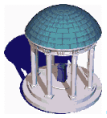


The University of North Carolina at Chapel Hill

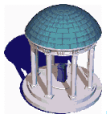
COMP 144 Programming Language Concepts

Spring 2002

Lecture 6: Introduction to Scripting Languages with Python

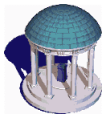
Felix Hernandez-Campos

Jan 23



Origin of Scripting Languages

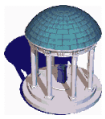
- Scripting languages originated as *job control languages*
  - 1960s: IBM System 360 had the Job Control Language
  - *Scripts* used to control other programs
    - » Launch compilation, execution
    - » Check return codes
- Scripting languages got increasingly more powerful in the UNIX world
  - Shell programming, AWK, Tcl/Tk, Perl
  - *Scripts* used to combine *components*
    - » *Gluing* applications [Ousterhout, 97]



System Programming Languages

- System programming languages replaced assembly languages
  - Benefits:
    - » The compiler hides unnecessary details, so these languages have a higher level of abstraction, increasing productivity
    - » They are *strongly typed*, *i.e.* meaning of information is specified before its use, enabling substantial error checking at compile time
    - » They make programs more portable
  - SPLs and ALs are both intended to write application from scratch
  - SPLs try to minimize the loss in performance with respect to ALs
  - *E.g.* PL/1, Pascal, C, C++, Java

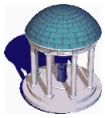
COMP 144 Programming Language Concepts
Felix Hernandez-Campos
3



Higher-level Programming

- Scripting languages provide an even higher-level of abstraction
  - The main goal is programming productivity
    - » Performance is a secondary consideration
  - Modern SL provide primitive operations with greater functionality
- Scripting languages are usually interpreted
  - Interpretation increases speed of development
    - » Immediate feedback
  - Compilation to an intermediate format is common

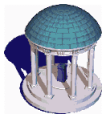
COMP 144 Programming Language Concepts
Felix Hernandez-Campos
4



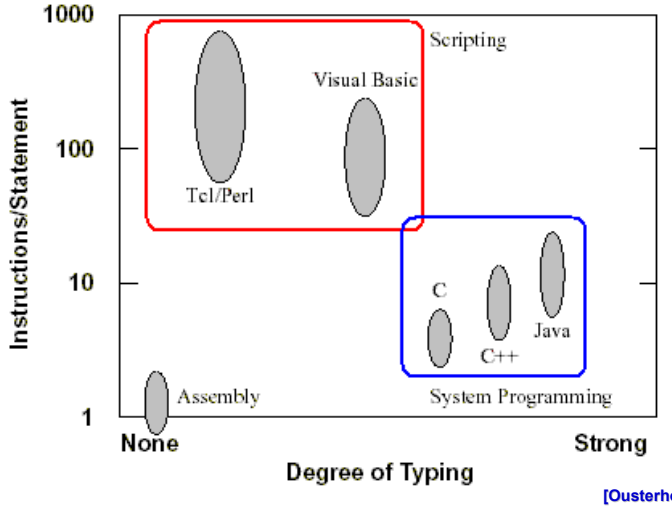
Higher-level Programming

- They are *weakly typed*
  - *I.e.* Meaning of information is inferred
  - ✖ Less error checking at compile-time
    - » Run-time error checking is less efficient, but possible
  - ✓ Weak typing increases speed of development
    - » More flexible interfacing
    - » Fewer lines of code
- They are not usually appropriate for
  - Efficient/low-level programming
  - Large programs

COMP 144 Programming Language Concepts
Felix Hernandez-Campos
5

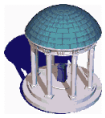


Typing and Productivity



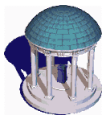
[Ousterhout, 97]

