Objectives

- Review: Precedence/Arithmetic, importing modules
- Definite for loops

Exponentiation

- Goal: compute -3^2
  - Suggested: **pow**(-3, 2)
    - pow is a built-in function
  - How else could we get that?
- For fun, what is 2 ** -3 ** 2

Python Libraries

- Python has a rich library of functions and definitions available for your use
  - The library is broken into **modules**
  - A **module** is a file containing Python definitions and statements
- Benefits of functions/definitions in modules
  - Don't need to rewrite someone else's code
  - If it's in a module, it is a very efficient (in terms of computation speed and memory usage)

Importing Modules

- To use the definitions in a module, you must first **import** the module
  - Example: to use the **math** module's definitions, use the the import statement: import math
  - Typically import statements are at *top* of program
- To use, prepend constant or function with "modulename."
  - Examples for constants:
    - math.e
  - Examples for functions:
    - math.sqrt

module_example.py

Using Modules

- Alternatively can import only a subset of the module:
  - Syntax:

  from <library> **import** <name1>, <name2>, …
  - Example:

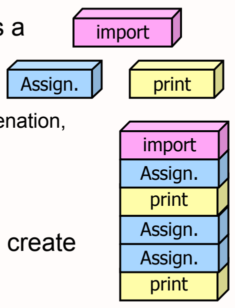
  from math **import** pi
  - Then, can use just pi instead of math.pi in program

Finding Modules To Use

- How do I know if some code that I want already exists?
  - Python Library Reference:
  - http://docs.python.org/lib/lib.html
- For example, `string` module has functions/constants for manipulating strings
- For the most part, to practice, in the beginning you will write most of your code from scratch

Programming Building Blocks

- Each type of statement is a building block
  - Initialization/Assignment
    - Arithmetic, string concatenation, input/raw_input
  - Print
  - Import
- We can combine them to create more complex programs
  - Solutions to problems

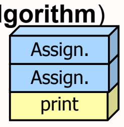


Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution

Design Patterns

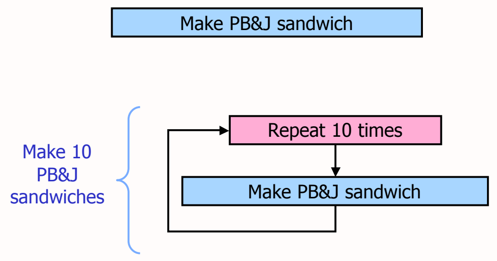
- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution
- Example (**Standard Algorithm**)
  - Get input from user
  - Do some computation
  - Display output



```
x = input("…")
ans = …
print ans
```

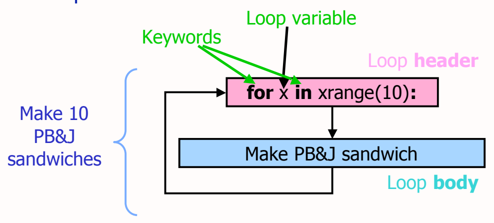
- **Today**: learn new building block, new design pattern

Looping/Repetition



The `for` Loop

- Good for when know how many times loop will execute
  - Repeat N times



Using the `For` Loop

- Good for when know how many times loop will execute
  - Repeat N times

Times to repeat

```
for x in xrange(10):
        statement_1
        statement_2
        …
        statement_n
```

- "Body" of for loop
- Gets repeated
- Note indentation

Using the `For` Loop

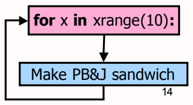
- If only ***one*** statement to repeat

```
for x in xrange(5): print "Hello!"
```

simple_for.py

What Goes in the Loop Body?

- Make PB&J Sandwich
  - Gather materials (bread, PB, J, knives, plate)
  - Open bread
  - Put 2 pieces of bread on plate
  - Spread PB on one side of one slice
  - Spread Jelly on one side of one slice
  - Place PB-side facedown on Jelly-side of bread
  - Close bread
  - Clean knife
  - Put away materials



What Goes in the Loop Body?

