ImgPL

Image Programming Language

Mattias Roback Magnus Suther

Innovativ Programmering VT 2011
TDP019 – Projekt: Datorspråk
Linköpings universitet

Innehåll

1. Inledning

Vi har i kursen TDP019 på Innovativ Programmering vårterminen 2011 skapat ett eget programmeringsspråk. Arbetet har gjorts i projektform i grupper om två. Vårt språk har vi valt att kalla ImgPL – Image Programming Language. ImgPL är ett domänspecifikt språk tänkt att användas för att redigera bilder i första hand, med hanterar även vanliga konstruktioner så som kontrollsatser, upprepningssatser och input/output. ImgPL är ett interpreterat språk, och har en enkel och ren syntax liknande den i Python. Språket är dessutom indenteringsstyrt, vilket bäddar för lättläst och tydlig kod.

Vi nämde att språket är interpreterat – det här innebär att språket kan användas på två sätt. Precis som andra språk kan programmet skrivas till fil som sedan kan exekveras, men språket kan även användas direkt i en terminal genom att mata in en kodrad i taget. Det interpreterade läget är smidigt som ett verktyg för att redigera bilder, medan programkod skriven till fil underlättar distribution till slutanvändare.

Eftersom språket har en enkel och indenteringsstyrd syntax är språket en bra startpunkt för nya glada programmerare, men kan lika gärna användas av avancerade användare, som kanske bäddar in språket i egna program skrivna i andra språk.

ImgPL är ett språk för bildredigering i första hand, och har ett antal funktioner för detta. Bilder redigeras i objektorienterad anda, där bilderna läses in i imageobjekt, och för dessa objekt körs bildoperationerna. ImgPL kan läsa enstaka bilder såväl som alla bilder ur en mapp. Operationerna görs på alla bilder i imageobjektet vilket gör det smidigt att utföra så kallade batch-jobb – likadana operationer för alla bilder i en mapp.

Vårt språk hanterar också objektorientering. Klasser kan definieras med metoder och medlemsvariabler vilka blir unika för alla olika instanser av klassen.

2. Användarmanual

ImgPL – eller Image Programming Language – är ett interpreterat och intenteringsstyrt språk med syfte att möjliggöra bearbetning av bilder på ett smidigt sätt. Språket är objektorienterat och har en enkel och ren syntax inspirerad av Python och är därmed lätt för nybörjare att lära sig.

Språket kan användas dels som ett enkelt verktyg för bildredigering direkt via den interpreterade miljön, dels i form av ett större program riktat mot en slutanvändare.

2.1. Interpreterat eller från fil?

Den interpreterade miljön är praktisk att använda för att genomföra enklare operationer på en eller flera bilder. Man kan även göra andra enklare operationer som variabeltilldelning och matematiska operationer. I den interpreterade miljön kan man även importera programfiler skrivna i ImgPL, för att på så sätt kunna utnyttja funktionaliteten i dem.

Att skriva sitt program i en fil är bra om programmet är tänkt att distribueras till slutanvändare. Ett program i fil kan interagera med användaren, köra upprepningssatser och kontrollsatser för att agera på användarens inmatningar.

För lite längre program rekommenderas att man skriver programmet i fil, eftersom program skrivna i den interpreterade miljön försvinner när ImgPL avslutas eller när någon oväntad inmatning leder till felmeddelande och avslut av ImgPL.

2.2. Installation

Installation av ImgPL innebär att språkets nödvändiga filer läggs in i systemets mappar. ImgPL kan därefter köras från en terminal, oavsett i vilken mapp terminalen står i. Sökvägar till och från programfiler och/eller bilder utgår från den mapp från vilken ImgPL startats.

Språket kan även startas direkt utan installation, genom att med terminalen navigera till mappen med språkets filer och starta ImgPL med ./ImgPL. För mer om hur ImgPL kan köras hänvisas till kapitel 2.3:Starta språket.

2.2.1. Förkrav

För att kunna använda språket och exekvera program skrivna i språket krävs att vissa program är installerade på din dator.

Språket kräver

- Ruby 1.8.*. ImgPL är testat med Ruby 1.8.7 med Ubuntu 10.04 och Ubuntu 10.10
- RMagick (librmagick-ruby) – ett rubygem som kan installeras via systemets pakethanterare.
- ImageMagick

2.2.2. Installation med makefile

Installation med makefile kräver att systemet har ett Unixliknande filsystem, där mapparna

- /usr/local/bin
- /usr/local/lib

måste finnas på systemet. Dessa mappar ska finnas i alla system baserade på Linux. Installationen placerar filen ImgPL i /usr/local/bin/imgpl och övriga filer i /usr/local/lib/ImgPL. Användaren måste ha skrivrättigheter till ovanstående mappar för att installationen ska fungera. Installationen görs med kommandot **make install** i en terminal från den mapp som språkets filer ligger i.

```
1.    sudo make install
```

2.3. Starta språket

Språket kan startas antingen interpreterat eller med en sökväg till en fil för att exekvera filen.

Om språket är installerat på datorn (rekommenderas), så startas språket på ett av följande sätt.

- imgpl
- imgpl program.imgpl

Det första kommandot startar språket i interpreterat läge och det andra exekverar den fil vars namn ges som argument. En fil som ska exekveras måste vara skrivet i ImgPL och måste ha filändelsen imgpl.

Språket kan användas även om det inte är installerat på datorn, genom att med terminalen navigera till mappen med språkfilerna och starta språket med ./ImgPL.

Man kan när som helst avsluta genom att trycka **ctrl-d** eller **ctrl-c** i terminalen. Detta fungerar oavsett om det är en fil som exekveras eller om språket körs i interpreterat läge.

Körs språket i interpreterat läge kan även kommandona **exit** och **quit** användas för att avsluta.

2.4. Importera filer

Om ImgPL används i det interpreterade läget, kan man importera programfiler skrivna i ImgPL, för att på så sätt kunna utnyttja funktioner, klasser och variabler som finns definierade i dem. Vid importering av en fil kommer den att evalueras, vilket innebär att funktioner, klasser och variabler kommer att definieras och sparas undan, men även att eventuella utskrifter och inläsningar görs.

Exempel 1 – Importering av filer

```
1.    import <file.imgpl>
```

2.5. Kommentarer

Språket hanterar både enradskommentarer och flerradskommentarer. Enradskommentarer används med fördel under utveckling och testning för att exekveringen ska ignorera en rad i programmet utan att man behöver ta bort den. Enradkommentarer skrivs enligt följande syntax.

Exempel 1 – Enradskommentarer

```
1.    // bortkommenterad text eller kod
2.    if a is 10 and isOK() then // bortkommenterad kod

  Texten från och med // fram till radens slut ignoreras under
  exekvering.
```

Flerradskommentarer används huvudsakligen till att kommentera kod och för att kommentera ut kodblock.

Exempel 2 – Flerradskommentarer

```
1.    /*
2.    Den här texten ignoreras.
3.    */
4.
5.    /*
6.    println << ”This line”
7.    println << ”and this line”
8.    println << ”will be ignored.”
9.    */

  Notera att /* och */ måste stå ensamma på egna rader för att
  flerradskommentarer ska fungera.
```

2.6. Reserverade nyckelord

ImgPL innehåller en del reserverade nyckelord. Reserverade nyckelord är precis som namnet antyder reserverade för att kunna skapa olika konstruktioner i språket, som exempelvis if-satser och klassdefinitioner. De orden kan därför inte användas till variabelnamn eller funktionsnamn.

Tabell 1: Reserverade nyckelord

if else then while for	while do break class func	return null print println errprint	get true false and or	not is in

2.7. Bildredigering med ImgPL

Bildredigering är språkets huvudsyfte och kan göras både genom interpretatorn och i större program skrivna i fil. Bildredigering görs alltid genom att ladda in en eller flera bilder till ett objekt, på vilket man sedan kör redigeringsfunktioner.

Bilder öppnas alltid genom att kalla på den inbyggda klassen Image

ImgPL hanterar följande operationer på bilder.

Tabell 2: Bildoperationer

Operation	Funktionsnamn	Parametrar
Öppna bilder	new	Ett eller flera filnamn. Ett eller flera mappnamn. Om mappnamn anges kommer alla bilder i mappen att laddas in till objektet.
Spara bilder	save	Mappnamn. Om mappnamn anges, sparas alla bilder i objektet till denna mapp, annars sparas alla bilder i objektet över till samma fil de lästes från.
Visa bildspel	slideshow	Fördröjning i millisekunder. Fördröjning innan nästa bild i objektet visas. Kan vara

		Integer och Float.
Visa bilden eller bilderna	display	Inga parametrar. Visar alla bilder i objektet en och en. Stega mellan bilderna med mellanslag.
Rotera	rotate	Gradtal. Rotera alla bilder i objektet angivet antal grader. Roterar höger om värdet är positivt.
Flippa	flip	Inga parametrar. Vänder bilden uppochner.
Skala	scale	Om ett argument ges, skala procentuellt med detta värde. Om två argument ges, skala till önskade värden. Förhållandet mellan höjd och bredd behålls om ett argument ges, annars anpassas bilden..
Konvertera till gråskala	bw	Inga parametrar. Konverterar alla bilder i objektet till gråskala.
Lägg på text	addText	addText() kan ta 10 parametrar, varav alla utom den första är valfri. Parametrarna anges i följande ordning: text – Den text som ska skrivas på bilden. Måste anges. x – position i x-led. Standardvärde: 0. Anges med utgångspunkt från mitten av bilden. y – position i y-led. Standardvärde: 0. Anges med utgångspunkt från mitten av bilden. pointsize – textstorlek. Standardvärde: 32. fill – textfärg. Standardvärde: white font – typsnitt. Standardvärde: Times font_weight – tjocklek på texten. Standardvärde: 400 font_style – normal eller kursiv. Standardvärde: normal

		stroke – konturlinje Standardvärde: none stroke_width – konturlinjens tjocklek Standardvärde: 0 Om parametrar ska hoppas över måste null anges för dessa parametrar.
Vattenmärkning	watermark	watermark() kan ta 7 parametrar varav alla är valfria utom den första. Parametrarna anges i följande ordning: text – texten som ska placeras på bilden. Måste anges. x – position i x-led Standardvärde: 0 Anges med utgångspunkt från mitten av bilden. y – position i y-led Standardvärde: 0 Anges med utgångspunkt från mitten av bilden. pointsize – textstorlek Standardvärde: 32 font – typsnitt Standardvärde: Times font_weight – tjocklek på texten. Standardvärde: 400 font_style – normal eller kursiv. Standardvärde: normal rotation – gradtal som texten ska roteras. Standardvärde: -90
Hämta bildstorlek	getDimensions	Inga parametrar. Returnerar en array med [x, y]-par för alla bilder i objektet.
Hämta filnamn	getFilename	Inga parametrar. Returnerar en array med filnamnen för alla bilder i objektet.
Konvertera mellan format	convert	Tar en parameter med den nya filtypen.

		Parametern måste vara en av jpg, jpeg, png, gif, och anges utan punkt. Convert sparar alla bilder i objektet med det nya formatet till samma mapp som repektive bild lästes in från. Orginalbilderna behålls.

Exempel 1 – Öppna en bild och rotera

```
1.    image = Image.new(”bild.jpg”)
2.    image.rotate(45)

  Den första raden öppnar och läser in bild.jpg till objektet
  image. På nästa rad körs funktionen rotate på objektet. Rotate
  tar ett argument med det gradtal man vill rotera bilden, i
  detta fall 45 grader åt höger.
```

Exempel 2 – Ett större exempel med bilder

```
1.    images = Image.new(”folder”, ”smiley.jpg”)
  Variabeln images innehåller nu alla bilder från mappen folder
  samt smiley.jpg.
  För att konvertera alla bilder till png-formatet skulle vi göra
  så här:

2.    images.convert(”png”)
  Detta kommer att medföra att alla bilder sparas om till det nya
  formatet, och sparas i samma mapp som orginalen kommer från.
  Mappen folder innehåller alltså orginalbilderna plus alla
  konverterade bilder förutom smiley.png, som ligger i samma mapp
  som smiley.jpg. Variabeln images innehåller fortfarande
  orginalbilderna.

  Vi kan visa alla bilder i en slideshow genom att helt enkelt
  göra:

3.    images.slideshow(3000)
  Parametern anges i millisekunder, och bilden i fönstret byts ut
  med detta intervall.

  För att lägga på en vattenmärkning på alla bilder i images gör
  man:

4.    images.watermark(”Hello World!”, 100, 20, 25)
  Detta kommer att vattenmärka bilderna med ”Hello World!” och
  placera texten 100 pixlar åt höger och 20 pixlar ner, räknat
  från mitten av varje bild i images. Textens storlek blir 20
  punkter, och är roterad -90 grader som standard, vilket innebär
  att texten kommer att stå på högkant.
  Funktionen watermark kan ta många olika parametrar, se tabellen
```

```
  ovan.

  För att ändra storlek på alla bilder i images, kan man göra så
  här:

5.    images.scale(20)
  Detta skalar bilderna till 20% av dess ursprungliga storlek.
  Man kan också skriva:

6.    images.scale(50, 50)
  Detta skalar bilderna tills dess att bildens x eller y-värde
  blir 50 pixlar. Bilden behåller sitt förhållande mellan bredd
  och höjd.
```

2.8. Datatyper

ImgPL hanterar datatyperna array, string, integer, float, range och bool, vilket är de vanligast förekommande typerna i alla större språk.

För information om vilka operationer som är tillåtna på de olika datatyperna, se kapitel 2.8.7: Typkontroller och operationer på datatyper.

2.8.1. Array

En array är en lista med element, där varje element har ett specifikt index, med vilket elementet kan nås. I en array ligger alla element i den ordning de lades in, och ordningen kan inte ändras.

Arrayer kan innehålla element av alla datatyper som språket stöder, inklusive variabler.

Exempel 1 – Grundläggande syntax

```
1.    [1, ”Hello”, ”World”]

  Den här arrayen innehåller alltså elementen 1, ”Hello” och
  ”World”
```

Exempel 2 – Lägga till element

```
1.    var = [1, 2, 3]
2.    var + ”Hello”

  Arrayen i variablen var kommer nu att vara [1, 2, 3, ”Hello”],
  där elementet ”Hello” lades till sist i arrayen.
```

Exempel 3 – Ta bort element

```
1.    var = [1, 2, 3, ”Hello”]
2.    var – ”Hello”

  Minusoperatorn tar i det här exemplet bort alla förekomster av
  ”Hello” i arrayen var.
```

Exempel 4 – Komma åt element

```
1.    var = [1, 2, 3]
2.    elem = var[2]

  Variablen elem kommer i det här exemplet att innehålla en 3:a,
  eftersom elementet på den första positionen alltid ligger på
  indexvärde 0. Om angivet indexvärde är negativt, kan ett
  element hämtas ut räknat från slutet av arrayen. Indexvärdet –1
  är sista elementet.
  Element kan även hämtas ut genom att skriva [1,2,3][0].
```

2.8.2. String

En string en följd av tecken inom citattecken. En string kan innehålla alla tecken utom backslash.

Exempel 1 – Grundläggande syntax

```
1.    ”Hello World!”
```

Exempel 2 – Lägga ihop strängar

```
1.    ”Hello ” + ”World”
  Resultatet blir ”Hello World”
```

Exempel 3 – Ta ut ett tecken

```
1.    Var = ”Hello World”
2.    h = var[0]

  Variabeln h kommer att innehåll bokstaven ”H”
  Om angivet indexvärde är negativt,  kan ett tecken hämtas ut
  räknat från slutet av strängen. Indexvärdet –1 är sista
  tecknet.
  Tecken kan även hämtas ut genom att skriva ”Hello World”[0].
```

2.8.3. Integer

En integer är alla positiva och negativa heltal.

Exempel 1 – Generell syntax

```
1.    2
2.    234
```

2.8.4. Float

En float är alla positiva och negativa decimaltal. Decimaltecknet består av en punkt.

Exempel 1 – Generell syntax

```
1.    2.5
2.    3.14
```

2.8.5. Range

En range fungerar som en array, med undantaget att en range används för att skapa en förifylld array med angivet start och slutvärde, där start och slutvärdet ingår i arrayen.

