



Avdeling for informatikk og e-l ring, H gskolen i S r-Tr ndelag

Variabler, konstanter og datatyper

Svend Andreas Horgen

L restoffet er utviklet for faget LN116D Programmering i Visual Basic

2. Variabler, konstanter og datatyper

Resym : Denne leksjonen tar for seg viktig l restoff. Variabler er selve grunnstenene i programmering og en dypere forst else av disse er helt n dvendig for l re   programmere bra. Vi ser p  datatyper etter   ha visualisert teorien bak variabler gjennom et enkelt eksempel. Med datatyper oppn r vi bedre kontroll i programmene v re. F r konstanter introduseres ser vi p  litt mer avansert behandling av variabler, dvs konvertering mellom datatyper og bruksomr det til variablene. Husk at i programmering er det du som bestemmer, men for   bestemme m  du kjenne til reglene og byggestenene. Jobb med dette stoffet, det er viktig!

2.1.	VARIABLER	2
2.1.1.	Tilordning av verdier	2
2.1.2.	Hvorfor det rare navnet "variabel"?	3
2.1.3.	Husk riktig navn.....	3
2.1.4.	Velg gode variabelnavn	4
2.2.	DATATYPER.....	5
2.2.1.	Datatypen string	5
2.2.2.	Numeriske datatyper.....	6
2.2.3.	Boolske variabler.....	8
2.2.4.	Date.....	8
2.2.5.	Verdimengder, plass og navngiving	9
2.2.6.	Hvorfor datatyper?	9
2.3.	AVANSERTE TING – LEVETID, BRUKSOMR�DE OG KONVERTERING	9
2.3.1.	Levetid og bruksomr�de	10
2.3.2.	Konvertering.....	13
2.4.	KONSTANTER	15
2.4.1.	Arealet til en sirkel.....	15
2.4.2.	Fleksibel moms p� egg fra frittg�ende h�ns.....	16
2.4.3.	Om god programmeringsskikk.....	16
2.4.4.	Om det � lese inn fra skjerm	17
2.5.	OPPSUMMERING	18

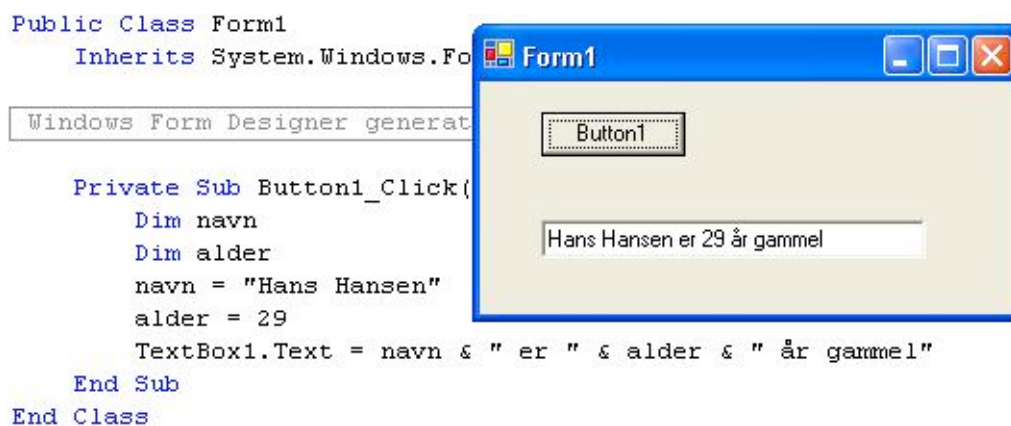
2.1. Variabler

Hvis du får i oppgave å legge sammen fem tall er det vanskelig å gjøre dette i en sleng. Det mest naturlige er å legge sammen de to første, ta vare på summen og legge til det tredje tallet, huske den nye summen og legge til det fjerde, huske denne summen og legge til det femte. Resultatet blir den endelige summen. Til hjelp for å huske informasjon (eller ta vare på mellom-summer) i programmering, kan variabler brukes.

En variabel er navnet på en fysisk adresse i internminnet, eller RAM, i datamaskinen. En variabel kan lagre informasjon for bruk i programmene vi lager, og vi kan bruke så mange variabler vi vil. Uten variabler kommer vi ikke langt i programmering.

2.1.1. Tilordning av verdier

For å kunne bruke en variabel må den opprettes, eller *deklarerer* som vi gjerne sier. Følgende programsnitt forklarer hvordan en variabel deklarerer og fylles med innhold. Dersom koden limes inn bak en knapp i et skjema som har et tekstfelt med navn TextBox1, vil følgende bli resultat når knappen trykkes ved kjøring:



Figur 1: Et enkelt program som gjør bruk av to variabler. Variablene inneholder informasjon.

Det er kodeordet `Dim` som deklarerer en variabel. Alle variabler må opprettes øverst i koden rett under `Private Sub Button1_Click()...` Variablene `navn` og `alder` deklarerer før de fylles med innhold. Ved å skrive setningen

```
    navn = "Hans Hansen"
```

fyller vi innholdet "Hans Hansen" inn i variabelen som heter `navn`. Alderen angir vi på lik måte ved å tilordne tallet 29 til variabelen `alder`. Til slutt bruker vi det som ligger i disse to variablene gjennom å si at kontrollen `TextBox1` sin `Text` egenskap skal endres til å få innholdet på høyre side av likhetstegnet. Altså vil innholdet i variabelen `navn` slås sammen med et mellomrom, ordet "er" og et nytt mellomrom, etterfulgt av innholdet i variabelen `alder` etterfulgt av et mellomrom, ordet "år", et nytt mellomrom og ordet "gammel". Vi ser at det er `&`-operatoren som slår sammen to tekststrenger til en større tekststreng.

2.1.2. Hvorfor det rare navnet "variabel"?

Det er fullt lov å la variabler først få en verdi, og så overskrive denne verdien. Det er nettopp derfor det heter variabler. Innholdet i en variabel kan gjerne byttes ut etter hvert som programmet går sin gang, altså "variabelt innhold". For eksempel:

```
Dim tall
tall = 24
MsgBox("Tallet er nå " & tall)
tall = -28
MsgBox("Tallet er nå endret til " & tall)
```

Her vil første meldingsboks skrive ut at tallet er 24, mens neste meldingsboks skriver ut at tallet er nå endret til -28.

Du kan se på variabelen som en boks som har plass til akkurat én bit av informasjon om gangen. Skal du representere mer informasjon må du ty til flere variabler som lever side om side. Nettopp dette er gjort i eksempelet med navn og alder på en person.

