LINKÖPINGS UNIVERSITET

ETL

Easy To Learn

TDP019 Projekt: Datorspråk

Språkdokumentation

Författare

Ahmed Sikh , ahmsi881@student.liu.se
Sayed Ismail Safwat, saysa289@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Vårterminen 2021
Version 1.0

18 maj 2021

Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.0	Första version av Språkdokumentation	210510
1.1	Andra version av Språkdokumentation	210518

Innehåll

1 Inledning

Detta är ett projekt på IP-programmet som är skapat under den andra terminen vid Linköpings universitet i kursen TDP019 Projekt: datorspråk. ETL (Easy To Learn) har inspirerats för det mesta från Ruby språket. Det har utvecklats för en nybörjare och är skrivet på ett sätt som liknar skriftligt engelska vilket ökar språkets läsbarhet.

1.1 Syfte

Syftet med detta projektet var att visa vilka komponenter ett programmeringsspråk består av och hur ett nytt programmeringsspråk byggs upp med de där komponenterna.

1.2 Målgrupp

ETL språket skall passa de nybörjare som inte har några förkunskaper inom programmering. Det passar perfekt dem som vill börja lära sig programmering på rätt sätt eftersom det täcker de elementära grunderna i programmering. Språket kommer även passa de lärare som vill lära ut programmering till nybörjare eller möjligtvis till en grupp av barn i grundskolan.

2 Användarhandledning

Den här sektionen innehåller instruktioner för att hur man ska komma igång med språket samt lära känna språkkonstruktioner.

2.1 Installation

För att kunna testa ETL krävs den att senaste versionen av Ruby är installerad.

För att kunna köra språket måste det laddas ner. Språket kan laddas ner via länken:

https://gitlab.liu.se/ahmsi881/tdp019/-/archive/master/tdp019-master.zip

Användaren behöver skriva kommandoraden ***ruby ETL.rb*** för att kunna köra programmet.

Det finns två sätt att köra ETL-språket på:

1. Att skriva kod genom terminalen, vilket är ett sätt om användaren vill skriva endast en enkel rad kod som inte består av flera saker samtidigt. Detta kan användaren göra i ETL.rb genom: se **Figur 1**.

Figur 1: Exempel på hur ska det se ut när användaren vill testa språket genom terminalen

```
238  checkEtl = Etl.new
239  checkEtl.log(false)
240  checkEtl.activate_terminal
241  #checkEtl.activate_file("etl.etl")
242  checkEtl.output.each { |segment|
243      if segment.class != Function and segment.class != FunctionCall
244          segment.eval()
245      end }
246
247
```

2. Andra sättet är att testa språket i sin helhet vilket innebär att användaren skriver sin kod i en fil som heter **etl.etl** och där kommer programmet ta hand om resten. Detta kan användaren göra i ETL.rb genom: se **Figur 2**.

Figur 2: Exempel på hur ska det se ut när användaren vill testa språket genom en test fil

```
238  checkEtl = Etl.new
239  checkEtl.log(false)
240  #checkEtl.activate_terminal
241  checkEtl.activate_file("etl.etl")
242  checkEtl.output.each { |segment|
243      if segment.class != Function and segment.class != FunctionCall
244          segment.eval()
245      end }
246
```

2.2 Variabler och Tilldelning

Variabler har en dynamisk typning där användaren inte behöver specificera datatypen när den ska deklareras. Tilldelningen i ETL betecknas endast med tilldelningsoperatorn "=". I ETL går det att tilldela en variabel till jämförelse, strängar och matematiska uttryck.

Det innebär att det ska finnas endast ett namn och dennes värde vilket visas i följande stil:

```
x = 5
y = "Hej"
z = "hej" plus "då"
d = 5 < 10
```

2.3 Matematiska Operationer

ETL kan utföra alla sedvanliga matematiska beräkningar såsom addition, subtraktion, multiplikation, division, potenser och modolo samt deras rätta prioriteter och associativiteten, det vill säga division och multiplikation ska utföras före addition och subtraktion. Samtliga beräkningar utförs oavsett om de är heltal eller flyttal. Språket stöder även beräkningarna inuti en parentes.

Exempel:

```
(5 + 4)
1 - 5
2 * 1.0
5 / 5
4 - 7 * (10 / 2)
5 ^ 2
10 % 3
```

Det går även att utföra matematiska beräkningar på variabler som har heltal eller flyttal som värde.

