Heftet til
TI-Nspire™-programredigering

Finn ut mer om TI Technology via online hjelp på education.ti.com/eguide.

Viktig informasjon

Med mindre annet er uttrykkelig angitt i lisensen som følger med et program, gir Texas Instruments ingen garanti, enten uttrykkelig eller underforstått, inkludert, men ikke begrenset til, noen underforståtte garantier for salgbarhet og egnethet til et bestemt formål, med hensyn til programmer eller bokmaterialer og gjør slike Materialer tilgjengelig utelukkende på en "as-is" basis. Under ingen omstendigheter skal Texas Instruments være ansvarlig for noen som har spesielle, sikkerhets-, tilfeldige eller følgeskader i forbindelse med eller som følge av kjøp eller bruk av disse materialene, og Texas Instruments eneste og eksklusive ansvar uansett hvilken form for handling, skal ikke overstige det beløpet som er angitt i lisensen for programmet. Videre skal Texas Instruments ikke holdes ansvarlig for ethvert krav av noe slag mot bruken av disse materialene av noen annen part.

© 2020 Texas Instruments Incorporated

TI-Nspire™ -programvaren bruker Lua som skriptmiljø. For informasjon om opphavsrett og lisens, se http://www.lua.org/license.html.

TI-Nspire™ -programvaren bruker Chipmunk Physics versjon 5.3.4 som simuleringsmiljø. For lisensinformasjon, se http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/.

Microsoft® og Windows® er registrerte varemerker for Microsoft Corporation i USA og / eller andre land.

Mac OS®, iPad® og OS X® er registrerte varemerker for Apple Inc.

Unicode® er et registrert varemerke for Unicode, Inc. i USA og andre land.

Faktiske produkter kan være litt annerledes enn på bilder.

Innhold

Komme i gang med Program Editor **1**

Definere et program eller en funksjon 2
Vise et program eller en funksjon 5
Åpne en funksjon eller et program for redigering 6
Importere et program fra et bibliotek 6
Opprette en kopi av en funksjon eller et program 6
Gi nytt navn til et program eller en funksjon 7
Endre tilgangsnivået til biblioteket 7
Finne tekst 7
Finne og erstatte tekst 8
Lukke aktuell funksjon eller program 8
Kjøre programmer og behandle funksjoner 8
Lese verdier inn i et program 11
Vise informasjon 14
Bruke lokale variabler 16
Forskjeller mellom funksjoner og programmer 18
Kalle opp et program fra et annet 19
Kontrollere flyten i en funksjon eller et program 20
Bruke If, Lbl og Goto for å kontrollere programflyt 20
Bruke loop for å gjenta en gruppe med kommandoer 23
Endre modusinnstillinger 27
Løse håndteringsfeil og feil i programmer 27

Generell informasjon **29**

Komme i gang med Program Editor

Du kan opprette egendefinerte funksjoner eller programmer ved å skrive inn definisjonsutsagn på kommandolinjen til Kalkulator eller ved å bruke Program Editor Program Editor har noen fordeler, og det er disse dette avsnittet handler om Les avsnittet *Kalkulator* for mer informasjon.

- Editor-funksjonen har programmeringssjabloner og dialogbokser som hjelper deg å definere funksjoner og programmer med riktig syntaks
- Program Editor lar deg legge inn programmeringsutsagn over flere linjer uten at du trenger å legge til en spesiell tastsekvens etter hver linje.
- Du kan enkelt opprette private og felles bibliotekobjekter (variabler, funksjoner og programmer). Se *Biblioteker* for mer informasjon.

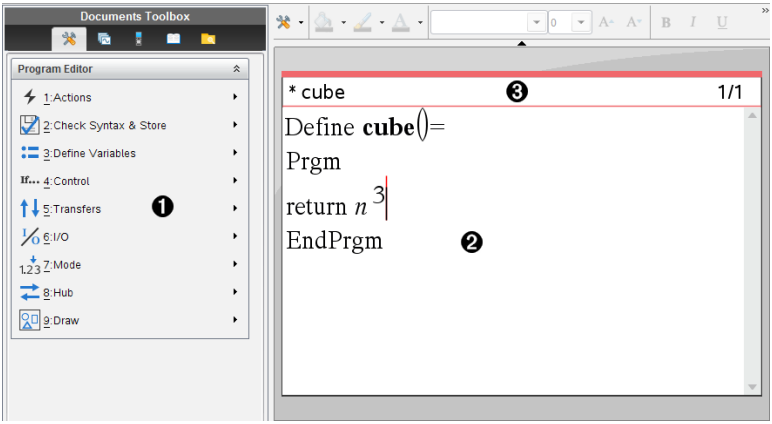
Starte Program Editor

▸ For å legge til en ny Program Editor-side i gjeldende oppgave:

Bruk verktøylinjen og trykk **Sett inn > Program Editor > Ny**.

Grafregner: Trykk [doc▾] og velg **Sett inn > Program Editor > Ny**.

Merk: Program Editor er også tilgjengelig fra **Funksjoner & Programmer** menyen på en kalkulator-side.



❶ Program Editor-meny – Denne menyen er alltid tilgjengelig når du befinner deg i arbeidsområdet til Programeditor og bruker normal visningsmodus.

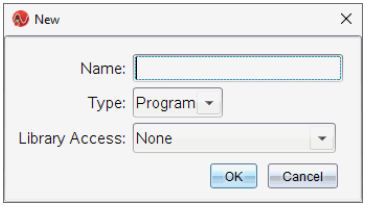
❷ Arbeidsområde i Program Editor

❸ Statuslinjen viser linjenummerinformasjon og navnet på den funksjonen eller det programmet som blir redigert En stjerne (*) viser at denne funksjonen er "uren", som betyr at den er blitt endret siden siste gang syntaksen ble kontrollert og funksjonen ble lagret.

Definere et program eller en funksjon

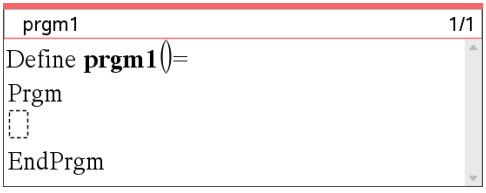
Starte en ny Programeditor

1. Pass på at du er i dokumentet og oppgaven der du vil opprette programmet eller funksjonen.
2. Klikk på **Sett inn**-knappen på programverktøylinjen, og velg **Programeditor > Ny**. (På grafregneren trykker du på doc▾ og velger **Sett inn > Programeditor > Ny**.)



