

G22.3033-002 Scripting Languages

7/24/2008
Other Scripting Languages

© Martin Hirzel G22.3033-002 NYU 7/24/2008 1

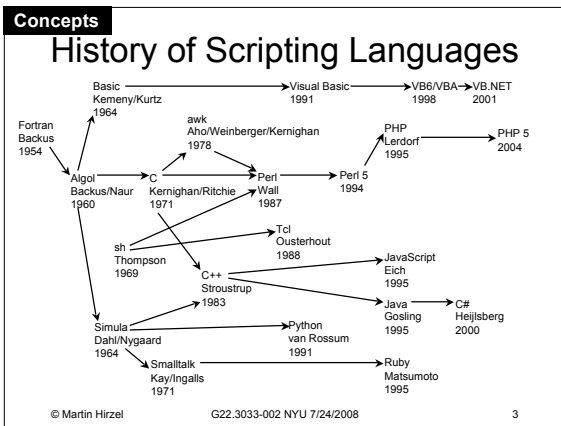
Concepts

Language Popularity Index

Position May 2008	Position May 2007	Delta in Position	Programming Language	Rating May 2008	Delta May 2007	Status
1	1	0	Java	20.170%	+1.04%	A
2	2	0	C	18.292%	+0.14%	A
3	5	2	(Visual) Basic	10.762%	+2.35%	A
4	4	0	PHP	10.657%	+1.90%	A
5	3	2	C++	10.484%	+0.37%	A
6	6	0	Perl	9.869%	-0.28%	A
7	7	0	Python	4.613%	+0.83%	A
8	8	0	C#	3.963%	+0.31%	A
9	10	1	Ruby	2.851%	+0.22%	A
10	11	1	Delphi	2.641%	+0.51%	A
11	9	2	JavaScript	2.411%	-0.60%	A
12	14	2	Perl 5	1.314%	-0.03%	A
13	12	1	SQL	0.739%	-1.34%	A
14	13	1	PL/SQL	0.629%	-1.36%	B

<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

© Martin Hirzel G22.3033-002 NYU 7/24/2008 2



Outline

- Bash
- Python
- Ruby

© Martin Hirzel G22.3033-002 NYU 7/24/2008 4

Bash

About Bash

- Bash is one of many Unix shells
- Unix shell = interactive command line interface
 - Used to manipulate files and control processes
 - Shell script = sequence of shell commands stored in a file and used like a program
 - As users wrote more elaborate shell scripts, shells grew into early scripting languages
- History / naming
 - Original Thompson shell (Ken Thompson 1969)
 - Replaced by Bourne shell (Stephen Bourne 1977)
 - Most shells today are Bourne-compatible
 - Bash = Bourne-Again SHell (Brian Fox 1987)
 - Bash is default on Linux, Mac OS X, and Cygwin

© Martin Hirzel G22.3033-002 NYU 7/24/2008 5

Bash

How to Write + Run Code

- Read-eval-print loop: **bash**
- Run script from file: **bash file.sh**
- Run script stand-alone: **#!/bin/bash**
- One-liner: **bash -c 'command'**
- Runs automatically:
 - .bashrc, .bash_profile**

© Martin Hirzel G22.3033-002 NYU 7/24/2008 6

Bash **Example**

```
#!/bin/bash
USAGE=$(cat <<EOF
Usage: compileall [options]
Options:
  -v | -verbose   Show executed commands.
  -p | -pretend   Don't actually compile, only show commands.
EOF
)
VERBOSE=false PRETEND=false
while [ 0 -lt $# ]; do
  case "$1" in
    -v | -verbose)  VERBOSE=true;;
    -p | -pretend)  PRETEND=true VERBOSE=true;;
    *)              echo "Unknown option $1."; echo "$USAGE"; exit 1;;
  esac
  shift
done
for f in *.c; do
  COMMAND="gcc -o ${f%.c}.exe $f"
  if [[ $VERBOSE == true || $PRETEND == true ]]; then echo $COMMAND; fi
  if [[ $PRETEND == false ]]; then $COMMAND; fi
done
```

© Martin Hirzel G22.3033-002 NYU 7/24/2008 7

Bash **Lexical Peculiarities**

- Single-line comments: #...
- Commands terminated by one of the control operators (;, &, &&, ||, &, *newline*)
- Expansion (see next slide)
 - Sigil \$ for variable use, but not when left-hand-side of assignment or declaration
 - Don't need quotes around strings
 - Don't need commas between parameters
- Heredocs

