Omgivning, runtime och organisation

TDP019 Projekt: Datorspråk
Föreläsning 4

LINKÖPINGS UNIVERSITET

Pontus Haglund
Institutionen för datavetenskap
2024-02-22

Översikt

- Omgivning
- Runtime
- Organisation

En kommentar om eval/exec

- När är det ok att använda eval i Ruby?
  Tänk på uttrycket 1+2*3

```
class ArithmeticExpression
def initialize(expr)
  @expr = expr
end
def evaluate
  return eval(expr)
end
```

En kommentar om eval/exec

- När är det ok att använda eval i Ruby?
  Tänk på uttrycket 1+2*3

```
class ArithmeticExpression
def initialize(expr)
  @expr = expr
end
def evaluate
  return eval(expr)
end
```

En kommentar om eval/exec

- När är det ok att använda eval i Ruby?
  Tänk på uttrycket 1+2*3

```
class ArithmeticExpression
def initialize(expr)
  @expr = expr
end
def evaluate
  return eval(expr)
end
```


```
class BinaryExpression
def initialize(lhs, op, rhs)
  @lhs = lhs
  @op = op
  @rhs = rhs
end
def evaluate
  return eval("#{@lhs.evaluate}
                #{@op}#{@rhs.evaluate}")
end
```

En kommentar om eval/exec

- När är det ok att använda eval i Ruby? Tänk på uttrycket 1+2*3

```
class ArithmeticExpression
def initialize(expr)
  @expr = expr
end
def evaluate
  return eval(expr)
end
```

Inte ok

```
class BinaryExpression
def initialize(lhs, op, rhs)
  @lhs = lhs
  @op = op
  @rhs = rhs
end
def evaluate
  return eval("#{@lhs.evaluate}
              #{@op}#{@rhs.evaluate}")
end
```

Ok!

En kommentar om eval/exec

- När är det ok att använda eval i Ruby? Tänk på uttrycket 1+2*3

```
class ArithmeticExpression
def initialize(expr)
  @expr = expr
end
def evaluate
  return eval(expr)
end
```

Inte ok

```
class BinaryExpression
def initialize(lhs, op, rhs)
  @lhs = lhs
  @op = op
  @rhs = rhs
end
def evaluate
  return eval("#{@lhs.evaluate}
              #{@op}#{@rhs.evaluate}")
end
```

Ok!

Vad är skillnaden?

En kommentar om eval/exec

- När är det ok att använda eval i Ruby? Tänk på uttrycket 1+2*3

```
class ArithmeticExpression
def initialize(expr)
  @expr = expr
end
def evaluate
  return eval(expr)
end
```

```
class BinaryExpression
def initialize(lhs, op, rhs)
  @lhs = lhs
  @op = op
  @rhs = rhs
end
def evaluate
  return eval("#{@lhs.evaluate}
                #{@op}#{@rhs.evaluate}")
end
```

- Det är alltså inte ok att eliminera delar av grammatiken genom att låta Ruby gör jobbet

Object#send att föredra här

- Behöver inte använda eval

```
class BinaryExpression
def initialize(lhs, op, rhs)
  @lhs = lhs
  @op = op
  @rhs = rhs
end
def evaluate
  return @lhs.evaluate.send(@op, @rhs.evaluate)
end
```

Omgivning
Miljö
Environment

LINKÖPINGS UNIVERSITET

Vad är det

- Vad finns tillgängligt var?
- Variabler, typer etc
- Byggs upp av ramar (frames)

När är det relevant

- Nya block
- Nya funktionsanrop
- Alla deklarationsställen
- Alla ställen där värden slås upp

Hur fungerar det?

- Vid varje ny ”nivå” läggs en ny ram till i kedjan
- Kedjan av frames bildar aktuell omgivning

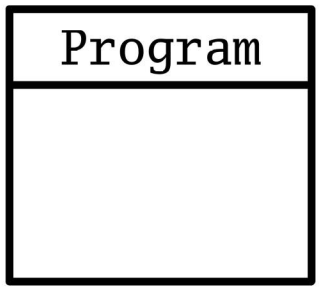
- Att leta efter en variabel → Leta i den här framen(nuvarande/närmsta) och fortsätt utåt
  - (Varierar, finns olika varianter…)

Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```

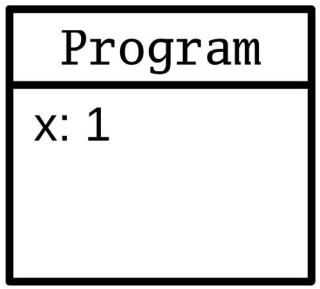
Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```



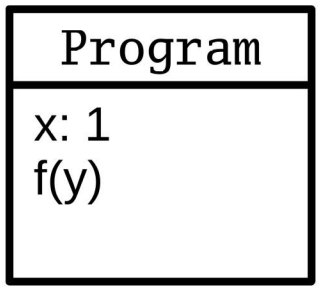
Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```



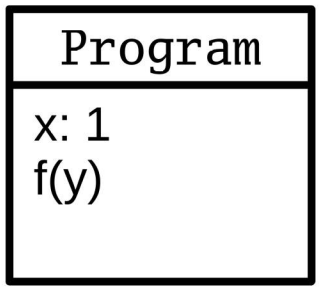
Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```



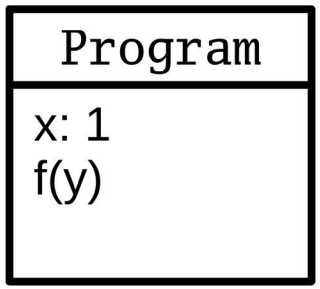
Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```



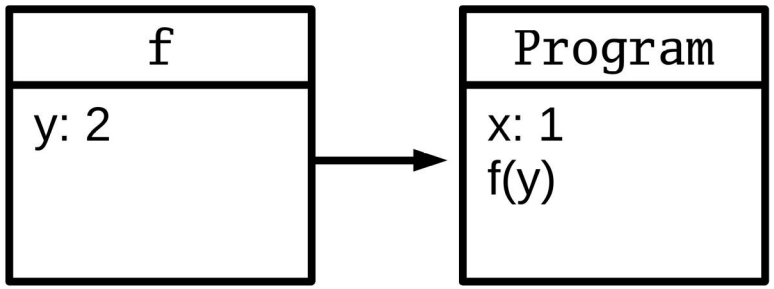
Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```



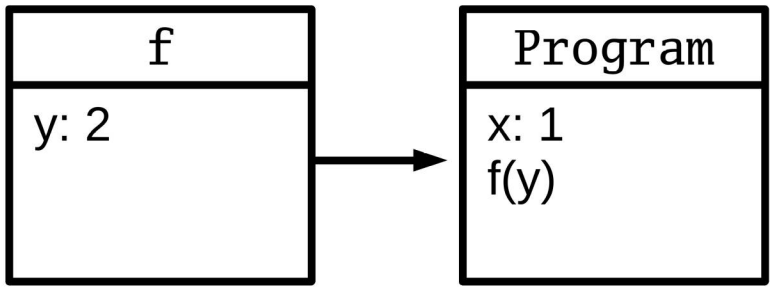
Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```



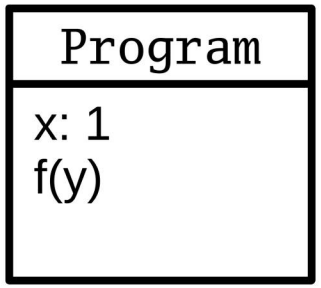
Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```



Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```

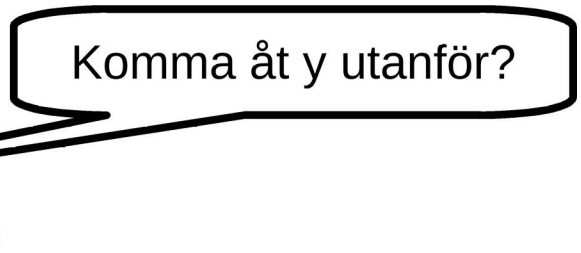


Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```

Omgivning ramar (frames) - exempel

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
end program
```

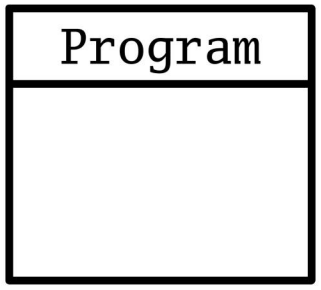


Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```

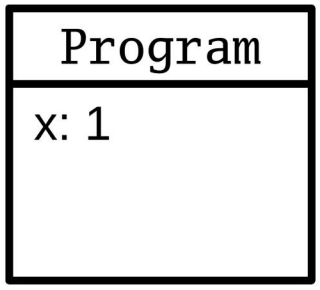
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



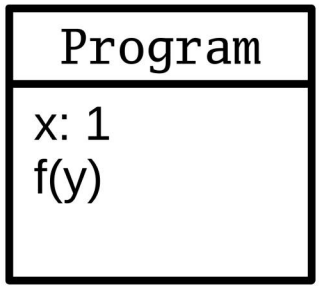
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



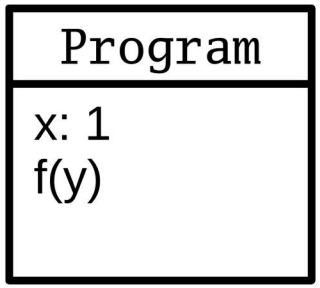
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



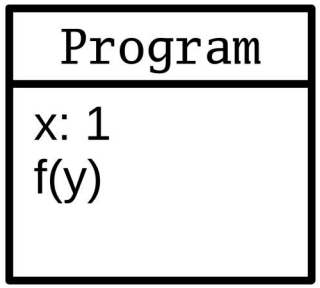
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



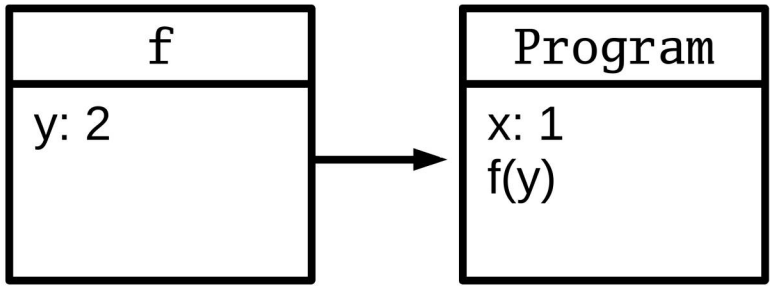
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



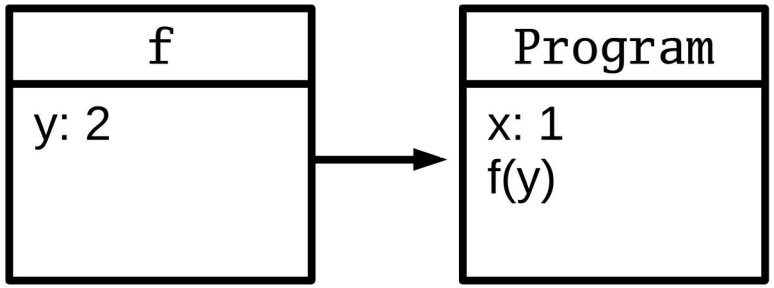
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



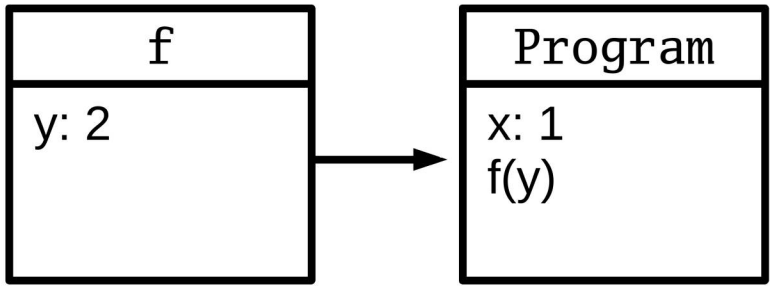
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



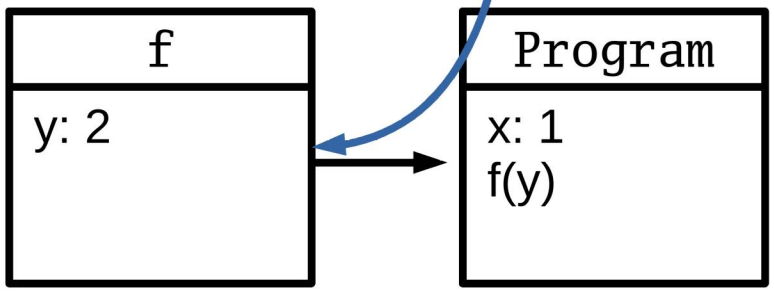
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```

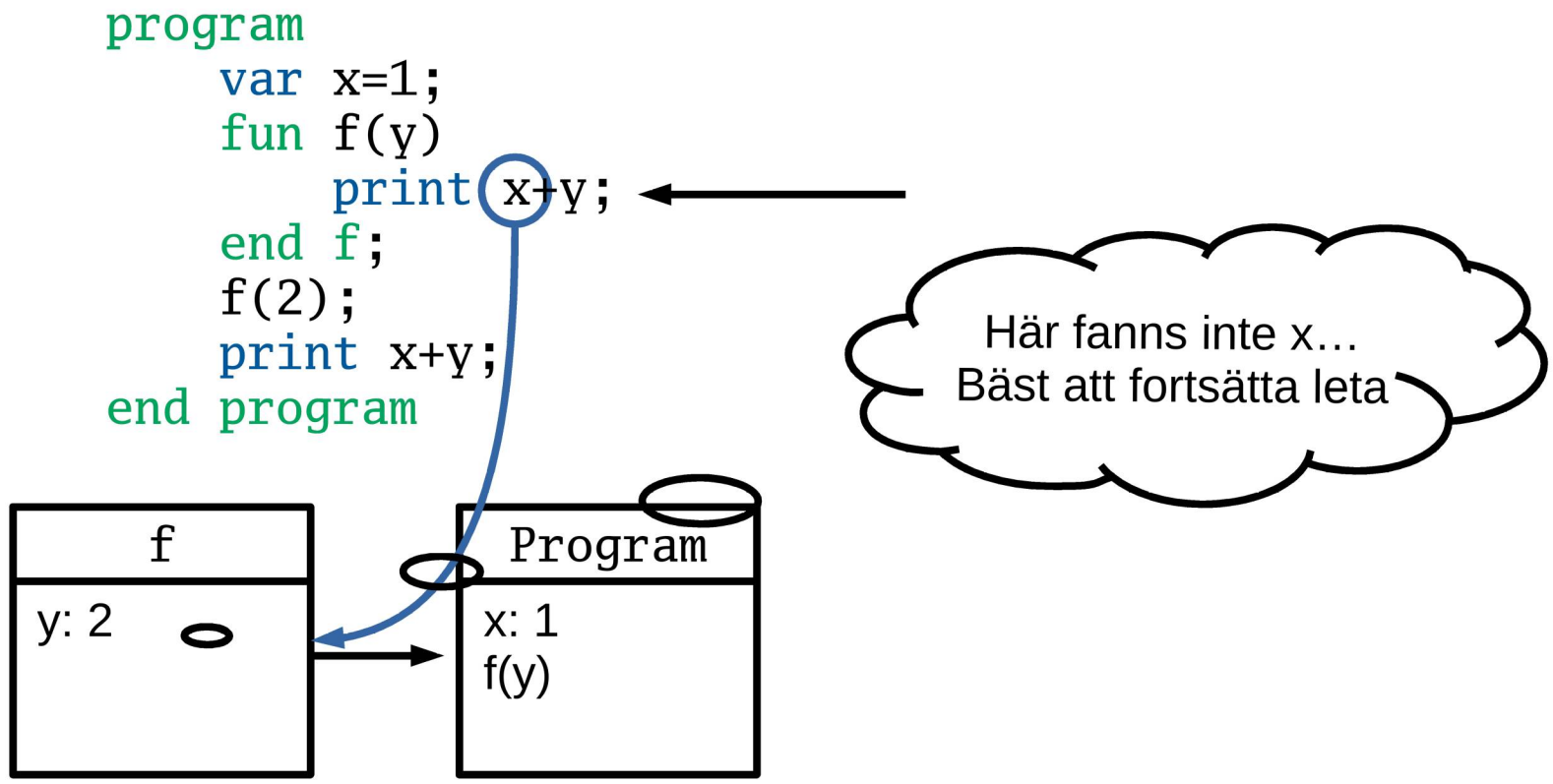


Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```

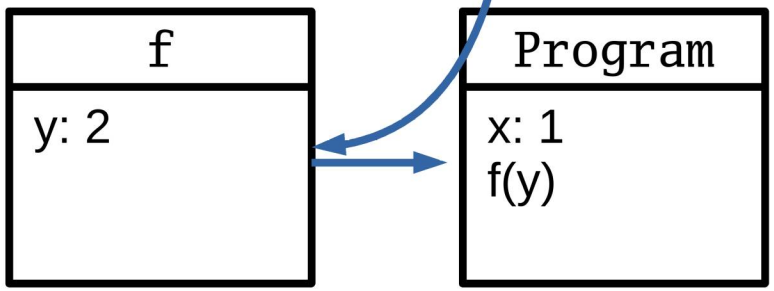


Omgivning ramar (frames)



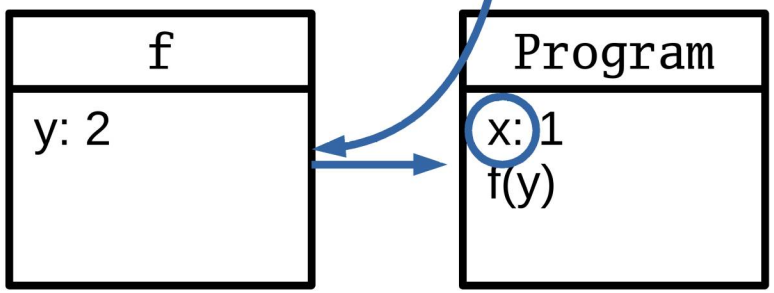
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



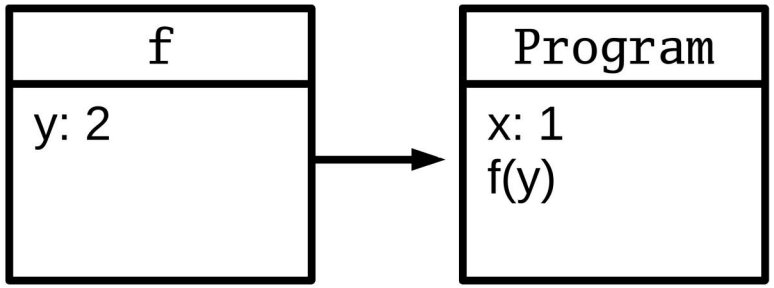
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



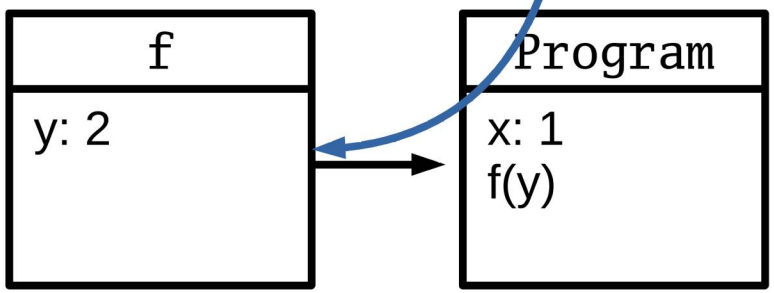
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



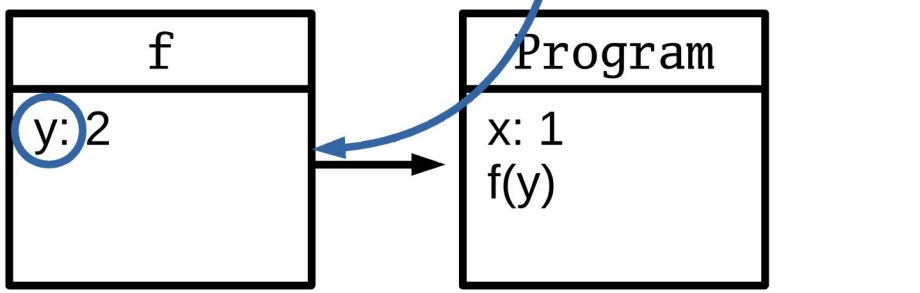
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



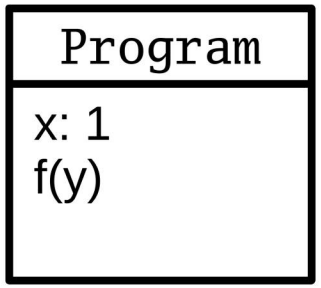
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



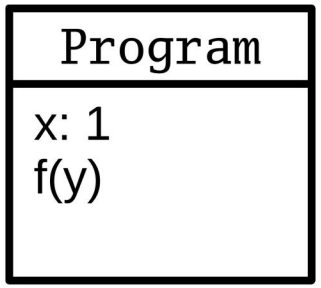
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



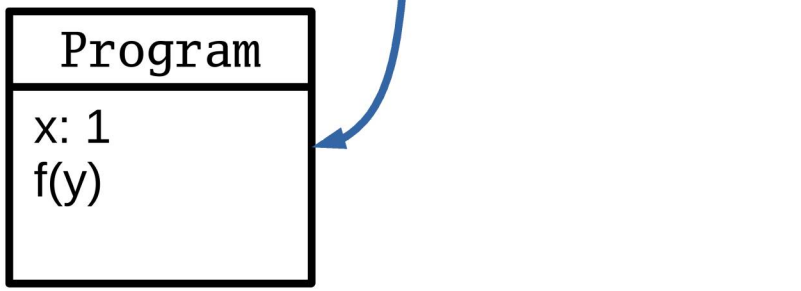
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



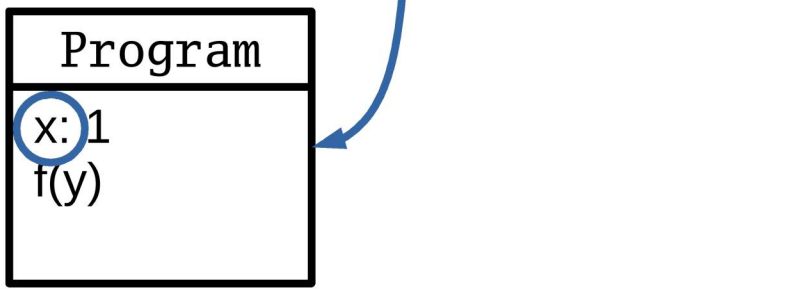
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



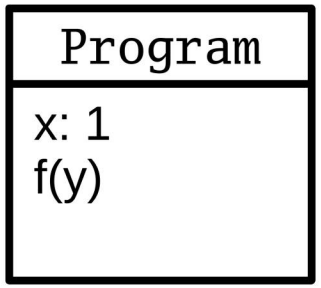
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



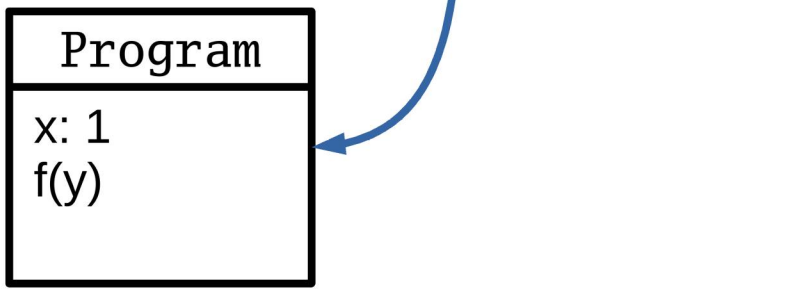
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



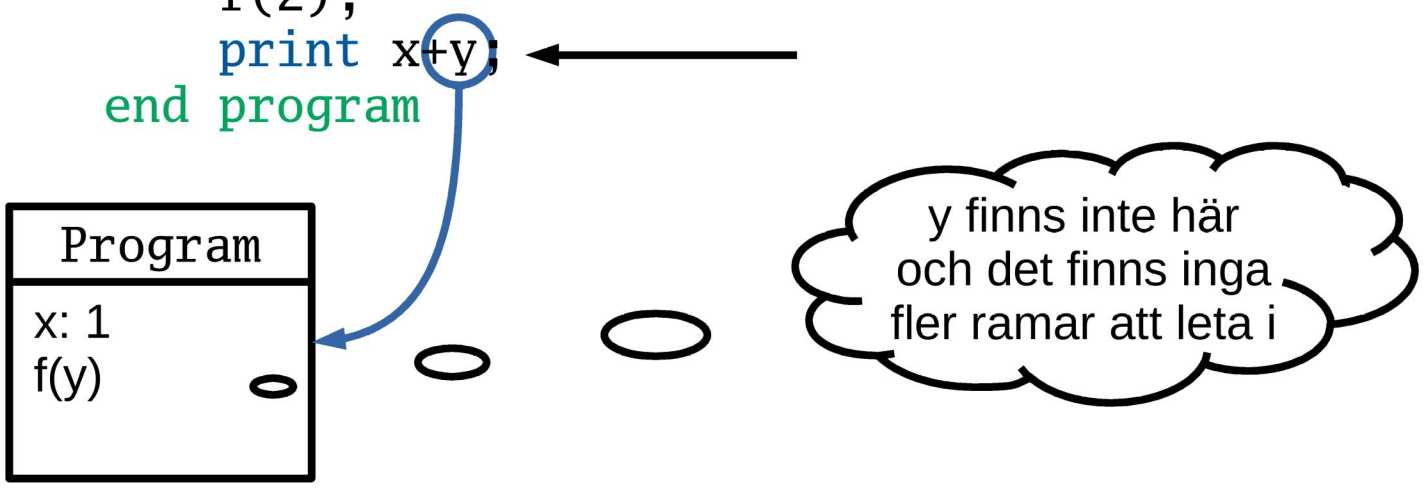
Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



Omgivning ramar (frames)

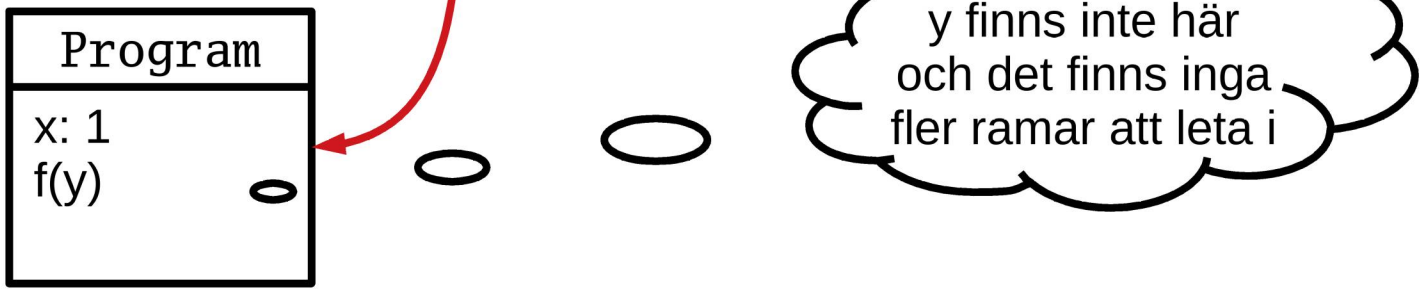
```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```



