Begreper

Program

Et sett med operasjoner som får en datamaskin til å utføre noe

Beskrivelse

Et program eller en app (applikasjon) er et sett med operasjoner som får en datamaskin til å utføre noe. Når dette skjer heter det at programmet *kjøres*. Når en mobiltelefon eller en datamaskin brukes kjøres alltid et eller annet program, for eksempel sms-appen til telefonen, eller Microsoft Word på datamaskinen. Det å lage et program heter å programmere eller å kode det og det kjøres ved at programmet lastes inn i arbeidsminnet til mobiltelefonen eller datamaskinen.

Arbeidsminne

Et sted i datamaskinen hvor programmer som kjøres lagres midlertidig

Beskrivelse

Når et program kjøres lastes det inn i arbeidsminnet til en datamaskin. Jo flere operasjoner et program har, desto større plass tar det i arbeidsminnet. Et annet navn på arbeidsminnet er RAM (Random Access Memory).

Programmere

Å lage et program

Beskrivelse

Det å lage et program er å programmerer eller å kode. Da settes ulike operasjoner sammen som får datamaskinen til å utføre noe. Operasjonene kan enten skrives i en tesktfil, tekstprogrammering eller settes sammen grafiskt gjennom blokker, blokkprogrammering. Blokkprogrammering kan være enklere å bruke når programmering læres og tekstprogrammering brukes gjerne for mer komplekse programmer. Det finnes flere ulike typer programmeringsspråk innen både blokkprogrammering, og tekstprogrammering.

Tekstprogrammering

Programmering hvor operasjoner skrives i tekstfiler

Beskrivelse

I tekstprogrammering består operasjoner av instruksjoner satt sammen av ulika ord og tall. Operasjonene utføres linje for linje fra topp til bunn av filen, og hver linje leses fra høyre mot venstre.

Det finnes to ulike typer tekstprogrammeringsspråk:

1. Tolkede språk: for eksempel Python eller JavaScript
2. Kompilerte språk: C eller C++

For tolkede språk kjøres tekstfilen gjennom en kommandotolk (*interpreter* eng.), da kalles filen ofte for et skript, eller kan filen gjøres om til et kjørbart program ved at den kompileres. Vanligvis lages skript bare når vi bruker tekstprogrammering, mens alle apper som installeres på en datamaskin er eksempler på kompilerte programmer.

Tekstprogrammering følger faste regler som bestemmes av hvilket programmeringsspråk du koder i. Disse reglene kalles for syntaksen til programmeringsspråket.

Kompilere

Lage et kjørbart program av en tekstfil

Beskrivelse

En eller flere tekstfiler blir kompilert når de blir oversatt til et kjørbart program. Dette programmet trenger da ikke noe annet programmet for å bli kjørt.

Blokkprogrammering

Programmering hvor operasjonene settes sammen grafisk gjennom ulike blokker

Beskrivelse

I blokkprogrammering består operasjoner av blokker som settes sammen grafisk i et eget program. Ofte er det lettere å lære seg å programmere gjennom å bruke blokkprogrammering, da betydningen til operasjonene kan visualiseres gjennom utseendet til de ulike blokkene.

Syntaks

Regler for hvordan operasjoner skrives i tekstprogrammering

Beskrivelse

I tekstprogrammering skrives alle operasjoner med bokstaver og tegn. Syntaksen til et tekst-programmeringsspråk er regler som bestemmer hvordan operasjonene kan skrives. På samme måte som syntaksen til skriftlig norsk bestemmer hvordan norsk kan skrives, for eksempel en setning starter med stor bokstav og slutter med et punktum, bestemmer syntaksen til tekstprogrammering hvordan operasjoner kan skrives i en tekstfil.

Syntaksen for å tildele to variabler hver sin verdi og å lage en kommentar er forskjellige for ulike programmeringsspråk. Under vises hvordan det ser ut for Python og JavaScript. Det er flere ting som er likt men også noe som skiller de to språkene.

Python-syntaks

```
# Alder og høyde til en person
alder = 9
høyde = 1.34
```

JavaScript-syntaks

```
// Alder og høyde til en person
let alder = 9;
let høyde = 1.34;
```

Syntaksfeil

En syntaksfeil er en programmeringsfeil som oppstår når en bruker skriver noe som ikke er lov i henhold til syntaksreglene. Program med syntaksfeil vil ikke kunne kjøres og man får som oftest informasjon om hvor i programmmet synktaksfeilen er.

```
print("Hei verden!"
```

Hvis denne pythonkoden kjøres vil føljende feilmelding vises:

```
SyntaxError: unexpected EOF while parsing
```

Meldingen sier først at det er en syntaksfeil og så hvorfor feilen oppsto.

Kommandotolk

Et program som kjører et skript.

Beskrivelse

Tolkede tekstprogrammeringsspråk trenger en kommandotolk for å kunne kjøres. Tekstfilene med programmet kalles da for skript. Kommandotolken til et pythonskript heter enkelt og greit `python`. Hvis Python er installert og tilgjengelig kan et pythonskript kjøres i en terminal gjennom å bruke kommandotolken `python` foran filnavnet til skriptet.

```
> python mitt-program.py
```

TODO: Definere terminal?