2.1.3. Husk riktig navn

Det er viktig å skrive riktig navn når vi skal bruke en variabel. Hvis vi hadde glemt en "r" og skrevet

```
alde = 29
```

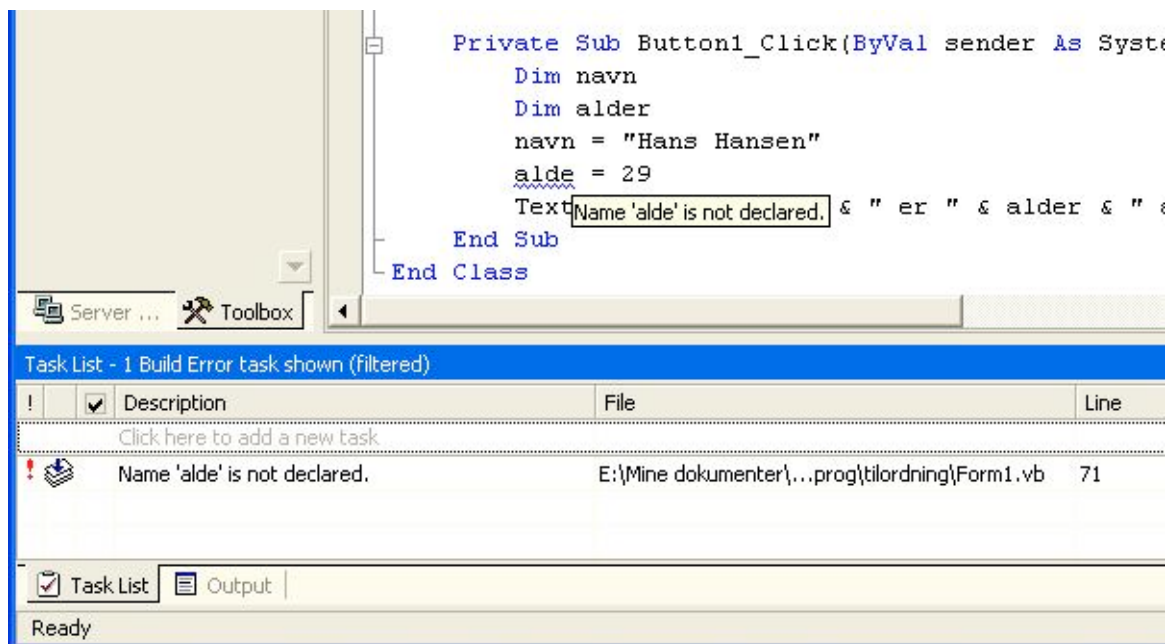
ville ikke programmet fungere. I utviklingsmiljøet vil følgende feilmelding komme fram når programmet kjøres:



Figur 2: Ved kjøring kommer det beskjed om eventuelle feil i programmet.

En bør da trykke på "Nei" for å få se hvor feilen ligger og rette opp. Feil skjer faktisk ganske ofte. Hvordan kan vi oppdage og rette opp feil som denne? I dette tilfellet vil variabelen `alde` være understreket med blå skrift siden det er rundt den Visual Studio oppdager at det er en feil. Dersom musa holdes i ro over dette stedet vil det komme fram en liten gul beskjed som sier at variabelnavnet `alde` ikke er deklarerert. Det samme kan vi se dersom vi ser i oppgaveruten (task-list) nederst på skjermen. Hvorfor er ikke variabelen deklarerert? er det naturlig å tenke da. Vi ser at vi har deklarerert en alder-variabel, men ved nærmere ettersyn ser vi også at det er en skriveleif ute og går.

Visual Basic tror at variabelen `alde` er en helt annen enn variabelen `alder`, men siden alle variabler må deklarereres før bruk og variabelen `alde` ikke er deklarerert, kommer feilmeldingen. Det er i så måte til stor hjelp for oss at vi må ta oss det ekstra bryet det er å deklarere variabler før bruk! (De som har programmert i Visual Basic 6.0 eller tidligere, vil vite at det før .NET kom ikke var nødvendig å deklarere variabler)



Figur 3: Utviklingsmiljøet tilbyr mange former for visuell hjelp som er nyttig for å oppdage, lete etter og korrigere feil.

2.1.4. Velg gode variabelnavn

Kort oppsummert er det nyttig å være bevisst på valget av variabelnavn:

- Vær konsekvent, for eksempel ved å alltid holde deg til variabelnavn som starter med små bokstaver. I praksis betyr små og store bokstaver i Visual Basic ingenting for om programmet kjører, men det er lettest å lese kode dersom hovedsakelig små bokstaver brukes.
- Bruk gode variabelnavn, og dersom du ønsker å bruke flere beskrivende navn i variabelen, så husk at et variabelnavn ikke kan ha mellomrom. Det er for eksempel mye lettere å lese `etVeldigLangtVariabelnavn` eller `et_veldig_langt_variabelnavn`, fremfor `etveldiglangtvariabelnavn`. Enig?
- Navn som i punktet over dokumenterer koden, det vil si at navnet er med på å forklare hva som skjer i programmet. Det er liten tvil om at variabelnavnet `alder` er mye bedre enn bare `a`. Er dette programmet lett å lese for deg?

```
Dim t1
Dim t2
Dim t3
t1 = 4
t2 = 8
t3 = t1 * t2 / 2
```

- De fleste vil nok uansett foretrekke følgende variant som gjør nøyaktig det samme:

```
Dim startTall
Dim sluttTall
Dim gjennomsnitt
startTall = 4
sluttTall = 8
gjennomsnitt = startTall * sluttTall / 2
```

- Mellomrom eller bindestrek skal ikke brukes i variabelnavn. Du kan bruke tall i variabelnavn, men variabler må starte med en bokstav. Du kan ikke bruke spesialtegn som komma, punktum, skigard (#) og liknende.

2.2. Datatyper

Vi vet nå at en variabel er et sted i minnet som har et *navn* og et innhold i form av en *verdi*. Dette innholdet kan være for eksempel tekst eller et tall. I forrige eksempel lagret vi først et navn og deretter et tall, i to ulike variabler.

Det er også mulig å legge andre ting i en variabel, og vi kan angi at en bestemt variabel skal inneholde en bestemt type informasjon. Vi kategoriserer i en rekke såkalte *datatyper*, som beskrives i det følgende.

2.2.1. Datatypen string

Forrige eksempel kunne like gjerne vært skrevet slik:

Vi definerer datatypen eksplisitt

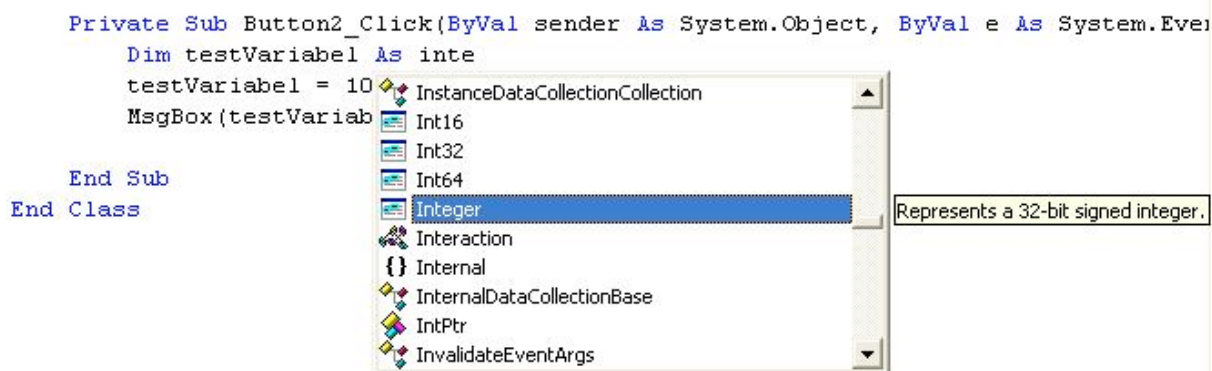
```
1. Dim navn As String
2. Dim alder As Integer
3. navn = "Hans Hansen"
4. alder = 29
5. TextBox1.Text = navn & " er " & alder & " år gammel"
```

De to første linjene er markert i grått for å påpeke at det er her endringen har skjedd fra forrige gang. I stedet for bare `Dim navn` har vi skrevet

```
Dim navn As String
```

Visual Basic vil oppfatte variabelen `navn` som å være av datatype *String*, som igjen betyr at det alltid må lagres tekst i denne variabelen. *String* betyr *tekst* eller *tekststreng* oversatt fra engelsk. Vi skal se mer på hva en streng egentlig er senere i leksjonen.

