University of Arizona

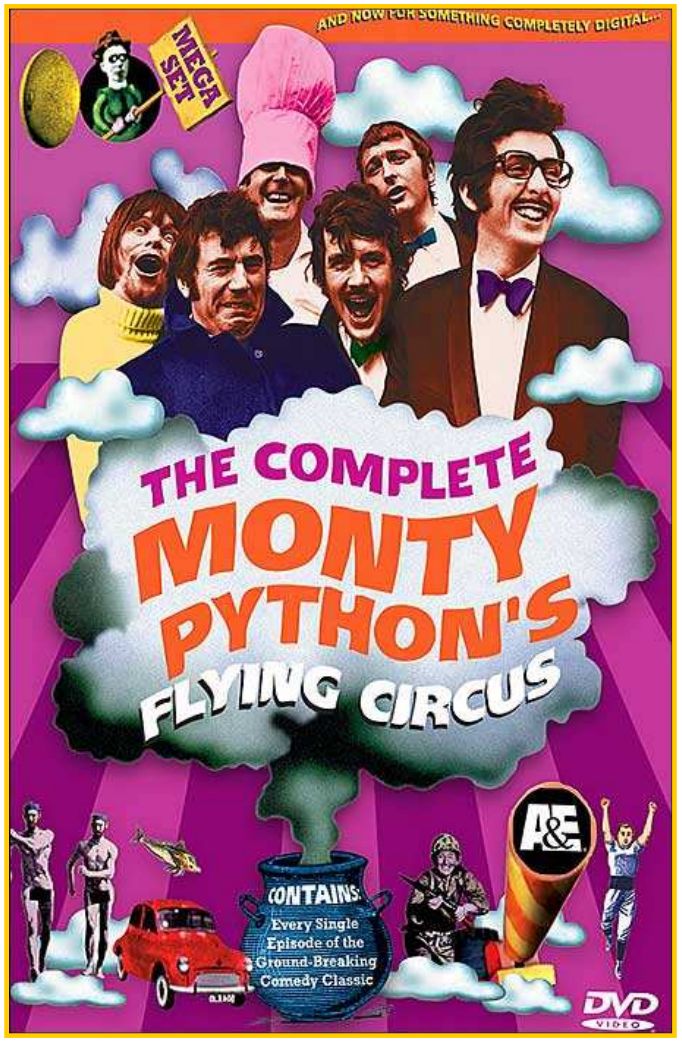
College of Science / Dept. of Computer Science

cs520 Prin. of Prog. Languages | Prof. Christian Collberg

Raquel Torres Peralta & Federico Cirett Galan

History

- Facts:
  - Creator: Guido van Rossum
  - Based on:
    - ABC
    - Modula-3
  - Name:
    - From a TV show



python

Python

- High-level scripting language
- Multi-paradigm
- Easy to learn
- Stack-based interpreter
- Garbage collection using reference count algorithm

Python

Goals:

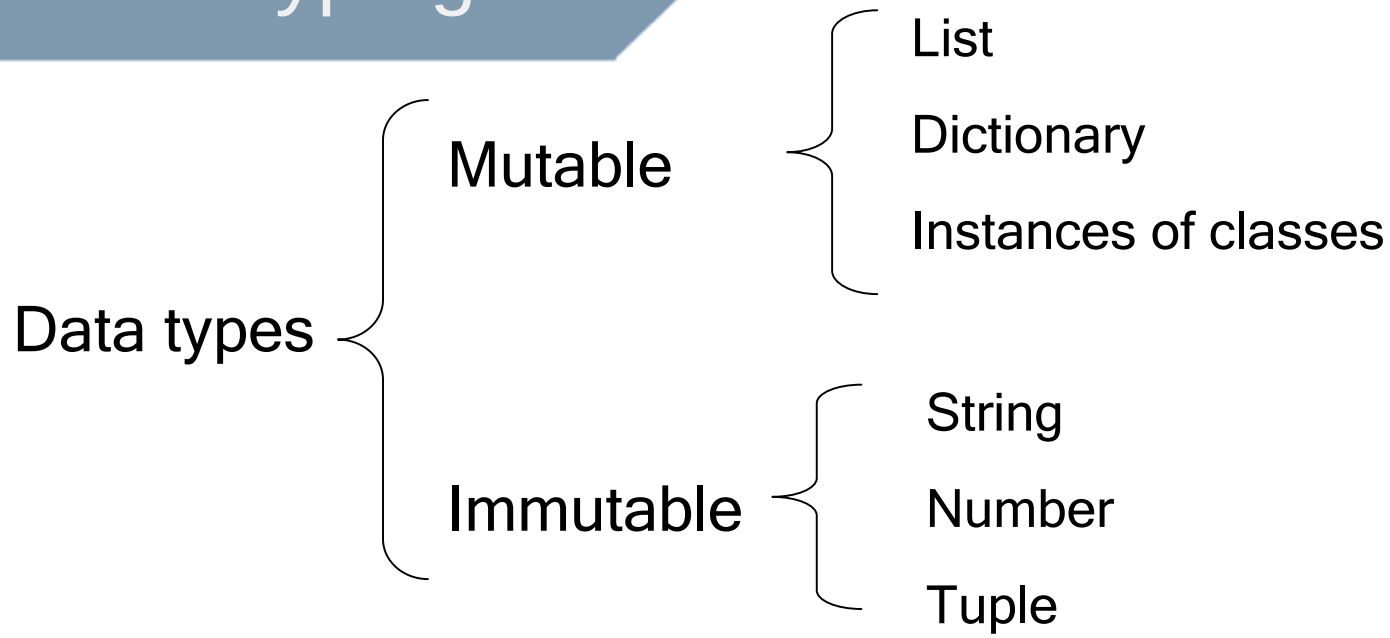
- An interpreted language useful for writing and testing administration scripts.

- A programming language intended for non-technical users.

Principal uses:

- Web programming.
- Software development.
- Building prototypes.

Dynamic typing




```
def sum(a,b):
    if type(a) == int:
            if type(b) == int:
                    return a+b
    if type(a) == str and type(b) == str:
            return a.upper()+ b.upper()
```

python

Data types

Python has a set of built-in types:

Lists and Tuples: a collection of objects which can be referenced by their position number within each object.

```
mylist= [1,44,[3,‘cat'],‘pet']
mytuple= (1, 2, ‘salad’)
```

python

Data types

Dictionary: a collection of associated *key:data* pairs

```
students={123:‘Michael‘, 134:‘Joanna'}
```

(Similar to Perl hashes)

File: disk files as in:

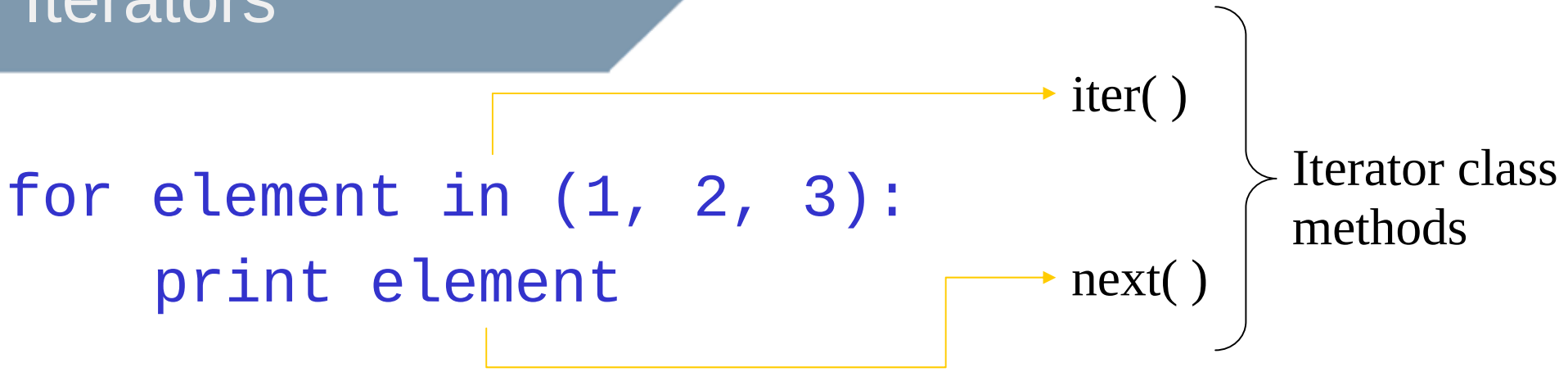
```
file1 = open('data.01','r')
data = file1.read()
```

Objects

```
class Point:
    def __init__(self, x, y):
                self.x = x
                self.y = y

class Paralellpiped:
        pass

>>> cube=Paralellpiped()
>>> cube.length = 10
>>> cube.center = Point(20, 20)
```

python

Iterators




```
>>> l = 'abc‘
>>> it = iter(l)
>>> it.next()
 'a‘
>>> it.next()
 'b‘
>>> it.next()
'c'
```

→ Returns an iterator object

python

Generators

Generators are functions that create and return iterators.

The instruction *yield* specifies the point of return of an intermediate value.

(Much like Ruby)

```
def fib():
  a, b = 0, 1
  while 1:
    yield b
    a, b = b, a+b

def func():
  c=0
  b=fib()
  while c<10:
    print b.next()
    c=c+1
```

python

Lists

List of objects

```
list = [1, 2, 3]
list.append(42.5)
```

stack

```
list.pop()
```

queue

```
list.pop(0)
```

Functional prog.

```
def square(x):
    return x*x

map(square,list)
```

List comprehension

```
[x*x for x in list]
```

python

Scope

Python built in

Global (module)

Local

```
path = '.'

def directories(path):
    a = []; i = []
    for root, dirs,files in os.walk(path):
        a.append([root,dirs,files])
    for list in a:
      for folders in list:
          for files in folders:
              i.append(files.split('.'))
    return i

def types(filelist):
    typelist = []
    for file in filelist:
        if len(file) == 2:
            typelist.append(file[1])
    return typelist

filetypes = Set(types(directories(path)))
```

Productivity

Sorting and seeking for a value within a set

```
sports = [ 'Football', 'Baseball', 'Cricket‘]
sports.sort()
print sports
['Baseball', 'Cricket', 'Football‘]
```

```
>> 'Cricket' in sports
True
```

python

Productivity

Our project in 220 lines!!

Python News Parser

Results for the search of the word 'olympic' in the RSS news feeds.

Economy

Politics

Religion

Sports

Olympic diary **(BBC News)**
Following the Olympic torch on its Everest journey

Major attractions along Olympic torch relay route in Ho Chi Minh city **(www.xinhuanet.com)**

Vietnamese eager to watch Beijing Olympic torch relay **(www.xinhuanet.com)**

Olympic Flame arrives in Vietnam **(www.xinhuanet.com)**

Syrian political, sports elite express support for Beijing Olympics **(www.xinhuanet.com)**

Olympic Flame reaches Mt. Qomolangma **(www.xinhuanet.com)**

Olympic torch arrives in Vietnam **(Indiatimes - Times of India)**
The Beijing Olympic flame arrived in Vietnam's Ho Chi Minh City from North Korea, on one of the final legs of its troubled worldwide journey.

Tibet shadow on Olympics **(Indiatimes - Times of India)**
The Olympic torch was greeted by protesters in Japan on Friday.

World

Tibet shadow on Olympics **(Indiatimes - Times of India)**
The Olympic torch was greeted by protesters in Japan on Friday.

python

In short…

- Python is a highly productive scripting language.
- Easy to learn, easy to read.
- It has “batteries included”.
- Useful for writing working prototypes.

python

Perl

in a Clamshell

Jordan Marshall & Matthew Ghigliotti

An Example...

```
open(FILE, shift @ARGV) ||
  die "I couldn't open the file.\n";

LOOP:
while ($line = <FILE>) {
  $line = lc $line;
  foreach $word (@ARGV) {
    $word = lc $word;
    if ($line =~ /(\W|\A)$word(\W|\Z)/) {
      print $line;
      next LOOP;
    }
  }
}
```

History of Perl

- Created by Larry Wall; first release in 1987
- Originally for text manipulation
- Created in response to
  - ↑ costs of programmers
  - ↓ costs of hardware
  - Superior compiler software
- Perl 5 released in 1994 – still the major version number
  - Introduced modules; inspired Comprehensive Perl Archive Network (CPAN) in 1995

Implementation

- Core interpreter written in C
- De facto standard
- Garbage collection scheme
- Interpreting and compiling operations interleaved
- Specialized tools to parse Perl
- Ported to most OSes
- Difficult to maintain Perl's source code

Language Features

- Objects
- Types
- Garbage collection
- Dynamic type checking
- Most of the advanced structures discussed
- Extensible through modules
- Ties closely to the UNIX shell and C

The Perl Philosophy

“More than one way to do something in Perl.”

- Imperative

```
$/ = ";";

while (<>) {
  /\A\s*(.*?)\s*\z/;
  print $1, "\n" if $1;
}
```

Paradigms (Functional)

```
sub quicksort {
  !@_ ? () : (quicksort(grep {$_ < $_[0]}
                        @_[1..$#_]),
                      $_[0],
                      quicksort(grep {$_ >= $_[0]}
                        @_[1..$#_]));
}
```

Paradigms (Logic)

```
use AI::Prolog;

my $input = <<'PROLOG';
   color(tree, green).
   color(sun, yellow).
   plant(X) :- color(X, green).
PROLOG

my $pro_inst = AI::Prolog->new($input);
$pro_inst->query("plant(sun).");

while (my $results = $pro_inst->results) {
   print $results->[2], "\n";
}
```

Type Conversion

- Operations are evaluated according to context
  - Depends on the operation and arguments

```
$string = "jordan";
@array = ("great","awesome","charming");
$string = $string.@array;
@array = $string;

# $string has the value 'jordan3'
# and @array has one element, $string.
```

Objects and Inheritance

```
package Thingy;
sub new {
  my $class = shift;
  my $self  = {};
  $self->{NAME} = undef;
  bless ($self, $class);
  return $self;
}
sub name {
  my $self = shift;
  if (@_) {$self->{NAME} = shift}
  return $self->{NAME};
}
1;
```