3. Skriv inn et navn for funksjonen eller programmet som du definerer.
4. Velg **Type** (**Program** eller **Funksjon**).
5. Still inn **bibliotektilgang**:
   - Hvis du kun vil bruke funksjonen eller programmet fra det aktuelle dokumentet og oppgaven, velg **Ingen**.
   - Hvis du vil at funksjonen eller programmet skal være tilgjengelig fra hvilket som helst dokument, men ikke synlig i katalogen, velg **LibPriv**.
   - Hvis du vil at funksjonen eller programmet skal være tilgjengelig fra hvilket som helst dokument og også synlig i katalogen, velg **LibPub (Vis i Katalog)**. Se *Biblioteker* for mer informasjon.
6. Klikk på **OK**.

   Et nytt element i Programeditor åpnes, med en sjablon som passer til de valgene du gjorde.



Legge inn linjer i en funksjon eller et program

Programeditoren utfører ikke kommandoer eller beregner uttrykk etter hvert som du legger dem inn. De utføres kun når du behandler funksjonen eller kjører programmet.

1. Hvis din funksjon eller ditt program krever at brukeren tilføyer argumenter, skriv inn parameternavn i parentesen som kommer etter navnet. Skill parametere med et komma.

```
* prgm1                                  0/1
Define prgm1(a,b)=
Prgm
EndPrgm
```

2. Mellom linjene Func og EndFunc (eller Prgm og EndPrgm), legg inn utsagnslinjene som utgjør din funksjon eller ditt program.

```
* prgm1                                  3/3
Define prgm1(a,b)=
Prgm
Disp "a=",a
Disp "b=",b
Disp "a^b=",a^b
EndPrgm
```

- Du kan enten skrive inn navnene på funksjonene og kommandoene eller sette dem inn fra Katalogen.
- En linje kan være lengre enn skjermens bredde. Hvis det er tilfelle, må du kanskje bla for å se hele utsagnet.
- Etter at du har skrevet inn hver linje, trykk på **Enter**. På den måten får du en ny, tom linje, og du kan fortsette å legge inn en ny linje.
- Bruk piltastene ◄, ►, ▲, og ▼ for å bla gjennom funksjonen eller programmet for å legge inn eller redigere kommandoer.

Sette inn kommentarer

Kommentarer kan være nyttige for den som ser eller redigerer programmet. De vises ikke mens programmet kjører og har ingen virkning på programflyten. ©-symbolet vises ved begynnelsen av linjen med kommentaren.

```
* volcyl                                   3/3
Define LibPub volcyl(ht,r)=
Prgm
©volcyl(ht,r) => volume of cylinder ❶
Disp "Volume=",approx(π·r^2·ht)
©This is another comment.
EndPrgm
```

❶ Kommentar som viser nødvendig syntaks. Siden dette bibliotekobjektet er felles og denne kommentaren er den første linjen i en Funk- eller Prgm-blokk, vises kommentaren i katalogen som hjelp. Se *Biblioteker* for mer informasjon.

For å sette inn en kommentar:

1. Plasser markøren på slutten av linjen der du vil sette inn en kommentar.
2. Fra **Handlinger**-menyen klikker du på **Sett inn kommentar** eller trykker på **Ctrl+T**.
3. Skriv inn kommentarteksten etter ©-symbolet.

Kontrollere syntaks

Programeditor lar deg kontrollere om funksjonen eller programmet har riktig syntaks.

- Fra **Kontroller syntaks og lagre** -menyen, velg **Kontroller syntaks**.

  Dersom syntakskontrollen oppdager syntaksfeil, vises en feilmelding, og kontrollen prøver å plassere markøren nær den første feilen så du kan korrigere den.

```
* prgm1                                    3/3
Define prgm
Prgm
Disp "a=",a
Disp "b=",b
Disp "a^b=",a^b
EndPrgm
```

Error
Syntax
OK

Lagre funksjon eller program

Du må lagre funksjonen eller programmet for å gjøre dem tilgjengelige. Programeditor kontrollerer automatisk syntaksen før lagring.

En stjerne (*) vises i det øvre, venstre hjørnet i Programeditor for å fortelle at funksjonen eller programmet ikke er blitt lagret.

- Fra **Kontroller syntaks og lagre** -menyen, velg **Kontroller syntaks og lagre**.

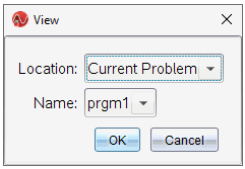
  Dersom syntakskontrollen oppdager syntaksfeil, vises en feilmelding, og kontrollen prøver å plassere markøren nær den første feilen.

  Hvis ingen syntaksfeil blir funnet, vises meldingen «Lagret» på statuslinjen øverst i Programeditor.

Merk: Hvis funksjonen eller programmet er definert som et bibliotekobjekt, må du også lagre dokumentet i den utpekte bibliotek-mappen og oppdatere bibliotekene for å gjøre objektet tilgjengelig for andre dokumenter. Se *Biblioteker* for mer informasjon.

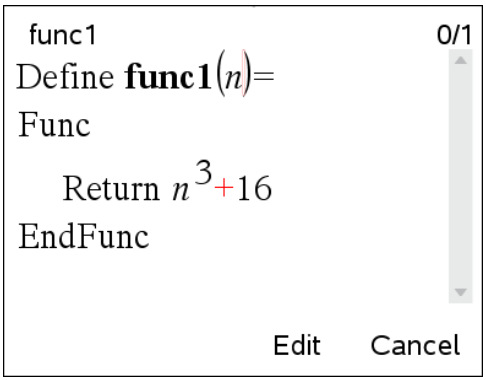
Vise et program eller en funksjon

1. Fra **Handlinger** -menyen, velg **Vis**.



2. Dersom funksjonen eller programmet er et bibliotekobjekt, velg biblioteket fra **Plassering**-listen.
3. Velg funksjonens eller programmets navn fra **Navn**-listen.

   Funksjonen eller programmet vises i et vindu.



