CSC148-Section:L0301
Week#3-Friday

Instructed by

AbdulAziz Al-Helali

a.alhelali@mail.utoronto.ca

Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap course material winter17

Computer Science
UNIVERSITY OF TORONTO

Outline

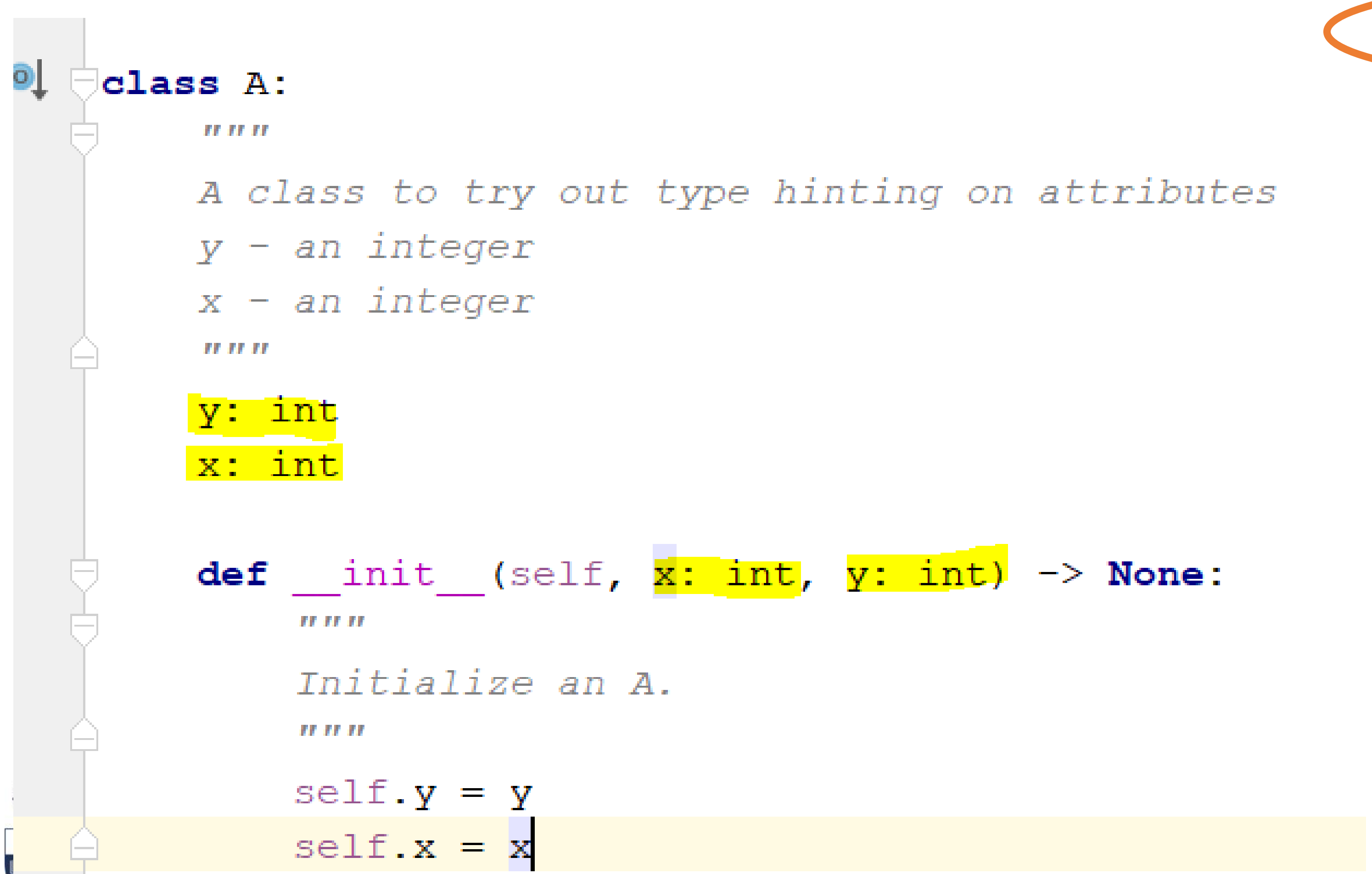
- Documentation, and Type hinting
- List comprehension
- abstract data types (ADTs)

