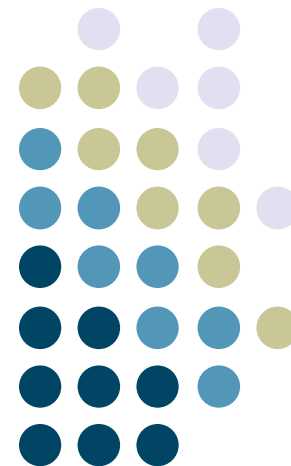


Introduction to Python (All the Basic Stuff)



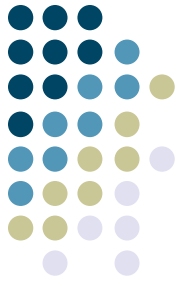
CS 6452
John Stasko

Learning Objectives

- Python program development
 - Command line, IDEs, file editing
- Language fundamentals
 - Types & variables
 - Expressions
 - I/O
 - Control flow
 - Functions & scoping

Python Environment

- You've gotten Python on your machine
- Run from command line
OR
- Use an Integrated Development Environment (IDE)



Command Line (Windows)

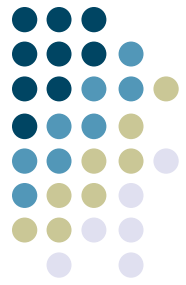
```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\John>python
Python 2.7.7 (default, Jun 1 2014, 14:21:57) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Windows command shell

Python command

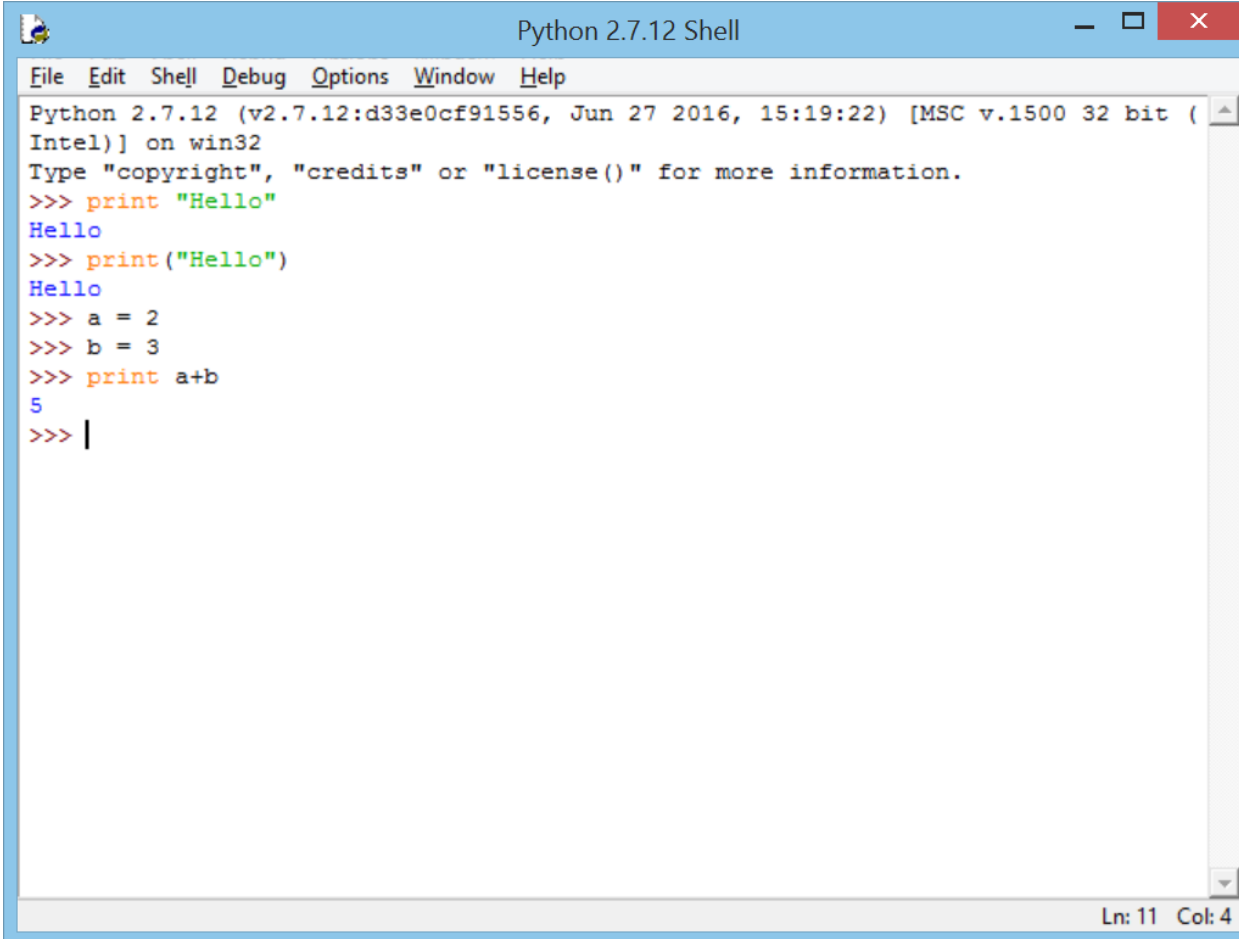
```
C:\Python27\python.exe
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```



