

---

---

## Python



Some material from: Stephen Ferg Bureau of Labor Statistics  
and Guido van Rossum Python Architect

Maria Hybinette, UGA

1

---

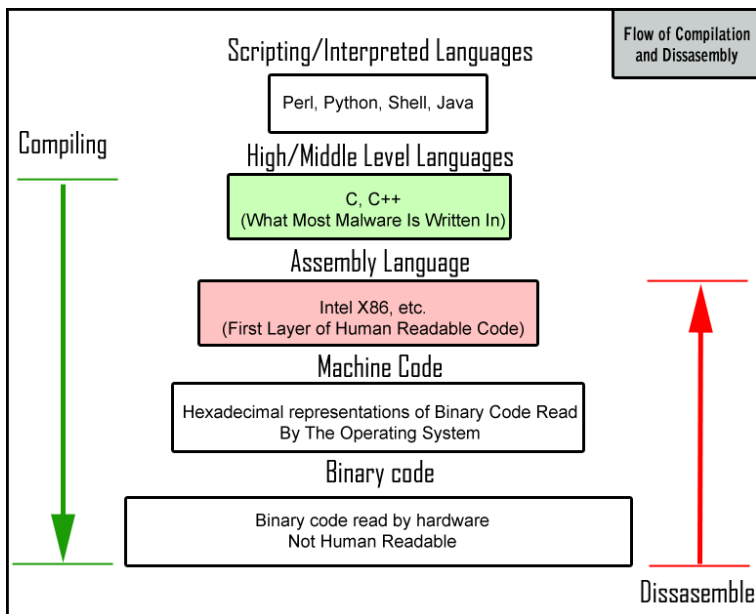
---

## Evolution of *Scripting* Languages

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

*Scripting Language (an interpretive glue) vs. Programming Language (compiled, glue). Being classified as a scripting language today is less relevant.*

2



Maria Hybinette, UGA

Image Credit: <https://blog.malwarebytes.com/security-world/2012/09/so-you-want-to-be-a-malware-analyst/>

3



Developed in 1991 by Guido van Rossum

- [PEP 3000](#) (December 2008)

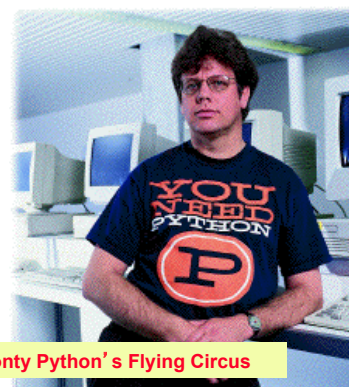
*"There should be one— and preferably only one —obvious way to do it." (remove old ways of doing stuff)*

- Mature
- Powerful / flexible
- Easy-to-learn / use
- Easy to **read** and intuitive (Perl)

- Open source, and free
- Lots of documentation
- Lots of tutorials
- Lots of libraries

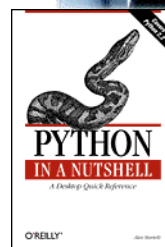
» Ruby - nice, purely object oriented, **but** harder to find libraries

Maria Hybinette



Monty Python's Flying Circus

Guido Van Rossum



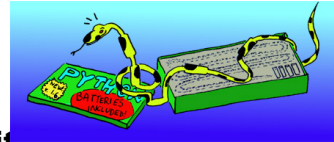
<https://www.youtube.com/watch?v=iV2ViNJEZC8>

4

# Python

---

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



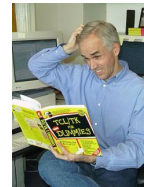
Maria Hybinette, UGA

5

## 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** 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)



[http://www.ferq.org/projects/python\\_java\\_side-by-side.html](http://www.ferq.org/projects/python_java_side-by-side.html) (February/2004)

Maria Hybinette, UGA

6

# Simplifies Explicit Data Typing

---

- Variable can be of *non-specific* data type.
- Variables are typed (by inferences) when used
- But be careful: Ruby, Python are **really strongly typed** (once a type is **inferred** you can't intermix types)

## Who is using Python?

---



- Industrial Light & Magic, maker of the Star Wars films, uses Python extensively in the computer graphics production process.
- Disney Feature Length Animation uses Python for its animation production applications.
- **Google**: youtube, Maps, Gmail
- **Yahoo** uses Python for its groups site, and in its **Inktomi** search engine.
- Reddit, BitTorrent
- New York Stock Exchange (NYSE) - uses it for developing on-line systems for the floor of the exchange and for the member firm's back offices
- The **National Weather Service** uses Python to prepare weather forecasts.
- Financial analysis