Documentation

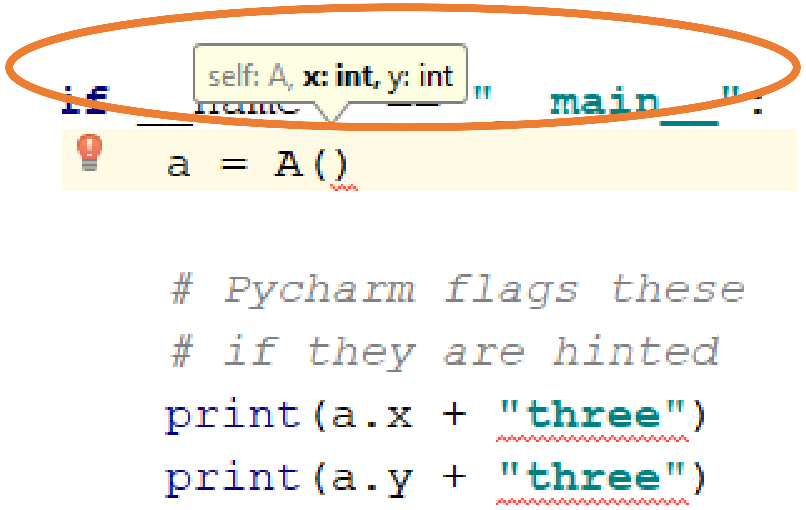
- don't maintain documentation in two places, e.g. superclass and subclass, unless there's no other choice:
  - inherited methods, attributes –
    - no need to document again
  - extended methods
    - User the super class method
    - Add new behavior needed in the subclass
    - Document only what is new
  - overridden methods, attributes –
    - document that they are overridden
    - write new docstring in subclass
- See Shape and Square code from last week.

Pycharm type hinting

type hinting is new in the Python world, and to get the benefit of Pycharm's inspector, some fussing may be needed. . .

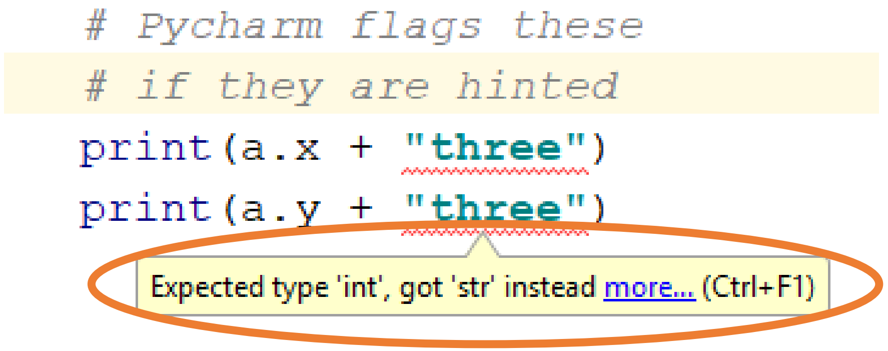


```
class A:
    """
    A class to try out type hinting on attributes
    y - an integer
    x - an integer
    """
    y: int
    x: int

    def __init__(self, x: int, y: int) -> None:
        """
        Initialize an A.
        """
        self.y = y
        self.x = x
```



self: A, **x: int**, y: int

```
if __name__ == "__main__":
    a = A()

    # Pycharm flags these
    # if they are hinted
    print(a.x + "three")
    print(a.y + "three")
```



```
    # Pycharm flags these
    # if they are hinted
    print(a.x + "three")
    print(a.y + "three")
```

Expected type 'int', got 'str' instead more... (Ctrl+F1)

List comprehension

- new lists from old
- suppose L is a list of the first hundred natural numbers:
  - L = list(range(100))
- if I want a new list with the squares of all the elements of L I could
  - new_list = []
  - for x in L:
    - new_list.append(x * x)
- or I could use the equivalent list comprehension
  - new_list = [x * x for x in L]

