Programmering i Ma1c
– ett specialanpassat utbildningsmaterial för lösning
av sannolikhetsproblem

Computer Programming in Mathematics in Swedish
Upper Secondary School – a Customized Educational
Material for Solving Probability Problems

Johnny Åberg

Handledare: Björn Textorius
Examinator: Peter Frejd
**Språk**  x Svenska/Swedish  
Engelska/English

**Rapporttyp**  Examensarbete, forskningsproduktion

**ISRN-nummer**  LiU-LÄR-MG-A--2019/09--SE

---

**Titel**  Programmering i Ma1c – ett specialanpassat utbildningsmaterial för lösning av sannolikhetsproblem

**Title**  Computer Programming in Mathematics in Swedish Upper Secondary School – a Customized Educational Material for Solving Probability Problems

**Författare**  Johnny Åberg

---

**Sammanfattning**
Från den 1 juli 2018 ingår matematisk problemlösning med programmering i ämnet matematik på gymnasieskolans teknik- och naturvetenskapsprogram. Den här studien utgår från den nya kursplanen för Matematik 1c (Ma1c), försöker göra en rimlig tolkning av Skolverkets skrivningar, presenterar ett specialanpassat utbildningsmaterial avsett för kursens programmeringsdel och beskriver hur materialet har utarbetats. Materialet består av en projecktorpresentation och ett elevhäfte med övningsuppgifter, fokuserar på sannolikhetsproblemen i Ma1c, är provat i tre mindre och osystematiska pilotförsök, är inte helt utprovat och behöver förbättras ytterligare. Givet studiens tolkning, och 9 timmar undervisning i programmering, finns det goda skäl att misstänka att eleverna inte kommer att hinna lära sig matematisk problemlösning med programmering i Ma1c.

**Abstract**
From July 1st 2018 solving mathematical problems with computer programming is included in mathematics in Swedish upper secondary school. This study tries to make a reasonable interpretation of the unclear wording in the new curriculum and develops a customized educational material for the programming part of the course based on the interpretation. The material consists of a projector presentation and a booklet with exercises, focuses on probability problems, has been tried in three minor and unsystematic trials, is not fully tested and needs to be further improved. Given the study’s interpretatio, and 9 hours of teaching in programming, there are good reasons to suspect that the pupils will not learn how to solve mathematical problems with computer programming due to shortage of time.

**Nyckelord**  gymnasiet, Matematik 1c (Ma1c), problemlösning, sannolikhetsproblem, programmering, utbildningsmaterial, specialanpassat, Python 3, teoretiska designprinciper, erfarenheter, pilotförsök
Förord


Den här studien genomfördes 2017/19 och är ett kompletterande matematikdidaktiskt examensarbete på 15 högskolepoäng. Med min bakgrund är det inte förvånande att mina handledare tidigt föreslog att studien skulle handla om programming i någon form. Det visade sig vara ett klokt förslag, då jag fick stor nytta av mina kunskaper och långa erfarenhet av programvaruutveckling inom främst svensk industri. Kanske blev studien intressantare av mitt utifrånperspektiv?

Att skriva en vetenskaplig rapport 28 år efter min civilingenjörsexamen, i ett icke-tekniskt och för mig nytt ämne, på distans och parallellt med ett heltidsarbete visade sig vara en stor utmaning. Så ett stort tack till matematiska institutionen på Linköpings universitet som trots att arbetet tog betydligt längre tid än planerat alltid var positiva, uppmuntrande, flexibla och hjälpsamma; i synnerhet min handledare Björn Textorius och examinator Peter Frejd.

Tack även till eleverna och övriga deltagare i pilotförsöken. Som vanligt var det ett nöje att undervisa.

Nyköping 22 april 2019

Johnny Åberg
Innehållsförteckning

1 Inledning .......................................................................................................................... 7
  1.1 Undervisning kring digitalisering ............................................................................. 7
  1.2 Nationell IT-strategi ................................................................................................. 7
  1.3 Förändrade kursplaner .............................................................................................. 8
  1.4 Kritik mot förändringen ............................................................................................ 9
  1.5 Specialanpassat utbildningsmaterial saknas ............................................................. 9

2 Bakgrund .......................................................................................................................... 10
  2.1 Litteraturöversikt .................................................................................................... 10
  2.2 Syfte och frågeställning ......................................................................................... 12
  2.3 Skolverkets förtydligande ....................................................................................... 12
  2.4 Skolverkets otydlighet ......................................................................................... 14

3 Forskningsmetod ............................................................................................................. 16
  3.1 Development Studies ............................................................................................... 16
  3.2 Spider’s Web ........................................................................................................... 17
  3.3 Urval av deltagare till pilotförsök ........................................................................... 17
  3.4 Etiska överväganden .............................................................................................. 18

4 Genomförande ............................................................................................................... 20
  4.1 Översikt .................................................................................................................... 20
  4.2 Tolka Skolverkets skrivningar ............................................................................... 21
    4.2.1 Stora frihetsgrader ............................................................................................ 21
    4.2.2 Hög ambitionsnivå ........................................................................................... 21
    4.2.3 Inga förkunskaper i programmering ................................................................ 21
    4.2.4 Traditionell undervisning tillåten ................................................................... 21
    4.2.5 Ett förenklat Yatzyproblem ........................................................................... 21
  4.3 Bestämma programmeringsundervisningens omfattning ........................................ 22
  4.4 Formulera designprinciper för utbildningsmaterialet ............................................. 22
    4.4.1 Sammanfattning ............................................................................................... 23
    4.4.2 D1: Omfattning ............................................................................................... 23
    4.4.3 D2: Delar ......................................................................................................... 23
    4.4.4 D3: Undervisningsspråk ............................................................................... 24
    4.4.5 D4: Förkunskaper ........................................................................................... 24
    4.4.6 D5: Programspråk ........................................................................................... 24
    4.4.7 D6: Utvecklingsmiljö ....................................................................................... 24
4.4.8 D7: Färdigheter .......................................................................................................................... 25
4.4.9 D8: Byggblock .......................................................................................................................... 25
4.4.10 D9: Utvecklingsmetod .......................................................................................................... 26
4.4.11 D10: Kodom .............................................................................................................. 27
4.4.12 D11: Framgångsfaktorer .................................................................................................... 27
4.5 Färdigställa materialet .............................................................................................................. 28
4.5.1 Övergripande undervisningssekvens ..................................................................................... 28
4.5.2 Större moment ....................................................................................................................... 28
4.5.3 Kopplingsar .......................................................................................................................... 29
4.6 Prova materialet ....................................................................................................................... 29
4.6.1 Pilotförsök 1: Programmeringskurs för elever på NT-programmen ...................................... 30
4.6.2 Pilotförsök 2: Studiedag för gymnasielärare i matematik .................................................. 30
4.6.3 Pilotförsök 3: Workshop för lärarstudenter ..................................................................... 30
5 Resultat ........................................................................................................................................ 31
5.1 Specialanpassat utbildningsmaterial .................................................................................... 31
5.2 Teoretiska designprinciper .................................................................................................... 31
5.3 Erfarenheter från pilotförsöken ........................................................................................... 31
5.3.1 Programmeringsundervisningen ......................................................................................... 31
5.3.1.1 Varierande undervisningsresultat ............................................................................... 31
5.3.1.2 Ingen löste [FY] ...................................................................................................... 31
5.3.2 Utbildningsmaterialet ......................................................................................................... 31
5.3.2.1 Inte färdigt ............................................................................................................. 31
5.3.2.2 För omfattande ....................................................................................................... 32
5.3.2.3 Utmanande designprinciper ............................................................................... 32
6 Diskussion ................................................................................................................................ 33
6.1 Studiens tillförlitlighet ............................................................................................................. 33
6.2 Studiens forskningsmetod ..................................................................................................... 33
6.3 Möjliga förbättringar .............................................................................................................. 33
6.3.1 Programmering i Malc. ........................................................................................................ 33
6.3.2 Utbildningsmaterialet ........................................................................................................ 33
6.3.3 Programmeringsundervisningen ...................................................................................... 34
6.4 Brister i studien ..................................................................................................................... 34
6.4.1 Spretighet .......................................................................................................................... 34
6.4.2 Dålig förankring i forskning ......................................................................................... 34
6.4.3 Osystematiska pilotförsök ............................................................................................. 34
1 Inledning

1.1 Undervisning kring digitalisering


1.2 Nationell IT-strategi

Många länder har de senaste åren visat stort intresse för IT, programmering och digitalisering av sitt utbildningsväsende. Stora investeringar har gjorts trots bland annat osäkra resultat (Skolverket, 2016a, s.11):

"Det finns alltså, trots det stora intresset för och de stora investeringarna i ökad digitalisering av skolväsendet, relativt lite samlad kunskap om hur olika länder har arbetat strategiskt kring dessa digitaliseringsfrågor och vilka resultat arbetet har givit."


"Det krävs betydande insatser för att mer systematiskt studera de digitala praktikernas effekter och vilka som bör spridas. Inte minst gäller detta om och hur programmering i skolan bidrar till ökat lärande, vilket i samband med nya skrivningar i styrdokumenten och därpå följande nya praktiker i skolan särskilt skulle kunna studeras."

Även Kjällander, Åkerfeldt och Petersen (2016, s. 30) påpekar bristen på forskning och att viss försiktighet är befogad:

"Denna översikt visar på bristen av forskning om programmering i skolan och det i sig skulle kunna tolkas som ett skäl att avvakta med att skriva in programmering i kursplanerna, eftersom undervisningen ska bygga på vetenskaplig grund."
Rolandsson (2012) skriver att den svenska gymnasieskolans undervisning i informatik, där programmering ingår, sällan undersökt inom didaktisk forskning. I studiens diskussion och slutsats framgår att ämnet är mycket besvärligt, att en didaktisk repertoar behöver utvecklas och att man kan ifrågasätta om programmeringsundervisning ska erbjudas i gymnasieskolan. En anledning är att många av lärarna i Rolandssons (2012, s. 57) studie har följande uppfattning:

"I uppsatsen framgår med all tydlighet att en majoritet av lärare anser att inte alla elever kan lära sig programmering, p.g.a. avsaknad av logiska och analytiska förmågor."


Bristen på forskning gäller inte bara svenska skolan. Även internationellt behövs mer forskning om till exempel undervisningsmetoder i grundläggande programmering (Adams, Bennedsen, Devlin, Malmi, Mannila, Paterson, Pears & Seidman, 2007, s. 211):

"Our literature review leads us to believe that larger-scale systematic studies are vital to a better long-term understanding of how to teach this [introductionary computer programming] key element of the field."

1.3 Förändrade kursplaner

I regeringens uppdrag ingick att vid behov ge förslag på förändringar av gymnasieskolans läroplan och dess ämnesplan för matematik. Trots bristen på forskning, och att Skollagen (SFS 2010:800) tydligt framhåller att utbildningen ska vila på vetenskaplig grund och beprövat erfarenhet, föreslog Skolverket att *programmering* skrivs in under rubriken *Problemlösning* i kursplanerna för Matematik 1c (Ma1c), 2c, 3c, 4 och 5. Den nya skrivningen i de förändrade kursplanerna, vilka fastställdes av regeringen den 9 mars 2017 och gäller fr.o.m. den 1 juli 2018, säger att undervisningen i kurserna ska behandla (Skolverket, 2019, s. 9):

[SS1] - ”Strategier för matematisk problemlösning inklusive modellering av olika situationer, såväl med som utan digitala verktyg och programmering.”

Denna skrivning i Ma1c är studiens utgångspunkt och kommer fortsättningsvis att refereras till med [SS1].
1.4 Kritik mot förändringen

Förändringen av kursplanerna är inte oproblematisk; flera remissinstanser och andra är kritiska. Bland annat finns en utbredd oro för att skolan är oförberedd samt att programmeringsundervisningen kommer att vara svårgenomförbar, resurskrävande och leda till försämrade matematikkunskaper (t.ex. Skolverket, 2016b; Larsson, 2017b; Sandell, 2017).

1.5 Specialanpassat utbildningsmaterial saknas

Många matematiklärare är oförberedda, osäkra och har bristande programmeringskunskaper (Larsson, 2017a), men behöver troligen ändå själva utarbeta delar av sitt utbildningsmaterial till programmeringsundervisning i Ma1c. Det är en utmanande och tidskrävande uppgift. De färdiga utbildningsmaterial i grundläggande programmering, vilka författaren till denna rapport utvärderade inför studien, visade sig nämligen enligt honom vara för generella och omfattande samt dessutom skrivna på engelska. De uppfylldes inte heller centrala krav som att utbildningsmaterialiet ska vara minimalt, anpassat till elevernas specifika förkunskaper, med en ytterst begränsad undervisningstid och behandla matematik som ingår i Ma1c.

Bristen på utbildningsmaterial för undervisning av matematisk problemlösning med programmering i Ma1c är en viktig anledning till studiens tillkomst. Förutom att lämpliga och beprövade utbildningsmaterial saknas, talar fler skäl för att utarbeta ett eget och specialanpassat material: det kan anpassas till studiens behov, kan snabbt ändras för att prova nya idéer, är gratis och har ingen upphovsrätt att ta hänsyn till.
2 Bakgrund

2.1 Litteraturöversikt

Inledningsvis är det viktigt att lägga på minnet ett påpekande i Kjällander m.fl. (2016) att begreppet *programmering* inom ramen för utbildning delvis skiljer sig åt i olika studier. Begreppet är sällan definierat, kan betyda olika saker och det är ofta oklart vilka aktiviteter som ingår och i vilken omfattning. Påpekandet gäller även här.

Som framgår av kapitel 1.2 finns få studier om programmering i den svenska skolan. Att hitta studier som förutom att behandla undervisning av gymnasieelever i grundläggande programmering även omfattar matematisk problemlösning, integration i ämnet matematik och utbildningsmaterial visade sig vara svårt med den tillgängliga tiden. Det är förklaringen till den begränsade litteraturöversikten och användningen av examensarbeten i studien.

Något som svensk forskning dock kommit fram till är svårigheten att undervisa elever i programmering i gymnasieskolan (Kjällander m.fl., 2016, s. 11):

"Under seminarierna, som Rolandsson genomfört, menar lärarna att trots att lärare undervisar i programmering på olika sätt, har många elever svårt att förstå ämnet."

Även Eckerdal (2009, s. 63) pekar på stora svårigheter kring undervisning i programmering, men på universitetsnivå:

"Forskning i datavetenskapens didaktik visar emellertid att nybörjarstudenter har stora svårigheter att lära sig programmering. De rapporterade svårigheterna gäller såväl praktik som begreppsförståelse."

Internationellt är forskningsläget bättre, speciellt på universitetsnivå, men det har ändå varit svårt att hitta forskning som är relevant och användbar i studien. En litteraturöversikt av Adams m.fl. (2007, s. 211) undersöker kursplaner, pedagogik, programspråk och utvecklingsmiljöer för undervisning av nybörjare i programmering på universitetsnivå och konstaterar:

"We conclude that despite the large volume of literature in this area, there is little systematic evidence to support any particular approach. For that reason, we have not attempted to give a canonical answer to the question of how to teach introductory programming."

Översikten är mest tillämplig för undervisning i avsnitten om programspråk och utvecklingsmiljöer, vilka innehåller följande tips: (a) att använda programspråket Python 3, (b) understryker behovet att visualisera program på olika sätt samt (c) nämner att en *debugger*
är ett sätt att bland annat undersöka hur ett program fungerar, felsöka programmet och visa hur variabelvärdena ändras när det exekveras.

Gries (1974) behandlar vilket innehåll som bör ingå i introduktionskurser i programmering på universitetsnivå. Ett område som lyfts fram är *How to solve problems* och studien har några förslag på hur undervisningen i generell problemlösning ska bedrivas. Andra delar av studien handlar om att dela in och bara använda en begränsad del av programspråket. En annan litteraturöversikt sammanfattar nybörjares svårigheter att lära sig programmera i introduktionskurser på universitetsnivå (Robins, Rountree & Rountree, 2003, s.162):

"It is clear that novice programmers face a very difficult task. Learning to program involves acquiring complex new knowledge and related strategies and practical skills."

Litteraturöversikten beskriver bland annat flera svårigheter, ger förslag på vad som kan göras för att förbättra undervisningen och presenterar ett ramverk för programmering som är tänkt att vara ett stöd för planering, genomförande och uppföljning av programmerings-undervisning. Förslagen omfattar att eleverna ska arbeta praktiskt och undervisas i att använda till exempel (a) *schema/plan* och en utvecklingsmetod för att underlätta skrivandet av program, (b) *close tracking* som innebär att ”ta datorns perspektiv”, exekvera och felsöka ett program radvis i huvudet, (c) *graphical languages* för att beskriva programmens kontrollflöde, (d) en *notional machine* som är en starkt förenklad modell av datorns samt (e) lämpliga generella problemlösningsmetoder.

Ytterligare en litteraturöversikt (Lehman & Qian, 2017) pekar ut tre områden där elever som ska lära sig programmering ofta har svårigheter: *syntactic knowledge* (bl.a. kunskap om programspråket, dess regler och hur ett programs byggblock ska skrivas på teckennivå), *conceptual knowledge* (hur varje byggblock fungerar, vad det gör och på vilket sätt det kan användas i program) och *strategic knowledge* (hur byggblock kan kombineras i program för att lösa matematiska problem samt nödvändiga färdigheter som planering, felsökning och testning av program). Studien undersöker orsaker till svårigheterna och ger några förslag på hur de kan överkommas genom att till exempel (a) presentera tydliga programexempel för eleverna, (b) undervisa explicit i programmeringsstrategier och användning av *schema/pattern*, (c) låta eleverna samarbeta, (d) utnyttja en *debugger* för felsökning av program samt (e) använda *syntax highlighting* och *code completion* för att markerar när ett byggblock är rätt skrivet enligt programspråkets regler.

Det finns några examensarbeten som behandlar ämnesområden av intresse i studien till exempel några lärares tolkningar av kursplaner i ämnet programmering (Depta & Orb, 2015), några gymnasielärares attityder till programmering i ämnet matematik (Arkå Nilsson, 2018; Sandell, 2017) samt några gymnasielärares och elevers uppfattningar om programmering i kursen Matematik 5 (Götling &Löfwenhamns, 2018). I brist på användbar forskning används dessa examensarbeten i studien, men restriktivt.
Sammanfattningsvis har forskningen inget entydigt svar på frågan om hur nybörjare bäst ska undervisas i programmering (Adams m.fl., 2007) eller matematisk problemlösning med programmering. Hur ett utbildningsmaterial för sådan programmeringsundervisning bör organiseras och presenteras för elever är ännu mer oklart.

2.2 Syfte och frågeställning

Studiens utgångspunkt är alltså [SS1]. Eftersom skrivningen nu ska tillämpas finns det goda skäl att genomföra studier som undersöker olika tolkningar och deras konsekvenser. Den här studien bidrar med en av många möjliga tolkningar och presenterar ett av alla möjliga specialanpassade utbildningsmaterial, för undervisning av matematisk problemlösning med programmering i Ma1c, som tolkningen kan leda till.

Studien kan vara av intresse för matematiklärare på teknik- och naturvetenskapsprogram (NT-programmen) och lärarstudenter, vilka ska implementera programmeringsundervisning i ämnet matematik. En förhoppning är att de får viss överblick, bättre förståelse för möjligheter och problem med programmering i ämnet matematik samt sparar tid genom att implementera, testa och utveckla delar av studiens utbildningsmaterial i sin egen undervisning.

Materialet fokuserar på sannolikhetsproblem i Ma1c på grund av Skolverkets förtydligande [SS2] i kapitel 2.3, men kan enkelt anpassas och kompletteras med fler matematiska problem som handlar om till exempel talföljder, primtal samt ränta och även användas i andra kurser än Ma1c. Trots studiens begränsade storlek är förhoppningen att den bidrar med ny kunskap om främst programmering i ämnet matematik och blir utgångspunkt för fler studier.