Objects and Inheritance

- Objects are equivalent to anonymous hashes

```
package Doodad;
use Thingy;
@ISA = ("Thingy");
1;

package Widget;
use Thingy;
use Doodad;
@ISA = ("Thingy", "Doodad");
1;
```

Text Processing

- Regular expressions built in
- Implicit global variables

```
while (<>) {
        s/<.+?>//g;
        print;
}

$/ = ";";
while (<>) {
        /\A\s*(.*?)\s*\z/;
        print $1, "\n" if $1;
}
```

Summary

- Built for ease of use
- Scripting and processing in conjunction with the UNIX shell
- Very powerful and capable
- Multiple programming paradigms
- Not a large learning curve from similar languages

I C  N

CSC 520

Lopamudra Sarangi

Parag Sarfare

Introduction

- Product of University of Arizona, Ralph Griswold et. al, influenced by SNOBOL4 and SL5
  - High level general-purpose programming language
  - Paradigm : Procedural
  - Compiled / Interpreted
- Language extensions
  - Graphics features, Unicon(+OOP), Jcon(Java based), MT Icon- (multitasking)
- Applications :
  - Scripting/String processing – compilers(LUCA), databases, word processors.
  - Research – AI, Natural language processing,
  - Expert systems, Rapid Prototyping
  - Graphic displays

The Answer is 42

```
procedure main()
        local a
        a := [3.14,42,1.2,1]
        s_sort(a)
        write("Ans: ",a[*a])
end

procedure s_sort(a)
        local i; local j
        every i := 1 to *a-1 do
           every  j := i+1 to *a do
             if a[j] < a[i] then a[i] :=: a[j]
end
```

- local variable passed-by-reference
- every creates generators
- a is a list
- *a gives the size of the structure
- :=: swap operator!

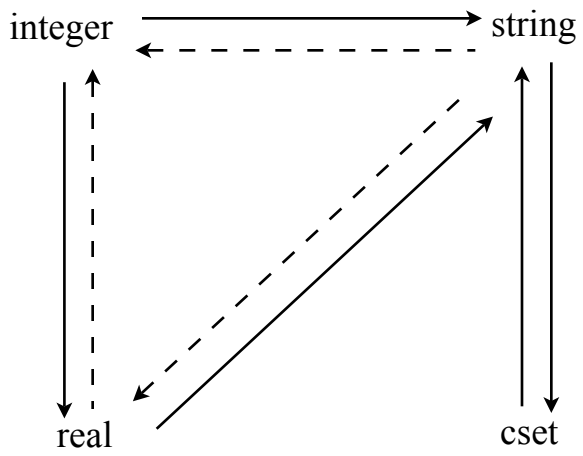
Procedures

```
procedure main()
        every write(Invert(-3, 3))
        every write(Invert(-5))
end

procedure Invert(i, j)
          /j := i
          while i <= j do {
                    suspend -i
                    i +:= 1
               }
          fail
end
```

```
3 2 1 0 -1 -2 -3 5
```

- No nesting
- Returns with
  - success giving a valid result
  - fail returning null value or
  - suspend saving the dynamic state which can be resumed
- Parameter Passing
  - Variables passed-by-value
  - If object mutable(eg. List, Table) then passed-by-reference
  - initial, &null, default parameters

Typing & Scope

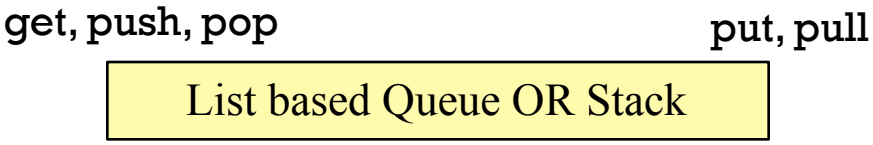


- Data Types
  - Objects have types, variables do not
  - Weak dynamic typing
  - Built-in types
    - numerics (integers,reals), csets, strings, sets, lists, tables, records and procedures
  - Type conversion
    - implicit - operation dependent
    - explicit - using (conditional) conversion routines
- Variables & Scope
  - Lexical scoping
  - local, static, global scope declarations
  - record, link, procedure

Structures

```
record imdb(title,rating)

list1 := [1,"Two",[3,4]]
r1 := imdb("Matrix",8)
r2 := imdb(21,6.6)

tbl := table(0)
tbl[1] := list1
tbl["2"] := r1
```

- Lists - queue/stack
  - ===, |||, ~===, |||:=
- Tables (Dictionaries)
  - key, value pairs
- Sets - unique elements
  - union ++, intersection **, difference --
- Records
  - declared global
  - fields can hold any type



Generators

```
str := "Python is like Icon"
# produces 5 and 18
every write(find("on", str)

list1 := [“i”,”c”,”o”,”n”]
writes(every(!list1))
#prints icon

i := 0; j := 1; k :=2
# generates 1 and 2
every write(i < j | i < k)

# generates 0 to 4
every write(seq(i) \ 5)
```

- Iterate through outcomes of expressions which produce multiple values
- every is used to create expression context for multiple values
- procedures can also act as generators using suspend

Expressions

```
4> 3 #success, result= 3
1=0  #failure

if find("on", str) > 10 then
write("Bingo!")

max := max < x
write("y=",(x|5) > y)

line := read() & write(line);
#implicit error handling

co-expr_name := create expr
@co-expr_name
```

- Success-Failure semantics
- Goal directed evaluation
  - failure drives control
  - automatic searching among alternatives
- Control Backtracking
  - limited to expr
  - `every e1 do e2` `drives' e1
- Co-expressions
  - capture state of an expr
  - suspend/resume an expr
  - transfer control, co-routines

String Operations

```
1  2  3  4
 i  c  o  n
-4 -3 -2 -1  0
```

```
s ? expr
#s-subject, expr-exprn/operator

line ? while tab(upto(&letters))
do write(tab(many(&letters)))
#prints all letters in line

s:=map("fg/ij/cd","cdefghij",&date)
```

- Strings - first class objects
  - cursor, subscripting
- String scanning
  - tab(),upto(),many()
- Editing & conversion
  - image(s) , left(s,i), map (),trim(),replace(s1,s2,s3)
- Character sets
  - &lcase,&letters,&digits
- Regex

More features...

