HT15 - VT16

Ludwig Bäcklund (ludli839) & Lukas Vikström (lukvi423)

Introduktion

CodiWeb är ett språk som låter dig programmatiskt konstruera HTML-filer som innehåller bilder, länkar och textblock. Målgruppen för språket är främst grundskolestudenter som ska lära sig programmering. Med CodiWeb så får dessa studenter omedelbar visuell återkoppling för koden de skriver.

Användarhandledning

I detta avsnitt går vi igenom hur man gör för att kunna kompilera och köra kod med CodiWeb samt hur man använder kodens konstruktioner.

Installation och körning

1. Ladda ner CodiWeb i zip-format.
2. Extrahera dess filer till lämplig plats.
3. Navigera till mappen i föregående steg i din kommandoprompt
4. Kompilera en fil genom att i kommandoprompten skriva: **ruby CodiComp.rb <filnamn>**
5. Öppna filen **Result.html** i valfri webläsare

Funktioner

Funktioner följer syntaxet:

```
def metod_namn (variabel_ett, variabel_två)
    #kod
/def
```

Variabler i metod-definitionen kommer att vara tillgängliga för alla konstruktioner inuti metoden. Valfritt antal variabler kan användas, så länge metodanropet matchar det antalet variabler.

Funktionsanrop

Funktionsanrop följer syntaxet:

```
metod_namn.call("sträng_ett", "sträng_två")
```

Tilldelningar

Tilldelning följer syntaxet:

```
variabel_ett = "hej"
variabel_två = variabel_ett
variabel_tre = 5
```

Skapande av HTML-objekt

Skapandet av HTML-objekt följer syntaxet:

```
create <pre_defined_object>
    #kod
/create
```

<pre_defined_object> är namnet på ett av de förbestämda objekten som representerar olika HTML-konstruktioner. Dessa är link, image, paragraph och title. De förbestämda objekten har olika förebestämda variabler som används för att definera deras position och utseende på HTML-sidan. Dessa är:

```
position = top-left | top-middle | top-right | middle-left |
middle-middle | middle-right | bottom-left | bottom-middle |
bottom-right
```

för link, image, paragraph och title.

```
path = "images/image_one.jpg"
path = "http://www.img-host.com/img.jpg"
```

för image

```
url = "http://www.google.com"
```

för link

```
text = "Länk text"
```

för link

```
color = "red" | "blue" | "yellow" | ...
color = "rgb(255, 0, 0)"
color = "ffffff"
```

för paragraph och title

Skapande av flera HTML-objekt

Skapandet av flera HTML-objekt följer syntaxet:

```
create_multiple <amount> <pre_defined_object> <index_variabel>
    #kod
/create
```

<amount> är antalet objekt som ska skapas.
<pre_defined_object> är namnet på ett av de förbestämda objekten som representerar olika HTML-konstruktioner. Dessa är link, image och paragraph.
<index_variabel> är namnet på en variabel som i varje loop får det nuvarande indexvärdet, alltså 1 i första kodexekveringen, 2 i andra, osv.
create_multiple använder förbestämda variabler på samma sätt som create.

Aritmetiska uttryck

Aritmetiska uttryck följer syntaxet:

```
a = 5 + 5 - 2
b = 2 * 4 / 2
```

De matematiska operatorerna evalueras i ordningen de förekommer, och inte i någon form av prioriteringsregler.

If-satser

If-satser följer syntaxet:

```
if a == b
    variabel_ett = "sträng_ett"
/if
```

Dessa if-satser kan t.ex. användas i create_multiple satser för att definera dess förbestämda variabler beroende på vad index-variabeln är för tillfället.

