Unit 6, Assignment 1— Which Language?

Task 1 (P2) – Why Choose a Particular Programming Language?

Thomas Robinson

Organisational Policy

- The company may be specialised and only make use of a handful of specific languages
- The organisation may not have the resources to procure new IDEs, software or computers

Suitability

- An application written for a desktop computer is unlikely to make use of a backend web-development language such as PHP and likewise, an application for the web will not make use of a desktop language like C#.
- An application that needs direct access to resources, such as a game or other high-performance program would use a lower level language rather than something like Java which runs sandboxed in a virtual machine.
- Developing for certain platforms suchs iOS restricts the languages that can be used

Availability of Staff

- Many languages are decades old and new programs have not been written in them in some time. It may be difficult to find someone with sufficient knowledge ate or maintain an application.
- Choosing a language the company’s employees are not familiar with can cause issues since the developers will need to learn the intricacies of a new language before confidently starting a project.

Reliability & Scalability

- How well will the language perform if presented with information or inputs it is not expecting?
- Can the program support significant, sustained usage now and in future?
- Could the application scale to support more users, locations, etc? Through the use of clustering or load-balancing?

Maintenance Costs

- How active is the community around the language?
- How easy is it to find and locate documentation?
- Will the language be supported far into the future?
- Costs of required hosting infrastructure.
- Costs of keeping the infrastructure maintained and updating the program to support newer OS versions

Unit 6, Assignment 1—Which Language?

Thomas Robinson

SOLIHULL COLLEGE

Thomas Robinson

Task One (P1) – Applications & Limitations of Programming Paradigms

Procedural

- Programs are structured and ran in a set-order, often line-by-line.
- Can include subroutines that can be called and ran multiple times

Examples of procedural languages include BASIC, COBOL, FORTRAN and Pascal. Most modern languages such as JavaScript can also be used procedurally.

Procedural programming is commonly used when specificity, reliability and repeatability is important, such as in the manufacturing and medical sectors.

Programs written in this way require less resources than programs written with other paradigms.

Downsides to the procedural paradigm include the difficulty in being able to store collections of data about a single item. Additionally, program source code can be more complicated with many separate parts that are not clearly linked. This can make it harder to debug and maintain.

Object-Oriented

- An "object" can contain properties and functions/methods inside of itself

Most modern languages have support for writing with the object-oriented paradigm, such as Java, Python, PHP, C#.

Common uses for object-oriented programming include games and complex data-driven applications.

Object-oriented languages are good for representing real-world objects and data. It is often easy to maintain and expand in future since objects can be expanded to have more properties and methods in future.

It is often harder to learn the concepts and ideas around object-oriented programming and the way in which objects are linked and interact can become confusing.

Event-Driven

- Actions are taken upon an event taking place, such as user input or a message from another program or physical hardware.

As with the object-oriented paradigm, many modern languages such as Visual Basic/.NET, Java, JavaScript and Python can support being event-driven.

This paradigm can make use of user input as well as input from other applications which makes it useful for applications such as interactive websites and applications.

The program flow of an event-driven program can be unintuitive and not obvious since tasks happen upon certain events taking place rather than in a certain, logical order. This can make a program harder to debug due to the large variety of potential inputs. Additionally, these programs could be slower if the task scheduler has to process many inputs at once.

Task Two (P4) – Data Types (C#)

Data Type (C#)	Description and example use
String	A word or phrase – eg: a name or address. Takes up a variable number of bytes depending on string length.
Integer	A whole number up to 32767. Takes up 4 bytes in memory.
Floating point	A number with a single decimal place, for example a percentage. Takes up 4 bytes in memory.
Byte	A single byte (of eight bits) that can store an int from 0-255.
Date	An object that stores a date, taking up 8 bytes.
Boolean	A variable with two vales – true or false (1 or 0). Takes up 2 bytes.
Single	An alternative name for a float.
Double	A number with more than one decimal point, for example a co-ordinate. Takes up 8 bytes.
Fixed Point	A number stored with an exact number of decimal places, for example a monetary amount.

Memory Management

The overhead of storing different types of data is different depending on the datatype. An empty string will take up a different amount of memory to an empty float which will take up a different amount of memory to an empty date and so on.

To enable to the program to be more efficient with its memory management, and therefore to be more performant and use less memory, it is necessary to specify data types for variables in some lower-level programming languages.

Data Validation

Strict data types for variables ensures that an improper value doesn't get assigned where it should not be, for example assigning a String where an Int should go.

It can also prevent impossible operations before they happen, for example adding a Date and a String. This makes debugging easier since potential improper operations can be spotted before one is attempted.

Task 3 (P3)

Selection

In computer programming, a selection is when a decision must be made. What code is executed next is determined by this decision.

```
let input = parseFloat(prompt("Enter a number!"));

if(input > 10) {
    alert("Greater than ten!");
} else {
    alert("Less than ten!");
}
```

If the number given by the user is greater than 10, then they will be alerted of this. Otherwise, they will be told the number is less than ten.

Iteration

Iteration refers to running code a certain number of times, or until a certain condition is met.

```
let input = parseInt(prompt("Run how many times?"));

for(let i = 0; i > input; i++) {
    console.log(`Ran ${i} times!`);
}
```

In the above example, the `console.log` statement is being ran as many times as the user has inputted.

Sequence

In a sequential program, the lines are ran in order and every step is executed or considered. In the below example, two user-inputted numbers are summed together.

```
let input1 = parseFloat(prompt("First number to add"));
let input2 = parseFloat(prompt("Second number to add"));

let sum = input1 + input2;

alert(`The sum of the two numbers is ${sum}!`);
```

Task Four (M1) – Quality of Code

Robustness

A robust piece of code is easy to understand with little code duplication and can handle unexpected inputs and conditions at the earliest opportunity. It will function well over time and will be able to be expanded upon in future if required. Regular reviews will make sure that this is the case.

```
// function to ask a user for a number
// and then return its square root

function getSqrt() {
    // ask the user for input and then try to parse it as a float
    const subject = parseFloat(prompt("Enter a number"));


    if(subject == NaN || !subject) {
        alert("Not a number!");
        // it's not a number so re-run the function
        // until the user does enter a number
        return getSqrt();
    } else if (subject < 0) {
        alert("Cannot be less than zero!");
        // it's less than zero so re-run the function
        // to ask for another number
        return getSqrt();
    } else {
        // passed the checks, run the operation
        // and do not ask for another number
        return alert(Math.sqrt(subject));
    }
}
```

An example of code that has checks for unexpected user input as well as comments to help explain certain lines.

Usability

Code usability refers to how easy the code is to understand and work with. It requires code to be clear and concise without excessive lines, functions and operations, with adequate documentation in the form of comments.

Portability

The portability of code is whether it can be ran on multiple platforms and architectures. For example, some programming languages can only be complied to binaries for certain operating systems while others are designed to be portable and write-once for all platforms. As an example, programs on macOS can be multi-architecture, supporting both Apple Silicon devices and older Intel-based Mac products.

Maintainability

Making code maintainable can be achieved by making it easy to modify in future. This can involve separating different aspects of a program into separate files and functions to allow them to be more easily inspected.

Keeping to coding conventions and standards for an organisation or the programming language itself can help make code be more readable and therefore more maintainable in the future.