Programmering 1 med Python

Patrik Berggren

2 december 2024

Innehåll

Kapitel 1

Introduktion

Välkommen till denna bok om programmering med **Python**. Här lär du dig de grundläggande byggstenarna för att skriva och förstå kod. Vi går steg för steg med enkla exempel och övningar.
Kom ihåg att stanna upp och läs igen om du inte är säker på om du förstått. Övningarna syftar till att kontrollera att man också själv kan använda det som avsnitten förklarar. I slutet på boken finns lösningsförslag på övningarna. I programmering kan det finnas många olika sätt att lösa en uppgift på. Lösningsförslagen är inte nödvändigtviss det ända eller ens det bästa sätten, men ett förslag på hur man kan tänka.

1.1 Kodexempel

Förutom kodexempel i varje avsnitt finns också ett appendix med fler kodexempel. Det kan vara bra att kolla på fler exempel för att se en större variation av användning av de programmeringskoncept vi kollar på i avsnitten.

1.2 Vad är programmering?

Programmering handlar om att skriva instruktioner som en dator kan förstå och följa. Python är ett av de mest populära programmeringsspråken eftersom det är lätt att läsa och använda. Det är också ett vanligt val inom maskininlärning, eller webbutveckling.

Python
Python är ett programmeringsspråk. Det används för att skapa allt från enkla program till avancerade system som spel och webbplatser.

Kapitel 2

Grunder i programmering med python

2.1 Utskrifter med print

2.1.1 Hello World Program

Ett av de första programmen man brukar skriva när man lär sig ett nytt programmeringsspråk är det så kallade ”hello world”-programmet. Det vill säga ett program som skriver ut texten **hello world** på skärmen. Så här ser det ut i Python:

```
1 print("Hello, world!")
```

Hello World-program

När du kör programmet visas texten `Hello, world!` i terminalen.

Terminal/Konsol

En terminal (eller konsol) är ett program där du kan skriva in och köra kommandon, till exempel för att köra ditt Python-program.

2.1.2 Syntax i programmering

När vi kör Hello World-programmet behöver datorn veta vad vi vill göra. Datorn kräver att vi skriver vårat program på ett väldigt exakt formatterat sätt, detta kallas syntax. Exempelvis skulle programmet ovan inte gå att köra om vi hade gjort om det på något av sätten nedan där enstaka tecken tagits bort eller till och med om vi använder stora istället för små bokstäver.

```
1 PRINT("Hello, world")
2 Print("Hello, world")
3 print("Hello, world"
4 print("Hello, world)
5 print(Hello, world")
6 print(Hello, world)
7 print "Hello, world"
8 skrivut(Hello, world)
```

Felaktig syntax i kod

Syntax

Syntax är reglerna som bestämmer hur kod ska skrivas för att datorn ska förstå den. Om vi bryter mot syntaxen, fungerar inte programmet.

Syntax highlighting

När våran editor färglägger olika delar av koden. Det gör det enklare för oss att läsa kod, eller att se när syntaxen är fel som i exemplena ovan.

2.1.3 Funktionen `print()`

Funktionen `print()` används för att skriva ut text eller resultat på skärmen. En funktion är en bit kod som utför en specifik uppgift. När vi använder en funktion, skickar vi in något den ska jobba med. Detta kallas ett **argument**. Här är ett exempel:

```
1 print("Hej på dig")
```

Syntax för print()

- **print** är funktionens namn.
- **paranteser** kommer alltid efter namnet på en funktion. Inom parantesen skriver vi det vi skickar in i funktionen (Argumentet). Detta kan liknas vid funktionsbegreppet i matematik där vi ofta skriver f(x). Där är f namnet och x argumentet.
- **”Hej på dig”** är argumentet. Vi behöver markera för python att det vi skickat in är en text. Det gör vi genom att använda citattecken. Vi kommer senare att förstå varför.

Funktion

En funktion är en bit kod som gör en uppgift. Vi kan använda funktioner genom att anropa dem med deras namn.

Argument

Ett argument är något vi skickar in i en funktion, till exempel en text eller ett tal. Argument ges alltid inom ett par paranteser.

2.1.4 Programmering kan räkna ut saker

Python kan användas för att göra matematiska beräkningar, till exempel addition, subtraktion, multiplikation och division. Här är några exempel:

```
1 print(5 + 3)  # Addition
2 print(10 - 4) # Subtraktion
3 print(7 * 2)  # Multiplikation
4 print(9 / 3)  # Division
```

Matematik i Python

Observera!

I programmet ser du att vi skrivit förklaring till koden efter tecknet #. Genom att använda # tecknet kan vi tala om för python att det som kommer efter inte är kod utan en förklaring. Det kan vara bra för att den som läser koden ska förstå den. Vi kommer prata mer om kodkommentarer i ett senare avsnitt.

När detta körs visas resultaten på skärmen:

Output från Matematik i Python - programmet

```
8
6
14
3.0
```

Output

Det som programmet skriver ut på skärmen

2.1.5 Övningar

Här är några övningar för att testa det du har lärt dig.

Uppgift 1 Skriv ett program som visar texten `Hej! Jag lär mig Python.` på skärmen.

Lösningar på sida 63 ◄

Uppgift 2 Använd funktionen `print()` för att visa resultaten av följande beräkningar:

12 + 5

20 - 8

4 * 3

16 / 4

Lösningar på sida 63 ◄

Uppgift 3 Skriv ett program som använder `print()` för att visa texten **Python är roligt!** och beräkningen `10 + 15` på olika rader.

Lösningar på sida 63 ◄

För fler kodexempel se appendix A.1

2.2 Variabler och Sekvensiell Exekvering

2.2.1 Vad är en variabel?

En variabel är som en namngiven låda där vi kan lagra data, som till exempel tal eller text. Vi kan senare ändra innehållet i lådan eller använda det i beräkningar.

```
1 minvariabel = 100+1
```

Tilldelning

I programmet ser vi hur vi sparar ett värde i en variabel. I exemplet heter våran variabel **minvariabel** och vi sparar värdet 101 i den. Vi kan senare i programmet återanvända våran variabel för att hämta det värde vi sparat i den. Notera att vi använder likamedtecknet = men inte på samma sätt som i matematik. När python ser = så kommer den först göra en uträkning av det som står på höger sida. Det blir alltid ett konkret värde, exempelvis ett tal, eller en text. I exemplet räknar python ut åt oss att $100+1$ blir 101. Python sparar sedan värdet 101 i variabeln som vi döpt till **minvariabel**.

Likamedtecknet

Likamedtecknet betyder i programmering att vi sparar värdet på högersidan i variabeln vi skrivit på vänstersidan.

Observera!

Likamedtecken i programmering betyder **INTE** samma sak som i matematik. Det kan se ut som att vi ställt upp en ekvation, men vi använder = bara för att spara ett uträknat värde.

2.2.2 Använda variabler

```
1 x = 5
2 y = 10
3 z = x + y
4 print(z)
```

Exempel på variabler

När detta program körs, kommer resultatet `15` att visas på skärmen eftersom `x` är 5 och `y` är 10.

Variabel

En plats i datorns minne där data, som tal eller text, lagras. Vi ger platsen ett namn för att enkelt kunna använda den i programmet.

2.2.3 Datatyper

Data i Python kan ha olika typer, som bestämmer vad vi kan göra med den. Några vanliga typer:

- `int`: Heltal (till exempel `5`)
- `float`: Decimaltal (till exempel `3.14`)
- `str`: Text, även kallad sträng (till exempel `"Hej!"`)

```
tal = 5           # int
decimaltal = 3.14 # float
text = "Hej!"     # str
```

Exempel på olika datatyper

Eftersom datorn bara ser 1:or och 0:or i minnet behöver den hålla reda på om det vi sparat är ett heltal eller en text. Vi kommer gå djupar kring datatyper senare, men nu vet du varför vi behöver citattecken när vi skriver en text. Datatyper är också förklaringen till varför `print(4/2)` blir 2.0 och inte 2. Python gör nämligen om alla divisioner till float (decimaltal).

Datatyp
En typ av värde, exempelvis heltal (integer), eller textvärden (string)

Integer
Variabler av datatypen integer (förkortat int) innehåller heltalsvärden

Float
Variabler av datatypen float innehåller decimaltal

String
Variabler av datatypen string (förkortat str) innehåller textvärden. På svenska säger vi också att datatypen är en sträng.

2.2.4 Sekvensiell exekvering

När ett program körs, läses koden rad för rad, uppifrån och ned. Detta kallas sekvensiell exekvering. Ordningen på kod spelar alltså mycket stor roll!!! Här är ett exempel:

```
x = 5
print(x)
x = 7
print(x)
```

Ordning i koden spelar roll

Resultatet blir först `5` och sedan `7`, eftersom värdet på `x` ändras innan den skrivs ut andra gången.

Exekvering

Exekvering betyder att man kör programmeringskod. Det vill säga att datorn steg för steg går igenom koden och utför våra instruktioner.

```
1 x = 5
2 print(x)
3 x = x+2
4 print(x)
```

Användning av samma variabel två gånger

Detta program skriver också ut 5 och sen 7. Det beror på att raden x=x+2 betyder att vi först räknar ut högersidan, vilket blir 5+2 eftersom x, när programmet ska köra rad 3, har värdet 5. Vi sätter sedan detta värde som nytt värde på variabeln x. Det går med andra ord bra att spara ett nytt värde i en variabel som beror på det gamla värdet. Det är också ganska vanligt att man vill göra just det, exempelvis för att öka en variabels värde med Ett, vilket vi alltså kan åstadkomma genom `x=x+1`.

2.2.5 Strängkonkatenering

Om vi har sparat textvärden, det vill säga strängar, i våra variabler kan vi sätta ihop texterna med + tecknet. Detta kallas konkatenering.

```
1 namn = "Kalle"
2 efternamn = "Anka"
3 hela = namn + efternamn
4 print(hela)
```

Konkatenering av strängar

Output

```
KalleAnka
```

I programmet ser vi att + tecknet används också för texter för att konkatenera eller slå ihop dom. Vi sparar resultatet i en ny variabel vi döpt till hela, som vi sedan skriver ut med `print(hela)`. Vi hade också kunnat addera textsträngarna direkt med `print("Kalle"+"Anka")`.

Konkatenering

Konkatenering av texter betyder att vi sätter ihop texterna direkt efter varandra. Konkatenering av texten **hej** och texten **då** blir **hejdå**.

2.2.6 Övningar

Uppgift 4 Tilldela värdet 7 till en variabel **a** och värdet 5 till en variabel med namnet **b**. Skriv sen ut summan av dem.

Lösningar på sida 63 ◄

Uppgift 5 Deklarera en variabel `x` och tilldela den värdet `10`. Ändra sedan värdet på `x` till `20` och skriv ut det nya värdet.

Lösningar på sida 63 ◄

Uppgift 6 Skapa en variabel med ditt namn och skriv ut följande med hjälp av variabeln: `Hej, [ditt namn]!`

Lösningar på sida 64 ◄

För fler kodexempel se appendix A.2

2.3 If-satser (Beslutsfattande)

2.3.1 Vad är en `if`-sats?

I programmering används `if`-satser för att fatta beslut baserat på vissa villkor. En `if`-sats gör att programmet endast kör en viss del av koden om ett villkor är sant.

```
x = 10
if x > 5:
    print("x är större än 5")
```

Exempel på if-sats

I detta exempel kontrollerar programmet om värdet på `x` är större än 5. Om villkoret är sant (vilket det är, eftersom `x = 10`), skriver programmet ut `x är större än 5`.

Villkor

Ett villkor är ett påstående som är antingen sant eller falskt, till exempel `x > 5`.

Observera!

En if-sats har alltid ett villkor följt av tecknet **:**
Därefter kommer en eller flera rader som är högerjusterade (se indentering nedan).

2.3.2 Indentering i koden

Python använder indragning eller högerjusterad kod för att markera vilken kod som hör till en `if`-sats. All kod som är indragen efter en `if`-sats körs om villkoret är sant. Detta kallas för indentering och markerar alltså vilken kod som tillhör vilket kodblock.

```
x = 10
if x > 5:
    print("Detta är indraget")
    print("Det körs om villkoret är sant")
```

Indragning är viktigt

I programmet är båda print indenterade vilket gör att dom bara körs när villkoret **x är större än 5** är sant.

Indentering

Högerjusterad eller indragen kod, vilket används i python för att markera vilken kod som tillhör ett kodblock. Exempelvis vilken kod som är inuti en if-sats.

2.3.3 Jämförelseoperatorer

För att skapa villkor används jämförelseoperatorer, som till exempel:

== Är lika med (till exempel `x == 5`)

!= Är inte lika med (till exempel `x != 5`)

> Större än

< Mindre än

>= Större än eller lika med

<= Mindre än eller lika med

Observera!

== betyder att vi jämför om två saker är likamed varandra. Jämför med ett likamedtecken (=) som används för att ge ett variabel ett värde (tilldelning). Vi använder == när vi gör en jämförelse för att inte förväxla med variabel-tilldelning.

```
x = 8
if x != 5:
    print("x är inte lika med 5")
```

Exempel med jämförelseoperatorer

Här är villkoret sant eftersom `x = 8` inte är lika med `5`, så texten skrivs ut.

2.3.4 `else` och `elif`

Ibland vill vi hantera flera olika fall. Då kan vi använda `else` för att göra något när villkoret inte är sant, eller `elif` (som står för ”else if”) för att lägga till fler villkor.

```
x = 10
if x < 5:
    print("x är mindre än 5")
elif x == 10:
    print("x är exakt 10")
else:
    print("x är större än 5 men inte 10")
```

Exempel med else och elif

När detta körs, skrivs `x är exakt 10` ut, eftersom det andra villkoret (`x == 10`) är sant.

Observera!

En elif körs bara om alla villkor ovanför är falska.

else

En `else`-sats körs när inget av de tidigare villkoren i en `if`-sats är sanna.

elif

En `elif`-sats används för att lägga till ytterligare villkor till en `if`-sats.

2.3.5 Övningar

Uppgift 7 Skriv en `if`-sats som kontrollerar om ett tal är större än 100. Om villkoret är sant, skriv ut `Talet är stort.`

Lösningar på sida 64 ◀

Uppgift 8 Använd en `if`-sats med `else`. Skriv ett program som kontrollerar om ett tal är negativt eller positivt, och skriver ut lämpligt meddelande.

Lösningar på sida 64 ◀

Uppgift 9 Skapa ett program som använder `elif` för att skriva ut olika meddelanden beroende på ålder: - Om åldern är under 13, skriv `Du är ett barn.` - Om åldern är mellan 13 och 19 (inklusive), skriv `Du är en tonåring.` - Annars skriv `Du är vuxen.`

Lösningar på sida 64 ◀

För fler kodexempel se appendix A.3

2.4 Input (Att ta emot data från användaren)

2.4.1 Vad är input?

I Python används funktionen `input()` för att ta emot data från användaren. Med `input()` kan program bli interaktiva genom att ställa frågor eller vänta på användarens svar.

```
1 name = input("Vad heter du? ")
2 print("Hej, " + name + "!")
```

Exempel på input

I det här exemplet: - Användaren blir tillfrågad `"Vad heter du?"`. - Svaret lagras i variabeln `name`. - Programmet använder `print()` för att hälsa på användaren med det inskrivna namnet.

