CSC207H: Software Design
Lecture 4

Wael Aboelsaadat

wael@cs.toronto.edu

http://ccnet.utoronto.ca/20075/csc207h1y/

Office: BA 4261

Office hours: R 5-7

Acknowledgement: These slides are based on material by Prof. Karen Reid

Python: functions & classes

Functions

- A function is a reusable piece of a program.
- Functions are defined with `def`

```
>>> def square(x):
...     return x*x
>>> print square(8)
64
```

- Optional arguments:

```
>>> def power(x, exp=2):          # exp defaults to 2
...     if x <= 0: return 1
...     else: return x*power(x, exp-1)
```

Classes

- A class is a kind of object (like lists or strings) that contains variables and operations (or methods)
- The simplest class:

  ```
  >>> class Simple: pass
  ```

- Class objects are created with the constructor, which has the same name as the class:

  ```
  >>> obj = Simple()
  ```

- Variables are accessed as `obj.var`

  ```
  >>> obj.x = 3
  ```

An Example Class

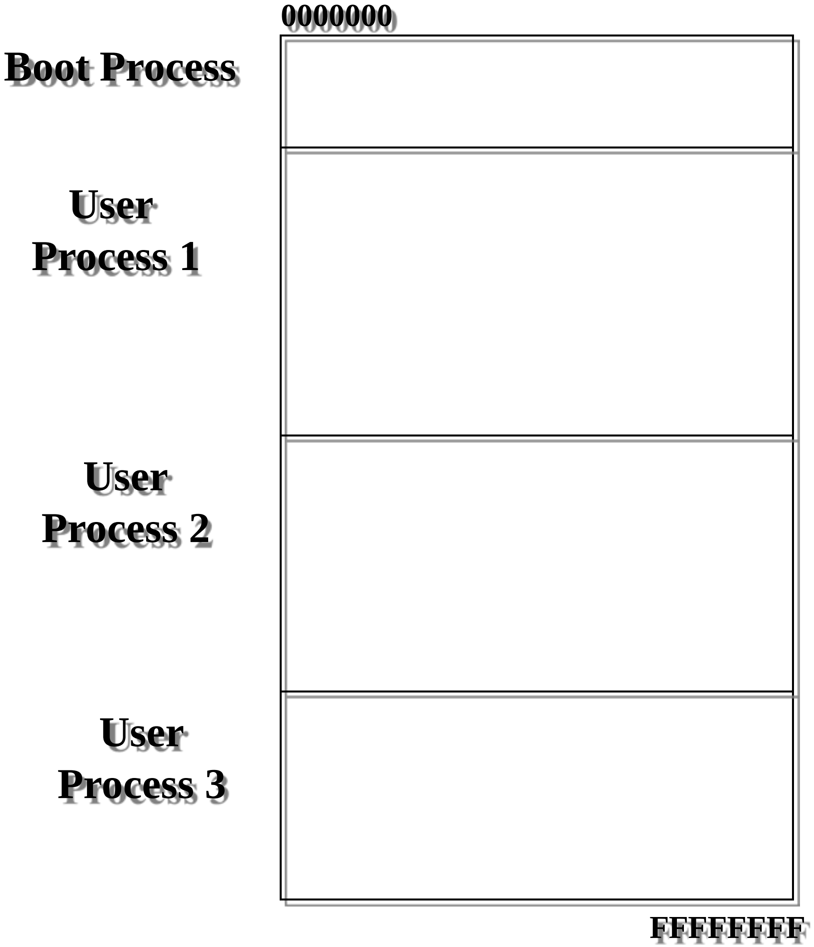
```
>>> class Account:
...      def __init__(self, initial):
...          self.balance = initial
...      def deposit(self, amt):
...          self.balance = self.balance + amt
...      def withdraw(self,amt):
...          self.balance = self.balance - amt
...      def getbalance(self):
...          return self.balance
```

- `__init__` defines the constructor
- `self` is the object that is being manipulated.
  - It is the first argument to every method.

Using the example class

```
>>> a = Account(1000.00)
>>> a.deposit(550.23)
>>> print a.getbalance()
1550.23
>>> a.deposit(100)
>>> a.withdraw(50)
>>> print a.getbalance()
1600.23
```

Why's the self?