Omgivning ramar (frames)

```
program
    var x=1;
    fun f(y)
        print x+y;
    end f;
    f(2);
    print x+y;
end program
```

Krashar här ERROR: y is not defined



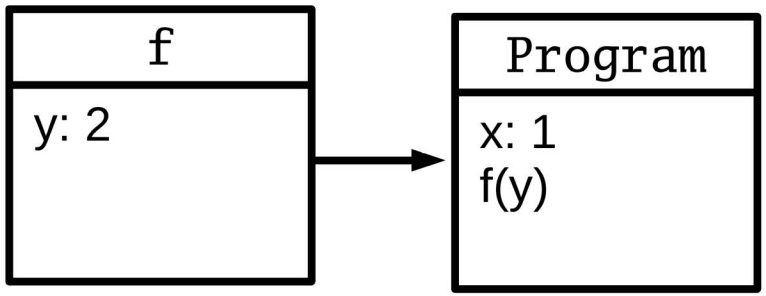
Tankar om implementation

LINKÖPINGS UNIVERSITET

Tankar kring implementation

UPPGIFT:

1) Hur ska vi representera frames
2) Hur lagrar vi våra frames
3) Hur får våra objekt (ast) tillgång till RÄTT saker



Dynamiskt och statiskt scope

LINKÖPINGS UNIVERSITET

dynamiskt/statiskt bindning (scope)

- Dynamiskt scope letar i anropskedjan

- Statiskt scope letar i det scope som tillhör blocket där funktionen definerades

dynamiskt/statiskt bindning (scope)

- Dynamiskt scope letar i anropskedjan

- Statiskt scope letar i det scope som tillhör blocket där funktionen definerades

dynamiskt/statiskt bindning (scope)

- Dynamiskt scope letar i anropskedjan
  - Bash, Dash, Powershell, elisp, logo…

- Statiskt scope letar i det scope som tillhör blocket där funktionen definerades
  - Python, C++, ruby(?)…

dynamiskt/statiskt bindning (scope)

- Dynamiskt scope letar i anropskedjan
  - Bash, Dash, Powershell, elisp, logo…

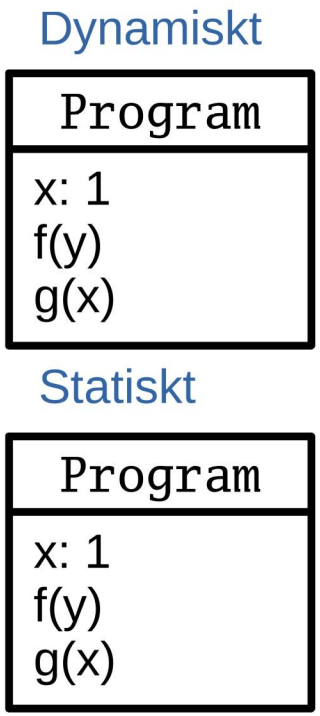
- Statiskt scope letar i det scope som tillhör blocket där funktionen definerades
  - Python, C++, ruby(?)…

Dynamisk/statisk

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    fun g(x)
        return f(3);
    end g;
    print g(2);
end program
```

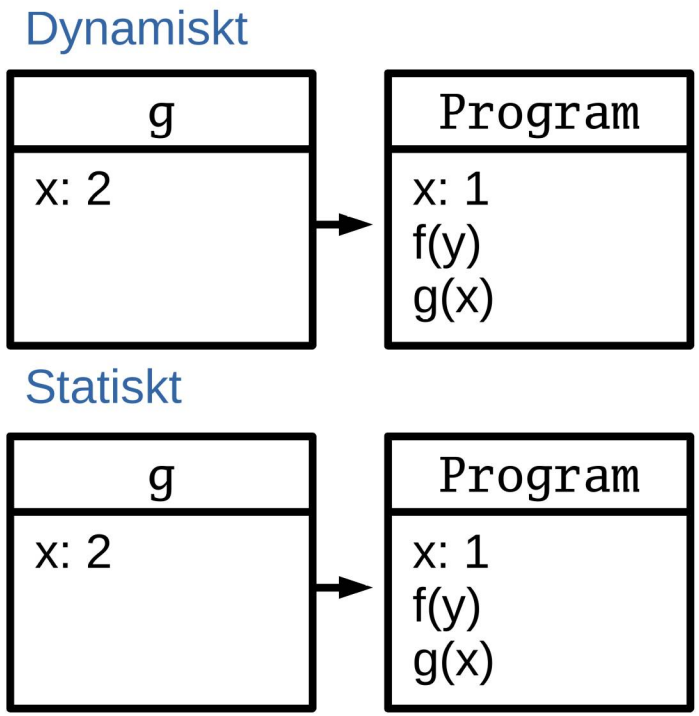
Dynamisk/statisk

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    fun g(x)
        return f(3);
    end g;
    print g(2);
end program
```



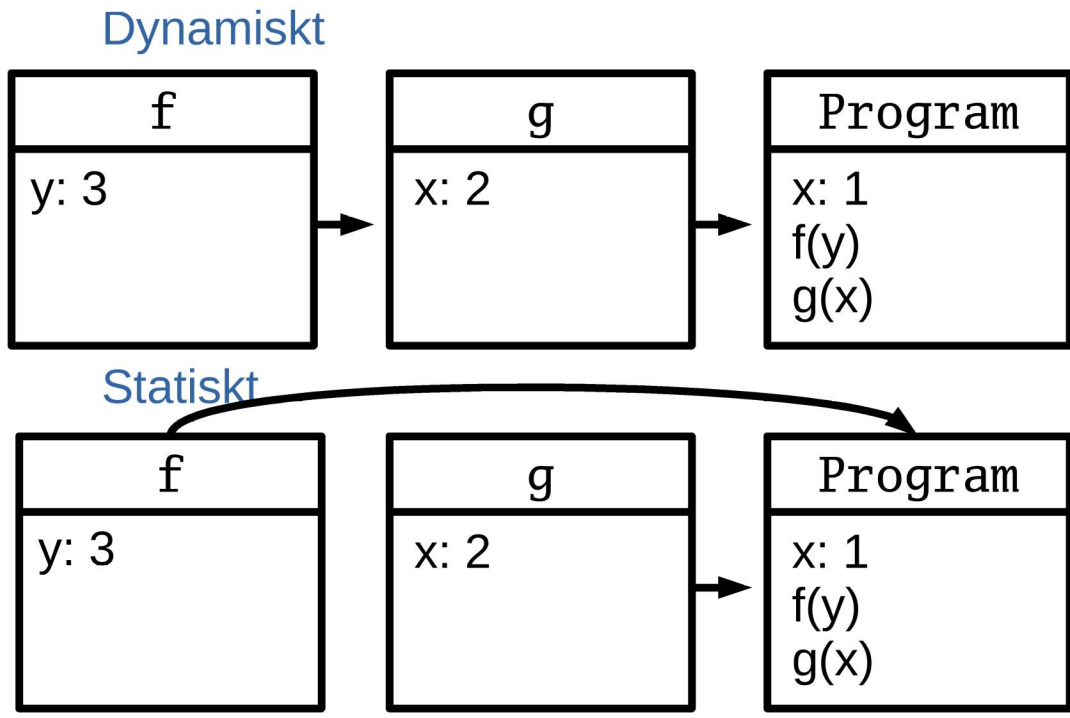
Dynamisk/statisk

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    fun g(x)
        return f(3);
    end g;
    print g(2);
end program
```



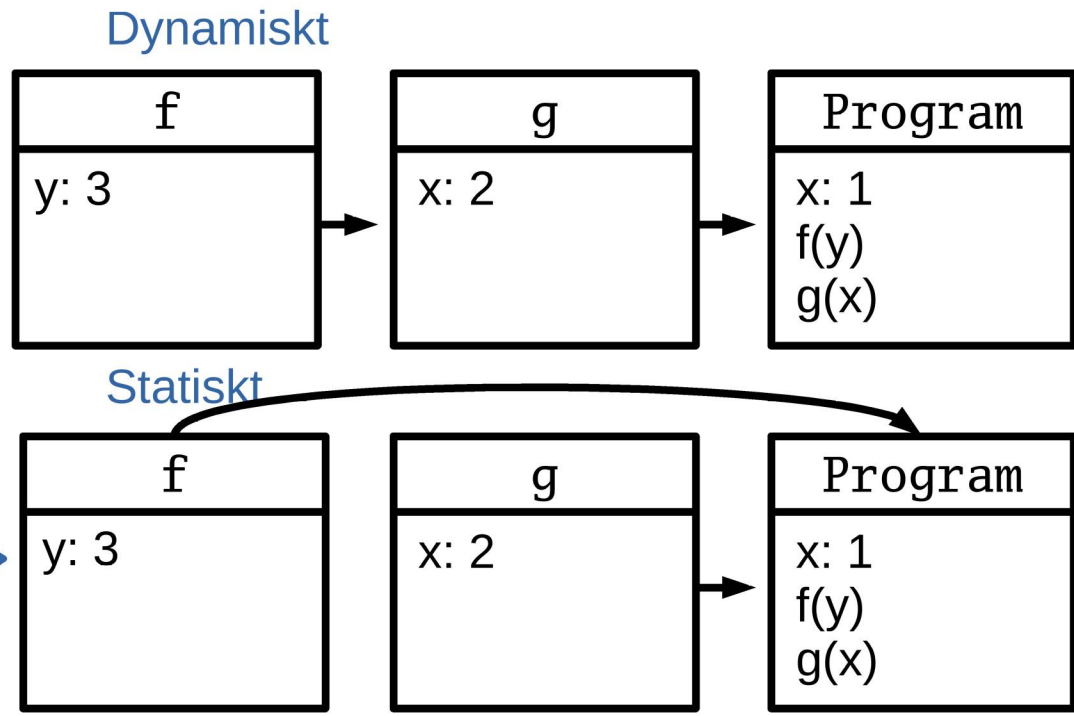
Dynamisk/statisk

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    fun g(x)
        return f(3);
    end g;
    print g(2);
end program
```



Dynamisk/statisk

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    fun g(x)
        return f(3);
    end g;
    print g(2);
end program
```



Denna länk behöver sättas upp vid deklarationstillfället.

dynamiskt/statiskt bindning (scope)

- Dynamiskt scope
  Enkelt att implementera
  Ofta praktiskt i mindre program

- Statiskt scope
  Lite mer tänkade att implementera
  Praktiskt för analys av större program

Runtime?

Runtime

- Vad är runtime?
  Allt som inte är ”kod”
  Semantik

- Vilken del är det i ert språk
  När det faktiska trädet av noder exekveras.
  Allt som händer som ett resultat/med anledning av det

Runtime och REPL (Read Evaluate Print Loop)

- För dessa språk måste runtime leva mellan varje rad
- Inte svårt men något att ta hänsyn till

Att lagra och anropa ”kod”

LINKÖPINGS UNIVERSITET

Att lagra och anropa kod

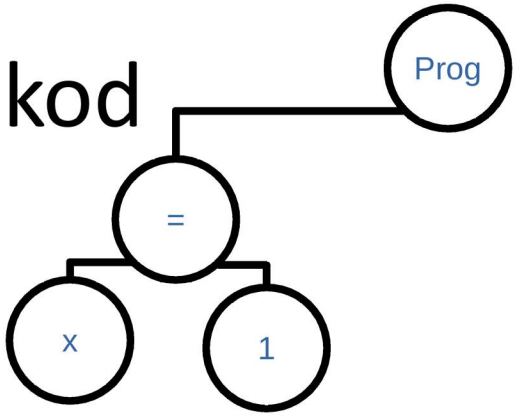
- Hur kan man lagra och anropa kod
- Exempelvis funktioner

Lagra och anropa kod