Etter at du har skrevet inn et mellomrom bak kodeordet `As` etter navnet på en variabel, vil det komme fram en nedtrekksliste av mulige ting å skrive. Denne listen er svært lang, som vist i illustrasjonen under:



Figur 4: Visual Studio hjelper til med å finne fram til lovlige ting å skrive til enhver tid. Legg merke til den gule lappen som her gir litt mer informasjon om datatypen Integer.

Skriv inn første bokstav i den datatypen du ønsker, for eksempel "i", og listen blar seg fram til alle muligheter som starter med bokstaven i. Fortsett med å skrive inn til du har teksten "inte" og du har plutselig fått markert datatypen *Integer* med blått. Trykk mellomrom, enter eller tab, og resten av ordet fyller seg inn automatisk. Du kan endre på antall listevalg som vises i denne hurtiglisten ved å endre i utviklingsmiljøets innstillinger under menyvalget Tools...Options...General.

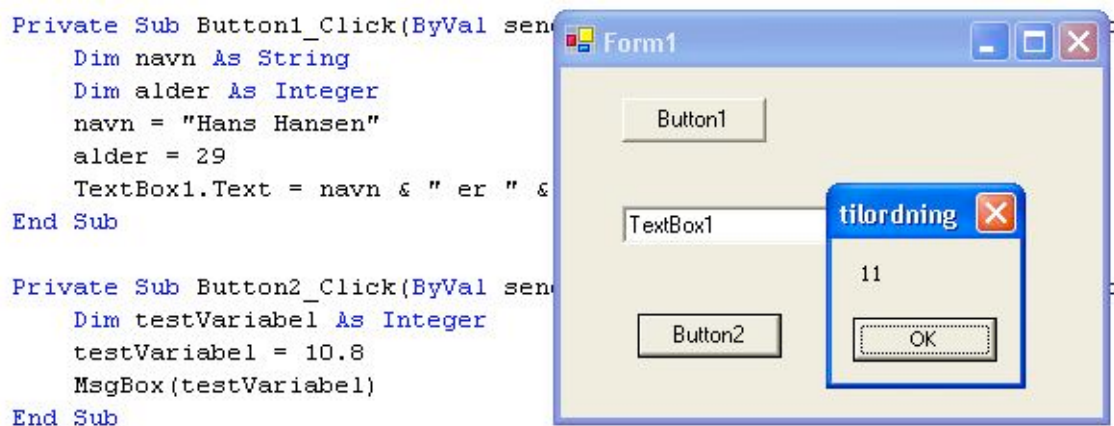
! Det er Visual Basic som programmeringsspråk som tilbyr deg å bruke variabler og datatyper. Det er Visual Studio (utviklingsmiljøet) som tilbyr en nedtrekksliste for å hjelpe deg å skrive inn riktig informasjon, eventuelt raskere. Her ser du et tydelig eksempel på forskjellen mellom et programmeringsspråk og et utviklingsmiljø.

2.2.2. Numeriske datatyper

Integer er en av mange numeriske datatyper. Short, long, single, double og decimal er andre. Disse har felles at de er konstruert for å lagre tall. Det fins mange ulike numeriske datatyper, men vi påpeker at i praksis vil valget nesten alltid stå mellom Integer og Double for å lagre henholdsvis heltall og desimaltall. Eksempel på heltall er 10, 24, 0, 1, -8, -10785, 3 og så videre, det vil si alle positive og negative naturlige tall (opp til en viss størrelse). Eksempel på tall som passer med datatypen Double, er 3.14, -25.899986547, 108757964.3, 0, 4, 6.25 og så videre. Både tall med og uten komma faller inn under datatypen Double. Merk at komma må skrives som punktum i Visual Basic. Grunnen til dette blir klart senere i kurset.

Integer vs Double

Dersom man prøver å legge et desimaltall inn i en variabel som er deklartert med datatype Integer, vil Visual Basic automatisk runde dette opp eller ned til nærmeste hele tall, noe kodesnutten under knapp 2 (Button2_Click) avslører. Se i dette eksempelet bort fra koden som ligger under knapp 1.

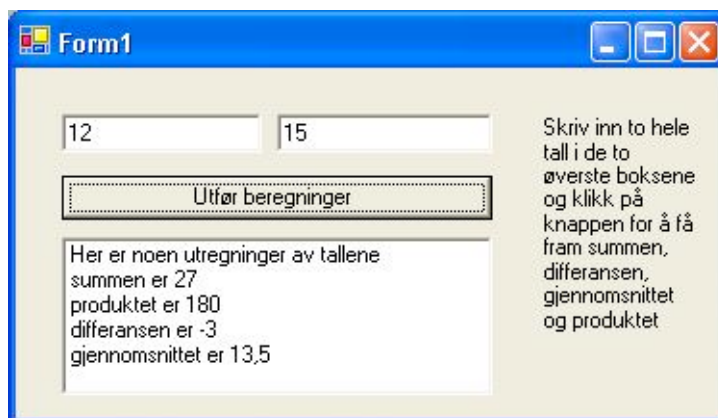


Figur 5: Når knapp nummer to trykkes, blir tallet 10,8 rundet opp til 11 siden testVariabel er av type Integer. Slike finurligheter kan føre til feil i programmet hvis du ikke er klar over det!

Vi lager en enkel kalkulator

Numeriske datatyper brukes ofte til å utføre aritmetiske operasjoner, så som å beregne sum, gjennomsnitt, differanse, gange sammen to tall osv. Skjermbildet under viser hvordan resultatet av mange matematiske beregninger kan presenteres for brukeren basert på to tall

som skrives inn. Det du her ser er altså en spesialutviklet kalkulator som du ikke akkurat vil finne innebygd i Windows :-)



Figur 6: En kalkulator hvor tallene 12 og 15 er skrevet inn i to tekstfelt, og knappen er trykket. Resultatet av de matematiske beregningene vises i listeboksen under knappen.

Vi har her satt inn en listeboks, som er en nyttig kontroll for å presentere mer enn en linje med data. Du finner den i verktøykassen i utviklingsmiljøet, under navnet "ListBox" og et ikon som ser ut som en hvit boks. Den forklarende teksten til høyre i skjermbildet har vi fått til ved å legge ut en *label* i skjemaet. Denne er gjort større i vertikal retning. Gjør det ved å dra i kanten på den. Deretter er tekstegenskapen til labelen endret til å inneholde den teksten som skal vises.

Nok om designet, det får du sikkert til selv. Følgende kode er lagt bak knappen:

Kode for å gjøre matematiske beregninger

```

1. Private Sub Button1_Click(ByVal sender As System.Object,
                             ByVal e As System.EventArgs)
    Handles Button1.Click
2.     Dim tall1 As Integer
3.     Dim tall2 As Integer
4.     Dim snitt As Double
5.     Dim sum, produkt, differanse As Integer
6.     Dim utData As String
7.
8.     tall1 = TextBox1.Text
9.     tall2 = TextBox2.Text
10.
11.    sum = tall1 + tall2
12.    produkt = tall1 * tall2
13.    differanse = tall1 - tall2
14.    snitt = sum / 2
15.
16.    ListBox1.Items.Add("Her er noen utregninger av tallene")
17.    ListBox1.Items.Add("summen er " & sum)
18.    ListBox1.Items.Add("produktet er " & produkt)
19.    ListBox1.Items.Add("differansen er " & differanse)
20.    utData = "gjennomsnittet er " & snitt
21.    ListBox1.Items.Add(utData)
22. End Sub

```

Hva skjer i koden bak kalkulatoren?

