# INT3095 Assignment 2 (15%)

## General instructions

This assignment serves to assess your learning outcomes in up to L06. Please do the following:

1. It is suggested to use Python IDE to complete this assignment so as I can read your code in local computer (e.g., Thonny or PyCharm)
   - Download Thonny: http://www.thonny.com
2. Write your own code. Do not copy from the web or others.
3. Provide a brief report to show your program's testing outcome. It will capture all the major screenshots of the testing outcomes.
4. Provide a heading in your programs that includes your name, student ID.
5. Submit the source code and report to Moodle. You may zip the files.

Reminders:

1. Make sure that all your codes are error-free.
2. Make sure that you fulfill all the functional requirements of the programs.
3. The comments inside the code will be assessed as well. Make sure that your comments are informative.
4. Make your code as simple and efficient as possible.
5. Follow the best programming practice as far as applicable in the assignment https://www.datacamp.com/tutorial/coding-best-practices-and-guidelines

## Your Works

Develop classes for the Text-mode Python City, which is a simplified text mode version of the famous game SimCity.

## Part 1: Create a class for an ordinary city

Follow the instructions below to write the class for an ordinary city.

### Functional requirements

An ordinary city is created with **population size of 11000**, along with **10 units of culture, food, productivity, and wealth respectively**. To keep things simple, an ordinary city consists of only four types of citizens: professionals, farmers, workers, and businessmen. The composition of citizens would affect the culture, food, productivity, and wealth levels of the city.

In each **ROUND of the game**, a certain type of citizens (in batch of 1009) will be added to the city. The addition of each type of citizens would affect the properties of the city in the following way:

| | Population | Culture (unit) | Food (unit) | Productivity (unit) | Wealth (unit) |
|---|---|---|---|---|---|
| Professional | +1009 | +1 | -1 | -1 | unchanged |
| Farmer | +1009 | unchanged | +1 | +1 | unchanged |
| Worker | +2009 | unchanged | -2 | +1 | -1 |
| Businessman | +1009 | unchanged | -2 | +2 | +1 |

In addition, there are two types of **special events: earthquakes and trade**. Earthquake affects one city at a time and changes the properties of the city concerned, while trade affects two cities at a time and changes the properties of both cities. The changes required are stipulated in the following table:

| | Population | Culture | Food | Productivity | Wealth |
|---|---|---|---|---|---|
| Earthquake | -8000 | -1 | -3 | -3 | -5 |
| Trade | | +5 | +2 | +3 | +2 |

The properties of the city should be printed when the city is created, and after each update.

## Technical requirements

The class needs to at least contain the following (e.g., you may add more if needed):
Instance variables:

1. A variable that represents the name of the city. It is randomly selected from the following list[]. Therefore, **each game the city name may be different**.
   **[Erith, Furness, Pella Wish, Gamsby, Dalelry, Bleakburn, Stratford, Wanborne]**

2. The variables that represent the property values of population, culture, food, productivity, and wealth respectively needs to be initialized when the class is created in the constructor. Besides, it will be used to initialize the name of the city.

Following methods in the City Class are suggested at least to be included:

1. A `__str__` method that returns the name of the city object and its property values in a pretty format.
2. Methods `add_professional(self)`, `add_farmer(self)`, `add_worker(self)`, and `add_businessman(self)` that add citizens to the city and update the values of population, culture, food, productivity, and wealth.
   [Note that the population should be capped at zero if it is reduced to below zero. Other property values could be negative.]
3. For better code reuse, you may need to use other helper method(s) to do the actual update of the properties as required.
4. Method `earthquake(self)` that updates the properties of the city according to the table above.
5. Method `trade_with(self, city)` that updates the properties of both cities concerned according to the table above.

## Part 2: Create a class for a big city

Develop a class for big cities, which **inherits the class for ordinary cities**. [Note: No marks in this part if not use inheritance]

### Functional requirements

Big cities are also cities, except that they have larger initial values of the properties. The initial population of a big city is 150000, while the initial property values are 100 units [there is 10 unit for ordinary city] for each property.

