

# TDP007 - Konstruktion av datorspråk

Introduktion till Ruby och kursen

Pontus Haglund

Institutionen för datavetenskap

- 1 Introduktion till kursen  
Utvecklarblogg
- 2 Introduktion till Ruby

# Mål med denna föreläsning

Du som student bör efter föreläsningen:

- veta vad målet med kursen är
- veta hur vi kommer arbeta under kursen
- veta hur ni examineras
- hur man kan köra Ruby-kod
- känna till några utvalda detaljer av Ruby

- 1 Introduktion till kursen  
Utvecklarblogg
- 2 Introduktion till Ruby

# Frågor

- Hur konstruerar man ett datorspråk?
- Vad är ett datorspråk?
- Varför skulle man vilja konstruera nya språk?
- Hur vet man om ett språk är bättre än ett annat?



# Konkreta mål

Ska kunna:

- Förklara och använda regex
- Använda verktyg för hantering av uppmärkningsspråk
- Använda och modifiera en tolk för ett enklare programspråk
- Redogöra för och tillämpa grundläggande principer för design av datorspråk



## En kommentar om upplägg och stöd

Kommer ni ihåg att jag pratade om mängden handhållning som finns i kurserna? (TDP001 föreläsning 1).

# Hur betygssätts ni

## Examination

- DAT1 - Datortentamen - U, 3, 4, 5 - 4 hp
- LAB1 - Laboration - U, G - 2 hp

## Betygsskala:

- Fyrgradig skala - U, 3, 4, 5

En kommentar om chat-gpt/co-pilot



# Hur tar du dig i mål

Kursen har olika delar. Till varje del hör följande moment:

- Föreläsningar
- Laborationer
- Inlämning av uppgifter (med **hårda** deadlines)
- Förberedelser inför seminarier
- Seminarier (**obligatorisk** närvaro)

# Föreläsningar

## **Utvärderingar och ändringar i kursen**

# Föreläsningar

## Arbetsgrupper

Checklista att göra (idag)

- Kontrollera att ni finns med i webreg
- Kontrollera att ni får kontakt med er partner
- Anmäl dig till tentamomentet i webreg.

# Laborationer och inlämningar

Programkod, enhetstester och utvecklarblogg

# Utvecklarblogg

- Dokumentera hur ni tänkt när ni arbetade fram lösningen.
- Reflektera kring hur det är att lära sig Ruby
- Reflektera kring de olika tekniker ni stöter på

## Utvecklarblog - Dokumentera arbetet

- Vad arbetade ni med vid varje labbpass?
- Hur har ni tolkat uppgiften?
- Vad var svårt och/eller lätt med uppgiften?

## Utvecklarblogg - Reflektioner runt lärande

- Hur har ni gått tillväga för att lära er Ruby?
- Vilka fel och misstag har ni gjort under tiden?
- Vad finns det för nya konstruktioner i Ruby som ni inte sett förut?
- Vad finns det för konstruktioner som ni känner igen men som ser annorlunda ut?
- Finns det något som ni gillar eller tycker om i Ruby?

## Utvecklarblog - Reflektioner runt tekniker

- Hur användbart verkar detta vara?
- Hur lätt/svårt är det att sätta sig in i?
- Hur har ni hittat informationen?



# Seminarier

Förbered opposition och var beredd att presentera:

- lösningen i grova drag
- tolkning av uppgiften
- användningen av tekniker
- kodstil, tester och utvecklarblogg

Seminarier är det som examinerar **LAB1**

# Material i kursen

- Kursboken
- Kurssidan (läs hela)
- Dokumentation
  - [ruby-lang documentation](#)

# Rails



- 1 Introduktion till kursen  
Utvecklarblogg
- 2 Introduktion till Ruby

# Mjukt

C++



->

ruby



# Hur använder man ruby

- Finns förinstallerat i PUL (2.7)
- Interaktivt **irb**
- Kör en fil **ruby filnamn.rb**

# Skapa script

hello.rb

```
#!/bin/env ruby w  
puts "Hello, world!"
```

shell

```
$ ./hello.rb  
Hello, world!
```

## Objekt - allting är objekt

```
7.class
(3.14).class
(3.14).round

'Pontus'.class
'Pontus'.reverse
'Pontus'.gsub('P', 'L')

["äpple", "banan", "apelsin"].class
["äpple", "banan", "apelsin"].length

1.+2
1+2
```

```
=> Integer
=> Float
=> 3
>>
=> String
=> "sutnoP"
=> "Lontus"
>>
=> Array
=> 3
>>
=> 3
=> 3
```



# Block

```
def foo(a, b, c)
  # do stuff
end
```

```
def max(a, b)
  max = a
  if b > a
    max = b
  end
  max
end

puts max 2, 3
```

```
def max(a, b)
  max = a
  if ( b > a )
    max = b
  end
  return max
end

p max(3, 2)
```

# Angående stil

**Var konsekvent**

# Klasser

```
class Component
  def initialize(name)
    @name = name
  end

  def display()
    "Component: #{@name}"
  end
end
```

```
c = Component.new("c1")
puts c.display
b = Battery.new("b1", 24)
puts b.volt
puts b.display
```

```
class Battery < Component
  def initialize(name, voltage)
    super(name)
    @voltage=voltage
  end

  def display()
    "Battery " + super
  end

  def volt()
    #...
  end
end
```

```
Component: c1
24
Battery Component: b1
```

