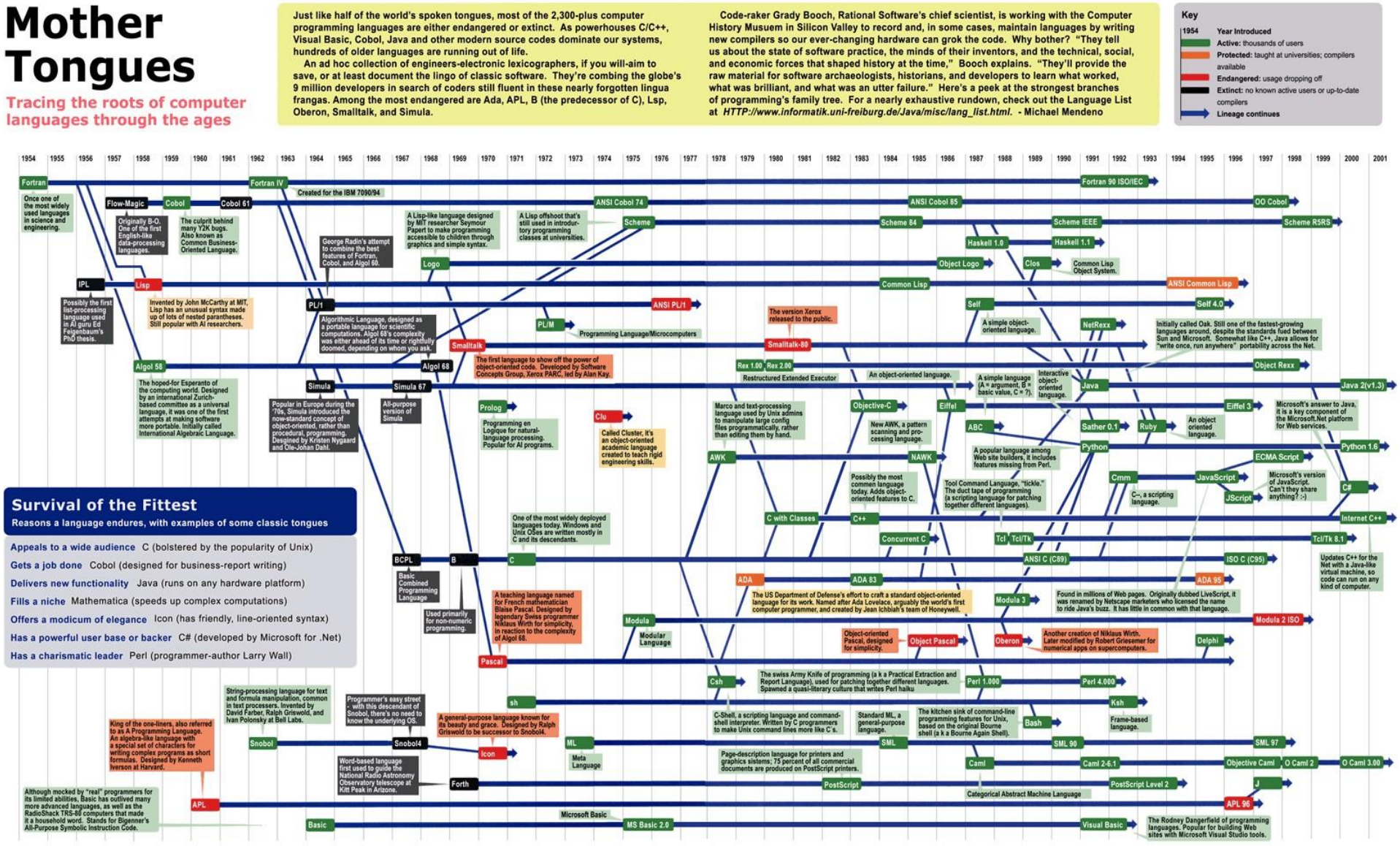
University of Southern California

Viterbi School of Engineering

Python Introduction

Review: High Level Languages



Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

http://www.digibarn.com/collections/posters/tongues/ComputerLanguagesChart-med.png

Concepts related to Python

PYTHON PROGRAMMING

Background

- Designed by the Guido van Rossum
  - Was conceived in the late 80s. Implementation was started in Dec. 89. First released in 91 (version 0.9.0)
  - Python 1.0 – Jan. 94
  - Python 3.6.3 – Oct. 17
  - Python 3.7.2 – Dec. 18

- Maintained by the non-profit PSF (Python Software Foundation)
- Python includes automatic memory management, dynamic type system
- Why Python? Why not Perl?

