Lecture 11 (Week 12): Subject Review and Exam Prep

Dr Simon D'Alfonso

School of Computing and Information Systems
Melbourne School of Engineering

Final Week of Semester

- A1 solutions have been released
- A2 due at the end of this week
- Practice exam material now up on LMS
- Consultation during exam prep weeks
- Any questions?

Today

1. List comprehensions, 2D-lists recap
2. Recursion recap
3. Exam stuff

Summary of collections

Operation	str	list	tuple	dict	set
Add item	N/A	l.append()	N/A	d[key] = value	s.add()
Indexing	s[index]	l[index]	t[index]	d[key]	N/A
Look Up	element in s	element in l	element in t	key in d	element in s
Slice	s[i:j]	l[i:j]	t[i:j]	N/A	N/A
Order preserving	Yes	Yes	Yes	No	No

List Comprehension

Basic form:

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

Is equivalent to:

```
for item in list:
    if conditional:
        expression
```

List Comprehension

In Python we often create lists from other lists.

Example:

```
words = ['HeLLo', 'GooDBYe']
lower_words = []
for w in words:
    lower_words.append(w.lower())
```

Python list comprehension allows us to perform these tasks in one line.

```
lower_words = [w.lower() for w in words]
```

List comprehension

```
#Create a list of the squares of
integers 1-10.

squares = [x**2 for x in range (1,
11)]

#Create a list of only the even
numbers in list n
n = [45, 2, 36, 77, 32, 19]

evens = [x for x in n if x%2 == 0]
```

2-Dimensional Lists

- Row-column 2-dimensional tables are a very common data structure. E.g., Excel spreadsheets, websites (HTML: <table>, <tr>, <td>), etc.

Product Name	Quantity in Stock	Price	How Many Sold
Product1	5	45.5	10
Product2	8	10.2	50
Product3	9	6.4	20
Product4	20	8.6	20
Product5	15	10	30

- How can this be represented as a Python list structure?

2-Dimensional Lists

```
headers = ["Product Name", "Quantity in
Stock", "Price", "How Many Sold"]
prod1 = ["Product1", 5, 45.5, 10]
prod2 = ["Product2", 8, 10.2, 50]
prod3 = ["Product3", 9, 6.4, 20]
prod4 = ["Product4", 20, 8.6, 20]
prod5 = ["Product5", 15, 10, 30]
products = [headers, prod1, prod2, prod3,
prod4, prod5]

import pprint #pretty printer
pprint.pprint(products)
```

2-Dimensional lists

Change the price of Product1 to $50 and change the quantity of Product3 to 10:

```
products[1][2] = 50 #row 1, #column 2
products[3][1] = 10 #row 3, #column 1
```

```
[
    ["Product Name", "Quantity in Stock", "Price", "How
Many Sold"],
    ['Product1', 5, 50, 10],
    ['Product2', 8, 10.2, 50],
    ['Product3', 10, 6.4, 20],
    ['Product4', 20, 8.6, 20], ['Product5', 15, 10, 30]
]
```

2-Dimensional Lists

Print each row (i.e., product)

```
for i in range(len(products)):
    list_row = []
    for j in range(len(products[i])):
        list_row.append(products[i][j])

    print(list_row)
```

Print each column

```
for i in range(len(products[0])):
    list_column = []
    for j in range(len(products)):
        list_column.append(products[j][i])

    print(list_column)
```

Recursion

Fibonacci sequence:
0,1,1,2,3,5,8,13,21,34,55,89,…

```
#calculate the nth Fibonacci number
def fib(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

Iterative Approach

```
def fib_iterative(n):
    a, b = 0, 1
    for i in range(n - 1):
        a, b = b, a + b
    return b
```

Recursion

Sometimes the record of a value needs to be stored throughout the recursion, so pass it as an argument in the recursive function calls:

```
#calculates max num of list
def recursion_max(my_list, curr_max=0):
    # check to see if we've reached the end of the list
    if my_list == []:
        return curr_max
    # check to see if current head element is greater than the current max
    elif my_list[0] > curr_max:
        return recursion_max(my_list[1:], my_list[0])
    # head element isn't greater than `curr_max` so check rest of my_list
    else:
        return recursion_max(my_list[1:], curr_max)
```

The Exam

- What's examinable:
  - Lecture/tutorial content and the topics covered throughout the semester
  - Grok worksheets mentioned in the weekly LMS modules
  - Ideas from assignments
- Like Mid-Semester Test - the link to the exam will be on the QUIZZES page of the LMS site.
- 3 hours (+ 15 mins reading time)
- Lecturer will be available via email and LMS quiz chat tool for any technical issues or questions during initial period
- Unimelb exams webpage, with everything you need to know about your exams - when they will be, how to prepare, and what to do on exam day: https://students.unimelb.edu.au/your-course/manage-your-course/exams-assessments-and-results/exams

The Exam

Remember that the examination is designed to help you express what you have learned in this subject. The aim of this subject is NOT for you to become professional programmers, but instead to become familiar with programming concepts and techniques. Although the exam will be challenging, you will be given significant marks for all of the sensible elements that you give as answers, including fragments of code, whether or not they fully solve the problem.

Types of questions that might be on exam

- The types of questions on the MST
- Multiple choice questions about the elements of Python programs, techniques of programming, and computational thinking.
- Short answer questions about the elements of Python programs, techniques of programming, and computational thinking.
- Presenting pieces of code and asking you to interpret their purpose, outputs, any errors, and other aspects.
- Coding challenges requiring a mixture of familiar patterns and deeper analysis of the problem.

Exam Points

- When we mark your answers, we think like a Python terminal (syntax errors may be *partially* penalized)
- We don't mark (or expect) comments in your code, though you are free to add them.
- You can use your own Python dev tools to compose/test your ideas, and then carefully transfer them into the LMS quiz boxes.
- Sample front page

Practice Question

An “abecedarian” word is one where all of its letters are in alphabetical order. For example, the words ‘deep’ and ‘adept’ are abecedarian. Write a function is_abecedarian(word), which returns True or False depending on whether the input word is an abecedarian word.

```
def is_abecedarian(word):

    for i in range(len(word)-1):
        if word[i].lower() > word[i+1].lower():
            return False

    return True
```

Practice Question

Write a function second_largest() which takes a list of numbers as input and returns the second largest number. Note that if the maximum/largest element occurs more than once in the list, then the function should still return the numerically second largest number. For example, if the list is [9,9,8,5], then the function would return 8.

Lecture Identification and Acknowledgement

Coordinator / Lecturer: Simon D'Alfonso

Semester: Semester 1, 2022

These slides include materials from 2020 - 2021 instances of COMP90059 run by Kylie McColl or Wally Smith