

## Evolution of Scripting Languages

- UNIX shell scripting
  - » awk, sed, ksh, csh
- Tck/Tk
- Perl
- Python
- PHP
- Ruby

Maria Hybinette, UGA

2

## CSCI: 4500/6500 Programming Languages

Python



Thanks to: Stephen Ferg, Bureau of Labor Statistics and Guido van Rossum, Python Architect

Maria Hybinette, UGA

1

## Python

## Python

Developed in 1991  
by Guido van Rossum

- Mature
- Powerful / flexible
- Easy-to-learn / use
- Easy to read (in contrast to Perl ☹)
- Open source
- Lots of documentation
- Lots of tutorials
- Lots of libraries
  - » Ruby - nice, purely object oriented, *but* harder to find libraries

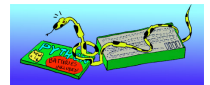


Monty Python's Flying Circus Fan



Guido Van Rossum

- Portable
  - » Mac, Windows, Unix (and installed on atlas.cs.uga.edu)
- Faster than C, C++, Java in *productivity*
  - » Compact language
  - » Batteries included (build in library)
- Python block indenting
  - » looks cleaner => easier to read
- Slower in execution
  - » but you can integrate C/C++/Java with Python



Maria Hybinette, UGA

3

Maria Hybinette, UGA

4

## Python vs Java (outdated)



Test	Java	Python	Comparison
Standard Output	138.85	30.58	Python 4.5X faster
Hashtable	17.00	8.22	Python 2X faster
I/O	56.72	47.36	Python 1.2X faster
List	5.94	14.32	Java 2.4X faster
Native Methods	2.475	7.92	Java 3.2X faster
Interpreter Initialization	0.25	0.04	Python 6.3X faster
Object Allocation	23.65	211.11	Java 8X faster
Interpreter Speed	0.43	2.29	Java 5.3X faster

<http://www.twistedmatrix.com/users/glyph/rant/python-vs-java.html> (April/2000)

Maria Hybinette, UGA

Matthew "Glyph" Lefkowitz

5

## More comparisons...

- [furryland.org/~mikec/bench/](http://furryland.org/~mikec/bench/)
- Doug Bagley's Great Computer Language Shootout
  - (<http://web.archive.org/web/20040611035744/http://shootout.allioth.debian.org/>)

Maria Hybinette, UGA

6

## Python vs Java

- Python programs run slower than Java
- Python programs take less time to develop
  - » Typically a 5-10 times difference (origin, Ousterhout)
- Python is **dynamically** (but **strongly**) typed
  - » Programmer don't have to deal with static typing
    - variable bound to type at **compile time** & optionally to an object (value of same type)
  - » Trend is now toward stronger static type checking, not less
    - However, this is a productivity win at the cost of some risk
- Python is **compact**
- Python is **concise** (not verbose, not superfluous)
- Closures (lambda)



Maria Hybinette, UGA [http://www.ferg.org/projects/python\\_java\\_side-by-side.html](http://www.ferg.org/projects/python_java_side-by-side.html) (February/2004)

7

## Weak vs Strong Typing

- Variable can be of non-specific data type.
- Strongly typed languages puts (many) restrictions on of different types interact with each other
  - » 3+ 3.5 may not be allowed (only integers adds)
- Weak typing means that a language will implicitly convert (or cast) types when used.
  - » C is weakly typed, you can easily override the type system using casts, PHP
  - » Python, Java strongly typed

Maria Hybinette, UGA

8

## Dynamically vs Static Typing

- Other people like static typing... trade-off performance



I think there is a trend ...

Dynamically typed languages such as Python, Ruby, and even Smalltalk will be mainstream industrial languages in the coming years.

Robert C. Martin, author of *Agile Software Development*

Maria Hybinette, UGA

9

## Quotes

- "When Java came out, I was excited -- I could write code twice as fast in Java as I could in C/C++. And with Python I can write code twice as fast as I can in Java."
  - Andy Hertzfeld (original apple computer software engineer now at google)
- *When a 20,000 line project went to approximately 3,000 lines overnight, and came out being more flexible and robust ... I realized I was on to something really good.*
  - Matthew "Glyph" Lefkowitz
- So the real punchline of the story is this: weeks and months after writing [python program], I could still read the code and grok what it was doing without serious mental effort.
  - » And the true reason I no longer write Perl for anything but tiny projects is that was never true when I was writing large masses of Perl code. I fear the prospect of ever having to modify keeper or anthologize again -- but [the above python] gives me no qualms at all.

