

Deliverable 1: Final Year Dissertation

An Online Tool to Choose a Programming Language

Ronan Smith, rs6, H00189534

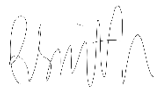
BSc Computer Science

Supervised by Fairouz Kamarreddine



Declaration

I, Ronan Smith, confirm that this work submitted for assessment is my own and has been expressed in my own words. At any point in this document where work of another author has been used in any form (e.g. ideas, equations, research data, programming code) this has been properly acknowledged with references.

Signed: 

Date: 27/11/2017

Abstract

This project looks at the design and creation of a website which can be used by programmers of any level to help them choose a programming language for their particular problem. This project discusses some useful topics, for example what is a good first programming language to learn? What languages are out there? And what actually makes a language 'good' in the first place?

Contents

1.0 Aims & Objectives	1
1.1 Carry out the relevant research.....	1
1.2 Deeply explore similar products that already exist	1
1.3 Build a solution.....	2
1.4 Evaluate the new solution and compare with existing products.....	2
2.0 Literature Review.....	3
2.1 What makes a good programming language?	3
2.1.1 Writability and Readability.....	3
2.1.2 Simplicity.....	5
2.1.3 Modularity	5
2.1.4 Definiteness.....	6
2.1.5 Efficiency	6
2.2 A good first programming language to learn	6
2.2.1 The Traditional Approach.....	7
2.2.2 Mini Languages	7
2.2.3 Next, make things more interesting	8
2.2.4 Programming at university	8
2.2.5 There's no right answer	9
2.3 Most popular programming languages.....	9
2.4 Programming Paradigms.....	10
2.4.1 Imperative Paradigm.....	10
2.4.2 Logical Paradigm.....	11
2.4.3 Object-Oriented Paradigm	13
2.4.4 Functional Paradigm.....	14
2.5 High vs low-level programming	16
2.5.1 Assembly Code (Low-Level Programming).....	16
2.5.2 C (System-Level Programming).....	17
2.5.3 Java (High-Level Programming)	17
3.0 Technical Literature Review.....	19
3.1 Codecademy	19
3.2 Stack Overflow	20
3.3 Best Programming Language for Me.....	21
4.0 Survey to Find the most Popular Languages for Programmers.....	23
5.0 Requirements Analysis	24
5.1 Use Case Diagram	24

5.2 Requirements	25
6.0 Project Management & Product Design	27
6.1 Gantt Chart	27
6.2 Risk Analysis	28
6.3 Professional, Legal, Ethical and Social Discussion.....	30
6.4 Early Product Design Ideas & Methodology.....	31
7.0 Evaluation Strategy	32
8.0 References	33
9.0 Appendices.....	35

1.0 Aims & Objectives

The ultimate aim of this project is to produce an online product that can be used to help programmers decide on a programming language to use based on the problem they are trying to solve. This product should be simple and easy to use, but at the same time it should be highly informative and should have a number of useful features. To achieve the aim, I define the following objectives:

1.1 Carry out the relevant research

I will be conducting research in related areas for this project, for example what makes a good programming language; Are certain languages better for specific tasks or are some languages just better than others in general? I will also be discussing what the best languages for beginners are, the different paradigms that exist and the most popular languages according to different sources. Furthermore, I will be carrying out a survey aimed at Computer Science and Information Systems students as well as programmers in general that will aim to find out what the most popular programming languages are.

1.2 Deeply explore similar products that already exist

I will be discussing and reviewing some similar products that are currently out there, more specifically online products used by computer programmers. This will include discussion on what I think are good and bad features of the products and what features I will add to my own solution based on this.

1.3 Build a solution

I will design and implement a website that extends and improves the current products that are out there. The website will do at least the following things:

- Effectively provide users with information about a wide range of programming languages.
- Ask users to answer a few questions which will ultimately lead them to a programming language relevant to their needs.
- Be simple and easy to use but also full of information about programming and programming languages.
- Be welcoming to programmers of all different levels, from beginners to experts.

1.4 Evaluate the new solution and compare with existing products

I will check that the website meets these aims and objectives through my own testing methods as well as a usability study. The study will be aimed at programmers of any level and people that are likely to use the website.

Furthermore, I will be revisiting the similar products discussed in section 3.0 (Technical Literature Review) to discuss whether or not I have successfully extended and improved them.

2.0 Literature Review

Computer programming can be a lot of fun, but also very challenging. An important aspect in programming is choosing the programming language you want to use. Depending on what the problem is you are trying to solve there may be hundreds or even thousands of possible languages to choose from. How can we decide which one suits us best and how can we be sure we are making a correct decision?

This is a problem that depends very much on the context of the situation. The relevant details may include the experience of the programmer, the type of problem being dealt with (is it a logical problem? Does it deal with functions? Does it deal with objects?) and the problem itself; a new problem with little to no documentation or a well-defined problem which has been dealt with in many different ways before?

The following sections discuss some areas that should be visited when choosing a programming language (or creating a tool to help do this for you) in much more detail.

2.1 What makes a good programming language?

This is a topic that is very much based on opinion and each programmer is likely to be biased towards certain languages based on their own previous experience and the reasons they generally use programming for. However, I think most programmers would agree on the following statement; “a programming language is good if it helps us to write programs that are easy to read, easy to understand, and easy to modify” [1].

There are a huge number of ways we could categorise what makes a good programming language, but for the purpose of this project we will stick to the following 5 [1]:

2.1.1 Writability and Readability

A program that can be easily understood by the programmer **writing** it and others checking/**reading** it is highly maintainable. This is something we like to see in a

programming language, as maintainable programs are easy to extend in the future and can save a lot of development time for future programmers using that program code. Although the readability of a program is mostly controlled by how well the programmer actually writes it, this can be helped along by a programming language that is designed to be more readable in the first place. For example, a simple program to add the numbers together from 1 to 10 and print the result looks like this in MIPS Assembly:

```

        .file    "adder.c"
        .section .rodata
.LC0:
        .string "The total is %d\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $0, -4(%rbp)
        movl    $1, -8(%rbp)
        jmp     .L2
.L3:
        movl    -8(%rbp), %eax
        addl    %eax, -4(%rbp)
        addl    $1, -8(%rbp)
.L2:
        cmpl    $10, -8(%rbp)
        jle     .L3
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        movl    $.LC0, %edi
        movl    $0, %eax
        call    printf
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
.ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
        .section .note.GNU-stack,"",@progbits

```

This is the equivalent C program:

```
#include <stdio.h>
int main(){
    int i;
    int total = 0;
    for (i = 1; i < 11; i++){
        total = total + i;
    }
    printf("The total is %d\n", total);
}
```

Of course, there are still uses for languages like MIPS Assembly in low-level programming and hardware manipulation. But for a programmer who has experience in high level programming and is working at a higher level, the block of C program code above would be much easier to read, understand and hence work with than the corresponding MIPS Assembly code, which is about 30 lines longer (depending on how you lay it out) and seems much more complicated to someone who is not an expert in Assembly.

2.1.2 Simplicity

Languages that are designed to be **simple** tend to be easier for programmers to use. “Simplicity is about reducing incidental complexity as much as possible in order to be able to focus on the complexity that is inherent to the problems we solve” [2]. In other words, having a programming language with a small number of easy to remember features allows the programmer to concentrate on what really matters – the program that is actually being written – rather than having to remember 200 or more useless extra features to remember.

2.1.3 Modularity

Modularity in programming (i.e. keeping code blocks relevant to one single task each) enhances simplicity and this may be one of the reasons that Object-Oriented Programming is so popular.

2.1.4 Definiteness

A programming language with a **definite philosophy**; meaning it should be designed for a definite and specific reason as much as possible, can be seen as a highly useful language for the purpose it was built for. For example, the Visual Basic language developed by Microsoft was designed to be easy to use for new programmers, and so is an ideal language to use in schools, however it would not be quite so useful for designing huge industrial applications. Prolog is great to use when working with logic, but might not apply to other areas quite so well. Both are designed for a specific purpose and philosophy in mind which they work well for.

2.1.5 Efficiency

Efficiency has a huge part to play in deciding how good a programming language is. Some languages, like C, allow the programmer to manually allocate and reallocate memory locations as they please, and this can be seen as much more efficient than if it is done automatically by the compiler, for example. However, some programmers would prefer not to have to deal with the overhead of doing things themselves quite so much. We look to find a balance where the program is making a good use of system resources and performing at fast speeds but the programmer is also able to create the program code to an efficient standard.

2.2 A good first programming language to learn

If you plan on working as a programmer, the first language you learn can be a very important one. For young people, learning through a language that they enjoy may help to encourage them into going for a career in programming, whereas learning through a language that is difficult to work with may put them off. However, a first programming language can't just be chosen because it is fun to work with (at least not in education,

anyway) but must also teach you the fundamentals that you need to know and that you can apply to all the other programming languages you use in the future. If your first language gives you skills you can take into another language, then you are off to a good start.

2.2.1 The Traditional Approach

The traditional approach to learning programming would be learning the basics of some large, general purpose language such as C, Pascal or LISP and being given a set of tasks that work with number or symbol processing. This has its problems, however [4]:

1. General purpose languages are too big and have too many features. These can be difficult for a new programmer to remember and may make them feel inadequate.
2. There is not much of a visual side to general purpose languages, and so it is not so easy for the programmer to see and understand what the code they write is actually doing.
3. The tasks that students tend to work on don't really appear relevant to their everyday life and so are not as interesting and appealing to them.

2.2.2 Mini Languages

This is where 'Mini Languages' come in. A mini language is designed to have a small number of features and a small syntax with simple semantics [4]. At a simple level, this keeps things easy to remember for the programmer. Furthermore, mini-languages tend to be visually appealing, often created in block format and often producing results that are interesting to look at.