0000000

Boot Process

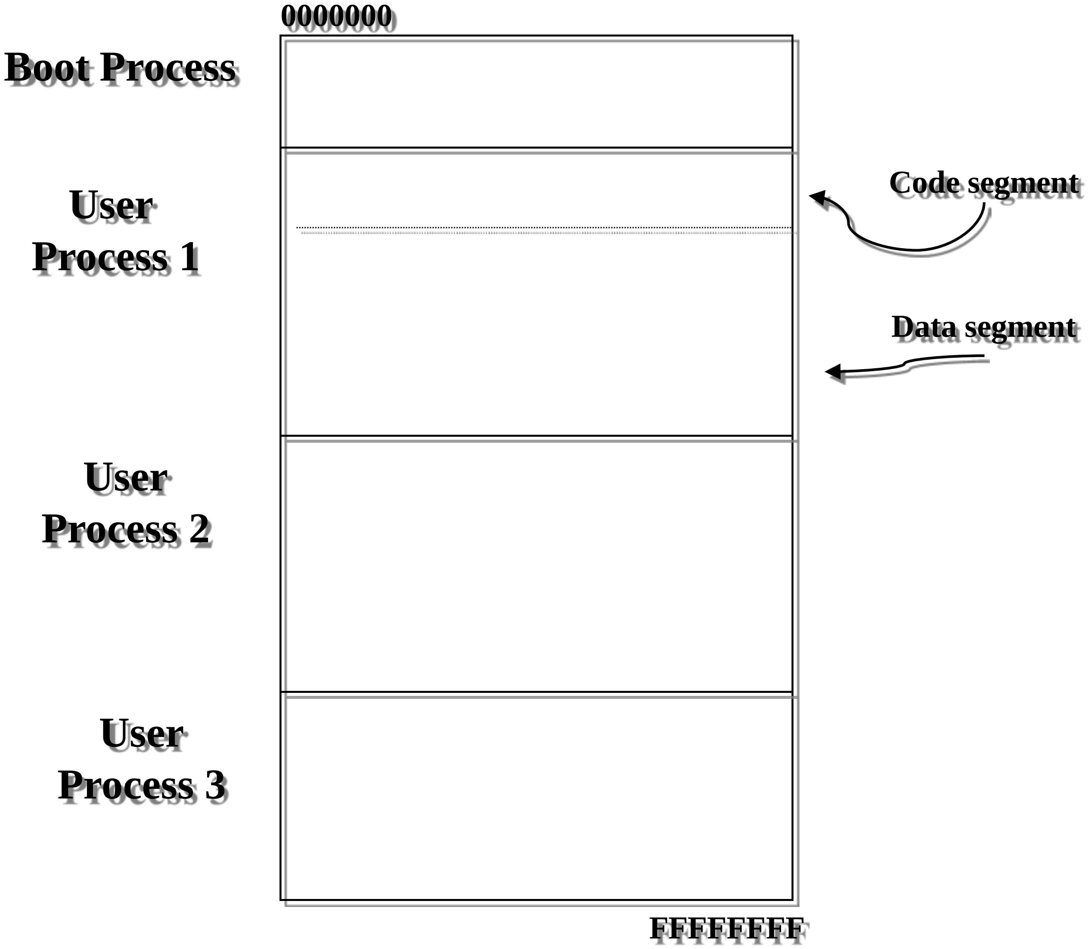
User
Process 1

User
Process 2

User
Process 3

FFFFFFFF

Why’s the self?



