Principer för (imperativa) programmeringsspråk

Human Centered Systems
Inst. för datavetenskap
Linköpings universitet

Översikt

- Motivering
- Syntax och semantik
- Imperativa språkets byggstenar – och Python
  - Datatyper
  - Tilldelning och uttryck
  - Kontrollstrukturer (på satsnivå)
  - Subprogram

Relaterade avsnitt: PL 3.1-3.2, 5.1-5.3, 5.8 intro, 6.1-6.3, 7.1-7.3, 8.1-8.3, 9.1-9.2, 9.5.2.4

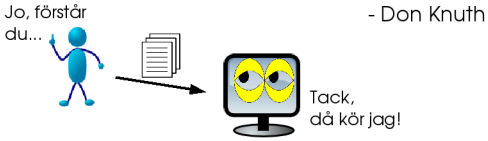
Varför lära sig detta?

- PL-boken Kap 1 (repetition):
  - Förmåga att uttrycka sig, förmåga att välja språk, snabbare inlärning, förståelse för implementation, ökad insikt om språk
- En hantverkare kan sina verktyg
  - Alla programmerare bör ha implementerat ett eget språk som minimum
- Förståelse för vad som krävs för att implementera ett datorspråk är användbar kunskap för alla program
  - Även t ex en modul av funktioner kan ses som ett "språk"

Förståelse för språk = förståelse för konstruktion av språk

"It has often been said that a person does not really understand something until after teaching it to someone else. Actually a person does not *really* understand something until after teaching it to a *computer*, i.e. expressing it as an *algorithm*"

- Don Knuth



Motiverande exempel

Exempel på ett korrekt uttryck i Python:

```
numb1 > 3 and numb2 != 4
```

Exempel på ett inkorrekt uttryck i Python:

```
numb1 > 3 and numb2
```

***Fråga**: Vår Pythontolk måste ha en algoritm i sig som kan särskilja dessa två fall. Hur ser en sådan algoritm ut?*
***Steg 1**: Vi behöver detaljerad förståelse för språkets syntax och semantik*

Syntax och semantik

Begreppen syntax och semantik

- **Syntax**: formatet på språkets konstruktioner
- **Semantik**: betydelsen hos språkets konstruktioner
- Varje programspråk har sin egen **unika syntax**
- **Paradigm**: språken har en gemensam semantisk kärna men kan variera i olika detaljer
- *I våra kurser*: Vi lär oss principerna – så kan vi paradigmet
  - Språket som vi väljer är ett exempel...om än valt med omsorg
  - Men då måste vi lära oss se **principerna i exemplet**

Python vs Java – en skillnad i syntax...

Python:

```
if x == 2 and y == 3
    print "hej"
```

Java:

```
if (x == 2 && y == 3) {
    System.out.println("hej");
}
```

"villkor"

```
for x in range(7):
    print x
```

```
for (int x=0; x < 7; x++) {
    System.out.println(x);
}
```

"iteration"

...men i många avseenden **samma semantik**

Hur beskriva ett programmeringspråks syntax?

- Ett språk kan ses som en mängd strängar/meningar/satser
- Ett programmeringsspråk är ett formellt språk
- Formella språk är definierade av syntaxregler
- Formella språk är enkla att hantera med algoritmer
  - Exaktheten ger enkelhet för datorn (svårighet för människan...)

Parser/igenkännare

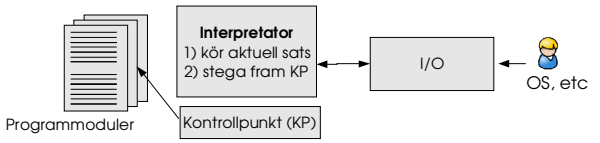
- Algoritmer som avgör om givna satser är korrekta eller ej
- Lexem: grundläggande klassificering av strängar
- Tokens: gruppering av lexem i principiella roller t ex "identifierare".
- Använder normalt en formell grammatik
- Backus-Naur-Form: grammatisk formalism för syntax (seminarium 3)

Hur beskriva semantiken för ett programmeringsspråk?

