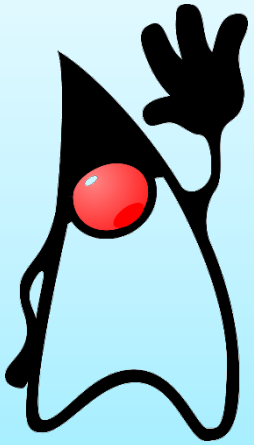
TDDD78 / 729A85

Objektorienterad programmering och Java



jonas.kvarnstrom@liu.se – 2016

Fråga – kommentera – avbryt!

140
160
180
TOO FAST!

TOO
SLOW

EMERGENC
STO

Inblandade i kursen

Inblandade

Examinator, kursledare: Jonas Kvarnström

Inblandade (2)

Handledare, sedan flera år

Mikael Nilsson
Daniel de Leng
Piotr Rudol
Mariusz Wzorek

D-studenter

U-studenter

Kogvet-studenter

…?

Bakgrund, mål och medel

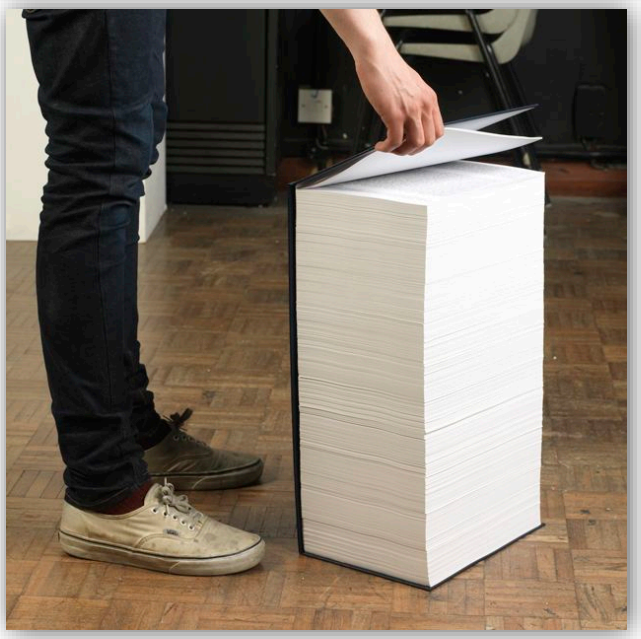
Kunskaper

Vad kan ni redan?

Grundläggande begrepp inom programmering

Konkreta programmeringskunskaper i Python, motsvarande t.ex. TDDD73

Läsa!



Objektorienterad programmering

ERA MÅL:

Få kunskap och färdigheter
inför
framtida programutveckling

Utan OO har vi **datastrukturer** för lagring, som ”manipuleras utifrån” av **funktioner**

```
int daysSinceToday(date)
boolean isLeapYear(date)
void add(date, days)
```

Date-struktur

year: 2021
month: 12
day: 31

Med OO har **objekt** både datalagring och egen funktionalitet, egna **metoder**

- ➔ **Fundamental** princip:
  “Objektet bestämmer över sig själv”
  (ingen manipulerar utifrån)
- ➔ **Grunden** för begrepp som *ärvning, overriding, polymorfism, inkapsling, …*

Date-objekt

year: 2021
month: 12
day: 31

daysSinceToday()
isLeapYear()
add(days)

En **klass** är en datatyp, t.ex. **Date**, som anger:
Vilken *information* lagras i objekt av typ Date?
Vad kan man göra med objekt av typ Date?

Objektorienterad programmering

Konkret OO-språk: Java

Generella språkbegrepp:

Stark typning,
…

ERA MÅL:

Få kunskap och färdigheter inför framtida programutveckling

Generella färdigheter:

Verktyg för utveckling

Programmeringsbegrepp:

Modularitet,
…

Programmeringsvana!

D-studenter

U-studenter

Kogvet-studenter

…?

Studenter, inte ***elever***

Kunskapstörstande

Aktiva och proaktiva

MEDEL:

Hur ska ni uppnå era mål?

Labbar och miniprojekt
med instruktioner

3 hp (utökat)
Godkänt / komplettering

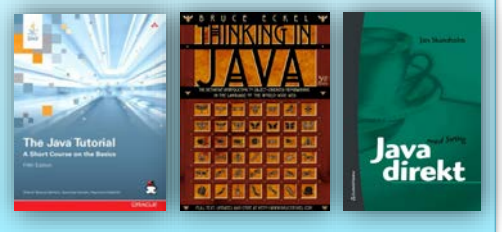
Egna tankar och reflektioner!

Projekt
med friare val

3 hp
3 / 4 / 5 / komplettering

Föreläsningar

Kurslitteratur, info på nätet



Automatisk kodinspektion
för att hitta förbättringsmöjligheter

Inlämning
Feedback
Komplettering

Handledare
att fråga och diskutera med

Föreläsningar

Introduktion
Fö 1-2: Kursinfo, labbinfo, Java för Python-programmerare
Utan objektorientering: Ändå mycket nytt…

Objektorienteringens grunder
Principer och begrepp
Konkret användning av OO i Java

Grafiska gränssnitt
med *komponenter* och *händelsehantering*

Funktionalitet i Java
Sammansatta datatyper: Listor, …
Signalera och hantera fel
Text, I/O, …

Projektinfo
Välja projekt, …

Mer objektorientering
Principer för "*bra*" *programkod*
Återanvändbara *designmönster*

