CSC148 winter 2018

functional programming, top-down

week 5

Danny Heap

heap@cs.toronto.edu / BA4270 (behind elevators)

http://www.teach.cs.toronto.edu/~csc148h/winter/

416-978-5899

February 5, 2018

Computer Science
UNIVERSITY OF TORONTO

Outline

idiomatic python

going with the (pep) tide

Python is more flexible than the community you are coding in. Try to figure out what the python way is

- don't re-invent the wheel (except for academic exercises), e.g. `sum`, `set`
- use comprehensions when you mean to produce a new list (tuple, dictionary, set, ...)
- **any** $\approx \exists$ **all** $\approx \forall$
- use ternary `if` when you want an expression that evalutes in different ways, depending on a condition

example: add (cubes of) first 10 natural numbers

- You'll be generating a new list from `range(1, 11)`, so use a comprehension

- You want to add all the numbers in the resulting list, so use `sum`

euclidean distance in 3 dimensions... or more

Suppose `L = [x, y, z]`, using `L`, compute:

$$\sqrt{x^2 + y^2 + z^2}$$

average string length

Suppose `L = ["my", "dog", "has", "fancy", "fleas"]`, compute the average string length using `L`

try big list with any/all

```
with open("/usr/share/dict/words", "r") as words_file:
    word_list = words_file.read().split("\n")
```

list differences, lists without duplicates

- python `lists` allow duplicates, python sets don't
- python `sets` have a set-difference operator
- python built-in functions `list()` and `set()` convert types

possible test topics

include...

- class design
- special methods
- subclasses
- inheritance
- testing, exceptions
- ADTs, stacks, queues, sacks
- linked lists

valid sudoku

what makes a sudoku square valid?

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

- valid rows
- valid columns
- valid subsquares

code it!

```
def valid_sudoku(grid, digit_set: set) -> bool:
"""
Return whether grid represents a valid, complete sudoku.
"""
assert all([len(r) == len(grid) for r in grid])
assert len(grid) == len(digit_set)
return (_all_rows_valid(grid, digit_set) and
        _all_columns_valid(grid, digit_set) and
        _all_subsquares_valid(grid, digit_set))
```

code those non-existent helpers!

```
def _all_rows_valid(grid, digit_set: set) -> bool:
    """
    Return whether all rows in grid are valid and complete.

    Assume grid has same number of rows as elements of digit_set
    and grid has same number of columns as rows.
    """
    assert all([len(r) == len(grid) for r in grid])
    assert len(grid) == len(digit_set)
    return all([_list_valid(r, digit_set) for r in grid])
```

code the helpers' helpers...

```
def _list_valid(r, digit_set: set) -> bool:
    """
    Return whether r contains each element of digit_set
    exactly once.

    Assume r has same number of elements as digit_set.
    """
    assert len(r) == len(digit_set)
    return set(r) == digit_set
```