```
program ←
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

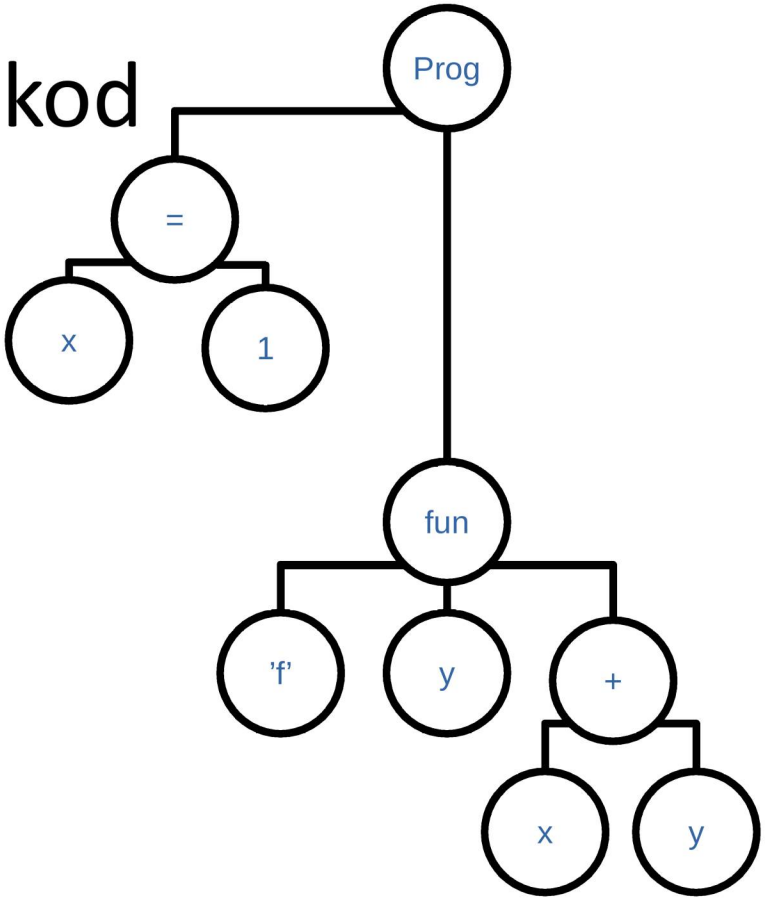
Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



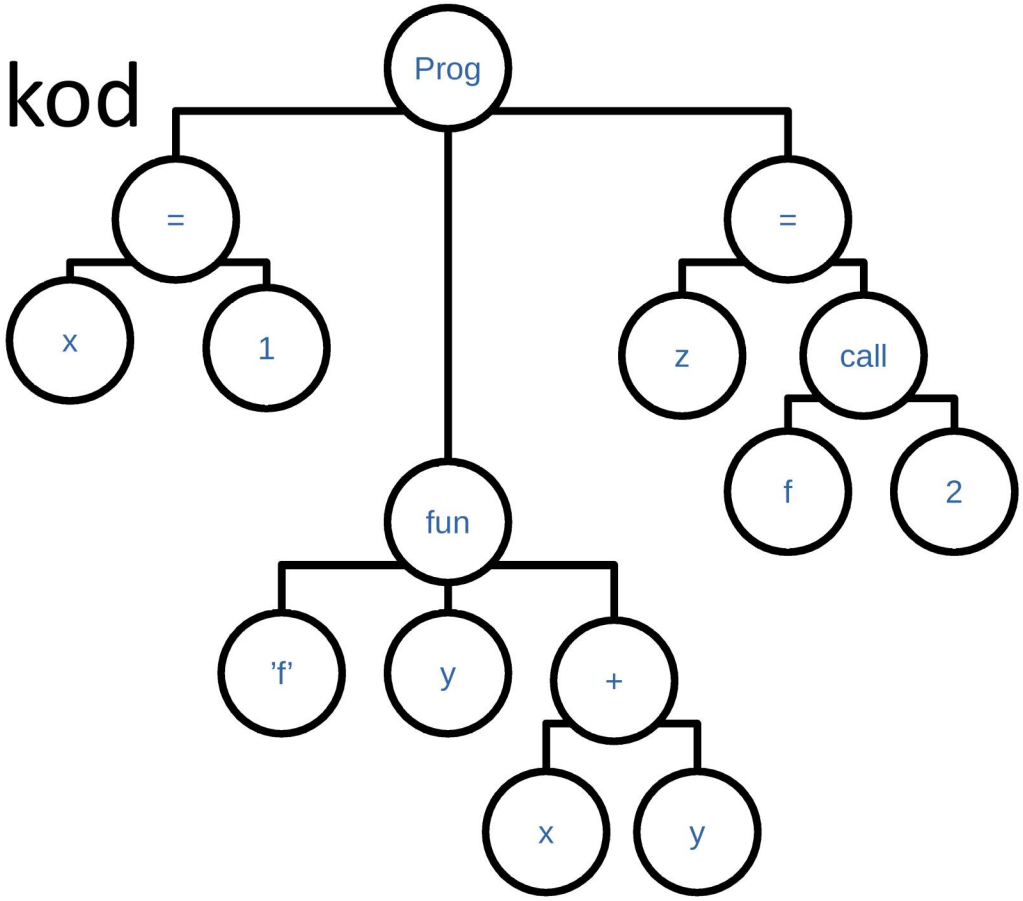
Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



