Opponering

Det skapade datorspråket

The whole “motto” of the language is - “easy to learn” thus the name ESL. One problem with this is that it is easy for someone with experience to create something very easy for themselves, but maybe not so much for complete beginners. Although the language itself is indeed constructed in a way that promotes this motto - “easy to learn”. For instance they use dynamic typing which allows the user to assign any value they want to a variable without explicitly using data types. One thing I did notice though, I assume that a language like this would be more of a starting ground to learn more advanced languages? I am therefore a bit confused why they choose to use “otherwise” instead of “else” on the if statements? Other than this, the whole language holds this standard, endif, endwhile and enddef so it is easy to assume how to write different constructions. Each construction follows the same order overall and tries to avoid any extra fluff. Printing is how you would assume it to be as a beginner probably, through typing write followed by a message. The mathematics follow the general standard in regards to the priority of the operators and parentheses. Personally I can very easily learn the language through their documentation, but that's a very different matter to a complete beginner.

A few other things I would like to mention about the language, the way they have named their methods doesn’t follow any kind of structure it seems, they are mixing camel case with snake case. The names on some of their methods are also quite confusing and when I first saw some of them, I thought they were some sort of inbuilt ruby function. For example, “receiveLevel” which with closer inspection is supposed to return the current scope, so why not just write “getCurrentScope” ? Receiving something is different from getting something and it causes confusion. The same goes for “incre”, while I do understand what this means, it seems unnecessary.

Språkbeskrivningen med grammatik

We find the BNF or the grammatics of the language to be correctly written and all the constructions to have the right names. Though some of them might be difficult to understand for the target group (beginners), like “multiple_strings”, or a “statement”. Maybe something like “sentence” could be used instead since string is a very programming-specific word. There are small differences between the BNF and the actual code. For example, WHILE_LOOP in the BNF is “whileIteration” in the code, other than that everything looks the same. There are also no extra constructions, everything that is in the BNF has its corresponding functionality in the language itself. Yes everything looks good for the most part, they explain in a lot of detail with both the reflection itself and the introduction. Although, they explain in their “kod standard” that after every if-statement you should write then. I am not 100% certain on the definitions of an if-statement and what they encompass, but when I read the first line which states

that you should write an endif after an “if-sats” and then write then after an “if-statement”, I can’t help myself but think that they mean the same thing.

Implementering av verktyg eller användning av verktyg för språket

From our perspective it seems that they have implemented the use of RDparser very well since everything is working as it should. Writing your own parser is quite hard so the choice to use the already defined parser for the course is a clever move and seems like the choice most of the participants in the course have made. There are always limitations in tools of course, but for such a simple language I don’t think it is a problem. They seem to use the tools they have very well, they make use of different methods and algorithms that work well with each other.

Metoder och algoritmer

Synpunkter på valda metoder och algoritmer, verkar de vara bra val? Kan det bli mycket ineffektivt? Finns det alternativ, som hade blivit bättre?

We think that everything looks good in the file, the code looks clean and we can’t find anything that would be particularly ineffective. Although everything is specific which means that there is usually more code. If we compare our code for example we have all the operators in several rules like their term, but we understand the need to have it like they do. If a problem occurs they will be able to find the issue directly seeing as they are quite specific as for us though it's been nested a few times so it might prove to be slightly more challenging.

In the language itself they are very specific with everything, which has both its pros and cons and in comparison to our language which is very generalised it is quite different. I therefore saw it as a great opportunity to compare the two. The ironic thing is, is that they are also quite general in what they do and seem to mix the two quite nicely. They use expr to call upon different operations and expr uses terms to get different operators. While our code is more generalised, it is also longer which is an attribute to their design and effectiveness.

They also lack base classes such as float and int, and sort them all in “Constant”, which I thought was a must, so it is cool that they got it to work without them, it makes for a very clean and tidy look that is easy to follow and relatively easy to understand.

Koden för implementeringen i Ruby.

We would definitely say that the language is well dispositioned. You can clearly see the divide between the different parts of the program. They keep it simple and exclude all of

the unnecessary items. Sometimes it might be a bit tricky to understand the exact definition and functionality of a class based on class names and function names, but generally it's good. We like the fact that it's very specific and most parts of the language have their own "part". When you want to look at specific functionality you can just go there in the classes.rb file. Though some of it is weirdly named, like "constant", which is in majority named "atom" in ETL.rb, so the dividing of functionality there could have been done differently.

Har Ruby använts på ett bra sätt? Har ni alternativa förslag på hur man kan använda andra Ruby-konstruktioner.
They haven't used too many built in ruby-constructions, but they don't need to either.

Code complete-boken eller andra kodstandarder

They describe in their documentation that the language doesn't use any specific indentation in their language which is true, and therefore it differs from any of the previous languages we've seen. Therefore it seems like they've created a little bit of their own code standard unlike any we've seen. They've used general typing standards like camel case and snake case. A unified typing standard throughout the code is recommended, they've used a mix of both along with one other standard they have made up.

Testkörning av språket

Var det lätt att komma igång med systemet?
They clearly state how to run the program. Although for a beginner it would prove to be more difficult, but assuming they had all of the same tools at hand then it wouldn't be a problem.

We found that after reading the user instructions once, we were already sufficient in the language.

They have a separate test file where you can write all of the code which is very nice. In this way you don't have to write separate tests for every case or run in interactively.

The only issue we found ist that their comments for multiple lines doesn't always work, other than that it works fine.