Week 3: ADTs

Arnamoy Bhattacharyya

Announcement

If you still want to change sessions, or missed some labs due to late enrolment, fill up the form and send email to

csc14818s@cs.toronto.edu

Overview

Documentation during inheritance

ADTs

Implementation of ADTs and Inheritance

Documentation

Avoid duplicate documentation

- Inherited: No need to document again
- Extended: Only document what is extra
- Overridden: Document details

```
help(RightAngleTriangle)
```

Type Signatures

1. You must be seeing a lot of different docstrings types
   a. @type, @param etc
   b. Feel free to choose any (I prefer @param -- allows description)
2. There is only one type of docstring for signatures (adapted from CSC108)
   a. def __init__ (self, corners: List['Point']) -> None:
   b. We do not annotate "self"
3. You should annotate public attributes

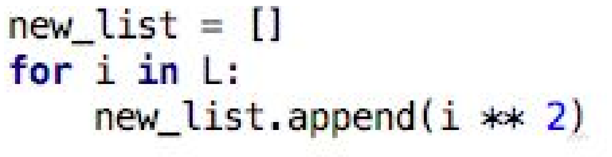
```
Class Shape:

    corners:List['Point']
```

List Comprehension

Suppose L is a list of first hundred natural numbers

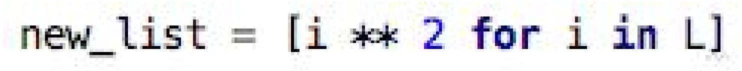
L = range(100)

Create a new list that are squares of each number of L



```
new_list = []
for i in L:
    new_list.append(i ** 2)
```

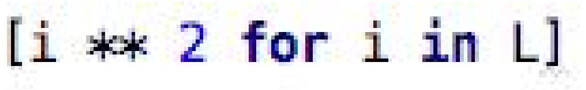
Comprehension



```
new_list = [i ** 2 for i in L]
```

Onto Pycharm

General form



```
[i ** 2 for i in L]
```

[expression **for** name **in** iterable]

You can have if condition at the end -- filter

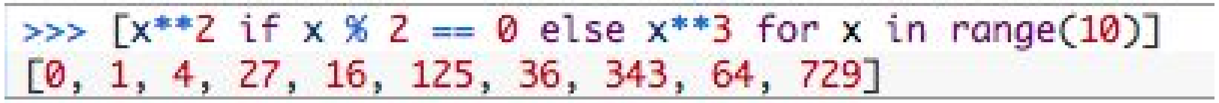
[expression **for** name **in** iterable **if** condition]

Exercise

Write a function odd_square that returns the square of the odd numbers in the range (0, 100)

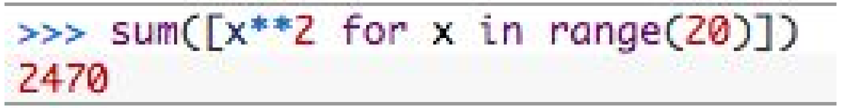
More examples

1. Squares of even numbers and cubes of odd numbers



```
>>> [x**2 if x % 2 == 0 else x**3 for x in range(10)]
[0, 1, 4, 27, 16, 125, 36, 343, 64, 729]
```

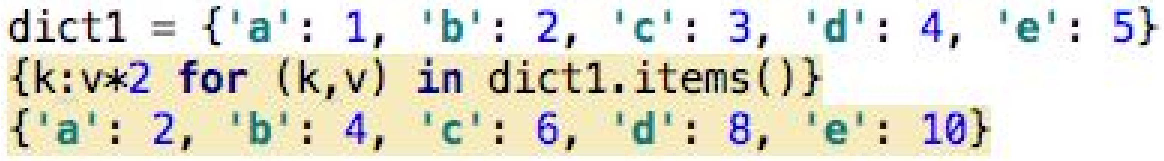
2. Sum of squares of a range of numbers



```
>>> sum([x**2 for x in range(20)])
2470
```