Motivation

- Rich lib/package support
  - "That's one of the nice things. I mean, part of the beauty of me is that I'm very rich"
- Good syntax (readable, takes fewer # code lines than C++, to express a concept)
  - "it's very hard for them to attack me on looks, because I'm so good looking"
- Job prospects
  - "I will be the greatest jobs president that God has ever created"

Features – OOP

- Supports encapsulation, inheritance, polymorphism
- Everything is public

Features – Interactive vs Scripts

- Can invoke Python and work interactively
  - % python

    >>> print "Hello World" // command prompt

    Ctrl-D (Linux/Mac) [Ctrl-Z Windows] at the prompt will exit.
- Can write code into a text file and execute that file as a script
  - % python myscript.py

```
print "Hello world"
```

myscript.py

- Can turn on executable bit and execute directly
  - % chmod u+x myscript.py
  - % ./myscript.py

```
#! /usr/bin/env python
print "Hello world"
```

myscript.py

Review: Compiled vs. Interpreted Languages

Compiled (C/C++)

- Requires code to be converted to the native machine language of the processor in the target system before it can be run
- Analogy: Taking a speech and translating it to a different language ahead of time so the speaker can just read it
- Faster
- Often allows programmer closer access to the hardware

Interpreted (Matlab / Python)

- Requires an interpreter program on the target system that will interpret the program source code command by command to the native system at run-time
- Analogy: Speaking through an interpreter where the speaker waits while the translator interprets
- Better portability to different systems
- Often abstracts HW functionality with built-in libraries (networking, file I/O, math

https://www.youtube.com/watch?v=qaj7nO1HUqA

Features – Interpreted

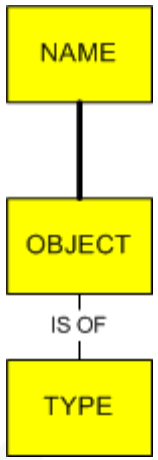
- Executes instructions directly
  - Does not compile the instructions to machine language beforehand
- The interpreter translates, in real-time, each instruction into a sequence of one or more subroutines already compiled in machine code
- Intermediate language 
  - Not compiled straight to machine code but to an intermediate byte code

    Byte code is then interpreted by the interpreter (Cpyton, Jython, etc.)

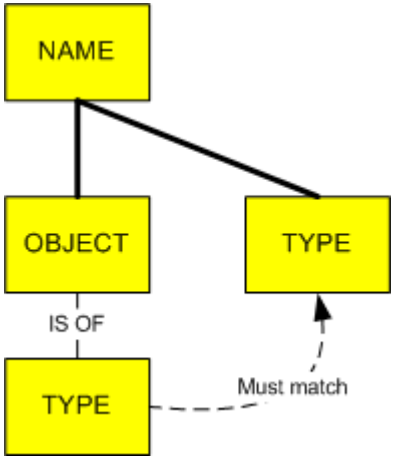
    Can write directly in the shell, or execute scripts

Features – Dynamic System

- **Dynamic typing or Dynamically typed**
  - **Performs type checking at run-time as opposed to Compile-time**



- **Question: Is a language with the following sample code, dynamic typed or statically typed:**
  - **VarName = 7**
  - **VarName = "Seven"**



- **Question: Is C++ a statically typed language?**

Features – Who is Strong?

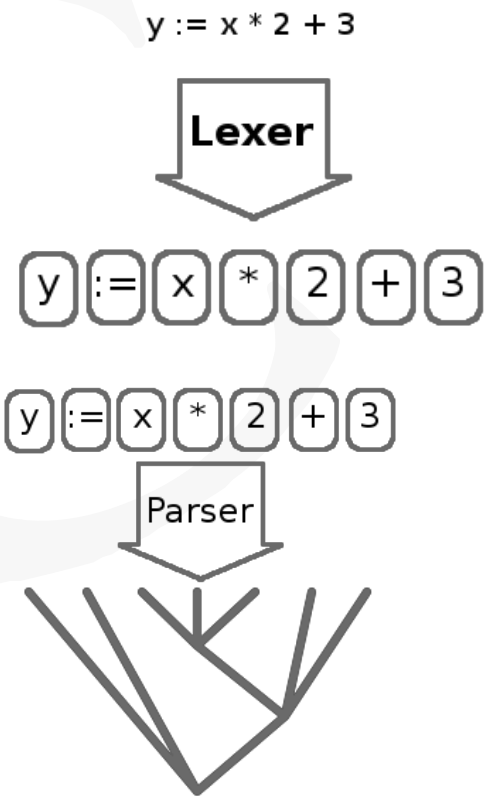
- A weakly typed language may produce unpredictable results or may perform implicit type conversion
- A strongly typed language is more likely to generate an error or refuse to compile if the argument passed to a function does not closely match the expected type

```
a  = 9
b = "9"
c = concatenate(  str(a),  b)
d = add(a,  int(b)  )
```

