Web Application Development in Python

ICT Enterprise Project

Supervisor: Catherine Woods

Submitted By: David Murphy (TL\_KCOMP\_GY6)

<<Submitted on: 28th April 2014>>

Table of Contents

[1 Introduction 1](#_Toc384923237)

[2 Requirements Specification 2](#_Toc384923238)

[2.1 Project goals 2](#_Toc384923239)

[3 Project Plan 3](#_Toc384923240)

[4 Python 5](#_Toc384923241)

[4.1 Introduction to Python 5](#_Toc384923242)

[4.1.1 Reasons for selection of Python as a programming language for this project 5](#_Toc384923243)

[4.2 History of Python 7](#_Toc384923244)

[4.2.1 Python Versions 7](#_Toc384923245)

[4.3 Python Philosophy 9](#_Toc384923246)

[4.4 Overview of Python Features 10](#_Toc384923247)

[4.4.1 Interpreted Language 10](#_Toc384923248)

[4.4.2 Multi-Paradigm 10](#_Toc384923249)

[4.4.3 Cross-Platform Usage 10](#_Toc384923250)

[4.4.4 Extensibility and Libraries 11](#_Toc384923251)

[4.5 Python Syntax and Semantics 12](#_Toc384923252)

[4.5.1 Comments 12](#_Toc384923253)

[4.5.2 Number Operations 12](#_Toc384923254)

[4.5.3 Strings 12](#_Toc384923255)

[4.5.4 Boolean operators 13](#_Toc384923256)

[4.5.5 Comparative operators 14](#_Toc384923257)

[4.5.6 Membership Operators 14](#_Toc384923258)

[4.5.7 Numeric data types 14](#_Toc384923259)

[4.5.8 Sequence types 14](#_Toc384923260)

[4.5.9 Variable usage 17](#_Toc384923261)

[4.6 Control Flow 18](#_Toc384923262)

[4.6.1 Indentation and variable scope 18](#_Toc384923263)

[4.6.2 if statements 18](#_Toc384923264)

[4.6.3 for loop 19](#_Toc384923265)

[4.6.4 while loop 20](#_Toc384923266)

[4.7 Functions 21](#_Toc384923267)

[4.7.1 Functions 21](#_Toc384923268)

[4.7.2 Methods 21](#_Toc384923269)

[4.7.3 Default Argument Values 22](#_Toc384923270)

[4.7.4 Keyword Arguments 22](#_Toc384923271)

[4.7.5 Returning values from functions 23](#_Toc384923272)

[4.7.6 Lambda functions 24](#_Toc384923273)

[4.7.7 Variable numbers of arguments to a function 24](#_Toc384923274)

[4.8 Exception Handling 26](#_Toc384923275)

[4.8.1 Try...except construct 26](#_Toc384923276)

[4.8.2 Exception types 27](#_Toc384923277)

[4.8.3 Raising exceptions and exception arguments 27](#_Toc384923278)

[4.9 Object-Oriented Programming 28](#_Toc384923279)

[4.9.1 Basic class definition and instantiation in Python 28](#_Toc384923280)

[4.9.2 Initialising and instantiating classes 28](#_Toc384923281)

[4.9.3 Data attributes 29](#_Toc384923282)

[4.9.4 Class attributes 30](#_Toc384923283)

[4.9.5 Method definition in classes 31](#_Toc384923284)

[4.9.6 Inheritance 31](#_Toc384923285)

[4.9.7 Polymorphism 32](#_Toc384923286)

[4.10 Python Library 33](#_Toc384923287)

[4.10.1 Mathematics 33](#_Toc384923288)

[4.10.2 Dates and times 33](#_Toc384923289)

[4.10.3 Data compression 33](#_Toc384923290)

[4.10.4 Testing 33](#_Toc384923291)

[4.10.5 Operating System interface 34](#_Toc384923292)

[4.10.6 Multi-threading 34](#_Toc384923293)

[4.10.7 Data analysis and manipulation 34](#_Toc384923294)

[4.10.8 Decimal floating point arithmetic 34](#_Toc384923295)

[5 Comparison of Python with Java 35](#_Toc384923296)

[6 Python Web Frameworks 36](#_Toc384923297)

[6.1 Introduction 36](#_Toc384923298)

[6.2 Python Web Frameworks 36](#_Toc384923299)

[6.2.1 Web Server Gateway Interface (WSGI) 36](#_Toc384923300)

[6.3 Flask 37](#_Toc384923301)

[6.3.1 Flask and Python extension used in this project 37](#_Toc384923302)

[7 Heroku Cloud Application Platform 39](#_Toc384923303)

[7.1 Introduction to Heroku 39](#_Toc384923304)

[7.2 Heroku Applications 39](#_Toc384923305)

[7.2.1 Web Servers 39](#_Toc384923306)

[7.2.2 Deploying to Heroku with Git 39](#_Toc384923307)

[7.2.3 Managing Heroku Applications 40](#_Toc384923308)

[7.3 Use of Heroku with Project Implementation 41](#_Toc384923309)

[8 Python/Flask Web Application 42](#_Toc384923310)

[8.1 Introduction 42](#_Toc384923311)

[8.2 Technologies 42](#_Toc384923312)

[8.3 Product Delivery 43](#_Toc384923313)

[8.4 Analysis 44](#_Toc384923314)

[8.4.1 Requirements Hierarchy Chart 44](#_Toc384923315)

[8.4.2 Requirements 44](#_Toc384923316)

[8.5 Design 46](#_Toc384923317)

[8.5.1 Initial Design Development 46](#_Toc384923318)

[8.5.2 System Architecture 47](#_Toc384923319)

[8.5.3 System sequence diagram 48](#_Toc384923320)

[8.5.4 System design narrative 49](#_Toc384923321)

[8.5.5 Implementation strategy and version control 49](#_Toc384923322)

[8.6 GitHub Repository History Information 50](#_Toc384923323)

[8.7 Implementation Increments 54](#_Toc384923324)

[8.7.1 Increment #01 54](#_Toc384923325)

[8.7.2 Increment #02 54](#_Toc384923326)

[8.7.3 Increment #03 55](#_Toc384923327)

[8.7.4 Increment #04 56](#_Toc384923328)

[8.7.5 Increment #05 57](#_Toc384923329)

[8.7.6 Increment #06 57](#_Toc384923330)

[8.7.7 Increment #07 58](#_Toc384923331)

[8.7.8 Increment #08 59](#_Toc384923332)

[8.7.9 Increment #09 59](#_Toc384923333)

[8.7.10 Final Implementation 60](#_Toc384923334)

[9 Application Testing 61](#_Toc384923335)

[9.1 Testing strategy 61](#_Toc384923336)

[9.2 Test conditions 61](#_Toc384923337)

[9.3 Unit Testing 62](#_Toc384923338)

[9.3.1 Unit Testing - Currency Conversion Client 62](#_Toc384923339)

[9.3.2 Unit Testing – Currency Conversion Web Service 63](#_Toc384923340)

[9.4 Black Box Testing and System Testing 64](#_Toc384923341)

[9.4.1 Currency Conversion Client 64](#_Toc384923342)

[9.4.2 Currency Conversion Web Service 66](#_Toc384923343)

[10 Project Conclusions 69](#_Toc384923344)

[11 References 70](#_Toc384923345)

[12 Bibliography 71](#_Toc384923346)

[13 APPENDICES 72](#_Toc384923347)

[13.1 Implementation Code 72](#_Toc384923348)

[13.1.1 Currency Converter Client 72](#_Toc384923349)

[13.1.2 Currency Converter Web Service 72](#_Toc384923350)

[13.2 Installation of Python and Flask in a VirtualEnv virtual environment 72](#_Toc384923351)

[13.3 Deployment of Python/Flask project to Heroku 72](#_Toc384923352)

[13.4 Creation of PostgreSQL Database on Heroku 72](#_Toc384923353)

REFERENCING – for citing

BIBLIOGRAPHY – other resources not cited

# Introduction

Introduction goes here.

# Requirements Specification

## Project goals

The goals of this project are as follows:

* Acquire programming skills in the Python programming language
* Acquire programming skills in a Python web framework (Flask)
* Compare and contrast Python with another high-level programming language (Java) in terms of programming paradigms, syntax, standard libraries, domain applications and web development
* Perform analysis, design, implementation and testing of a web application programmed in a Python/Flask environment. The application will consist of two parts, a Currency Conversion Client (client-side application) and a Currency Conversion Web Service (RESTful web service). The development will make use of various technologies including deployment of the web service to the cloud, transmittal of JSON data objects between client and server and connection to a database. Source control for the implementation will be maintained using Git.

# Project Plan

The following is a detailed breakdown of the tasks required to achieve the project goals outlined above:

|  |  |
| --- | --- |
| **PROJECT GOALS** | **TIMELINE - Wk1 – Wk12 (college term)** |
| **Acquire Python programming skills**  Tasks to complete:   * study official Python.org documentation * complete official Python.org tutorial * complete ‘Dive Into Python’ online course   All research conducted in this phase will be documented in this section accordingly. | This research will be carried out continuously over Wks1 – Wks6 |
| **Acquire skills in a Python web framework (Flask)**  Tasks to complete:   * read Flask official documentation * complete Flask online tutorial * complete other online tutorials   All research conducted in this phase will be documented in this section accordingly. | This research will be carried out continuously over Wks3 – Wks7 |
| **Comparison of Python with another high-level programming language (Java)**  A critical comparison of Python with Java in terms of the following headings will be presented:   * programming paradigms * features of the language * syntax and semantics * standard libraries * domain applications * web development | Work in this section will be carried out over Wk5 – Wk8 |
| **Analysis, design, implementation & testing of a web application programmed in a Python/Flask environment**  Summary of implementation  The application to be developed will allow a client-side user to log on to a web application, select from a list of currencies, input a monetary amount and perform a currency conversion using live currency rates.  When instructed, the web application will connect RESTfully to a remotely hosted web service using the required input data. The web service will perform the necessary business logic calculation and RESTfully return data to the web application for presentation to the user.  The web service could be enhanced by connecting to a database to retrieve currency data.  Technologies to be employed   * The web application and web service will be developed in Python using the Flask web micro-framework for web development. * The web service will be remotely hosted on a suitable hosting service, such as Heroku * Analysis and design of the application will include analysis of the problem and a description of the solution design, and will also include UML diagrams as required. * Source control will be achieved through Git/GitHub.   Product delivery  Due to the nature of the research into Python and Flask being carried out the application will be developed incrementally, with each increment adding further functionality and/or improvements to the application until a satisfactory solution is implemented.  Testing strategy  A testing strategy will be designed and the application tested accordingly in conjunction with each increment. | Work in this section will be carried out over Wk3 – Wk11  Analysis & design to be completed by Wk4  First increment delivered Wk4  Weekly increments thereafter until satisfactory implementation is achieved  Testing strategy to be completed by Wk4, and carried out in conjunction with the various increments  Final submission 28/04/2014 |

# Python

## Introduction to Python

Python is a powerful, high-level and dynamic programming language. It is a cross-platform interpreted object-oriented language . As a language it is extensible, allowing users to add to the language with their own source code, modules and components which can be reused in other applications.[[1]](#footnote-2)

Python supports multiple programming paradigms including object-oriented, procedural and functional styles.[[2]](#footnote-3) It is very flexible in its usage in that it is suitable for use in large-scale application development, or equally well as a scripting language.

Python promotes itself as being a very powerful, high-level language while being expressive and easy to learn. Some of its features include:[[3]](#footnote-4)

* clear, readable syntax
* intuitive object orientation
* natural expression of procedural code
* full modularity
* very high-level dynamic data types
* comprehensive standard library

Python has a vast array of domain applications, notably web development, database-driven development, GUI development, scientific applications, education, general software development and game development.[[4]](#footnote-5)

Python is developed under an approved open-source license meaning that the language is free for all to download and use. The intellectual property rights behind the Python language are owned by the Python Software Foundation. [[5]](#footnote-6) This Foundation forms the core of a wider community of developers and contributors that maintains and continually develops the language.[[6]](#footnote-7)

### Reasons for selection of Python as a programming language for this project

Python was chosen as a programming language for research and development in this project for the following reasons:

* Python is an expressive language which is relatively easy to learn
* Python is a powerful language that is suitable for many kinds of application development
* The language has a large and growing community of users, and is constantly being developed and improved by a large community of open-source contributors
* At the time of writing (March 2014) Python was ranked 8th in the TIOBE Index, which is an indicator of the popularity of programming languages. The language has grown in popularity since its introduction to the index[[7]](#footnote-8) and is currently ranked above comparable languages such as Ruby and Perl.
* For the above reasons, the author considers that it would be a great advantage to have Python programming skills as a computer science post-graduate entering the workplace. Therefore, Python is the language of choice for this project.