0000000

Boot Process

User Process 1

Code segment

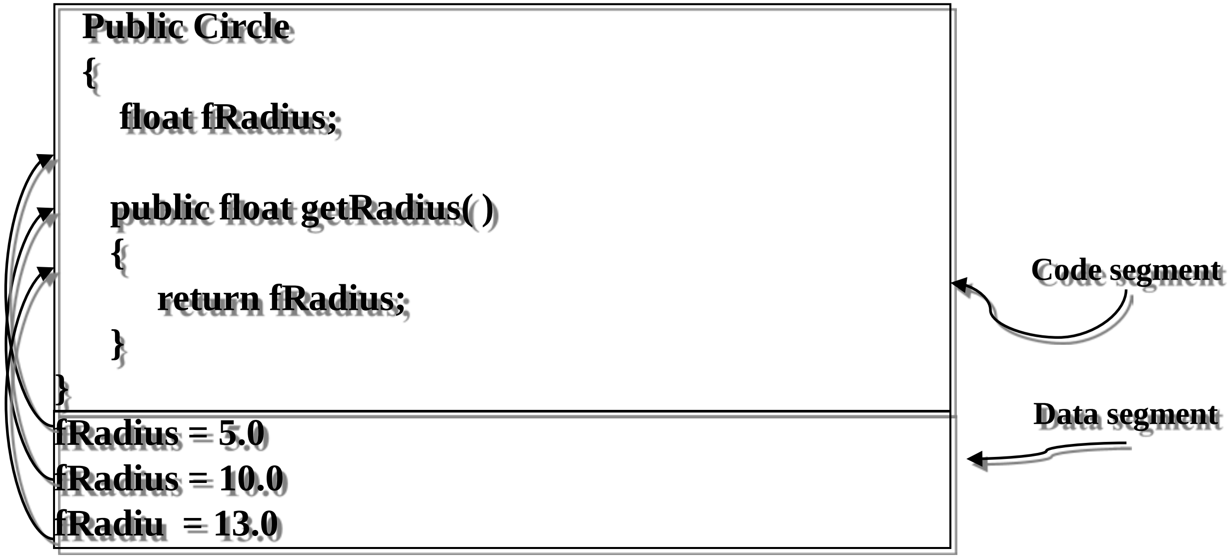
Data segment

User Process 2

User Process 3

FFFFFFFF

Why’s the self?



```
  Public Circle
  {
     float fRadius;

    public float getRadius( )
    {
        return fRadius;
    }
}
fRadius = 5.0
fRadius = 10.0
fRadiu  = 13.0
```

```
Circle circle1 = new Circle( 5.0 );
Circle circle2 = new Circle(10.0 );
Circle circle3 = new Circle( 13.0 );

System.out.println( circle1.getRadius( ) );

System.out.println( circle2.getRadius( ) );

System.out.println( circle3.getRadius( ) );
```

Creating a class in Python

- A source file may define any number of classes
  - Start definition using the class keyword followed by name of class
  - Contents of class are indented

Methods in Python

- Methods
  - Define using def; parameter list follows in parentheses
  - Indent body of method
  - No return type, no types for parameters
  - Finish at any time with return
    - Methods without return statement return None
  - Instance methods must have at least one parameter
    - The first parameter represents the particular instance of the class (i.e. this object)
    - Convention: call this parameter self (sort of like this in Java)
    - self is not given as an argument when this method is called (except within the class itself)

Creating a class in Python

- Class members (cont'd):
  - Methods (cont'd):
    - methods can be called anything
    - but method names beginning and ending with double underscore mean special things
      - For example, __init__ is the class’s constructor
  - Instance Variables:
    - Created by assignment to self.varname within a method
    - No declaration, just use!

Simple Counter class

```
class Counter:
  def __init__(self):
      self.value = 0
  def step(self):
      self.value += 1
  def current(self):
      return self.value

# Testing the class definition:
c = Counter()
print "initial value", c.current()
c.step()
print "after one step", c.current()
c.nonExistentMethod()
```

Output:

```
initial value 0
after one step 1
AttributeError Counter instance
has no attribute
'nonExistentMethod'
```

Encapsulation

- Python does not enforce encapsulation
  - No equivalent of *protected* or *private*
  - Anyone can happily execute
    - obj.value = "abc"
- Generally a bad idea
- Remember: the things that make it easy to write code quickly in Python make it harder to maintain.

Inheritance

- Extend a parent class to create a child class
  - put parent’s name in parentheses after child’s
- Must invoke parent’s constructor explicitly
  - Unlike Java, it can be called like any other method

```
– from counter import Counter
– class Stepper(Counter):
–   def __init__(self):
–      Counter.__init__(self)
–   def reset(self):
–     self.value = 0
```

Example: overriding Counter

- Methods defined in child take precedence over those defined in parent

Example: overriding Counter

```
class Incrementer(Counter):
  def __init__(self, increment=1):
     Counter.__init__(self)
     self.increment = increment
  def step(self):
     self.value += self.increment

# Test the class:
obj = Counter()
for i in range(2):
   obj.step()
   print "Counter (parent) ", i, ":",obj.current()
obj = Incrementer(3)
for i in range(2):
   obj.step()
   print "Incrementer (child) ", i, ":",obj.current()
```