Exempel:

```
x = 5
y = x + 2
z = x * y
```

2.4 Kommentarer

I språket finns det möjligheten att ignorera en rad eller flera rader ifall användaren inte vill att de raderna ska köras. Detta görs genom att skriva **"«"** för att ignorera en rad och för att ignorera fler rader måste det skrivas **"<comment"** i början av raden och **"<end"** i slutet av raden.

Exempel på flerradskommentar:

```
<comment
Detta är en flerradskommentar och allt som skrivs i det här utrymmet kommer ignoreras
och inte köras.
Som det syns här går det att skriva vad som helst. ?!"#€%&123456789
Det är jätteviktigt att inte glömma skriva <end i slutet av raden.
<end
```

Exempel på enkelradskommentar:

```
<< Här ignoreras bara en rad som skrevs med << i början av raden.
<< Varje rad måste ha << i början för att den ska ignoreras.
```

Kommenterar används ofta av programmerare som en påminnelse om hur de har kommit fram till den specifika koden.

2.5 Print

I ETL går det att skriva ut datatyper som strängar, tal, logiska uttryck och flera strängar samtidigt förutsatt att de är tilldelade till en variabel innan utskriften. För att skriva ut används ordet **write** innan variabelnamnet.

Exempel:

```
a = "Printing should be easy!"
write a
------------------------------
b = 3 < 4
write b
------------------------------
c = 1234
write c
```

Skriver ut följande:

```
-->> Printing 'Printing should be easy!'
----------------------------------------
-->> Printing 'true'
----------------------------------------
-->> Printing '1234'
```

2.6 Villkor/If-satser

Att skriva villkor eller if-satser i ETL språket är inte avancerad. Användaren bör börja med **"if"** i början av raden, sedan öppna en parentes där användaren kan skriva en eller flera logiska uttryck som kan ge **falskt** eller **sant**, efter det stänger användaren parentesen och skriver därefter ordet **"then"**. Då börjar användaren på en ny rad för att skriva den satsen eller de satserna som ska utföras ifall de logiska uttrycken som finns inuti parentesen ska returnera **sant**. I slutet av en if-sats ska användaren skriva **"endif"** för att säga att här slutar villkoret.

Exempel på en if-sats:

```
x = 7
y = 8
if (x > 6 and y == 8) then
write "if-sats fungerar"
endif
```

Skriver ut följande:

```
-->> Printing 'if-sats fungerar'
```

I ETL kan användaren skriva en elseif-sats som följer av en if-sats. Detta kan användaren göra genom att skriva **"elseif"** i en ny rad. Det betyder att om de logiska uttrycken som finns inuti **if-sats-parentes** returnerar **falsk**, så kommer **elseif-satsen** nu utföra den eller de satserna som finns efter ordet **"elseif"**.

Exempel på en elseif-sats:

```
j = 2
if (j != 2) then
write "if-sats fungerar"
elseif (j == 2) then
write "elseif-sats fungerar"
otherwise
write "otherwise fungerar"
endif
```

Skriver ut följande:

```
-->> Printing 'elseif-sats fungerar'
```

I ETL kan också användaren skriva en **else-sats** som följs av antingen en **if-sats** eller **elseif-sats**. Detta kan användaren göra genom att skriva **"otherwise"** i en ny rad. Det betyder att om de logiska uttrycken som finns inuti **if-sats** eller **elseif-sats-parentesen** returnerar **falsk**, så kommer **else-satsen** nu utföra den eller de satser som finns efter ordet **"otherwise"**. I slutet kommer användaren också att göra samma sak här, det vill säga att skriva **"endif"** för att visa att här slutar villkoret.

Exempel på en if-sats som följer av en else-sats:

```
x = 7
y = 8
if (x less than 6 or y equal 9) then
write "if-sats fungerar"
otherwise
write "otherwise-sats fungerar"
endif
```

Skriver ut följande:

```
-->> Printing 'otherwise-sats fungerar'
```

I ETL kan användaren skriva logiska operator i tecken.

Exempel 1:

<, >, <=, >=, **!=** och ==

eller att skriva logiska operator i ord, exempelvis:

less than, **greater than**, **less than or equal to**, **greater than or equal to**, **not equal to** och **equal**

ETL kan även hantera **or**, **and** och **not**, se exempel 1.