- Polymorphism
  - polymorphic operations
  - operation depends upon the operand types
  - 1.s := “ad”

    s[3:3] := “a” # concatenation followed by assignment, s is “ada”

    write(s[1:3]) # substring operation

    2.write(*str) # gives the size of the object, be it a string, list or table
- Preprocessing
  - $define PI 3.14 (macro expansion - done before syntactic and semantic analysis)
  - $include “mathlib.icn”

Storage Management

- Data allocated in separate regions
- Static, Non-static blocks & strings
- Co-expressions in static blocks
- Garbage collection: marking & compaction
- Predictive Need

  + assures adequate amount of space before allocation

  + garbage collector can be invoked at a safer time

  - maximum amount of storage must be determined

  - predictive need requests prior to allocation

Summary

- Missing features
  - Modules with distinct namespaces
  - Object oriented programming
    - Unicon adds OO features
  - Exception Handling
  - Concurrency
    - MT Icon
- Comparison with other scripting languages
    - Python (heavily influenced), Perl, Ruby, gawk
- Overall
    - Good for scripting, advanced features
    - Too many operators, readability issues

GAWK

Amit Wadhwa

Nithya Chandrasekaran

Introduction

History

- Aho, Weinberger, Kernighan 1977
- GNU AWK – 1986
  - Paul Rubin & Jay Fenlason
  - Arnold Robbins
- NAWK
  - Kernighan

Text Processing and Pattern Matching

Formatted reporting

One to Thousand line programs

Walk, Talk & GAWK

General Users

- As a power tool on command line
- Excellent for work on databases and tabular data

Bigger things

- Networks
- AI
- XML
- Bio Informatics

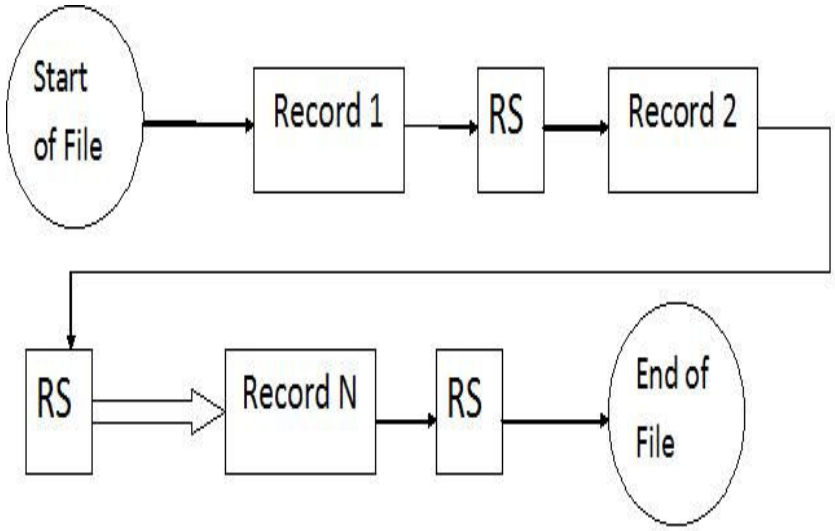
Getting Started

Records and Fields

- File = records + fields
  - Each line is a record
  - Fields are delimited by a special character
    - Whitespace
    - Change with
      "–F" (command line) or
      FS (special varaible)

Generic GAWK Command

- /pattern/{ action}
- BEGIN and END Blocks



Getting Started …

Example 1

- **Command:**
  - ps aux|gawk '$11 == "ssh"
  - {print $2}’
- **Output :**
  - List all pids running ssh

Example 2

- **Command:**
  - awk 'NF > 0' file
- **Output :** Print every line that has at least one field

Two Ways to run Gawk

From the Command line

- cat file | gawk ‘(pattern){action}’
- cat file | gawk -f program.awk

From a script

```
#!/usr/bin/gawk –f
# This is a comment
 (pattern) {action}
```

Interesting Aspects

- Data Driven Paradigm
- *nix to Windows
- POSIX compliant
- Can access standard UNIX stream
- Includes TCP/IP networking functions
- No objects, Not functional; No built-in logic programming!

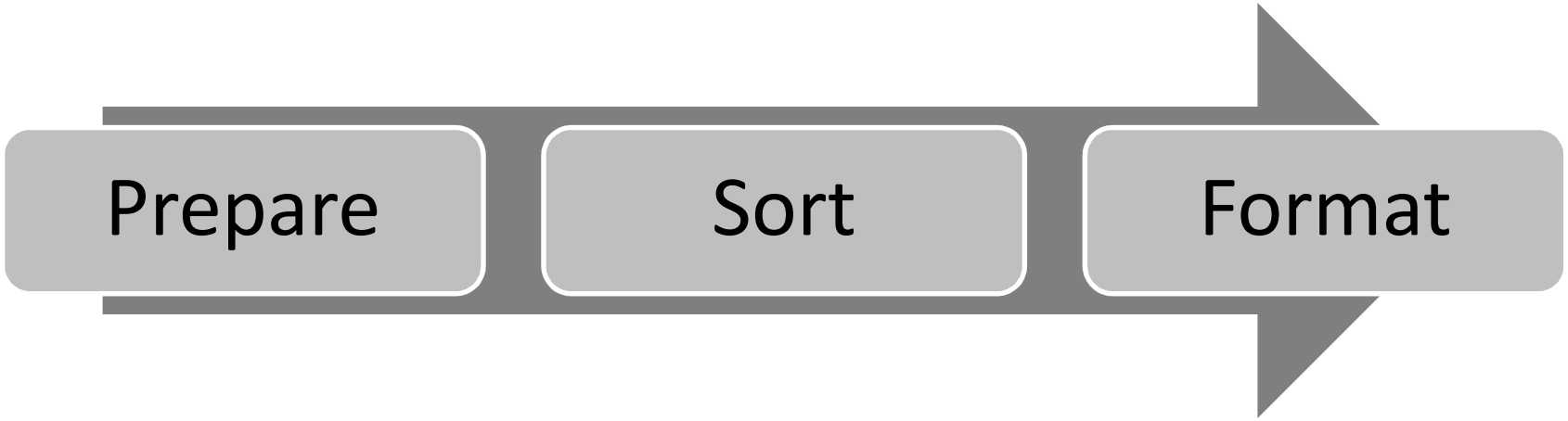
- Asks the machine about the current time

```
BEGIN
    "/inet/tcp/0/localhost/daytime"
    |& getline
     print $0
    close("/inet/tcp/0/localhost/dayti
    me")
       }
```

Aspects…

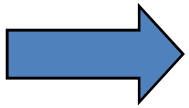
- Interpreted
- No predefined memory limits
- Associative Arrays
- Lint Checking
- Automatic initialization
- Implicit Coercion
- No pointers ☺

Program



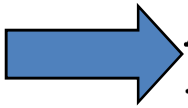

```
A:1:1.0
A:2:1.2
B:1:4.0
B:2:5.0
```

grep output



```
Name:1:2
A:1.0:1.2
B:4.0:5.0
```

Array of numbers



```
Name |   1 |   2
A    | 1.0 | 1.2
B    | 4.0 | 5.0
```

Human readable

Vars and Arrays

Variables

- Predefined variables (NF,NR,SUBSEP…)
- No need for declaration
- Implicitly set to 0 AND Empty String
- Implicit coercion

Arrays

- Dynamic
- Associative
- String indexing
- Built in Array Iterator “in”

One type

- X=x+1
- X=“ppl”

Array Assignment

- A[1] = “foo”;
- B[“foo”] = “bar”;

Iterator

- For (x in myarray) {

More about Arrays

The arrays in gawk can be used to implement almost any data structure

Set:	• myset[“1”]=1; myset[“a”]=1; • If (“x” in myset)
Multi-dimensional array:	• myarray[1,3] = “a”; myarray[1,”ppl”] = 5;
List:	• mylist[1,”value”]=2; mylist[1,”next”] = 3;

