Skriptspråk

F15

Skriptspråk

- Skript = manus, en konversation med datorn
- Skriptspråk (ofta, något felaktigt, synonymt med dynamiska språk) är en klass av programmeringsspråk som blir allt viktigare
  - Webben (Perl, Python, Ruby, JavaScript, ActionScript)
  - Scientific computing (Python)
  - Textmanipulering (Perl)
  - Byggskript, skalskript (Bash, Perl, Python, Ruby)
- Några karaktäristika
  - Kort utvecklingstid till priset av lång exekveringstid
  - Otypade (eller dynamiskt typade)
  - Intepreterade
  - Hög nivå och hög grad av föränderlighet (t.ex. strängevaluering, byta ut funktioner under körning, etc.)
  - Domänspecifika (t.ex. `Make`)
- Blir vanligare att bygga stora (delar) av system med skriptspråk
  - Ex. Spotify (Python), PPM (Perl), Amazon (Python)

Skriptspråk i traditionell utveckling

Används för olika typer av automatisering

- Byggprocessen
- Testning
- Kodgenerering
- Olika hjälpprocesser, t.ex. vid versionshantering

På denna kurs använder lärarlaget följande skript:

- Bashskript för publicering av webbsidan efter `hg push`
- Bashskript för epostnotifikationer vid incheckningar
- Pythonskript för RSS-flöde, nyheter och schemat
- Rubyskript för att generera innehållet i det publika `hg`-repot
- 30+ makefiler för kompilering av LaTeX-, C- och Java-kod

Fotnot: Finsk studie visar att programmerare anser att skriptspråk är mer användbart än diskret matematik, funktionell programmering, XML och databaser

Från `publish.rb` som publicerar kursens `hg`-repo (Ruby)

```ruby
if ARGV.size > 0
  result = ARGV.collect do |filter|
    if filter == "upload"
      upload = true; nil
    else
      makefiles.collect { |entry| entry if entry.end_with?(filter+"/Makefile") }
    end
  end
  makefiles = result.flatten.compact
end

for f in makefiles do
  $stderr.print "Processing #{f} ... "
  makefile = Pathname.new(f)
  directory = makefile.dirname
  temp = `pushd .&& cd #{directory}&& make publish&& popd;`
  unless $?.to_i == 0
    logname = directory.basename.to_s + ".log"
    puts "Error! (Log written to #{logname})"
    File::open(logname, "w") { |log| log << temp }
  else
    puts "OK"
  end
end
```

Några exempel*

Språk	Domän	Nyckelabstraktioner
sh, csh, ...	*nix	pipes, omdirigering, text
AWK	radorienterad text	strängar, regulära uttryck
Make	applikationsutv.	Mål, beroenden
Applescript	Macprogram	applikationskataloger
Javascript	webb (klientsida)	DOM
UnrealScript	3D-spel	aktörer, ljus
ActionScript	Flash	bilder, filmer, ljud, tid
PHP	webb (serversida)	HTML
Groovy	Java	Javaobjekt, listor, mappar
Perl, Python, Ruby	generella	objekt, listor, mappar

(* Via Nate Nystrom)

Mikroexempel 1*

```
#include <stdio.h>
int main(void) {
  puts("Hello, world");
  return 0;
}
```

```
object Hello extends Application {
  Console.println("Hello, world");
}
```

```
puts "Hello, world"
```

```
print "Hello, world"
```

```
print "Hello, world\n";
```

```
<?php
print "Hello world\n";
?>
```

```
println "Hello, world"
```

(* Via Nate Nystrom)

Mikroexempel 1*

```
#include <stdio.h>                              // C
int main(void) {
  puts("Hello, world");
  return 0;
}

object Hello extends Application {              // Scala
  Console.println("Hello, world");
}

puts "Hello, world"                             // Ruby

print "Hello, world"                            // Python

print "Hello, world\n";                         // Perl

<?php
print "Hello world\n";                          // PHP
?>

println "Hello, world"                          // Groovy
```

