LINKÖPINGS UNIVERSITET

TDP019 Projekt: Datorspråk

PLEASE

Författare

Samuel Jönsson, samjo479@student.liu.se
Ronja Eriksson, roner992@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Vårterminen 2020
Version 0.1

2020-05-28

Innehåll

1 Inledning

Detta dokument innehåller all information som krävs för att använda programspråket PLEASE. Det ger information om hur språket installeras, hur det änvänds det och hur det är uppbyggt.

1.1 Syfte

Detta programspråk skapades under kursen TDP019, Projekt: Datorspråk som en del av kanditatprogrammet Innovativ Programmering. Syftet med arbetet är att skaparna ska få kunskap om följande områden:

- Konstruera ett mindre datorspråk.
- Diskutera och motivera designval i det egna datorspråket med utgångspunkt i teori och egna erfarenheter.
- Implementera verktyg (interpretator, kompilator, etc) för det egna datorspråk
- Formulera teknisk dokumentation av det egna datorspråket
- Presentera en uppgift muntligt inför publik enligt angivna förutsättningar

1.2 Språkidé

Programspråket PLEASE är tänkt att vara ett programspråk som får användaren att fundera kring människa-dator interaktionen. Genom att ha en syntax som kräver ett artigt språkbruk i alla skrivna program eds användaren till att reflektera kring skillnaden i hur en dator bes följa instruktioner och i hur vi kommunicerar människor emellan.

Språket har en syntax som är nära engelskt skriftspråk. Artighetsfraser markerar början och slut av funktioner och styrsatser. Instruktioner i språket skrivs som meningar och avslutas med punkt.

2 Användarmanual

Nedan följer instruktioner till hur PLEASE installeras och körs.

2.1 Installation

För att använda PLEASE krävs att användaren har programspråket Ruby installerat. Implementationen av PLEASE skapades med hjälp av Ruby version 2.5.1. Programmets filer är komprimerade i en zip-fil.

Placera zip-filen i valfri katalog och extrahera sedan filerna. Skriv följande kommando i terminalen:

```
echo "alias PLEASE='ruby bnf.rb'" >> .bashrc
```

Användaren kan nu skriva kod och spara filen med fileändelsen ".thx". Vid körning av koden skrivs kommandot PLEASE följt av filens namn.

2.2 Ett första PLEASE-program

För att skapa ett PLEASE-program skapar du en fil i en texteditor där du skriver din programkod. Du sparar sedan din fil med filändelsen .thx. Kod för ett enkelt program kan se ut som det för hello_world.thx:

```
Dear Computer,
  please write the following "Hello world".
Sincerely Me
```

"Dear Computer" markerar starten av huvudfunktionen och "Sincerely Me" slutet. "Me" kan här ersättas med utvecklarens namn för att signera sin kod. "Please write the following: "Hello world"." Skriver ut "Hello world" i terminalen.

Denna fil kan köras med kommandot `PLEASE hello_world.thx`.

2.3 Konstruktioner

PLEASE innehåller ett antal konstruktioner som kan användas för att utföra olika uppgifter. En introduktion till dessa följer nedan.

2.3.1 Main-funktion

Ett program kräver en huvudfunktion för att kunna köras. Huvudfunktionen utgörs av startfrasen "Dear Computer", följt av slutfrasen "Sincerely, Signatur". Mellan dessa nyckelord skrivs instruktioner programmet ska utföra. Exempel på ett enkelt program ser ut som följer:

```
Dear computer,
  please write the following: "This is the main function".
Sincerely, User
```

2.3.2 Variabler

Variabler i PLEASE deklareras med

```
please create the variable named {variable_name} containing {data}.
```

En variabel kan innehålla siffror (med eller utan decimaler) samt ord eller meningar. Ord och meningar omgärdas av citattecken och kan innehålla siffror. Ett variabelnamn kan bestå av bokstäverna a-z, siffrorna 0-9 samt understreck. Här följer några exempel på giltiga variabeldeklarationer.

```
please create the variable named ten containing 10.
please create the variable named pi containing 3.14.
please create the variable named words containing "This is a sentence".
```

En variabel kan också deklareras med resultatet av en matematisk uträkning eller ett funktionskall, se 2.3.4 samt sektion 2.3.8.

