TDP007 - Konstruktion av datorspråk

Mer om Ruby och om språk

Pontus Haglund

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1. Mål
2. Bedöma språk
3. Testning
4. Likhet
5. Regex
6. Första seminariet

LINKÖPING UNIVERSITY

1 Mål
2 Bedöma språk
3 Testning
4 Likhet
5 Regex
6 Första seminariet

LINKÖPING UNIVERSITY

Mål med föreläsningen

Målet är att du som student efter denna föreläsning:

- får lite verktyg för att tänka på hur man kan bedöma ett språk
- förstår hur likhet fungerar i Ruby
- använda enhetstestning i Ruby
- använda regex i Ruby
- har med sig tips inför första seminariet

1 Mål
2 Bedöma språk
3 Testning
4 Likhet
5 Regex
6 Första seminariet

LINKÖPING UNIVERSITY

Varför

- Du bör kunna göra ett informerat val
- Du bör kunna anpassa dig till situationen

Hur kan man bedöma ett språk

- Språk
- Verktyg
- Dokumentation
- Utvecklare
- Historik
- Exempel

Språk

- Kodbibliotek
- Sårskilda konstruktioner
- Baskonstruktioner

Verktyg

- Utvecklingsmiljö
- Körmiljö

Dokumentation

- Dokumentation
- Utbildningsmaterial

Utvecklare

- Community
- Ägare

Historik

- Bakgrund
- Trender

Exempel

- Tillgängliga program
- Framgångsexempel

1 Mål
2 Bedöma språk
3 Testning
4 Likhet
5 Regex
6 Första seminariet

LINKÖPING UNIVERSITY

Varför

Varför testar vi?

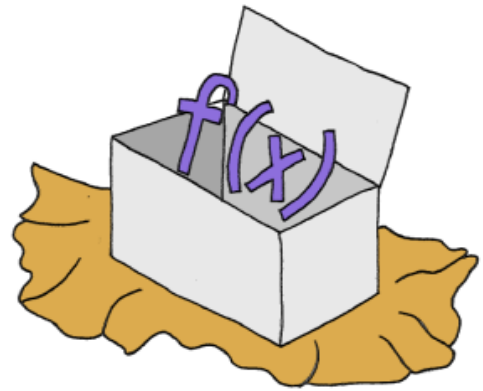
Fyra nivåer

- Enhetstest (unit test)
  - Varje enhet, t.ex. funktion, testas enskilt.
- Integrationstest
  - Test av flera enheter tillsammans
- Systemtest
  - Hela systemet för säkerhet etc.
- Acceptanstest
  - Sluttest av kunden

Två strategier

- Black box test
  - Utgår ifrån specifikation

- White box test
  - Utgår från analys av programkod



Enhetstestning

- Styrkor
  - Underlättar förändring
  - Underlättar integrationstestning
  - Ersätta viss design och dokumentation
  - Mindre fel
- Enhetstestning i Ruby
  - Vi kommer använda ramverket Test::Unit
  - Testfall är en uppsättning förväntningar på utdata

Testklass

```
require './faculty'
require 'test/unit'
class TestFaculty < Test::Unit::TestCase
  def test_simple
    assert_equal(1, faculty(0))
    assert_equal(1, faculty(1))
    assert_equal(120, faculty(5))
    assert_equal(2432902008176640000, faculty(20))
  end
end
```

Kör tester

- Kör genom kommandoraden

```
ruby test_faculty.rb
```

```
Run options:
# Running tests:
.
Finished tests in 0.008113s, 123.2628 tests/s
1 tests, 3 assertions, 0 fails, 0 errors, 0 skips
```

Fler sätt att testa

Titta i dokumentationen för att se alla olika sätt. Test::Unit

Övning

- Hämta filerna `faculty.rb` och `test_faculty.rb` från kurssidan (under föreläsningar). Kör testet.
- Hämta filen `some_functions.rb` och urforma testfall för funktionerna i filen

1 Mål
2 Bedöma språk
3 Testning
4 Likhet
5 Regex
6 Första seminariet

LINKÖPING UNIVERSITY

Likhet i Ruby

- a.equal?(b) - är a samma objekt som b?
- a.eql?(b) - har a samma hashvärde som b?
- a === b - är a samma case som b? (ofta samma som ==)
- a == b - är värdet på a samma som värdet på b?

1 Mål
2 Bedöma språk
3 Testning
4 Likhet
5 Regex
6 Första seminariet

LINKÖPING UNIVERSITY

Varför?

- Regex är den enklaste formen av formellt språk
- Söka i textsträngar
  - wildcards
  - grep
  - scriptspråk
- Många olika uppsättningar

I Ruby

- Ruby har ett mycket bra stöd för Regex.
- Framåt slash "/" används för att skilja Regex från andra konstruktioner
- Mycket bättre variant än Python

Exempel

```
>> /be/ =~ "To be, or not to be..."
=> 3
>> /o+/ =~ "Moo mooo moooo!"
=> 1
>> "23, 18, 45" =~ /1[0-9]/
=> 4
>> /\w\d/ =~ "ABC123"
=> 2
>> /kul/ =~ "Inget roligt här!"
=> nil
```

Testa reguljära uttryck i Ruby själv på webbsidan rubular.com

Repetition av regex

Speciella tecken:

```
\^/$|.+*?()[]{}
```

Alla andra tecken matchar sig själva i uttryck

Funktioner som använder regex

```
>> s = "The stars, like dust"
>> s.sub(/e/,'$')
=> "Th$ stars, like dust"

>> s.gsub(/[aeiouy]/,'*')
=> "Th* st*rs, l*k* d*st"

>> s.gsub(/(^|,|\s)\w/) { |m| m.upcase }
=> "The Stars, Like Dust"

>> s.split(/[\s,]+/)
=> ["The", "stars", "like", "dust"]

>> s.scan(/[aeiouy]\w/)
=> ["ar", "ik", "us"]
```

Resultat av matchning

```
>> /[aeiouy]{2,}/ =~ "the moon is a cheese"
=> 5

>> [$`, $&, $']
=> ["the m", "oo", "n is a cheese"]

>> /(\d\d):(\d\d):(\d\d)/ =~ "Hon är 12:35:40 nu"
=> 12

>> [$1, $2, $3]
=> ["12", "35", "40"]
```

Resultat av matchning

```
>> s = "the moon is a cheese"
>> re = /[aeiouy]{2,}/

>> md = re.match(s)
=> #<MatchData:0xdf65ed4>

>> [md.pre_match, md[0], md.post_match]
=> ["the m", "oo", "n is a cheese"]

>> md2 = /(\d\d):(\d\d)/.match("14:45")
=> #<MatchData:0xdb94f94>

>> md2.captures
=> ["14", "45"]
```

Övning

- Hitta på ett reguljärt uttryck som matchar en svensk postadress, t.ex. 584 31 Linköping. Finns det några postorter vars namn innehåller något annat än bara vanliga bokstäver?
- Baka in det i en funktion som svarar sant eller falskt, givet en sträng som kanske innehåller en postadress.
- Gör en testklass med lämpliga testfall. Ta även med negativa tester, dvs det reguljära uttrycket får inte matcha strängar som inte är postadresser.
- Testa reguljära uttryck i Ruby på rubular.com

1 Mål
2 Bedöma språk
3 Testning
4 Likhet
5 Regex
6 Första seminariet

LINKÖPING UNIVERSITY

Inför första seminariet

- Hur kan man snabbt sätta sig in i den andra gruppens kod med begränsad tid?
- Hur gör man en bra opposition?

www.liu.se

LINKÖPING UNIVERSITY