COMPSCI 101
Principles of Programming

Lecture 26 - Test2 Revision

2020 S0 Test2

- Worth 25% of your final mark for CompSci 101
- 2 hours, 12 Qs
- No calculators

Look over labs, assignments, code runner exercises, lecture exercises, the test.

Topics

- Bits and pieces: slicing lists, tuples and strings, evaluating an arithmetic expression, len(), min(), max(), round(), sum(), int(), float(), str(), input(), mutable objects, in, and, or, not, tuples, object types, passing objects as parameters,
- `if` … `elif` … `else` statements, `while` loops, for … in range(…) and for … in … loops, range(start, end, step)
- Process a string, string methods: rfind(), find(), split(), manipulate and reassemble a string, slicing a string
- Some questions on lists, define a function which manipulates list objects
- Some questions on dictionary, define a function which creates a dictionary, and prints the dictionary
- Some questions on tuples, define a function which manipulates tuples
- Read text from a text file, process the text information, return result

Questions

- while Loop
- Lists
- for loop
- Dictionaries
- Tuples
- File Reading
- Nested loops

Q1: String manipulation

- Write a function called print_title(word) which takes a string as a parameter and prints the word in a series of lines. Each line of the word is shortened by removing the first and the last character until the word contains no more characters. The left indent is made up of an increasing number of '-' characters. The characters of the word are all in uppercase characters.
- print_title('marvellous')

```
MARVELLOUS
-ARVELLOU
--RVELLO
---VELL
----EL
```

- print_title('fantastic')

```
FANTASTIC
-ANTASTI
--NTAST
---TAS
----A
```

Algorithm

- Parameter: word
  - marvellous
  - Length of the word: 10
  - Number of rows = 5
  - 1st row: zero '-'
  - 2nd row: 1 * '-' and 8 letters, index: 1 to 9 (i.e. 10-1)
  - 3rd row: 2 * '-' and 6 letters, index: 2 to 8 (i.e. 10-2)
  - 4th row: 3 * '-' and 4 letters, index: 3 to 7 (i.e. 10-3)
  - 5th row: 4 * '-' and 2 letters

```
MARVELLOUS
-ARVELLOU
--RVELLO
---VELL
----EL
```

Length of the word: 9
Number of rows = 5
1st row: zero '-'
2nd row: 1 '-' and 7 letters
3rd row: 2 '-' and 5 letters
4th row: 3 '-' and 3 letters
5th row: 4 '-' and 1 letter

```
FANTASTIC
-ANTASTI
--NTAST
---TAS
----A
```

Q2: Python Lists

- Complete the convert_first_letter() function which is passed a list of names as a parameter. The function changes the first letter of each name in the list to uppercase, leaving the rest of the name unchanged. You can assume that each element of the list contains at least one character.

```
1. names: ['karl', 'Orlando', 'carlo', 'zAC']
2. names: ['Karl', 'Orlando', 'Carlo', 'ZAC']
```

- Common mistake: for-each loop <- DO NOT USE

```
for word in names_list:
    word = word[0].upper() + word[1:]
```

Q3: Python Lists

- write a function called sum_over(a_list_of_lists, target) which takes a list of integer lists and an integer as parameters, and returns the sum of all entries in the parameter list of lists which are greater than a specified amount, target.

```
the_list = [[2, 4, 16, 80, 27], [1, 4, 120, 18, 7],
                                 [20, 14, 70, 8, 130]]
print(sum_over(the_list, 50))

                                400
                                (i.e.80 + 120 + 70 + 130

print(sum_over(the_list, 100))
                                250
                                (i.e. 120 + 130)
```

Algorithm

- Parameter: list_of_lists

```
result = 0
    for each list in the list of lists
        for each item in the list
            if item is bigger than the target
                Add item to the result
```

Q4: Python Tuples -

- Write a function called get_left_most_x(points) which takes a list of tuples as a parameter and returns the smallest left-most position of any tuple in the list. You can assume that
  - every element of the list contains more than 1-element tuple
  - the size of the list is at least one, and
  - Values are all >=0

```
points = [(100,5), (20, 100), (140, 200), (70, 100), (25,
0)]
print("Smallest x value:", get_left_most_x(points))
```

- Steps:
  - Set a min value
  - for each tuple in the list of tuples
  - …

```
Smallest x value: 20
```

Q5: Python dictionaries

```
def main():
   dict1 = {"A": [1, 2, 3, 5], "B": [1, 2, 8], "X": [0, 9], "N": [8]}
   dict2 = {"A": [5], "B": [2, 4, 7], "T": [5, 6], "N": [3, 8]}
   add_dict2_values(dict1, dict2)

def add_dict2_values(dict1, dict2):
```

```
A : [1, 2, 3, 5]
B : [1, 2, 4, 7,
8]
N : [3, 8]
X : [0, 9]
```

- Complete the merge(dict1, dict2) function which is passed two dict objects as parameters, dict1 and dict2. Both parameter dictionaries have a single character as the keys and a list of integers as the corresponding values.
  - The function looks at the lists corresponding to the **same** key in both dictionaries.
    - For any key which is the same in both dictionaries, then any integer in the corresponding list of dict2 which is **not** already in the corresponding list of dict1 is added to the dict1 corresponding list. All the corresponding lists of dict1 are kept in sorted order.

dict1 = {"A": [1, 2, 3, 5], "B": [1, 2, 8], "X": [0, 9], "N": [8]}

dict2 = {"A": [5], "B": [2, 4, 7], "T": [5, 6], "N": [3, 8]}

```
A : [1, 2, 3, 5]
B : [1, 2, 4, 7, 8]
N : [3, 8]
X : [0, 9]
```

Algorithm

- Parameters: dict1, dict2

```
For key1 and list1 in dict1
  • if key1 exists in dict2
    • Get the corresponding list from dict2
    • For all values in the list
      • If it is not exists in the list from dict1
        • append it to list1
Sort all values in the list1
```

Q6: File Reading

- Complete the get_lines_from_file(filename) function. This function takes a filename as a parameter, open and reads the contents of the file specified in the parameter. This file contains several webpages and links. The file contents should then be converted into a list of strings.
- Input:

```
home:news
home:calendar
home:enrolments
our_people:staff
...
```

$\Rightarrow$

```
['home:news', 'home:calendar', 'home:enrolments',  ... ]
```

Q7: Print pyramid

- Write a function called print_pyramid(number) to produce a triangle of empty spaces surrounded by "+"s.
- If the parameter is 4, the function should produce:

```
+++++++++
++++ ++++
+++   +++
++     ++
+       +
+++++++++
```

- If the parameter is 1, the method should produce:

```
+++
+ +
+++
```

Check the Pattern

```
+++++++++
++++ ++++
+++   +++
++     ++
+       +
+++++++++
```

- number = 4
  - Total 6 lines
  - 1st and the last line: 9 "+" => number * 2 + 1
  - 4 rows:
    - 1st row (i=0) : 4 "+" and 1 space and 4 "+"
    - 2nd row (i=1) : 3 "+" and 3 spaces and 3 "+"
    - 3rd row (i=2) : 2 "+" and 5 spaces and 2 "+"
    - 4th row (i=3) : 1 "+" and 7 spaces and 1 "+"
    - i.e. _____ "+" and i*2 + 1 spaces and _______ "+"
    - Use nested for loops