© Martin Hirzel G22.3033-002 NYU 7/24/2008 8

Bash **Expansion**

Left-to-right	Brace expansion	a{b,c}d ⇒ abd acd
	Tilde expansion	~ ⇒ /home/hirzel
	Parameter and variable expansion	\$foo ⇒ Like interpolation in Perl, value of \$foo
	Arithmetic expansion	\$((2+2)) ⇒ 4
	Command substitution	\$(..., `...`) ⇒ Output of executing
	Word splitting	Split unless grouped by "..." or '...'
	Pathname expansion	foo.* ⇒ foo.ppt foo.pdf
	Quote removal	'a\b\\' ⇒ a\b\
Command execution	Treat first word as function, builtin, or program, rest as parameters	

© Martin Hirzel G22.3033-002 NYU 7/24/2008 9

Bash **Structure of a Bash Application**

- *command* ::= (id=*expr*)* wordsAndRedirections controlOp
- *pipeline* ::= command (| *command*)*
- *list* ::= pipeline ((;|&&| |) *pipeline*)* (;|&|*newline*)
- *compoundCommand* ::=
 - (*list*) -- execute in subshell
 - {*list*;} -- execute in current shell environment
 - ((*expr*)) -- arithmetic; exit status 0 iff expression value !=0
 - [[*expr*]] -- conditional, with (...), !, &&, ||
 - controlStatement*
- Include file (run commands in current environment):
 - (source | .) *filename args*
- Exit status
 - Every command has an exit status
 - 0 treated as true by if, while, conditionals

© Martin Hirzel G22.3033-002 NYU 7/24/2008 10

Bash **Control Statements**

Conditional	if ...; then ...; elif ...; then ...; else ...; fi case id in pat ...) ...; ... esac
Fixed Loop	select id [in ...]; do ...; done for id [in ...]; do ...; done
Indefinite Loop	for ((...; ...; ...)); do ...; done while ...; do ...; done until ...; do ...; done
Unstructured control	break continue [n] return [<i>expr</i>] exit [n]

© Martin Hirzel G22.3033-002 NYU 7/24/2008 11

Bash **Types**

```

graph TD
    Root(( )) --- S((String))
    Root --- A((Array))
    Root --- I((Integer))
  
```

© Martin Hirzel G22.3033-002 NYU 7/24/2008 12

Bash

Variable Declarations

Implicit	<code>id=...</code>	Create at assignment
Explicit	<code>declare [attr] id=...</code>	-a: array -i: integer
	<code>typeset [attr] id=...</code>	-r: readonly -x: export
	<code>export id[=...]</code>	Set environment for later commands
	<code>readonly [attr] id=...</code>	-a: array
	<code>local [attr] id=...</code>	Same options as declare
Remove	<code>unset id</code>	Also for arrays
	<code>declare -n id</code>	Not for arrays

© Martin Hirzel G22.3033-002 NYU 7/24/2008 13

Bash

Type Conversions

Type	Value	String	Integer	Array
String	Integer	Identity	As number	One-element array
	Other single word		Error	
	Blank separated words	Error	Multi-element array	
Integer		As string	Identity	Singleton array
Array		First element	First element	Identity

© Martin Hirzel G22.3033-002 NYU 7/24/2008 14

Bash

Writing Subroutines

- Declaration: `[function] id () { ...; }`
 - Can contain explicit `return`
 - Can declare variables with `local`
- Arguments: not declared
 - Positional parameters in `$1, $2, ...`
 - At top level, `$1, $2, ...` = command line arguments
- Listing functions:
 - `declare -f / typeset -f`
 - List names only: `declare -F / typeset -F`
- Exporting functions to environment: `export`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 15

Bash

Operators

- Arithmetic operators (in `((...))`):
 - C operators, including `++, !, ~, +, -, *, /, <<, >>, <=, <`, etc., `&, |, &&, ||, ?:, =, *=`, etc.
- Conditional operators (in `[...]`):
 - File test operators: e.g., `-e` exists, `-d` is directory
 - Unary string operators: e.g., `-z` is zero
 - String comparison: `==, !=, <, >`
 - Arithmetic comparison: `-eq, -ne, -lt, -le, -gt, -ge`
- Control operators (in pipeline):
 - `||, &, &&, ;, ; ;, (,), |, newline`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 16