Syntaxen för att skapa ett range-object är två heltal separerade med två punkter.

Det andra talet måste vara större än eller lika med det första talet, annars kommer en tom array att skapas

Exempel 1 – Generell syntax

```
1.    1..2
2.    => [1, 2]
3.    155..159
4.    => [155, 156, 157, 158, 159]
```

2.8.6. Bool

En bool är en datatyp som enbart kan innehålla värdena true och false. En bool används ofta som en flagga för att indikera status i programmets körning, som till exempel i kodsnutten nedan, vilken hoppar ur en upprepningssats om flag har satts till false.

Exempel 1 – Användning av bool

```
1.    if not flag then
2.      break

  Variabeln flag innehåller true eller false, och if-satsen körs
  endast om flag har värde false.
```

2.8.7. Typkontroller och operationer på datatyper

ImgPL är ett dynamiskt typat språk, vilket innebär att man i deklarationen av variabler inte behöver ange vilken typ variabeln ska innehålla. Det gör att det blir lättare att byta typ på variablerna. Allt man behöver göra är att tilldela variabeln en annan datatyp.

Innehållet i variabeln kommer att kastas bort när något nytt tilldelas till den.

Exempel 1 – Dynamisk tilldelning

```
1.    var = 2
2.    var = ”hello”
```

Tabell 3: Operationer på aritmetiska uttryck

Operation	Resultat
Integer +-/*% Integer	Nytt heltal av uträkningen
Integer +-/*% Float Float +-/* %Integer Float +-/*% Float	Heltal görs om till decimaltal, och resultatet blir ett decimaltal.

String + String	Strängarna blir ihopslagna till en sträng.
String - String	Om högerledet har exakt ett tecken, tas alla förekomster av tecknet bort från vänsterledet, annars lämnas vänsterledet oförändrat.
String */ String	Stöds ej!
String * Integer String * Float	Decimaltal trunkeras och resultatet blir en sträng med vänsterledet upprepat så många gånger som högerledet anger.
Integer +-/* String Integer +-/* Float	Stöds ej!
Array + Array	Elementen i högerledet läggs till sist i arrayen i vänsterledet.
Array - Array	Alla element i vänsterledet som finns i högerledet tas bort från arrayen i vänsterledet.
Array /* Array Range /* Range	Stöds ej!
Array + String Array + Integer Array + Float Range + String Range + Integer Range + Float	Högerledet läggst till som ett element i slutet av datatypen i vänsterledet.
Array + Range	Högerledet konverteras till en array från vilken alla element läggs till i slutet av arrayen i vänsterledet.
Array - Integer Array - Float Range - Integer Range - Float	Alla förekomster av högerledet tas bort från vänsterledet.
Array - String Array - Range Range - String Range - Range	Alla element från högerledet tas bort från datatypen i vänsterledet.
Range + Range Range + Array	Båda leden konverteras till arrayer och resulterar i att alla element i högerledet läggs till sist i vänsterledet.

2.9. Variabler

Det här avsnittet handlar om variabler utanför klassdefinitioner. För information om variabler i klasser hänvisas till kapitel 2.14.2: Medlemsvariabler.

En variabel fungerar som en identifierare i vilken man kan spara datatyper för senare åtkomst. En variabel måste börja på en gemen (a-z) men första bokstaven kan följas av versaler och gemener, samt siffror och understreck.

2.9.1. Tilldelning

En variabel tilldelas på följande sätt.

Exempel 1 – Tilldelning och åtkomst

```
1.    number = 5
2.    hello = ”Hello World”

  Här har variablerna sparats undan och kan kommas åt genom att
  referera till variablen.

3.    println << hello

  Ovanstående sats kommer att hämta ut värdet på variablen,
  alltså ”Hello World”, och skriva ut den på skärmen.
```

2.9.2. Användning i uttryck

Alla variabler kan användas i uttryck och när uttrycken evalueras hämtas först datatypen som variabeln pekar på.

Ett aritmetiskt uttryck är matematiskt uttryck där båda leden är siffror. I dessa kan man använda integer, float och även variabler som innehåller dessa datatyper.

2.10. Aritmetiska uttryck

Om både integer och float används i samma uträkning så blir svaret alltid en float, för att behålla precisionen.

I uttryck kan även parenteser användas för att ange prioritet.

2.10.1. Addition och subtraktion

För användning av additions- och subtraktionsoperatorn mellan andra datatyper, se kapitel 2.8.7: Typkontroller och operationer på datatyper.

Addition och subtraktion använder sig av tecknen + och -, precis som vanligt. Vid uträkning gäller de vanliga matematiska reglerna, vilket betyder att addition och subtraktion har lägre prioritet än de andra operatorerna så om man t.ex. vill utföra addition före exempelvis multiplikation så kan man använda sig av parenteser.

Exempel 1 – Addition och subtraktion

```
1.    3+3+2
2.    => 8
3.    3-3+2.0
4.    => 2.0
5.    1-(2+8)
6.    => -9
```

2.10.2. Multiplikation och division

För användning av multiplikations- och divisionsoperatorn mellan andra datatyper, se kapitel 2.8.7: Typkontroller och operationer på datatyper.

Mutiplikation och division använder sig av tecknen * och /. De har också högre prioritet än addition och subtraktion precis enligt de matematiska reglerna.

Exempel 1 – Multiplikation och division

```
1.    2*2
2.    => 4
3.    2/3.0
4.    => 0.6666667
5.    2*(5+1)
6.    => 12
```

2.10.3. Upphöjt

Upphöjt använder sig av tecknet ** (två multiplikationsoperatorer precis efter varandra). Upphöjt har högst prioritet av de aritmetiska operatorerna och beräknas alltså först.

Om man vill upphöja exempelvis 3 i 3+1 så får man använda sig av parenteser för att ange prioritet.

Exempel 1 - Upphöjt

```
1.    2**2
2.    => 4
3.    2**2+5
4.    => 9
5.    3**(1+2)
6.    => 27
```

2.10.4. Modulo

Modulo ger resten vid heltalsdivision. Prioriteten är samma som för multiplikation och division.

```
1.    5 % 2
2.    => 1
3.    37.5 % 2
4.    => 1.5
5.    10 % 3
6.    => 1
```

2.11. Logiska uttryck

Logiska uttryck är uttryck som kan evalueras till ett sanningsvärde, alltså true eller false. Uttryck av typen true och false sätts samman till sammansatta uttryck med hjälp av logiska operatorer och evalueras i sin tur till ett sanningsvärde. NOT-operatorn har högst prioritet, AND och OR har samma prioritet.

Exempel 1 – OR-operatorn

```
1.    true or false
2.    true or true
3.    false or true
4.    false or false

  Or-operatorn innebär att ett av uttrycken på endera sidan om
  operatorn måste vara sann för att uttrycket som helhet ska vara
  sant. Alltså är alla uttryck ovan sanna utom den sista.
```

Exempel 2– AND-operatorn

```
1.    true and true
2.    true and false
3.    false and true
4.    false and false
```

```
  AND-operatorn innebär att båda uttrycken på endera sidan
  operatorn måste vara sann för att uttrycket som helhet ska vara
  sant. I exemplet ovan finns alltså bara ett uttryck som är sant
  och tre som är falska.
```

Exempel 3 – NOT-operatorn

```
1.    not true
2.    not false

  NOT-operatorn inverterar uttrycket direkt efter operatorn. Om
  uttrycket till höger om operatorn är sant blir uttrycket om
  helhet falskt, och vice versa.
```

Exempel 4 – Sammansatta logiska uttryck

```
1.    not true or false
2.    true and not false

  Det översta uttrycket blir falskt, det andra sant.
```

2.12. Funktioner

Funktioner är ett smidigt sätt att kunna återanvända sin kod, om samma operation behöver göras på flera ställen i programmet.

Till en funktion kan ett eller flera värden skickas in, och funktionen gör sin operation på eller med dessa värden. Från en funktion kan man returnera ett värde, vilket man senare kan använda i sitt program. Mer om det i kapitel 2.12.3: Returvärden.

Notera att funktioner måste definieras innan de kan anropas.

2.12.1. Definiera funktioner

En funktion måste vara definierad innan den kan anropas, vilket innebär att den måste skrivas ovanför funktionsanropet.

Exempel 1 – Definiera en funktion

```
1.    func sayHello()
2.          println << ”Hello!”

  Här har vi en mycket enkel funktion, vars enda syfte är att
  skriva ut en text på skärmen. Funktionens namn är sayHello och
  tar inga argument och inget returneras.
```

Exempel 2 – Definiera en funktion med argument

```
1.    func sayHello(fname, lname)
2.      println << ”Hello ” + fname + ” ” + lname

  Den här funktionen är en utökning av exemplet ovan. Till
  funktionen är det tänkt att man ska skicka in två strängar, en
  med ett förnamn och en med ett efternamn. Funktionen kommer
  sedan att skriva ut ”Hello” följt av förnamn och efternamn.

  Notera att båda argumenten måste skickas in, annars kommer den
  ena variabeln att vara odefinierad när den ska användas i
  println-satsen. Detta orsakar ett felmeddelande och att
  exekveringen av programmet omedelbart avslutas.
```

2.12.2. Funktionsanrop

För att ett funktionsanrop ska vara möjligt måste funktionen vara definierad innan anropet görs.

Ett funktionsanrop innebär att man exekverar kod som är placerad på ett annat ställe i programmet och möjliggör därigenom återvinning av kod, samt renare kod.

Exempel 1 – Funktionsanrop utan parametrar

```
1.    func sayHello()
2.      println << ”Hello”
3.    sayHello()

4.    Den tredje raden är alltså ett funktionsanrop. Koden i
  funktionen skulle inte köras om inte den tredje raden fanns.
```

Exempel 2 – Funktionsanrop med parametrar

```
1.    func add(n, m)
2.      println << n + m
3.    addTen(20, 10)
```

Exempel 3 – Funktionsanrop till returnerande funktion

```
1.    func power(number)
2.          return number**2
3.    value = power(2)

  Funktionen power returnerar här ett värde som vi sparar i
  variabeln value. Värdet som sparas är 2 upphöjt till 2, alltså
```

Exempel 4 – Funktionsanrop i uttryck och satser

```
  Funktionsanrop kan användas mitt i uttryck, under förutsättning
  att funktionen returnerar ett värde som kan användas i det
  aktuella uttrycket.

1.    welcome = sayHello() + getName(id)
  Variabeln welcome kommer att innehålla ihopslagningen av de
  returnerade värdena från sayHello() och getName(id), där vi kan
  tänka oss att sayHello() returnerar ”Hello ” och getName(id)
  returnerar ett namn som hämtas ur en databas med nyckeln id.

  Funktionsanrop kan även göras i kontrollsatser och
  upprepningssatser.

2.    if stillNotCrashed() then
3.      return crash()

  Anropet stillNotCrashed() evalueras till ett sanningsvärde,
  alltså true eller false, som if-satsen använder för att avgöra
  om blocket ska köras eller ej. Detta kräver att
  stillNotCrashed() returnerar ett sanningsvärde, och inget
  annat.
  Funktionen stillNotCrashed() returnerar det värde som
  returneras från funktionen crash().
```

2.12.3. Returvärden

En funktion kan returnera värden av alla datatyper som stöds av språket, och ingen returtyp behöver specificeras i förväg.

När en returnsats påträffas kommer det efterföljande uttrycket att omedelbart returneras från funktionen och efterföljande rader i funktionen ignoreras.

Exempel 1 – Användning av return i funktioner

```
1.    func testNumber(number)
2.      if number < 10 then
3.        return true
4.      else
5.        return false
6.
7.    var = testNumber(9)

  Efter körning av exemplet ovan kommer variabeln var att ha
  värdet false. Detta eftersom den första if-satsen evalueras
  till false, och därmed körs return false. Uttrycket efter
  return kommer alltså att sparas i var. Om det inte finns någon
  variabel som tar emot ett returnerat uttryck försvinner det
  returnerade värdet och kan inte användas senare.
```

Endast en datatyp kan returneras i varje returnsats. Fler objekt eller värden kan returneras samtidigt om de först placeras i en Array.

Om inget returneras från funktionen returneras null.

2.13. Rekursivitet

Rekursivitet innebär att en funktion kan kallas inifrån sig själv. Funktionen kommer att fortsätta att anropa sig själv rekursivt tills dess att ett slutvärde returneras, och först då kommer språket att evaluera och räkna ut slutresultatet med början längst ner i den anropsstack som bildas.

ImgPL har stöd för detta som följande exempel visar. Koden finns även i examples/recursivity.imgpl.

Exempel 1 – Rekursivitet med Fibonnacci

```
1.    func fib(input)
2.      if input is 0 then
3.        return 0
4.      if input is 1 then
5.        return 1
6.      return fib(input-1) + fib(input-2)

7.    print << "Enter a number: "
8.    get >> number
9.    println << "The " + number + "th fibonnacci number is " +
  fib(number)
```

För mer information om rekursivitet, hänvisas till kapitel 2.13: Rekursivitet.

2.14. Klasser

Med klasser kan man skapa egna objekt som inehåller både variabler och funktioner.

Om du har en klass som heter Bil, och vill skapa en ny instans av den så skriver du volvo = Bil.new(). Då kommer ett nytt bil-objekt att läggas i variabeln volvo.

För att man ska kunna göra något intressant med en instans så måste man ange funktioner som hämtar ut variabler eller skriver ut något. Mer om detta i nästa avsnitt.

2.14.1. Definiera klasser

För att skapa en ny klass så använder man sig av nyckelordet class. Efter det så anger man namnet på klassen. Ett klassnamn måste, till skillnad från variabelnamn och funktionsnamn, börja med stor bokstav.

På raderna under klassnamnet så kan man ange medlemsvariabler. Dessa måste börja med ett semikolon för att skilja dem från andra variabler.

Efter medlemsvariblerna kan man deklarera de funktioner som ska finnas med i klassen.

En funktion måste ha namnet new för att man ska kunna skapa en ny instans av klassen. Observerna att klassern bara kan definieras en gång. Dubbel definition ger ett felmeddelande.

Exempel 1 – Klassdefinition

```
1.    class Car
2.      ;color
3.      func new(color)
4.        ;color = color
5.      func getColor()
6.        return ;color
7.      func printCar()
8.        println << ”This car is ” + ;color
```

2.14.2. Medlemsvariabler

Medlemsvariabler är variabler som är unika för varje instans av klassen.

Om du i exemplet ovan skapar två olika instanser av klassen Car, så kan den ena ”bilen” kan ha färgen gul och den andra ”bilen” kan ha färgen röd.

Vilka olika medlemsvariabler som finns i en klass måste deklareras på raden efter klassnamnet (se exempel 1 nedan). Man kan inte tilldela dem på samma rad utan det måste göras inuti en funktion, så som exempelvis new-funktionen.

En klass behöver däremot inte innehålla medlemsvariabler, vilket man kan se i exempel 2 nedan.

Exempel 1 – Deklarering av medlemsvariabler

```
1.    class Car
2.          ;color
3.          ;regnr
4.          ;manifacturer
5.          func new(color,regnr,manif)
6.            ;color = color
7.            ;regnr = regnr
8.            ;manifacturer = manif
9.         func printCar()
10.          println << ”Manifacturer: ” + ;manifacturer
11.
12.   println << ”Color: ” + ;color
13.   println << ”Reg. number: ” + ;regnr
```

Exempel 2 – En klass utan medlemsvariabler

```
1.    class Hello
2.      func new()
3.        println <<  ”New instance created.”
        func printHello
4.        println << ”Hello!”
```

2.14.3. Metoder

För att kunna skapa nya instanser av en klass så måste klassen innehålla metoden new().

Utöver den metoden så kan man skapa andra metoder som t.ex. skriver ut till skärmen, byter värde på instansvariabler eller hämtar instansvariabler.

För att skapa en ny metod i en klass deklarerar man dem precis som vanliga funktioner. Enda skillnaden är att metoderna som är deklarerade i en klass kan komma åt instansens medlemsvariabler.

2.14.4. Instansiering av objekt

För att skapa en ny instans av en klass så skriver man **instansnamn = Klassnamn.new()**. Då skapas en ny instans av klassen vilken läggs i variabeln instansnamn. På den kan man senare anropa de funktioner som finns definierade för den klassen.

