

Introduktion till Java

– för Pythonprogrammerare

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 += 1" or
"C = C + 1"

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

1989: **C++ 2.0**



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



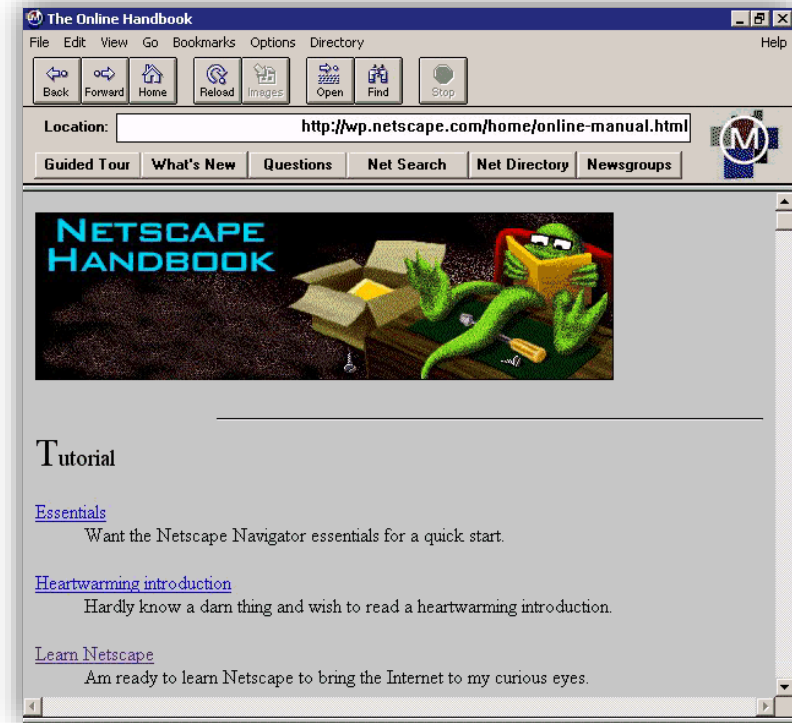
- Ville ha ett objektorienterat programmeringsspråk
 - C++ sågs som problematiskt
 - James Gosling utvecklade sitt eget: "C++ ++ --"
 - Omdöpt till "Oak"
 - 1991–1994:
 - Skapa spinoff-företag
 - Bygga, programmera, demonstrera
 - Gå i konkurs



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*)

*Scala, Smalltalk, Eiffel,
Ruby, JADE, Emerald, ...*

OO-språk med vissa
procedurella aspekter

*Java, C++, C#, Python,
VB.NET, ...*

Procedurella språk med
OO-utökningar

*Fortran 2003, Perl,
COBOL 2002, PHP, ABAP,
...*

Allgemeiner Berichts-Aufbereitungs-
Prozessor

Objektorientering kan också kombineras med andra paradigmer!

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

Inte Python igen: Behöver **breda** kunskaper, skilda perspektiv!

Learn at least a half dozen programming languages.

– Peter Norvig (2001): *Teach Yourself Programming in Ten Years*, <http://norvig.com/21-days.html>

Likheter med andra vanliga språk:

C, C++, C#,
(PHP, Javascript, ...)

Använder vanliga begrepp, tankesätt:
Typning, klasser,
statisk scoping, ...

Språk:
Varför Java?

Vanligt:
#1 på TIOBE index
#2 på GitHub och
RedMonk (efter
JavaScript)

Medelkomplexitet:
Mer att tänka på än Python,
lättare att börja än C++/...

Passar till undervisning:
Färre fallgropar än vissa språk,
bra utvecklingsmiljöer,
bra grundbibliotek av kod/klasser, ...

Python och Java: Hur skriver vi kod?

Många nya begrepp, mycket ny syntax...



Python:

Flera programmeringsparadigm

Inklusive imperativ procedurell kod

HelloWorld.py

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

Kod på toppnivå körs
när man "kör filen"

Enkel uppgift – enkel kod

Java:

Klassbaserad objektorientering

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

HelloWorld.java

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

main() startas
när man "kör klassen"

Mer "overhead" runtomkring...

Märks mindre i större program!

Kan inte förklara allt på en gång!

Acceptera class, static, ... tills vidare –
förklaras mer senare...

klass = class
metod = method

Interaktivitet (1)



- Python kan köras med kod sparad i en fil...

```
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 = (guess + x/guess) / 2
        print(guess)

find_root()
```

zaza4 <23> **python3 newton.py**

This program tries to find the square root of a number.