Concepts

Regular Expressions

Case Sensitive Matching

- x = "aB"
  if (x ~ /ab/) ... # this test will fail

This one will ignore case

- IGNORECASE = 1
  if (x ~ /ab/) ... # now it will succeed

Conditional flow

Dynamic Expressions

- BEGIN
  { regexp= "[A-Za-z_][A-Z0-9]+" }
  $0 ~ regexp { print }

Function calls and recursion

Summary

Text Processing

- Regular Expressions

Reporting

- Prepare, Sort, Format

TCP/IP and String functions available

Simple and smart variables/arrays

- No types, Associative, Iterator

Niche

- No objects, logic programming, functional concepts

References

- GAWK manual
  http://www.cs.utah.edu/dept/old/texinfo/gawk/gawk_toc.html
- Introduction to GAWK
  http://www.linuxjournal.com/article/1156
- Getting started with GAWK
  https://www6.software.ibm.com/developerworks/education/au-gawk/index.html
- Sams Red Hat 6 unleashed

CS520 – Javascript

"The Worlds most misunderstood **Programming** Language" - Douglas Crockford

Presented By:
Pallavi Chilappagari
Natasha Gaitonde

Road Map

- Features and Applications
- Functions
- Objects
- Prototype Based Inheritance
- Closures
- Garbage Collection
- Conclusion

Features

- Light weight, interpreted scripting
- Weakly typed
- Dynamically Typed
- First class Functions
- Object Oriented
- Javascript Engines

Applications

- Validating user input
- Interactive, dynamic content
- Client environment customization
- Cookie interaction

HelloWorld.html

```
<script type="text/javascript”
  src=“Validate.js"/>
<body onload=“welcome()”>
```

Validate.js

```
function welcome(){
  var txt="Hello World!";
  document.title=txt;
  setTimeout('startTime()',500);

  var x= new Array(2);
  x[0]="Eric";x[1]=“Stan"; x[2]=“Kyle";
  document.getElementById('listtxt').val
  ue=x.sort();}
```

Functions

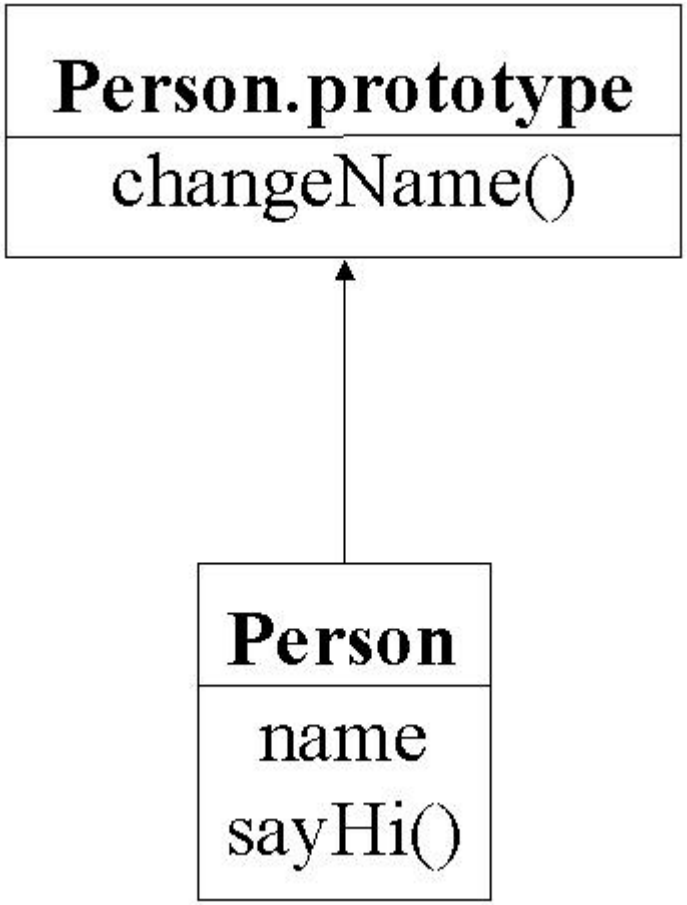
- Functions are Objects
- Higher Order Functions

```
Function myMap(f, arr){
    for (x in arr){
        arr[x] = f(arr[x]);}}
```

- Custom class Constructors and Methods
- Event handling using Callbacks

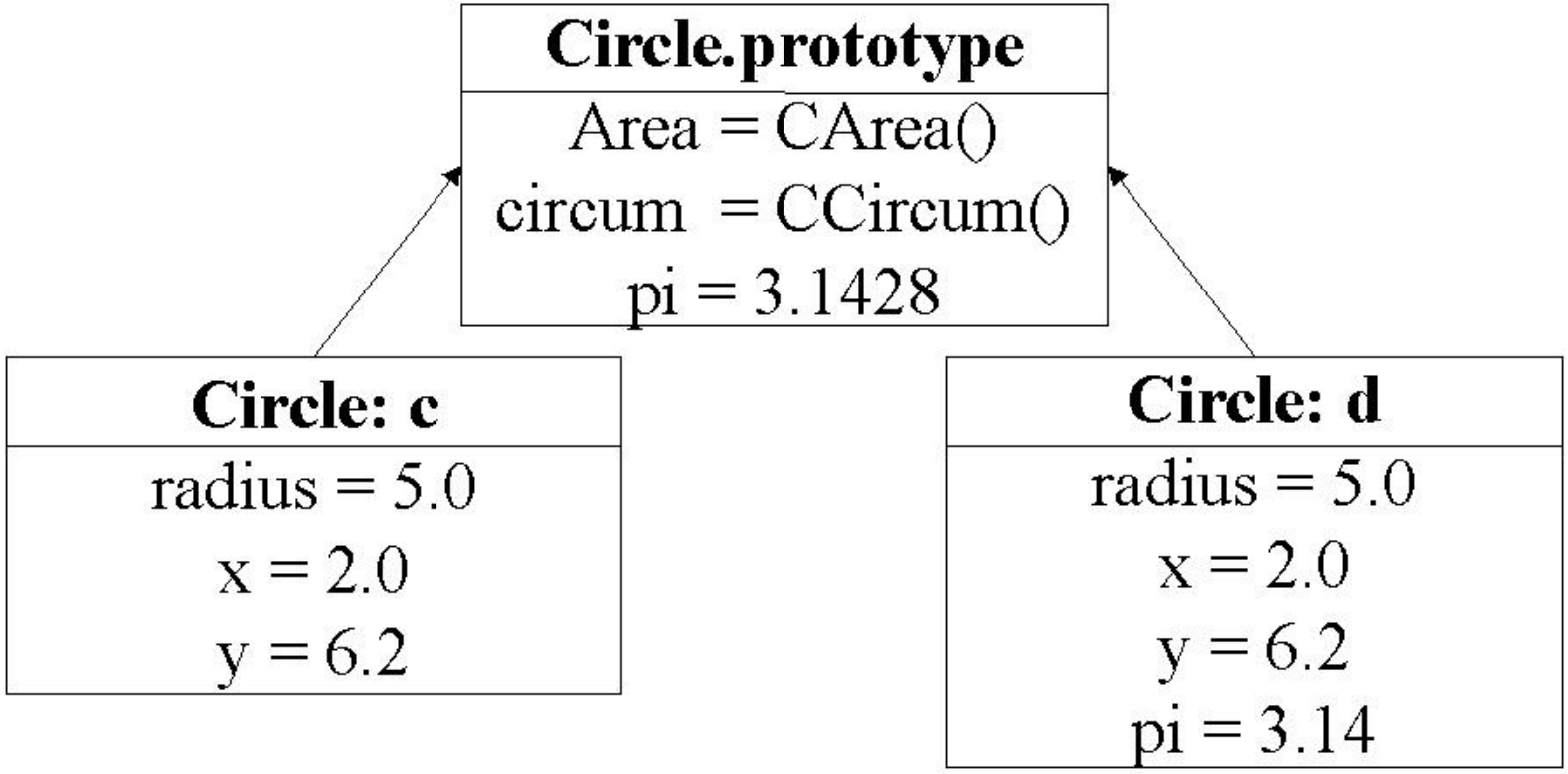
Custom Objects

```
function Person(name){
    this.name = name;
    this.sayHi = function(){
    alert( this.name + " says
hi");}}
x = new Person("Jerry");
x.sayHi();

Person.prototype.changeNam
= function(name) { this.nam
= name; }

y = new Person("Ben");
y.changeName("Tom");
y.sayHi();
```