4. Bruk piltastene for å vise funksjonen eller programmet.
5. Hvis du vil redigere programmet, klikk på **Rediger**.

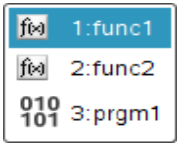
Merk: Valget **Rediger** er kun tilgjengelig for funksjoner og programmer som er definert i den aktuelle oppgaven. For å redigere et bibliotekobjekt, må du først åpne bibliotekdokumentet.

Åpne en funksjon eller et program for redigering

Du kan kun åpne en funksjon eller et program fra den aktuelle oppgaven.

Merk: Du kan ikke modifisere et låst program eller en funksjon. For å låse opp et objekt, gå til Kalkulator-siden og bruk **unLock**-kcommandoen.

1. Vise listen over tilgjengelige funksjoner og programmer.
   - Fra **Handlinger** -menyen, velg **Åpne**.

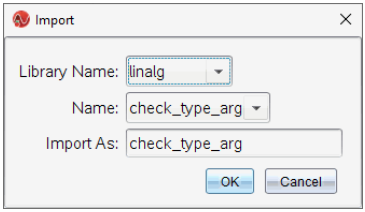


2. Velg det elementet du vil åpne.

Importere et program fra et bibliotek

Du kan importere en funksjon eller et program som er definert som et bibliotekobjekt inn i en Programeditor i den aktuelle oppgaven. Den importerte kopien er ikke låst, selv om originalen er låst.

1. Fra **Handlinger** -menyen, velg **Importer**.



2. Velg **biblioteknavn**.
3. Velg **navnet** på objektet.
4. Hvis du vil at det importerte objektet skal ha et annet navn, skriv navnet inn under **Importer som**.

Opprette en kopi av en funksjon eller et program

Når du oppretter en ny funksjon eller et nytt program, vil du kanskje synes at det er lettere å starte med en kopi av den/det aktuelle. Den kopien som du oppretter er ikke låst, selv om originalen er låst.

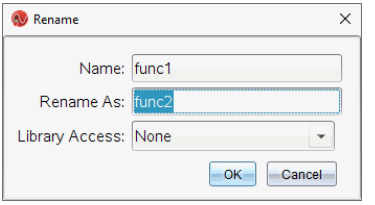
1. Fra **Handlinger** -menyen, velg **Opprett kopi**.
2. Skriv inn et nytt navn, eller klikk på **OK** for å akseptere det foreslåtte navnet.

3. Hvis du vil endre tilgangsnivået, velg **Bibliotektilgang**, og velg et nytt nivå.

Gi nytt navn til et program eller en funksjon

Du kan gi nytt navn til og (alternativt) endre tilgangsnivået til aktuell funksjon eller program.

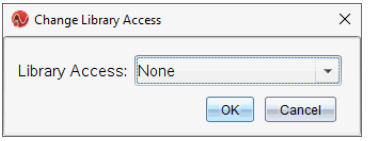
1. Fra **Handlinger** -menyen, velg **Gi nytt navn**.



2. Skriv inn et nytt navn, eller klikk på **OK** for å akseptere det foreslåtte navnet.
3. Hvis du vil endre tilgangsnivået, velg **Bibliotektilgang**, og velg et nytt nivå.

Endre tilgangsnivået til biblioteket

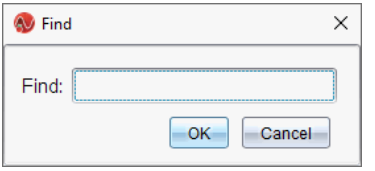
1. Fra **Handlinger** -menyen, velg **Endre bibliotektilgang**.



2. Velge **Bibliotektilgang**:
   - Hvis du kun vil bruke funksjonen eller programmet fra aktuell Kalkulator-oppgave, velg **Ingen**.
   - Hvis du vil at funksjonen eller programmet skal være tilgjengelig fra andre dokumenter, men ikke synlig i Katalogen, velg **LibPriv**.
   - Hvis du vil at funksjonen eller programmet skal være tilgjengelig fra andre dokumenter og også synlig i Katalogen, velg **BiblOff**.

Finne tekst

1. Fra **Handlinger** -menyen, velg **Finn**.

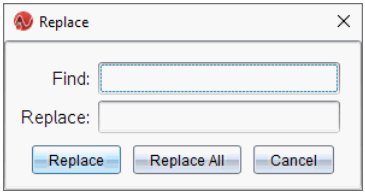


2. Skriv inn den teksten du vil finne, og klikk på **OK**.
   - Hvis teksten blir funnet, er den markert i programmet.

- Hvis teksten ikke blir funnet, vises en melding.

Finne og erstatte tekst

1. Fra **Handlinger** -menyen, velg **Finn og erstatt**.



2. Skriv inn teksten du vil finne.
3. Skriv inn den nye teksten.
4. Klikk på **Erstatt** hvis du vil erstatte den første forekomsten etter markørposisjonen, eller klikk på **Erstatt alle** hvis du vil erstatte alle forekomster.

   Merk: Hvis teksten blir funnet i en matematisk sjablon, får du en advarsel om at erstatningsteksten vil erstatte hele sjablonen, ikke bare teksten som ble funnet.

Lukke aktuell funksjon eller program

▸ Fra **Handlinger** -menyen, velg **Lukk**.

Hvis funksjonen eller programmet inneholder endringer som ikke er lagret, blir du bedt om å kontrollere syntaks og lagre før lukking.

Kjøre programmer og behandle funksjoner

Etter å ha definert og lagret et program eller en funksjon, kan du bruke det fra en applikasjon. Alle applikasjonene kan evaluere funksjoner, men kun Kalkulator- og Notat-applikasjonen kan kjøre programmer.

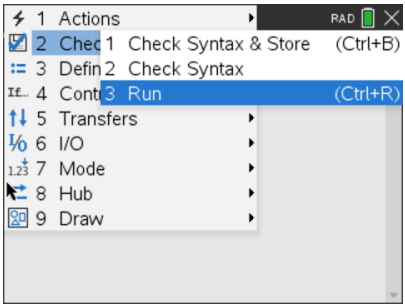
Programutsagnene utføres i sekvensrekkefølge (selv om noen kommandoer endrer programflyten). Utmatingen, om noen, vises i applikasjonens arbeidsområde.

