

G22.3033-002 Scripting Languages

7/31/2008
Review

© Martin Hirzel G22.3033-002 NYU 7/31/2008 1

Outline

- How to Describe a Language
- Dynamic Typing
- Text Processing
- Data Manipulation
- Closures and Objects
- Web Programming
- Security
- Debugging

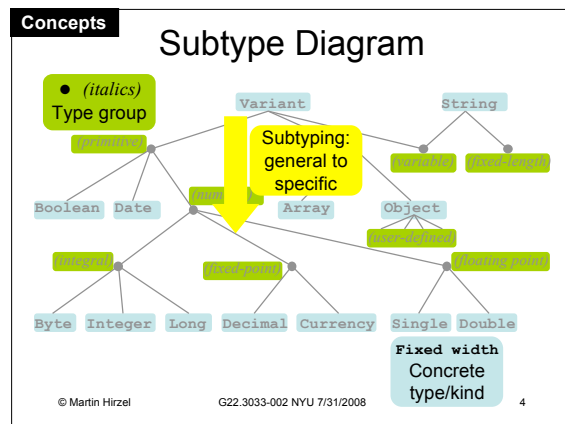
© Martin Hirzel G22.3033-002 NYU 7/31/2008 2

Soap-Box

How to Describe a Language

- About the language
- Related languages
- Example
- How to write + run code
- Lexical peculiarities
- Types **Subtype diagram**
- Variable declarations
- Type conversions **Conversion table**
- Input and output
- Operators **Operator table**
- Arrays
- Hashes
- Control statements **BNF**
- Writing subroutines **BNF**
- Using objects **UML object diagram**
- Structure of an application
- Scopes and visibility
- Embedded object model
- Defining classes **Object model**
- Inheritance **Class diagram**
- Call-backs **Sequence diagram**
- Library functions
- Reference documentation
- Common mistakes
- Suggestions for practice
- Evaluating the language

© Martin Hirzel G22.3033-002 NYU 7/31/2008 3



Concepts

Conversions Table

Column: target context

Value	bool	number	string
bool	Identity	0	""
number	false	Identity	"1"
string	true	false	0
bool	0	0	Identity
number	1	0	Identity
string	Other	true	Numeric prefix
bool	Other	false	0
number	Other	true	Numeric prefix
string	Other	false	0
bool	Other	true	Numeric prefix
number	Other	false	0
string	Other	true	Numeric prefix
bool	Other	false	0
number	Other	true	Numeric prefix
string	Other	false	0
bool	Other	true	Numeric prefix

Diagonal: Identity

© Martin Hirzel G22.3033-002 NYU 7/31/2008 5

Concepts

BNF = Backus Naur Form

Construct	Description	Example
Terminal	concrete code	Courier Bold
Non-terminal	placeholder	<i>Times Italic</i>
Ellipsis	omitted code	...
Rule	non-terminal definition	<i>Lhs ::= Rhs</i>
Alternative	choose one	<i>Alt1 Alt2</i>
Optional	zero or one times	[Square brackets]
Repeat	zero or more times	Kleene star*
Repeat	one or more times	Kleene plus+
Grouping	treat as unit	(parentheses)