Prototype Based Inheritance

- Classical Inheritance
- Runtime alteration of Prototypes



Closures

- *A closure is a special kind of object that combines two things: a function, and the environment in which that function was created “*

- Can be used to emulate private methods in JavaScript

- Used in ‘trimbreakpoint’, a debugging utility for JavaScript, to inspect variables during execution

Emulating private methods with closures

```
var Counter = (function() {
    var privateCounter = 0;

    function changeBy(val) { privateCounter
    += val; }

    return { increment: function()
    { changeBy(1); },

            value: function(){return
    privateCounter; }})();
alert(Counter.value()); /* Alerts 0 */
Counter.increment();
```

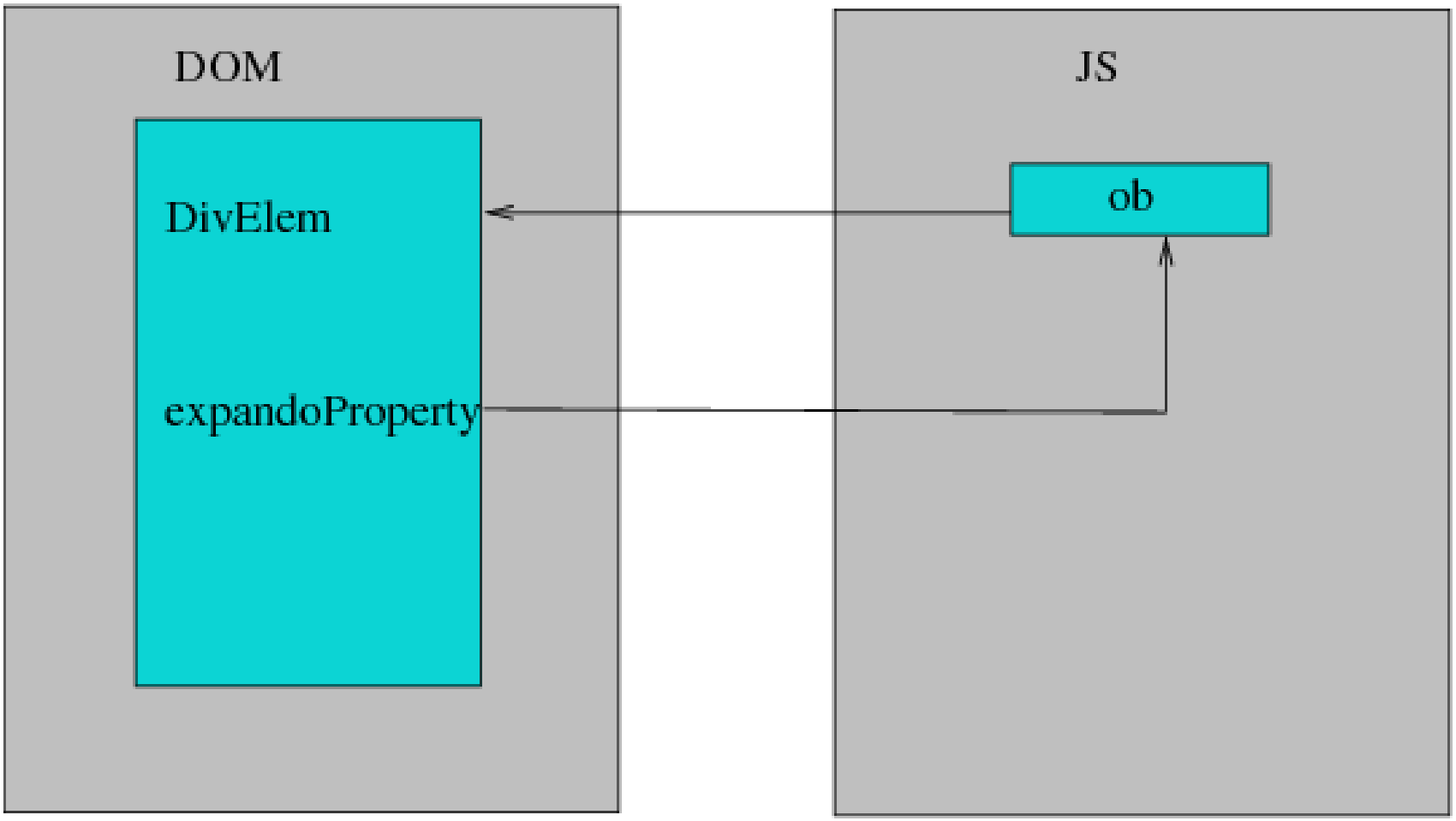
Garbage Collection

- JavaScript implements a variant of mark-and-sweep algorithm
- Browsers use reference counting to handle the memory allocated for DOM objects
- Common to have cyclic references between the JavaScript variables and DOM objects
- Cyclic references and closures can result in memory leaks