For me, one of my earliest memories of programming was using Scratch to write simple block programs to move a cat around the screen. MIT App Inventor [5] is another example of a block programming system, and this can be used to create very simple to very complex Android apps without seeing a single piece of source code. These are great ways to learn

because they provide an interface where programmers can see the changes they make coming into action.

2.2.3 Next, make things more interesting

Once source code starts being revealed to novice programmers' things can quickly become complicated. It is important to gradually introduce them to what source code looks like and to try to build a bridge between what they have learned in block programming, for example, and what it looks like under the hood. Visual Basic is a useful language at this stage as it allows the programmer to create an interface containing buttons, labels, text/input boxes and much more, simply through dragging and dropping items. Changing the view in the Visual Basic or Visual Studio IDE (Integrated Development Environment) can then allow them to start writing program code for each of the interface items in turn.

2.2.4 Programming at university

At the beginning of university things can become interesting for what programming languages should be taught as a programming introduction. The problem at this stage is that everyone has come from different backgrounds, countries and education systems, and so not everyone is at the same level of expertise. It could be easy for the university to look at industry and to decide to start teaching languages that are used in software development companies, however, this isn't necessarily the best idea. As Doug Grant said in 1992, "despite the apparent industry demand for students familiar with C and C++, these need not be the languages of choice for early computer science courses" [6]. This is still relevant today. It is not necessarily important to know a specific language that is used in industry, but it is important to learn the skills and principles needed to program in the real world.

2.2.5 There's no right answer

There is no set way of doing things, and even the SQA (Scottish Qualifications Authority) have not set a specific language to work with in the new 'Curriculum for Excellence', feeling that it should be up to the teacher to decide based on their own experience and knowledge. They have, however, defined their own pseudocode reference language called SQARL (SQA Reference Language – formerly called HAGGIS) to set the general idea of structure in programming. At university level in the UK, Java dominates introductory programming courses, although Python is easier to use [7] and potentially a better option. This may be because Java has been around for longer, or due to university lecturers having more experience with Java, or because there is more documentation out there for it. At the end of the day the specific language taught is not important, but what can be learnt from it.

2.3 Most popular programming languages

What are the most popular programming languages? Well, this isn't an easy question to answer straight off. Programming languages are tools, and one tool may be extremely useful in one situation, but not so useful in another. The popularity of a programming language depends largely on the domain (the type of problem) that it is being used for and possibly the paradigm it will be working in (see section 2.4).

In 2015, most rankings of programming languages would see the "big 5" right at the top or around about the top. These were Java, C, C++, Python and C# [14], and you would often expect languages like JavaScript and PHP in the top 10 somewhere too. However, a more recent article discussing the top languages according to GitHub [13] puts JavaScript right at the top with 2.3 million pull requests (requests to view code of that type using GitHub) in comparison with 1 million for its closest contender - Python - and shows that more recently languages like C and C# have dropped in the rankings with languages such as Ruby, R (a language mostly used for handling and visualising big data) and Swift (Apple's relatively new programming language that can be used for creating mobile apps) climbing up the way.

The most popular languages are always changing as time goes on, with more modern solutions constantly being worked on that can provide improvements on older languages. The language that you choose to use is up to you, and should be chosen based on what you want to do with it, however, it is always useful to know what everyone else is using.

2.4 Programming Paradigms

A programming paradigm is a style, or “way” of programming [8]. Some languages may be better suited to solving different problems depending on what paradigm or paradigms they fit in to, and some problems may be suited more to certain programming paradigms. For the purpose of this project, I will discuss 4 different paradigms; imperative, logical, object-oriented and functional.

2.4.1 Imperative Paradigm

In an imperative programming language, programs are written as a structured series of steps, which are executed in order, line by line, top to bottom, left to right. At the end of the sequence of steps a problem should have been solved or at least part of a problem and any change to the order of execution in an imperative program will change the results. This is one of the simplest paradigms and often one of the first taught in schools.

Some advantages of the imperative programming paradigm are as follows [9] [10]:

- **Efficient;** This is certainly true with fairly small programs as there is no need for function calls and a huge number of pointers around the program. The next instruction is always on the next line.
- **Close to machine code;** Low level programming like Assembly can be seen as imperative. Assembly languages are closely linked to machine code and the imperative paradigm is closely related to assembly code. This makes it easier to work with the imperative paradigm at low levels.

- **Iterative Process;** The process is made up of step by step instructions which should be fairly easy for the programmer to follow, especially in small programs.
- **Familiar;** The style of writing down instructions line by line is similar to the way the programmer might write down their tasks for the day in real-life and so it makes sense to them.

And some disadvantages [9] [10]:

- **Debugging can be harder than other paradigms;** This is due to the fact that most variables in an imperative program are likely to be global, and so they can be changed a lot over the course of execution, making them harder to keep track of.
- **Abstraction is limited;** Due to this, it is much more difficult hide irrelevant data from view and thus ends up with a lot of mess on the screen.
- **Order is crucial;** This can be a problem as it means making changes to the lines of program code have to be done with much more thought.
- **Complex processes lead to “spaghetti code”;** This is a way of describing code that gets out of control. Complex processes in a language following the imperative paradigm may not be as easy to break down into modules and therefore can get very complicated very quickly.

Some examples of languages that fit into the imperative programming paradigm are C, Fortran, Pascal and (non-object-oriented) Python.

2.4.2 Logical Paradigm

In the logical programming paradigm, programs are written in a more declarative way. This means instead of the traditional approach of writing line by line instructions in how to solve certain problems, a set of facts and rules is defined that can be applied to certain types of problem to try and solve them. The program is not told exactly how to solve certain problems, but instead applies the rules it has been provided with to find the solutions itself.

There are two main advantages to using the logical paradigm [9]:

- **The system solves the problem, not the programmer;** This keeps the actual programming of the system to a minimum and keeps things fairly simple.
- **Proving the validity of a given program is simple;** A program is made up of a set of facts and possibly a set of rules to connect facts. To run the program a series of queries can be written which will simply return the answer 'true', 'false' or the value of a variable.

Some disadvantages of the logical programming paradigm are as follows:

- **Programming in this paradigm is substantially different from most other paradigms** [11]; If the programmer has not worked with a logical language before, then it can be difficult for them to get into the right way of thinking to use this paradigm. Using a logical programming language requires a certain type of thinking which may not suit some programmers well.
- **The logical paradigm is rather deficient** [9]; It does not have a lot of the built-in characteristics that you would expect to see in, for example, the object-oriented paradigm.

The logical paradigm is great for dealing with logical formulae and is useful for creating tools such as expert systems; where a knowledge base is used to store facts and rules, and queries (structured questions based on the data) can be sent to the knowledge base through a means of communication called an inference engine. However, the logical programming paradigm does not fit well to general programming problems due to it being so different from other paradigms. Prolog is one of the most well-known logical programming languages but another example is Mercury, which can also be considered as a functional programming language (see section 2.4.4). It is important to point out that many languages actually cover a number of different paradigms rather than just the one.

2.4.3 Object-Oriented Paradigm

In the object-oriented paradigm, programs are defined through an abstraction of real-life objects. Everything that can be defined in some way to be its own entity is implemented as a 'class' with its own attributes and behaviours in an object-oriented program. Objects are kept as independent as possible, but must also interact with each other to produce interesting results.

Some advantages or useful features of the object-oriented paradigm are as follows [9] [11]:

- **Inheritance;** The capability to allow classes to inherit from parent classes is a useful one. It gives a subclass (a class inheriting from a parent or superclass) the ability to inherit all (through extension), or some (through inheritance) of, the attributes of the superclass, allowing better control of structure of the class and less code repetition.
- **High amount of cohesion is possible;** The concept of objects allows programs to be written in a very cohesive way (i.e. tasks are split up into code fragments that are created for only that task). Highly cohesive programs can also be described as highly modular, which was a feature I said was important in section 2.1 'What makes a good programming language?'.
- **Encapsulation and Abstraction;** These are two features of object oriented programming that keep the data that can be seen and accessed at any point in the program relevant and secure. This is useful as it stops parts of the program interfering with others; for example, one class changing another classes variable values.

The object-oriented (or OO) paradigm does have some disadvantages however, for example [12]:

- **Steep learning curve;** For people who are new to programming, or who haven't dealt in an object-oriented way of working before, the thought process involved in

object-oriented programming can be difficult to get their head around and may take a while to understand fully.

- **Suitability to general problems;** The object-oriented paradigm is only suitable for use on problems that can be broken down into distinct objects. Some problems just don't work with this paradigm and although they could probably be written in OO, they would not be very efficient.
- **Slower programs;** In general, an object-oriented program will be much larger than its imperative counterpart. This is due to the fact that everything is broken up into classes. The size of OO programs can cause them to be much slower as well as the fact that lots of function calls are necessary in comparison with an imperative program.

Object-Oriented programming languages such as Java and C++ are among the most popular in the world, numbers 3 and 6 respectively according to Github, sometimes referred to as the "Facebook for programmers" [13]. They are extremely useful and can be used to produce some outstanding pieces of software, but it should be remembered that they don't apply well to all types of problem.

2.4.4 Functional Paradigm

In the functional programming paradigm, "computation proceeds by rewriting functions, not by changing state" [11]. In other words, the functional paradigm has been built around a notion of not having any memory. There are no global variables for example (variables that can change throughout the scope of the program) and once the environment is decided, it cannot change unless it is overwritten. This is a very different approach to most other paradigms, where operations like assigning a value to a variable; an impossible situation in functional programming, are completely normal and expected.

