CSC 108H5 S 2019 Term Test 2
Duration — 50 minutes
Aids allowed: none

Student Number: |_|_|_|_|_|_|_|_|_|_|_|_|_|

Last Name: _____ **First Name:** _____

---

*Do **not** turn this page until you have received the signal to start.*
(Please fill out the identification section above and read the instructions below.)
Good Luck!

This midterm consists of 3 questions on 8 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.*

- Comments are not required except where indicated, although they may help us mark your answers.
- No error checking is required: assume all user input and all argument values are valid.
- If you use any space for rough work, indicate clearly what you want marked.
- You may use a pencil; however, work written in pencil will not be considered for remarking.

1: _____/10

2: _____/ 6

3: _____/ 7

TOTAL: _____/23

Question 1. [10 marks]

Assume each of the pieces of code below is being entered into the Python console. Each subquestion is independent of the others.

In each space, write what would be displayed on the console, if anything. If a line of code would not cause anything to be displayed, leave the space blank.

If the code would cause an error, write ERROR, and provide a brief EXPLANATION.

Part (a) [1 mark]

```
>>> names = ["Bob", "Lisa"]
>>> names.append("Jim") + ["Dan", "Sally"]
>>> print(names)
```

Part (b) [1 mark]

```
>>> nums = [1, 2]
>>> nums.append(3)
>>> nums2 = nums + [4, 5]
>>> print(nums)



>>> print(nums2)
```

Part (c) [2 marks]

```
>>> L1 = [[1, 2], [3, 4]]
>>> for i in range(len(L1)):
          for j in range(len(L1[i])):
                print(L1[i][j])
```

```
>>> L2 = L1[:]
>>> L1.append(8)
>>> print(L1)



>>> print(L2)
```

Part (d) [2 marks]

```
>>> partners = [("Jane", "Joe"), ("Bill", "Jack"), ("Nick", "Sam")]
>>> print(len(partners))



>>> print(type(partners[0]))



>>> partners[0][0] = "Jim"
>>> print(partners[0])
```

Part (e) [4 marks]

```
>>> marks = {"Abby": 90, "Kay": 80, "Bob": 49}
>>> print(marks["Abby"])



>>> marks["Bob"] = 50
>>> print(marks)



>>> print(len(marks))



>>> print(80 in marks)



>>> print(marks.get("Abby", None))



>>> print(marks.get("Jon", 0))



>>> print(marks[0])
```

Question 2. [6 MARKS]

Part (a) [4 MARKS]

Complete the following function according to the docstring, by filling in the blanks within the given code. For full marks, you must complete this by only filling in the blanks with lines of code as appropriate. You should **not** be re-writing the function.

```
def most_seen(d: dict) -> str:
    """Return the word in the given dictionary <d> that has been seen on the most number of
    pages. The dictionary contains words as the key, and a list of page numbers on which the
    word appears as a value. If there are multiple most frequent words, return either one.

    >>> most_seen({"a": [1, 3, 6, 8], "b": {5, 6}, "c": [8]})
    "a"

    >>> s = most_seen({"happy": [1, 19, 67], "sad": [1, 5, 7], "angry": [99]})
    >>> s == "happy" or s == "sad"
    True
    """

    curr_max = ____________________

    curr_word = ""

    for _________________ in _________________:

        if ________________________ > curr_max:

            curr_max = ___________________________

            curr_word = ___________________________

    return ______________________________
```

Part (b) [2 MARKS]

Complete the following test case table for the above function by adding two more test cases, their expected values, and a purpose for the test case. Make sure each test case in the table has its own unique purpose. The first row is completed for you as an example.

Parameter Value (d)	**Expected Return Value**	**Purpose of Test Case**
`{"a": [1, 2, 3], "b": [2, 5, 7], "c": [99]}`	'a' or 'b'	Multiple most frequent words

Question 3. [7 marks]

We are keeping track of student grades on each test and assignment in our course using separate text files, all of which are formatted as below. The first line contains headings, and each line afterward contains a student ID followed by an integer grade that student received. The ID and grade are separated by a comma.

An example of such a file is below. The format of all our grades files will look just like this, although the actual student IDs, grades, and the number of students there are in a given file will differ and thus cannot be hard-coded into any of your functions.

```
Student ID,Grade
hanna672,98
jacksonx,73
liucarl1,100
bobson3,48
```

Again, the above is just one example of what a file like this could look like. A marks file can have any number of students and grades in them, not just four like the example does.

Complete the function below which takes in the name of a file as a string (the file is a marks file, formatted like above) then calculates and returns the average grade for the test (that is, take the grades of all the students in the file and return their average).

Note: Remember to **open the file before reading it and close it** appropriately.

```
def get_average(filename: str) -> float:
    """Calculate and return the average grade for all the students
    in the marks file that has the given filename.
    """
```

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

Short Python function/method descriptions:

```
__builtins__:
  sorted(sequence) -> sequence
     Return the elements in the given sequence in sorted (non-decreasing) order.
  range([start], stop, [step]) -> list-like-object of int
    Return the integers starting with start and ending with stop - 1 with step
    specifying the amount to increment (or decrement).  If start is not specified,
    the sequence starts at 0.  If step is not specified, the values are incremented by 1.
  str(x) -> str
    Return an object converted to its string representation, if possible.
str:
  S.find(sub[,i]) -> int
    Return the lowest index in S (starting at S[i], if i is given) where the
    string sub is found or -1 if sub does not occur in S.
  S.split([sep]) -> list of str
    Return a list of the words in S; use string sep as the separator and
    any whitespace string if sep is not specified.
list:
  L.append(object) -> NoneType
    Append object to end of L.
  L.extend(iterable) -> NoneType
     Extend list L by appending elements from the iterable. Strings and lists are
     iterables whose elements are characters and list items respectively.
  L.pop([index]) -> item
    Remove and return item at index (default last).
    Raises IndexError if list is empty or index is out of range.
  L.index(value, [start, [stop]]) -> integer
     Return first index of value. Raises ValueError if the value is not present.
dict:
  D[k] -> object
    Return the value associated with the key k in D.
  D.get(k, [optional=None]) -> object
    Return the value associated with the key k in D.
    If this doesn't exist, return the second argument (by default this is None).
  D.keys() -> list-like-object of object
    Return the keys of D.
  D.values() -> list-like-object of object
    Return the values associated with the keys of D.
  D.items() -> list-like-object of tuple of (object, object)
    Return the (key, value) pairs of D, as 2-tuples.
file:
  F.open(s, [optional='r']) -> file object
      Open and return the file named the given filename s. By defualt,
      file is opened with mode 'r' for reading. If mode 'w' is given, the file is opened for writing.
  F.read() -> str
     Read the contents of the file and return as one string. Each line
     in the file is separated by the newline character \n.
  F.readline() -> str
     Read the next line in the file and return as a string ending
     the newline character \n.
  F.write(s) -> int
     Write the given string s to the file F. Return the number of characters written.
  F.close() -> None
     Close the file F.
```