## History of Python

Python first emerged in the late 1980’s when Dutch programmer, Guido van Rossum was working with a language called ABC on the Amoeba operating system, which at the time was an experimental distributed operating system. ABC had similarities to BASIC or Pascal; it was a useful teaching language, it did not require variable declarations and used indentation for nesting of statements. Van Rossum liked these aspects of ABC. However, other aspects of the language were not so positive; the language had poorly structured error handling, and was monolithic in nature such that it could not be easily extended or modified.[[8]](#footnote-9)

Van Rossum decided to take the best features of ABC and incorporate them into a new language called Python. Python was designed to be very readable, using English keywords where other languages might use punctuation, and adopting an uncluttered code layout. It had fewer syntactic constructions than other languages such as C, Perl or Pascal.8

Some years later, Van Rossum and his team of programmers worked in an open-structured programming laboratory, allowing the language to be developed in new and interesting ways. The team borrowed ideas and concepts from other programming languages, eventually releasing Python version 1.0 to the programming community in 1994. Python 2.0 was released in 2000, with notable introductions to this version being the use of list comprehensions (borrowed from the Haskell programming language) and a garbage collection system.[[9]](#footnote-10)

The introduction of Version 2.0 heralded the advent of a more open, transparent and community-backed development of the language.[[10]](#footnote-11)

Because the language was developed to be deliberately simple in structure and syntax, it is easy to learn and is suitable as a first language for beginners. In spite of this, it is a very powerful language that can accomplish impressive tasks with a minimum amount of coding.8

### Python Versions

Python is currently being developed in two strands, versions 2.x and 3.x. Version 2.7.6 is the current version 2.x, while version 3.4.0 is the latest version 3.x available at the time of writing. Version 3.x has broken backward compatibility with Versions 2.x and under, and Version 2.7 will be the last major version 2.x to be released.[[11]](#footnote-12)

Version 3.x has introduced some improvements over Version 2.x. One of the major improvements is the introduction of better Unicode support (all text strings are Unicode by default). Some aspects of the core language have also been adjusted to be easier for beginners and more consistent with the rest of the language.

Python Version 2.7.6 has been chosen as the version for research and development in this project. This version is chosen over Version 3.3.4 due to the fact that Version 2.7 currently has wider usage in the Python community in general, and therefore better third-party support. While this is likely to change in the future, it is considered appropriate that, for this project, Version 2.7.6 is used. All references to code forthwith refer to Version 2.7.6.

## Python Philosophy

Python was designed to be easily extensible, having a relatively small core language, with further extended functionality being added as required. It can also be embedded in existing applications that need a programmable interface. Python philosophy rejects complicated or complex code and exuberant syntax in favour of a sparser, less-cluttered grammar. The language is intended to be fun to use, and this spirit is often reflected in the official documentation.[[12]](#footnote-13)

The endeavour towards simple code constructs, intuitive syntax and readable code has lead to the so-called *Zen of Python*, in which the language’s guiding principles have been packaged into 20 aphorisms, some of which are repeated here:[[13]](#footnote-14)

* Beautiful is better than ugly
* Explicit is better than implicit
* Simple is better than complex
* Complex is better than complicated
* Sparse is better than dense
* Readability counts
* Special cases aren’t special enough to break the rules
* There should be one—and preferably only one –obvious way to do it

## Overview of Python Features

### Interpreted Language

Python is an interpreted language. Rather than compiling the source code into machine code and executing that, Python source code is loaded directly into the Python interpreter and executed there. The interpreter converts the source code into bytecode but does not process the file at this point. Once the user requests that the interpreted code be executed, the interpreter begins to read each line of bytecode and validates it. It is at this run-time stage that any syntax errors will become apparent (as opposed to a compiled language, syntax errors become apparent at compile-time).[[14]](#footnote-15)

Python provides a REPL (read-eval-print loop), in which a user can enter Python commands and statements directly to be evaluated and executed in real-time.

### Multi-Paradigm

Python supports various programming paradigms:

* Object-oriented – Python includes such object-oriented concepts as class definition defining object state and behaviour, data members, instance variables, function overloading, instantiation of objects, inheritance and aggregation. In fact, everything in Python is an object.[[15]](#footnote-16) Extensibility and code reuse are natural by-products of the object-oriented paradigm. Python does not force the use of object-oriented features.[[16]](#footnote-17)
* Procedural – Python is useful as a procedural language. Lists of instructions can be quickly drafted and executed. Python is widely used for scripting purposes.
* Functional – Python supports functional programming, in which functions strive for side-effect free behaviour. Advantages of this programming style include formal provability, modularity, composability, and ease of debugging and testing.15

### Cross-Platform Usage

The Python interpreter has been developed for many operating platforms including Windows, Linux and OSX. Python comes pre-installed on most Linux distributions. A script written in Python on any platform will happily execute on another supported platform without any alterations to the code; the code in a Python program is the same no matter what the underlying architecture. The concept of “write once, run anywhere” is justifiably applicable to Python.

Python itself is written in C, and the maintained version of the language engine itself is known as CPython. Other implementations of Python exist, such as Jython (which is implemented in Java and compiles to Java bytecode) and IronPython (which runs on Microsoft’s CLR).

### Extensibility and Libraries

The modularity provided by object-oriented concepts allows for easy extension of functionality of the language. This is evidenced by the large array of frameworks developed by third parties for providing particular functionalities, in particular open-source web frameworks such as Django, Bottle and Flask for web application development.

In terms of general-purpose libraries, Python maintains a “batteries included” philosophy; a rich and versatile standard library is immediately available to the user, without having to download separate packages.[[17]](#footnote-18)

## Python Syntax and Semantics

As indicated by the various aphorisms in the *Zen of Python* language philosophy (see Section 4.3), Python was designed to be a very readable language. It has an uncluttered visual layout, using English keywords where other languages may use punctuation.[[18]](#footnote-19) There follows an introduction to the syntax and semantics of the language (all information in this section is adapted from the official Python documentation, located at <http://docs.python.org/2/>).

### Comments

Single-line comments are written by commencing the line with the hash (#) character. The end-of-line indicates the end of the comment.

For example:

1 # this is a comment in Python

2

3 # this is another comment

### Number Operations

Binary operators + - \* / and % (modulo) operate in the same mathematical manner as other languages. Some points should be noted regarding numbers and associated operations:

* Integer division returns a floor value. For example 7/3 returns 2.
* If one of the arguments in division is a float, then the operation returns a float. For example, 7.0/3 returns 2.3333...
* \*\* operator returns an exponential power. For example, 2\*\*5 returns 32.
* Complex numbers are supported using the suffix ‘j’ or ‘J’, as in (real + imag j), where j is the square root of -1, and ‘real’ and ‘imag’ are the real and imaginary components of the vector respectively.

### Strings

In Python, a string is a *Sequence Type.* String objects can be enclosed in single or double quotes (as in ‘String’ or “String”). Quote characters required in strings can be escaped with the ‘\’ character. Other escape sequences include \n for new line, \t for horizontal tab and \v for vertical tab. Individual strings can span multiple lines if surrounded by triple quotes (‘’’ or “””).

Strings can be concatenated with the ‘+’ operator, and repeated with the ‘\*’ operator. Strings immediately next to each other are automatically concatenated. For example, ‘ABC’ ‘xyz’ yields ‘ABCxyz’.

x = Python

x[0] *....returns* ‘P’

x[2] ....*returns* ‘t’

x[2:4] ....*returns* ‘ty’

x[3:] ....*returns* ‘hon’ *(ie, the remainder of the string from index 3)*

x[:3] *....returns* ‘Pyt’ *(ie, the string as far as index 3)*

There is no ‘character’ or ‘char’ data type in Python. A single character is represented by a string of length 1. However, characters in a string are indexed, and can be accessed using ‘slice notation’, with the first character at index 0. For example:

x[-1] *....returns* ‘n’

x[-5] *....returns* ‘y’

Negative indices are counted from the right-hand-side, starting at -1. Therefore:

x[-1] *....returns* ‘n’

x[-5] *....returns* ‘y’

It is important to note that strings are immutable. In other words, x[0] = ‘W’ would return an error. Placing a ‘u’ character before a string literal encodes the string as Unicode, for example: u’Python’.

### Boolean operators

Python includes Boolean operators as follows:

|  |  |
| --- | --- |
| Operator | Result |
| x or y | If x is false, then y, else x |
| x and y | If x is false, then x, else y |
| not x | If x is false, then True, else False |

### Comparative operators

Python provides for object identity comparison with the ‘is’ and ‘is not’ operators.

Logical operators < > <= >= == and != are similar to those used in mathematics and other languages.

### Membership Operators

Python provides membership operators. The operator ‘in’ indicates whether a given sequence contains a value. The operator ‘not in’ indicates whether a given sequence does not contain a given value.

### Numeric data types

There are four distinct numeric types in Python; plain integers (int), long integers (long), floating point numbers (float) and complex numbers (complex). Plain integers have 32-bit precision, long integers have unlimited precision. Floats are usually equivalent to a *double* in C, and precision can vary depending on the host machine. Both real and imaginary parts of a complex number are floating point values.

There are additional numeric types included in the Python standard library, namely fraction, which holds rational numbers, and decimal, which holds floating point numbers with user-definable precision.

The decimal data type is particularly important for applications which work with money and monetary values; decimal numbers can be represented exactly, while float type numbers do not have exact representations in binary floating point. The decimal data type provides the exactness required in the arithmetic involved in important financial calculations.

The built-in numeric types support various operations such as:

abs (x) returns the magnitude of x

int (x) casts value of x to an int (similar for long and float)

pow (x, y) returns x to the power of y

### Sequence types

There are various sequence types in Python, among the most common of which are strings, lists and tuples.

#### Lists

The most versatile sequence type is the list. A list is a set of comma separated values surrounded by square brackets. Items in a list do not have to be of the same data type. Similarly to strings, list item indices start at 0, and can be sliced and concatenated. For example:

mylist = [‘dog’, ‘cat’, 1234, 500]

mylist[3] ....*returns* 500

mylist[-2] ....*returns* 1234

mylist[1:-1] ....*returns* [‘cat’, 1234]

mylist[:2] + [‘giraffe’, 3\*4] ....*returns* [‘dog’, ‘cat’, ‘giraffe’, 12]

mylist[:] ....*returns* [‘dog’, ‘cat’, 1234, 500]

It is possible to work with values of individual items in a list. For example:

mylist[3] = mylist[3] + 250 ....*returns* [‘dog’, ‘cat’, 1234, 750]

Items can be replaced, inserted or removed as follows:

mylist[0:2] = [1, 10] ....*returns* [1, 10, 1234, 750]

mylist[0:2] = [] ....*returns* [1234, 750]

mylist[1:1] = [‘xyz’, ‘pqr’] ....*returns* [1234, ‘xyz’, ‘pqr’, 750]

List length can be determined using the len() function:

len(mylist) ....*returns* 4

Lists can be nested. For example:

q = [2,3]

p = [1, q, 4] ....*returns* [1, [2, 3], 4]

Items can be appended to lists with the append() method:

p.append(5) ....*returns* [1, [2, 3], 4, 5]

q.append(‘xtra’) ....*returns*  [1, [2, 3, ‘xtra’], 4, 5]

#### Tuples

A tuple is a series of comma separated values, usually surrounded by parenthesis (although parenthesis are not required if the tuple assignment is the only statement on that line of code). Tuples are immutable, though it is possible to assign values to tuples which are mutable, such as lists.

Examples of tuples include:

tuple1 = (a, b, c) tuple of variable names

tuple2 = (‘a’, ‘b’, ‘c’) tuple of strings

tuple3 = (1, 2, [‘a’, ‘b’, ‘c’], 3) tuple of numbers and a list

#### Dictionaries

A dictionary is an unordered collection of key-value pairs. The key can be any immutable data type, such as a strings, numbers or tuples containing only immutable types. Lists cannot be used as keys. Keys must be unique within a dictionary, and each key references a value. The value can be any data type, including strings, numbers, lists, tuples or other dictionaries.

Dictionaries are created using curly braces {}. A pair of curly braces {} on its own will create an empty dictionary. Dictionary data can be initialised by inserting key:value pairs separated by commas within the braces, as follows:

dict1 = {‘id1’: 1000, ‘id2’: 1001, ‘id3’: 1002}

dict2 = {‘person1’: ‘John’, ‘person2’: ‘Mary’}

dict3 = {‘person2’: {‘name’: ‘Jack’, ‘age’: 23}, ‘person3’: [‘Jane’, 35]}

In the third dictionary, dict3, the value in the first key-value pair is another dictionary, while the value in the second key-value pair is a list.

The main operations associated with a dictionary are storing a value with an associated key, and retrieving a value associated with a particular key. Using the above examples, the following show how to manipulate dictionaries:

dict1[‘id4’] = 1003 #insert a value to dict1

print dict1 ....{‘id1’: 1000, ‘id2’:1001, ‘id3’: 1002, ‘id4’: 1003}

print dict1[‘id1’] ....1000

The keys() method of a dictionary object returns a list of all the keys used in that dictionary in arbitrary order:

print dict1.keys() ....[‘id1’, ‘id2’, ‘id3’, ‘id4’]

Dictionaries can also be created using the dict() constructor, which takes as an argument a sequence of key-value pairs in tuple format.

### Variable usage

Python is not a “strongly typed language”, in that variables do not have specific types which they can only be assigned to for the life of the execution.[[19]](#footnote-20) Variables do not have to be declared; a variable exists as soon as its name is referred to in the code. Variables are not assigned to a type, so a variable name can happily refer to different object types throughout its scope.

Naming convention in Python dictates that variable names should be all in lower case, with individual words separated by the underscore (\_) character. It must begin with a letter or underscore, and the name is case-sensitive. Values are assigned with the assignment (=) operator.

Some examples:

x = 23

y = x + 4

print y ....*returns* 27

x = ‘hello’

y = ‘ there’

print x + y ....*returns* ‘hello there’

In this example, the variable x and y are assigned to integers initially, and then to strings without causing any difficulty.

Python caters for multiple assignment, as in the following example where a and b are assigned the values 1 and 2:

a, b = 1, 2

Variables can be defined as global with the ‘global’ keyword. Global variables have scope throughout the code.

## Control Flow

### Indentation and variable scope

In Python, code blocks are controlled using indentation. There are no curly braces ({}) required to define code blocks. End-of-line is determined by carriage returns, rather than semi-colons found in other languages.