This approach has a number of advantages, as described below [9]:

- **A High Level of Abstraction;** This means that irrelevant data is hidden from the programmer. At any point in programming with a functional language the irrelevant data that they don't need to know about is there but hidden from view. This is an advantage because it reduces the possibility of the programmer committing a number of different types of errors.
- **Lack of dependence on assignment operators;** With a lack of dependence on assignment operators, the order of execution in a program is no longer important. This is extremely useful as it means functional languages are ideal for use in parallel environments, where the order instructions are executed in tends not to be fixed. Parallel environments are becoming more and more common as well, with the rise of multi-core computers.
- **Referential Transparency;** It is easy to translate a program written in the functional paradigm back to mathematical functions and this makes it easy to carry out mathematical proofs and analysis in comparison to programs written in another paradigm, as well as making it easier to test for correctness against the original functions.

Some disadvantages of the functional paradigm are as follows [9]:

- **Efficiency;** Certain data structures such as arrays are not as easy to implement in functional programming languages and may have to be created with much less efficient code.
- **Lack of support for variables;** Due to the fact that global variables and assignment can't be used in functional languages, it can be difficult to solve a problem that contains many different variables. The programmer should go with a different paradigm if this is the case.

Functional languages, like SML New-Jersey are ideal for use in problems that contain a lot of functions. They work well with mathematical problems, due to how easy it is to translate

between mathematical functions and the functional paradigm, and they give you a different approach than other paradigms by not having global variables or state. The functional paradigm can be inefficient however, and it should not be used to solve problems that contain a large number of variables.

2.5 High vs low-level programming

Computers store data in a base-2 numeral system called 'binary', where every possible value is stored using a combination of only two distinct values; 0 and 1. Binary representation is very effective for computers for a number of reasons, for example consider the following 2; First, it is simple to store. The values for 1 and 0 can be stored by transistor switches which can be made very small and are very cheap to mass produce. Second, the binary rules of arithmetic are simple (there are only four); $0+0=0$, $0+1=1$, $1+0=1$ and $1+1=10$ and this means they are easier to store and represent in hardware. The problem with binary representation comes only when some human needs to understand it. Binary representation, or machine code as it is also known is very difficult – almost impossible – for a human to read and understand right away and this is why programming languages are necessary. From low-level to high-level, programming languages provide more and more abstraction over machine code, becoming easier for a programmer to read and work with as they get higher. Here, I will discuss the different levels of programming with reference to some languages that are used at that level.

2.5.1 Assembly Code (Low-Level Programming)

Assembly has been mentioned before in section 2.1.1, where I highlighted that for many programmers, it is very difficult to read and understand at first glance when compared with a higher-level programming language like C, but it is much closer to machine code and therefore can be used for much more low-level tasks such as explicit memory allocation or direct hardware manipulation. Assembly languages are specific to the hardware they are written on. This means they cannot be freely moved around different computers without

thinking about the underlying hardware, in other words they are not portable. Assembly languages like ARM and MIPS are useful for tasks like building operating systems or compilers, reverse engineering (trying to understand someone else's compiled code) or building device drivers [15], as all these tasks involve communicating with and manipulating computer hardware.

2.5.2 C (System-Level Programming)

C and other system-level programming languages attempt to bridge the gap between low-level and high-level programming languages. They are an attempt to keep most of the low-level features of an assembly language available while making the syntax a bit easier to work with. C is much more portable than MIPS or ARM Assembly although it can still give different results when it is run on different hardware. C is a widely used language, making it a good one to learn. It also forms the basis of many of the high-level languages that are around today.

2.5.3 Java (High-Level Programming)

High-level programming languages like Java are about as far away from hardware as you can get in terms of abstraction. All memory allocation in Java is carried out automatically meaning the programmer does not need to think about the hardware at all. This has advantages as it allows the programmer to concentrate on the programming task in hand without having to worry about the extra overheads of hardware manipulation. However, as the compiled assembly code is produced automatically by Java Virtual Machine (JVM) it can be much less efficient than if a human programmer had written it themselves. Java is highly portable, and can be used on any machine architecture that has JVM installed. JVM compiles Java source code into an intermediate language called Java Byte Code before compiling it to the relevant assembly language for the architecture it is running on. The underlying architecture, for example the processor that is being used in the computer the

Java code is being written on, is not something that has to even be considered by the programmer; everything is dealt with behind the scenes.

Java is ideal for high-level programming tasks such as mobile/web/desktop application development, but not so useful if you need direct control to hardware.

3.0 Technical Literature Review

For this section I chose to discuss other online resources that are commonly used by programmers. All three sites I have chosen have at least some features that I would like to include in my own final product. The resources that I will discuss are Codecademy [16], Stack Overflow [17] and Best Programming Language for Me [18].

3.1 Codecademy

Codecademy is an online resource for learning programming languages. It provides lessons and tutorials for learning specific languages as well as tasks that you might need to do as a programmer such as setting up a live website. Another feature of Codecademy, other than the lessons is choosing what you learn by area (programming, web development or data science) or language, which is something that I would look to implement in my own website. Another interesting feature of Codecademy is that when you choose an area of programming, it also tells you what the average salary is for someone who works in that area.

Advantages:

- **Lots of Detail;** Codecademy provides lots of detailed information about each of the languages it has on its system, and it provides a range of tutorials or lessons that can take anything between 4 and 40 hours to complete.
- **User Accounts;** Giving users the ability to create their own account can be useful as it allows them to keep track of what they have learned as well as increasing the level of your account as you progress.

Disadvantages:

- **Only a small number of languages;** While this might be suitable for a tool like Codecademy, the website I am building should have information on a much larger set of languages.

- **Lack of choosing a language by level;** Although there are lots of different levels of lessons on the website it is not always clear what areas of the website are better for beginners or experienced programmers for example. Further to this, if you choose to do a course in a language you already know, you can't skip the easy parts at the beginning and so this can be quite tedious to work through.
- **Need an account to use the service;** The fact you need to create an account to use any of the services on Codecademy can be frustrating for programmers as it is another set of login details to remember and it may put them off if they only want to use a small number of functions on the website.

3.2 Stack Overflow

Stack Overflow is an online community for programmers to come together and learn about programming, mostly through asking and answering questions that all other members of the system can get access to. Related questions usually appear from your Google (or other search engine queries) but you can also search for questions by 'tag' which usually means the name of the programming languages that are involved in the question. This system also provides job listings for software developers.

Advantages:

- **Widely used;** Stack Overflow is very popular with programmers, meaning that most questions you would ask can more than likely be answered by some other member.
- **No need to sign up;** If you simply want to use this tool to view questions that have already been asked and answered, you don't need to create an account.
- **Voting system;** Users can 'vote-up' or 'vote-down' questions and answers, which helps you as a programmer to easily identify what the best answers are.

Disadvantages:

- **Not designed for beginners;** Beginners using the system and asking a question for the first time need to be prepared that some of the more experienced users aren't very friendly. Typically, a new user might get their questions downvoted and told that their questions don't make any sense.
- **User Accounts;** On Stack Overflow you must have a user account to ask questions. My solution will have the option to ask questions anonymously or without anonymity, meaning users can ask questions more freely without getting embarrassed when asking simple questions. These discussions will be reviewed by an administrator to avoid spam and disruptive users.

3.3 Best Programming Language for Me

This website is designed specifically to allow users to choose a programming language based on what they need it for, a similar task to that of this project. Best Programming Language for Me has a simple interface, and allows users to reach some language after answering a series of questions, getting closer to a specific answer after each question.

Advantages:

- **Short number of questions lead to an answer;** This is useful as it avoids the user getting bored going through a huge number of questions. Usually on this website, you can have an answer after only 4 or 5 questions.
- **Explanation;** When you are given an answer to what language you should use, you also get a description of the language and some information of why it would be good for your situation.
- **Well broken down;** Getting from the start of the website to some programming language is made simple by the fact that at each stage languages are broken down into different types. For example, a breakdown at one stage might be mobile

development, games development or desktop development, then if you selected mobile development, you would have the choice between iOS, Android or Windows.

Disadvantages:

- **Lack of tutorials on site;** To actually start learning a language you have to follow a link to a different website.
- **Static;** The webpages are static which means adding a new language might be more time consuming than if it were in a database for example. This is due to the fact that the database could dynamically update the structure of the website when a new language is added.
- **Only one feature;** The questionnaire is the only function of the website. It is fit for purpose but would be more interesting if it had added on features such as a discussion board.

In summary, I have discussed three online resources that are used by programmers and due to what I have discussed I will add a number of useful features similar to the ones on these websites, for example:

- Detailed information pages about programming languages.
- A questionnaire function which tells users the best programming language to use for their task.
- A user-friendly interface that is welcoming for both beginners and experienced programmers.
- A dynamic website that can be easily updated to accommodate new languages.
- The option to create a user account, however this won't be needed for most of the functions available.
- A discussion board where registered users can discuss problems they are having and answer each other's questions.
- A display of the average salary in different fields of computer science related work.

4.0 Survey to Find the most Popular Languages for Programmers

Between the 13th and 22nd November I carried out a survey to find out what the most popular programming languages are for programmers. The survey was aimed mostly at students, but also anyone with programming experience. After an initial pilot survey which I asked 4 of my colleagues to carry out I made some minor changes to the wording of some of the questions to make them more clear and added a new question at the end to discuss how likely programmers are to use a new programming language if given the choice. 29 people took part in the final survey, with 82.8% of them being under 25 and a fair mix of students between years 1 and 4.

The survey contained 4 scenario questions, each of which had 4 options of programming languages to use in that situation as well as an 'other' option for participants to give their own suggestions. This allowed me to get some concrete answers for languages as well as opinions on ones I may not have thought of. Participants were also asked to explain their answer for each scenario.

