Objectives

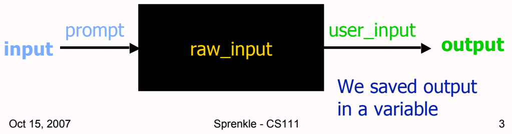
- Creating your own functions

Why write functions?

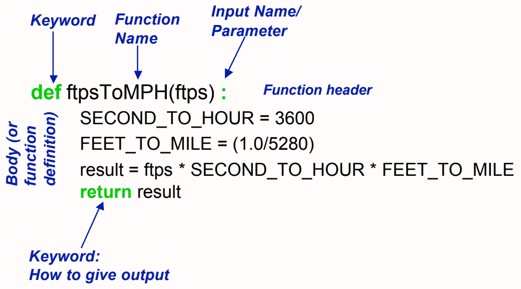
- Allows you to break up a hard problem into smaller, more manageable parts
- Makes your code easier to understand
- Hides implementation details (*abstraction*)
  - Provides interface (input, output)
- Makes part of the code reusable so that you:
  - Only have to type it out once
  - Can debug it all at once
    - Isolates errors
  - Can make changes in one function (maintainability)

Functions

- Function is a **black box**
  - Implementation doesn't matter
  - Only care that function generates appropriate output, given appropriate input
- Example:
  - Didn't care how **raw_input** function was implemented



Syntax of Function Definition



```
def ftpsToMPH(ftps) :
    SECOND_TO_HOUR = 3600
    FEET_TO_MILE = (1.0/5280)
    result = ftps * SECOND_TO_HOUR * FEET_TO_MILE
    return result
```

Where are functions in the code?

- Can be defined in script before use (calling it)
- Could be in separate **module**
  - Import to use in script
  - Example: menu.py
  - Define in modules when functions are reusable for many different programs
    - Benefits: shorter code (no function defns), isolate testing of function, write "test driver" scripts to test functions separately from use in script

Parameters

- The inputs to a function are called ***parameters*** or ***arguments***
- When ***calling***/using functions, parameters must appear in same order as in the function header
  - Example: round(x, n)
    - **x** is float to round
    - **n** is integer of decimal places to round to

Parameters

- **Formal Parameters** are the variables named in the the function definition.
- **Actual Parameters** are the variables or literals that really get used when the function is called.

Formal *Actual*

def round(x, n) :
roundCelc = round(celc,2)

Formal & actual parameters must match in order, number, and type!

Practice: Old McDonald

- A verse of the song goes

  Old McDonald had a farm, E-I-E-I-O
  And on that farm he had a dog, E-I-E-I-O
  With a ruff, ruff here
  And a ruff, ruff there
  Here a ruff, there a ruff, everywhere a ruff, ruff
  Old McDonald had a farm, E-I-E-I-O

- Write a function to print a verse
  - Why does it make sense to write a function for the verse?
  - What is input?
  - What is output?

Function Output

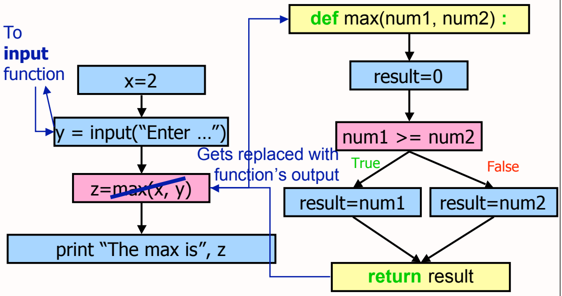
- When the code reaches a statement like

  return x

  x is the output returned to the place where function called and the function stops
  - For functions that don't have explicit output, return does not have a value with it
    - **return**
    - Optional: don't need to have output/return

Flow of Control

num1 gets the value of x
num2 gets the value of y



To **input** function

x=2

y = input("Enter ...")