När en variabel deklareras skiljer sig inte syntaxen åt beroende på innehåll. Det finns dock skillnader i hur variabler kan användas. En variabel innehållandes en mening kan inte användas i matematiska uträkningar medan en variabel innehållandes ett tal kan användas till detta.

2.3.3 Listor

Listor i PLEASE är en sammansatt datastruktur som kan innehålla variabler, klassobjekt och listor, eller en godtycklig kombination av dessa. En lista är indexerad där varje objekt återfinns på sin egen plats i listan och indexeringen räknas från 1. En lista skapas som följer:

```
please create the list named example_list containing "hello", 1, 5, "item", variable.
```

Objekt i en lista kan nås via sitt index med hjälp av följande syntax:

```
please access the item at index 1 in example_list.
```

Objektet längst fram i listan skulle då returneras, i detta fall ”hello”.

Objekt kan läggas till i listor antingen vid ett specificerat index eller längst bak i listan.

```
please add "stuff" to example_list at index 3.
```

Detta lägger till ”stuff” på plats 3 och flyttar allt på plats 3 och efter ett steg bak i listan. Listan skulle då se ut såhär:

”hello”, 1, ”stuff”, 5, ”item”, variable

```
please add blue to example_list.
```

blue läggs till sist i listan. Nu ser listan ut som följer:

”hello”, 1, ”stuff”, 5, ”item”, variable, blue

Objekt kan tas bort ur listan med hjälp av index.

```
please remove the element at index 4 from example_list.
```

Detta skulle ta bort 5 ur listan och såhär skulle listan då se ut:

”hello”, 1, ”stuff”, ”item”, variable, blue

Variabler kan också tas bort via namn.

```
please remove blue from example_list.
```

blue tas då bort och såhär ser då listan ut:

”hello”, 1, ”stuff”, ”item”, variable

2.3.4 Matematik

PLEASE har funktionalitet för grundläggande matematiska uttryck. Addition, subtraktion, multiplikation och division är inkluderat och har rätt prioritetsordning. Dessa operationer utförs som följer:

```
adding x to y

subtracting x from y

multiplying x with y

dividing x by y
```

X och y kan här antingen vara ett tal eller en variabel innehållandes ett tal.

En variabel kan deklareras som resultatet av en matematisk uträkning med följande syntax:

```
please create the variable named {math_variabel} containing the result of adding 2 to 5.
```

math_variable skulle i detta fall ha värdet 7.

2.3.5 Villkor

I PLEASE används villkor för att utvärdera styrstrukturer och repetitionssatser. De olika villkorssatserna är följande:

```
or is
and is
greater than
less than
equal to
not
```

Dessa returnerar true eller false baserat på resultatet vid ett uttrycks utvärdering. Nedan följer ett några exempel.

```
5 greater than 3
```

utvärderas till true

```
3 greater than 5
```

utvärderas till false

```
5 greater than 3 and is 6 greater than 3
```

utvärderas till true

```
7 greater than 10 or is 3 greater than 5
```

utvärderas till false

```
not true
```

utvärderas till false

2.3.6 Consider-Otherwise

PLEASE tillåter användning av styrstrukturer i form av Consider-otherwise, denna konstruktion motsvarar if-else i andra programspråk. Likt if-else tillåter detta instruktioner i programkod att genomföras baserat på vilka villkor som uppfylls. En consider-otherwise-sats kan ha olika struktur och nedan följer ett exempel.

```
please consider the following, is 3 greater than 5?
  then
    please write the following: "3 is greater than 5".
  otherwise, is 4 greater than 6?
  then
    please write the following: "4 is greater than 6".
  otherwise
    please write the following: "The conditions above are false".
thank you
```

Koden utvärderar de olika villkoren och kommer fram till att varken det första eller andra är sanna, därmed genomför koden instruktionerna i den tredje grenen. Consider-otherwise initieras alltid med frasen “please consider the following, is {condition}?” och avslutas med thank you. Strukturen kan innehålla en godtycklig mängd “otherwise, is {condition}?” med tillhörande instruktioner och kan men behöver inte avslutas med “otherwise”. Kodblocket behöver däremot alltid avslutas med “thank you”.