Maria Hybinette, UGA

Eric S. Raymond, author of *The Cathedral and the Bazaar*

10

## Is it Safe?



Python is an "open-source" language.

It has no vendor.

Does that mean we'll have support problems?

What about...

- Vendor longevity?
- Consulting & training support?
- Books and reference materials?
- Tools? IDEs, debuggers, screen-painters?

Maria Hybinette, UGA

11



## Who is using open-source software?

"LAMP"

- Linux
- Apache
- MySQL
- PHP | Perl | Python

- Apache has overwhelmingly dominated the Web server market since 1996.
- PHP is the most popular Apache module, running on almost 10 million domains (over a million IP addresses).

... and then there's ...

Maria Hybinette, UGA

12



## Installing Python

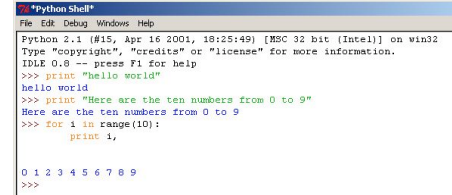
- Already exists of atlas.cs.uga.edu
- Easy to get and install for Win/Mac from <http://www.python.org>

Maria Hybinette, UGA

19

## IDLE Development Environment

- Shell for interactive evaluation
- Text editor with color-coding and smart indenting for creating python files.
- Menu commands for changing system settings and running files.



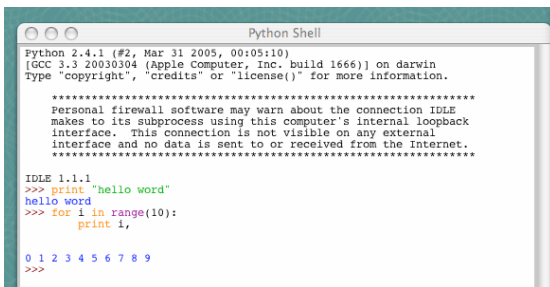
```
Python Shell
Python 2.1 (#15, Apr 16 2001, 18:25:49) [MSC 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
IDLE 0.8 -- press F1 for help
>>> print "hello world"
hello world
>>> print "Here are the ten numbers from 0 to 9"
Here are the ten numbers from 0 to 9
>>> for i in range(10):
        print i,
0 1 2 3 4 5 6 7 8 9
>>>
```

[http://hkn.eecs.berkeley.edu/~dyoo/python/idle\\_intro/index.html](http://hkn.eecs.berkeley.edu/~dyoo/python/idle_intro/index.html)

Maria Hybinette, UGA

20

## Interpreter: On my Mac



```
Python 2.4.1 (#2, Mar 31 2005, 00:05:10)
[GCC 3.3 20030304 (Apple Computer, Inc. build 1666)] on darwin
Type "copyright", "credits" or "license()" for more information.
*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****
IDLE 1.1.1
>>> print "hello world"
hello world
>>> for i in range(10):
        print i,
0 1 2 3 4 5 6 7 8 9
>>>
```

Maria Hybinette, UGA

21

## Working with a file.py

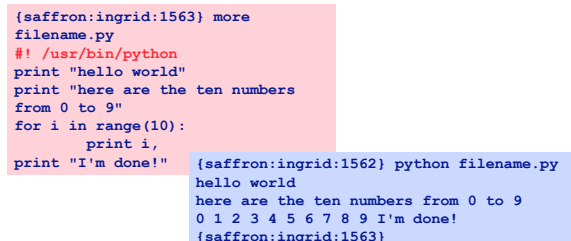
- IDLE -
  1. File -> new window
  2. type commands in new window area
  3. save as "file name".py (typical extension)
  4. Run module