The following is a simple example:

x = 10

if x == 10:

y = ‘y has local scope within indented block’

print y

This will print the value of y (‘y has local scope within indented block’).

However, the following will generate an error:

x = 10

if x == 10:

print x

y = ‘y has no scope outside the indented block’

print y ....error

y has local scope only and cannot be used unassigned outside the indented block.

### if statements

If-statements must be followed immediately by a colon (:) and an indented code block. elif is a keyword used for else-if. For example:

x = 4

if x < 0:

print ‘x is negative’

elif x == 0:

print ‘x is zero’

else:

print ‘x is positive’

The above example obviously prints ‘x is positive’.

### for loop

Unlike other languages, the Python for-statement iterates over the items of any sequence types (such as a list or string) in the order that they appear in the sequence.

For example:

languages = [‘Python’, ‘Java’, ‘Ruby’]

for my\_item in languages:

print my\_item, len(my\_item)

This will return the following output:

Python 6

Java 4

Ruby 4

If a sequence needs to be modified as a for-loop iterates over it, slice notation can be used to create a copy of the sequence for iteration, as follows:

for my\_item in languages[:]:

if len(my\_item) > 5:

languages.insert(0, my\_item)

print my\_item

Here the for-loop iterates over a copy of languages, and inserts any item with length > 5 at index 0, as follows:

[‘Python’, ‘Python’, ‘Java’, ‘Ruby’]

If a for-loop is needed over an arithmetic progression, the range() function can be used:

range(6) ....*returns* [0, 1, 2, 3, 4, 5]

range(5, 10) ...*.returns* [5, 6, 7, 8, 9]

range (0, 10, 3) ....*returns* [0, 3, 6, 9]

Combined with a for-loop:

for i in range(10):

print i

The break and continue statements can be used to break out of a loop and continue with the next iteration respectively.

### while loop

The while-loop is executed similarly to the for-loop, as follows:

a = 1

while a < 10:

print a,

a = a + 1

The comma in the print statement prevents the newline return. The following is the output expected:

1 2 3 4 5 6 7 8 9

## Functions

### Functions

Functions are defined in Python with the ‘def’ keyword, an end-of-statement colon (:) and indentation to define the code block. For example, the following function definition to print a Fibonacci series from the python.org tutorial nicely demonstrates a function definition:

def fib(n):

a, b = 0, 1

while a < n:

print a,

a, b = b, a + b

A sample call of the function, fib(50) will print the following output:

0 1 1 2 3 5 8 13 21 34

‘Docstrings’ can be included as the first statement of a function, and must be a string literal describing the function surrounded by triple-quotes (‘’’docstring’’’). This docstring is used for automatically generating documentation (similar to Javadoc in Java).

Global variables cannot be directly assigned a value within a function, but they can be referenced.

Arguments are passed to a function using the ‘call-by-value’ method. This means that the actual parameter passed to the formal parameter in the function is always an *object reference* rather than the actual value of the object.

A function definition creates a new entry in the program *symbol table*, which is a table of all variables used in the program. Therefore, a function name is essentially a variable name, and can be assigned to a new name as required. For example, taking the fib() function above:

new\_fib = fib

Now, when the functinon new\_fib(50) is called we get the same output as when we called fib(50).

The ‘return’ keyword returns a value similar to other languages.

### Methods

In Python, a *method* is a function that ‘belongs’ to an object, and is called using dot notation, as in obj.methodname(). For example, if we have a list object ‘my\_list’, we can call the append() method as follows:

my\_list.append(‘an example’)

which appends the argument to the list.

### Default Argument Values

In function definition, default values can be specified for one or more formal parameters. This allows the function to be called using some or all of these arguments. When arguments are not passed, the default value in the formal parameter is assigned to the argument. Mandatory arguments must appear first in the parameter list.

For example, take the following function definition[[20]](#footnote-21):

def ask\_ok(prompt, retries=4, complaint=’Yes or no, please’):

while True:

ok = raw\_input(prompt)

if ok in ('y', 'ye', 'yes'):

return True

if ok in ('n', 'no', 'nop', 'nope'):

return False

retries = retries – 1

if retries < 0:

raise IOError('refusenik user')

print complaint

This function can be called in several ways:

* Provide the mandatory argument (prompt) :

ask\_ok(‘Do you want to quit?’)

* Provide one of the optional arguments:

ask\_ok(‘Do you really want to quit?’, 3)

* Provide all the arguments:

ask\_ok(‘Overwrite file?’, 2, ‘Must be yes or no!’)

### Keyword Arguments

Python allows for the sending of a keyword name with an argument to a function. For example, take the following function:

def say\_hello(a=’hello’, b=’there’):

print a + b

Now, we can call the function in the following ways:

say\_hello()

....*returns* ‘hello there’

say\_hello(a=’hi’)

....*returns* ‘hi there’

say\_hello(b=’hello’)

....*returns* ‘hello hello’

say\_hello(b=’again’, a=’hi’)

....*returns* ‘hi again’

Keyword arguments must follow positional arguments.

Functions can also be passed as arguments to another function.

### Returning values from functions

Single values are returned from functions in Python using the return keyword, similar to other languages. In addition, Python functions can return multiple values of varying data types. For example, the following function will return the argument value, the value squared and the value cubed:

def multi\_return(x):

return x, x\*x, x\*x\*x

The following function will return multiple data types (numeric and string):

def another\_multi\_return(a,b):

a=21

b=’Python’

return a, b

What happens in reality within the function is that a tuple with the required returned values is created within the function, and then unpacked into its component parts on exiting the function.[[21]](#footnote-22) This is seamlessly carried out behind the scenes, and the values are neatly returned to the user individually.

### Lambda functions

Lambda functions can be thought of as small anonymous functions. Typically, a lambda function is implemented to do something very trivial[[22]](#footnote-23), such as multiplying a number or modifying a string. The syntax of a lambda function is as follows:

name = lambda (arguments) : code

The following are some simple examples of lambda functions:

get\_sum = lambda (a, b) : a + b

get\_sum(2,3) ....*returns* 5

get\_square = lambda (a, b) : a \*\* b

get\_square(5, 2) ....*returns* 25

### Variable numbers of arguments to a function

Python allows for the coding of a function that accepts a variable number of arguments. This is useful if the number of arguments at the time of function definition is unknown. A single asterisk ‘\*’ is placed in front of the parameter name in order to achieve this. Consider the following function:[[23]](#footnote-24)

def average( \*args ):

total = 0

count = 0

for i in args :

total = total + i

count = count + 1

return total / count

The function calculates the average of all values passed to it. Thus, the following calls to the function can be made:

average(1, 3) ....*returns* 4

average(1, 2, 3) ....*returns* 2

average(5, 4, 3, 2, 1) ....*returns* 3

## Exception Handling

### Try...except construct

Exception handling is easily accomplished in Python by means of the try...except...else...finally construct, similar to try...catch constructs found in other languages.

The basic syntax of a try...except construct is as follows:[[24]](#footnote-25)

try:

# some python statements

except [ExceptionName]:

# some error handling if exception raised

else:

# some other statement if no exceptions are raised

finally:

# do something when all is done in any case

The try clause and one except clause are required as a minimum in an exception handler. Any number of except clauses can be included, and the else and finally clauses are optional.

The following example[[25]](#footnote-26) takes user input from the console using the raw\_input() function, and demonstrates the use of multiple except clauses:

try:

v = int( raw\_input("Enter a value: "))

print "We got some valid input!"

except ValueError:

print "Invalid input, please enter a value"

except KeyboardInterrupt:

print "Please don't hit ctrl-c"

Here, the ValueError and KeyboardInterrupt exceptions are handled separately. Alternatively, multiple exceptions can be handled in the same except clause:

try:

v = int( raw\_input("Enter a value: "))

print "We got some valid input!"

except (ValueError, KeyboardInterrupt):

print "Invalid input, or Keyboard Interrupt"

### Exception types

There are a number of standard exception types provided in Python. Some of these include:

* Exception the base for all other exception types in Python
* FloatingPointError error in processing floating point value
* MemoryError raised when system runs out of memory while running
* NameError raised when a name cannot be found
* ValueError value assigned to a given type is not valid
* ZeroDivisionError raised when division by zero occurs

User-defined exceptions can also be defined in Python.

### Raising exceptions and exception arguments

Exceptions are easily raised in Python using the raise keyword. For example, to raise a ValueError the following could be coded:[[26]](#footnote-27)

def myFunction( a, b, c ) :

if a < 0 or a > 10 :

raise ValueError, "a must be between 0 and 10"

if b > 50 :

raise ValueError, "b must be less than 50"

return a\*b + c

This simple example raises a ValueError if the various user-defined conditions are not met. The exception is also passed a single string argument with a useful user-defined error message which is presented to the user in a traceback if the exception is raised.

## Object-Oriented Programming

Python is a fully-object oriented language. Classes are easily defined, can inherit from other user-defined or library classes and are easily instantiated.[[27]](#footnote-28) The object-oriented concepts of abstraction, inheritance, polymorphism and encapsulation are supported by the language.

### Basic class definition and instantiation in Python

The basic structure of a class in Python is as follows:

class MyClass:

# class definition here

A class is defined using the reserved keyword class, followed by the class name. By convention, class names begin with a capital letter, with the leading letter in each word in the name also capitalised.[[28]](#footnote-29) Similarly to function definitions, the code within a class definition should be indented.

To create an instance of a class a simple reference to the class is made as follows:

instance = MyClass()

### Initialising and instantiating classes

Classes are initialised using the special \_\_init\_\_ method; this method is called immediately after the creation of an instance of the class. The method is used to define and initialise data attributes of an instance of the class. This method is similar to a ‘constructor’ in other languages, but it is not in fact a constructor; a reference to an instance of the class already exists when the initialisation method is called. However, in practice it fulfils much the same role as a constructor.[[29]](#footnote-30) It is not a requirement to include an initialisation method in a class definition.

The syntax for the \_\_init\_\_ method is shown in the following example:

class MyClass:

def \_\_init\_\_(self, value1 = 5):

self.x = value1

self.y = 10

There are a number of interesting aspects to this example code. Firstly, in class method headers, the first argument is always named self. self refers to the newly created instance in the initialisation method, and to the instance whose method was called in other class methods. When calling a method on an object however, it is not necessary to specify the self argument, as Python will add it automatically.[[30]](#footnote-31)

The initialisation method here shows a second argument, value1, given a default value of 5. In this way, the initialisation of a class can vary depending on the arguments passed in the object instantiation. Initialisation methods can take any number of arguments.

Two data attributes are defined in the method, x and y. The self keyword is again used to represent the instance of the class concerned, and data attributes, functions and methods in a class definition must use the self keyword in a similar fashion. The following are simple examples of creating instances of the class:

instance = MyClass()

print instance.x ....5

print instance.y ....10

another\_instance = MyClass(8)

print another\_instance.x ....8

print another\_instance.y ....10

The second object created passes a value of 8 for the value1 argument, and the initialisation function initialises the attribute x with this value. However, the first object does not pass any initialisation data and so x is set to the default value of 5.

### Data attributes

The previous example demonstrated the use of data attributes (known as ‘instance variables’ in other languages). By default, these attributes are accessible and mutable; data attributes of an instance are accessed by using dot notation in conjunction with the instance reference. Alternatively, Python provides two methods, getattr() and setattr() which act as accessor and mutator functions on data attributes. An example of accessing and modifying data attributes, using the previous class definition, is as follows:

object = MyClass()

print a.x ....5

object.x = 100

print a.x ....100

object.z = ‘hello’

print a.z ....’hello’

It should be noted here that we have added a third attribute to our object ‘object’ with a reference ‘z’. From here onwards, this object will have a data attribute with reference z. However, this data attribute an attribute of this instance only, and not of the class.[[31]](#footnote-32)

What is interesting about data attributes in Python is that, unlike other object-oriented languages, by default they are not hidden from outside the class. This is in keeping with the Python philosophy of “trusting the user”, and as such assumes that the programmer best understands their code and how to handle instance attributes.[[32]](#footnote-33)

However, Python does provide for true encapsulation by allowing for the definition of private attributes, functions and methods in a class. A private element in Python is declared as such by its name; if a method, function or data attribute is pre-pended with two underscores, then they are only visible within their own class (for example, ‘\_\_an\_attribute’ is an attribute visible only within its own class). Accessor and mutator methods would then be required to work with private attributes outside the class.

### Class attributes

Class attributes (known as static attributes in other languages) are defined within a class scope but outside of methods or initialisation functions. Take the following example:

class MyClass:

my\_class\_attr = ‘some value’

def \_\_init\_\_(self, value1 = 5):

self.x = value1

self.y = 10

In this class definition, the class attribute my\_class\_attr is defined, and given the string value ‘some value’. This attribute is common to all instances of the class, and can be accessed using dot notation on either the class name or the instance reference, as follows:

Print MyClass.my\_class\_attr ....’some value’

a = MyClass()

print a.my\_class\_attr .... ’some value’

Similarly to other languages, it is good practice to access class attributes via their respective class name, rather than on an instance of the class.