- Make PB&J Sandwich
  - Gather materials (bread, PB, J, knives, plate)
  - Open bread

Initialization

Loop Body

  - Put 2 pieces of bread on plate
  - Spread PB on one side of one slice
  - Spread Jelly on one side of one slice
  - Place PB-side facedown on Jelly-side of bread

Finalization

  - Close bread
  - Clean knife
  - Put away materials

Using the `For` Loop

- Good for when know how many times loop will execute
  - Repeat N times

Times to repeat

```
for x in xrange(10):
        statement_1
        statement_2
        …
        statement_n
```

- "Body" of for loop
- Gets repeated
- Note indentation

Analyzing `xrange()`

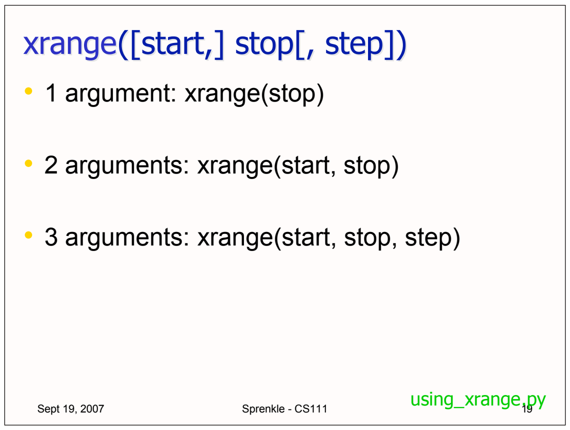
- `xrange` is a built-in function

- What does `xrange` do, exactly?

xrange_analysis.py

xrange([start,] stop[, step])

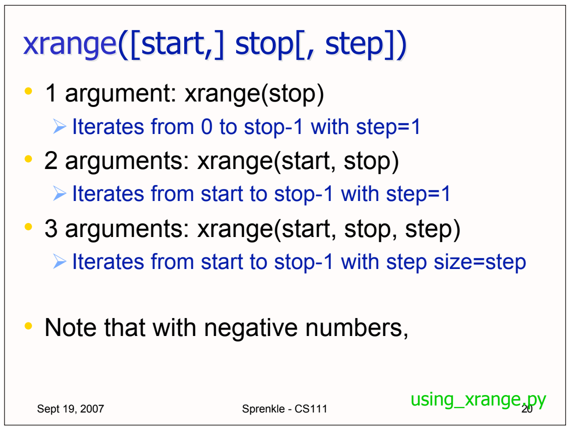
- What does the above signature mean?



xrange([start,] stop[, step])

- 1 argument: xrange(stop)
- 2 arguments: xrange(start, stop)
- 3 arguments: xrange(start, stop, step)

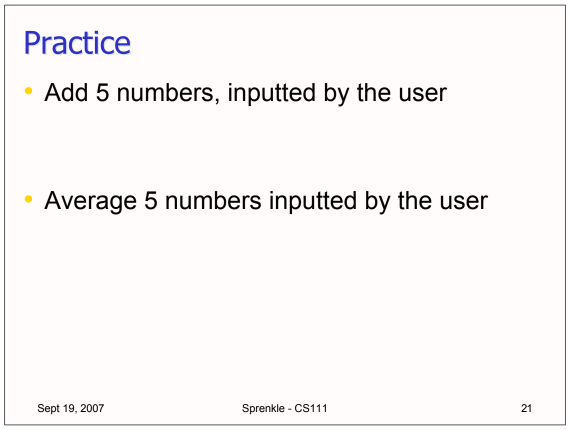
using_xrange.py



xrange([start,] stop[, step])

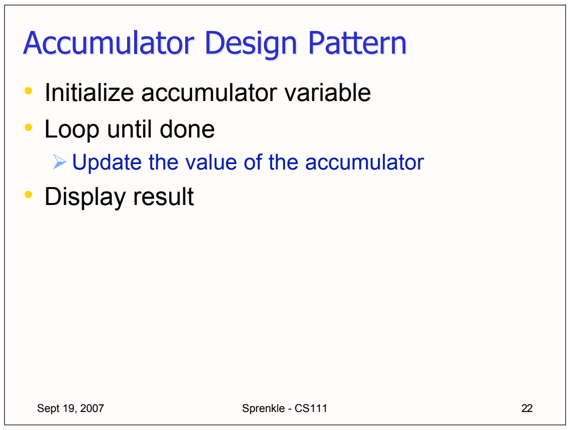
- 1 argument: xrange(stop)
  - Iterates from 0 to stop-1 with step=1
- 2 arguments: xrange(start, stop)
  - Iterates from start to stop-1 with step=1
- 3 arguments: xrange(start, stop, step)
  - Iterates from start to stop-1 with step size=step
- Note that with negative numbers,

using_xrange.py



Practice

- Add 5 numbers, inputted by the user
- Average 5 numbers inputted by the user



Accumulator Design Pattern

- Initialize accumulator variable
- Loop until done
  - Update the value of the accumulator
- Display result