Enter a number:

2

1.5

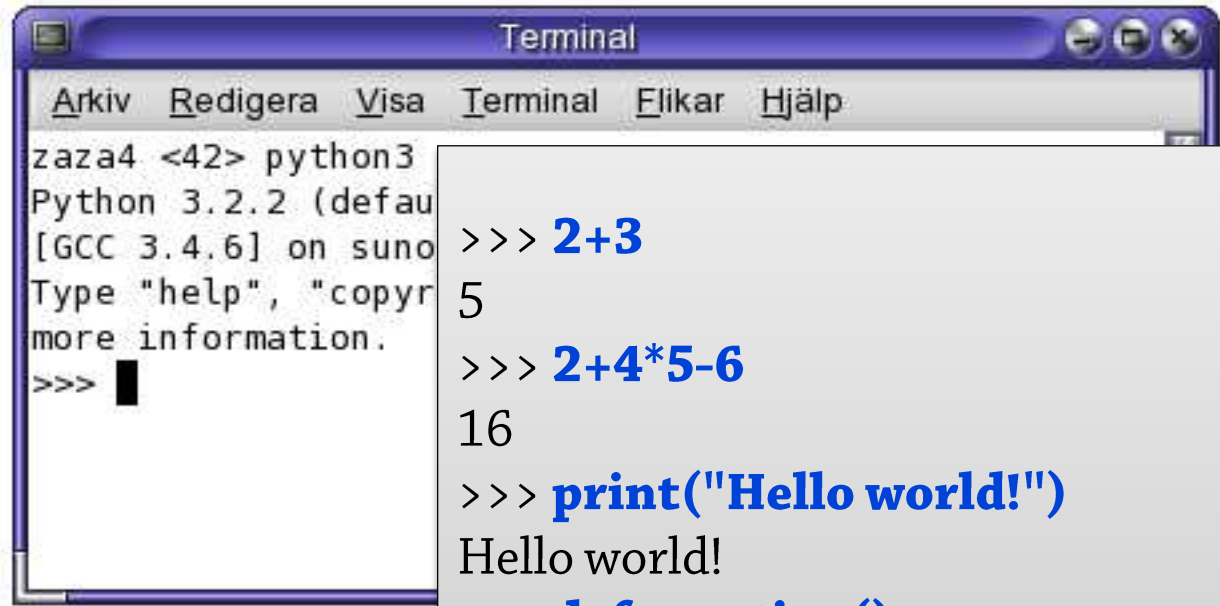
1.4166666666666665

1.4142156862745097

1.4142135623746899

1.414213562373095

- ...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.
```

- Java körs *normalt* med kod sparad 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();  
    }  
}
```

- ...men numera finns också **jshell** (här i verbose-läge):

```
jshell> 2+3  
$1 ==> 5  
| created scratch variable $1 : int
```

```
jshell> 2+4*5+6  
$2 ==> 28  
| created scratch variable $2 : int
```

```
jshell> System.out.println("Hello world!")  
Hello world!
```

```
jshell> void greeting() {  
...> System.out.println("Nobody expects the Spanish Inquisition.");  
...> System.out.println("Our chief weapon is surprise... and fear.");  
...> }  
| created method greeting()
```

```
jshell> greeting();  
Nobody expects the Spanish Inquisition.  
Our chief weapon is surprise... and fear.
```

Har en del smarta funktioner
(tab-komplettering, hjälptexter, ...):
<http://cr.openjdk.java.net/~rfield/tutorial/JShellTutorial.html>

Python och Java: Skillnader i yttlig struktur

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 */
```


Fil: Newton.py

Fil: Newton.java

```
public class Newton {
```

All kod måste ligga i en klass...

Det som ingår 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

```
}
```

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*)

```
}
```

Fil: Newton.py

```
def find_root():
```

Fil: Newton.java

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

Klamrar runt metodens kod: { ... }

**Namnstandard:
storBokstavUtomFörstaOrdet**



Syntax: Strängar, utskrifter



Fil: Newton.py

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

Python-strängar:
"Hello" eller 'Hello'

Jämförelse: str1 == str2

Fil: Newton.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'**

```
    }  
}
```

Syntax: Semikolon, indentering



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 läsbarhet
Ignoreras av språket**

}

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

Fil: Newton.py

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

Fil: Newton.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)**

```
}  
}
```

Deklarera variabler



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 deklareras:
"Här kommer en ny variabel"
(annars säger kompilatorn
att "*guess* finns inte")

Java har explicit typning:
Ange alltid vilken typ en
variabel ska ha
(*double* = "decimaltal")

deklarera = declare

Fil: Newton.py

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

Fil: Newton.java

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

**Namnstandard:
storBokstavUtomFörstaOrdet**

```
    }  
}
```


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)  
}
```

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

```
    }  
}  
}
```

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
```

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;
```

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

```
        }  
    }  
}
```

Referens: Operatörer i Python och Java



Python

Räknesätt: + - * / %

Upphöjt till: **

Division avrundad nedåt: //

Jämförelser: == != > < >= <=

Olikhet: <>

Tilldelning: = += -= *= /= %=

Tilldelning: i += 1

Tilldelning: i -= 1

Bitoperatorer: & | ~ << >>

Villkor: and

or

not

Java

Samma

Använd: Math.pow(bas, exponent)

Använd: (int) (x / y)

Samma

Använd !=

Samma

Använd i++

Använd i--

Samma (men finns flera)

Använd: &&

||

!

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)
```

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);  
        }  
    }  
}
```

Utskrift igen...

```
}  
}
```

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!"
```

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!");  
    }  
    ...  
}
```

Sammanfattning: Satser och gruppering



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
klamrar { }

Indentering ändå viktigt
för läsbarhet!

- 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

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*...

```
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:

När ~~variabler~~ har datatyper: I Java och många andra språk

Objektorientering...