Operatorer

Följande operatorer är tillgängliga:

Operator	Beskrivning
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
<	Mindre än
>	Större än
<=	Mindre än eller lika med
>=	Större än eller lika med
==	Lika med
!=	Inte lika med
and	Och
or	Eller

Exempelkod

```
def menu (url1, imgPath, url2, url3)
    create_multiple 3 link id
        url = url3
        text = "This is a link"
        test = "Ja"
        position = middle-left
        if id == 3 and test == "Ja"
            position = bottom-left
            text = "This is another link"
            url = url2
        /if
        if id == 1
            position = top-left
            text = "This link is here"
        /if
    /create_multiple
    create title
        text = "Welcome"
        color = "green"
        position = top-middle
    /create
    create link
        text = "I am here"
        url = url1
        position = bottom-right
    /create
    create image
        path = imgPath
        position = middle-middle
    /create
/def
create paragraph
    text = "This is an ordinary paragraph with fancy color"
    color = "red"
    position = middle-right
/create
```

```
menu.call("http://google.com", "http://i.imgur.com/6yH9Fkx.gif",
"http://example.com", "http://bing.com")
```

Systemdokumentation

I detta avsnitt går vi igenom hur språket är implementerat och vilka verktyg som användes för att möjliggöra det.

CodiWeb är implementerat i Ruby med hjälp av en s.k “recursive descent parser” (RDParse) som vi använde sen tidigare i kursen TDP007: Konstruktion av datorspråk.

Lexikalisk analys

I den lexikaliska analysen så använder RDParse de tokens som vi har specificerat för att separera och specificera de kodstycken vi vill använda oss av i språket. Dessa tokens sparas undan och de tokens som inte ska användas förkastas.

Ett exempel av vad den lexikaliska analysen utför ser ut på följande sätt:

Innan lexikalisk analys

```
def metod_namn (variabel_ett, variabel_två)
    #kod
/def
```

Efter lexikalisk analys

```
["def", "metod_namn", "(", "variabel_ett", ",", "variabel_två",
")", ..., "/", "def"]
```

Parsning

I parsnings-steget så använder RDParse de tokens som nu finns tillgängliga för att kunna matcha dem med de regler vi har specificerat i vår BNF-grammatik. Detta bestämmer vilken kod som egentligen ska utföras och i vilken ordning. I varje s.k regel där tokens överensstämmer med det som är specificerat så skapas objekt av olika klasser för att sedan evalueras efter att alla tokens har matchat. Dessa klassobjekt returneras sen uppåt i grammatiken tillbaka till den regeln som matchade först vilket innebär att alla klassobjekt till slut kommer vara del av ett annat objekt. Ett exempel på detta är en simpel tilldelning:

Kodstycke

```
variabel_ett = "sträng_ett"
```

Pseudokod för regelmatchning i RDParse

```
:begin
```

```
    match(:stmt_list)
        match(:stmt)
            match(:construction)
                match(:assignment)
                    match(:var, "=", :value)
                        Assignment.new(var, value)
```

I detta exempel så skulle Assignment-objektet returneras uppåt till :stmt som då skapar ett Stmt-objekt som innehåller Assignment-objektet. Detta repeteras för :stmt_list som skapar ett Stmt_list-objekt som då innehåller Stmt-objektetet.

Exekvering

Varje klass har en eval-funktion som kallas på i slutet av parsningssteget av det Stmt_list-objekt som ligger längst upp i hierarkin av klassobjekt. Dessa eval-funktioner innehåller all kod som faktiskt utför det som behövs för att det till slut ska resultera i en färdig HTML-sida.

HTML

Det ligger ett antal olika steg bakom det som till slut resulterar i en HTML-sida som innehåller alla objekt som specificerats i koden. Det första steget är att skapa en mall på en tom HTML-sida. Det är på denna mall vi arbetar för att placera alla HTML-objekt. I implementationen så är varje HTML-objekt representerat av ett objekt av klassen Link, Image, Paragraph och Title. Dessa objekt innehåller en s.k pre_tag, end_tag och content. pre_tag och end_tag innehåller start- och sluttaggar för HTML-objektet, t.ex. <img> och </img> för Image. content innehåller det som ska stå mellan pre_tag och end_tag, i Link's fall är detta texten som är klickbar på HTML-sidan. Alla CSS-egenskaper ligger i en style-tagg inuti pre_tag.