Command Line (Mac)

```
Terminal - Python - 80x34
Last login: Thu Aug 25 12:19:05 on console
caledonia% python
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 3
>>> print(x)
3
>>> █
```

IDLE IDE

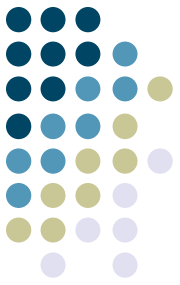
A screenshot of a Python 2.7.12 Shell window. The window title is "Python 2.7.12 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the following code and output:

```
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print "Hello"
Hello
>>> print("Hello")
Hello
>>> a = 2
>>> b = 3
>>> print a+b
5
>>> |
```

The status bar at the bottom right indicates "Ln: 11 Col: 4".

Other Python IDEs

- Pydev (Eclipse)
- PyCharm
- Wing
- Spyder Python
- ...

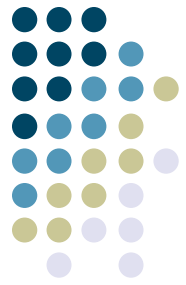


Creating/Editing Programs

- Text editor: Notepad, WordPad, Sublime text, Emacs, ...

```
C:\Users\John\Desktop\Python\family.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
family.py
1 import random
2
3
4 def main():
5     print "Starting Family Simulator"
6     keep_going = 'y'
7     while keep_going == 'y':
8         new_run()
9         keep_going = raw_input("Run again? (y/n) ")
10
Starting Family Simulator
How many families? 5
Line 1, Column 1 Tab Size: 4 Python
```

Doesn't work
ie, not interactive



Running Programs

Running an existing file

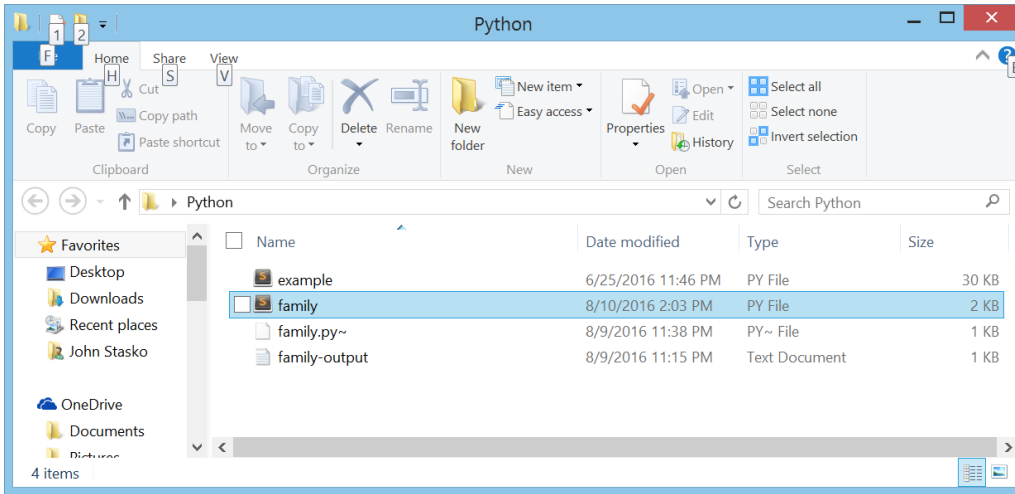
```
C:\Windows\system32\cmd.exe

C:\Users\John\Desktop\Python>python family.py
Starting Family Simulator
How many families? 5
Simulating families
1 - G G G G B
2 - B G
3 - B G
4 - B G
5 - B G
The average number of children born was 2.6 and the maximum was 5 .
A total of 8 girls were born and 5 boys.
Run again? (y/n) n

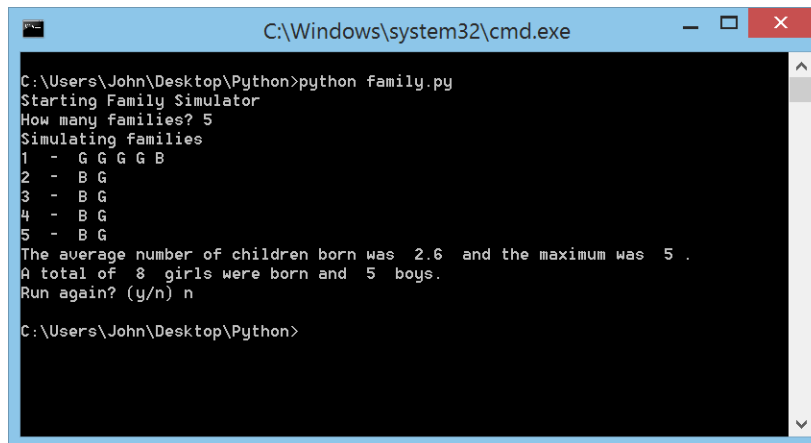
C:\Users\John\Desktop\Python>_
```