- Programutførelse fortsetter inntil det når det siste utsagnet eller en **Stopp**-kommando.
- Funksjonsutførelsen fortsetter inntil den når en **Retur**-kommando.

Kjøre et program eller en funksjon fra Programeditor

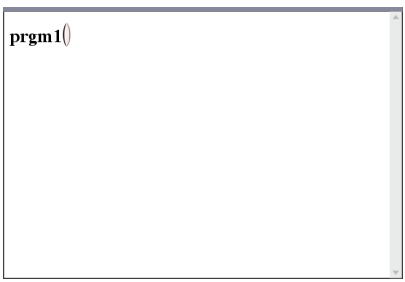
1. Pass på at du har definert et program eller en funksjon, og at Programeditor er den aktive fanen (datamaskinen) eller siden (grafregner).
2. På verktøylinjen klikker du på **Dokumentverktøy**-knappen og velg **Kontroller Syntaks og lagre > Kjør**.
   —eller—
   Trykk på **Ctrl+R**.

   Grafregner: Trykk på [menu] [2] [3], eller trykk på [ctrl] [R].



Dette vil automatisk:

- kontrollere syntaksen og lagre programmet eller funksjonen,
- lime inn program- eller funksjonsnavnet på den første tilgjengelige linjen i Kalkulator-applikasjonen umiddelbart etter Programeditor. Hvis det ikke eksisterer noen Kalkulator i den posisjonen, settes det inn en ny.



3. Hvis programmet eller funksjonen krever at du tilføyer ett eller flere argumenter, skriver du inn verdiene eller variabelnavnene inne i parentesen.

4. Trykk på enter.

Merk: Du kan også kjøre et program eller en funksjon i Kalkulator-eller Notat-applikasjonen ved å skrive navnet på programmet med parenteser og eventuelle nødvendige argumenter og trykke på enter.

Bruke korte og lange navn

Når du er i samme problem der et objekt er definert, kan du få tilgang til det ved å skrive inn dets korte navn (navnet oppgitt i objektets **Define**-kommando). Dette er tilfellet for alle definerte objekter, inkludert private, offentlige og ikke-bibliotekobjekter.

Du kan få tilgang til et bibliotekobjekt fra et hvilket som helst dokument ved å skrive inn objektets lange navn. Et langt navn består av navnet på objektets bibliotekdokument etterfulgt av en omvendt skråstrek "\" etterfulgt av navnet på objektet. For eksempel er det lange navnet på objektet definert som **funk.1** i

bibliotekdokumentet **bib1 bib1\funk.1**. For å skrive "\"-tegnet på grafregneren, trykker du på [⇧shift] [÷].

Merk: Hvis du ikke kan huske det nøyaktige navnet eller rekkefølgen for argumentene som er nødvendig for et privat bibliotekobjekt, kan du åpne bibliotekdokumentet eller bruke Programeditor for å vise objektet. Du kan også bruke **getVarInfo** for å se en liste over objekter i et bibliotek.

Bruke et offentlig bibliotekprogram eller -funksjon

1. Forsikre deg om at du har definert objektet i dokumentets første problem, arkivert objektet, lagret bibliotekdokumentet i MyLib- mappen og oppdatert bibliotekene.
2. Åpne TI- Nspire™-applikasjonen du vil bruke programmet eller funksjonen i.

   Merk: Alle applikasjoner kan behandle funksjoner, men bare Kalkulator- og Notat-applikasjonen kan kjøre programmer.

3. Åpne Katalogen og bruk bibliotekfanen for å finne og sette inn objektet.
   —eller—
   Skriv navnet på objektet. Hvis det dreier seg om et program eller en funksjon, setter du alltid parenteser etter navnet.

   ```
   bib.2\funk.1()
   ```

4. Hvis programmet eller funksjonen krever at du tilføyer ett eller flere argumenter, skriver du inn verdiene eller variabelnavnene inne i parentesen.

   ```
   bib.2funk.1(34,effekt)
   ```

5. Trykk på [enter].

Bruke privat bibliotekprogram eller -funksjon

For å bruke et privat bibliotekobjekt, må du kjenne det lange navnet. For eksempel er det lange navnet på objektet definert som **funk.1** i bibliotekdokumentet **bib1 bib1\funk.1**.

Merk: Hvis du ikke kan huske det nøyaktige navnet eller den rekkefølgen som argumentene må stå i for et privat bibliotekobjekt, kan du åpne bibliotekdokumentet eller bruke Programeditor for å vise objektet.

1. Forsikre deg om at du har definert objektet i dokumentets første problem, arkivert objektet, lagret bibliotekdokumentet i MyLib- mappen og oppdatert bibliotekene.
2. Åpne TI- Nspire™-applikasjonen du vil bruke programmet eller funksjonen i.

   Merk: Alle applikasjoner kan behandle funksjoner, men bare Kalkulator- og Notat-applikasjonen kan kjøre programmer.

3. Skriv navnet på objektet. Hvis det dreier seg om et program eller en funksjon, setter du alltid parenteser etter navnet.

   ```
   bib.2\funk.1()
   ```

4. Hvis objektet krever at du tilføyer ett eller flere argumenter, skriver du inn verdier eller variabelnavn i parentesen.

```
bib.2funk.1(34,effekt)
```

5. Trykk på [enter].

Avbryte et program som kjører eller en funksjon

Mens et program eller en funksjon kjører, vises opptatt-pekeren ⏲.

▶ For å stoppe programmet eller funksjonen,

- Windows®: Hold nede tasten **F12**, og trykk på **Enter** flere ganger.
- Mac®: Hold nede tasten **F5**, og trykk på **Enter** flere ganger.
- Grafregner: Hold nede tasten [on], og trykk på [enter] flere ganger.

En melding vises. For å redigere programmet eller funksjonen i Programeditor, velger du **Gå til**. Markøren vises ved kommandoen hvor pausen skjedde.

Lese verdier inn i et program

Du kan velge mellom mange metoder for å lese inn de verdiene som en funksjon eller et program bruker i en beregning.

Integrere verdiene i programmet eller funksjonen

Denne metoden er først og fremst nyttig for verdier som må være de samme hver gang programmet eller funksjonen blir brukt.