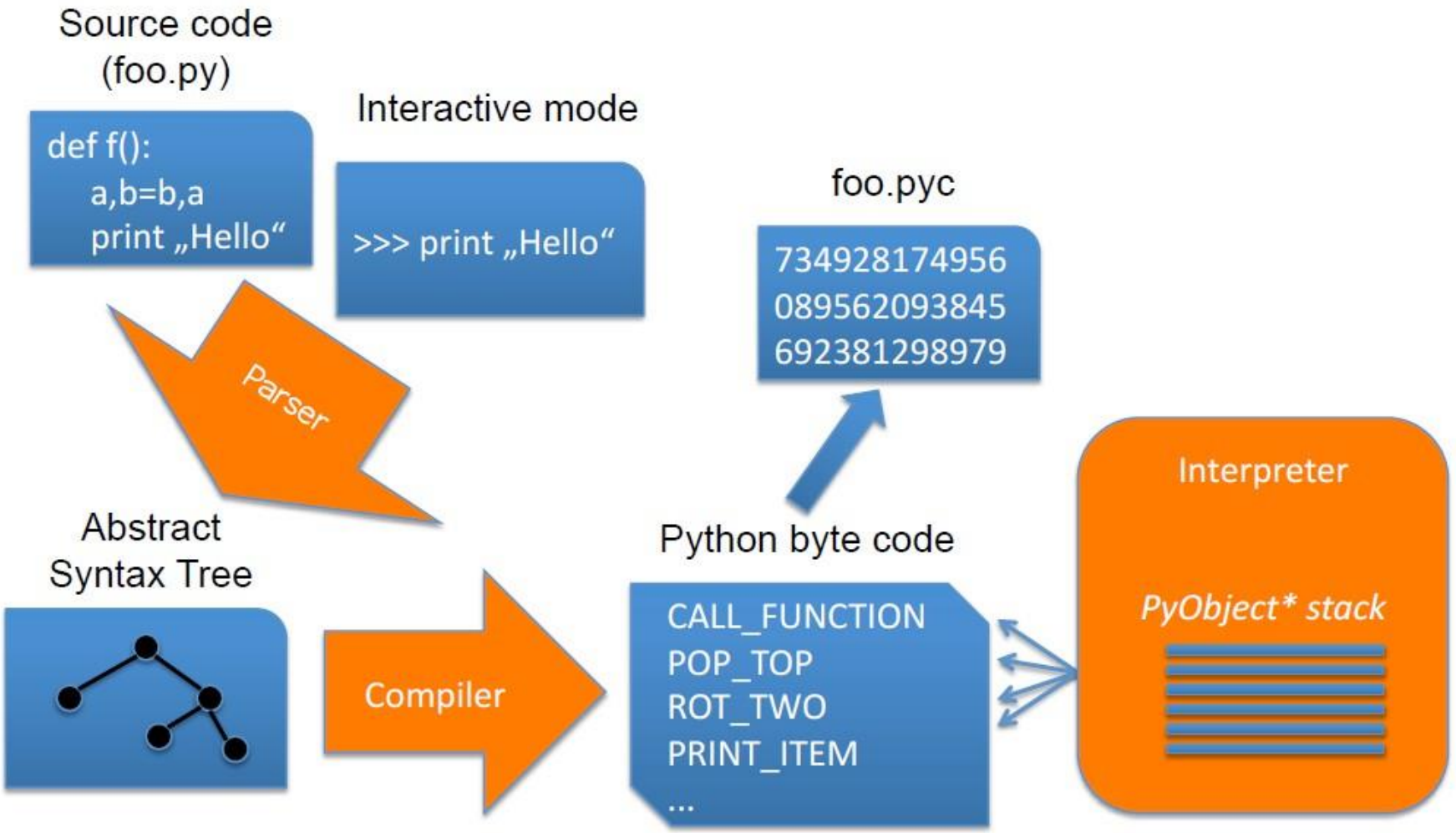
```
a  = 9
b = "9"
c = concatenate(a, b)  // produces "99"
d = add(a, b)          // produces 18
```