z=~~max(x, y)~~

print "The max is", z

Gets replaced with function's output

def max(num1, num2) **:**

result=0

num1 >= num2

result=num1

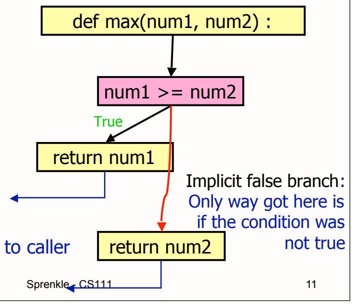
result=num2

return result

Flow of Control: Using return

```
def max(num1, num2) :
    if num1 >= num2 :
        return num1
    return num2
```

```
x=2
y=6
z = max( x, y )
```



def max(num1, num2) :

num1 >= num2

return num1

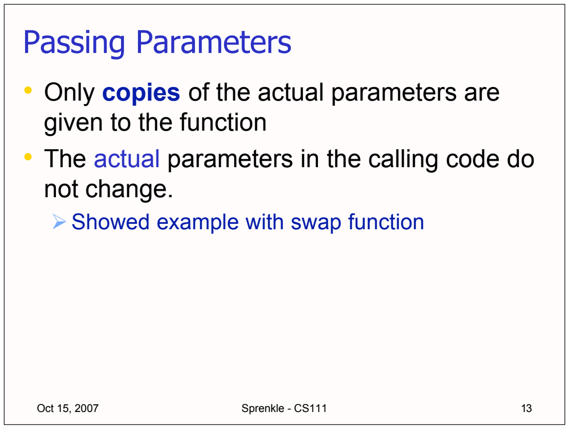
Implicit false branch:
Only way got here is if the condition was not true

return to caller

return num2

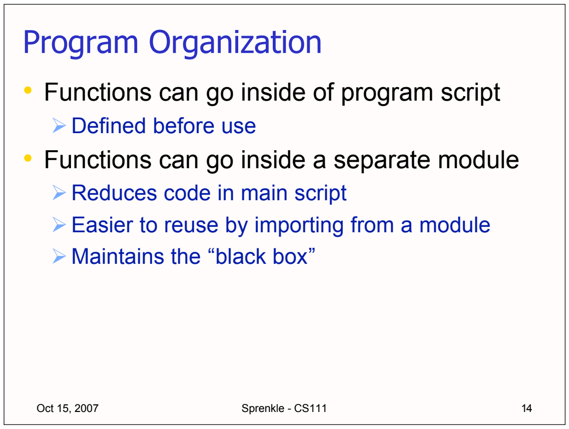
Using return

- Use return to "shortcut" function
  - Return output as soon as know answer
  - Compare efficiency of two functions in binaryToDecimal.py



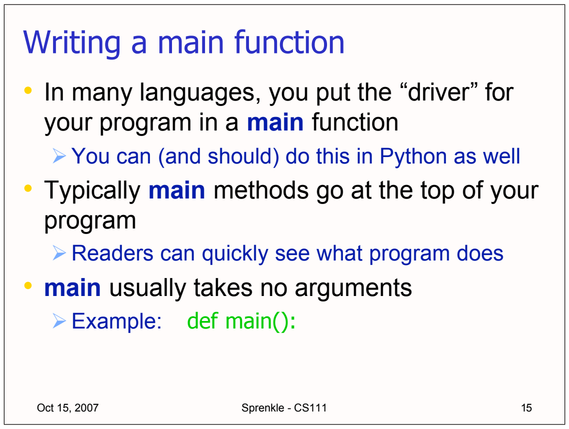
Passing Parameters

- Only **copies** of the actual parameters are given to the function
- The actual parameters in the calling code do not change.
  - Showed example with swap function



