# Coding Standards

# for

# Traffic Light Optimization
# Imagine Interactive Systems (Iminsys)

**Prepared by Team Alpha:**

1. **TK Lebelo u15209190 (team leader)**
2. **RW van Graan u16040865**
3. **R Rammbuda u16207948**
4. **OM Mogoathle u14254922**
5. **TR Sithole u15360441**

**Submitted to : Stacey Baror**
**Avinash Singh**
**Eunice Hammond**

**May - November 2020**

# Contents

# 1 Naming Conventions

## 1.1 Classes

- All class names must begin with a capital letter
- The name must be descriptive of the classes functions.
- An underscore must be used to separate the different words in the class, function names.

## 1.2 Functions

- The name must be descriptive of the functions purpose to the class.
- An underscore must be used to separate the different words in the function names.
- The function names must be kept as short as possible.

## 1.3 Variables

- Global variables names must begin with a capital letter
- Constant variable names must be all capitals.
- An underscore must be used to separate the different words in the variable, function names.

# 2 File and folder Naming and Organization

## 2.1 Framework

For this project we will be using a Software framework which will be providing generic functionality as a starting point.
**Django framework will be used**

- All developers are to adhere to the basic project structure provided by the framework
- In cases of a single file not enough to provide specific functionality, a folder must be created for those functions, descriptive of what it contains.

## 2.2 Independent Subsystems and Third Party Software

- All subsystem should be treated as independent applications unless their purpose or functions to the system is too small to be recognised as a separate application.
- All third party software (NOT libraries) directly be used by the system would be treated as independent applications within Django.

# 3 Formatting and Indentation

- 4 spaces tabs must be used in all files for indentation of code for readability. Example:
  - Functions in a classes must be on a different level from the classes.
  - Operations, loops and conditional statements within a function must be on a different level to the function.
  - Operations within a loop or conditional statement must be on a different level to the respective parent head.
- A new line must be provided between functions in a class.
- Global Variables must be declared outside the classes
- When using any mathematical operators, there must be space between the operator and the operands.
- Avoid writing long operations for calculations or conditional statements. Operations must be broken down for readability.
-

# 4 Commenting and Documenting

## 4.1 Classes and functions

The comments within code file can be written in two ways. Before Class

- **Before Classes Functions.**
  A clear description **MUST** be given about the purpose of classes to the (sub)system and functions to the class.

- **Inline Comments.**
  Developers should try to use as little inline comments as possible. Only in times where the purpose of a statement could be confused to something different i.e. behaviour, an inline comments should be added to clear the jargon.

## 4.2 System Documentation

Developers are to constantly monitor the incremental states of the system development life-cycle in order to update the complimenting documentation to reflect what the current system is and what it provides. The documents include:

- Software Requirements Documentation
- System Architecture Design
- User Manual

# 5 Classes and Functions

## 5.1 Size

The length of classes and functions should not be very large i.e. Exceed:

- 100 Lines functions.
- 400 lines classes.

If it happens that functions or classes grow big, Developers are to further abstract the function or classes respectively.

## 5.2 Error return values and exception handling conventions

- Functions are to return appropriate error values or messages i.e. Error:
  - messages must be descriptive of what caused the problem.
  - Values must close enough to the expected return value e.g. when an integer value is expected, a negative value should be returned to signify an error
- Exceptions should be thrown only in cases where an alternative return cant be provide i.e. a totally erroneous request has been made

# 6 Testing

## 6.1 Unit Testing

- Testing must be written by the respective developer for individual models and views within the system.
- The test must reflect potential scenarios which might suggest different behaviour by the the models or views.

## 6.2 Integrated testing

The system wide testing should be provided when new functionality is added to the system. Testing integration ability of the new functions.