2.14.5. Metodanrop

För att kunna anropa en metod som är deklarerad inuti en klass så måste man först skapa en instans utav klassen som man lägger i en variabel.

Efter att man har skapat en instans så kan man anropa en funktion på den instansen genom att skriva **variabel.funktionsnamn(inparametrar)**.

Exempel 1 – Metodanrop på en instans

```
1.    class Person
2.          ;name
3.        func new(name)
4.          ;name = name
5.        func changeName(newName)
6.          ;name = newName
7.    me = Person.new(”Leif”)
8.    me.changeName(”Gunnar”)

  Innan me.changeName har körts så innehåller me en instans av
  klassen Person där variabeln innehåller strängen ”Leif”.

  Efter att funktionen changeNmane har körts så har ;name bytt
  värde till metoden changeNames inparameter, vilken i detta fall
  är ”Gunnar”.
```

2.15. Inläsning och Utskrift

Inläsning och utskrift innebär att man kan kommunicera med användaren genom att skriva meddelanden, ställa frågor och agera på användaren svar.

2.15.1. Utskrift till användare

Utskrift sker med hjälp av tre printsatser. Dessa är print, println och errprint. Print och println skriver ut till stdout, medan errprint skriver ut till stderr.

Exempel 1 – print

```
1.    print << ”Hello world!”

  Den här printsatsen kommer inte att orsaka en radbrytning efter
  utskriften, vilket kan vara bra om man vill ta input från
  användaren, då frågan står på samma rad som svaret skrivs på.
```

Exempel 2 – println

```
1.    println << ”Hello World!”

  Skriver ut texten med en radbrytning efter. Eventuell utskrift
  efter denna kommer på egen rad.
```

Exempel 3 – errprint

```
1.    errprint << ”Error”

  Skriver ut ett felmeddelande. Utskriften är lite mer detaljerad
  än de övriga eftersom de skrivs ut med escape-tecken som inte
  syns i den vanliga utskriften.
```

2.15.2. Inläsning från användare

Inläsning sker med get. Get kan läsa in text till en existerande variabel, eller kan skapa en ny om variabelnamnet inte finns sedan tidigare.

Exempel 1 – get

```
1.    get >> answer

  Om variabeln answer inte är definierad tidigare kommer den nu
  att skapas. Om den text som användaren skriver in enbart består
  av siffror eller ett decimaltal kommer den att konverteras till
  antingen Integer eller Float, annars behålls texten oförändrad.
```

Exempel 2 – Användning av användares input

```
1.    print << ”Guess a number: ”
2.    get >> answer
3.    if answer is 10 then
4.      println << ”You guessed right!”
5.    else
6.      println << ”You guessed wrong”

  Om användaren matar in en siffra kommer konverteras den till
  ett heltal eller decimaltal, och ifsatsen kommer att fungera.
```

2.16. Kontrollsatser

Kontrollsatser är kodblock som kontrollerar om ett visst villkor är uppfyllt. Endast om villkoret är uppfyllt körs koden i blocket. En kontrollsats i ImgPL består av en if-sats, vilken kan innehålla if else-satser och else-satser.

2.16.1. If-satser

If-satser är nödvändiga för att kunna kontrollera värden som returneras från funktioner eller matas in från användaren, så att programmet kan reagera på detta och ändra sitt beteende.

En if-sats består alltid av if-satsen själv, men kan följas av en eller flera if else-satser och en else-sats.

Exempel 1 – En enkel if-sats

```
1.    if 2 < 10 then
2.      println << ”2 is less than 10”

  Endast om uttrycket 2 < 10 är uppfyllt så körs blocket och
  texten skrivs ut.
```

Exempel 2 – If-sats med fler villkor

```
1.    continue = true
2.    correct = 10
3.    get >> answer
4.    if continue and answer is correct and notCrashed() then
5.      println << ”Congratulations ” + getName()

  Ovanstående if-sats evalueras till true om och endast om answer
  har värdet 10 och notCrashed() returnerar true.
```

Exempel 3 – If else-sats

```
1.    get << answer
2.    if answer < 10 then
3.           println << ”Higher”
4.    else if answer > 10 then
5.           println << ”Lower”
6.    else if answer is 10 then
7.           println << ”You found the secret number!”

  If-satsen kontrollerar om användaren gissade för lågt eller för
  högt och ger i så fall ledtrådar. Om användaren gissar rätt
  skrivs en annan text ut.

  Eftersom den andra if else-satsen måste vara sann om de andra
  satserna är falska, skulle vi lika gärna kunna använda en else-
  sats istället.
```

Exempel 4 – Else-satsen

```
1.    if 30 > 20 then
2.          println << ”Mathematics don’t work anymore!”
3.    else
4.          println << ”Mathematics still work!”

  En if-sats kan endast innehålla en else-sats, och det är denna
  som kommer att exekveras om inget av de andra villkoren är
  uppfyllt.
  Eftersom 30 inte är mindre än 20 kommer else-satsen att köras,
  och vi kan konstatera att matematiken fortfarande fungerar.
```

2.17. Upprepningssatser

Upprepningssatser används när en viss operation behöver genomföras fler än en gång. Det kan till exempel vara att upprepa en fråga till användaren tills dess att denne svarar rätt på frågan.

I ImgPL finns det två typer av upprepningssatser. For-loopen används huvudsakligen när man vet hur många gånger upprepningen behöver ske eller om man vill stega igenom en array eller range.

While-loopen används huvudsakligen när man inte vet när upprepningen ska avslutas.

2.17.1. For-loopar

For-loopar används för att iterera genom exempelvis arrayer och strängar. För varje varv i loopen kommer variabeln att tilldelas nästa värde i den datatyp som itereras.

Exempel 1 – Iterera en Range

```
1.    for i in 1..50 do
2.          print << i**2

  Eftersom 1..50 är en Range och kommer att göras om till en
  Array fylld med siffror mellan 1 och 50, kommer den här loopen
  att skriva ut alla siffror mellan 1 och 50, upphöjda till 2.
  Rangen kan tilldelas en variabel innan den används:

3.    range = 1..50
4.    for i in range do
5.          println << i**2
```

Exempel 2 – Iterera en Array

```
1.    array = [”Hej ”, ”hej ”, ”leverpastej!”]
2.    for i in array do
3.           print << I

  Den här arrayen kommer att skriva ut texten ”Hej hej
  leverpastej” på en och samma rad.
```

2.17.2. While-loopar

En while-loop fortsätter tills dess att ett villkor är uppfyllt. Loopen kan avbrytas dels genom att ändra villkoret inne i loopen, t.ex. räkna upp en variabel, eller genom att använda break.

Exempel 1 – While-loop med break

```
1.    while true do
2.           print << ”Enter a number: ”
3.           get >> answer
4.           if answer is 10 then
5.             break

  Den här loopen kommer att fortsätta tills dess att användaren
  matar in rätt tal, först därefter avslutas loopen med break.
```

Exempel 2 – While-loop med slutvärde

```
1.    count = 0
2.    while count <= 20 do
3.           get >> answer
4.           if answer is not 10 then
5.             count = count + 1

  Den här loopen kommer att fortsätta tills dess att användaren
  svarat någonting annat än 10 20 gånger.
```

2.18. Räckvidd

I detta kapitel behandlas variablers, funktioners och klassers räckvidd. Med detta menas varfrån kan man nå en variabel. Om man har en variabel i en funktion, kan man då nå den utanför funktionen?

2.18.1. Variablers scope

De variabler som deklareras i vänstermarginalen utav koden, d.v.s. de som inte är indenterade, kan man komma åt var man än är i programmet. De är alltså globala variabler. Om man däremot skriver en if-sats där man deklarerar en ny variabel så kommer man inte åt den utanför if-satsen.

Ett indenterat block, som exempelvis en if-sats, kommer alltså åt alla de variabler som är deklarerade i blocken som som ligger innan.

Exempel 1 – Variabelscope i if-satser

```
1.    a = ”global” // Denna variabel är global.
2.    if a is ”global” then
3.      b = ”not global” // Denna variabel är inte global
4.      if b is ”not global” then
5.        print << b // Man kommer åt b här
6.      print << b // B är definierad här också
7.    print << b // Här är b inte definierad, vilket ger ett
  felmeddelande.
8.    print << a // Ok
```

2.18.2. Funktioners scope

En funktion kan deklareras var som helst i programmet, även i if-satser och inuti andra funktioner. Alla funktioner som deklareras är globala, d.v.s. att om man deklarerar en funktion inuti en if-sats så kommer man senare åt den var som helst i sitt program.

Exempel 1 – Funktioners scope

```
1.    z = 0
2.    func add(x=0,y=0)
3.           return x+y+z
4.    println << add(1,2) // skriver ut 3

  Notera att vi kommer åt variabeln z inne i funktionen.
```

2.18.3. Klassers scope

Liksom en funktion kan klasser deklareras var som helst i programmet, och dessa blir globala och kan alltså kommas åt överallt. Det innebät även att en klass definierad inuti en funktion blir åtkomlig utanför funktionen, under förutsättning att funktionen har körts en gång.

2.19. Indentering

ImgPL är ett indenteringsstyrt språk. Detta innebär att koden måste skrivas i form av block, där kod som tillhör blocket är indenterat ett steg in. Detta borgar för renare och mer lättläst kod. Att koden är korrekt indenterad är nödvändigt för att koden ska tillhöra rätt block.

2.19.1. Koda indenteringsstyrt

Om koden tillhör ett block måste den indenteras ett steg, men vad är ett block? Ett block är ett antal rader kod som tillhör en kontrollsats, upprepningssats, funktion eller klass. Block fungerar även som en begränsning av räckvidden för variabler, läs mer om det i
kapitel 2.18: Räckvidd.

Exempel 1 – Indenterade block

```
1.    if input < 10 then
2.      println << ”Hello World!”

  Utskriftsraden är ett block, som tillhör if-satsen. Den här if-
  satsen skulle mycket väl kunna vara ett block i exempelvis en
  upprepningssats. Blocken kan alltså vara nästlade oändligt
  många nivåer.
```

Exempel 2 – Nästlade block

```
1.    func outer()
2.      func inner()
3.        println << ”Hello World!”
4.      inner()

  Printsatsen är ett block som tillhör inner(), vilket i sin tur
  är ett block i outer(). Funktionsanropet till inner tillhör
  outers block, men anropet hade lika gärna kunnat finnas utanför
  outerfunktionen, eftersom funktioner är globala. Dock krävs det
  att ett anrop till outer körs innan anropet till inner, annars
  kommer inte inner att vara definierad, vilket resulterar i ett
  fel.
```

Exempel 3 – Felaktig indentering

```
1.    for i in [1, 2, 3, 4] do
2.      print << ”Type name: ”
3.    get >> name
4.      println << name

  Här har vi ett kodexempel som inte kommer att fungera. Eftersom
  get-satsen inte är indenterad kommer for-loopen att betraktas
  som avslutad. Detta kommer att skriva ut ”Type name: ” fyra
  gånger innan exekveringen når get-satsen.
  Efter get-satsen uppstår en krash – printsatsen är felaktigt
  indenterad. Förmodligen kan man anta att koden var tänkt att se
  ut så här istället:

5.    for i in [1, 2, 3, 4] do
6.      print << ”Type name: ”
7.      get >> name
8.      println << name
```

2.20. Felhantering

Om något fel uppstår antingen i den interpreterade miljön eller i ett program skrivs ett felmeddelande ut. Om det är ett program som körs från fil kommer dessutom programmet att omedelbart avslutas efter att felet uppstått. I interpreterat läge avslutas inte interpretatorn, men felmeddelandet visas och operationen genomförs inte.

2.20.1. Klassificering

ImgPL har följande typer av felmeddelanden.

Tabell 4: Klassificering av felmeddelanden

Klass	Betydelse
<IO_ERROR>:	Uppstår när en fil inte finns eller inte kan öppnas eller skrivas.
<TYPE_ERROR>:	Uppstår när en operation inte kan genomföras för objektet eller objekten.
<IMAGE_ERROR>:	Generellt felmeddelande som uppstår om funktioner på bildobjekt inte fungerar.
<NOT_DEFINED_ERROR>:	Uppstår om en variabel, funktion eller klass inte är definierad.
<ARGUMENT_ERROR>:	Uppstår om antalet argument som en funktion anropas med inte stämmer med funktionsdefinitionen.
<INDEX_OUT_OF_RANGE_ERROR>:	Uppstår när indexvärdet är för högt eller för lågt, och det därmed inte finns något element på positionen.
<NO_INSTANCE_ERROR>:	Uppstår när vid ett anrop till en medlemsfunktion i en klass, och variabeln funktionen kallas på inte tillhör klassen.
<MEMBER_FUNCTION_CALL_ERROR>:	Uppstår när man kallar på en klassmetod utan man har skapat en instans av klassen.
<SYNTAX_ERROR>:	Uppstår när det finns ett syntaktiskt fel i programkoden.
<PARSE_ERROR>:	Uppstår när inmatat rad i interpretatorn inte kan hanteras eller innehåller syntaktiska fel.
<DOUBLE_DEFINITION_ERROR>:	Uppstår när en klass definieras om det redan finns en klass med det namnet.
<ZERO_DIVISION_ERROR>:	Uppstår endast vid division med noll.

3. Systemdokumentation

ImgPL är ett språk skrivet i Ruby. Det använder sig av en Recursive Descent-parser, vilken ingick som verktyg i kursen, för att parsa och sedan bygga upp ett syntaxträd utav de noder som är definierade i filen tree.rb.

Efter att parsern har byggt upp ett syntaxträd så evalueras detta genom att köra funktionen eval på noderna. Vid inläsningen av en fil så läggs begin- och end-tokens till för att underlätta vid parsningen av indenterade block. Om då indenteringen skulle vara fel så läggs begin- och end-tokens in på fel plats, vilket sedan parsern kommer att upptäcka och ge ett felmeddelande.

Efter att begin- och end-token har lagts till så kommer koden att delas upp i lexikaliska enheter (tokens). De token som inte har filtrerats bort, som exempelvis mellanslag kommer att fortsätta igenom parsern, som matchar en följd av tokens som regler. Utifrån dessa regler kommer senare ett syntaxträd att skapas.

Syntaxträdet består utav olika klasser, som alla har en eval-funktion. Den funktionen kommer sedan att köra eval-funktionen på noderna som ligger under den i hierarkin. För mer information om parsern, syntaxträdet och tokens se kapitel 3.1: Implementering.

3.1. Implementering

Språket är implementerat med hjälp av en lexer, en parser och ett syntaxträd. När programkoden lästs in som en sträng skickas den först genom lexern vilken delar upp koden i tokens, som sedan används av parsern. Parsern i sin tur skapar ett syntaxträd i vilken programmet evalueras.

3.1.1. Lexer

Lexern tar hand den sträng som kommer och delar upp den i mindre delar. Det handlar om att matcha förekomster av ord och mönster i strängen och dela upp dessa i sina egna beståndsdelar, så kallade tokens. Uppdelningen görs med funktionen token(), vilken tar ett reguljärt uttryck. Från den här funktionen returnerar vi instanser av superklassen Token. Vid matchning av en token sparas denna undan och används senare för matchning mot språkets grammatik.

3.1.2. Parser

Vid parsning kommer de tokens som lexern sparat undan att matchas mot språkets grammatik, som specificeras med hjälp av funktionen match(). Att en regel har matchats innebär att parsern har hittat ett mönster av ord i koden, vilka bildar uttryck eller andra konstruktioner. När en regel har matchats returnerar vi instanser till klasser i syntaxträdet. Parsern sparar undan dessa tills dess att matchningen är klar för alla tokens. Sen kommer parsern att evaluera den första instansen.

Eftersom instansen i sin tur evaluerar instansen för den regel som följer och så vidare kommer eval() att köras på alla klasser vars instanser parsern har sparat. Vid parsningen används en Recursive Descent-parser.

3.1.3. Syntaxträd

Syntaxträdet finns i filen tree.rb, och består av klasser med funktionerna initialize() och eval().