# Namngivning

```
# Variabler, metoder: i, i_min
# Konstanter       : I, I_MIN
# klasser          : Battery, ResultThing

# Instansvariabler : @i, @i_min
# Klassvariabler   : @@log
# Globala variabler : $glob, $i_min
# Symboler         : :name, :age
```

## Arrayer (lista)

```
>> [3, "little", "pigs"]  
=> [3, "little", "pigs"]  
  
>> [3, "little", "pigs"].join("")  
=> "3littlepigs"  
  
>> a = Array.new  
=> []  
>> a << "append" << "stuff"  
=> ["append", "stuff"]  
  
>> Array.new(2, 5)  
=> [5, 5]
```

# Arrayer documentation

[ruby-lang documentation](#)

## Hashtabeller (dict/map)

```
>> {"Anna" => 5, "Alex" => 4, "Anders" => 3}  
=> {"Anna"=>5, "Alex"=>4, "Anders"=>3}
```

```
>> {"Anna": 5, "Alex": 4, "Anders": 3}  
=> {:Anna=>5, :Alex=>4, :Anders=>3}
```

```
>> {Anna: 5, Alex: 4, Anders: 3}  
=> {:Anna=>5, :Alex=>4, :Anders=>3}
```

```
>> {:Anna => 5, :Alex => 4, :Anders => 3}  
=> {:Anna=>5, :Alex=>4, :Anders=>3}
```

## Block och iteratorer

Metoder kan ta block som argument. Ett block är kod som innehåller den aktuella omgivningen. En iterator är en metod som tar ett block och kör det för varje element. Ungefär som algoritmer och lambdafunktioner tidigare.

```
>> ['i', 'am', 'Don', 'Quixote'].each do | entry | print entry, ' ' end  
i am Don Quixote => ["i", "am", "Don", "Quixote"]  
>>  
>> fac = 1  
=> 1  
>>  
>> 1.upto(5) do | i | fac *= i end  
=> 1  
>>  
>> [1, 2, 3, 4, 5].map { | entry | entry * entry }  
=> [1, 4, 9, 16, 25]  
>>  
>> (0..100).inject(0) { | result, entry | result + entry }  
=> 5050
```



## Två sätt att skriva block

```
[1,2,3,4,5].each do | e | puts e end  
# DO och END när sidoeffekterna är viktiga  
# eller när koden tar flera rader
```

```
[1,2,3,4,5].map { | e | e * e }  
# { och } när koden tar en rad och  
# returvärdet är det viktiga
```

## Skriv en egen iterator

```
def enumerate(list, &block)
  counter = 0
  for x in list do
    block.call(counter, x)
    counter += 1
  end
end

enumerate(["hej", "svej"]) do |x, y|
  puts "index: #{x}, value: #{y}"
end
```

# Villkor

```
if (1 + 1 == 2)
  puts "Like in school."
else
  puts "What a surprise!"
end

puts "Like in school." if (1 + 1 == 2)
puts "Surprising!" unless (1 + 1 == 2)

puts (1 + 1 == 2) ? "Working" : "Defect"

spam_probability = rand(100)
case spam_probability
when 0...10 then puts "Lowest probability"
when 10...50 then puts "Low probability"
when 50...100 then puts "High probability"
end
```

# Upprepning

```
while (i<10)
  i*=2
end

i*=2 while (i<100)

begin
  i*=2
end while (i<100)

i*=2 until (i>=1000)
```

```
loop do
  break i if (i>=4000)
  i*=2
end

4.times do i*=2 end

r=[]
for i in 0..7
  next if i\%2==0
  r<<i
end

(0..7).select { |i| i\%2!=0 }
```

## Skapa klasser

```
class Cell
  def initialize
    @state = :empty
  end
end

class Board
  def initialize(width, height)
    @width = width; @height = height
    @cells = Array.new(height) {
      Array.new(width) { Cell.new }
    }
  end
end
```

# Åtkomst

```
class Cell
  def state
    @state
  end
end

class Cell
  attr_reader :state
end

class Board
  def size
    self.width * self.height
  end
end
```

```
class Cell
  def state=(new_state)
    @state=new_state
  end
end

class Cell
  attr_writer :state
end

class Cell
  attr_accessor :state
end
#
#
```

## Array-liknande

```
class Board
  def [](col, row)
    @cells[col][row]
  end
end

class Board
  def []=(col, row, cell)
    @cells[col][row] = cell
  end
end
```

# Läxa

På alla föreläsningar finns det som sagt *uppgifter* att göra. Dessa är inte examinerande men viktiga att göra ändå. Assistenterna kan kommentera/hjälpa till med dem och vissa är förberedelser till dugga/tenta som inte förekommer i seminarie-serien.

Uppgiften från denna föreläsning är att komma igång med grundläggande Ruby så att ni får en känsla för hur saker och ting fungerar. Gå in på sidan med föreläsningsslides och titta i `fo1.rb` den innehåller kodexempel från föreläsningen. Titta, kör(eventuellt delar) och förstå.



[www.liu.se](http://www.liu.se)