© Martin Hirzel G22.3033-002 NYU 7/31/2008 6

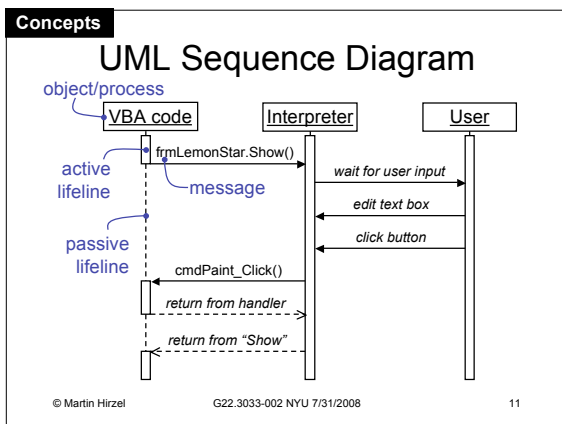
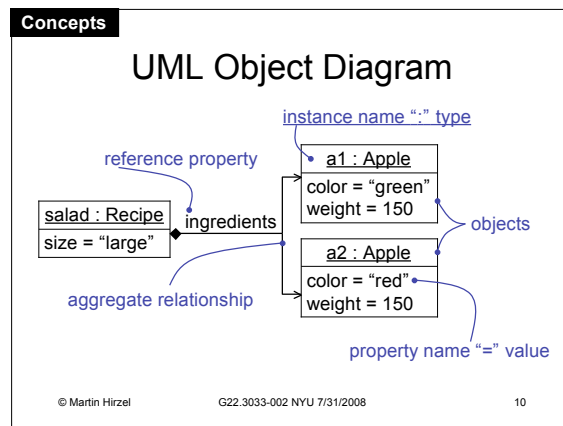
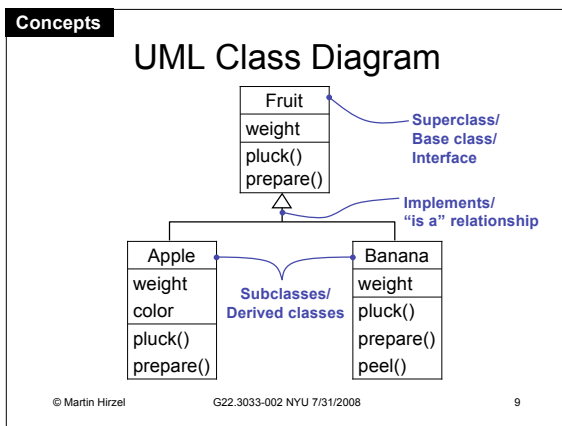
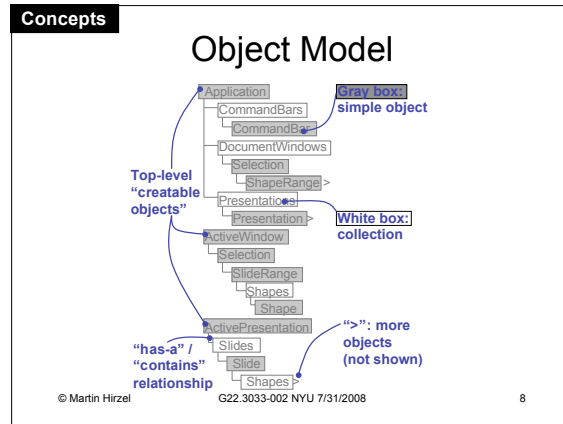
Concepts

Operator Table

(...)	Arity: 1 = unary 2 = binary	2	Associativity: all; Indexing L = left R = right
^		2	
+, -		1	
*, /		2	Multiplicative
\		2	Integer division
Mod		2	Modulus
+, -		2	Additive
&		2	String concatenation
<<, >>		2	Bit shift
=, <, <=, >, >=, Is		2	Comparison
Not		1	Negation
And, Or, Xor, Eqv, Imp		2	Logic (not all same precedence)
Set ... = ...		2	Assignment statement

Precedence: from high to low

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



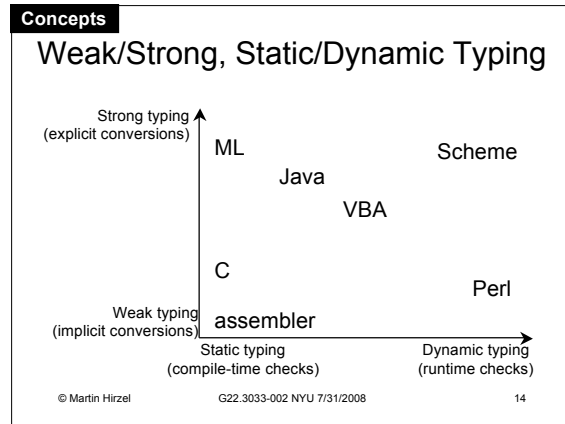
- Outline**
- How to Describe a Language
 - Dynamic Typing
 - Text Processing
 - Data Manipulation
 - Closures and Objects
 - Web Programming
 - Security
 - Debugging
- © Martin Hirzel G22.3033-002 NYU 7/31/2008 12