Lagra och anropa kod

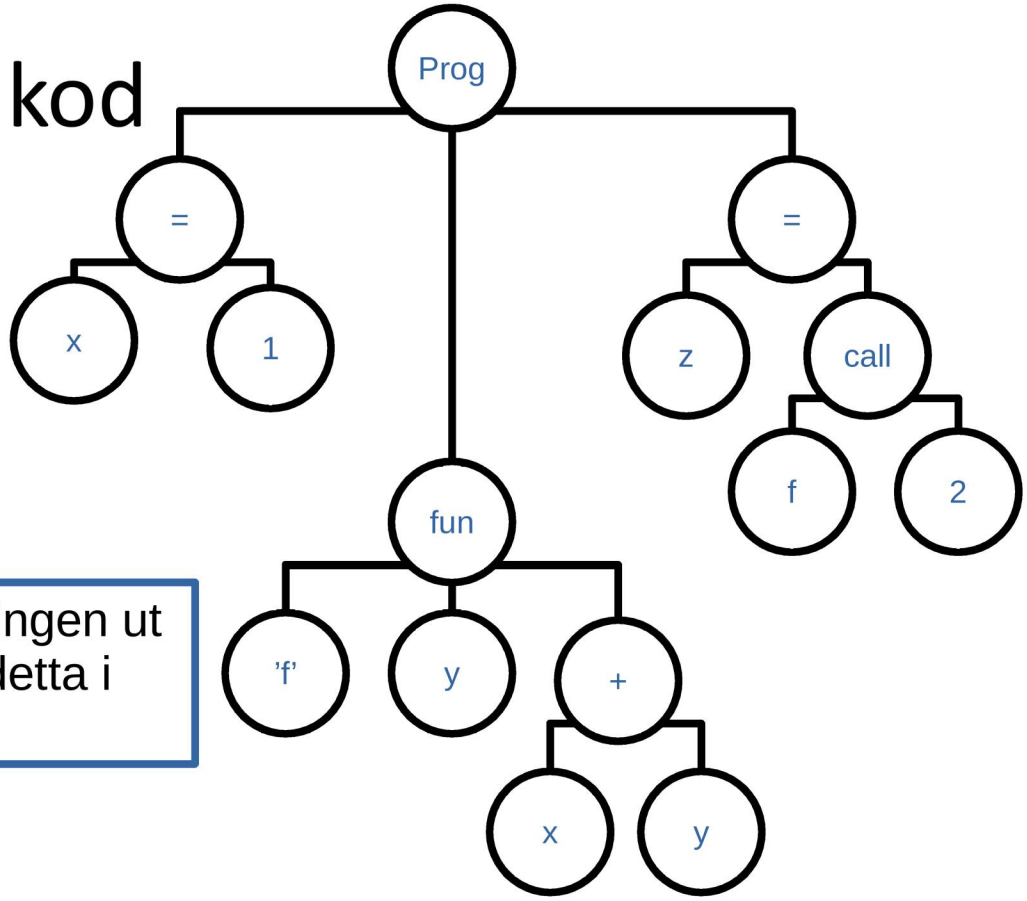
```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

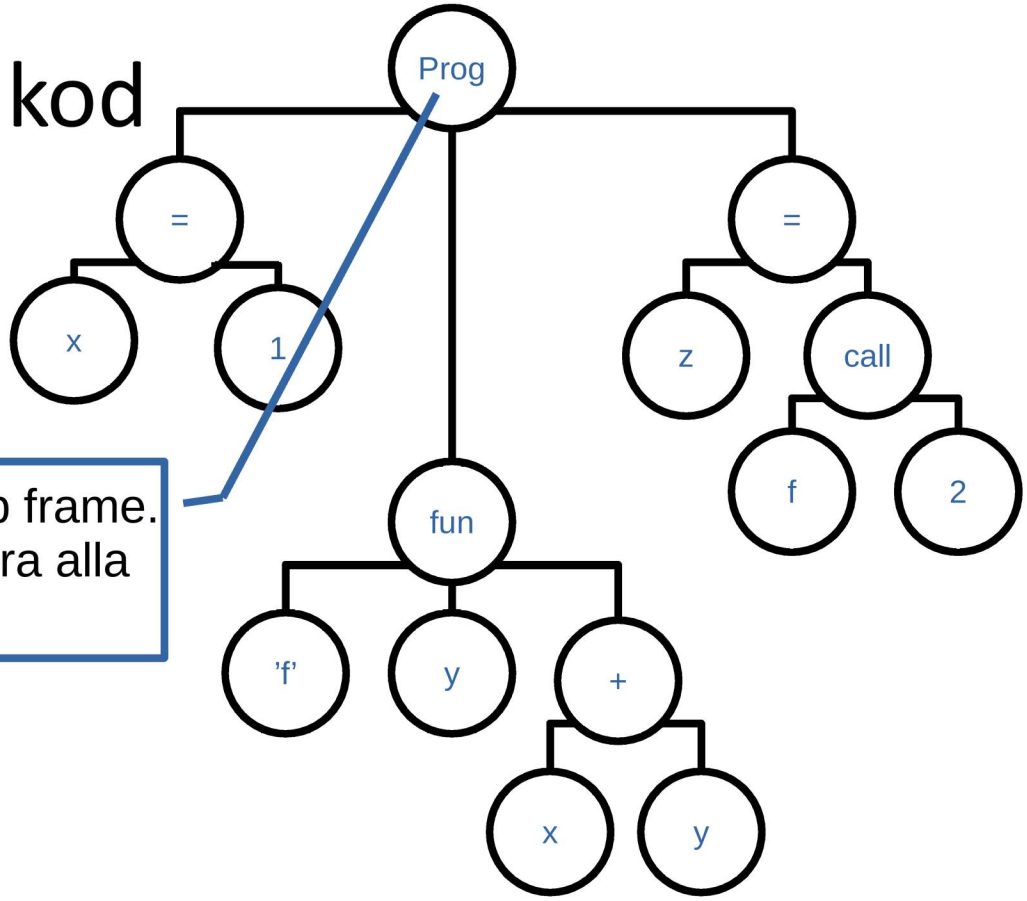
Så ser inläsningen ut men hur blir detta i runtime?

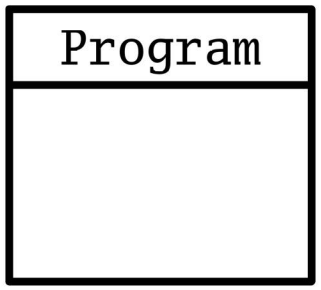


Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

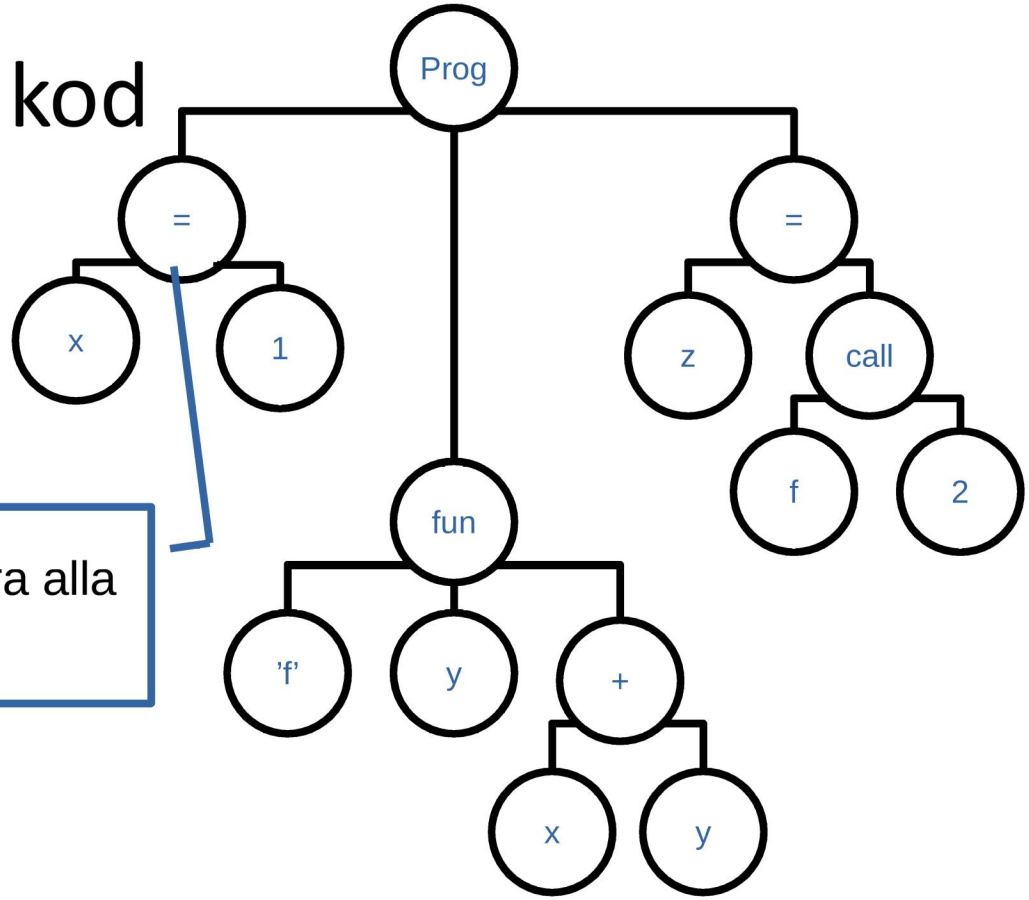
Sätt upp frame. Utvärdera alla grenar.



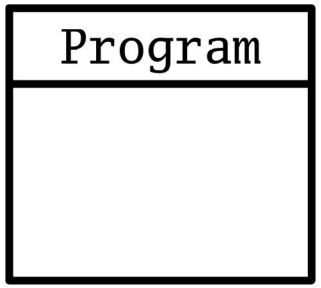


Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



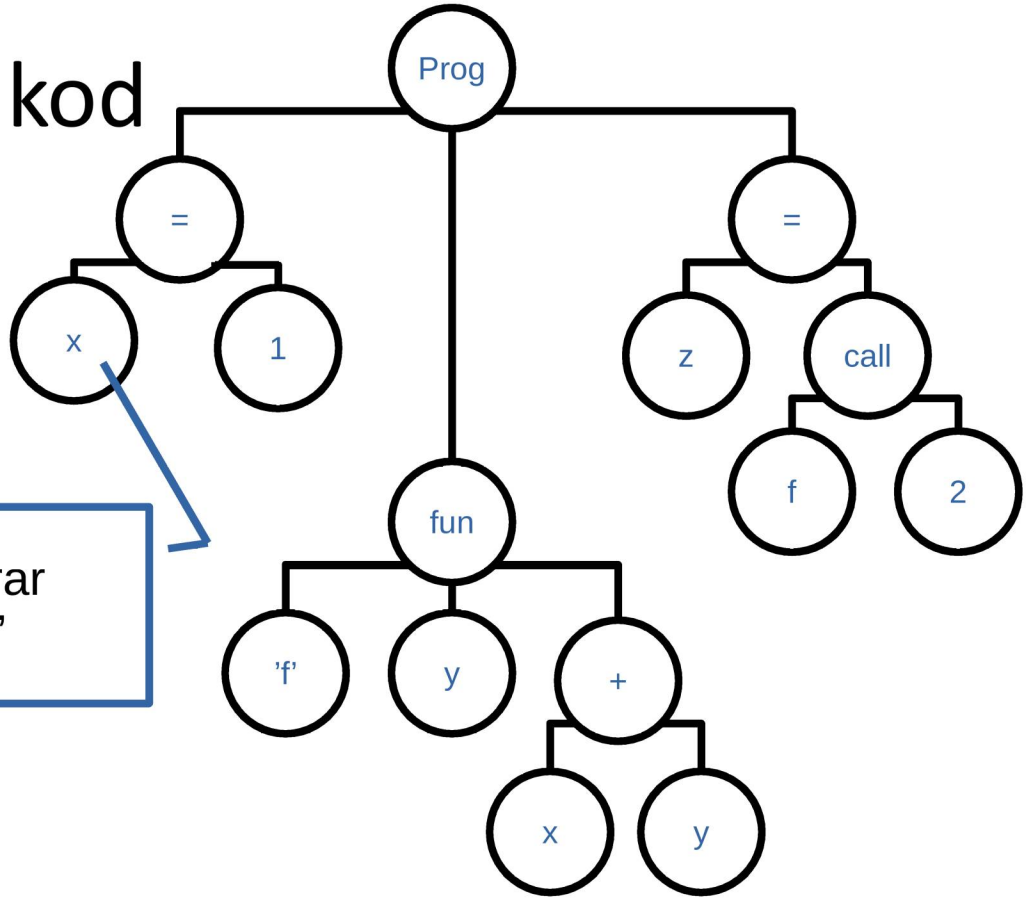
Utvärdera alla grenar.

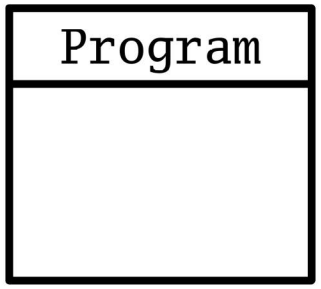


Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

Returnerar namn ”x”

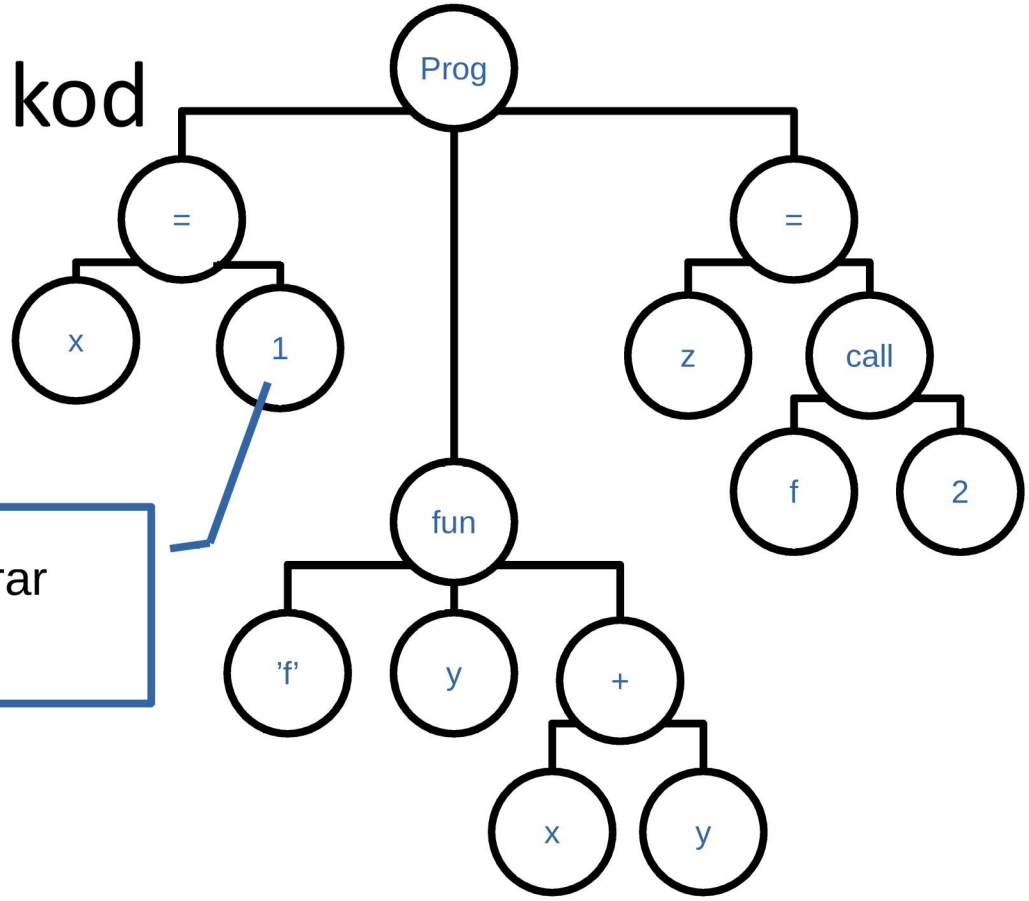


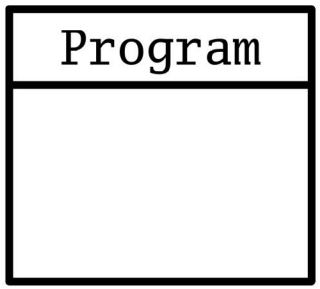


Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

Returnerar heltal 1

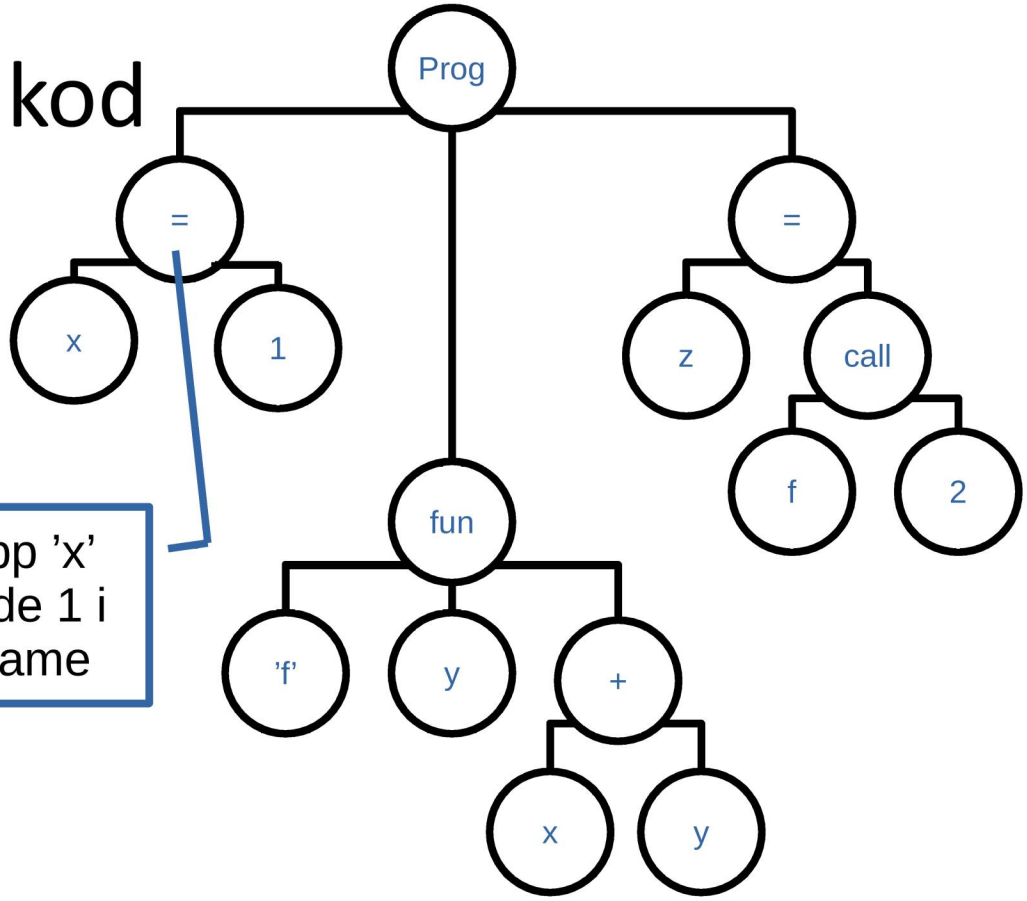




Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

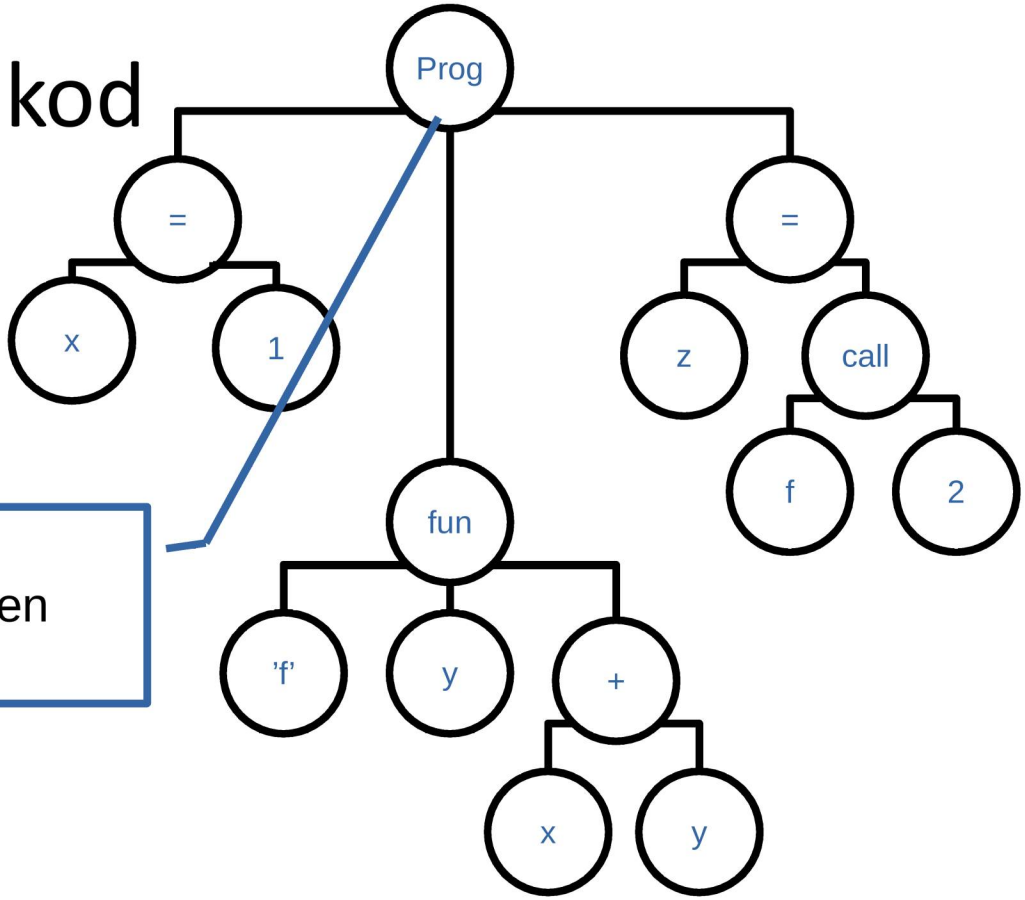
Sätter upp 'x' med värde 1 i aktuell frame



Program
x: 1

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

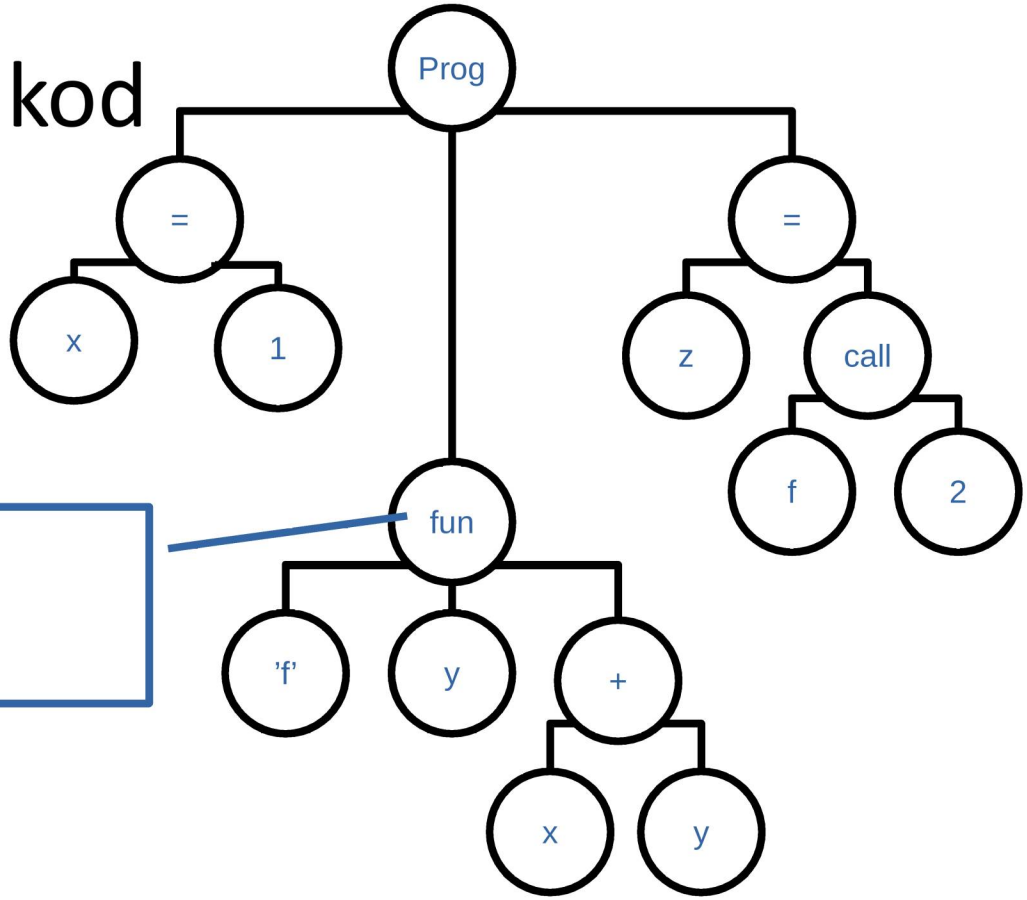


Nästa gren

Program
x: 1

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

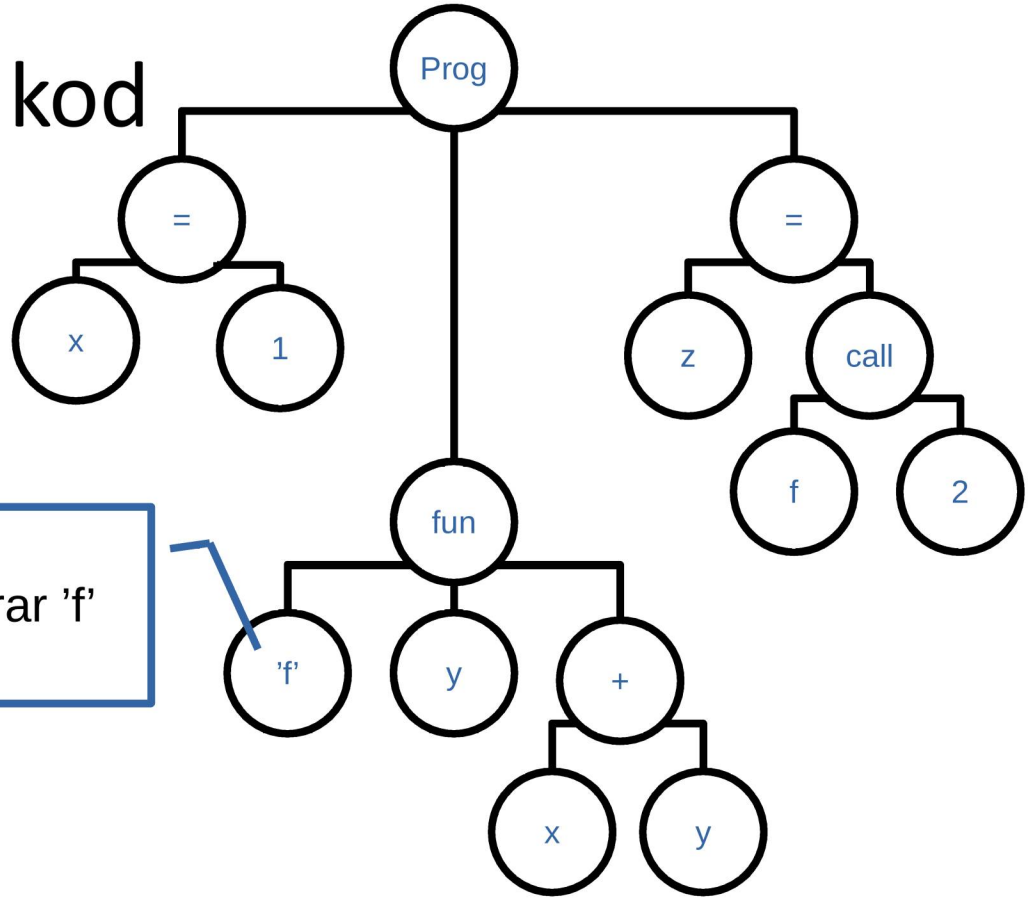


Program
x: 1

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

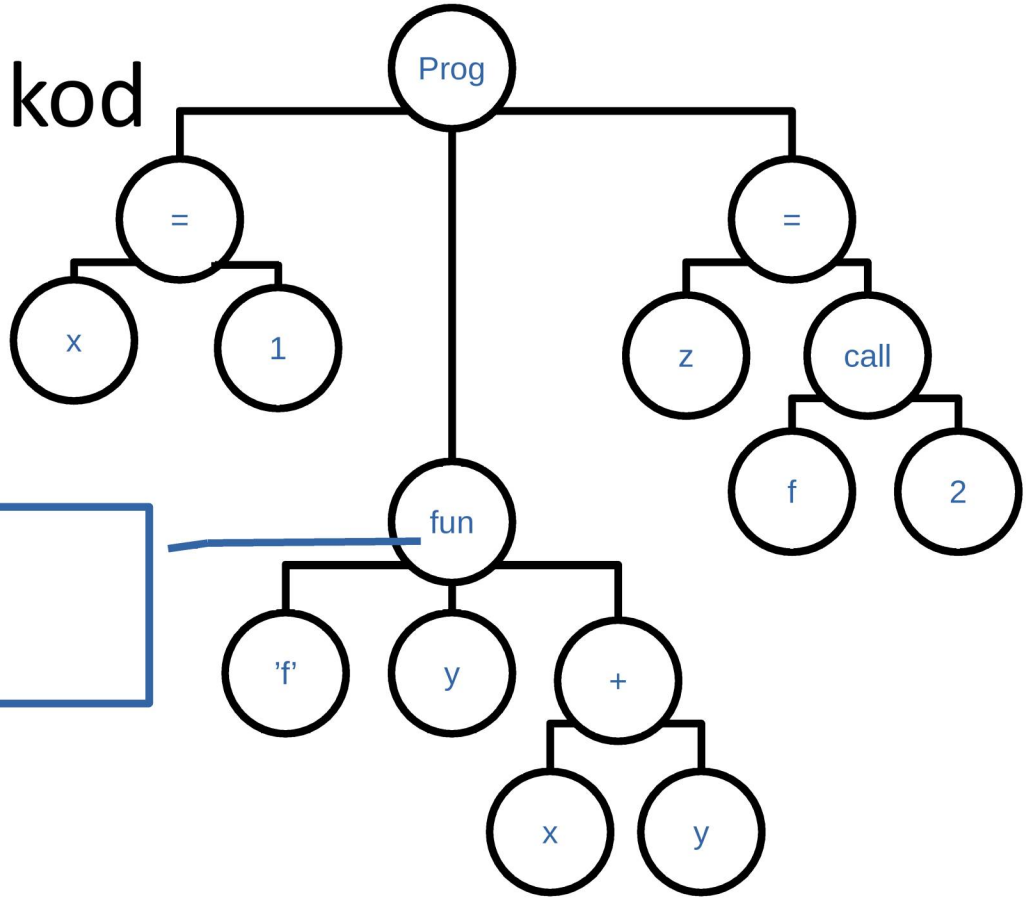
Returnerar ’f’



Program
x: 1

Lagra och anropa kod

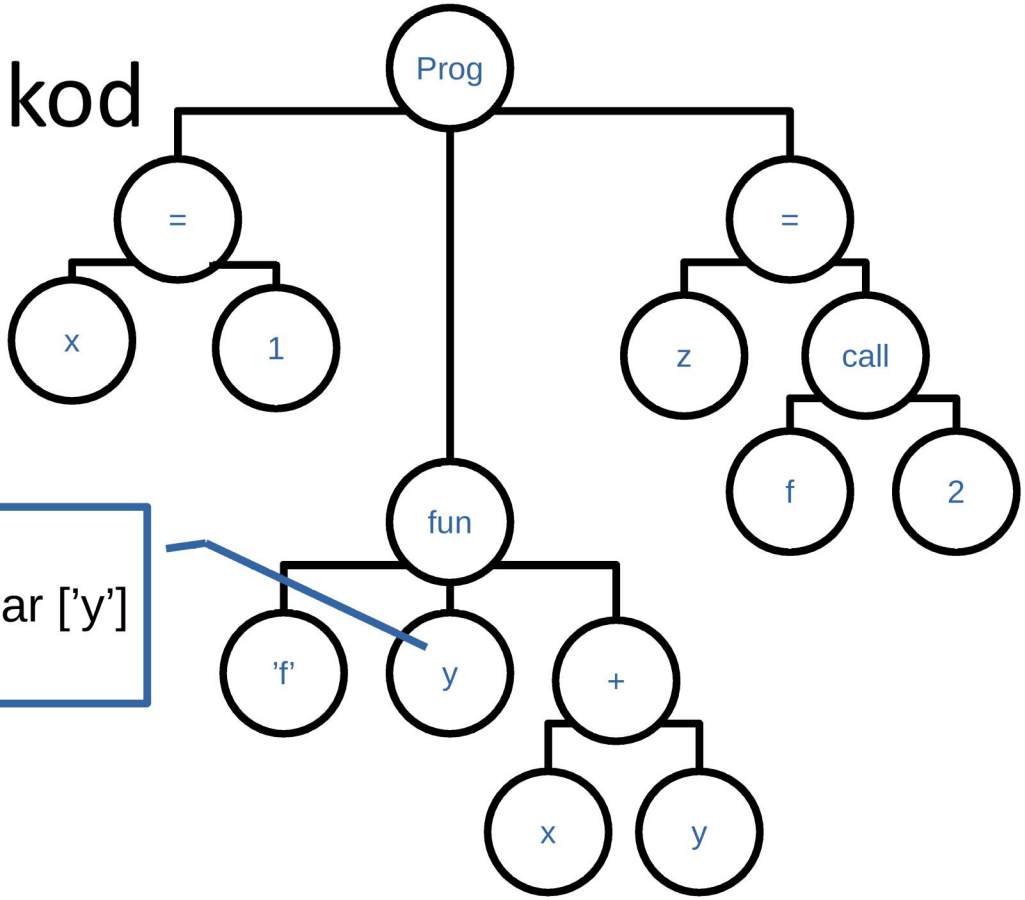
```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