Grammatik

Teckenförklaring

Sym	Innebörd
+	Följt av
|	Eller

BNF

```
<begin> ::= <stmt_list>
<create> ::= "create" + <pre_def_obj> + <stmt_list> + "/" +
"create"
```

```
<loop> ::= "create_multiple" + <num> + <pre_def_obj> + <var> +
<stmt_list> + "/" + "create_multiple"

<pre_def_obj> ::= <link>
                | <paragraph>
                | <title>
                | <image>

<if_stmt> ::= "if" + <expr> + <stmt_list> + "/" + "if"

<expr> ::= <comp_operation>

<stmt_list> ::= <stmt_list> + <stmt>
              | <stmt>

<stmt> ::= <loop>
         |<construction>
         |<operation>
         |<comp_operation>

<comp_operation> ::= <comp_operation> + "and" + <comparison>
                   | <comp_operation> + "or" + <comparison>
                   | <comparison>

<comparison> ::= <var> + <comp_operator> + <value>

<comp_operator> ::= "=="
                   |"!="
                   |">="
                   |"<="
                   |">"
                   |"<"

<function> ::= "def" + <func_name> + "(" + <var_list> + ")" +
<stmt_list> + "/" + "def"
             | "def" + <func_name> + <stmt_list> + "/" + "def"

<function_call> ::= <func_name> + "." + "call" + "(" +
<value_list> + ")"
                   |<func_name> + "." + "call"

<func_name>  ::= /\w+?/

<construction> ::= <function_call>
                 | <function>
                 | <loop>
                 | <if_stmt>
                 | <create>
                 | <assignment>

<assignment> ::= <var> + "=" + <value>
               | <var> + "=" + <operation>
```

```
<operation> ::= <num>
              | <operation> + <math_operation> + <num>

<math_operation> ::= "*"
                   | "/"
                   | "+"
                   | "-"

<value> ::= <num>
          | <string>
          | <var>

<string> ::= /".+"/

<var_list> ::= <var>
             | <var_list> + "," + <var>

<value_list> ::= <value_list> + "," + <value>
               | <value>

<var> ::= <pre_def_var>
        | /\w+?/

<num> ::= Integer

<pre_def_var> ::= "url"
                | "text"
                | "path"
                | "color"
                | "position"
                | "bottom-left"
                | "bottom-middle"
                | "bottom-right"
                | "middle-left"
                | "middle-middle"
                | "middle-right"
                | "top-left"
                | "top-middle"
                | "top-right"
```

Reflektion

Under projektets gång hade vi ett antal problem som resulterade i ett långsammare arbetsflöde under vissa perioder. Den allmänna principen av vad vårt språk skulle göra var väldigt klart för oss, men hur detta skulle översättas till kod samt hur det skulel samarbeta med RDParse var något av ett pussel för oss. För att försöka underlätta detta började vi med att implementera grundläggande grammatik till språket. Om vi kollar tillbaka på den grammatiken nu så ser man en del förändringar som skett under utvecklingens gång. Flera gånger i projektet bestämde vi oss för

att gå tillbaka och ändra på strukturen eller grunden vars element i språket är byggda på, grammatiken var inget undantag även om det inte krävdes större förändringar.

När grammatiken såg acceptabel ut så började vi arbeta med variabler och variabeltilldelning. Vid denna tidpunkt insåg vi att vi var väldigt osäkra på hur våra klasser skulle se ut, och när vi kallar på dessa klasser från grammatiken. I tidigare arbeten inom RDParse hade vi inte använt oss utav klasser utan istället byggt logiken direkt i grammatiken. För att förstå oss på hur detta skulle gå till behövde vi testa ett antal olika lösningar och struktureringssätt för att sedan komma till en slutsats.