Execution Steps

- Question: How is a line of Python code executed? What are the steps when you hit return
  - Lexing
    - Breaking the lines of code you just typed into tokens
  - Parsing
    - Taking those tokens and generating an AST (Abstract Syntax Tree) to present their relationship
  - Compiling
    - Taking AST and turning it into one or more code objects
  - Interpreting
    - Taking each code object and executing the code it represents



CPython Interpreter



https://troeger.eu/files/teaching/pythonvm08.pdf

Python Setup

- Invoke the interpreter using the command ‘python’ on the server
- In addition to the platform we use in our class, you may consider the following:
  - To install on your local machine visit www.python.org
  - There are also various Python IDEs you can install on your local machine

Example Reference

- http://docs.python.org/tutorial/introduction.html

Example: Hello World

Python Code

```
>>> print 'Hello World!'
>>> print '1+1 = ', 1+1
```

Output

```
Hello World!
1+1 =  2
```

Executing scripts

- Write in any text editor
- Put extension .py
- Execute using

```
python <filename>.py
```

```
#!/usr/bin/python
print 'Hello World!'
```

Indentation

- No braces ({}) or semicolon
- Blocks are identified by looking at their indentation

Python Code

```
if x > y :
   print ‘X is greater’
else :
   print ‘y is greater’
```

Comments

- A hash sign is used for a single line comment
- There are no multi-line comments in Python

Python Code

```
is the first comment
is the second!
'Hello World!'
```

Types

- Types
  - Bool
  - Integers
  - Floats
  - Complex
  - Strings
- Dynamically typed
  - No need to "type" a variable
  - Python figures it out based on what it is assigned
  - Can change when re-assigned

```
>>> 2 + 1
3
>>> 2.5 + 1.0
3.5
>>> 2+4j + 3-2j
(5+2j)
>>> "Hello world"
'Hello world'
>>> 5 == 6
False
```

```
>>> x = 3
>>> x = "Hi"
>>> x = 5.0 + 2.5
```

myscript.py

Python Code

```
x = 'abcde'
print x
```

Output

```
abcde
```

Strings (cont.)

- Enclosed in either double or single quotes
  - The unused quote variant can be used within the string
- Can concatenate using the '+' operator
- Can convert other types to string via the str(x) method

```
>>> 'spam eggs'
'spam eggs'

>>> 'doesn\'t'
"doesn't"

>>> "doesn't"
"doesn't"

>>>'"Yes," he said.'
'"Yes," he said.'

>>> "Con" + "cat" + "enate"
'Concatenate'

>>> i = 5
>>> j = 2.75
>>> "i is " + str(i) + " & j is" + str(j)
'i is 5 & j is 2.75'
```

Python Code

```
answer = 101
pi = 3.141592654
print 'The answer is %i and the value of pi is %f'
%(answer,pi)
print 'The answer is %i and the value of pi is
%.2f' %(answer,pi)
```

Output

```
The answer is 101 and the value of pi is 3.141593
The answer is 101 and the value of pi is 3.14
```

