# CS2110 Lecture 21 Mar. 12, 2021

- HW5 due Mon. 3/15
  - Q3 and Q4: it is fine to use loops as long as the function is also recursive.

## Last time

- Grades so far
- More recursion examples

## Today

- A couple more recursion examples
- A short introduction to exceptions

COURSE GRADE GUARANTEES: If, at the end you have:
85%+: A
70%+: B
50%+: C
40%+: D.

*Notes:*
*1) for each letter grade, it really means "some kind of", e.g. for A: A-, A, or A+. 85% does not guarantee a "regular" A*
*2) "guarantee" means that the percentages won't increase. They might be lowered a little. E.g. A could go to 84%+*

| Student | HW1 | HW2 | HW3 | HW4 | | DS1 | DS2 | DS3 | DS4 | DS5 | Quiz 1 | Quiz 2 | | Homework Total | DS Total | Quiz Total | | Total | % |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Points Possible | 6.00 | 6.00 | 6.00 | 6.00 | | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 15.00 | 20.00 | | 24 | 15 | 35 | | 74 | 100 |
| | | | | | | | | | | | | | | | | | | | |
| | 6.00 | 6.00 | 6.00 | 6.00 | | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 14.00 | 20.00 | | 24.00 | 15.00 | 34.00 | | 73.00 | 98.65 |
| | 6.00 | 5.00 | 4.00 | 5.00 | | 3.00 | 3.00 | 3.00 | 2.00 | 3.00 | 14.00 | 20.00 | | 20.00 | 14.00 | 34.00 | | 68.00 | 91.89 |
| | 6.00 | 6.00 | 5.00 | 3.00 | | 3.00 | 3.00 | 0.00 | 2.00 | 3.00 | 15.00 | 20.00 | | 19.00 | 11.00 | 35.00 | | 65.00 | 87.84 |
| | 6.00 | 6.00 | 4.00 | 5.00 | | 3.00 | 2.00 | 2.00 | 3.00 | 3.00 | 14.00 | 16.00 | | 20.00 | 13.00 | 30.00 | | 63.00 | 85.14 |
| | 5.00 | 5.00 | 3.00 | 0.00 | | 3.00 | 3.00 | 3.00 | 2.00 | 3.00 | 14.00 | 19.00 | | 13.00 | 14.00 | 33.00 | | 60.00 | 81.08 |
| | 6.00 | 6.00 | 2.00 | 3.00 | | 3.00 | 3.00 | 3.00 | 2.00 | 2.00 | 10.00 | 18.00 | | 17.00 | 13.00 | 28.00 | | 58.00 | 78.38 |
| | 4.00 | 6.00 | 2.00 | 2.00 | | 3.00 | 3.00 | 3.00 | 3.00 | 2.00 | 14.00 | 13.00 | | 14.00 | 14.00 | 27.00 | | 55.00 | 74.32 |
| | 4.00 | 6.00 | 4.00 | 0.00 | | 3.00 | 1.00 | 3.00 | 0.00 | 1.00 | 11.00 | 20.00 | | 14.00 | 8.00 | 31.00 | | 53.00 | 71.62 |
| | 5.00 | 5.00 | 4.00 | 4.00 | | 3.00 | 3.00 | 2.00 | 3.00 | 3.00 | 3.00 | 14.00 | | 18.00 | 14.00 | 17.00 | | 49.00 | 66.22 |
| | 4.00 | 6.00 | 3.00 | 2.00 | | 3.00 | 3.00 | 3.00 | 3.00 | 3.00 | 8.00 | 11.00 | | 14.00 | 15.00 | 19.00 | | 48.00 | 64.86 |
| | 6.00 | 6.00 | 3.00 | 0.00 | | 3.00 | 2.00 | 2.00 | 3.00 | 0.00 | 7.00 | 17.00 | | 13.00 | 10.00 | 24.00 | | 47.00 | 63.51 |
| | 2.00 | 5.00 | 4.00 | 4.00 | | 3.00 | 2.00 | 2.00 | 2.00 | 2.00 | 8.00 | 12.00 | | 15.00 | 11.00 | 20.00 | | 46.00 | 62.16 |
| | 3.00 | 4.00 | 2.00 | 3.00 | | 3.00 | 2.00 | 2.00 | 2.00 | 2.00 | 10.00 | 8.00 | | 12.00 | 11.00 | 18.00 | | 41.00 | 55.41 |
| | 5.00 | 5.00 | 5.00 | 2.00 | | 3.00 | 3.00 | 0.00 | 2.00 | 0.00 | 6.00 | 8.00 | | 17.00 | 8.00 | 14.00 | | 39.00 | 52.70 |
| | 3.00 | 5.00 | 0.00 | 2.00 | | 2.00 | 3.00 | 2.00 | 2.00 | 2.00 | 8.00 | 6.00 | | 10.00 | 11.00 | 14.00 | | 35.00 | 47.30 |
| | 4.00 | 4.00 | 0.00 | 3.00 | | 3.00 | 3.00 | 2.00 | 2.00 | 2.00 | 6.00 | 4.00 | | 11.00 | 12.00 | 10.00 | | 33.00 | 44.59 |
| | 4.00 | 3.00 | 0.00 | 0.00 | | 3.00 | 0.00 | 0.00 | 0.00 | 2.00 | 5.00 | 5.00 | | 7.00 | 5.00 | 10.00 | | 22.00 | 29.73 |
| | 3.00 | 0.00 | 1.00 | 0.00 | | 2.00 | 2.00 | 2.00 | 0.00 | 2.00 | 3.00 | 4.00 | | 4.00 | 8.00 | 7.00 | | 19.00 | 25.68 |

