What is a “Dynamic Language”

- A language in which many behaviors are deferred to run time.
  - Type checking
    - Type consistency of every expression checked at the time the expression is evaluated.
  - Code writing
    - Strings of characters can be interpreted as program text.
    - Precise definition of program entities depend on user input.
  - Code linking
    - Modules located and loaded at run time.
Pros and Cons

- **Pros**
  - Flexibility
    - Program can adapt as it runs to account for run time environment, user input, or errors that are encountered.
  - Easy of development
    - Compilation step is simple (and fast) because less work is done at compile time.

- **Cons**
  - Slow execution
    - Extra run time work requires processor cycles.
  - Less reliable
    - Static checking provides early bug detection.
Examples

- The “scripting” languages are usually dynamic.
  - Python
  - Perl
  - Ruby
  - ... a cast of others

- The “compiled” languages are usually static.
  - C/C++, Java, Scala, Ada, etc.
Distinction Can Be Unclear

- Many compiled languages do allow certain dynamic features.
  - Dynamic Link Libraries (*.dll) or Shared Object files (*.so) allow static languages to load code dynamically.
    - Requires OS support; feature exists outside the language.
  - Dynamic type checking can be simulated.
    - For example, in C using unions

- Some dynamic languages also support static features
  - Boo allows both static and dynamic type checking
Python Dynamic Type Checking

• Consider:
  
  - "Hello" + 1
    
    • It's a run time exception: TypeError: cannot concatenate 'str' and 'int' objects
  
  - if 1 < 2:
    
      print "Hello"
  
      else:
        
          print "Hello" + 1

    • It works fine, no type error because the bad expression isn't evaluated.
Python Dynamic Evaluation

- The `exec` statement lets you execute strings as program text.
  
  ```python
  exec "for i in range(1, 3):
      print i"
  
  The contents of the string is parsed and then executed.
  String could be built at run time based on user input, etc.

- The `eval` function lets you evaluate strings as Python expressions.
  
  ```python
  result = eval("1 + 2")
  
  The expression in the string is parsed and evaluated.
  String could be built at run time based on user input, etc."
Python Dynamic Definitions

- Precise class definition depends on condition

  - if 1 < 2:
    class Example:
      def method_1(self):
        print "I'm in method_1"
  else:
    class Example:
      def method_2(self):
        print "I'm in method_2"

- After the if statement executes, what methods does class Example have?
Dynamic Defs (Continued)

- Let's find out...
  - `ex = Example()`
    `ex.method_1()`
    - Print's “I'm in method_1”
  - `ex = Example()`
    `ex.method_2()`
    - **Raises:** `AttributeError: Example instance has no attribute 'method_2'`

- Methods in a class are checked dynamically.
  - Python runtime system verifies the existence of each method just before every call.
Python Import

• Modules brought into your program with import
  - `import mystuff`
    • *At run time*, Python searches for mystuff.py (or mystuff.pyc) and executes it.
    • Names defined in the module are now available for use in the importing module.

• Importing the same module more than once has no effect.
  - Module code only executed once.
  - BUT... names in the module still available!
Dynamic Module Selection

- Combine exec with modules.
  ```python
  if 1 < 2:
      module_name = "amod"
  else:
      module_name = "bmod"
  
  exec "import " + module_name
  ```
- Constructs the module name at run time.
- Uses `exec` to execute the necessary import.
- This is rarely done, but it illustrates Python's dynamic nature.