Studiens övergripande syfte är att utifrån [SS1] utarbeta, och i några mindre pilotförsök prova, ett specialanpassat utbildningsmaterial i programmering avsett för instruktionsorienterad undervisning (Skolverket, 2018a). Målsättningen med utbildningsmaterialet är att utveckla elevers problemlösning förmåga, med fokus på sannolikhetsproblem i Ma1c, så att eleverna efter avslutad undervisning kan skriva ett program som löser det förenklade Yatzyproblemet [FY] i kapitel 4.2.5. Två aspekter av det övergripande syftet är i fokus: utvecklandet av designen av utbildningsmaterialet och erfarenheter från dess användning.

Frågeställningen är:

- Utgående från studiens tolkning av [SS1], vilka teoretiska designprinciper kan användas för att utarbeta utbildningsmaterialet?
- Vilka är erfarenheterna från användningen av utbildningsmaterialet i pilotförsöken?

2.3 Skolverkets förtydligande

Delar av de förändrade kursplanerna (Skolverket, 2019) är otydligt formulerade. Det gäller även [SS1] eftersom Skolverket inte har specificerat vad som avses med matematisk problemlösning med programmering (Skolverket, u.å., s. 13):

[SS3] - "Formuleringen som används är medvetet öppen för att tillåta stor variation både i vilken utsträckning programmering förekommer i undervisning och i vilka former."

12
utan förtydligar med följande exempel:

[SS2] - ”Programmering kan användas för att simulera situationer för att göra uppskattningar av sannolikheter, snarare än algebraisk-matematiska beräkningar. ”Vad är en sannolik värdeutveckling för en fond vars kurs ändras slumpvis enligt vissa mönster? Vad är chansen att slå Yatzy på högst 3 slag?” Programmering kan också användas för att utforska variation i sannolikheter – ”Hur vanligt är det att en 50/50-sannolikhet faller ut 60/40 eller mer skevt vid 100 observationer? 1000 observationer?”

Exemplet ovan ingår i det centrala innehållet sannolikhet och statistik i Ma1c, men förekommer sällan i vanliga matematikböcker på gymnasiet (t.ex. Alfredsson, Brolin, Erixon, Heikne & Ristamäki, 2009; Gennow, Gustafsson & Silborn, 2011; Holmström, Sjunnesson & Smedhamre, 2011) och har en högre svårighetsnivå jämfört med många andra sannolikhetsproblem i Ma1c.

Skolverket (2016b, s. 7) beskriver även kortfattat vad som avses med programmering:

[SS4] - ”I vissa sammanhang kan programmering vara synonymt med ’skrivande av kod’ medan det i andra sammanhang avses ett vidare perspektiv på programmering där även problemformulering, val av lösning, att pröva och ompröva samt att dokumentera räknas in i programmeringsaktiviteten. För detta krävs förmåga till kreativ problemlösning, logiskt tänkande, ett strukturerat arbetssätt och förmåga att generalisera [sic]. Det senare perspektivet på programmering har varit utgångspunkt för förslagen till förändringar i styrdokumenten.”

och i Skolverket (2017, s. 9) finns ytterligare en beskrivning:

[SS5] - ”I programmering ingår att skriva kod men det ska också ses i ett vidare perspektiv som omfattar kreativt skapande, styrning och reglering, simulering samt demokratiska dimensioner. Det här vidare perspektivet på programmering är en viktig utgångspunkt i undervisningen och programmering ingår därmed i alla aspekter av digital kompetens.”

Skolverket (2018a, s. 8) indikerar en mindre omfattning av begreppet programmering än [SS4] och [SS5]:

[SS6] - ”I gymnasieskolans matematikundervisning är huvudsyftet inte att utbilda eleverna i programmering utan att lära dem att använda programmering när de arbetar med att lösa matematiska problem. Ändå måste läraren understundom också fokusera på att bygga upp elevernas kunskaper om programmering och på att göra det möjligt för eleverna att utforma och utveckla egna program och att få saker att ske med hjälp av programmering.”
I modulen *Matematikundervisning med digitala verktyg II* framgår att programmeringsundervisningen behöver anpassas till nybörjare (Skolverket, 2018a, s. 2):

[SS7] - “Detta innebär att programmeringsundervisningen behöver anpassas till nybörjare (Skolverket, 2018a, s. 2):”

Skolverket (2018b, s. 7) innehåller en märkelig skrivning, vilken tillsammans med [SS7], skulle kunna tolkas som att eleverna ska lösa matematiska problem med programmering trots att de inte kan programmera och inte heller ska undervisas i programmering:

[SS8] - “Ämnesplanernas skrivningar innebär att programmering ska användas till att lösa matematiska problem, inte utbilda eleverna i programmering.”

Modulen *Matematikundervisning med digitala verktyg II* presenterar två olika sätt att undervisa i programmering, projektiorienterat och instruktionsorienterat, och uppmuntrar främst användning av den projektiorienterade varianten med aktiviteter som tinkering och orkestrering (Skolverket, 2018a, s. 9):


Hur matematiklärare tolkar [SS1], och de övriga skrivningarna [SS2-9], kommer i hög grad att påverka programmeringsundervisningen, utbildningsmaterialet, dess innehåll och omfattning. Skrivningarna kommer även fortsättningsvis att refereras till med [SS1-9].

2.4 Skolverkets otydlighet

Att det är svårt att tolka de knappa skrivningarna [SS1-9], eftersom tolkningsutrymmet är så stort, och att det finns många frågetecken kring hur programmeringsundervisningen ska implementeras i ämnet matematik på gymnasiet framgår tydligt i många examensarbeten (t.ex. Arkå Nilsson, 2018; Sandell, 2017). Götlöv och Löfwenhamn (2018, s. 9) skriver lite tillspetsat:

"...att det i styrdokumenten [Skolverkets] inte finns något som vittnar om syn på eller ställningstaganden om programmeringens natur."
Att skrivningarna är otydligt formulerade medför att Skolverkets målsättning och ambitionsnivå med [SS1] är högst oklar. Det kan bland annat skapa osäkerhet, försvåra forskning och diskussion samt i slutändan påverka både matematik- och programmeringsundervisningen i Ma1c. Behovet av förtydliganden från Skolverket är stort och till exempel typiska programmeringsuppgifter i nationella prov och bedömningsgrunder skulle bidra till minskad otydlighet. Författaren till denna rapport har ägnat ansenlig tid åt att försöka förstå vad Skolverket menar med skrivningarna, men osäkerheten i studien är fortfarande betydande. Det gäller speciellt innebörden av begreppet *programmering* i [SS1]. Götling och Löfwenhamn (2018, s. 14) får avsluta:

"Eftersom att Skolverket (2018) inte har tagit ett uttalat ställningstagande gällande rekommenderad nivå, val av kodspråk eller på explicit vis uppmärksammat vilka kunskaper som hos eleverna väntas komma av den programmeringsrelaterade undervisningen blir resultatet bristande direktiv till lärarkåren. De lärare som har omfattande kunskap om programmering och programmeringsdidaktik må ha friheter att föra in ämnet på sina lektioner som de vill men andra faller mellan stolarna."
3 Forskningsmetod

3.1 Development Studies

Studien tillämpar den generella forskningsmetoden Development Studies, vilket är en typ av Educational Design Research (EDR), med definitionen:

"...the systematic analysis, design and evaluation of educational interventions with the dual aim of generating research-based solutions for complex problems in educational practice, and advancing our knowledge about the characteristics of these interventions and the processes of designing and developing them." (van den Akker, Bannan, Kelly, Nieveen & Plomp, 2013, s. 16)

Den iterativa metoden som ingår i EDR visas i Figur 1 och omfattar fyra systematiska processer.

Metoden tillämpas i studien på nedanstående översiktliga sätt. Av tidsskäl är studien begränsad till en iteration av processerna.

- **Problem**: Problemet studien försöker lösa är att det saknas lämpliga utbildningsmaterial för undervisning av matematisk problemlösning med programmering i Malc. Behovet har uppstått efter Skolverkets nya skrivning [SS1] i den förändrade kursplanen.
- **Analysis**: Först identifieras programmeringsundervisningens innehåll och omfattning utifrån Skolverkets andra skrivningar [SS2-9].
- **Design & develop prototype**: Därefter formuleras teoretiska designprinciper innan ett specialanpassat utbildningsmaterial färdigställs.
- **Evaluation**: Till sist provas materialet i tre mindre pilotförsök.

Kapitel 4.1 beskriver mer detaljerat hur metoden har tillämpats i studien.
3.2 Spider´s Web

Som stöd i analysen i föregående avsnitt använder studien ramverket Curricular Spider´s Web i Figur 2 (van den Akker m.fl., 2013). Ramverket hjälper till att identifiera, beskriva, skapa balans och kompatibilitet mellan komponenterna i programmeringsundervisningen.

![Curricular Spider’s Web diagram](image.png)

Figur 2: Curricular Spider’s Web i educational design research (van den Akker m.fl., 2013, s. 58).

Analysen fokuserar främst på följande komponenter i figuren:

- **Rationale** (studiens översättning: Motiv)
  Den ursprungliga och huvudsakliga anledningen till att undervisningen behövs.

- **Aims and objectives** (Mål)
  Syftet med undervisningen och vad eleverna ska lära sig.

- **Content** (Innehåll)
  Undervisningens omfattning och begränsningar.

- **Learning activities** (Aktiviteter)
  Hur eleverna ska undervisas; metoder och former

- **Teacher Role** (Lärarroll)
  Lärarens uppgift och arbetssätt i undervisningen.

- **Materials and resources** (Resurser)
  Materiel, utrustning och andra resurser som behövs i undervisningen.

De utelämnade komponenterna påverkar inte utarbetandet av det specialanpassade utbildningsmaterialet, vilket är studiens övergripande syfte, i någon högre utsträckning. Tabell 1 på sida 22 visar hur ramverket Curricular Spider’s Web har använts i studien.

3.3 Urval av deltagare till pilotförsök

Det var tillfälligheter som bestämde deltagarna i studiens tre olika pilotförsök. Det första pilotförsöket (i vilket ca 3 elever deltog) genomfördes efter att en gymnasielärare i matematik
visade intresse för en kurs i grundläggande programmering för hennes elever, det andra (ca 10 gymnasielärare i matematik) genomfördes efter en kortare diskussion med huvudläraren i matematik och det tredje (ca 16 lärarstudenter) hade sin upprinnelse i en förfrågan från matematiska institutionen på Linköpings universitet.

Urvälet är vare sig stort, slumpmässigt eller representativt för Ma1c och det är ett typiskt exempel på bekvämlighetsurval (Bryman, 2013). I det första pilotförsöket finns det dessutom anledning att misstänka att eleverna har högre motivation, och bättre förutsättningar att tillgodogöra sig programmeringsundervisningen, än genomsnittseleven i Ma1c. I andra och tredje pilotförsöken är kunskapsnivån i matematik och programmering avsevärt högre jämfört med genomsnittseleven i Ma1c.

Trots bristerna är författarens bedömning att urvalet är acceptabelt, då pilotförsökens syfte endast var att få återkoppling och identifiera möjliga förbättringar i utbildningsmaterialet.

### 3.4 Etiska överväganden

Studien följer god forskningssed (Vetenskapsrådet, 2017) och de rekommenderade etiska principerna inom den humanistisk-samhällsvetenskapliga forskningen (Vetenskapsrådet, 2002). Det omfattar bland annat ett grundläggande individskydd formulerat i informationskravet, samtyckeskravet, konfidentialitetskravet och nyttjandekravet. Individskyddet i studien realiserades genom att informera alla elever, och övriga deltagare i pilotförsöken, om:

- Studien genomförs under 2017/18 och ingår i ett examensarbete vid Linköpings universitet om lösning av matematiska problem med programmering i Ma1c.
- Deltagarnas enda uppgift är att gå en programmeringskurs i programspråket Python 3 med fokus på lösning av sannolikhetsproblem. Syftet är främst att prova ett utbildningsmaterial.
- Deltagandet är frivilligt och kan avbrytas när som helst utan motivering eller påföljd.
- Anteckningar och annat arbetsmaterial kommer under hela studien att förvaras på ett säkert sätt och skyddas från obehöriga.
- Arbetsmaterialet kommer endast att användas i studien, därefter förstöras och bara leva vidare i studiens rapport.
- Deltagarna är anonyma i rapporten.
- Rapporten kommer att publiceras, kan läsas av många och användas i annan forskning.

Informationen ovan finns med i projektorpresentationens introduktion (se bilaga 5) som visades i början av varje pilotförsök. Eleverna kan inte förväntas känna till individskyddet och fick därför en noggrannare genomgång än övriga deltagare. Elevernas matematiklärare och rektor är informerade om studien och elevernas deltagande på en övergripande nivå.
Författarens bedömning är att ovanstående är tillräcklig för att uppfylla individskyddet, då studien inte behandlar någon känslig eller personlig information och programmeringskurser får betraktas som okontroversiella.
4 Genomförande

4.1 Översikt

För att uppfylla studiens övergripande syfte och besvara frågeställningen i kapitel 2.2 genomfördes följande steg med stöd av forskningsmetoden Development Studies i kapitel 3.1:


3. Formulera teoretiska designprinciper för utbildningsmaterialet utifrån komponenterna samt författarens mångåriga arbete som programmerare och egna erfarenheter kring undervisning i programmering. Se kapitel 4.4.


5. Prova och förbättra utbildningsmaterialet samt samla erfarenheter i tre mindre och osystematiska pilotförsök. Se kapitel 4.6.


Studien är främst byggd på författarens mångåriga arbete som programmerare, preferenser och egna erfarenheter kring undervisning i matematik och programmering. Genom åren har han använt många olika utvecklingsmodeller, programmeringsmetoder, hjälpmedel samt verktyg och utvecklat ett fungerande arbetssätt. Studien bygger även på hans undervisning i matematik och fysik på NT-programmen under senare år.
4.2 Tolka Skolverkets skrivningar
   För att tolka den otydliga skrivningen [SS1], och försöka förstå Skolverkets målsättning och ambitionsnivå på programmeringsundervisning i Ma1c, tar författaren hjälp av skrivningarna [SS2-9].
   Detta avsnitt redogör för författarens tolkning av skrivningarna och är det första steget i att utarbeta utbildningsmaterialet.

4.2.1 Stora frihetsgrader
   Författaren tolkar [SS3], [SS6], [SS8] och [SS9] som att frihetsgraderna är stora och att Skolverket inte ställer krav på någon minimal programmeringsundervisning. Varje matematiklärare får med andra ord själv bestämma omfattningen och hur undervisningen ska bedrivas.

4.2.2 Hög ambitionsnivå

4.2.3 Inga förkunskaper i programmering
   Författaren tolkar [SS7] som att programmeringsundervisningen och utbildningsmaterialet ska anpassas till nybörjare utan förkunskaper i programmering.

4.2.4 Traditionell undervisning tillåten

4.2.5 Ett förenklat Yatzyproblem
   Författaren tolkar [SS2] som exempel på sannolikhetsproblemen med högst svårighetsnivå, vilka eleverna förväntas kunna lösa med programmering i Ma1c, och vad Skolverket menar med matematisk problemlösning i [SS1]. Av skrivningens fyra problem har författaren valt att utgå från Vad är chansen att slå Yatzy på högst 3 slag? och skapa ett nytt med lägre svårighetsnivå. Det nya och förenklade Yatzyproblemet är mer representativt för sannolikhetsproblem i Ma1c samt styr i hög grad utbildningsmaterialets innehåll och omfattning. Yatzyproblemet är centralt i studien och kommer fortsättningsvis att refereras till med [FY].

   [FY] - Du kastar 5 tärningar 3 gånger och får inte spara någon tärning mellan kasten. Vad är sannolikheten att få Yatzy minst 1 gång?
Notera att [FY] bryter mot spelreglerna i Yatzy, då inga tärningar får sparas mellan kasten utan alla kast måste göras med 5 tärningar. Det innebär att även om 3 av tärningarna visar två ögon efter första kastet, så får de inte sparas utan nästa kast måste göras med 5 tärningar ändå.

Trots att [FY] är betydligt enklare att lösa än det ursprungliga, både analytiskt och med programmering (se bilaga 1), är problemet troligen utmanande för de flesta eleverna i Ma1c. Som jämförelse, och för att understryka att alla problemen i [SS2] har hög svårighetsnivå för nybörjare i programmering och jämfört med typiska sannolikhetsproblem i Ma1c, visar bilaga 2 exempel på program som löser Hur vanligt är det att en 50/50-sannolikhet faller ut 60/40 eller mer skevt vid 100 observationer? och Vad är chansen att slå Yatzy på högst 3 slag? om spelreglerna i Yatzy följs.

4.3 Bestämma programmeringsundervisningens omfattning

Med hjälp av [FY] och föregående avsnitt identifierar författaren komponenterna i programmeringsundervisningen med hjälp av ramverket Curricular Spider’s Web i kapitel 3.2.

Detta avsnitt sammanfattar komponenterna och är det andra steget i att utarbeta utbildningsmaterialet.

<table>
<thead>
<tr>
<th>Komponent</th>
<th>Sammanfattning</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1: Motiv</td>
<td>Skolverkets nya skrivning [SS1] i kursplanen för Ma1c innebär att programmering numera ingår i det centrala innehållet.</td>
</tr>
<tr>
<td>K2: Mål</td>
<td>Eleverna ska efter avslutat programmeringsundervisning kunna skriva ett program som löser [FY].</td>
</tr>
<tr>
<td>K4: Aktiviteter och</td>
<td>Undervisningen ska vara instruktionsorienterad (Skolverket, 2018a) och bedrivs på ett sätt som eleverna känner igen: lärarledda gemensamma genomgångar och därefter eget arbete med att lösa övningsuppgifter.</td>
</tr>
<tr>
<td>K5: Lärarroll</td>
<td>Genomgångarna bedrivs på ett sätt som eleverna känner igen: lärarledda gemensamma genomgångar och därefter eget arbete med att lösa övningsuppgifter.</td>
</tr>
</tbody>
</table>

Tabell 1: Programmeringsundervisningens komponenter enligt Curricular Spider’s Web.

4.4 Formulera designprinciper för utbildningsmaterialet

Utifrån komponenterna i Tabell 1 formulerar författaren ett antal teoretiska designprinciper för utbildningsmaterialet. Designprinciperna preciserar materialets innehåll och omfattning samt redovisar några större val som gjorts. När inte eftersökt eller relevant information har hittats inom rimlig tid hos Skolverket eller i andra studier, har författarens preferenser, kunskaper och erfarenheter i programmering samt egna erfarenheter kring lärande i
programmering använts i arbetet. Många av designprinciperna härstammar från när författaren skrev programmet som löser [FY] i bilaga 1.

Detta avsnitt beskriver, motiverar och sammanställer designprinciperna samt valen och är det tredje steget i att utarbeta utbildningsmaterialet.

### 4.4.1 Sammanfattning

<table>
<thead>
<tr>
<th>Designprincip</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1: Omfattning</td>
<td>minimal</td>
</tr>
<tr>
<td>D2: Delar</td>
<td>digitalt, projektorpresentation, elevhäfte med övningsuppgifter</td>
</tr>
<tr>
<td>D3: Undervisningsspråk</td>
<td>svenska</td>
</tr>
<tr>
<td>D4: Förkunskaper</td>
<td>nybörjare, sannolikhetsläran i Ma1c</td>
</tr>
<tr>
<td>D5: Programspråk</td>
<td>Python 3</td>
</tr>
<tr>
<td>D6: Utvecklingsmiljö</td>
<td>PC, Windows 10, IDLE</td>
</tr>
<tr>
<td>D7: Färdigheter</td>
<td>analysera, skriva, ändra, starta, avbryta, testa, felsöka</td>
</tr>
<tr>
<td>D8: Byggblock</td>
<td>visa på skärmen, vänta på inmatning från tangentbordet, beräkna matematiska uttryck, generera slump tal, välja rader, repetera rader, använda kommentarer, variabler</td>
</tr>
<tr>
<td>D9: Utvecklingsmetod</td>
<td>iterativ, inkrementell</td>
</tr>
<tr>
<td>D10: Kodmall</td>
<td>slumpförsök</td>
</tr>
<tr>
<td>D11: Framgångsfaktorer</td>
<td>praktisera, kopiera, experimentera, repetera, samarbeta</td>
</tr>
</tbody>
</table>

Tabell 2. Teoretisk designprinciper som använts för att utarbeta utbildningsmaterialet.