1. Definere programmet.

```
* calculatearea                    4/4
Define calculatearea()=
Prgm
w:=3
h:=23.64
area:=w·h
Disp area
EndPrgm
```

2. Kjør programmet.

```
calculatearea()
                                    70.92
                                    Done
```

La brukeren tildele verdiene til variabler

Et program eller en funksjon kan henvise til variabler som er blitt opprettet på forhånd. Denne metoden krever at brukeren husker variabelnavnene og tildeler verdier til dem før objektet blir brukt.

1. Definere programmet.

```
* calculatearea                                    2/2
Define calculatearea ()=
Prgm
area:=w· h
Disp area
EndPrgm
```

2. Les inn variablene, og kjør så programmet.

```
w:=3                                                3
h:=23.64                                        23.64
calculatearea()
                                    70.92
                                    Done
```

La brukeren lese inn verdiene som argumenter

Denne metoden lar brukeren sende en eller flere verdier som argumenter innenfor uttrykket som kaller opp programmet eller funksjonen.

Følgende program, **volcyl**, beregner volumet av en sylinder. Det krever at brukeren leser inn to verdier: sylinderens høyde og radius.

1. Definere **volcyl**-programmet.

```
* volcyl                                   1/1
Define volcyl(height,radius)=
Prgm
Disp "Volume =", approx(π· radius^2· height)
EndPrgm
```

2. Kjør programmet for å vise volumet av en sylinder med en høyde på 34 mm og en radius på 5 mm.

```
volcyl(34,5)
                          Volume =2670.35
                                    Done
```

Merk: Du trenger ikke å bruke parameternavnene når du kjører **volcyl**-programmet, men du må tilføye to argumenter (som verdier, variabler eller uttrykk). Den første må representere høyden, og den andre må representere radius.

Forespørre verdiene fra brukeren (gjelder kun programmer)

Du kan bruke **Request** og **RequestStr** -kommandoene i et program for å stanse programmet og vise en dialogboks som ber brukeren om informasjon. Denne metoden krever ikke at brukeren husker variabelnavn eller i hvilken rekkefølge det er behov for dem.

Du kan ikke bruke **Request** eller **RequestStr** -kommandoen i en funksjon.

1. Definere programmet.

```
* calculatearea                                  3/3
Define calculatearea()=
Prgm
Request "Width: ", w
Request "Height: ", h
area:=w·h
EndPrgm
```

2. Kjør programmet og svar på forespørslene.

```
calculatearea(): area

Width:  3
Height:  23.64

                                          70.92
```

Bruk **RequestStr** istedenfor **Request** når du vil at programmet skal tolke brukerens svar som en tegnstreng istedenfor et matematisk uttrykk. Dermed må ikke brukeren sette svaret i anførselstegn (“”).

Vise informasjon

En funksjon eller et program som kjører viser ikke beregnede mellomresultater hvis du ikke legger til en kommando for å vise dem. Dette er en viktig forskjell mellom å uføre en beregning på kommandolinjen og utføre den i en funksjon eller et program.

Følgende beregninger viser for eksempel ikke et resultat i en funksjon eller et program (selv om de gjør dette fra kommandolinjen).

```
prgm2                                   0/2
Define prgm2()=
Prgm
x:=12·6
cos(π/4)
EndPrgm
```

Vise informasjon i loggen

Du kan bruke **Disp** -kommandoen i et program eller en funksjon for å vise informasjon, inkludert mellomresultater, i loggen.

```
* prgm2                                 2/2
Define prgm2()=
Prgm
Disp 12·6
Disp "Result:",cos(π/4)
EndPrgm
```

Vise informasjon i en dialogboks

Du kan bruke **Text** -kommandoen til å stanse et program som kjører og vise informasjon i en dialogboks. Brukeren velger **OK** for å fortsette, eller velger **Avbryt** for å stoppe programmet.

Du kan ikke bruke **Text** -kommandoen i en funksjon.

```
* sample                                 1/1
Define sample()=
Prgm
Text "Area=" & area
EndPrgm
```

Merk: Selv om et resultat vises med **Disp** eller **Text**, blir ikke resultatet lagret. Hvis du vil referere til et resultat senere, må du lagre det til en global variabel.

```
* sample                                  2/2
Define sample()=
Prgm
cos(π/4) → maximum
Disp maximum
EndPrgm
```

```
sample()
                                  0.707107
                                      Done
```

Bruke lokale variabler

En lokal variabel er en midlertidig variabel som kun eksisterer mens en egendefinert funksjon er i ferd med å bli behandlet eller et egendefinert program kjører.

Eksempel på en lokal variabel

Følgende programsegment viser en **For...EndFor loop** (som blir drøftet senere i denne modulen). Variabelen i er loop-telleren. I de fleste tilfeller brukes variabelen i kun mens programmet kjører.

```
* loop_prog                                  0/5
Define loop_prog()=
Prgm
Local i  ❶
For i,0,5,1
  Disp i
EndFor
Disp i
EndPrgm
```

❶ Erklærer variabelen i som lokal.

Merk: Hvis mulig, erklær som lokal alle variabler som kun brukes i programmet og som ikke trenger å være tilgjengelige etter at programmet stopper.

Hva er årsaken til en feilmelding om udefinert variabel?

En feilmelding om **udefinert** variabel kommer til syne når du behandler en egendefinert funksjon eller kjører et egendefinert program som refererer til en lokal variabel som ikke er blitt initialisert (tildelt en verdi).

For eksempel:

```
* fact                                        5/5
Define fact(n)=
Func
Local m  ❶
While n>1
  n·m→m: n−1→n
EndWhile
Return m
EndFunc
```

❶ Lokal variabel m er ikke tildelt en utgangsverdi.

Initialisere lokale variabler

Alle lokale variabler må være tildelt en utgangsverdi før det kan refereres til dem.

```
* fact                                  5/5
Define fact(n)=
Func
Local m: 1→m  ❶
While n>1
  n·m→m: n−1→n
EndWhile
Return m
EndFunc
```

❶ 1 er lagret som utgangsverdien for m.

Merk (CAS): Funksjoner og programmer kan ikke bruke en lokal variabel for å utføre symbolske beregninger.