2.3.7 Repetitionssatser

PLEASE har flera sorters repetitionssatser med lite olika egenskaper. Alla repetitionssatser avslutas med ’thank you’.

```
please repeat the following until {condition}.
  {instructions}
thank you
```

I denna form av repetition är condition ett villkor. Så länge detta villkor är sant kommer koden inuti repetitionssatsen att köras, vilket motsvarar en while-loop i andra programmeringsspråk.

```
please repeat the following {x} times.
  {instructions}
thank you
```

X är här det antal gånger instruktionerna i repetitionssatsen kommer att köras.

```
please repeat the following {x} times counting with {i}.
  {instructions}
thank you
```

X är här hur många gånger instruktionerna kommer köras och i är en variabel som motsvarar vilket ”varv” i repetitionssatsen programmet befinner sig i. Detta tillåter användning av indexbaserade repetitionssatser.

```
please repeat the following counting from {x} up to {y} with {i}.
  {instructions}
thank you
```

Denna repetitionssats fungerar som den ovan, med skillnaden att varven räknas från x istället för 1.

```
please repeat the following for each {item} in {list}.
  {instructions}
thank you
```

Denna repetitionssats itererar över objekt i en lista direkt. Detta tillåter samma instruktioner att utföras på varje objekt i en lista.

Alla repetitionssatser kan avbrytas. Detta genom att inuti en repetitionssats skriva något av följande:

```
please don't
please stop
```

Repetitionssatsen kommer då att sluta iterera.

2.3.8 Funktioner

Funktioner i PLEASE deklareras alltid ovanför Main-funktionen och kan likt main-funktionen innehålla en godtycklig mängd instruktioner.

Funktioner utgörs av nyckelfraserna `please remember the following instructions named {function_name}.` och måste alltid avslutas med frasen “thank you”. I slutet av funktionen skrivs även vad denna ska returnerna efter körning. `please return {variable_name}` returnerar exempelvis en variabels värde. Om en funktion inte returnerar något skrivs detta uttryckligen med `please return nothing`.

Nedan följer ett exempel på hur en funktion kan deklareras.

```
please remember the following instructions named {function_name}.
  please return nothing
thank you
```

För att kalla på funktionen skulle följande syntax användas:

```
please follow the instructions named {function_name}.
```

PLEASE tillåter även överföring av ett godtyckligt antal parametrar. Vid användning av dessa måste ordet “with” inkluderas i funktioners deklarationer och anrop följt av parametrarnas namn, skiljt med kommatecken. Exempel på en funktionsdeklaration med parametrar följer nedan.

```
please remember the following instructions named {function_name} with {parameter1},
{parameter2}, {parameter3}.
```

```
  please return nothing
thank you
```

För att anropa denna funktion används följande syntax:

```
please follow the instructions named {function_name} with {argument1},
{argument2}, {argument3}.
```

För att fånga upp en funktions returvärde måste funktionens anrop kombineras med variabeltilldelning.

```
please follow the instructions named {function_name} and remember the result
as {variable_name}.
```

Även denna variabeltilldelning kan kombineras med parametrar.

```
please follow the instructions named {function_name} with {argument} and remember
the result as {variable_name}.
```

2.3.9 Klasser

Klasser i PLEASE deklareras högst upp i filen, över eventuella funktioner och main-funktion. Klasser kan innehålla både medlemsfunktioner medlemsvariabler och en godtycklig mängd av dessa. PLEASE stödjer inte klassvariabler. En klass i PLEASE deklareras som följer;

```
please remember the following template for objects of the type {class_name}.
  it has the following variables:
    {variable_name}

  it has the following instructions:
    please remember the following instructions named {function_name} with {parameter}.
      please write the following: "hello".
      please create the variable named {variable_name_2} containing the result of
      adding {parameter} to 2.
    please return {variable_name_2}
  thank you
thank you
```

Medlemsinstruktioner deklareras på samma vis som vanliga instruktioner och en klass kan ha ett godtyckligt antal.

För att skapa ett klassobjekt används följande syntax.

```
please create an object using the template {class_name} named {object_name} with 1.
```

Här har {object_name}s {variable_name} värdet 1. När ett klassobjekt skapas måste alla medlemsvariabler tilldelas ett värde. Medlemsvariabler tilldelas värden i den ordning de är deklarerade.