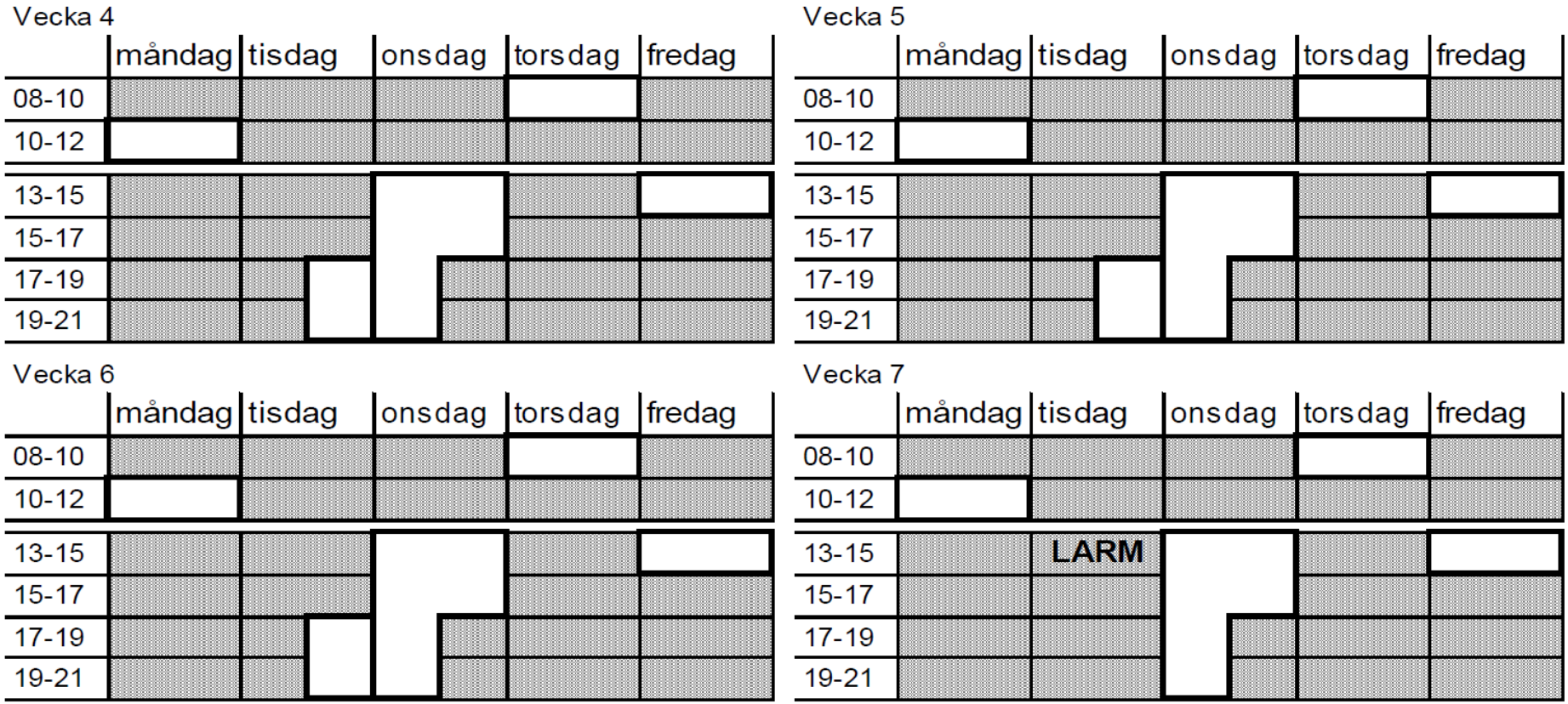
Bonus (nästa period)
Överkurs för de som vill veta mer

Om föreläsningarna

- Föreläsningarna utgår från *förkunskapskraven*
  - Ni som har programmerat i Java förut:
    - Titta på innehållet på förhand
    - Kan redan ➔ kanske *läsa* istället för att gå till föreläsningen
    - Tillräckligt mycket nytt/osäkert:
      Kom hit – och var beredda på att vissa avsnitt känns långsamma
  - Övriga:
    - Tveka aldrig att stoppa föreläsningen och fråga!

- Bilderna innehåller *mycket text*
  - Vanliga kommentarer:
    - Bra att man kan läsa efteråt, kan förstå sammanhanget
    - Känns ibland som att föreläsaren läser innantill

Blockschema



Ibland 8-10, 17-19

Steg 0: Labbanmälan

Snarast:
Anmälan till steg 1 i WebReg, <u>enskilt</u>

8 grupper (A-B-C-D-E-F-G-H), olika labbtider:

D1A → A, B

D1B → C, D

D1C → E, F

U1 → G, H

Kogvet, övriga → A, B, E, F

Steg 1: Grunder, labbmiljö, era första Java-program

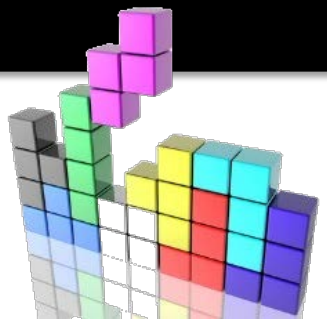
- Vecka 3
- Vecka 4
- Vecka 5
- Vecka 6

4 labbar, enskilt
Lugnare, föreläsningstakt styr:
Gör klart i tid!

Steg 2: Fortsättning, miniprojekt

- Vecka 7
- Vecka 8
- Vecka 9

1 miniprojekt, enskilt
Tyngre, mycket att hinna med



Steg 3: Utforska, skapa ett eget projekt!

- Vecka 10
- Vecka 11
- Vecka 12
- Vecka 13
- Vecka 14
- Vecka 15
- Vecka 16
- Vecka 17

Grupper om 2 rekommenderas
Slut mitt i perioden:
Andra kurser att fokusera på!

Uppföljning

Vad kommer sedan?

OOP

Begrepp

Java

Mer programmeringsvana!

U:
TDDD80 (11 hp)
Mobila och sociala applikationer / Java

D, U:
TDDD86 (11 hp)
Datastrukturer, algoritmer och programmeringsparadigm / C++

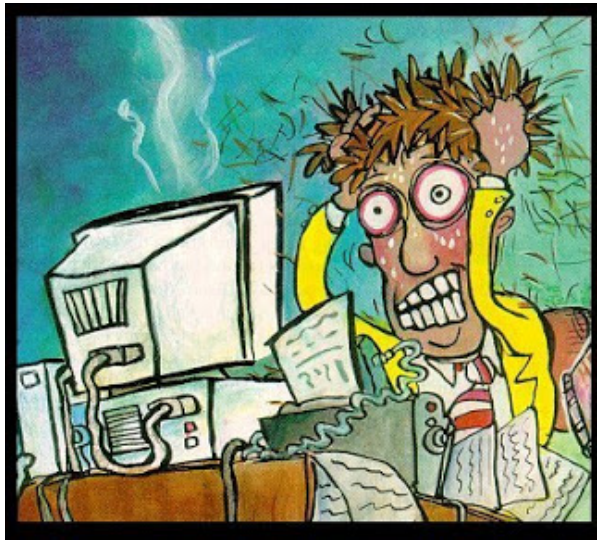
TDDB68 Processprogrammering och operativsystem / OO