### Method definition in classes

Methods are defined in a class definition similarly to defining functions using the def keyword. In fact, a method can be thought of as a function that belongs to an object. As previously mentioned, the first parameter in a method definition is always self, with further parameters included as required. The following is a simple example showing a method definition in a class, and the instantiation and calling of a method on the instance:

class AnotherClass:

def square(self, number):

number\_squared = number \* number

return number\_squared

....

instance = AnotherClass()

print instance.square(4) ....16

### Inheritance

Inheritance is easily achieved in Python; the base class is simply passed as a parameter in the class definition header. All attributes and methods of the base class are inherited by the derived class. This is best demonstrated by example. Take the earlier example class definition, MyClass; a second derived class, MyDerivedClass is created as follows:

class MyClass:

def \_\_init\_\_(self, value1 = 5):

self.x = value1

self.y = 10

class MyDerivedClass(MyClass):

def \_\_init\_\_(self):

MyClass.\_\_init\_\_(self)

self.z = 20

The most important point to note here is that when defining a derived class, the \_\_init\_\_ method of the derived class must explicitly call the \_\_init\_\_ method of the base class (in this case MyClass.\_\_init\_\_(self)). If this is not called, the data attributes and associated initialisation will not be inherited and an exception may be thrown at runtime if the inherited data attribute is required.[[33]](#footnote-34)

The derived class can be used as follows:

object = MyDerivedClass()

print object.x ....5

print object.y ....10

print object.x + object.z ....25

Python also supports a limited form multiple inheritance. This is achieved through passing multiple parameters in the class definition header.[[34]](#footnote-35)

### Polymorphism

Python fully supports polymorphism. Behaviours of methods called on objects depend on the type of object on which the method is called. Similarly to other languages, the appropriate code for the given object type is executed at runtime when the type of object is known.

## Python Library

Python has a “batteries included” philosophy.[[35]](#footnote-36) It provides a rich and varied library containing many modules enabling the user to accomplish all kinds of programming tasks. The following is a small sample of the modules available with a brief description of each, in no particular order.

### Mathematics

The math module provides functions for carrying out many mathematical operations such as power and logarithmic functions, trigonometric functions (sin, cos and tan, etc), angular conversion and mathematical constants.

### Dates and times

The datetime module in Python “supplies classes for manipulating dates and times in both simple and complex ways”.[[36]](#footnote-37) The module provides functions for date and time arithmetic and for output formatting and manipulation. The module is useful for dealing with date or time stamps provided as data by other sources, such as database management systems.

The datetime module is used in the project implementation discussed later in this document.

### Data compression

Data compression file formats are easily handled in Python with the zlib module.[[37]](#footnote-38) The module provides tools to easily create and decompress common compression file formats such as zip files and tarballs.

### Testing

The unittest module (also known as PyUnit) is a Python version of the well-known Java unit testing library, JUnit. The module provides a framework for performing automated unit tests on Python functions, methods and classes. The framework provides shared setup and teardown functions for automated tests and the aggregation of tests into test suites.[[38]](#footnote-39)

The unittest module is used is used for unit testing of code in the project implementation discussed later in this document.

### Operating System interface

The os module “provides dozens of functions for interacting with the operating system”.[[39]](#footnote-40) The module provides functions for reading and writing files, creating file objects, working with paths and directories and accessing system information.

### Multi-threading

The threading module provides functionality allowing for multi-threading, in which tasks which are not sequentially dependent within a process can be executed simultaneously.

Use is made of the threading module in the project implementation discussed later in this document.

### Data analysis and manipulation

Python provides numerous modules which provide tools for aiding data analysis and manipulation. The re module provides “regular expression tools for advanced string processing”.[[40]](#footnote-41) The struct module provides functions for working with binary data record layouts.[[41]](#footnote-42) Other modules, such as array and collections, provide very useful tools for working with and analysing data collections such as arrays, lists and dictionaries.

### Decimal floating point arithmetic

The decimal module provides a Decimal datatype for decimal floating point arithmetic. The Decimal datatype is particularly important for financial applications which require exact decimal representation that binary floating point arithmetic cannot provide. The datatype and associated module also provide control over precision, rounding and tracking of significant decimal places.[[42]](#footnote-43)

The Decimal datatype is used in financial monetary calculations in the project implementation discussed later in this document.

# Comparison of Python with Java

# Python Web Frameworks

## Introduction

A web framework can be defined as “a collection of packages or modules which allow developers to write web applications or services without having to handle such low-level details as protocols, sockets or process/thread management.”[[43]](#footnote-44)

A typical web framework provides tools that allow the developer to quickly build web-based applications, resources and services. Most frameworks provide libraries for functions such as web security, session management, database access and mapping, web templates, URL mapping and tools for creating web services.[[44]](#footnote-45)

A web framework allows a developer to concentrate on building the necessary logic required in their application without having to worry about writing low-level code to deal with communications and associated infrastructure. The framework provides the scaffold around which a web application can be successfully built.

Most web frameworks are based on the *model-view-controller* (MVC) pattern. This is an architectural pattern that aims to separate the data model with business rules from the user interface. This pattern is generally considered good practice due to its modularity, promoting code reuse and allowing multiple interfaces to be applied.27

## Python Web Frameworks

There are numerous web frameworks available for the Python language. Popular “batteries included”- type frameworks include Django, TurboGears and web2py. These frameworks typically provide libraries for most or all features that may be required in a web application, such as a template engine, database mapping and session management.

There are numerous other “micro-frameworks” such as Flask, web.py and Bottle that provide minimal core web application functions. These frameworks typically provide a limited library of functions to build a basic web application. There are typically many third-party extensions available for these frameworks providing various functionalities such that the developer can extend the core framework library as the need dictates.

### Web Server Gateway Interface (WSGI)

The Web Server Gateway Interface (WSGI) is “a specification  for simple and universal [interface](http://en.wikipedia.org/wiki/Interface_(computer_science)) between [web servers](http://en.wikipedia.org/wiki/Web_server) and [web applications](http://en.wikipedia.org/wiki/Web_application) or [frameworks](http://en.wikipedia.org/wiki/Web_application_framework) for the [Python programming language](http://en.wikipedia.org/wiki/Python_(programming_language))”. [[45]](#footnote-46) It is a standard for Python web application development, and was developed to act as a low-level interface between web servers and web applications providing a framework-independent common vocabulary for client-server communication.

All of the Python web frameworks mentioned in the previous section support the WSGI interface.

## Flask

Flask is the web framework of choice for use in this project. This has been chosen in conjunction with the author’s work placement enterprise.

Flask is a micro-framework which “aims to keep the core simple but extensible”[[46]](#footnote-47). It provides a template engine and development web server out of the box. Additional functionality such as a database abstraction layer or form validation is not provided by default; such features are available if required through third-party extensions.

Flask itself is built on the Jinja2 template engine and Werkzeug WSGI utility library

Flask has been chosen for use in this project for the following reasons:

* The micro-framework is considered suitable while Python is concurrently being learned. The core framework allows for simple applications to be quickly developed, with increased complexity added as knowledge of the language increases.
* The framework provides support for RESTful request dispatching, which is one of the features of the proposed implementation in this project.
* The framework provides support for secure client-side sessions, which is another aim of the proposed implementation in this project.
* The framework provides integrated unit testing support.
* The framework is well documented, and is well supported amongst the Python community considering the many extensions to the framework available.[[47]](#footnote-48)

### Flask and Python extension used in this project

The following extensions to the Flask framework Python standard libraries are used in this project:

* Flask-WTF – provides integration of Flask and WTForms, which itself provides convenient form-building capabilities and form validation in Python applications.
* Flask-SQLAlchemy – integrates Flask and SQLAlchemy, which is itself a Python SQL toolkit and Object Relational Mapper. The extension enables a Python application to integrate with a database in a database-independent syntax.
* Requests – a Python extension which allows for easy HTTP request generation, including get/post methods, custom headers and HTTP user authorisation parameters.
* ConfigObj – a Python extension allowing a user to easily place application-specific configuration data in a configuration file for use in a Python application.
* Eventlet / Gunicorn – lightweight Python web servers used for server-side application deployment in this project. Gunicorn is used when deploying from a Ubuntu platform while Eventlet is used when deploying from a Windows platform.
* Flask-Testing – a Flask extension which builds on Python’s native unittest unit testing framework to provide unit testing facilities for a Flask application.
* Blinker – a Python extension which provides fast and simple object-to-object and broadcast signaling for Python objects. This extension is required for performing certain unit tests using Flask-Testing.
* Pytz – a Python extension which allows for accurate timezone calculations and configuration, including daylight saving.

# Heroku Cloud Application Platform

## Introduction to Heroku

Heroku is a cloud platform as a service (PaaS which supports several programming languages. The platform provides services that allow a user to deploy, run and manage web applications. Supported languages include Ruby, Node.js, Java, Python, Clojure and Scala.[[48]](#footnote-49)

## Heroku Applications

With reference to the languages listed in Section 7.1, in Heroku, an application is:

*a collection of source code written in one of these languages, perhaps a framework, and some dependency description that instructs a build system as to which additional dependencies are needed in order to build and run the application.*

In Python, application dependencies are specified by the inclusion of a text file, *requirements.txt,* which lists the dependencies required in order for the application to execute. In general, the source code along with the specified dependencies are all that is required to be deployed to Heroku in order to successfully build and run an application.

### Web Servers

An application hosted on Heroku must specify the appropriate web server to use to host the application. There are many web servers available for Python applications, and suitable web servers include Gunicorn (Unix-based systems only), Gevent and Eventlet. [[49]](#footnote-50) These servers are embedded in the application when deployed.

### Deploying to Heroku with Git

Applications are deployed to Heroku using Git, the popular version control system widely used by software developers. Git provides developers with a remote repository for code and other documentation, and provides version control by providing ‘snapshots’ of a project every time a new or updated file is pushed to the remote repository. A Git repository is securely associated with a local repository on a contributor’s local machine.

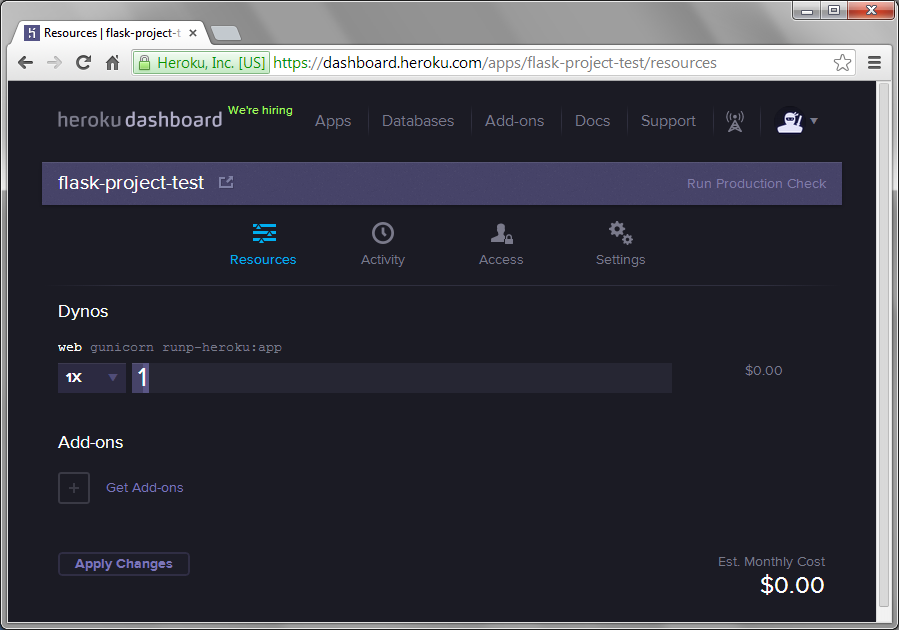
When an application is created on Heroku, a new Git remote repository is associated with the local repository of the application. When the application is pushed to the remote repository for the first time, Heroku commences the build process for the application. The build process involves firstly installing all dependencies listed in the application requirements, and creating any other assets required by the application. Once these are installed, the application is executed.

### Managing Heroku Applications

Heroku executes applications on *dynos*, which are defined as “isolated, secure, virtualised Unix containers that provide the environment to run the application”.[[50]](#footnote-51)

Dynos can easily be scaled in Heroku. A newly created application will automatically be operating on one dyno. As web traffic increases or more processing power required, the dyno count can be scaled upwards accordingly. Heroku provides 750 free dyno-hours per month per application, thus allowing a single-dyno application to run continuously for free. Additional usage incurs charges.

Dyno scaling and other application management functions can be carried out at the command-line, or alternatively by using the GUI Heroku dashboard tool.



**Figure 7.xx – Heroku dashboard showing application running at 1x dyno**

By default, Heroku assigns a random name to projects. The application is hosted at http://<project\_name>.herokuapp.com. The project name can be changed by the user and user URL’s can be assigned to applications as required.

## Use of Heroku with Project Implementation

Heroku will be used in the implementation part of this project. It is intended to create a RESTful web service, implemented in Python using the Flask web framework, and this will be deployed to Heroku in the final implementation. A locally hosted web application will connect with the web service as required.

# Python/Flask Web Application

## Introduction

This section describes the analysis, design, implementation and testing of a web application developed in the Python programming language using the Flask “microframework” for web development.

The implementation was designed such that it would make use of the technologies studied as part of this project.

The web application to be developed is a currency conversion application, and is to be developed in two parts: the Currency Conversion Client and Currency Conversion Web Service.

The Currency Conversion Client will allow a user to securely log in through a browser to a web application, hosted locally. The web application will present a selection of currencies to the user, and allow a user to input a monetary amount, with the user selecting the “from” currency and the “to” currency. The user submits the information, a call is made to the Currency Conversion Web Service.

The Currency Conversion Web Service will act as remotely hosted RESTful web service, exposing services to provide live currency exchange rates and currency conversions for available currencies. The web service will carry out the necessary business logic and return the resulting data to the Currency Conversion Client for presentation to the user.

The design for this project has been kept deliberately simple in order to create a suitable working product commensurate with the new technologies explored earlier. It is considered that the proposed implementation will demonstrate a significant knowledge of Python application programming, Python/Flask web application development and remote API/web service deployment to a suitable host in the cloud. The focus is not on what the system does, but rather on *how* it does it.