Programmeringsspråk

Et språk som beskriver hvordan operasjoner settes sammen i et program.

Beskrivelse

Når man programmerer må man velge et programmeringsspråk man skal bruke for å sette sammen de ulike operasjonene. Det finnes mange ulike programmeringsspråk som kan brukes til enten tekstprogrammering

eller blokkprogrammering. De finnes aller flest tekst-programmeringsspråk i verden da disse er enklere å bruke når man skal lage større program.

Eksempler på tekst-programmeringsspråk som trenger en kommandotolk

- Python - brukes for eksempel til å regne matematikk
- JavaScript - brukes for eksempel til å styre en nettside

Eksempler på tekst-programmeringsspråk som må kompileres

- C - brukes for eksempel til å programmere kjernen i de fleste operativsystemene
- C++ - brukes for eksempel til å programmere dataspill

Eksempler på blokk-programmeringsspråk

- Scratch - brukes for eksempel til å programmere enkle spill
- Trinket-block - brukes for eksempel til å lære seg programere
- Blokkuino - brukes til å styre en arduino

Algoritme

En liste med operasjoner som utfører en oppgave.

Beskrivelse

En algoritme er en liste med operasjoner som utfører en oppgave. Oppgaven kan være en matematisk oppgave som å beregne et gjennomsnitt eller å finne typetallet (tallet med høyest frekvens i en tallmengde). Oppgaven kan også være noe hverdagslig som å velge den beste oppskriften i en samling med oppskrifter.

Listen med operasjoner må komme i en gitt rekkefølge, for eksempel er følgende to operasjoner fra en oppskrift:

- blande ingrediensene sammen
- sett inn formen i ovnen Her er det opplagt at man må blande ingrediensene i en oppskrift før man setter inn formen i ovnen, men for en som starter å lære seg programmering kan det være vanskelig å forstå i hvilken rekkefølge operasjonene i en algoritme skal skrives.

En algoritme kan være noe konkret, som noen linjer med kode som utfør oppgaven eller en abstrakt beskrivelse av hvilke operasjoner som skal utføres og i hvilken rekkefølge de skal gjøres. TODO: Gi eksempler eller ta bort

Mange internettbedrifter bruker algoritmer når de skal presentere varer eller artikler til deg. For eksempel bruker strømmetjenester som Netflix og Spotify slike når de skal presentere filmer eller musikk de mener passer deg. Disse algoritmene baserer seg på at bedriftene husker hva du har valgt tidligere og hva andre personer som likner deg (samme alder, kjønn, bosted) har valgt.

Eksempel: gjennomsnitt

- Lag en list-variabel `alle_tall` hvor du samler alle tall du vil beregne gjennomsnittet på.
- Summer alle tallene i `alle_tall` og lagre resultatet i variabelen `sum`.

- Divider verdien i `sum` med antallet tall i `alle_tall` og lagre verdien i variabelen `gjennomsnitt`

```
alle_tall = [4, 5, 3, 5, 6, 4, 3, 5]
sum = 0
for tall in alle_tall:
    sum += tall
gjennomsnitt = sum/len(alle_tall)
```

Operasjon

En enkelt ting en datamaskin skal gjøre

Beskrivelse

En operasjon forteller datamaskinen at den skal *gjøre noe*. Et program er satt sammen av mange operasjoner etter hverandre. En operasjon er enkelt fortalt, det å *gjøre noe* med en eller flere verdier, å bruke kontrollstrukturer eller å brukefunksjoner.

Det finnes 4 ulike hovedvarianter av operasjoner som gjør noe med verdier:

1. Bearbeider en verdi (*process* eng.)
2. Lagre en verdi i en variabel
3. Hente inn en verdi (inndata), input
4. Vise en verdi (utdata), output

Bearbeide verdier

Når et program bearbeider en eller flere verdier gjøres det noe med verdiene. To ulike måter å bearbeide verdier på er å bruke en operator for eksempel `5 + 6` eller å bruke verdiene som argumenter til en funksjon for eksempel `gjennomsnitt([3, 6, 8])`.

Flere operasjoner som bearbeider verdier kan settes sammen til uttrykk for eksempel:

```
round(sqrt(3)/2, 3)
```

Lagre verdi

Verdier lagres i arbeidsminnet som variabler og kan deretter brukes senere i programmet.

Når en verdi lagres i en variabel brukes en tilordningsoperator (`=`).

```
alder = 9
navn = "Marie"
```

Her lagres to ulike verdier: `9` og `"Marie"`, til variablene: `alder` og `navn`.

Hente en verdi, input

En datamaskin hadde vært ganske verdiløs hvis den ikke kunne hente verdier enten fra en bruker eller fra et annet sted, for eksempel en avstandssensor på en bil.

```
navn = input("Hva heter du?")
```

Her brukes funksjonen `input` til å spør brukeren om å skrive inn sitt navn. Programmet stopper opp og venter på at brukeren skal skrive inn sitt navn.

Vise en verdi, output

En verdi kan vises til en bruker for eksempel ved at den vises på skjermen, eller skrives til en fil.

```
print("Jeg er 13 år!")
```

Her sendes verdien `"Jeg er 13 år!"` til funksjonen `print` som så viser den på skjermen.