Circular Reference Example

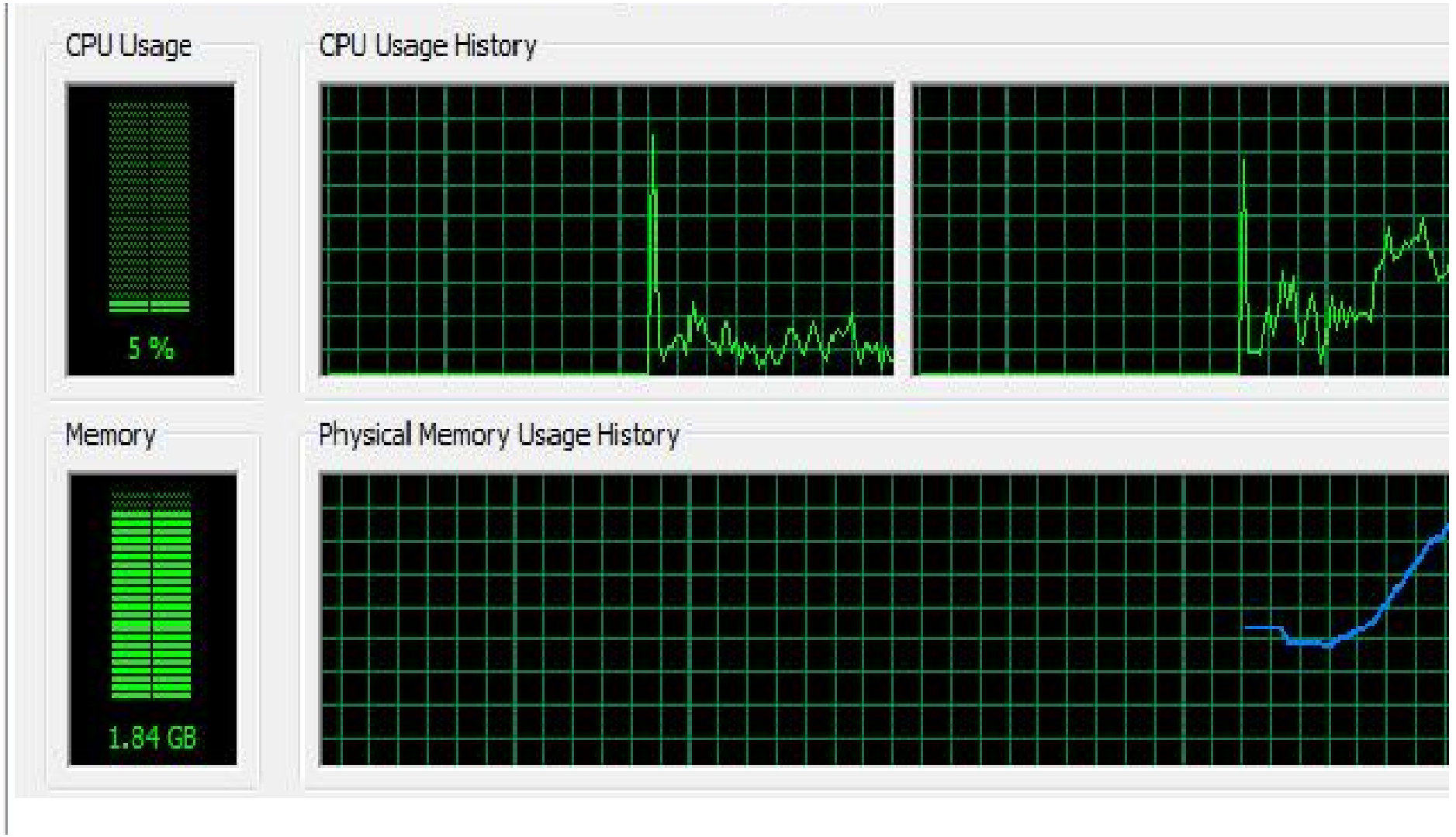
```
<script type="text/javascript">
  var ob;
  window.onload = function(){
  ob=document.getElementById("DivElem
  ");
  document.getElementById(“DivElem”)
      .expandoProperty=ob;
  ob.bigString=new Array(1000).join(new
  Array(2000));};
</script>
```

Circular Reference illustrated



Memory leak due to Closures

```
<script type="text/javascript">
    function LeakMemory(){
    var parentDiv =
  document.createElement("div");
    parentDiv.onclick=function(){ foo(); };
    parentDiv.bigString =  new Array(1000)
      .join(new Array(2000).join("XXXXX"));}
</script>
<input type="button" value="Memory
  Leaking Insert" onclick="LeakMemory()"/>
```


CPU Usage
5 %
CPU Usage History
Memory
1.84 GB
Physical Memory Usage History

Conclusion

Please refer to our paper for more on

- Scope
- Generators, Iterators and Array Comprehensions
- Exception Handling
- Closures

Thank you

TCL/TK

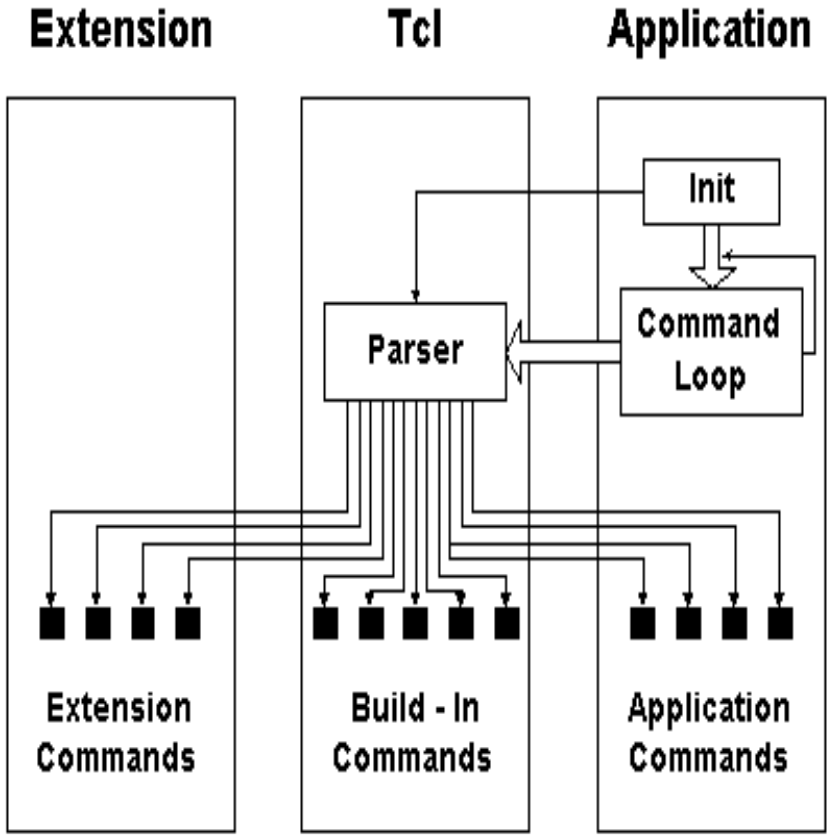
CS520 Final presentation

- Balaji Ramamurthy
- Xuchen Wang

Introduction

- **Tcl** is the abbreviation of "Tool Command Language".It is a scripting language created by John Ousterhout. Originally "born out of frustration"

- **Tk** is one of the most important Tcl extensions, written in C,for rapid prototyping of GUI interfaces for X windows system.