## Technologies

The web application will employ the following technologies in development:

* client-side web application: to be developed in Python employing the Flask web framework
* server-side web service: to be developed in Python employing the Flask web framework. The web service is to comply with the requirements of REST (REpresentational State Transfer) architectural principles
* remote hosting: the web service will be hosted on the Heroku Cloud Application Platform. Heroku provides support for Python applications developed with the Flask web framework
* Git/GitHub: a GitHub remote repository will be used for the duration of the project. Source code and documentation will be pushed to the repository such that a full history of incremental development will be recorded. Heroku also requires the use of Git and GitHub for the deployment of applications
* Sublime Text 2 has been chosen as a simple and lightweight text editor for the development of Python source code for this project. This is deliberately chosen in lieu of more advanced IDE’s because it is felt that such an approach is appropriate while learning a new language; there is no language-specific intellisense bundled with Sublime Text, and so the user is forced to consider the syntax carefully as it is being written
* The application will be developed on a Windows 7 platform, with Python scripts executed using the Python interpreter from Windows Powershell

## Product Delivery

The web application will be developed incrementally. It is considered that this is an appropriate method of delivering the final product, given the concurrent research into the various technologies that is taking place.

The first implementation will be a very simple version of the application, locally hosted, performing minimal calculation. It is intended that increments will be delivered approximately weekly, with each increment adding further functionality/usability/style until a satisfactory solution is implemented which achieves all of the requirements detailed in the following section.

## Analysis

### Requirements Hierarchy Chart

### Requirements

The requirements of Currency Conversion Client are identified as follows:

* Currency Conversion Client will allow a new user to create a user account for logging into the application
* Currency Conversion Client will allow a previously registered user to securely log in to the application
* Currency Conversion Client will carry out a currency conversion on a specified amount between currencies of the user’s choice
* Currency Conversion Client will allow a user to logout of the application

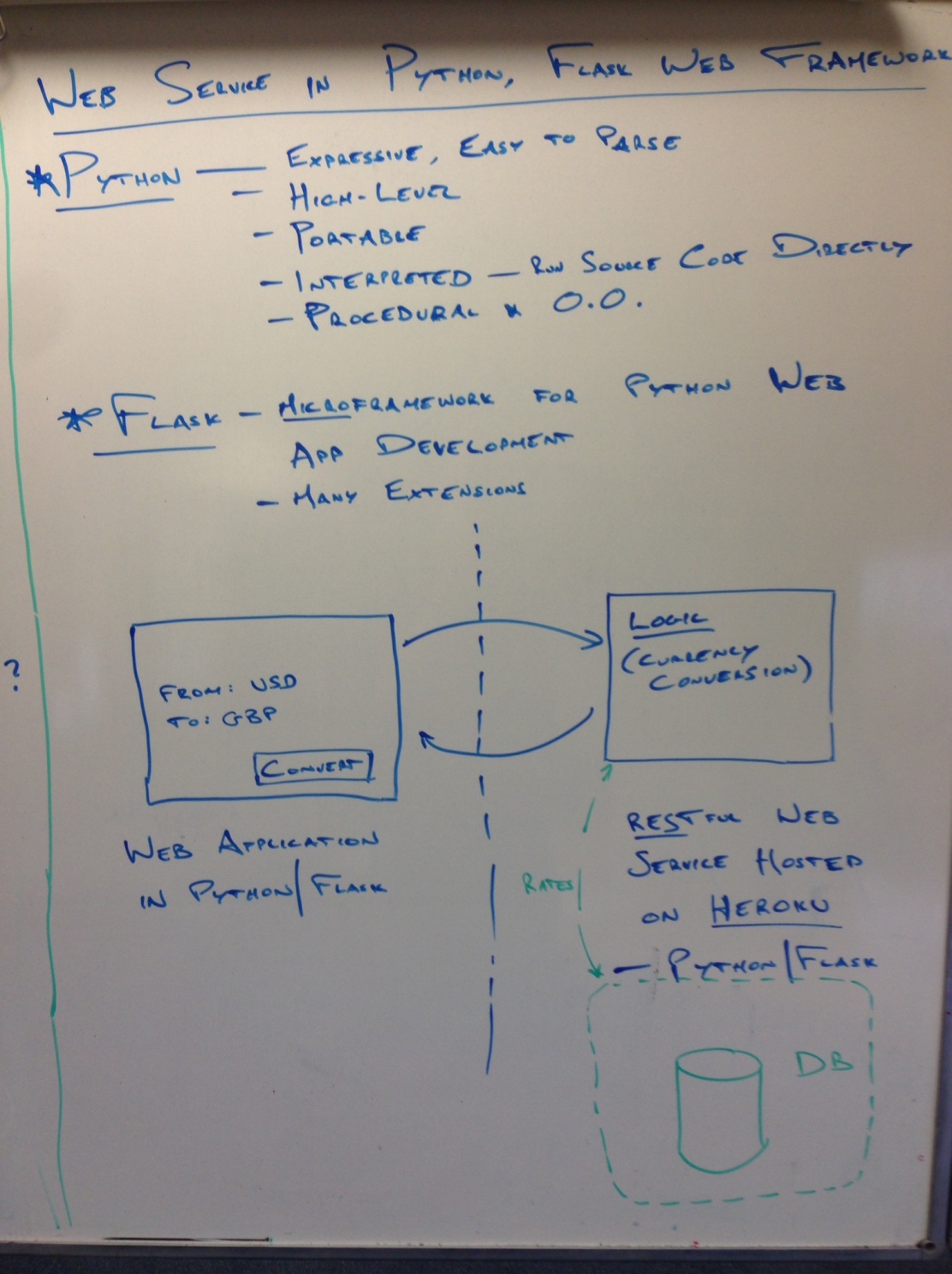
The requirements of Currency Conversion Web Service are as follows:

* Currency Conversion Web Service will expose a service to return a list of available currencies to a client
* Currency Conversion Web Service will expose a service to perform a currency conversion using live exchange rates

## Design

### Initial Design Development

Figure 8.xx following is a snapshot of the original design developed in the author’s work placement enterprise. The diagram outlines the system architecture for both Currency Conversion Client and Currency Conversion Web Application.



**Figure 8.xx – initial design**

The diagram shows the separation of Currency Conversion Client (web application in Python/Flask) and Currency Conversion Web Service (RESTful web service hosted on Heroku), with flow of data via HTTP between them.

This diagram is expanded upon in the next Section, which presents in detail the system architecture for the implementation.

### System Architecture

**Currency Conversion Web Service**

**Currency Conversion Client**

HOSTED ON localhost

HOSTED ON **HEROKU**

HTTP

**Client-side web app**

From currency: \_\_\_\_\_\_

To currency: \_\_\_\_\_\_

Amount: \_\_\_\_\_\_

**RESTful web service**

*/getcurrencies*

provide list of available currencies

*/convert*

currency conversion

CONVERT

HTTP

Currency

rates

User email and

password

Database

Database

Currency Conversion Client

* Developed in Python/Flask
* Hosted locally
* Accessed through a browser
* Supply and receive data to and from web service in JSON format (HTTP methods GET / POST)
* User login required

Currency Conversion Web Service

* Developed in Python/Flask
* Hosted on Heroku Cloud Application Platform
* Supply and transmit data in JSON format to any interested client
* HTTP basic authorisation required to access exposed API services

### System sequence diagram

The following is a sequence diagram for the system:

### System design narrative

The design shows the separation of the client-side web application and the remotely hosted web service. The design is explained as follows:

* Currency Conversion Client, developed in Python using the Flask web framework, will be accessed by a registered user through a browser. The application allows the user to select a “from” currency from a list, select a “to” currency from a list and enter a monetary amount to be converted. The text fields presented to the user must only accept a valid monetary amount (i.e. cannot be negative, must be numeric)
* The data is validated and posted RESTfully over HTTP to Currency Conversion Web Service.
* Currency Conversion Web Service will be hosted on the Heroku Cloud Application Platform. The service will accept posted data from the web application and perform the currency conversion. Currency exchange rates for use in the calculation will be downloaded periodically from a live currency feed. The converted amount will be posted RESTfully over HTTP to the client.
* Currency Conversion Client will display the converted amount to the user.
* A database could be used on the server-side to store currency exchange rates. A database could be used on the client side to store user registration data. The use of databases will be explored as development progresses.

### Implementation strategy and version control

The application will be developed incrementally. Development will commence with a simple implementation, and features and functionality will be improved on an ongoing basis for the duration of the project.

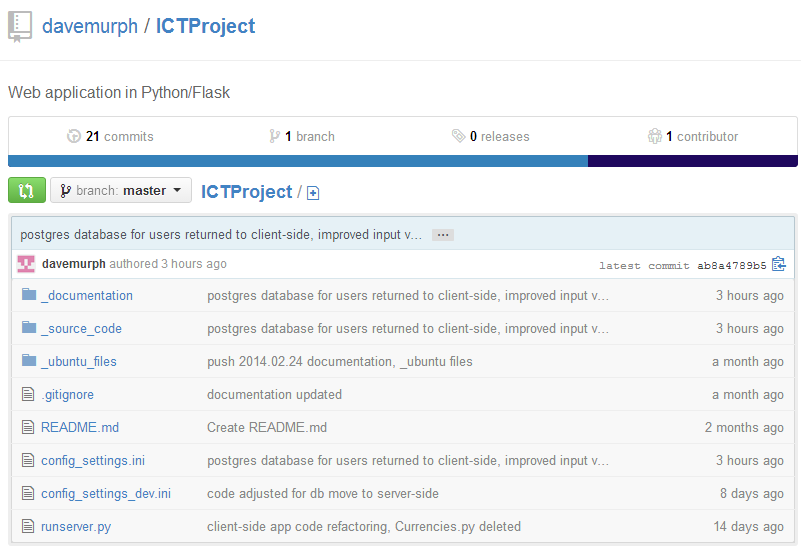
Version control is to be used during the development process. Git and GitHub will be used for maintaining version control. Incremental code and documentation development will be regularly pushed to a Git repository, allowing full historical version control for the application development. The GitHub repository history will demonstrate a continuous work output for the duration of the project and application development.

## GitHub Repository History Information

GitHub maintains a complete history of repository activity over the course of application development. Information on each commit is recorded and provides a useful indicator of progress through the project from start to finish. GitHub provides useful metrics that graphically show the number of contributors and contributions to a project over time, commit activity over time, additions and deletions over time and time/date commit activity.

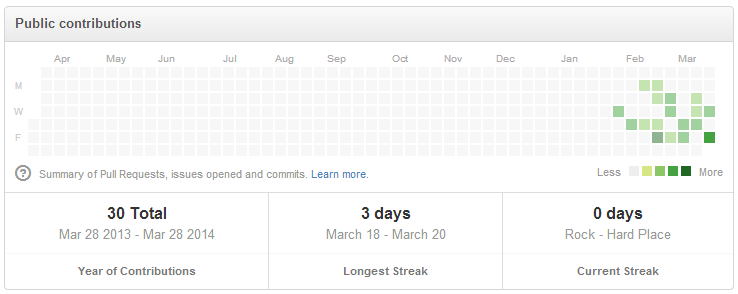
The full list of commits to a project is available, with each commit providing a ‘snapshot’ of the project at the time of the commit. Any commit can be checked-out in order to revisit the application and associated code.

Figure 8.xx following shows the GitHub repository for Currency Conversion Client (repository name ICTProject):



**Figure 8.xx – GitHub repository for Currency Conversion Client**

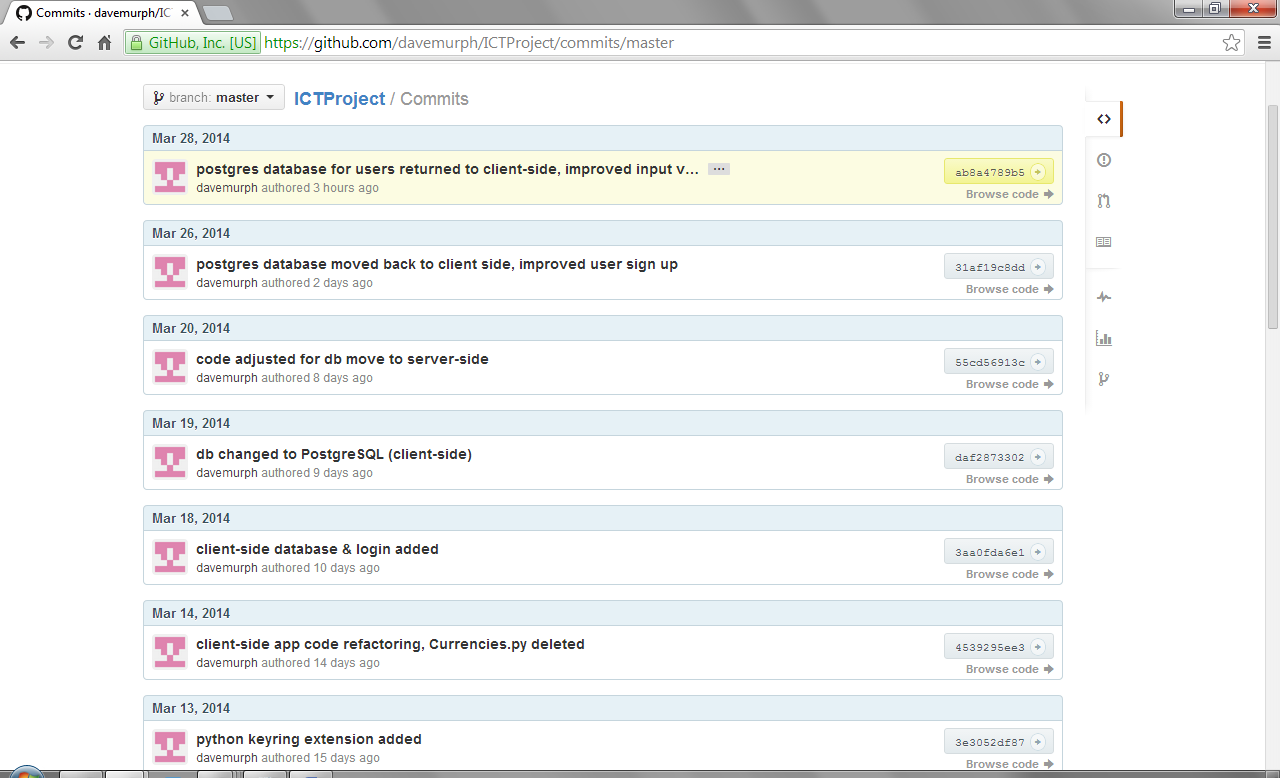
Figure 8.xx following shows a calendar with project contributions to all projects indicated. Note contributions in February/March during the application development.