Kontrollstruktur

Bestemmer hvilke deler av et program som skal utføres

Beskrivelse

Vanligvis utføres operasjonene i et program fra toppen av tekstfilen og nedover. Hver linje i tekstfilen utføres en gang. Kontrollstrukturer består av if-setninger og løkker og de kan bestemme at noen linjer skal utføres og andre ikke, eller at noen linjer skal utføres mange ganger. Slik bestemmer kontrollstrukturer flyten i et program.

Uttrykk

En eller flere operasjoner som er satt sammen og som returnerer en verdi

Beskrivelse

Et uttrykk (*expression* eng.) består av en eller flere operasjoner som er satt sammen. Alle enkeltoperasjoner i et uttrykk resulterer i en verdi.

```
navn = "Petter"
setning = "Jeg heter " + navn + " og jeg er " + 16 + " år."
```

Her er to linjer som begge tilordner to variabler verdier. Verdiene er begge resultater av to uttrykk. Den første linjen består av et uttrykk som resulterer i verdien: `"Petter"`. Den andre linjen består av et uttrykk som resulterer i verdien `"Jeg heter Petter og jeg er 16 år."`.

Variabel

Er et navngitt sted i arbeidsminnet som lagrer verdier.

Beskrivelse

En variabel er et navngitt sted i arbeidsminnet som lagrer en eller flere verdier i et program. Navnet til variabelen brukes så i uttrykk for å representere verdien til variabelen.

Du bestemmer selv hva navnet til en variabel skal være, men syntaksen til programmeringsspråket legger begrensninger på hva et variabelnavn kan være. Når man lager en variabel for første gangen sier man at man definerer den og man kan ikke bruke en variabel før den er definert. Navnet til en variabel brukes når du:

1. definerer variablen
2. bruker verdien til variablen
3. endrer verdien til variablen.

Disse tre måtene kan illustreres med følgende eksempel:

```
lån = 500000.
rente_prosent = 0.05
rente = lån*rente_prosent
lån = lån + rente
```

Her defineres første tre variabler: `lån`, `rente_prosent` og `rente`. På tredje rad brukes variabelen `lån` og `rente_prosent` i et uttrykk for å regne ut hva renten skal være. På siste rad endres verdien til variabelen `lån` ved at renten legges til det første lånet.

Variabel i matematikken

Ordet variabel brukes også i algebra i matematikken. Da betegner det ofte en ukjent tallverdi, for eksempel *x, y*. Innen programmering har en variabel *alltid* en verdi og representerer altså ikke noe ukjent.

Variabelnavn

Navnet til en variabel.

Beskrivelse

En variabel har altid et navn. Syntaksen til et programmeringsspråk legger noen krav til navnet.

For Python gjelder disse kravene til et variabelnavn

1. kan inneholde bokstaver, siffer og understrek.
2. kan IKKE inneholde mellomrom , bindestrek `-` eller punktum `.`: Bruk heller `_` mellom ord i variabelnavnet.
3. kan IKKE starte med et siffer.
4. må være unikt. Flere variabler kan altså ikke ha samme navn.

Tips til variabelnavn

Bruk variabelnavn som beskriver hva verdien til variabelen skal brukes til. For eksempel er navn som `a`, `b` oftest dårlige navn, mens `lån`, `navn` eller `poengsum` er bra navn.

Verdi

En konkret representasjon av data

Beskrivelse

En verdi representerer en størrelse som kan manipuleres av et program. Verdier er ofte det man til daglig kaller data. En datamaskin er altså en maskin som bruker verdier! Verdier har ulike datatyper som beskriver hva verdien kan brukes til:

Eksempler på datatyper er:

- tall: `2, -5, 0.45, 1e-3`
- tekst eller strenger: `"A", "Jeg heter Lise."`
- boolean: `True, False`
- liste: `[5, 3, 8], ["ost", 2, "melk", 5]`

Datatype

Beskriver hva en verdi kan brukes til

Beskrivelse

Alle verdier har en datatype som beskriver hva den kan brukes til. De vanligste datatypene er:

- tall: Brukes i all matematikk, se for eksempel [aritmetiske operatorer](#aritmetisk operatorer).
- strenger: Brukes til å representere tekst-verdier.
- boolean: Brukes til å beskrive om noe er *sant* (`True`) eller *falskt* (`False`).
- lister: Brukes å samle verdier i en ordnet rekkefølge.
- [assosiative lister](#assosiativ liste). Brukes til å samle verdier som er assosiert med en nøkkel.

Definere

Operasjon som lager en variabel eller funksjon.

Beskrivelse

For å kunne bruke en variabel eller funksjon må de først defineres. Når en variabel defineres knyttes et variabelnavn til en verdi som lagres i arbeidsminnet. Når en funksjon defineres knyttes et funksjonsnavn til et sett med operasjoner. Navnet brukes så i programmet for å representere enten verdien til variabelen eller alle operasjonen til funksjonen.

I tillegg til å få et navn må en variabel også bli tilordnet en verdi (tilordnes) når den defineres. I noen programmeringsspråk kan man si ifra at man skal bruke en variabel uten å gi den en verdi. Da heter det å *deklarere* en variabel.