# Learning Python

- We will cover the *highlights* of python.
  - » You will have to learn more on your own.
  - » “Dive into Python”, Mark Pilgrim
    - download a local copy pdf and on-line read available
    - <http://diveintopython.net>
- The Official Python Tutorial  
<https://docs.python.org/3/tutorial/>
- The Python Quick Reference  
<http://rgruet.free.fr/#QuickRef>



# Python: Batteries included

---

---

- Large Collection of proven modules included in standard distribution

## numPy

---

---

- Offers Matlab-ish capabilities within Python
- **Fast** array operations
- 2D Arrays, multi-D arrays, linear algebra and more.
- Tutorial:
  - » [http://www.scipy.org/Tentative\\_NumPy\\_Tutorial](http://www.scipy.org/Tentative_NumPy_Tutorial)

# pandas

---

- **Pandas uses a**
  - » **DataFrame** object which can be thought of as a table of data
  - » **Handles Time Series**
- **It was built by the finance sector to aid with data manipulation and data analysis**
- **It has loads of brilliant functions to dig into your data**
- **It has useful functions for reading and writing to file types such as csv and Excel**

# sciPY: Scientific Python

---

- **Gathers a variety of high level science and engineering modules together:**
- **stats:** statistical functions
- **spatial:** KD-trees, nearest neighbors, distance functions
- **interpolate:** interpolation tools e.g. IDW, RBF
- **optimize:** optimization algorithms including linear programming

# matplotlib

- plotting library to make graphs.
- easily customized and produce publication quality plots
- Using the Matplotlib, NumPy and Pandas libraries together make data analysis much easier and reproducible than in Excel

```
#!/usr/bin/env python
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

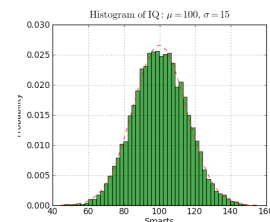
mu, sigma = 100, 15
x = mu + sigma*np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='green',
alpha=0.75)

# add a 'best fit' line
y = mlab.normpdf( bins, mu, sigma)
l = plt.plot(bins, y, 'r--', linewidth=1)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'\mathrm{Histogram of IQ:} \ \mu=100, \ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)

plt.show()
```



Maria Hybinette, UGA

## python™ Versions

- **Version History**
  - » Python 0.9.0 (1991 first published version of code)
  - » Python 1.x (1994 legacy)
  - » Python 2.7.x (2000, list comprehensions, Haskell)
  - » Python 3.2.x (3 branch started in 2008, remove redundancies in code, only one “obvious” way to do it)
- **Developing environments:**
  - » **IDLE (basic)**
    - coded in 100% pure Python, using the tkinter GUI toolkit
    - cross-platform: works on Windows and Unix
    - Python shell window (a.k.a. interactive interpreter)
    - debugger (not complete, but you can do the basics, set breakpoints, view and step)
  - » **ipython, Spyder (Anaconda)**
  - » **Eclipse module**

Maria Hybinette, UGA



# Installing Python

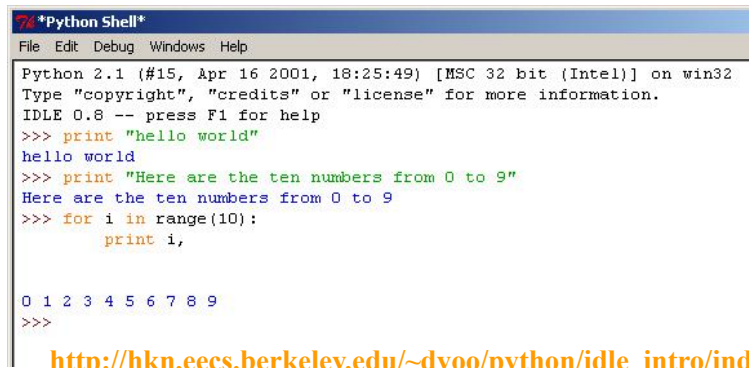
---

- Already exists of nike.cs.uga.edu (version 2.6)
- Easy to get and install for Win/Mac from (2.6)  
<http://www.python.org>
- Intro: [Wikipedia's Python](#)
- We recommend Anaconda installation. See Class schedule page.
  - » Demonstrate ...