Bash

Input and Output

- Output: `echo "hello, world!"`
 - Suppress newline: `echo -n "hi"`
- Input from stdin to `id`: `read id`
 - Input from another file: `read -u fd id`
- Menu/read loop:

```

1) apple
2) banana
#? 2
You selected banana
#? 1
You selected apple
#? ^D
    
```

```

#!/bin/bash
select myInput in apple banana; do
  echo "You selected $myInput"
done
    
```

© Martin Hirzel G22.3033-002 NYU 7/24/2008 17

Bash

String Manipulation

- The bash manual talks about all variables as "parameters", not just positional parameters
 - Besides simple interpolation (`$id` or `${id}`), bash also manipulates strings during parameter expansion
 - Expand at top level or in `"..."`, but not in `'...'`
- Return length: `${#id}`
- Extract substring: `${id:offset:length}`
- Remove matching prefix or postfix:
 - Shortest prefix: `${id#pat}`
 - Longest prefix: `${id##pat}`
 - Shortest suffix: `${id%pat}`
 - Longest suffix: `${id%%pat}`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 18

Bash

Arrays

- Initialization:
 - From zero: `id= (expr ... expr)`
 - Indexed: `id= ([index]=expr ... [index]=expr)`
- Indexing:
 - Zero-based, non-contiguous, integers only
 - Write: `id [index]=expr`
 - Read: `$id [index]`
- Deleting:
 - Element: `unset id [index]`
 - Entire array: `unset id`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 19

Reference

Bash Documentation

- Manual page: `man bash`
- <http://www.gnu.org/software/bash/manual/bashref.html>

© Martin Hirzel G22.3033-002 NYU 7/24/2008 20

Bash

Evaluating Bash

<p><u>Strengths</u></p> <ul style="list-style-type: none"> • Good at manipulating files and processes • Same language for interaction and script 	<p><u>Weaknesses</u></p> <ul style="list-style-type: none"> • Lack of data structures • Difficult to manipulate strings • No debugger • No static checker
--	---

© Martin Hirzel G22.3033-002 NYU 7/24/2008 21

Outline

- Bash
- Python
- Ruby

© Martin Hirzel G22.3033-002 NYU 7/24/2008 22

Python

About Python

- First released 1991 by Guido van Rossum
 - Named after Monty Python comedy group; documentation uses examples from sketches
- General-purpose language
 - Not focused on just one domain only, such as application extension or web scripting
- Main focus: minimalist syntax, readability, comprehensive libraries (“batteries included”)
 - There should be only one obvious way to do it
- Used by YouTube and Google

© Martin Hirzel G22.3033-002 NYU 7/24/2008 23

Python

How to Write + Run Code

- One-liner: `python -c 'command'`
- Run script from file: `python file.py`
- Debugger: `python -m pdb file.py`
- Read-eval-print loop: `python`
- Run stand-alone: `#!/usr/bin/env python`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 24

Python

Example

```
#!/usr/bin/env python
import re, sys
cup2g = { 'flour': 110, 'sugar': 225, 'butter': 225 }
volume = { 'cup': 1, 'tbsp': 16, 'tsp': 48, 'ml': 236 }
weight = { 'lb': 1, 'oz': 16, 'g': 453 }
for line in sys.stdin.readlines():
    qty, unit, ing = re.search(r'([0-9.]+) (\w+) (\w+)', line).groups()
    if ing in cup2g and unit in volume:
        qty = float(qty) * cup2g[ing] / volume[unit]
        unit = 'g'
    elif unit in volume:
        qty = float(qty) * volume['ml'] / volume[unit]
        unit = 'ml'
    elif unit in weight:
        qty = float(qty) * weight['g'] / weight[unit]
        unit = 'g'
    sys.stdout.write('qty %d, unit %s, ing %s\n' % (int(qty),unit,ing))
```