List comprehension example:

```
>>> mylist = [1, 2, 3]
>>> result = [c+3 for c in mylist]
>>> mylist
[1, 2, 3]
>>> result
[4, 5, 6]
```

Creates new list

The for loop will go through mylist every round putting a new value for c. Then c+3 will be evaluated and the value will be put in the result list as follows:
[1+3, 2+3, 3+3]
The result will be
[4, 5, 6]

```
self.corners =[c + offset_point for c in self.corners]
```

+ calls __add__ in Point class
The resulting is a list of Points

Filtering with [...]

- I can make sure my new list only uses specific elements of the old list. by adding a condition. . .

```
>>> L = ["one", "two", "three", "four", "five", "six"]
>>> new_list = [ s * 3
                  for s in L
                   if s <= "one"]
>>> new_list
['oneoneone', 'fourfourfour', 'fivefivefive']
```

notice that a comprehension CAN **span several lines**, if that makes it easier to understand

general comprehension pattern

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

- Python expressions evaluate to values, name refers to each element of **iterable** (**list**, **tuple**, **dictionary**, ...) in turn, and a condition evaluates to either **True** or **False**

- see Code like Pythonista
    - http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html#list-comprehensions

Abstract Data Types (ADTs)

- An ADT species the intended meaning of the data it stores, and the operations it provides on that data. It **DOES NOT** talk about the how to store and manipulate the data in a particular programming language.

We want to focus on the meaning of the real-world entity being represented rather than the details of how this is implemented for two reasons:

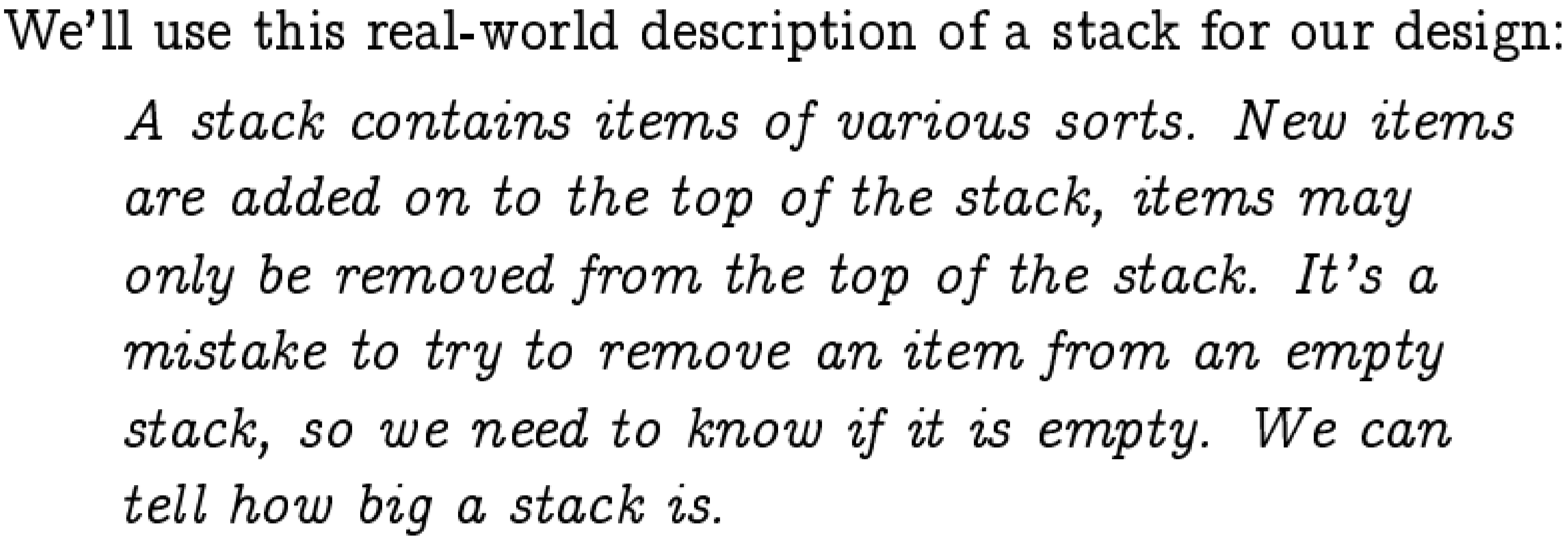
1. We can think about algorithms, or recipes, for solving problems more freely if we don't have to include all the details of how our objects are implemented.
2. Details of how objects, and their components, are stored and accessed vary between programming languages, whereas a really good algorithm can be translated into any programming language.