# IDLE Development Environment

---

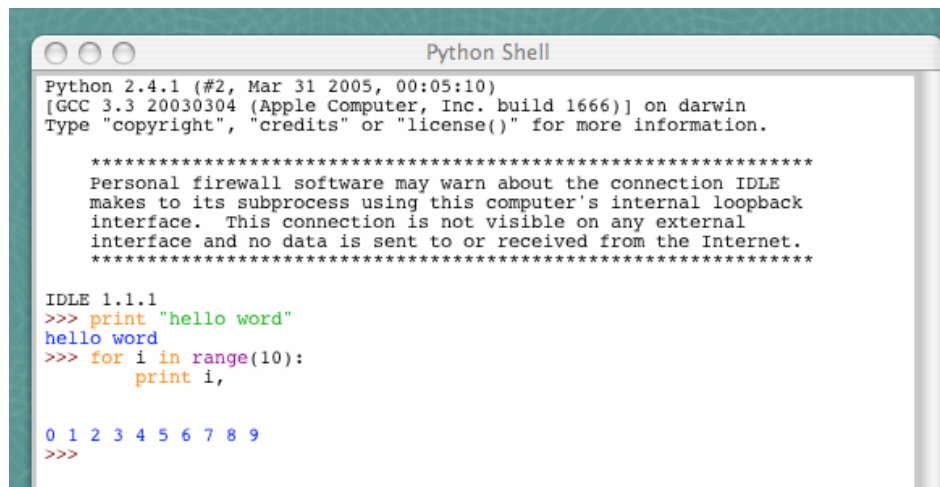
- Shell for interactive evaluation
- Text editor with **color-coding** and **smart indenting** for creating python files.
- Menu commands for changing system settings and running files.



```
*Python Shell*
File Edit Debug Windows Help
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)

# 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 word"
hello word
>>> for i in range(10):
    print i,

0 1 2 3 4 5 6 7 8 9
>>>
```

- Type “python” to start interpreter
- Type CTRL-D to exit the interpreter
- Python evaluates all inputs dynamically

Maria Hybinette, UGA

19

# IDLE: 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) – if you don’t you don’t see ‘colors’ in IDLE – but programs still run.
  4. Run module

Maria Hybinette, UGA

20

# Anaconda's Spyder Editor

---

- Debugger
- Help/Documentation easily accessible

# Running Programs on UNIX

---

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

```
{saffron:ingrid:1563} more
filename.py
#! /usr/local/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} filename.py
// what will happen?
```

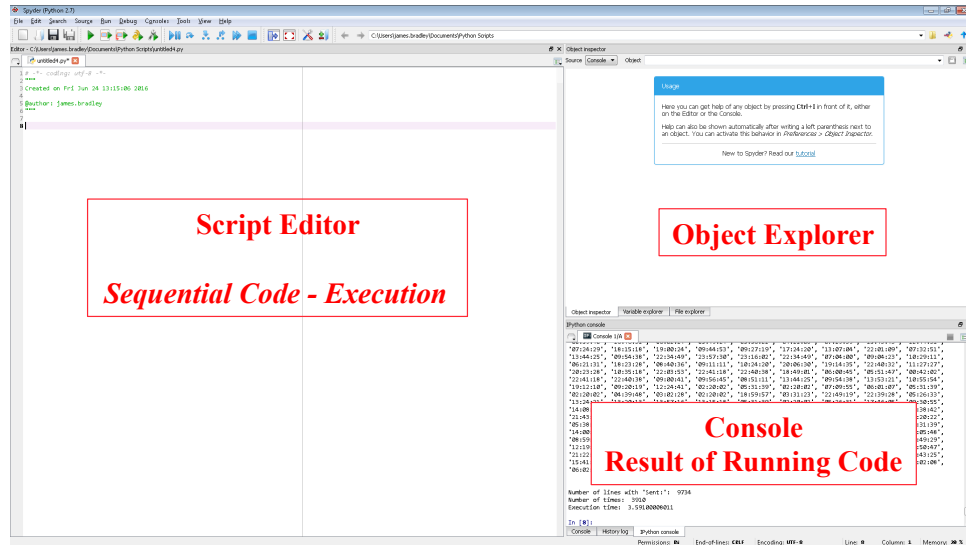
```
0 1 2 3 4 5 6 7 8 9
>>> ===== RESTART =====
=
>>>
hello word
here are the ten numbers from 0 to 9
0 1 2 3 4 5 6 7 8 9 I'm done!
>>> ===== RESTART =====
=
>>>
hello word
here are the ten numbers from 0 to 9
0 1 2 3 4 5 6 7 8 9 I'm done!
>>> ID

firstprogram.py - /Volumes/User/Users/in
print "hello word"
print "here are the ten numbers from 0 to 9"
for i in range(10):
    print i,
print "I'm done!"
```