**Big cities inherit all the methods of an ordinary city**, but they can also attack other cities, big or ordinary. When an attack occurs, the attacking city and the city being attacked will have their property values changed as follows:

| | Population | Culture | Food | Productivity | Wealth |
|---|---|---|---|---|---|
| Attacking city | -100 | unchanged | +5 | +3 | +5 |
| Being attacked | -500 | -2 | -5 | -3 | -4 |

### Technical requirements

The `Big_City` class should inherit from `City`. Other requirements are listed below (you may add more if needed):

Instance variables:

1. A variable that represents the name of the city. Randomly generated from the list in ordinary city but cannot the identical name as the ordinary city in a particular game.
2. The variables that represent the property values of population, culture, food, productivity, and wealth respectively. These should be initialized when the class is created, according to the big city requirements.
3. A constructor `__init__(self, name)` that overrides the constructor of the parent class. It should initialize the name of the city, as well as its properties values.

Methods:

1. Method `attack (self, city)` that attacks another city and updates the properties as required.

## Test cases

You should run the following code and generate an output looks like as follows in a particular run.

**(Suppose the random ordinary city name is Erith and the name of big city is Furness**

```
c = City("Erith")
c.earthquake()
c.earthquake()
c.add_professional()
c.add_farmer()
c.add_worker()
c.add_businessman()

bc = Big_City("Furness")
c.trade_with(bc)
bc.add_professional()
bc.add_farmer()
bc.add_worker()
bc.attack(c)
bc.earthquake()
```

The output

**NOTE: followings output just for reference only. The data will be different as I have changed some data in the descriptions. For example, the first created city may be {any city name by random} [11000, 10, 10, 10, 10]**

```
City Erith created. [note Erith is random, your project may NOT by this city]
Erith: [10000, 10, 10, 10, 10]

Earthquake in Erith!
Erith: [2000, 9, 7, 7, 5]

Earthquake in Erith!
Erith: [0, 8, 4, 4, 0]

Message: Adding 1000 professionals to Erith.
Erith: [1000, 9, 3, 3, 0]

Message: Adding 1000 farmers to Erith.
Erith: [2000, 9, 4, 4, 0]

Message: Adding 1000 workers to Erith.
Erith: [3000, 9, 3, 5, 0]
```

Message: Adding 1000 businessmen to Erith.
Erith: [4000, 9, 1, 7, 1]

Big city Furness created.
Furness: [100000, 100, 100, 100, 100]

Erith trades with Furness.
Erith: [4000, 14, 3, 10, 3]
Furness: [100000, 105, 102, 103, 102]

Message: Adding 1000 professionals to Furness.
Furness: [101000, 106, 101, 102, 102]

Message: Adding 1000 farmers to Furness.
Furness: [102000, 106, 102, 103, 102]

Message: Adding 1000 workers to Furness.
Furness: [103000, 106, 101, 104, 102]

Furness attacks Erith!
Furness: [102900, 106, 106, 107, 107]
Erith: [3500, 12, -2, 7, -1]

Earthquake in Furness!
Furness: [94900, 105, 103, 104, 102]

Marking Guide

The grading of this assignment will be based on the following aspects. You can refer to the significant of every part to pay of efforts of it.

**[Note]: Fail Grade will be given if do not submit source code.**

| | |
|---|---|
| Correctness of the program (50%) | There is No Bugs in the program. In addition, it can correctly use python features to complete the solution. (e.g., class, inheritance, methods …)<br><br>The running outcome is correct according to the designed operations. |
| Layout and detail of screen captures with appropriate discussions. (20%) | Good organization of the screen captures in the report and with meaningful/detailed explanations.<br><br>Good format, consistent style, font size, style... shown in the report. |
| Following of good program practice. (20%) | A detailed heading to describe your program. Including purpose of this program, your name, student ID, and Date, version.<br><br>Provide effective comments to explain your codes.<br><br>Following a lot of base features on 'best program practice' |
| Additional Smart features (10%) | You may design some relevant smart feature in your programs.<br><br>For example, add one more interested operation in the game. (e.g., alliance of two ordinary cities so as to increase the properties for defense of attack)<br><br>Note, if you have this, please specify in your report so as I can be aware of your work. |