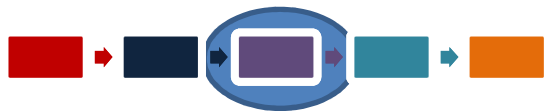


# CS61A Lecture 13

## *Object-Oriented Programming*

Jom Magrotker  
UC Berkeley EECS

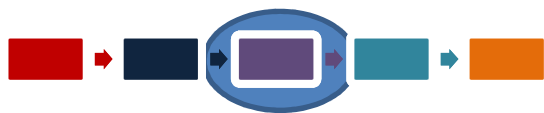
July 10, 2012



# THOUGHTS ON THE MIDTERM?

Using thumbs up/thumbs down...

What did everyone think of the midterm?



# COMPUTER SCIENCE IN THE NEWS

Home > IT Business > News

IT Business

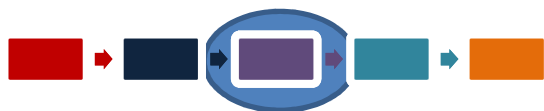


## Algorithmic pricing on Amazon 'could spark flash crash'

High-speed trading tools traditionally used on the stock market are helping to shape Amazon's price movements

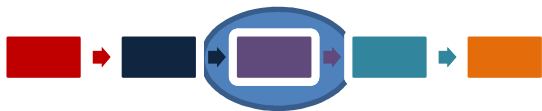
By *Derek du Preez* | *Computerworld UK* | Published 12:17, 09 July 12

<http://www.computerworlduk.com/news/it-business/3368640/algorithmic-pricing-on-amazon-could-spark-flash-crash/>



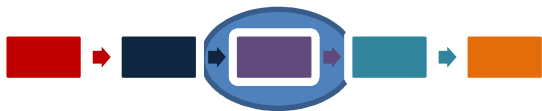
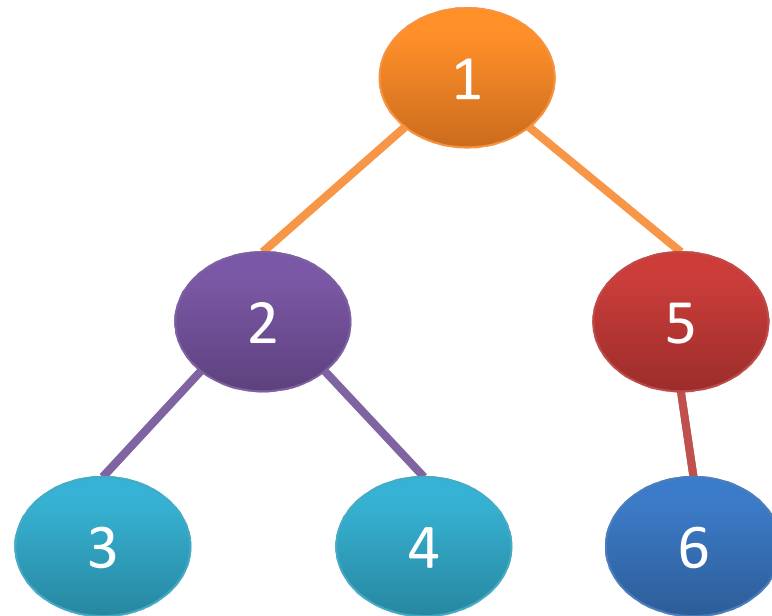
# TODAY

- Review: Binary Search Trees
- Demo: Project 2
- Object-Oriented Programming
  - Defining our own data types!
  - Data with functions!
  - State!



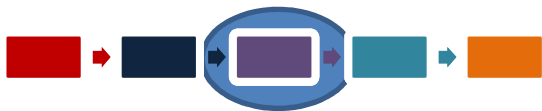
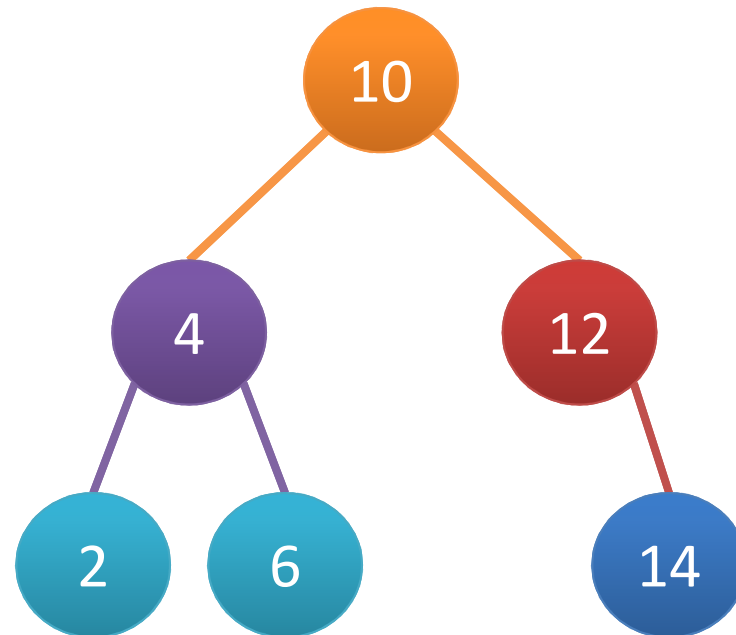
# REVIEW: BINARY TREES

Trees where each node has *at most* two children are known as ***binary trees***.