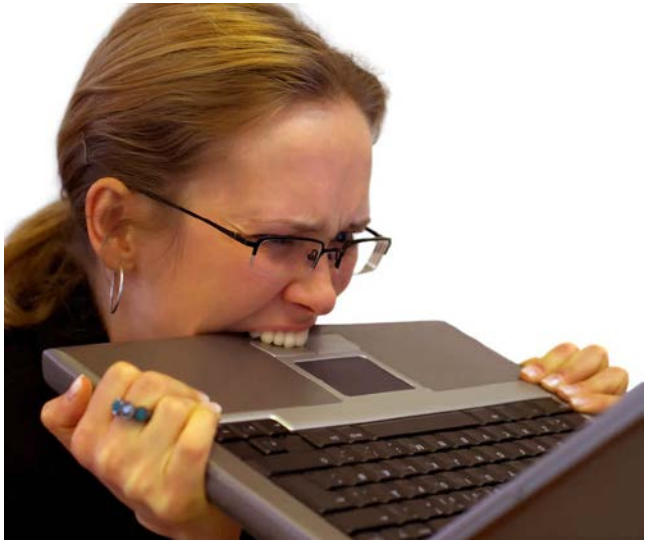
Kursutveckling

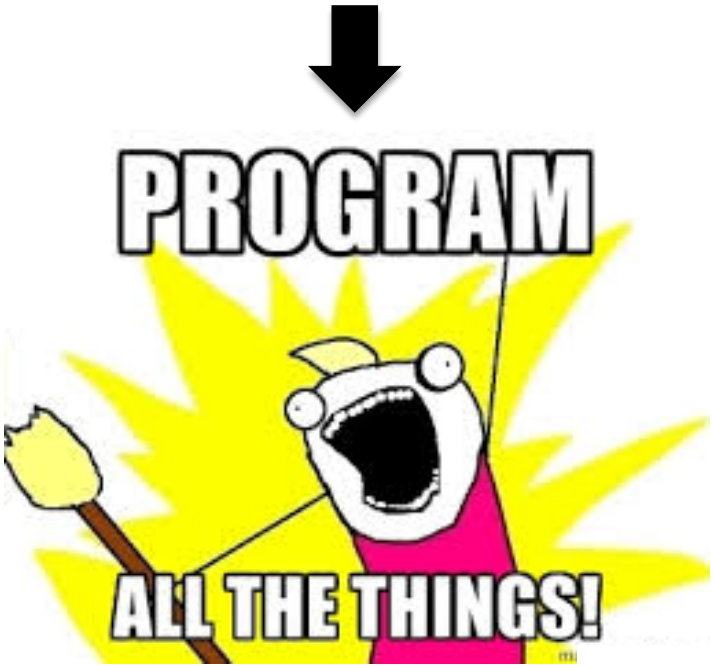
- **<u>Nytt för i år</u>:**
  - **Omarbetning av föreläsningar**
    - Strömlinjeformning, mer fokus på de allra viktigaste begreppen
    - Mer fokus på *bra programmering*, hur man *använder* det nya på bästa sätt (utgångspunkt i vanliga misstag och kompletteringar)
  - **Mer tid för grundläggande labbar: 3+3 HP istället för 2+4**
    - Önskat av studenter + oss
    - Mer övning på vissa *nya begrepp och tekniker*
    - Mer övning på *bra programmering*
  - **Designmönster flyttade från projekt till labbar**
    - Knepigt att genomföra själv för första gången

  - **Extra *redovisningstillfällen* för labbar**
    - Enbart redovisning – även möjligt på vanliga labbar, men *handledning prioriteras*

Början och slutet...







Bakgrund:
Varifrån kommer Java? Varför använda det?

Historia: C, C++

1960-talet: **CPL** → **BCPL** → **B**

↓

1969-1973: **C** → skrev om delar av Unix-kärnan

↓

1978: **K&R C** – "*The C Programming Language*" av Kernighan & Ritchie

↓

1979: **C with Classes** – OO-finesser från Simula (Bjarne Stroustrup)

↓

1983: **C++** – Vidareutveckling

"C++":
"C = C + 1"

↓

1985: **The C++ Programming Language** (bok)

↓

1989: **C++ 2.0**

Historia: Oak 1

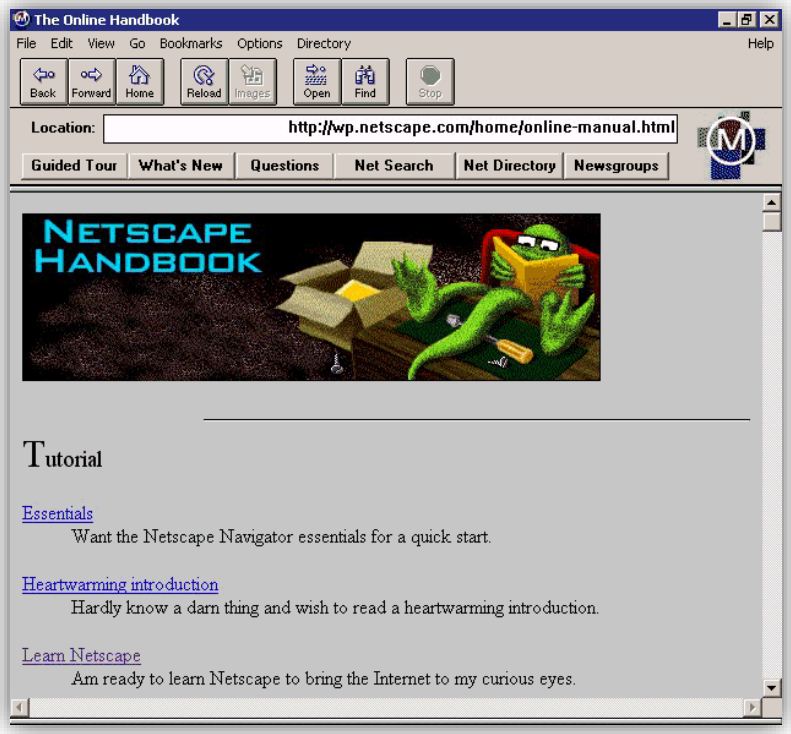
- 1990: Sun Microsystems påbörjade projekt **StarSeven**
  - Avancerad interaktiv ”multikontroll”: TV, video, satellit, …
    - *Touchskärm*, dra program i listan till en video, …
    - *Trådlöst nätverk*
    - Unix-OS

Historia: Oak 2

- Ville ha ett objektorienterat **programmeringsspråk**
  - C++ sågs som problematiskt
  - James Gosling utvecklade sitt eget: "**C++ ++ – –**"
    - Omdöpt till "Oak"
  - 1991–1994:
    - Skapa spinoff-företag
    - Bygga, programmera, demonstrera
    - Gå i konkurs

Historia: Oak 3

- Hitta en **annan användning**!
  - 1994: WWW på gång
    - Nov 1993: 500 WWW-servrar!
    - Okt 1994: Netscape Mosaic 0.9!
  - Webben var väldigt statisk
    - Interaktivitet? Formulär!
    - JavaScript? Slutet av 1995…
    - Flash? 1996…
  - **Oak passade perfekt**!
    - Portabelt, plattformsoberoende
    - Bra stöd för nätverkskommunikation
  - Språket döptes om till **Java**
    - Många arv kvar från C++, C, till och med B (1969)



Varför Java?

- Varför **Java**, när det finns så många OO-språk?

”Rena” OO-språk (*allt* är objekt)	OO-språk med vissa procedurella aspekter	Procedurella språk med OO-utökningar
Scala, Smalltalk, Eiffel, Ruby, JADE, Emerald, …	*Java, C++, C#, Python, VB.NET, …*	*Fortran 2003, Perl, COBOL 2002, PHP, ABAP, …*
		Allgemeiner Berichts-Aufbereitungs-Prozessor

Objektorientering kan också kombineras med andra paradigm!