- Formella metoder – matematiskt avancerade modeller
  - Inte i denna kurs, men nämns några i PL-boken
- Pragmatisk metod
  - 1. lär dig hur interpretatorer och kompilatorer fungerar i stora drag
  - 2. Lär dig informellt vad som händer för olika konstruktioner

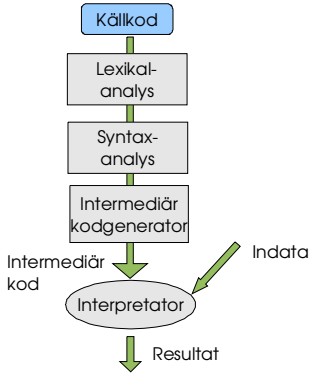
Hur exekveras ett Pythonprogram?

Grundläggande bild:



En sådan modell kallas ibland för operationell semantik.
I detta fall för ett s k interpreterat språk.

Hybridmodellen

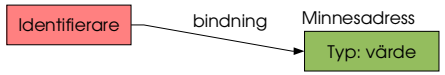


Imperativa språkets byggstenar

Namn

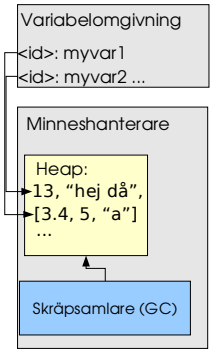
- Namn kallas ofta identifierare
- Identifierar variabler, moduler, funktioner etc
- Exempel på designval:
  - känslighet för stor/liten bokstav
  - Nyckelord som inte får vara identifierare
- Konventioner: särskilj delar av namnet med _ eller kamelNotation.
- Python:
  - _ eller bokstav + sekvens av _, bokstav, siffra
  - Skillnad på stor och liten bokstav.
  - For inte vara ett reserverat ord, t ex **for**, **break**, **def**

Variabelbindning



- Variabler har namn, typ och värde
- Variabelbindning är associationen mellan namnet och dess "typ+värde"

Heap: Python-objektens minnesarea



- Variabelomgivningen består av aktuella identifierare
- På Heap lagras skapade objekt
  - antal och storlek ibland okänt
  - binär representation
  - refereras/skapas via *uttryck*
- Skräpsamlare – Garbage Collector

Räckvidd (engelska: scope)

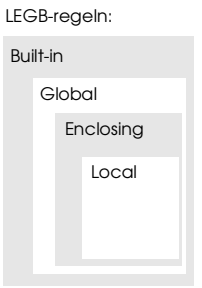
- Det räckvidd av satser där en identifierare/variabel är möjlig att synlig/refererbar
- Variabler är lokala i ett källkodsblock om de införts där.
- Icke-lokala variabler är variabler som är nåbara trots att de införts lokalt
- Skuggning måste hanteras så att man kan hålla isär på icke-lokala och lokala variabler

Variablers räckvidd i Python

- Variabler har begränsad räckvidd
  - Globala variabler – i modulen
  - lokala variabler – i nuvarande funktion
- Satsblock markeras i Python med indentering
  - samma indentering i följd samma block
  - markerar en svit av satser som ska exekveras "tillsammans"
- I explicit typade språk t ex Java/C++ är räckvidd och block förbundna
  - ...men inte i Python

När och hur får lokala variabler värden?

- Funktioner kan definieras nästlat i varandra
  - Omgivande (enclosing) funktion
- Lokala variabler har s k statisk räckvidd
  - Vi kan referera till variabler i funktionen som är kontext



Ett exempel

```
var = "hej"

def foo():
    print var

def foo1():
    var = 7

def foo2():
    global var
    var += " hopp"

def foo3():
    var = " hoppsan"
    import mymodule
    mymodule.var += var
```

Skriv ut icke-lokala variabeln var

Lokal variabel införs som sedan skuggar den globala

Explicit referens till global variabel

Möjlighet att referera variablerna i samma omfång

Datatyper

- En mängd datavärden som samlat för ett visst syfte
- Primitiva datatyper: värden består inte av värden av annan typ
  - Tal, tecken och Booleska typer
  - Strängar ett gränsfall
  - I Python: taltyperna och boolean
- Sammansatta datatyper: typer som inte är primitiva
  - Arraytyper och associativa arrayer/hashes viktiga exempel
  - I Python: sekvenstyper och Dictionary
- Dynamiska datatyper: minnestilldelningen sker vid körtid
  - Python har dynamiska datatyper