Sakta men säkert började våra klasser ta form, och vi kunde börja satsa på att utveckla kärnelementen i språket, nämligen create, och create_multiple. Hur indatan skulle nå dessa operationer var vi osäkra på, och vi ändrade på vår lösning ett antal gånger under utvecklingen. Vi bytte bl.a. mellan att create eller pre_def_obj skulle ta hand om indatan, tills vi till slut beslöt för pre_def_obj. För oss var det dock ett viktigt steg att behöva tänka om, det gjorde det lättare att se fördelar samt nackdelar med olika strukterer även om det var tröttsamt.

Framåt slutet när våra kärnfunktioner existerade insåg vi att vi borde tänka om hur vi lagrar variablerna från de olika operationerna, och vi bestämde oss för att göra grundläggande scoping. Denna förändring innebar en omstrukturering av flertal klasser, och var en lång process fylld av irriterande error meddelanden. Efter denna övergång från vanlig lista till grundläggande scoping med hashning så insåg vi att detta var något som hade underlättat om vi gjorde det lite tidigare under utvecklingen.

Under projektets gång lärde vi oss en hel del om hur ett programmeringsspråk kan vara uppbyggt, och problem som kan uppstå om man inte tänker igenom saker innan man utför dem. CodiWeb är dessutom ett väldigt specifikt språk vilket innebär att det inte riktigt fanns något vi kunde gå efter, utan vi behövde lista ut det mesta själva genom att försöka, misslyckas och sedan tänka om.

I överlag är vi relativt nöjda med slutresultatet men med vi önskar att vi hade mer tid för kunna implementera mer avancerade HTML-objekt. Några saker vi skulle vilja implementera är knappar som man kan klicka på för att förändra utseendet på HTML-sidan, t.ex. bakgrundsfärgen. HTML divs är också något vi skulle vilja implementera för att kunna arbeta på flera objekt samtidigt.

Kod

CodiComp.rb

```
require_relative 'CodiGrammar.rb'
puts ARGV[0]
if ARGV[0]
  CodiWeb.new(ARGV[0]).go
else
```

```
  puts "Missing file argument"
end
```