Output:

```
Ctr (parent) 0 : 1
Ctr (parent) 1 : 2
Incrnter (child) 0 : 3
Incrnter (child) 1 : 6
```

Example: __add__

- Specially-named methods associated with every arithmetic operator
  - __add__ for +
  - __mul__ for *
  - If x is an object, x+2 is really x.__add__(2)
- Operators also have right-hand methods
  - *E.g.* __radd__, __rmul__
  - So 2+x is x.__radd__(2)
- Execution order for a+b is:
  - If a has a method __add__, call a.__add__(b)
  - If b has a method __radd__, call b.__add__(a)
  - Else use Python’s built-in default

Example: __add__

```
# modInt: only has values in the range 0..base-1
class modInt:
    def __init__(self, base):
        self.base = base
        self.value = 0
    def __add__(self, other):
        self.value += other
        self.value %= self.base
        return self
    def val(self):
        return self.value

a = modInt(3)
    for i in range(5):
        a = a + 1
        print a.val(),
```

1 2 0 1 2

Some other __special__ methods

__str__(self)	Convert to string
__getitem__(self, index)	Indexing([])
__contains__(self, item)	Membership test (in)
__len__(self)	Length (len)
__int__(self)	Convert to integer (int)

Python and Java: differences

- Java:
  - each file is a class
  - execution starts with the main method of the class that is loaded first
- Python:
  - no need for classes in a file
  - execution starts with the first executable statement in a file
  - The execution of a class indentation block is storing the set of statements that define the class
  - Similarly, a def indentation block inside a class stores the definition of a method

Creating and loading modules

- Any Python file can be loaded as a module using import module
  - File called xyz.py becomes module xyz
- Statements are executed as module loads
  - Libraries typically just define constants and functions
- Module contents referred to as module.content
  - E.g. sys.argv
- Can also use
  - from module import name1, name2
  - from module import *

Module: example

- # stuff.py
- value = 123
- def printVersion():
-     print "Stuff Version 2.2 "

- # loader.py
- import stuff
- print stuff.value
- stuff.printVersion()

- *$ python stuff.py*
- *$ python loader.py*
- ***123***
- ***Stuff Version 2.2***

Python Sequences

Lists

- List: a mutable sequence of objects
- mutable: can be changed
- sequence: can be indexed (start at 0)
- Same idea as the List interface in Java
- *A Python list is a heterogeneous collection*
  - This is a fancy (but quick) way of saying that its contents need not all be the same type: a list can contain just about anything

Syntax

- Elements are inside square brackets separated by commas:
  - lst = [1, 'Fred', 2, [], '999']
- List elements can be referred to by index:
  - print lst[0], lst[3]
  - print lst[5]

- **1 []**
- **IndexError: list index out of range**

Updating lists

- Modify lists by assigning to their elements
- Built-in function len() returns length of sequence

```
– x = ['a', 'b', 'c', 'd']
– i = 0
– while i < len(x):
–     x[i] = i
–     i += 1
– print x
```

  - ***[0, 1, 2, 3]***

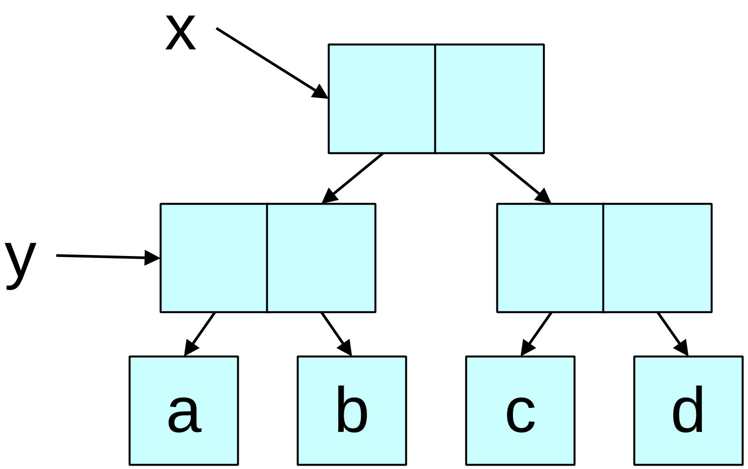
Nesting lists

- Lists of lists of lists of …
- Literals: [[1, 2], [3, 4]]
- Index from the outside in
  - x = [[13, 17, 19], [23, 29]]
  - print x[1]
  - print x[0][1:3]