### 4.4.2 D1: Omfattning

**Val:** Utbildningsmaterialets omfattning är minimal.

**Motivering:** Skälet är att (a) minimera undervisningstiden och påverka matematikundervisningen i Ma1c så lite som möjligt, (b) underlätta för eleverna; ett mindre material är enklare att lära sig, överblicka och använda när de ska programera själva.

**Kommentar:** Trots att Skolverket lagt till programmering i Ma1c har de inte tagit bort något i kursplanens centrala innehåll, ökat undervisningstiden eller minskat kunskapskraven. Några konsekvenser är att omfattningen på matematikundervisningen behöver minskas för att skapa utrymme för programmeringsundervisningen och att eleverna måste lära sig kursens matematik på kortare tid för att nå de oförändrade kunskapskraven. Robins m.fl. (2003) pågår i sin summering att utbildningsmaterialet i grundläggande programmering bör vara enkla.

### 4.4.3 D2: Delar

**Val:** Utbildningsmaterialet är digitalt och består av en projektorpresentation samt ett elevhäfte med övningsuppgifter.

**Motivering:** (a) Eleverna är vana vid lärarledda genomgångar följa av eget arbete med övningsuppgifter samt (b) digitala texter är lättare att kopiera och söka i.
Kommentar: Elevhäftet är en kortfattad lärobok, förberedd för utskrift på papper, som är tänkt att användas främst av eleverna. Presentationen är ett komplement till elevhäftet som läraren visar med projektor under de gemensamma genomgångarna.

4.4.4 D3: Undervisningsspråk
Val: Utbildningsmaterialet är skrivet på svenska.
Motivering: Det är skolans och elevernas normala undervisningssspråk.
Kommentar: Det är inte självklart att använda svenska i materialet. Python 3, IDLE och många facktermer har engelskt ursprung och saknar bra svenska översättningar. Det gör att språken blandas friskt i materialet; funktionellt men inte alltid så välvårdat.

4.4.5 D4: Förkunskaper
Val: Utbildningsmaterialet förutsätter att eleverna är nybörjare i programmering och behärskar sannolikhetsläran i Ma1c (t.ex. Alfredsson m.fl. 2009).
Motivering: (a) Skolverket är tydliga med att eleverna inte kan förutsättas kunna någon programmering [SS7], (b) programmering ingick inte i grundskolans kursplaner före 1 juli 2018 samt (c) det kommer att ta år innan undervisningen i Ma1c kan bygga vidare på grundskolans programmeringsundervisning.
Kommentar: Elevernas förväntade förkunskaper påverkar självklart utbildningsmaterialet innehåll, omfattning och detaljrikedom; ju sämre kunskaper och mindre erfarenhet, desto mer programmering och sannolikhetslära behöver materialet behandla.

4.4.6 D5: Programspråk
Motivering: Det är (a) textbaserat, representativt och framtaget för utbildningsändamål, (b) lämpligt för matematiska beräkningar, (c) populärt, växer och används på många universitet och företag, (d) gratis och utvecklat av en erkänd och uthållig icke-komersiell organisation samt (e) används i Skolverket (2018a, 2018b).
Kommentar: För att skriva program behövs ett programspråk. Många språk som Java, C och C++ har stora likheter (se bilaga 1), är enkla att lära sig när väl ett av språken behärskas och vilket som väljs spelar mindre roll (Adams m.fl., 2007). Det gäller i synnerhet i studien där en mycket begränsad del av Python 3 används (se kapitel 4.4.9).

4.4.7 D6: Utvecklingsmiljö
Motivering: (a) Eleverna får låna varsin PC med Windows 10 på gymnasieskolan där författaren undervisar, programmet och utvecklingsmiljön IDLE (www.python.org) (b) är enkel, lättanvänd och lämplig för nybörjare, (c) följer med då Python 3 installeras på en dator, (d) är stabil, snabb och representativ samt (e) tillhandahåller syntax highlighting och code completion vilket underlättar undervisning (t.ex. Lehman & Qian, 2017).
**Kommentar**: Förutom ett programspråk behövs även en utvecklingsmiljö för att skriva program. Den består ofta av en dator, ett operativsystem och ett speciellt program som måste vara installerat på datorn. IDLE finns för andra operativsystem som Linux och Mac OS och det går att hitta liknande utvecklingsmiljöer för mobila enheter som mobiltelefoner och läsplattor. Inte heller valet av utvecklingsmiljö spelar någon större roll (Adams m.fl., 2007), då eleverna i studien är nybörjare i programmering och bara behöver använda en mindre del av IDLE.

### 4.4.8 D7: Färdigheter

**Val**: Se Tabell 3.

**Motivering**: Färdigheterna (a) är fundamentala i programmering, (b) behövs för att lösa [FY] och materialets övningsuppgifter, (c) behövs för att uppfylla [SS4] samt (d) testa och felsöka kan användas för att förtvivla hur program byggs upp och fungerar (t.ex. Adams m.fl., 2007; Lehman & Qian, 2017).

**Kommentar**: Programmering är ett generellt och svårdefinierat begrepp (t.ex. Kjällander m.fl., 2016). Erfarna programmerare har sällan en gemensam uppfattning om vilka färdigheter som ingår och i vilken omfattning. Skolverkets användning av begreppet är också otydlig, vilket framgår av kapitel 2.3 och 2.4. Tabell 3 beskriver översiktligt några färdigheter som ingår i studiens användning av begreppet programmering.

<table>
<thead>
<tr>
<th>Färdighet</th>
<th>Beskrivning</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysera</td>
<td>Analysera och inse att ett matematiskt problem är lämpligt att lösa med programming. Färdigheten inbegriper att välja mellan olika alternativ och bestämma hur programmet ska fungera, byggas upp och testas.</td>
</tr>
<tr>
<td>skriva, ändra, starta, avbryta</td>
<td>Skriva, ändra, starta och avbryta programmet i utvecklingsmiljön. Att skriva omfattar att dokumentera programmet med enkla kommentarer, använda en kodmall och utvecklingsmetod.</td>
</tr>
<tr>
<td>testa</td>
<td>Kontrollera stegvis att programmet fungerar som tänkt och till sist ger rätt svar på det matematiska problemet.</td>
</tr>
<tr>
<td>felsöka</td>
<td>Hitta och rätta fel i programmet om det inte fungerar som tänkt.</td>
</tr>
</tbody>
</table>

*Tabell 3. Färdigheter som behandlas i utbildningsmaterialet.*

### 4.4.9 D8: Byggblock

**Val och motivering**: Se Tabell 4.

materialet även ut variabler, vilka är nödvändiga i många byggblock och för att binda samman dem. Byggblocken har konstruerats för att de ska vara enkla att förstå, använda, kombinera och undervisa. Byggblocken välja och repetera rader har ägnats speciell uppmärksamhet eftersom de har pekats ut som problematiska (Robins m.fl., 2003).

<table>
<thead>
<tr>
<th>Byggblock</th>
<th>Motivering</th>
</tr>
</thead>
<tbody>
<tr>
<td>visa på skärmen</td>
<td>Behövs för att lösa [FY].</td>
</tr>
<tr>
<td>vänta på inmatning från tangentbordet</td>
<td>Lämpligt byggblock i enklare och illustrativa övningsuppgifter.</td>
</tr>
<tr>
<td>beräkna matematiska uttryck</td>
<td>Behövs för att lösa [FY].</td>
</tr>
<tr>
<td>generera slumptal</td>
<td>Behövs för att lösa [FY].</td>
</tr>
<tr>
<td>välja rader</td>
<td>Behövs för att lösa [FY].</td>
</tr>
<tr>
<td>repetera rader</td>
<td>Behövs för att lösa [FY]. Valet av while, istället för till exempel for, har stöd i forskning (Robins m.fl., 2003)</td>
</tr>
<tr>
<td>använda kommentarer</td>
<td>Behövs enligt [SS4].</td>
</tr>
</tbody>
</table>

Tabell 4. Byggblock som behandlas i utbildningsmaterialet.

```python
from random import choice

# [Generera slumptal]...

h = 0
n = 0

while n < 1000000:
    antalYatzy = 0
    antalKast = 0
    while antalKast < 3:
        ögon1 = choice(range(6)) + 1
        ögon2 = choice(range(6)) + 1
        ögon3 = choice(range(6)) + 1
        ögon4 = choice(range(6)) + 1
        ögon5 = choice(range(6)) + 1
        if ögon1==ögon2==ögon3==ögon4==ögon5:
            antalYatzy = antalYatzy + 1
            antalKast = antalKast + 1
        if antalYatzy > 0:
            h = h + 1
            n = n + 1
            f = h/n
            print("P(Yatzy minst 1 gång): ca", f) [Skriv ut på skärmen]
```


4.4.10 D9: Utvecklingsmetod

Val: Utbildnings materialet inkluderar en iterativ och inkrementell utvecklingsmetod.

Motivering: Den (a) är konkret och effektiv, (b) visar hur program byggs upp och fungerar, (c) är lämplig för att lösa [FY] och många av materialets övningsuppgifter samt (d) följer praxis (Larman & Basili, 2003).

Kommentar: För att skriva lite större och mer avancerade program är det viktigt, speciellt för oerfarna programmerare, att ta hjälp av en beprövd utvecklingsmetod som visar hur program kan utformas, byggas upp och testas. Detta antyds av Robins m.fl. (2003) samt Lehman och Qian (2017). Metoden som används i studien bygger på att dela upp ett program i små, väl avgränsade och greppbara delar som var för sig är enklare att förstå, skriva, testa...

4.4.11 D10: Kodmall

**Motivering:** (a) [FY] är ett slumpförsök, (b) utbildningsmaterialet är fokuserat på sannolikhetsproblem samt (c) kodmallen är tydlig, illustrativ och kan enkelt konkretiseras med tärningar, papper och miniräknare.

**Kommentar:** Erfarna programmerare kopierar och återanvänder bra idéer, smarta lösningar samt delar av sina egna och andras program hela tiden. En kodmall är ett sätt att tydliggöra och beskriva en abstraktionsnivå och generell struktur ovanför byggblocken (se kapitel 4.4.9) som kan användas gång på gång för att lösa närliggande matematiska problem. Mallen innehåller ett embryo som kan kopieras och bli startpunkt för nya program; vilket sparar tid, underlättar och ibland är avgörande för att lyckas överhuvudtaget. Vikten av tydliga programexempel påpekas även av Lehman och Qian (2017). Kodmallen kan liknas vid pattern (Gamma, Helm, Johnson och Vlissides, 1995) och schema/plan (Robins m.fl., 2003; Lehman & Qian, 2017). Bilaga 3 innehåller kodmallen och visar exempel på dess användning.

4.4.12 D11: Framgångsfaktorer

**Motivering:** Framgångsfaktorerna är baserade på författarens mångåriga arbete som programmerare och egna erfarenheter kring lärande i programmering.


<table>
<thead>
<tr>
<th>Framgångsfaktor</th>
<th>Beskrivning</th>
</tr>
</thead>
<tbody>
<tr>
<td>praktisera</td>
<td>Läsa, skriva om och skapa nya program samt öva, öva och öva.</td>
</tr>
<tr>
<td>kopiera</td>
<td>Återanvända idéer, lösningar och program.</td>
</tr>
<tr>
<td>experimentera</td>
<td>Våga hitta på egna program och inte bara läsa övningsuppgifter.</td>
</tr>
<tr>
<td>repetera</td>
<td>Behärska grunderna ordentligt.</td>
</tr>
<tr>
<td>samarbeta</td>
<td>Hjälpa andra, arbeta tillsammans och fråga när man fastnar.</td>
</tr>
</tbody>
</table>

Tabell 5. Framgångsfaktorer som behandlas i utbildningsmaterialet.
4.5 Färdigställa materialet

Utifrån komponenterna i programmeringsundervisningen (se Tabell 1, sida 22) och designprinciperna (se Tabell 2, sida 23) kunde utbildningsmaterialet färdigställas med viss precision och säkerhet.

Detta avsnitt beskriver arbetet och är det fjärde steget i att utarbeta utbildningsmaterialet.

4.5.1 Övergripande undervisningssekvens

Ordningsföljden i utbildningsmaterialet liknar den som många läroböcker i grundläggande programmering använder (t.ex. Wentworth, Elkner, Downey & Meyers, 2012) och designprinciperna realiseras i utbildningsmaterialet i följande ordning:

1. Ett enkelt program i Python 3 (D5: Programspråk), hur det fungerar och några enklare fel som kan inträffa när det skrivs.
2. Utvecklingsmiljön IDLE (D6: Utvecklingsmiljö) och hur den används för att skriva, ändra, starta, och avbryta (D7: Färdigheter) program.
3. Byggblocken (D8: Byggblock) ett i taget och hur de ser ut, fungerar och kan användas i program.
4. Hur matematiska problem kan lösas med programmering genom att välja och kombinera byggblocken på olika sätt i program.
5. En utvecklingsmetod (D9: Utvecklingsmetod) som gör det enklare, och snabbare, att skriva mer avancerade program.
7. Några enkla och effektiva tillvägagångssätt för att testa och felsöka program.

Svårighetsnivån på materialet ökar successivt, där steg 4 till 7 är områden med högre komplexitet, vilket rekommenderas av Robins m.fl. (2003). Varje steg i sekvensen inbegriper lösning av många övningsuppgifter.

4.5.2 Större moment

Tabell 6 visar större moment i arbetet med att färdigställa utbildningsmaterialet utifrån den övergripande undervisningssekvensen, komponenterna och designprinciperna. Ordningen mellan momenten är inte rättvisande, då materialet till stora delar skrivits iterativt; avsnitt har tagits bort, lagts till och skrivits om, omdisponerats och rättats massor av gånger i en process som är svår att beskriva i detalj. Vägledande principer har varit bland annat att skriva det enkla och färdigtänkta först, fokusera på texterna och förbättra den grafisk utformningen senare.
Tabell 6. Större moment i arbetet med att färdigställa utbildningsmaterialet.

4.5.3 Kopplingar
Tabell 7 visar ungefär var designprinciperna har realiserats i projektorpresentationen och elevhäftet.

<table>
<thead>
<tr>
<th>Designprinciper</th>
<th>Sidor i projektorpresentationen</th>
<th>Sidor i elevhäftet</th>
</tr>
</thead>
<tbody>
<tr>
<td>D11: Framgångsfaktorer</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>D5: Programspråk</td>
<td>7-12</td>
<td>2-6, 29-32</td>
</tr>
<tr>
<td>D6: Utvecklingsmiljö</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D7: Färdigheter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D8: Byggblock</td>
<td>13-28</td>
<td>6-17</td>
</tr>
<tr>
<td>D9: Utvecklingsmetod</td>
<td>29-34</td>
<td>17-21</td>
</tr>
<tr>
<td>D10: Kodmall</td>
<td>35-38</td>
<td>21-28</td>
</tr>
</tbody>
</table>

Tabell 7 Kopplingar mellan designprinciper, projektorpresentationen och elevhäftet. Utelämnade designprinciper har ingen tydlig koppling till specifika sidor.

4.6 Prova materialet

För att få återkoppling, väcka nya idéer samt hitta fel, brister och möjliga förbättringar i utbildningsmaterialet genomfördes tre mindre och osystematiska pilotförsök utan ambition att efterlikna verklighetsnära förhållanden i Ma1c. Inte i något pilotförsök räckte tiden till för att prova hela materialet och någon mätning av deltagarnas kunskaper, eller deras upplevelser av
programmeringsundervisningen, gjordes inte. Kapitel 6.4.3 beskriver anledningen till de osystematiska pilotförsöken och kapitel 3.3 deltagarurvalet.

Undervisningen i pilotförsöken styrdes i hög grad av projektorpresentationen, dess undervisningsssekvens och innehåll. Den kompletterades vid behov med genomgångar på whiteboard eller papper. Varje pilotförsök resulterade i en omarbetad version av materialet och gav en indikation på hur materialet fungerade i programmeringsundervisningen samt vilken undervisningstid som krävdes.

Detta avsnitt beskriver pilotförsöken och är det femte och sista steget i att utarbeta utbildningsmaterialet.

4.6.1 Pilotförsök 1: Programmeringskurs för elever på NT-programmen

4 elever i årskurs 1 anmälde sig till en frivillig och grundläggande programmeringskurs som författaren anordnade. Vi träffades tre helger och programmerade i totalt cirka 12 timmar. En av eleverna hoppade av på första träffen, men de övriga visade stort intresse och ville fortsätta programera efter kursens slut. En elev använder nu programmering för att lösa ett matematiskt problem i sitt gymnasiearbete.

Författarens bedömning är att eleverna som genomförde kursen är mer motiverade och bättre i matematik än genomsnittet. Ändå hade eleverna svårigheter att förstå och tillämpa begrepp som relativ frekvens, experimentell sannolikhet och slumpförsök i flera steg trots att de hade läst sannolikhetsläran i Ma1c några månader tadigare. Alla elever var nybörjare, följde med bra i programmeringsundervisningen och arbetade i huvudsak ensam.

4.6.2 Pilotförsök 2: Studiedag för gymnasielärare i matematik

10 matematiklärare, varav hälften inte undervisar på NT-programmen, hade studiedag. Cirka 5 timmar av dagen användes till en grundläggande programmeringskurs under författarens ledning. Ett fåtal lärare på yrkesprogram var lite negativa, ointresserade och hade svårt att förstå varför de skulle lära sig programmering.

Vissa lärare kunde redan programmera, andra hade läst programmering på högskolan för många år sedan och några var nybörjare. Majoriteten följde med bra i undervisningen och valde att arbeta tillsammans med en kollega.

4.6.3 Pilotförsök 3: Workshop för lärarstudenter

16 studenter deltog i en workshop på cirka 7 timmar om grundläggande programmering som MA1 anordnade. Många av studenterna kunde redan programmera, och vissa hade t.o.m. arbetat som programmerare i flera år, så de avvek direkt eller inom några timmar.

Det märktes snabbt att studenterna som stannade var nyfikna, intresserade och lärde sig snabbt vilket gjorde att studietakten blev högre än i övriga pilotförsök. Även här varierade förkunskaperna stort och de flesta valde att arbeta tillsammans med en eller flera kamrater.
5 Resultat

Studiens övergripande syfte är enligt kapitel 2.2 att utarbeta ett specialanpassat utbildningsmaterial avsett för programmeringsundervisning i Ma1c. Frågeställningen handlar om vilka teoretiska designprinciper som kan användas för att utarbeta materialet och dess användning i några mindre och osystematiska pilotförsök. Resultatet presenteras nedan.

5.1 Specialanpassat utbildningsmaterial

Det specialanpassade utbildningsmaterialet visas i bilaga 5 och 6.

5.2 Teoretiska designprinciper

De teoretiska designprinciperna sammanfattas i Tabell 2 på sida 23.

5.3 Erfarenheter från pilotförsöken

5.3.1 Programmeringsundervisningen

5.3.1.1 Varierande undervisningsresultat

Studietakten i pilotförsöken varierade, deltagarna hann olika långt och all a lärde sig programmera i någon utsträckning. Spridningen var dock mycket stor och beror gissningsvis på skillnader i bland annat förkunskaper och intresse. Påståendena är osäkra eftersom pilotförsöken är osystematiska (se kapitel 6.4.3).