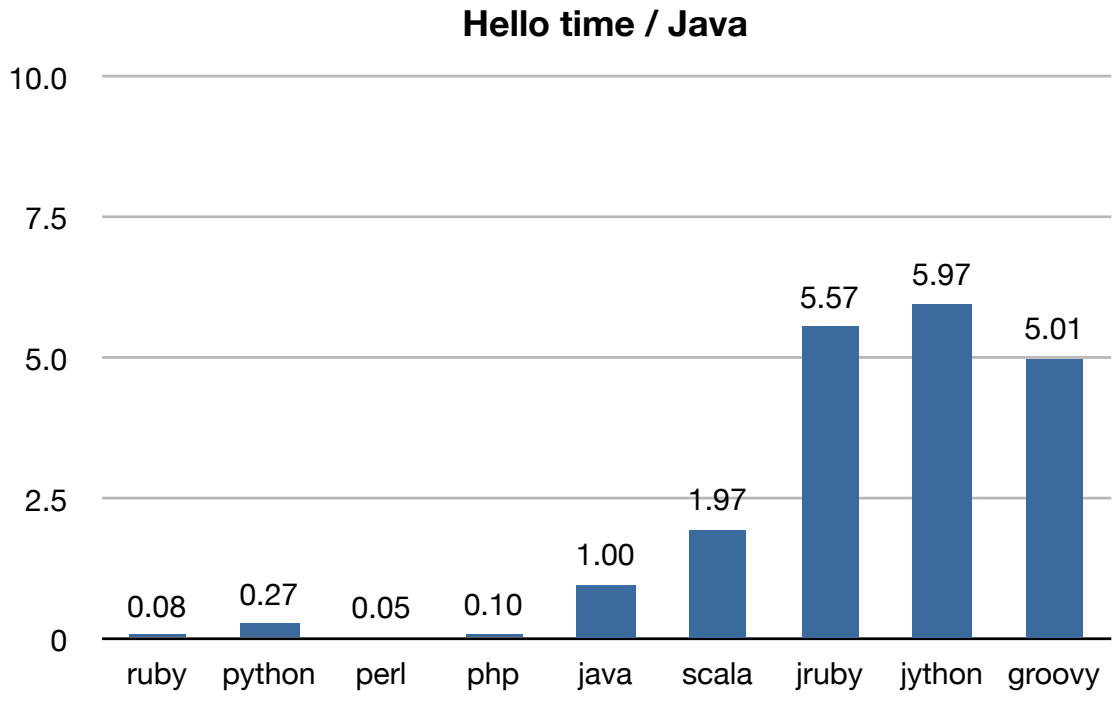
(* Via Nate Nystrom)

Mikroexempel 2*

```
#include <unordered_map>
unordered_map<string,int> *m = new unordered_map<string,int>();
m->put("one", 1);                                   // C++0x

val m = new HashMap[String,Int]();                  // Scala
m += "one" -> 1;

m = {}                                              // Ruby & Python
m["one"] = 1

%m = ();                                            // Perl
$m{"one"} = 1;

$m = array();                                       // PHP
$m["one"] = 1;

def m = [:]                                         // Groovy
m["one"] = 1
```

(* Via Nate Nystrom)

