# 301AA - Advanced Programming

Lecturer: **Andrea Corradini**

andrea@di.unipi.it

http://pages.di.unipi.it/corradini/

***AP-24**: A brief intro to Scripting languages*

*Based on Chapter 13 of Programming Language Pragmatics*
by Michael L. Scott, 3rd edition

# Origin of Scripting Languages

- Modern scripting languages have two principal sets of ancestors.
  1. command interpreters or “shells” of traditional batch and “terminal” (command-line) computing
     - IBM’s **JCL**, **MS-DOS** command interpreter, Unix **sh** and **csh**
  2. various tools for text processing and report generation
     - IBM’s **RPG**, and Unix’s **sed** and **awk**.
- From these evolved
  – **Rexx**, IBM’s “Restructured Extended Executor,” ~1979
  – **Perl**, originally devised by Larry Wall in the late 1980s
  – Other general purpose scripting languages include **Tcl** (“tickle”), **Python**, **Ruby**, **VBScript** (for Windows) and **AppleScript** (for Mac)
  – **PHP** for server-side web scripting (and **JSP**, **VBScript**, **JavaScript**…)
  – And several others….

# Scripting Language: Common Characteristics

- Both batch and interactive use
  - Compiled/interpreted line by line
- Economy of expression
  - Concise syntax, avoid top-level declarations

```
class Hello { // Java
        public static void main(String[] args) {
            System.out.println("Hello, world!");
        }
}
-----------------------------------------
print "Hello, world!\n” # Python
```

- Lack of declarations
- Simple default scoping rules, which can be overruled via explicit declarations

# Scripting Language: Common Characteristics

– Dynamic typing, due to lack of declarations

– Flexible typing: a variable is interpreted differently depending on the context (kind of coercion)

```
$a = "4";         # Perl
print $a . 3 . "\n";         # '.' is concatenation
print $a + 3 . "\n";         # '+' is addition

        will print
43
7
```

– Easy access to system facilites

- Eg: **Perl** has more than 100 built-in commands for I/O, file/directory manipulation, process management, …
- *Note, **Perl** means **Perl 5**. **Perl 6** (now **Raku**) is different.*

## Scripting Language: Common Characteristics

- Sophisticated pattern matching and string manipulation
  - From text processing and report generation roots
  - Based on extended regular expressions
- High level data types
  - Built-in support for associative arrays implemented as hash tables.
  - Storage is garbage collected
- Quicker development cycle than industrial-quality languages (like Java, C++, C#, …)
  - Able to include state-of-the-art features (E.g., Python includes several new constructs seen in Java and Haskell)

# Problem Domains

- Some general purpose languages (eg. **Scheme** and **Visual Basic**) are widely used for scripting
- Conversely, some scripting languages (eg. **Perl**, **Python**, and **Ruby**) are intended for general use, with features supporting "programming in the large"
  - modules, separate compilation, reflection, program development environments
- But most scripting languages have principal use in ***well defined problem domains***:
  1. Shell languages
  2. Text Processing and Report Generation
  3. Mathematics and Statistics
  4. "Glue" Languages and General-Purpose Scripting
  5. Extension Languages
  6. Scripting the World Wide Web

# Variable declarations and Dynamic typing

- Most scripting languages (**Scheme** is an exception) do not require variables to be declared
- With or without declarations, most scripting languages use dynamic typing
- **Perl** and **JavaScript**, permit optional declarations - sort of compiler-checked documentation
- **Perl** can be run in a mode (**`use strict 'vars'`**) that requires declarations
- The interpreter can perform type checking at run time, or coerce values when appropriate
- **Tcl** is unusual in that all values—even lists—are represented internally as strings