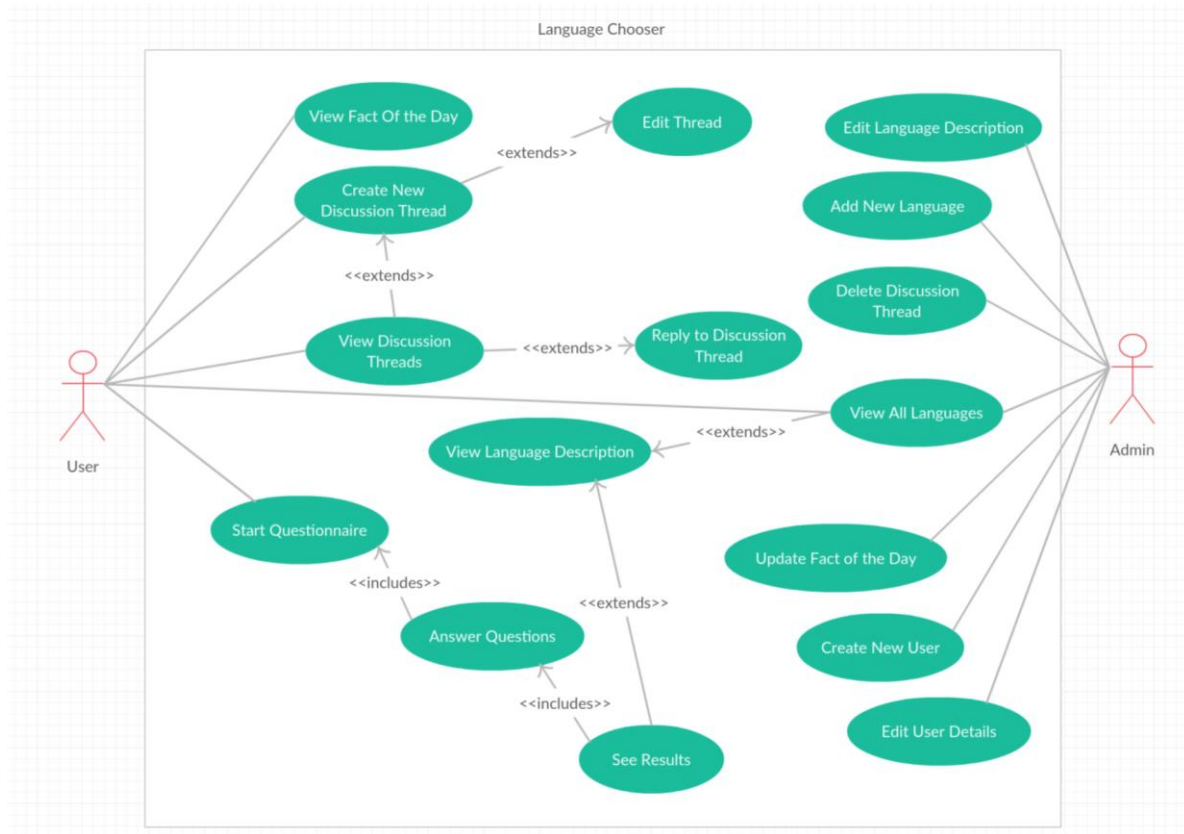
The next few questions looked for languages in terms of their type, for example object-oriented programming, server-side scripting and database manipulation. The top answers for these ones were Java, PHP, and SQL respectively, and most of these answers came down to the fact they are widely used, because the participant had more experience with them or because there is a wide range of documentation available for them.

Participants were then asked to name the first language they thought of for each paradigm and this provided a range of responses including a few languages I had never heard of before such as OCAML and F#. The final question asked how likely they would be to switch to a new language if it was better suited to their purpose and this resulted in an average rating of 3.71 out of 5, suggesting that most would be happy to switch languages.

For the full set of results on this survey see appendix 1.

5.0 Requirements Analysis

5.1 Use Case Diagram



There will be two actors/users on this system. They are the Admin (who has a login account and special permissions) and the User (a regular visitor to the site with no login details)

Textual descriptions for each of the use cases can be seen at appendix 2.

5.2 Requirements

To categorise the requirements, I have used MoSCow analysis, where each requirement falls into one of 4 categories:

1. **M - Must have;** The final product must fulfil this requirement to be complete.
2. **S - Should have;** The final product should fulfil these requirements but does not necessarily have to in order to be successful.
3. **C - Could have;** This category covers requirements that are extra, optional requirements that could be implemented if I have spare time.
4. **W – Would like to have;** Extra requirements that would have been fulfilled if there had been more time available to complete the project.

The requirements can also be described in one of the following two ways:

1. **Functional;** Requirements that refer to the specific functions and services that are expected of the completed system.
2. **Non-Functional;** Requirements that are not concerned with specific system functionality. Instead they deal with performance or external constraints such as availability and security.

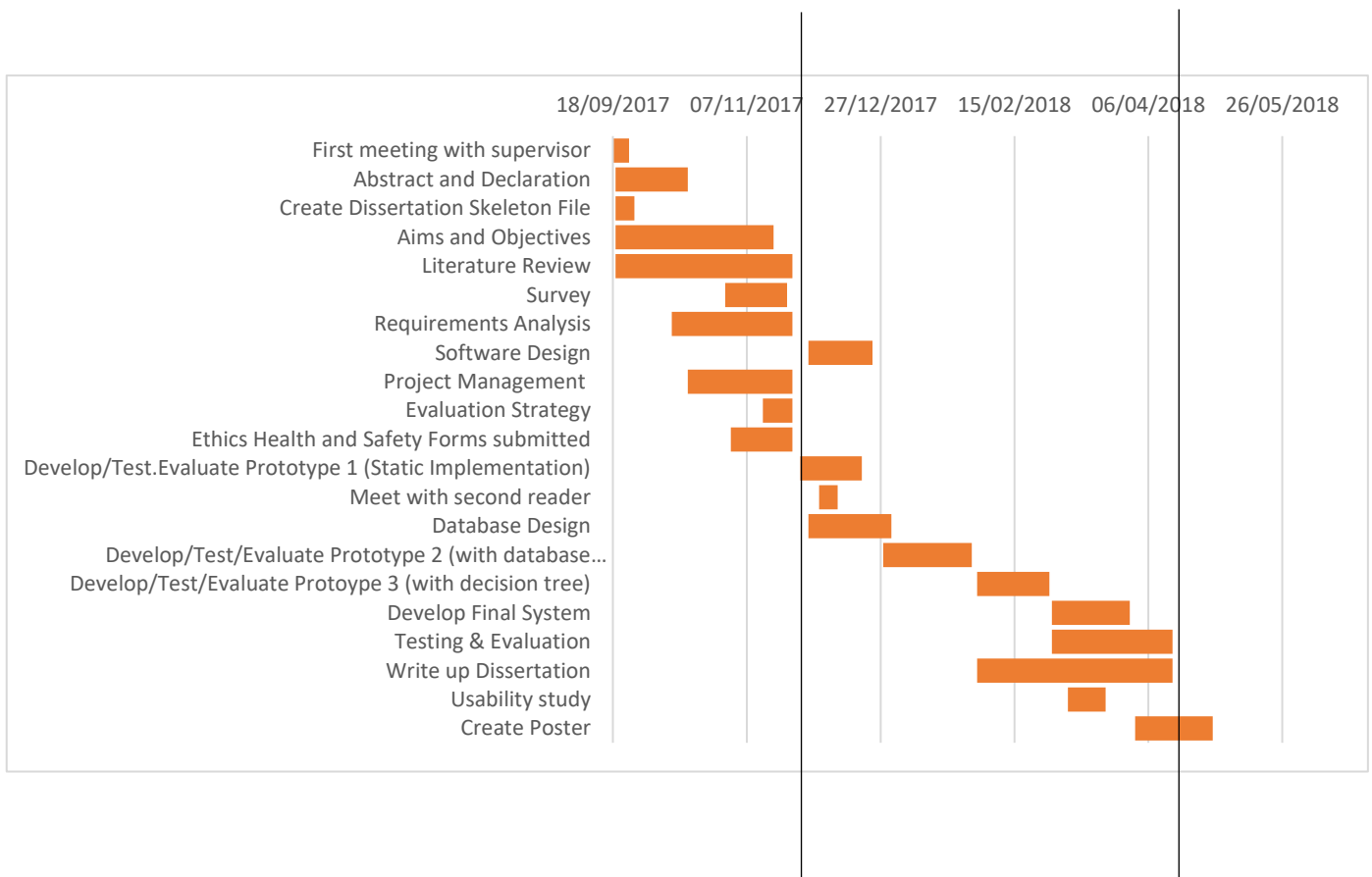
Functional requirements have the letters FU at the start of their ID whereas non-functional requirements are marked with NF.

ID	Description	MoSCoW
FU-01	The system shall provide an information page about every language in its database.	M
FU-02	The system shall provide a questionnaire-style function, whereby a user can answer a series of short closed-end questions to come to one or more suitable answers.	M
FU-03	The system shall provide the user with informative answers after they have completed their questionnaire which tells them a bit about the languages that are being advised to them and links them to tutorial or information sites.	M
FU-04	The system shall provide a 'fact of the day' for its users. For example: "The first high-level programming language was FORTRAN. It was invented in 1954."	S
FU-05	The system shall allow admin users to update the 'fact of the day'.	S
FU-06	The system shall allow admin users to add and remove languages from the underlying database.	S
FU-07	The system shall allow admin users to edit the description for programming languages in the database.	S
FU-08	The system shall allow the user to see a comprehensive list of all programming languages in the database.	M
FU-09	The system shall have a discussion board.	C
FU-10	The system shall allow users to create their own discussion threads.	C
FU-11	The system shall allow users to view other discussion threads.	C
FU-12	The system shall allow users to make comments on other users' discussion threads.	C
FU-13	The system shall allow admin users to delete discussion threads.	C
FU-14	The system shall allow a user to delete their own discussion threads.	C
FU-15	The system shall allow a user to rate a language and leave a comment.	W
FU-16	The system shall provide its own language-specific tutorials for users to take part in.	W
FU-17	The system shall allow users to vote for their favourite programming languages.	W
FU-18	The system shall allow users to browse programming languages by popularity.	W
NF-01	The system shall be available online, 24 hours per day, 7 days per week.	M
NF-02	The system shall run correctly on all the main Web Browsers (Chrome, Microsoft Edge, Firefox, Safari).	C
NF-03	The system shall be usable and will adjust itself to work on all platforms including desktop, mobile and tablet.	C
NF-04	The system shall be visually pleasing for the user.	M
NF-05	The system shall be secure to access.	C
NF-06	The system shall use https.	W
NF-07	The system shall be described as 'Secure' in a Google Chrome web browser, which means it passes all data in forms securely.	W
NF-08	The system shall be extendable. I.e. it should be designed in such a way that new features can be added later.	M
NF-09	The system shall be able to deal with multiple concurrent users.	M

6.0 Project Management & Product Design

6.1 Gantt Chart

The chart below shows the main tasks of the project and the dates they should be completed by. The two vertical lines show the main deadlines; Deliverable 1 deadline on 27th November 2017 and Deliverable 2 deadline on 23rd April 2018. In both cases I have planned to complete all the important tasks a few days before to avoid last minute changes.