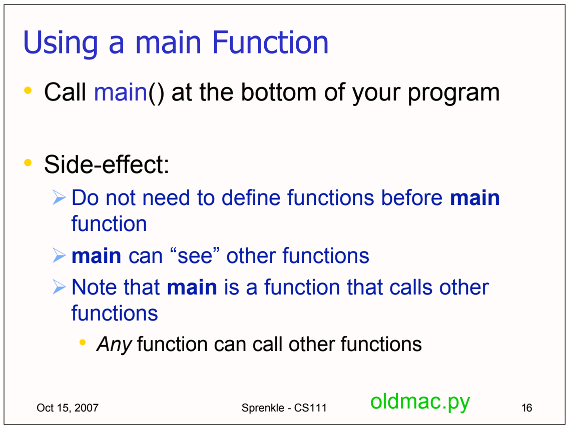
Program Organization

- Functions can go inside of program script
  - Defined before use
- Functions can go inside a separate module
  - Reduces code in main script
  - Easier to reuse by importing from a module
  - Maintains the “black box”



Writing a main function

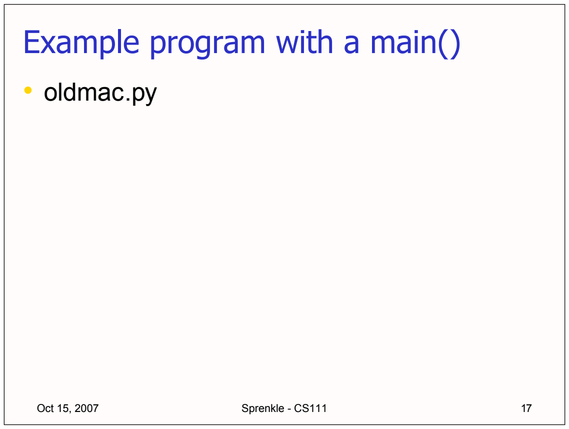
- In many languages, you put the “driver” for your program in a **main** function
  - You can (and should) do this in Python as well
- Typically **main** methods go at the top of your program
  - Readers can quickly see what program does
- **main** usually takes no arguments
  - Example: def main():



Using a main Function

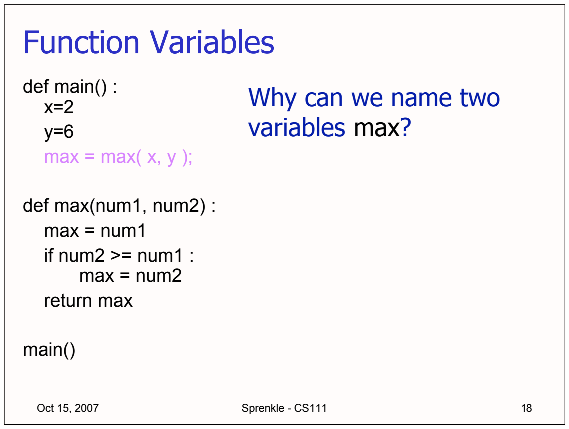
- Call main() at the bottom of your program
- Side-effect:
  - Do not need to define functions before **main** function
  - **main** can “see” other functions
  - Note that **main** is a function that calls other functions
    - *Any* function can call other functions

oldmac.py



Example program with a main()

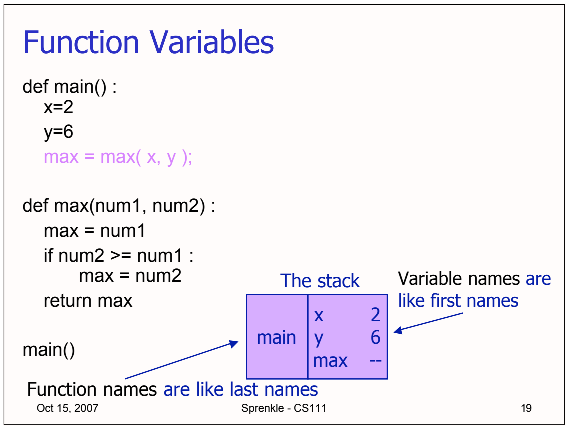
- oldmac.py



Function Variables

```
def main() :
   x=2
   y=6
   max = max( x, y );

def max(num1, num2) :
   max = num1
   if num2 >= num1 :
        max = num2
   return max

main()
```

Why can we name two variables max?