Medlemsfunktioner kallas på följande vis:

```
please follow the instructions named {function_name} in {object_name} with
{argument} and remember the result as id.
```

Ett klassobjekts medlemsvariabler skrivs ut på följande vis.

```
please write {object_name}'s {variable_name}.
```

Klasser kan ärva från varandra. Den ärvande klassen deklareras som följer.

```
please remember the following template for objects of the type {child_class_name} which
inheirits from {class_name}.
```

Barnklassen kommer att ha alla variabler som förälderklassen har. Dessa variabler tilldelas värden först när ett klassobjekt skapas, före eventuella egna variabler som barnklassen kan innehålla. Den ärvande klassen har tillgång till alla instruktioner som finns i klassen den ärver från. En klass kan endast ärva från en klass, men arv kan ske i flera led.

2.3.10 Kommentarer

I PLEASE kommenteras kod med symbolen ~. All text på samma rad efter ~kommer ignoreras av interpretatorn. Detta gör att kod kan kommenteras och förklaras i koden samt att kod som inte behövs för stunden men kan behövas senare kan kommenteras ut.

```
~denna text kommer inte att tolkas som kod
~please remember the variable named test containing "testing".
```

2.3.11 Exempel

Detta är ett exempel på ett lite större program med klasser och instruktioner.

```
please remember the following template for objects of the type character.
  it has the following variables:
  name,
  level,
  class

  it has the following instructions:
  please remember the following instructions named level_up.
    please change the value of level to the result of adding 1 to level.
    please return nothing
  thank you
thank you

please remember the following instructions named write_names with names.
  please write the following: "These are the people playing today.".
  please repeat the following for each name in names.
    please write the following: name.
  thank you
  please return nothing
thank you

Dear computer,
  please create an object using the template character named Firya with
  "Firya Il'idhren", 3, "Warlock".
  please create an object using the template character named Vendela with
  "Vendela Ciel", 15, "Sorceror".

  please create the list named player_names containing "Johanna", "Tony", "Mikaela",
  "Jennifer", "Matija".

  please follow the instructions named write_names with player_names.

  please create an object using the template character named Lillian with "Lillian", 15,
  "Fighter".
Sincerely, DM
```

3 Systemdokumentation

I kommande underrubriker beskrivs programspråkets uppbyggnad.

3.1 Noder

Vid körning av programkod i PLEASE används rdparse för att parsea koden och skapa noder utifrån given syntax. Dessa noder bildar gemensamt en trädstruktur där varje nod har en funktion för evaluering (eval()). Denna funktion utför olika uppgifter beroende på vad noden ska returnera för värde. Den första noden som skapas blir programmets rotnod, i PLEASE kommer denna alltid vara en av de fyra Program_Nodes. När noden evalueras kommer denna i sin tur kalla på nästa nods eval(), vilket resulterar i en kedjereaktion där alla noders evalueringar blir sammankopplade.

3.2 Lexikalisk Analys

Programkod som skrivs i PLEASE kommer analyseras av rdparse för att plocka ut tokens. Dessa tokens består av reserverade nyckelord och fraser för att kunna tolka programkoden på ett korrekt sätt. Även icke-reserverade ord så som variabelnamn kommer analyseras.