Tilordne

Gir eller endrer en verdi til en variabel

Beskrivelse

Når en verdi lagres i en variabel heter det at denne variabelen tilordnes en verdi. Dette gjøres med en tilordningsoperator, som i Python er `=`. På venstre side av tilordningsoperatoren må variabelens navn skrives. På høyre side må det stå en verdi eller et uttrykk som resulterer i en verdi. Denne verdien blir så lagret i variabelen.

Tall

Datatype som brukes til numeriske verdier

Beskrivelse

Verdier av datatypen tall brukes i programmering til utføre operasjoner som vi i hverdagen tenker på som matematiske operasjoner. Når to verdier skal legges sammen gjennom addisjon må disse verdiene for eksempel ha datatypen tall.

En del programmeringsspråk for eksempel Python skiller på datatypen heltall og flyttall. Flyttal brukes til å reprsentere tall med desimaler. Andre programmeringsspråk, som for eksempel JavaScript eller Scratch, skiller i utgangspunktet ikke på heltall eller flyttal.

Heltall

Datatype som brukes til å representere hele tall

Beskrivelse

Heletall er tall som ikke har desimaler. Heltall brukes for eksempel til å representere et antall for eksempel antallet spiller i et spill eller en posisjon i en rekkefølge. Heltall er mer naturlig for en datamaskin å arbeide med enn flyttal. Det er fordi alle heltall kan representeres med et binært tall som er tall satt sammen av sifrene 1 og 0.

I programmeringsspråket Python heter datatypen for helttall `int`, som er kort for integer og er engelsk for heltall. Funksjonen `int()` kan brukes til å lage et heltall fra for eksempel en streng.

```
tall_streng = input("Skriv inn ett tall:")
tall_int = int(tall_streng)
```

I eksemplet over hentes en streng inn fra brukeren og så blir den gjort om til et heltall med `int()` funksjonen. Hvis brukeren skriver en streng som *ikke* går ann å gjøre om til et heltall sier kommandotolken ifra ved å reise en programmeringsfeil.

Flyttall

Datatype som brukes til å representere desimaltall

Beskrivelse

Et flyttal er et desimaltall. Innen all programmering bruker man `"."` istedefor `","` for å angi desimalen i et desimaltall eller flyttal.

Ordet flyttal henspiller på måten tallet er representert i arbeidsminnet til en datamaskin. Det som er viktig å nevne her er at flyttal ikke representeres eksakt i arbeidsminnet. For eksempel består desimaltallet $\sqrt{2}$ av uendelig mange desimaler og må i en datamaskin representeres på en avrundet og unøyaktig måte. I Python er $\sqrt{2}\simeq 1.4142135623730951$.

Flyttal er vanskeligere å representere eksakt for en datamaskin enn heltall. Dette er fordi en datamaskin i utgangspunktet bare forstår binære tall. Heltall kan representeres ved binære tall, mens flyttall må representeres på en annen måte. For eksempel må alle desimaltall representeres med et endelig antall desimaler

Ordet *flyt* (engelske *float*) kommer fra at plasseringen av desimaltegnen flyter. Vi kan for eksempel tenke oss at `0.03` kan representers med heltallet 3 hvor desimaltegnet er flyttet 2 plasser. Hvordan et flyttal faktisk representeres på går vi ikke inn på her men i Python kan dette med at flyttall er vanskelige å representer vises med følgende eksempel:

```
print(3*0.1)
```

Dette resulterer i utskriften: `0.30000000000000004` som nesten er `0.3` men det *er* ikke eksakt `0.3`.

Programmeringsfeil

En feil som oppstår når et program gjøre en feil

Beskrivelse

Det er vanlig å skrive program med feil (*bug* eng.) i seg. Det finnes ulike typer programmeringsfeil og her deler vi inn i to ulike typer:

1. Syntaksfeil: Når kode bryter mot syntaksen til et tekstprogrammeringsspråk.
2. Logiske feil: Når et program gjør noe som ikke gir mening. Når et feil oppstår er det viktig å kunne feilsøke (*debug* eng.).

Feilsøke

Finne programmeringsfeil i et program

Beskrivelse

Å feilsøke (*debug* eng.) et program er å prøve å finne programmeringsfeil. Dette kan ofte være frustrerende og tidkrevende, men en enkel og god læringsalgoritme for å lære seg programmering er å:

1. prøve programmere noe
2. gjøre feil
3. finn feilene

4. rette feilene Det finnes ulike strategier for å feilsøke. TODO: Forklare ulike strategier for feilsøking.

Binære tall

Tall som representeres med sifferene 1 og 0

Beskrivelse

Binære tall er tall som representerers med sifferene 1 og 0 for eksempel 1011 er et binært tall. For å regne ut verdien til tallet ganger vi sifferverdien med plassverdien, akkurat som vi gjør med *vanlige tall* fra titall systemet. Plassverdien til de 4 første posisjonene i et binært tall er: 8, 4, 2, 1. Verdien til 1011 kan vi derfor regne ut til å være: $1\cdot8 + 0\cdot4 + 1\cdot 2 + 1\cdot 1 = 11$. Inne i en datamaskin lagres **allt** som binæretall. Dette er fordi en datamaskin består av veldig mange strømbrytere som enten kan være av (0) eller på (1). TODO: Skal vi skrive mer dette temaet eller vise til annen kilde!