Input
Data som användaren skickar in i programmet, ofta via tangentbordet.

2.4.2 Arbeta med olika datatyper

Funktionen `input()` returnerar alltid text (strängar). Om du vill använda det inskrivna värdet som ett tal måste du omvandla det med `int()` (heltal) eller `float()` (decimaltal).

```
1 age = int(input("Hur gammal är du? "))
2 age = age + 1
3 print("Nästa år fyller du " + str(age) + " år.")
```

Exempel på input med tal

I detta exempel:

1. `input()` tar emot en text. Det vill säga en string.
2. `int()` konverterar string-värdet till ett heltal. Eftersom vi omvandlat våran variabel till en int kan vi sen göra en beräkning med den **age=age+1**.
3. `str()` används för att omvandla talet tillbaka till text i utskriften.

2.4.3 Övningar

Uppgift 10 Skriv ett program som frågar användaren efter deras favoritfärg och sedan skriver ut `Din favoritfärg är [färgen]"`.

Lösningar på sida 64 ◀

Uppgift 11 Skapa ett program som frågar användaren efter två tal och sedan skriver ut summan av dem.

Lösningar på sida 64 ◀

Uppgift 12 Skriv ett program som frågar användaren efter deras ålder. Om åldern är mindre än 18, skriv ut `Du är inte myndig.`". Annars, skriv ut `Du är myndig.`".

Lösningar på sida 64 ◀

2.4.4 Sammanfattning

- Funktionen `input()` används för att ta emot data från användaren.
- Data från `input()` är alltid text (strängar). Omvandling behövs för att arbeta med tal.

För fler kodexempel se appendix A.4

2.5 Kodkommentarer

Kodkommentarer är ett viktigt verktyg för att göra din kod lättare att förstå för både dig själv och andra som läser den. Kommentarer används för att förklara vad koden gör och varför den gör det, utan att påverka programmets körning. Kommentarer kan också användas för att tillfälligt inaktivera kod under utveckling eller felsökning.

2.5.1 Enkla kommentarer

I Python skrivs kommentarer med ett #-tecken i början av raden. Allt efter # på samma rad ignoreras av Python och påverkar inte programmets exekvering. Här är ett exempel:

```
# Detta är en kommentar
print("Hello, world!")  # Detta är också en kommentar
```

Exempel på en enkel kommentar

I exemplet ovan används kommentarer för att förklara kodraderna. Kommentarerna är bara för människor att läsa och påverkar inte hur programmet fungerar.

Kommentar
En kommentar är en rad i koden som inte påverkar körningen av programmet. Den används för att förklara vad koden gör.

2.5.2 Kommentarer för att förklara kod

Kommentarer hjälper till att förklara mer komplex kod och gör det lättare för andra att förstå vad du har gjort. Här är ett exempel på hur kommentarer kan användas för att beskriva vad en kodbit gör:

```
# Skapa en variabel för användarens ålder
alder = 25

# Om användaren är 18 år eller äldre
if alder >= 18:
    print("Du är vuxen")
else:
    print("Du är minderårig")
```

Kommentarer för att förklara kod

I detta exempel förklarar kommentarerna vad varje del av koden gör, vilket gör det lättare att förstå syftet med varje kodrad.

2.5.3 Multiradskommentarer

Python har inte en specifik syntax för multiradskommentarer, men det finns två sätt att skriva kommentarer på flera rader. Den ena metoden är att använda flera #-tecken, en per rad:

```
# Detta är en kommentar
# som sträcker sig över
# flera rader
```

Exempel på multiradskommentarer med #

En annan metod för att skriva kommentarer på flera rader är att använda en trippel-citat”(`'''` eller `"`):

```
"""
Det här är en kommentar
som kan sträcka sig
över flera rader.
"""
```

Exempel på multiradskommentarer med trippel-citat

Observera att trippel-citat inte tekniskt sett är kommentarer, utan strängar som inte används. De används ofta för dokumentation, men fungerar också bra som multiradskommentarer under utveckling.

2.5.4 Bra praxis för kommentarer

Även om kommentarer är bra, bör du undvika att kommentera uppenbar kod. Kommentera endast när det behövs för att förklara varför något görs, särskilt om koden är komplex eller otydlig.

- Kommentera inte varje rad kod. Förklara varför koden gör något, inte vad den gör om det är uppenbart.
- Skriv kortfattat men tydligt. Kommentarer ska vara lätta att förstå på första läsningen.
- Använd kommentarer för att förklara logik som kan vara svår att förstå eller som har särskild betydelse.

2.5.5 Alternativ användning av kodkommentarer

Två alternativa användningsssätt för kodkommentarer är under felsökning, eller för att markera saker som du som programmerare ska programmera senare (En så kallad **ToDo**).

```
print("Välkommen till frågesportsspelet")
print("Vad heter Sveriges huvudstad?")
svar = input()
```

```
#TODO Gör om svaret till bara små bokstäver
if svar == "stockholm":
  print("Grattis")
else:
  print("Tyvärr fel")
```

ToDo

I programmet har vi skrivit en `#TODO` kommentar vilket används för att komma ihåg saker som behöver programmeras framöver. En TODO-kommentar skiljer sig inte från en vanlig kommentar utan är bara en vanligt förekommande användning av kommentarer hos programmerare.

```
print("Välkommen till frågesportsspelet")
print("Vad heter Sveriges huvudstad?")
svar = input()
#svar.Lower()
if svar == "stockholm":
  print("Grattis")
else:
  print("Tyvärr fel")
```

Felsökning genom kodkommentarer

I programmet ovan har en kommentar används för att tillfälligt inaktivera en rad kod som verkar generera ett fel när vi kör programmet. Eftersom vi satt ett # tecken på raden kör python den inte. Genom att kommentera flera rader kan vi i ett avancerat program lista ut var något går fel. Det är generellt inte rekommenderat då det finns bättre sätt att felsöka program, men eftersom metoden är så pass vanligt förekommande är det något man bör känna till. För enklare program kan det också gå snabbast att göra på det viset. Programmet blir korrekt om vi ändrar svar.Lower() till svar.lower()

2.5.6 Övningar

Uppgift 13 Skriv ett program som skriver ut användarens namn och ålder. Lägg till kommentarer för att förklara varje steg i programmet.

Lösningar på sida 64 ◀

Uppgift 14 Lägg till kommentarer i ett program som kontrollerar om ett tal är jämnt eller udda. Kommentera varje steg för att förklara vad som händer.

Lösningar på sida 65 ◀

För fler kodexempel se appendix A.5

2.6 Att använda slumptal och modulen `random`

2.6.1 Vad är en modul?

En modul i Python är en samling fördefinierad kod som vi kan använda i våra program. Moduler är som bibliotek som innehåller användbara funktioner och verktyg som kan spara oss tid och arbete. För att använda en modul måste vi först **importera** den.

```
1 import random
```

Exempel på import

I detta exempel importeras modulen `random`, som innehåller funktioner för att arbeta med slumpmässighet.

Modul
En modul är en samling färdiga funktioner och verktyg som vi kan använda i våra program.

2.6.2 Slumptal med `random.randint()`

Funktionen `randint()` från modulen `random` genererar ett heltal mellan två givna värden. Det är användbart när vi vill simulera slumpmässiga val, som att kasta en tärning.

```
1 import random
2
3 tarning = random.randint(1, 6)
4 print("Du slog:", tarning)
```

Exempel med `randint()`

I detta program:

- `random.randint(1, 6)` genererar ett heltal mellan 1 och 6 (inklusive båda).
- Det slumpmässiga talet lagras i variabeln `tarning` och skrivs ut med `print()`.

Slumptal
Ett nummer som genereras slumpmässigt av datorn.

Observera!
randint är en funktion på samma sätt som print. Funktionen randint finns däremot inte förinladdad när vi startar ett program och vi måste därför importera den från random-modulen. Vi ser också att vi till en funktion kan skicka in flera argument. I detta fall skickar vi in två tal som anger mellan vilka tal vi vill få ett slumpat tal. När vi skickar flera argument separerar vi dom med kommatecken.

2.6.3 Exempel: Ett enkelt spel

Här är ett program som använder `randint()` för att skapa ett enkelt spel där användaren ska gissa ett tal.

```
import random

hemligt_tal = random.randint(1, 10)
gissning = int(input("Gissa ett tal mellan 1 och 10: "))

if gissning == hemligt_tal:
    print("Grattis! Du gissade rätt.")
else:
    print("Fel! Det rätta talet var", hemligt_tal)
```

Gissa ett tal-spel

2.6.4 Andra användbara funktioner i `random`

Modulen `random` innehåller fler funktioner än bara `randint`.
Här är några exempel:

- `random.random()` genererar ett slumptal mellan 0 och 1.
- `random.choice()` väljer slumpmässigt ett objekt från en lista. Vi kommer kolla mer på listor senare.

```
import random

farger = ["röd", "blå", "grön", "gul"]
vald_farg = random.choice(farger)
print("Den valda färgen är:", vald_farg)
```

Exempel med `random.choice()`

För att ta reda på vad för funktioner som finns brukar det vara bäst att göra en sökning. Vanligen får man bättre resultat om man söker på engelska, exempelvis sökning ”python generate random number”.

2.6.5 Övningar

Uppgift 15 Skriv ett program som slumpar ett tal mellan 1 och 100 och skriver ut det.

Lösningar på sida 65 ◄

Uppgift 16 Skapa ett program som kastar två tärningar (med värden mellan 1 och 6) och skriver ut summan av deras resultat.

Lösningar på sida 65 ◄

Uppgift 17 Gör ett program som väljer slumpmässigt en aktivitet från en lista olika aktiviteter, till exempel ”läsa”, ”spela spel”, eller ”träna”.

Lösningar på sida 65 ◄

2.6.6 Sammanfattning

- Moduler som `random` låter oss använda fördefinierade funktioner för specifika ändamål.
- Funktionen `randint()` används för att skapa slumptal inom ett visst intervall.
- Interaktiva program kan skapas genom att kombinera `input()` och slumpfunktioner.

För fler kodexempel se appendix A.6

2.7 Flödesdiagram

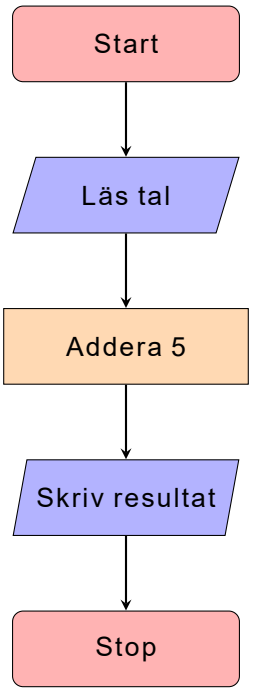
Flödesdiagram är ett visuellt verktyg för att beskriva algoritmer och programflöden. De hjälper till att strukturera och förstå hur ett program är uppbyggt och fungerar.

Flödesdiagram

Ett flödesdiagram är en grafisk representation av en process eller algoritm. Det använder symboler som representerar olika steg och pilar för att visa flödet mellan dessa steg.

Symboler i flödesdiagram I flödesdiagram används följande standardiserade symboler:
- **Oval (start/stop):** Representerar början eller slutet på ett flöde.
- **Rektangel (process):** Representerar ett steg i processen, till exempel en beräkning.
- **Parallelltrapets (in-/utdata):** Representerar inmatning eller utmatning, t.ex. att läsa in eller skriva ut data.
- **Romb (beslut):** Representerar ett beslut, t.ex. ett val baserat på en if-sats.
Exempel: Enkel sekvens

Flödesdiagram:



För att följa ett flödesdiagram så börjar vi i start-rutan, och följer pilarna. Vid varje ny symbol så utför vi det som står där, exempelvis fråga användaren efter ett tal. Så småningom så kommer vi nå stop-symbolen och då är programmet slut. Flödesdiagram är till för att beskriva ett program, men behöver inte skrivas på ett lika exakt sätt som pythonkod. Vi kan därför skriva saker som ”Läs tal” och räkna med att den som läser flödesdiagrammet förstår att när vi sen skriver ”Addera 5” menar vi ”Addera 5 till talet och spara det nya värdet till tal”. Hur

formellt och noga man ska skriva sina flödesdiagram är delvis en smaksak. Målet är däremot att det ska vara tydligt hur det fungerar. Kanske tycker du inte det är självklart vad ”Addera 5” innebär i flödesdiagrammet ovan.

Vi kan också översätta vårat flödesdiagram till pythonkod:

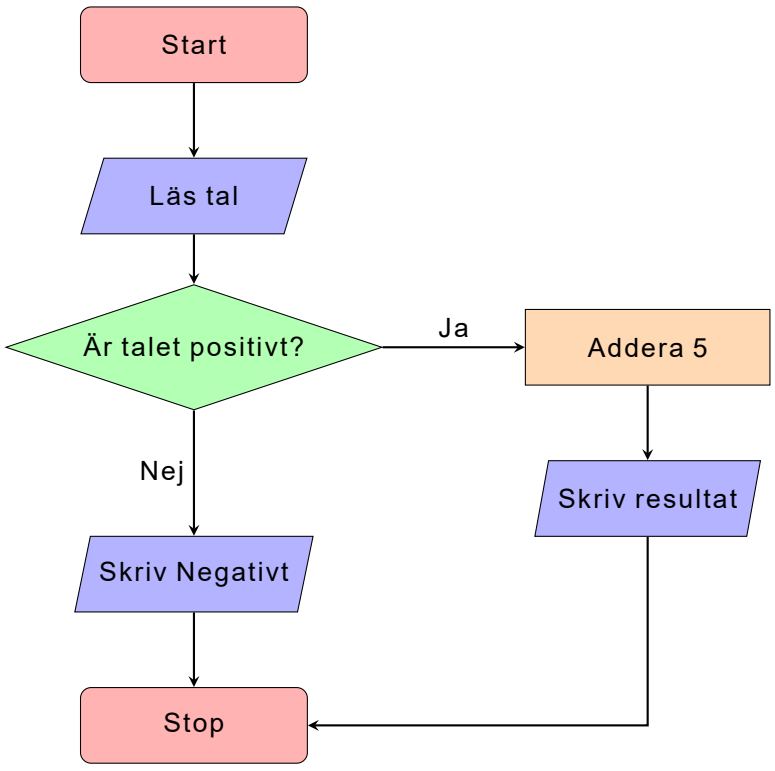
Motsvarande Python-kod:

```python
# Läs ett tal från användaren
nummer = int(input("Ange ett tal: "))

# Lägg till 5
resultat = nummer + 5

# Skriv ut resultatet
print("Resultatet är:", resultat)
```

Exempel output
Ange ett tal: 10
Resultatet är: 15

Exempel: Beslutsstruktur I många program måste beslut tas beroende på data. Beslutsstrukturer visas i flödesdiagram med en romb.