3.3 Grammatik

```
program ::= <main_func>
          | <class_declarations><func_declarations><main_func>
          | <class_declarations><main_func>
          | <func_declarations><main_func>

class_declarations ::= <class_declarations><class_declaration>
                     | <class_declaration>

class_declaration ::= please remember the following template for objects of the type
                      <identifier> which inherits from <identifier>.
                      <class_content> thank you
                    | please remember the following template for objects of the type
                      <identifier>. <class_content> thank you

class_content ::= it has the following variables: <class_variable_declarations>
                  it has the following instructions: <class_func_declarations>
                | it has the following variables: <class_variable_declarations>
                | it has the following instructions: <class_func_declarations>

class_variable_declarations ::= <class_variable_declarations><class_variable_declaration>
                              | <class_variable_declaration>

class_variable_declaration ::= /\w+/

class_func_declarations ::= <class_func_declarations><class_func_declaration>
                          | <class_func_declaration>

class_func_declaration ::= please remember the following instructions named <identifier>
                           with <param_identifiers>. <instructions><return_statement>
                           thank you
                         | please remember the following instructions named <identifier>.

func_declarations ::= <func_declarations><func_declaration>
                    | <func_declaration>

func_declaration ::= please remember the following instructions named <identifier>
                     with <parameters>. <instructions> <return_statement> thank you
                   | please remember the following instructions named <identifier>.
                     <instructions> <return_statement> thank you

func_call ::= please follow the instructions named <identifier> in <accessor>
              with <parameters> and remember the result as <identifier>.
            | please follow the instructions named <identifier> in <accessor>
              and remember the result as <identifier>.
            | please follow the instructions named <identifier> in <accessor>
              with <parameters>.
            | please follow the instructions named <identifier> in <accessor>.
            | please follow the instructions named <identifier> with <parameters>
              and remember the result as <identifier>.
```

```
            | please follow the instructions named <identifier> and remember
              the result as <identifier>.
            | please follow the instructions named <identifier> with <parameters>.
            | please follow the instructions named <identifier>.

param_identifiers ::= <param_identifiers>, <param_identifier>
                    | <param_identifier>

param_identifier ::= /\w+/

parameters ::= <parameters>, <parameter>
             | <parameter>

parameter ::= <variable>
            | <accessor>

return_statement ::= please return nothing
                   | please return <identifier>

main_func ::= Dear computer <instructions> Sincerely

instructions ::= <instructions><instruction>
               | <instruction>

instruction ::= <class_object_initializer>
              | <variable_declaration>
              | <variable_assignment>
              | <func_call>
              | <control_statement>
              | <print_statement>
              | <container_management>

class_object_initializer ::= please create an object using the template <identifier>
                             named <identifier> with <parameters>.

variable_declaration ::= please create the list named <identifier> containing <list>.
                       | please create the variable named <identifier> containing
                         the result of <expr>.
                       | please create the variable named <identifier> containing
                         <variable>.

variable_assgnment ::= please change the value of <identifier> to <data>.
                     | please change the value of <identifier> by <variable_expr>.

control_statement ::= <if_statement>
                    | <while_statement>
                    | <for_statement>
                    | <break_statement>

if_statement ::= please consider the following, is <condition>? then
                 <instructions> thank you
```

```
                | please consider the following, is <condition>? then
                  <instructions> otherwise <instructions> thank you
                | please consider the following, is <condition>? then
                  <instructions> <otherwise_statements> thank you
                | please consider the following, <condition>? then
                  <instructions> <otherwise_statements> otherwise
                  <instructions> thank you

otherwise_statements ::= <otherwise_statements><otherwise_statement>
                       | <otherwise_statement>

otherwise_statement ::= otherwise, is <condition>? then <instructions>

for_statement ::= please repeat the following for each <variable> in <container>.
                  <instructions> thank you
                | please repeat the following <integer> times. <instructions>
                  thank you
                | please repeat the following <integer> times counting with
                  <variable>. <instructions> thank you
                | please repeat the following counting from <integer> up to
                  <integer> with <variable>. <instructions> thank you

while_statement ::= please repeat the following until <condition>.
                    <instructions> thank you

break_statement ::= please don't
                  | please stop

condition ::= <or_condition>
            | <and_condition>
            | <bool_condition>

or_condition ::= <bool_condition> or is <bool_condition>

and_condition ::= <bool_condition> and is <bool_condition>

bool_condition ::= <factor> greater than <factor>
                 | <factor> less than <factor>
                 | <factor> equal to <factor>
                 | not <factor>
                 | <bool>

print_statement ::= please write the following: <output>.
                  | please write the following: <accessor>.
                  | please write <identifier>'s <identifier>.

container_management ::= <list_insertion>
                       | <list_item_removal>
                       | <list_access>
                       | <list_index_insertion>
                       | <list_index_removal>
```

```
list_insertion ::= please add <varible> to <identifier>.

list_item_removal ::= please remove <variable> from <identifier>.

list_access ::= please access the item at index <data> in <identifier>.

list_index_insertion ::= please add <variable> to <identifier> at index <data>.

list_index_removal ::= please remove the element at index <data> from <identifier>.

return_statement ::= please return <variable>.
                   | please return nothing.


expr ::= adding <term> to <term>
       | subtracting <term> from <term>
       | <term>

term ::= multiplying <factor> with <factor>
       | dividing <factor> with <factor>
       | <factor>

factor ::= <accessor>
         | <variable>

accessor ::= /\w+/

list ::= <elements>

elements ::= <elements>,<element>
           | <element>

element ::= Float
          | Integer
          | /".*"/
          | <identifier>

variables ::= <variables>,<variable>
            | <variable>

variable ::= Float
           | Integer
           | /".*"/

data ::= Integer
       | String

output ::= /\".*"/
```

```
identifier ::= /\w+/
```