5.3.1.2 Ingen löste [FY]

Ingen av pilotförsöken deltagare hann, eller ens försökte, lösa [FY]. De som kom längst hann lösa ett fåtal andra sannolikhetsproblem med lägre svårighetsnivå. Studien kan inte belägga påståendet med säkerhet, men mycket pekar på att programmeringsundervisningen som behövs för att eleverna i Ma1c ska kunna skriva ett program som löser [FY] inte kommer att hinnas med i kursen. Förutom erfarenheterna från pilotförsöken stöds påståendet av att undervisningstiden inte kommer att räcka till för det omfattande utbildningsmaterialet (se kapitel 5.3.2.2) och den långa undervisningssekvensen (se kapitel 4.5.1). Hur många timmar som kan avsättas för programmeringsundervisningen bestäms av respektive matematiklärare. Studien utgår från 9 timmar, vilket motsvarar ca 10% av den genomsnittliga totala undervisningstiden i Ma1c enligt Lärarnas Riksförbund (2013), vilket är otillräckligt.

Det finns därför goda skäl att hävda, givet studiens tolkning av [SS1-9] och 9 timmars undervisningstid, att eleverna inte kommer att lära sig matematisk problemlösning med programmering i Ma1c.

5.3.2 Utbildningsmaterialet

5.3.2.1 Inte färdigt

Både projektorpresentationen (se bilaga 5) och elevhäfte med övningsuppgifter (se bilaga 6) har brister och behöver rättas, omarbetas samt förbättras generellt (se kapitel 6.3.2). Alla
delar i utbildnings materialet är inte heller jämkade med rapporten, då viss terminologi behövde förändras och nya insikter gjordes under rapportskrivandet.

5.3.2.2 För omfattande
Trots att ambitionen är att utbildningsmaterialet ska vara minimalt består det av ca 75 sidor. Det går kanske att göra mindre (se kapitel 6.3.2), men sidantalet ger ändå en tydlig indikation på vilken omfattning som krävs för att undervisa i matematisk problemlösning med programmering i Ma1c givet studiens tolkning av [SS1-9]. Det finns även risk att materialet kommer att växa: Fler brister, nödvändiga kompletteringar och önskvärda förbättringar kommer sannolikt att upptäckas när materialet provas i fler försök.

5.3.2.3 Utmanande designprinciper
De delar av utbildningsmaterialet som provades i pilotförsöken fungerade väl. Tabell 8 visar de designprinciper som inte provades, behöver realiseras på ett bättre sätt i materialet eller var speciellt utmanande att undervisa.

<table>
<thead>
<tr>
<th>Designprincip</th>
<th>Kommentar</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2: Delar</td>
<td>Undervisningen med projektor och elevhäftets omfattning kan ifrågasättas. Gissningsvis är fler gemensamma genomgångar på whiteboard att föredra och elevhäftet bör delas upp i mindre, mer specialiserade och praktiskt användbara delar. Deltagarna använde bara en liten del av elevhäftet; främst övningsuppgifterna.</td>
</tr>
<tr>
<td>D7: Färdigheter</td>
<td>Mer fokus på framförallt felsökning, men även testning, tidigare i utbildningsmaterialet. Även analysen behöver integreras på ett bättre sätt. I synnerhet färdigheterna analysera, testa och felsöka är speciellt utmanade att undervisa.</td>
</tr>
<tr>
<td>D9: Utvecklingsmetod</td>
<td>Avsnittet provades inte i pilotförsöken.</td>
</tr>
<tr>
<td>D10: Kodmall</td>
<td>Avsnittet provades inte i pilotförsöken.</td>
</tr>
<tr>
<td>D11: Framgångsfaktorer</td>
<td>Materialet behöver fokusera mer på att få eleverna att experimentera, själva repetera bristande kunskaper och samarbeta. Denna designprincip är också speciellt utmanande att undervisa.</td>
</tr>
</tbody>
</table>

Tabell 8. Designprinciper som inte provades, behöver realiseras på ett bättre sätt i utbildningsmaterialet eller var speciellt utmanande att undervisa.
6 Diskussion

6.1 Studiens tillförlitlighet


Faktorer som har stor påverkan på resultatet inkluderar: (a) tolkningen av [SS1-9], (b) formuleringen av de teoretiska designprinciperna och deras användning för att utarbeta utbildningsmaterialet, (b) tillgänglig undervisningstid i pilotförsöken och deltagarnas förkunskaper, motivation och antal samt (d) författarens preferenser, kunskaper och erfarenheter i programmering, pedagogik och undervisning. Inslaget av subjektivitet, gissningar och tillfälligheter är stort.

6.2 Studiens forskningsmetod

Den valda forskningsmetoden Development Studies (se kapitel 3.1) medgav ett naturligt arbetssätt, gav ett bra stöd i studien samt har den tilltalande målsättning att både bidra till forskningen och lösa praktiska problem i lärarens vardag.

Ramverket Curricular Spider’s Web (se kapitel 3.2) gav också ett bra stöd, men visade sig ha vissa brister när det användes i studien. Till exempel hade ett mer fullständigt, detaljerat och tydligare ramverk med konkreta checklistor varit värdefullt för att bestämma programmeringsundervisningens innehåll och omfattning.


6.3 Möjliga förbättringar

6.3.1 Programmering i Ma1c

För att hinna med en givande och meningsfull programmeringsundervisning i Ma1c kan en kombination av följande beaktas: (a) göra en annan och lindrigare tolkning av [SS1-9] (se kapitel 4.2), (b) avsätta mer än 9 timmar för programmeringsundervisningen (se kapitel 5.3.1.2) samt (c) arbeta för en omformulering av [SS1-9] och sänka Skolverkets målsättning och ambitionsnivån till ”få se och prova på lite enklare matematisk problemlösning med programmering” istället för ”lösa matematiska problem med programmering” i [SS1].

6.3.2 Utbildningsmaterialet

Förutom att färdigställas behöver utbildningsmaterialet även förbättras på andra sätt. Helt färdigt eller felfritt blir materialet aldrig, men några idéer är (a) att minska omfattningen och
fokusera än mer på det som eleverna använder för att lösa övningsuppgifterna, (b) utöka avsnitten om felsökning och testning; speciellt i början av materialet, (c) dela upp, omarbeta och förenkla elevhäftet genom att till exempel skapa ett referensblad för byggblocken, placera övningsuppgifterna i ett separat dokument och radera det som även projektorpresentationen behandlar, (d) komponera fler och bättre övningsuppgifter som till exempel handlar om att förstå, ändra och rätta exempel på välskrivna program, (e) minska projektorpresentationens omfattning kraftigt och öka undervisningen på whiteboard samt (f) förbättra materialets grafiska utformning.

6.3.3 Programmeringsundervisningen
Även programmeringsundervisningen behöver förbättras. Några idéer är att integrera (a) en notional machine (Robins m.fl., 2003), (b) close tracking (Robins m.fl., 2003), (c) utvecklingsmiljön Thonny (www.thonny.org), vilken har en bättre debugger (t.ex. Adams m.fl., 2007; Lehman & Qian, 2017) än IDLE, samt (d) en förenklad variant av parprogrammering (Hanks, McDowell & Werner, 2003).

6.4 Brister i studien
Detta avsnitt tar upp några brister i studien vilka insågs främst under rapportskrivandet. Om de hade åtgärdats hade studien blivit mer avgränsad, balanserad, trovärdig och användbar.

6.4.1 Spretighet
Studien har växt successivt, trots att ett antal begränsningar har gjorts sedan starten, och borde ha delats upp i flera separata studier. Vid en uppdelning hade varje separat studie blivit mer avgränsad, detaljerad och fördjupad.

6.4.2 Dålig förankring i forskning
Trovärdigheten hade ökat om fler relevanta vetenskapliga avhandlingar, rapporter, artiklar och annan typ av litteratur hade kunnat hittas och kopplas till studien. I synnerhet saknas tillräckligt bra referenser till forskning om grundläggande programmeringsundervisning och utbildningsmaterial avsedda för den typen av undervisning. Bristen är speciellt påtaglig i litteraturöversikten i kapitel 2.1.

6.4.3 Osystematiska pilotförsök
Mer genomtänkta och bättre genomförda pilotförsök; till exempel ett bättre urval av deltagare (se kapitel 3.3), mer undervisningstid (se kapitel 5.3.1.2) och systematisk utvärdering av undervisningens resultat och effekter hade bidragit med fler och värdefullare erfarenheter från programmeringsundervisningen och användningen av utbildningsmaterialet.

Anledningen till att pilotförsöken inte genomfördes systematiskt är att det i början av studien inte fanns någon plan att ta vara på erfarenheter från pilotförsöken. Tanken var att erfarenheterna skulle dokumenteras först efter pilotförsöken i ett efterföljande försök med en strikt kontrollerad undervisningssekvens. Det var först efter pilotförsöken, när studien
begränsades och dess inriktning ändrades på grund av tidsbrist, som de odokumenterade erfarenheterna från pilotförsöken blev intressanta och skulle användas i rapporten.

6.4.4 Fåtal perspektiv

Studien hade blivit mer balanserad med en eller flera medförfattare vilka hade kunnat bidra med idéer, vidare resonemang, fler motargument och andra perspektiv. Det är sällan bra att arbeta ensam med omfattande och komplexa uppgifter.

6.5 Personliga reflektioner

Detta avsnitt redogör för några av de funderingar och synpunkter som arbetet med studien har väckt hos författaren. Funderingarna är subjektiva, lite spekulativa och löst kopplade till studiens övergripande syfte och forskningsfråga. Avsnittets syfte är att väcka idéer och tankar, ge nya perspektiv och kanske inspirera till ny forskning.

6.5.1 Programmering på NT-programmen

Matematik och programmering, men även fysik, är nog en bra kombination i Ma4, Ma5 och gymnasiarbetet på NT-programmen. Kopplingarna mellan ämnena, likheterna i tankesätt, nödvändiga förmågor samt nytan sinsemellan är troligen stora (t.ex. Heintz, Färnqvist & Thorén, 2015; Sandell, 2017). I senare kurser har eleverna bättre förkunskaper, är mer mogna och intresserade, vilket ökar möjligheterna att hinna med en mer givande och meningsfull programmeringsundervisning än i Ma1c. Att undervisningen ska ske i ämnet matematik, eller vara obligatoriskt, är inte lika säkert.

6.5.2 Regeringens beslut

Det är lätt att få uppfattningen att beslutet att skriva in programmering i kursplanerna för matematik (se kapitel 1.3) är förhastat och dåligt underbyggt. Många var kritiska (se kapitel 1.4), remisstiden var bara ca 4 veckor (Skolverket, 2016b) och den konsekvensanalys som Westerholm (2018) refererar till i en debattartikel har inte kunnat hittas:

"…det är många viktiga saker som eleverna ska lära sig i ämnet matematik i gymnasieskolan. Denna fråga behöver ständig uppmärksammas. Vi gör alltid en konsekvensanalys vid ändringar i styrdokumenten. Innehållet i den analysen kan de som önskar ta del av i samband med remissförfarande som alltid sker samt i redovisningen till regeringen."

Vad Skolverkets konsekvensanalys omfattar är därför oklart för författaren. En minimal konsekvensanalys bör enligt författarens mening inkludera att utreda, och tydligt kommunicera, följande: (a) Vilken undervisningstid och andra resurser behövs för att elever ska kunna lösa matematiska problem med programmering, (b) vilka positiva och negativa effekter på elevernas matematikkunskaper kan förvántas samt (c) vilka krav ställer programmeringsundervisningen på matematiklärare?
6.5.3 Skolverkets underskattning

Det går att ana en underskattning från Skolverkets sida när det gäller utmaningen för elever att lära sig matematisk problemlösning med programmering. Bara att lära sig grundläggande programmering är svårt (se kapitel 2.1). Götling och Löfwenhamn (2018, s. 12) konstaterar försiktigt:

"Detta, i sig, vittnar om en viss trivialisering från Skolverket gällande utmaningarna relaterade till syntax och konceptförståelse från såväl lärares håll som från elever.”

Några exempel på underskattning kan ses i Skolverket (2018b) som avhandlar en komplex rekursiv funktion, listoperatorer och ett 30 rader långt icke-trivialt program; allt ovanliga och olämpliga i nybörjarsammanhang enligt författarens erfarenhet. Att dessutom tillämpa sina kunskaper i programmering, och matematik, för att lösa matematiska problem är ännu svårare. Robins m.fl. (2003, s.156) beskrivning av kedjan eleverna behöver behärskar visar på svårigheterna:

"This chain starts with the features of the language being taught. The second link is design skills, including templates (schemas/plans), and the procedural skills of planning, testing and reformulating code. The third link is problem-solving skills, knowledge and strategies (including the use of the procedural skills) abstracted from the specific language taught that can be applied to new languages and situations.”

Ytterligare exempel på underskattning är Skolverkets svävande och motsägelsefulla skrivningar [SS6-8], vilka kan tolkas som att eleverna ska kunna lösa matematiska problem med programmering (se [SS6]) utan att kunna programmera (se [SS7]) eller få tid att lära sig det i tillräcklig omfattning (se [SS6-8]).

Jämför med om lärare i kursen Fysik 1 skulle undervisa elever med enbart rudimentära kunskaper i matematik utifrån följande påhittade skrivning: Åmnesplanernas skrivningar innebär att matematik ska användas till att lösa fysikaliska problem, inte utbilda eleverna i matematik. Den tillspetsade formuleringen utgår från [SS8] där matematik och programmering är utbytta mot fysik och matematik.

6.5.4 Potentiella samordningsvinster

Det finns goda möjligheter att spara tid och pengar samt underlätta implementeringen av programmeringsundervisning i ämnet matematik genom bättre samordning. Om Skolverket till exempel bestämmer att alla ska använda samma programspråk och utvecklingsmiljö är det enklare att utarbeta ett gemensamt utbildningsmaterial, spreda lämpliga undervisningsmetoder samt samarbeta för lärarutbildningar, skolor och matematiklärare.
6.6 Förslag på fortsatt forskning

Detta avsnitt presenterar några utvalda idéer på ytterligare forskning som skulle fördjupa och komplettera studien. De har sitt ursprung från delar av studien som blev nedprioriterade av tidsbrist eller hamnade utanför studiens begränsningar.

6.6.1 Granska designprinciperna

De teoretiska designprinciperna (se Tabell 2, sida 23) behöver skärskådas. Är de (a) heltäckande eller saknas några, (b) relevanta eller är några onödiga samt (c) entydiga eller behöver några omformuleras? Idealt ska tabellen innehålla de viktigaste och effektivaste designprinciperna enligt den senaste forskningen. Ett komplement till forskningen kan vara att jämföra erkända och beprövade utbildningsmaterial i grundläggande programmering oavsett programspråk. Då borde likheter, skillnader och andra återkommande mönster i bra utbildningsmaterial framträda: till exempel vad som lärs ut, hur och i vilken ordning. Genom att generalisera och beskriva dessa återkommande mönster bör designprinciperna kunna utvärderas, förbättras och utgöra underlag för nya och bättre utbildningsmaterial.

6.6.2 Utarbeta alternativa utbildningsmaterial

Det går att utarbeta nästan hur många utbildningsmaterial som helst utifrån Skolverkets skrivningar [SS1-9], komponenterna i programmeringsundervisningen (se Tabell 1, sida 22) och de teoretiska designprinciperna (se Tabell 2, sida 23). Att göra andra tolkningar av skrivningarna, modifiera komponenter, prova andra designprinciper och undersöka vad det leder till för material är angeläget och skulle belysa programmeringsundervisningen i Ma1c ytterligare.

6.6.3 Utvärdera utbildningsmaterialet i systematiska försök

Pilotförsöken i kapitel 4.6 är otillräckliga och utbildningsmaterialet, men även alternativa material, behöver studeras i fler och mer verklighetsnära försök. Nya studier kan till exempel undersöka och mäta undervisningstiden, programmeringsundervisningens resultat (med ett prov) samt elevernas åsikter och upplevelser (med en enkät). Systematiska försök i Ma1c, eller så verklighetsnära som möjligt, är oersättliga för att samla data, få mer erfarenhet och öka kunskapen om både undervisningen och materialet.
7 Referenser


8 Bilagor

Programmering kräver ibland goda matematikkunskaper. Bilagorna förutsätter att läsaren är insatt i gymnasienformatik i allmänhet och sannolikhetsläran i Ma1c i synnerhet till exempel begreppen utfall, händelse, komplementinhändelse, relativ frekvens och slumpförsök i flera steg. Även vissa programmeringskunskaper förutsätts.

8.1 Bilaga 1: Exempel på lösningar av det förenklade Yatzyproblemet

8.1.1 Ett program i Python 3

Det finns många olika program som löser [FY] och det är omöjligt att bedöma, rangordna och objektivt komma fram till vilket som är det bästa. Figur 5 visar ett förslag som uppskattar P(Yatzy minst 1 gång) genom att nytta kodmallen (Figur 11, sida 47), simulera slumpförsöket många gånger samt beräkna den relativa frekvensen för händelsen. Figuren inkluderar ett flödesschema och pseudokod för att underlätta förståelsen för hur programmet fungerar och är uppbyggt. Grafiska och alternativa sätt att åskådliggöra program är något som rekommenderas i forskning (t.ex. Adams m.fl., 2007; Robins m.fl., 2003).

Flödesschema

Start

År n < 10^6?

nej

ja

Simulera att 5 tärningar kastas 3 gånger

Yatzy minst 1 gång?

nej

ja

h = h + 1

n = n + 1

Skriv ut f

Stopp

Python 3

```python
from random import choice
h = 0
n = 0
while n < 100000:
    antalYatzy = 0
    antalKast = 0
    while antalKast < 3:
        ögon1 = choice(range(6)) + 1
        ögon2 = choice(range(6)) + 1
        ögon3 = choice(range(6)) + 1
        ögon4 = choice(range(6)) + 1
        ögon5 = choice(range(6)) + 1
        if ögon1==ögon2==ögon3==ögon4==ögon5:
            antalYatzy = antalYatzy + 1
        antalKast = antalKast + 1
    if antalYatzy > 0:
        h = h + 1
    n = n + 1
f = h/n
print("P(Yatzy minst 1 gång): ca", f)
```

Pseudokod

Reperera många gånger:

Simulera att 5 tärningar kastas 3 gånger.

Öka en räknare om kasten gav Yatzy minst 1 gång.

Uppskatta P(Yatzy minst 1 gång) genom att beräkna och visa den relativa frekvensen på skärmen.

Figur 5: Ett program i Python 3 som löser [FY] genom att simulera ”Du kastar 5 tärningar 3 gånger och får inte spara någon tärning mellan kasten.” och uppskatta P(Yatzy minst 1 gång).
Figur 6: Ett exempel på vad som visas på datorns skärm när programmet ovan har avslutats.

8.1.2 Analytiskt

Den analytiska lösningen av [FY] är ca 0,0023 och beräknas genom att använda komplementhändelsen $P(\text{Ingen Yatzy})$ längst till höger i träddiagrammet nedan.

$$P(\text{Yatzy minst 1 gång}) = 1 - P(\text{Ingen Yatzy}) = 1 - [1 - P(\text{Yatzy})]^3,$$

där $P(\text{Yatzy}) = P(\text{alla tärningar visar samma antal ögon}) = 6^5[1/6]^5$

Figur 7: Träddiagrammets nedersta rad representerar alla utfall i slumpförsöket "Du kastar 5 tärningar 3 gånger och får inte spara någon tärning mellan kasten". Varje förgrening representerar att 5 tärningar kastas med början överst. En svart nod symboliserar att kastet gav Yatzy och en vit övriga kombinationer.