# (last time) Important rules for recursive functions

- When writing a recursive function:
  - MUST have base case(s), situations when code *does not* make recursive call.
  - MUST ensure that recursive calls *make progress toward base cases.* I.e. you need to convince yourself that recursive call is "closer to" base case than the original problem you are working on
  - SHOULD ensure you don't unnecessarily repeat work. Ignoring this contributes to recursion's bad reputation. E.g. direct recursive implementation of Fibonacci is extremely and unnecessarily inefficient

# (last time) Recursion examples

- More basic recursion examples (lec19.py)
  - Print the items of a list, one per line
  - Print the items of a list, one per line, in reverse order
    - Idea?
    - Consider list as: theFirstItem <the rest of the list>
    - Reverse is: reverse(<the rest of the list>) theFirstItem
    - Consider list as: <list from start to near end> theLastItem
    - Reverse is: theLastItem reverse(<list from start to near end>)
  - return a string that is the reverse of the given string
  - sum the items in a list
  - return True/False depending on whether given string is a palindrome ( e.g. Was it a car or a cat I saw?)
  - return num of digits in an integer
  - return sum of digits in an integer
  - return string with each occurrence of a particular character with a different character
  - return count of number of substrings that have same first and last characters
  - compute nth Fibonacci number:
    - 1, 1, 2, 3, 5, 8, 13, …
    - Definition: fib(1) = 1, fib(2) = 1,
    - fib(n) = fib(n-1) + fib(n-2) for n > 2

# More recursion examples

- generate a string pattern
- “flatten” a list
  - E.g. [[[[[3,[2,4]]], 0], [‘a’]], 23] -> [3,2,4,0,’a’, 23]
- Towers of Hanoi problem
- nesting depth
- Drawing shapes
  - https://en.wikipedia.org/wiki/Koch_snowflake

lec20.py
ToHcomplete.py

# Nesting depth

- Def of nestingDepth of a list
  - 0 if list has no lists as elements
  - 1 more than maximum nestingDepth among all items of the list that are lists
- Examples
  - [] -> 0
  - [[]] -> 1
  - [[[]]] -> 2
  - [[[]], 1, 2] -> 2, because [[]] is 1 so the whole list is 1 + 1 = 2
  - [ [[]], 1, 2, [[[100]]] ] -> 3, because [[]] is 1, [[[100]]] is 2, max is 2, so whole list is 2 + 1 = 3

```
def nestingDepth(inList):
    # initialize a variable for max nesting level seen (among sublists)
    # iterate over items in inList
    #     if item is not a list, just skip it
    #     if item is a list, recursively compute its nesting level and
    #       compare that item's nesting level with max seen so far,
    #       updating max if nec
    # if max has not been updated in the loop, return 0
    # otherwise return 1 more than max
```

# Handling errors, raising errors, monitoring presumptions

- try/except
- raise
- assert – good practice to ”sprinkle” asserts throughout your code. Catch cases of presumed conditions not being met before they result in mysterious, hard-to-track down errors

Try/except and raise covered well in basic Python documentation

Textbook Ch 13 covers exceptions. I won’t test you on this material but it is very useful.

exceptions.py

# Next Week

- Next big topic: Classes and object-oriented programming (Chapters 17, 18, and 19)