Flödesdiagram:



I programmet ser du romb eller diamantsymbolen som motsvarar en if-sats i programmeringskod. Det är den ända symbolen i flödesdiagram som det kan gå ut mer än en pil från. Notera också att pilarna som går ut från diamantsymbolen är markerade med Ja och Nej för att visa vid vilka fall vi ska gå vilken väg.

I detta fall så går vi på Ja-pilen om talet är positivt. Eller Nej pilen om talet inte är positivt. Det som står i en diamantsymbol är alltså alltid en ja-nej-fråga.

Motsvarande Python-kod:

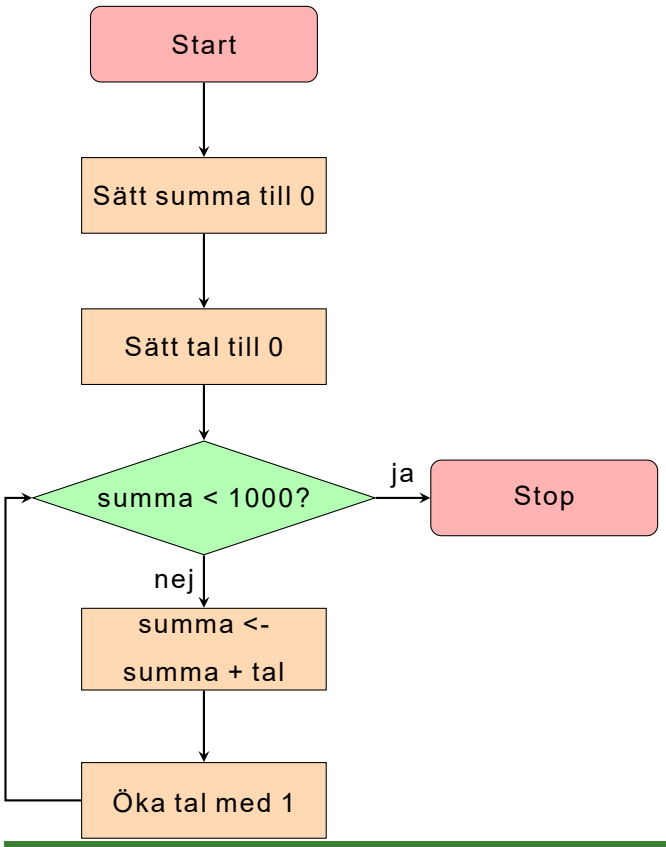
```python
# Läs ett tal från användaren
nummer = int(input("Ange ett tal: "))

if nummer > 0:
    # Om talet är positivt, addera 5
    resultat = nummer + 5
    print("Resultatet är:", resultat)
else:
    # Om talet är negativt, skriv "Negativt"
    print("Talet är negativt")
```

Exempel output 1
Ange ett tal: 3
Resultatet är: 8

Exempel output 2
Ange ett tal: -5
Talet är negativt

Om vi har ett program med en while-loop är det ända vi behöver göra att dra en pil tillbaka till ett tidigare steg. En while-loop avbryts när ett villkor är falskt så vi kan använda våran vanliga diamantsymbol. Genom att på detta sätt återanvända delar av ett flödesdiagram kan vi på samma sätt som i programmeringskod beskriva mera komplicerade processer.

Flödesdiagram:



Python-kod:

```
# Räknar 1+2+3+4+5... så länge summan är
    mindre än 1000
summa = 0
tal = 1
while summa < 1000:
    summa = summa + tal

print("Summa 1+2+3+...+",tal,">1000")
```

Listing 2.1: Summera positiva tal

Räknar ut vilket tal vi behöver räkna till så att $1+2+3+\cdots+tal > 1000$.

Observera!

I detta flödesdiagram har vi varit lite mer noga med att beskriva variabler än i tidigare exempel. Det är som sagt inte en exakt vetenskap hur det ska beskrivas, men eftersom vi i programmet håller reda på både summan av alla tal, och vilket tal vi ligger på, kan det vara bra att vara lite extra tydlig. Notera också att vi skrivit **summa <- summa + tal** istället för **summa = summa + tal**. Det är vanligt att man använder en pil för att beskriva tilldelning till en variabel när man skriver flödesdiagram, eller psuedokod vilket vi tar upp i ett senare avsnitt

Sammanfattning: Flödesdiagram är ett kraftfullt verktyg för att planera och analysera programlogik. Genom att översätta diagram till kod (och vice versa) kan du få en bättre förståelse för hur program fungerar.

För fler kodexempel se appendix A.7

2.8 Logiska uttryck och booleanska värden

2.8.1 Vad är ett logiskt uttryck?

Ett logiskt uttryck är ett uttryck som antingen är **sant** (`True`) eller **falskt** (`False`). Logiska uttryck används ofta i beslutsfattande, till exempel i `if`-satser och loopar, för att avgöra om en viss kod ska köras.

```
x = 5
y = 10
print(x < y)  # True
print(x == y) # False
```

Exempel på logiska uttryck

I exemplet ovan är `x < y` sant eftersom `5` är mindre än `10`. Däremot är `x == y` falskt eftersom `5` inte är lika med `10`.

Boolean
En datatyp i Python som kan vara `True` (sant) eller `False` (falskt). Den används i logiska uttryck.

2.8.2 Logiska operatorer

Python har tre logiska operatorer: `and`, `or`, och `not`. Dessa används för att kombinera eller invertera logiska uttryck.

and

Operatorn `and` returnerar `True` om båda uttrycken är sanna.

```
x = 5
y = 10
z = 15

print(x < y and y < z)  # True (båda villkoren är sanna)
print(x > y and y < z)  # False (det första villkoret är falskt)
```

Exempel med `and`

or

Operatorn `or` returnerar `True` om minst ett av uttrycken är sant.

```
x = 5
y = 10

print(x > y or x < y)   # True (det andra villkoret är sant)
```

```
print(x > y or y > 20)  # False (båda villkoren är falska)
```

Exempel med `or`

not

Operatorn `not` inverterar värdet av ett logiskt uttryck.

```
x = 5
y = 10

print(not x < y)  # False (inverterar värdet av "x < y", som är True)
```

Exempel med `not`

2.8.3 Praktiskt exempel: Kontrollera behörighet

Här är ett exempel där logiska operatorer används för att avgöra om en person är behörig att rösta.

```
alder = int(input("Hur gammal är du? "))

if alder >= 18 and alder < 120:
    print("Du får rösta!")
else:
    print("Du är inte behörig att rösta.")
```

Exempel med behörighetskontroll

I detta program kontrolleras om åldern är 18 år eller äldre och mindre än 120.

2.8.4 Övningar

Uppgift 18 Skriv ett program som frågar användaren efter ett nummer. Programmet ska skriva ut `True` om numret är mellan 10 och 20, annars `False`.

Lösningar på sida 65 ◀

Uppgift 19 Skriv ett program som frågar användaren efter två tal. Programmet ska skriva ut `True` om minst ett av talen är större än 50.

Lösningar på sida 65 ◀

Uppgift 20 Skriv ett program som frågar användaren efter ett lösenord. Om lösenordet är `hemligt123` ska programmet skriva `Access granted`, annars `Access denied`.

Lösningar på sida 66 ◀

För fler kodexempel se appendix A.8

2.9 Matematik i Python: `math`-modulen och specialoperatorer

Python har inbyggda verktyg för matematiska beräkningar. Förutom grundläggande operatorer som `+`, `-`, `*`, och `/` finns fler avancerade funktioner i `math`-modulen. Här ska vi även titta på specialoperatorerna `**`, `%`, och `//`.

2.9.1 `math`-modulen

`math`-modulen är en inbyggd modul i Python som erbjuder en mängd matematiska funktioner och konstanter.

Importera `math`-modulen

För att använda modulen måste den importeras i din kod:

```
import math
```

Exempel: Importera `math`

Efter att du har importerat `math` kan du använda dess funktioner.

Vanliga funktioner i `math`

Här är några av de mest använda funktionerna i `math`:

```
import math

# Roten ur ett tal
print(math.sqrt(16))  # 4.0

# Upphöjning till en viss potens
print(math.pow(2, 3))  # 8.0

# Heltalsdel av en division
print(math.floor(7.8))  # 7
print(math.ceil(7.2))   # 8

# Värdet av pi och e
print(math.pi)  # 3.141592653589793
print(math.e)   # 2.718281828459045
```

Exempel på `math`-funktioner

Modul

En modul är ett bibliotek av funktioner och variabler som kan importeras och användas i ett program.

2.9.2 Specialoperatorer

Python har flera operatorer som är användbara för matematiska beräkningar utöver de grundläggande.

** (Exponentiering)

Operatorn ** används för att upphöja ett tal till en viss potens.

```
print(2 ** 3)  # 8 (2 upphöjt till 3)
print(5 ** 2)  # 25 (5 upphöjt till 2)
```

Exempel på **

% (Modulus)

Operatorn % returnerar resten vid en division.

```
print(10 % 3)  # 1 (10 dividerat med 3 ger resten 1)
print(15 % 4)  # 3 (15 dividerat med 4 ger resten 3)
```

Exempel på %

Modulus
En operation som returnerar resten vid division av två tal.

// (Heltalsdivision)

Operatorn // utför division och returnerar endast heltalsdelen av resultatet.

```
print(10 // 3)  # 3 (heltalsdelen av 10 dividerat med 3)
print(15 // 4)  # 3 (heltalsdelen av 15 dividerat med 4)
```

Exempel på //

Heltalsdivision
En operation som returnerar heltalsdelen av en division, utan decimaler.

2.9.3 Praktiskt exempel: Beräkna cirkelns area

Här använder vi math-modulen och en av specialoperatorerna för att beräkna en cirkels area.

```
import math

# Fråga användaren efter radien
radie = float(input("Ange radien: "))
```

```
5
6 # Beräkna arean
7 area = math.pi * (radie ** 2)
8 print("Cirkelns area är:", area)
```

Beräkna cirkelns area

2.9.4 Övningar

Uppgift 21 Skriv ett program som använder `math.sqrt()` för att beräkna kvadratroten av ett tal som användaren anger.

Lösningar på sida 66 ◄

Uppgift 22 Skriv ett program som använder % för att avgöra om ett tal som användaren anger är jämnt eller udda.

Lösningar på sida 66 ◄

Uppgift 23 Skriv ett program som frågar användaren efter två tal och skriver ut resultatet av // och %.

Lösningar på sida 66 ◄

För fler kodexempel se appendix A.9

Kapitel 3

Funktioner och datastrukturer

3.1 Listor i Python

Listor är en av de mest användbara datatyperna i Python. De används för att lagra flera värden i en och samma variabel. Till exempel kan en lista innehålla en samling av tal, strängar eller till och med andra listor.

3.1.1 Skapa en lista

En lista skapas genom att använda hakparenteser [] och separera elementen med kommatecken:

```
# En lista med tal
tal = [1, 2, 3, 4, 5]

# En lista med text
djur = ["katt", "hund", "kanin"]

# En blandad lista
blandad = [1, "äpple", True]
```

Exempel: Skapa en lista

Lista
En lista är en samling av värden som kan lagras i en variabel.

Element
Ett värde i listan kallas för ett element

3.1.2 Åtkomst till element i en lista

Varje element i en lista har ett index som börjar på `0` för det första elementet. Du kan komma åt ett element genom att ange dess index inom hakparenteser:

```
djur = ["katt", "hund", "kanin"]
print(djur[0])  # Skriv ut "katt"
print(djur[1])  # Skriv ut "hund"
```

Exempel: Åtkomst via index

Index
Index är platsen på ett av listans värden. I python anges första elementet med index 0.

3.1.3 Modifiera en lista

Du kan ändra värdet på ett element, lägga till nya element eller ta bort befintliga element.

Ändra värde:

```
djur = ["katt", "hund", "kanin"]
djur[1] = "hamster"  # Ändra "hund" till "hamster"
print(djur)  # ["katt", "hamster", "kanin"]
```

Exempel: Ändra värde

Lägga till element:

```
djur = ["katt", "hund"]
djur.append("kanin")  # Lägg till "kanin" sist i listan
print(djur)  # ["katt", "hund", "kanin"]
```

Exempel: Lägga till element

Ta bort element:

```
djur = ["katt", "hund", "kanin"]
djur.remove("hund")  # Ta bort "hund"
print(djur)  # ["katt", "kanin"]
```

Exempel: Ta bort element

3.1.4 Loopar genom en lista

Det är vanligt att använda loopar för att bearbeta alla element i en lista. Vi kommer i nästa avsnitt se att man vanligen gör detta med en for-loop. Vi använder i detta exempel en while-loop eftersom det är vad vi sett hittills.

```
djur = ["katt", "hund", "kanin"]
i = 0
while i<djur.len(): #Kör loopen tills i når slutet på listan
  print(djur[i])
  i = i + 1

# Skriver ut:
# katt
# hund
# kanin
```

Exempel: For-loop med en lista

Notera att vi kör programmet till variabeln **i** har ett värde som är ett mindre än listans längd. Eftersom vi börjar räkna på 0 får vi ändå med alla tal. Inuti loopen använder vi våran variabel **i** för att skriva ut ett element i listan i taget.

Observera!
Vi använder vanligen en variabel med namnet **i** för att beskriva ett index

3.1.5 Index out of bounds

Ett vanligt fel när man jobbar med listor är ett ”Index out of bounds”-error. Det betyder att vi har försökt hämta ett element (Ett värde i listan) som ligger utanför listan. Om vi i en lista med 4 element försöker hämta det 5 elementet får vi det felet.

```
1 tal = [1,2,3,4]
2 print(tal[5])
```

Index out of bounds error

Index out of bounds-error
En typ av fel vi får när vi försöker hämta ett element från en lista som ligger utanför listan.

3.1.6 Vanliga metoder för listor

Python erbjuder flera metoder för att arbeta med listor:

- `append()`: Lägg till ett element i slutet.
- `remove()`: Ta bort ett specifikt element.
- `pop()`: Ta bort och returnera det sista elementet (eller ett specifikt index).
- `sort()`: Sortera listan.
- `len()`: Returnera antalet element i listan.

```
1 djur = ["kanin", "hund", "katt"]
2 djur.sort()  # Sortera listan
3 print(djur)  # ["hund", "kanin", "katt"]
```

Exempel: Använda lista-metoder

3.1.7 Övningar

Uppgift 24 Skapa en lista med namnen på tre frukter och skriv ut dem en och en med en loop.

Lösningar på sida 66 ◀

Uppgift 25 Skapa en lista med siffrorna 1 till 5. Lägg till talet 6 och skriv sedan ut hela listan.

Lösningar på sida 67 ◄

Uppgift 26 Skapa en lista med talen 10, 20 och 30. Använd `len()` för att skriva ut antalet element i listan.

Lösningar på sida 67 ◄

För fler kodexempel se appendix **??**

3.2 For-loop (Upprepa saker med Python)

En `for`-loop används när vi vill upprepa något för varje sak i en lista, varje bokstav i en text, eller över en rad nummer. Tanken är enkel: loopen går igenom varje sak i listan, en i taget, och gör något med den.