Funktionen initialize används enbart när instanserna skapas av parsern när den matchar tokens. Funktionen eval() körs bara när en instans kör en annan instans evalfunktion. I syntaxträdet kommer alla instanser att hålla minst en instans av en annan nod i trädet.

3.1.4. Kodstandard

Vi har inte använt oss av någon specifik kodstandard, utan vi har kodat efter en del kodkonventioner som vi själva har kommit överens om.

- Variabelnamn börjar med liten bokstav med orden separerade med understreck.
- Funktionsnamn följer samma standard som variabelnamn.
- Klassnamn börjar med stor bokstav och även där är orden separerade med understreck, precis som funktions- och variabelnamn.

3.2. Grammatik

```
<newline> ::= “\n”+
<indent>
<dedent>
<dent>

<comment> ::= “//” /.*/ <newline>

            | “/*” <newline> /.*/ <newline> “*/”


<class_name> ::= /^[A-Z]+\w*/
<identifier> ::= /^[a-z]+\w*/
<class_var> ::= /^;[a-z]+\w*/
<begin>
<end>
<program> ::= <lines>


<lines> ::= <line> <lines>
            | <line>

<line> ::= <dent> <newline>
            | <dent> <expr> <newline>
            | <dent> <stmt> <newline>

<stmt> ::= <class_def>
            | <func_def>
            | <if_else_stmt>
            | <assign_stmt>
            | <print_stmt>

            | <get_stmt>

            | <while_stmt>

            | <for_stmt>

            | <break>

            | <return> <expr>
```

```
<class_def> ::= “class” <class_name> <newline> <indent> <begin>
                <newline> <dent> <class_vars_def> <dent> <class_funcs_def>
                <newline> <dedent> <end>

<class_call> ::= <class_name> “.” “new” “(“ <arg_list> “)”
           | <class_name> “.” “new” “(“ “)”


<class_funcs_def> ::= <class_funcs_def> <newline> <dent> <func_def>
           | <func_def>

<class_vars_def> ::= <class_vars_def> <dent> <class_var_def>
           | <class_var_def>

<class_var_def> ::= <class_var> <newline>

<func_def> ::= “func” <identifier> “(“ <param_list> “)” <newline> <block>
           | “func” <identifier> “(“ “)” <newline> <block>

<param_list> ::= <param_list> “,” <param>
           | <param>

<param> ::= <assign_stmt>
           | <identifier>

<while_stmt> ::= “while” <condition> “do” <newline> <block>

<for_stmt> ::= “for” <identifier> <rel_op> <datatypes> “do” <newline> <block>

<if_else_stmt>::= <if_stmt> <newline> <dent> <else_if_stmts>
                <newline> <dent> <else_stmt>
           | <if_stmt> <newline> <dent> <else_if_stmts>

           | <if_stmt> <newline> <dent> <else_stmt>

           | <if_stmt>


<if_stmt> ::= “if” <condition> “then” <newline> <block>
```

```
<else_if_stmts> ::= <else_if_stmts> <newline> <dent> <else_if_stmt>
            | <else_if_stmt>

<else_if_stmt> ::= “else” “if” <condition> “then” <newline> <block>
<else_stmt> ::= “else” <newline> <block>
<block> ::= <indent> <begin> <newline> <lines> <dedent> <end>


<assign_stmt>::= <identifier> “=” <class_call>
            | <identifier> “=” <expr>

            | <class_var> “=” <class_call>

            | <class_var> “=” <expr>


<print_stmt> ::= “print” “<<” <expr>
            | “println” “<<” <expr>
            | “errprint” “<<” <expr>

<get_stmt> ::= “get” “>>” <identifier>

<expr> ::= <add_expr>
            | <condition>

<add_expr> ::= <add_expr> <add_operator> <multi_expr>
            | <multi_expr>

<multi_expr> ::= <multi_expr> <multi_operator> <unary_expr>
            | <unary_expr>

<unary_expr> ::= <add_operator> <unary_expr>
            | <power_expr>

<power_expr> ::= <datatypes>
            | “(“ <add_expr> “)”
            | <power_expr> “**” <power_expr>

<datatypes> ::= <int_type>
            | <float_type>
            | <range_type>
            | <null>

            | <array>
```

```
            | <string>

            | <datatypes> “[“ expr “]”

            | <class_var> “.” <identifier> “(“ <arg_list> ”)”

            | <class_var> “.” <identifier> “(“”)”

            | <identifier> “(“ <arg_list> “)”

            | <identifier> “.” <identifier> “(“”)”

            | <identifier> “(“ <arg_list> “)”
            | <identifier> “(“ “)”
            | <identifier>
            | <class_var>

<array> ::= “[“ <elements> ”]”
            | “[“ “]”

<elements> ::= <elements> “,” <expr>
            | <expr>

<arg_list> ::= <arg_list> “,” <arg>
            | <arg>

<arg> ::= <expr>
<add_operator> ::= “+” | “-”
<multi_operator> ::= “*” | “/” | “%”
<rel_operator> ::= “not” “in” | “is” “not” | “<=” | “>=” | “in”

            | “is” | “<” | “>”

<condition> ::= <or_expr>

<or_expr> ::= <or_expr> “or” <and_expr>
            | <and_expr>

<and_expr> ::= <and_expr> “and” <not_expr>
            | <not_expr>

<not_expr> ::= “not” <not_expr>
            | <rel_expr>

<rel_expr> ::= <add_expr> <rel_operator> <add_expr>
```

```
| “(“ <condition> “)”
| <datatypes>

| “true”

| “false”
```

3.3. Packetering

Paketet består av följande filer.

- ImgPL
  - Exekveras först användaren när språket startas. Ämnad för /usr/local/bin. Här avgörs om språket ska köras i interpreterat läge eller ej.
- ImgPL.rb
  - Innehåller lexer, grammatikregler och förmanipulering av koden.
- tree.rb
  - Innehåller syntaxsträdet.
- scopehandler.rb
  - Innehåller hjälpklass för att hantera räckvidd.
- image.rb
  - Innehåller funktioner för bildhantering.
- rdparse.rb
  - Innehåller själva parsern, med tokenizer.
- examples/
  - Exempelfiler skrivna i ImgPL:
  - classes.imgpl
  - functions.imgpl
  - images.imgpl
  - math.imgpl
  - recursivity.imgpl
  - faculty.imgpl
  - smiley.jpg
- testcase.rb
  - Testfall
- error.txt
  - Vår allas hjälte Homer Simpsson i ASCII
- makefile
  - Innehåller installationsscript

4. Reflektion

Vi började med att skriva planeringen för hur vårt språk skulle se ut och implementeras. Vi drömde om att skapa ett helt objektorienterat språk i stil med Ruby. Samtidigt skulle språket vara indenteringsstyrt i stil med Python. Indenteringsstyrt är det fortfarande, men det krävde en del tänkande och modifierande av strängar för att det skulle fungera. Det är nämligen nödvändigt att hålla reda på hur mycket varje kodblock är indenterad. Vi löser detta genom att matcha token för ”\\t”, och i samband med det så konstaterade vi att vi var tvugna att matcha denna token för varje rad för att kunna avgöra om indenteringsnivån minskat eller ökat. Samtidigt lade vi till begin och end-tokens före och efter ett indenterat kodblock. Detta lade vi till vid inläsningen av filen för att förenkla parsningen. Fullständigt objektorienterat blev det dock inte. Det visade sig för svårt att implementera i reglerna. Vi kom i alla fall inte på hur det skulle gå till.

Vår grammatik för språket var helt perfekt och felfri. Trodde vi. Det visade sig att parsern var betydligt noggrannare angående prioritetsordningen än vad vi förväntade oss, och vi blev tvungna att bryta ner reglerna i mindre regler. I början bestod grammatiken av <stmts> och <exprs>, men detta medförde att programmet enbart kunde innehålla statements eller expressions och inte båda samtidigt.

Till en början körde vi på som vi lärt oss i kursen TDP007 men skapade inget syntaxträd. Vi lyckades få en if-sats att fungera även utan syntaxträd, men saker och ting evaluerades vid matchningen av reglerna vilket medförde att programmet evaluerades i fel ordning.

Först efter att ha funderat ett tag på hur vårt språk inte kunde fungera som vi tänkt, tjuvtittar vi på några projekt gjorda förra året, och konstaterar snabbt att den stora skillnaden är just syntaxträdet. Efter att ha skrivit om det mesta och implementerat ett syntaxträd började saker och ting att fungera nästan av sig självt. Implementeringen innehåller många specialfall, inte minst för <return_stmt> och <break_stmt> som omedelbart måste hindra efterföljande kod att evalueras. Och så var det alla typkontroller som måste spridas ut i noderna. Tyvärr blev inte noderna så små som vi trodde från början, då de snabbt fylldes av kontrollsatser för typkontrollerna och specialfallen.

För att vårt språk ska fungera med indentering har vi varit tvugna att göra ganska mycket textformatering innan vi släpper in strängen med programmet till parsern. På grund av

formateringen har vi fått problem med att matcha tokens. Det blir inga lättlästa reguljära uttryck. Speciellt har vi stött på problem med att matcha kommentarer och blockkommentarer, så vi har löst detta genom att ta bort dessa innan lexern tar över. Även tomma rader tas bort innan lexning.

Det svåraste har varit att förstå sig på hur parsern fungerar, och hur prioriteten för reglerna påverkar hur parsern tolkar koden. Att det skulle vara svårt visste vi, men det var värre än vi förväntade oss. Vi trodde att bildredigeringen skulle vara den svåraste delen att implementera, men det visade sig faktiskt vara det lättaste. Vi hade inte gjort någon undersökning av vilka möjligheter och externa bibliotek som vi kunde använda innan vi bestämde oss för att göra ett språk för bildredigering.

5. Bilagor

Bifogat till rapporten är:

- Programkod
    - ImgPL
    - ImgPL.rb
    - tree.rb
    - scopehandler.rb
    - image.rb
- Makefile

```ruby
#!/usr/bin/env ruby
# -*- coding: utf-8 -*-
############################
# File:    ImgPL           #
# Authors:                 #
# Mattias Roback (matro414) #
# Magnus Suther (magsu191)  #
############################

# Require the right file depending on if the
# package is installed or not
if File.exists?("/usr/local/lib/ImgPL") then
  require '/usr/local/lib/ImgPL/ImgPL'
else
  if not File.exists?("ImgPL.rb") then
    abort("Could not find required file 'ImgPL.rb'! Aborting.")
  else
    require 'ImgPL'
  end
end

if ARGV.length > 1 or ARGV[0].eql?("-help") or ARGV[0].eql?("-h") then
  # Show help and usage information
  puts "Welcome to ImgPL - Image Programming Language!"
  puts "Usage: "
  puts "One argument: Evaluate a program (argument must be a path to a .imgpl file)"
  puts "No argument: Run in interpreted mode"
  puts "-h, -help: Show this text"
  exit
end

if not ARGV[0].nil? then
  # A file was given as argument, lets evaluate it
  if not File.exists?(ARGV[0]) then
    puts "File not found, check spelling!"
    exit
  elsif not File.extname(ARGV[0]).eql?(".imgpl")
    puts " #{File.extname(ARGV[0])} is not a valid file extension, only .imgpl is supported!"
    exit
  end
  ImgPL.new.eval_file(ARGV[0])
else
  # No file given, start in interpreted mode
  ImgPL.new.run()
end