4 Reflektion

4.1 Generella erfarenheter

Vi valde ganska tidigt att utföra ett projekt där programkodens tema bestod av "artig" syntax. Vi båda upplevde tidigt under projektet att det var svårt att förstå hur ett programspråk skapas med hjälp av en parser. Till exempel var vi oroliga för att vår långa syntax skulle resultera i problem eftersom vi inte visste hur parsning skulle gå till. Det visade sig att de långa tokens vi valde inte hade någon betydelse. Vår förståelse för varför syntax i vanliga programspråk är kort och koncis förbättrades, eftersom vi märkte att våra långa tokens gjorde det mer komplicerat att hitta fel i syntax bland annat. Att skriva kod är dessutom mycket mer tidskrävande i PLEASE, och det är ibland svårt att komma ihåg vår egen syntax.

Vi hade många föreläsningar under TDP019 som var relevanta för projektet. Vi båda ansåg dock att dessa var svåra att greppa eftersom vi upplever att man faktiskt behöver se exempel i rdparse gällande matchningar av tokens och hantering av dessa för att kunna applicera kunskapen. Detsamma gällde föreläsningen om scope. I efterhand förstår vi tydligt vad dessa föreläsningar ville förmedla. Vi tar med oss lärdomen att man ibland inte behöver stressa över att förstå saker direkt, utan ofta kan kunskapen appliceras vid ett senare tillfälle.

4.2 Parsning

Det var också en utmaning att skapa vår BNF i början och det var inte förrän vi började arbeta med parsning under TDP007 som detta blev lite tydligare för oss. När vi faktiskt fick se hur tokens plockas ut och hur man hanterar dessa matchningar blev allt mycket klarare. Denna kunskap kunde vi också använda för att skriva en tydlig BNF, vilket vi anser förenklade vårt arbete enormt när vi väl började skriva källkoden för vårt programspråk. Eftersom vår BNF var väl uppdaterad när vi började skriva koden kunde vi i stort sätt bara översätta denna till vår parser, och sedan spendera mer tid på att fundera över hur vi skulle lösa olika problem. Efter några dagar av arbete med rdparse kände vi oss bekväma med hur filen fungerade och hur vi skulle använda den för att skapa vårt syntaxträd.

4.3 Parametrar

Den första konstruktionen vi stötte på som var utmanande för oss var parametrar. Det som gjorde det utmanande var att vi behövde fundera över vår scope-hantering. Vi löste detta genom att skapa en lista i form av en klassvariabel @@variables. I denna har varje scope ett eget index med tillhörande variabler. När vi utför en funktions instruktioner tittar vi alltid i det sista scopet för att hitta rätt variabler. Vi lyckades genomföra parameteröverföring med denna metod och vi är nöjda med implementationens resultat.

4.4 Variabelhantering

När vi valde att implementera klasser började vi fundera på om vår hantering av variabler behövde göras om. Eftersom vi i klassvariabeln @@variables sparade värden trodde vi att det skulle bli svårt att blanda dessa med klassobjekt, eftersom objekten själva inte har något definitivt värde. På grund av detta bestämde vi oss för att göra om variabelhanteringen och därmed spara noderna själva i hashtabellerna. Efter att vi hade arbetat en tid med detta insåg vi att vi i processen hade skapat andra problem. Eftersom olika variabler behövde evalueras olika antal gånger kunde vi inte få detta att fungera. Exempelvis behöver en Integer_Node endast evalueras en gång för att returnera ett värde, medan en variabelnod skulle behöva evalueras två gånger (en gång för variabelnamnet, ytterligare en gång för dess värde). Vi ansåg att detta skulle skapa mer problem än vår första variabelhantering, och valde därför att återgå till föregående version. Vi lärde oss av detta att noga tänka igenom stora beslut innan man bestämmer sig för att ändra något. Vi spenderade onödig tid på

detta, och hade troligtvis kunnat förutse konsekverserna om vi hade spenderat mer tid över att fundera hur beslutet skulle påverka oss.