Streng

Datatype som brukes til å representere tekst-verdier

Beskrivelse

Innen programmering kalles en tekst for en streng (*string* eng.). Ordet kommer fra at man lagrer en *streng* med enkeltbokstaver. Mange ganger kan det være nyttig å tenke på en streng som en liste med enkelt bokstaver (*characters* eng.) men til forskjell fra lister kan ikke enkeltelementene endres uten å lage en ny streng.

Strenger kan legges sammen med streng-operatoren +. Denne operasjonen kalles da å konkatenere to strenger, men oftest sier man bare å *legge sammen* to strenger.

```
fornavn = "Elise"
etternavn = "Sandberg"
print("Eleven heter " + fornavn + " " + etternavn)
```

Det finnes flere ulike operasjoner som kan gjøres på og med strenger... TODO: Legge til noe flere operasjoner.

Boolean

Datatype som brukes til å representere sanne og falske verdier

Beskrivelse

Verdier av datatypen boolean kalles boolske verdier og de brukes til å representere sanne og falske verdier. Man bruker boolske verdier til å svare på betingelser eller til å representere svar på *ja- og nei-spørsmål*. Boolske verdier er også resultatet av logiske operatorer.

Liste

En samling av verdier som er ordnet i en rekkefølge

Beskrivelse

En liste (*array* eng.) brukes når du har mange verdier som hører sammen. Verdiene i en liste kalles elementer og er ordnet i en rekkefølge. Verdien til hvert element hentes ut fra listen gjennom å bruke nummeret i rekkefølgen elementet har i listen. Dette nummeret kalles for indeksen til elementet. Du kan lagre listen i en variabel slik at du kan bruke verdiene seinere i et program.

```
deltagere = ["Rebecca", "Erik", "Selma", "Amanda"]
deltager = deltagere[2]
deltagere[1] = "Svein"
```

`deltagere` er en list-variabel med fire elementer. Hvert element representerer navnene til deltagerne i en konkuranse. `deltager` er en variabel som har det *tredje* (ikke det andre!) elementet fra listen. Til slutt endres verdien til det andre elementet til `"Svein"`. Deltager `"Erik"` er altså byttet ut med `"Svein"`.

Element - liste

Verdiene i en liste kalles for elementer

Indeks - liste

En indeks brukes når man skal hente ut eller endre på elementene i en liste. Indeksen betegner plasseringen til verdien i listen og må alltid angis med et heltall (flyttal virker ikke, da plassering 2.59 ikke gir mening). Det første elementet har indeksen `0`, det andre har indeksen `1` og så videre. På rad to over brukes indeksen `2` til å hente ut verdien til det tredje elementet i listen `deltager`. På rad tre over brukes indeksen `1` til å endre det andre elementet i listen til verdien `"Svein"`.

Assosiativ liste

En samling verdier som hver er assosiert med en nøkkel

Beskrivelse

En assosiativ liste (*maps*, *dictionaries* eng.) brukes når du har mange verdier som hører sammen og hvor hver verdi knyttes til en (nøkkel). For eksempel kan man lagre verdier knyttet til en elev i en assosiativ liste hvor nøklene kan være `fornavn`, `etternavn`, `klasse` og `resultater`. Et element i en assosiativ liste er altså et nøkkel - verdi par hvor nøkkelen altid er en streng og verdien kan ha en vilkårlig datatype.

Assosiative lister finnes i de fleste programmeringsspråk men heter da gjerne noe annet.

- python: `dict` - kort for *dictionary*
- JavaScript: `Object`
- Scratch: Det finnes ikke assosiative lister i Scratch (enn?!)

```
elev = {
    "fornavn": "Salma",
    "etternavn": "Eriksen",
    "klasse": "8A",
```

```
    "resultater": [4, 5, 3, 6, 5, 4]
}

# Hente ut element-verdier
fornavn = elev["fornavn"]
resultater = elev["resultater"]

# Endre element-verdier
elev["klasse"] = "9A"
```

Element - assositativ liste

Et element i en assositativ liste er en nøkkel - verdi par. Nøkkelen sier noe om hva verdien er (tenk variabelnavn) og verdien er en verdi som er akseptert i programmeringsspråket. I eksemplet over ser vi at verdiene er 3 strenger og en liste.

Operatorer

Enkelt tegn som bearbeider en eller flere verdier og returnerer en verdi

Beskrivelse

En operator er et enkelt tegn som brukes til å bearbeide en eller to verdier. Operatorer er ofte en kilde til misforståelser når man skal lære seg programmering fordi de likner tegn som brukes i hverdagsspråk og matematikk, og i tillegg kan de ulike operatorene likne hverandre.

Likt for alle operatorer er at de bearbeider (opererer på) en eller flere verdier og de returnerer *alltid* en verdi. Addisjons-operatoren `+` opererer på to tall og returnerer resultatet av addisjonen. For eksempel `3 + 4` adderer verdiene `3` og `4` og returnerer `7`. Addisjons-operatoren er en aritmetisk operator.