## Other IDE(s): Anaconda's Spyder

- **Why we recommend Anaconda Python?**
  - » We could use Python IDLE ... some code ahead is depicted using this interface ... BUT! Then
- **Anaconda... arrived ....**
  - » Already has many packages installed
  - » Has a Script Editor and Console Window
  - » Allows for efficient debugging
  - » Breakpoints, using Console
- **Also has a Notebook feature**
- **Other IDEs – PyCharm, iPython Notebook.**

# Anaconda Spyder



Maria Hybinette, UGA

25

Look at a sample of code...  
(use your favorite development environment)

```
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
```

script.txt

Colors Please

Maria Hybinette, UGA

26

## 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
```

script.py

```
>>>
12
HelloWorld
```

## 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 (inference).**

# 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"""`

# 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.)

# 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...  
  
    x = y + 1  
    return x
```

## Look at more 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
```



# Python and Types

Python determines the data types  
in a program **automatically at runtime**.

“Dynamic Typing”

But Python is 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

33



# 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

34

# 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
```

# Multiple Assignment

---

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

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

```
>>> x
```

```
2
```

```
>>> y
```

```
3
```

# 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.**

# 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 separated by a comma (,) , it will concatenate them with a space **between** them.

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

# Strings

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

Maria Hybinette, UGA

39

# 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

40

# Lists

---

```
>>> alist = [631, "maria" , [ 331, "maria" ]]  
>>> print alist  
[123, 'maria', [331, 'maria']]
```

- List items need not have the same type
- Same operators as for strings
- operations `append()`, `insert()`, `pop()`, `reverse()` and `sort()`

## 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]
```

# 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 first match 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
```

# Dictionaries

---

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

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

- **Lookup:**

```
- d["duck"]                # "bird"
```

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

```
- d["bird"]                # ?
```

- **Delete, insert, overwrite:**

```
- del d["bee"]             # delete
```

```
- d["lion"] = "cat"        # insert
```

```
- d["duck"] = "unknown"    # overwrites
```

# 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 (needs to be immutable – tuples OK, but not lists).**

```
{
  "name": "Maria",
  "age": 25,
  42: "yes",
  "flag": ["red", "white", "blue"]
}
```

# Dictionary Details

---

- **Keys must be immutable:**

- » numbers, strings, tuples of immutables

- these **cannot be changed** after creation

- Keys are *hashed* (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

# Tuples

---

- **Immutable lists**
- **Faster than lists**

- `key = ("lastname", "firstname" )`
- `lastname = key[0]`
- `point = x, y, z`                   # parentheses optional
- `singleton = (1,)`               # trailing comma!!!  
  # (required otherwise  
a value)
- `empty = ()`                       # parentheses!

# 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



# Reference Semantics

---

- **Assignment *manipulates references***

- $x = y$

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

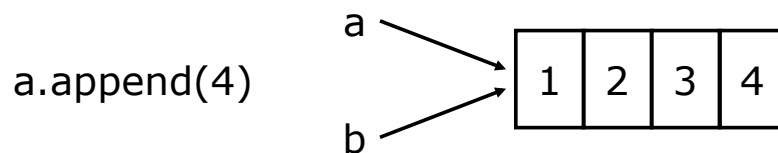
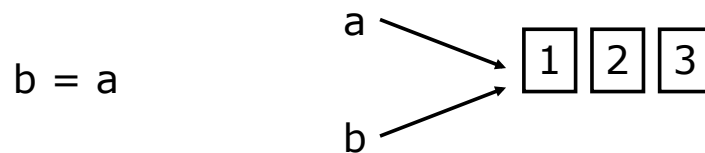
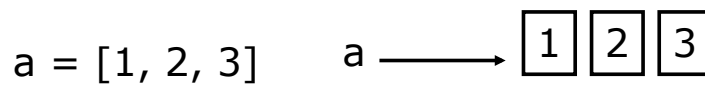
- **Very useful; but beware!**

