The exam may cover any topic from class or the assigned readings, but it will focus on material covered since Exam 2.

To review, you should carefully go through the lecture notes, labs, and homework assignments to be sure that you understand all of the concepts. For extra practice, we're providing these problems.

In today's review session, I'll give you 5–10 minutes to work on each problem, individually or with the student next to you. Then we'll discuss solutions as a group before moving on to the next problem. I'll leave some time at the end for a general Q&A.

For all problems, be sure to practice writing type annotations, docstrings, and test functions unless the problem says otherwise.

1 Slicing and recursion

a. Design a function first_is_unique that takes a non-empty string s as input and returns a Boolean as output. The function should return True if the first character in s appears is unique – it doesn’t appear anywhere else in s – and False otherwise (that is, the first character does appear elsewhere in s).

While this could be solved recursively or iteratively, neither is necessary! Instead, use string slicing and Python’s in operator to write a very short solution.
b. Write a recursive function called `every_other_letter` that takes a string as input and returns a string that contains every other letter in the string starting with the first letter. The intended behavior is demonstrated by the provided test function.

```python
def test_every_other_letter():
    assert every_other_letter("a") == "a"
    assert every_other_letter("kale") == "kl"
    assert every_other_letter("banana") == "bnn"
    assert every_other_letter("computer") == "cmue"
```
2 Iteration

a. Consider the following function:

```python
def mystery(l: list):
    for i in l:
        return i
```

What does this function return when [1, 2, 3] is passed in?

What does the same function return when [] is passed in?

b. Consider the following function:

```python
def mystery(l: list):
    result = 0
    for i in l:
        result = i
    return result
```

What does this function return when [1, 2, 3] is passed in?

What does the same function return when [] is passed in?
c. Consider the following Pyret function definition:

```pyret
fun short-strings(lst :: List<String>, threshold :: Number) -> List<String>:
    filter(lam(word): string-length(word) < threshold end, lst)
where:
    short-strings([list: ], 3) is [list: ]
    short-strings([list: "aaa", "b", "cccc", "dd"], 3) is [list: "b", "dd"]
end
```

Write a Python function `short_strings` that accomplishes the same task as `short-strings`.

**Note:** Your function should use a `for` loop to iterate through the elements of the input list. It should not use `filter` (or other higher-order functions) or a list comprehension.
d. Write a function `count_changes` that takes in a string `s` of "0"s and "1"s and returns the number of times there is a change from a "0" to a "1" or vice versa in that input string.

For example, `count_changes("110110000")` would return 3.
3  Structured data and memory

a. Consider the following program:

```python
@dataclass
class Time:
    hour: int
    mins: int

noon = 12
quarter = 15

twelve_fifteen = Time(noon, quarter)
lunch = twelve_fifteen
phone_call = Time(noon, quarter)

# Can't eat till the call's over!
lunch.min = 30
```

List the entries in the directory and the heap after evaluating the program.
b. Consider the following program:

```python
@dataclass
class Course:
    code: str
    students: list  # of strings
    term: str

current_term = "Fall 2022"
next_term = "Spring 2023"

course1 = Course("cmpu101", ["Ada", "Alan"], current_term)
course2 = Course("cmpu102", ["Grace", "Winifred"], next_term)
course3 = course1

course1.students.append("Matthew")
course2 = Course("cmpu102", ["Grace", "Winifred"], next_term)
course3 = course1

print(course1)
print(course2)
print(course3)
```

Write the results of the print statements.
4 Dictionaries

a. What output is printed by the following code?

```python
d = {"May": "spring"}
d["July"] = "summer"
d["April"] = "spring"

for month in ["March", "April", "May", "June", "July"]:
    if month in d:
        print(d[month])
    else:
        print("error")
```
b. Consider a Python dictionary whose keys are words, and where the associated values are definitions of those words. For example,

```python
animals = {
    "zebra": "a striped animal",
    "aardvark": "an animal that likes to eat ants",
    "skunk": "a striped animal"
}
```

Write a function called `reverse_lookup` that takes as input a dictionary and a definition string. The function then returns the list of all keys that have that definition.

For example, if we passed in the dictionary `animals` and the string "a striped animal", we would get back a list containing "zebra" and "skunk".
5 Tables revisited

Consider the table students, which has columns for the first name, birth month, and favorite color of several students:

<table>
<thead>
<tr>
<th>name</th>
<th>month</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alice&quot;</td>
<td>&quot;Jan&quot;</td>
<td>&quot;red&quot;</td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td>&quot;Aug&quot;</td>
<td>&quot;red&quot;</td>
</tr>
<tr>
<td>&quot;Cathy&quot;</td>
<td>&quot;Jan&quot;</td>
<td>&quot;blue&quot;</td>
</tr>
</tbody>
</table>

a. Write a line or two of code to add a column called house to the (updated) students table, choosing whichever Vassar houses you’d like for Alice, Bob, and Cathy. The resulting table should have the same name, students.

b. Write a function whose_birth_month that takes in a string mnth indicating a month and returns an array of the people in the students table who were born in month mnth.

c. Write a line of code that calls whose_birth_month to create an array named similar_to_me that has the names of all the people listed in students who were born in the same month as you. (Pass your birth month in as a string, like "Jan", "Feb", etc.)