6.2 Risk Analysis

Risk analysis is an extremely important part of most large software projects. I have identified a number of risks to this project and arranged them in the table below. Each risk in the table has a description, to explain in more detail what it is, and an impact which describes if its impact on the project will be tolerable; a minor inconvenience, severe; problematic to the project but not causing it to fail, or catastrophic; which would cause the project to fail completely if it occurred. Each item in the table also has a likelihood; how likely it is to occur (high, medium or low) and a mitigation strategy which explains how I will attempt to avoid running into this risk. It is important to keep the likelihood of high impact risks as low as possible.

Risk	Description	Impact	Likelihood	Mitigation
Time Constraints	Not enough time to complete all of the defined requirements	Tolerable	High	Prioritise requirements using MoSCoW analysis.
Legal Issues	Issues relating to using someone else's source code or images and not making it clear when I am using someone else's work.	Catastrophic	Low	Make sure all work owned by another author is properly cited and don't use someone else's work without their permission.
Ethical Issues	Not receiving the correct ethical approval for the project	Catastrophic	Low	Make sure the necessary ethical approval for the project has been carried out

				before carrying out any surveys.
Compatibility	The product may not be compatible over all available browsers and platforms.	Tolerable	Low	Test the product on as many browsers and platforms as possible; at least all the main browsers.
Workload	Inability to balance workload between this project and others on the course.	Severe	Medium	Keep to the project plan as much as possible and adjust it if necessary. Organise the available time and prioritise important tasks.
Data Loss	Corruption or loss of source code or other project related data.	Catastrophic	Low	Regularly backup all important project data both on an external USB drive and online.
Low survey response rate	Not enough participants in the survey to achieve interesting results.	Severe	Low	Advertise the survey as much as possible through email and social media.

Problems with working off-campus	Inability to work from home due to having different software installed or a different setup to systems like MySQL.	Severe	Medium	Set up all necessary software on personal and university machines as early as possible.
Not sticking exactly to the project plan	Not completing tasks on time or in the planned order.	Tolerable	Medium	Allow plan to be changeable. Make sure tasks on the critical path are completed on time.

6.3 Professional, Legal, Ethical and Social Discussion

My survey, discussed in section 4.0 was carried out professionally using an official survey-building website and participants were invited to voluntarily take part. It was made clear to them that there was no obligation to take part or to answer all questions if they didn't want to. Small amounts of personal data were collected about participants including their age range, and if they were a student their year and course. This data has been stored securely and anonymously on the Survey Planet website [19].

Since all participants gave consent by taking part in the study, all participants are over 16 years of age and there was no information taken from the participants that they do not know about there are no other legal, ethical or social issues to discuss for this project.

6.4 Early Product Design Ideas & Methodology

In the early stages of this project I thought up 3 main ideas of how the questionnaire system in this online tool could work:

1. **Static Web Links;** At each stage the user is asked some question and given a few possible answers. Upon selecting an answer, they will be taken to the next question until a final result is reached. Each link between questions is static meaning it doesn't change. This is simple to design and implement but would be difficult to maintain when, for example, new languages need to be added.
2. **Points Based League Table;** In this situation, each time a question was answered, points would be allocated to all the languages relevant to that answer. At the end of a set number of questions, a list would be shown of the languages with the highest points.
3. **Decision Tree;** This is a way to implement item 1 in this list in a more dynamic and interesting way. Although it would be harder to implement, it would be much more maintainable and adaptable to change.

Based on these ideas, development of the final product will be done in 4 stages, with each stage ending with a prototype that can be evaluated and works towards the end system:

1. Prototype 1 – This will be a static website with no database connection that has a simple but usable interface and contains simple versions of the proposed features. It should implement at least item 1 in the above list.
2. Prototype 2 – At this stage a database will be created to store all the information for the website. This will make it much more dynamic.
3. Prototype 3 – This will be the final prototype and should result in a website with the ability for administrative users to login and update information on the site as well as some implementation of items 2 and 3 in the list above.
4. The final product.

7.0 Evaluation Strategy

To evaluate that my product meets the aims and objectives of this project I plan to carry out a usability study on the final system. This will aim to find general opinions from end-users on the interface of the system and will also include questions that relate to the aims and objectives defined in section 1.0.

In addition to this evaluation of the final system I will be carrying out evaluations at the end of each prototype stage mentioned in section 6.4. These will include small usability studies with only a small number of participants or discussions with my supervisor on what works well and what can be improved. At each evaluation I will review the aims and objectives to make sure I am on track.

8.0 References

Most of my references were found using Google, Google Scholar or Heriot-Watt University's 'Discovery' service: discovery.hw.ac.uk.

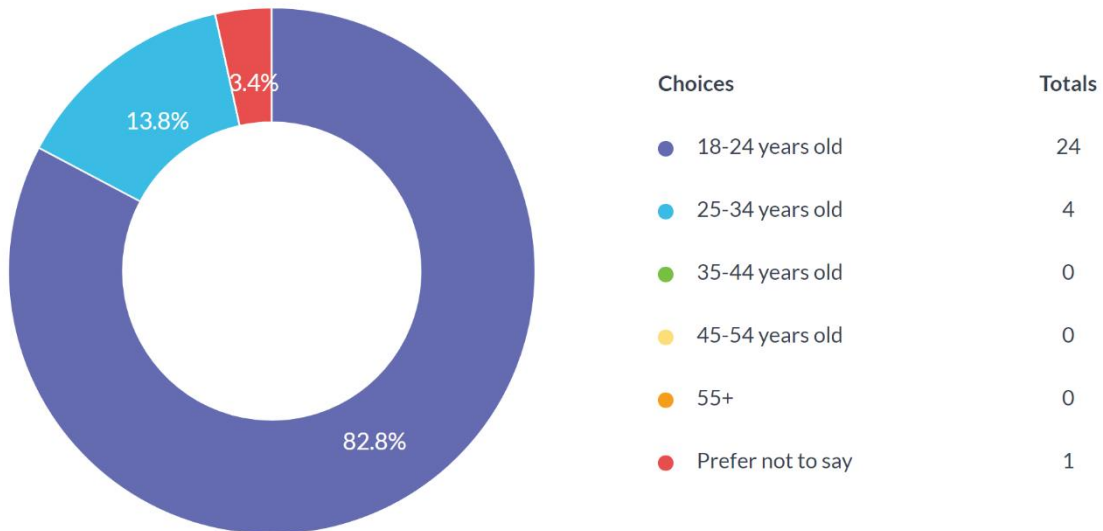
1. Khedker, U.P., 1997. *What makes a good programming language*. Technical Report TR-97-upk-1, Department of Computer Science University of Pune.
2. Chris Webb (2015) *The Pursuit of Simplicity in Programming* [online]. Available from <http://blog.mediumequalsmessage.com/simplicity-in-programming> [Last accessed 9th October 2017].
3. Kate Springer, CNN Tech (2016) *Japan's Fukuoka poised to be the country's Silicon Valley* [online] Available from <http://money.cnn.com/2016/11/16/technology/fukuoka-startup-city/index.html> [Last accessed 9th October 2017].
4. Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A. and Miller, P., 1997. Mini-languages: a way to learn programming principles. *Education and Information Technologies*, 2(1), pp.65-83.
5. MIT App Inventor: Available from <http://appinventor.mit.edu/explore/> [Last accessed 9th October 2017].
6. Rosson, M.B., 1993. How might the object-oriented paradigm change the way we teach introductory programming? *ACM SIGPLAN OOPS Messenger*, 4(2), pp.313–314.
7. Murphy, E., Crick, T. & Davenport, J.H., 2016. An Analysis of Introductory Programming Courses at UK Universities., pp.The Art, Science, and Engineering of Programming, 2017, Vol. 1, Issue 2, Article 18.
8. Ray Toal, Loyola Marymount University *Programming Paradigms* [online] <http://cs.lmu.edu/~ray/notes/paradigms/> [Last accessed 11th October 2017].
9. Gary T. Leavens (1997) *Major Programming Paradigms* [online] Available from www.eecs.ucf.edu/~leavens/ComS541Fall97/hw-pages/paradigms/major.html [Last accessed 11th October 2017].