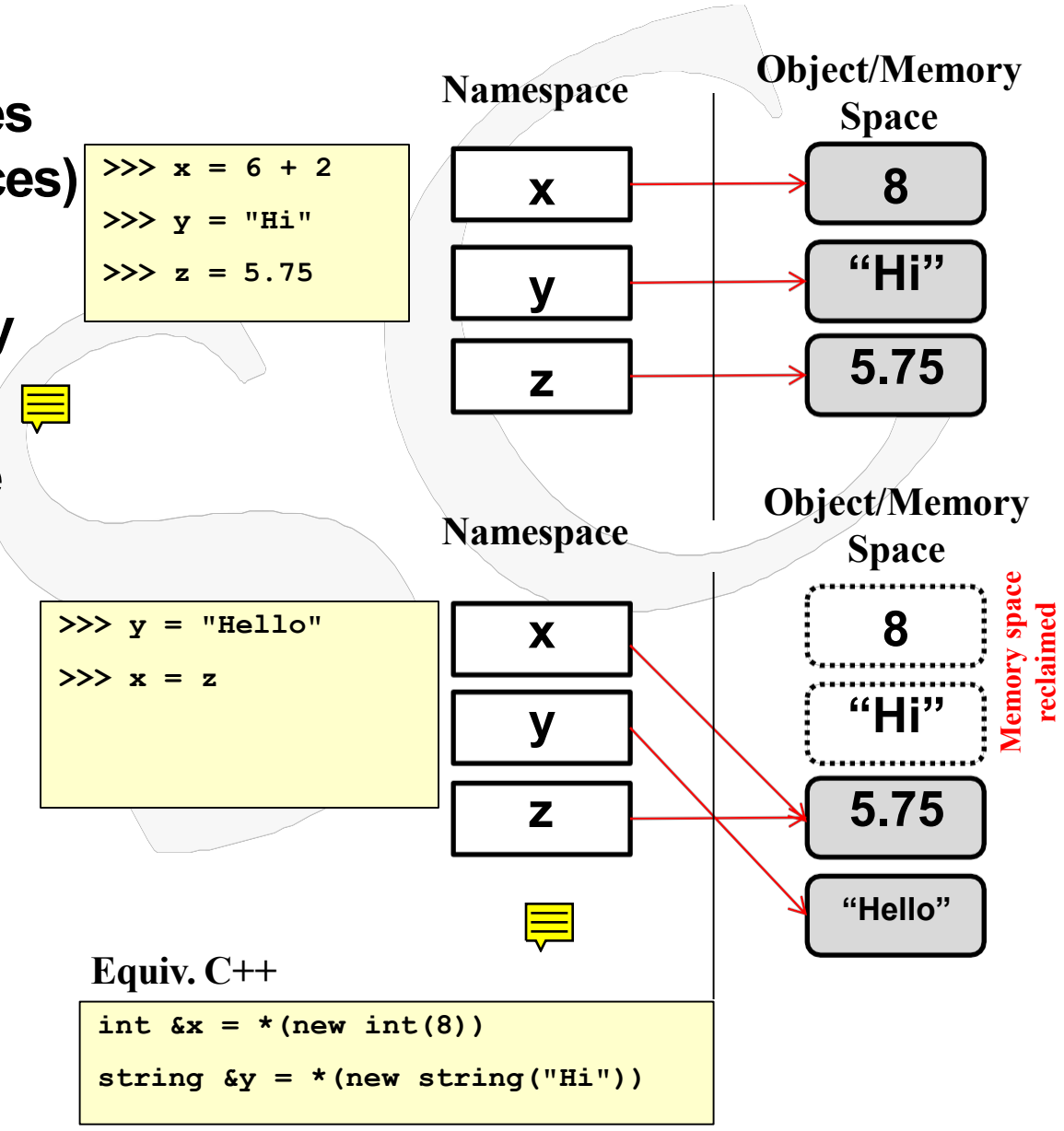
Simple Console I/O

- Print to display using 'print'
  - If ended with comma, no newline will be output
  - Otherwise, will always end with newline
- raw_input allows a prompt to be displayed and whatever text the user responds with will be returned as a string
- Use int(*string_var*) to convert to integer value
- Use float(*string_var*) to convert to float/double value

```
>>> print 'A new line will'
>>> print 'be printed'
A new line will
be printed

>>> print 'A new line will',
>>> print ' not be printed'
A new line will not be printed

>>> response = raw_input("Enter text: ")
Enter text: I am here

>>> print response = raw_input("Enter a num: ")
Enter a num: 6
6

>>> x = int(response)

>>> response = raw_input("Enter a float: ")
Enter a float: 6.25

>>> x = float(response)
```

Variables & Memory Allocation

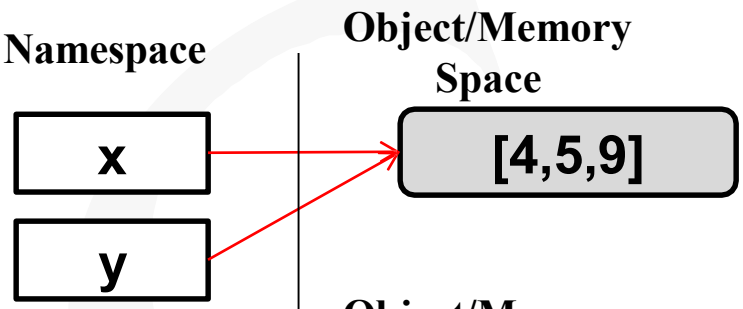
- All variables can be thought of as references (i.e., C++ style references)
- RHS expressions create/allocate memory for resulting object
- LHS is a variable name associated with that object
- Assignment changes that allocation
- When no variable name is associated with an with an object, that object is deallocated