Comparison

How to Write + Run Code

		Bash	Perl	Python	Ruby	PHP	JavaScript	VBA
REPL	Read-eval-print loop (interactive interpreter)	✓	✓	✓	✓	✓	✓	✓
Batch	Run script from file (explicitly call interpreter)	✓	✓	✓	✓	✓		
Shebang	Run script from file (implicitly using #!interp.)	✓	✓	✓	✓	✓		
One-liner	Run short script from argument to interpreter	✓	✓	✓	✓			
Embedded (application)	Handle events (e.g., button click, timer)		✓					✓
Embedded (HTML)	HTML generation (server) or HTML parsing (client)				✓	✓	✓	✓

© Martin Hirzel G22.3033-002 NYU 7/31/2008 13



Comparison

Type Conversions

Source value	Target context	Bash	Perl	Python	Ruby	PHP	JavaScript	VBA
Number	Boolean	N/A	✓	✓	✓	✓	✓	✓
	String	✓	✓	✗	✗	✓	✓	✓
String	Boolean	N/A	✓	✓	✓	✓	✓	✓
	Number	✓	✗	✗	✓	✓	✓	✓
Array	Boolean	N/A	✓	✓	✓	✓	✓	✗
	Number	✓	✗	✗	✗	✗	✓	✗
	String	✓	✓	✗	✗	✓	✓	✗

© Martin Hirzel G22.3033-002 NYU 7/31/2008 15

- Perl**
- ### Context
- When you use a value in a different context than the value's type, Perl converts it
 - E.g., using array as scalar → int length
 - Context provided by:
 - Operator: e.g., `string . string`
 - Function: e.g., `print list`
 - Assignment: e.g., `@a = list`
 - Statement: e.g., `if (boolean) {...}`
 - Functions provide context for parameters, and can react to context for return value
- © Martin Hirzel G22.3033-002 NYU 7/31/2008 16

Perl

Context Example

Source code	Original Value	Context provider	Context	Converted value
<code>3+(10,11,12)</code>	List literal (10,11,12)	Operator +	Scalar (number)	<code>3+12 = 15</code>
<code>@a=(10,11,12); 3 + @a</code>	Array variable @a	Operator +	Scalar (number)	<code>3+3 = 6</code>
<code>@a=(10,11,12); length @a</code>	Array variable @a	Function <code>length</code>	Scalar (string)	<code>length "3" = 1</code>
<code>@a = "hi";</code>	Scalar literal "hi"	Assignment	List	<code>("hi")</code>
<code>sort 12,10,11</code>	List literal (12,10,11)	Function <code>sort</code>	List	<code>(10,11,12)</code>

© Martin Hirzel G22.3033-002 NYU 7/31/2008 17

Perl

List Conversions

Value	Context	
	Scalar	List
Scalar	identity	1-element list
List literal	last element	flattened sublists
Array	length	identity
Hash	empty	<code>"0"</code>
	non-empty	<code>"size/buckets"</code> $k_1, v_1, k_2, v_2, \dots$
<code>m/ /</code>	true iff match	all groups
<code><...></code>	end of file	<code>" "</code>
	not EOF	line