10. Curricular Linux Environment at Rice (CLEAR) *Lec02: Programming Paradigms*
[online] Available from <https://www.clear.rice.edu/comp310/f12/lectures/lec02/> [Last accessed 11th October 2017].
11. Gabbrielli, M. et al., 2010. *Programming Languages Principles and Paradigms*,
London: Springer London.
12. The Saylor Foundation *Advantages and Disadvantages of Object-Oriented Programming* [online] Available from <https://www.saylor.org/site/wp-content/uploads/2013/02/CS101-2.1.2-AdvantagesDisadvantagesOfOOP-FINAL.pdf>
[Last accessed 20th October 2017].
13. Matt Weinberger (2017) *The 15 Most Popular Programming Languages according to "Facebook for Programmers"* [online] <http://uk.businessinsider.com/the-9-most-popular-programming-languages-according-to-the-facebook-for-programmers-2017-10/#8-c-8> [Last Accessed 20th October 2017].
14. Cass, S., 2015. The 2015 top ten programming languages. IEEE Spectrum, July, 20.
15. Quora (2016) *"In which fields assembly language is used?"* [online] Available from <https://www.quora.com/In-which-fields-assembly-language-is-used> [Last accessed 9th November 2017].
16. Codecademy [online]. Available from <https://www.codecademy.com/> [Last accessed 14th November 2017].
17. Stack Overflow [online]. Available from <https://stackoverflow.com/> [Last accessed 14th November 2017].
18. Best Programming Language for Me [online]. Available from <http://www.bestprogramminglanguagefor.me/> [Last accessed 14th November 2017]
19. Survey Planet [online]. Available from <https://app.surveyplanet.com> [Last accessed 16th November 2017].

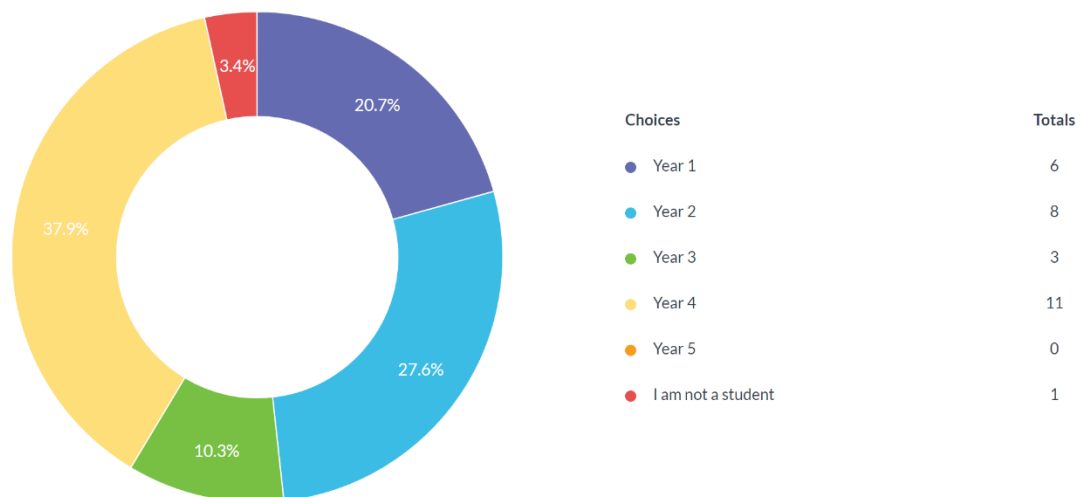
9.0 Appendices

Appendix 1 – Survey results

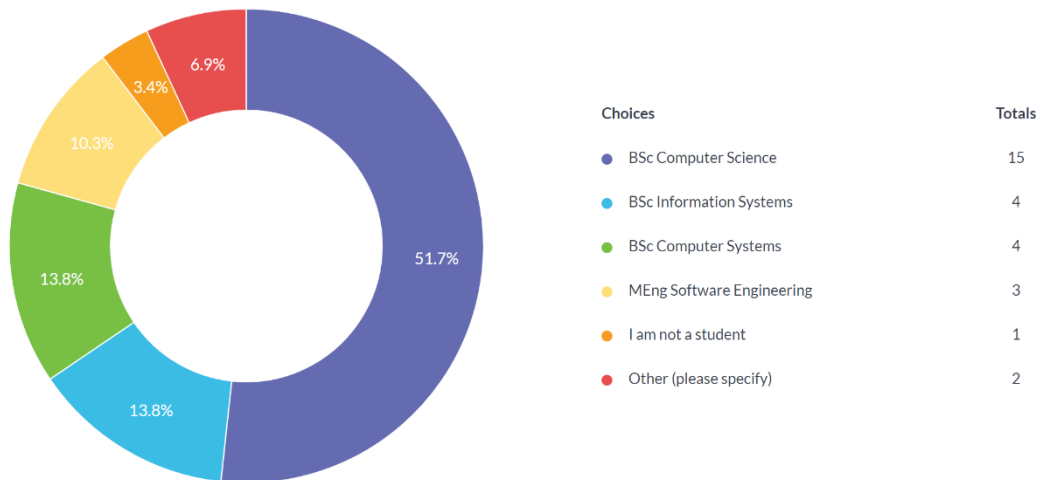
Q1. Please select your age range from the list below.



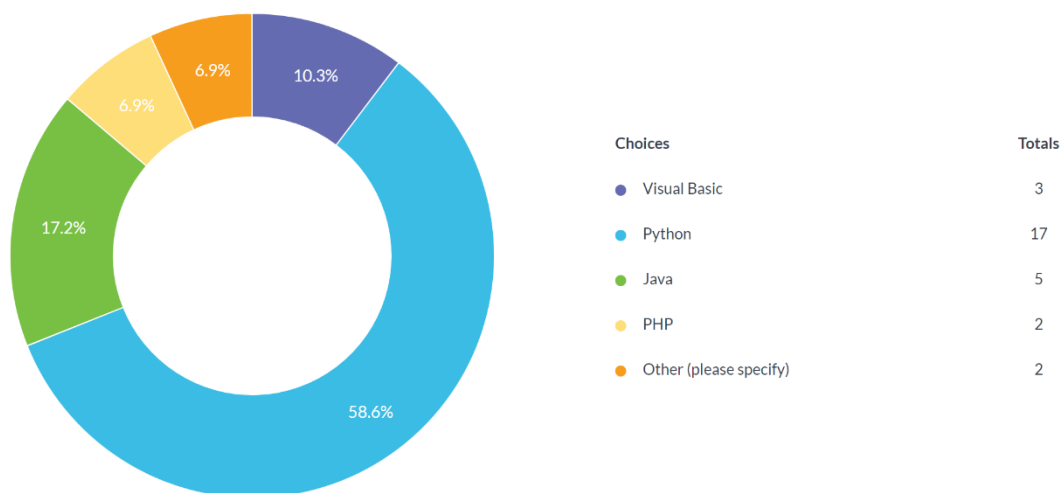
Q2. If you are a student, please select your year of study.



Q3. If you are a student, please select the programme you are studying.



Q4. Scenario 1: You have been asked by a local high school to come in and teach their 4th year students a new programming language over a time period of around 3/4 weeks. They are relatively new to programming but have learned the basics through the visual programming language Scratch and they have also done some small websites with HTML. What language do you choose to teach the students? Be ready to explain your answer for the next question.

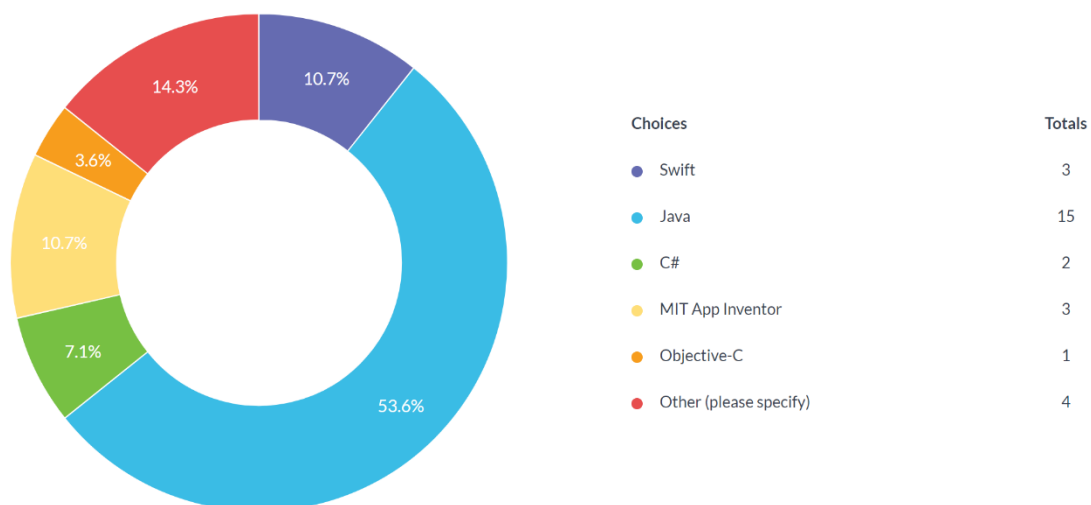


Q5. Please explain the answer you gave for this scenario in the previous question.

Some common answers:

- (Python) Syntax is easy to understand.
- (Python) Easy to learn for beginners and can be applied to many different areas of software development.
- (Python) Forces them to think about code readability through indentation.
- (Java) Some people find it easier to teach the concept of object-oriented programming.
- (Java) Widely used and has good libraries.
- (Visual Basic) Simple syntax and the Visual Studio IDE makes things simpler for beginners than writing in a text file directly or in the terminal.

Q6. Scenario 2: You have just started working at a company that develops mobile apps. Your boss says that he wants you to take the lead on building a new app that can be used to track the users' fitness (counting steps, recording what they eat etc.). It's up to you which mobile platform you build the app for (between iOS, Android and Windows), but once it has been built the company will recreate it for other platforms if it is successful. What language would you choose to work in? Please be prepared to answer why in the next question.