Program
x: 1

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

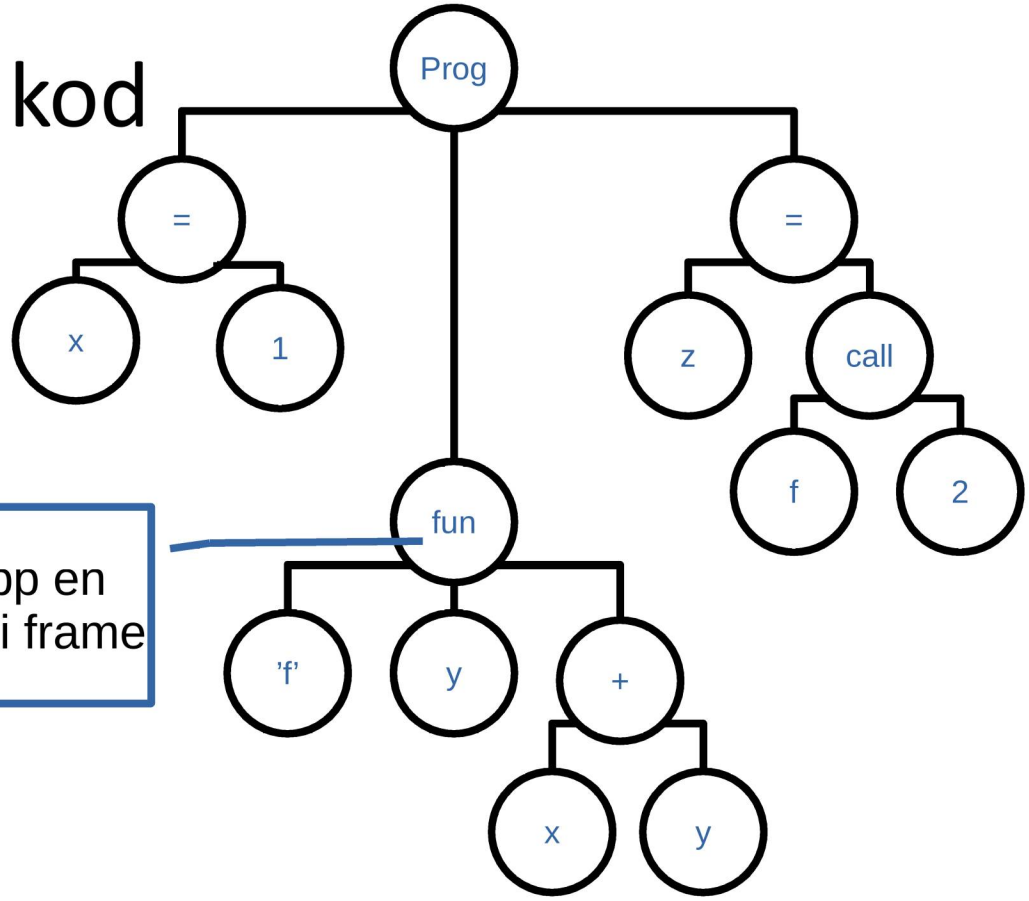


Returnerar ['y']

Program
x: 1

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

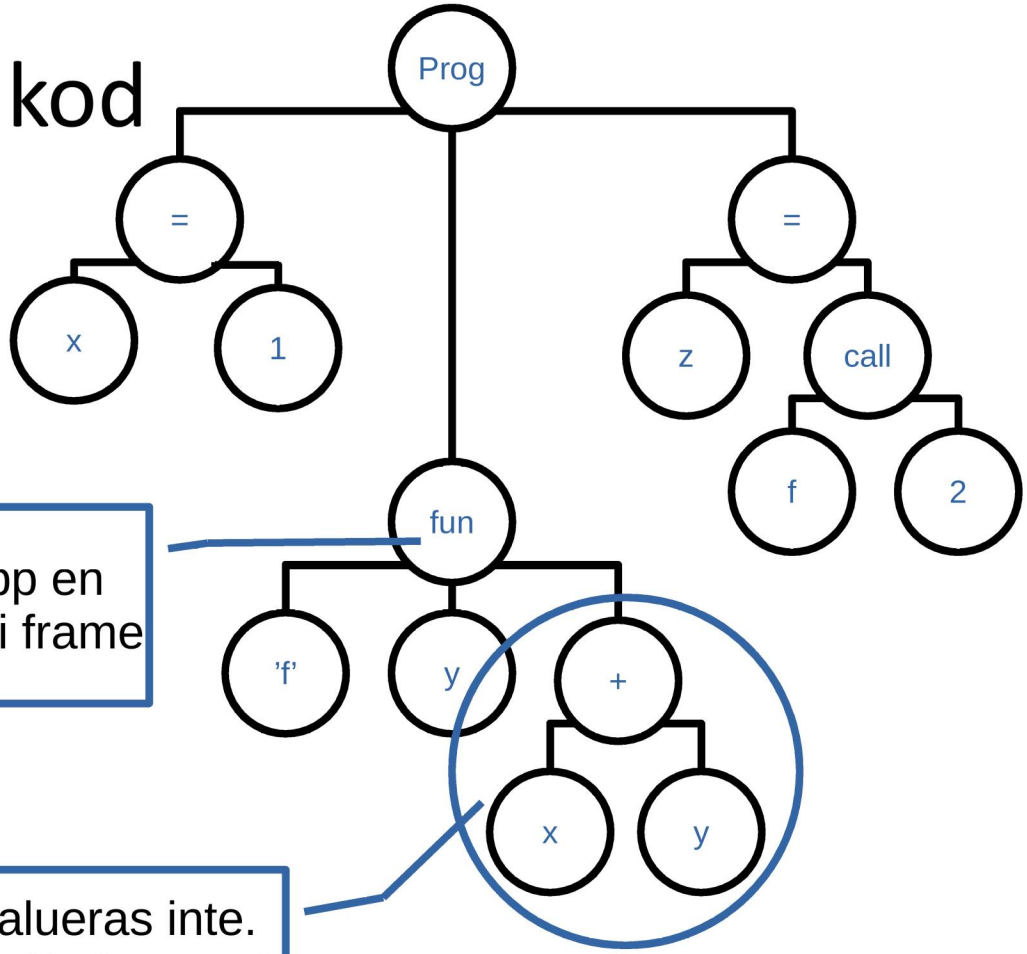


Sätter upp en funktion i frame

Program
x: 1 f(y)

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



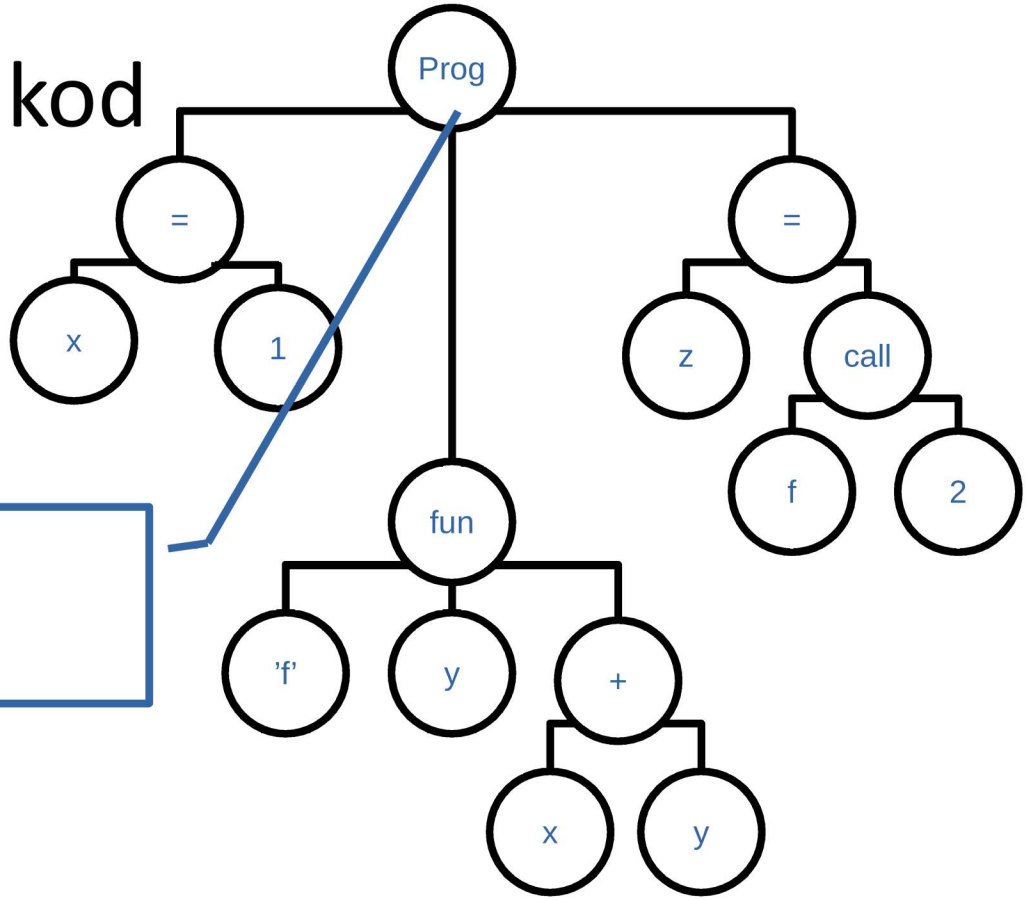
Sätter upp en funktion i frame

Program
x: 1 f(y)

Denna del evalueras inte. Sparas undan för ”senare”

Lagra och anropa kod

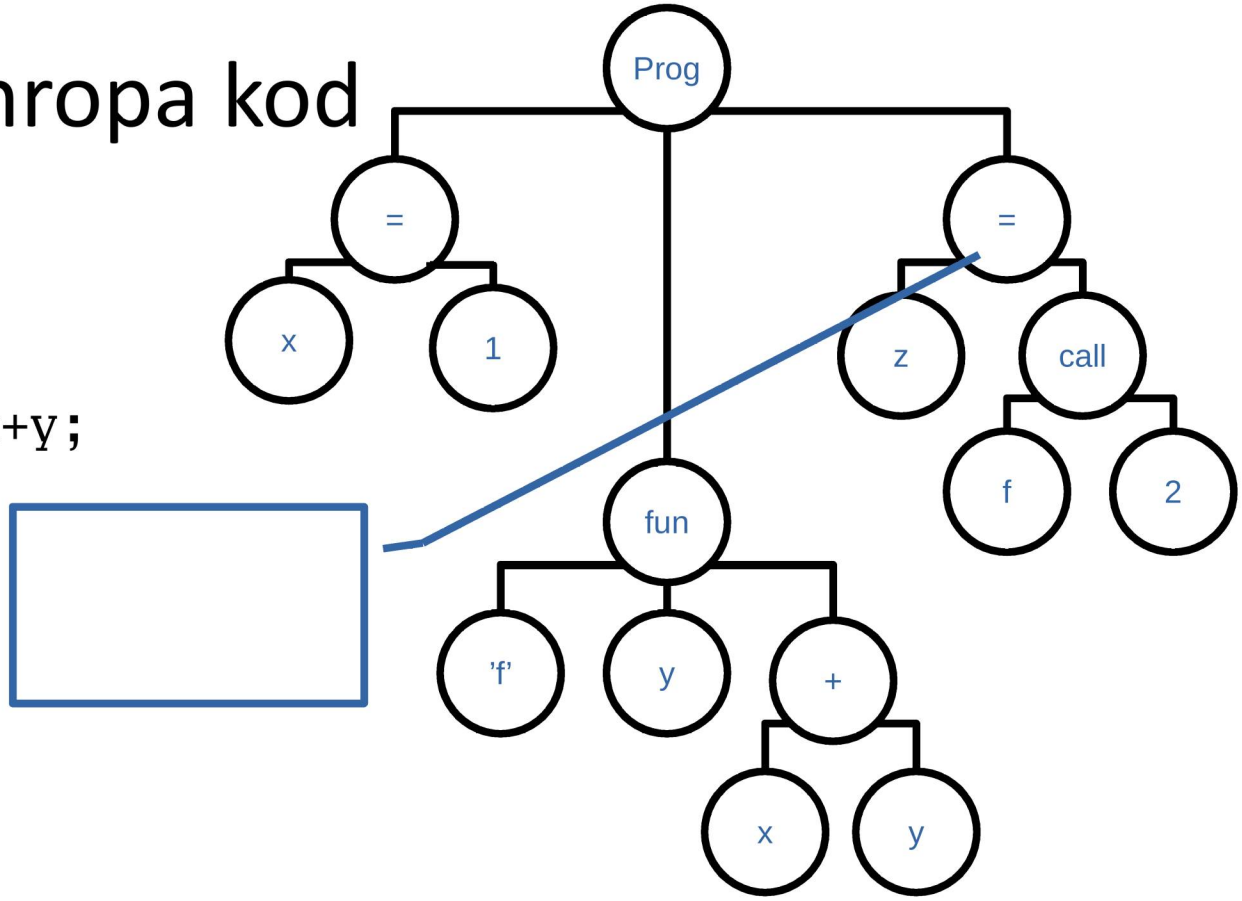
```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



Program
x: 1 f(y)

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

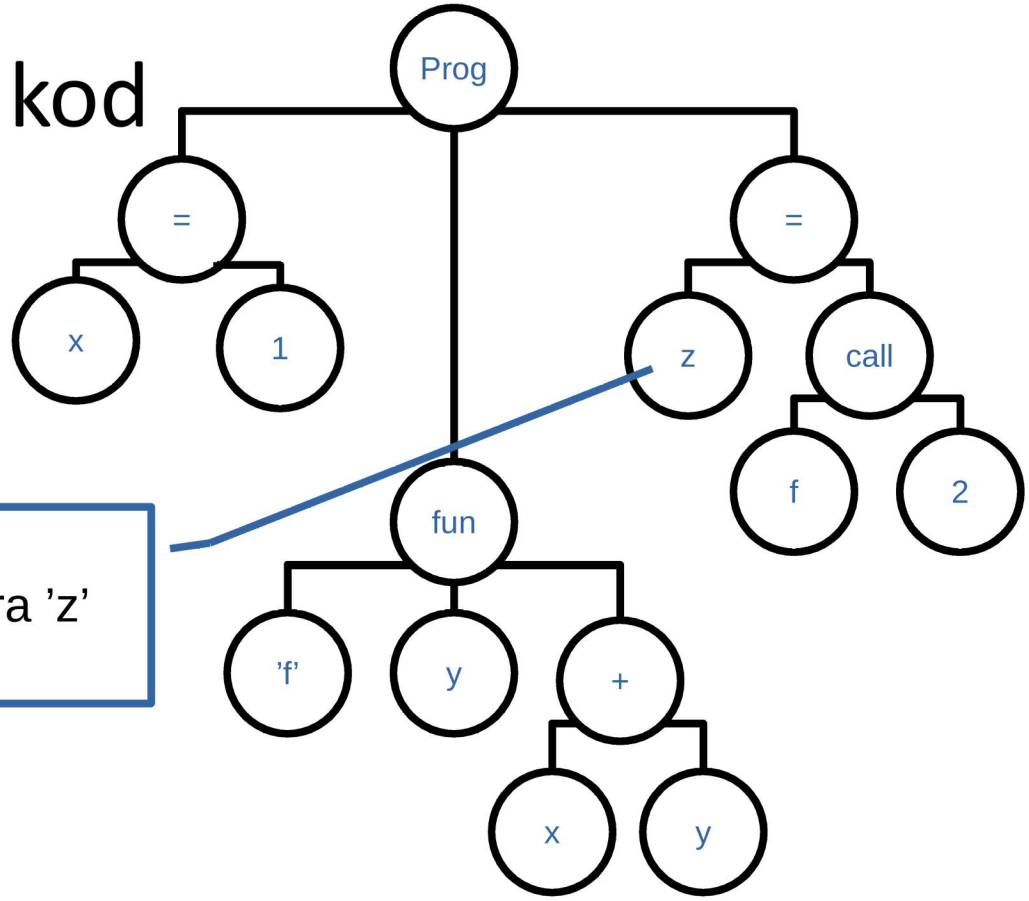


Program
x: 1 f(y)

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

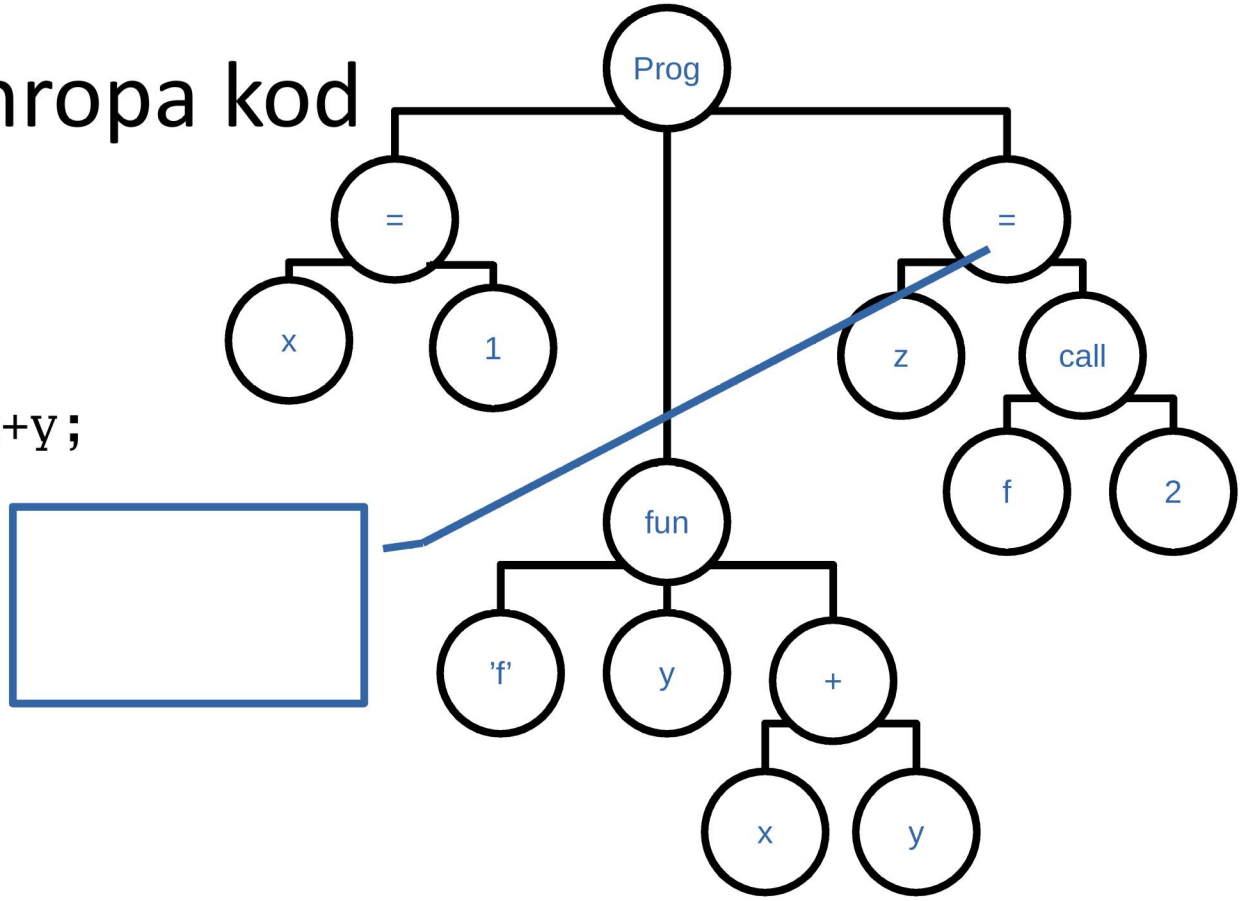
Returnera ’z’



Program
x: 1 f(y)

Lagra och anropa kod

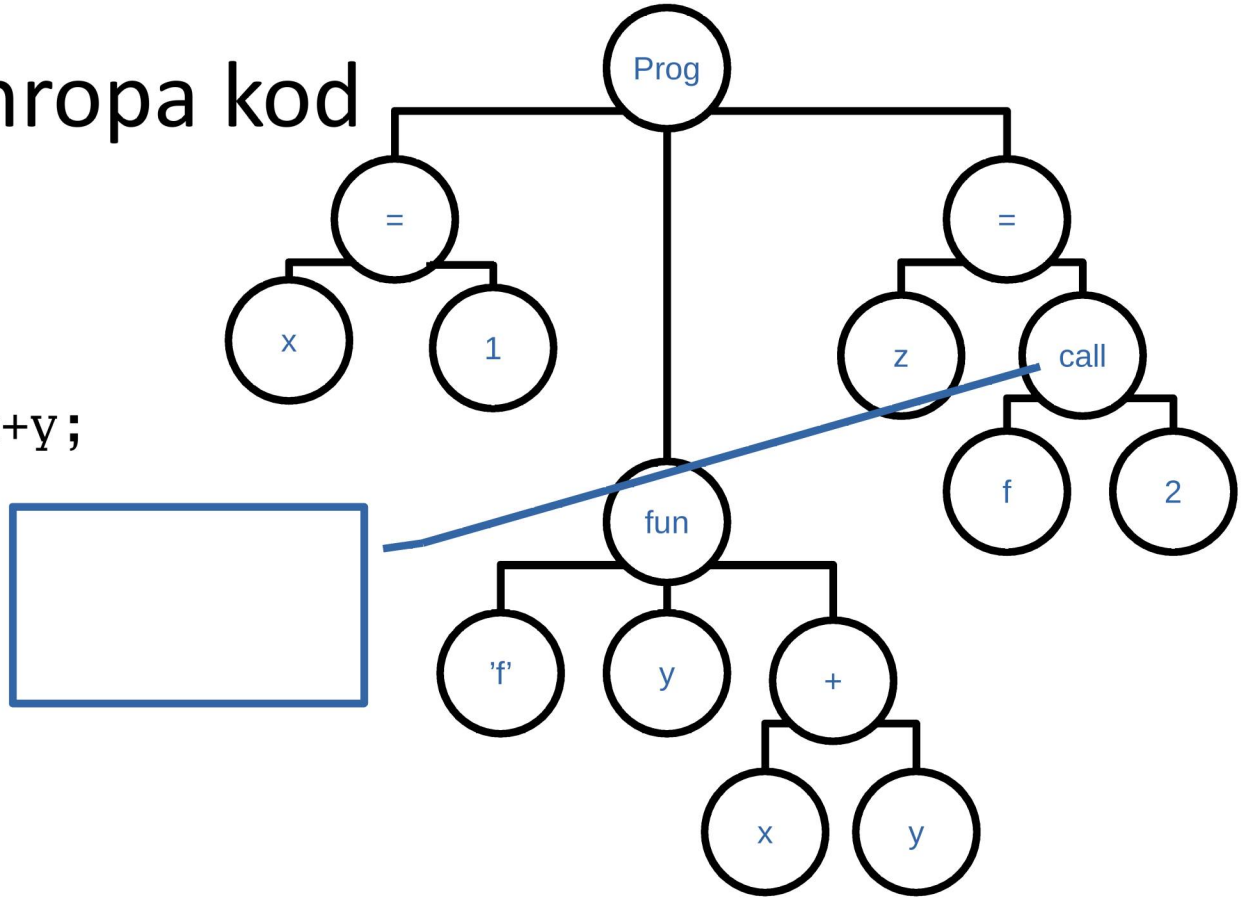
```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