© Martin Hirzel G22.3033-002 NYU 7/31/2008 18

Concepts

Type System Variations

- Duck Typing
 - Object-oriented dynamic typing
 - Requires more dynamic method lookup than Java-like languages
 - Examples: Ruby, Python, PHP, Perl, VBA
- Gradual Typing
 - User chooses between static and dynamic typing on a per-variable basis (e.g. VBA)
- Pluggable Type Systems
 - User chooses to enable additional static checks on a per-file basis (e.g. Perl pragmas)

© Martin Hirzel G22.3033-002 NYU 7/31/2008 19

Outline

- How to Describe a Language
- Dynamic Typing
- **Text Processing**
- Data Manipulation
- Framework Support
- Web Programming
- Security
- Debugging

© Martin Hirzel G22.3033-002 NYU 7/31/2008 20

Concepts

Text Processing Example

Python

```
for line in sys.stdin.readlines():
    qty,unit,ing = re.search(r'([0-9.]+) (\w+) (\w+)', line).groups()
    ...
    print 'qty %d, unit %s, ing %s\n' % (int(qty), unit, ing)
```

Perl

```
while (<>) {
    my ($qty, $unit, $ing) = /([0-9.]+) (\w+) (\w+)/;
    ...
    printf("qty %d, unit %unit, ing %ing\n", $qty + .5);
}
```

Ruby

```
while gets
    $ =- /([0-9.]+) (\w+) (\w+)/
    $qty, $unit, $ing = [$1, $2, $3]
    ...
    puts "qty #{$qty.to_i}, unit #{$unit}, ing #{$ing}\n"
end
```

© Martin Hirzel G22.3033-002 NYU 7/31/2008 21

Comparison

Text Processing Features

	Bash	Perl	Python	Ruby	PHP	JavaScript	VBA
Implicit variable \$ _		✓		✓			
RegExp literal /.../		✓		✓		✓	
RegExp match operator =~		✓		✓			
Implicit variables \$1, \$2, ...		✓		✓			
String interpolation: sigils-var.	✓	✓			✓		
String interpolation: expression				✓	✓		
String formatting operator %			✓	✓			

© Martin Hirzel G22.3033-002 NYU 7/31/2008 22

Concepts

Chomsky Hierarchy

Type	Languages	Automata	Rules
0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$
1	Context sensitive	Linear bounded Turing machine	$\alpha B \gamma \rightarrow \alpha \delta \gamma$
2	Context free	Pushdown automaton	$A \rightarrow \beta$ (BNF)
3	Regular	Finite state machine	$A \rightarrow b, C \rightarrow dE$ (regexp)

- Formal regexp only has “essentials”
- Perl adds shortcuts and non-regular features

© Martin Hirzel G22.3033-002 NYU 7/31/2008 23

Perl

Inside a Regular Expression

Kind	Construct	Syntax	Example	
			Pattern	Matches
Essentials	Character	Itself	b	b
	Concatenation	$e_1 e_2$	bc	bc
	Alternative (or)	$e_1 e_2$	a bc	a, bc
	Repetition (≥ 0)	e^*	a*	, a, aa, aaa
	Grouping	(e)	(a b)c	ac, bc
Quantifier	Optional	$e?$	(a b)?	, a, b
	Repetition (≥ 1)	e^+	a+	a, aa, aaa
Char class	Custom class	[...]	[a-c]	a, b, c
	Wildcard character	.	.	a, 3, :
	Shortcut class	\...	\d	0,1,2,...,9
Assertion	Zero-width anchor	$\$, ^, \w, \dots$	\w	(word boundary)

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

Outline

- How to Describe a Language
- Dynamic Typing
- Text Processing
- **Data Manipulation**
- Closures and Objects
- Web Programming
- Security
- Debugging

© Martin Hirzel G22.3033-002 NYU 7/31/2008 25

Control Statements

	Bash	Perl	Python	Ruby	PHP	JavaScript	VBA
look-and-feel	<code>if expr; then stmt* fi</code>	<code>if (expr) { stmt* }</code>	<code>if expr: stmt* #end suite #[:indent]</code>	<code>if expr stmt* end</code>	<code>if (expr) stmt #alt. syntax: #endif;</code>	<code>if (expr) stmt</code>	<code>If expr stmt* End If</code>
for-in loop	✓	✓	✓	✓	✓	✓	✓
try...catch		✓	✓	✓		✓	
modifiers		✓		✓			
block args.		✓		✓			