Haskell++ (t.ex.) är *funktionellt* + *OO*

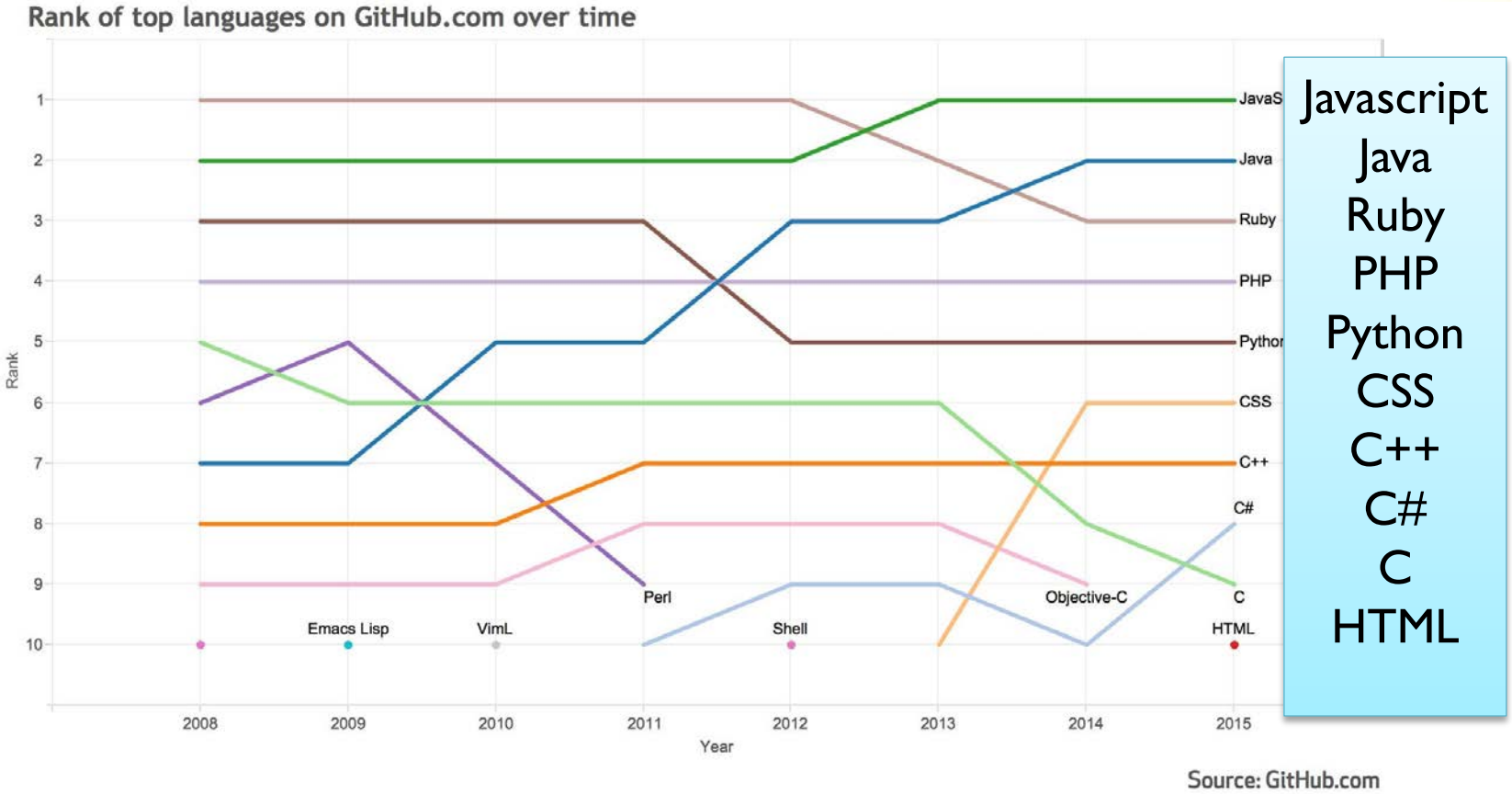
Varför 2

- Inte Python igen…
  - Vi behöver **breda** kunskaper, skilda perspektiv!

…Lär dig åtminstone ett halvt dussin olika programmeringsspråk.
Peter Norvig (2001) Teach Yourself Programming in Ten Years.
http://norvig.com/21-days.html