Det finnes flere ulike typer operatorer. Disse skilles ved at de bearbeider og returnerer ulike typer verdier:

- aritmetiske operatorer: bearbeider og returnerer tall-verdier.
- strengoperatorer: bearbeider streng og tall-verdier og returnerer nesten alltid streng-verdier.
- logiske operatorer: bearbeider og returnerer boolske-verdier.
- sammenligningsoperatorer: bearbeider ulike typer verdier men returnerer alltid en boolsk-verdi.
- tilordningsoperatorer: tilordner en variabel som er på venstre side av operatoren en verdi.

Operator-rekkefølge

Operatorer kan brukes til å sette sammen uttrykk: `7-2*3+2`. For at slike sammensatte uttrykk skal evalueres entydig finnes det regler for i hvilken rekkefølge operatorene skal utføres i (*operator precedence* eng.). I uttrykket `4-2*3+2` brukes de tre aritmetiske operatorene `-`, `+` og `*` og de utføres i samme rekkefølge som man gjør i matematikken. Hvert programmeringsspråk har egne regler for hvilke rekkefølge operatorer utføres i. Disse bør en avansert programmererer kjenne til. Rekkefølgen for de aritmetiske operatorene er de samme for de aller fleste programmeringsspråk:

1. parentser
2. potenser
3. multiplikasjon og divisjon

4. addisjon og subtraksjon Når to operatorer som har samme rekkefølge kommer etter heverandre i et uttrykk utføres operatorene fra venstre til høyre. `7-2*3+2` som: 1.`7-2*3+2`: Utfører ganger-operatoren 2.`7-6+2` 3.`1+2` 4.`3` Merk at hvis ikke operatorene utføres fra venstre til høyre ville for eksempel rad 2 bli noe annet: `7-6+2` $\rightarrow$ `7-8` som blir `-1`.

Unære og binære operatorer

En operator som bearbeider to verdier kalles en binær operator og en operator som bearbeider en verdi kalles en unær operator. `+` er et eksempel på en binær aritmetisk operator da den brukes til å legge sammen to verdier. `-` er et eksempel på en aritmetisk operator som både kan være unær og binær. `-` er unær når den representerer et negativt fortegn: `-5`, og binær når den representerer substraksjon: `7-4`.

Vanlige misforståelser ved bruk av operatorer.

- Tilordningsoperator må stå til venstre for et uttrykk. Man kan altså ikke skrive `4*6=x`
- Tilordningsoperatoren blandes ofte sammen med likhets operatoren
- `4a` betyr ikke `4*a`. Mener du variabelen `a` ganger 4 skriver du det eksplisit ved å bruker multiplikasjons-operatoren `*`.

Aritmetiske operatorer

Tegn som bearbeider en eller to tall-verdier

Beskrivelse

Inne programmering representeres de fire regneartene ved aritmetiske operatorer. Disse bearbeider da bare tall-verdier og returnerer et annet tall-verdi. I tilegg til operatorer for de fire regneartene finnes også andre aritmetiske operatorer:

- `+`: addisjon; `5+2` resulterer i `7`
- `-`: subtraksjon (binært) og minus fortegn (unært); `5-2` resulterer i `3`.
- `*`: multiplikasjon; `5*2` resulterer i `10`.
- `/`: divisjon; `5/2` resulterer i `2.5`

Operatorene over finnes i både Python og Scratch. Innen Python har vi i tilegg disse aritmetiske operatorene:

- `**`: potens; `5**2` resulterer i `25`.
- `//`: heltalsdivisjon; `5//2` resulterer i `2`.
- `%`: modulus; rest ved heltalsdivisjon, `5%2` resulterer i `1`.

Strengoperatorer

Tegn eller funksjoner som bearbeider streng-verdier og returnerer en streng

Beskrivelse

Strengoperatorer brukes til å bearbeide strenger og da det er en operator returnerer den alltid en annen streng.

Logiske operatorer

Tegn eller ord som bearbeider boolske verdier og returnerer en boolsk verdi

Beskrivelse

Den enkleste måten å bearbeide en eller to verdier er å bruke en operator.

Sammenligningsoperator

Tegn som sammenligner to verdier og returnerer en boolsk verdi

Beskrivelse

En sammenligningsoperator sammenligner to verdier. Basert på verdiene og type operator returneres så resultatet av sammenligningen som en boolskverdi: `true` eller `false`.

Det finnes 5 ulike sammenligningsoperatorer:

`==`: likhet; `4==6` returnerer verdien `false` da 4 ikke er lik 6. `<`: mindre enn; `4<6` returnerer verdien `true` da 4 er mindre enn 6. `<=`: mindre enn eller lik; `7<=7` returnerer verdien `true` da 7 er mindre enn eller lik 7. `<`: større enn; `4>6` returnerer verdien `false` da 4 ikke er større enn 6. `>=`: større enn eller lik; `7>=4` returnerer verdien `true` da 7 er større enn eller lik 4.

Tilordningsoperatorer

Tegn som brukes til å tilordner en variabel en verdi

Beskrivelse

En variabel tilordnes en verdi gjennom en tilordningsoperator. Når en verdi får en ny verdi brukes

Er man ny til programmering blandes dette tegnet ofte sammen med likhetstegnet fra matematikk som tilsynelatende er det samme. I matematikk betyr likhetstegnet at det som står på venstre side om likhetstegnet er lik det som står på høyre side.