Function Variables

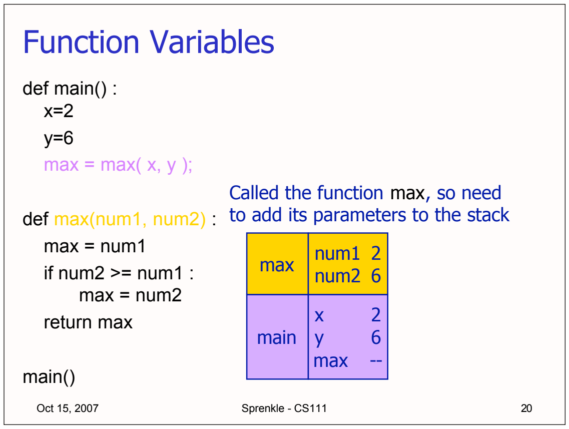
```
def main() :
   x=2
   y=6
   max = max( x, y );

def max(num1, num2) :
   max = num1
   if num2 >= num1 :
        max = num2
   return max

main()
```

The stack

main	x	2
	y	6
	max	--

Variable names are like first names

Function names are like last names

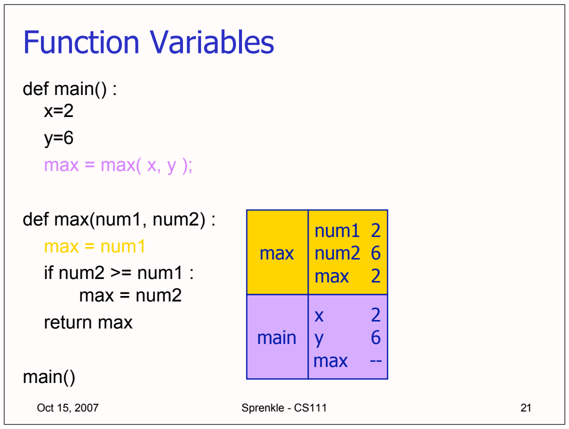


Function Variables

```
def main() :
   x=2
   y=6
   max = max( x, y );

def max(num1, num2) :
   max = num1
   if num2 >= num1 :
        max = num2
   return max

main()
```

Called the function max, so need to add its parameters to the stack

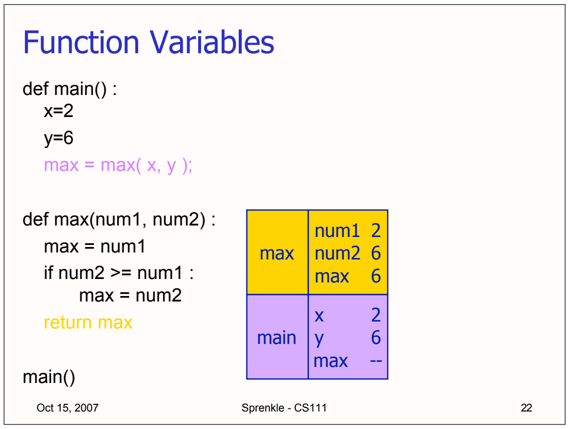
max	num1	2
	num2	6
main	x	2
	y	6
	max	--



Function Variables

```
def main() :
   x=2
   y=6
   max = max( x, y );

def max(num1, num2) :
   max = num1
   if num2 >= num1 :
        max = num2
   return max

main()
```