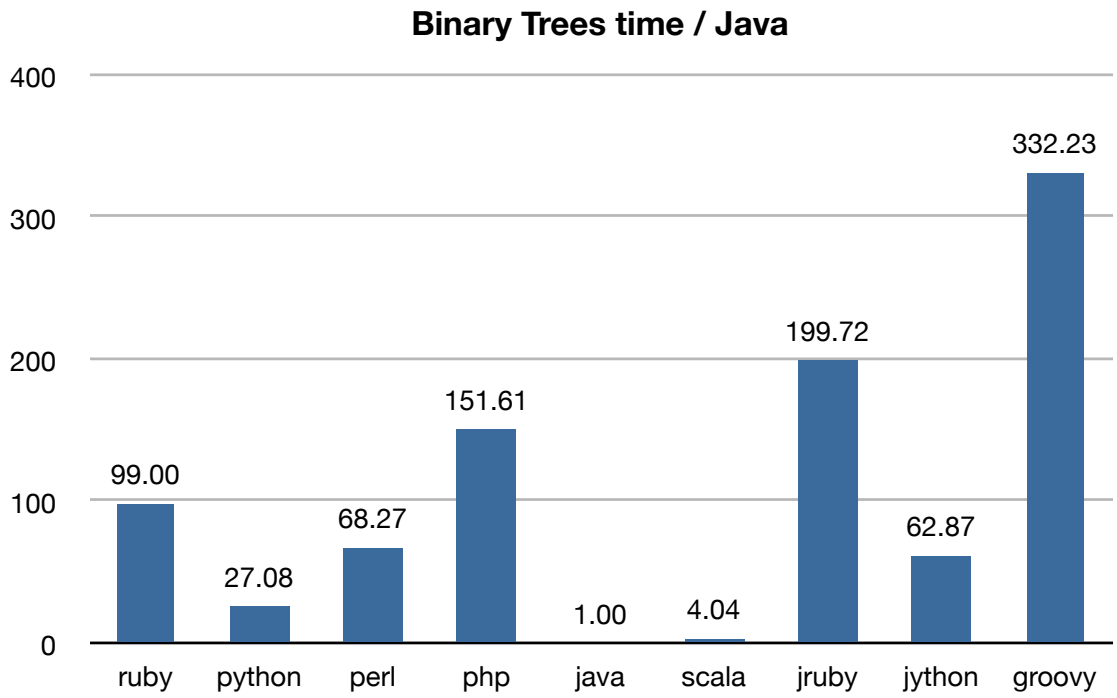
Startup-tid för Hello World*



- C-interpretatorer ca. 4–20x snabbare än Java
- Java-interpretatorer 5-6x långsammare än Java

(* Via Nate Nystrom)

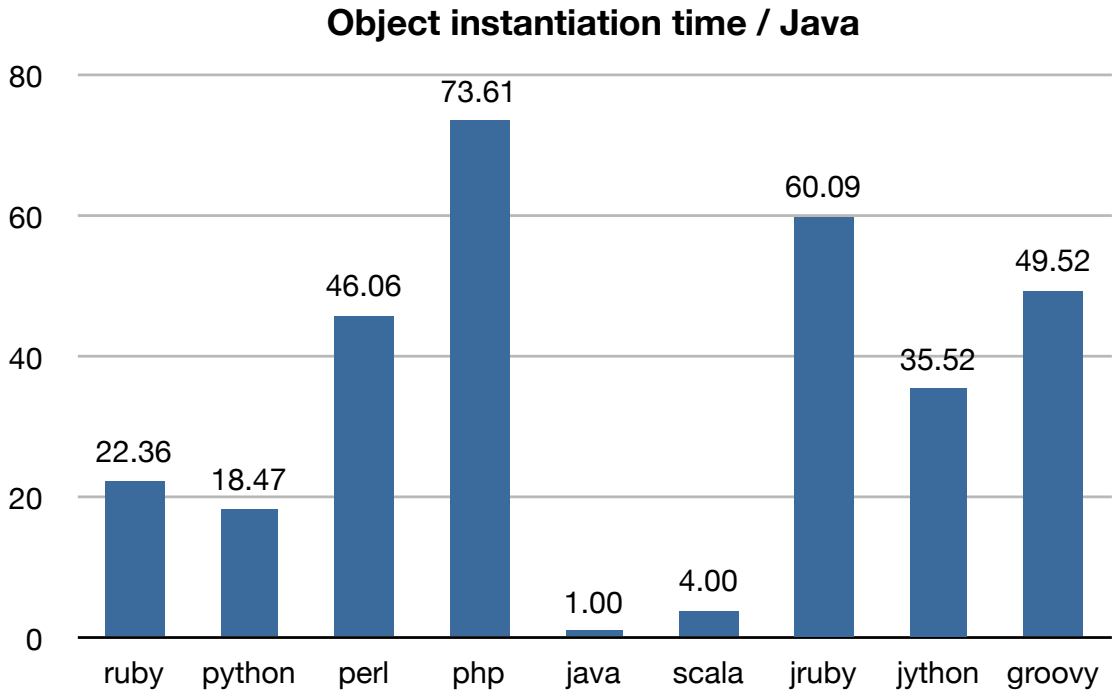
Körtider: skapa och traversera ett binärträd*



- C-interpretatorer ca. 27–152x långsammare än Java
- Java-interpretatorer 63-332x långsammare än Java

(* Via Nate Nystrom)

Minnesallokering och initialisering*



- C-interpretatorer ca. 18–74x långsammare än Java
- Java-interpretatorer 35–60x långsammare än Java

(* Via Nate Nystrom)

Utvecklingstid

- Tidiga experiment av t.ex. John K Ousterhout visar på mellan 4–60 gånger högre utvecklingstid i C/C++ kontra Tcl för motsvarande program, och mellan 2–50 gånger så många rader kod
- Informella experiment med Ruby kontra Java av Bruce Tate ger ca. 60 gånger långsammare utveckling med Java
- Studier av bl.a. Lutz Prechelt för små (<500 LOC) program visar
  - Skript ca 1–5 gånger kortare än motsvarande Java- el. C-program
  - Mediantid på 2–4 timmar för skriptspråk konta 8–11 timmar för Java/C++/C
  - Produktivitet i LOC/h är relativt oberoende av programspråk

- När är det rimligt att ge upp prestanda för kortare utvecklingstid?