Program
x: 1 f(y)

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

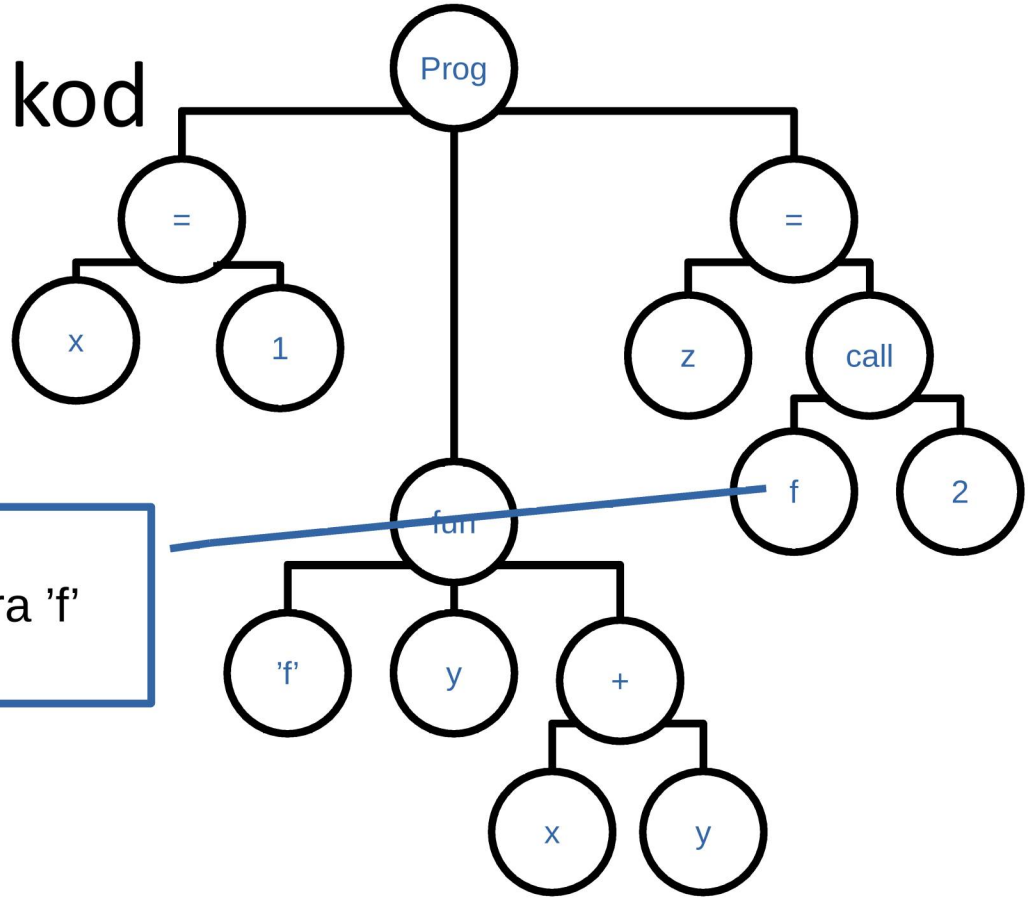


Program
x: 1 f(y)

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

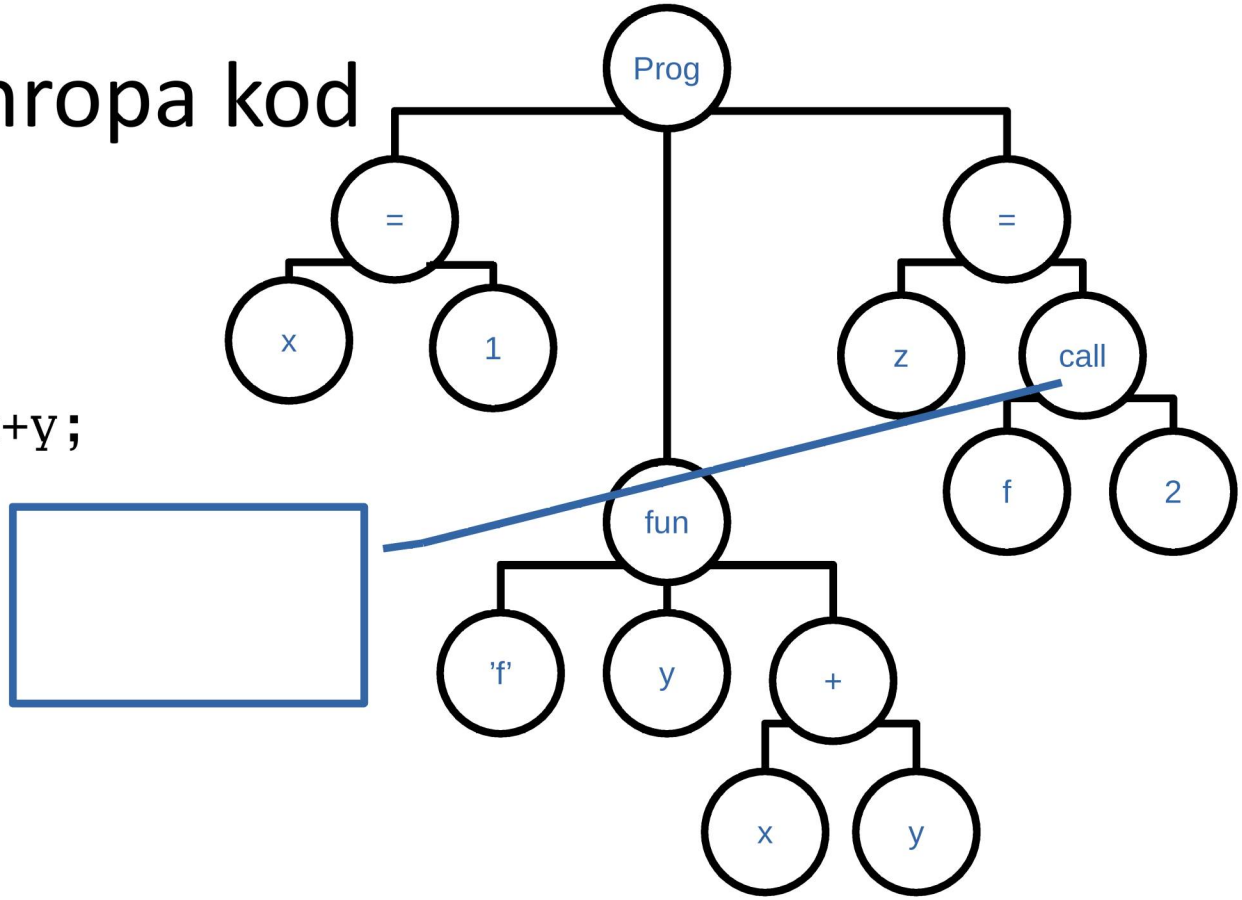
Returnera ’f’



Program
x: 1 f(y)

Lagra och anropa kod

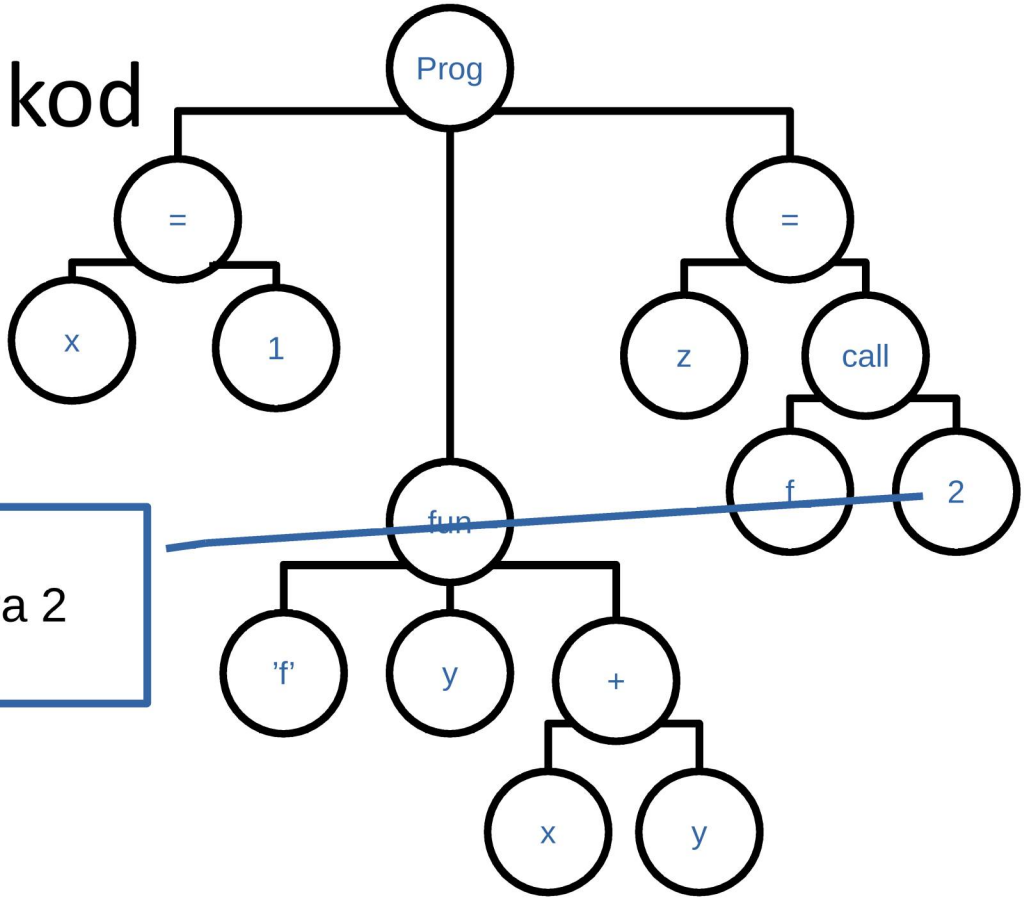
```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



Program
x: 1 f(y)

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

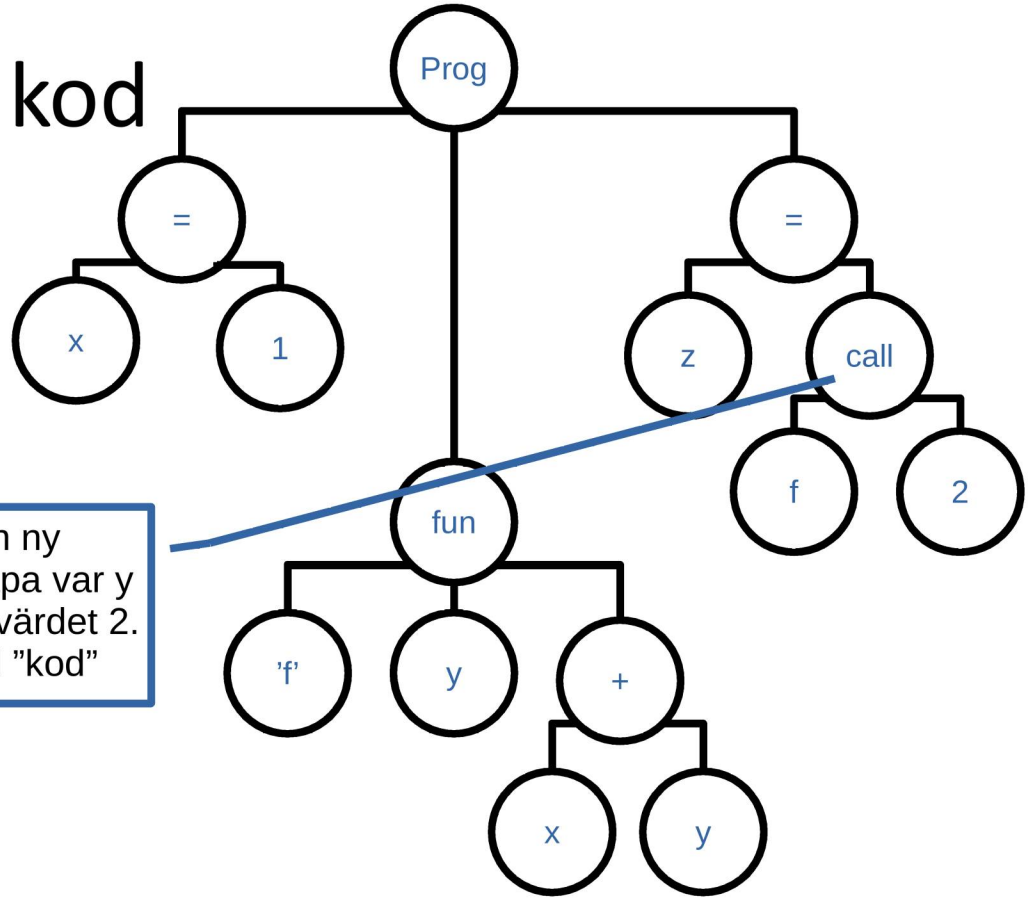


Returnera 2

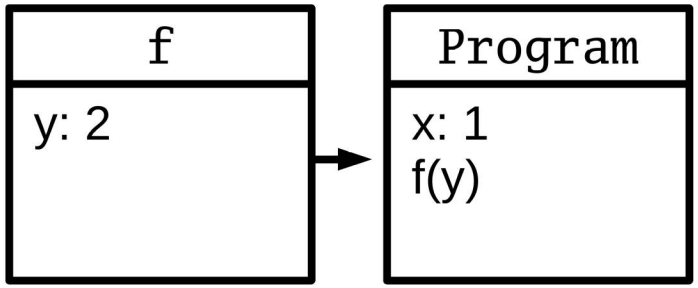
Program
x: 1 f(y)

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

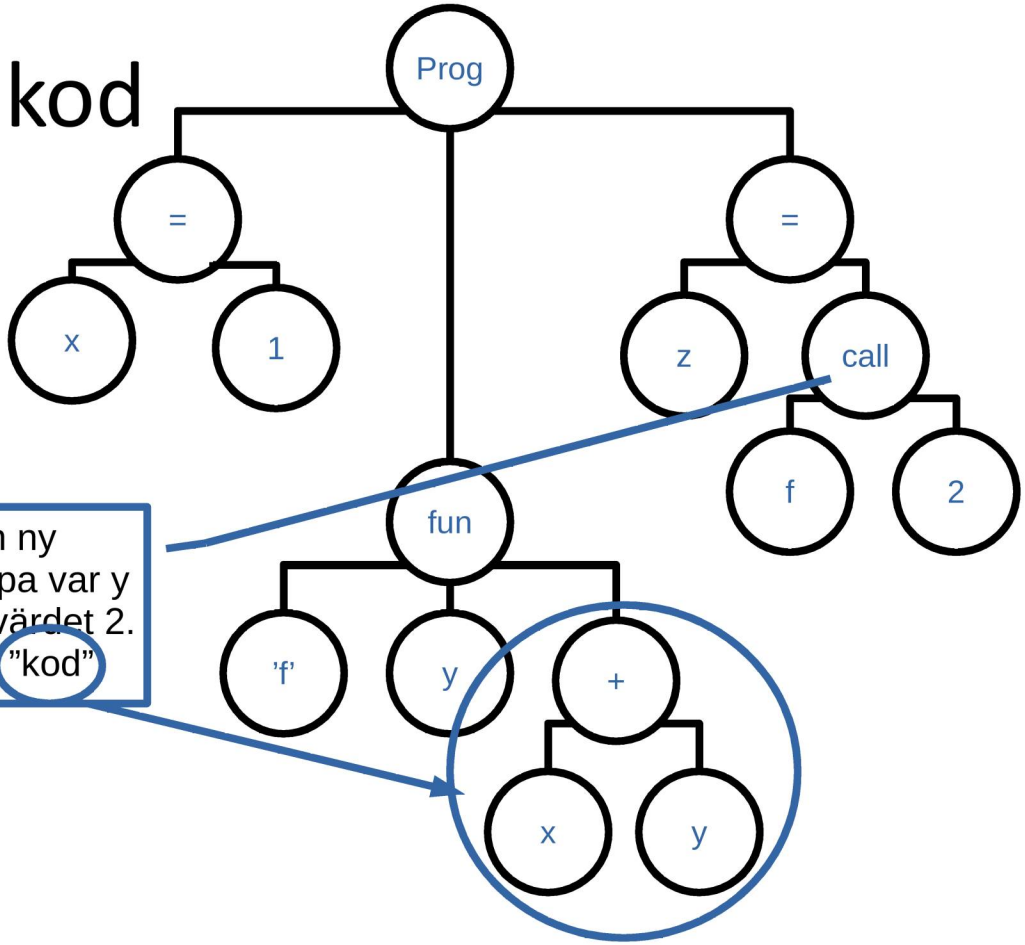


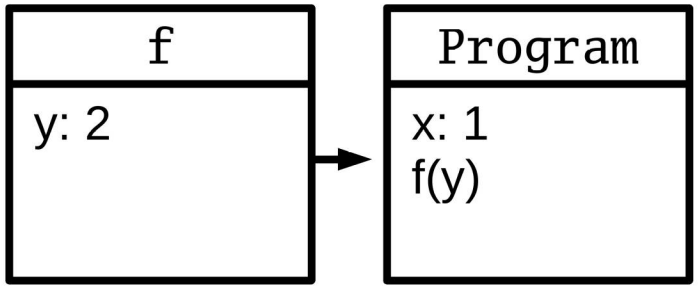
Sätt upp en ny frame. Skapa var y i den med värdet 2. Kör sparad ”kod”



Lagra och anropa kod

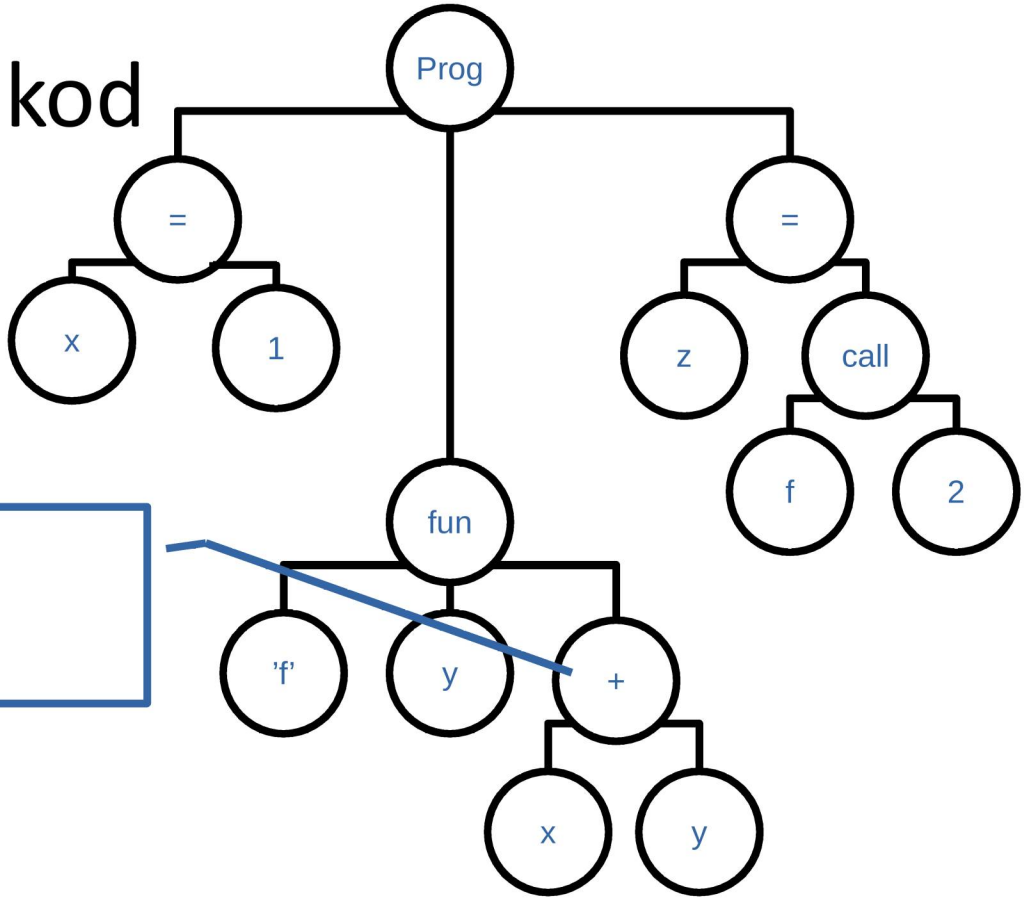
```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

