK`** and then **`JFK`** to **`PNQ`** – just one stop!

`can_travel("travel.json", "SFO", "JFK")` would also return **`True`** since you can reach **`JFK`** directly from **`SFO`.**

`can_travel("travel.json", "SFO", "PVG")` would return **`False`** since you cannot reach **`PVG`** from **`SFO`** with just one stop.

Some notes on this problem:

- You may assume that **`start`** and **`end`** are valid keys (airport codes) within the dictionary represented by the JSON file.
- You may assume that all of the flight codes in the inner dictionaries are valid keys within the dictionary represented by the JSON file.

_____ (Your SUID, e.g. 006892456, required on every page)

Please write your solution for Problem 5 here.

```
def can_travel(filename, start, end):
```

_____ (Your SUID, e.g. 006892456, required on every page)

Use this page as additional space for Problem 5 if needed.

This page intentionally left blank.

Problem 6: Timing (16 points)

In this problem you will write and use a `Clock` class that represents a time using hours and minutes. Someone could use this class as follows:

```
time1 = Clock(14, 42) # 14:42 or 2:42PM
time2 = Clock(4, 13)  #  4:13 or 4:13AM
```

6a. Class Implementation (8 points)

Your first task is to write three methods for the `Clock` class:

1. The constructor `__init__` which takes in two integers as parameters: `hour` and `minute`. You may assume that these values are well-formed (i.e. you do not have to confirm that the values are valid hour and minute numbers).
2. `get_hour()` returns the hour value associated with the instance of the `Clock`.
3. `get_minute()` returns the minute value associated with the instance of the `Clock`.

Write your implementations for Problem 6a here:

```
class Clock:

    def __init__(self, hour, minute):




    def get_hour(self):




    def get_minute(self):
```

6b. Get Time in Minutes (4 points)

Write a new method **`get_time_in_minutes`** for the **`Clock`** class that returns the total number of minutes that have elapsed since the start of that day. For example:

```
time = Clock(2, 31) # 2:31 or 2:31AM

# This prints 151:
#    * Hour being 2 means that 120 minutes have elapsed
#    * Minute being 31 means we add 31 to the total
print(time.get_time_in_minutes())
```

Write your implementations for Problem 6b here:

```
def get_time_in_minutes(self):
```

6c. Minutes Between (4 points)

Write a function **`minutes_between`** that takes in two **`Clock`** objects and returns the total number of minutes between the two times represented. For example:

```
time1 = Clock(15, 0)  # 15:00 or 3PM
time2 = Clock(16, 20) # 16:20 or 4:20PM
mins = minutes_between(time1, time2)
print(mins) # Prints 80, 1 hour and 20 minutes between the two
```

You may assume that the time represented by **`time2`** is *later than* the time represented by **`time1`**.

Write your implementations for Problem 6c here:

```
def minutes_between(time1, time2):
```

This page intentionally left blank.

This page intentionally left blank.