# Deliverables and Submission

There are two significant deliverables that must be handed in: a document that provides a description of your language, and a .ZIP file (*not* a .RAR file or a .7Z file!!!) containing your language implementation. Both are described in more detail below.

# The Task

This assignment is about designing a computer language and building an implementation of it. The implementation may be an interpreter or compiler (or a “transpiler”) – the choice is yours.

There are three deliverables:

* The language design (aka “two sides of A4”); **INCOMPLETE**
* The language presentation (in class, Week 8); **COMPLETE**
* The language implementation (a .ZIP file, submitted the usual way.) **INCOMPLETE**

# The Language Design

The first thing you need to do is design your language. In doing so you must answer at least the following questions:

1. What is the model of computation? Is it a logic language, an imperative language (i.e., conceptually based on a Turing Machine), or a functional language (conceptually based on lambda calculus)[[1]](#footnote-1) or something else?
2. What language paradigm(s) does it implement? Is it object-oriented? Is it procedural? Is it imperative or declarative? How does the type system work?
3. What are the rules for the syntax? What are the rules for constants, identifiers, control structures and so on?
4. What are the semantics? This is a big one that involves deciding at least:
5. How variables work (are they mutable?);
6. How “type” is defined, and how types are defined;
7. The rules for handling and evaluating values, both literal and those assigned to variables presumably by representing them as expressions;
8. How code can be reused in the form of subroutines or predicates or functions;
9. What operators there are and whether they are syntactic sugar for something else, such as for method calls in C++ or for predicates in Prolog;
10. How identifiers are resolved and the scoping rules; and
11. Whether there are pointers or references or something similar, or not. If not, perhaps your language implements “value semantics”?
12. Is there any interesting additional functionality that doesn’t fall into the usual categories?

For example, C/C++ and C# have pre-processor directives; Java has annotations.

1. Does your language expect or target a certain environment, such as compiling to JavaScript that will run only in a web browser?
2. What is it intended to do better – or be better for – than any other language?

You don’t need an extensive (or even much of a) standard library, and it would be an unreasonable burden to expect you to provide one. However, you should at least be able to print things to the console. The exact nature of these “things” is a matter for you to design. Will you go the way of iostreams in C++, something like printf() in C (and C++), or a different direction entirely like Haskell’s monads?

You should be able to describe your language within two sides of A4 at most – I am not looking for a massive language specification behemoth like that for Java or C++!

For specific areas, it is fine to refer to another language (e.g., “identifiers follow the rules used in Java”) but you’re courting disappointment if your design is something like “the language is Java 1.2”.

Although trivial, it will be helpful to give your language a name. It’s easier to talk about things that have names. **EasyPython**

# The Implementation

Your implementation should be an interpreter or a compiler for your language. In either case the input will be a text file containing source code (written in your language). The output of an interpreter will be the execution of the program; a compiler will produce code suitable for execution on a real or virtual machine. A transpiler could be either.

Your language implementation does not need to implement everything you described or demonstrated in your presentation (above), but it ideally should demonstrate at least some connection.

You are encouraged to use javacc, ANTLR v4, or flex/bison to help you build your implementation; though this is not a hard requirement, use of a parser generator removes quite a bit of drudgery involved in tokenising and parsing source, leaving you with a syntax tree to work with.

Your implementation must run on either 64-bit Windows 10 or Linux (Ubuntu) [[2]](#footnote-2) or both. Your implementation must be a self-standing executable. In other words, the marking tutor shouldn’t have to install anything to run it. If the marking tutor is required to install some missing component to run your examples, or if your language otherwise can’t be used to run the examples in the *examples* directory, **the highest grade this assignment**

# Assessment

Assessment will be broken into two parts.

10% will come from your presentation and will be assigned as follows:

* 10 out of 10: Your presentation was convincing; you made everyone want to use your language.
* 7 out of 10: Your presentation was OK. It is obvious that you know what you are doing but some doubts remain.
* 4 out of 10: You gave a presentation.
* 0 out of 10: You didn’t give a presentation.

90% of your grade shall be assigned as follows. Note that these are percentages of the remaining 90%, not the final grade.

|  |  |  |
| --- | --- | --- |
| % mark | Mark descriptors | Class |
| 70-100% | **Excellent** 90–100% Your language implements a novel paradigm with elegance or blends two or more paradigms cleanly. Furthermore, your implementation allows programs to be good citizens of their environment, interoperating with those written in other languages, perhaps by embedding your interpreter, linking against object code produced by your compiler, or by using operating system IPC facilities or the network. Your two sides of A4 clearly describes how this all works, and you have a set of example programs. Programs written in your language, when run, do no more and no less than their source code suggests. 80–89% Your language either implements a novel paradigm with elegance or blends two paradigms cleanly; or your language allows useful programs to be written using a programming model that is fairly standard, but allows these programs to be good citizens of their environment, interoperating with those written in other languages, perhaps by embedding your interpreter, linking against object code produced by your compiler, or by using operating system inter-process communication facilities or the network. Your two sides of A4 clearly describes how this all works, and you have a set of example programs. Programs written in your language, when run, do no more and no less than their source code suggests. 70–79% Your language allows useful programs to be written, perhaps using an approach like that employed by existing languages. Your implementation supports all features of your language and the programs that result do exactly what their source code suggests, but these programs are self-contained and do not interact with the outside world beyond printing things. Your language supports a rich collection of composite data structures, allowing programs to be concise and expressive. Any restrictions inherent in the language, such as maximum identifier length, are very reasonable. Two sides of A4 describe your language thoroughly. | First |
| 60-69% | **Very good**  Your language allows useful programs to be written, perhaps using a conventional approach. Your implementation supports all features of your language and the programs that result do exactly what their source code suggests, but these programs are self-contained and do not interact with the outside world beyond printing things. Your language supports one composite data structure (such as arrays, tuples, dictionaries, etc.). Any restrictions inherent in the language, such as maximum identifier length, are very reasonable. Two sides of A4 describe your language well. | Second Division 1 |
| 50-59% | **Good**  Your language is effective and useful but may place some severe restrictions on the programmer, so that facilities that are necessary to effective programming in the given paradigm – such as not supporting recursion in a conventional imperative language, or only allowing use of built-in variables. Nevertheless, your implementation supports most the features of the language and allows for the creation of meaningful, though perhaps not particularly useful programs. The features that are missing, while useful, are not absolutely essential for getting work done, but getting work done might be laborious. The two sides of A4 that you provide may be confusing or unclear in places. | Second Division 2 |
| 40-49% | **Satisfactory**  Your language is useful but shows a lack of originality, perhaps bearing a strong resemblance to the tutorial examples, but enough semantic differentiation to be recognisably a different language – if only slightly – and not just one or two keywords with a new spelling. Some fundamental language features are not supported by your implementation, making it difficult to write effective programs. What is implemented behaves correctly. The two sides of A4 describe most of the language, but some aspects are unclear or obscure. | Third |

1. “Functional” doesn’t just mean it has something called “functions”. For example, C and JavaScript define “functions” using similar syntax, but C is not a functional programming language, and whilst JavaScript has some functional facilities, it’s properly considered a multi-paradigm language. [↑](#footnote-ref-1)
2. I am an advocate of cross-platform development, but as I don’t have OS X installed in a virtual machine (I run it natively on Apple hardware) and I only run assignments (and most executables of un-proven origin) on virtual machines, I cannot accept OS X as a target. [↑](#footnote-ref-2)