CodiGrammar.rb

```
require_relative "parse.rb"
require_relative "CodiLogic.rb"
class CodiWeb
  attr_accessor :variable_list

  def initialize(file)
    @file = file
    @ruleparser = Parser.new("CodiWeb") do
      token(/\s/)
      token(/\d+/) { |x| x.to_i }
      token(/(\w+-\w+|\w+|"{1}.+?"{1})/) { |x| x }
      token(/(==|<=|>=|<|>|!=)/) { |x| x }
      token(/./) { |x| x }

      start :begin do
        match(:stmt_list) { |a| a.eval }
      end


      rule :create do
        match("create", :pre_def_obj, :stmt_list, "/", "create")
{ |_, a, b| Create.new(a, b) }
      end

      rule :loop do
        match("create_multiple", :num, :pre_def_obj, :var,
:stmt_list, "/", "create_multiple") { |_, a, b, c, d, _|
Create_Multiple.new(a, b, c, d) }
      end

      rule :pre_def_obj do
        match("link") { |_| Link.new() }
        match("paragraph") { |_| Paragraph.new }
        match("title") { |_| Title.new }
        match("image") { |_| Image.new() }
      end

      rule :if_stmt do
        match("if", :expr, :stmt_list, "/", "if") {|_, a, b, _|
If.new(a, b)}
      end

      rule :expr do
        match(:comp_operation) { |a| a }
      end

      rule :stmt_list do
```

```
        match(:stmt_list, :stmt){ |a, b| Stmt_list.new(a, b) }
        match(:stmt) {|a| Stmt.new(a) }
      end

      rule :stmt do
        match(:loop) { |a| a }
        match(:construction) { |a| a }
        match(:operation) { |a| a }
        match(:comp_operation) { |a| a }
      end

      rule :comp_operation do
        match(:comp_operation, "and", :comparison) { |a, b, c|
ExtComparison.new(a, b, c) }
        match(:comp_operation, "or", :comparison) { |a, b, c|
ExtComparison.new(a, b, c) }
        match(:comparison) { |a| a }
      end

      rule :comparison do
        match(:var, :comp_operator, :value) { |a, b, c|
Comparison.new(a, b, c) }
      end

      rule :comp_operator do
        match("==") { |a| a }
        match("!=") { |a| a }
        match(">=") { |a| a }
        match("<=") { |a| a }
        match(">") { |a| a }
        match("<") { |a| a }
      end

      rule :function do
        match("def", :func_name, "(", :var_list, ")",
:stmt_list, "/", "def") { |_, a, _, b, _, c| Function.new(a, c,
b) }
        match("def", :func_name, :stmt_list, "/", "def") { |_,
a, b| Function.new(a, b) }
      end

      rule :func_name do
        match(/\w+?/) { |a| a }
      end

      rule :function_call do
        match(:func_name, ".", "call", "(", :value_list, ")") {
|a, _, _, _, b|  FunctionCall.new(a, b) }
        match(:func_name, ".", "call") { |a, _, _|
FunctionCall.new(a) }
      end
      rule :construction do
        match(:function_call) { |a| a }
        match(:function) { |a| a }
```

```
        match(:loop) { |a| a }
        match(:if_stmt) { |a| a }
        match(:create) { |a| a }
        match(:assignment){|a| a}
      end

      rule :assignment do
        match(:var, "=", :value) { |var, _, value|
Assignment.new(var, value) }
        match(:var, "=", :operation) { |a, _, b|
Assignment.new(a, b) }
      end

      rule :operation do
        match(:num) { |a| a }
        match(:operation, :math_operation, :num) { |a, b, c|
Operation.new(a, b, c) }
      end

      rule :math_operation do
        match("*") { |a| Multiplier.new }
        match("/") { |a| Divider.new }
        match("+") { |a| Adder.new }
        match("-") { |a| Subtractor.new }
      end

      rule :value do
        match(:num) { |a| a }
        match(:string) {|a| a }

        match(:var){|a| a }
      end

      rule :string do
        match(/".+"/) {|a| a }
      end
      rule :var_list do
        match(:var) { |a| a }
        match(:var_list, ",", :var) { |a, _, b| MultVar.new(b,
a) }
      end
      rule :value_list do
        match(:value_list, ",", :value) { |a, _, b|
MultVar.new(b, a) }
        match(:value) { |a| a }
      end
      rule :var do
        match(:pre_def_var) {|a| a }
        match(/\w+?/) {|a| Var.define(a) }
      end

      rule :pre_def_var do
        match("url") { |a| Url.new(a) }
        match("text") { |a| Text.new(a) }
```

```
        match("path") { |a| Path.new(a) }
        match("color") { |a| Color.new(a) }
        match("position") { |a| Position.new(a) }
        match("bottom-left") { "left:5%;top:60%;" }
        match("bottom-middle") { "left:35%;top:30%;" }
        match("bottom-right") { "left:75%;top:60%;" }
        match("middle-left") { "left:5%;top:30%;" }
        match("middle-middle") { "left:35%;top:30%;" }
        match("middle-right") { "left:75%;top:30%;" }
        match("top-left") { "left:5%;top:5%;" }
        match("top-middle") { "left:35%;top:5%;" }
        match("top-right") { "left:75%;top:5%;" }
      end

      rule :num do
        match(Integer) { |a| a }
      end

    end

  end

  def go
    block = ""
    File.foreach(@file) do |line|
      p line
      block += line
    end
    puts "CodiWeb Constructed the file => #{@ruleparser.parse
block}"
  end

end
```