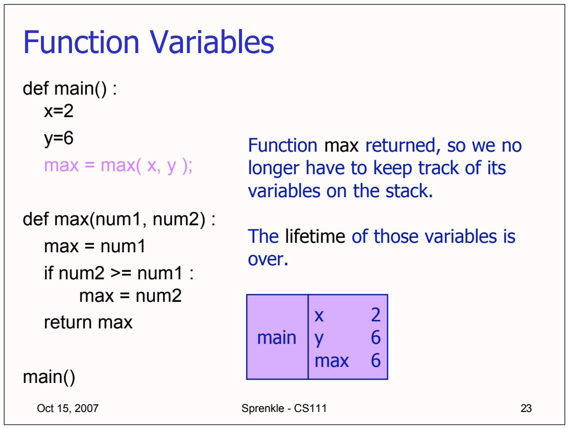
max	num1	2
	num2	6
	max	2
main	x	2
	y	6
	max	--



Function Variables

```
def main() :
   x=2
   y=6
   max = max( x, y );

def max(num1, num2) :
   max = num1
   if num2 >= num1 :
        max = num2
   return max

main()
```

max	num1	2
	num2	6
	max	6
main	x	2
	y	6
	max	--



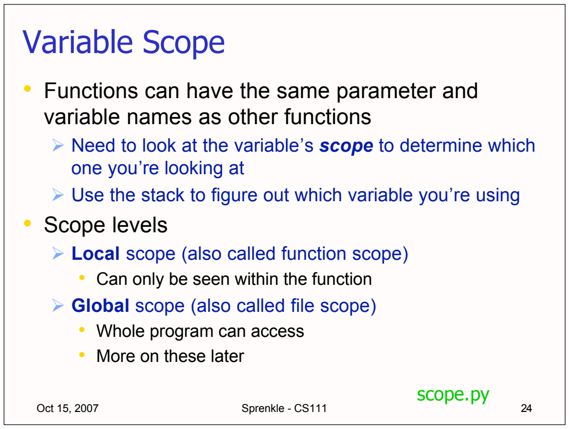
Function Variables

```
def main() :
   x=2
   y=6
   max = max( x, y );

def max(num1, num2) :
   max = num1
   if num2 >= num1 :
        max = num2
   return max

main()
```

Function max returned, so we no longer have to keep track of its variables on the stack.

The lifetime of those variables is over.

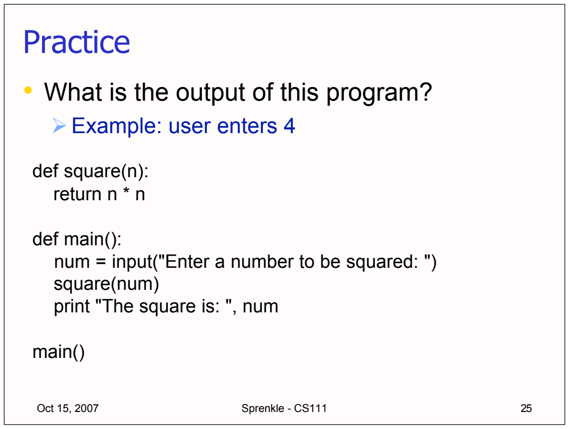
main	x	2
	y	6
	max	6



Variable Scope

- Functions can have the same parameter and variable names as other functions
  - Need to look at the variable's ***scope*** to determine which one you're looking at
  - Use the stack to figure out which variable you're using
- Scope levels
  - **Local** scope (also called function scope)
    - Can only be seen within the function
  - **Global** scope (also called file scope)
    - Whole program can access
    - More on these later