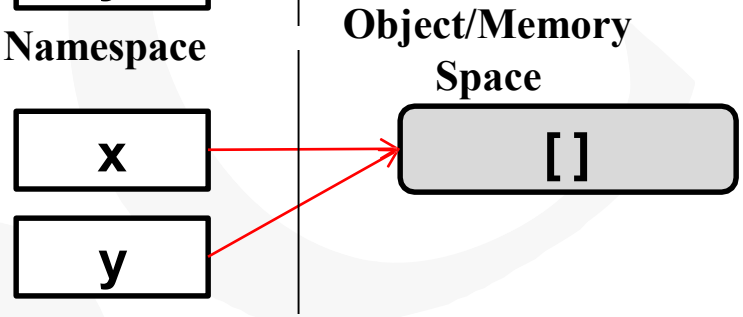
Variable/Reference Gotcha

- **Consider the following code**
- **What will the length of y be in the 2nd code box**

```
>>> x = [4,5,9]
>>> y = x
```

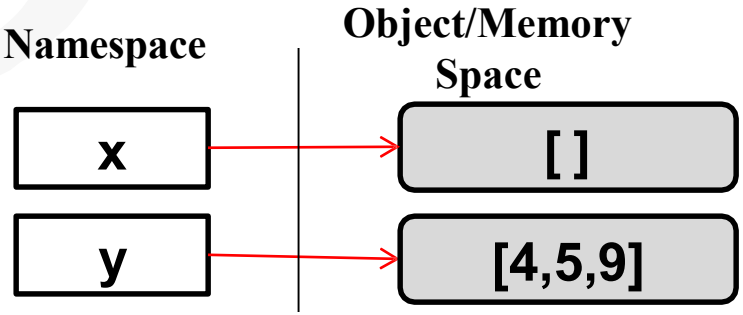



```
>>> x = []
>>> print len(y)
```



You think this is what you have…

```
>>> x = []
>>> print len(y)
3
```



…when, in fact, you have this

Integers and Floating points

Python Code

```
i = 2
j = 3
k = 3.0
print j/i
print k/i
```

Output

```
1
1.5
```

Tuple

- Like an array but can stores heterogeneous types of objects
- Comma separated values between parentheses
- Immutable
  - After initial creation, cannot be changed

```
>>> x = ('Hi', 5, 6.5)
>>> print x[1]
5
>>> y = x[2] + 1.25
7.75
>>> x[2] = 9.5
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item
assigned
```

List

- Like tuples but individual elements can be reassigned
- Comma separated values between square brackets
- Similar to vectors
  - Efficient at appending and popping back of the list
- Can contain heterogeneous types of objects
- Basic functions

```
>>> x = ['Hi', 5, 6.5]
>>> print x[1]
5
>>> y = x[2] + 1.25
7.75
>>> x[2] = 9.5
>>> x
['Hi', 5, 9.5]
>>> x.append(11)
['Hi', 5, 9.5, 11]
>>> y = x.pop()
>>> x
['Hi', 5, 9.5]
>>> y = x.pop(1)
>>> x
['Hi', 9.5]
>>> len(x)
2
```

Example

Python Code

```
x = [ 1,2,3,4,5]
print x
x[0] = 'a'
x[1] = (4,5,6)
print x
```

Output

```
[1, 2, 3, 4, 5]
['a', (4, 5, 6), 3, 4, 5]
```

Handy String Methods

- Can be subscripted
  - No character type…everything is a string
- Can be sliced using ':'
  - [Start:End]
  - Note end is non-inclusive
- Len() function
- Immutable type
  - To change, just create a new string

```
>>> word = 'HelpA'
>>> word [4]
'A'
>>> word[0:2]
'He'
>>> word[2:4]
'lp'

>>> word[:2]
'He'

>>> word[3:]
'pA'

>>> len(word)
5

>>> word[0] = 'y'

Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object does not support item assignment
>>> word = 'y' + word[1:]
```

Handy String Methods (cont.)