Det er flere ting å merke seg her. Variablene `tall1` og `tall2` er deklarerert som `Integer`, og disse får innholdet som skrives inn i de to tekstboksene når programmet starter. Det er mulig å deklare flere variabler av samme datatype på formen som vist i linje 5, noe som sparer oss for både skriving og plass i tillegg til at det kan gi bedre oversikt over programkoden for en utenforstående. Variablene `sum`, `produkt` og `differanse` er naturlig nok satt opp som `Integer`. Hvorfor det tror du? ¹Variabelen `snitt` derimot, er definert som `Double`. Grunnen til dette er at gjennomsnittet kan bli et desimaltall selv om begge grunntallene som skrives inn er vanlige hele tall. Prøv selv å regne ut snittet av tallene 24 og 25 om du er i tvil. Den observante leser kan sikkert svare på hva som hadde skjedd dersom også `snitt` var definert som `Integer`... ²

Utrekningene starter i linje 11, først etter at variablene er fylt med de tallene som brukeren skal skrive inn i tekstboksene. Aritmetiske operasjoner i Visual Basic, som i alle andre språk, kan vi utføre ved å bruke såkalte *operatorer*, så som `+` (addisjon), `-` (subtraksjon), `*` (multiplikasjon) og `/` (divisjon). I linje 14 regner vi ut gjennomsnittet av de to tallene ved å dele innholdet i variabelen `sum` (som allerede har fått innhold i linje 11) med 2 og legge resultatet i variabelen `snitt`.

Det er litt tungvindt å skrive ut en og en linje i en listeboks, siden hele setningen `Listbox1.Items.Add(ettEllerAnnet)` må skrives hver gang. Likevel er listebokser veldig godt egnet til å presentere data, og vi kommer derfor til å bruke slike i dette kurset. Det som står innenfor parentesene (`ettEllerAnnet`) legges i neste linje i listeboksen på skjermen. Det er mulig å erstatte `ettEllerAnnet` med bare en tekststreng, en blanding av tekst og variabler, eller bare variabler. Linje 16 viser det første, 17-19 er et eksempel på det andre og linje 21 er et eksempel på det siste. Variabler gjør hverdagen mer fleksibel for en som programmerer!

En siste ting å merke seg er at siden differansen som regnes ut består i å trekke tallet som skrives i den andre tekstboksen fra det som skrives i den første, vil resultatet, som i skjermbildet over, kunne bli negativt. Det er ingenting i veien for dette, Visual Basic regner like godt med negative som med positive tall.

2.2.3. Boolske variabler

En boolsk variabel kan inneholde en av de to verdiene `true` eller `false`. Boolske variabler benyttes i forbindelse med kontrollstrukturer, og vi kommer derfor tilbake til disse senere i kurset.

2.2.4. Date

Datoer og klokkeslett legges inn i en egen datatype i Visual Basic, nemlig datatypen `date`. Dato systemet bygger på seriedatoer, som gjør det enkelt å regne med datoer. Vil du finne antall dager fra i dag til år 2025 er det en smal sak ved hjelp av innebygde datametoder. Vi introduserer ikke metoder før senere i kurset, og derfor trenger du ikke legge noe vekt på datoer foreløpig.

¹ Det er helt kurant å bruke `Integer` til `sum`, `produkt` og `differanse`, siden de to tallene `tall1` og `tall2` er av type `Integer`. Da kan ikke disse resultatene bli annet enn `Integer`. Divisjon, derimot, i likhet med `snitt`, kan bli desimaltall. Legg merke til at det ikke vil være galt å bruke `Double` for `sum`, `produkt` og `differanse`.

² Dersom `snitt` hadde vært `Integer`, og tallene var for eksempel 1 og 2, ville gjennomsnittet blitt rundet opp fra 1.5 til 2, noe som åpenbart ikke er helt riktig. Se også Figur 5.

2.2.5. Verdimengder, plass og navngiving

Verdimengden til en datatype angir hvilke elementer som faller inn under dens lovlige verdier. Datatypen boolean kan bare akseptere innlegging av de to verdiene True og False, mens datatypen Short bare aksepterer tall mellom -32.768 og + 32.768. Prøver du å utføre setningene

```
Dim test As Short
Test = 1000000
```

vil programmet stoppe og gi feilmeldingen ”Constant expression not representable in type 'Short' ”.

En oversikt over verdimengdene til de mest brukte datatypene står oppført på ressursiden til denne leksjonen som lenke til Microsoft-dokumentasjonen på Internett. Her kan du blant annet se hvor stor plass hver datatype opptar. Siden det kreves flere byte for å lagre et veldig langt tall enn å lagre et kort tall, tar hver variabel deklarerert som datatype `double` 4 ganger så mye plass som datatypen `short`. Det er derfor anbefalt å bruke de datatypene som best fanger opp de verdiene som en forventer legges inn i variablene. Merk at argumentet om plass ikke er særlig aktuelt i dagens situasjon med mye internminne og store harddisker, men det er såkalt ”god programmeringsskikk” å ikke sløse med plass.

Mange oppfordrer til bruk av **prefiks** som angir hvilken datatype en variabel har. Det kan være greit å bruke prefiks konsekvent i en del tilfeller, men det er også mer tidkrevende å skrive og endre koden. Det er opp til hver enkelt om denne stilen følges eller ikke, og vi gjør det bare av og til i leksjonene i dette kurset. (Bruk av prefiks i navnene til kontrollere som knapper (btn), listebokser (lst), labeler (lbl) og tekstbokser (txt), er derimot en smart ting).

2.2.6. Hvorfor datatyper?

Hvorfor bør en angi datatype for en variabel? Her er en kort argumentasjonsliste:

- Redusert minnebruk
- Raskere kode
- Bedre datavalidering (mindre feil, bedre sikkerhet)
- Selvdokumenterende kode (lettere å lese og forstå koden)
- Færre feil i koden

I noen programmeringsspråk trenger en ikke å angi datatype, men da vil det bak kulissene likevel trolig brukes datatyper. Datatyper er nødvendige av ovennevnte grunner.

2.3. Avanserte ting – levetid, bruksområde og konvertering

Vi har nå introdusert variabler og sett at ulike datatyper skal benyttes til å legge inn forskjellige typer av informasjon i variablene. For å få en fullgod forståelse av variabler må vi også se på spørsmål rundt eksistens – hvor lenge eksisterer en variabel (*levetid*) og hvor er den gyldig (*bruksområde*). *Konvertering* tillater oss å legge innholdet fra en variabel av en datatype inn i en annen variabel av en annen datatype, noe som er helt nødvendig i mange tilfeller.

2.3.1. Levetid og bruksområde

En variabel må deklarerer før den kan benyttes. Vi har sett at dette gjøres med kodeordet `Dim`. I Visual Basic er det også mulig å bruke kodeordene `Static`, `Private` og `Public` i forbindelse med opprettelsen av en variabel.

Deklarere en variabel med kodeordet Dim – enkelt og lokalt

Variabler som deklarerer med kodeordet `Dim`, vil bare eksistere innenfor en prosedyre, vi sier at variabelen er *lokal*. Dette er mest vanlig praksis og brukes i nesten alle tilfeller. Prøv gjerne å trykke på knappen i følgende kodesnutt tre ganger etter hverandre.

```
Private Sub Button3_Click(ByVal sender As System.Object,  
                          ByVal e As System.EventArgs)  
    Handles Button3.Click  
    Dim total As Integer  
    total = total + 3  
    MsgBox(total)  
End Sub
```

I det du starter programmet vil variabelen `total` opprettes. Denne vil samtidig få verdien 0. En fin ting med Visual Basic .NET er at når nye variabler opprettes, vil de automatisk nullstilles samtidig. Tallvariabler får verdi 0, menst tekstvariabler får en tom tekststreng "".