Tips & tricks (Windows)



Shift right click in background of window to get menu with “Open command window here” operation



Then run
python foobar.py

Types, Variables

- `int, float, str`
- No variable declarations
- Weakly typed language
- ```
x = 1
x = "hello"
print x
```
- Boolean values `True` and `False`

# Math Expressions

- `+`, `-`, `*`, `/`, `%`, `**`  
(`%` - remainder, `**` - exponentiation)

`5 % 3 is`

`3 / 2 is`

`3 / 2.0 is`

- Assignment statements

`x = (a + 10) * (b**3 / d * f)`

`tax = tax + .01`

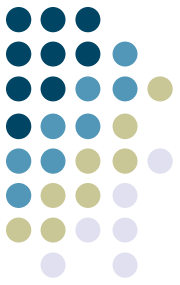
# Type Conversions

- You have two integers  $a$  and  $b$   
You want  $a/b$  (the floating point value)

`a / float(b)`

- Mixed type expression, the integer is converted to a float:

`17.5 / 6`



# Output

```
print 'Hello'
print "Hello"
print distance
print a, b
print "The value is ", val
```

Suppresses the newline

```
print a, ←
```

---

## Version 3

```
print('Hello')
print(a, b)
print(a, end=" ") ←
```

Suppresses the newline

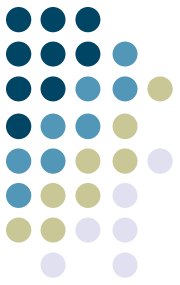
# Input (Python 2)

- Reading strings

```
firstName = raw_input('Enter your first name')
```

- Reading numbers

```
age = input('How old are you?')
```



# Input (Python 3)

- Reading strings

```
firstName = input('Enter your first name')
```

- Reading numbers

```
age = eval(input('How old are you?'))
```



# Intermission

- Admin stuff



# Control Flow

- Not all programs are “straight line”
  - Conditional statements
  - Iteration
  - Functions and procedures

# Conditional Expressions

- Conditionals: *if*

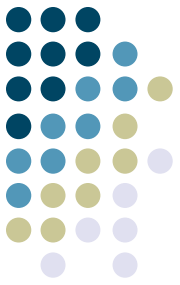
```
if x == 1:
 print "X is one"
elif x == 0:
 print "X is zero"
else:
 print "X is something else"
```

# Indentation

- Blocks are denoted by exact indentation

```
if x == 10:
 a = b * 5
 print "Hello"
 y = y + 1
print a, b
```

A source of headaches...



# What is X after the code runs?

```
x = 7
if x > 22:
 x = 5
elif x <= 7:
 x = x*3
elif x == 21:
 x = x*2

print(x)
```

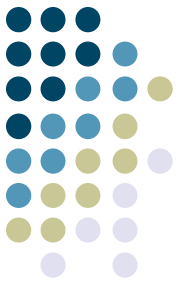
1. 5
2. 7
3. 21
4. 42
5. I have no idea

# Iteration - while

- General form: *while* *<test>*: will continue to iterate as long as *<test>* is true (not 0 or *none*)
- Special keywords to control looping
  - *break* - jumps out of the loop
  - *continue* - jumps back to the top of the loop (the *while* statement)

```
x = 15
while x > 0:
 print x
 x = x - 1
```

Watch that indentation!