Minus:

- Avsaknad av statisk typinformation problem för verktyg och IDE:er
- Koden kan vara svårläst pga få ”inkörsportar” eller deklarationer att luta sig tillbaka på
- Hög grad av föränderlighet kan göra det svårt att debugga eller resonera om koden
- Svårt att optimera

Plus

- Snabb utvecklingtid, enkelt att få något upp och snurra
- Läsbar kod på hög abstraktionsnivå
- Korta program
- Flexibilitet
- Interpretatorer gör det enkelt att testa, debugga och inspektera ett körande program

cap i C (1/3)

```
int main(int argc, char** argv) {
  unsigned int limit;

  switch (argc) {
  case 1:
    limit = DEFAULT_LIMIT;
    break;
  case 2: // Parse cmd line arguments
    if (***++argv == '-' && ***++argv == 'l') {
      limit = parseLimit(++*argv);
    }
    if (limit)
      break;
  default:
    puts("Usage: cap [-l<ns>] where n is an int and s in [bkM]. Examples: \n\n\t"
 "cap -l10M (stops at 10 megabytes)\n\t"
 "cap -l1024b (stops at 1024 bytes)\n\t"
 "cap -l1k (same as above) \n");
    return 2;
  }
```

cap i C (2/3)

```
  if (cp(limit)) {
    return 0;
  } else {
    fprintf(stderr, "Limit reached (%d bytes), capping\n", limit);
    return 1;
  }
}

int cp(int limit) {
  for (int c=getchar(); c != EOF && --limit; c = getchar()) putchar(c);
  return limit;
}

int multiplier(char c) {
  switch (c) {
  case 'M': return 1024 * 1024;
  case 'k': return 1024;
  default : return 1;
  }
}
```

cap i C (3/2)

```
int parseLimit(char* arg) {
  char *c = arg;
  int isNumber = 1;
  int mult = 0;
  while (*c && isNumber) {
    switch (*c) {
    case ' ': ++c; continue; // Skip spaces
    case 'M':
    case 'k':
    case 'b':
      mult = multiplier(*c);
      *c = '\0';
      continue;
    default:
      isNumber = isdigit(*c++);
    }
  }
  return isNumber ? atoi(arg) * mult : 0;
}
```

cap i Python (1/2)

```
#!/usr/bin/env python
from sys import argv, stdin, stdout

if len(argv) < 2:
    print "Usage: cap -l[<size>]"
else:
    arg = argv[1].strip()
    if arg[0:2] == "-l" and arg[-1] in "kbM":
        print cap(limit(int(arg[2:-1]), arg[-1]))
    else:
        print "Could not understand limit argument '" + arg + "'"
```

cap i Python (2/2)

```
def cap(limit):
    read = 0
    while read < limit:
        temp = stdin.read(1)
        read = read + 1
        if len(temp) == 0:
            return "Reaced EOF"
        stdout.write(temp)
    else:
        return "Limit reached, capping"

def limit(prefix, suffix):
    if suffix == 'k':
        prefix = prefix * 1024
    elif suffix == 'M':
        prefix = prefix * 1024 * 1024
    return prefix
```

Programmet `compare` som ni använde i lab 1 (Python)

```
import sys
from os import popen

if len(sys.argv) < 4:
    print "Usage: compare program-name input expected-output"
else:
    program = sys.argv[1]
    input = sys.argv[2]
    expected = sys.argv[3]

    # If no absolute or relative path was given, assume
    # program is in current directory (prepend ./)
    try:
        program.index("/")
    except:
        program = "./" + program

    output = popen(program + " " + input).read().strip()
    print "Testing", program, input, "against", expected, ".......",

    if (output == expected):
        print "OK"
    else:
        print "Error, expected", expected, "got", output
```