2.7 Iteration

Det finns en sorts loop i ETL-språket som kallas för en while-loop där användaren har möjlighet att iterera igenom exempelvis ett tal tills det villkoret i loopen har uppfyllts. För att skriva en while-loop skrivs först ordet **while** sedan villkoret inuti en parentes. Efter det går det att skriva den satsen eller de satserna när villkoret som finns inuti parentes uppfylls, därefter för att avsluta while-loopen måste ordet **endwhile** skrivas i en ny rad. Exempel:

```
y = 1
while ( y < 4)
write "while-loop fungerar"
y = y + 1
endwhile
```

Skriver ut följande:

```
-->> Printing 'while-loop fungerar'
-->> Printing 'while-loop fungerar'
-->> Printing 'while-loop fungerar'
```

I exemplet ovan, skrevs **'while-loop fungerar'** ut samt lägger till en etta till variabeln **y** så länge den uppfyller villkoret **(y < 4)**. Detta innebär att satserna kommer att utföras tills variabeln **y** är mindre än fyra.

I ETL går det även att avbryta while-loopen genom att skriva **stop** efter de satserna som ska utföras för första gången. Detta gör while-loopen utföra de satserna endast en gång, därefter kommer det avbrytas.

Exempel:

```
y = 1
while ( y < 4)
write "while-loop fungerar endast en gång"
y = y + 1
stop
endwhile
```

Skriver ut följande:

```
-->> Printing 'while-loop fungerar endast en gång'
```

2.8 Funktioner

ETL-språket har två sorts funktioner:

1. Funktioner utan parametrar:

   För att definiera en funktion utan parameter i ETL behöver användaren skriva **define** och sedan sätta ett namn på den funktionen. Därefter måste användaren skriva töm parentes så att kodraden kommer att se ut så här: **define name()** i slutändan.

   Efter definitionen av funktionen kan användaren skriva en eller flera satser inuti funktionskroppen. Funktionskroppen avslutas med ordet **return** beroende på vad användaren vill returnera.

   I slutet bör alltid användaren skriva ordet **enddef** för att avsluta funktionskroppen.

   Användaren kan anropa funktionen genom att skriva exempelvis **write name()** på en ny rad, där kommer terminalen skriva ut det som funktionen returnerar.

Exempel på funktioner utan parameter:

```
define add()
a = 4
b = 5
c = a + b
return c
enddef

write add()
```

Skriver ut följande:

```
-->> Function 'add' returning '9'
```

2. Funktioner med parametrar:

   För att definiera en funktion med parametrar i ETL behöver användaren skriva **define** sedan sätta ett namn på den funktionen. Därefter måste användaren öppna en parentes för att skriva parameters namnet. Funktionen kan ta flera parametrar som har kommatecken emellan. Kodraden kommer att se ut så här: **define name(a ,b)** i slutändan.

   Efter definitionen av funktionen kan användaren skriva en eller flera satser inuti funktionskroppen. Funktionskroppen avslutas med ordet **return** beroende på vad användaren vill returnera.

   I slutet bör alltid användaren skriva ordet **enddef** för att avsluta funktionskroppen.

   Användaren kan anropa funktionen med parametrar genom att skriva exempelvis **write name(2, 5)** på en ny rad, där parametrarna som finns inuti parentesen ska ta sina värden. I slutet kommer terminalen skriva ut det som funktionen returnerar.

   Exempel på funktioner med parameter:

```
define add(a, b)
s = a + b
return s
enddef

write add(25, 75)
```

   Skriver ut följande:

```
-->> Function 'add' returning '100'
```

2.9 Multiple Strings

ETL har en konstruktion som heter *Multiple Strings*. Denna finns för att låta användaren addera antingen två eller flera variabler som innehåller strängar (som i Exempel 1) eller två eller flera strängar (som i Exempel 2) med varandra genom att skriva ordet **plus** mellan de variablerna/strängarna som ska adderas.

Exempel 1:

```
x = "ETL" plus " är enkelt."
write x
```

Skriver ut följande:

```
-->> Printing 'ETL är enkelt.'
```

Exempel 2:

```
y = "ETL"
z = " är"
w = " lätt att lära sig!"
write y plus z plus w
```

Skriver ut följande:

```
-->> Printing 'ETL är lätt att lära sig!'
```