# REVIEW: BINARY SEARCH TREES

***Binary search trees*** are binary trees where all of the items to the ***left*** of a node are ***smaller***, and the items to the ***right*** are ***larger***.



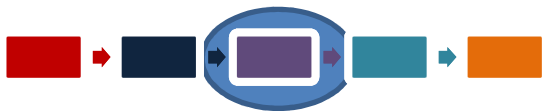
# REVIEW: BST ADT

```
empty_bst = None  
def make_bst(datum, left=empty_bst,  
            right=empty_bst):  
    return (left, datum, right)
```

```
def bst_datum(b):  
    return b[1]  
def bst_left(b):  
    return b[0]  
def bst_right(b):  
    return b[2]
```

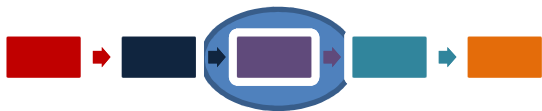
CONSTRUCTOR

SELECTORS



# REVIEW: SEARCHING WITH BSTs

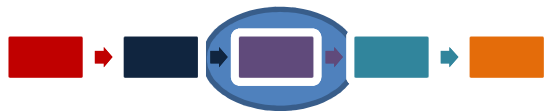
```
def bst_find(b, item):  
    if b == empty_bst:  
        return False  
    elif bst_datum(b) == item:  
        return True  
    elif bst_datum(b) > item:  
        return bst_find(bst_left(b),  
                        item)  
    return bst_find(bst_right(b),  
                    item)
```





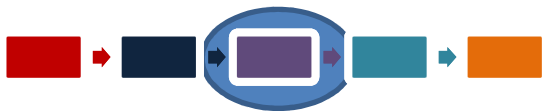
# ANNOUNCEMENTS

- Homework 6 is due today, **July 10**.
  - Starting with the next homework, we will mark questions as *core* and *reinforcement*.
  - The *core* questions are ones that we suggest you work on to understand the idea better.
  - The *reinforcement* questions are extra problems that you can practice with, but are not as critical.
- Homework 7 is due **Saturday, July 14**.
  - It will be released this afternoon.
- Make sure you fill out a survey!
  - You must give us a completed survey to get back your exam once it has been graded.



# PROJECT 2 DEMO

Get started if you haven't already!



# DEFINING OUR OWN DATA TYPES

So far, we've been defining new data types like this:

```
def make_account(owner, balance):  
    """Makes a bank account for the given owner with  
    balance dollars  
    """  
    return (owner, balance)
```

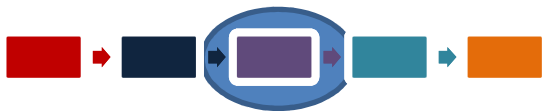
Constructor

```
def account_balance(acct):  
    """Return acct's balance"""  
    return acct[1]
```

```
def account_owner(acct):  
    """Return acct's owner"""  
    return acct[0]
```

Can we group all of this into a single description of the data type?

Selectors



# DEFINING OUR OWN DATA TYPES

```
class Account:
```

Define the new data type Account

```
def __init__(self, owner, balance):  
    self.__owner = owner  
    self.__balance = balance
```

Constructor

self refers to the specific *instance* of a class that we're manipulating.

```
def get_owner(self):  
    return self.__owner
```

```
def get_balance(self):  
    return self.__balance
```

Methods are operations that are associated with a class or object.

Selectors

Two underscores at the beginning of an instance variable means that the variable should only be accessed inside the class definition.

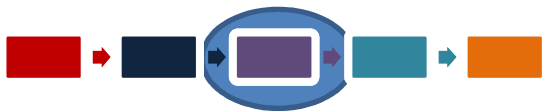


# OBJECT-ORIENTED PROGRAMMING: CREATING OBJECTS

We say that `toms_account` is an *instance* of the `Account` class.

```
>>> toms_account = Account("Tom", 50)
```

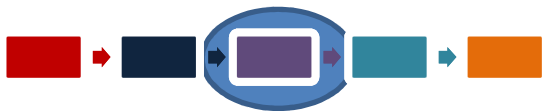
We call the process of creating an instance of a class *instantiation*.