© Martin Hirzel G22.3033-002 NYU 7/24/2008 25

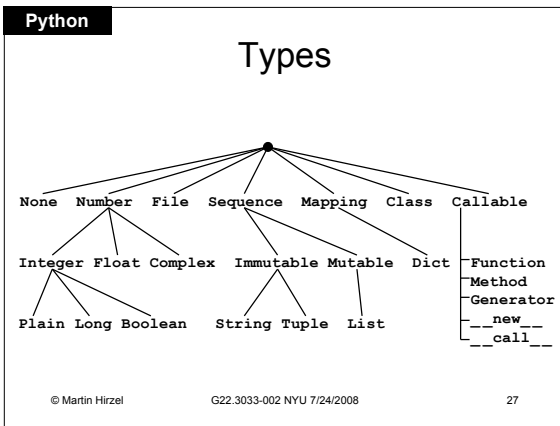
Python

Lexical Peculiarities

- Line break and indentation sensitive (!)


```
for line in sys.stdin.readlines():
    qty, unit, ing = re.search( \
        r'([0-9.]+) (\w+) (\w+)', line).groups()
    if ing in cup2g and unit in volume:
        qty = float(qty) * cup2g[ing] / volume[unit]
        unit = 'g'
    # "suite" ends at dedent
```
- Single-line comments: #...
- Literals
 - True, False, None, 123, 12.34, 's', "s"
 - No variable interpolation (use % operator instead)
 - Raw strings r'...' don't even interpolate backslash
 - Triple-quoted strings """...""" can contain newlines

© Martin Hirzel G22.3033-002 NYU 7/24/2008 26



Python

Variable Declarations

- There are no explicit variable declarations
- Reading an undefined variable is an error
- "Local if written" rule:
 - If function contains assignment "x=y", then "x" is a local variable, even if there is another global "x"
 - If function only reads "x", it uses the global "x"
 - If function contains statement "global x", then assignment "x=y" writes the global "x" if any
- No implicit "this"
 - Field and method access need explicit receiver even from within same class

© Martin Hirzel G22.3033-002 NYU 7/24/2008 28

Python

Type Conversions

Type	Value	Boolean	Number	String
bool	False	False	0	Error
	True	Identity	1	
int	0	False	Identity	Error
	Other	True		
str	""	False		Identity
	"0"	True	Error	
	Other	True		
	None	False	Error	Error
list	Empty	False	Error	Error
	Other	True		
dict	Empty	False	Error	Error
	Other	True		

Explicit conversions: int(x), float(x), str(x), ...

© Martin Hirzel G22.3033-002 NYU 7/24/2008 29

Python

Input and Output

- Output:
 - Statement: print "hi"
 - Statement without newline: print "hi",
 - Method call:


```
import sys; sys.stdout.write("hi")
```
- Input:


```
import sys;
line = sys.stdin.readline()
allLines = sys.stdin.readlines()
```

© Martin Hirzel G22.3033-002 NYU 7/24/2008 30

Python

Operators

[]	Array subscript
.	Attribute reference
**	2 Exponentiation
~	1 Bitwise negation
+, -	1 Positive, negative
*, /, %	2 Multiplicative
+, -	2 Additive
<<, >>	2 Shifts
!, ^, &	2 Bitwise (not all same precedence)
<, <=, >, >=, <>, !=, ==	2 Value comparison
is, is not	2 Identity comparison
in, not in	2 Membership test
or, and, not	2 Logical (not all same precedence)
lambda	Anonymous function

Assignment (=) is not an expression, but a statement

© Martin Hirzel G22.3033-002 NYU 7/24/2008 31

Python

Arrays

```

graph TD
    Sequence --> Immutable
    Sequence --> Mutable
    Immutable --> str["str  
'abc'"]
    Immutable --> tuple["tuple  
(1,2)"]
    Mutable --> list["list  
[1,2]"]
    
```

- More general concept: sequence
- All sequences support
 - Membership test: `m in s`
 - Concatenation: `s1 + s2`
 - Indexing (0-based): `s[i]`
 - Slicing: `s[lb:ub], s[lb:ub:step]`
 - Built-in functions: `len(s), min(s), max(s)`
- Constructing a list of numbers: `range(ub)`
- List mutation: `s[i]=m, s[lb:ub]=t, s.append(m), s.pop(), s.sort(), etc.`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 32

Concepts

List Comprehensions

- Python calls them “generator expressions”
- Concise syntax for generating lists:


```
listCompr ::= [expr forClause comprClause*]
forClause ::= for id in expr
comprClause ::= forClause | ifClause
ifClause ::= if expr
```
- Example:


```
l = [1,2,3,4]
t = 'a', 'b'
c1 = [x for x in l if x % 2 == 0]
c2 = [(x, y) for x in l if x < 3 for y in t]
print str(c1) # [2, 4]
print str(c2) # [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')]
```

```

c2 = []
for x in l:
    if x < 3:
        for y in t:
            c2.append((x,y))
            
```