© Martin Hirzel G22.3033-002 NYU 7/31/2008 26

Arrays

	Bash	Perl	Python	Ruby	PHP	JavaScript	VBA
Creation	<code>a=(4 9)</code>	<code>@a=(4,9)</code>	<code>l=[4,9]</code>	<code>a=[4,9]</code>	<code>\$a=array(4,9)</code>	<code>a=[4,9]</code>	<code>Dim a(2) a(0)=4:...</code>
Length	<code>\${#a[*]}</code>	<code>0+@a</code>	<code>len(l)</code>	<code>a.length</code>	<code>count(\$a)</code>	<code>a.length</code>	<code>a.Length</code>
Indexing	<code>\${a[0]}</code>	<code>\$a[0]</code>	<code>l[0]</code>	<code>a[0]</code>	<code>\$a[0]</code>	<code>a[0]</code>	<code>a(0)</code>
Deletion	<code>unset a</code>	<code>undef @a</code>	<code>del l</code>	<code>a=nil</code>	<code>unset(\$a)</code>	<code>a=null</code>	<code>Erase a</code>
Content type	Scalars	Scalars	Any	Any	Any	Any	Multi-dim.
Missing idx.	Insert	Insert	*	Insert	Insert	Insert	*
String idx.	*	*	*	*	✓	✓	*
Negative idx.	*	From end	From end	From end	As string	*	*
Slicing	*	<code>@a[0..2]</code>	<code>l[0:2]</code>	<code>a[0..2]</code>	Function	Method	*

© Martin Hirzel G22.3033-002 NYU 7/31/2008 27

Associative Arrays ("Hash"/"Dict")

	Bash	Perl	Python	Ruby	PHP	JavaScript	VBA
Creation	*	<code>%h=(x>4, y>9)</code>	<code>d={'x':4, 'y':9}</code>	<code>h={'x':>4, 'y':>9}</code>	<code>\$a=array('x'=>4, 'y'=>9)</code>	<code>o={'x':4, 'y':9}</code>	*
Length	*	<code>0+keys %h</code>	<code>len(d)</code>	<code>h.length</code>	<code>count(\$a)</code>	*	*
Indexing	*	<code>\$h(x)</code>	<code>d['x']</code>	<code>h</code>	<code>\$a['x']</code>	<code>o['x']</code>	*
Deletion	*	<code>undef %h</code>	<code>del d</code>	<code>h=nil</code>	<code>unset(\$a)</code>	<code>o=null</code>	*
Iteration	*	<code>for %k(keys %h)</code>	<code>for k in d.keys()</code>	<code>for k in h.keys</code>	<code>foreach(\$a as \$k=>\$v)</code>	<code>for(k in o)</code>	*
Content type	*	Scalars	Any	Any	Any	Any	*
Missing key	*	Insert	Insert	Insert	Insert	Insert	*
Key type	*	String	Immutable	Any	String	String	*

© Martin Hirzel G22.3033-002 NYU 7/31/2008 28

Perl Array and Hash Gotchas

- Subscripts use scalar sigil
 - `@a = (4,9)` # create array @a
 - `print $a[0]` # index into array @a, result 4
 - `print $a` # error: undefined variable \$a
- Contents must be scalars
 - Can't directly nest arrays/hashes
 - Work-around: use references, they are scalars
 - `@a = (4,9)` # create array @a
 - `$x = [2,7]` # create reference \$x to array
 - `@n = (@a, $x)` # create array @n with two refs
 - `print $n[1][0]` # same as `$x->[0]`, result 2