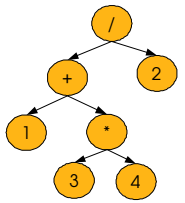
Uttryck

- Aritmetiska uttryck oftast likartat i alla imperativa språk
- Booleska uttryck lite mer syntaxskillnader
- Precedence: vilken operator ska appliceras först
  - Ex: 1 + 3 * 4
- Vänster/höger-ssociativitet: vilken ordning operatorer med samma precedence ska appliceras
  - Ex: 1 + 2 + 3
- Överlagrade operatorer: samma symbol fungerar för många typer
  - T ex + i Python för både tal och strängar

Evaluering av uttryck

Uttryck/term: (1 + (3 * 4)) / 2

Trädstruktur:



- Trädstrukter används för att representera uttryck
- Evaluering av ett uttrycks görs av algoritmer som **traverserar** trädet
- Enklast nerifrån och upp och kom ihåg delresultaten

T ex som lista av listor: ['/', ['+', 1, ['*', 3, 4]], 2]

Python: Booleska och relationsoperatorer

Booleska operatorer

- `not` negation
- `and` logisk och
- `or` logisk eller

Booleska konstanter

True värde 1 "sant"
False värde 0 "falskt"

Relationsoperatorer

- `x == y` likhet
- `x != y` icke-likhet
- `x < y` mindre än
- `x > y` större än
- `x <= y` mindre eller lika
- `x >= y` större eller lika

Booleska uttryck i Python

```
x = 1
y = 2

x < y            # True
(x >= (y – 1))   # True
(x > (y – 1))    # False
(x != y)         # True
(y == x)         # False
(y == (x * 2))   # True

(x > y) or (x != y * 2) # True
not ((x < y) and (x != y)) # False
```


Tilldelningssatsen

- Enkel tilldelning: variabel tilldelas uttryckets värde
- Semantik: identifieraren binds till uttryckets värde
- Sammansatt tilldelning: +=, -= etc
- Sekvenstilldelning: flera variabler får värden samtidigt
  - Kan användas för att undvika temporär variabel vid "swap"
- Tilldelning är exempel på en enkel sats.

Exempel på enkla satser i Python

```
area = math.pi *
5**2
print area
list_1 = list("abba")
del list_1[1:3]
my_function("hej")

break
return list_1
```

- Instruktioner
- Atomär kontroll
- Kommandot reserverat ord/symbol
- Enkla satser = kommandoskript

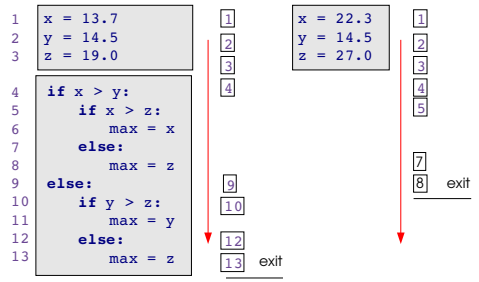
Programmets kontrollflöde

- Kontrollflöde: ordningen som programsatserna exekveras i
  - Kontrollflödet kallas ibland dynamisk struktur
  - För rak sekvens av (enkla) satser: uppifrån och ner
  - **if**, **while**, **for** styr detta flöde - olika flöden vid val/villkor
- Kontrollpunkt: kallas ibland den punkt där programmet befinner sig vid ett visst ögonblick
  - på maskinkodsnivå: "program räknaren" håller adressen till nästa sats som ska exekveras
  - Vid subprogram (funktioner) måste anropsposition sparas, dvs vi behöver en s k anropsstack

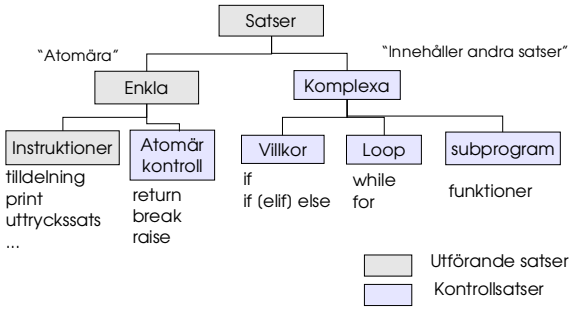
Kontrollsatser

- Satser vars syfte är att styra programflödet
- Sammansatta kontrollsatser styr lokalt inom sig
  - T ex villkor och loopar
- Enkla kontrollsatser utgör del av andra konstruktioner
  - T ex return och break
- Villkor: innehåller 1, 2, eller fler villkorade klausuler
- Iterativa satser: definitiva och indefinita loopar
  - Normalt for och while-loop