Maria Hybinette, UGA

22

## Running Programs on UNIX

- `#!/opt/sfw/bin/python` (makes it runnable as an executable)



```
{saffron:ingrid:1563} more
filename.py
#!/usr/bin/python
print "hello world"
print "here are the ten numbers
from 0 to 9"
for i in range(10):
    print i,
print "I'm done!"
{saffron:ingrid:1562} python filename.py
hello world
here are the ten numbers from 0 to 9
0 1 2 3 4 5 6 7 8 9 I'm done!
{saffron:ingrid:1563}
```

Maria Hybinette, UGA

23

Maria Hybinette, UGA

24

## Look at a sample of code...

```
x = 34 - 23           # A comment.
y = "Hello"          # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"   # String concat.
print x
print y
```

Maria Hybinette, UGA

25

## Look at a sample of code...

```
x = 34 - 23           # A comment.
y = "Hello"          # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"   # String concat.
print x
print y
```

```
>>>
12
HelloWorld
```

Maria Hybinette, UGA

26

## Enough to Understand the Code

- Assignment uses = and comparison uses ==.
- For numbers +-\*/% are as expected.
  - » Special use of + for string concatenation.
  - » Special use of % for string formatting.
- Logical operators are words (and, or, not) not symbols (&&, ||, !).
- The basic printing command is "print."
- First assignment to a variable will create it.
  - » Variable types don't need to be declared.
  - » Python figures out the variable types on its own.

Maria Hybinette, UGA

27

## Basic Datatypes

- Integers (default for numbers)
  - `z = 5 / 2` # Answer is 2, integer division.
- Floats
  - `x = 3.456`
- Strings
  - Can use "" or '' to specify. "abc" 'abc' (Same thing.)
  - Unmatched ones can occur within the string. "maria's"
  - Use triple double-quotes for multi-line strings or strings that contain both ' and " inside of them: ""a'b'c""

Maria Hybinette, UGA

28

## Whitespace

- Whitespace is meaningful in Python: especially indentation and placement of newlines.
  - » Use a newline to end a line of code. (Not a semicolon like in C++ or Java.) (Use \ when must go to next line prematurely.)
  - » No braces {} to mark blocks of code in Python... Use consistent indentation instead. The first line with a new indentation is considered outside of the block.
  - » Often a colon appears at the start of a new block. (We'll see this later for function and class definitions.)

Maria Hybinette, UGA

29

## Comments

- Start comments with # – the rest of line is ignored.
- Can include a "documentation string" as the first line of any new function or class that you define.
- The development environment, debugger, and other tools use it: it's good style to include one.

```
def my_function(x, y):
    """This is the docstring. This
    function does blah blah blah."""
    # The code would go here...
```

Maria Hybinette, UGA

30

## Look at a sample of code...

```
x = 34 - 23          # A comment.
y = "Hello"         # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World" # String concat.
print x
print y
```

Maria Hybinette, UGA

31

## Python and Types

Python determines the data types  
in a program automatically.

"Dynamic Typing"

But Python's not casual about types, it  
enforces them after it figures them out.

"Strong Typing"

So, for example, you can't just append an integer to a string. You  
must first convert the integer to a string itself.

```
x = "the answer is " # Decides x is string.
y = 23               # Decides y is integer.
print x + y         # Python will complain about this.
```

```
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <toplevel>
    print x + y
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

Maria Hybinette, UGA

32

## Naming Rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.

```
bob Bob _bob _2_bob_ bob_2 BoB
```

- There are some reserved words:

```
and, assert, break, class, continue, def,
del, elif, else, except, exec, finally, for,
from, global, if, import, in, is, lambda,
not, or, pass, print, raise, return, try,
while
```

Maria Hybinette, UGA

33

## Accessing Non-existent Name

- If you try to access a name before it's been properly created (by placing it on the left side of an assignment), you'll get an error.

```
>>> y
```

```
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <toplevel>
    y
NameError: name 'y' is not defined
>>> y = 3
>>> y
3
```

Maria Hybinette, UGA

34

## Multiple Assignment

- You can also assign to multiple names at the same time.

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

Maria Hybinette, UGA

35

## String Operations

- We can use some methods built-in to the string data type to perform some formatting operations on strings:

```
>>> "hello".upper()
'HELLO'
```

- There are many other handy string operations available. Check the Python documentation for more.

Maria Hybinette, UGA

36

## Printing with Python

- You can print a string to the screen using “print.”
- Using the % string operator in combination with the print command, we can format our output text.

