# *PYTHON CODING STANDARDS*

Version 1.1.0

*FIT1045 – Adapted from JavaScript coding standards (ENG1003)*

*Monash University | 25 Exhibition Walk, Clayton Campus, Monash University 3800*

# Table of Contents

# Purpose

This document serves as the Python coding standards document for all documentation and coding done for FIT1045: Algorithms and Programming Fundamentals in Python at Monash University. You should follow the advice in this document for your Assignment and workshop work.

The version of this document is 1.1.0.

This document was last updated on the September 21, 2017.

# Coding Standards

## Naming Conventions

Criteria:

- Function and variable names (including function parameters) must be **meaningful**, and be in **lowerCamelCase.**
- The name of each function and variable should to be **indicative** of its **purpose**.
- Class names should be **UpperCamelCase**.

See example below.

**BAD EXAMPLE**

```
# This function moves the image c to position represented by a,b
def A(a, b, c):
```

**GOOD EXAMPLE**

```
# This function moves the image 'imageToMove' to a position
#     represented by coordinates xCoordinate, yCoordinate
def moveToLocation(xCoordinate, yCoordinate, imageToMove):
```

## Magic Numbers:

Criteria:

- Unnamed **constants** (i.e., values that don't change) should be **assigned to named variables**
- This is *particularly* important for regularly used values
- As with any variable, the name of this should indicate its purpose
- The convention for constants is that their names are in **ALL_CAPS** (with words separated by underscores) to indicate that they are fixed

See example below.

**BAD EXAMPLE**

```
area = radius * radius * (3.14159)
circumference = 2 * radius * (3.14159)
kilometers = 1.60934 * miles
```

**GOOD EXAMPLE**

```
PI = 3.14159
MILES_TO_KM = 1.60934

area = radius * radius * PI
circumference = 2 * radius * PI
kilometers = MILES_TO_KM * miles
```

## Variable Scope

Criteria:

- Variables should be defined with the **most restrictive** (smallest) scope possible
- Global variables should only be used when **absolutely necessary**
- Global variables should be defined at the **top of file**, above function definitions and after any file documentation

See examples of scoping within functions below.

**BAD EXAMPLE**

```
currentPiece=[0,2]
attemptedMove = [3,5]
cellsJumped = 0
def checkValidMove():
      global cellsJumped
      i = currentPiece[0]
      while i < attemptedMove[0]:
            j = currentPiece[1]
            while j < attemptedMove[1]:
                  if (isEnemy(i,j)):
                        cellsJumped += 1
                  j = j + 1
            i = i + 1
      return ((cellsJumped % 2) == 0)
# cellsJumped, currentPiece and attemptedMove unnecessarily global
```

**GOOD EXAMPLE**

```
def checkValidMove(attemptedMove,currentPiece):

      cellsJumped = 0
      i = currentPiece[0]
      while i < attemptedMove[0]:
            j = currentPiece[1]
            while j < attemptedMove[1]:
                  if (isEnemy(i,j)):
                        cellsJumped += 1
                  j = j + 1
            i = i + 1
      return ((cellsJumped % 2) == 0)
# cellsJumped, i and j all local to checkValidMove and can't be accessed
# outside the function (appropriate as they are only for this function)
# attemptedMove and currentPiece passed in as arguments so that they need
# not be global either but local to the scope checkValidMove is called from
```

## Function Declarations

Criteria:

- Like with magic numbers, functions should always be **given a name** (not anonymous)
- Functions should be defined at the **top** of a file, just below any global variables or constants, rather than immediately before they are used
- Functions should be defined in the order of dependency, with the functions up the top having no dependencies and those lower down requiring some of the functions above
- Functions should **not** be defined inside other functions

See examples below.

**BAD EXAMPLE**

```
“””  This function is designed to find the maximum value in an
 array and change all elements to be that maximum value”””
def maximise(anArray):
      “””  The following function is generally useful, but can
       only be used from inside maximise() due to scope.”””
      def getMaxValue(theArray):
            theMax = theArray[0]
            for i in range(len(theArray)):
                  if (theArray[i] > theMax):
                        theMax = theArray[i]
            return theMax

      arrayMax = getMaxValue(anArray)
      for i in range(len(anArray)):
            anArray[i] = arrayMax
      return anArray
```

**GOOD EXAMPLE**

```
“”” The following function can now be reused by other code.”””
def getMaxValue(anArray):
      theMax = anArray[0]
      for i in range(len(anArray)):
            if (anArray[i] > theMax):
                  theMax = anArray[i]
      return theMax

“”” This function is designed to find the maximum value in an
 array and change all elements to be that maximum value”””
def maximise(anArray):
      var arrayMax = getMaxValue(anArray)
      for i in range(len(anArray)):
            anArray[i] = arrayMax
      return anArray
```

## Cohesion and Coupling - Writing Modular Code

Criteria:

- Always ensure that any function you create performs only one logical task.