Du stusser kanskje på setningen

```
total = total + 3
```

Dette er en finurlig måte å oppdatere innholdet i en variabel på. I begynnelsen kan det være litt vanskelig å forstå syntaksen, men om du får vite at Visual Basic først regner ut høyresiden, og deretter legger resultatet i venstresiden, blir det kanskje lettere? På høyre side av likhetstegnet blir innholdet i `total` plussset på med 3. Siden `total` i utgangspunktet er 0, blir svaret 0+3, med andre ord 3. Dette blir så lagt i `total`, og initialverdien på 0 forsvinner til fordel for den oppdaterte verdien på 3. Denne måten å øke innholdet i en variabel ved å gi den sin egen verdi plussset på med en annen verdi, vil du ofte se i programmering, både med tekst- og tallvariabler.

Som du kanskje aner, vil resultatet bli det samme hver gang knappen trykkes, nemlig at tallet 3 skrives ut til skjerm³. For hver gang knappen trykkes starter koden bak prosedyren `Button3_Click`, og da vil en ny variabel som heter `total` opprettes, få initialverdi 0, og så utføres resten av koden. En viktig karakteristikk ved bruk av kodeordet `Dim` er at slike variabler eksisterer fra de blir opprettet til prosedyren er ferdig med å kjøre. Da slettes de fra minnet i datamaskinen, og verdien går tapt for all framtid.

Deklarere en variabel med kodeordet Static – vedvarer over tid

Av og til er det hendig å ta vare på en verdi i en variabel selv om prosedyren er ferdig utført. Dersom vi modifierer koden til å benytte stikkordet `static` i stedet, vil noe merkelig skje:

```
Private Sub Button3_Click(ByVal sender As System.Object,  
                          ByVal e As System.EventArgs)  
    Handles Button3.Click  
    Static total As Integer
```

³ Når vi sier "skrives ut til skjerm" mener vi at noe vises på skjermen. Dette uttrykket stammer fra teori i operativsystemfag. En kan nemlig skrive data til en skriver eller til en annen utenhet, for eksempel en skjerm.

```

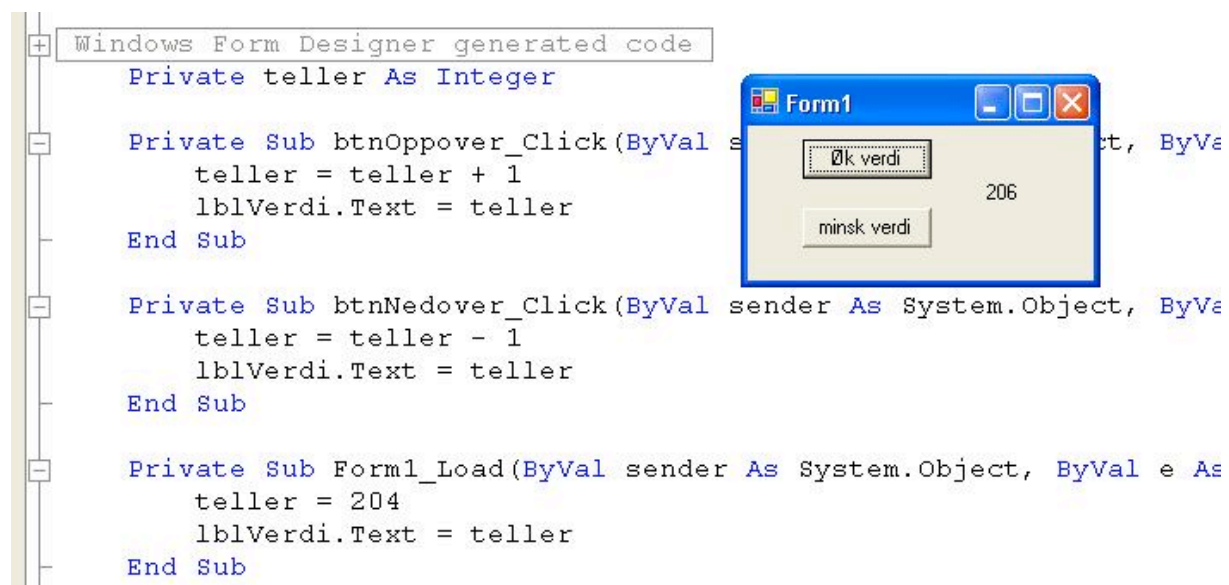
        total = total + 3
        MsgBox(total)
    End Sub

```

Første gang knappen trykkes, vises tallet 3 i boksen. Neste gang vises tallet 6, og tredje gang vises tallet 9. Verdien blir altså tatt vare på selv om prosedyren er ferdigkjørt, noe som skyldes at stikkordet `Static` ble benyttet. Variabler som er deklarerert med `Static`, bevarer sitt innhold til neste gang prosedyren kjøres. Dersom hele programmet avsluttes (trykk på rødt kryss eller stopp-knappen) vil slike variabler miste sitt innhold / slutte å eksistere.

Deklarere en variabel med kodeordet Private – globale variabler

Det er, som vi har sett, mulig å lage flere knapper innenfor et skjermbilde. Hver knapp har en tilhørende click-prosedyre med kode som utføres når knappen trykkes (klikkes) på. Neste eksempel illustrerer hvordan én og samme variabel kan brukes av flere ulike prosedyrer. Normalt skal det ikke gå, siden en variabel bare eksisterer innenfor den prosedyren den er opprettet i.



Figur 7: Et program hvor tre prosedyrer benytter samme variabel.

Øverst i koden er det satt inn følgende setning (rett under Windows Form Designer...)

```
Private teller As Integer
```

Her deklarerer variabelen `teller`, den settes til datatype `Integer`, og den kan benyttes i alle prosedyrer i gjeldende skjema. Grunnen til dette er at den har kodeordet `Private` foran seg, og at den ikke er deklarerert innenfor en prosedyre. Denne typen variabler kaller vi gjerne for *globale variabler*, siden de kan benyttes av flere prosedyrer. Globale variabler skal benyttes med omhu, og bør deklarereres langt opp i koden, helst over første prosedyre. Slik er det enkelt å lokalisere de globale variablene når du senere en gang vender tilbake til koden.

Det er flere ting å merke seg i koden over:

- Hendelsen `Load` til et skjema slår til/utføres i det samme øyeblikk som programmet starter ("load" betyr å "laste inn" på engelsk). Det første som skjer i programmet vårt er derfor at prosedyren

```
Private Sub Form1_Load(ByVal sender As System.Object,  
                      ByVal e As System.EventArgs)  
    Handles MyBase.Load  
    teller = 204  
    lblVerdi.Text = teller  
End Sub
```

som befinner seg nederst i programmet, starter automatisk opp som det første som skjer. (Rekkefølgen av hvordan prosedyrer står oppført i kodemiljøet har altså ingenting å si for når de blir utført – hvorfor er det slik, tror du? ⁴).