**Figure 8.xx – contributions to all GitHub repositories over time**

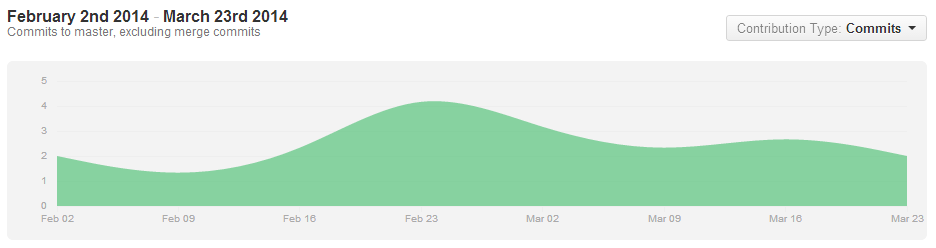
#### Project commit history

Figures 8.xx following is a screenshot from Github showing a sample of commits to the Currency Conversion Client code repository. The history demonstrates regular pushes to the repository, with appropriate description messages given for each commit.



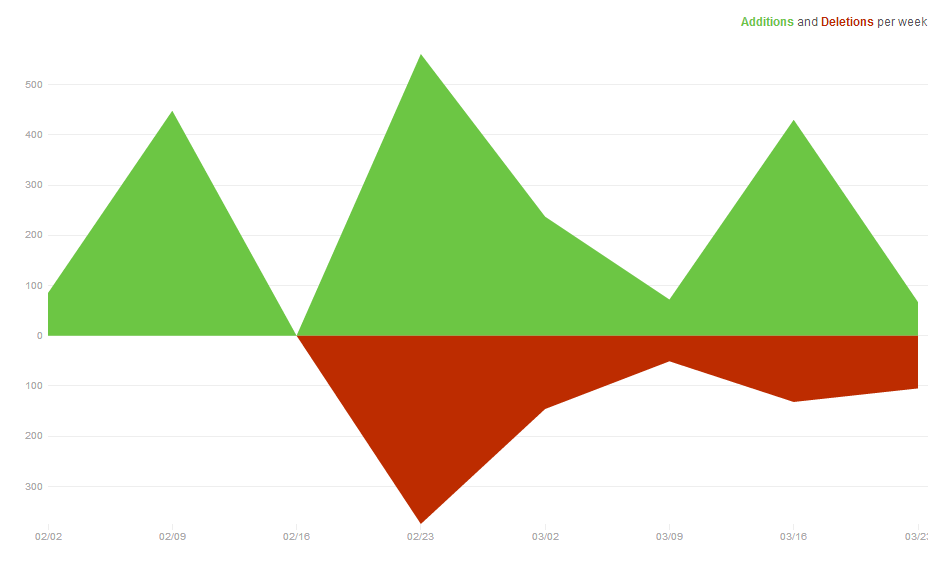
**Figure 8.xx – Currency Conversion Client commit history**

Figure 8.xx following shows a graph of commit activity over time for the project duration.

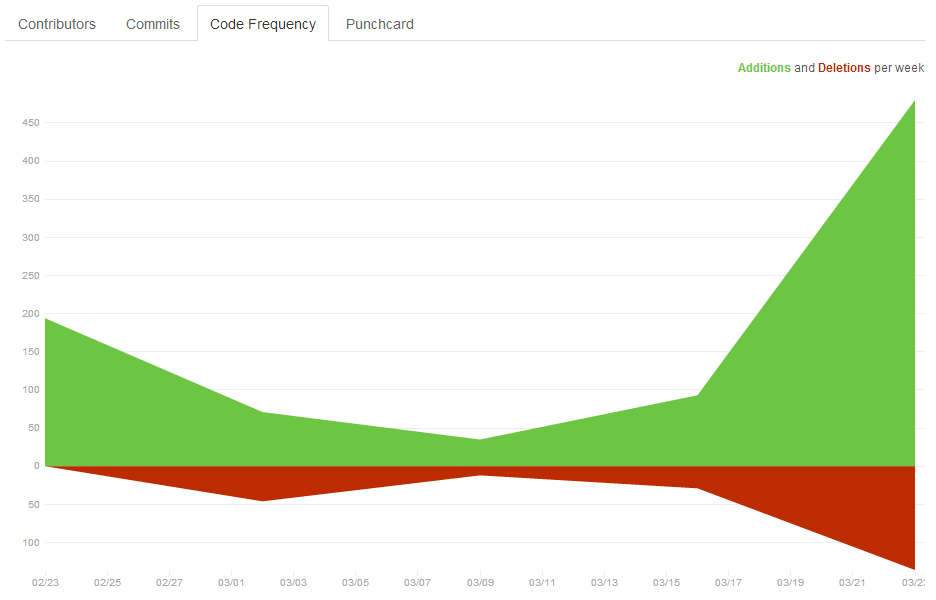


**Figure 8.xx – Currency Conversion Client commit activity**

Figures 8.xx and 8.xx following graphically indicate code frequency, showing code additions and deletions over time. Interestingly, code additions increase in the Currency Conversion Web Service repository relative to the Client towards the end of the project.



**Figure 8.xx – Currency Conversion Client code frequency**

****

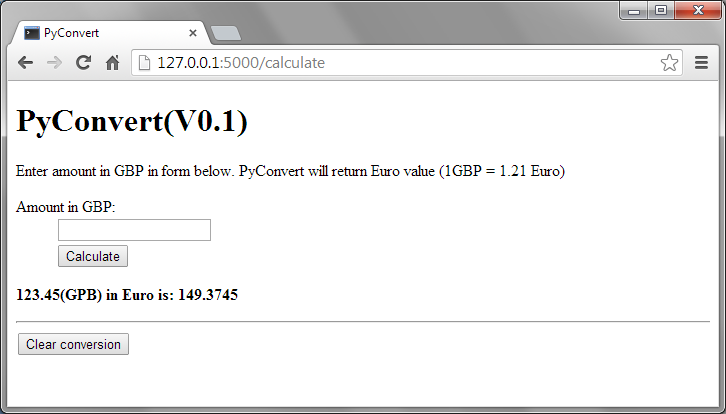
**Figure 8.xx – Currency Conversion Web Service code frequency**

## Implementation Increments

### Increment #01

The first increment of the application involved developing a simple self-contained locally-hosted web application. The application requests the user to input a monetary amount. A nominal currency conversion is carried out and the result displayed to the user. If the user enters an invalid (non-numeric) value, an error message is displayed and the calculation is not carried out. The user can click a button to clear the current conversion if they so wish.

The code makes use of the core features of the Flask framework, including view functions, template rendering and the Flask object, of which the application is an instance. A sample screenshot is provided in Figure 8.xx below.

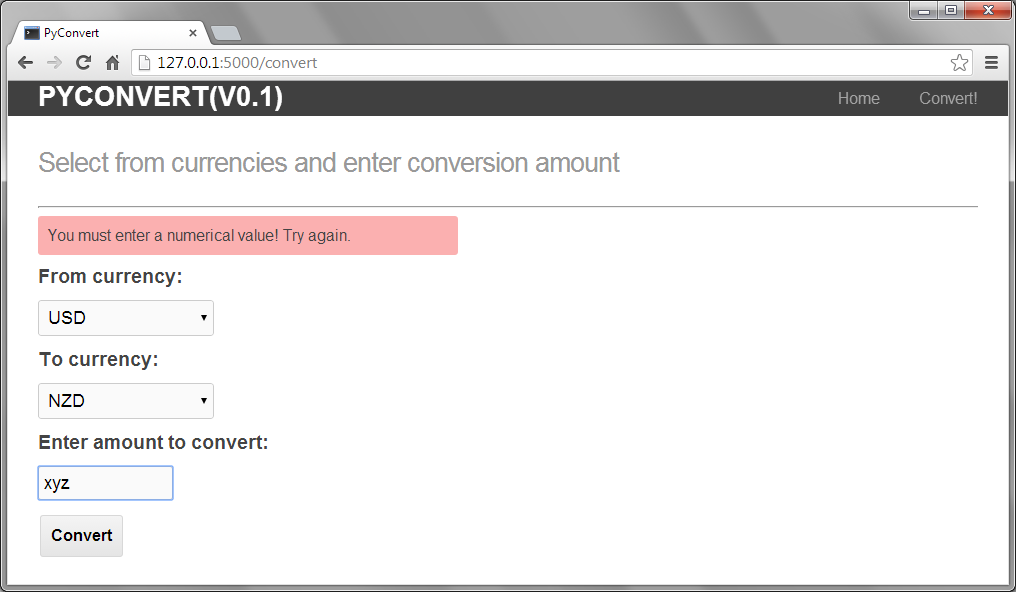


**Figure 8.xx – Increment #01 sample execution**

### Increment #02

The code from the first increment was re-written in order to move towards separating presentation from the business logic in an effort to move towards the *Model-View-Controller* software pattern. In addition, the project directory was reorganised to mirror that recommended in the Flask documentation [[51]](#footnote-52)(which is that of a Python module).

The application now allows the user to select a ‘from’ currency, ‘to’ currency and an amount to be converted; conversions are carried out using hard-coded exchange rates. If invalid input is entered appropriate error messages are displayed using Flask’s message flashing feature. CSS styling was also added to the application. The application makes use of the Flask-WTForms extension for building the necessary web forms. A sample screenshot is provided in Figure 8.xx below.



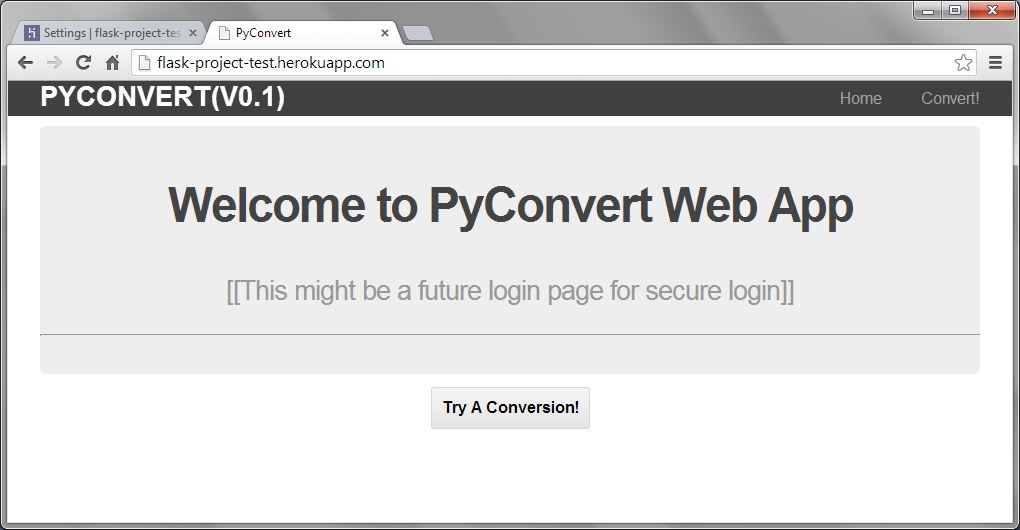
**Figure 8.xx – Increment #02 sample execution**

### Increment #03

The goal of the third increment in the development of the application was to successfully deploy a sample python application to the Heroku cloud application platform. One of the goals of the implementation is to host a RESTful web service on Heroku and as such it is considered appropriate to try to get a sample application running on Heroku prior to developing the web service (information on deploying a Python application to Heroku is presented in Appendix ??).

A simple ‘Hello World’ web application written in Python/Flask was initially deployed as an exercise. The currency conversion application in its current state was then successfully deployed. In order to use the Unix-only Gunicorn web server used in Heroku’s online tutorial the application was deployed from a Ubuntu virtual machine hosted on Windows (using Oracle VirtualBox).

A sample execution of the application running on the Heroku platform is shown in Figure 8.xx below; note the URL of the application with the herokuapp.com suffix.



**Figure 8.xx – execution of application on Heroku platform**

### Increment #04

The aim of this increment was to create the RESTful web service (Currency Conversion Web Service) that would interact with the client-side web application over HTTP. It was decided to encode all data in JSON format to comply with REST requirements.

The web service was tested using cURL (again on a Linux/Ubuntu platform), which is a command-line tool for transferring data using various protocols, and in this case HTTP methods. cURL allows the user to construct HTTP methods (e.g. GET, POST, PUT, DELETE etc) and specify the data format, associated parameters, user authentication data, etc.