CAS: Utføre symbolske beregninger

Hvis du vil at en funksjon eller et program skal utføre symbolske beregninger, må du bruke en global variabel istedenfor en lokal. Du må imidlertid være sikker på at den globale variabelen ikke eksisterer allerede utenfor programmet. Følgende metoder kan være en hjelp.

- Referer til et globalt variabelnavn, vanligvis med to eller flere tegn, som sannsynligvis ikke eksisterer utenfor funksjonen eller programmet.
- Inkluder **SlettVar** i et program for å slette den globale variabelen, hvis den eksisterer, før det blir referert til den. (**SlettVar** sletter ikke låste eller forbundne variabler.)

Forskjeller mellom funksjoner og programmer

En funksjon som er definert i Programeditor, er lik de funksjonene som er bygget inn i TI-Nspire™ -programvaren.

- Funksjoner må returnere et resultat som kan graftegnes eller settes inn i en tabell. Programmer kan ikke returnere et resultat.
- Du kan bruke en funksjon (men ikke et program) inne i et uttrykk. For eksempel: **3 • func1(3)** er gyldig, men ikke **3 • prog1(3)**.
- Du kan kun kjøre programmer fra Kalkulator- og Notes-applikasjonene. Men du kan behandle funksjoner i Kalkulator, Notes, Lister og regneark, Grafer og geometri, og Data og statistikk.
- En funksjon kan referere til enhver variabel; men den kan kun lagre en verdi til en lokal variabel. Programmer kan lagre til lokale og globale variabler.

  Merk: Argumenter som pleide å sende verdier til en funksjon, blir automatisk behandlet som lokale variabler. Hvis du vil lagre til en annen vilkårlig variabel, må du erklære dem som **local** innenfra i funksjonen.

- En funksjon kan ikke kalle opp et program som en delrutine, men den kan kalle opp en annen egendefinert funksjon.
- Du kan ikke definere et program inne i en funksjon.
- En funksjon kan ikke definere en global funksjon, men den kan definere en lokal.

Kalle opp et program fra et annet

Et program kan kalle opp et annet program som en delrutine. Delrutinen kan være ekstern (et eget program) eller intern (inkludert i hovedprogrammet). Delrutiner kan være nyttige når et program må gjenta den samme gruppen med kommanoder på forskjellige steder.

Kalle opp et separat program

For å kalle opp et separat program, bruk den samme syntaksen som du bruker for å kjøre programmet fra kommandolinjen.

```
Define subtest1()=
Prgm
 For i,1,4,1
  subtest2(i,i*1000)
EndFor
EndPrgm
```

```
Define subtest2(x,y)=
Prgm
 Disp x,y
EndPrgm
```

Definere og kalle opp en intern delrutine

For å definere en intern delrutine, bruk **Define**-kommandoen med **Prgm...EndPrgm**. Fordi en delrutine må defineres før den kan kalles opp, er det lurt å definere delrutinene når hovedprogrammet startes.

En intern delrutine blir kalt opp og utført på samme måte som et separat program.

```
* subtest1                                  9/9
Define subtest1()=
Prgm
  Local subtest2 ❶
  Define subtest2(x,y) ❷
  Prgm
    Disp x,y
  EndPrgm
© Beginning of main program
For i,1,4,1
  subtest2(i,i·1000) ❸
EndFor
EndPrgm
```

❶ Erklærer delrutinen som en lokal variabel.

❷ Definerer delrutinen.

❸ Kaller opp delrutinen.

Merk: Bruk Programeditorens **Var** -meny for å legge inn **Define**r og **Prgm...EndPrgm**-kommandoene.

Merknader til bruk av delrutiner

Ved slutten av en delrutine returnerer utførelsen til det oppkallende programmet. For å avbryte en delrutine før den er slutt, bruk **Return** uten argumenter.

En delrutine har ikke tilgang til lokale variabler som er erklært i oppkallingsprogrammet. På samme måte har oppkallingsprogrammet ingen tilgang til lokale variabler som er erklært i en delrutine.

Lbl-kommandoer er lokale til de programmene som de er plassert i. Derfor kan en **Goto**-kommando i oppkallingsprogrammet ikke knyttes til et navn i en delrutine eller omvendt.

Unngå sirkulære definisjonsfeil

Når du behandler en egendefinert funksjon eller kjører et program, kan du spesifisere et argument som inkluderer den samme variabelen som ble brukt for å definere funksjonen eller opprette programmet. Men for å unngå sirkulære definisjonsfeil, må du tildele en verdi for variabler som blir brukt når du behandler funksjonen eller kjører programmet. For eksempel:

$x+1 \rightarrow x$ ❶

– eller –

```
For i,i,10,1
  Disp i ❶
EndFor
```

❶ Forårsaker en **sirkulær definisjons**-feilmelding hvis x eller i ikke har noen verdi. Feilen oppstår ikke hvis x eller i allerede er blitt tildelt en verdi.

Kontrollere flyten i en funksjon eller et program

Når du kjører et program eller behandler en funksjon, utføres programlinjene i sekvensrekkefølge. Men noen kommandoer endrer programflyten. For eksempel:

- Kontrollstrukturer, som f.eks. **If...EndIf**-kommandoer bruker en betingelsestest for å bestemme hvilken del av programmet som skal utføres.
- Loop-kommandoer, som f.eks. **For...EndFor** gjentar en gruppe av kommandoer.

Bruke If, Lbl og Goto for å kontrollere programflyt

If-kommandoen og flere **If...EndIf**-strukturer lar deg utføre et betinget utsagn eller en blokk av utsagn, det vil si, basert på resultatet av en test (som f.eks. x>5). **Lbl** (etikettnavn) og **Goto**-kommandoene lar deg forgreine eller hoppe fra et sted til et annet i en funksjon eller et program.

If-kommandoen og flere **If...EndIf**-strukturer finner du i Programeditorens **Kontroll** -meny.

Når du setter inn en struktur, som f.eks. **If...Then...EndIf**, settes det inn en sjablon der markøren befinner seg. Markøren er plassert slik at du kan legge inn en betingelsestest.

If-kommando

For å utføre en enkel kommando dersom en betingelse er sann, bruk den generelle formen:

```
If x>5
  Disp "x is greater than 5" ❶
Disp x ❷
```

❶ Utføres kun hvis x>5; ellers forkastes den.