  - ***[23, 29]***
  - ***[17, 19]***

Indexing hands back actual value

- Nested lists are objects in their own right
- Outer list points to inner list



```
x = [["a", "b"], ["c", "d"]]
y = x[0]
y[0] = 123
print y
print x
```

[123, “b”]
[[123, “b”], [“c”, “d”]]

Adding lists

- Adding lists concatenates them
- You can multiply a list by an integer (recall multiplying the string "ho" by 3)
  - x = ["a", "b"] + ["c", "d"]
  - y = 2 * x
  - print x
  - print y
  - **['a', 'b', 'c', 'd']**
  - **['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']**

Strings

- An immutable sequence of characters
- No separate character type
- Immutable: cannot be modified in place
  - Safety
  - Efficiency

String indexing

```
element = "boron"
i = 0
while i < len(element):
    print element[i]
    i += 1
```

- ***b***
- ***o***
- ***r***
- ***o***
- ***n***

String methods

Strings are objects

(Yes, it does look a lot like Java, doesn't it?)

s.capitalize()	Capitalize the first letter.
s.lower()	Convert all letters to lower case.
s.strip()	Remove leading and trailing white space.
s.rstrip()	Remove trailing (right-hand) white space.
s.upper()	Convert all letters to upper case.
s.count(pat, start, end)	Count occurrences of pat; start and end optional.
s.find(pat, start, end)	Return index of first occurrence of pat, or -1; start and end optional.
s.replace(old, new, limit)	Replace occurrences of old with new; limit is optional.

Negative string indices

- Negative indices count backward from the end of the string
  - x[-1] is the last character
  - x[-2] is the second-last character

- Example:
  - val = "carbon"
  - print val[-2], val[-4], val[-6]

  - ***o r c***

Negative list indices, and a slice

- Python sequence indices allow manipulations that we don't have in Java
- Negative indices
  - Negative indices count backward from the end of the string or other sequence:
- Indexed just like strings
  - x = ["a", 2, "bcd"]
  - print x[0], x[-1], x[1:-2]

  - ***a bcd []***

For loops

- Python’s for loop works like Java's new for loop
  - for item in collection
    - sets item to each element of collection in turn
  - for c in "lead":
    - print "[" + c + "] ",
  - print
  - ***[l] [e] [a] [d]***

Breaking and continuing

- End loop prematurely using break
  - only exits one level of loop
- Use continue to skip immediately to the next iteration of the loop
- (Java and Python inherited these from C)
  - for element in aVeryLongList:
    - if element < 0:
      - break
    - print element

Membership

- x in c is True if the value x is in the collection c
  - Works on all collections
  - Uses linear search on sequences

  - vowels = "aeiou"
  - for v in vowels:
    - if v in "uranium":
      - print v

a
i
u

Python Dictionaries and Functions

Dictionaries

- Another name for maps
    - Also called hashes and associative arrays
- Built into the language
    - Handy to be able to just write them

Creating and indexing

- Create by putting key/value pairs inside {}
  - birthdays = {"Newton":1642,
"Darwin":1809}
- Empty dictionary written as {}
- Index using []
  - print birthdays["Darwin"]
  - ***1809***

Access

- Can only access keys that are present
  - birthdays = {"Newton":1642,"Darwin":1809}
  - print birthdays["Turing"]
  - ***KeyError: Turing***
- Test for presence of key using k in d
  - if "Turing" in birthdays:
    - print birthdays["Turing"]
  - else:
    - print "Who?"
  - ***Who?***

Getting Help

- The pydoc module can be used to get information about objects, functions, etc.

```
>>> from pydoc import help
>>> help(re)
```

- pydoc can also be used from the command line, to provide manpage-like documentaiton for anything in Python:

```
% pydoc re
```

- dir() lists all operations and variables contained in an object (list, string, etc):

```
>>> dir(re)
[‘DOTALL’, ‘I’, …, ‘split’, ‘sub’, ‘subn’, ‘template’]
```