8.1.3 Ett program i C

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main()
{
    int h, n, antalYatzy, antalKast, ogon1, ogon2, ogon3, ogon4, ogon5;
    float f;
    srand(time(NULL));
    h = 0;
    n = 0;
    while(n < 1000000) {
        antalYatzy = 0;
        antalKast = 0;
        while(antalKast < 3) {
            ogon1 = rand()%6 + 1;
            ogon2 = rand()%6 + 1;
            ogon3 = rand()%6 + 1;
            ogon4 = rand()%6 + 1;
            ogon5 = rand()%6 + 1;
            if ((ogon1 == ogon2) && (ogon2 == ogon3) && (ogon3 == ogon4) && (ogon4 == ogon5)) {
                antalYatzy = antalYatzy + 1;
            }
        }
        antalKast = antalKast + 1;
        h = h + 1;
    }
    return 0;
}
```
antalKast = antalKast + 1;
}
if (antalYatzy > 0) {
    h = h + 1;
}
    n = n + 1;
}    f = (float)h/n;
printf("P(Yatzy minst 1 gång): ca %f", f);
}

Figur 8: Programmet i Figur 5 på sida 43 skriven i programspråket C istället för Python 3.

8.2 Bilaga 2: Exempel på lösningar av Skolverkets problem

Nedanstående program löser Hur vanligt är det att en 50/50-sannolikhet faller ut 60/40 eller mer skevt vid 100 observationer? i [SS2].

```python
from random import choice
h = 0    n = 0
while n < 100000:
    antalEttor = 0
    antalObservationer = 0
    while antalObservationer < 100:
        slumptal = choice(range(2)) + 1
        if slumptal == 1:
            antalEttor = antalEttor + 1
            antalObservationer = antalObservationer + 1
        if not (40 < antalEttor < 60):
            h = h + 1
            n = n + 1
    f = h/n
print("P(skevare än 60/40 vid 100 observationer): ca", f)
```

Figur 9: Ett exempel på vad som visas på datorns skärm när programmen ovan har avslutats.

Nedanstående program löser Vad är chansen att slå Yatzy på högst 3 slag? i [SS2] och följer spelreglerna i Yatzy. Problemet kan förtydligas: Du har 5 tärningar, får kasta 3 gånger och spara tärningar mellan kasten. Din strategi är att efter varje kast gruppera tärningarna utifrån antalet ögon de visar, spara den grupp som innehåller flest tärningar och kasta om övriga. Vad är sannolikheten att få Yatzy?

```python
from random import choice
antalYatzy = 0
for antalFörsök in range(1, 100000):
```
ögonRäknare = [0, 0, 0, 0, 0, 0]
for antalKast in range(0, 3):
    # Kasta ej sparade tärningar (första gången 5 stycken).
    for tärning in range(0, 5 - max(ögonRäknare)):
        ögon = choice(range(6))
        ögonRäknare[ögon] += 1
    # Spara lämpliga tärningar, d.v.s. flest med lika antal ögon.
    for ögon in range(0, 6):
        if ögon != ögonRäknare.index(max(ögonRäknare)):
            ögonRäknare[ögon] = 0
    # Resulterade de 3 kasten i Yatzy?
    if max(ögonRäknare) == 5:
        antalYatzy += 1
print("P(Yatzy): ca", antalYatzy/antalFörsök)

Figur 10: Ett exempel på vad som visas på datorns skärm när programmen ovan har avslutats.

Programmet är svårare att skriva, jämfört med programmet som löser [FY] i Figur 5 på sida 43, beroende på att tärningar får sparas mellan kasten. Att få programmet att spara rätt grupp av tärningar, och bara kasta om övriga, är lite klurigt: Det krävs 3 nästlade loopar samt några byggblock och instruktioner vilka inte ingår i utbildningsmaterialet; till exempel listan [0,0,0,0,0,0], listfunktionerna index och range, loopinstruktionen for och den matematiska funktionen max. Programmet kan skrivas enbart med byggblocken som ingår i utbildningsmaterialet, men det skulle bli onödigt komplicerat och svårare att förstå.
8.3 Bilaga 3: Exempel på kodmall för slumpförsök

Figur 11: En kodmall för att skriva program som löser slumpförsök i ett eller flera steg. Mallen bygger på experimentell sannolikhet och uppskattar P(H) genom att simulera ett slumpförsök många gånger, räkna antalet gånger händelsen H inträffar och beräkna den relativa frekvensen för H på skärmen.

```python
from random import choice
h = 0
n = 0
while n < 1000000:
    # Simulera slumpförsöket en gång
    if choice(range(2)) + 1:
        h = h + 1
    n = n + 1
f = h/n
print("Uppskattad P(H) =", f)
```


```python
from random import choice
h = 0
n = 0
while n < 1000000:
    ögon1 = choice(range(2)) + 1
    ögon2 = choice(range(2)) + 1
    ögon3 = choice(range(2)) + 1
    if ögon1 + ögon2 + ögon3 == 8:
        h = h + 1
    n = n + 1
f = h/n
print("Uppskattad P(H) =", f)
```
8.4 Bilaga 4: Förslag på utvecklingsmetoder

Utvecklingsmetoden exemplifieras med programmet som löser [FY] i Figur 5 på sida 43.

**Steg 1: Simulerar att 1 tärning kastas**


Avsluta med att radera testkoden när testerna är gjorda och programmet fungerar som tänkt.

```python
from random import choice
ögon1 = choice(range(6)) + 1
print(ögon1)
```

**Steg 2: Simulerar att 5 tärningar kastas**

Lägg till de nya raderna i programmet och raden med temporär testkod. Starta programmet några gånger, kontrollera att skärmen efter varje gång visar fem tal mellan 1 och 6 och avsluta med att radera testkoden när programmet verkar fungera som tänkt. Varje tal representerar antalet ögon som en av tärningarna visar efter kastet.

```python
from random import choice
ögon1 = choice(range(6)) + 1
ögon2 = choice(range(6)) + 1
ögon3 = choice(range(6)) + 1
ögon4 = choice(range(6)) + 1
ögon5 = choice(range(6)) + 1
print(ögon1, ögon2, ögon3, ögon4, ögon5)
```

**Steg 3: Simulera att tärningarna kastas 3 gånger**

Fortsätt på samma sätt som tidigare, men kontrollera att skärmen visar fem tal mellan 1 och 6 på tre rader. Varje rad representerar ett kast med tärningarna.

Indenteringen under while styr vilka rader som repeteras 3 gånger.

```python
from random import choice
ögon1 = choice(range(6)) + 1
ögon2 = choice(range(6)) + 1
ögon3 = choice(range(6)) + 1
ögon4 = choice(range(6)) + 1
ögon5 = choice(range(6)) + 1

while True:
    print(ögon1, ögon2, ögon3, ögon4, ögon5)
    Ögon1 = choice(range(6)) + 1
    Ögon2 = choice(range(6)) + 1
    Ögon3 = choice(range(6)) + 1
    Ögon4 = choice(range(6)) + 1
    Ögon5 = choice(range(6)) + 1
    print(ögon1, ögon2, ögon3, ögon4, ögon5)
```

**Figur 13:** Ett exempel på vad som visas på datorns skärm när programmen ovan har avslutats.
from random import choice
antalKast = 0
while antalKast < 3:
    ögon1 = choice(range(6)) + 1
    ögon2 = choice(range(6)) + 1
    ögon3 = choice(range(6)) + 1
    ögon4 = choice(range(6)) + 1
    ögon5 = choice(range(6)) + 1
    print(ögon1, ögon2, ögon3, ögon4, ögon5)
    antalKast = antalKast + 1

Steg 4: Simulera många kast och räkna antal gånger tärningarna visar Yatzy
    Fortsätt på samma sätt, men kontrollera istället att skärmén då och då visar texten ”Yatzy”.
    Texten representerar att ett kast gav Yatzy.
    Indenteringen under if styr vilka rader som utförs om villkoret är sant.

from random import choice
n = 0
while n < 1000000:
    antalYatzy = 0
    antalKast = 0
    while antalKast < 3:
        ögon1 = choice(range(6)) + 1
        ögon2 = choice(range(6)) + 1
        ögon3 = choice(range(6)) + 1
        ögon4 = choice(range(6)) + 1
        ögon5 = choice(range(6)) + 1
        if ögon1 == ögon2 == ögon3 == ögon4 == ögon5:
            antalYatzy = antalYatzy + 1
            print("Yatzy")
    antalKast = antalKast + 1
    n = n + 1

Steg 5: Beräkna och visa relativa frekvensen för händelsen ”Yatzy minst 1 gång”
    Fortsätt på samma sätt, men kontrollera istället att skärmén visar en uppskattning på
    sannolikheten P(Yatzy minst 1 gång). En bra test är att jämföra uppskattningen med den
    analytiskt beräknade sannolikheten i bilaga 1.
    Notera att kodmallen för slumpförsök (Figur 11, sida 47) dyker upp i det färdiga
    programmet.

from random import choice
h = 0
n = 0
while n < 1000000:
    antalYatzy = 0
    antalKast = 0
    while antalKast < 3:
        ögon1 = choice(range(6)) + 1
        ögon2 = choice(range(6)) + 1
        ögon3 = choice(range(6)) + 1
        ögon4 = choice(range(6)) + 1
        ögon5 = choice(range(6)) + 1
        if ögon1 == ögon2 == ögon3 == ögon4 == ögon5:
antalYatzy = antalYatzy + 1
antalKast = antalKast + 1
if antalYatzy > 0:
    h = h + 1
n = n + 1
f = h/n
print("P(Yatzy minst 1 gång): ca", f)

En alternativ utvecklingsmetod, vilken kräver mer programmeringserfarenhet och bygger på kopiering, exempliferas också med programmet som löser [FY].

**Steg 1: Inse att problemet är ett slumpförsök**

Det första, viktigaste och kanske svåraste steget är att inse att [FY] är ett slumpförsök och att det blir enklare att skriva programmet genom att återanvända delar av andra program som löser närliggande sannolikhetsproblem alternativt utgå från kodmallen för slumpförsök (Figur 11, sida 47).

**Steg 2: Kopiera och utgå från ett program**

Hitta och kopiera ett tidigare skrivet, bekant och lättförståeligt program som är lämpligt att använda som startpunkt. Undersök till exempel tidigare skrivna program och utbildningsmaterialets exempel. Nedanstående kommer från Figur 12 på sida 47 och uppskattar sannolikheten för ”3 tärningar kastas. Vad är sannolikheten att summan av tärningarnas ögon är exakt 8?”.

```python
from random import choice
h = 0
n = 0
while n < 1000000:
    ögon1 = choice(range(6)) + 1
    ögon2 = choice(range(6)) + 1
    ögon3 = choice(range(6)) + 1
    if ögon1 + ögon2 + ögon3 == 8:
        h = h + 1
    n = n + 1
f = h/n
print("Uppskattad P(H) =", f)
```

**Steg 3: Ändra programmet så att det uppskattar sannolikheten för Yatzy**

Ändra först så att programmet simulerar 5 tärningar istället för 3 och noterar händelsen ”Yatzy” istället för ”summan av tärningarnas ögon är exakt 8”. Tillaga eller ändrade rader är markerade med fet stil. Testa programmet och kontrollera svaret genom att jämföra med den analytiska lösningen av ”Du kastar 5 tärningar. Vad är sannolikheten att få Yatzy?”.

```python
from random import choice
h = 0
n = 0
while n < 1000000:
    ögon1 = choice(range(6)) + 1
    ögon2 = choice(range(6)) + 1
    ögon3 = choice(range(6)) + 1
    ögon4 = choice(range(6)) + 1
    ögon5 = choice(range(6)) + 1
    if ögon1 + ögon2 + ögon3 + ögon4 + ögon5 == 6:
        h = h + 1
    n = n + 1
f = h/n
print("Uppskattad P(H) =", f)
```
ögon3 = choice(range(6)) + 1
ögon4 = choice(range(6)) + 1
ögon5 = choice(range(6)) + 1
if ögon1 == ögon2 == ögon3 == ögon4 == ögon5:
    h = h + 1
n = n + 1
f = h/n
print("Uppskattad P(H) =", f)

Steg 4: Ändra programmet så att det löser [FY]
Ändra slutligen så att programmet simulerar att tärningarna kastas 3 gånger istället för 1 gång och noterar händelsen ”Yatzy minst 1 gång” istället för ”Yatzy”.

from random import choice
h = 0
n = 0
while n < 1000000:
    antalYatzy = 0
    antalKast = 0
    while antalKast < 3:
        ögon1 = choice(range(6)) + 1
        ögon2 = choice(range(6)) + 1
        ögon3 = choice(range(6)) + 1
        ögon4 = choice(range(6)) + 1
        ögon5 = choice(range(6)) + 1
        if ögon1 == ögon2 == ögon3 == ögon4 == ögon5:
            antalYatzy = antalYatzy + 1
            antalKast = antalKast + 1
    if antalYatzy > 0:
        h = h + 1
        n = n + 1
        f = h/n
    print("Uppskattad P(H) =", f)

8.5 Bilaga 5: Projektorpresentation

Grundläggande programmering
i
Python 3
- lösning av sannolikhetsproblem i Ma1c
Introduktion: Om mig

• Johnny Åberg, är civilingenjör i Datateknik vid Chalmers Tekniska Högskola och har programmerat på fritiden, i skolan och arbetslivet sedan 1980.
• Har arbetat ca 25 år med inbyggda system inom telekom, fordons- och försvarsindustri; t.ex. basstationer, vägtullar och stridsflygplan.
• Arbetar extra som gymnasielärare i matematik och fysik på tekniska och naturvetenskapliga program i Nyköping sedan 2016.
• Arbetar på ett förberedande examensarbete “Lösning av sannolikhetsproblem i Ma1c med programmering” som Matematiska institutionen på Linköpings Universitet handleder.
• Målet är att gå KPU och bli behörig gymnasielärare i Matematik och Teknik om några, eller kanske troligare, många år.

Introduktion: God forskningssed

Omfattar bl.a. ett grundläggande individskydd:

• Informationskrav – alla deltagare (du och andra) ska veta studiens syfte, sina uppgifter i studien och villkoren för deltagandet.
• Samtyckeskrav – allt deltagande är frivilligt och kan avbrytas när som helst utan motivering eller påföljd.
• Konfidentialitetskrav – alla deltagare är anonyma, insamlat material skyddas mot obehöriga och tystnadsplikt tillämpas vid behov.
• Nytjandekrav – insamlat material får endast användas i studien och annan forskning.
• Framtagen av Vetenskapsrådet som är en statlig myndighet. Frågor?

Introduktion: Kursöversikt

• Introduktion (kursens avsnitt är fetmarkerade)
• Vad är ett program och vilka verktyg behövs för att programmera? Python 3
• Vad är program uppbyggda av? Byggblock
• Hur kan byggblocken kombineras och anpassas för att lösa häftets problem? Algoritmer
• Hur arbetar programmerare? Arbetsätt
• Hur kan närbesläktade problem lösas enklare? Typisk algoritm
• Avslutning
Introduktion: Lite studieteknik

• Det krävs tid, ansträngning, koncentration och engagemang för att lära sig programmera - speciellt i början.
• Om du upptäcker att du missat eller glömt något viktigt: fråga och repetera - att behärska grunderna är nödvändigt.
• Kursens fokus är praktik; du lär dig programmera genom att läsa och skriva många program – så öva, öva och öva!
• Våga experimentera, prova idéer, utforska och ändra dina program - kanske det bästa sättet att lära sig?
• Notera likheterna med att lära sig matematik.

Introduktion: Praktiska detaljer

• Kursen är uppdelad i avsnitt. Varje avsnitt består av:
  - genomgång och introduktion av avsnittet
  - läsa häftet och lösa övningsuppgifter
  - kortare fråge- och diskussionsstund
  - kortare rast
• Lunch ca 12:00 – 13:00.
• Kursen slutar senast klockan 17:00.
• Avbryt mig gärna, kommentera och ställ frågor.
• Några frågor, funderingar eller kommentarer?

Vad är ett program och vilka verktyg behövs för att programmera?

Python 3
Python 3: Ett exempel

• Syftet med programmering är att skriva ett program (tänk app) till en dator (tänk iPhone).
• Programmet bestämmer vad datorn ska göra; utan program är en dator helt oanvändbar.
• Ett program är en textfil uppdelad på rader som följer de strikta regler som specificeras i ett programspråk. Jag har valt Python 3.

```python
x = 1
while x <= 1000:
    y = x*x
    print(y)
    x = x + 1
```

Filén "program.py"

Python 3: En rad i taget

• När du har startat ett program läser, tolkar och utför datorn rad efter rad i filen; från den första raden till den sista då programmet avslutas. Vissa rader exekveras flera gånger.
• Programpekaren visar vilken rad datorn arbetar med.
• Använd ett finger och exekvera programmet manuellt rad för rad, t.ex. för att förstå eller felsöka ett program.

```python
x = 1
while x <= 1000:
    y = x*x
    print(y)
    x = x + 1
```

Filén "program.py"

Python 3: Program innehåller fel

• Program innehåller nästan alltid fel (tänk appars som kraschar eller inte svarar); många upptäcks aldrig eller för sent.
• Några enklare fel som är lätt att hitta är t.ex. felstavningar, otillåtna och missade tecken samt felaktiga indragningar.

```python
x = 1,
while x <= 1000:
    y = x*x
    print(y)
    x = x + 1
```

Program med 3 enklare fel.

• Svårare fel är t.ex. att programmet visar fel svar eller kraschar.
• Att upptäcka ett fel, hitta raden och rätta felet kan ta lång tid även för erfarna programmerare.
Python 3: Verktyget IDLE

- För att programmera behöver du en utvecklingsmiljö. Jag har valt IDLE.
- IDLE är en smart ordbehandlare för Python 3 med särskilda kommandon för att hantera program.

<table>
<thead>
<tr>
<th>IDLE</th>
<th>Word2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Skriv ett program&quot;</td>
<td>&quot;Skapa och skriv en text i ett dokument&quot;</td>
</tr>
<tr>
<td>&quot;Spara programmet&quot;</td>
<td>&quot;Spara dokumentet i en fil&quot;</td>
</tr>
<tr>
<td>&quot;Ändra programmet&quot;</td>
<td>&quot;Öppna filen och ändra texten i dokumentet&quot;</td>
</tr>
<tr>
<td>Hanterar program (*.py)</td>
<td>Hanterar dokument (*.docx)</td>
</tr>
<tr>
<td>Programmets text måste följa reglerna i Python 3</td>
<td>Texten får se ut hur som helst, men stavfel markeras</td>
</tr>
<tr>
<td>Programmets text kan tolkas av IDLE</td>
<td>Texten betyder inget för Word2016</td>
</tr>
<tr>
<td>&quot;Starta programmet&quot;</td>
<td>-</td>
</tr>
</tbody>
</table>

- Filer med program hanteras som vanligt i t.ex. Windows utforskaren.

Python 3: Eget arbete

- Nu är det dags att prova IDLE, skriva ditt första program samt öva på lite felsökning.
- Läs kapitel 1 "Utvecklingsmiljön IDLE" i häftet, följ instruktionerna och lös övningsuppgifterna tillsammans med en kamrat. Glöm inte att turas om att sitta vid tangentbordet.

Vad är program uppbyggda av?

**Byggblock**
Byggblock: Små, enkla och generella

• Ett program består av små, enkla och generella byggblock (tänk Lego), där varje byggblock har en specifik funktion.

\begin{verbatim}
x = 1 while x <= 1000:
    y = x*x
    print(y)
    x = x + 1:
\end{verbatim}

Programmet är uppbyggt av 5 byggblock, varav 3 är anpassningar av ”Beräkna matematiska uttryck”.

• Det är byggblocken, hur de kombineras och anpassas som bestämmer vad programmet gör.

• Varje problem i häftet kräver en viss kombination av byggblock och anpassning för att lösas – utmaningen är att komma på vilken.

2018-11-09
Johnny Åberg, Linköpings Universitet

Byggblock: Exempel ”Visa en dikt på skärmn”

• Programmet använder bara byggblocket ”Visa på skärmen” (markerat med rött).

\begin{verbatim}
print("Den mätta dagen, den är aldrig störst.
Den bästa dagen är en dag av törst.
Nog finns det mål och mening i vår färd -
men det är vägen, som är mödans värld.
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
"
Byggblock: Exempel ”Beräkna ett uttryck”

• Programmen använder 2 nya byggblock: ”Vänta på inmatning från tangentbordet” och ”Beräkna matematiska uttryck”.