See examples below.

**BAD EXAMPLE**

```
# This function is set up to try and race a series of cars until one
# wins.  It returns the top 5 cars
# theCars is a list of instances of a Car class with properties fuel,
# xPosition and fuelEconomy
def raceAllTheCars(theCars):
    win = 100
    topFive = []
    for i in range(len(theCars)):
        xChange = Math.random() * 5
        theCars[i].fuel -= xChange / theCars[i].fuelEconomy
        theCars[i].xPosition += xChange
        worstTopFive = -1
        worstTopFiveDist = -1
        for j in range(len(topFive)):
            if topFive[j].xPosition > worstTopFiveDist:
                worstTopFive = j
                worstTopFiveDist = topFive[j].xPosition
        if topFive.length < 5:
            topFive.append(theCars[i])
        else:
            if worstTopFiveDist < theCars[i].xPosition:
                topFive[worstTopFive] = theCars[i]
        if theCars[i].xPosition > win:
            return topFive
```

**GOOD EXAMPLE**

```
# This function is set up to try and race a series of cars until one
# wins.  It returns the top 5 cars
# theCars is a list of instances of a Car class with properties fuel,
# xPosition and fuelEconomy
def raceAllTheCars(theCars):
    win = 100
    topFive = []
    for i in range(len(theCars)):
        xChange = Math.random() * 5
        theCars[i].move(xChange)
        if topFive.length < 5:
            topFive.append(theCars[i])
        else:
            replaceSmallestWith(topFive, theCars[i])
        if theCars[i].xPosition > win:
            return topFive

def replaceSmallestWith(anArray, replacement):
    smallestIndex = indexOfMinimumValue(anArray)
    if anArray(smallestIndex) < replacement.xPosition:
        anArray[smallestIndex] = replacement


“”” There would also be
    +  a move function
    +  an indexOfMinimumValue function
  which have been omitted for conciseness.
“””
```

## Comments and Documentation

### Documenting Files

Each of these must:

- Explain the purpose of the file
- List the authors and team responsible for the code/content
- Include a last modified date and version number, if applicable
- List any other relevant details

See example below.

```
“””
 * Purpose: This file is designed to allow for easy changes to vision
 *          impaired modes for CSS-based websites
 * Organization/Team: Synergistic Site Solutions
 * Author: Preity Webb
 * Last Modified: 29 November 2014
 * Version: 3.0.4
 * Licence: Creative Commons Attribution; Non-Commercial (BY-NC)
“””
```

## Function Comments

Each of these must:

- Explain the purpose of the function,
- Describe the parameters as well as the return value (if any), and
- Be completely clear to someone unfamiliar with the code

See example below.

```
“”” moveToLocation()

 This function moves the image imageToMove to position represented by
 xCoordinate, yCoordinate. It does this treating 0,0 as the origin at
 the centre of the screen; for images more than 1 pixel by 1 pixel, this
 function places the centre of the image at the location specified. The
 centre is defined as the point halfway between the top and bottom and
 left and right sides, where there is any doubt between several pixels,
 the upper right one is chosen.

 argument: xCoordinate: this represents the final location along the
      x-axis for the object to be placed at.

 argument: yCoordinate: this represents the final location along the
      y-axis for the object to be placed at.

 argument: imageToMove: this is the image, passed by reference, that
      will be placed in the location specified by xCoordinate and
      yCoordinate. Note that this image cannot exceed a size of 40
      pixels by 100 pixels

 preconditions:
      image must exist and be at least 1 pixel by 1 pixel
      image cannot be larger than 40x100 pixels
      coordinates must be integers as they represent individual pixels

 postconditions:
      image will be at the location specified by the two coordinates

 returns:
      This function does not return anything
“””
function moveToLocation(xCoordinate, yCoordinate, imageToMove)
{
    [...]
```

## Inline Comments

Each of these must:

- Explain why particular choices were made, and
- Explain any complex code that another software engineer may find confusing
- Should match the indentation of code within that block
- Good use of variable names may reduce need for comments for certain lines (eg. If you can understand by reading the variable name what is happening)

See example below.

```
# [...]
def upperTriangularForm(coefficientMatrix,resultantVector):
    if isNumeric(coefficientMatrix):
        # We set up a specific function here, to ensure that all parts
        # of the matrix were numeric, as python isn't strict with
        # types.

        [...]

        # Note that we update the final elimination matrix as we go to
        # save time, considering we'll need this to perform LU
        # decomposition. This way we don't need to store each
        # elimination matrix used, we just need to know the most
        # recent one.
        matrixMultiplication(coefficientMatrix,eliminationMatrix)
        finalEliminationMatrix = matrixMultiplication(
                finalEliminationMatrix, eliminationMatrix);

        [...]
    }
}
```

**~ End of Document ~**