Overview

1) Tcl as general-purpose scripting language
   - special features

2) Tcl `s Embeddability &Extensibility
   - i.e :Integrated with C

3) Advanced topic
   - FP,OOP,GC in TCL

Tcl as scripting language

- As a **scripting language**, Tcl is similar to other UNIX shell language such as the C shell. Here is what a typical Tcl program looks like:

- proc add {a b} {
  puts stdout $x + $y = [expr $x + $y];
  }

---

same effect as above

- puts stdout " $x + $y = [expr $x + $y] "

Regular Expression

- The most powerful way to express patterns is with regular expressions. Here is an example:

---

set x http://www.arizona.edu;
regexp {^[^:]+(?=.*\.edu$)} $x match;

---

match => http

Scope

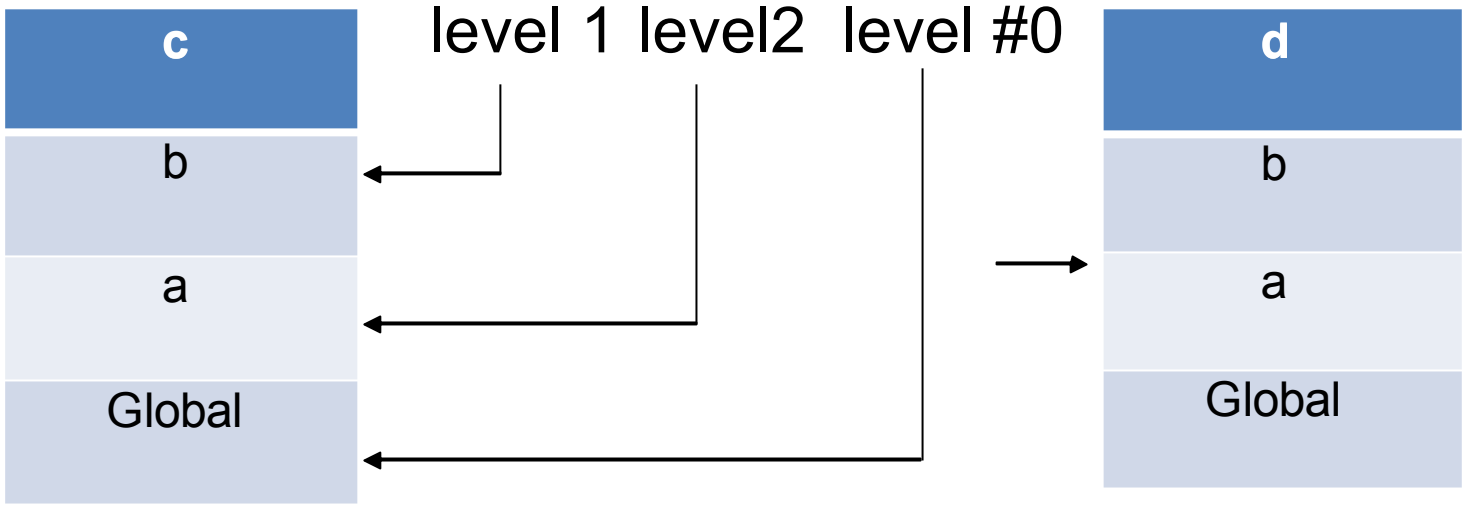
- **Scope:**
- Variables defined outside a procedure are not visible to the procedure,unless upvar or **global** scope commands are used.

```
proc ObjInit { o x y } {
        global obj
        set obj($o,x) $x
        set obj($o,y) $y
        set obj($o,dist) [expr sqrt($x* $x + $y * $y)]
}
proc ObjMove { o dx dy } {
        global obj
        incr obj($o,x) $dx
        incr obj($o,y) $dy
        set obj($o,dist) [expr sqrt($obj($o,x)* $obj($o,x) + \
        $obj($o,y) * $obj($o,y))]
}
```

Dynamic Scope

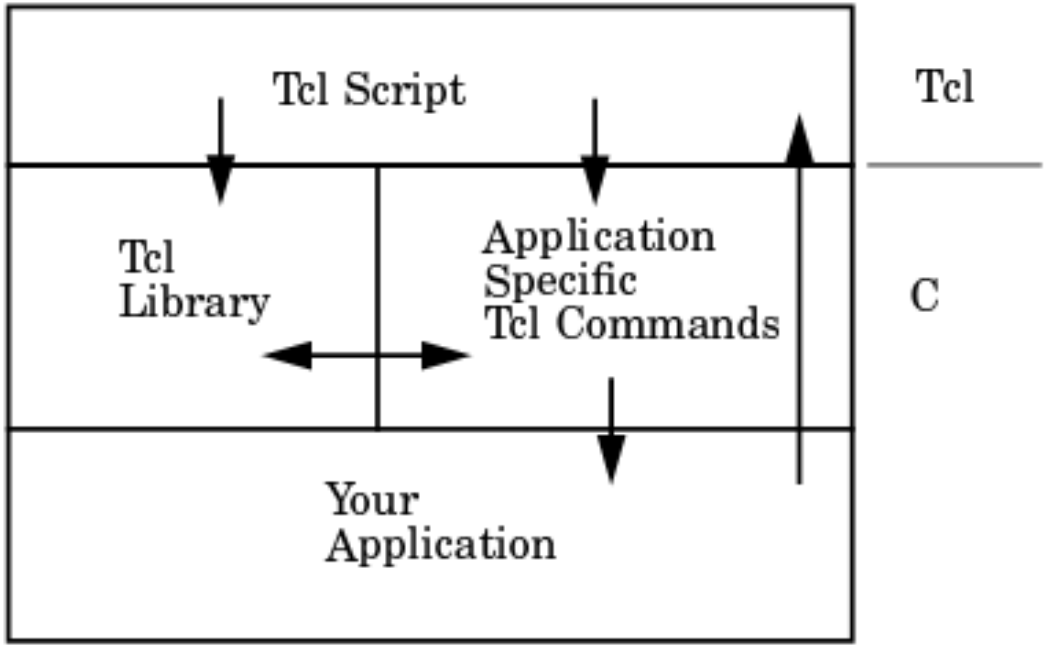
- **upvar && uplevel command**

  The Tcl upvar and uplevel commands allow a procedure to modify the local variables of any procedure on the call stack. i.e:



Embeddability and Extensibility

- Tcl is designed to be easily extensible by writing new command implementations in C.



Functional programming

- **Functional programming in Tcl**

  Tcl is often used imperatively, but at bare-bones it's fully functional --just by the fact that every command returns a result.

```
# used imperatively
proc readfile name {
        set fp [open $name]
        set data [read $fp]
        close $fp
        return $data
}
```

```
# used functionally
proc readfile name {[lambda fp {K [read $fp] [close $fp]}] [open $name]}
proc K {a b} {set a}
```

OOP

- **OOP in Tcl**
  - There was no build-in OOP support beforeTcl8.5
  - In Tcl 8.5, XOTcl extension is added to the Tcl core.

```
class Stack {              # "tcl++" or "smallTcl"
        variable s {}
        method push args {eval lappend s $args}
        method pop {} {
                if ![llength $s] {error "stack underflow"}
                K [lindex $s end] [set s [lrange $s 0 end-1]]
        }
}
```

Garbage Collection

- **Garbage collection**

  1) Tcl does not need memory managment when it is used in a "Native" way

  2) GC is absolutely needed to work with OO.

"An object or two may sometimes be nice, just like a glass of beer. But one shouldn't start drinking at breakfast."

-John Ousterhout

Summary

1) Tcl as general-purpose scripting language

- string substitution
- regular expression
- scope
- upvar&&uplevel commands

2) Tcl `s Embeddability &Extensibility

- integration Tcl with C

3) Advanced topic

- functional programming
- OOP
- GC