COMP 144 Programming Language Concepts
Felix Hernandez-Campos
6



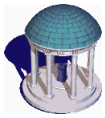
Python

- Guido van Rossum created Python in the early 90s
  - Named after *Monty Python's Flying Circus*
- Python is easy to learn
  - Simple, clean syntax
  - Elegant object-orientation
  - Good documentation
  - Friendly community
- Python is powerful
  - Efficient high-level data structures are part of the language
  - It has a very comprehensive set of standard libraries
  - It is easy to implement new functions in C or C++



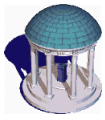
Feeding a 26-foot python

The python absolutely refused to eat anything, and while it is possible for a snake to refrain from food for a considerable period, there is an end even to the endurance of a snake. The authorities decided that extreme measures must be taken. The snake was firmly grasped by twelve men, and food, consisting of two rabbits and four guinea pigs, was pushed into its mouth by the aid of a pole. He was then put back into the cage to allow the processes of digestion to resume. (SciAm, 1902)



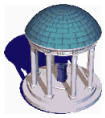
IDLE

- Integrated Development Environment
- We will follow a short tutorial: *One Day of IDLE Toying*
  - http://hkn.eecs.berkeley.edu/~dyoo/python/idle_intro/index.html
  - Download IDLE+Python 2.2: http://www.python.org/2.2/



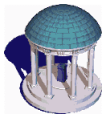
Built-in Data Structures: Numbers

- Integers, floating-point numbers, complex numbers, arbitrarily long integers
  - 345
  - 3.45
  - 3+45j
  - 59802738573890250873 45L
- Operators
  - +, –, *, /, **, %,...
  - `abs()`, `floor()`,...



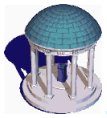
Built-in Data Structures: Strings

- Quotes sequences of characters
  - `'Comp 144\nProgramming Language Concepts'`
  - `"Python's tricks"`
- Raw mode
  - `r'Comp 144\nProgramming Language Concepts'`
- Operators
  - Concatenation `+`
    - » `"Programming " + "Language" + "Concepts"`
  - Repetition `*`
    - » `"COMP144 " * 5`



Built-in Data Structures: Strings

- Positional operators
  - Index `string[i]`
  - Slice `string[i:j]`
  - Length `len(string)`
- Formatting (extended `printf` notation)
  - `"This is %s %.1f" % ("python", 2.2)`
  - ```
    name = "python"
    ver = 2.2
    "This is %(name)s %(ver).3f" % vars()
    ```
    ```

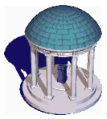


Built-in Data Structures: Lists

- Ordered collection of objects
  - They can contain *any* type of object
  - They are *mutable*
  - E.g.

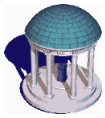
[]	Empty list
[1, ”2”, 3.0]	Three-element list
[1, [”2”, 4], 3.0]	Nested list

- Operators
  - Access `list[index]`
  - Deletion `del list[index]`
  - Length `len(list)`



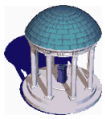
Built-in Data Structures: Lists

- Operators
  - Concatenation `+`
    - » `[1, 2] + [3, 4] + [5]`
  - Repetition `*`
    - » `[1, 2] * 5`
- Positional operators
  - Index `list[i]`
  - Slice `list[i:j]`
  - Length `len(list)`
- Generation
  - Ranges `range(start,end,step)`



Reading Assignment

- John K. Ousterhout, *Scripting: Higher-Level Programming for the 21st Century*, 1997
  - http://home.pacbell.net/ouster/scripting.html
- Guido van Rossum and Fred L. Drake, Jr. (ed.), *Python tutorial*, PythonLabs, 2001.
  - Read chapters 1 to 2
  - http://www.python.org/doc/current/tut/tut.html
  - Try some examples in IDLE



Additional References

- Mark Lutz and David Ascher, *Learning Python*, O'Really, 1999.
- Guido van Rossum and Fred L. Drake, Jr. (ed.), *Python Reference Manual*, PythonLabs, 2001.
  - *http://www.python.org/doc/current/ref/ref.html*