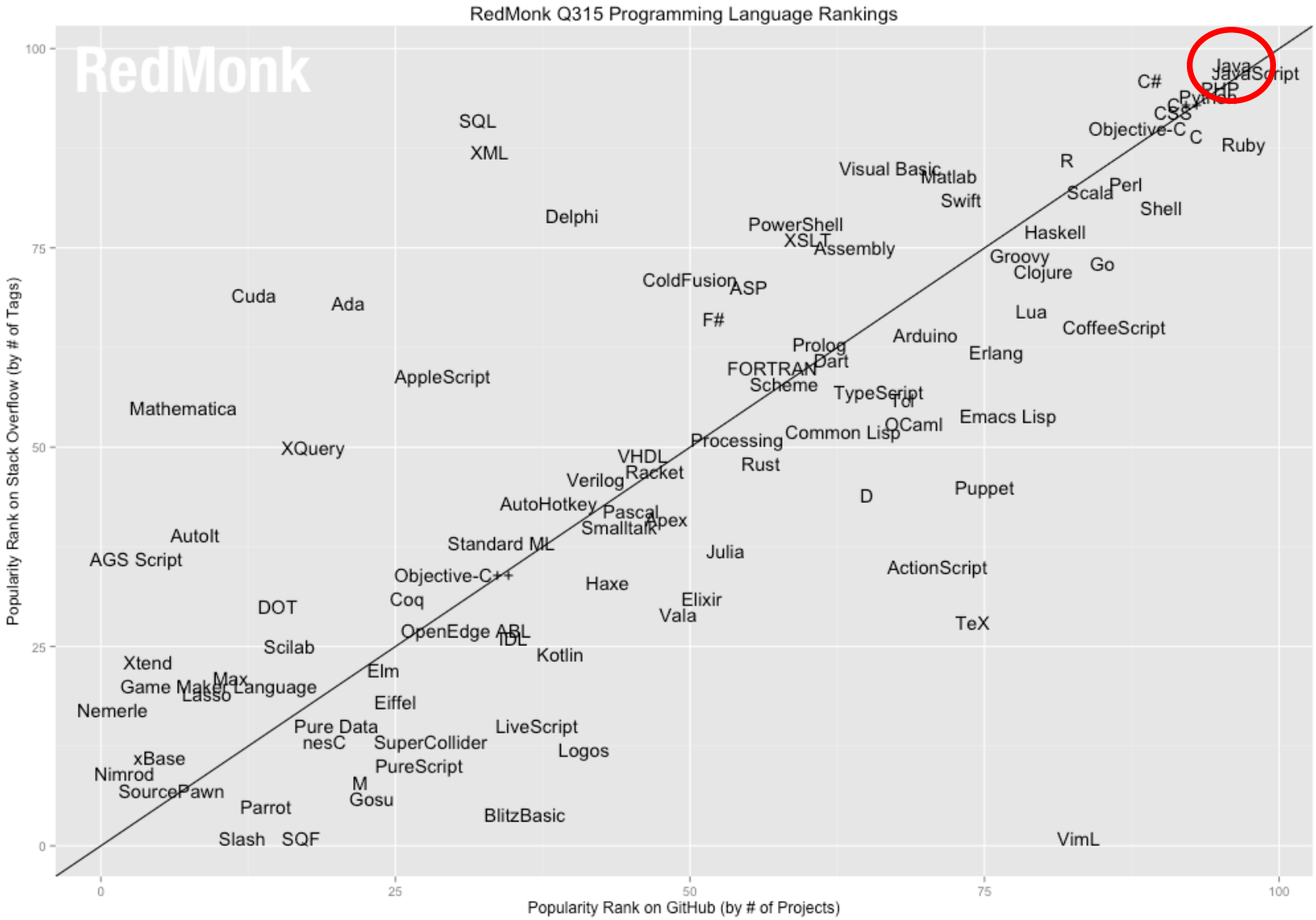
Varför 3

- Ett av de mest **<u>använda</u>** OO-språken, i **<u>många områden</u>**
  - Webapplikationer
  - Viktiga delar av Googles system
  - Android-appar (med några skillnader)



Javascript
Java
Ruby
PHP
Python
CSS
C++
C#
C
HTML

GitHub + Stack Overflow



Varför 4

- Många **likheter** med **andra vanliga språk**
  - …men "enklare grunder" än många av dessa

  - http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Position jan-15	Position jan-16	Language	Ratings jan-15
2	1	**Java**	16.703%
1	2	**C**	15.528%
4	3	**C++**	6.953%
5	4	**C#**	6.705%
8	5	**Python**	5.045%
6	6	**PHP**	3.784%
16	7	**Visual Basic .NET**	3.274%
7	8	**Javascript**	2.613%

C och dess efter-följare

Varför 5

- Många egenskaper passar bra för **undervisning**

Mindre omfattande än vissa andra språk	Jämförelsevis ”lätt att analysera”	Skräpsamling, minnesskydd
"C++ is designed to give the programmer choice, even if this makes it possible for the programmer to choose incorrectly"	*Bra utvecklingsmiljöer, avancerat stöd*	*Svårare att göra fel, lättare att hitta fel*

Fungerar på många plattformar	Stort klassbibliotek	…
C# mer bundet till Windows	*Funktionalitet som finns överallt*	…

Introduktion till Java

– för Pythonprogrammerare

jonas.kvarnstrom@liu.se – 2016

Två sammanhängande delar

Föreläsning 1-2:

Allmän orientering, teoretisk kunskap

En första känsla för skillnaderna

Labb 1:

Prova på, få praktisk erfarenhet

Förstå <u>i detalj</u>
hur språken skiljer sig

Många nya begrepp, allt beror på allt annat...

Förenkling:
Börja med Java utan objektorientering,
fokusera på de andra skillnaderna

Ändå lite knepigt:
Java förutsätter objektorientering, klasser, ...

Kompromiss:
Använd vissa nyckelord utan att riktigt förstå dem än...

Python och Java:
Skillnader i mål och "filosofi"

Skillnad i mål och "filosofi"

Python	Java
Enkelt att skriva små program	Fokus på större program
Ska vara lätt att komma igång med	OK om det kräver lite mer att komma igång
	Mer fokus på prestanda (!)

Har många konsekvenser!

Kod på toppnivån (1)

Python:
Kod *kan* skrivas på "toppnivån"

Java:
All kod är i en klass,
alla satser i en metod (funktion)

HelloWorld.py

```
print("Hello World")
```

Kod på toppnivå körs
när man "kör filen"

HelloWorld.java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

main() startas
när man "kör klassen"

Enkel uppgift – enkel kod

Mer "overhead" runtomkring
Märks mindre i större program!
Har vissa fördelar…

klass = class
metod = method

Interaktivitet (1)

- Python kan köras med kod **sparad i en fil**…

emacs@mina4.ida.liu.se

File Edit Options Buffers Tools Python Help

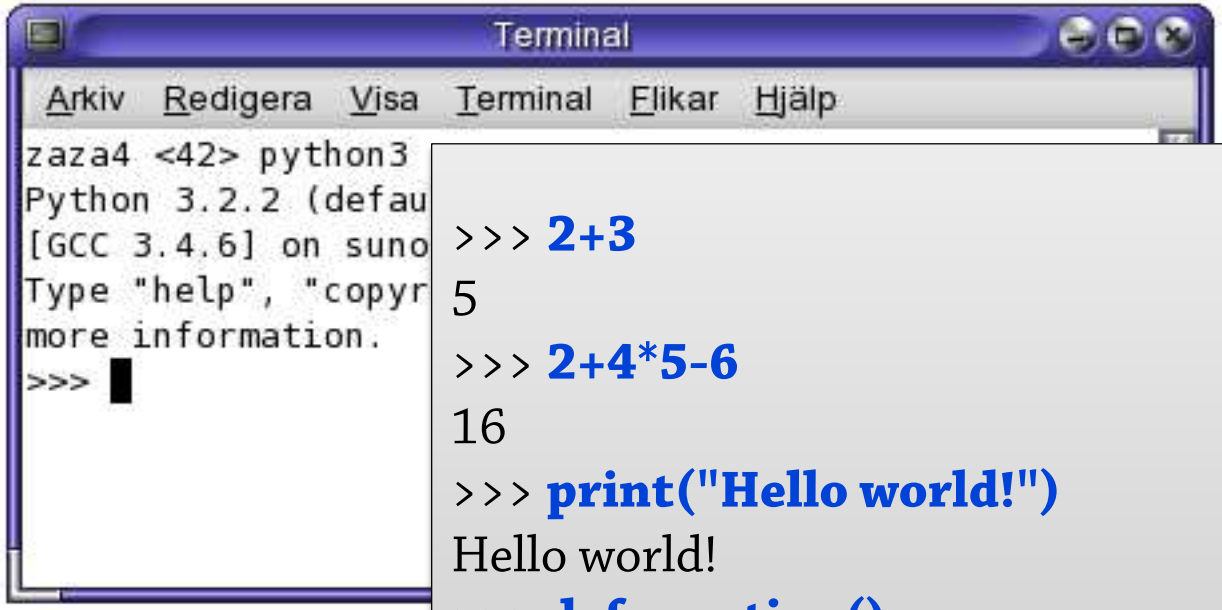
```
def find_root():
    print("This program tries to find the square root of a number.")
    x = eval(input("Enter a number: "))
    guess = x/2
    for i in range(5)
        guess = (gues
        print(guess)

find_root()
```