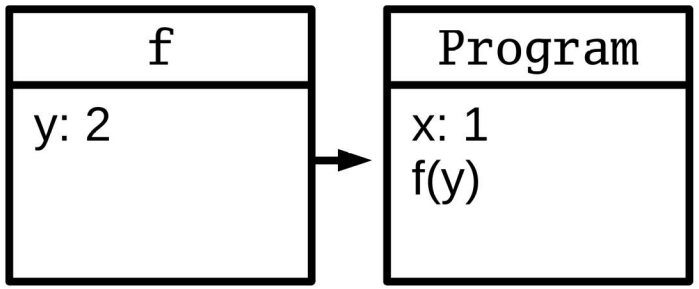




Lagra och anropa kod

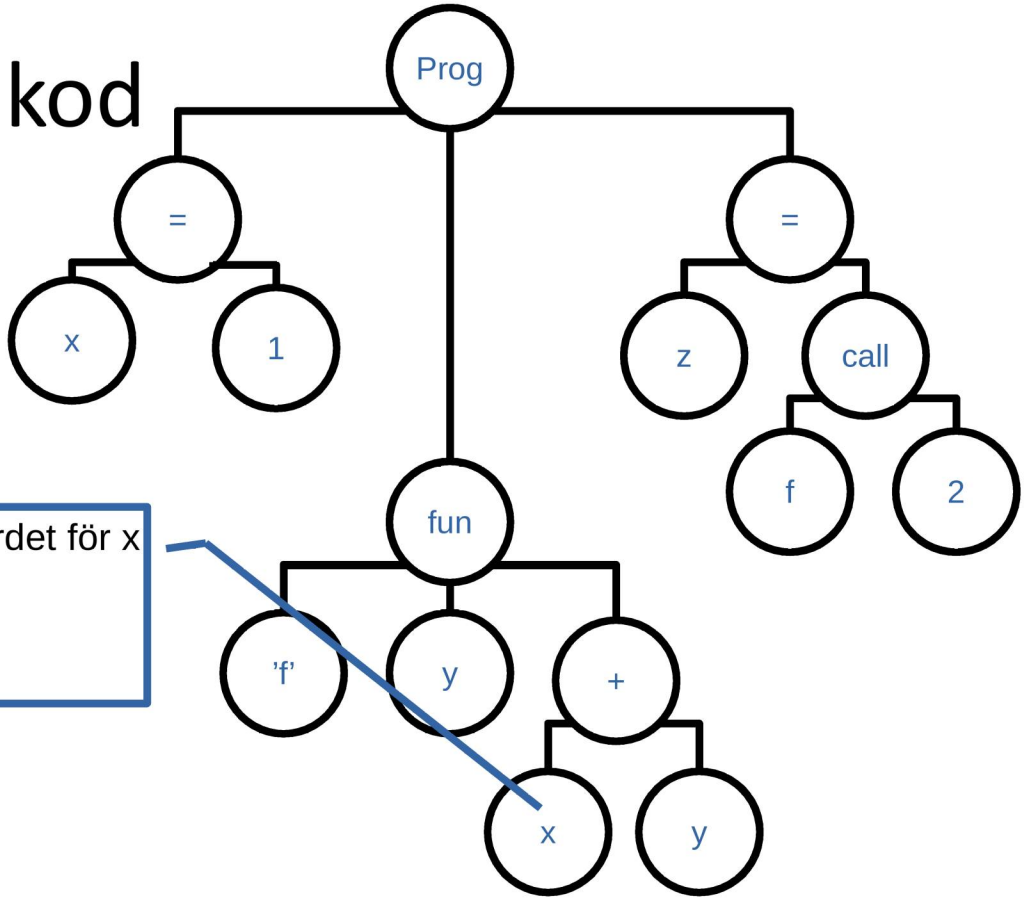
```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



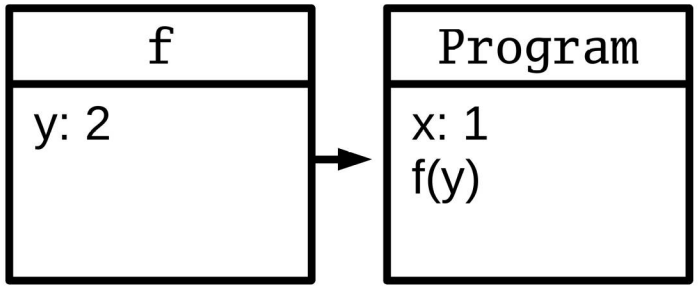


Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

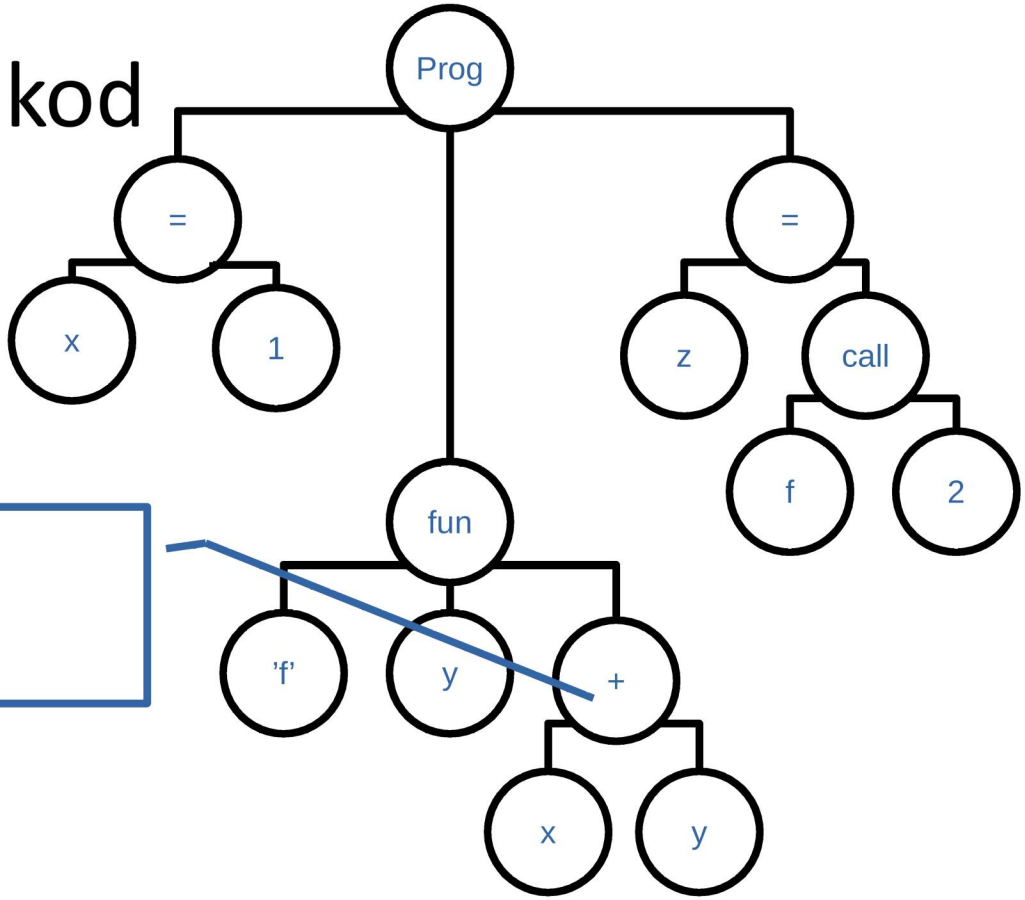


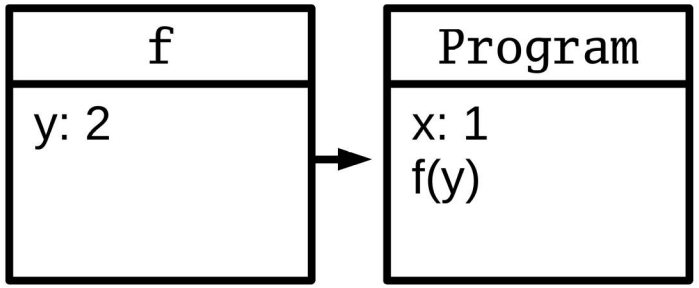
Slå upp värdet för x



Lagra och anropa kod

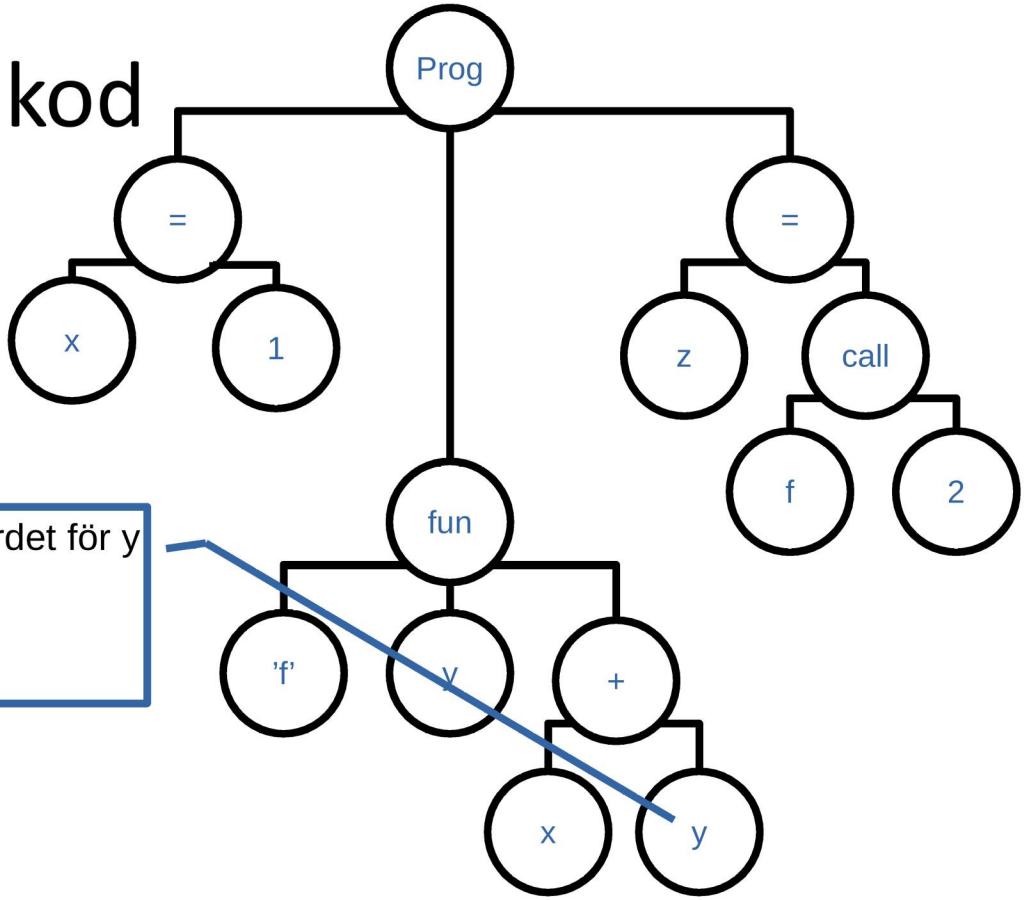
```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