The web service exposes two resources, /getcurrencies and /convert, which return a list of available currencies and perform a currency conversion respectively. Exchange rates are hard-coded in the application. The resources employ HTTP Basic authentication requiring a subscriber to validate themselves with a username and password.

### Increment #05

The main aim of Increment #05 was to modify the client-side web application to call into the exposed services of Currency Conversion Web Service, thus creating the Currency Conversion Client.

The client-side application was modified and improved to ensure robust interfacing with the web service. Additional functionality was developed to allow for displaying currency labels in addition to currency codes.

Functionality of Currency Conversion WebService was improved to include periodic connection to a live currency feed to download up-to-date currencies in JSON format for use in currency conversions. The application spawns a separate thread for managing the currency feed. In addition, the web service code was moved to its own application folder and an associated repository created on GitHub.

A sample execution of the code is presented in Figure 8.xx below.

****

**Figure 8.xx – currency conversion result with live rates, Increment #05**

### Increment #06

Increment #06 made some minor changes to the code, including refactoring of the client-side application code. Use is made of configuration files and deployment of the web service from a Windows environment.

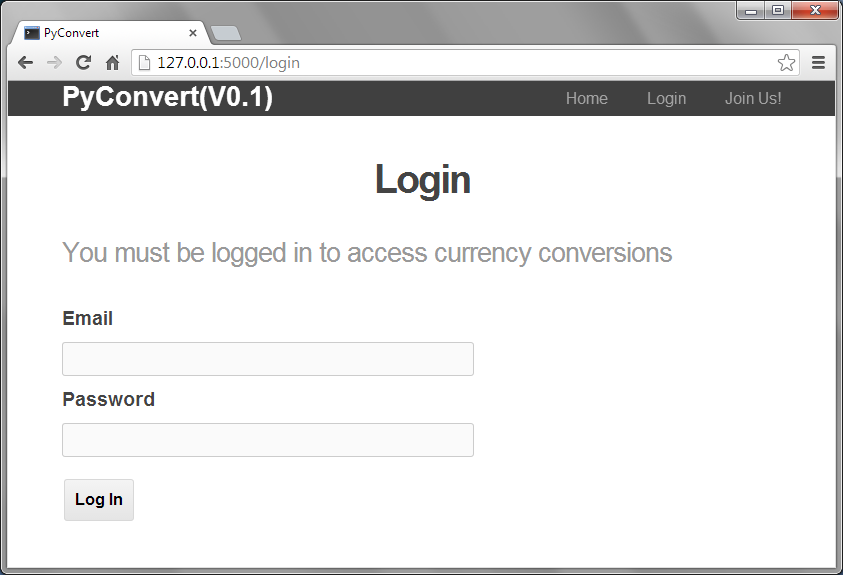
Configuration data (including HTTP Basic authentication username and password details and web service URLs) is now included for both applications in separate configuration files. In order to deploy Currency Conversion Web Service to Heroku from a Windows environment, use is made of an alternative web server, Eventlet.

### Increment #07

This increment involved adding some database connectivity to the project. Database functionality is easily included in a Flask project through use of the Flask-SQLAlchemy extension. User login and registration was introduced to Currency Converter Client, with a user’s email address and hashed password stored in a local database. Access to the web application is only allowed if the user successfully authenticates, and a session cookie is stored in the user’s browser.

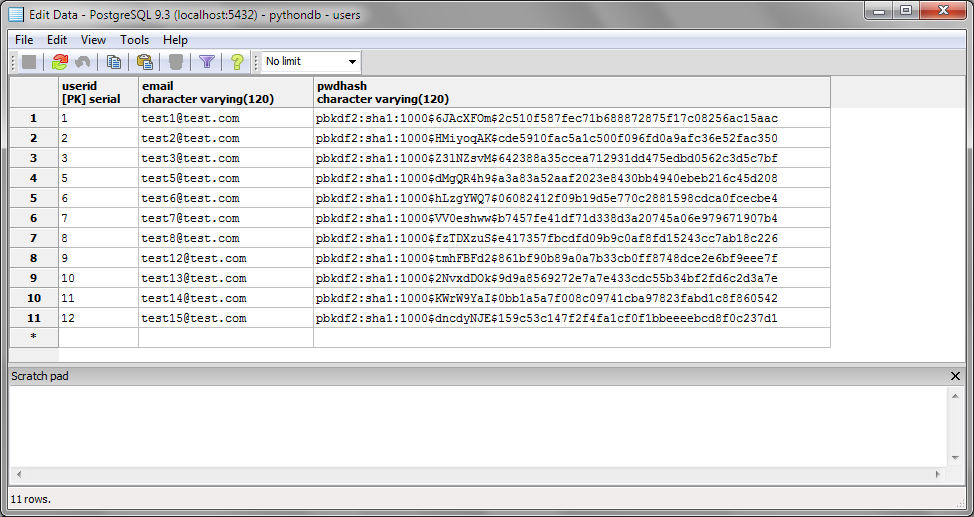
The application was first set up to connect with an SQLite database. Once the application was successfully connecting with SQLite, the SQLite database was swapped out for a PostgreSQL database with little or no difficulty.

Figure 8.xx below shows the login page of Currency Converter Client.



**Figure 8.xx – Currency Converter Client login page**

Figure 8.xx below presents the ‘users’ table in the PostgreSQL database, showing unique user ID, user email address and user hashed and salted password.



**Figure 8.xx – local PostgreSQL database table ‘users’**

### Increment #08

Increment #08 involved introducing database connectivity to Currency Converter Web Service. PostgreSQL was used as this is the recommended database for use on Heroku. Data models were created for ‘rates’ and ‘subscribers’ to the service, and a database was created to store this data.

Rather than storing live currency rates in memory the web service writes exchange rate data to a database when a live rates are received. When a call is made to the service from a client, the service connects to the database and retrieves the relevant data required for the client call. In addition, username and hashed password of clients who subscribe to the web service are stored in the database.

An additional resource, /addsubscriber was also created in order to easily register subscribers on the system. An administration password is required in order to register a new subscriber; this password is stored in the web service configuration file. Subscriber’s username and password is used to authenticate the client for the exposed services.

Once the web service was successfully running locally and connecting with the database, a PostgreSQL database was created on Heroku and the web service successfully deployed.

### Increment #09

The final increment in the development introduced unit testing to the project. The native unittest module (also known as ‘PyUnit’) in addition to a Flask extension, Flask-testing, were used for unit testing both the client and web service. The Flask-testing extension provides functions for easily creating test-clients for a given Flask application instance. A temporary in-memory SQLite database is configured for client unit testing such that it is not necessary to alter a development or production database.

A typical testing class inherits from the Flask-testing TestCase class. The class defines setUp() and teardown() methods which execute before and after each test, and a special method, create\_app() must also be defined in order to configure and return an instance of the Flask application. Individual tests are defined as methods, and a test method name *must* begin with the word ‘test’ in order for PyUnit to recognise it as a unit test.

Unit tests were successfully executed using HTMLTestRunner, a freeware utility which executes PyUnit test suites and creates presentable HTML test result output.

Unit testing is discussed in further detail in Section 9.0.

### Final Implementation

The final implementation of the project has met the requirements and generally conformed to the architectural design set out earlier in this chapter.

Currency Converter Client runs locally and stores user accounts information in a locally hosted PostgreSQL database. Users can sign up for an account, and log in and out of the system to access the live currency conversion feature. User password data is stored hashed and salted. A session is created for a logged in user, storing a cookie in the user’s browser while the user is logged in. The currency conversion page interfaces over HTTP with Currency Conversion Web Client.

Currency Conversion Web Service is hosted on Heroku. The application stores authorised subscriber information in a PostgreSQL database, also hosted on Heroku. The exposed services are available at the following locations:

* <http://currency-converter-api.herokuapp.com/currencyapi/getcurrencies>
  + returns a list of available currencies for currency conversion
* <http://currency-converter-api.herokuapp.com/currencyapi/convert>
  + performs a live currency conversion for given currency code and monetary amount in JSON format. HTTP request parameters to be in the form of the following example:

{“from\_currency”:”EUR”, “to\_currency”:”GBP”, “amount”:”50.99”}

A subscriber to the web service must be authorised over HTTP Basic authorisation to access the service. An administration resource is available for adding subscribers to the database. An administration password is required to add a subscriber to the database. This administration password is stored in the application configuration file. The administration resource is available at:

* <http://currency-converter-api.herokuapp.com/currencyapi/addsubscriber>

# Application Testing

## Testing strategy

The testing plan for the project is developed incrementally as functionality is added to the application. However, as an overall testing strategy the following is to be adopted from the outset:

* Unit testing – unit testing of client-side and server-side modules using an appropriate Python unit testing tool is to be carried out. Unit testing and results are to be appropriately documented.
* Black box testing – both Currency Conversion Client and Currency Conversion Web Service will be tested in isolation. Currency Conversion Client will be tested with a browser using appropriate input values. Currency Conversion Web Service will be tested from the command line with cURL using appropriate input values. All tests will be compared with expected behaviour.
* System testing – the integrated system will be fully tested. All user input fields are to be tested and results compared with expected behaviour.

In all cases on this project, the agile approach of ‘test-early, test-often’ is assumed during application development.

## Test conditions

The following conditions are to be tested in Currency Conversion Client:

* Numerical fields should be tested for in-range values, boundary values, just out-of-bounds values, out-of-range values, negative values, alphabetical values, non-alpha-numeric values, etc.
* String fields should be tested for empty strings, maximum string length, etc.
* Email fields should be tested for correct email format

The following conditions are to be tested in Currency Conversion Web Service:

* All exposed resources are to be tested for appropriate responses with the following combination parameters:
* Valid and invalid HTTP authorisation
* Valid and invalid JSON request parameters

## Unit Testing

Unit testing is carried out using the Flask-testing extension which provides unit testing facilities for Flask web applications. The extension is built upon unittest (the unit testing framework provided in Python’s standard library), and unittest is used to drive the tests. Use is also made of a test runner, HTMLTestRunner, to automatically generate a HTML test results document which collates formatted test results.

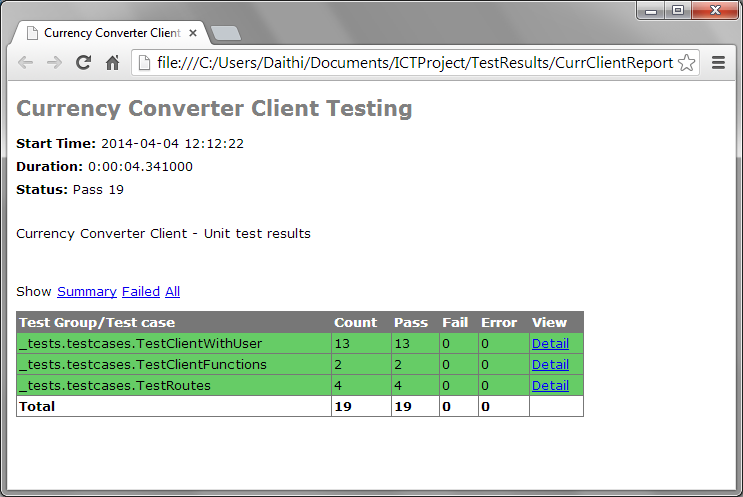
### Unit Testing - Currency Conversion Client

Unit tests were executed on the URLs in the application that interface with the client database. Tests were carried out in which automatic HTTP ‘POST’ and ‘GET’ methods were executed to test user login, logout and signup, using valid and invalid email and password details.

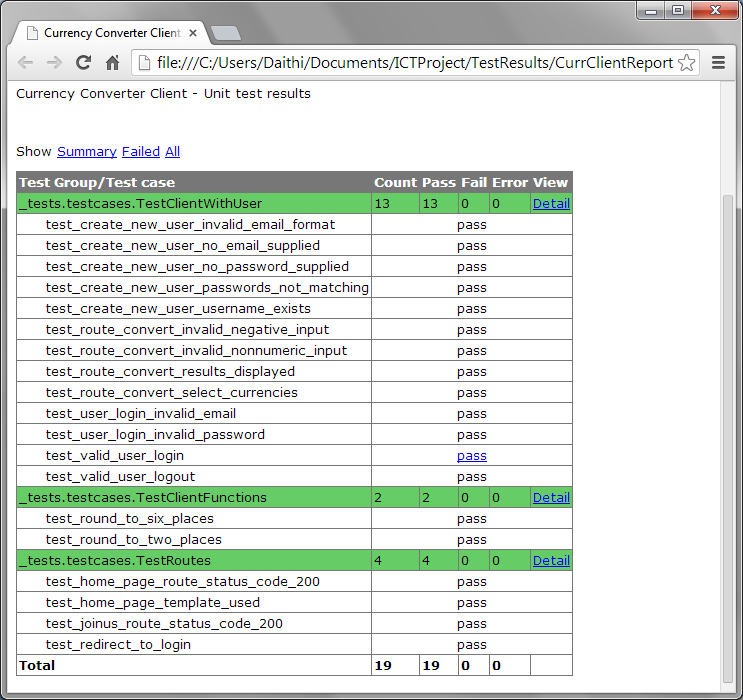
Unit tests were also carried out on the view function that interfaces with the web service. Again, automated HTTP requests were made using valid and invalid numeric conversion amounts.

Standalone functions were tested for correct output. Some assert methods particular to the Flask-testing extension were used, such as assert\_template\_used(), which tests if a particular html template is rendered for a particular route, and assert\_200(), which checks that a HTTP response to a particular request has a status code of 200.