- **Example:**

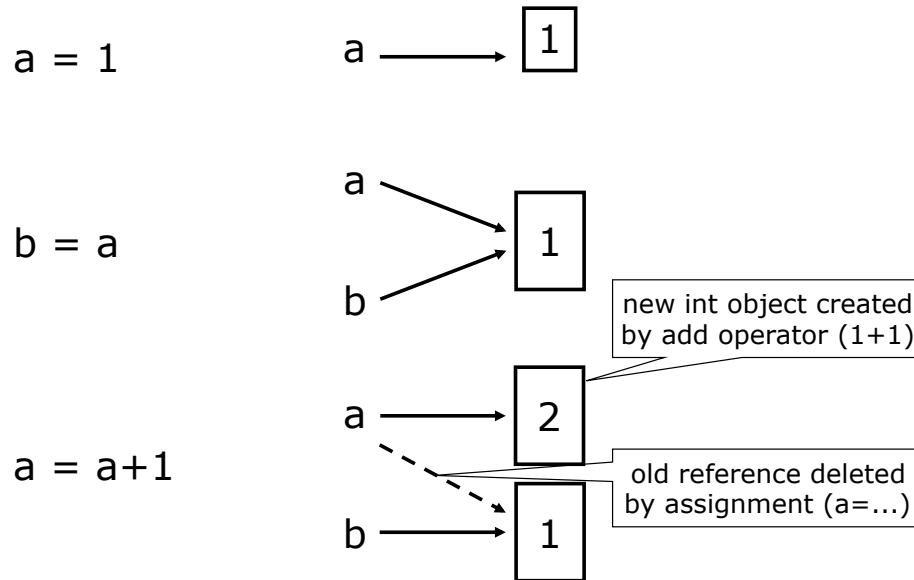
```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print b
[1, 2, 3, 4]
```

# Changing a Shared List

---



# Changing an Integer



# Control Structures

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

```
while condition:  
    statements
```

```
for var in sequence:  
    statements
```

**break**  
**continue**

# 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 'built in' dictionary
  - » `import os`
  - » `for k,v in os.environ.items():`
  - » `print "%s=%s" % (k,v)`
- `os.environ` is a dictionary of environment variables`

## Exercise I

---

Print (on separate lines)

`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):`

...

# Output

---

- $1 \times 1 = 1$
- $1 \times 2 = 2$
- $1 \times 3 = 3$
- $1 \times 4 = 4$
- $1 \times 5 = 5$
- $1 \times 6 = 6$
- ....

# 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)
```

**Don't really need  
str here**

# 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
---
---
```

Maria Hybinette, UGA

57

# Functions, Procedures

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

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

Maria Hybinette, UGA

58

## 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

---

- Write script in the Editor window to convert a Fahrenheit temperature to a Celsius temperature and print out the result in the Console window

$$^{\circ}\text{C} = (^{\circ}\text{F} - 32) \times \frac{5}{9}$$

# Exercise III

- 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:
```

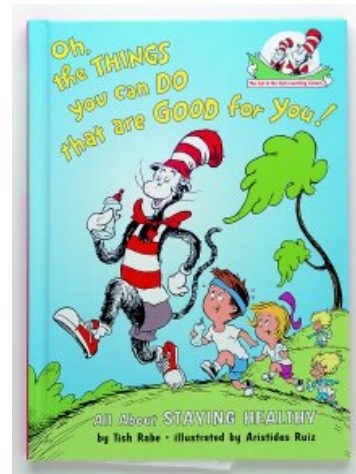
## On your own...

---

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



Maria Hybinette, UGA



63

## Hands On

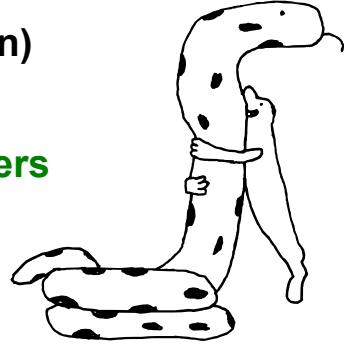
---

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



# 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/>

Maria Hybinette, UGA

65

## 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

66



## 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



## 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

# Bruce Eckel's Top 10

---



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