```
>>> print "%s xyz %d" % ("abc", 34)
abc xyz 34
```

“Print” automatically adds a newline to the end of the string. If you include a list of strings, it will concatenate them with a space between them.

```
>>> print "abc"      >>> print "abc", "def"
abc                  abc def
```

Maria Hybinette, UGA

37

## Strings

- » Concatenation
  - “Hello” + “World” -> ‘HelloWorld’
- » Repetition
  - “UGA” \* 3 -> ‘UGAUGAUGA’
- » Indexing
  - “UGA”[0] -> ‘U’
- » Slicing
  - “UGA”[1:3] -> ‘GA’
- » Size
  - len(“UGA”) -> 3
- » Comparison
  - “Maria” < “maria” -> True
- » Search
  - “i” in “maria” -> True

Maria Hybinette, UGA

38

## Container Types

- ( 100, 200, 300 ) # Tuple
- [ 42, 3.14, “hello” ] # List
- { ‘x’:42, ‘y’:3.14 } # Dictionary

### Tuple

- » a simple *immutable* ordered sequence of items

### List

- » a mutable ordered sequence with more powerful manipulations

### Dictionary -

- » a lookup table of key-value pairs

Maria Hybinette, UGA

39

## Lists

```
>>> alist = [631, “maria” , [ 331, “maria” ]]
>>> print alist
[123, ‘maria’, [331, ‘maria’]]
```

- List items need not have the same type
- Flexible arrays not Lisp-like linked list
- Same operators as for strings
- operations append(), insert(), pop(), reverse() and sort()

Maria Hybinette, UGA

40

## More List Operations

```
>>> a = range(5)          # [0,1,2,3,4]
>>> a.append(5)          # [0,1,2,3,4,5]
>>> a.pop()              # [0,1,2,3,4]
5
>>> a.insert(0, 42)     # [42,0,1,2,3,4]
>>> a.pop(0)            # [0,1,2,3,4]
42
>>> a.reverse()         # [4,3,2,1,0]
>>> a.sort()            # [0,1,2,3,4]

>>> a.append([22,33])   # [0,1,2,3,4,[22,33]]
>>> a.extend([10,20])  # [0,1,2,3,4,[22,33],10,20]
```

Maria Hybinette, UGA

41

## More Lists

- List multiplication
  - » list = [“aa”, “bb”] \* 3
- Printing out lists
  - » print “\n”.join(list) # better formatting
- More operations
  - » list.count(“aa”) # how many times
  - » list.index(“bb”) # returns the location
- More on slices
  - » list[-1] # last element
  - » list[0:3] # starting ele 0 and up to 2
  - » list[3:] # starting ele 3 to end of list
  - » list[:] # a complete copy of the list

Maria Hybinette, UGA

42

## Dictionaries

- Hash tables, "associative arrays" with key/value pairs

```
- d = {"duck": "bird", "bee": "insect"}
```

- Lookup:

```
- d["duck"]          # "bird"
- d["lion"]         # raises KeyError exception
```

- Delete, insert, overwrite:

```
- del d["bee"]       # delete
- d["lion"] = "cat" # insert
- d["duck"] = "unknown" # overwrites
```

Maria Hybinette, UGA

43

## More Dictionary Ops

- Keys, values, items:

```
- d.keys()          # returns dictionary keys
- d.values()        # returns all values
- d.items()         # returns a list of
                    # key/value pairs
```

- Presence check:

```
- d.has_key("duck") # True
- d.has_key("spam") # False
```

- Values of any type

- Keys almost any type

```
{
  "name": "Maria",
  "age": 25,
  ("hello", "world"): 1,
  42: "yes",
  "flag": ["red", "white", "blue"]
}
```

Maria Hybinette, UGA

44

## Dictionary Details

- Keys must be immutable:

- » numbers, strings, tuples of immutables
  - these cannot be changed after creation
- » reason is *hashing* (fast lookup technique)
- » not lists or other dictionaries
  - these types of objects can be changed "in place"
- » no restrictions on values

- Keys will be listed in arbitrary order

- » again, because of hashing

Maria Hybinette, UGA

45

## Tuples

- Immutable lists

- Faster than lists