Funksjon

En funksjon lagrer en eller flere operasjoner

Beskrivelse

En funksjon brukes til å lagre et sett med operasjoner til et navn. Operasjonene som lagres i en funksjon må utføre en spesifikk oppgave slik at man bruker funksjonens navn istedenfor alle operasjonene når programmet skal utføre oppgaven. Når funksjonen brukes heter det at man kaller funksjonen. Når en funksjon blir kallet blir alle operasjonene til funksjonen utført og slik blir også oppgaven til funksjonen utført.

En funksjon likner på en variabel med at den har et navn og noe blir lagret til navnet, men en funksjon lagrer operasjoner hvor en variabel lagrer verdier. TODO: Legg in noen eksempler

Funksjonsnavn

Navnet til en funksjon

Beskrivelse

En funksjon har nesten altid et navn. Syntaksen til et programmeringsspråk legger noen krav til navnet. Disse krav er de samme som for variabelnavn.

Tips til funksjonsnavn

Bruk funksjonsnavn som beskriver hva funksjonen skal gjøre. Ettersom en funksjon utfører noe er det altid lurt å bruke et verb som første ord i navnet. For eksempel er `beregn_nytt_lån` eller `hent_resultater` gode navn til to funksjoner.

Kalle funksjon

Utføre operasjoner knyttet til en funksjon

Beskrivelse

En funksjon lagrer operasjoner og når disse skal utføres må funksjonen kalles. Andre ord som ofte brukes for dette er å utføre funksjonen eller å kjøres den. Innen tekstprogrammering kalles en funksjon gjennom å skrive to parenteser etter funksjonsnavnet.

```
#
```

Argument

Verdier som overføres til en funksjon når den kalles

Beskrivelse

Returverdi

Beskrivelse

Kommentar

Tekst som ikke utfører noen operasjoner

Beskrivelse

Valg

Ulik kode kjøres på grunnlag av en betingelse

Beskrivelse

Et program tar et valg når ulik kode kjøres på grunnlag av en betingelse. Innen programmering tas valg gjennom en if-setning.

If-setning

En kontrollstruktur som gjør et valg på grunnlag av en betingelse

Beskrivelse

En if-setning er en kontrollstruktur som kan brukes til å ta et valg. Valget gjøres på bakgrunn av verdien til en betingelse. Et valg kan være å utføre en blokk med kode hvis en bruker har gjetter korrekt tall (betingelsen er da sann) i en gjettelek, og en kodeblokk hvis brukeren gjetter feil tall.

```
if gjettet == korrekt:
    print("Du gjettet riktig :)")
else:
    print("Du gjettet feil :(")
```

En if-setning kan brukes for å ta ett valg, istedenfor to som i eksemplet over. Da sløyfes bare `else` setningen.

```
pris = 30
if alder < 15:
    pris = 15
print("Du skal betale {pris} kr".format(pris))
```

En if-setning kan brukes til å ta flere enn to valg også. Da legges flere betingelser inn gjennom `elif` setningen.

```
if alder < 4:
    pris = 0
elif alder < 15:
    pris = 15
elif alder < 63:
    pris = 30
else:
    pris = 15
print("Du skal betale {pris} kr".format(pris))
```

Kodeblokk

Linjer med kode som hører sammen

Beskrivelse

En kodeblokk er et sett med linjer som hører sammen. I python brukes mellomrom, eller så kallte innrykk, i starten på hver linje for å vise et kodeblokk. Linjer som har samme innrykk i python tilhører samme kodeblokk.

```
for i in range(1,11):
    # Dette er en kodeblokk
    # En kodeblokk kan gå over flere linjer
```

```
    print("Jeg tilhører kodeblokk 1")
    if i < 5:
        print ("Jeg tilhører kodeblokk 2")
    else:
        print ("Jeg tilhører kodeblokk 3")
```

I andre tekst-programmeringsspråk, som for eksempel C++, Java, JavaScript, brukes krøllparenteser `{<kodeblokk>}` til å avgrense en kodeblokk.

Innrykk

Mellomrom først i en kodelinje

Beskrivelse

Innrykk er et mellomrom før de første tegnene i en kodelinje. Mellomrommet kan bestå av enten tegnet *mellomrom* eller av tegnet *tab*. Innrykk brukes i de fleste programmeringsspråk til å gjøre kode bedre lesbar. I Python brukes innrykk til å avgrense en kodeblokk

Betingelse

Noe som må være sant for at kode skal utføres

Beskrivelse

I programmering brukes betingelser blandt annet til å ta valg. En betingelse er noe som enten har verdien sant eller verdien falskt. Her er noen måter en betingelse kan lages på i python.

```
# Sammenligning
if alder <= 12:
    print("Du er et barn!")

# Kombinerte sammenligninger
if 13 <= alder and alder < 20:
    print("Du er en tennåring!")

# Sjekker om en verdi er i en liste
if navn in ["Ole", "Dole", "Doffen"]:
    print("Du er en nevø av Donald!")

# Sjekker verdien til en variabel
if ferdig:
    print("Der var du ferdig!")
```

Løkke

Kontrollstruktur som gjentar en kodeblokk

Beskrivelse

En løkke er en kontrollstruktur som gjentar en kodeblokk null til flere ganger. I python finnes det to ulike typer løkker:

1. for-løkke: Brukes når man vet hvor mange ganger en kodeblokk skal gjentas og løkken har alltid en tellevariabel
2. while-løkke: Brukes når gjentagelsen av kodeblokken er avhengig av en betingelse

Nestede løkker

To eller flere løkker som er plassert inni hverandre

Beskrivelse

Med nestede løkker plasseres en løkke inne i en annen løkke og man får da en yttre og en indre løkke. Den indre løkken vil gjentas sammen med den yttre kodeblokken.

```
for i in range(1, 11):
    # Yttre kodeblokk
    for j in range(1, 11):
        # Indre kodeblokk
        print(i*j)
```

Hver av de to løkkene vil her gentas 10 ganger, men da den indre løkken inngår i den yttre kodeblokken, vil den indre kodeblokken totalt gjentas: 10x10=100 ganger.

for-løkke

En løkke som bruker en tellevariabel til å gjentar en kodeblokk et gitt antall ganger og som bruker en tellevariabel

Beskrivelse

En for-løkke brukes til å gjenta en kodeblokk når man vet hvor mange ganger kodeblokken skal gjentas. Spesielt for en for-løkker er at den alltid bruker en tellevariabel som oppdateres for hver gang en kodeblokk gjentas.

To eksempler hvor en for-løkke brukes er når

1. en enkel tallfølge skal lages, og
2. en liste med verdier skal gås igjennom

for-løkke med tallfølge

I de tre følgende eksempelene brukes funksjonen `range` til å lage tre ulike tallfølger.

```
# Tellevariabelen i får tallverdiene 0, 1, 2, 3, 4
for i in range(5):
    print("tall:", i)
```

Får `range` et tallverdi som argument, her 5, lages en tallfølge som starter på 0 og går opp til (men ikke inklusive) 5, og hvert tall er 1 større enn det forrige.

```
# Tellevariabelen j får tallverdiene 5, 6, 7, 8, 9
for j in range(5, 10):
    print("tall:", j)
```

Får `range` to tallverdier som argument, her 5 og 10, lages en tallfølge som starter på 5 og går opp til (men ikke inklusive) 10, og hvert tall er 1 større enn det forrige.

```
# Tellevariabelen k får partallverdiene 0, 2, 4, 6, 8
for k in range(0, 10, 2):
    print("partall:", k)
```

Får `range` tre tallverdier som argument, her 0, 10 og 2, lages en tallfølge som starter på 0 og går opp til (men ikke inklusive) 10, og hvert tall er 2 større enn det forrige.

for-løkke med en liste

Tellevariabel

Variabel som oppdateres for hver gang en kodeblokk i en løkke gjentas

Beskrivelse

while-løkke

En løkke som gjentar en kodeblokk så lenge en betingelse har verdien True

Beskrivelse

Modul

Funksjonalitet som må importeres for å brukes

Beskrivelse

En modul eller bibliotek inneholder funksjoner eller verdier som kan brukes innen et spesifikt område. For eksempel gir modulen `math` i python tilgang på en del matematiske funksjoner, for eksempel: `sin()`, `exp()` og `ceil()` og tallverdier: `pi` og `e`. For å bruke en modul må den første importeres.

```
# Importerer hele module math
from math import sin, exp, ceil, pi, e

if exp(1) == e**1:
    print("exp(1) er det samme som e**1")
```

```
if ceil(pi) == 4:
    print("ceil avrunder et flyt-tall til nærmeste heltall over tallet.")
```

Importere moduler

Få tilgang på operasjoner fra en modul

Beskrivelse

Når en modul eller bibliotek importeres får en tilgang på funksjoner og verdier, knyttet til en spesifikk funksjonalitet. De fleste programmeringsspråk kommer med ett sett med grunnleggende operasjoner. Vil du for eksempel generere et tilfeldig tall må man i python importere denne operasjonen fra modulen random.

I python kan moduler importeres på flere ulike måter.

```
# Importere bare funksjonen randint fra modulen random
from random import randint

# Importere alle funksjonen fra modulen random
from random import *

# Importere hele modulen men beholder funksjonene i modulen
import random
random.randint(1, 10)

# Importere hele modulen men gir den et kortere navn
import random as r
r.randint(1, 10)
```

Det er å foretrekke å importere bare de funksjoner man trenger for eksempel `from random import randint` og å unngå å importere alle funksjoner som `from random import *`.

Tilfeldig tall

Et tall som velges tilfeldig

Beskrivelse

Tilfeldige tall kan brukes i spill eller i noen matematiske simuleringer. For å få tilgang på funksjoner som genererer tilfeldige tall må de i python importeres fra modulen `random`. Det finnes flere ulike måter å generere tilfeldige tall og her er noen eksempel:

```
from random import random, randint, choice

# Genererer et tilfeldig flyttall mellom 0 og 1
random()
```

```
# Genererer et tilfeldig heltall mellom 1 og 10
randint(1,10)

# Velger et element fra en liste
choice(["Janne", "Jarl", "Snorre"])
```

Python

Beskrivelse

JavaScript

Beskrivelse

C

Beskrivelse

C++

Beskrivelse

Scratch

Beskrivelse