Unit testing results output is presented in Figures 9.xx and 9.xx following. The client test script is provided in Appendix ??.



**Figure 9.xx – Currency Converter Client unit test, summary results output**

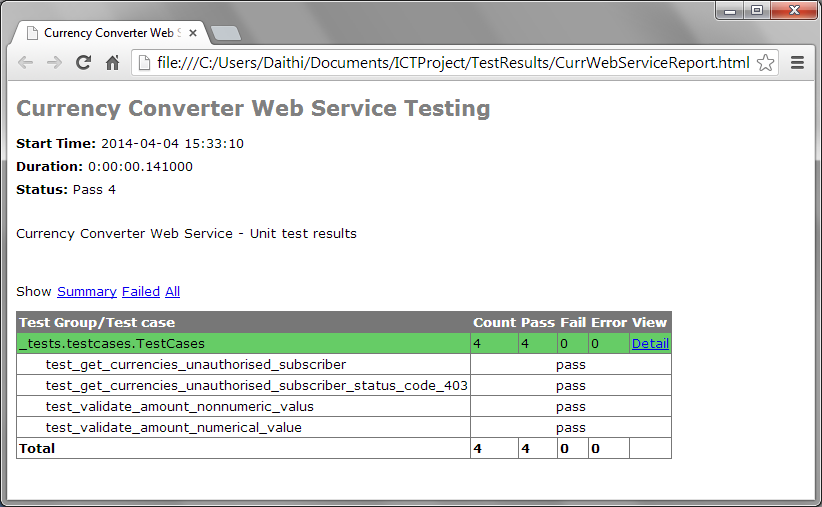
****

**Figure 9.xx – Currency Converter Client unit test, detailed results output**

### Unit Testing – Currency Conversion Web Service

Limited unit testing was carried out on Currency Conversion Web Service; tests included testing exposed resources for an unauthorised user, and testing the function which validates an incoming numerical amount.

Unit testing results output is presented in Figure 9.xx following. The web service test script is provided in Appendix ??.



**Figure 9.xx – Currency Converter Web Service unit test, detailed results output**

## Black Box Testing and System Testing

### Currency Conversion Client

During development, the following test conditions were identified and tested on the client. Tests were manually carried out as appropriate during each developmental increment. Tables 9.xx, 9.xx and 9.xx following present the test conditions identified for black box and system testing during development.

**Table 9.xx – test conditions for ‘/convert’ route, Currency Converter Client**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Ref | Route | User Input | Test Value | Expected Result |
| 1.01 | /convert | Conversion amount | Valid numerical amount (123.45) | Currency conversion |
| 1.02 | /convert | Conversion amount | Non-numerical value (‘xyz’) | Error message |
| 1.03 | /convert | Conversion amount | Empty string (“”) | Error message |
| 1.04 | /convert | Conversion amount | Negative amount  (-123.45) | Error message |

**Table 9.xx – test conditions for ‘/joinus’ route, Currency Converter Client**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Ref | Route | User Input | Test Value | Expected Result |
| 2.01 | /joinus | Email, password & confirm password | Valid email format (‘test@test.com’) & password (x2) | User created in database |
| 2.02 | /joinus | Email, password & confirm password | No email entered, password entered (x2) | Error message |
| 2.03 | /joinus | Email, password & confirm password | Valid email format (‘test1@test.com’), no password entered | Error message |
| 2.04 | /joinus | Email, password & confirm password | Invalid email format (‘test’), password entered (x2) | Error message |
| 2.05 | /joinus | Email, password & confirm password | Valid email format (‘test1@test.com’), password entered not confirmed | Error message |
| 2.06 | /joinus | Email, password & confirm password | Valid email format (‘test1@test.com’), 2x passwords entered but not matching | Error message |
| 2.07 | /joinus | Email, password & confirm password | Existing user email (‘test@test.com’) & password (x2) | Error message |

**Table 9.xx – test conditions for ‘/login’ route, Currency Converter Client**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Ref | Route | User Input | Test Value | Expected Result |
| 3.01 | /login | Email & password | Existing user email (‘test@test.com’) & correct password | User logged in |
| 3.02 | /login | Email & password | Unknown user email (‘xyz@test.com’) & some password | Error message |
| 3.03 | /login | Email & password | Invalid email format (‘test’) and some password | Error message |
| 3.04 | /login | Email & password | No email supplied, some password entered | Error message |
| 3.05 | /login | Email & password | Existing user email (‘test@test.com’) & no password | Error message |
| 3.06 | /login | Email & password | Existing user email (‘test@test.com’) & incorrect password | Error message |

### Currency Conversion Web Service

During development, the following test cases were identified and tested on the web service. Tests were manually carried out on a Ubuntu platform using cURL to generate HTTP requests to the exposed services with appropriate header data, HTTP Basic authentication and JSON parameters.

A sample cURL test command, constructed for test case 4.01 below, would be as follows:

$ curl –u username:password –i –H “Content-Type: application/json” –X POST –d ‘{“from\_currency”:”GBP”, “to\_currency”:”NZD”, “amount”:100.00}’ http://localhost:8000/testapi/convert

The command generates a POST request (marked by the –X switch) with appropriate JSON parameters (marked by the –d switch). HTTP Basic authentication credentials are marked with the –u switch, and the request headers are marked with the –H switch.

Table 9.xx following presents the test cases identified for testing the web service with cURL.

**Table 9.xx – test cases for Currency Converter Web Service, tested with cURL**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Ref | Route | Request parameters | HTTP Method | Request status code & expected result |
| 4.01 | /convert | Username: valid\_username  Password: valid\_pass  from\_currency: ‘GBP’  to\_currency: ‘NZD’  amount: ‘100.00’ | POST | 200 OK  Converted amount & unit rate |
| 4.02 | /convert | Username: valid\_username  Password: valid\_pass  from\_currency: ‘GBP’  to\_currency: ‘NZD’  amount: ‘xyz’ | POST | 400 Bad request |
| 4.03 | /convert | Username: valid\_username  Password: valid\_pass  from\_currency: ‘GBP’  to\_currency: NULL  amount: 100.00 | POST | 400 Bad request |
| 4.04 | /convert | Username: valid\_username  Password: valid\_pass  from\_currency: NULL  to\_currency: ‘USD’  amount: 100.00 | POST | 400 Bad request |
| 4.05 | /convert | Username: valid\_username  Password: valid\_pass  from\_currency: ‘CAD’  to\_currency: ‘USD’  amount: 100.00 | POST | 400 Bad request |
| 4.06 | /convert | Username: valid\_username  Password: invalid\_pass  from\_currency: ‘CAD’  to\_currency: ‘USD’  amount: ‘100.00’ | POST | 403 Forbidden |
| 4.07 | /convert | Username: invalid\_username  Password: valid\_pass  from\_currency: ‘CAD’  to\_currency: ‘USD’  amount: ‘100.00’ | POST | 403 Forbidden |

Table 9.xx cont...

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Ref | Route | Request parameters | HTTP Method | Request status code & expected result |
| 4.08 | /convert | Username: NULL  Password: NULL  from\_currency: ‘GBP’  to\_currency: ‘NZD’  amount: ‘100.00’ | POST | 403 Forbidden |
| 4.09 | /getcurrencies | Username: valid\_username  Password: valid\_pass | GET | 200 OK  List of exchange rates |

# Project Conclusions

# References

# Bibliography

# APPENDICES

*(in no particular order)*

## Implementation Code

### Currency Converter Client

### Currency Converter Web Service

## Installation of Python and Flask in a VirtualEnv virtual environment

## Deployment of Python/Flask project to Heroku

Instructions are provided on the Heroku website for deploying, building and executing a web application developed in Python using the Flask web framework (instructions can be found at <https://devcenter.heroku.com/articles/getting-started-with-python>).

A user must authenticate their machine using SSH on first login to Heroku. A Python project must be created inside a “virtual environment”, using Virtualenv. Virtualenv is a tool used “to create isolated environments for Python in which you can install packages without interfering with the other virtualenvs nor with the system Python's packages”.[[52]](#footnote-53) All dependencies (i.e. Python and Flask extensions) are installed within the virtual environment and are specific to the Python project in question. Gunicorn was chosen as the web server. The sample application was developed in a Linux environment (Ubuntu 12.04) as Gunicorn is only available for Unix systems.

A simple ‘Hello World’-type Python-Flask web application was created in the project folder. Dependencies were specified in requirements.txt, and the Gunicorn web server and point of entry to the application were specified in a special text file called Procfile. The application was then deployed to Heroku via Git and successfully executed.

## Creation of PostgreSQL Database on Heroku

1. Telles, M. 2008, p.1 [↑](#footnote-ref-2)
2. http://en.wikipedia.org/wiki/Python\_(programming\_language) [↑](#footnote-ref-3)
3. http://www.python.org/about/ [↑](#footnote-ref-4)
4. http://www.python.org/about/apps/ [↑](#footnote-ref-5)
5. http://www.python.org/psf/ [↑](#footnote-ref-6)
6. http://www.python.org/community/ [↑](#footnote-ref-7)
7. http://www.tiobe.com/index.php/content/paperinfo/tpci/Python.html [↑](#footnote-ref-8)
8. Telles, M. 2008 p.3 [↑](#footnote-ref-9)
9. http://en.wikipedia.org/wiki/History\_of\_Python [↑](#footnote-ref-10)
10. http://docs.python.org/2/whatsnew/2.0.html [↑](#footnote-ref-11)
11. http://www.python.org/download/releases/2.7/ [↑](#footnote-ref-12)
12. http://en.wikipedia.org/wiki/Python\_(programming\_language) [↑](#footnote-ref-13)
13. http://www.python.org/dev/peps/pep-0020/ [↑](#footnote-ref-14)
14. Telles, M. 2008 p.8 [↑](#footnote-ref-15)
15. http://www.diveintopython.net/getting\_to\_know\_python/everything\_is\_an\_object.html [↑](#footnote-ref-16)
16. http://docs.python.org/2/howto/functional.html [↑](#footnote-ref-17)
17. http://legacy.python.org/dev/peps/pep-0206/ [↑](#footnote-ref-18)
18. http://en.wikipedia.org/wiki/Python\_syntax\_and\_semantics [↑](#footnote-ref-19)
19. Telles, M. 2008 p.34 [↑](#footnote-ref-20)
20. http://docs.python.org/2/tutorial/controlflow.html#default-argument-values [↑](#footnote-ref-21)
21. Telles, M. 2008 p.209 [↑](#footnote-ref-22)
22. Telles, M. 2008 p.213 [↑](#footnote-ref-23)
23. Telles, M. 2008 p.215 [↑](#footnote-ref-24)
24. Telles, M. 2008 p.226 [↑](#footnote-ref-25)
25. Telles, M. 2008 p.229 [↑](#footnote-ref-26)
26. Telles, M. 2008 p.237 [↑](#footnote-ref-27)
27. http://www.diveintopython.net/object\_oriented\_framework/defining\_classes.html [↑](#footnote-ref-28)
28. Telles, M. 2008 p.267 [↑](#footnote-ref-29)
29. http://www.diveintopython.net/object\_oriented\_framework/defining\_classes.html [↑](#footnote-ref-30)
30. http://www.diveintopython.net/object\_oriented\_framework/defining\_classes.html [↑](#footnote-ref-31)
31. Telles, M. 2008 p.269 [↑](#footnote-ref-32)
32. Telles, M. 2008 p.268 [↑](#footnote-ref-33)
33. Telles, M. 2008 p.289 [↑](#footnote-ref-34)
34. https://docs.python.org/2/tutorial/classes.html#multiple-inheritance [↑](#footnote-ref-35)
35. https://docs.python.org/2/tutorial/stdlib.html#batteries-included [↑](#footnote-ref-36)
36. https://docs.python.org/2/tutorial/stdlib.html#dates-and-times [↑](#footnote-ref-37)
37. https://docs.python.org/2/tutorial/stdlib.html#data-compression [↑](#footnote-ref-38)
38. https://docs.python.org/2/library/unittest.html#module-unittest [↑](#footnote-ref-39)
39. https://docs.python.org/2/tutorial/stdlib.html#operating-system-interface [↑](#footnote-ref-40)
40. https://docs.python.org/2/tutorial/stdlib.html#string-pattern-matching [↑](#footnote-ref-41)
41. https://docs.python.org/2/tutorial/stdlib2.html#working-with-binary-data-record-layouts [↑](#footnote-ref-42)
42. https://docs.python.org/2/tutorial/stdlib2.html#decimal-floating-point-arithmetic [↑](#footnote-ref-43)
43. https://wiki.python.org/moin/WebFrameworks [↑](#footnote-ref-44)
44. http://en.wikipedia.org/wiki/Web\_application\_framework [↑](#footnote-ref-45)
45. http://en.wikipedia.org/wiki/Web\_Server\_Gateway\_Interface [↑](#footnote-ref-46)
46. http://flask.pocoo.org/docs/foreword/#what-does-micro-mean [↑](#footnote-ref-47)
47. http://flask.pocoo.org/extensions/ [↑](#footnote-ref-48)
48. https://devcenter.heroku.com/articles/how-heroku-works#defining-an-application [↑](#footnote-ref-49)
49. https://devcenter.heroku.com/articles/python-faq#do-python-applications-run-behind-nginx [↑](#footnote-ref-50)
50. https://devcenter.heroku.com/articles/how-heroku-works#running-applications-on-dynos [↑](#footnote-ref-51)
51. http://flask.pocoo.org/docs/patterns/packages/ [↑](#footnote-ref-52)
52. https://wiki.archlinux.org/index.php/Python\_VirtualEnv [↑](#footnote-ref-53)