```
● key = (lastname, firstname)
● lastname = key[0]
● point = x, y, z          # parentheses optional
● x, y, z = point         # unpack
● singleton = (1,)       # trailing comma!!!
● empty = ()             # parentheses!
```

Maria Hybinette, UGA

46

## Variables

- Need to assign (initialize)

- use of uninitialized variable raises exception

- No need to declare type (dynamically typed)

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```

- » However once set the type matters

- Can't treat integer as a string

Maria Hybinette, UGA

47

## Reference Semantics

- Assignment manipulates references

- x = y
  - does not make a copy of y
  - makes x *reference* the object y references

- Very useful; but beware!

- Example:

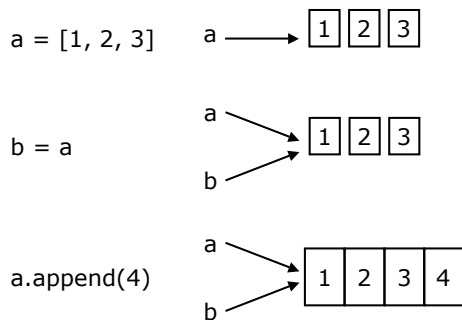
```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

Maria Hybinette, UGA

48



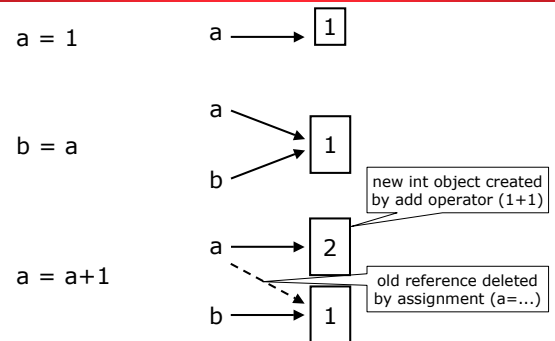
## Changing a Shared List



Maria Hybinette, UGA

49

## Changing an Integer



Maria Hybinette, UGA

50

## Control Structures

```
if condition:
    statements
[elif condition:
    statements] ...
else:
    statements

while condition:
    statements

for var in sequence:
    statements

break
continue
```

Maria Hybinette, UGA

51

## More For Loops

- looping through list
  - » `for item in list:`
  - » `print item`
- looping through counter
  - » `for i in range(5):`
  - » `print i`
- Iterating through a dictionary
  - » `import os`
  - » `for k,v in os.environ.items():`
  - » `print "%s=%s" % (k,v)`
- `os.environ` is a dictionary of environment variables`
- `dict.items()` returns key value pair tuples

Maria Hybinette, UGA

52

## Exercise I

Print:

`1x1=1 1x2=2 1x3=3 .... 8x9=72 9x9=81`

but don't repeat. For example

print only `3x5=15`

but don't print `5x3=15`

so print only if `first_number <= second_num`

Hint: use range

```
for num in range(1,10):
```

...

Maria Hybinette, UGA

53

## Exercise Answer

```
a = range(1,10)
```

```
b = range(1,10)
```

```
for anum in a:
```

```
    for bnum in b:
```

```
        if ( anum <= bnum ):
```

```
            print str(anum), "x", str(bnum), "=", str(anum*bnum)
```

Maria Hybinette, UGA

54

## Grouping Indentation

In Python:

```
for i in range(20):
    if i%3 == 0:
        print i
    if i%5 == 0:
        print "Bingo!"
print "---"
```

In C:

```
for ( i = 0; i < 20; i++ )
{
    if (i%3 == 0) {
        printf("%d\n", i);
    }
    if (i%5 == 0) {
        printf("Bingo!\n");
    }
}
printf("---\n");
```

```
0
Bingo!
...
3
...
6
...
9
...
12
...
15
Bingo!
...
18
...
```

## Functions, Procedures

```
def name(arg1, arg2, ...):
    """documentation""" # optional doc string
    statements

return # from procedure
return expression # from function
```