```

```ruby
# -*- coding: utf-8 -*-
############################
# File:    ImgPL.rb        #
# Authors:                 #
# Mattias Roback (matro414) #
# Magnus Suther (magsu191)  #
############################

# Require the right files, depending on if the language is
# installed to /usr/local/* or not.
if File.exists?("/usr/local/lib/ImgPL") then
  require '/usr/local/lib/ImgPL/rdparse'
  require '/usr/local/lib/ImgPL/tree'
  require '/usr/local/lib/ImgPL/scopehandler'
else
  if not File.exists?("rdparse.rb") then
    abort("Could not find required file 'rdparse.rb'! Aborting.")
  else
    require 'rdparse'
  end
  if not File.exists?("tree.rb") then
    abort("Could not find required file 'tree.rb'! Aborting.")
  else
    require 'tree'
  end
  if not File.exists?("scopehandler.rb") then
    abort("Could not find required file 'scopehandler.rb'! Aborting.")
  else
    require 'scopehandler'
  end
end

begin
  require 'RMagick'
rescue
  abort("Could not find required module 'RMagick'! Aborting.")
end
require 'fileutils'


def printMascot()
  # Prints our mascot: Homer in ascii art!
  if File.exist?("/usr/local/lib/ImgPL/error.txt") then
    puts File.read("/usr/local/lib/ImgPL/error.txt")
  else
    puts File.read("error.txt") if File.exist?("error.txt")
  end
end

##### Modifications to Ruby-classes #####
class Integer
  def split()
    # Split the number, and return an array with each
    # number as separate items. Used to loop through each number, ie
    # for a in 1234
    a = self.to_s.split("").to_a.map{|a|a.to_i}
    a
  end
  def include?(rhs)
    # Check if the value of self contains the number in rhs
    self.to_s.include? rhs.to_s
  end
end

class Fixnum
```

```ruby
  def length()
    self
  end
end

class Float
  def split()
    # Split the number, and return an array with each
    # number as separate items. Used to loop through each number, ie
    # for a in 1234
    self.to_s.split(".").to_a.map{|a|a.to_i}
  end
  def length()
    self
  end
  def include?(rhs)
    # Check if the value of self contains the number in rhs
    self.to_s.include? rhs.to_s
  end
end

class Array
  # Expand the Array class with functions to compare
  # the length of self and rhs
  def <(rhs)
    return self.length < rhs.length
  end
  def >(rhs)
    return self.length > rhs.length
  end
  def <=(rhs)
    return self.length <= rhs.length
  end
  def >=(rhs)
    return self.length >= rhs.length
  end
end

class String
  # Expand the String class with functions to compare
  # the length of self and rhs
  def <(rhs)
    return self.length < rhs.length
  end
  def >(rhs)
    return self.length > rhs.length
  end
  def <=(rhs)
    return self.length <= rhs.length
  end
  def >=(rhs)
    return self.length >= rhs.length
  end
end

class Range
  # Expand the Range class with functions to compare
  # the length of self and rhs
  def length()
    self.end-self.begin
  end
  def <(rhs)
    return self.length < rhs.length
  end
  def >(rhs)
```

```
    return self.length > rhs.length
  end
  def <=(rhs)
    return self.length <= rhs.length
  end
  def >=(rhs)
    return self.length >= rhs.length
  end
end


##### Tokens #####
class Token
  # Super class for tokens. When tokens are matched
  # with regular expressions, some of then are returned
  # as instances of the subclasses.
  def initialize(value=nil)
    @value = value
  end

  def value
    @value
  end
end

# Define the subclasses of class Token.
# These are all just using the methods in Token,
# no other functionality is added.
class Indent < Token;end
class Dent < Token;end
class Dedent < Token;end
class Newline < Token;end
class Begin < Token;end
class End < Token;end
class While < Token;end
class For < Token;end
class Do < Token;end
class Break < Token;end
class Print < Token;end
class Get < Token;end
class If < Token;end
class Else < Token;end
class Then < Token;end
class Add_op < Token;end
class Multi_op < Token;end
class Rel_op < Token;end
class Func < Token;end
class Return < Token;end
class Out_op < Token;end
class In_op < Token;end
class Class_token < Token;end
class Not_token < Token;end
class Or_token < Token;end
class And_token < Token;end
class New_token < Token;end
class Power_op < Token;end

class ImgPL
  def initialize
    @parser = Parser.new("ImgPL") do
      @indent_level = 1
      ##### Strings #####
      token(/^\"([^\"])*\"/) {|m|String_type.new(m)}
      ##### Newlines #####
      token(/^(\\n)+/) {|_|Newline.new}
```

```
196         ##### Indents, Dedents & Dents #####
197         token(/^(\\t)+/)do|m|
198           # When we find a tab token, we need to count
199           # the number of tabs to keep track of the current
200           # indentation level.
201           # If the indentation increases, returns a Indent.
202           # If indentation decreases, returns a Dedent.
203           # If the indentation is unchanged, returns a Dent.
204           new_indent = m.split("t").length
205           dif = (new_indent-@indent_level).abs
206           if new_indent > @indent_level
207             @indent_level += dif
208             Indent.new(dif)
209           elsif new_indent == @indent_level
210             Dent.new(dif)
211           else new_indent
212             @indent_level -= dif
213             Dedent.new(dif)
214           end
215         end
216         ##### Operators #####
217         token(/^[\+|\-]/){|m| Add_op.new(m)}
218         token(/^\*\*/) {|m| Power_op.new} # Match square root operator
219         token(/^\<\</){|_| Out_op.new}
220         token(/^\>\>/){|_| In_op.new}
221         token(/^[\*|\/|\%]/){|m| Multi_op.new(m)}
222         token(/^((<=)|(>=)|<|>) /){|m| Rel_op.new(m.split(" ").join(" "))}
223         ##### Begin & End Tokens #####
224         token(/^_begin/){|_| Begin.new}
225         token(/^_end/){|_| End.new}
226         ##### Keywords and Identifiers #####
227         token(/^not\s+in\s+/){|m| Rel_op.new(m.split(" ").join(" "))}
228         token(/^is\s+not\s+/){|m| Rel_op.new(m.split(" ").join(" "))}
229         token(/^[A-Za-z]\w*/) do |m|
230           case m
231           when "null" then Null.new
232           when "in","is" then Rel_op.new(m)
233           when "true","false" then Bool_type.new(m)
234           when "or" then Or_token.new(m)
235           when "and" then And_token.new(m)
236           when "not" then Not_token.new(m)
237           when "println","print","errprint" then Print.new(m)
238           when "get" then Get.new
239           when "if" then If.new
240           when "else" then Else.new
241           when "then" then Then.new
242           when "while" then While.new
243           when "for" then For.new
244           when "do" then Do.new
245           when "break" then Break.new
246           when "return" then Return.new
247           when "class" then Class_token.new
248           when "func" then Func.new
249           else
250             if m[0..0].scan(/[A-Z]/).empty?
251               Identifier.new(m)
252             else
253               Class_name.new(m)
254             end
255           end
256         end
257         ##### Spaces #####
258         token(/ +/)
259         ##### Int, Float, Range #####
260         token(/^\d+\.\.\d+/){|m|s,e = m.split("..");Range_type.new(s.to_i,e.t
```

```
_i)}
261         token(/^\d+\.\d+/) {|m| Float_type.new(m.to_f) }
262         token(/^\d+/) {|m| Int_type.new(m.to_i) }
263         ##### Class variables #####
264         token(/^;[a-z]+\w*/){|m| Class_var.new(m)}
265         ##### The rest #####
266         token(/^./) {|m| m }
267 
268         ##### Rules #####
269         start :program do
270           match(:lines){|line|Program.new(line).eval()}
271         end
272 
273         rule :lines do
274           match(:line,:lines){|line,lines|Lines.new(line,lines)}
275           match(:line){|line|Lines.new(line)}
276         end
277 
278         rule :line do
279           match(Dent,Newline){|_,_|Line.new()}
280           match(Dent,:expr,Newline){|_,expr,_|Line.new(expr)}
281           match(Dent,:stmt,Newline){|_,stmt,_|Line.new(nil,stmt)}
282         end
283 
284         rule :stmt do
285           match(:class_def){|stmt|Stmt.new(stmt)}
286           match(:func_def){|stmt|Stmt.new(stmt)}
287           match(:if_else_stmt){|stmt|Stmt.new(stmt)}
288           match(:assign_stmt){|assign|Stmt.new(assign)}
289           match(:print_stmt){|print|Stmt.new(print)}
290           match(:get_stmt){|get|Stmt.new(get)}
291           match(:while_stmt){|stmt|Stmt.new(stmt)}
292           match(:for_stmt){|stmt|Stmt.new(stmt)}
293           match(Break){|_|Break_stmt.new()}
294           match(Return,:expr){|_,expr|Return_stmt.new(expr)}
295         end
296 
297         rule :class_def do
298           match(Class_token,Class_name,Newline,Indent,Begin,Newline,Dent,:cla
    ss_vars_def,Dent,:class_funcs_def,Newline,Dedent,End){|_,id,_,_,_,_,_,vars,
    _,funcs,_,_,_|Class_def.new(id,funcs,vars)}
299           match(Class_token,Class_name,Newline,Indent,Begin,Newline,Dent,:cla
    ss_funcs_def,Newline,Dedent,End){|_,id,_,_,_,_,_,block,_,_,_|Class_def.new(
    id,block)}
300         end
301 
302         rule :class_call do
303           match(Class_name,'.',Identifier,'(',:arg_list,')'){|class_name,_,fu
    nc_name,_,args,_|Class_call.new(class_name,func_name,args)}
304           match(Class_name,'.',Identifier,'(',')'){|class_name,_,func_name,_,
    _|Class_call.new(class_name,func_name)}
305         end
306 
307         rule :class_funcs_def do
308           match(:class_funcs_def,Newline,Dent,:func_def){|funcs,_,_,func|Clas
    s_funcs_def.new(func,funcs)}
309           match(:func_def){|func|Class_funcs_def.new(func)}
310         end
311 
312         rule :class_vars_def do
313           match(:class_vars_def,Dent,:class_var_def){|vars,_,var|Class_vars_d
    ef.new(var,vars)}
314           match(:class_var_def){|var|Class_vars_def.new(var)}
315         end
316 
```

```
317       rule :class_var_def do
318         match(Class_var,Newline){|var,_|Class_var_def.new(var)}
319       end
320
321       rule :func_def do
322         match(Func, Identifier,'(',:param_list,')', Newline, :block){|_,id,
    _,params,_,_,block| Func_def.new(id, params, block)}
323         match(Func, Identifier,'(',')', Newline, :block){|_,id,_,_,_,block|
     Func_def.new(id, nil, block)}
324       end
325
326       rule :param_list do
327         match(:param_list,',', :param){|params,_,param| Param_list.new(para
    ms, param)}
328         match(:param){|param| Param_list.new(nil, param)}
329       end
330
331       rule :param do
332         match(:assign_stmt) {|stmt| Param.new(stmt)}
333         match(Identifier) {|id| Param.new(id)}
334       end
335
336       rule :while_stmt do
337         match(While,:condition,Do,Newline,:block){|_,cond,_,_,block|While_s
    tmt.new(cond,block)}
338       end
339
340       rule :for_stmt do
341         match(For,Identifier,Rel_op,:datatypes,Do,Newline,:block){|_,id,op,
    data,_,_,block|For_stmt.new(id,op,data,block)}
342       end
343
344       rule :if_else_stmt do
345         match(:if_stmt,Newline,Dent,:else_if_stmts,Newline,Dent,:else_stmt)
    do|stmt,_,_,stmt2,_,_,stmt3|
346           If_else_stmt.new(stmt,stmt2,stmt3)
347         end
348         match(:if_stmt,Newline,Dent,:else_if_stmts){|stmt,_,_,stmt2|If_else
    _stmt.new(stmt,stmt2)}
349         match(:if_stmt,Newline,Dent,:else_stmt){|stmt,_,_,stmt2|If_else_stm
    t.new(stmt,nil,stmt2)}
350         match(:if_stmt){|stmt|If_else_stmt.new(stmt)}
351       end
352
353       rule :if_stmt do
354         match(If,:condition,Then,Newline,:block){|_,cond,_,_,block|If_stmt.
    new(cond,block)}
355       end
356
357       rule :else_if_stmts do
358         match(:else_if_stmts,Newline,Dent,:else_if_stmt){|stmt2,_,_,stmt|El
    se_if_stmts.new(stmt,stmt2)}
359         match(:else_if_stmt){|stmt|Else_if_stmts.new(stmt)}
360       end
361
362       rule :else_if_stmt do
363         match(Else,If,:condition,Then,Newline,:block){|_,_,cond,_,_,block|E
    lse_if_stmt.new(cond,block)}
364       end
365
366       rule :else_stmt do
367         match(Else,Newline,:block){|_,_,block|Else_stmt.new(block)}
368       end
369
370       rule :block do
```

```
371         match(Indent,Begin,Newline,:lines,Dedent,End){|_,_,_,line,_,_|Block
    .new(line)}
372       end
373
374       rule :assign_stmt do
375         match(Identifier,'=',:class_call){|id,_,instance|Assign_stmt.new(id
    ,instance)}
376         match(Identifier,'=',:expr){|id,_,expr|Assign_stmt.new(id,expr)}
377         match(Class_var,'=',:class_call){|id,_,instance|Assign_stmt.new(id,
    instance)}
378         match(Class_var,'=',:expr){|id,_,expr|Assign_stmt.new(id,expr)}
379       end
380
381       rule :print_stmt do
382         match(Print,Out_op,:expr){|print,_,expr|Print_stmt.new(print.value,
    expr)}
383       end
384
385       rule :get_stmt do
386         match(Get,In_op,Identifier){|_,_,id|Get_stmt.new(id)}
387       end
388
389       rule :expr do
390         match(:add_expr){|add_expr|Expr.new(add_expr)}
391         match(:condition){|cond|Expr.new(cond)}
392
393       end
394
395       rule :add_expr do
396         match(:add_expr,:add_operator,:multi_expr){|add,op,multi|Add_expr.n
    ew(add,op,multi)}
397         match(:multi_expr){|multi|Add_expr.new(multi)}
398       end
399
400       rule :multi_expr do
401         match(:multi_expr,:multi_operator,:unary_expr){|multi,op,unary|Mult
    i_expr.new(multi,op,unary)}
402         match(:unary_expr){|unary|Multi_expr.new(unary)}
403       end
404
405       rule :unary_expr do
406         match(:add_operator,:unary_expr){|op,unary|Unary_expr.new(unary,op)
    }
407         match(:power_expr){|power|Unary_expr.new(power)}
408       end
409
410       rule :power_expr do
411         match(:datatypes){|datatype|Power_expr.new(datatype)}
412         match('(',:add_expr,')'){|_,expr,_|Power_expr.new(expr)}
413         match(:power_expr,Power_op,:power_expr){|power,_,expr| Power_expr.n
    ew(expr,power)}
414       end
415
416       rule :datatypes do
417         match(Int_type){|int|Datatypes.new(int)}
418         match(Float_type){|float|Datatypes.new(float)}
419         match(Range_type){|range|Datatypes.new(range)}
420         match(Null){|null|Datatypes.new(null)}
421         match(:array){|array|Datatypes.new(array)}
422         match(String_type){|string|Datatypes.new(string)}
423         match(:datatypes,'[',:expr,']'){|expr1,_,expr2,_|Subscript_expr.new
    (expr1,expr2)}
424         match(Class_var,'.',Identifier,'(',:arg_list,')'){|var,_,func,_,arg
    s,_|Instance_func_call.new(var,func,args)}
425         match(Class_var,'.',Identifier,'(',')'){|var,_,func,_,_|Instance_fu
```

```
c_call.new(var,func)}
426           match(Identifier,'.',Identifier,'(',:arg_list,')'){|var,_,func,_,ar
      gs,_|Instance_func_call.new(var,func,args)}
427           match(Identifier,'.',Identifier,'(',')'){|var,_,func,_,_|Instance_f
      unc_call.new(var,func)}
428           match(Identifier,'(',:arg_list,')'){|id,_,args,_|Func_call.new(id,a
      rgs)}
429           match(Identifier,'(',')'){|id,_,_|Func_call.new(id)}
430           match(Identifier){|ident|Get_var.new(ident)}
431           match(Class_var){|var|Get_var.new(var)}
432         end
433
434         rule :array do
435           match('[',:elements,']'){|_,elements,_| Array_type.new(elements)}
436           match('[',']'){|_,_|Array_type.new()}
437         end
438
439         rule :elements do
440           match(:elements,',',:expr){|elements,_,expr|Elements.new(expr,eleme
      nts)}
441           match(:expr){|expr|Elements.new(expr)}
442         end
443
444         rule :arg_list do
445           match(:arg_list,',', :arg){|args,_,arg| Arg_list.new(arg,args)}
446           match(:arg){|arg| Arg_list.new(arg)}
447         end
448
449         rule :arg do
450           match(:expr) {|expr| Arg.new(expr)}
451         end
452
453         rule :add_operator do
454           match(Add_op){|op|op.value}
455         end
456
457         rule :multi_operator do
458           match(Multi_op){|op|op.value}
459         end
460
461         rule :rel_operator do
462           match(Rel_op){|op|op.value}
463         end
464
465         rule :condition do
466           match(:or_expr)
467         end
468
469         rule :or_expr do
470           match(:or_expr,Or_token,:and_expr){|expr,_,expr2|Or_expr.new(expr,e
      xpr2)}
471           match(:and_expr){|expr|Or_expr.new(expr)}
472         end
473
474         rule :and_expr do
475           match(:and_expr,And_token,:not_expr){|expr,_,expr2|And_expr.new(exp
      r,expr2)}
476           match(:not_expr){|expr|And_expr.new(expr)}
477
478         end
479         rule :not_expr do
480           match(Not_token,:not_expr){|token,expr|Not_expr.new(expr,token)}
481           match(:rel_expr){|expr|Not_expr.new(expr)}
482         end
483
```

```
484         rule :rel_expr do
485           match(:add_expr,:rel_operator,:add_expr){|expr1,op,expr2|Rel_expr.n
    ew(expr1,op,expr2)}
486           match('(',:condition,')'){|_,expr,_|Rel_expr.new(expr)}
487           match(:datatypes){|expr|Rel_expr.new(expr)}
488           match(Bool_type){|bool|Datatypes.new(bool)}
489         end
490       end
491     end
492
493     ##################### INTERPRETED MODE ####################
494     def run(command = nil)
495       begin
496         # Make the logger quiet
497         @parser.logger.level = Logger::WARN
498
499         if command.nil? then
500           loop do
501             str = getInput(0)
502
503             if str.eql?("exit") or str.eql?("quit") then
504               $stdout.puts "Bye!"
505               exit
506             end
507
508             # Find import command, and import the given file.
509             if not str.scan(/^import\s+\<.+\>/).empty? then
510               importFile(str)
511               str = ""
512             end
513
514             # Add a tab at the beginning of every line.
515             # Thanks to this, the token for tab is matched on
516             # every line, and we can keep track of the indentation.