-1:--- newton.py Al

```
zaza4 <23> python3 newton.py
This program tries to find the square root of a number.
Enter a number:
2
1.5
1.4166666666666665
1.4142156862745097
1.4142135623746899
1.414213562373095
```

Interaktivitet (2)

- …eller i **interaktivt läge**




```
>>> 2+3
5
>>> 2+4*5-6
16
>>> print("Hello world!")
Hello world!
>>> def greeting():
... print(”Nobody expects the Spanish Inquisition.")
... print(”Our chief weapon is surprise… and fear.")
...
>>> greeting()
Nobody expects the Spanish Inquisition.
Our chief weapon is surprise… and fear.
```

Interaktivitet (3)

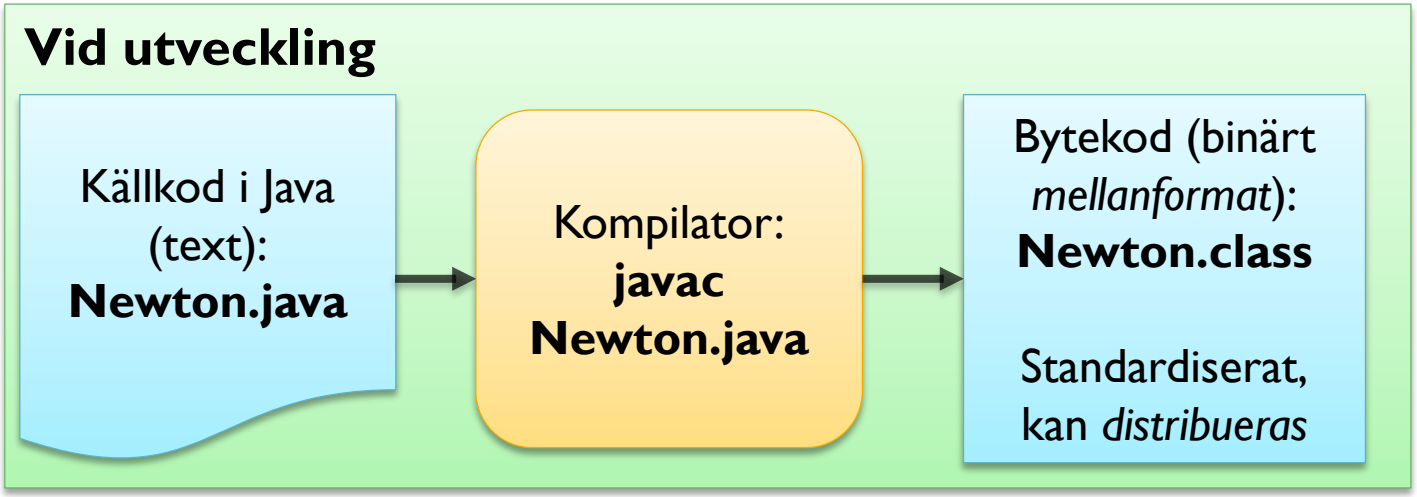
- Java har inget interaktivt läge!
  - Skriv *alltid* kod i en fil

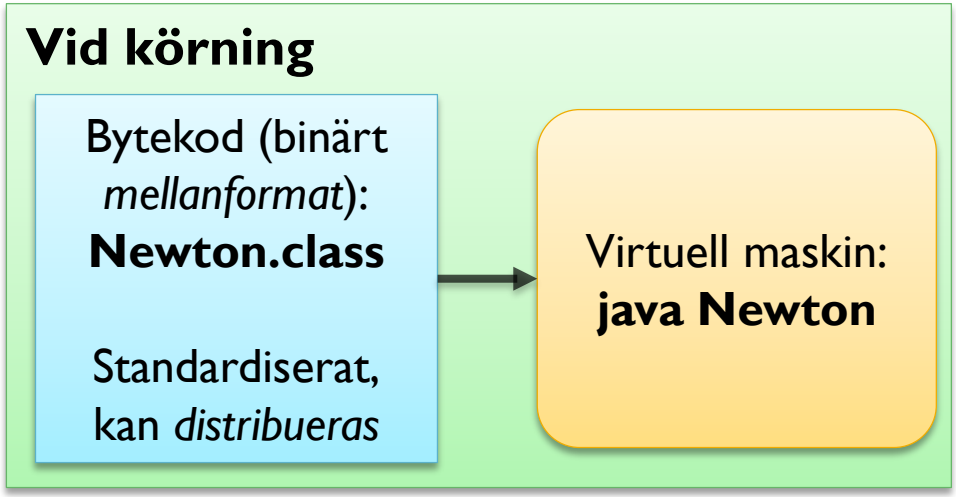
Fil: Newton.java

```
public class Newton {
   public static void findRoot() {
      System.out.println("This program tries to find...");
      System.in.read......;
      double guess = x/2;
      for (int i  = 1; i < 5; i++) {
         guess = (guess + x/guess) / 2;
         System.out.println(guess);
      }
   }
   public static void main(String[ ] args) {
      findRoot();
   }
}
```