CodiLogic.rb

```
require 'rubygems'
@@html = nil

class Begin
  attr_reader :html, :css

  def initialize()
    @htmlname = "Result.html"
    @@html = File.new("Result.html", "w")
    IO.copy_stream('template.html', @htmlname)
    return @@html
  end

  def html
    return @htmlname
  end
```

```
end
@@html = Begin.new().html()
class VariableList
  attr_reader :list

  def initialize()
    @list = [{}]
  end

  def add(var)
    @list[@@scope_counter][var.name] = var.value
    @list
  end

  def increase()
    if @@scope_counter == 0
      @list << {}
      @@scope_counter = @list.length - 1
    end
  end

  def clear()
    @list[@@scope_counter].clear()
  end

  def findValue(var)
    @list.each do |varObj|
      if varObj.name == var
        return varObj.value
      end
    end
    return false
  end

  def findVar(var)
    @list.each do |varObj|
      if varObj.name == var
        return varObj
      end
    end
    return false
  end
end

@@variableList = VariableList.new()
@@scope_counter = 0

class MultVar
  def initialize(var, multvar = nil)
    @var = var
    @varlist = multvar if multvar
    @variables = []
  end
```

```
  def eval()
    @varlist.eval if @varlist
    @var.eval
  end

  def variables()
    @variables.clear
    if @varlist.class == MultVar
      @variables += @varlist.variables
    else
      @variables << @varlist
    end
    if @var.class == MultVar
      @variables += @var.variables
    else
      @variables << @var
    end
    @variables.reverse
  end
end
class Var
  attr_reader :value, :name, :type

  def initialize(name)
    @name = name
    @value = nil
    @type = nil
  end

  def variables
    return [self]
  end

  def Var.define(name)
    var = @@variableList.list[@@scope_counter][name]
    if var
      return var
    else
      Var.new(name)
    end

  end

  def update(value)
    @value = value
    @type= value.class
  end

  def eval()
    return self
  end

  def print
    return @value
```

```
  end
end
class Url < Var
  def update(value)
    if value.class == String
      @type = "pre_def_var"
      @value = 'href= '+ value
    elsif value.class == List
      @type = "pre_def_var"
      @value = value
    end

  end

  def Url.define(name)
    url = @@variableList.list[@@scope_counter][name]
    if url
      return url
    else
      Url.new(name)
    end
  end
end

class Text < Var
  def update(value)
    @type = "text"
    value.gsub!(/\"/, "")
    @value = value
  end
end
class Create
  attr_reader :value

  def initialize(obj, stmt_list)
    @value = obj
    @obj = obj
    @stmt_list = stmt_list
  end

  def eval
    scope_before = @@scope_counter
    @@variableList.increase
    scope_after = @@scope_counter
    @stmt_list.eval
    @obj.update(@stmt_list)
    @obj.eval()
    @@scope_counter = 0 if scope_before != scope_after
    @stmt_list
  end

  def variables
    @stmt_list.variables
  end
```

```ruby
end

class Stmt_list
  attr_reader :variables

  def initialize(obj, stmt_list = nil)
    @stmt = obj
    @stmt_list = stmt_list
    @variables = []
  end

  def eval()
    @stmt.eval() unless @stmt.class == Function
    stmtVariables = retrieve_values(@stmt.variables)
    if @stmt_list
      @stmt_list.eval unless @stmt_list.class == Function
    end
    true
  end

  def retrieve_values(stmt_list)
    variables = []
    if stmt_list.class == Array
      stmt_list.each { |x| variables << x }
    else
      variables << stmt_list
    end
    return variables
  end

  def reset_values()
    @variables.clear
  end

end

class Stmt
  attr_reader :variables

  def initialize(obj)
    @stmt = obj
    @variables = []
  end

  def eval()
    @stmt.eval() unless @stmt.class == Function
    retrieve_values(@stmt.variables)
    true
  end

  def retrieve_values(stmt_list)
    variables = []
    variables << stmt_list
    return variables
```

```
  end