517             str = str.gsub(/(\\n)/,"\\n\\t").insert(0,"\\t")+"\\n"
518
519             if not str.empty? then
520               out = @parser.parse(str)
521               # Output the value returned, if any
522               $stdout.puts "=> #{out.inspect}" if !out.nil? and !out.is_a?(Nu
    ll)
523             end
524
525           end
526         else
527           # Lets the testcases use this function to pre-format the string
528           # before evaluating
529
530           # Add a tab at the beginning of every line.
531           # Thanks to this, the token for tab is matched on
532           # every line, and we can keep track of the indentation.
533           command = command.gsub(/(\\n)/,"\\n\\t").insert(0,"\\t")+"\\n"
534           if not command.empty? then
535             print "=> "
536             # Always return whatever is returned
537             @parser.parse(command)
538           end
539         end
540         rescue NoMethodError, Interrupt
541           # The user most likely pressed
542           # ctrl-d or ctrl-c
543           $stdout.puts "Bye!"
544           exit
545         rescue Parser::ParseError
546           printMascot()
```

```
        $stderr.puts "<SYNTAX_ERROR>: Failed to interpret input."
        retry
      rescue => e
        printMascot()
        $stderr.puts e
        retry
      end
    end


  ##################### EVAL FILE ####################
  def eval_file(path, quiet = false)
  begin
    # Make the logger quiet
    @parser.logger.level = Logger::WARN

    file = File.read(path)
    str = file.chomp

    # Add a tab at the beginning of every line.
    # Thanks to this, the token for tab is matched on
    # every line, and we can keep track of the indentation.
    str = str.gsub(/(\n)/,"\\n\\t").insert(0,"\\t")+"\\n\\t\\n"
    str.gsub!(/\t/,"\\t")
    lines = str.split("\\n")
    final_lines = []
    previous_tabs = 1
    code_lines = []

    # Removes empty lines and block-comments.
    count = 0
    lines.each do |line|
      count += 1 if line.eql?("\\t/*") and count == 0
      code_lines << line if count.eql?(0) and
        !line.gsub(/\\t+/,"").gsub(/\s*/,"").empty?
      count -= 1 if line.eql?("\\t*/")
    end

    # Removes one-line-comments
    lines = []
    raise("<SYNTAX_ERROR>: Block comments do not match.") if count != 0
    code_lines.each do |line|
      index = line.index(/\/\//)
      if index.nil?
        lines << line
      else
        lines << line[0..index-1]
      end
    end
    lines << "\\t"

    #Adds begin and end-tokens before and after a
    #block, in order to keep track of indentation level
    #when several blocks is closed at the same time.
    lines.each do |line|
      tabs = line.scan(/\\t/).length
      if tabs == previous_tabs+1
        final_lines << ("\\t"*tabs)+"_begin "
        final_lines << line.clone
      elsif tabs < previous_tabs
        (previous_tabs-1).downto(tabs)do |i|
          final_lines << ("\\t"*i)+"_end "
        end
        final_lines << line
      else
```

```
612         final_lines << line
613       end
614       previous_tabs = tabs.to_i
615     end
616     str = final_lines.join("\\n")+"\\n"
617
618     out = @parser.parse(str)
619     $stdout.puts "Mission complete!" if not quiet
620
621   rescue NoMethodError,Interrupt
622     # Most likely, the user has pressed ctrl-d or ctrl-c
623     # somewhere, lets abort without a crash.
624     $stdout.puts "Bye!"
625     exit
626   rescue SystemStackError
627     printMascot()
628     $stderr.puts "<STACK_LEVEL_ERROR>: Stack level to deeeeeeeeeeeeeeep."
629   rescue Parser::ParseError
630     printMascot()
631     $stderr.puts "<SYNTAX_ERROR>: Program contains syntactic errors!"
632   rescue => e
633     printMascot()
634     $stderr.puts e
635   end
636   end
637 end
638
639
640 def getInput(indent)
641   # Let the user type multiline programs in the interpreter.
642   # Since we need to keep track of the indentation to be able to
643   # insert special characters and tokens, we solve this recursively.
644
645   $stdout.print "[ImgPL] " + ("··· " * indent)
646   input = gets.chomp
647
648   if input.start_with?("for ","if ","else if ","else ",
649                        "while ", "func ", "class ") then
650     input += "\\n" + ("\\t"*(indent+1)) + "_begin \\n" + ("\\t"*(indent+1))
     + getInput(indent+1)
651   end
652
653   # If there is no block, this is an expression that
654   # should be returned immediately. But only if it
655   # is not already inside a block.
656   return input if indent.eql?(0) or input.empty?
657
658   # Get the block. Block is considered complete when
659   # an empty line is input. The code below
660   # is reached only if a block is expected.
661   loop do
662     $stdout.print "[ImgPL] " + ("··· " * indent)
663     line = gets.chomp
664     if line.start_with?("for ","if ","else if ","else ",
665                         "while ","func ","class ") then
666       # Only run if line is not the first line in the block.
667       line += "\\n\\t_begin \\n" + ("\\t"*(indent+1)) + getInput(indent+1)
    + "\\n"
668     end
669
670     return input + "\\n" + ("\\t"*(indent-1)) + "_end " if line.empty?()
671     input += "\\n" + ("\\t"*(indent)) + line
672   end
673 end
674
```

```
675 def importFile(str)
676   # The import command lets users import/evaluate ImgPL files
677   # from the interpreter. This is used to make use of functions
678   # and/or classes declared in the imported file.
679   # Syntax for the import command: import <file.imgpl>
680 
681   file = str.split(/\s+/)[1].gsub(/[\<|\>]/, "")
682   if not file.nil? then
683     if not File.extname(file).eql?(".imgpl") then
684       file += ".imgpl"
685     end
686     if File.exists?(file) then
687       # Evaluate the file to store classes, functions and variables
688       # declared in it.
689       eval_file(file, true)
690     else
691       # File not found, raise error
692       raise("Import failed, file #{file.inspect} not found!")
693     end
694   end
695 end
696 
```

```ruby
# -*- coding: utf-8 -*-
############################
# File:    tree.rb         #
# Authors:                 #
# Mattias Roback (matro414) #
# Magnus Suther (magsu191)  #
############################

# Require the right files, depending on if the language is
# installed to /usr/local/* or not.
if File.exists?("/usr/local/lib/ImgPL") then
  require '/usr/local/lib/ImgPL/scopehandler'
  require '/usr/local/lib/ImgPL/image'
else
  if not File.exists?("scopehandler.rb") then
    abort("Could not find required file 'scopehandler.rb'! Aborting.")
  else
    require 'scopehandler'
  end
  if not File.exists?("image.rb") then
    abort("Could not find required file 'image.rb'! Aborting.")
  else
    require 'image'
  end
end


class Program
  def initialize(lines)
    @lines = lines
  end
  def eval()
    return_value = nil
    return_value = @lines.eval()
    return_value = nil if return_value.is_a?(Break_stmt) or
      return_value.is_a?(Return_stmt)
    return_value
  end
end

class Lines
  def initialize(line,lines=nil)
    @line = line
    @lines = lines
  end
  def eval()
    return_value = nil
    if @lines.nil?
      return_value = @line.eval()
    else
      return_value = @line.eval()
      return_value = @lines.eval() if !return_value.is_a?(Return_stmt) and
        !return_value.is_a?(Break_stmt)
    end
    return_value
  end
end

class Line
  def initialize(expr=nil,stmt=nil)
    @expr=expr
    @stmt=stmt
  end
  def eval()
    return_value = nil
```

```ruby
    if !@expr.nil?
      return_value = @expr.eval()
    elsif !@stmt.nil?
      return_value = @stmt.eval()
    end
    return_value
  end
end

class Expr
  def initialize(add_expr)
    @add_expr = add_expr
  end
  def eval()
    @add_expr.eval()
  end
end

class Add_expr
  def initialize(add_expr,op=nil,multi_expr=nil)
    @add_expr=add_expr
    @op=op
    @multi_expr=multi_expr
  end
  def eval()
    if @multi_expr.nil?
      return @add_expr.eval()
    else
      add = @add_expr.eval()
      multi = @multi_expr.eval()
      case @op
        #Addition
      when "+" then
        if add.is_a?(Array)
          if multi.is_a?(Array)
            #If both is array they are combined
            add + multi
          else
            #Else, put it in the last element
            add << multi
          end
        elsif add.is_a?(String)
          add + multi.to_s
        else
          begin
            add + multi
          rescue
            raise("<TYPE_ERROR>: Can't add #{multi.inspect} to " +
            " #{add.inspect}.")
          end
        end
        #Subtraction
      when "-" then
        if add.is_a?(Array)
          if multi.is_a?(Array)
            #Removes the elements in multi from add
            add - multi
          else
            #Removes all elements thats equal to multi
            add.delete_if{|elem|elem == multi}
          end
        elsif add.is_a?(String)
          #Removes all occurences of multi in add
          add.gsub(multi.to_s,"")
        else
```

```ruby
          begin
            add - multi
          rescue
            raise("<TYPE_ERROR>: Can't subtract #{multi.inspect} from "+
            "#{add.inspect}.")
          end
        end
      end
    end
  end
end

class Multi_expr
  def initialize(multi_expr,op=nil,unary_expr=nil)
    @multi_expr = multi_expr
    @op= op
    @unary_expr = unary_expr
  end
  def eval()
    if @unary_expr.nil?
      return @multi_expr.eval()
    else
      multi = @multi_expr.eval()
      unary = @unary_expr.eval()
      case @op
        #Multiplication
      when "*" then
        if unary.is_a?(Array) or unary.is_a?(String)
          raise("<TYPE_ERROR>: Can't multiply #{multi.inspect} with " +
          "#{unary.inspect}.")
        else
          multi * unary
        end
        #Division
      when "/" then
        if (multi.is_a?(Integer) or multi.is_a?(Float)) and
            (unary.is_a?(Integer) or unary.is_a?(Float))
          begin
            multi / unary
          rescue
            raise("<ZERO_DIVISION_ERROR>: Only Chuck Norris can " +
            "divide #{multi.inspect} by 0.")
          end
        else
          raise("<TYPE_ERROR>: Can't divide #{multi.inspect} by " +
          "#{unary.inspect}.")
        end
        #Modulo
      when "%" then
        if (multi.is_a?(Integer) or multi.is_a?(Float)) and
            (unary.is_a?(Integer) or unary.is_a?(Float))
          multi % unary
        else
          raise("<TYPE_ERROR>: Can't use modulo for #{multi.inspect} " +
          "and #{unary.inspect}.")
        end
      end
    end
  end
end

class Unary_expr
  def initialize(unary_expr,op=nil)
    @op = op
    @unary_expr = unary_expr
```

```ruby
  end
  def eval()
    if @op.nil?
      return @unary_expr.eval()
    else
      unary = @unary_expr.eval()
      if unary.is_a?(Integer) or unary.is_a?(Float)
        case @op
        when "+" then +unary
        when "-" then -unary
        end
      else
        raise("<TYPE_ERROR>: Can't use #{@op.inspect} for " +
        "#{unary.inspect}.")
      end
    end
  end
end

class Power_expr
  def initialize(expr,power_expr=nil)
    @power_expr = power_expr
    @expr = expr
  end
  def eval()
    if @power_expr.nil?
      return @expr.eval()
    else
      power = @power_expr.eval()
      expr = @expr.eval()
      if power.is_a?(Integer) or power.is_a?(Float) and
        expr.is_a?(Integer) or expr.is_a?(Float)
        return power ** expr
      else
        raise("<TYPE_ERROR>: Can't calculate #{power.inspect} ** " +
        "#{expr.inspect}.")
      end
    end
  end
end

class Rel_expr
  def initialize(expr1,op = nil,expr2 = nil)
    @expr1 = expr1
    @op = op
    @expr2 = expr2
  end
  def eval()
    expr1 = @expr1.eval()
    if !@expr2.nil?
      expr2 = @expr2.eval()
      len1 = expr1.length if !expr1.eql?(true) and !expr1.eql?(false)
      len2 = expr2.length if !expr2.eql?(true) and !expr2.eql?(false)

      raise("<TYPE_ERROR>: Operator <,>,<=,>= not " +
      "defined for boolean.") if ["<",">","<=",">="].include?(@op) and
      (len1.nil? or len2.nil?)

      # Compares length or objects depending on the operator
      case @op
      when "<" then len1 < len2
      when ">" then len1 > len2
      when "<=" then len1 <= len2
      when ">=" then len1 >= len2
      when "is" then expr1 == expr2
```

```ruby
      when "is not" then expr1 != expr2
      when "in" then expr2.include? expr1
      when "not in" then !expr2.include? expr1
      end
    else
      return expr1
    end
  end
end

class Datatypes
  def initialize(value)
    @value = value
  end
  def eval()
    return @value.eval()
  end
end

class Int_type
  def initialize(value)
    @value = value
  end
  def eval()
    @value
  end
end

class Float_type
  def initialize(value)
    @value = value
  end
  def eval()
    @value
  end
end

class String_type
  def initialize(value)
    @value = value
    @value = value[1..-2] if value =~ /\"/
  end
  def eval()
    @value
  end
end

class Bool_type
  def initialize(value)
    if value == "true"
      @value = true
    else
      @value = false
    end
  end
  def eval()
    @value
  end
end

class Range_type
  def initialize(s,e)
    @start = s
    @end = e
  end
```

```ruby
  def eval()
    return (@start..@end).to_a
  end
end

class Array_type
  def initialize(elements = nil)
    @elements = elements
  end
  def eval()
    @elements = @elements.eval() if !@elements.nil?
    @elements = [] if @elements.nil?
    @elements
  end
end

class Elements
  def initialize(expr,elements=nil)
    @expr = expr
    @elements = elements
  end
  def eval()
    if !@elements.nil?
      a = @elements.eval()
    else
      a = []
    end
    a << @expr.eval()
  end
end

class Identifier
  def initialize(var_name)
    @var_name = var_name
  end
  def eval()
    @var_name
  end
end

class Null
  def eval()
    Null.new()
  end
end

class Class_var
  def initialize(var_name)
    @var_name = var_name
  end
  def eval()
    @var_name
  end
end

class Stmt
  def initialize(stmt)
    @stmt = stmt
  end
  def eval()
    return_value = nil
    return_value = @stmt.eval()
    return_value
  end
end
```

```ruby

class Assign_stmt
  attr_accessor :identifier, :expr
  def initialize(id,expr)
    @identifier = id
    @expr = expr
  end
  def eval()
    id = @identifier.eval()
    Scope_handler.assign_variable(id, @expr.eval())
  end
end

class Print_stmt
  def initialize(type,expr)
    @type = type
    @expr = expr
  end
  def eval()
    expr = @expr.eval()
    if expr.is_a?(Hash)
      #Only happens if expression is a class defined by user.
      str = "Class:\t#{expr["class"]}"
      str += "\nVars: "
      expr["vars"].each{|key,value|str+="\t#{key} = #{value.inspect}\n"}
      expr = str
    elsif expr.is_a?(Magick::ImageList)
      #Happens if an Image is printed
      str = "Class:\tImage\n"
      expr.each{|image|str+="#{image.filename()}\n"}
      expr = str
    end
    case @type
    when "println" then $stdout.puts expr
    when "print" then $stdout.print expr
    when "errprint" then $stderr.puts "<ERRPRINT> #{expr}"
    end
    nil
  end
end

class Get_stmt
  def initialize(id)
    @id = id
  end
  def eval()
    in_value = $stdin.gets.chomp
    value = String_type.new(in_value)
    case in_value
    when /\A\d+\.\d+\z/ then value = Float_type.new(in_value.to_f)
    when /\A\d+\z/ then value = Int_type.new(in_value.to_i)
    end
    Assign_stmt.new(@id,value).eval()
  end
end

class Get_var
  def initialize(identifier)
    @identifier = identifier
  end
  def eval()
    variable = nil
    id = @identifier.eval()
    variable = Scope_handler.get_variable_value(id)
    variable
```

```ruby
  end
end

class If_else_stmt
  def initialize(if_stmt,else_if_stmts=nil,else_stmt=nil)
    @if_stmt = if_stmt
    @else_if_stmts = else_if_stmts
    @else_stmt = else_stmt
  end
  def eval()
    return_value = @if_stmt.eval()
    continue_else = true
    if return_value == true and !@else_if_stmts.nil?
      continue_else = @else_if_stmts.eval()
    end
    if continue_else == true and return_value == true and !@else_stmt.nil?
      return_value = @else_stmt.eval()
    end
    return_value = nil if return_value == true
    return_value = continue_else if continue_else.is_a?(Break_stmt) or
      continue_else.is_a?(Return_stmt)
    return_value
  end
end

class If_stmt
  def initialize(cond,block)
    @cond = cond
    @block = block
  end
  def eval()
    return_value = true
    Scope_handler.new_variable_scope()

    if(@cond.eval())
      return_value = @block.eval()
    end

    Scope_handler.pop_variable_scope()
    return_value
  end
end

class Else_if_stmts
  def initialize(else_if_stmt,else_if_stmts=nil)
    @else_if_stmt = else_if_stmt
    @else_if_stmts = else_if_stmts
  end
  def eval()
    return_value = true
    return_value = @else_if_stmt.eval()
    if(!@else_if_stmts.nil? and return_value==true)
      return_value = @else_if_stmts.eval()
    end
    return_value
  end
end

class Else_if_stmt
  def initialize(cond,block)
    @cond = cond
    @block = block
  end
  def eval()
    return_value = true
```

```
521     Scope_handler.new_variable_scope()
522
523     if(@cond.eval())
524       return_value = @block.eval()
525     end
526
527     Scope_handler.pop_variable_scope()
528     return_value
529   end
530 end
531
532 class Else_stmt
533   def initialize(block)
534     @block = block
535   end
536   def eval()
537     Scope_handler.new_variable_scope()
538     return_value = @block.eval()
539     Scope_handler.pop_variable_scope()
540     return_value
541   end
542 end
543
544 class While_stmt
545   def initialize(cond,block)
546     @cond = cond
547     @block = block
548   end
549   def eval()
550     return_value = nil
551     Scope_handler.new_variable_scope()
552
553     while(@cond.eval())
554       return_value = @block.eval()
555       break if return_value.is_a?(Break_stmt) or
556       return_value.is_a?(Return_stmt)
557     end
558
559     Scope_handler.pop_variable_scope()
560     return_value = nil if return_value.is_a?(Break_stmt)
561     return_value
562   end
563 end
564
565 class For_stmt