- find(), rfind()
  - Finds occurrence of substring and returns index of first character or -1 if not found
- in
  - Returns boolean if substring occurs in string
- replace()
- isalpha(), isnum(), isalnum(), islower(), isupper(), isspace()
  - return boolean results
- split(*delim*)
  - returns a list of the string split into multiple strings at occurrences of delim

```
>>> "HelpA".find('elp')
1

>>> "yoyo".rfind("yo")
2

>>> "yoyo".find("ooo")
-1

>>> "yo" in "yoyo"
True

>>> "yoyo".replace("yo","tu")
'tutu'

>>> "!$%".isalphnum()
False

>>> "A 123"[2:].isnum()
True

>>> x = "The cow jumped over the moon"
>>> x.split()
>>> x
['The','cow','jumped','over','the','moon']

>>> x[2]
'jumped'
```

Handy List Methods

- Can be sliced using ':'
  - [Start:End)
  - Note end is non-inclusive
- len() function
- item in list
  - Return true if item occurs in list
- sort()
  - Sorts the list in ascending order
  - Works in place
- reverse()
- min(), max(), sum() functions
- count(item)
  - Counts how many times item occurs in the list

```
>>> x = [4, 2, 7, 9]
>>> x[3]
9
>>> x[0:2]
[4,2]
>>> x[2:]
[7, 9]

>>> 2 in x
True

>>> min(x)
2

>>> x.sort()
[2, 4, 7, 9]

>>> x.append(4)
[9, 7, 4, 2, 4]

>>> x.count(4)
2
```

Selection Structures

- if…elif…else
- Ends with a ":" on that line
- Blocks of code delineated by indentation (via tabs)

```
#! /usr/bin/env python

myin = raw_input("Enter a number: ")

x = int(myin)

if x > 10:
     print "Number is greater than 10"

elif x < 10:
     print "Number is less than 10"

else:
     print "Number is equal to 10"
```

Example

Python Code

```
x=5
if x > 5 :
    print ‘x > 5’
else :
    print ‘x <= 5’
```

Output

```
x <= 5
```

Python Code

```
for num in range(4) :
      x = num+10;
      print x
print ‘done’
```

Output

```
10
11
12
13
done
```

Loops (cont.)

Python Code

```
names = ['Bob', 'Michael', 'Tom']
for aName in names :
      print ' name :',aName
```

Output

```
name : Bob
name : Michael
name : Tom
```

Loops (cont.)

Python Code

```
flag = False
x = 0
while not flag:
        x = x +3
        if x>9:
                flag = True
        print x
```

Output

```
3
6
9
12
```

Iterative Structures

- while <cond>:
- Again code is delineated by indentation

```
#! /usr/bin/env python

myin = raw_input("Enter a number: ")
i = 1
while i < int(myin):
     if i % 5 == 0:
          print  i
     i += 1
```

Iterative Structures (cont.)

- for item in collection:
- collection can be list, tuple, or some other collection we will see later
- For a specific range of integers just use range() function to generate a list
  - range(stop)
    - 0 through stop-1
  - range(start, stop)
    - start through stop-1
  - range(start, stop, stepsize)
    - start through step in increments of stepsize

```
#! /usr/bin/env python
# Prints 0 through 5 on separate lines
x = [0,1,2,3,4,5] # equiv to x = range(6)
for i in x:
     print i
# Prints 0 through 4 on separate lines
x = 5
for i in range(x):
     print i
# Prints 2 through 5 on separate lines
for i in range(2,6):
     print i
# Prints 0,2,4,6,8 on separate lines
for i in range(0,10,2):
     print i
```

List Iteration

- Can iterate through a list of any type item

```
#! /usr/bin/env python

# Prints each word on a separate line
x = "The cow jumps over the moon"
y = x.split()
for i in y:
     print i
```

Import Modules

- To import non-standard Python library support, use import
- sys module contains system access methods including access to command line arguments
- sys.argv is a list where each item is a string representing the i-th command line argument

myscript.py

```
#! /usr/bin/env python
import sys
print "Num args is " + str(len(sys.argv))
print "Command line args are:"
for arg in sys.argv:
     print arg
```

Command line

```
% python myscript.py hello 5 2.5
```

Output

```
Num args is 4
Command line args are:
myscript.py
hello
5
2.5
```

Python Code

```
import random
numbers = [1,2,3,4,5]
random.shuffle(numbers)
print numbers
```

Output

```
[4, 1, 3, 2, 5]
```

Assignment: Write a script

- Create a python script 'ex1.py'
- The script should let the user enter a list of numbers 1 at a time via 'raw_input', terminated with -1 (but don't store -1)
  - Actually store them in a list
  - Remember 'raw_input' returns a string representation…convert to integer via: int(*mystr*)
- Print the sum of all the elements
- Print the min and max of the elements
  - Try to do this the easy way…Don't iterate