```python
x = int(input("Ange ett heltal x? "))
y = x*x + 3*x + 8
print(y)
```

Ange ett heltal x? 1
2

• För att kunna använda många matematiska funktioner, t.ex. sin, asin, och log, måste du inkludera modulen math i dina program.

Byggblock: Exempel ”Singla slant”

• Programmet använder 2 nya byggblock: ”Generera slump tal” och ”Välj rader”.

```python
from random import choice
x = choice(range(2)) + 1
if x == 1:
    print("krona")
else:
    print("klave")
```

Exempel på logiska uttryck

• Det logiska uttryckets värde bestämmer vägen genom programmet: är det sant exekveras en rad och är det falskt exekveras en annan.

Byggblock: Exempel ”Multiplikationstabell”

• Programmet använder 1 nytt byggblock: ”Repetera rader”.

```python
x = 1
while x <= 9:
y = x*9
print(x,"\* 9 =", y)
x = x + 1
print("Klar!")
```

1 * 9 = 9
2 * 9 = 18
3 * 9 = 27
4 * 9 = 36
5 * 9 = 45
6 * 9 = 54
7 * 9 = 63
8 * 9 = 72
9 * 9 = 81
Klar!

• Indenteringen grupperar och bestämmer vilka rader som exekveras när det logiska uttrycket är sant respektive falskt.
### Byggblock: Exempel ”Singla slant flera gånger”

- Programmen använder 1 nytt byggblock: ”Använda kommentarer”

```python
# Programmet simulerar att en slant singlas flera gånger
from random import choice
n = 0  # Antal gånger slanten singlats
krona = 0  # Antal gånger "krona" hamnade uppåt
önskat = int(input("Ange antal gånger slanten ska singlas?"))
while n < önskat:
    x = choice(range(2)) + 1
    if x == 1:
        krona = krona + 1
        n = n + 1
print("Krona kom upp: ", krona, " gånger")
```

Ange antal gånger slanten ska singlas? 25
Krona kom upp: 11 gånger

### Byggblock: Eget arbete

- Först löser vi övningsuppgift 3 i kapitel 3.9 på tavlan.
- Nu är det dags att börja använda byggblocken, kombinera dem lite och skriva dina första enkla program.
- Läs kapitel 3 ”Utvalda byggblock i Python 3” i häftet ochlös övningsuppgifterna tillsammans med en kamrat.

### Hur kan byggblocken kombineras och anpassas för att lösa häftets problem?

**Algoritmer**
Algoritmer: Finns överallt

• En algoritm definieras i häftet som: ”ett begränsat antal steg utförda i en bestämd ordning, där varje steg består av en entydig och elementär instruktion som kan tolkas och utföras.”

• Algoritmer som du tolkar och utför finns överallt; t.ex. bakrecept, lång division med trappa och IKEA:s monteringsanvisningar.

• Ett program skrivet i Python 3 är en algoritm som en dator kan tolka och utföra extremt snabbt.

• Skillnaden är instruktionerna, deras egenskaper och vem som utför dem.

```python
x = 1
while x <= 1000:
    y = x*x
    print(y)
    x = x + 1
```

2018-11-09 Johnny Åberg, Linköpings Universitet

Algoritmer: Översiktlig beskrivning

• Jag har valt flödesschema och pseudokod som är baserade på byggblocken. Beskrivningarna kompletterar varandra, visar algoritmens uppbyggnad och egenskaper på lite olika sätt.

• En dator förstår inte flödesscheman eller pseudokod.

• Det är ofta enkelt att översätta ett flödesschema till ett program skrivet i Python 3.

```plaintext
<table>
<thead>
<tr>
<th>Sätt ugnen på 200 grader.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blanda en nypa socker med äggen. Visa luftigt.</td>
</tr>
<tr>
<td>Häll smeten i en form och grädda i ugnen i ca 15 min.</td>
</tr>
</tbody>
</table>
```

2018-11-09 Johnny Åberg, Linköpings Universitet

Algoritmer: Välja, kombinera och anpassa

• För att skriva ett program som löser ett av häftets problem måste du alltså:
  a) välja vilka (och antal) av häftets 7 byggblock som ska användas.
  b) kombinera alla byggblocken på rätt sätt.
  c) anpassa varje byggblock.

• Ovanstående kallas för ”att ta fram en algoritm till ett problem”.

• Alla frihetsgrader gör att enormt (oändligt?) många olika problem kan lösas med häftets 7 byggblock.

• Python 3 innehåller dessutom fler byggblock som kan kombineras och skapa ännu fler byggblock...
Algoritmer: Komplicerad aktivitet

• Att ta fram algoritmer är fundamental i programmering, kreativt och kan vara tidskrävande, eller omöjligt, även för erfarna programmerare.
• Hur det går till teoretiskt och praktiskt beskrivs ytterst knapphändigt i häftet. Det är svårt att beskriva och ännu svårare att lära ut.
• Häftet innehåller endast några exempel, vilka du kan studera och utgå ifrån, samt några övningsuppgifter med tips.
• Ovanstående är kanske häftets, och kursens, största brist?
• Notera likheten med problemlösning i matematik.

Algoritmer: Eget arbete

• Först löser vi övningsuppgift 1 i kapitel 4.5 på tavlan.
• Nu är det dags för att ta fram och beskriva några enklare algoritmer.
• Läs kapitel 4 ”Om algoritmer” i häftet och lös övningsuppgifterna tillsammans med en kamrat.

Hur arbetar programmerare?

Arbetssätt
Arbetssätt: En enkel metod

- Ett förslag du kan använda för att lösa häftets problem med programmering.
- Endast några större och viktigare aktiviteter ingår. Den vanliga aktiviteten ”Testa programmet” är t.ex. utelämnad.
- ”Skriv ett program” inkluderar att ta fram en algoritm.
- Förhoppningsvis ger metoden ett visst stöd.
- Notera likheten med matematisk problemlösning.

Arbetssätt: Kopiera från häftet

- Problem: ”Vilka är de 100 första kubiktalen?”.
- Problemet är väldigt likt ett exempel i häftet nämligen ”Vilka är de 1000 första kvadrattalen?”. Undersök om exemplet går att återanvända; helt eller delvis.
- Kopiera exemplet och ändra det som behövs.

```
x = 1
while x <= 1000:
    y = x*x*x
    print(y)
    x = x + 1
```

- Alla ändringar är rödmarkerade.

Arbetssätt: Återanvänd dina program

- Problem: ”Skriv ett program som visar ett antal kvadrat- och kubiktal på skärmen, där det första talet är 1. Antalet ska matas in på tangentbordet”.
- Problemet påminner om ”Vilka är de 100 första kubiktalen?”. Prova om den lösningen kan återanvändas och kanske kompletteras med något byggblock (nyckelord ”matas in”).
- Utgå från programmet, ändra och komplettera det som behövs.

```
x = 1
antal = int(input("Ange antal?"))
while x <= antal:
    y = x**2
    y1 = x**3
    print(y, y1)
    x = x + 1
```

2018-11-08 Johnny Åberg, Linköpings Universitet
Arbetssätt: Skriv och testa i steg

• Problem: ”Skriv ett program som visar de 25 första termerna i följande talserie, och deras summa, på skärmen: 1 + 1/2 + 1/4 + 1/8 + ...”

• Inget i häftet, eller i något program, går att återanvända. Utgå från nyckelorden ”visa på skärmen”, ”25 termer” och ”summa”, analysera problemet och skissa en algoritm.

• Prova att skriva och testa programmet i 4 steg.

```python
y = 1
print(y)
y = 1
x = 1
while x<=25:
    print(y)
x = x + 1

s = 0
y = 1
x = 1
while x<=25:
    print(y)
s = s + y
y = y/2
x = x + 1
print(s)
```

Arbetssätt: Eget arbete

• Först löser vi övningsuppgift 5 i kapitel 5.5 på tavlan.

• Nu är det dags att fortsätta öva på att använda byggblocken, ta fram algoritmer samt skriva lite mer avancerade program.

• Prova att skriva och testa programmet på föregående bild i de föreslagna stegen: Bra övning på ett modernt, iterativt och evolutionärt arbetssätt.

• Läs kapitel 5 ”En enkel metod” och lös övningsuppgifterna tillsammans med en kamrat.

Hur kan närbesläktade problem lösas enklare?

Typisk algoritm
Typisk algoritm: Problemgrupper

- Närbesläktade problem i en problemgrupp kan ofta lösas med samma algoritm.
- Om du inser att ett problem tillhör en problemgrupp, och känner till gruppens typiska algoritm, kan problemet lösas betydligt snabbare och enklare.
- Det gör du genom att kopiera och anpassa den typiska algoritmen så att den löser problemet. Utgå från häftet eller dina program.
- Exempel på problemgrupper i Ma1c: slumpförsök, talföljder och ränta.
- Notera likheten med typtal i matematik.

Typisk algoritm: Slumpförsök

- Baserad på experimentell sannolikhet: "Uppskatta sannolikheten P(H) genom att simulera ett slumpförsök många gånger, räkna antalet gånger händelsen H inträffar och beräkna den relativa frekvensen för H".
- Exempel på slumpförsök: kasta tärningar en eller flera gånger, singla mynt och dra lotter.
- Notera matematiken som algoritmen baseras på. Matematik är användbart!

Typisk algoritm: Eget arbete

- Först löser vi övningsuppgift 4 i kapitel 6.1.3 på tavlan.
- Nu är det dags att använda den typiska algoritmen för slumpförsök, och allt annat du lärt dig i häftet, för att lösa några avslutande sannolikhetsproblem.
- Läs kapitel 6 ”Problemgrupper” och lös övningsuppgifterna tillsammans med en kamrat.
Avslutning: Tips på fortsättning

- Det finns mängder av självstudiematerial på Internet: hemsidor, kurser, böcker och filmer om t.ex. grundläggande programmering, Python 3 och matematiska problem med olika svårighetsgrader.
- Mycket är gratis och många gånger av hög kvalitet.
- Fler tips hittar du i häftets Appendix C "Mer om Python och programmering".

Avslutning: Kursutvärdering

- Enkäten är anonym.
- Syftet är att utvärdera och förbättra kursen, undervisningen och häftet.
- Läs igenom den snabbt - några oklarheter eller andra frågor?
- Lägg den i kuvertet när du är färdig.

Avslutning: Tack

- Stort tack för att du var med på kursen!
Elevens förberedelser inför kursen

• Ta med dator, mus, matematikbok för Ma1c, papper och penna.
• Repetera sannolikhetsläran i Ma1c; speciellt begreppen som är uppräknade i kurshäftets kapitel 2.3 "Matematiska förkunskaper".
• Hitta en kamrat att arbeta med. Delar av kursen är grupparbete.
• Lär gärna igenom häftet och presentationen.
• Kontakta mig på jasjas1818@gmail.com om du har några frågor eller kommentarer.

8.6 Bilaga 6: Elevhäfte med övningsuppgifter

Det var komplicerat att inkludera elevhäftet i rapporten. Ett försök att visa 2 sider av elevhäftet på varje sida i rapporten misslyckades, sidnumreringen har försvunnit, innehållsförteckningen och rubriknumreringen är korrigerad manuellt. Det finns fler problem med främst formateringen, men texterna och bilderna är korrekta och det är det viktigaste.
Grundläggande programmering
i
Python 3

- lösning av sannolikhetsproblem i Ma1c

Sammanfattning
Det här häftet ingår i en studie som handlar om lösning av matematiska problem med programmering i kursen Matematik 1c (Ma1c) på gymnasieskolans teknik- och naturvetenskapsprogram. Häftet är mycket kortfattat och behandlar enbart det eleverna behöver för att skriva program som löser några avslutande sannolikhetsproblem rörande slumpförsök. Det som behövs är t.ex. en minimal del av utvecklingsmiljön IDLE, några utvalda byggblock i programspråket Python 3 samt en enkel metod för programmering. Förhoppningen är att häftet, i kombination med lärarledd undervisning och övning, lär eleverna lite grundläggande programmering och exemplifierar hur programmering kan användas för problemlösnings i Ma1c. Målgruppen är främst elever som läst Ma1c och saknar erfarenhet av programmering. Eleverna förutsätts ha tillgång till varsin dator, Windows 10 på engelska med administratörsrättigheter samt Internet. Häftet kan enkelt kompletteras med fler problemgrupper i Ma1c som t.ex. talföljder, primtal och ränta.

Innehållsförteckning
1 Utvecklingsmiljön IDLE ..............................................................................................2
  1.1 Installera ...............................................................................................................2
  1.2 Skapa en mapp för dina program ........................................................................3
  1.3 Arbeta med program ............................................................................................3
    1.3.1 Skriva och spara ............................................................................................3
    1.3.2 Starta och avbryta ..........................................................................................3
    1.3.3 Ändra ...............................................................................................................3
    1.3.4 Felsöka ..........................................................................................................4
  1.4 Övningsuppgifter .....................................................................................................4
2 Om programmering ........................................................................................................5
  2.1 Definition ...............................................................................................................5
  2.2 Svårt att beskriva .................................................................................................6
  2.3 Matematiska förkunskaper ..................................................................................6
  2.4 Lite studieteknik ....................................................................................................6
  2.5 Samarbete .............................................................................................................6
3 Utvalda byggblock i Python 3 ....................................................................................6
  3.1 Inledning ...............................................................................................................7
    3.1.1 Små, enkla och generella ..............................................................................7
    3.1.2 Variabler .......................................................................................................7
  3.2 Beräkna matematiska uttryck ..............................................................................8
  3.3 Visa på skärm .......................................................................................................8
  3.4 Vänd på inmatning från tangentbordet .................................................................9
  3.5 Generera slumptal ...............................................................................................9
1 Utvecklingsmiljön IDLE

1.1 Installera


1. Försök hitta programmet ”IDLE (Python 3.6 32-bit)” på din dator. Använd startknappen i Windows 10, leta efter ”Python 3.6” eller sök efter ”idle” i Windows sökfunktion.

2. Om du inte hittar programmet måste du installera det:
   a) Starta Internet Explorer eller någon annan webbläsare.
   b) Gå till www.python.org/downloads och välj ”Download Python 3.6.4”.
   c) Starta den hämtade filen ”python-3.6.4.exe” och följ instruktionerna.

1.2 Skapa en mapp för dina program

För att du snabbt ska hitta dina program är det bra att spara dem i en separat mapp på datorns skrivbord. Du kommer att återanvända dina program många gånger t.ex. för att kopiera delar av dem när du skriver nya program.

1. Högerklicka på skrivbordet, välj New->Folder och ge mappen namnet ”Mina program”.

1.3 Arbeta med program

1.3.1 Skriva och spara

När IDLE är startat kan du börja skriva program. Ett program är en textfil uppdelad på rader som följer de strikta regler som specificeras i Python 3 [Ref. 2]. En felstavning eller oavsiktlig indentering räcker för att orsaka ett fel, så var noggrann och kontrollera allt du skriver. Var observant på textens färg som ger ett visst stöd.

1. Välj File->New File i fönstret ”Python 3.6.4 Shell” för att skapa en tom textfil till ditt program.
2. Välj File->Save As... i det nya fönstret, välj mappen ”Mina program” på skrivbordet och ge filen namnet ”program.py”. Notera att en fil som innehåller ett program i Python 3 alltid slutar på ”.py”.
3. Skriv nedanstående i fönstret ”program.py”:

   ```python
   print("Hejsan världen!")
   ```

4. Välj File->Save i ”program.py” för att spara programmet.

Om du ger filerna unika och beskrivande namn, samt sparar dem i mappen ”Mina program”, blir det enklare att hålla ordning på dina program. T.ex. kan du ge filen som innehåller programmet kopplat till problem 1 namnet ”problem1.py”.

1.3.2 Starta och avbryta

När du skrivit och sparat programmet kan du starta det:

1. Välj Run->Run Module i ”program.py”.
2. Kontrollera att Hejsan världen! skrivs ut i ”Python 3.6.4 Shell”.
3. När prompten ”>>>“ visas i ”Python 3.6.4 Shell” vet du att programmet har avslutats.


1.3.3 Ändra

Du kommer att ändra dina program många gånger t.ex. för att rätta fel eller få dem att lösa nya problem. Du ändrar ett program genom att lägga till eller ta bort text i filen som innehåller programmet.

Johnny Åberg, Linköpings Universitet 2018
1. Ändra och lägg till två rader under den östra raden i ditt program:

    print("Det här är rad 2 i programmet...")
    print("...och här kommer rad 3.")

2. Spara och starta programmet enligt ovan.
3. Notera i vilken ordning texterna skrivs ut.

Ett program är i stort sett alltid uppdelat på flera rader. När ett program har startats
exekveras den första raden i filen först, sedan raden efter, därefter nästa och så vidare.
När den sista raden i filen har exekverats avslutas programmet.

1.3.4 Felsöka

Nyskrivna och ändrade program innehåller nästan alltid fel. Vissa fel går snabbt att
hitta, medan andra fel kan ta lång tid att hitta även för erfarna programmerare. Det här
kapitlet behandlar fel som IDLE hittar direkt när du har startat ditt program.
1. Ändra och inför ett stavfel på den första raden i ditt program:

    prin("Hejsan världen!")

2. Spara och starta programmet enligt ovan.
3. Notera felmeddelandet.
4. Rätta programmet genom att ändra och återställa raden:

    print("Hejsan världen!")

5. Spara och starta programmet enligt ovan.

Läs felmeddelandet som visas på skärmen noggrant, då det ofta innehåller tillräckligt
med information för att du ska hitta den felaktiga raden och förstå vad felet beror på.
Några vanliga fel som IDLE hittar är felstavningar, otillåtna eller missade tecken (t.ex
parenteser, kolonn eller citationstecken), felaktiga indragningar samt okända variabler.

Traceback (most recent call last):
  File "test.py", line 12, in <module>
  if (t1 == t2):
NameError: name 't1' is not defined

Exempel 1: Felmeddelande i IDLE.

1.4 Övningsuppgifter

Syftet med övningsuppgifterna är att du ska lära dig använda IDLE, hitta och rätta några
enkla fel i dina program. Kopiera programmen i uppgifterna till IDLE och kontrollera
dina svar innan du tittar i facit. OBS! Om du kopierar från ett Word-dokument måste du
ta bort och skriva om alla citationstecken och minustecken. Ibland behöver du även
justera indenteringen. Kopiering från PDF-dokument kräver andra korrigeringar.
1. Hitta och rätta felen i nedanstående program. Notera hur IDLE ändrar färgen på
texten när du rättar felen.
print("Hejsan världen!")
print("Det här är rad 2 i programmet...")
print("...och här kommer rad 3!")

2. Hitta och rätta fel i nedanstående program. Starta programmet när det är felfritt. Vad skriver programmet ut för talserie?