© Martin Hirzel G22.3033-002 NYU 7/31/2008 29

Perl Data Structures

```

$recipe = [
  {
    qty => 2,
    unit => "cup",
    ing => "flour",
  },
  {
    qty => 4,
    unit => "oz",
    ing => "sugar",
  },
  {
    qty => 3,
    unit => "tbsp",
    ing => "milk",
  },
];
    
```

© Martin Hirzel G22.3033-002 NYU 7/31/2008 30

Python 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 7/31/2008 31

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 7/31/2008 32

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 7/31/2008 33

Outline

- How to Describe a Language
- Dynamic Typing
- Text Processing
- Data Manipulation
- Closures and Objects
- Web Programming
- Security
- Debugging

© Martin Hirzel G22.3033-002 NYU 7/31/2008 34

Comparison Writing Subroutines

	Bash	Perl	Python	Ruby	PHP	JavaScript	VBA
Positional parameters	✓	✓	✓	✓	✓	✓	✓
Variable arguments	✓	✓	✓	✓	✓	✓	✓
Named parameters			✓				✓
Default values			✓		✓	✓	✓
Call-by-value	✓	✓	✓	✓	✓	✓	✓
Call-by-reference		✓			✓		✓
Nested subroutines	✓	✓	✓	✓	✓	✓	✓
Subroutine references		✓	✓	✓	✓	✓	✓
Anonymous subroutines		✓	✓	✓	✓	✓	✓

JavaScript, PHP, and Anonymous subroutines are grouped under Closures.

© Martin Hirzel G22.3033-002 NYU 7/31/2008 35

Concepts Closures

- Closure = subroutine reference + environment
- Motivation: keep state, e.g., for AJAX call-back
- Subroutine reference may be named or anonymous
- Environment = variable bindings from old scope chain
 - Scope chain > 1 requires nested subroutines

Example in JavaScript Example in Python Explanation

```
function f(y) {
    function g() {
        document.write(y);
    }
    return g;
}
var c = f(1);
var d = f(2);
c();
d();
```

```
def f(y):
    def g():
        print y;
    return g;
c = f(1)
d = f(2)
c()
d()
```

#c1→(fn, env)
#c→(→g, y→1)
#c→(→g, y→2)
print 1
print 2

© Martin Hirzel G22.3033-002 NYU 7/31/2008 36

Concepts

Emulating Objects using Closures

```
#!/usr/bin/env python
def stack():
    arr = [] # empty list
    def push(x):
        arr.append(x)
    def pop():
        return arr.pop()
    return (push, pop)
(s_push, s_pop) = stack()
s_push("apple")
s_push("pear")
print s_pop() # pear
print s_pop() # apple
```

Java does the opposite: emulating closures using objects
(e.g., java.util.Arrays.sort(Object[]a, Comparator c))

© Martin Hirzel G22.3033-002 NYU 7/31/2008 37

VBA

Using (Real) Objects

```
Dim a1 As Apple ' declare variables a1 and a2
Dim a2 As Apple ' of class Apple
Set a1 = New Apple ' allocate new objects, assign
Set a2 = New Apple ' references (need "Set" keyword)
a1.color = "green" ' set property, differently
a2.color = "red" ' for a1 and a2
a1.prepare("slice") ' call method, passing
a2.prepare("squeeze") ' string parameter
```

a1 : Apple	a2 : Apple
color = "green"	color = "red"
weight = 150	weight = 150

© Martin Hirzel G22.3033-002 NYU 7/31/2008 38

Comparison

Properties vs. Fields

- Both: dot notation look&feel
 - Writable: a1.color = "red"
 - Readable: Debug.print a1.color
- Properties only: associated getter/setter behavior
 - E.g., update graphical representation
- Realization:
 - Perl: ties
 - Python: __get__/_set__ (new-style classes)
 - Ruby: private fields, mandatory getter/setter
 - PHP: __get/_set
 - JavaScript: __defineGetter__, __defineSetter__
 - VBA: Property Get/Let/Set