3.2.1 Hur fungerar en for-loop?

En `for`-loop i Python ser ut så här:

```
for variabel in lista:
    # Gör något med variabeln
```

Syntax för for-loop

Vad händer steg för steg?

1. **Variabeln** är ett namn vi väljer. Den kommer att innehålla en sak från listan i taget.
2. **Listan** är det vi vill gå igenom, t.ex. en samling siffror eller bokstäver.
3. Loopen tar varje sak i listan och lagrar den i variabeln. Sedan kör den koden inuti loopen.

Exempel: Skriv ut siffror i en lista

Här är ett enkelt exempel där vi har en lista med siffror:

```
tal_lista = [1, 2, 3, 4, 5]

for tal in tal_lista:
    print(tal)
```

Exempel: Skriv ut siffror

Vad händer när programmet körs?

1. Första gången i loopen är `tal = 1`. Koden `print(tal)` körs och skriver ut 1.
2. Sedan hoppar loopen till nästa tal i listan, `tal = 2`, och skriver ut 2.
3. Detta fortsätter för alla tal i listan.

Resultatet blir:

Exempel: Output

```
1
2
3
4
5
```

Observera!

Det är viktigt att koden inuti loopen är indenterad (indragen), annars får du fel. Python använder indrag för att veta vad som hör till loopen.

3.2.2 Ett tydligare exempel: Steg för steg

Låt oss se ett ännu tydligare exempel. Här använder vi en lista med namn:

```
namn_lista = ["Anna", "Björn", "Cecilia"]

for namn in namn_lista:
    print("Hej, " + namn + "!")
```

Exempel: Skriv ut namn

Vad händer när programmet körs?

1. **Första varvet**: Variabeln `namn` blir `''Anna''`. Programmet skriver ut `"Hej, Anna!"`.
2. **Andra varvet**: Variabeln `namn` blir `''Björn''`. Programmet skriver ut `"Hej, Björn!"`.
3. **Tredje varvet**: Variabeln `namn` blir `''Cecilia''`. Programmet skriver ut `"Hej, Cecilia!"`.

Outputen blir:

Exempel: Output
Hej, Anna!
Hej, Björn!
Hej, Cecilia!

3.2.3 For-loop med `range()`

Vi kan också använda `range()` för att skapa en serie siffror automatiskt. Till exempel kan vi skriva ut talen från 1 till 5 utan att skapa en lista först:

```
for tal in range(1, 6):
    print(tal)
```

Exempel: range()

Här skapar `range(1, 6)` siffrorna 1 till 5 (men inte 6). Resultatet blir detsamma som med listan ovan.

Steg i `range()`

Du kan lägga till ett steg i `range()` för att hoppa över tal. Till exempel:

```
for tal in range(2, 11, 2):
    print(tal)
```

Här hoppar `range()` två steg åt gången och skriver ut 2, 4, 6, 8, 10.

3.2.4 For-loop med text

En sträng i Python kan behandlas som en lista av bokstäver. Vi kan använda en `for`-loop för att skriva ut varje bokstav i en sträng:

```
1 text = "Python"
2
3 for bokstav in text:
4     print(bokstav)
```

Exempel: Loopa över en sträng

Outputen blir:

Exempel: Output

```
P
y
t
h
o
n
```

3.2.5 Övningar

Uppgift 27 Skriv ett program som skriver ut alla siffror från 1 till 10, en per rad.

Lösningar på sida 67 ◀

Uppgift 28 Skriv ett program som skriver ut alla multiplar av 3 mellan 1 och 30.

Lösningar på sida 67 ◀

Uppgift 29 Skriv ett program som skriver ut varje bokstav i strängen ”Programmering” på en ny rad.

Lösningar på sida 67 ◀

För fler kodexempel se appendix **??**

3.3 Dictionary: En samling av nyckel-värde-par

En **dictionary** (eller ”ordbok”) i Python är en datastruktur som används för att lagra data i form av nyckel-värde-par. Detta innebär att varje värde i en dictionary är kopplat till en unik nyckel, som används för att referera till värdet. Detta gör dictionaries särskilt användbara när vi behöver hantera data som har en logisk koppling, till exempel namn och telefonnummer eller ord och deras definitioner.

3.3.1 Skapa en dictionary

En dictionary skapas genom att använda klamrar `{}` och separera nycklar och värden med kolon (:). Här är ett exempel:

```
# Skapa en dictionary med information om en person
person = {
    "namn": "Alice",
    "ålder": 25,
    "stad": "Stockholm"
}
```

Skapa en dictionary

3.3.2 Åtkomst till värden

För att hämta ett värde från en dictionary använder vi nyckeln inom hakparenteser `[]`:

```
# Hämta värdet för nyckeln "namn"
print(person["namn"])  # Output: Alice
```

Hämta värden från en dictionary

Om vi försöker använda en nyckel som inte finns i dictionaryn, kommer Python att generera ett `KeyError`. För att undvika detta kan vi använda metoden `get()`, som låter oss ange ett standardvärde:

```
# Försök hämta en nyckel som inte finns
print(person.get("jobb", "Okänd"))  # Output: Okänd
```

Använda get-metoden

3.3.3 Lägga till och ändra värden

Vi kan lägga till nya nyckel-värde-par eller ändra existerande värden:

```
# Lägg till en ny nyckel "jobb"
person["jobb"] = "Programmerare"

```

```
# Ändra värdet för nyckeln "stad"
person["stad"] = "Göteborg"

print(person)
```

Lägga till och ändra värden

3.3.4 Ta bort värden

Vi kan använda pop() för att ta bort ett nyckel-värde-par:

```
# Ta bort nyckeln "ålder"
person.pop("ålder")

print(person)
```

Ta bort värden från en dictionary

3.3.5 Loopa genom en dictionary

Vi kan loopa igenom en dictionary för att få åtkomst till dess nycklar och värden:

```
# Loopa genom nycklar och värden
for nyckel, värde in person.items():
    print(f"{nyckel}: {värde}")
```

Loop genom en dictionary

3.3.6 Exempel på användning

Här är ett exempel på hur dictionaries kan användas för att lagra och analysera data:

```
# Räkna antalet förekomster av varje bokstav
text = "programmering"
bokstavsfrekvens = {}

for bokstav in text:
    bokstavsfrekvens[bokstav] = bokstavsfrekvens.get(bokstav, 0) + 1

print(bokstavsfrekvens)
```

Exempel: Räkna antalet bokstäver i en text

Output

```
{'p': 1, 'r': 3, 'o': 1, 'g': 2, 'a': 1, 'm': 2, 'e': 1, 'i': 1, 'n': 1}
```

3.3.7 Övningar

Uppgift 30 Skapa en dictionary som innehåller tre städer och deras befolkning. Skriv ut befolkningen för en av städerna.

Lösningar på sida 67 ◄

Uppgift 31 Skriv ett program som använder en dictionary för att lagra elevbetyg. Programmet ska kunna lägga till nya elever och deras betyg samt visa alla sparade betyg.

Lösningar på sida 68 ◄

Uppgift 32 Skapa ett program som räknar hur många gånger varje bokstav förekommer i en given mening. Implementera detta genom att använda en `for`-loop och metoden `get()`.

Lösningar på sida 68 ◄

För fler kodexempel se appendix **??**

3.4 Funktioner: Strukturera och Återanvänd Kod

Funktioner är en grundläggande del av programmering. De låter oss strukturera kod genom att dela upp den i mindre, hanterbara delar. Funktioner kan också återanvändas, vilket gör vårt arbete effektivare och vår kod lättare att förstå och underhålla.

3.4.1 Vad är en funktion?

En funktion är en bit kod som utför en specifik uppgift. Vi definierar funktioner med nyckelordet `def` och ett namn, följt av eventuella parametrar inom parentes. Funktionen kan köras genom att vi anropar dess namn.

```
def hälsa():
    print("Hej! Välkommen till Python.")

hälsa()  # Anropa funktionen
```

En enkel funktion

3.4.2 Parametrar och argument

Vi kan ge funktioner indata genom att använda parametrar. När vi anropar funktionen skickar vi värden, som kallas argument, till parametrarna.

```
def hälsa(namn):
    print(f"Hej, {namn}! Välkommen till Python.")

hälsa("Alice")  # Output: Hej, Alice! Välkommen till Python.
```

Funktion med parametrar

Parameter
En parameter är en plats för indata i en funktions definition.

Argument
Ett argument är det faktiska värde vi skickar till en funktions parameter.

3.4.3 Returvärden

En funktion kan också ge tillbaka ett värde till det ställe där den anropas. Detta görs med nyckelordet `return`.

```
def addera(a, b):
    return a + b

```

```
resultat = addera(5, 7)
print(resultat)  # Output: 12
```

Funktion med returvärde

Returvärde
Ett returvärde är det värde som en funktion skickar tillbaka till anropsplatsen med hjälp av `return`.

3.4.4 Main-funktionen

I större program är det vanligt att använda en `main`-funktion för att tydligt visa var programmet börjar. Här är ett exempel:

```
def hälsa():
    print("Hej! Välkommen till Python.")

def main():
    hälsa()

if __name__ == "__main__":
    main()
```

Ett program med en main-funktion

När vi använder `if __name__ == "__main__"`, ser Python till att bara köra `main()` om programmet körs direkt, inte om det importeras i ett annat program.

Main-funktion
En `main`-funktion fungerar som programmets startpunkt, vilket gör koden tydligare och mer strukturerad.

3.4.5 Namespace: Variabler i funktioner

Varje funktion har sitt eget **namespace**, vilket innebär att variabler definierade i funktionen inte påverkar eller är kända utanför den. Här är ett exempel:

```
def ändra_värde():
    x = 10  # Lokal variabel
    print(f"Inne i funktionen: {x}")

x = 5  # Global variabel
ändra_värde()
print(f"Utanför funktionen: {x}")
```

Namespace och variabler

Output:

```
Inne i funktionen: 10
Utanför funktionen: 5
```

I detta exempel är variabeln `x` inne i funktionen skild från variabeln `x` utanför. Detta förhindrar att funktioner oavsiktligt ändrar värden i resten av programmet.

Namespace

Ett namespace är ett område där variabler och deras värden lagras. En funktions namespace är separat från resten av programmet.

3.4.6 Övningar

Uppgift 33 Skriv en funktion `dubbla()` som tar ett tal som parameter och returnerar dess dubbla värde.

Lösningar på sida 69 ◄

Uppgift 34 Skapa ett program med en `main`-funktion som anropar en funktion `hälsa(namn)` och skriver ut en personlig hälsning.

Lösningar på sida ?? ◄

Uppgift 35 Experimentera med namespace genom att skapa en funktion som definierar och ändrar en variabel. Kontrollera om förändringen påverkar en variabel med samma namn utanför funktionen.

Lösningar på sida ?? ◄

För fler kodexempel se appendix ??

Kapitel 4

Algoritmer

Introduktion till algoritmer

Algoritmer är grunden för all programmering. En algoritm är en serie steg som beskriver hur ett problem ska lösas. Genom att följa dessa steg kan vi skriva program som automatiserar beräkningar, sorterar data, söker efter information och mycket mer.

I detta kapitel kommer vi att utforska enkla och användbara algoritmer. Vi börjar med sorteringsalgoritmer, som hjälper oss att arrangera data i en viss ordning. Dessa algoritmer är inte bara praktiska, utan de lär oss också viktiga koncept som loopar, jämförelser och effektivitet.

Att förstå algoritmer handlar inte bara om att skriva kod, utan också om att tänka logiskt och lösa problem steg för steg. Oavsett om du programmerar ett spel, analyserar data eller skapar en hemsida, är algoritmer en central del av programmeringen.

4.1 Bubble Sort

Vad är Bubble Sort?

Bubble Sort är en enkel sorteringsalgoritm som fungerar genom att jämföra två intilliggande element i en lista och byta plats på dem om de är i fel ordning. Den upprepar detta tills hela listan är sorterad. Namnet Bubble Sort"kommer från att det största (eller minsta) elementet bubblar"upp till sin rätta plats i varje iteration.

Hur fungerar Bubble Sort?

Algoritmen går igenom listan flera gånger:

- Jämför två intilliggande element.
- Om de är i fel ordning, byt plats på dem.
- Fortsätt jämföra nästa par tills slutet av listan är nådd.
- Upprepa detta tills hela listan är sorterad.

Kodexempel

Här är en implementation av Bubble Sort i Python:

```python
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n-i-1):
            if lista[j] > lista[j+1]:
                # Byt plats
                lista[j], lista[j+1] = lista[j+1], lista[j]

# Exempel
data = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(data)
print("Sorterad lista:", data)
```

Bubble Sort i Python

Förklaring av koden

1. `for i in range(n)`: Loopar genom listan flera gånger.
2. `for j in range(0, n-i-1)`: Ser till att algoritmen inte jämför element som redan är sorterade.
3. `if lista[j] > lista[j+1]`: Kontrollerar om elementen är i fel ordning.
4. `lista[j], lista[j+1] = lista[j+1], lista[j]`: Byter plats på elementen.

Visualisering

Bubble Sort kan visualiseras som att sorteringen sker steg för steg, där varje större element flyter upptill toppen.

Övning

Uppgift 36 Implementera Bubble Sort för en lista som innehåller följande värden: `[10, 8, 2, 7, 1, 3]` och skriv ut den sorterade listan.

Lösningar på sida 69 ◄

Nackdelar med Bubble Sort

Bubble Sort är enkel att förstå, men den är inte särskilt effektiv för långa listor eftersom den har en tidskomplexitet på $O(n^2)$. Detta innebär att tiden det tar att sortera listan ökar kvadratiskt med dess storlek. För större datamängder är mer avancerade algoritmer, som Quick Sort eller Merge Sort, bättre val.

För fler kodexempel se appendix A.10

4.2 Sökning

När vi arbetar med data är det vanligt att vi behöver hitta ett specifikt värde i en samling, som en lista. Det finns olika sätt att göra detta, beroende på hur data är organiserad. Två grundläggande sökalgoritmer är **linjär sökning** och **binär sökning**.

4.2.1 Linjär sökning

Linjär sökning är den enklaste sökalgoritmen. Den går igenom varje element i listan, ett i taget, tills det hittar det sökta värdet eller når slutet av listan.

- Fördel: Fungerar på både sorterade och osorterade listor.
- Nackdel: Kan vara långsam för långa listor eftersom varje element måste kontrolleras.

Kodexempel

Här är ett exempel på linjär sökning i Python:

```python
def linjar_sokning(lista, mål):
    for index, värde i enumerate(lista):
        if värde == mål:
            return index  # Returnerar positionen
    return -1  # Returnerar -1 om värdet inte finns

# Exempel
data = [10, 20, 30, 40, 50]
print(linjar_sokning(data, 30))  # Output: 2
```

Linjär sökning

4.2.2 Binär sökning

Binär sökning är en mer effektiv algoritm för att hitta ett värde i en sorterad lista. Algoritmen delar listan på mitten och avgör om det sökta värdet är mindre eller större än mittpunkten. Därefter upprepas processen på den relevanta halvan av listan.