# USING OUR NEW DATA TYPE

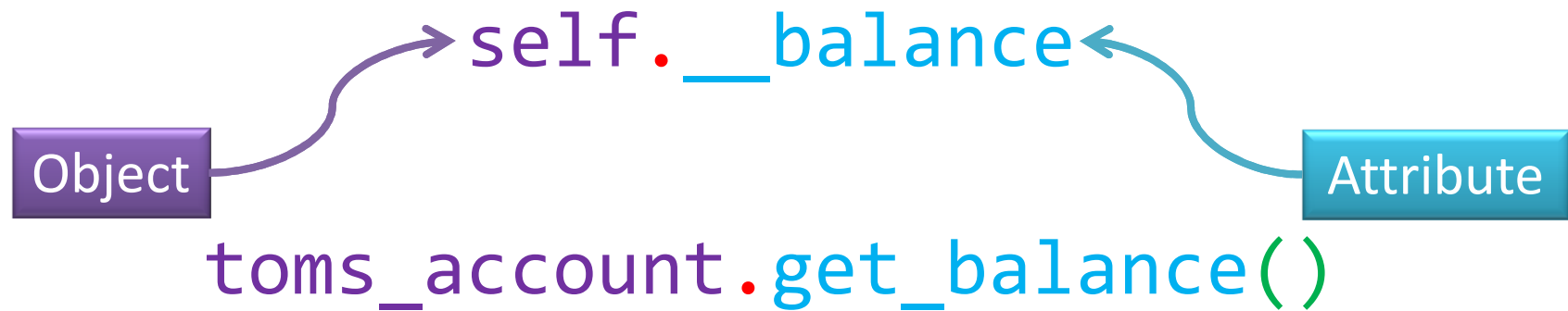
So how do we use it?

```
>>> toms_account = Account("Tom", 50)
>>> toms_account
<__main__.Account object at 0x01...>
>>> type(toms_account)
<class '__main__.Account'>
>>> Account.get_balance(toms_account)
50
>>> toms_account.get_balance()
50
>>> toms_account.get_owner()
"Tom"
```

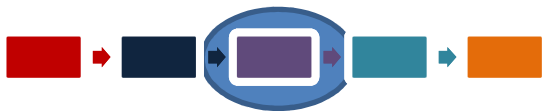


# DOT NOTATION

“Dot Notation” is used for referring to the data that is associated with a specific object.



As we saw on the previous slide, the dot notation also allows us to call an object's associated functions without having to pass the object itself as the first argument.



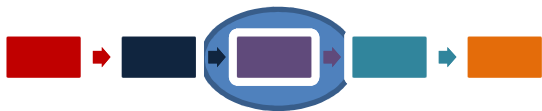
# OPERATING ON OUR DATA

How would we have implemented `withdraw` and `deposit` operations before?

```
def acct_withdraw(acct, amount):  
    new_balance = acct.get_balance() - amount  
    return Account(acct.get_owner(), new_balance)
```

```
def acct_deposit(acct, amount):  
    new_balance = acct.get_balance() + amount  
    return Account(acct.get_owner(), new_balance)
```

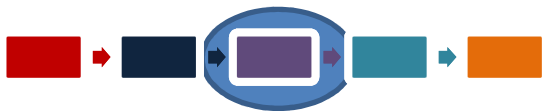
Can we associate the *operations* on our data type with the *definition* of the data type?





# OPERATING ON OUR DATA

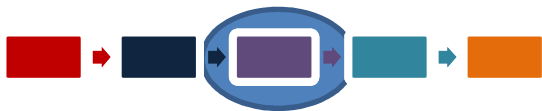
```
class Account:  
    def __init__(self, owner, balance):  
        self.__owner = owner  
        self.__balance = balance  
    def get_owner(self):  
        return self.__owner  
    def get_balance(self):  
        return self.__balance  
    def withdraw(self, amount):  
        self.__balance -= amount  
    def deposit(self, amount):  
        self.__balance += amount
```



# OPERATING ON OUR DATA

Using our operations for our accounts:

```
>>> toms_account = Account("Tom", 50)
>>> toms_account.get_balance()
50
>>> toms_account.withdraw(25)
>>> toms_account.get_balance()
25
>>> toms_account.deposit(500)
>>> toms_account.get_balance()
525
```



# OBJECT-ORIENTED PROGRAMMING

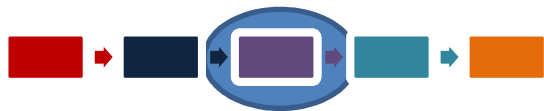
We now have a new *programming paradigm!*

Before – Functional Programming:

- Think of our program as a series of functions.
- Think in terms of inputs and outputs of each function.
- There is no change over time, no *state*.

Now – Object-Oriented Programming:

- Think of our program as a series of objects.
- Think in terms of the ways in which objects interact with each other using methods.
- Objects can change over time in our program. They have *state*.



# CONCLUSION

- Object-oriented programming: a brand new paradigm.
- **Preview:** Classes that inherit traits from other classes and variables shared by all data of the same class.