File I/O

- open(filename,mode)
  - "r" = read text
  - "w" = write text
- readline()
  - Returns a single line of text
- readlines()
  - Returns a list of strings where each string is a line of text
- write(str)
  - Writes the string to a file

```
>>> infile = open("data.txt","r")
>>> outfile = open("new.txt","w")
>>> all_lines = infile.readlines()
['4 5 6','The cow jumped over the moon']
>>> first_line = all_lines[0]
>>> first_vals = first_line.split()
>>> sum = 0
>>> for v in first_vals:
      sum += int(v)
>>> outfile.write(str(sum) + '\n')
>>> num_words = len(all_lines[1].split())
>>> outfile.close()
>>> infile.close()
```

data.txt

```
4 5 6
The cow jumped over the moon
```

Example

Python Code

```
fileIn = open('file.in','r')
for aLine in fileIn:
      print aLine
fileIn.close()
```

Output

```
Line1
Line2
```

Example

Python Code

```
fileOut = open('file.out','w')
data = [1,2,3,4,5]
for item in data:
      fileOut.write( '%i\n' %(item))
fileOut.close()
```

Output (in file)

```
1
2
3
4
5
```

Dictionary (C++ Maps)

- Associates key, value pairs
- Key needs to be something hashable (string, int, etc.)
- Value can be anything
- Can get keys or values back in a list
- Can check membership

```
>>> grades = {}
>>> grades['Bill'] = 85
>>> grades["Jan"] = 100
>>> grades['Tom'] = 74
>>> for k in grades:
      print k
### Prints each key: 'Bill','Jan','Tom'
>>> grades.keys()
['Bill','Jan','Tom']
>>> grades.values()
[85,100,74]
>>> for k,v in grades.iteritems():
      print k + " got the score " + str(v)
>>> 'Jan' in grades
True
>>> grades['Jan'] + grades['Bill']
>>> grades['Tom'] = 75
```

Example

Python Code

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class':
'First'};
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

Output

dict['Name']: Zara
dict['Age']: 7

List Comprehensions

- Easy way to create new lists from current ones
- [new_expr for item in old_list]

```
>>> x = [1, 4, 7, 8]
>>> plus1 = [i+1 for i in x]
>>> names = ["Jane","Tom",”Bill”]
>>> namelen = [len(name) for name in names]
[4, 3, 4]
>>> lower_names = [name.lower() for name in names]
["jane","tom","bill"]
```

Functions and Modules

- def func(…):
  - Code
  - return statement
- To use in another python script or from interactive shell, use import
- Import only
  - Need to precede function name with module name (script name)
- From module import *
  - Just call function name

odd_even.py

```
#! /usr/bin/env python
def count_odds(inlist):
    num = 0
    for i in inlist:
        if i % 2 == 1:
            num += 1
    return num

def count_evens(inlist):
    return len(inlist) - count_odds(inlist)
```

```
>>> import odd_even
>>> odd_even.count_odds([3,4,8,9,5])
3
```

```
>>> from odd_even import *
>>> count_odds([3,4,8,9,5])
3
```

Python Code

```
import os
cmd = ‘echo Hello World!’
os.system(cmd)
```

Output

```
Hello World!
```

Python Code

```
#!/usr/bin/python
import sys
print 'Number of arguments:', len(sys.argv),
'arguments.'
print 'Argument List:', str(sys.argv)
```

Output

$ python test.py arg1 arg2 arg3
Number of arguments: 4 arguments.
Argument List: ['test.py', 'arg1', 'arg2', 'arg3']

Classes

- Place prototype and definition together in class
  - Variables initialized there are data members
- Everything is public
- Every member function takes a first argument of 'self' (like *this* pointer in C++)
- Within member functions, access data members by preceding with self

student.py

```
#! /usr/bin/env python
class Student:
   def __init__(self, n):
      self.name = n
      self.scores = []
   def get_name(self):
      return self.name
   def add_score(self, score):
      self.scores.append(score)
   def sum_scores(self):
      return sum(self.scores)
```

```
>>> import student
>>> s = student.Student("Bill")
>>> s.get_name()
"Bill"
```

More Topics

- OOP
- Parallel Computing
- Complexity

Resources

- Python documentations:
  - http://docs.python.org/2/tutorial/index.html
- Stack overflow
- Tutorials Point
- Just Google It