© Martin Hirzel G22.3033-002 NYU 7/24/2008 33

Python

Hashes

- Python calls them “dictionary”, not “hash”
- Initialization:


```
d = {'lb': 1, 'oz': 16, 'g': 453}
or d = dict(lb=1, oz=16, g=453)
```
- Indexing: `d['lb']`
 - Can use any immutable type as key, including `int, str, tuple`
- Deleting: `del weight['lb']`
- Membership test: `k in d`
- Lots of other methods, e.g., `d.keys(), d.values(), d.has_key(k)`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 34

Python

Control Statements

Conditional	<code>if expr: ... [elif ...] [else: ...]</code>
Loops	Fixed <code>for target in expr: ... [else: ...]</code>
	Indefinite <code>while expr: ... [else: ...]</code>
Unstructured control	<code>break</code>
	<code>continue</code>
	<code>return [expr]</code>
	<code>yield [expr]</code>
Exception handling	<code>try: ... (except [Cls[, id]]: ...)* \</code> <code>[else: ...] [finally: ...]</code>
	<code>raise [expr]</code>

© Martin Hirzel G22.3033-002 NYU 7/24/2008 35

Python

Writing Subroutines

- Declaration: `def id(arg*): ...`
 - Can be nested
 - First-class: can store closure in variable
- Arguments: `arg ::= id | id = expr | *id | **id`
 - Mandatory argument: `id`
 - Optional argument: `id = expr`
 - Variable positional arguments: `*id`
 - Variable keyword arguments: `**id`
- Anonymous function: `lambda arg*: ...`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 36

Python>=2.5

Generators

```
#!/usr/bin/env python # 1
def myGenerator(x): # 2
    x = x + 3 # 3
    yield x # 4
    x = x + 3 # 5
    yield x # 6
    x = x + 3 # 7
    yield x # 8
myCoroutine = myGenerator(1) # 9
print '1st call:' #10
print myCoroutine.next() #11
print '2nd call:' #12
print myCoroutine.next() #13
print '3rd call:' #14
print myCoroutine.next() #15
print 'after 3rd call' #16
```

Python can also treat a generator result as an iterator:
for y in myGenerator(1): print y

© Martin Hirzel G22.3033-002 NYU 7/24/2008 37

Concepts

Co-Routines

- Coroutine = subroutine that suspends execution, but remembers state for later resumption
 - Requires saving and restoring (partial) stack
 - Useful for separating logic of producer (e.g., tree walk) from consumer (e.g., adding values in tree)
- Python "generators" are a new implementation of the old concept coroutines

© Martin Hirzel G22.3033-002 NYU 7/24/2008 38

Python>=2.2

Using Objects

```
a1 = Apple(150, 'green')
a2 = Apple(150, 'green')
a2.color = 'red'
print a1.prepare('slice')
print a2.prepare('squeeze')
```

Constructor calls
Field assignment
Method calls

© Martin Hirzel G22.3033-002 NYU 7/24/2008 39

Python>=2.2

Defining Classes

```
class Fruit:
    def __init__(self, weight):
        self.weight = weight
    def pluck(self):
        return 'fruit(' + self.weight + 'g)'
    def prepare(self, how):
        return how + 'd ' + self.pluck()
```

Fruit
weight
__init__()
pluck()
prepare()

© Martin Hirzel G22.3033-002 NYU 7/24/2008 40

Python>=2.2

Inheritance in Python

```
class Fruit:
    def __init__(self, weight):
        self.weight = weight
    def pluck(self):
        return 'fruit(' + self.weight + 'g)'
    def prepare(self, how):
        return how + 'd ' + self.pluck()

class Apple(Fruit):
    def __init__(self, weight, color):
        Fruit.__init__(self, weight)
        self.color = color
    def pluck(self):
        return self.color + ' apple'
```

© Martin Hirzel G22.3033-002 NYU 7/24/2008 41

Reference

Python Documentation

- Tutorial: <http://docs.python.org/tut/tut.html>
- Library reference: <http://docs.python.org/lib/lib.html>
- Language reference: <http://docs.python.org/ref/ref.html>
- Book: Dive Into Python. Mark Pilgrim. Apress, 2004. Available for free online: <http://diveintopython.org/>

© Martin Hirzel G22.3033-002 NYU 7/24/2008 42

Soap-Box

Evaluating Python

<p><u>Strengths</u></p> <ul style="list-style-type: none"> • Readability • Libraries • Simple yet powerful built-in data types • Good for larger projects 	<p><u>Weaknesses</u></p> <ul style="list-style-type: none"> • Indentation-sensitive • Unusual syntax • RegExps less tightly integrated
---	---