# Iteration - for

- General form: the *for* statement iterates over a sequence of items, such as a list

```
for x in range(5):
 print x
```

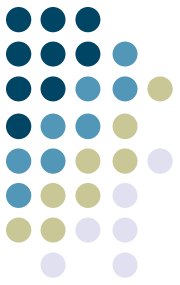
prints 0, 1, 2, 3, 4

```
for x in range(2, 5):
 print x
```

prints 2, 3, 4

```
for x in ["spam", "eggs", "ham"]:
 print x
```

prints spam, eggs, ham



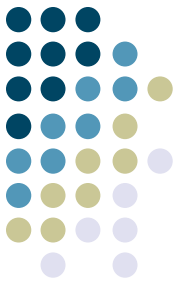
# What is z after the code runs?

```
x = 5
z = 0
for x in range(5):
 y = 6
 if x == 3:
 break
 while y > 0:
 y = y - 1
 z = z + 1

print(z)
```

1. 0
2. 18
3. 24
4. 64
5. I have no idea





# Exercise

- Write a program that:

Takes integer input from the user, until -1 is entered  
Calculates and prints the average of those numbers  
(excluding the -1)

# Functions

- It's helpful to break your code into smaller pieces that each do some specific function or operation
  - Can test each one individually
  - Makes debugging easier

- Python uses the

```
def fct_name():
 statement
 statement
 etc
 return expression
```

**syntax**

# Example

- ```
def sum(num1, num2):  
    result = num1 + num2  
    return result
```

Elsewhere in code

```
a = 3  
b = 12  
c = sum(a, b)
```

Modularity

- We typically have multiple functions in a file
 - Name one main as the “starter”
 - Call it to get things going



Example

```
def main():  
    code goes here  
  
def fct1():  
    code goes here  
  
def fct2():  
    code goes here  
  
...  
  
main()
```

Random Numbers

- Built-in functions for that
- `import random`

```
number = random.randint(1, 100)
```

Gives random integer between 1 and 100, inclusive

- `number = random.uniform(1.0, 10.0)`

Gives random float between 1.0 and 10.0

Modules

- Each separate file is called a *module*
- Usually contains a set of related functions that provide some useful operations
- You then `import` these modules
- When you downloaded Python, you got a bunch of them
 - `random, math`

Scoping

- What is scoping?
- Scoping is a fancy word that just means “the rules about what you can see from where” in a program
- The *namespace* is the collection of stuff that you can see from any given point in a program

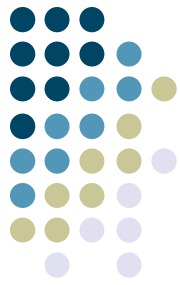


An Example Scoping Error

```
welcomeMsg = "Hello!"  
def changeWelcomeMsg():  
    welcomeMsg = "Bonjour!"  
    print "New welcome  
message is", welcomeMsg
```

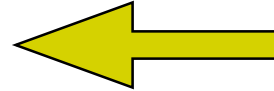
```
changeWelcomeMsg()  
>>> New welcome message is  
Bonjour!  
print welcomeMsg  
Hello!
```

Why?



An Example: Scoping Error

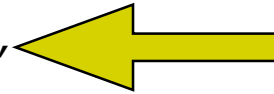
- `welcomeMsg = "Hello!"`



`welcomeMsg` is defined in the **global scope**

- `def changeWelcomeMsg():`

- `welcomeMsg = "Bonjour!"`



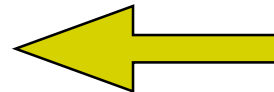
This line defines a *new* variable with the same name, in the **local scope!**