  def reset_values()
    @variables.clear
  end

end
class Assignment
  attr_reader :value

  def initialize(var, value)
    @var = var
    @value = value
  end

  def eval()
    if @value.class == Var
      @var.update(@@variableList.list[@@scope_counter]
[@value.name])
    elsif @value.class == String
      @var.update(@value)
    elsif @value.class == Operation
      @var.update(@value.eval)
    else
      @var.update(@value)
    end
    @@variableList.add(@var)
    @var
  end

  def variables
    return [@var]
  end
end

class Predefined_object
  def initialize()
    @content = ""
    @htmlpage = File.open(@@html)
    @css = 'style='
    @html = ""
    @stmt_list = nil
  end

  def add_css(value)
    @css += value
  end

  def add_html(value)
    @html += value
  end

  def add_content(value)
    @content += value
```

```
  end

  def eval()
    l = @@variableList.list[@@scope_counter]
    l.each do |key, value|
      if @attributes.has_key?(key)
        add_css(value) if @attributes[key] == "css"
        add_html(value) if @attributes[key] == "html"
        add_content(value) if @attributes[key] == "content"
      end
    end
    build
    true
  end

  def build()
    final = @pre_tag.insert(-2, +" " + @html+ " " + @css) +
@content + @end_tag
    fil = File.open(@@html)
    read = fil.read
    topSpan = findWriteLoc(read)
    temp = (read).insert(topSpan, "\n"+final)
    File.write(@@html, temp)
    fil.close
  end

  def update(stmt_list)
    @stmt_list = stmt_list
  end

  def findWriteLoc(string)
    body_index = string.index("<body>")
    if body_index
      return body_index + 6
    else
      raise Exception.new("Invalid Template file, make sure
<body> is included")
    end
  end
end

class Link < Predefined_object
  def initialize
    super
    @attributes = {"url" => "html", "position" => "css", "text"
=> "content"}
    @pre_tag = "<a >"
    @end_tag = "</a>"
  end

  def clone
    Link.new()
  end
end
```

```
class Image < Predefined_object
  def initialize
    super
    @attributes = {"path" => "html", "position" => "css"}
    @pre_tag = "<img >"
    @end_tag = "</img>"
  end

  def clone
    Image.new()
  end
end

class Paragraph < Predefined_object
  def initialize
    super
    @attributes = {"text" => "content", "position" => "css",
"color" => "css"}
    @pre_tag = "<p >"
    @end_tag = "</p>"
  end

  def clone
    Paragraph.new
  end
end

class Title < Predefined_object
  def initialize
    super
    @attributes = {"text" => "content", "position" => "css",
"color" => "css"}
    @pre_tag = "<h1 >"
    @end_tag = "</h1>"
  end

  def clone
    Title.new
  end
end
class Path < Var
  def update(value)
    @type = "pre_def_var"
    @value = "src=" + value
  end

  def Path.define(name)
    var = @@variableList.list[@@scope_counter][name]
    if var
      return var
    else
      Path.new(name)
```

```
    end
  end
end

class If
  def initialize(expr, stmt_list)
    @expr = expr
    @stmt_list = stmt_list
    @comparison_return = false
  end

  def eval()
    @comparison_return = @expr.eval
    if @comparison_return
      @stmt_list.eval
    else
      @expr
    end
  end

  def variables
    if @comparison_return
      @stmt_list.variables
    else
      []
    end
  end
end

class Comparison
  def initialize(var, comp_oper, value)
    @var = var
    @comp_oper = comp_oper
    @value = value
  end

  def eval
    var = @@variableList.list[@@scope_counter][@var.name]
    if var
      if @comp_oper == "=="
        return var == @value
      elsif @comp_oper == "!="
        return var != @value
      elsif @comp_oper == ">="
        return var >= @value
      elsif @comp_oper == ">"
        return var > @value
      elsif @comp_oper == "<="
        return var <= @value
      elsif @comp_oper == "<"
        return var < @value
      end
    end
  else
```

```