- Fördel: Mycket snabbare än linjär sökning för stora listor.
- Nackdel: Kräver att listan är sorterad.

Hur fungerar binär sökning?

1. Dela listan på mitten.
2. Kontrollera mittpunkten:

- Om värdet är det sökta, är vi klara.
- Om värdet är mindre än det sökta, leta i den högra halvan.
- Om värdet är större, leta i den vänstra halvan.

3. Upprepa tills värdet hittas eller listan är tom.

Kodexempel

Här är ett exempel på binär sökning i Python:

```python
def binar_sokning(lista, mål):
    vänster, höger = 0, len(lista) - 1
    while vänster <= höger:
        mitten = (vänster + höger) // 2
        if lista[mitten] == mål:
            return mitten
        elif lista[mitten] < mål:
            vänster = mitten + 1
        else:
            höger = mitten - 1
    return -1  # Returnerar -1 om värdet inte finns

# Exempel
data = [10, 20, 30, 40, 50]
print(binar_sokning(data, 30))  # Output: 2
```

Binär sökning

4.2.3 Jämförelse mellan linjär och binär sökning

Egenskap	Linjär sökning	Binär sökning
Krav på sortering	Nej	Ja
Effektivitet	Långsam för långa listor	Snabb för långa listor
Komplexitet	$O(n)$	$O(\log n)$

Tabell 4.1: Jämförelse mellan linjär och binär sökning.

4.2.4 Övning

Uppgift 37 Skriv en Python-funktion som implementerar linjär sökning och använd den för att hitta värdet 25 i listan `[5, 15, 25, 35, 45]`.

Lösningar på sida **??** ◀

Uppgift 38 Använd kodexemplet för binär sökning för att hitta värdet 50 i listan `[10, 20, 30, 40, 50]`.

Lösningar på sida **??** ◀

För fler kodexempel se appendix A.11

4.3 Pseudokod

Pseudokod är en metod för att beskriva algoritmer på ett sätt som är lätt att läsa och förstå, utan att behöva använda den exakta syntaxen i ett programmeringsspråk. Det fungerar som en mellanliggande representation som hjälper oss att planera och strukturera vår kod innan vi implementerar den.

4.3.1 Varför använda pseudokod?

Pseudokod är användbart eftersom det:

- Hjälper till att fokusera på logiken i en algoritm, utan att fastna i språkspecifik syntax.
- Är enkelt att läsa och förstå, även för personer som inte är programmerare.
- Underlättar planeringen av mer komplexa program.

4.3.2 Hur skriver man pseudokod?

Det finns inga fasta regler för pseudokod, men här är några riktlinjer:

- Använd beskrivande namn för variabler och steg.
- Håll det kortfattat och tydligt.
- Strukturera koden med indragningar för att visa block som hör ihop.
- Använd enkla termer som LOOP", "IF", och "ELSE".

4.3.3 Exempel: Summera en lista

Låt oss skriva pseudokod för att summera alla värden i en lista.

Pseudokod:

```
START
SET summa TO 0
FOR varje element i listan:
        ADD element TO summa
END FOR
PRINT summa
END
```

Python-implementation:

```python
lista = [1, 2, 3, 4, 5]
summa = 0
for element in lista:
    summa += element
print(summa)  # Output: 15
```

Summera en lista

4.3.4 Exempel: Hitta det största talet i en lista

Här är ett exempel på pseudokod för att hitta det största talet i en lista.

Pseudokod:

```
START
SET största TO första elementet i listan
FOR varje element i listan:
        IF element > största:
                SET största TO element
        END IF
END FOR
PRINT största
END
```

Python-implementation:

```python
lista = [10, 20, 5, 30, 15]
största = lista[0]
for element in lista:
    if element > största:
        största = element
print(största)  # Output: 30
```

Hitta största talet i en lista

4.3.5 Övningar

Uppgift 39 Skriv pseudokod för att räkna hur många jämna tal som finns i en lista.

Lösningar på sida ?? ◀

Uppgift 40 Implementera följande pseudokod i Python:

```
START
SET antal TO 0
FOR varje element i listan:
IF element är större än 10:
ADD 1 TO antal
END IF
END FOR
PRINT antal
END
```

Lösningar på sida ?? ◀

Pseudokod
Ett sätt att beskriva algoritmer med ord och strukturer som påminner om kod, men utan att följa strikt syntax.

För fler kodexempel se appendix A.12

4.4 Rekursion

Rekursion är en viktig programmeringsteknik där en funktion anropar sig själv för att lösa ett problem. Det kan vara användbart när ett problem kan delas upp i mindre, likartade delproblem.

4.4.1 Hur fungerar rekursion?

En rekursiv funktion måste alltid ha:

1. **Basfall:** En villkorssats som avslutar rekursionen när ett visst kriterium uppfylls.
2. **Rekursivt fall:** Ett anrop till sig själv, med ett argument som gradvis närmar sig basfallet.

4.4.2 Exempel: Faktorial

Faktorial av ett heltal n definieras som:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

Faktorial av 0 är definierat som 1 ($0! = 1$).
Med rekursion kan vi definiera detta som:

$$n! = \begin{cases} 1 & \text{om} \quad n = 0 \\ n \times (n-1)! & \text{om} \quad n > 0 \end{cases}$$

Kodexempel

```python
def faktorial(n):
    if n == 0:  # Basfall
        return 1
    else:       # Rekursivt fall
        return n * faktorial(n - 1)

# Exempel
print(faktorial(5))  # Output: 120
```

Rekursiv faktorialfunktion

Rekursiv funktion
En funktion som anropar sig själv. Den använder basfall för att avsluta anropen.

4.4.3 Exempel: Rekursiv binär sökning

Binär sökning kan också implementeras rekursivt. Istället för att använda en loop delar den rekursiva versionen listan i mindre delar tills det sökta värdet hittas eller listan är tom.

Kodexempel

```python
def rekursiv_binar_sokning(lista, mål, vänster, höger):
    if vänster > höger:  # Basfall: Målet finns inte
        return -1
    mitten = (vänster + höger) // 2
    if lista[mitten] == mål:  # Basfall: Målet hittas
        return mitten
    elif lista[mitten] < mål:
        return rekursiv_binar_sokning(lista, mål, mitten + 1, höger)
    else:
        return rekursiv_binar_sokning(lista, mål, vänster, mitten - 1)

# Exempel
data = [10, 20, 30, 40, 50]
print(rekursiv_binar_sokning(data, 30, 0, len(data) - 1))  # Output: 2
```

Rekursiv binär sökning

4.4.4 Rekursion jämfört med iteration

Rekursion och iteration (loopar) kan ofta lösa samma problem. Här är några skillnader:

- **Rekursion:** Kan vara enklare och mer intuitiv för vissa problem, t.ex. träd eller grafproblem.
- **Iteration:** Mer minneseffektiv, eftersom rekursion kräver att varje anrop lagras på ett stackminne.

Fördelar med rekursion	Nackdelar med rekursion
Intuitiv för vissa problem	Kräver mer minne
Kortare kod i vissa fall	Risk för stack overflow

Tabell 4.2: Jämförelse av rekursionens fördelar och nackdelar.

4.4.5 Övningar

Uppgift 41 Skriv en rekursiv funktion som beräknar summan av alla heltal från 1 till n.

Lösningar på sida **??** ◀

Uppgift 42 Implementera binär sökning rekursivt och använd den för att hitta värdet 45 i listan `[5, 15, 25, 35, 45, 55]`.

Lösningar på sida **??** ◄

För fler kodexempel se appendix A.13

Kapitel 5

Lösningsförslag

Lösning till Uppgift 1 (Från sida 12)

```
print("Hej! Jag lär mig Python.")
```

Lösning till Uppgift 2 (Från sida 12)

```
print(12 + 5)
print(20 - 8)
print(4 * 3)
print(16 / 4)
```

Output

```
17
12
12
4.0
```

Lösning till Uppgift 3 (Från sida 12)

```
print("Python är roligt!")
print(10+15)
```

Lösning till Uppgift 4 (Från sida 16)

```
a = 7
b = 3
print(a + b)
```

Lösning till Uppgift 5 (Från sida 16)

```
x = 10
x = 20
print(x)
```

Lösning till Uppgift 6 (Från sida 16)

```
namn = "Patrik"
print("Hej, " + namn + "!")
```

Lösning till Uppgift 7 (Från sida 19)

```
x = 150
if x > 100:
    print("Talet är stort.")
```

Lösning till Uppgift 8 (Från sida 19)

```
x = -10
if x >= 0:
    print("Talet är positivt.")
else:
    print("Talet är negativt.")
```

Lösning till Uppgift 9 (Från sida 19)

```
age = 15
if age < 13:
    print("Du är ett barn.")
elif age <= 19:
    print("Du är en tonåring.")
else:
    print("Du är vuxen.")
```

Lösning till Uppgift 10 (Från sida 20)

```
color = input("Vad är din favoritfärg? ")
print("Din favoritfärg är " + color + ".")
```

Lösning till Uppgift 11 (Från sida 20)

```
num1 = int(input("Ange det första talet: "))
num2 = int(input("Ange det andra talet: "))
print("Summan är: " + str(num1 + num2))
```

Lösning till Uppgift 12 (Från sida 20)

```
age = int(input("Hur gammal är du? "))
if age < 18:
    print("Du är inte myndig.")
else:
    print("Du är myndig.")
```

Lösning till Uppgift 13 (Från sida 24)

```python
# Fråga användaren om deras namn
namn = input("Vad heter du? ")

# Fråga användaren om deras ålder
alder = input("Hur gammal är du? ")

# Skriv ut ett meddelande med namn och ålder
print("Hej, " + namn + "! Du är " + alder + " år gammal.")
```

Lösning till Uppgift 14 (Från sida 24)

```python
# Fråga användaren om ett tal
tal = int(input("Skriv ett tal: "))

# Kontrollera om talet är jämnt eller udda
if tal % 2 == 0:
    print("Talet är jämnt.")
else:
    print("Talet är udda.")
```

Lösning till Uppgift 15 (Från sida 26)

```python
import random
tal = random.randint(1, 100)
print("Slumptalet är:", tal)
```

Lösning till Uppgift 16 (Från sida 26)

```python
import random
tarning1 = random.randint(1, 6)
tarning2 = random.randint(1, 6)
print("Summan är:", tarning1 + tarning2)
```

Lösning till Uppgift 17 (Från sida 26)

```python
import random
aktiviteter = ["läsa", "spela spel", "träna", "rita", "baka"]
vald_aktivitet = random.choice(aktiviteter)
print("Idag ska du:", vald_aktivitet)
```

Lösning till Uppgift 18 (Från sida 33)

```python
nummer = int(input("Skriv in ett nummer: "))
print(nummer >= 10 and nummer <= 20)
```

Lösning till Uppgift 19 (Från sida 33)

```
tal1 = int(input("Skriv in det första talet: "))
tal2 = int(input("Skriv in det andra talet: "))
print(tal1 > 50 or tal2 > 50)
```

Lösning till Uppgift 20 (Från sida 33)

```
losenord = input("Skriv in ditt lösenord: ")

if losenord == "hemligt123":
    print("Access granted")
else:
    print("Access denied")
```

Lösning till Uppgift 21 (Från sida 36)

```
import math

# Fråga användaren efter ett tal
tal = float(input("Ange ett tal: "))

# Beräkna och visa kvadratroten
print("Kvadratroten av", tal, "är:", math.sqrt(tal))
```

Lösning till Uppgift 22 (Från sida 36)

```
# Fråga användaren efter ett tal
tal = int(input("Ange ett tal: "))

# Kontrollera om talet är jämnt eller udda
if tal % 2 == 0:
    print("Talet är jämnt.")
else:
    print("Talet är udda.")
```

Lösning till Uppgift 23 (Från sida 36)

```
# Fråga användaren efter två tal
tal1 = int(input("Ange det första talet: "))
tal2 = int(input("Ange det andra talet: "))

# Beräkna heltalsdivision och resten
print("Heltalsdivision:", tal1 // tal2)
print("Rest:", tal1 % tal2)
```

Lösning till Uppgift 24 (Från sida 40)

```
frukter = ["äpple", "banan", "apelsin"]
for frukt in frukter:
    print(frukt)
```

Lösning till Uppgift 25 (Från sida 40)

```
tal = [1, 2, 3, 4, 5]
tal.append(6)
print(tal)  # [1, 2, 3, 4, 5, 6]
```

Lösning till Uppgift 26 (Från sida 41)

```
tal = [10, 20, 30]
print(len(tal))  # 3
```

Lösning till Uppgift 27 (Från sida 44)

```
for tal in range(1, 11):
    print(tal)
```

Lösning till Uppgift 28 (Från sida 44)

```
for tal in range(3, 31, 3):
    print(tal)
```

Lösning till Uppgift 29 (Från sida 44)

```
text = "Programmering"
for bokstav in text:
    print(bokstav)
```

Lösning till Uppgift 30 (Från sida 47)

```
# Skapa en dictionary med städer och befolkning
stad_befolkning = {
    "Stockholm": 975551,
    "Göteborg": 583056,
    "Malmö": 347949
}

# Skriv ut befolkningen för en stad
print("Befolkning i Stockholm:", stad_befolkning["Stockholm"])
```

Output:

```
Befolkning i Stockholm: 975551
```

Lösning till Uppgift 31 (Från sida 47)

```python
# Skapa en tom dictionary för elevbetyg
elev_betyg = {}

# Lägg till nya elever och deras betyg
elev_betyg["Anna"] = "A"
elev_betyg["Björn"] = "B"
elev_betyg["Cecilia"] = "C"

# Visa alla elever och deras betyg
print("Elevbetyg:")
for elev, betyg i elev_betyg.items():
    print(f"{elev}: {betyg}")
```

Output:

```
Elevbetyg:
Anna: A
Björn: B
Cecilia: C
```

Lösning till Uppgift 32 (Från sida 47)

```python
# Mening att analysera
mening = "Detta är en enkel mening"

# Skapa en tom dictionary för att räkna bokstäver
bokstav_räknare = {}

# Gå igenom varje bokstav i den rensade meningen
for bokstav in rensad_mening:
    # Använd get() för att öka antalet eller sätta det till 1 om bokstaven är ny
    bokstav_räknare[bokstav] = bokstav_räknare.get(bokstav, 0) + 1

# Visa resultatet
print("Antal bokstäver:")
for bokstav, antal in bokstav_räknare.items():
    print(f"{bokstav}: {antal}")
```

Output:

```
Antal bokstäver:
d: 1
e: 7
t: 4
a: 2
ä: 1
n: 4
k: 1
l: 1
m: 1
g: 1
```

Lösning till Uppgift 33 (Från sida 50)

```
data = [10, 8, 2, 7, 1, 3]
bubble_sort(data)
print("Sorterad lista:", data)
```

Output: [1, 2, 3, 7, 8, 10]

Lösning till Uppgift 36 (Från sida 53)

```
data = [10, 8, 2, 7, 1, 3]
bubble_sort(data)
print("Sorterad lista:", data)
```

Output: [1, 2, 3, 7, 8, 10]

Bilaga A

Exempelkod

Här finns exempelkod från varje kapitel. Syftet är att du ska kunna kolla på programmen för att förstå hur programmering kan användas.