© Martin Hirzel G22.3033-002 NYU 7/24/2008 43

Outline

- Bash
- Python
- Ruby

© Martin Hirzel G22.3033-002 NYU 7/24/2008 44

Ruby

About Ruby

- Invented 1995 by Yukihiro “Matz” Matsumoto
- Influenced by SmallTalk
 - Everything is an object (even e.g., integers)
 - Blocks and user-defined control constructs
- Influenced by Perl
 - RegExp match =~ /.../, default variable \$_
- Common use: RoR (Ruby on Rails) framework for web programming

© Martin Hirzel G22.3033-002 NYU 7/24/2008 45

Ruby

How to Write + Run Code

- One-liner: `ruby -e 'command'`
- Run script from file: `ruby file.rb`
- Debugger: `ruby -rdebug file.rb`
- Check syntax only: `ruby -c file.rb`
- Read-eval-print loop (interactive Ruby): `irb`
- Run stand-alone: `#!/usr/bin/env ruby`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 46

Ruby

Example

```
#!/usr/bin/env ruby
$cup2g = { 'flour'=>110, 'sugar'=>225, 'butter'=>225 }
$volume = { 'cup'=>1, 'tbsp'=>16, 'tsp'=>48, 'ml'=>236 }
$weight = { 'lb'=>1, 'oz'=>16, 'g'=>453 }
while gets
  $_ =~ /([0-9.]+) (\w+) (\w+)/
  $qty, $unit, $ing = [$1, $2, $3]
  if $cup2g.key?($ing) and $volume.key?($unit)
    $qty = $qty.to_f * $cup2g[$ing] / $volume[$unit]
    $unit = 'g'
  elsif $volume.key? $unit
    $qty = $qty.to_f * $volume['ml'] / $volume[$unit]
    $unit = 'ml'
  elsif $weight.key? $unit
    $qty = $qty.to_f * $weight['g'] / $weight[$unit]
    $unit = 'g'
  end
  puts "qty #{ $qty.to_i }, unit #{ $unit }, ing #{ $ing }\n"
end
```

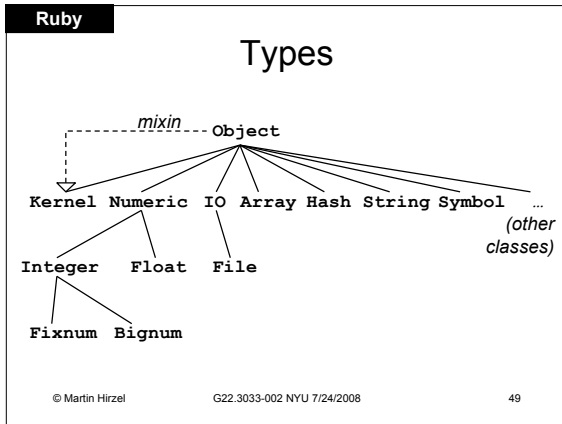
© Martin Hirzel G22.3033-002 NYU 7/24/2008 47

Ruby

Lexical Peculiarities

- Case sensitive
- Single-line comment: `#...`
- Multi-line comment: `=begin ... =end`
- Line break ends statement, optional semicolon (`;`)
- Sigils indicate scope, not type
- Heredocs
- Expression interpolation: `"... #{ ... } ..."`
- Literals: `"s"`, `'s'`, `true`, `nil`, RegExp `/.../`, Array `[1, 2, 3]`, Hash `{ 'x'=>1, 'y'=>2 }`, Symbol `:foo`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 48



Ruby

Variable Declarations

- There are no explicit variable declarations
- Reading an undefined variable is an error
- Scope

Scope	First letter
Local variable or method	Lowercase or <code>_</code>
Object variable (private)	<code>@</code>
Class variable (static)	<code>@@</code>
Global variable	<code>\$</code>

Constant; convention:

- MixedCase = class name
- ALL_CAPS = value

© Martin Hirzel G22.3033-002 NYU 7/24/2008 50

Ruby

Type Conversions

Class	Value	Boolean	Number	String
FalseClass	false	Identity	Error	Error
TrueClass	true	Identity	Error	Error
Fixnum	0	true	Identity	Error
	Other			
String	""			
	"0"	true	Error	Identity
NilClass	nil	false	Error	Error
Array	Empty			
	Other	true	Error	Error
Object		true	Error	Error

Explicit conversions: `x.to_i()`, `x.to_f()`, `x.to_s()`, ...