Kontrollflöden för if-sats



Taxonomi över viktigaste satstyperna i Python



Subprogram

- Namngivna källkodsblock
- Subprogram definieras och anropas
- Vi skiljer på definitionshuvud och definitionskropp
- Parametrar används för skicka med indata vid anrop
- Returvärde används för att skicka med utdata vid retur från subprogram

Funktion och procedur

OBS: dessa begrepp används lite olika av olika författare/språk

- Två typer av subprogram
  - Funktion: ett subprogram som returnerar ett värde
  - Procedur: returnerar inte ett värde – I Python värdet None
- Funktioner kan ingå som del i uttryck
- Sidoeffekt: effekter förutom returnerat värde
  - t ex om subprogrammet skriver ut på konsol
  - parametrar av referenstyp (mer detaljer senare)
  - en "ren" funktion har inga sideffekter
  - procedurer har "bara" sideffekter
- "Funktionell procedur": både returvärde och sideffekter

I Python – modulen består av satser

Rätt:

```
def power(x,y):
    result = 1
    for _ in range(y):
        result = times(result,x)
    return result

def times(x,y):
    return x * y

print power(2,4)
```

Fel:

```
print times(4,5)

def times(x,y):
    return x * y
```


Parametrar och argument

- (Formella) parametrar i en definition
  - beter sig som lokala variabler i funktionen
  - binds på nytt vid varje anrop
  - frigörs från bindning vid anropets slut
- (Aktuella parametrar) argument i ett anrop
  - består av uttryck
  - argumentets värde överförs till parametern

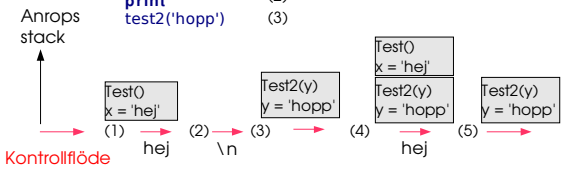
Anropsstacken

```
def test():
    x = 'hej'
    print x,

def test2(y):
    test()          (4)
    print y         (5)

test()              (1)
print               (2)
test2('hopp')       (3)
```

- Vid varje anrop av en funktion skapas en variabelomgivning
- Anropen läggs på en s k anropsstack med "funktionsomgivningar"



Metoder för parameteröverföring

- Tekniken som används för att tilldela parametrar värden av argument-uttryck
  - Olika språk har olika principer (det finns 3-4 vanliga)
  - C++ kommer visa på en annan variant än Python
- I Python: "genom objekt-referens" även kallat "genom tilldelning" (LP-boken: pass-by-reference)
  - argumentutryckets värde är en objekt-referens – den tilldelas parametern
  - Innebär strukturdelning för muterbara strukturer

Exempel: önskad sideffekt

```
def assign_plus(val, index, list):
    val += 1
    list[index] = val

val = 34
mylist = [12, 3, 7]
assign_plus(val, 2, mylist)
print 'val: ', val
print 'mylist: ', mylist
```

```
val:  34
mylist:  [12, 3, 35]
```


Exempel: blockera sideffekt - kopiera

```
def safe_filter(val, mylist):
    mylist = mylist[:]
    if val in mylist:
        mylist.remove(val)
    return mylist

mylist = [7,3,6,3]
mycopy = safe_filter(3, mylist)
print 'mylist: ', mylist
print 'mycopy: ', mycopy
```

```
mylist:  [7, 3, 6, 3]
mycopy:  [7, 6, 3]
```

Summering

- Algoritmisk förståelse – den djupaste förståelsen
- Syntax och semantik – BNF och operationell semantik
- Namn – binds, har räckvidd, dynamiskt på heap
- Datatyper – primitiva och sammansatta
- Uttryck – precedence, associativitet, överlagring och uträkning
- Tilldelning: enkel och sammansatt, värde binds till identifierare
- Kontrollstrukturer: styr programflödet
- Subprogram/