```python
x = 1
while x <= 10:
    y = x * x
    print(y)
    x = x + 1
```

3. Utgå från det felfria programmet i uppgift 2, inför ett nytt fel genom att radera kolonet, starta programmet och notera felmeddelandet i IDLE. Ändra och återställ programmet samt kontrollera att felmeddelandet har försvunnit. Repetera, men inför följande fel i tur och ordning: (a) stava while fel, (b) ta bort indenteringen på rad 4, (c) radera rad 1, (d) ändra likhetstecknet på rad 5 till "$==" och (e) var kreativ, experimentera och hitta på egna fel.

Facit


3. -

2 Om programmering
Nu har du provat IDLE, felsökt och arbetat med två enklare program och är redo att lära dig lite mer om vad programmering innebär.

2.1 Definition
Programmering är ett oklart begrepp som kan betyda en hel del. Häftet definierar programmering som "ett antal aktiviteter, utförda i en viss ordning, vilka krävs för att skapa ett felfritt program". Ska du t.ex. lösa problemet "Vilka är de första 1000 kvadrattalen" med programmering, så måste även en erfaren programmerare göra ganska mycket från det han läser problemtexten första gången tills programmet visar kvadrattalen på skärmen. Några exempel på aktiviteter är att analysera problemet, ta fram en algoritm, skriva och testa programmet samt eventuellt felsöka lösningen.
Exempel 2: Ett av många möjliga program i Python 3 som löser problemet ”Vilka är de första 1000 kvadrattalen?”.

2.2 Svårt att beskriva
Av bl.a. utrymme-skäl kan häftet inte räkna upp och beskriva allt som ingår i begreppet programmering, så endast några större och viktigare aktiviteter finns med. Det innebär att många mindre eller omedvetna delaktiviteter saknas. Trots det är förhoppningen att du kommer att ha nytta av häftet.

2.3 Matematiska förkunskaper
Programmering kräver ofta att man har goda kunskaper i matematik. Längre fram i häftet är det särskilt viktigt att du kan sannolikhetsläran i Ma1c [Ref. 3]. Repetera i synnerhet begreppen utfall, händelse, komplementhändelse, teoretiskt, exakt och uppskattat värde på sannolikhet, relativ frekvens, experimentell sannolikhet, enkla slumpförsök, slumpförsök med flera steg, träddiagram samt likformig sannolikhetsfördelning.

2.4 Lite studieteknik

2.5 Samarbete
Det är viktigt att hjälpa och samarbeta med dina kamrater. Att programmera är många gånger ett lagarbete som blir roligare, är lärorikare och går fortare om man hjälps åt. Ta hjälp av läraren om du kör fast.

3 Utvalda byggblock i Python 3
Nu är förberedelserna avklarade och det är snart dags att skriva ditt första program, men först behöver du lära dig vad program är uppbygdda av.


3.1 Inledning

3.1.1 Små, enkla och generella

Exempel 2 visar hur ett program består av små, enkla och generella byggblock, där varje byggblock har en specifik funktion. Ett ensamt byggblock gör sällan någon nytta, men om flera byggblock kombineras och anpassas på rätt sätt kan massvis av problem lötas. För att kunna skriva program som löser häftets problem, behöver du lära dig följande utvalda byggblock i Python 3:

- beräkna matematiska uttryck
- visa på skärmen
- vänta på inmatning från tangentbordet
- generera slumptal
- välja rader
- repetera rader

Det är viktigt att du lär dig alla byggblock ordentligt; vilka som finns och hur de kan användas. Appendix A exemplifierar hur byggblocken kan kombineras och anpassas i program.

3.1.2 Variabler

Variabler är centrala i all programmering och ingår ofta i byggblocken. Utan variabler är de flesta program omöjliga att skriva. Ett program kan ha många olika variabler och de har stora likheter med matematikens namne. En variabel i Python 3 kan liknas vid en låda med namn där ett tal i taget kan sparas. Variabeln:

- kan tilldelas ett värde - ett tal läggs i lådan.
- kan användas i beräkningar - talet plockas upp ur lådan, används och läggs sedan opåverkat tillbaka i lådan.
- behåller sitt värde tills en ny tilldelning sker - talet plockas upp ur lådan, slängs och ett nytt tal läggs i lådan. Det gamla värdet försvinner eftersom endast ett tal i taget kan sparas i lådan.

En variabel skapas när den tilldelas ett värde första gången och försvinner när programmet avslutas. Namnet på en variabel kan vara nästan hur långt som helst och bestå av både bokstäver och siffror. Börja alltid med en bokstav och namnge dina variabler omsorgsfullt utifrån deras innehåll eller användning, så blir ditt program lättare att läsa och förstå.

```python
# Tilldela variabeln x ett värde.
# -----------------------------------------------
x = 1

# Använd x i en beräkning.
# -----------------------------------------------
y = 2*x + 3

# Tilldela x ett nytt värde (det gamla värdet försvinner).
# -----------------------------------------------
x = y + x

# Ge variabler bra namn.
# -----------------------------------------------
```

Johnny Åberg, Linköpings Universitet 2018
\[ x_2 = 3 \]
\[ \text{antalKast} = 100 \]
\[ \text{ögon} = 6 \]

*Exempel 3: Variabelanvändning i Python 3.*

### 3.2 Beräkna matematiska uttryck

**Beräkna, räkna ut, öka, minska**

\[ <\text{variabel}> = <\text{matematiskt uttryck}> \]

De flesta numeriska och algebraiska uttryck är enkla att beräkna i ett program. Python 3 innehåller många användbara matematiska funktioner, men i det här häftet kommer du bara använda kombinationer av de fyra räknesätten. Använd samma prioriteringsregler som i matematiken och ta hjälp av parenteser vid behov eller om du är osäker på räkneordningen.

```python
# Addera, subtrahera, multiplicera och dividera
# --------------------------------------------------------------------
y = 1 + 1
y = 1 - 1
y = 1*1
y = 1/1

# Kombinera ovanstående och använda parenteser. Notera att punkt
# används i decimaltal.
# --------------------------------------------------------------------
x = 3.7
y = (y + 1)*2 - 3/(x - 4)
```

*Exempel 4: De fyra räknesätten i Python 3.*

### 3.3 Visa på skärmen

**Skriva ut, visa på skärmen**

\[ \text{print(<text och/eller variabel>)} \]

För att skriva ut text och variabelvärdet i ett program använder du i Python 3 `print`. Utskriften visas på en rad i fönstret ”Python 3.6.4 Shell” på dators skärm.

```python
# Skriva ut text på en rad.
# -------------------------------------------
print("Hejsan världen!"))

# Skriva ut en tom rad.
# -------------------------------------------
print("")

# Skriva ut ett variabelvärde.
```
# Kombinera ovanstående.
# ------------------------------------------------------------------
print("Körd sträcka är", x, "m, då hastigheten är 28 km/h")

Exempel 5: Användning av print i Python 3.

### 3.4 Vänta på inmatning från tangentbordet

Ibland vill man att ett program ska vänta på att någon matar in ett heltal på datorns tangentbord t.ex. genom att en fråga visas på skärmen, personen skriver in talet på tangentbordet och avslutar med att trycka på returtangenten. I Python 3 kan du använda int och input som väntar på en inmatning i fönstret ”Python 3.6.4 Shell”. Det inmatade heltala sparas i <variabel> som du därefter kan använda på vanligt sätt i t.ex. beräkningar.

# Vänta på att ett heltal ska matas in på tangentbordet.
# --------------------------------------------------------------------
x = int(input("Ange ett heltal? "))

Exempel 6: Inmatning av heltal i Python 3.

### 3.5 Generera slumpal

För att skriva ett program som simulerar och analyserar slumpmässiga händelser och föllopp är slumpal synnerligen användbart. I det här häftet kommer du bara att använda slumpal med likformig sannolikhetsfördelning och återläggning t.ex. för att simulera tärningskast. I Python 3 kan du göra det med choice och range i modulen random. Om du bestämmer antalet möjliga utfall med <antal>, så kommer ett slumpmässigt valt heltal i intervallet 1 till <antal> att sparas i <variabel>. Varje heltal representerar då ett utfall i ett slumpförsök.

# Inkludera modulen random endast en gång i varje program.
# ---------------------------------------------------------------------
from random import choice

# Simulera ett tärningskast genom att slumpmässigt välja ett av
# heltalen i intervallet 1 till 6 och spara det i variabeln x. Låt
# slumptalet representera antalet ögon tärningen visar.
# --------------------------------------------------------------------
x = choice(range(6)) + 1

**Exempel 7:** Slumptalsgenerering i Python 3.

### 3.6 Välja rader

Ofta vill man välja vilka rader som ska exekveras beroende på vad som händer i ett program. Python 3 använder *if-else* som exekverar en eller flera indragna rader om ett logiskt uttryck är sant och några andra indragna rader om det är falskt. Vägen genom programmet bestäms alltså av uttryckets värde. Uttrycket består av jämförelser mellan tal och variabler, t.ex. `==` (lika med) och `>` (större än), vilka kan kombineras med `and` eller `or`.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A and B</th>
<th>A or B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Falskt</td>
<td>Falskt</td>
<td>Falskt</td>
<td>Falskt</td>
</tr>
<tr>
<td>Falskt</td>
<td>Sant</td>
<td>Falskt</td>
<td>Sant</td>
</tr>
<tr>
<td>Sant</td>
<td>Falskt</td>
<td>Falskt</td>
<td>Sant</td>
</tr>
<tr>
<td>Sant</td>
<td>Sant</td>
<td>Sant</td>
<td>Sant</td>
</tr>
</tbody>
</table>

_Figur 1: Sanningstabell för logiskt `and` och `or`. A och B är jämförelser mellan tal och variabler._

# I det logiska uttrycket kan bl.a. följande jämförelser göras:
# === “lika med”    > “större än”
# !== “avskilt från” < “mindre än”
# och därefter kombineras med:
# or  “logiskt eller” and “logiskt och”

```python
n = 3
if n < 4:
    print("Exekveras, eftersom 3 < 4 är sant.")
else:
    print("Exekveras inte.")
```

# Använd parenteser i sammansatta logiska uttryck
# --------------------------------------------------------------------

```python
if (n > 2) or (n == 0):
    print("Exekveras, eftersom 3 > 2 är sant.")
```

# En annan kombination av logiska uttryck.
# --------------------------------------------------------------------

```python
if 1 < n < 9:
    print("Exekveras, eftersom 1 < 3 < 9 är sant.")
```

**Exempel 8:** Användning av *if-else* i Python 3. Notera att *else* kan utelämnas.
3.7 Repetera rader

Ibland vill man repetera något ett visst antal gånger eller vänta tills en speciell händelse inträffar i ett program. I Python 3 kan du lösa det med `while` som exekverar en eller flera indragna rader gång på gång så länge ett logiskt uttryck är sant. När uttrycket blir falskt, avbryts `while` och exekveringen fortsätter efter de indragna raderna. Vägen genom programmet och antalet repetitioner bestäms alltså av uttryckets värde. Uttrycket är uppbyggt på samma sätt som i föregående kapitel.

# Repetera 10 gånger och skriv ut alla heltal mellan 0 och 9.
# --------------------------------------------------------------------
 n = 0
while n < 10:
    print(n)
    n = n + 1
# Vänta på att heltal 1 matas in på tangentbordet.
# --------------------------------------------------------------------
 x = 0
while x != 1:
    x = int(input("Ange ett heltal?"))
# Repetera tills vidare.
# --------------------------------------------------------------------
while True:
    print("Exekveras tills programmet avbryts med Ctrl-C.")

Exempel 9: Användning av `while` i Python 3.

3.8 Använda kommentarer

Kommentarer påverkar inte vad ett program gör utan används främst till att göra programmet enklare att läsa, komma ihåg och förstå. Det blir speciellt tydligt när du läser andras program. Kommentera dina program ofta och rikligt, t.ex. vad programmet gör och hur det fungerar. En kommentar inleds med `#` och följs av en valfri text.

# En kommentar som består av en rad.
print("Hejsan världen!") # Kan även placeras sist på raden.

"""
Ett alternativ till `#` är att skriva 3 citationstecken före och efter en kommentar. Då kan kommentaren skrivas på flera rader.
"""

Exempel 10: Kommentarer i Python 3.
3.9 Övningsuppgifter
Syftet med övningsuppgiftarna är att du ska lära dig använda byggblocken, kombinera dem lite och skriva dina första program. Använd kapitel 3 på samma sätt som en formelsamling och ta hjälp av exempelen i Appendix A. Glöm inte att kommentera dina program och spara dem i mappen ”Mina program”. Svårare uppgifter är markerade med *.

1. Skriv ett program som visar följande på skärmen:
   a) texten ”Glöm inte citationstecken för texter.”
   b) värdet av $1 + 2*3 – 4/5$
   c) Värdet av några andra numeriska uttryck. Hitta på uttrycken själv.
   d) texten ”$1 + 2*3 – 4/5$”

2. Vilka tal kommer nedanstående program att visa på skärmen? Räkna först på papper och kontrollera därefter dina uträkningar genom att kopiera programmen till IDLE.
   a) $n = 9$
      ```python
      print(n - 4)
      ```
   b) $\pi = 3.14$
      ```python
      radie = 1.25
      area = pi*radie*radie
      print("Cirkelns area =", area)
      ```
   c) $n = 10$
      ```python
      m = n/2
      m = n + m
      print("m =", m)
      ```
   d) $x = 1$
      ```python
      y = 3
      # x = x + x + 4*x
      y = x + x + y
      z = y
      z = 2*y – x + z
      print("x =", x, "y =", y, "z =", z)
      ```


   a) summan av talen
   b) differensen
   c) produkten
   d) kvoten

5. Visa en funktionstabell på skärmen för $y = 3*x + 10$, där $x$ är alla heltal i intervallet 10 till 20. Tabellen ska innehålla både x- och y-värden.

6. Skriv ett program som simulerar och visar följande slumpmässiga försök på skärmen:
   a) 1 slant singlas
b) 1 tärning kastas
c) 2 åttasidiga tärningar kastas
d) 1 tärning kastas 10 000 gånger

7. Skriv ett program som avgör om ett heltal är positivt, negativt eller 0. Talet ska matas in på tangentbordet och ”positivt”, ”negativt” eller ”noll” skrivas ut på skärmnen.

8. * Var kreativ, experimentera och undersök byggblocken själv. Ändra dina program och lägg till, ta bort eller kombinera byggblocken på nya sätt. Ett utmärkt sätt att lära sig programmera.

Facit (programmen är förslag)

1. –

2. a) 5       b) 4.90625   c) m = 15.0   d) x = 1 y = 5 z = 14

3. pi = 3.14
   radie = int(input("Ange en radie? "))
   omkrets = pi*radie*2
   area = pi*radie*radie
   print("Cirkelns omkrets =", omkrets)
   print("Cirkelns area =", area)

4. x1 = int(input("Ange första talet? "))
   x2 = int(input("Ange andra talet? "))
   print("Summa =", x1 + x2)
   print("Differens =", x1 - x2)
   print("Produkt =", x1*x2)
   print("Kvot =", x1/x2)

5. x = 10
   while x <= 20:
      y = 3*x + 10
      print(x, y)
      x = x + 1

6. a) from random import choice
    myntsida = choice(range(2)) + 1
    if myntsida == 1:
        print("Krona")
    else:
        print("Klave")

b) from random import choice
    ögon = choice(range(6)) + 1
    print("Antal ögon =", ögon)

c) from random import choice
    ögon1 = choice(range(8)) + 1
    ögon2 = choice(range(8)) + 1
    print("Antal ögon =", ögon1, ögon2)

d) from random import choice
    n = 0
while n < 10000:
    ögon = choice(range(6)) + 1
    print(ögon)
    n = n + 1

7. x = int(input("Ange ett heltal? "))
   if x > 0:
      print("Positivt")
   if x < 0:
      print("Negativt")
   if x == 0:
      print("Noll")

8. -

4 Om algoritmer

4.1 Definition
En algoritm definieras i det här häftet som ”ett begränsat antal steg utförda i en bestämd ordning, där varje steg består av en entydig och elementär instruktion som kan tolkas och utföras.”. De algoritmer du ska ta fram måste alltså vara ändliga, kompletta, bestå av byggblocken i föregående kapitel och utföras i rätt ordning från start till stopp. Algoritmer används inte bara vid programmering utan finns även i vardagslivet. Några exempel är matlagningsrecept, lång division med trappa och IKEA:s monteringsanvisningar.

4.2 Flödesschema och pseudokod
Exempel 11: En algoritm i vardagslivet för en hungrig tonårigs morgonbestyr fram till skolan börjar 08:30? Flödesschemat till höger visar ”At frukost” i detalj.

4.3 Program i Python 3


4.4 Kreativt och utmanande

4.5 Övningsuppgifter
Syftet med övningsuppgifterna är att du ska öva på att ta fram, förstå och beskriva algoritmer.

1. Utgå från ditt program som löste uppgift 3 i kapitel 3.9 och beskriv programmets algoritm översiktligt med pseudokod och ett flödesschema.

2. Översätt nedanstående flödesschema och pseudokod till ett program i Python 3.

3. Utgå från ditt program som löste uppgift 7 i kapitel 3.9 och beskriv programmets algoritm översiktligt med pseudokod och ett flödesschema.

4. * Ta fram en algoritm som löser problemet att din mobiltelefon är urladdad. Beskriv algoritmen i pseudokod och med ett flödesschema. Utgå t.ex. från ”Är batteriet fulladdat?”, ”Är något eluttag ledigt?” och ”Finns det en sladd?”. Se Exempel 11.

Facit (pseudokod, flödesschematic och program är förslag)

1. Se enkel funktionsräknare i Appendix A, men ändra variabelnamn, uttryck och utskrifter i pseudokod och flödesschema.

2. while True:
   x = int(input("Ange ett heltal? "))
   if x == 0:
      print("Otillåtet tal")
   else:
      y = 1/x
      print("Talet inverterat=", y)

3. Se nedan.
4. En enkel metod


**Pseudokod**

Vänta på inmatning av ett tal.
Visa "Positivet" på skärmen om talet är större än 0.
Visa "Negativt" om talet är mindre än 0.
Visa "Noll" om talet är 0.

**Figur 3: En enkel metod för att lösa häftets problem med programmering.**
5.1 Analysera problemet
Läs problemtexten sakt, flera gånger och tänk igenom problemet noggrant. Det är viktigt att du förstår problemet och alla förutsättningar ordentligt. Några kontrollpunkter du kan använda är:

- Är begrepp och formuleringar i texten entydiga eller kan jag ha missförstått något?
- Gör experiment med penna, papper, tärningar och annat för att undersöka hur problemet kan lösas för hand. Det kan många gånger leda fram till en algoritm som kan beskrivas i ett flödesschema och översättas till Python 3.
- Våga prova även halvfärdiga lösningar. Problemförståelse och program växer ofta fram successivt genom försök, misslyckanden och små förbättringar.

5.2 Skriv ett program
När du analyserat problemet omsorgsfullt är det dags att ta fram en algoritm, eventuellt beskriva den i t.ex. ett flödesschema och översätt det till ett program i Python 3. Fortsätt använda byggblocken på samma sätt som tidigare och försök tillämpa följande viktiga principer (fallande prioritet):

- Återanvänd så mycket som möjligt från dina program.
- Undersök och kopiera om möjligt valda delar av häftets olika exempel t.ex. pseudokod, flödesscheman och program.
- Komplettera din lösning med det som saknas. Att ta fram en helt ny algoritm och översätta den till Python 3 är din sista utväg.

Ett avslutande råd är att inte skriva hela programmet på en gång utan istället göra det i steg. Då kan du testa och felsöka programmet i mindre delar, vilket ofta är betydligt enklare. Se övningsuppgift 1 och 7 i kapitel 6.1.3.

5.3 Starta programmet
När du skrivit färdigt ditt program är det dags att starta det. Om IDLE skriver ut ett felmeddelande, eller svaret som visas på skärmen är fel, måste du felsöka det vilket beskrivs i nästa kapitel. När programmet visar rätt svar är du färdig: Du har skrivit ett program som löser ett problem!

5.4 Felsök lösningen
Innehåller ditt program ett eller flera fel måste du hitta och rätta dem. Kapitel 1.3.4 och Appendix B presenterar några metoder som du kan prova. Metoderna är effektiva, men det tar tid att lära sig använda dem.

5.5 Övningsuppgifter
Syftet med övningsuppgifterna är att du ska fortsätta öva på att använda byggblocken, ta fram algoritmer samt skriva lite mer avancerade program. Läs, återanvänd och kopiera så mycket du kan från dina program, häftets exempel i Appendix A och kapitel 3; pseudokod, flödesscheman och program.
1. Beräkna summan \( s \) av de 10 första termerna i talserien 1, 2, 3, … genom att:
   a) addera talen för hand på papper.
   b) använda formeln \( s = \frac{n*(n+1)}{2} \) och en miniräknare, där \( n \) är antalet termer.
   c) skriva ett program som inte använder formeln ovan. Börja med att ta fram en
      algoritm, t.ex. ett flödesschema, och översätt den därefter till Python 3.
   d) Gör om uppgiften men summera de 1 000 000 000 första termerna.

2. Skriv ett program som visar 25 slumpmässigt valda namn på skärmen.
   Sannolikheten för ”Kalle”, ”Valle” och ”Viktor” ska vara 1/3 vardera.

3. Skriv ett program som visar ett antal multiplikationstabeller på skärmen t.ex. 1
   till 12.

4. Skriv ett program som bestämmer och visar medelvärdet av 10 heltal, samt det
   minsta och största talet, på skärmen. Talen ska matas in på tangentbordet.

5. * Skriv ett program som visar en slumpmässig tipsrad på skärmen. En tipsrad
   består av 13 rader. På varje rad ska sannolikheten för ”1” vara 60%, ”x” 10%
   och ”2” 30%.

   ska ligga i intervallet \([1..10]\) och väljas utan återläggning. Programmet ska
   fortsätta tills det avbryts med Ctrl-C.

7. * Skriv ett program som vägleder dig fram till ett hemligt, och slumpmässigt
   genererat, heltal mellan 1 och 12. Programmet ställer frågan ”Gissa vilket mitt
   hemliga tal är?”, du matar in din gissning på tangentbordet och programmet
   skriver ut ”Rätt svar”, ”För stort” eller ”För litet”. Så länge du gissar fel
   fortsätter programmet och ställer frågan gång på gång.


Facit (pseudokod, flödesscheman och program är förslag)

1. a och b) 45

   c) 
   ```python
   n = 0
   summa = 0
   while n < 10:
       summa = summa + n
   n = n + 1
   print(n, "termer ger summan", summa)
   ```

   d) 4999999950000000

2. ```python
   from random import choice
   n = 0
   while n < 25:
       namn = choice(range(3)) + 1
       if namn == 1:
           print("Kalle")
       if namn == 2:
   ```
print("Valle")
if namn == 3:
    print("Viktor")
n = n + 1