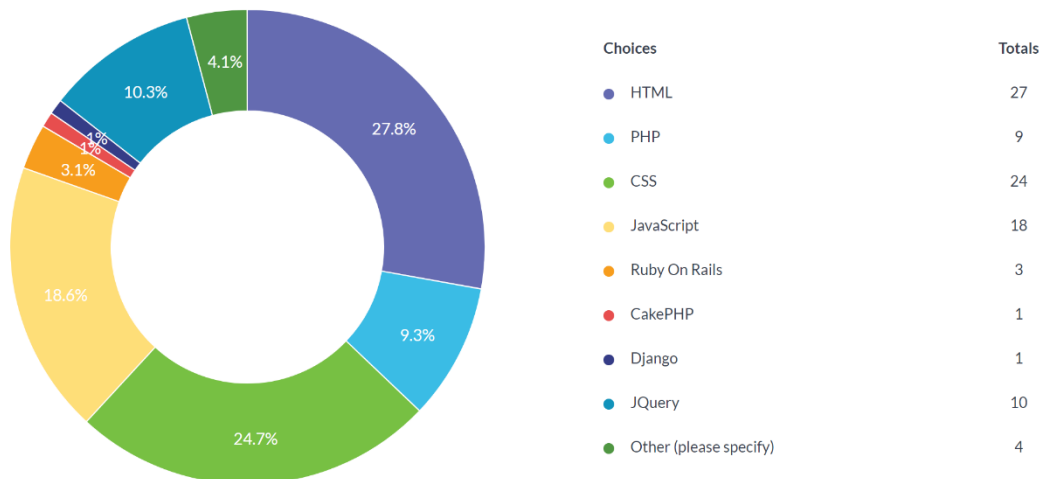


Q7. Please explain the answer you gave for this scenario.

Some common answers:

- (Python) It is a powerful language and is easy to use, it also has a huge amount of documentation to help you.
- (Swift) as this is currently used for iOS applications.
- (Java) Android has the largest market share.

Q8. Scenario 3: You are working part-time at a small cafe in Edinburgh to earn some extra cash while you study. The cafe manager finds out you study computing at university and asks if you could build a website for the cafe to extend their reach for customers. You gladly accept (this will be a great personal project to put on your CV). What languages/frameworks will you choose? You may choose multiple. Please be prepared to explain your answer in the next question.

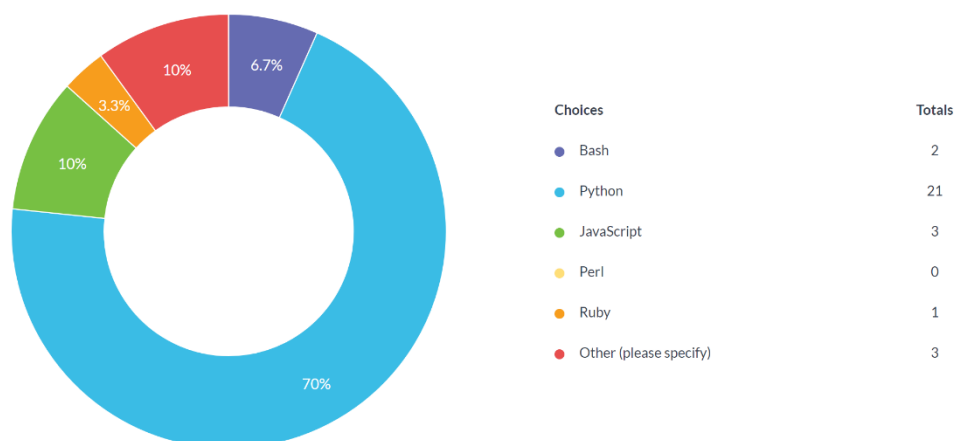


Q9. Please explain the answer you gave for this scenario.

Some common answers:

- (HTML, CSS, JavaScript) Widely used.
- (WYSIWYG Service Square Space) Only a small business and it allows the developer to offload website maintenance to less skilled users.
- (HTML) Not all browsers support all formats, so sticking with the classics is a good way to go.

Q10. Scenario 4: As part of a coursework task at university, you have to search a large text file for certain words and phrases and count the number of times each one occurs. You have also been told to do this using a scripting language. Which language would you use?

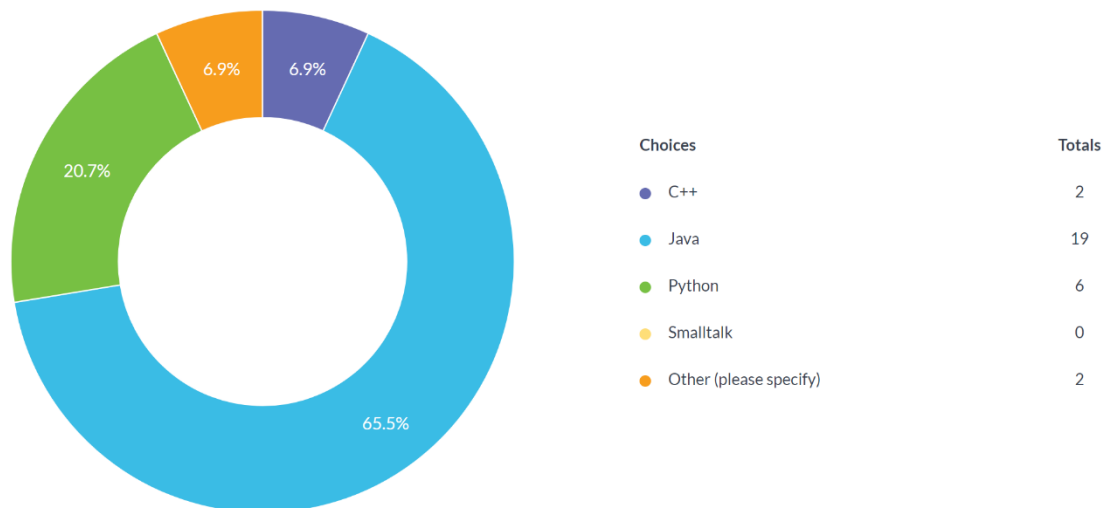


Q11. Please explain the answer you gave for this scenario.

Some common answers:

- (Python) Simple to use.
- (JavaScript) Allows you to use regexes to do this sort of task.
- (Bash) Easy to use in the command line.

Q12. What is your favourite language for object-oriented programming?

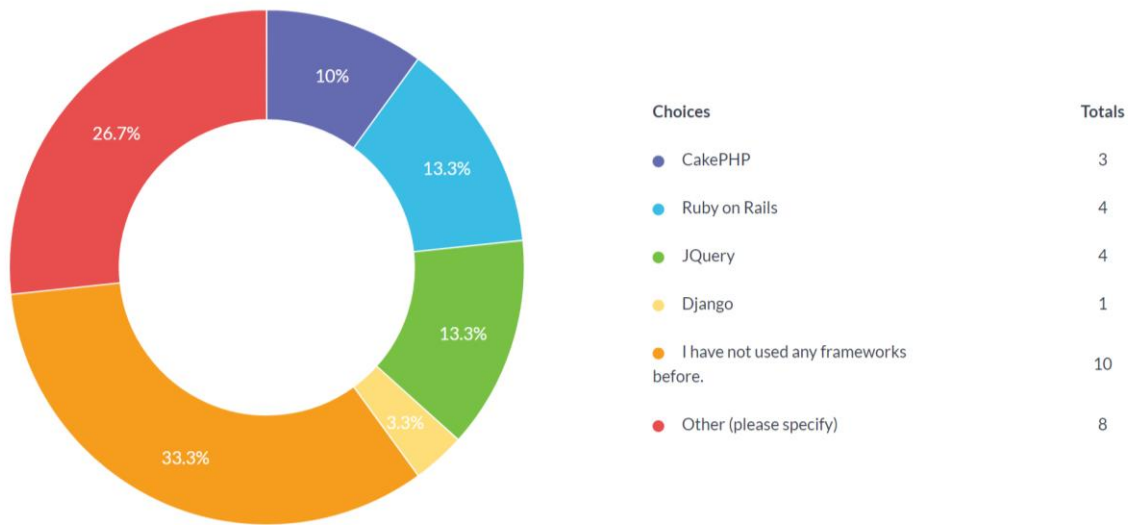


Q13. Please explain your answer to the question 'What is your favourite language for object-oriented programming?'

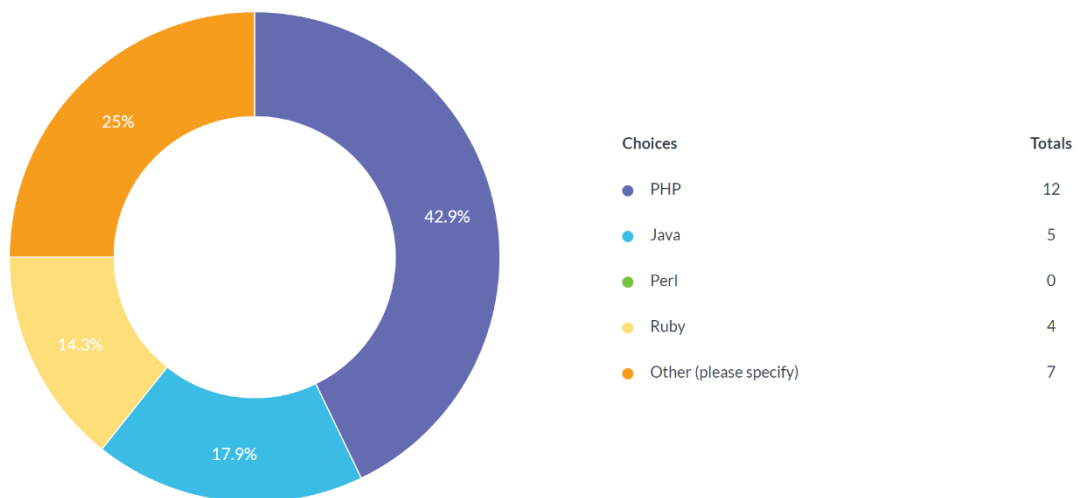
Some common answers:

- (Java) Has good libraries and syntax.
- (Java) More experience with Java.
- (C++) Extra features such as async and await.
- (C++) Good low-level control.
- (Python) Good syntax.