## Example Function

```
def gcd(a, b):
    "greatest common divisor"
    while a != 0:
        a, b = b%a, a # parallel assignment
    return b

>>> gcd.__doc__ # 2 __ of these
'greatest common divisor'
>>> gcd(12, 20)
4
```

## Exercise II

- Phone book application
  - » 1) add
    - Ask for name and phone number
  - » 2) print phone book
- To get input:
  - » answer = raw\_input("Enter your selection: ")

```
intro = """
Welcome to the phone book application
choices:
1) add new entry
2) print phone book
3) exit
"""

print intro

ph_d = {} # phone book dictionary

def add_entry():
    """ add new entry into phone book"""
    name = raw_input("give me a name:")
    number = raw_input("give me a number:")
    ph_d[name] = number

def print_pb():
    print "name".rjust(30)+"number".rjust(30)
    for name,num in ph_d.items():
        print name.rjust(30),num.rjust(30)

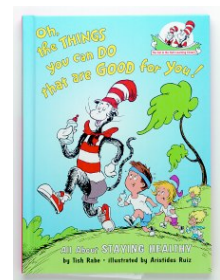
while (True):
    response = raw_input("Enter your command: ")
    if (response == '1'):
        add_entry()
    elif (response == '2'):
        print_pb()
    elif (response == '3'):
        break
    else:
        print "invalid command"
```

```
>>>
Welcome to the phone book application
choices:
1) add new entry
2) print phone book
3) exit

Enter your command: 1
give me a name:maria
give me a number:555-1212
Enter your command: 2
name number
maria 555-1212
Enter your command:
```

## On your own...

- modules & packages
- exceptions
- files & standard library
- classes & instances



## Hands On

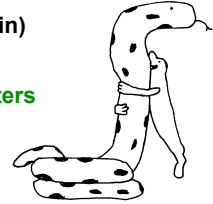
- [www.python.org/doc/current/tut/tut.html](http://www.python.org/doc/current/tut/tut.html)

Maria Hybinette, UGA

61

## Python Slogans

- Python Fits Your Brain, **Bruce Eckel**
- Life is Better Without Braces, **Bruce Eckel**
- Import This
- Batteries included (Tcl origin)
- Powered by Python
- Readability counts, **Tim Peters**



<http://mindview.net/>

62

## Bruce Eckel's Top 10

### 10. Reduced clutter.

Programs are read more than they are written  
Consistent formatting is important  
readability & compactness  
conversation of compactness  
Consistent use of programming idioms



### 09. It's not backward-compatible with other languages. (This came with some hilarious one-liners:

"C++'s backward compatibility with C is both its strength and its bane";  
"Java causes repetitive-strain syndrome";  
"Perl is compatible with every hacky syntax of every UNIX tool ever invented";  
"C# and Microsoft .NET are backward-compatible with Microsoft's previous marketing campaigns"; and  
"Javascript is not even compatible with itself".)



### 08. It doesn't value performance over my productivity. C++ memory leaks primitive types require awkward coding

Maria Hybinette, UGA

63

## Bruce Eckel's Top 10



### 07. It doesn't treat me like I'm stupid.

Java insists operator overloading is bad because you can make ugly code with it.

Bruce observes, "And we all know there's no ugly Java code out there."

### 06. I don't have to wait forever for a full implementation of the language. features invented in C+ takes a long time to appear in languages Unused features don't get tested

### 05. It doesn't make assumptions about how we discover errors. Is strong static type checking really the only way to be sure? Lack of good static typing in pre-ANSI C was troublesome Doesn't mean it's the best solution



Maria Hybinette, UGA

64

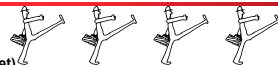
## Bruce Eckel's Top 10

### 04. Marketing people are not involved in it (yet).

Java is flawless  
Microsoft happens "Visual" C++  
Of-course Python isn't immune

### 03. I don't have to type so much. But what I do type is the *right* typing.

### 02. My guesses are usually right. I still have to look up how to open a file every time I do it in Java Most things I do in Java, I have to look up. Remember Python Idioms easier because they are simpler



Maria Hybinette, UGA

65

## Bruce Eckel's Top 10

### 01. Python helps me focus on my concepts rather than on fighting with the language.



Maria Hybinette, UGA

66