A.1 Kodexempel: Print

Exempel från avsnitt 2.1.

Grundläggande exempel

```
print("Hej, världen!")
```

Exempel 1: Skriv ut en enkel text

```
print('Python är roligt!')
```

Exempel 2: Enkelcitat för text

```
print("Han sa: 'Hej där!'")
```

Exempel 3: Kombinera citattecken

Matematiska operationer

```
print(2 + 3)  # Output: 5
```

Exempel 4: Addition

```
print(10 - 4)  # Output: 6
```

Exempel 5: Subtraktion

```
print(7 * 2)  # Output: 14
```

Exempel 6: Multiplikation

```
print(9 / 3)  # Output: 3.0
```

Exempel 7: Division

```
print((5 + 3) * 2)  # Output: 16
```

Exempel 8: Parenteser ändrar ordning

```
print("Antal äpplen per låda:", 24 / 6)
# Output: Antal äpplen per låda: 4.0
```

Exempel 9: Praktisk tillämpning

Flera värden i samma utskrift

```
print("Antal:", 5, "Pris per styck:", 20)
# Output: Antal: 5 Pris per styck: 20
```

Exempel 10: Skriva ut flera saker med komma

```python
print("Summan är:", 10 + 15)
# Output: Summan är: 25
```

Exempel 11: Beräkningar och text tillsammans

Felaktiga syntaxexempel

```python
print "Detta fungerar inte!"
```

Exempel 12: Saknar parantes

Förklaring: Parenteser måste alltid användas för att omge argumentet.

```python
print("Oj då!')
```

Exempel 13: Omatchade citattecken

Förklaring: Enkla och dubbla citattecken måste matcha varandra.

Avancerade exempel

```python
print("Hej!\nVälkommen till Python.")
# Output:
# Hej!
# Välkommen till Python.
```

Exempel 14: Skriva ut flera rader

```python
print("Det här är en backslash: \\")
# Output: Det här är en backslash: \
```

Exempel 15: Specialtecken i text

```python
print("Avrundat resultat:", 10 // 3)
# Output: Avrundat resultat: 3
```

Exempel 16: Matematiskt avrundat resultat

```python
print("Resterande antal:", 10 % 3)
# Output: Resterande antal: 1
```

Exempel 17: Resten av division

```python
print("Resultat:", 2 ** 3)
# Output: Resultat: 8
```

Exempel 18: Användning av exponent

A.2 Kodexempel: Variabler och sekventiell exekvering

Exempel från avsnitt 2.2.

Grunderna för variabler

```
name = "Alice"
print(name)
# Output: Alice
```

Exempel 1: Skapa en variabel

```
age = 25
print(age)
# Output: 25
```

Exempel 2: Variabel med heltal

```
price = 19.99
print(price)
# Output: 19.99
```

Exempel 3: Variabel med flyttal

Datatyper och deras användning

```
name = "Bob"
age = 30
print("Namn:", name)
print("Ålder:", age)
# Output:
# Namn: Bob
# Ålder: 30
```

Exempel 4: Kombinera variabler och text

```
first_name = "Alice"
last_name = "Smith"
full_name = first_name + " " + last_name
print(full_name)
# Output: Alice Smith
```

Exempel 5: Konkatenering av strängar

```
age = 20
print("Jag är " + age + " år gammal.")
# Ger ett TypeError! (sträng + int fungerar inte)
```

Exempel 6: Implicit datakonvertering är inte tillåten

```
age = 20
print("Jag är " + str(age) + " år gammal.")
# Output: Jag är 20 år gammal.
```

Exempel 7: Konvertera till sträng för att undvika fel

Sekventiell exekvering

```
counter = 0
print(counter)  # Output: 0
counter = counter + 1
print(counter)  # Output: 1
```

Exempel 8: Variabler kan ändra värde

```
a = 10
b = 3
result = a * b
print("Resultat:", result)
# Output: Resultat: 30
```

Exempel 9: Beräkningar med variabler

```
x = 5
y = x + 2
print(y)  # Output: 7
x = 10
print(y)  # Output: 7 (ändringen av x påverkar inte y)
```

Exempel 10: Variabler beroende av varandra

Blandade exempel

```
word = "Hej"
print(word * 3)
# Output: HejHejHej
```

Exempel 11: Multiplicera text

```
print(name)
# Ger ett NameError! (variabeln är inte definierad)
```

Exempel 12: Variabeln används innan den skapas

```
balance = 100
balance = balance - 20
balance = balance + 50
```

```
print(balance)
# Output: 130
```

Exempel 13: Variabelns värde ändras stegvis

```
x = 10
y = 3.5
print("Summan är:", x + y)
# Output: Summan är: 13.5
```

Exempel 14: Variabler med flyttal och heltal tillsammans

```
name = input("Vad heter du? ")
print("Hej, " + name + "!")
```

Exempel 15: Använda input och variabel tillsammans

```
pi = 3.14159
rounded_pi = round(pi, 2)
print("Pi avrundat:", rounded_pi)
# Output: Pi avrundat: 3.14
```

Exempel 16: Avrundning med variabel

```
value = 42
print(type(value))
# Output: <class 'int'>
```

Exempel 17: Kontrollera en variabels datatyp

```
x = 4
x *= 3  # Samma som x = x * 3
print(x)
# Output: 12
```

Exempel 18: Multiplicera och uppdatera samtidigt

```
total = 100
totaal = 200  # Felstavning skapar en ny variabel!
print(total)   # Output: 100
print(totaal)  # Output: 200
```

Exempel 19: Förväxla inte namn på variabler

```
width = 5
height = 10
area = width * height
perimeter = 2 * (width + height)
print("Area:", area)
print("Omkrets:", perimeter)
```

```
# Output:
# Area: 50
# Omkrets: 30
```

Exempel 20: Variabler i längre beräkningar

A.3 Kodexempel: If-satser (Beslutsfattande)

Exempel från avsnitt 2.3.

Grunderna för if-satser

```
x = 10
if x > 5:
    print("x är större än 5")
# Output: x är större än 5
```

Exempel 1: En enkel if-sats

```
y = 8
if y == 8:
    print("y är lika med 8")
# Output: y är lika med 8
```

Exempel 2: Kontrollera likhet

```
age = 17
if age >= 18:
    print("Du är vuxen.")
else:
    print("Du är inte vuxen.")
# Output: Du är inte vuxen.
```

Exempel 3: Använda else

elif och flera villkor

```
temperature = 15
if temperature > 25:
    print("Det är varmt.")
elif temperature > 10:
    print("Det är svalt.")
else:
    print("Det är kallt.")
# Output: Det är svalt.
```

Exempel 4: Använda elif för fler alternativ

```
score = 85
if score >= 90:
    print("Betyg: A")
elif score >= 75:
    print("Betyg: B")
elif score >= 60:
```

```
    print("Betyg: C")
else:
    print("Betyg: F")
# Output: Betyg: B
```

Exempel 5: If-elif-else med gränsvärden

Indentering är viktigt

```
x = 10
if x > 5:
print("x är större än 5")  # Ger ett IndentationError
```

Exempel 6: Felaktig indentering

```
x = 10
if x > 5:
    print("x är större än 5")  # Rätt indentering
# Output: x är större än 5
```

Exempel 7: Korrekt indentering

Jämförelseoperationer

```
x = 12
if x >= 10:
    print("x är minst 10")
# Output: x är minst 10
```

Exempel 8: Kontrollera större än eller lika med

```
y = 7
if y < 10:
    print("y är mindre än 10")
# Output: y är mindre än 10
```

Exempel 9: Kontrollera mindre än

```
z = 15
if z != 20:
    print("z är inte 20")
# Output: z är inte 20
```

Exempel 10: Kontrollera olika värden

Blandade exempel

```
a = 12
b = 9
if a > b:
    print("a är större än b")
else:
    print("b är större än eller lika med a")
# Output: a är större än b
```

Exempel 11: Använda if för att hitta det största värdet

```
balance = 100
withdrawal = 120
if withdrawal <= balance:
    print("Uttaget är godkänt")
else:
    print("Otillräckligt saldo")
# Output: Otillräckligt saldo
```

Exempel 12: If-sats med variabler i flera steg

```
number = 6
if number % 2 == 0:
    print("Talet är jämnt")
else:
    print("Talet är udda")
# Output: Talet är jämnt
```

Exempel 13: Kontrollera om ett tal är jämnt

```
price = 200
if price > 100:
    discount = 20
else:
    discount = 10
print("Rabatten är:", discount, "kr")
# Output: Rabatten är: 20 kr
```

Exempel 14: If-sats för flera steg av diskontering

```
x = 15
if 10 <= x <= 20:
    print("x är inom intervallet 10 till 20")
else:
    print("x är utanför intervallet")
# Output: x är inom intervallet 10 till 20
```

Exempel 15: Kontrollera intervall med if-satser

A.4 Kodexempel: Input (Att ta emot data från användaren)

Exempel från avsnitt 2.4.

Grunderna för `input()`

```
name = input("Vad heter du? ")
print("Hej, " + name + "!")
# Om användaren skriver: Anna
# Output: Hej, Anna!
```

Exempel 1: Läsa in text från användaren

```
age = input("Hur gammal är du? ")
print("Du är " + age + " år gammal.")
# Om användaren skriver: 25
# Output: Du är 25 år gammal.
```

Exempel 2: Mata in ett heltal

```
age = input("Hur gammal är du? ")
print(age + 5)  # Ger ett TypeError: Kan inte addera sträng och heltal
```

Exempel 3: Fel vid aritmetiska operationer utan typkonvertering

Datatypkonvertering

```
age = int(input("Hur gammal är du? "))
print("Om fem år är du", age + 5, "år gammal.")
# Om användaren skriver: 25
# Output: Om fem år är du 30 år gammal.
```

Exempel 4: Konvertera input till ett heltal

```
height = float(input("Hur lång är du i meter? "))
print("Du är", height, "meter lång.")
# Om användaren skriver: 1.75
# Output: Du är 1.75 meter lång.
```

Exempel 5: Konvertera input till ett flyttal

Kontroll av datatyper

```
number = int(input("Ange ett heltal: "))
print("Datatypen är:", type(number))
# Om användaren skriver: 42
# Output: Datatypen är: <class 'int'>
```

Exempel 6: Kontrollera datatyp efter konvertering

Fler exempel

```
a = int(input("Ange det första talet: "))
b = int(input("Ange det andra talet: "))
print("Summan är:", a + b)
print("Produkten är:", a * b)
# Om användaren skriver: 3 och 5
# Output: Summan är: 8
# Output: Produkten är: 15
```

Exempel 7: Utföra matematiska operationer

```
name = input("Vad heter du? ")
birth_year = int(input("Vilket år föddes du? "))
current_year = 2024
age = current_year - birth_year
print("Hej", name + ", du är", age, "år gammal.")
# Om användaren skriver: Anna och 2000
# Output: Hej Anna, du är 24 år gammal.
```

Exempel 8: Kombinera text och beräkningar

Felkänslig inmatning

```
try:
    number = int(input("Ange ett heltal: "))
    print("Du skrev:", number)
except ValueError:
    print("Det där var inte ett heltal!")
# Om användaren skriver: hej
# Output: Det där var inte ett heltal!
```

Exempel 9: Hantera felaktig inmatning

```
while True:
    try:
        number = int(input("Ange ett heltal: "))
        print("Du skrev:", number)
        break
    except ValueError:
        print("Det där var inte ett heltal. Försök igen.")
# Användaren kan försöka flera gånger tills korrekt värde anges.
```

Exempel 10: Begära inmatning tills den är korrekt

A.5 Kodexempel: Kodkommentarer

Exempel från avsnitt 2.5.

Grunderna i kommentarer

```
# Detta är en kommentar. Den körs inte av programmet.
print("Hej!")  # Detta skriver ut "Hej!" till skärmen.
# Kommentarer kan användas för att förklara vad koden gör.
```

Exempel 1: Enkel kommentar med #

```
"""
Detta är en kommentar som kan sträcka sig
över flera rader. Den används ofta för att
beskriva ett program eller en funktion.
"""
print("Hej från ett program med kommentarer!")
```

Exempel 2: Multiradkommentarer med trippelcitat (")

```
# Följande rad är tillfälligt avstängd för att felsöka programmet.
# print("Den här raden är inaktiverad.")
print("Den här raden körs.")
# Kommenterad kod kan aktiveras igen om det behövs.
```

Exempel 3: Kommentarer för felsökning

God kommentarpraxis

```
# Beräknar summan av två tal och skriver ut resultatet
a = 5
b = 7
summan = a + b
print("Summan är:", summan)
```

Exempel 4: Klara och tydliga kommentarer

```
# Dåligt exempel:
a = 5  # Sätt a till 5
b = 7  # Sätt b till 7
# Det här är onödiga kommentarer eftersom koden redan förklarar sig själv.
```

Exempel 5: Kommentera inte självklarheter

```
# Dåligt exempel:
a = 5  # Sätt a till 5
# Bra exempel:
```

```
a = 5  # Antalet användare som får tillgång samtidigt
```

Exempel 6: Förklara varför och inte bara vad

TODO-kommentarer

```
# TODO: Lägg till inmatning för användaren
a = 10
b = 20
print("Summan är:", a + b)
```

Exempel 7: Använd TODO-kommentarer för framtida uppgifter

Tillämpning av kommentarer

```
"""
Detta program läser in två tal från användaren,
beräknar summan och visar resultatet.
"""
# Läser in två tal från användaren
a = int(input("Ange det första talet: "))  # Konverterar till heltal
b = int(input("Ange det andra talet: "))  # Konverterar till heltal

# Beräkna och skriv ut summan
summan = a + b
print("Summan är:", summan)  # Skriver ut resultatet
```

Exempel 8: Kombinera olika typer av kommentarer

Fler exempel

```
# Kontrollera om ett tal är jämnt eller udda
number = 7
# Om resten av division med 2 är 0, är talet jämnt
if number % 2 == 0:
    print("Talet är jämnt.")
else:
    print("Talet är udda.")
```

Exempel 9: Kommentarer för komplexa delar av koden

```
"""
Detta är en enkel kalkylator som kan addera, subtrahera,
multiplicera och dividera två tal. Just nu är funktionerna
för multiplikation och division inte implementerade.
"""
```

```
# TODO: Implementera multiplikation och division
a = 10
b = 5

print("Summan är:", a + b)  # Addition
print("Skillnaden är:", a - b)  # Subtraktion
```

Exempel 10: Sätt din kod i kontext med kommentarer

A.6 Kodexempel: Slumptal med random

Exempel från avsnitt 2.6.