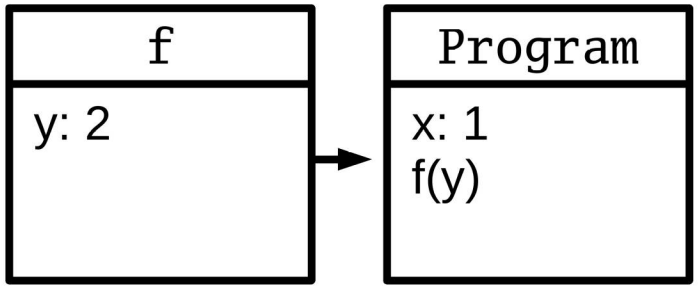


Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

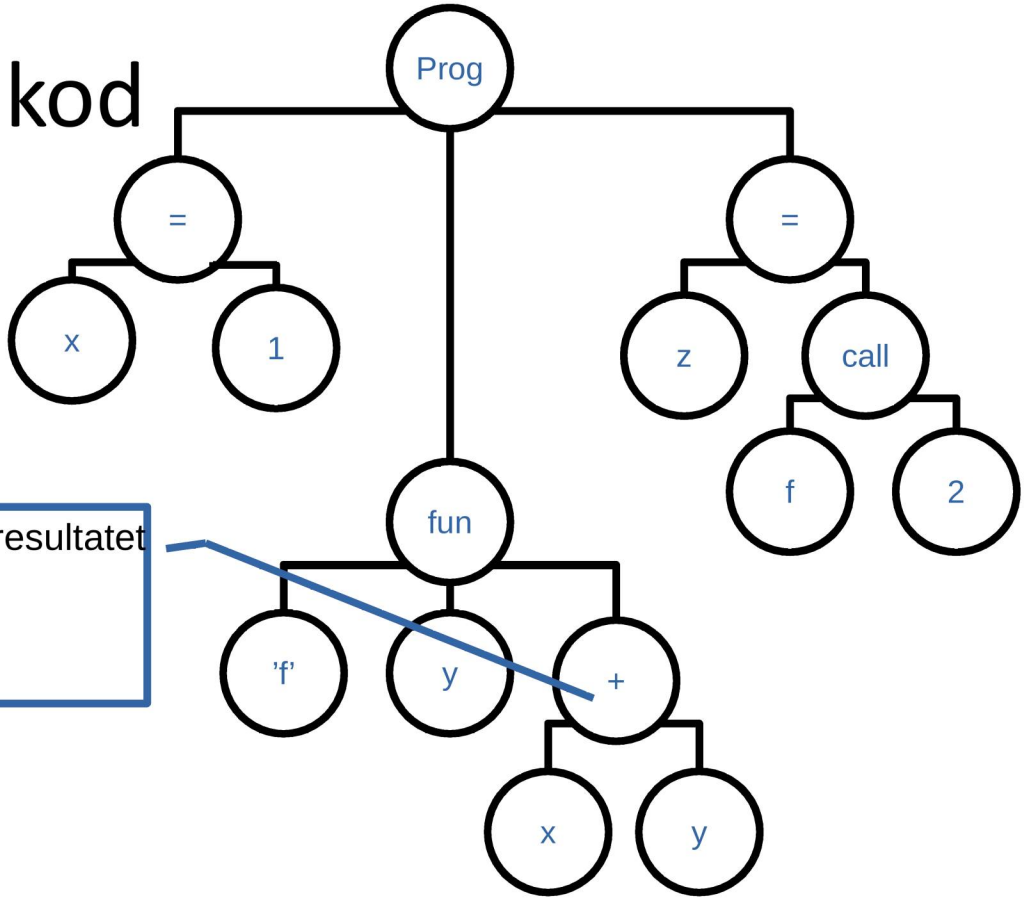


Slå upp värdet för y

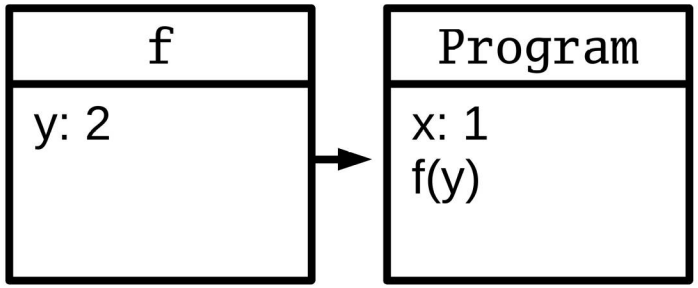


Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

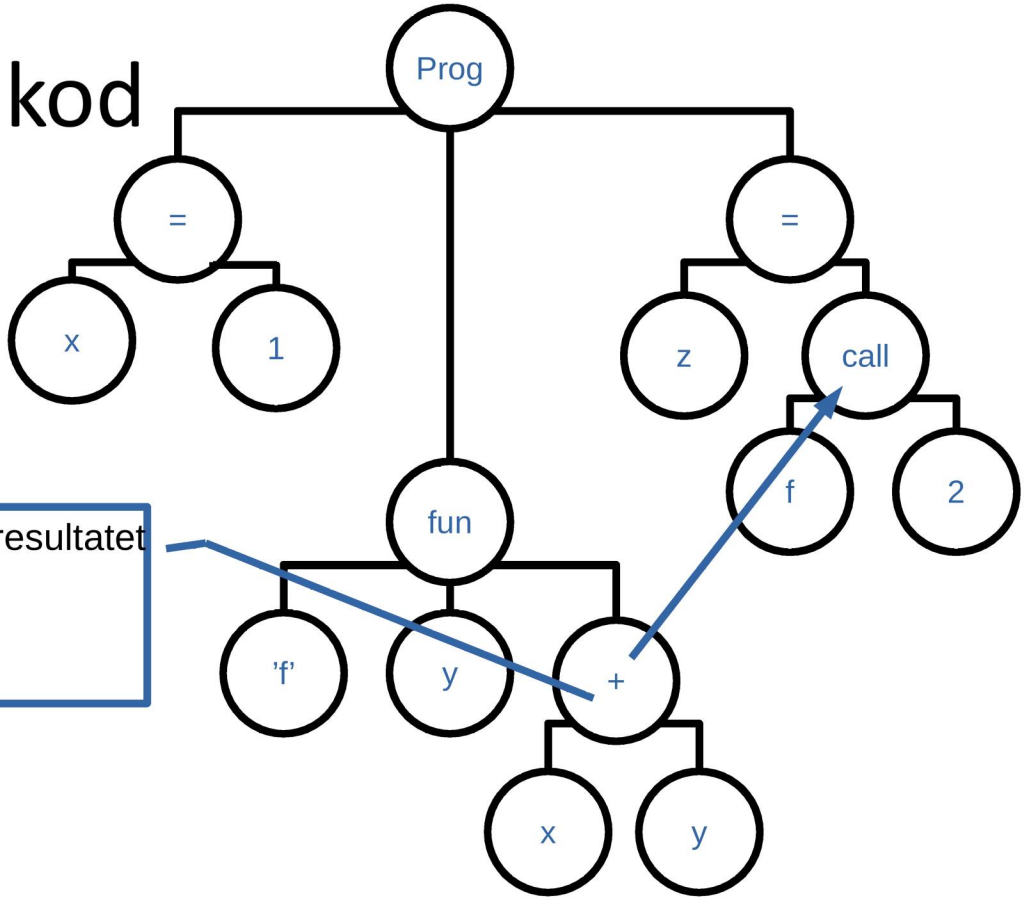


Returnera resultatet av 1+2

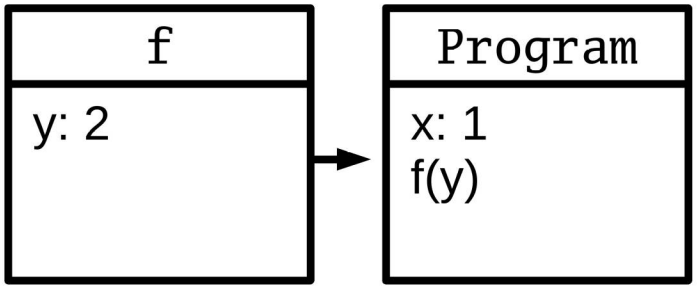


Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

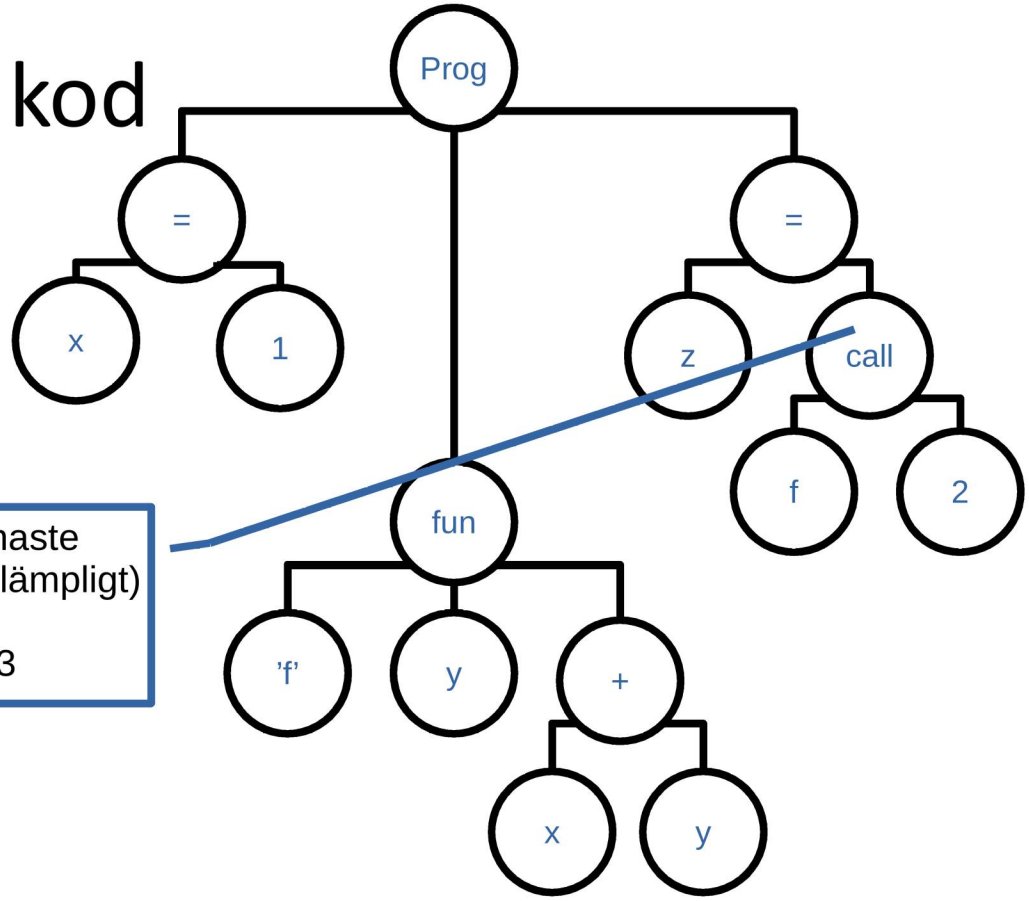


Returnera resultatet av 1+2



Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



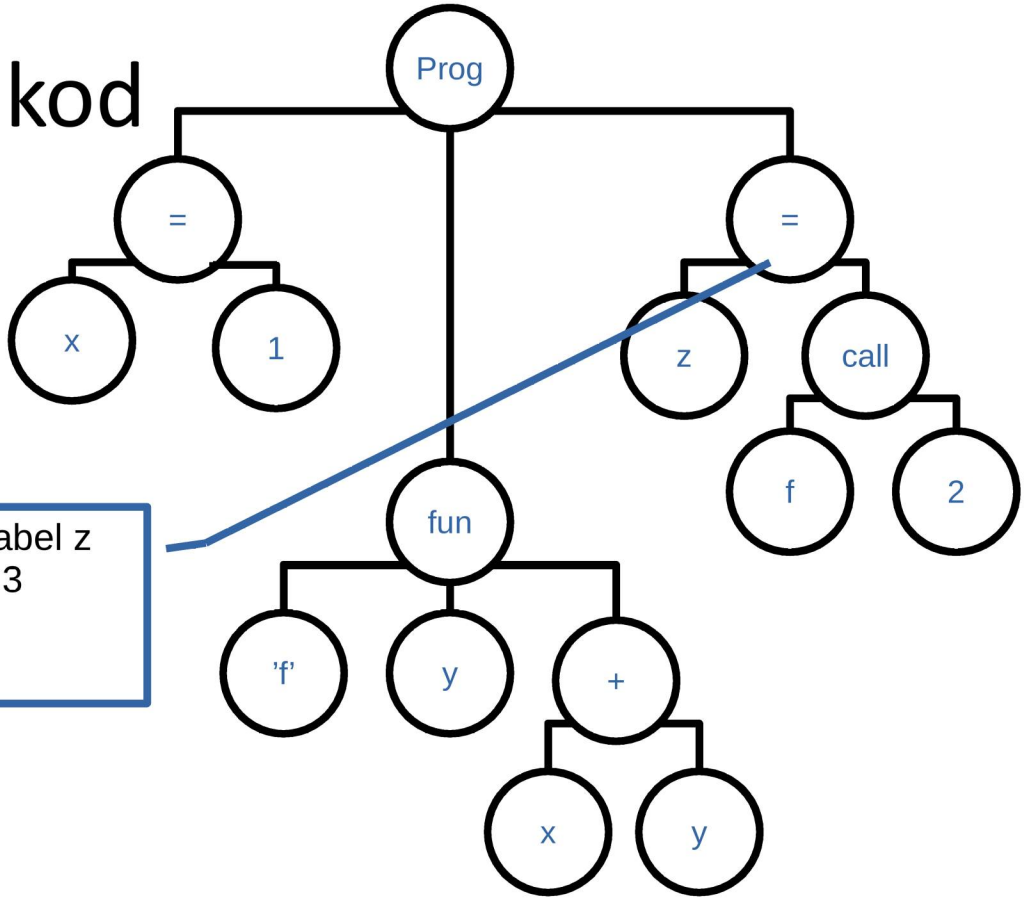
Riv ner senaste frame (om lämpligt)

Returnera 3

Program
x: 1 f(y)

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```

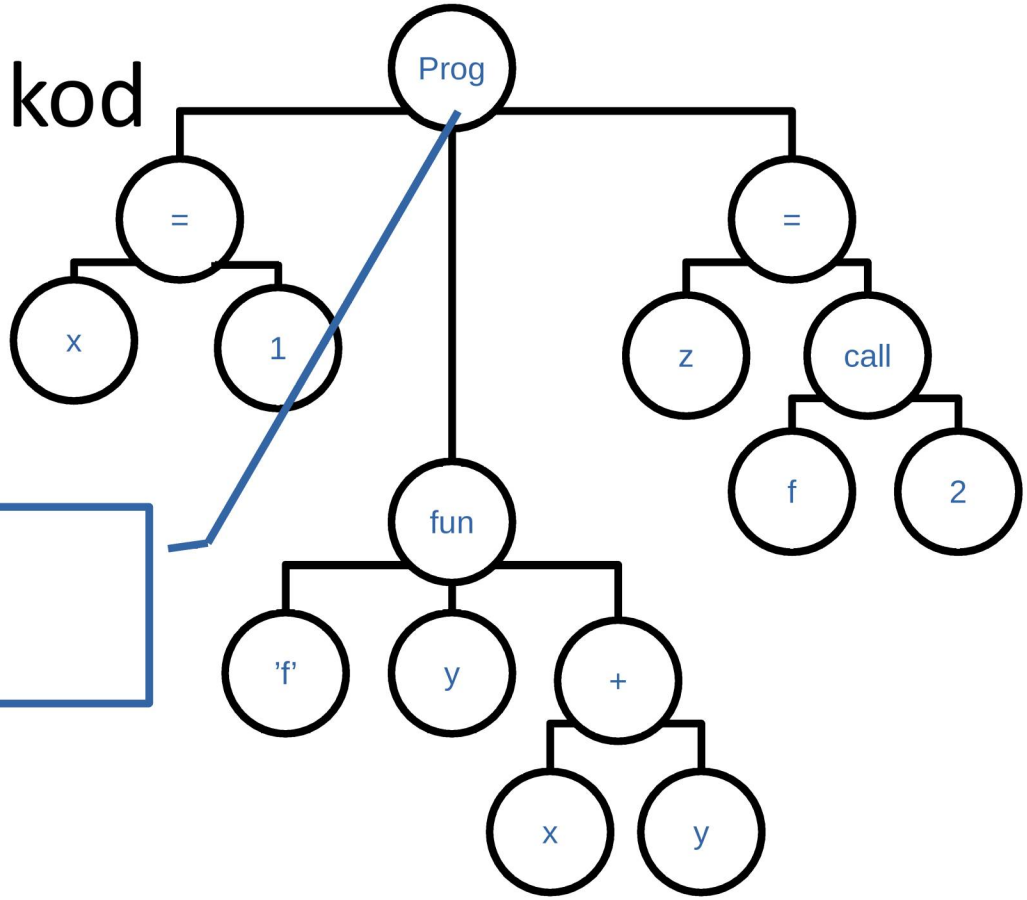


Skapa variabel z med värde 3

Program
x: 1 f(y) z: 3

Lagra och anropa kod

```
program
    var x=1;
    fun f(y)
        return x+y;
    end f;
    z = f(2);
end program
```



Att lagra och anropa kod

- Vad finns det för andra exempel där detta kan vara användbart?
- Olika typer av funktioner?

Referenser och kopior

LINKÖPINGS UNIVERSITET

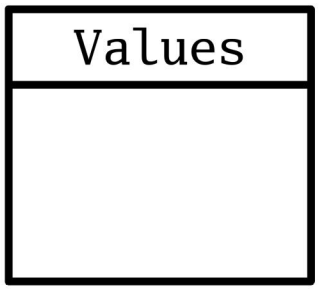
Referenser och kopior

- I vissa språk överförs parametrar som referenser, i andra som kopior
- Ibland får programmeraren välja/specificera

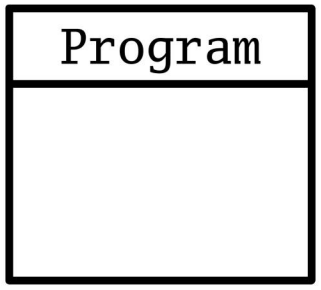
- Hur styr vi detta?
  När vi implementerar runtime

- Ni kommer behöva bestämma hur det ska fungera och implementera ert språk på det sättet

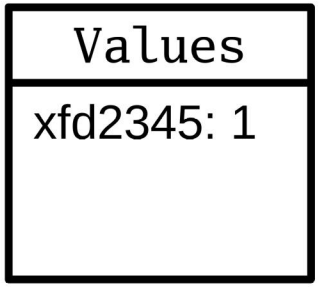
Ett exempel referens



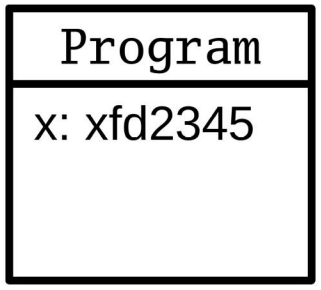

```
program
    var x=1;
    ref y=x;
    y = 3;
end program
```



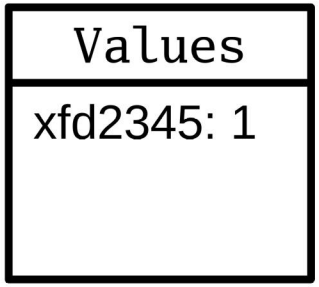
Ett exempel referens



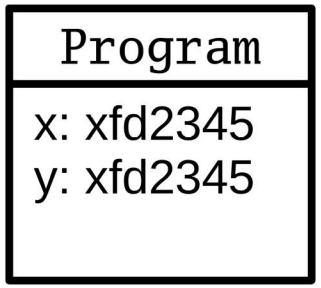

```
program
    var x=1;
    ref y=x;
    y = 3;
end program
```



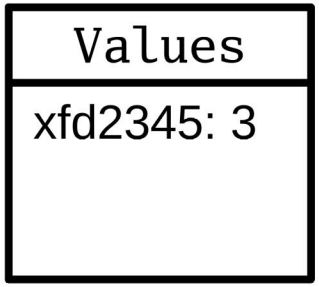
Ett exempel referens



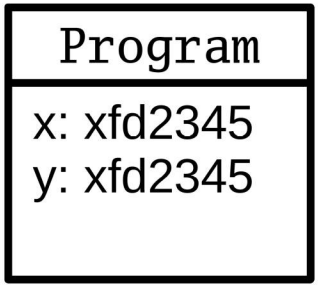

```
program
    var x=1;
    ref y=x;
    y = 3;
end program
```



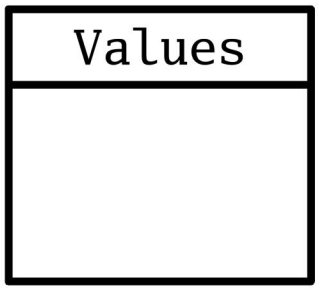
Ett exempel referens




```
program
    var x=1;
    ref y=x;
    y = 3;
end program
```



Ett exempel referens




```
program
    var x=1;
    ref y=x;
    y = 3;
end program
```

Organisera sina filer

LINKÖPINGS UNIVERSITET

Organisera projektet

- Grammatik
  En fil för parser (rdparse)
  En för grammatik

- Syntaxträd

- Klasser (semantik)
  En eller flera filer med klassdeklarationer
  Eval-metoder

- Runtime
  Ska beskrivas hur detta fungerar

Några siffror

LINKÖPINGS UNIVERSITET

- 7
- 45
- 160
- 46

- 7 Kurser
- 45
- 160
- 46

- 7 Kurser
- 45 st Föreläsningar (90 timmar)
- 160
- 46

- 7 Kurser
- 45 st Föreläsningar (90 timmar)
- 160 st Andra schemalagda pass (ca 320 timmar)
- 46

- 7 Kurser
- 45 st Föreläsningar (90 timmar)
- 160 st Andra schemalagda pass (ca 320 timmar)
- 46 högskolepoäng (1226 timmar studier)

- 7 Kurser
- 45 st Föreläsningar (90 timmar)
- 160 st Andra schemalagda pass (ca 320 timmar)
- 46 högskolepoäng (1226 timmar studier)

“Variety is the very spice of life, that gives it all its flavor.”

- William Cowper's (1785)

Tack för deltagandet

www.liu.se

LINKÖPINGS UNIVERSITET