© Martin Hirzel G22.3033-002 NYU 7/31/2008 39

Concepts

Progressive Disclosure

- Many users only use objects, but do not know how to define classes
 - Sufficient for small script using library/framework
- Of those users who define classes, many don't know how to define properties
 - Sufficient for large script or small library/framework
- Language design encourages this:
 - Learn subset of language to do most important tasks
 - Learn more to do more
- Progressive disclosure** user experience
 - Disclose (reveal) progressively (step by step)

© Martin Hirzel G22.3033-002 NYU 7/31/2008 40

JavaScript

Prototypes Example

```
function Fruit(weight)
{ this.weight = weight; }
Fruit.prototype.pluck = function()
{ return "fruit(" + this.weight + "g)"; }
Fruit.prototype.prepare = function(how)
{ return how + "d " + this.pluck(); }

function Apple(weight, color) {
    this.weight = weight;
    this.color = color;
}
Apple.prototype = new Fruit();
delete Apple.prototype.weight;
Apple.prototype.constructor = Apple;
Apple.prototype.pluck = function()
{ return this.color + " apple"; }

a = new Apple(150, "green");
document.write(a.prepare("slice"));
```

© Martin Hirzel G22.3033-002 NYU 7/31/2008 41

Outline

- How to Describe a Language
- Dynamic Typing
- Text Processing
- Data Manipulation
- Closures and Objects
- Web Programming
- Security
- Debugging

© Martin Hirzel G22.3033-002 NYU 7/31/2008 42

Concepts

HTML (HyperText Markup Language)

- Comment `<!-- ... -->`
- Element `<table>...</table>`
- Start tag `<table>`
- Attribute `border=5`
- Contents (between matching tags)
- End tag `</table>`
- Empty element `<hr .../>`
- Entity `&`

© Martin Hirzel G22.3033-002 NYU 7/31/2008 43

Concepts

HTTP (HyperText Transfer Protocol)

- HTTP GET: parameters encoded in URL
- HTTP POST: parameters in message header

© Martin Hirzel G22.3033-002 NYU 7/31/2008 44

Concepts

Embedding Code in Web Pages

- Server replaces PHP code by its own output, browser does not see it sent:
 - XML style (preferred!) `<?php ... ?>`
 - SGML style `<? ... ?>`
 - SGML style (implicit echo) `<?= ... ?>`
- User can trigger new page load
- Browser runs JavaScript while parsing HTML, user does not see it rendered:
 - Script style `<script>...</script>`
- User can trigger JavaScript event handler
- JavaScript can trigger new page load

© Martin Hirzel G22.3033-002 NYU 7/31/2008 45

Concepts

AJAX (Asynchronous JavaScript And XML)

© Martin Hirzel G22.3033-002 NYU 7/31/2008 46

Concepts

Scope in Server-Side Scripts

© Martin Hirzel G22.3033-002 NYU 7/31/2008 47

Concepts

Session State

© Martin Hirzel G22.3033-002 NYU 7/31/2008 48

Concepts

Relational Databases

- Database = collection of tables
- Table = relation = set of rows w/ same columns
- Row = tuple = one value per column
- Column = attribute = name+primitive type
- Only store primitive values, never nest tables

Recing				Cup2G		Volume		Weight	
rec	ing	qty	unit	ing	g	unit	cup	unit	lb
cake	flour	2.5	cup	flour	110	cup	1	lb	1
cake	milk	3	tbsp	sugar	225	tbsp	16	oz	16
gravy	salt	2	tsp	butter	225	tsp	48	g	453
gravy	sugar	10	g			ml	236		

© Martin Hirzel G22.3033-002 NYU 7/31/2008 49

Outline

- How to Describe a Language
- Dynamic Typing
- Text Processing
- Data Manipulation
- Closures and Objects
- Web Programming
- Security
- Debugging