❷ Viser alltid verdien av x.

I dette eksemplet må du lagre en verdi til x før du kan utføre **If** -kommandoen.

If...Then...EndIf-strukturer

For å utføre en gruppe av kommandoer dersom en betingelsestest er sann, bruk strukturen:

```
If x>5 Then
  Disp "x is greater than 5" ❶
  2·x→x ❶
EndIf
Disp x ❷
```

❶ Utføres kun hvis x>5.

❷ Viser verdien av:
2x if x>5
x if x≤5

Merk: EndIf markerer slutten på **Then**-blokken som er utført dersom betingelsen er sann.

If...Then...Else... EndIf-strukturer

For å utføre en gruppe av kommandoer dersom en betingelsestest er sann og en annen gruppe dersom betingelsen er usann, bruk denne strukturen:

```
If x>5 Then
  Disp "x is greater than 5 " ❶
  2·x→x ❶
Else
  Disp "x is less than or equal to 5 " ❷
  5·x→x ❷
EndIf
Disp x ❸
```

❶ Utføres kun hvis x>5.

❷ Utføres kun hvis x≤5.

❸ Viser verdien av:
2x if x>5
5x if x≤5

If...Then...ElseIf... EndIf-strukturer

En mer kompleks form av If-kommandoen lar deg teste flere betingelser. Anta at du vil at et program skal teste et egentilføyd argument som innebærer ett av fire alternativer.

For å teste for hvert alternativ (If Choice=1, If Choice=2, og så videre), bruk **If...Then...ElseIf...EndIf**-strukturen.

Lbl og Goto -kommandoer

Du kan også kontrollere flyten ved å bruke **Lbl** (navn) og **Goto** -kommandoer. Disse kommandoene finner du i Programeditorens **Overføringer** -meny.

Bruk **Lbl** -kommandoen for å navngi (tildele et navn til) en spesiell posisjon i funksjonen eller programmet.

Lbl *labelName*	navn som skal tildeles til denne posisjonen (bruk de samme reglene for navngivning som for variabelnavn)

Deretter kan du bruke **Goto** -kommandoen på et vilkårlig punkt i funksjonen eller programmet for å forgreine til den posisjonen som samsvarer med det spesifiserte etikettnavnet.

Goto *labelName*	spesifiserer hvilken **Lbl** -kommando det skal forgreines til

Siden en **Goto** -kommando ikke er betinget (den forgreiner alltid til det spesifiserte navnet), blir den ofte brukt med en **If** -kommando slik at du kan spesifisere en betingelsestest. For eksempel:

```
If x>5
  Goto GT5 ❶
Disp x

------------

Lbl GT5 ❷
Disp "The number was > 5"
```

❶ If x>5, forgreiner direkte til etikettnavn GT5.

❷ For dette eksemplet må programmet inkludere kommandoer (som f.eks. **Stop**) som hindrer **Lbl** GT5 i å bli utført hvis x≤5.

Bruke loop for å gjenta en gruppe med kommandoer

For å gjenta den samme gruppen av kommandoer etter hverandre, bruk en av loop-strukturene. Det er flere loop-typer tilgjengelig. Alle typene har forskjellige måter å avslutte loopen på, basert på en betingelsestest.

Loop og loop-relaterte kommandoer finner du i Programeditorens **Kontroll** og **Overføringer** -menyer.

Når du setter inn en av loop-strukturene, settes sjablonen inn ved markørens posisjon. Så kan du begynne å legge inn de kommandoene som vil bli utført innenfor loopen.

For...EndFor-loop

En **For...EndFor**-loop bruker en teller for å kontrollere hvor mange ganger loopen gjentas. Syntaksen i **For** -kommandoen er:

Merk: Sluttverdien kan være mindre enn startverdien, dersom trinnet er negativt.

For *variabel*, *start*, *slutt* [, *trinn*]

❶ ❷ ❸ ❹

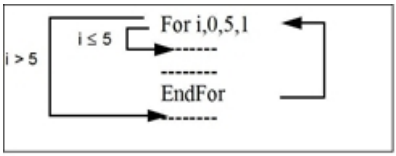
❶ *Variabel* brukt som teller

❷ Tellerverdi brukt den første gangen **For** utføres

❸ Avslutter loopen hvis *variabelen* overgår denne verdien

❹ Addert til telleren hver etterfølgende gang **For** utføres (dersom denne alternative verdien utelates, er *trinnet* 1.)

Hvis **For** utføres, blir *variabelens* verdi sammenliknet med *slutt*verdien. Hvis *variabelen* ikke overgår *sluttverdien*, utføres loopen; ellers hopper kontrollen til kommandoen som følger **EndFor**.



Merk: For -kommandoen øker tellervariabelens trinn automatisk, slik at funksjonen eller programmet kan avslutte loopen etter et visst antall gjentakelser.

På slutten av loopen (**EndFor**), hopper kontrollen tilbake til **For** -kommandoen, hvor variabelen er økt trinnvis og sammenliknet med *slutt*.

For eksempel:

```
For i,0,5,1
  Disp i ❶
EndFor
Disp i ❷
```

❶ Viser 0, 1, 2, 3, 4 og 5.

❷ Viser 6. Dersom *variabelen* økes trinnvis til 6, blir ikke loopen utført.

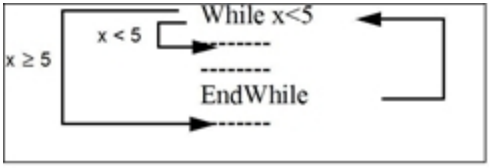
Merk: Du kan erklære tellervariabelen som lokal, dersom du ikke må lagre den etter at funksjonen eller programmet stopper.

While...EndWhile-looper

En **While...EndWhile**-loop gjentar en blokk av kommandoer så lenge en spesifisert betingelse er sann. Syntaksen i **While** -kommandoen er:

While -*betingelse*

Når **While** utføres, behandles *betingelsen*. Hvis *betingelsen* er sann, utføres loopen; ellers hopper kontrollen til kommandoen som følger **EndWhile**.



Merk: While -kommandoen endrer ikke betingelsen automatisk. Du må inkludere kommandoer som lar funksjonen eller programmet avslutte loopen.