566   def initialize(id,op,data,block)
567     @id = id.eval()
568     @op = op
569     @data = data.eval()
570     @block = block
571     case @data
572     when Bignum,Fixnum then @data = @data.split()
573     when Float then @data = @data.split()
574     when String then
575       # If the string contains spaces, we loop word by word,
576       # otherwise we loop letter by letter.
577       if @data =~ /\s/
578         @data = @data.split(/\s/)
579       else
580         @data = @data.split(//)
581       end
582     end
583   end
584   def eval()
585     return_value = nil
```

```
586     Scope_handler.new_variable_scope()
587
588     for i in @data
589       Scope_handler.assign_variable_in_last_scope(@id, i)
590       return_value = @block.eval()
591       break if return_value.is_a?(Break_stmt) or return_value.is_a?(Return
    _stmt)
592     end
593
594     Scope_handler.pop_variable_scope()
595     return_value = nil if return_value.is_a?(Break_stmt)
596     return_value
597   end
598 end
599
600 class Block
601   def initialize(lines)
602     @lines = lines
603   end
604   def eval()
605     return_value = nil
606     return_value = @lines.eval()
607     return_value
608   end
609 end
610
611 class Condition
612   def initialize(cond)
613     @cond = cond
614   end
615   def eval()
616     return_value = @cond.eval()
617
618     if return_value != false and return_value != true
619       raise("<TYPE_ERROR>: Condition #{return_value.inspect} cannot " +
620       "be resolved to boolean.")
621     end
622
623     return_value
624   end
625 end
626
627 class Or_expr
628   def initialize(expr1,expr2 = nil)
629     @expr1 = expr1
630     @expr2 = expr2
631   end
632   def eval()
633     if @expr2.nil?
634       return @expr1.eval()
635     else
636       return @expr1.eval() || @expr2.eval()
637     end
638   end
639 end
640
641 class And_expr
642   def initialize(expr1,expr2 = nil)
643     @expr1 = expr1
644     @expr2 = expr2
645   end
646   def eval()
647     if @expr2.nil?
648       return @expr1.eval()
649     else
```

```
650       return @expr1.eval() && @expr2.eval()
651     end
652   end
653 end
654 
655 class Not_expr
656   def initialize(expr,flag = nil)
657     @expr = expr
658     @flag = flag
659   end
660   def eval()
661     if @flag.nil?
662       return @expr.eval()
663     else
664       return !@expr.eval()
665     end
666   end
667 end
668 
669 class Param_list
670   def initialize(params, param)
671     @params = params
672     @param = param
673   end
674   def eval()
675     if not @params.nil?
676       a = @params.eval()
677     else
678       a = []
679     end
680     a << @param.eval()
681   end
682 end
683 
684 class Param
685   def initialize(stmt)
686     @stmt = stmt
687   end
688   def eval()
689     if @stmt.is_a?(Assign_stmt)
690       id = @stmt.identifier.eval()
691       expr = @stmt.expr.eval()
692       [id,expr]
693     else
694       [@stmt.eval(),nil]
695     end
696   end
697 end
698 
699 class Func_call
700   def initialize(id,args=nil,class_name=nil)
701     @id = id
702     @args = args
703     @class_name = class_name
704   end
705   def eval()
706     id = @id.eval()
707     params = []
708     if @class_name.nil?
709 
710       raise("<NOT_DEFINED_ERROR>: Function \"#{id}()\" not " +
711       "defined.") if not Scope_handler.function_defined(id)
712 
713       # Marshal is used to do a "deep copy" of an object.
714       if Scope_handler.function_has_parameters(id) then
```

```
715                 func_params = Scope_handler.get_function_parameters(id)
716         params = Marshal.load(Marshal.dump(func_params))
717       end
718
719     else
720       function_defined = Scope_handler.class_function_defined(@class_name,
     id)
721       raise("<NOT_DEFINED_ERROR>: Function \"#{id}()\" "+
722       "not defined for class #{@class_name.inspect}.") if not function_def
    ined
723
724       if Scope_handler.class_function_has_parameters(@class_name, id) then
725         func_params = Scope_handler.get_class_function_parameters(@class_n
    ame, id)
726         params = Marshal.load(Marshal.dump(func_params))
727       end
728
729     end
730     if !@args.nil?
731       args = @args.eval()
732       raise("<ARGUMENT_ERROR>: Wrong number of "+
733       "arguments for function \"#{id}()\".") if args.length > params.lengt
    h
734       args.each_with_index{|elem,i|params[i][1] = elem if !elem.is_a?(Null
    )}
735       # Checks if a parameter is missing from the user
736       params.each{|e| raise("<ARGUMENT_ERROR>: Wrong number "+
737       "of arguments for function \"#{id}()\".") if e[1].nil?}
738     end
739     # Maps function parameters together with their identifier and
740     # puts them last in the variables scope.
741     scope = {}
742     params.map{|key,value|scope[key] = value}
743         scope.each{|a,e| raise("<ARGUMENT_ERROR>: Wrong number "+
744       "of arguments for function \"#{id}()\".") if e.nil?}
745
746     # Makes sure that a function cannot use
747     # variables from other scopes (only global).
748     number_of_scopes = Scope_handler.get_number_of_variable_scopes()
749     tmp_variables = Scope_handler.pop_variable_scope(number_of_scopes - 1)
750     Scope_handler.insert_variable_scope(tmp_variables.pop) if !@class_name
    .nil?
751     Scope_handler.insert_variable_scope(scope)
752
753     if @class_name.nil?
754       #If the function does not belong to a class
755       return_value = Scope_handler.get_function_block(id).eval()
756     else
757       #If the function is a class function.
758       return_value = Scope_handler.get_class_function_block(@class_name, i
    d).eval()
759     end
760     return_value = return_value.value.eval() if return_value.is_a?(Return_
    stmt)
761     Scope_handler.pop_variable_scope()
762
763     # Puts the other scopes variables back into the variables scope
764     tmp_variables << Scope_handler.pop_variable_scope() if !@class_name.ni
    l?
765     Scope_handler.merge_variable_scopes(tmp_variables)
766
767     return_value = Null.new if return_value.nil?
768     return_value
769   end
770 end
```

```ruby

class Arg_list
  def initialize(arg,args=nil)
    @arg = arg
    @args = args
  end
  def eval()
    if not @args.nil?
      array = @args.eval()
    else
      array = []
    end
    array << @arg.eval()
  end
end

class Arg
  def initialize(arg)
    @arg = arg
  end
  def eval()
    @arg.eval()
  end
end

class Break_stmt
  def eval()
    self
  end
end

class Return_stmt
  attr_accessor :value
  def initialize(value)
    @value = value
  end
  def eval()
    self
  end
end

class Subscript_expr
  def initialize(expr,index)
    @expr = expr
    @index = index
  end
  def eval()
    expr = @expr.eval()
    index = @index.eval()
    if !index.is_a?(Integer)
      raise("<TYPE_ERROR>: Index must be an integer, found " +
      "#{index.inspect}.")
    elsif !expr.is_a?(String) and !expr.is_a?(Array)
      raise("<TYPE_ERROR>: Can't use subscript operator for " +
      "#{expr.inspect}.")
    else
      if (index < expr.length) and (index >= -expr.length)
        if expr.is_a?(Array)
          expr[index]
        else
          expr[index..index]
        end
      else
        raise("<INDEX_OUT_OF_RANGE_ERROR>: Index #{index.inspect} " +
        "for #{expr.inspect} is out range.")
```

```ruby
      end
    end
  end
end

class Class_def
  def initialize(id,funcs,vars=nil)
    @id = id
    @funcs = funcs
    @vars = vars
  end
  def eval()
    id = @id.eval()
    raise("<DOUBLE_DEFINITION_ERROR>: Class #{id.inspect} already " +
    "defined.") if Scope_handler.class_defined(id) or id.eql?("Image")

    Scope_handler.define_class(id)

    @funcs.eval(id)
    @vars.eval(id) if !@vars.nil?
  end
end

class Class_vars_def
  def initialize(var,vars=nil)
    @var = var
    @vars = vars
  end
  def eval(id)
    @var.eval(id)
    @vars.eval(id) if !@vars.nil?
    nil
  end
end

class Class_name
  def initialize(name)
    @name = name
  end
  def eval()
    @name
  end
end

class Class_var_def
  def initialize(var)
    @var = var
  end
  def eval(class_name)
    var = @var.eval()
    Scope_handler.assign_class_variable(class_name, var, nil)
  end
end

class Class_funcs_def
  def initialize(func,funcs=nil)
    @func = func
    @funcs = funcs
  end
  def eval(id)
    @func.eval(id)
    @funcs.eval(id) if !@funcs.nil?
    nil
  end
end
```

```
901
902 class Func_def
903   def initialize(id, params, block)
904     @id = id
905     @params = params
906     @block = block
907   end
908   def eval(class_name=nil)
909     id = @id.eval()
910     params = nil
911     params = @params.eval() if !@params.nil?
912     if class_name.nil?
913       Scope_handler.define_function(id, params, @block)
914     elsif
915       Scope_handler.define_class_function(class_name, id, params, @block)
916     end
917     nil
918   end
919 end
920
921 class Class_call
922   def initialize(class_name,func_name,args=nil)
923     @class_name = class_name
924     @func_name = func_name
925     @args = args
926   end
927   def eval()
928     name = @func_name
929     func_name = @func_name.eval()
930     class_name = @class_name.eval()
931
932     #If a function is called on an image-object.
933     if class_name.eql?("Image")
934       raise("<MEMBER_FUNCTION_CALL_ERROR>: Must create new instance "+
935       "before calling member " +
936       "function \"#{func_name}()\".") if not func_name.eql?("new")
937       args = @args.eval()
938       imagelist = Magick::ImageList.new()
939
940       # Searches for images in the given directories
941       #-Block begin-
942       args.each do |arg|
943         arg = arg[0..-2] if arg[-1..-1].eql?("/")
944         filetypes = [".jpg",".jpeg",".gif",".bmp",".png"]
945         if File.exists?(arg) and File.directory?(arg) then
946           # Add all images in arg to imagelist, non-recursive
947           entries = Dir.entries(arg)
948           entries.each do |ent|
949             file = arg + "/" + ent
950             # Opens images with the correct filetype.
951             if filetypes.include?(File.extname(ent).downcase)
952               imagelist.read(file)
953             end
954           end
955         else
956           raise("<IO_ERROR>: File #{arg.inspect} not found.") if
957           !File.exists?(arg)
958
959           if !(filetypes.include?(File.extname(arg).downcase))
960             raise("<IO_ERROR>: #{arg.inspect} is not a valid filetype.")
961             return Null.new
962           else
963             imagelist.read(arg)
964           end
965
```

```
966           end
967         end
968         #-Block end-
969         return imagelist
970
971         #If function is called on another object.
972       else
973         if func_name.eql?("new")
974           scope = Scope_handler.get_class_variables(class_name)
975           Scope_handler.insert_variable_scope(scope.clone())
976
977           Func_call.new(name,@args,class_name).eval()
978         else
979           raise("<MEMBER_FUNCTION_CALL_ERROR>: Must create new instance "+
980           "before calling member function \"#{func_name}()\".")
981         end
982         return {"class"=>class_name, "vars"=>Scope_handler.pop_variable_scop
    e()}
983       end
984     end
985   end
986
987   class Instance_func_call
988     def initialize(var_name,func_name,args=nil)
989       @var_name = var_name
990       @func_name = func_name
991       @args = args
992     end
993     def eval()
994       var = Get_var.new(@var_name).eval()
995       if var.is_a?(Magick::ImageList)
996         func = @func_name.eval()
997         args = @args.eval() if !@args.nil?
998         return image(func,args,var,@var_name)
999       else
1000        class_name = var["class"] if var.is_a?(Hash)
1001        class_hash = Scope_handler.get_class(class_name)
1002        if !class_name.nil?
1003          Scope_handler.insert_variable_scope(var["vars"])
1004          return_value = Func_call.new(@func_name,@args,class_name).eval()
1005          Scope_handler.pop_variable_scope()
1006        else
1007          raise("<NO_INSTANCE_ERROR>: Variable " +
1008          "#{@var_name.eval().inspect} is not a class instance.")
1009        end
1010      end
1011      return_value
1012    end
1013  end
1014
1015  class Image
1016    def initialize(image)
1017      @image = image
1018    end
1019    def eval()
1020      @image
1021    end
1022  end
1023
```

```ruby
# -*- coding: utf-8 -*-
############################
# File:    scopehandler.rb  #
# Authors:                  #
# Mattias Roback (matro414) #
# Magnus Suther (magsu191)  #
############################

class Scope_handler
  # A helping hand to handle scopes. All methods are static, so we can
  # handle scopes in all files with ease, without messing with imports.

  # @@variables is an array containing hashes. The last hash in this
  # array is the current scope, the first is the global scope.
  # The hash consists of keys(variable identifiers) and
  # their evaluated value.
  @@variables = Array.new()
  @@variables << Hash.new()

  # @@functions is a hash, where the key is the name of the function,
  # and the value is an array containing two elements. The first element
  # is an array containing arrays with the function parameters and their
  # associated value, and the second element contains the
  # function block/body.
  @@functions = Hash.new()

  # @@classes is a hash, where the key is the name of a class.
  # The value is also a hash, with two key-value pairs. The first key is
  # "vars", and stores a hash with the instance variables. The second
  # key is "funcs", which stores a hash of the member functions. The key
  # will be the name of the function, and the value will be an array
  # with the function parameters and the class body/block.
  @@classes = Hash.new()


  ########## Handle variable scopes ##########
  def Scope_handler.new_variable_scope()
    # Create a new scope, ie add a new hash to @variables
    @@variables << Hash.new()
    nil
  end

  def Scope_handler.insert_variable_scope(scope)
    # Append scope to @variables
     @@variables << scope
     nil
  end

  def Scope_handler.merge_variable_scopes(scope)
    # Merge scope into @@variables, scope should be an array
     @@variables += scope
     nil
  end

  def Scope_handler.pop_variable_scope(number_of_scopes=nil)
    # Pop, ie remove and return the last scope (or the number of scopes
    # given).
    if number_of_scopes.nil?() then
      return @@variables.pop()
    else
      return @@variables.pop(number_of_scopes) # Returns array
    end
  end

  def Scope_handler.assign_variable(id, value)
```

```ruby
    # If the variable is assigned already, reassign it.
    define_new = true
    @@variables.reverse_each do |scope|
      if scope.has_key?(id)
        scope[id] = value
        define_new = false
        break
      end
    end
    @@variables.last()[id] = value if define_new
    nil
  end

  def Scope_handler.assign_variable_in_last_scope(id, value)
    # Assign the variable in the last scope only
    @@variables.last()[id] = value
    nil
  end

  def Scope_handler.get_variable_value(id)
    # Return the value of the variable in the closest scope possible.
    variable = nil
    @@variables.reverse_each do |scope|
      variable = scope[id] if scope.has_key?(id)
    end
    raise("<NOT_DEFINED_ERROR>: Variable #{id.inspect} not " +
    "defined.") if variable.nil?
    return variable
  end

  def Scope_handler.get_number_of_variable_scopes()
    return @@variables.length()
  end

  def Scope_handler.print_variables()
    $stdout.puts @@variables.inspect
  end

  ########## Handle function scopes ##########
  def Scope_handler.function_defined(id)
    # Check if a function is defined.
    return false if @@functions[id].nil?
    return true
  end

  def Scope_handler.function_has_parameters(id)
    # Check if a parameter list is associated with a function
    return false if @@functions[id][0].nil?
    return true
  end

  def Scope_handler.get_function_parameters(id)
    # Return the function parameter list
    return @@functions[id][0]
  end

  def Scope_handler.get_function_block(id)
    # Return the function block
    return @@functions[id][1]
  end

  def Scope_handler.define_function(id, params, block)
    # Define a function
    @@functions[id] = [params, block]
    nil
```

```ruby
  end

  ########## Handle class scopes ##########
  def Scope_handler.class_function_defined(class_name, id)
    # Check if a class member function is defined
    return false if @@classes[class_name]["funcs"][id].nil?
    return true
  end

  def Scope_handler.class_function_has_parameters(class_name, id)
    # Check if a class member function has parameters
    return false if @@classes[class_name]["funcs"][id][0].nil?
    return true
  end

  def Scope_handler.class_defined(class_name)
    # Check if a class is already defined
    return false if @@classes[class_name].nil?
    return true
  end

  def Scope_handler.get_class_function_parameters(class_name, id)
    # Return the parameters of a class function
    return @@classes[class_name]["funcs"][id][0]
  end

  def Scope_handler.get_class_function_block(class_name, id)
    # Return the block of a class function
    return @@classes[class_name]["funcs"][id][1]
  end

  def Scope_handler.get_class_variables(class_name)
    # Get the class variables defined in a class
    return @@classes[class_name]["vars"]
  end

  def Scope_handler.get_class(class_name)
    # Return everything associated with a class
    return @@classes[class_name]
  end

  def Scope_handler.define_class(class_name)
    # Define a class
    @@classes[class_name] = {"vars"=>{},"funcs"=>{}}
    nil
  end

  def Scope_handler.define_class_function(class_name, func_name, params, bl
ock)
    # Define a class function
     @@classes[class_name]["funcs"][func_name] = [params, block]
    nil
  end

  def Scope_handler.assign_class_variable(class_name, var_name, value)
    # Assign a value to an instance variable
    @@classes[class_name]["vars"][var_name] = value
    nil
  end
end