Interaktivitet (4): Explicit kompilering

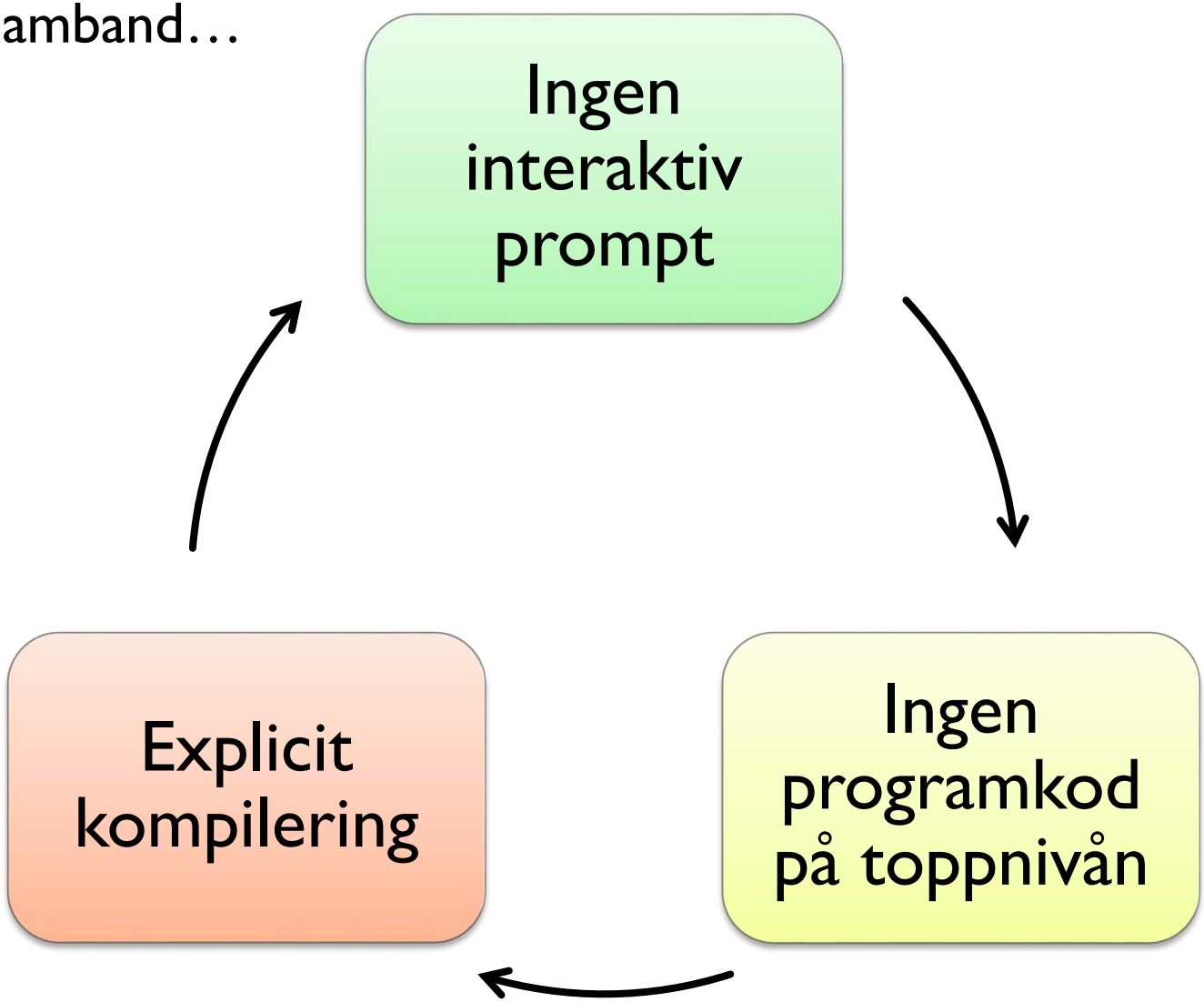
- Java har **explicit kompilering** – ett extra steg





Interaktivitet (5)

- Nära samband…



Python och Java:
Skillnader i ytlig struktur

Syntaxskillnader

Fil: Newton.py

```
# Vad ska vi skriva här?
```

Fil: Newton.java

```
// Vad ska vi skriva här?

/* En lång kommentar
   som kan sträcka sig
   över flera rader */
```

Syntaxskillnader

Fil: Newton.py

Fil: Newton.java

```
public class Newton {
}
```

All kod <u>måste</u> ligga i en klass…

Det som <u>ingår</u> i klassen läggs inom { … }
(Java bryr sig inte om indentering!)

Namnstandard för klasser:
VarjeOrdHarStorBokstav
JavaTest, ArrayList, …

Unicode – kan använda
å, Θ, π, Σ, Я, Õ

Syntaxskillnader

Fil: Newton.py

```
def find_root():
```

Fil: Newton.java

```
public class Newton {
    public static void findRoot() {
    }
}
```

Just nu vill vi inte använda objekt
Ändå måste metoder vara inuti klasser

public →
vem som helst får använda

static →
behöver inget objekt av typen Newton
("vanlig funktion, inte objektorienterad")

void →
returnerar inget värde (*procedur*)

Syntaxskillnader

Fil: Newton.py

```
def find_root():
```

Fil: Newton.java

```
public class Newton {
    public static void findRoot() {
```

Klamrar runt metodens kod: { … }

Namnstandard:
storBokstavUtomFörstaOrdet

Syntaxskillnader

Fil: Newton.py

```python
def find_root():
    print("This program tries to find....")
```

Python-strängar:
"Hello" eller 'Hello'

Jämförelse: str1 == str2

Fil: Newton.java

```java
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
```

Java-strängar:
"Hello"

Jämförelse: str1.equals(str2)

Apostrofer kan användas för enskilda *tecken*: 'H'

```
    }
}
```

Syntaxskillnader

Fil: Newton.py

```
def find_root():
    print("This program tries to find....")
```

Fil: Newton.java

```
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
}
```

Utskrift är en *metod*, "println", i ett objekt, "System.out"
➔ Mer info senare!

Semikolon efter satser
Radbrytning räcker inte!

Vi indenterar för <u>läsbarhet</u>
Ignoreras av språket

```
public class Newton { public static void findRoot() { System.out.println("This..."); } }
```

Syntaxskillnader

Fil: Newton.py

```python
def find_root():
    print("This program tries to find....")
    x = eval(input("Enter a number: "))
```

Fil: Newton.java

```java
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
        ...;

    }
}
```

Java är mindre anpassat för "textprogram"

Enklare att visa en dialogruta (kommer på labben)

Syntaxskillnader

Fil: Newton.py

```
def find_root():
    print("This program tries to find....")
    x = eval(input("Enter a number: "))
    guess = x/2
```

Va? Det kan väl datorn fatta själv?

Ja, men tänk om du skriver *guss* nästa gång.
Mer att skriva men extra säkerhetsbälte.

Va? Det kan väl datorn fatta själv?

Delvis, men det har också nackdelar.
Vi återkommer till typning nästa gång!

Fil: Newton.java

```
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
        ...;
        double guess = x/2;

}
```

Variabler måste <u>deklareras</u>:
"Här kommer en <u>ny</u> variabel"
(annars säger kompilatorn att "*guess* finns inte")

Java har <u>explicit typning</u>:
Ange alltid vilken typ en variabel ska ha
(double = "decimaltal")

deklarera = declare

Syntaxskillnader

Fil: Newton.py

```python
def find_root():
    print("This program tries to find....")
    x = eval(input("Enter a number: "))
    guess = x/2
```

Fil: Newton.java

```java
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
        ...;
        double guess = x/2;

    }
}
```

Namnstandard:
storBokstavUtomFörstaOrdet

Syntaxskillnader

Fil: Newton.py

```
def find_root():
    print("This program tries to find....")
    x = eval(input("Enter a number: "))
    guess = x/2
    for i in range(5):
```

Fil: Newton.java

```
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
        ...;
        double guess = x/2;
        for (int i  = 0; i < 5; i++) {
```

Heltalstyp: int

Annan loopsyntax: (start; villkor; steg)