# Scoping rules and nesting

- **Scheme**, **Python**, **JavaScript** provide the classic combination of nested subroutines and static (lexical) scope
- **Tcl** allows subroutines to nest, but uses dynamic scope
- Named subroutines (methods) do not nest in **PHP** or **Ruby**
  - **Perl** and **Ruby** join **Scheme**, **Python**, **JavaScript**, in providing first class anonymous local subroutines
- Nested blocks are statically scoped in **Perl**
  - In **Ruby** they are part of the named scope in which they appear
- **Scheme**, **Perl**, **Python** provide for variables captured in closures
- **PHP** and the major glue languages (**Perl**, **Tcl**, **Python**, **Ruby**) all have sophisticated namespaces
  - mechanisms for information hiding and the selective import of names from separate modules

# String and Pattern Manipulation

- **Regular expressions** are present in many scripting languages and related tools employ extended versions of the notation
  - extended regular expressions in sed, awk, Perl, Tcl, Python, and Ruby
  - grep, stand-alone Unix command, is a pattern-matching tool
- Two main groups.
  - The first group includes awk, egrep (the most widely used of several different versions of grep), the regex routines of the C standard library, and older versions of Tcl
    - These implement REs as defined in the **POSIX standard**
  - Languages in the second group follow Perl, which provides a large set of extensions, sometimes referred to as "**advanced REs**"

## Data Types

- As we have seen, scripting languages don't generally require (or even permit) the declaration of types for variables
- Most perform extensive run-time checks to make sure that values are never used in inappropriate ways
- Some languages (e.g., Scheme, Python, and Ruby) are relatively strict about this checking
  - When the programmer wants to convert from one type to another he must say so explicitly
- Perl (and likewise Rexx and Tcl) takes the position that programmers should check for the errors they care about
  - in the absence of such checks the program should do something "reasonable"

## Numeric types: “numeric values are simply numbers”

- In **JavaScripts** all numbers are double precision floating point
- In **Tcl** are strings
- **PHP** has double precision float and integers
- To these **Perl** and **Ruby** add ***bignums*** (arbitrary precision integers)
- **Python** also has complex numbers
- **Scheme** also has *rationals*
- Representation transparency varies: best in **Perl**, minimal in **Ruby**

## Composite data types

- Mainly **associative arrays** (based on hash tables)
- **Perl** has fully dynamic arrays indexed by numbers, and hashes, indexed by strings. Records and objects are realized with hashes
- **Python** and **Ruby** also have arrays and hashes, with slightly different syntax.
- **Python** also has sets and tuples
- **PHP** and **Tcl** eliminate distinction between arrays and hashes. Likewise **JavaScript** handles in a uniform way also objects.

# Object orientation

- Perl 5 has features that allow one to program in an object-oriented style
- PHP and JavaScript have cleaner, more conventional-looking object-oriented features
  - both allow the programmer to use a more traditional imperative style
- Python and Ruby are explicitly and uniformly object-oriented
- Perl uses a value model for variables; objects are always accessed via pointers
- In PHP and JavaScript, a variable can hold either a value of a primitive type or a reference to an object of composite type.
  - In contrast to Perl, however, these languages provide no way to speak of the reference itself, only the object to which it refers

# Object Orientation (2)

- Python and Ruby use a uniform reference model
- Classes are types in PHP, much as they are in C++, Java, or C#
- Classes in Perl are simply an alternative way of looking at packages (namespaces)
- JavaScript, remarkably, has objects but no classes
  - its inheritance is based on a concept known as *prototypes*
- While Perl's mechanisms suffice to create object-oriented programs, dynamic lookup makes both PHP and JavaScript more explicitly object oriented
- Classes are themselves objects in Python and Ruby, much as they are in Smalltalk
- In Ruby, **`2 * 4 + 5`** is syntactic sugar for **`(2.*(4)).+(5)`**, which is in turn equivalent to

```
(2.send('*', 4)).send('+', 5).
```

## Summary

- Scripting languages evolve quickly
- Able to incorporate latest features of programming language technology
- Quick learning curve
  - Widely used in teaching
- Huge libraries
- Very widely used, but pros and cons should be evaluated carefully…