- `print "New welcome message is", welcomeMsg`

- `changeWelcomeMsg()`

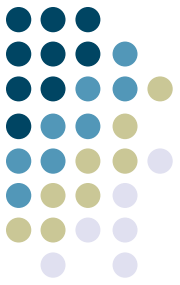
- `>>> New welcome message is Bonjour!`

- `print welcomeMsg`



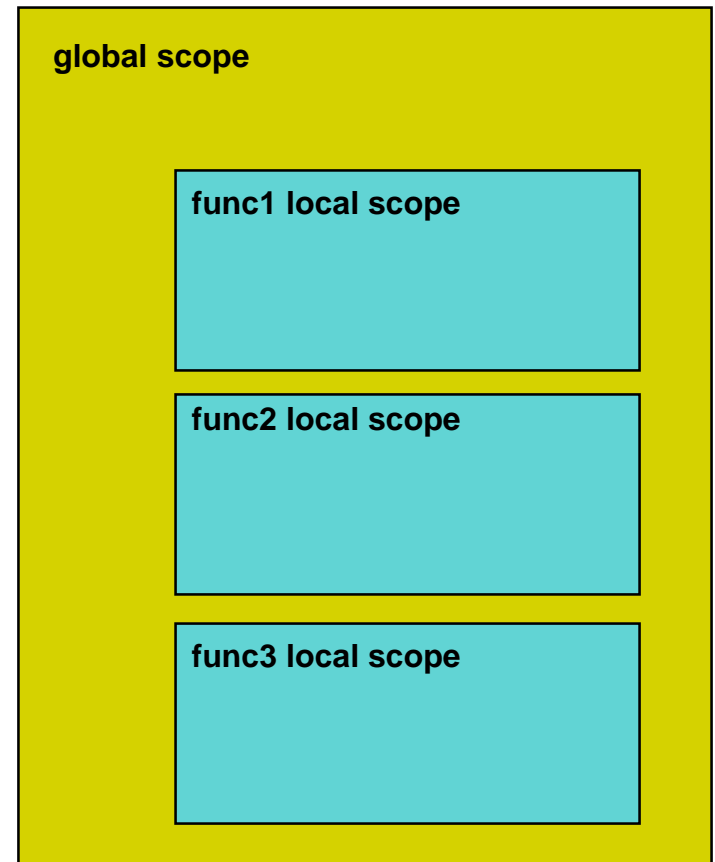
Since this call to *print* is outside the function `changeWelcomeMsg()`, it refers to the `welcomeMsg` variable in the global scope.

- `"Hello!"`



Thinking About Scopes

- Variables named in the global scope are available to statements in any scope
 - Unless they have been “hidden” by a local variable with the same name, as in the error example
- Variables named in a local scope are only available to statements in that scope
- The first assignment to a variable determines the scope it is in



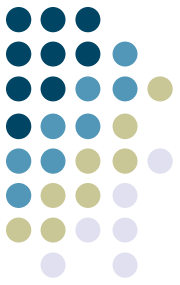


More on Scopes

- “Global” really means the file the variable is in
 - When you start developing with multiple files, each file defines its own scope that is “global” for that file
- Each call to a function creates a new local scope
 - Thus if a variable `foo` is defined in function `func()`, each call to `func()` has its own new “namespace” and its own separate `foo`
- By default, *all* assignments that you make in a function create names in the local scope
 - Advanced: you can use the *global* statement if you want to change a global variable from within a function
 - Dangerous, but useful. We’ll talk about it in a later lecture
- Names not assigned to in a function are assumed to be globals

Still More on Scopes

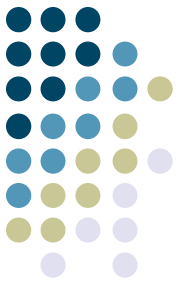
- What all this boils down to is...
 - Local variables (those first assigned to within a function) serve as temporary names you need only when a function is running
 - This helps modularity of your program ("hide" details within a function)
- But:
 - You need to be careful when using a name within a function that's defined outside
 - Subtle and hard to track bugs...
 - Just don't do it



A common bug

```
def print0to9():  
    maxmum = 10  
    for i in range(maximum):  
        print(i)
```

```
maximum = 5  
for i in range(1, maximum):  
    print(i)  
print0to9()
```



Scoping Gotchas

- Subtly different than some other languages

- 1. Local scopes don't nest

```
def outerfunc(x, y):  
    def innerfunc(z):  
        if z > 0:  
            print x, y  
    innerfunc(x)
```

- x and y aren't available inside the local scope for innerfunc
- 2. There are actually *three* scopes: global, local, and `__builtin__`
 - First, the local scope is checked
 - Then, the global scope
 - Finally, the scope defined by the module called `__builtin__`
 - `len`, `abs`, `max`, `min`, ...

HW



- Family simulation (see website)
- Due Tuesday at class time
- Turn in via t-square
- If you're having a tough time with it, seek help from Amit or John

More Administratia

- Late policy for assignments:
 - Clear with me *first* if you have a valid excuse for missing a due date
 - Examples: medical or family emergency
 - My policy is -10% per late day, maximum 3 days late