Modulimport och `randint`

```
# Vi måste importera modulen random innan vi kan använda dess funktioner
import random

# Generera ett heltal mellan 1 och 10
slumptal = random.randint(1, 10)
print("Ett slumptal mellan 1 och 10:", slumptal)
```

Exempel 1: Importera modulen `random`

```
import random

# Generera tre olika slumptal
print("Första slumptalet:", random.randint(1, 10))
print("Andra slumptalet:", random.randint(1, 10))
print("Tredje slumptalet:", random.randint(1, 10))
```

Exempel 2: Generera flera slumptal

`random.random()`

```
import random

# random.random() genererar ett flyttal mellan 0 och 1
flyttal = random.random()
print("Ett slumptal mellan 0 och 1:", flyttal)
```

Exempel 3: Generera ett flyttal mellan 0 och 1

```
import random

# Simulera ett kast med en tärning, där 0.5 är gränsen för "framgång"
slumptal = random.random()
if slumptal > 0.5:
    print("Framgång!")
else:
    print("Misslyckande.")
```

Exempel 4: Använd slumptal för simulering

random.choice()

```
import random

# En lista med möjliga alternativ
alternativ = ["äpple", "banan", "körsbär", "druva"]

# Välj ett slumpmässigt element från listan
val = random.choice(alternativ)
print("Det slumpmässiga valet är:", val)
```

Exempel 5: Välja ett slumpmässigt element från en lista

```
import random

# Definiera en lista med möjliga tärningssidor
tärning = [1, 2, 3, 4, 5, 6]

# Välj en slumpmässig sida
kast = random.choice(tärning)
print("Resultatet av tärningskastet är:", kast)
```

Exempel 6: Simulera ett tärningskast med `random.choice()`

Fler funktioner i `random`

```
import random

# En lista med tal
lista = [1, 2, 3, 4, 5]

# Blanda om listan slumpmässigt
random.shuffle(lista)
print("Den omblandade listan är:", lista)
```

Exempel 7: Slumpa om en lista med `random.shuffle()`

```
import random

# random.uniform() genererar ett slumptal med decimaler mellan 5 och 15
slumptal = random.uniform(5, 15)
print("Ett slumptal mellan 5 och 15:", slumptal)
```

Exempel 8: Generera ett slumptal inom ett intervall med decimaler

Syntaxfel och vanliga misstag

```
# Fel: random är inte importerat
slumptal = random.randint(1, 10)
print(slumptal)
# Lösning: Lägg till "import random" högst upp i programmet
```

Exempel 9: Glöm inte att importera modulen!

```
import random

# Fel: randint() kräver två heltal som argument
# slumptal = random.randint("1", "10")
# Lösning: Skicka in heltal, inte strängar
slumptal = random.randint(1, 10)
print(slumptal)
```

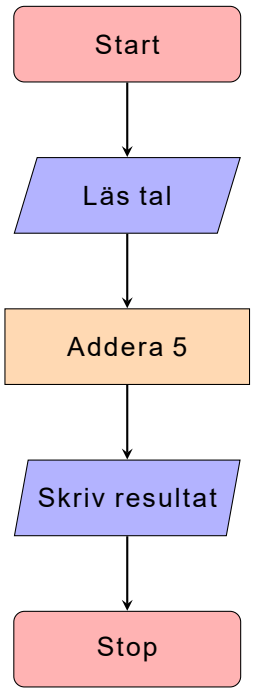
Exempel 10: Fel typ av argument till `randint()`

A.7 Kodexempel: Flödesdiagram

Exempel från avsnitt 2.7.

Exempel 1: Enkel sekvens

Flödesdiagram:



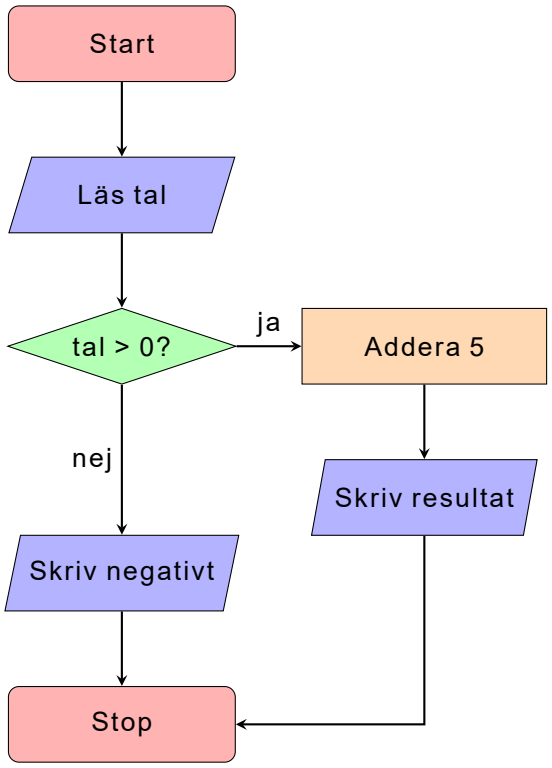
Python-kod:

```python
# Läs ett tal från användaren
nummer = int(input("Ange ett tal: "))

# Lägg till 5
resultat = nummer + 5

# Skriv ut resultatet
print("Resultatet är:", resultat)
```

Enkelt program som läser in ett tal från användaren, adderar 5 till det, och skriver ut resultatet igen.

Exempel 2: Beslutsstruktur

Flödesdiagram:



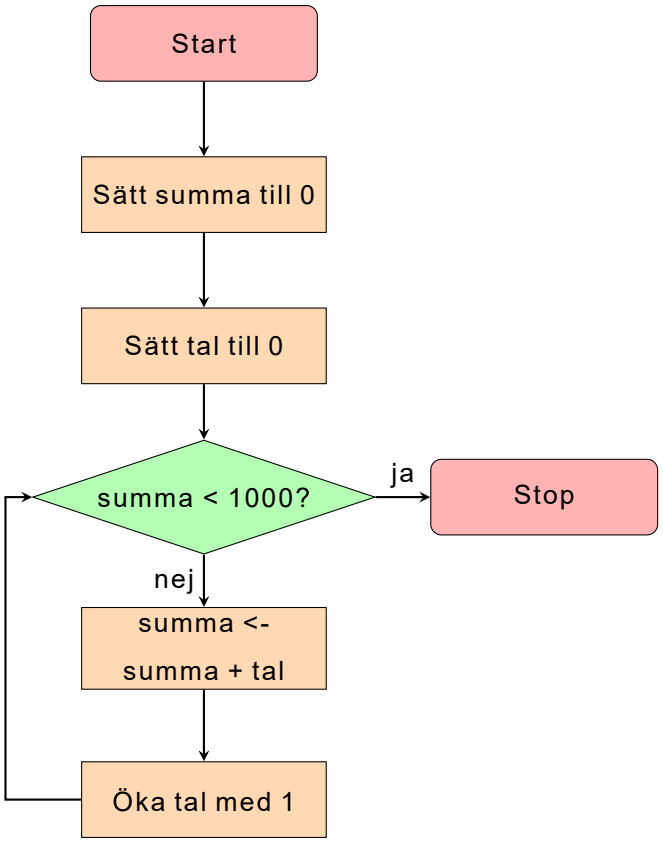
Python-kod:

```python
# Läs ett tal från användaren
nummer = int(input("Ange ett tal: "))

if nummer > 0:
    # Addera 5 om talet är positivt
    resultat = nummer + 5
    print("Resultatet är:", resultat)
else:
    # Annars skriv "negativt"
    print("Talet är negativt")
```

Användaren skriver in ett tal. Om det är större än 0 adderas fem till talet, och resultatet skrivs ut. Annars skrivs endast ut att talet är negativt utan att addera något.

Exempel 3: While-loop med villkor

Flödesdiagram:



Python-kod:

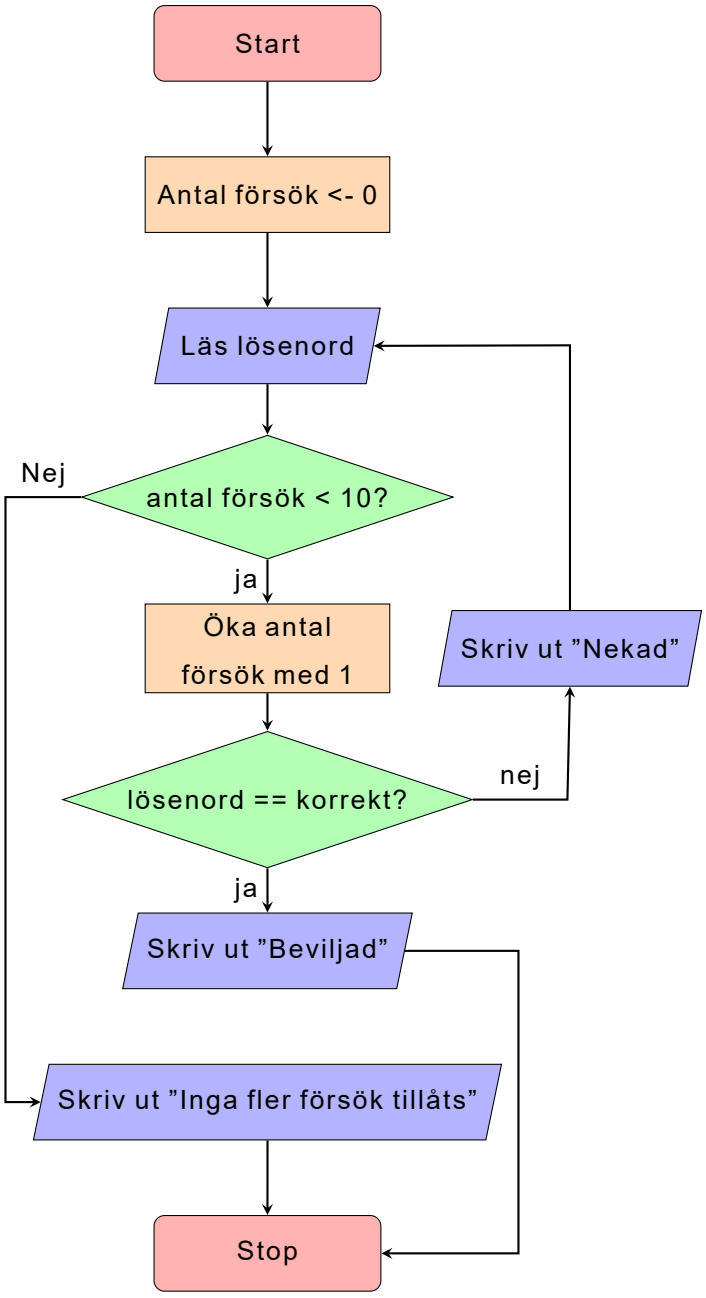
```
# Räknar 1+2+3+4+5... så länge summan är
    mindre än 1000
summa = 0
tal = 1
while summa < 1000:
    summa = summa + tal

print("Summa 1+2+3+...+",tal,">1000")
```

Listing A.1: Summera positiva tal

Räknar ut vilket tal vi behöver räkna till så att $1+2+3+\cdots+tal > 1000$.

Exempel 4: Kombinerad if-sats och while-loop

Flödesdiagram:



Python-kod:

```
# Enkel lösenordskontroll
korrekt_losenord = "python123"
antal_försök=0

while antal_försök<10:
    losenord = input("Ange lösenord: ")
    if losenord == korrekt_losenord:
        print("Beviljad!")
        break
    else:
        print("Nekad")
    antal_försök = antal_försök + 1
if antal_försök == 10:
  print("Inga fler försök tillåts")
```

Låter användaren mata in ett lösenord tills dess att användaren skriver in ”python123”. Programmet skriver ut meddelanden vid varje försök som säger om lösenordet var rätt eller inte. Programmet ger max 10 försök till användaren. Notera att i flödesdiagrammet finns ingen direkt motsvarighet till if-satsen efter loopen. Det beror på att flödesdiagramet beskriver vad programmet gör logiskt och inte i en direktöversättning. Eftersom loopen i flödesdiagramet avslutas genom **Antal försök > 10?** behövs inte en extra koll.

A.8 Kodexempel: Logiska uttryck och booleanska värden

Exempel från avsnitt 2.8.

Datatypen `bool`

```
# Booleanska värden är antingen True eller False
a = True
b = False
print("a är:", a)
print("b är:", b)
```

Exempel 1: Sanna och falska värden

```
# Jämförelser returnerar ett booleskt värde
print(5 > 3)   # True
print(10 < 7)  # False
print(3 == 3)  # True
print(4 != 5)  # True
```

Exempel 2: Booleanska värden från jämförelser

Logiska operatorer: and, or, not

```
# and returnerar True om båda villkoren är sanna
print(True and True)    # True
print(True and False)   # False
print(5 > 3 and 2 < 4)  # True
print(5 > 3 and 2 > 4)  # False
```

Exempel 3: Använda `and`

```
# or returnerar True om minst ett villkor är sant
print(True or False)    # True
print(False or False)   # False
print(5 > 3 or 2 > 4)   # True
print(10 < 7 or 1 == 1) # True
```

Exempel 4: Använda `or`

```
# not inverterar ett booleskt värde
print(not True)   # False
print(not False)  # True
print(not (5 > 3)) # False
```

Exempel 5: Använda `not`

Kombination av logiska uttryck

```
# Kombinera flera logiska operatorer
x = 10
y = 5
print((x > y) and (y > 2))     # True
print((x > y) or (y < 2))      # True
print(not (x == y))            # True
```

Exempel 6: Kombinera and or och not

```
# not har högre prioritet än and/or
print(not True and False)  # False
print(not (True and False)) # True
print(True or not False)    # True
print((True or False) and not False) # True
```

Exempel 7: Operatorns prioritet

Vanliga misstag

```
x = 10
y = 5

# Fel: operatorprioritet ger oväntat resultat
print(not x > y and y < 2)  # Detta är False

# Rätt: använd parenteser för tydlighet
print((not (x > y)) and (y < 2))  # Detta är True
```

Exempel 8: Glöm inte parenteser vid komplexa uttryck

```
x = (5 > 3)

# Onödigt sätt
if x == True:
    print("x är sant!")

# Bättre sätt
if x:
    print("x är sant!")
```

Exempel 9: Jämför inte direkt med `True`/`False`

Exempel i praktisk användning

```python
# Villkor: Användaren måste vara inloggad och ha adminrättigheter
inloggad = True
admin = False

if inloggad and admin:
    print("Välkommen, admin!")
else:
    print("Åtkomst nekad.")
```

Exempel 10: Kontrollera tillgång till system

```python
# Kontrollera om ett tal är mellan 10 och 20
tal = 15

if tal >= 10 and tal <= 20:
    print("Talet är inom intervallet.")
else:
    print("Talet är utanför intervallet.")
```

Exempel 11: Kontrollera om ett tal är i intervall

A.9 Kodexempel: Matematik i Python

Exempel från avsnitt 2.9.