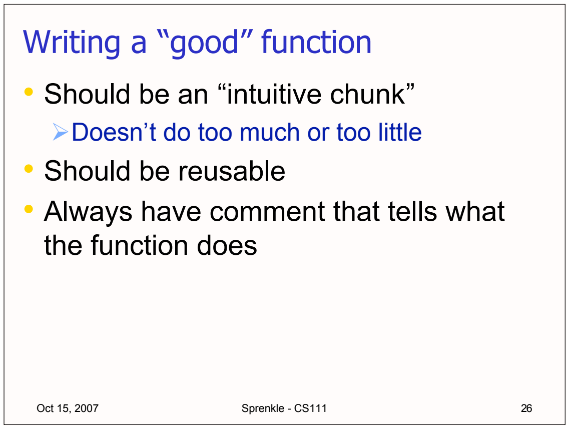
scope.py



Practice

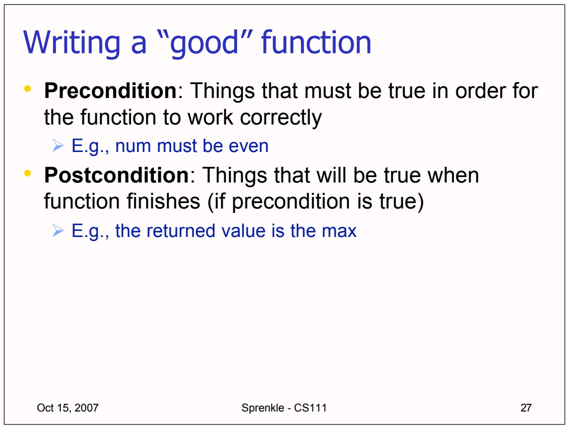
- What is the output of this program?
  - Example: user enters 4

```
def square(n):
    return n * n

def main():
    num = input("Enter a number to be squared: ")
    square(num)
    print "The square is: ", num

main()
```



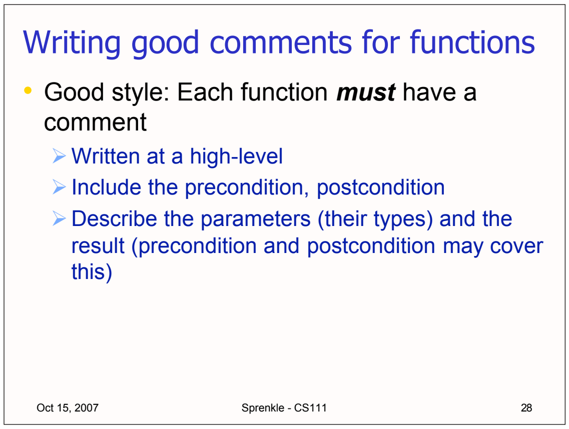
Writing a "good" function

- Should be an "intuitive chunk"
  - Doesn't do too much or too little
- Should be reusable
- Always have comment that tells what the function does



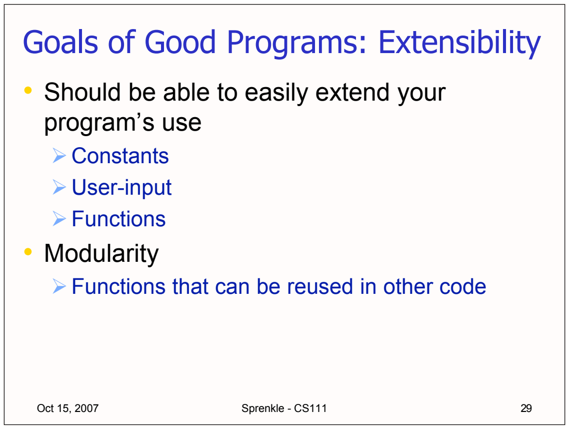
Writing a "good" function

- **Precondition**: Things that must be true in order for the function to work correctly
  - E.g., num must be even
- **Postcondition**: Things that will be true when function finishes (if precondition is true)
  - E.g., the returned value is the max



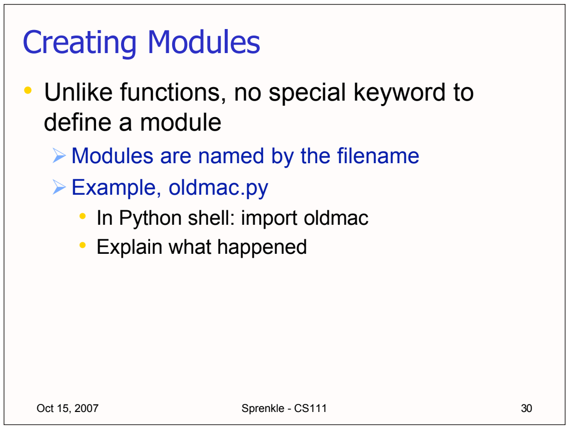
Writing good comments for functions

- Good style: Each function ***must*** have a comment
  - Written at a high-level
  - Include the precondition, postcondition
  - Describe the parameters (their types) and the result (precondition and postcondition may cover this)