Q14. Have you worked with frameworks before for web development? If so, please select your favourite from the list below.



Q15. What is your favourite language for server-side scripting?

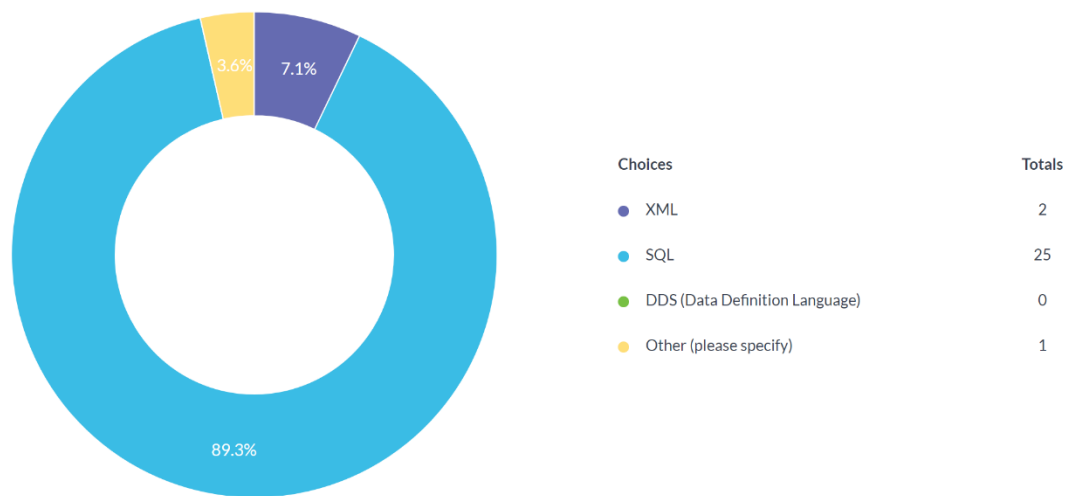


Q16. Please explain your answer to 'What is your favourite language for server-side scripting?'

Some common answers:

- (PHP) Widely used.
- (Python) The Flask framework is useful.
- (Ruby) Experience.

Q17. What language would you prefer to write and manipulate databases with?



Q18. Please name the first programming language you think of for each of these paradigms. You can miss some out if you can't think of an example at this time. Object-Oriented, Procedural/Imperative, Declarative, Functional, Logical, Event-Driven, Parallel. The most common for each paradigm has an asterisk* beside it.

Language	Object-Oriented	Procedural/Imperative	Declarative	Functional	Logical	Event Driven	Parallel
Java	* *						
C++							
Prolog			*		 *		
JavaScript						*	
Python		*					
SML				*			
Haskell							
ML							
C#							
Basic							
SQL							
PHP							
C							*
Cuda							
F#							
Pascal							
Visual Basic							
OCAML							
NodeJS							
Go							
PDDL							
MatLab							
Erlang							
Event-ML							
Pascal							
Lisp							

Q19. If you are working on a new task and you discover that a language you have not learned before is much better suited for this task than languages that you already know, how likely are you to learn this new language from 1 – not likely at all to 5 – Extremely likely?

Average: 3.71

Appendix 2 – Use Case Textual Descriptions

Textual descriptions for each of the use cases in the use case diagram 'Language Chooser'.

Each textual description has a unique ID number, a name, actors who will be able to carry out the use case, a general description, a main flow of steps that will be involved in carrying out the use case, preconditions and postconditions (conditions that must be true before and after the use case). The main reason for the ID number is so that each use case can be uniquely identified. This ID will not change throughout the project, even if the name of the use case does.

ID: 1.0

Name: View Fact of the Day

Actors: User

Description: The act of the user seeing and reading the fact of the day that will be displayed on the website's homepage.

Main Flow:

1. The user navigates to the website homepage.
2. The fact of the day is displayed on screen.

Preconditions: The user must be on the website homepage.

Postconditions: The user now knows the fact of the day

ID: 2.0

Name: Create New Discussion Thread

Actors: User

Description: The act of the user creating a new discussion thread which can be used to post problems that you are having or questions you have for other users on the system.

Main Flow:

1. The user clicks on the 'Create new discussion thread' option.
2. The user enters details for the discussion thread:
 - a. Subject
 - b. Description/question
3. The user selects the 'Submit' option to make the thread active.
4. extensionPoint: Edit Thread

Preconditions: The user must be on the discussion threads page.

Postconditions: A new discussion thread has now been created that can be edited, viewed and replied to.

ID: 3.0

Name: View Discussion Thread

Actors: User

Description: The act of a user viewing some discussion thread that has been created by themselves or another user.

Main Flow:

1. The user will navigate to the thread that they want to view.
2. The thread information and replies will be displayed.
3. extension: Reply to Discussion Thread.

Preconditions: The user must be on the discussion threads page.

Postconditions: The user has now viewed the discussion thread they were interested in.

ID: 4.0

Name: Start Questionnaire

Actors: User

Description: The act of a user taking part in the questionnaire function of the website.

Main Flow:

1. The user selects the 'Start Questionnaire' button.
2. include(Answer Questions)

Preconditions: The user must be on the main page of the website.

Postconditions: The user has started the questionnaire.

ID: 5.0

Name: Edit Language Description

Actors: Admin

Description: The act of an administrator editing the description associated with a particular programming language in the system's database.

Main Flow:

1. The administrator will navigate to the language they want to change the description for.
2. The administrator will type into an input form the new description that they want the language to have.
3. The administrator will select 'Submit'.

Preconditions: The actor must have an admin account and must be logged in.

Postconditions: The language now has a new description.

ID: 6.0

Name: Add New Language

Actors: Admin

Description: The act of an administrator adding a new programming language to the database.

Main Flow:

1. The administrator will navigate to the admin languages page.
2. The administrator will select the 'Add new language' option.
3. The administrator will enter details about the language into an input form and select the 'Submit' option.

Preconditions: The actor must have an admin account and must be logged in.

Postconditions: A new language has been added to the database.

ID: 7.0

Name: Delete Discussion Thread

Actors: Admin

Description: The act of an administrator removing a discussion thread from the system. This may be done at request from a user or due to it being outdated.

Main Flow:

1. The administrator will navigate to the admin threads page.
2. The administrator will select the thread they wish to delete, and click the 'delete thread' option.

Preconditions: The actor must have an admin account and must be logged in.

Postconditions: The discussion thread has now been deleted.

ID: 8.0

Name: View All Languages

Actors: Admin, User

Description: The act of an actor viewing a summary of all languages in the database. The view will be slightly different for admin users and regular users.

Main Flow:

1. The actor will navigate to the 'view all languages' page.

2. extensionPoint: View Language Description.

Preconditions: A User must be on the website. An Admin must be on the website, have an admin account, and must be logged in.

Postconditions: The actor has viewed a list of all languages in the database.

ID: 9.0

Name: Update Fact of the Day

Actors: Admin

Description: The act of an admin manually updating the fact of the day.

Main Flow:

1. The actor will navigate to the administrator view of the fact of the day.
2. The actor will then select the 'edit fact of the day' option.
3. The actor will fill an input form with the new fact of the day and select 'submit'.

Preconditions: The actor have an admin account, and must be logged in.

Postconditions: The fact of the day has been updated.

ID: 10.0

Name: Create New User

Actors: Admin

Description: The act of an administrator creating a new administrator account.

Main Flow:

1. The actor will navigate to the 'users' page.
2. The actor will select 'Create new User' option.
3. The actor will fill in a form with information about the new user and press 'submit'.

Preconditions: The actor have an admin account, and must be logged in.

Postconditions: A new admin user has been created.

ID: 11.0

Name: Edit User Details

Actors: Admin

Description: The act of an administrator updating the details for a particular admin user.

Main Flow:

1. The actor will navigate to the 'users' page.
2. The actor will select a user to edit.

3. The actor will fill in a form with the information they want to change and press 'submit'.

Preconditions: The actor have an admin account, and must be logged in.

Postconditions: A user's details have now been updated.

ID: 2.1

Name: Edit Thread

Actors: User

Description: The act of the user editing the details of a discussion thread that they have created.

Main Flow:

1. The user navigates to the thread they created.
2. The user edits the details of the thread through input forms and clicks 'submit'.

Preconditions: The user must have created a discussion thread already.

Postconditions: The details of the thread have now been updated.

ID: 3.1

Name: Reply to Discussion Thread

Actors: User

Description: The act of a user posting their own comments in reply to a discussion thread that has already been created.

Main Flow:

1. The user selects 'reply'.
2. The user enters their comment into a form and selects 'submit'.

Preconditions: The user must be viewing a specific thread.

Postconditions: The user has now replied to the discussion thread.

ID: 8.1

Name: View Language Description

Actors: Admin, User

Description: The act of an actor viewing the description for a specific language in the database. The view will be slightly different for admin users and regular users.

Main Flow:

1. The actor selects a language to view.
2. The description will be shown on screen.

Preconditions: None

Postconditions: The actor has viewed the description for a particular language.

ID: 4.1

Name: Answer Questions

Actors: User

Description: The act of a user answering questions in the questionnaire function of the website.

Main Flow:

1. The user answers the current question to take them closer to a specific programming language.
2. The user repeats step one until a result is obtained.
3. include(See Results)

Preconditions: The user must have started the questionnaire already.

Postconditions: The user has answered question(s) on their problem and possibly been given a result.

ID: 4.2

Name: See Results

Actors: User

Description: The act of a user viewing the results of their answers to the questions they answered.

Main Flow:

1. The user completes the questionnaire.
2. A programming language or list of programming languages are displayed on the screen.
3. extensionPoint: View Language Description

Preconditions: The user must have started the questionnaire already.

Postconditions: The user has answered question(s) on their problem and possibly been given a result.