`math`-modulen

```python
# För att använda funktioner från math-modulen måste vi importera den
import math

# Exempel: Beräkna kvadratroten av ett tal
tal = 25
resultat = math.sqrt(tal)
print("Kvadratroten av", tal, "är", resultat)
```

Exempel 1: Importera math-modulen

```python
import math

# math.pow() beräknar bas upphöjt till exponent
bas = 2
exponent = 3
resultat = math.pow(bas, exponent)
print(bas, "upphöjt till", exponent, "är", resultat)
```

Exempel 2: Exponentiering med `math.pow()`

```python
import math

# math.floor() rundar ner till närmaste heltal
print(math.floor(5.7))  # 5

# math.ceil() rundar upp till närmaste heltal
print(math.ceil(5.7))   # 6
```

Exempel 3: Heltalsavrundning med `math.floor()` och `math.ceil()`

```python
import math

# math.pi är ett konstant värde för pi
radie = 5
omkrets = 2 * math.pi * radie
print("Omkretsen av en cirkel med radie", radie, "är", omkrets)
```

Exempel 4: Använda matematiska konstanter som `math.pi`

Specialoperationer

```python
# Operatorn ** används för att beräkna potenser
bas = 3
exponent = 4
resultat = bas ** exponent
print(bas, "upphöjt till", exponent, "är", resultat)  # 81
```

Exempel 5: Exponentiering med **

```python
# Modulus operatorn \% beräknar resten av en division
a = 10
b = 3
rest = a % b
print("Resten av", a, "delat med", b, "är", rest)  # 1
```

Exempel 6: Modulusoperatorn %

```python
# Heltalsdivision returnerar endast heltalsdelen av resultatet
a = 10
b = 3
kvot = a // b
print(a, "heltalsdividerat med", b, "är", kvot)  # 3
```

Exempel 7: Heltalsdivision med //

Kombinerade exempel

```python
import math

# Pythagoras sats: c = sqrt(a^2 + b^2)
a = 3
b = 4
c = math.sqrt(a**2 + b**2)
print("Hypotenusan för en triangel med sidorna", a, "och", b, "är", c)  # 5.0
```

Exempel 8: Beräkna hypotenusan i en triangel

```python
# Modulus kan användas för att kontrollera jämnhet
tal = 15
if tal % 2 == 0:
    print(tal, "är ett jämnt tal.")
else:
    print(tal, "är ett udda tal.")
```

Exempel 9: Kontrollera om ett tal är jämnt eller udda

```python
import math

# Kombinera floor och ceil för att göra en egen rundningsfunktion
```

```
tal = 5.5

if tal - math.floor(tal) < 0.5:
    avrundat = math.floor(tal)
else:
    avrundat = math.ceil(tal)

print("Rundning av", tal, "ger", avrundat)  # 6
```

Exempel 10: Rundning av tal till närmaste heltal

```
import math

# Beräkna area av en cirkel
radie = 7
area = math.pi * radie**2
print("Arean av en cirkel med radie", radie, "är", area)
```

Exempel 11: Area av en cirkel

A.10 Kodexempel: Bubble Sort

Exempel från avsnitt 4.1.

Exempel 1: Grundläggande Bubble Sort-algoritm

```python
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                # Byt plats om elementet är större än det nästa
                lista[j], lista[j + 1] = lista[j + 1], lista[j]

# Testa med en lista
tal_lista = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(tal_lista)
print("Sorterad lista:", tal_lista)
# Output: Sorterad lista: [11, 12, 22, 25, 34, 64, 90]
```

Bubble Sort - Grundläggande implementation

Exempel 2: Visualisering av varje steg i sorteringen

```python
def bubble_sort_med_steg(lista):
    n = len(lista)
    for i in range(n):
        print(f"Pass {i + 1}: {lista}")  # Visa listan vid varje pass
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]

tal_lista = [5, 3, 8, 6]
bubble_sort_med_steg(tal_lista)
# Output:
# Pass 1: [5, 3, 8, 6]
# Pass 2: [3, 5, 6, 8]
# Pass 3: [3, 5, 6, 8]
# Pass 4: [3, 5, 6, 8]
```

Bubble Sort - Visualisering av stegen

Exempel 3: Optimera Bubble Sort med tidig avbrytning

```python
def bubble_sort_optimerad(lista):
    n = len(lista)
```

```
    for i in range(n):
        bytt = False
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]
                bytt = True
        if not bytt:
            break  # Avbryt om listan redan är sorterad

tal_lista = [1, 2, 3, 4, 5]
bubble_sort_optimerad(tal_lista)
print("Sorterad lista:", tal_lista)
# Output: Sorterad lista: [1, 2, 3, 4, 5]
```

Bubble Sort - Optimerad version

Exempel 4: Bubble Sort på strängar

```
def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j] > lista[j + 1]:
                lista[j], lista[j + 1] = lista[j + 1], lista[j]

sträng_lista = ["banan", "äpple", "citron", "druva"]
bubble_sort(sträng_lista)
print("Sorterad lista:", sträng_lista)
# Output: Sorterad lista: ['äpple', 'banan', 'citron', 'druva']
```

Bubble Sort - Sortera strängar

Exempel 5: Praktisk användning av Bubble Sort

```
# Sortera elever baserat på deras poäng
elever = [("Anna", 85), ("Björn", 75), ("Cecilia", 90), ("David", 80)]

def bubble_sort(lista):
    n = len(lista)
    for i in range(n):
        for j in range(0, n - i - 1):
            if lista[j][1] > lista[j + 1][1]:  # Sortera efter poäng
                lista[j], lista[j + 1] = lista[j + 1], lista[j]

bubble_sort(elever)
print("Sorterade elever:", elever)
```

```
# Output: Sorterade elever: [('Björn', 75), ('David', 80), ('Anna', 85), ('Cecilia', 90)]
```

Bubble Sort - Praktisk användning

Tips och rekommendationer

- Bubble Sort är enkel att förstå och implementera, men ineffektiv för stora datamängder. - Använd optimerad version för att förbättra prestanda om listan kan vara delvis sorterad.

A.11 Kodexempel: Linjär och Binär Sökning

Exempel från avsnitt 4.2.

Exempel 1: Linjär sökning i en lista

```python
def linjar_sokning(lista, mål):
    for index, värde i enumerate(lista):
        if värde == mål:
            return index  # Returnera index om målet hittas
    return -1  # Returnera -1 om målet inte finns

tal_lista = [10, 20, 30, 40, 50]
mål = 30
resultat = linjar_sokning(tal_lista, mål)
print(f"Elementet finns vid index: {resultat}")
# Output: Elementet finns vid index: 2
```

Linjär sökning i en lista

Exempel 2: Linjär sökning i en dictionary

```python
elever = {"Anna": 85, "Björn": 75, "Cecilia": 90, "David": 80}

def linjar_sokning_dict(dictionary, mål):
    for nyckel, värde i dictionary.items():
        if värde == mål:
            return nyckel
    return None

mål = 90
resultat = linjar_sokning_dict(elever, mål)
print(f"Eleven med poäng {mål} är: {resultat}")
# Output: Eleven med poäng 90 är: Cecilia
```

Linjär sökning i en dictionary

Exempel 3: Binär sökning i en sorterad lista

```python
def binar_sokning(lista, mål):
    vänster, höger = 0, len(lista) - 1
    while vänster <= höger:
        mitten = (vänster + höger) // 2
        if lista[mitten] == mål:
            return mitten
        elif lista[mitten] < mål:
```

```
            vänster = mitten + 1
        else:
            höger = mitten - 1
    return -1

tal_lista = [10, 20, 30, 40, 50]
mål = 40
resultat = binar_sokning(tal_lista, mål)
print(f"Elementet finns vid index: {resultat}")
# Output: Elementet finns vid index: 3
```

Binär sökning i en sorterad lista

Exempel 4: Visualisering av binär sökning

```
def binar_sokning_visualisering(lista, mål):
    vänster, höger = 0, len(lista) - 1
    steg = 0
    while vänster <= höger:
        mitten = (vänster + höger) // 2
        print(f"Steg {steg}: Vänster={vänster}, Höger={höger}, Mitten={mitten}")
        if lista[mitten] == mål:
            return mitten
        elif lista[mitten] < mål:
            vänster = mitten + 1
        else:
            höger = mitten - 1
        steg += 1
    return -1

tal_lista = [10, 20, 30, 40, 50]
mål = 40
binar_sokning_visualisering(tal_lista, mål)
# Output:
# Steg 0: Vänster=0, Höger=4, Mitten=2
# Steg 1: Vänster=3, Höger=4, Mitten=3
# Output: Elementet finns vid index: 3
```

Binär sökning - Visualisering

Exempel 5: Tidskomplexitet

Tidskomplexitet för sökning

Linjär sökning: $O(n)$, där n är antalet element i listan.

Binär sökning: $O(\log n)$, men listan måste vara sorterad.

Exempel 6: Binär sökning med rekursion

```python
def binar_sokning_rekursiv(lista, mål, vänster, höger):
    if vänster > höger:
        return -1
    mitten = (vänster + höger) // 2
    if lista[mitten] == mål:
        return mitten
    elif lista[mitten] < mål:
        return binar_sokning_rekursiv(lista, mål, mitten + 1, höger)
    else:
        return binar_sokning_rekursiv(lista, mål, vänster, mitten - 1)

tal_lista = [10, 20, 30, 40, 50]
mål = 50
resultat = binar_sokning_rekursiv(tal_lista, mål, 0, len(tal_lista) - 1)
print(f"Elementet finns vid index: {resultat}")
# Output: Elementet finns vid index: 4
```

Binär sökning - Rekursiv implementation

Praktiska tips

- Linjär sökning fungerar för osorterade listor eller dictionaries. - Binär sökning kräver en sorterad lista och är betydligt snabbare för större datamängder.

A.12 Kodexempel: Pseudokod

Exempel från avsnitt 4.3.

Exempel 1: Summera tal i en lista

Pseudokod:

```
START
    Lista = [10, 20, 30, 40]
    SUMMA = 0
    FOR varje TAL i Lista
        SUMMA = SUMMA + TAL
    END FOR
    Skriv SUMMA
END
```

Python-kod:

```python
tal_lista = [10, 20, 30, 40]
summa = 0

for tal in tal_lista:
    summa += tal

print(f"Summan är: {summa}")
# Output: Summan är: 100
```

Summera tal i en lista

Exempel 2: Kontrollera om ett tal är jämnt

Pseudokod:

```
START
    Läs in TAL
    IF TAL modulo 2 = 0 THEN
        Skriv "Talet är jämnt"
    ELSE
        Skriv "Talet är udda"
    END IF
END
```

Python-kod:

```python
tal = int(input("Ange ett tal: "))

if tal % 2 == 0:
```

```
    print("Talet är jämnt")
else:
    print("Talet är udda")
# Testa med tal som 4 och 7
```

Kontrollera om ett tal är jämnt

Exempel 3: Hitta det största talet i en lista

Pseudokod:

```
START
    Lista = [12, 45, 23, 89]
    MAX = Lista[0]
    FOR varje TAL i Lista
        IF TAL > MAX THEN
            MAX = TAL
        END IF
    END FOR
    Skriv MAX
END
```

Python-kod:

```
tal_lista = [12, 45, 23, 89]
största_talet = tal_lista[0]

for tal in tal_lista:
    if tal > största_talet:
        största_talet = tal

print(f"Det största talet är: {största_talet}")
# Output: Det största talet är: 89
```

Hitta det största talet i en lista

Exempel 4: Beräkna fakultet av ett tal

Pseudokod:

```
START
    Läs in TAL
    PRODUKT = 1
    FOR i = 1 TILL TAL
        PRODUKT = PRODUKT * i
    END FOR
    Skriv PRODUKT
END
```

Python-kod:

```python
tal = int(input("Ange ett tal: "))
fakultet = 1

for i in range(1, tal + 1):
    fakultet *= i

print(f"Fakulteten av {tal} är: {fakultet}")
# Testa med 5 (output: 120)
```

Beräkna fakultet av ett tal

Exempel 5: Fibonaccital (Iterativt)

Pseudokod:

```
START
    Läs in N
    A = 0
    B = 1
    FOR i = 1 TILL N
        Skriv A
        TEMP = A + B
        A = B
        B = TEMP
    END FOR
END
```

Python-kod:

```python
n = int(input("Hur många Fibonaccital vill du visa? "))
a, b = 0, 1

for _ in range(n):
    print(a, end=" ")
    a, b = b, a + b
# Testa med 7 (output: 0 1 1 2 3 5 8)
```

Fibonaccital - Iterativ metod

A.13 Kodexempel: Rekursion

Exempel från avsnitt 4.4.

Exempel 1: Rekursionens grunder

En rekursiv funktion anropar sig själv för att lösa mindre delar av ett problem. En rekursiv funktion måste innehålla: - Ett **basfall** som avslutar rekursionen. - Ett **rekursivt fall** som löser problemet genom att anropa sig själv.

Exempel på rekursiv nedräkning:

```python
def countdown(n):
    if n == 0:  # Basfall
        print("Klar!")
    else:  # Rekursivt fall
        print(n)
        countdown(n - 1)

countdown(5)
# Output: 5 4 3 2 1 Klar!
```

Nedräkning med rekursion

Exempel 2: Faktorial med rekursion

Faktorial av ett tal n definieras som $n! = n \cdot (n-1) \cdot (n-2) \cdots 1$.

Rekursiv definition:

$$\text{faktorial}(n) = \begin{cases} 1 & \text{om} \quad n = 0 \\ n \cdot \text{faktorial}(n-1) & \text{om} \quad n > 0 \end{cases}$$

Python-kod:

```python
def faktorial(n):
    if n == 0:  # Basfall
        return 1
    else:  # Rekursivt fall
        return n * faktorial(n - 1)

print(faktorial(5))  # Output: 120
```

Faktorial med rekursion

Exempel 3: Binärsökning med rekursion

Binärsökning hittar ett element i en sorterad lista genom att dela sökområdet i två delar vid varje steg.

Rekursiv algoritm:

```
START
    Om lista är tom: returnera False
    Beräkna mittenindex
    Om nyckel = mitten: returnera True
    Om nyckel < mitten: sök i vänstra halvan
    Om nyckel > mitten: sök i högra halvan
END
```

Python-kod:

```python
def binsok(lista, nyckel, vänster, höger):
    if vänster > höger:  # Basfall
        return False

    mitten = (vänster + höger) // 2

    if lista[mitten] == nyckel:  # Hittat nyckeln
        return True
    elif nyckel < lista[mitten]:  # Sök i vänstra halvan
        return binsok(lista, nyckel, vänster, mitten - 1)
    else:  # Sök i högra halvan
        return binsok(lista, nyckel, mitten + 1, höger)

# Testa med en sorterad lista
sorterad_lista = [1, 3, 5, 7, 9, 11]
nyckel = 7
print(binsok(sorterad_lista, nyckel, 0, len(sorterad_lista) - 1))  # Output: True
```

Binärsökning med rekursion

Exempel 4: Rekursivt fibonaccital

Ett exempel som visar rekursiv ineffektivitet:

```python
def fibonacci(n):
    if n <= 1:  # Basfall
        return n
    else:  # Rekursivt fall
        return fibonacci(n - 1) + fibonacci(n - 2)

print(fibonacci(6))  # Output: 8
```

Fibonaccital med rekursion

Observera: Rekursiva algoritmer som denna kan vara ineffektiva och bör förbättras med memoization eller iteration för stora n.