    raise Exception.new("Undeclared Variable #{@var.name}")
  end
end
class ExtComparison
  def initialize(comp, comp_oper, comp2)
    @comp_one = comp2
    @extender = comp_oper
    @comp_list = comp
  end

  def eval
    if @extender == "or"
      expr1 = @comp_one.eval
      expr2 = @comp_list.eval
      bool = expr1 || expr2
      return bool
    end
    if @extender == "and"
      expr1 = @comp_one.eval
      expr2 = @comp_list.eval
      bool = expr1 && expr2
      return bool
    end
  end
end

class Create_Multiple
  def initialize(number, obj, var, stmt_list)
    @obj = obj
    @times = number
    @obj_list = []
    @var = var
    @stmt_list = stmt_list
  end

  def eval
    scope_before = @@scope_counter
    @@variableList.increase
    scope_after = @@scope_counter
    obj_amount = @times
    obj_amount.times do
      @obj_list << @obj.clone
    end
    rawStmt = @stmt_list
    @obj_list.each_with_index do |obj, index|
      @stmt_list = rawStmt
      variable = Assignment.new(@var, index+1)
      variable.eval
      @stmt_list.eval
      obj.update(@stmt_list)
      obj.eval()
    end
    @@scope_counter = 0 if scope_before != scope_after
    return true
```

```
  end

  def variables
    @stmt_list.variables
  end
end

class Position < Var
  def update(value)
    @type = "css"
    @value = "position:absolute;#{value}"
  end
end


class Color < Var
  def update(value)
    @type = "css"
    value.gsub!(/\"/, "")
    @value = "color:#{value};"
  end
end

class Function
  attr_reader :name, :status, :variables, :value

  def initialize(name, stmt_list, varlist = nil)
    @name = name
    @stmt_list = stmt_list
    @variables = varlist
    @value = self
    @@variableList.add(self)
    true
  end

  def variables
    return @variables.variables if @variables
    return [] if not @variables
  end

  def eval
    @stmt_list.eval
  end
end

class FunctionCall
  def initialize(name, varlist = nil)
    @name = name
    @varlist = varlist if varlist
  end

  def eval
    l = @@variableList.list[@@scope_counter]
    @@variableList.increase
```

```
    l.clone.each do |key, value|
      if key == @name
        function_variables = l[key]
        if (@varlist && function_variables) &&
@varlist.variables.length == function_variables.variables.length
          @varlist.variables.each_with_index do |x, i|
            Assignment.new(function_variables.variables[i],
x).eval
          end
        elsif not @varlist

        elsif @varlist.variables.length !=
function_variables.variables.length
          raise Exception.new("Wrong number of arguments for
function #{function_variables.name}")
        end
        return function_variables.eval
      end
    end
  end
end

class Operation
  attr_reader :variables

  def initialize(operation, math_operation, value)
    @value, @math_operation, @operation = value, math_operation,
operation
    @variables = []
  end

  def variables
    @variables << @value
    if @math_operation
      @variables << @math_operation
    end
    if @operation
      @variables += @operation.variables if @operation.class ==
Operation
      @variables << @operation if @operation.class == Fixnum
    end
    return @variables
  end

  def eval
    variables()
    @math_operation.set_values(@operation.eval, @value) if
@operation.class == Operation
    @math_operation.set_values(@operation, @value) if
@operation.class == Fixnum || @operation.class == Float
    @math_operation.eval
  end
end
```

```ruby
class MathExpr
  def initialize
    @l_h = nil
    @r_h = nil
  end

  def set_values(l_h, r_h)
    @l_h, @r_h = l_h, r_h
  end
end

class Multiplier < MathExpr
  def eval
    return @l_h * @r_h
  end
end

class Divider < MathExpr
  def eval
    return @l_h / @r_h
  end
end

class Adder < MathExpr
  def eval
    return @l_h + @r_h
  end
end

class Subtractor < MathExpr
  def eval
    return @l_h - @r_h
  end
end
```