Example of ADTs

- List
  - sequences of items; can be added, removed, accessed by position
- Stack
  - specialized list where we only have access to most recently added item
- Dictionary
  - collection of items accessed by their associated keys

Computer Science
UNIVERSITY OF TORONTO

stack class design



We'll use this real-world description of a stack for our 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.*

Take a few minutes to identify the main noun, verb, and attributes of the main noun, to guide our class design.

stack class design

- Name: Stack
- Public Attributes: None
- Methods: add, remove, is_empty

implementation possibilities

- The public interface of our Stack ADT should be constant, but inside we could implement it in various ways such as:

1. Use a python list, which already has a pop method and an append method
   - Very easy
2. Use a python list, but add and remove from position 0
   - Easy but will have performance problems
3. Use a python dictionary with integer keys 0, 1, . . . , keeping track of the last index used, and which have been removed
   - Good to practice using dict and its methods

Implementation using list

```
""" implement stack ADT
"""
from container import Container, EmptyContainerException


class Stack(Container):
    """ Last-in, first-out (LIFO) stack.
    """

    def __init__(self) -> None:
        """ Create a new, empty Stack self.
        """
        self._storage = []

    def add(self, obj: object)-> None:
        """ Add object obj to top of Stack self.
        """
        self._storage.append(obj)
```

Implementation using list

```
def remove(self) -> object:
    """
    Remove and return top element of Stack self.

    Assume Stack self is not empty, otherwise
    raises EmptyStackException
    >>> s = Stack()
    >>> s.add(5)
    >>> s.add(7)
    >>> s.remove()
    7
    """
    if self.is_empty():
        raise EmptyContainerException
    else:
        return self._storage.pop()
```

Implementation using list

```
def is_empty(self) -> bool:
    """
    Return whether Stack self is empty.
    >>> s = Stack()
    >>> s.is_empty()
    True
    >>> s.add(s)
    >>> s.is_empty()
    False
    """
    return len(self._storage) == 0
```

Implementation using dictionary

```
""" implement stack ADT
"""
from container import Container, EmptyContainerException

class Stack(Container):
    """ Last-in, first-out (LIFO) stack.
    """

    def __init__(self) -> None:
        """ Create a new, empty Stack self.
        """
        self._key = -1
        self._storage = {}

    def add(self, obj: object)-> None:
        """ Add object obj to top of Stack self.
        """
        self._key += 1
        self._storage[self._key] = obj
```

Implementation using dictionary

```python
def remove(self) -> object:
    """
    Remove and return top element of Stack self.

    Assume Stack self is not empty, otherwise
    raises EmptyStackException
    >>> s = Stack()
    >>> s.add(5)
    >>> s.add(7)
    >>> s.remove()
    7
    """
    if self.is_empty():
        raise EmptyContainerException
    else:
        self._key -= 1
        return self._storage.pop(self._key + 1)
```

Implementation using dictionary

```
    def is_empty(self) -> bool:
        """
        Return whether Stack self is empty.
        >>> s = Stack()
        >>> s.is_empty()
        True
        >>> s.add(s)
        >>> s.is_empty()
        False
        """
        return len(self._storage) == 0


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Where Can I find the code presented in class

- You can find the full code for Stack as list and as dictionary in the course website under section **MWF2 (L0301)**
- with the following file names:
  - stack_as_dic.py
  - stack_as_list.py
- Download them Try different things with them and practice
  - Do not be afraid of doing mistakes