© Martin Hirzel G22.3033-002 NYU 7/31/2008 50

Concepts

SQL Injection

× Untrusted input, may be malicious:
`*'drop Table y;--`

Space → Reading user input (HTML form)

Time ↓ Information flow (assignments, parameter passing)

Inject input into SQL:
`"select * from t + where name=$name"`

Sanitizer: change
× suspect input to
✓ make it harmless
`addslashes($name)`

Security breach:
e.g., database damaged

© Martin Hirzel G22.3033-002 NYU 7/31/2008 51

Concepts

XSS (Cross-Site Scripting)

× Untrusted input, may be malicious:
`name=<script>.</script>`

Space → Reading user input (URL-encoded args)

Time ↓ Information flow (assignments, parameter passing)

Inject input into HTML:
`echo "Welcome back," + " $name "`

Sanitizer: change
× suspect input to
✓ make it harmless
`htmlspecialchars($name)`

Security breach:
e.g., victim's password sent back to attacker

© Martin Hirzel G22.3033-002 NYU 7/31/2008 52

Concepts

Downgrading

- Information-security policy can establish that:
 - Certain parts of secret information can be declassified and revealed to public listeners. E.g.:
 - Last 4 digits of SSN can be revealed to bank teller
 - Result of a password check can be revealed to anyone
 - Certain parts of untrusted input can be endorsed and used in certain trusted computations. E.g.:
 - Untrusted user input can be used in web application if it is properly formatted ("sanitized")

	Integrity	Confidentiality
High	Untrusted	Secret
Low	Trusted	Public
Downgrading	Endorsement	Declassification

© Martin Hirzel G22.3033-002 NYU 7/31/2008 53

Outline

- How to Describe a Language
- Dynamic Typing
- Text Processing
- Data Manipulation
- Closures and Objects
- Web Programming
- Security
- Debugging

© Martin Hirzel G22.3033-002 NYU 7/31/2008 54

Concepts

Defects, Infections, Failures

✓ Sane State
 × Infection = wrong state
 Defect = wrong code that turns sane state into infection
 Infection spreads.
 Failure = wrong output observed by user
 Zeller avoids the word "bug", since it could mean any of the above.

© Martin Hirzel G22.3033-002 NYU 7/31/2008 55

Concepts

The Scientific Method

(Backup first)
 Hypothesis E.g., from looking at code
 Experiment E.g., run with certain input
 Observation E.g., using print statement
 Conclusion E.g., step of infection chain
 Repeat
 (Fix similar problems)

© Martin Hirzel G22.3033-002 NYU 7/31/2008 56

Concepts

Delta Debugging Example

```

sub sort_ref_to_array { #buggy!
    my @sorted = sort @_;
    return $sorted[0];
}

sub test_sort {
    my $arrayref = sort_ref_to_array($_[0]);
    for (my $i=0; $i+1 < @$arrayref; $i++) {
        if ($arrayref->[$i] > $arrayref->[$i+1]) {
            return 'fail'; # *
        }
    }
    return 'pass'; # ✓
}

our $min = ddmmin([1,3,5,2,4,6], \&test_sort);
print "minimized to ", @$min, "\n";
    
```

135246	*
246	✓
135	✓
5246	*
46	✓
52	*
2	✓
5	✓
52	*

© Martin Hirzel G22.3033-002 NYU 7/31/2008 57

Concepts

Software Vise

- Vise = holds an artifact firm for working on it
- Perl makes it easy to build vise for batch application
- VBA allows you to build vise for GUI application

© Martin Hirzel G22.3033-002 NYU 7/31/2008 58

Administrative

Last Slide

- hw10 due tomorrow = Friday 1 August
- Final exam Thursday 7 August
 - During regular class hours = 6pm-8:20pm
 - In the regular class room = CIWW 109
- Now: Course evaluations
- Today's lecture
 - Review
- Next week
 - Final exam

© Martin Hirzel G22.3033-002 NYU 7/31/2008 59