- Det første som skjer i prosedyren `Form1_Load` er at variabelen `teller` får verdien 204. Siden `teller` er en global variabel som er deklarerert med datatype `Integer` helt øverst i koden, går dette i orden. `Form1_Load` har altså tilgang til variabelen `teller`.
- En label som vi har kalt for `lblVerdi` får innholdet i variabelen `teller`. Vi husker å spesifisere at det er *tekstegenskapen* som skal få verdien.
- Ingenting skjer etter det. Programmet har kjørt ferdig prosedyren og venter på at brukeren skal trykke på en av de to knappene for å henholdsvis øke eller minke verdien i variabelen `teller`.
- Når så brukeren klikker på knappen for å øke verdien, vil koden under knappen `btnOppover` utføres og variabelen `teller` får verdien $204 + 1$ som er 205. Det samme gjentas en gang til, og verdien 206 ligger plutselig i variabelen. Resultatet skrives ut i labelen, slik at vi kan se hva som skjer til enhver tid.

Eksempelet illustrerer hvordan flere prosedyrer kan være med på å endre verdien til en variabel. Det var helt nødvendig å deklare variabelen `teller` som `Private` for å få ønsket funksjonalitet til programmet vårt. Hver gang flere prosedyrer skal bruke en felles informasjonsressurs, må globale variabler brukes. En global variabel tillater altså kommunikasjon på tvers av prosedyrer, mens en lokal variabel kun er synlig innenfor den prosedyren den er deklarerert i.

Public – global over flere moduler

Vi skal ikke gå inn på den siste muligheten her, men nevner kort at `Public` kan brukes for å opprette variabler som ikke bare er tilgjengelige i alle prosedyrer i et skjema, men også mellom flere skjema eller flere moduler. Vi kommer tilbake til slik oppdeling senere i kurset når vi møter litt større problemstillinger som krever et mer gjennomtenkt design og oppdeling av programmet i flere mindre moduler og skjermbilder.

Viktig – bruk mest mulig lokale variabler

En fin regel er å bruke så lokale variabler som mulig. Det er mange grunner til det, hovedbegrunnelsen er kontroll og å unngå potensiale for feil. Bruk altså `Dim` så mye som mulig, og deklarer slike variabler *innenfor* den prosedyren de brukes i. Unntaksvis kan du bruke `Private` (eventuelt `Public`) for å få globale variabler, men bare dersom samme verdi

⁴ Rekkefølgen spiller ingen rolle – det er navnet på prosedyrene som avgjør hvilken kode som tilhører hva. Egentlig er det heller ikke navnene, men det som står bak ”Handles” som avgjør hvilken hendelse hver kodesnutt er knyttet opp mot. Mer om dette senere i kurset!

skal kunne være tilgjengelig for flere prosedyrer. Bruk av `Static` er hendig når informasjon må huskes over tid innenfor en prosedyre, og samtidig beskyttes fra andre prosedyrer.

2.3.2. Konvertering

Alt som skrives ut til skjerm i Visual Basic er i form av tekst. Selv tallet 29, som vi skrev ut i første eksempel, vil Visual Basic tolke som teksten "29". Teksten "29" består av de to tegnene "2" og "9" fra ASCII-tabellen. ASCII-tabellen er spesiell, siden den har en lang liste av en rekke tegn. I forbindelse med hvert tegn er det assosiert et unikt nummer. Store bokstaver finnes i tabellen med nummer 65 til og med 91, der 65 betyr A, 66 betyr B og 91 betyr Z. Tilsvarende finnes de små bokstavene i tabellen, og tegnene æ, ø og å. Numeriske tegn, dvs 0, 1, 2, ..., 9 finnes fra plass nummer 48 til 57, og andre tegn som \$, ", #, £,) og [finnes også. Er du interessert i hele tabellen, så søk etter "ascii table" på internett.

Bli 11 + 12 = 23?

Hvorfor tar vi med dette som en introduksjon på konvertering mellom datatyper? ASCII-tabellen er viktig for å forstå hvordan samspillet mellom tekst og tall foregår. Et eksempel illustrerer dette meget godt:

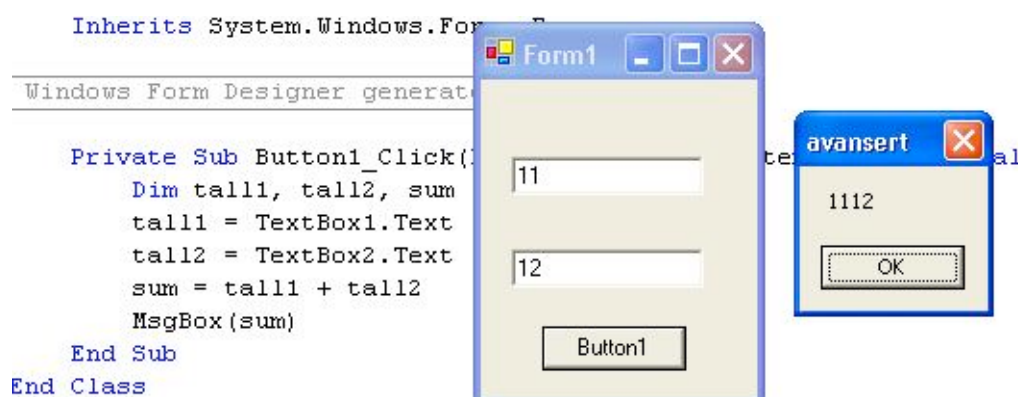
Ingen konvertering gir feil svar

```

1. Private Sub Button1_Click(ByVal sender As System.Object,
2.     ByVal e As System.EventArgs)
3.     Handles Button1.Click
4.     Dim tall1, tall2, sum
5.     tall1 = TextBox1.Text
6.     tall2 = TextBox2.Text
7.     sum = tall1 + tall2
8.     MsgBox(sum)
9. End Sub

```

Hva blir resultatet av koden over gitt at tallene 11 og 12 skrives inn i de to tekstboksene i Figur 8? Vi har tidligere sett at `&`-operatoren brukes for å slå sammen to tekststrenger, som vi gjorde i eksempelet med Hans Hansen og kalkulatoren. Visual Basic tillater også at `+` benyttes for å slå sammen to tekststrenger. Det bør nå være klarere hva som skjer: Det er ikke 23, men 1112 som skrives ut i meldingsboksen. Hva har skjedd?



Figur 8: Er $11 + 12$ virkelig 1112? Ja, men hvorfor, og hva kan gjøres for å unngå denne feilen?

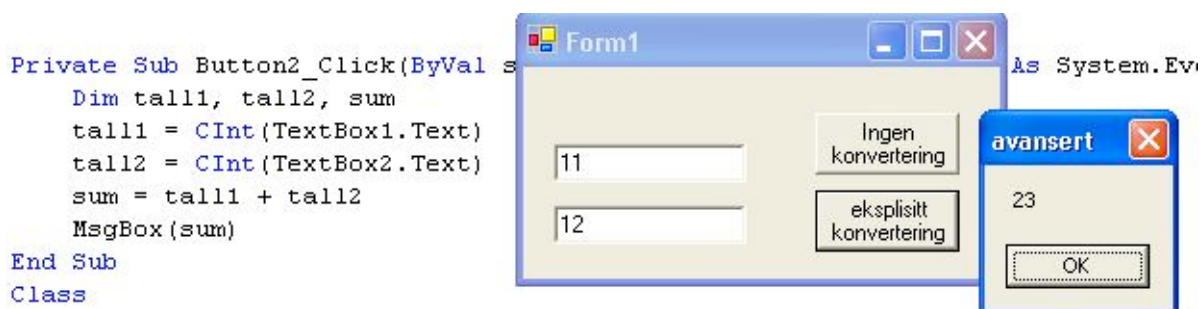
Forklaringen ligger i hvordan Visual Basic oppfatter kildematerialet. Alt som kommer fra skjerm er tekst, og dermed tolkes de to tallene 11 og 12 som tekst. Tenk på ASCII-kodene så

ser du at 11 og 12 gjerne kan være tekststrengene "11" og "12", bestående av enkelttegnene "1", "1", "1" og "2". Hva skjer når to strenger slås sammen? Jo, den første settes foran den andre. "11" + "12" betyr det samme som "11" & "12", og Visual Basic slår dermed de to sammen til resultatet "1112". Den årvåkne leser har fått med seg at kalkulatoren vi lagde la sammen to tall uten problemer ved å bruke operatoren +. Hvordan unngår vi problemet og får riktig svar som resultat også i dette eksempelet?