© Martin Hirzel G22.3033-002 NYU 7/24/2008 51

Concepts

Duck Typing

- If it walks like a duck and quacks like a duck, it's a duck.
- If a value supports the operations for a certain type, Ruby treats it as that type.
- This is normal for dynamic typing, but unusual for object-oriented languages.

© Martin Hirzel G22.3033-002 NYU 7/24/2008 52

Ruby

Input and Output

- Output: `p()`, `print()`, `printf()`, `puts()`
- Input: `gets()`, `readline()`, `readlines()`
- Like Perl, Ruby has implicit variables
 - E.g., `$1`, `$2`, ... are groups from pattern match, and `$_` is result of `gets` or `readline`
- Many operators are actually methods
 - E.g., `+` and `-` are methods on number objects, and `=~` is method on string objects for RegEx match

© Martin Hirzel G22.3033-002 NYU 7/24/2008 53

Ruby

Operators

<code>::</code>	2	Scope resolution
<code>[].[]=</code>	2	Read from array, write to array
<code>**</code>	2	Exponentiation
<code>+, -, !, ~</code>	1	Positive, negative, negation, complement
<code>*, /, %</code>	2	Multiplicative
<code>+, -</code>	2	Additive
<code><<, >></code>	2	Shifting
<code>&, , ^</code>	2	Bitwise (not all same precedence)
<code>>, >=, <, <=</code>	2	Comparison
<code><=>, ==, ===, !=, =~, !~</code>	2	Identity, pattern matching
<code>&&, </code>	2	Logical (not all same precedence)
<code>... ..</code>	2	Range inclusive, exclusive
<code>?:</code>	3	Conditional
<code>=, +=, -=, ...</code>	2	Assignment
<code>not, and, or</code>		Logical (not all same precedence)
<code>defined?</code>	1	(no precedence)

© Martin Hirzel G22.3033-002 NYU 7/24/2008 54

Ruby

Control Statements

Conditional	<code>if expr then ... elsif ... else ... end</code> <code>unless expr then ... elsif ... else ... end</code> <code>case expr when expr: else ... end</code>
Fixed	<code>for var in expr do ... end</code>
Loops	<code>while expr do ... end</code> <code>until expr do ... end</code>
Unstructured control	<code>break, redo, next, retry</code> <code>return [expr]</code> <code>yield</code>
Exception handling	<code>begin ... rescue Cls ... ensure ... end</code> <code>raise [Cls]</code>
Modifiers	<code>stmt [if unless while until] expr</code>

© Martin Hirzel G22.3033-002 NYU 7/24/2008 55

Ruby

Alternative Control Syntax

```

if x==5: puts 'five' end
if x==5 then puts 'five' end
if x==5
  puts 'five'
end
    
```

Less variation for other control statements, e.g.,

- while with do or newline
- for with do or newline

© Martin Hirzel G22.3033-002 NYU 7/24/2008 56

Ruby

Writing Subroutines

- Declaration: `def id [(arg*)] ... end`
 - Explicit `return`, or value of last expression in body
 - Convention: when `id` ends with `?`, return boolean; `=`, set field; `!`, make destructive update
 - `yield` executes block parameter; when block terminates, resume current subroutine after `yield`
- Arguments: `arg ::= [*]id`
 - Splat (`*`) gives variable number of arguments
- Deleting a subroutine: `undef id`
- Duplicating a subroutine: `alias id1 id2`
- Block (parameter closure): `block ::= { [arg*] ... }`
- Procs (first-class closure): `(proc | lambda) block`

© Martin Hirzel G22.3033-002 NYU 7/24/2008 57

Ruby

Iterators

```

#!/usr/bin/env ruby # 1
def myIterator(x) # 2
  x += 3 # 3
  yield x # 4
  x += 3 # 5
  yield x # 6
  x += 3 # 7
  yield x # 8
end # 9
$i = 0 #10
myIterator(1){|y| #11
  $i += 1 #12
  puts "call #{$i}:" #13
  puts y #14
} #15
puts 'done' #16
    
```

Many Ruby classes have iterator methods that take a block parameter.