På slutten av loopen (**EndWhile**), hopper kontrollen tilbake til **While** -kommandoen, hvor betingelsen behandles på nytt.

For å utføre loopen for første gang, må betingelsen være sann ved start.

- Alle variabelreferanser i betingelsen må være innstilt før **While** -kommandoen. (Du kan bygge verdiene inn i funksjonen eller programmet, eller du kan be brukeren om å legge inn verdiene.)
- Loopen må inneholde kommandoer som endrer verdiene i betingelsen, eventuelt forårsake at den blir usann. Ellers er betingelsen alltid sann og funksjonen eller programmet kan ikke avslutte loopen (kalt en infinit loop).

For eksempel:

```
0→x ❶
While x<5
  Disp x ❷
  x+1→x ❸
EndWhile
Disp x ❹
```

❶ Stiller inn x ved start.

❷ Viser 0, 1, 2, 3 og 4.

❸ Trinnvis x.

❹ Viser 5. Når x øker trinnvis til 5, utføres ikke loopen mer.

Loop...EndLoop-looper

En **Loop...EndLoop** oppretter en infinit loop, som gjentas uendelig. **Loop** -kommandoen har ikke noen argumenter.

```
Loop
--------
--------
EndLoop
--------
```

Vanligvis setter du inn kommandoer i den loopen som gjorde at programmet avsluttet fra loopen. Vanlig brukte kommandoer er: **If**, **Exit**, **Goto** og **Lbl** (navn). For eksempel:

```
0 → x
Loop
  Disp x
  x+1 → x
  If x>5 ❶
    Exit
EndLoop
Disp x ❷
```

❶ En **If** -kommando kontrollerer betingelsen.

❷ Avslutter loopen og hopper til hit når x øker trinnvis til 6.

Merk: Exit -kommandoen avslutter fra aktuell loop.

I dette eksemplet kan **If** -kommandoen være plassert hvor som helst i loopen.

Når If -kommandoen er:	Er loopen:
Ved starten av loopen	Utført kun hvis betingelsen er sann.
Ved slutten av loopen	Utført minst én gang og kun gjentatt hvis betingelsen er sann.

If -kommandoen kan også bruke en **Goto** -kommando for å overføre programkontrollen til en spesifisert **Lbl** (etikettnavn) -kommando.

Gjenta en loop øyeblikkelig

Cycle -kommandoen overfører programkontrollen øyeblikkelig til den neste iterasjonen i en loop (før den aktuelle iterasjonen er fullført). Denne kommandoen arbeider med **For...EndFor**, **While...EndWhile** og **Loop...EndLoop**.

Lbl og Goto-looper

Selv om **Lbl** (etikettnavn) og **Goto** -kommandoene ikke er gjennomførte loop -kommandoer, kan de brukes til å opprette en infinit loop. For eksempel:

```
Lbl START
--------
--------
Goto START
--------
```

Som for **Loop...EndLoop** bør loopen inneholde kommandoer som lar funksjonen eller programmet avslutte fra loopen.

Endre modusinnstillinger

Funksjoner og programmer kan bruke **setMode()** for å innstille midlertidig spesifikk beregningsmodus eller resultatmodus. Programeditorens **Modus** -meny gjør det enkelt å legge inn riktig syntaks uten å kreve at du må huske numeriske koder.

Merk: Modusendringer som er foretatt innenfor en funksjons- eller programdefinisjon, fortsetter ikke utenfor funksjonen eller programmet.

Stille inn en modus

1. Plasser markøren der hvor du vil sette inn **settModus** -funksjonen.
2. Fra **Modus** -menyen, velg den modusen du vil endre, og velg den nye innstillingen.

   Riktig syntaks er lagt inn ved markørens posisjon. For eksempel:

```
setMode(1,3)
```

Løse håndteringsfeil og feil i programmer

Når du har skrevet en funksjon eller et program, kan du bruke mange teknikker for å finne og korrigere feil. Du kan også bygge en feilhåndteringskommando inn i selve funksjonen eller programmet.

Hvis funksjonen eller programmet lar brukeren velge fra flere alternativer, pass på at du kjører den/det og test hvert alternativ.

Teknikker for løsning av feil

Meldinger som forteller om kjøretidsfeil kan lokalisere syntaksfeil, men ikke feil i programlogikken. Følgende teknikker kan være nyttige.

- Sett inn foreløpige **Disp** -kommandoer for å vise verdiene med de kritiske variablene.
- For å bekrefte at en loop er blitt utført riktig antall ganger, bruk **Disp** for å vise tellervariabelen eller verdiene i betingelsestesten.
- For å bekrefte at en delrutine er blitt utført, bruk **Disp** for å vise meldinger, som. f.eks. “Legge inn delrutine” og “Avslutte delrutine” ved begynnelsen og slutten av en delrutine.
- Stoppe et program eller en funksjon manuelt,
  - Windows®: Hold nede tasten **F12** og trykk på **Enter** flere ganger.
  - Macintosh®: Hold nede tasten **F5** og trykk på **Enter** flere ganger.
  - Grafregner: Hold nede tasten [on] og trykk på [enter] flere ganger.

Feilhåndteringskommandoer

Kommando	Beskrivelse
Try...EndTry	Definerer en blokk som lar en funksjon eller et program utføre en

Kommando	Beskrivelse
	kommando og, om nødvendig, gjenopprette fra en feil som oppsto på grunn av den kommandoen.
ClrErr	Tømmer feilstatus og stiller systemvariabelen *errCode* til null. For et eksempel på bruk av *errCode*, se kommandoen**Try** i referanseveiledningen.
SendFeil	Sender en feil til det neste nivået i **Try...EndTry**-blokken.

Generell informasjon

Hjelp på nettet (online)

education.ti.com/eguide

Velg ditt land for mer produktinformasjon.

Kontakt TIs brukerstøtte

education.ti.com/ti-cares

Velg ditt land for tekniske og andre støtteressurser.

Service og garantiinformasjoner

education.ti.com/warranty

Velg landet ditt for informasjon om lengden og vilkårene for garantien eller om produkttjenester.

Begrenset garanti. Denne garantien påvirker ikke dine lovmessige rettigheter.

Texas Instruments Incorporated

12500 TI Blvd.

Dallas, TX 75243