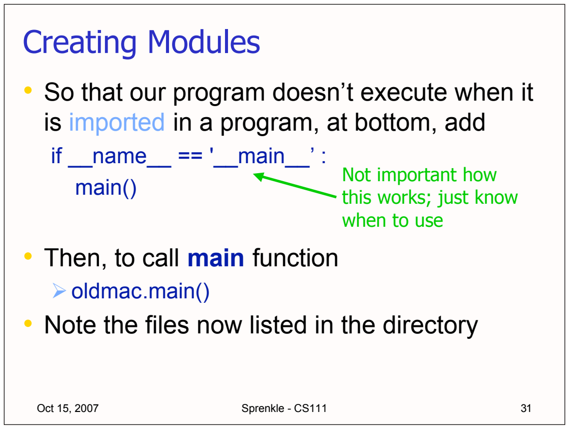
Goals of Good Programs: Extensibility

- Should be able to easily extend your program's use
  - Constants
  - User-input
  - Functions
- Modularity
  - Functions that can be reused in other code



Creating Modules

- Unlike functions, no special keyword to define a module
  - Modules are named by the filename
  - Example, oldmac.py
    - In Python shell: import oldmac
    - Explain what happened



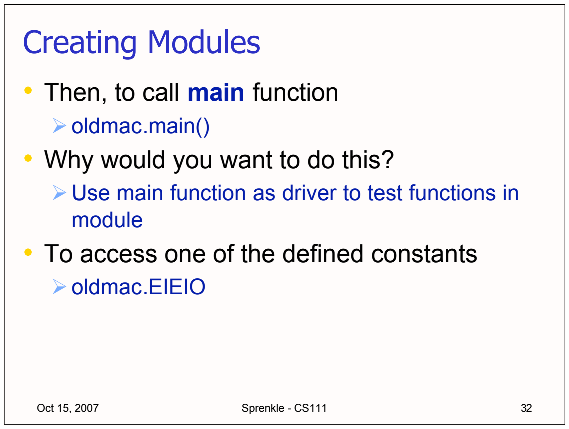
Creating Modules

- So that our program doesn’t execute when it is imported in a program, at bottom, add

```
if __name__ == '__main__' :
    main()
```

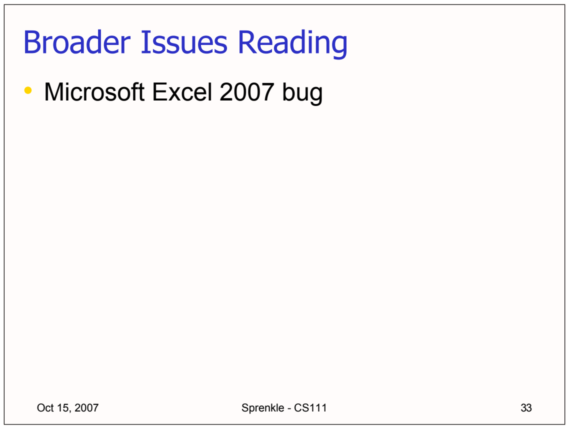
Not important how this works; just know when to use

- Then, to call **main** function
  - oldmac.main()
- Note the files now listed in the directory



Creating Modules

- Then, to call **main** function
  - oldmac.main()
- Why would you want to do this?
  - Use main function as driver to test functions in module
- To access one of the defined constants
  - oldmac.EIEIO



Broader Issues Reading

- Microsoft Excel 2007 bug