Grunnen til at alt gikk bra i kalkulator-eksempelet, var at vi der hadde deklarerert de to variablene `tall1` og `tall2` som datatype `Integer`. Siden bare hele tall kan ligge lagret i en variabel av datatype `Integer`, konverterte Visual Basic teksten som ble mottatt fra skjerm (string) automatisk til tall (integer). Når vi nå i siste eksempel deklarerer variablene uten å angi datatype (jmf. linje 2) tolket VB det hele som at vi ønsket å slå sammen to tekststrenger, og ingenting skjedde.

Implisitt vs eksplisitt konvertering

Av og til vil VB oppdage at det er nødvendig å konvertere mellom datatyper, og da skjer dette automatisk, underforstått, eller *implisitt* som vi også sier. Vi kan forutse når dette kommer til å skje, men det beste er å *eksplisitt* angi at vi ønsker en konvertering. Følgende modifiserte eksempel illustrerer dette.



Figur 9: Her er 11 + 12 korrekt summert til 23. Med eksplisitt konvertering eller bevisst bruk av datatyper unngås feil som eksempelet fra Figur 8.

Koden er nesten lik den i det eksempelet som gav feil svar. Endringen ligger i det stedet i koden hvor `tall1` og `tall2` fylles med innhold. Funksjonen `CInt()` er brukt for å konvertere det som står mellom parentesene til datatypen `Integer`. Dette kalles *eksplisitt konvertering*. Med andre ord blir de "tekst-tallene" vi skriver inn i tekstboksene konvertert til ekte heltall, før de legges i variablene. Siden VB ser at det er to tall som skal summeres ved hjelp av operatoren +, blir summen riktig beregnet, og tallet kan skrives ut på skjerm. Alt som skal skrives ut på skjerm må også være tekst, men her slår VB til og foretar konverteringen mellom tallet som ligger i variabelen `sum` til den teksten som skal vises på skjerm. Dette kalles *implisitt konvertering*. Det nevnes at vi kan fremtvinge konvertering fra en hvilken som helst numerisk datatype til `String` ved å benytte konverteringsfunksjonen `Str()`.

Eksempelet illustrerer hvor viktig det er å tenke over hva VB tror det håndterer: Tekst eller tall? Vi så at i det første eksempelet ble *ingen konvertering* foretatt og + ble tolket som sammenslåing av tekststrenger. I det andre sørget vi selv for å *eksplisitt* fremtvinge konverteringen, noe som gjorde at + ble tolket som addisjon. I tilfellet med kalkulatoren ble teksten *implisitt* konvertert til tall siden variablene var av datatype `Integer` (kun tall kan legges i slike), og + ble dermed tolket som addisjon.

2.4. Konstanter

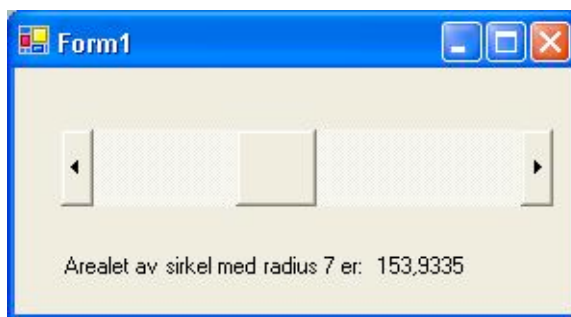
Variabler kjennetegnes med at verdiene i dem endres underveis i programmet. Konstanter tilbyr oss en mulighet for å lagre verdier som ikke skal endres særlig ofte. Et eksempel på en fin konstant er verdien til π . π har som vi kjenner fra matematikkens verden alltid verdien 3.14159265... og vil aldri få noen annen verdi. Momsen er kanskje på 24%, og vil ikke endres så ofte. Programmer som gjør bruk av konstanter blir fleksible i bruk.

2.4.1. Arealet til en sirkel

Hvis vi skal regne ut arealet til en sirkel, må vi bruke formelen

$$\text{Arealet} = \pi * \text{radius} * \text{radius}$$

der π er 3.14 og radius kan endres fra gang til gang. La oss lage et program som lar brukeren dra i et rullefelt og slik endre verdien til radiusen. Skjermbildet kan se slik ut:



Figur 10: Her er en horisontal scrollbar brukt. Dobbelklikk den, og skriv inn kode.

For hver gang brukeren klikker på en av pilene eller drar i firkanten, vil egenskapen `.Value` endre seg tilsvarende. Ved å programmere slik at denne verdien leses av, kan vi regne ut nåværende radius og legge den i en label under rullefeltet. For å få til at koden blir kjørt for hver gang du endrer på rullefeltet, er det altså selve rullefeltet som må ha en bakenforliggende kode. Det er `_Scroll` hendelsen som skal utføres. Koden kan være slik:

Vi lager konstanten PI og bruker den i arealberegning

```

1.  Const PI = 3.1415
2.  Private Sub hsbRadius_Scroll(ByVal sender As System.Object,
                                ByVal e As System.Windows.Forms.ScrollEventArgs)
                                Handles hsbRadius.Scroll
3.      Dim radius As Integer
4.      Dim areal As Double
5.      Dim utskrift As String
6.      radius = hsbRadius.Value
7.      utskrift = "Arealet av sirkel med radius " & radius
8.      areal = PI * radius * radius
9.      utskrift = utskrift & " er: " & areal
10.     lblAreal.Text = utskrift
11. End Sub

```

I linje 1 oppretter vi en konstant, kaller denne for `PI` og gir den verdien 3.1415. Merk at vi bruker punktum til å angi desimaltall, og ikke komma. I linje 3, 4 og 5 oppretter vi de variablene vi trenger, hvilket er tre stykker. `Utskrift` skal ha den teksten som skal skrives på skjerm.

Vi har laget rullefeltet ved å klikke på kontrollen ”Horisontal Scrollbar” i verktøykassen, eller *HscrollBar* som den i utgangspunktet heter. Denne har vi endret navn på, til `hsbRadius`. Når vi i designmodus dobbeltklikket på denne kom vi inn i prosedyren til hendelsen `Scroll` knyttet til dette rullefeltet. Hver gang vi ruller i rullefeltet ved å trykke på høyre/venstre-delen eller dra i klossen midt i, vil koden som står skrevet inn i `hsbRadius_Scroll` utføres.

Radiusen får verdien som rullefeltet har i linje 6. Denne verdien hentes ut ved å benytte *Value*-egenskapen til rullefeltet. Vi har på forhånd satt inn maksimal verdi til å være 20, og minste verdi til å være 0. Disse og andre egenskaper kan du selv utforske i egenskapsvinduet. Arealet beregnes i linje nummer 8 ved å bruke konstanten π og multiplisere med den radiusen som blir lest fra rullefeltet ($\text{radius} * \text{radius} * \pi = \text{areal}$). I eksempelet i Figur 10 er tallet 7 valgt som radius.