© Martin Hirzel G22.3033-002 NYU 7/24/2008 58

Concepts

Code Blocks as Parameters

- Code block = `{ [arg*] stmt* }`
syntax for anonymous function
 - Similar to a lambda
 - Can be passed as parameter to iterator method
 - The code block is a closure (has access to caller's environment)

© Martin Hirzel G22.3033-002 NYU 7/24/2008 59

Ruby

Using Objects

<code>a1 = Apple.new(150, "green")</code> <code>a2 = Apple.new(150, "green")</code>	Constructor calls
<code>a2.color = "red"</code>	Setter call
<code>puts a1.prepare("slice") + "\n"</code> <code>puts a2.prepare("squeeze") + "\n"</code>	Method calls

© Martin Hirzel G22.3033-002 NYU 7/24/2008 60

Ruby Defining Classes

```
class Fruit
  def initialize(weight_)
    @weight = weight_end
  end
  def weight
    @weight end
  def weight= (value)
    @weight = value end
  def pluck
    "fruit(" + @weight + "g)" end
  def prepare(how)
    how + "d " + pluck end
end
```

Fruit
@weight
initialize()
pluck()
prepare()

- All fields are private, external use requires accessors (e.g., @weight, weight, weight=)
- Classes are open, can add additional fields+methods

© Martin Hirzel G22.3033-002 NYU 7/24/2008 61

Ruby Inheritance in Ruby

```
class Fruit
  def initialize(weight_)
    @weight = weight_end
  end
  def weight
    @weight end
  def weight= (value)
    @weight = value end
  def pluck
    "fruit(" + @weight + "g)" end
  def prepare(how)
    how + "d " + pluck end
end

class Apple < Fruit
  def initialize(weight_, color_)
    @weight = weight_
    @color = color_
  end
  def color
    @color end
  def color= (value)
    @color = value end
  def pluck
    self.color + " apple" end
end
```

Fruit
@weight
initialize()
pluck()
prepare()

Apple
@weight
@color
initialize()
pluck()
prepare()

© Martin Hirzel G22.3033-002 NYU 7/24/2008 62

Ruby Scopes and Visibility

- Visibility of class members
 - All instance variables are private
 - Methods can be private, protected, or public
- Accessor generation
 - Generates @weight field and weight/weight= methods

```
class Fruit
  attr_accessor :weight
  def initialize(weight_)
    @weight = weight_
  end
  def pluck
    "fruit(" + @weight + "g)"
  end
  def prepare(how)
    how + "d " + pluck
  end
end
```

Fruit
@weight
initialize()
pluck()
prepare()

© Martin Hirzel G22.3033-002 NYU 7/24/2008 63

Ruby Structure of a Ruby Application

- require file
- Module = like class, but can't be instantiated
 - Class can include ("mix in") one or more modules
 - Members of mix-in module are copied into class
 - Later definition with same name overrides earlier
 - Module can inherit from other module, but not class
 - Module can contain methods, classes, modules
- Module Kernel is mixed into class Object
- Top-level subroutines are private instance methods of the Kernel module
 - Visible everywhere, can't call with explicit receiver

© Martin Hirzel G22.3033-002 NYU 7/24/2008 64

Ruby Arrays

- Initialization: \$a=[1, 2, 3]
 - With block: \$a=Array.new(10){|e|2*e}
- Indexing: \$a[...]
 - Zero-based, contiguous, integers only
 - Negative index counts from end
- Deleting: \$a.clear(), \$a.compact(), \$a.delete_at(i)
- Lots of other methods

© Martin Hirzel G22.3033-002 NYU 7/24/2008 65

Ruby Hashes

- Initialization:


```
$h = {'1b'=>1, 'oz'=>16, 'g'=>453}
```
- Indexing: \$h['1b']
 - Can use any object as key, not just strings
- Deleting: \$h.clear(), \$h.delete(k)
- Lots of other methods
- Can have a "default closure":


```
return value for keys not explicitly stored
```

© Martin Hirzel G22.3033-002 NYU 7/24/2008 66

Reference

Ruby Documentation

- <http://www.ruby-lang.org>
- <http://www.rubyonrails.org>
- Book: The Ruby Programming Language. David Flanagan, Yukihiro Matsumoto. O'Reilly, 2008

© Martin Hirzel

G22.3033-002 NYU 7/24/2008

67

Soap-Box

Evaluating Ruby

Strengths

- Rails
- Purely object oriented
- Perl-like =~ and default variables

Weaknesses

- Less popular than Java and PHP
- Unusual syntax

© Martin Hirzel

G22.3033-002 NYU 7/24/2008

68

Administrative

Last Slide

- No announcements.
- Today's lecture
 - Bash
 - Python
 - Ruby
- Next lecture
 - Review
 - Requests?

© Martin Hirzel

G22.3033-002 NYU 7/17/2008

69