3. x1 = 0
   while x1 <= 12:
      x2 = 0
      while x2 <= 12:
          print(x1, "\times", x2, "=", x1*x2)
          x2 = x2 + 1
      print("")
      x1 = x1 + 1

4. summa = 0
   n = 1
   while n < 10:
      x = int(input("Ange ett tal? "))
      if n == 1:
          minsta = x
          största = x
      if x < minsta:
          minsta = x
      if x > största:
          största = x
      summa = summa + x
      n = n + 1
      print("Minsta talet: ", minsta)
      print("Största talet: ", största)
      print("Medelvärde: ", summa/n)

5. from random import choice
   n = 0
   while n < 13:
      x = choice(range(10)) + 1
      if x <= 6:    # 6 av 10 möjlig utfall ger sannolikhet 60%
          print("1")
      if 6 < x < 8: # 1 av 10 ger 10%
          print(" x")
      if x >= 8:    # 3 av 10 ger 30%
          print("   2")
      n = n + 1

6. from random import choice
   while True:
      tal1 = choice(range(10)) + 1
      tal2 = choice(range(10)) + 1
      if (tal1 != tal2):
          print(tal1, tal2)
      else:
          # Visa inte slumptalen på skärmen. De är lika, vilket
          # är otillåtet om de väljs utan återläggning.
          print("Nytt försök!")

7. from random import choice
   gissat = 0
   hemligt = choice(range(12)) + 1
   while gissat != hemligt:
      gissat = int(input("Gissa vilket mitt hemliga tal är?")))
if gissat > hemligt:
    print("För stort")
if gissat < hemligt:
    print("För litet")
print("Rätt svar")

8. Se matematikbokens facit.

6 Problemgrupper


6.1 Slumpförsök

6.1.1 Allmänt

Den här problemgruppen omfattar uppskattningar av sannolikheter i slumpförsök med 1 eller flera steg, vilka båda är vanliga i sannolikhetsläran [Ref. 3]. Ett försök med flera steg kan beskrivas med ett träddiagram som har 1 nivå för varje steg. Ett exempel är en tärning som kastas 10 gånger.

6.1.2 Typisk algoritm

Figur 4 visar en typisk algoritm för att simulera och uppskatta sannolikheter i slumpförsök. Algoritmen är baserad på experimentell sannolikhet och kan sammanfattas "Uppskatta en sannolikhet $P(H)$ genom att simulera ett slumpförsök många gånger, räkna antalet gånger händelsen $H$ inträffar och beräkna den relativa frekvensen för $H$. Den relativa frekvensen är en uppskattning av $P(H)$ som blir bättre ju fler gånger slumpförsöket simuleras."
Figur 4: En typisk algoritm för ett slumpförsök.

Den typiska algoritmen är generell och en bra utgångspunkt för att lösa många av problemen i nästa kapitel, men du måste anpassa den till varje problem. Exempel 15 i Appendix A visar hur det kan göras.

6.1.3 Övningsuppgifter

Syftet med övningsuppgifterna är att du ska lära dig lösa några sannolikhetsproblem genom att använda den typiska algoritmen för slumpförsök. Använd även metoden i föregående kapitel och allt annat du lärt dig i häftet. Studera häftet, kopiera exemplen och återanvänd dina program.

1. Du kastar 1 tärning. Vad är sannolikheten att tärningen visar fler än 4 ögon?
   a) Bestäm sannolikheten (exakt) analytiskt med matematik.
   b) Uppskatta sannolikheten (närmevärde) experimentellt med papper, penna och en tärning.
   d) Ändra programmet, lägg till och räkna antal gånger händelsen ”fler än 4 ögon” inträffar.
   e) Ändra programmet och simulera tärningskastet 100 000 gånger.
   f) Ändra programmet, beräkna och skriv ut den relativa frekvensen för händelsen. Den relativa frekvensen är en uppskattning av sannolikheten.

2. Du kastar 1 tärning. Skriv ett program som simulerar och uppskattar sannolikheten för nedanstående händelser. Återanvänd och kopiera program, så går det fortare.
   a) H₁: Tärningen visar 6 ögon.
b) $H_2$: Tärningen visar inte 6 ögon.
c) $H_3$: Tärningen visar färre än 4 ögon.
d) $H_1$ eller $H_3$
e) $H_1$ och $H_3$

3. Du singlar 2 slantar. Vad är sannolikheten att du får 1 krona och 1 klave?
   a) Bestäm sannolikheten analytiskt.
   b) Skriv ett program som simulerar och uppskattar sannolikheten.

4. Du kastar 1 tärning 2 gånger efter varandra. Skriv ett program som simulerar och uppskattar sannolikheten att:
   a) tärningarna visar samma antal ögon.
   b) åtminstone en av tärningarna visar 6 ögon.
   c) ingen tärning visar fler än 3 ögon.
   d) det andra tärningskastet visar fler ögon än det första.

5. Du kastar 4 sjusidiga tärningar. Skriv ett program som simulerar och uppskattar sannolikheten att:
   a) Summan av tärningarnas ögon är 13.
   b) Summan är mindre än 8.
   c) Summan är större än 4 och mindre än 16.
   d) det första tärningskastet visar flest ögon.
   e) antalet ögon ökar i varje tärningskast.

6. Du singlar 1 slant. Skriv ett program som uppskattar P(krona) genom att simulerar slumpförsöket 10 gånger. Starta programmet 5 gånger och räkna ut ett medelvärde. Upprepa detta, men öka i tur och ordning antalet simuleringsar till 1000, 100 000 och 10 000 000. Vad gör du för observationer?

   a) Bestäm sannolikheten analytiskt.
   b) Skriv ett program som simulerar och uppskattar sannolikheten. Skriv och testa programmet i delar. Börja med att simulera ett tärningskast 20 gånger.
   c) Ändra programmet och räkna antal gånger tärningen visar fler än 4 ögon.
   d) Ändra programmet och räkna antal gånger händelsen ”tärningen visar fler än 4 ögon minst 5 gånger” har inträffat.
   e) Ändra programmet och simulera de 20 tärningskasten 100 000 gånger samt beräkna och skriv ut den relativa frekvensen för händelsen.


9. * Du kastar 1 tärning ett antal gånger. Skriv ett program som simulerar och uppskattar sannolikheten att tärningen visar 6 ögon minst 5 gånger om antalet tärningskast är:
   a) 10
   b) 50
   c) 100
10. ** Du kastar 1 tärning flera gånger tills summan av alla ögon tärningen visat är minst 12. Antalet tärningskast kommer då att variera mellan 2 och 12, vilket motsvarar att tärningen visar 6 ögon 2 gånger respektive 1 öga 12 gånger. Skriv ett program som simulerar och uppskattar sannolikheten för alla 11 möjliga utfall d.v.s. att antalet tärningskast som krävs är a) 2, b) 3, c) 4, d) 5, e) 6, f) 7, g) 8, h) 9, i) 10, j) 11 eller k) 12.


Facit (programmen är förslag)

1. a) 2/6

b) Något i närheten av 1/3. Kasta tärningen många gånger (n), notera antalet gånger 5 eller 6 ögon kommer upp (h) och beräkna relativa frekvensen h/n.

c) `from random import choice
ön = choice(range(6)) + 1`

d) `from random import choice
h = 0
ön = choice(range(6)) + 1
if ön > 4:
    h = h + 1`

e) `from random import choice
h = 0
n = 0
while n < 100000:
    ön = choice(range(6)) + 1
    if ön > 4:
        h = h + 1
    n = n + 1`

f) `from random import choice
n = 0
while n < 100000:
    ön = choice(range(6)) + 1
    if ön > 4:
        h = h + 1
    n = n + 1
f = h/n
print("Uppskattad P(H) =", f)`

2. a) 1/6

```
from random import choice
h = 0
n = 0
while n < 100000:
    ön = choice(range(6)) + 1
    if ön == 6:
```
h = h + 1
n = n + 1
f = h/n
print("Uppskattad P(H) =", f)

b) 5/6
Ändra rad 6 till: if ögon != 6:

c) 3/6
Ändra rad 6 till: if ögon < 4:

d) 5/6
Ändra rad 6 till: if (ögon == 6) or (ögon < 4):

e) 0/6
Ändra rad 6 till: if (ögon == 6) and (ögon < 4):

3. a) 2/4

b) from random import choice
h = 0
n = 0
while n < 100000:
    myntsida1 = choice(range(2)) + 1
    myntsida2 = choice(range(2)) + 1
    if myntsida1 != myntsida2:
        ögon = choice(range(6)) + 1
        if ögon == 6:
            h = h + 1
            n = n + 1
        f = h/n
        print("Uppskattad P(H) =", f)

4. a) 6/36

from random import choice
h = 0
n = 0
while n < 100000:
    ögon1 = choice(range(6)) + 1
    ögon2 = choice(range(6)) + 1
    if ögon1 == ögon2:
        h = h + 1
        n = n + 1
    f = h/n
    print("Uppskattad P(H) =", f)

b) 11/36
Ändra rad 7 till: if (ögon1 == 6) or (ögon2 == 6):

c) 9/36
Ändra rad 7 till: if (ögon1 <= 3) and (ögon2 <= 3):

d) 15/36
Ändra rad 7 till: if ögon2 > ögon1:
5. a) ca 0.075

```python
from random import choice
h = 0
n = 0
while n < 100000:
    ögon1 = choice(range(7)) + 1
    ögon2 = choice(range(7)) + 1
    ögon3 = choice(range(7)) + 1
    ögon4 = choice(range(7)) + 1
    summa = ögon1 + ögon2 + ögon3 + ögon4
    if summa == 13:
        h = h + 1
        n = n + 1
    f = h/n
print("Uppskattad P(H) =", f)
```

b) ca 0.014
Ändra rad 10 till: if summa < 8:

c) ca 0.45
Ändra rad 10 till: if 4 < summa < 16:

d) ca 0.18
Ändra rad 10 till:
if (ögon1 > ögon2) and (ögon1 > ögon3) and (ögon1 > ögon4):

e) ca 0.015
Ändra rad 10 till:
if ögon1 < ögon2 < ögon3 < ögon4:

6. Uppskatningen av P(krona) blir stabilare och noggrannare ju fler gånger försöket simuleras, men det tar längre tid innan programmet avslutas.

```python
from random import choice
h = 0
n = 0
while n < 10:
    myntsida = choice(range(2)) + 1
    if myntsida == 1: # 1 representerar krona
        h = h + 1
        n = n + 1
    f = h/n
print(”Uppskattad P(H) =”, f)
```

Ändra rad 4 och byt ut 10 mot 1000, 100 000 och 10 000 000 för att öka antalet simuleringar.

7. a) 1-[4/6]²⁰ (komplemethändelsen är ”tärningen visar aldrig fler än 4 ögon”)

b) from random import choice
```python
steg = 0
while steg < 20:
    ögon = choice(range(6)) + 1
    steg = steg + 1
```

d) from random import choice
från ögon = 0
steg = 0
while steg < 20:
    ögon = choice(range(6)) + 1
    if ögon > 4:
        från ögon = från ögon + 1
    steg = steg + 1
from random import choice
h = 0
från ögon = 0
steg = 0
while steg < 20:
    ögon = choice(range(6)) + 1
    if ögon > 4:
        från ögon = från ögon + 1
    steg = steg + 1
    if från ögon >= 1:
        h = h + 1
n = n + 1
f = h/n
print("Uppskattad P(H) =", f)

8. a) ca 0.013
from random import choice
h = 0
n = 0
while n < 100000:
    från ögon = 0
    steg = 0
    while steg < 20:
        ögon1 = choice(range(6)) + 1
        ögon2 = choice(range(6)) + 1
        if 3 < (ögon1 + ögon2) < 10:
            från 3 och 10 = från 3 och 10 + 1
        steg = steg + 1
    if från 3 och 10 == 15:
        h = h + 1
    n = n + 1
f = h/n
print("Uppskattad P(H) =", f)

9. a) ca 0.015
from random import choice
h = 0
n = 0
while n < 100000:
exakt6ögon = 0
steg = 0
while steg < 10:
    ögon = choice(range(6)) + 1
    if ögon == 6:
        exakt6ögon = exakt6ögon + 1
        steg = steg + 1
    if exakt6ögon >= 5:
        h = h + 1
        n = n + 1
f = h/n
print("Uppskattad P(H) =", f)
b) ca 0.093
Ändra rad 7 till: while steg < 50:
c) ca 0.99
Ändra rad 7 till: while steg < 100:

10. a) ca 0.028, b) ca 0.35, c) ca 0.38, d) ca 0.18, e) ca 0.049, f) ca 0.0088, g) ca 0.0010, h) ca 0.00010, i) ca 0.000003 j) ca 0 och k) ca 0 (exakt 1/6^12)

from random import choice
h = 0
n = 0
while n < 100000:
kast = 0
   summa = 0
while summa < 12:
    ögon = choice(range(6)) + 1
    summa = summa + ögon
    kast = kast + 1
if kast == 2:
    h = h + 1
    n = n + 1
f = h/n
print("Uppskattad P(H) =", f)

Sannolikheterna i deluppgift b) till k) uppskattas genom att ändra siffran 2 på rad 11 till 3, 4, 5, 6, 7, 8, 9, 10, 11 respektive 12.

11. a) ca 6

from random import choice
kast = 0
n = 0
while n < 1000000:
    ögon = 0
    while ögon != 2:
        ögon = choice(range(6)) + 1
        kast = kast + 1
        n = n + 1
print("Antal kast i genomsnitt =", kast/n)
7 Referenser

Appendix A: Exempel på program
Ett program kan skrivas på många olika sätt, så exemplen nedan är bara förslag.

Enkel funktionsräknare

Exempel 12: Beräkna och skriv ut två matematiska uttryck.

Singla slant

Pseudokod
Vänta på inmatning av ett x-värde.
Beräkna värdet av 2*x - 15.
Beräkna värdet av 1/(1 + x).
Visa y-värdena på skärmen.

Python 3
x = int(input("Ange ett heltal x? "))
y1 = 2*x - 15
y2 = 1/(1 + x)
print("y1 =", y1)
print("y2 =", y2)

Pseudokod
Simulera att en slant singlas.
Visa ”krona” eller ”klave” på skärmen.

Python 3
from random import choice
x = choice(range(2)) + 1
if x == 1:
    print("krona")
else:
    print("klave")
Exempel 13: Singla slant.

**Summera 100 tärningskast**

Pseudokod
Repetera 100 gånger:
Simulera ett tärningskast (x: antal ögon).
Visa ordningsnummer och x på skärmen.
Addera x till summan.
Visa summan på skärmen.

Python 3
```python
from random import *

summa = 0
n = 0
while n < 100:
    x = choice(range(6)) + 1
    print("Tärningskast", n, "=", x)
    summa = summa + x
    n = n + 1
print("Summa =", summa)
```

Exempel 14: Summera 100 tärningskast.

**Sannolikhet för 3 ögon vid ett tärningskast**

Pseudokod
Repetera många gånger:
Simulera ett tärningskast (x: antal ögon)
Öka en räknare om x = 3.
Uppskatta P(3 ögon) genom att beräkna och visa den relativa frekvensen på skärmen.

Python 3
```python
from random import choice

h = 0
n = 0
while n < 10000:
    x = choice(range(6)) + 1
    if x == 3:
        h = h + 1
    n = n + 1
f = h/n
print("Uppskattad P(H) =", f)
```

Kommentar:
Tillhör problemgruppen slumpförsök.
Appendix B: Mer felsökning

Du kommer att leta efter fel i dina program åtskilliga gånger. Det finns många metoder för felsökning varav en första presenterades i kapitel 1.3.4. Här presenteras ytterligare några metoder som kan hjälpa dig att förstå varför ditt program inte visar rätt svar på skärmen och hitta raden där felet finns:

- granska programmet
- lägg till spårutskrifter
- förenkla programmet

Granska programmet

Läs rad för rad, fundera och kontrollera ditt program omsorgsfullt. Några kontrollpunkter du kan använda är:

- Har du förstått problemet rätt och insett alla förutsättningar?
- Är algoritmen komplett och entydig eller finns det situationer du inte tänkt på?
- Är eventuell pseudokod och flödesschema korrekt översatta till Python 3?
- Fungerar programmet som du tror? Låtsas vara en dator och exekvera programmet rad för rad i ditt huvud och på papper. Fokusera på programmets alla variabler och hur deras värden förändras under exekveringen.
- Har en kamrat granskat ditt program? Att läsa och diskutera andra program är både effektivt och utvecklande.

Lägg till spårutskrifter


Förenkla programmet

Exempel 16: Ett modifierat och förenklat program med en spårutskrift.

Appendix C: Mer om Python och programmering

Det här häftet ger bara en mycket kortfattad introduktion till programmering och använder enbart några få delar av Python 3. Vill du lära dig mer, finns det ett flertal bra kurser, böcker, filmer, interaktiva hemsidor och annat på Internet. Det mesta materialet är gratis och skrivet på engelska, men delar finns översatt till svenska och andra språk:

Visuella och interaktiva kurser för nybörjare: www.hourofpython.com
En bra bok att fortsätta med: www.openbookproject.net/thinkcs/python/english3e
Python för både nybörjare och experter: www.python.org/about/gettingstarted
Hjälp, frågor och tips: www.python.org/about/help
Avancerade matematiska problem när du lärt dig mer: www.projecteuler.net


Ett program
n = 0
summa = 0
while n < 1000000:
    summa = summa + 3*n
n = n + 1
print("Summan=" summa)

Programmet modifierat
n = 0
summa = 0
while n < 10:
    summa = summa + 3*n
print(n, summa)
    n = n + 1
print("Summan=" summa)