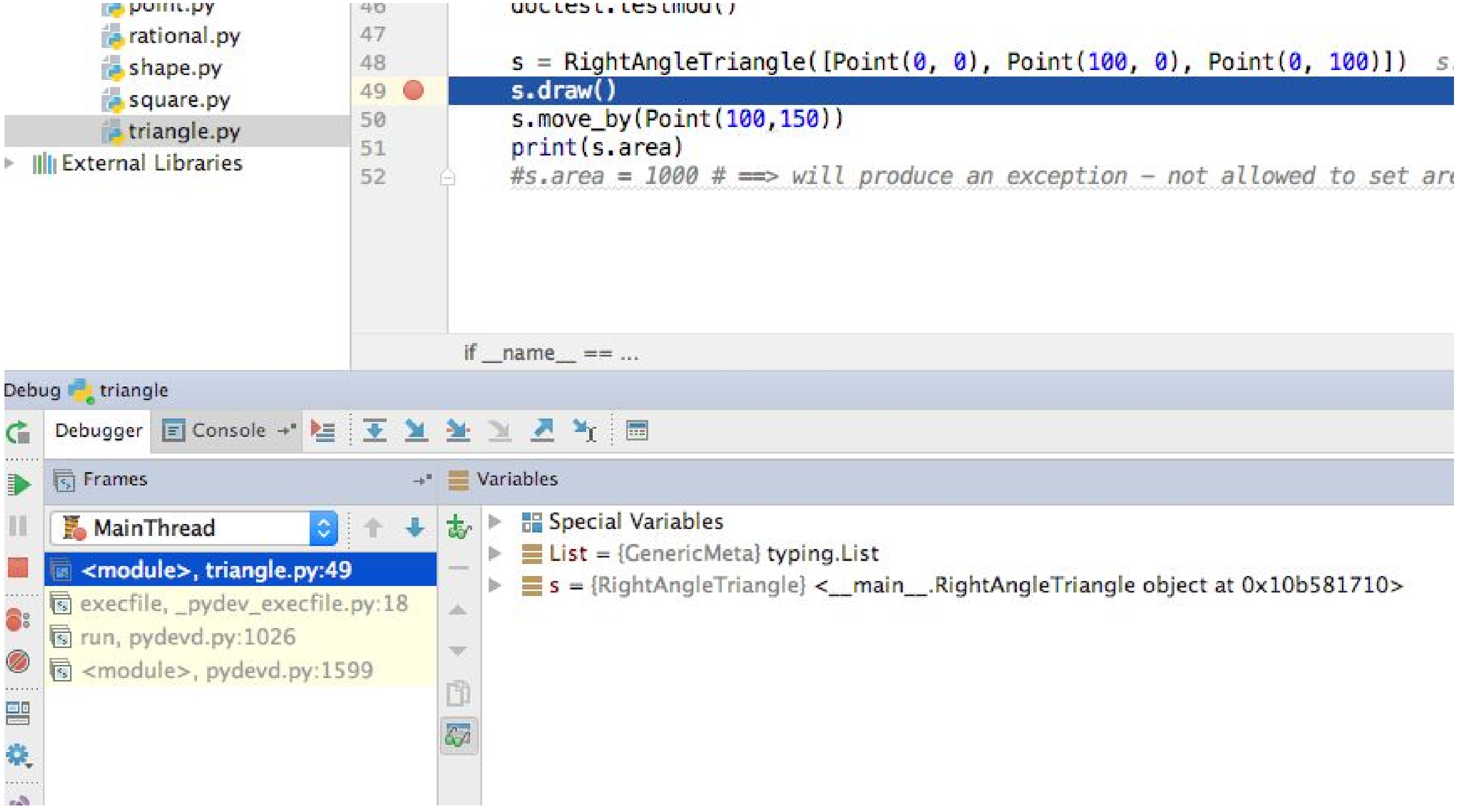
Iterables in Comprehension

Can be dicts



```
dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
{k:v*2 for (k,v) in dict1.items()}
{'a': 2, 'b': 4, 'c': 6, 'd': 8, 'e': 10}
```

Method resolution order



```
    doctest.testmod()

    s = RightAngleTriangle([Point(0, 0), Point(100, 0), Point(0, 100)])  s
    s.draw()
    s.move_by(Point(100,150))
    print(s.area)
    #s.area = 1000 # ==> will produce an exception - not allowed to set are
```

```
<module>, triangle.py:49
execfile, _pydev_execfile.py:18
run, pydevd.py:1026
<module>, pydevd.py:1599
```

```
Special Variables
List = {GenericMeta} typing.List
s = {RightAngleTriangle} <__main__.RightAngleTriangle object at 0x10b581710>
```

Abstract Data Types (ADTs)

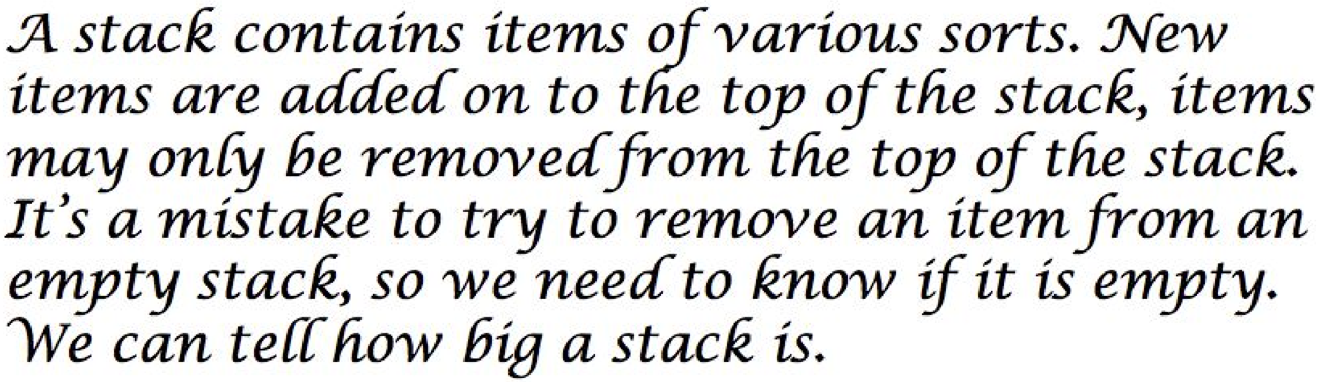
- In CS, we recycle our intuition about the outside world as ADTs
- We abstract data and operations, and suppress the implementation

  - Sequences of items; can be added, removed, accessed by position, etc.

  - Specialized collection of items where we only have access to most recently added item

  - Collection of items accessed by their associated keys

Stack Class Design


A stack contains items of various sorts. New items are added on to the top of the stack, items may only be removed from the top of the stack. It's a mistake to try to remove an item from an empty stack, so we need to know if it is empty. We can tell how big a stack is.

Stack Class

Public Attributes:

None (why?)

Different ways of implementing:

1. Using List, and use the append() and pop() methods
2. Using a List and adding and removing from first **position**
    a. Inefficient -- why?
3. Using a dictionary with integer keys 0, 1, ..., keeping track of the last index used, and which have been removed

Why we hide the “list”?

1. To the users, the data structure is “abstract”
2. We may change implementation we want
    a. From List to Dict implementation
3. We may change the variable name
    a. list to cool_list
4. If users just use the add(), remove() and is_empty() provided, they should be fine

Stack Implementation

Onto PyCharm