```

```ruby
# -*- coding: utf-8 -*-
############################
# File:    image.rb        #
# Authors:                 #
# Mattias Roback (matro414) #
# Magnus Suther (magsu191)  #
############################


def image(func,args,object,var)
  # This function handles methods on Image objects.

  length = 0
  length = args.length if args.is_a?(Array)
  #Calls the right function depending on which is called.
  #(Functions are defined below)
  return_value = Null.new
  case func
  when "save" then save(args,object,var,length)
  when "slideshow" then slideshow(args,object,var,length)
  when "display" then display(args,object,var,length)
  when "rotate" then rotate(args,object,var,length)
  when "flip" then flip(args,object,var,length)
  when "scale" then scale(args,object,var,length)
  when "bw" then bw(args,object,var,length)
  when "addText" then add_text(args,object,var,length)
  when "watermark" then watermark(args,object,var,length)
  when "getDimensions" then return_value = get_dimensions(args,object,var,length)
  when "getFilename" then return_value = get_filename(args,object,var,length)
  when "convert" then convert(args,object,var,length)
  when "polaroid" then polaroid(args, object, var, length)
  else
    raise("<NOT_DEFINED_ERROR>: Function #{func.inspect}() not " +
    "defined for class Image.")
  end
  return return_value
end

#################### IMAGE MANIPULATION FUNCTIONS ####################

########## Save ##########
def save(args,object,var,length)
  raise("<ARGUMENT_ERROR>: Wrong number of arguments for "+
  "function #{var.eval()}.save().") if length < 0 or length > 1
  #If a folder is given
  begin
    # If a new folder is given as an argument
    if length == 1
      raise("<TYPE_ERROR>: Argument to #{var.eval().to_s}.save() must "+
      "be a string.") if !args[0].is_a?(String)
      FileUtils.mkdir_p(args[0])
      #If folder name does not end with "/" one is added.
      args[0] += "/" if !args[0].end_with?("/")
    end
    object.each do |image|
      dir = File.dirname(image.filename)
      if length == 1 #Writes image to directory if given.
        image.write(args[0].to_s+(image.filename.split("/")).last.to_s)
      else #Writes over the old image.
        image.write(image.filename)
      end
    end
  rescue (SystemCallError)
```

```
    raise("<IO_ERROR>: Unable to create directory #{dir.inspect}.")
  rescue (Magick::ImageMagickError)
    raise("<IO_ERROR>: Unable to save image to file #{args[0].inspect}.")
  rescue
    raise("<IMAGE_ERROR>: #{var.eval()}.save() failed.")
  end
end

########## Slideshow ##########
def slideshow(args,object,var,length)
  #Creates a slideshow of the images
  raise("<ARGUMENT_ERROR>: Wrong number of arguments for "+
  "function #{var.eval()}.slideshow().") if length != 1
  begin
    object.delay = args[0]
    object.animate
  rescue TypeError
    raise("<TYPE_ERROR>: Wrong argument type for function "+
    "#{var.eval()}.slideshow(), expected Integer.")
  rescue
    raise("<IMAGE_ERROR>: #{var.eval()}.slideshow() failed.")
  end
end

########## Display ##########
def display(args,object,var,length)
  # Display all images in object, one at a time in
  # the same window. Step through images with space.
  raise("<ARGUMENT_ERROR>: Wrong number of arguments for "+
  "function #{var.eval()}.display().") if length != 0
  begin
    object.display()
  rescue
    raise("<IMAGE_ERROR>: #{var.eval()}.display() failed.")
  end
end

########## Rotate ##########
def rotate(args,object,var,length)
  # Rotate each of the images in object, and
  # put the rotated images back into object.
  raise("<ARGUMENT_ERROR>: Wrong number of arguments for "+
  "function #{var.eval()}.rotate().") if length != 1
  begin
    0.upto(object.length-1) do |n|
      object[n].rotate!(args[0])
    end
  rescue TypeError
    raise("<TYPE_ERROR>: Wrong argument type for function "+
    "#{var.eval()}.rotate(), expected Integer or Float.")
  rescue
    raise("<IMAGE_ERROR>: #{var.eval()}.rotate() failed.")
  end
end

########## Flip ##########
def flip(args,object,var,length)
  # Flip each image in object
  raise("<ARGUMENT_ERROR>: Wrong number of arguments for "+
  "function #{var.eval()}.flip().") if length != 0
  begin
    0.upto(object.length-1) do |n|
      object[n].flip!()
    end
  rescue
```

```
    raise("<IMAGE_ERROR>: #{var.eval()}.flip() failed.")
  end
end

########## Scale ##########
def scale(args,object,var,length)
  # Scale each image in object.
  # If one argument is given, scale by percentage size change.
  # If two arguments, scale by the desired width and height.

  raise("<ARGUMENT_ERROR>: Wrong number of arguments for "+
  "function #{var.eval()}.scale().") if length < 1 or length > 2
  begin
    if length == 1
      raise("<ARGUMENT_ERROR>: Argument(s) to scale() must be a "+
      "positive integer or float.") if args[0] < 0
      0.upto(object.length-1) do |n|
        object[n].scale!(args[0])
      end
    else
      raise("<ARGUMENT_ERROR>: Argument(s) to scale() must be a "+
      "positive integer or float.") if args[0] < 0 or args[1] < 0
      0.upto(object.length-1) do |n|
        object[n].scale!(args[0],args[1])
      end
    end
  rescue TypeError
    raise("<TYPE_ERROR>: Wrong argument type for "+
    "function #{var.eval()}.scale(), expected Integer or Float.")
  rescue
    raise("<IMAGE_ERROR>: #{var.eval()}.scale() failed.")
  end
end

########## Black & White ##########
def bw(args,object,var,length)
  # Convert all images in object to grayscale
  raise("<ARGUMENT_ERROR>: Wrong number of arguments for
  function #{var.eval()}.bw().") if length != 0
  begin
    0.upto(object.length-1) do |n|
      object[n] = object[n].quantize(256, Magick::GRAYColorspace)
    end
  rescue
    raise("<IMAGE_ERROR>: #{var.eval()}.bw() failed.")
  end
end

########## Add Text ##########
def add_text(args,object,var,length)
  # Add a text on top of every image in object.
  # Default parameters is:
  # text,x=0,y=0,pointsize=32,fill="white",font="Times",font_weight=400,
  #font_style=Magick::NormalStyle, stroke="none",stroke_width=0

  # When skipping parameters, null has to be provided as argument in
  # place for the parameters to be skipped.

  raise("<ARGUMENT_ERROR>: Wrong number of arguments for
  function #{var.eval()}.addText()") if length < 1 or length > 11

  # Set all attributes to their default values
  text = args[0]
  x = 0
  y = 0
```

```
  pointsize = 32
  fill = "white"
  font = "Times"
  font_weight = 400
  font_style = "normal"
  stroke = "transparent"
  stroke_width = 0

  # Set attributes to values sent as parameters, if given
  x = args[1] if !args[1].nil? and !args[1].is_a?(Null)
  y = args[2] if !args[2].nil? and !args[2].is_a?(Null)
  pointsize = args[3] if !args[3].nil? and !args[3].is_a?(Null)
  fill = args[4] if !args[4].nil? and !args[4].is_a?(Null)
  font = args[5] if !args[5].nil? and !args[5].is_a?(Null)
  font_weight = args[6] if !args[6].nil? and !args[6].is_a?(Null)
  font_style = args[7] if !args[7].nil? and !args[7].is_a?(Null)
  stroke = args[8] if !args[8].nil? and !args[8].is_a?(Null)
  stroke_width = args[9] if !args[9].nil? and !args[9].is_a?(Null)

  if font_style != "normal" and font_style != "italic"
    raise("<TYPE_ERROR>: Font style must be normal or italic.")
  end
  begin
    # Put text on each image in object.
    object.each do |image|
      gc = Magick::Draw.new
      gc.annotate(image, 0, 0, x, y, text) do
        self.gravity = Magick::CenterGravity
        self.pointsize = pointsize
        self.font_family = font
        self.font_style = Magick::NormalStyle
        self.font_style = Magick::ItalicStyle if font_style == "italic"
        self.fill = fill
        self.stroke = stroke
        self.stroke_width = stroke_width
        self.font_weight = font_weight
      end
    end
  rescue (TypeError)
    raise("<TYPE_ERROR>: #{var.eval()}.addText() should have parameters "+
    "(text(string),x(int),y(int),pointsize(int),fill(string),font(string),"
+
    "font_weight(int),font_style(string),stroke_color(string),"+
    "stroke_width(float)) or null for default.")
  rescue
    raise("<IMAGE_ERROR>: #{var.eval()}.addText() failed.")
  end
end

########## Watermark ##########
def watermark(args,object,var,length)
  # Put a watermark on each image in object.
  # A watermark always has white as font color.
  # Default parameters is:
  # x=0, y=0, pointsize=32, font='Times', font_weight=400,
  # font_style='normal', rotation=-90

  raise("<ARGUMENT_ERROR>: Wrong number of arguments for "+
  "function #{var.eval()}.watermark().") if length < 1 or length > 8

  text = args[0]
  x = 0
  y = 0
  pointsize = 32
  font = "Times"
```

```ruby
  font_style = "normal"
  font_weight = 400
  rotation = -90

  # Set attributes to values sent as parameters, if given
  x = args[1] if !args[1].nil? and !args[1].is_a?(Null)
  y = args[2] if !args[2].nil? and !args[2].is_a?(Null)
  pointsize = args[3] if !args[3].nil? and !args[3].is_a?(Null)
  font = args[4] if !args[4].nil? and !args[4].is_a?(Null)
  font_weight = args[5] if !args[5].nil? and !args[5].is_a?(Null)
  font_style = args[6] if !args[6].nil? and !args[6].is_a?(Null)
  rotation = args[7] if !args[7].nil? and !args[7].is_a?(Null)

  raise("<TYPE_ERROR>: Font style must be normal or
  italic.") if font_style != "normal" and font_style != "italic"

  begin
    object.each_with_index do |image,index|
      # For each image, create a new image to which the text is
      # annotated, and then use this image as watermark on
      # the images in object.
      mark = Magick::Image.new(image.columns,image.rows) do
        self.background_color = 'none'
      end
      gc = Magick::Draw.new
      mark.rotate!(-rotation)
      gc.annotate(mark, 0, 0, 0, 0, text) do
        self.gravity = Magick::CenterGravity
        self.pointsize = pointsize
        self.font_family = font
        self.font_weight = font_weight
        self.font_style = Magick::NormalStyle
        self.font_style = Magick::ItalicStyle if font_style == "italic"
        self.fill = "white"
        self.stroke = "none"
      end
      mark.rotate!(rotation)
      object[index] = image.watermark(mark,0.15,0,Magick::CenterGravity,x,y
)
    end
  rescue (TypeError)
    raise("<TYPE_ERROR>: #{var.eval()}.addText() should have parameters "+
    "(text(string),x(int),y(int),pointsize(int),fill(string),font(string),"
+
    "font_weight(int),font_style(string),stroke_color(string),"+
    "stroke_width(float)) or null for default.")
  rescue
    raise("<IMAGE_ERROR>: #{var.eval()}.watermark() failed.")
  end
end

########## Get Dimensions ##########
def get_dimensions(args,object,var,length)
  # Get an array with [x, y] for each image in object.
  dimensions = []
  object.each do |image|
    dimensions << [image.columns,image.rows]
  end
  return dimensions
end

########## Get Filename ##########
def get_filename(args,object,var,length)
  # Get an array with [filename] for each image in object.
  filenames = []
```

```ruby
  object.each do |image|
    filenames << image.filename
  end
  return filenames
end

########## Convert ##########
def convert(args,object,var,length)
    # Convert each image in object to the format given as
    # argument.
    # The images will be saved with this new extension, but the converted
    # images will not be loaded back into object.
  if length != 1
    raise("<ARGUMENT_ERROR>: Wrong number of arguments "+
    "for #{var.eval()}.convert().")
  else
    raise("<TYPE_ERROR>: Wrong argument type for function "+
    "#{var.eval()}.convert(), expected String.") if !args[0].is_a?(String)
    format = args[0].downcase().gsub(".","")
    if !["jpg","jpeg","png","gif","bmp"].include?(format)
      raise("<TYPE_ERROR>: Can only convert to jpg, jpeg, png, gif or bmp."
)
    end
    object.each_with_index do |image,index|
      # Substitute the file extension and let RMagick do the
      # convert for each image. Note that this saves each image to the
      # same folder it was read from, but with a new extension.
      ext_length = File.extname(object[index].filename).length
      object[index].write(object[index].filename[0..-(ext_length)]+format)
    end
  end
end

########## Polaroid ##########
def polaroid(args,object,var,length)
  # Make all images in object polaroid.

  raise("<ARGUMENT_ERROR>: Wrong number of arguments "+
    "for #{var.eval()}.polaroid().") if not (length.eql?(1) or
     length.eql?(0))

  raise("<TYPE_ERROR>: Wrong argument type for function "+
    "#{var.eval()}.polaroid(), expected String") if not args.nil? and
    args[0].is_a?(Null)

  # Set default value, and get the argument if given.
  text = ""
  text = args[0] if not args.nil? and not args[0].is_a?(Null)

  begin
    object.each_with_index do |image, index|
      object[index][:caption] = text
      object[index] = image.polaroid{self.gravity = Magick::CenterGravity}
      object[index].change_geometry!("#{image.columns}x#{image.rows}") do
      |ncols, nrows, img|
        img.resize!(ncols, nrows)
      end
    end
  rescue
    raise("<IMAGE_ERROR>: #{var.eval()}.polaroid() failed.")
  end
end

```

```
# -*- coding: utf-8 -*-
############################
# File:    makefile        #
# Authors:                 #
# Mattias Roback (matro414) #
# Magnus Suther (magsu191)  #
############################

install:
  cp ImgPL /usr/local/bin/imgpl
  mkdir -p /usr/local/lib/ImgPL
  cp ImgPL.rb /usr/local/lib/ImgPL
  cp tree.rb /usr/local/lib/ImgPL
  cp rdparse.rb /usr/local/lib/ImgPL
  cp scopehandler.rb /usr/local/lib/ImgPL
  cp image.rb /usr/local/lib/ImgPL
  cp error.txt /usr/local/lib/ImgPL
  echo "Installation complete! You can now run 'imgpl' in a terminal."

uninstall:
  rm -rf /usr/local/lib/ImgPL
  rm -f /usr/local/bin/imgpl
  echo "\nUninstall complete!"

```