Vi skal skrive ut mye tekst i labelen med navn `lblAreal`, og derfor bruker vi variabelen `utskrift` som lagringssted for denne teksten. Etter å ha fylt den med tekst i linje 7, regner vi ut arealet, og ønsker så å legge til mer tekst i linje 9. På samme måte som med telleren som ble oppdatert i eksempelet med globale variabler, skjøter vi sammen tekststrengen som allerede ligger i variabelen `utskrift` med seg selv og ordet ”er” og arealet som ble regnet ut. Resultatet blir vist under rullefeltet.

Tips: Du kan sette maksimum- og minimumverdi på rullefeltet, samt steg for hver endring. Dette gjør du for eksempel i egenskapsvinduet eller i koden under prosedyren `form_load`.

2.4.2. Fleksibel moms på egg fra frittgående høns

Sett at du skal lage et program som beregner priser for en butikk. Programmet inneholder en rekke prosedyrer som behandler summene – en viser pris uten moms, en annen viser prisen med moms, en tredje regner sammen summen av mange varer og finner den totale momsen, og en fjerde viser den merverdiavgiften som gjelder for egg fra frittgående høns :-)

Momsen varierer med jevne mellomrom. Noen ganger er den 23%, så kanskje 24%, 23.5% eller liknende. Når endringen skjer er det tungvint å lete seg fram til alle steder hvor dette tallet er benyttet i koden. Det er bedre å definere momsen som en konstant en gang for alle øverst i koden, og så referere til denne konstanten konsekvent i de prosedyrene som bruker moms i sine utregninger.

Vi kommer tilbake til bruk av konstanter i forbindelse med filbehandling, databaser, etc, men det er generelt greit å tenke på muligheten for å benytte slike i programmene du skriver. Ikke overdriv – du skal, som en god regel, kunne begrunne bruk av konstanter minst like godt som bruk av globale variabler og stå for dette valget.

! Konstanter kan ha samme type innhold som variabler (altså samme datatyper), men det er ikke mulig å endre innholdet i en konstant etter at den har fått verdi. Den får verdi

- når programmet starter. Variabler derimot kan få og miste innhold flere ganger i løpet av kjøretiden. Du ser sikkert nå at navnene ”konstant” og ”variabel” egentlig snakker for seg selv. De er begge bokser i programmeringsverdenen med mulighet for å lagre informasjon, men det er likevel stor forskjell. En konstant er for øvrig tilgjengelig over hele det skjemaet den er definert i, og lever så lenge programmet kjører. Skal du ha konstanter som gjelder for flere skjema, kan du legge slike i en egen modul med stikkordet `Public` foran konstanten.

2.4.3. Om god programmeringsskikk

Vi lærer folkeskikk i oppveksten, og på samme måte er det på sin plass å introdusere begrepet ”god programmeringsskikk”. Dette er regler som er ansett som ”riktige” innenfor fagmiljøene

som driver opplæring i grunnleggende programmering. Vi kan komme med mange flere punkter i løpet av kurset, men lister opp noen så langt:

- Legg informasjon fra skjerm i variabler før videre behandling. Det er ikke god programmeringsskikk å multiplisere to tekstfelter med hverandre, men mye bedre å multiplisere to variabler som først har fått innholdet fra de to tekstfeltene.
- Velg gode variabelnavn.
- Angi datatype for de variablene du deklarerer.
- Bruk lokale variabler hvis det er mulig.
- Bruk konstanter med måte og dersom en verdi gjentas ofte i et program uten å endres i løpet av programmet, men der det er en mulighet for at verdien kanskje kan endres for eksempel årlig eller hver måned (programmet må i så fall kompiles på nytt for hver endring).
- Unngå implisitt konvertering med mindre du er helt sikker på at det blir riktig.
- Rykk inn koden slik at den blir oversiktlig å lese for andre. Bruk et ekstra linjeskift der det er mest naturlig.

2.4.4. Om det å lese inn fra skjerm

Teksten "les inn fra skjerm" forekommer ofte i oppgaveformuleringer. Å lese inn fra skjerm betyr å motta data fra brukeren. Brukeren skriver inn noe i enten et tekstfelt, eller en inputbox (som vi ikke har brukt enda). For den saks skyld kan også et rullefelt brukes, som i eksempelet med arealet av en sirkel. Når forskjellig informasjon kan mates til programmet hver gang, vil resultatet bli forskjellig. Brukeren kan påvirke resultatet. Dette er vist i flere eksempler så langt, for eksempel programmer som mottar to tall og finner summen, differansen, produktet og gjennomsnittet.

- ! Å lese inn fra fil eller database betyr å hente data fra enten en fil eller en database. Behandlingen av disse dataene kan være nøyaktig den samme som om de ble lest inn fra skjerm.

Det enkleste, men minst fleksible (som en aller helst bør unngå i virkelige løsninger), er å *hardkode* data rett i koden. Dersom oppgaven er formulert slik:

"Lag et program som skriver ut alder og navn i en msgbox"

vil mange nybegynnere hardkode, og foreslå noe ala dette som løsning:

```
msgbox "Jeg heter Ola Nordmann og er 25 år"
```

Dette fungerer bra, men er ikke særlig fleksibelt. Noe bedre er følgende praksis, som vi har brukt en del i eksemplene. Merk at vi har brukt dette bare for å illustrere prinsippet med variabler og for å holde programmene så enkle som mulig:

```
dim alder = 25
dim navn = Ola Nordmann
msgbox "Jeg heter " & navn & " og er " & alder & " år"
```

Det aller mest fleksible, og *det du stort sett alltid bør gjøre*, er:

```
dim alder as integer
dim navn as string
navn = txtNavn.text
alder = Cint(txtAlder.text)
```

```
msgbox "Jeg heter " & navn & " og er " & alder & " år"
```

Forskjellen er stor, meget stor. Merk likevel at du i enkelte eksamensoppgaver vil kunne hardkode uten å bli trukket (eksempler på dette vil framgå av tidligere løsninger). Dermed sparer du tiden det ellers ville tatt å skrive kode for tekstbokser o.l. Dessverre bommer noen grovt på eksamen ved å hardkode når oppgaven helt tydelig legger opp til å ikke hardkode. Øvelse gjør mester her! Et tips er å tenke på formålet med programmet som skal lages – skal det være fleksibelt eller bare illustrere et poeng/forståelse?

2.5. Oppsummering

Denne leksjonen har introdusert variabler og konstanter. I tillegg til teori har vi prøvd å illustrere bruken av variabler gjennom morsomme eksempler som bruker spennende kontrollere. Det er ikke mulig å beskrive alt som skjer like godt gjennom hele teksten, noe som krever at du som leser selv aktivt prøver nye muligheter rundt nytt lærestoff.

Temaet omkring konvertering mellom datatyper er interessant, men kan være vanskelig. Dvel gjerne en stund ved dette, bruk variabler i praksis i øvingen og andre oppgaver, og les så teorien på nytt igjen for en økt forståelse rundt dette viktige aspektet. Også bruksområdet til en variabel vil bli mer tydelig gjennom kurset.

Variabler er noe du skal bruke gjennomgående, så å si ALLTID når du programmerer. Lær hvordan de fungerer, så har du et godt grunnlag. Variabler og datatyper vil være helt nytt stoff for mange. Dersom du ikke forstår alt nå, så fortvil ikke. Vi kommer tilbake til variabler hele tiden, og når du prøver i praksis vil det bli mer klart hvorfor du bruker slike og hvordan de fungerer.