Deklarera heltal i = 0
Så länge som i < 5 {
Utför "kroppen" av loopen
i++ (öka värdet på i)
}

Syntaxskillnader

Fil: Newton.py

```
def find_root():
    print("This program tries to find....")
    x = eval(input("Enter a number: "))
    guess = x/2
    for i in range(5):
```

Fil: Newton.java

```
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
        ...;
        double guess = x/2;
        for (int i  = 0; i < 5; i++) {
```

Loopens kropp läggs också inom klamrar { … }

Efter klamrar: Inget semikolon

```
        }
    }
}
```

Syntaxskillnader

Fil: Newton.py

```python
def find_root():
    print("This program tries to find....")
    x = eval(input("Enter a number: "))
    guess = x/2
    for i in range(5):
        guess = (guess + x/guess) / 2
```

Fil: Newton.java

```java
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
        ...;
        double guess = x/2;
        for (int i  = 0; i < 5; i++) {
            guess = (guess + x/guess) / 2;
```

Nu är *guess* redan deklarerad!
Många operatorer liknar Python…

```java
        }
    }
}
```

Operatorer i Python och Java

Python		Java
Räknesätt:	+ - * / %	*Samma*
Upphöjt till:	**	Använd Math.pow(bas, exponent)
Division avrundad nedåt:	//	Använd (int) (x / y)
Jämförelser:	== != > < >= <=	*Samma*
Olikhet:	<>	Använd !=
Tilldelning:	= += -= *= /= %=	*Samma*
Tilldelning:	i += 1	Använd i++
Tilldelning:	i -= 1	Använd i--
Bitoperatorer:	& | ~ << >>	*Samma (men finns flera)*
Villkor:	and or not	Använd: && || !

Förväxla inte med & eller |
IDEA varnar – förklaring i labb 1!

Syntaxskillnader

Fil: Newton.py

```python
def find_root():
    print("This program tries to find....")
    x = eval(input("Enter a number: "))
    guess = x/2
    for i in range(5):
        guess = (guess + x/guess) / 2
        print(guess)
```

Fil: Newton.java

```java
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
        ...;
        double guess = x/2;
        for (int i  = 0; i < 5; i++) {
            guess = (guess + x/guess) / 2;
            System.out.println(guess);
        }
```

Utskrift igen…

```
    }
}
```

Syntaxskillnader

Fil: Newton.py

```python
def find_root():
    print("This program tries to find....")
    x = eval(input("Enter a number: "))
    guess = x/2
    for i in range(5):
        guess = (guess + x/guess) / 2
        print(guess)


    print "Done!"
```

Fil: Newton.java

```java
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
        ...;
        double guess = x/2;
        for (int i  = 0; i < 5; i++) {
            guess = (guess + x/guess) / 2;
            System.out.println(guess);
        }

        System.out.println("Done!");
    }
    ...
}
```

Syntaxskillnader

Fil: Newton.py

```
def find_root():
    print("This program tries to find....")
    x = eval(input("Enter a number: "))
    guess = x/2
    for i in range(5):
        guess = (guess + x/guess)/2
        print(guess)
    print "Done!"
```

Radbrytning avslutar sats

Gruppering via indentering

Fil: Newton.java

```
public class Newton {
    public static void findRoot() {
        System.out.println("This...");
        ...;
        double guess = x/2;
        for (int i  = 0; i < 5; i++) {
            guess = (guess + x/guess) / 2;
            System.out.println(guess);
        }
        System.out.println("Done!");
    }
    ...
}
```

Semikolon eller {} avslutar sats

Gruppering via måsvingar {}

Indentering ändå viktigt för läsbarhet!

Funktioner

- Pythons funktioner på toppnivå kan "emuleras" i Java
  - Använd en **public static**-funktion
  - *Bara till vi har hunnit läsa om objekt!*

Fil: Newton.java

```
public class Newton {
    public static int findRoot() {
        System.out.println("This program tries to find...");
        System.in.read......;
        double guess = x/2;
        for (int i  = 1; i < 5; i++) {
            guess = (guess + x/guess) / 2;
        }
        return guess;
    }
    public static void main(String[ ] args) {
        System.out.println(findRoot());
    }
}
```

Ange alltid returtyp
Om inget returneras: void

Returnera värde med **return**

Mer Java:
Villkorssatser

sats = statement

Villkor: if

Python

```
if condition:
    statement1
    statement2
    ...
```

Java

```
if (condition) {
    statement1
    statement2
    ...
}
```

Python

```
if condition:
    statement1
    statement2
    ...
else:
    statement1
    statement2
    ...
```

Java

```
if (condition) {
    statement1
    statement2
    ...
} else {
    statement1
    statement2
    ...
}
```

Villkor: if (2)

Python

```
if condition:
    statement1
    statement2
    ...
elif condition2:
    statement1
    statement2
    ...
elif condition3:
    statement1
    statement2
    ...
```

Java

```
if (condition) {
    statement1
    statement2
    ...
} else if (condition2) {
    statement1
    statement2
    ...
} else if (condition3) {
    statement1
    statement2
    ...
}
```

Test av samma uttrycks värde


```
if (x + y == 0) {
    System.out.println("Exakt noll");
} else if (x + y == 1 || x + y == 2) {
    System.out.println("Ett eller två");
} else if (x + y == 3) {
    System.out.println("Exakt tre");
    System.out.println("Fler satser här");
} else {
    System.out.println("Något annat");
}
```

Här jämför vi värdet på x+y med några kända *konstanter*…

konstanter = constants

Satser: switch


```
switch (x + y) {
    case 0:
        System.out.println("Exakt noll");
        break;
    case 1:
    case 2:
        System.out.println("Ett eller två");
        break;
    case 3:
        System.out.println("Exakt tre");
        System.out.println("Fler satser här");
        break;
    default:
        System.out.println("Något annat");
}
```

Kan göras med en switch-sats! Fungerar för heltal, strängar och *enum-konstanter*

"Klar": Hoppa ur switch-satsen

Flera fall (1 och 2) ger samma kod att utföra

Frivilligt: Vad händer när inget av fallen passar?

Tydligt att man testar värdet på ***ett*** uttryck; skriv uttrycket (x+y) ***en gång***

Kräver extra "break"…

Satser: switch med fallthrough


```
switch (x + y) {
    case 0:
        System.out.println("Exakt noll");
        break;
    case 1:
    case 2:
        System.out.println("Ett eller två");
    case 3:
        System.out.println("Ett, två eller tre");
        break;
    default:
        System.out.println("Något annat");
}
```

Inget 'break' här,
så vi *fortsätter* in i nästa fall

Fallthrough kan förvirra – oftast bra att skriva på annat sätt!

Kommande föreläsningar:

Explicit typning: I Java och många andra språk
Mer kursinformation och labbinformation

Objektorientering...