# Examen del módulo Programación 1

Emiliano Dalla Verde Marcozzi

# Presentación de la actividad

Se requiere realizar un proyecto para el curso masivo abierto MOOC[1] Python Para Zumbies del profesor Fernando Masanori[2]. El proyecto debe contemplar el poder ser ejecutado bajo la tecnología VPL Virtual Programming Lab[3] para la plataforma de aprendizaje Moodle[4].
En este trabajo se desarrolla como proyecto el juego “Piedra, Papel o Tijera”[5], el cual invita al alumno a ejercitar varios conceptos de Python como ser clases, herencia, *properties*, bucles, etc. En el ejercicio se brinda un archivo con un programa incompleto, con secciones comentadas para que el alumno tenga una guía de lo que debe completar.

# Análisis, especificación, diseño, implementación y prueba básica del ejercicio.

## Análisis y especificación

Se pretende que el alumno complete el programa con el código faltante, a fin de que al ejecutarse el programa, el mismo tenga implementado lo necesario para jugar al juego “Piedra, Papel o Tijera”.
El programa debe ser capaz de poder procesar en orden las jugadas de dos jugadores. El programa debe ser capaz de determinar quién es el ganador de la partida a partir de comparar en orden las jugadas de cada jugador. Las reglas que determinan quién es el ganador por partida son las siguientes[6]:

- Piedra gana a Tijera
- Tijera gana a Papel
- Papel gana a Piedra

A partir de evaluar en orden las jugadas de cada jugador, el programa debe retornar como número entero el jugador ganador, o el número cero en caso de empate.

[1] https://es.wikipedia.org/wiki/MOOC
[2] https://about.me/fmasanori
[3] http://vpl.dis.ulpgc.es/
[4] https://moodle.org/?lang=es
[5] https://es.wikipedia.org/wiki/Piedra,_papel_o_tijera
[6] https://es.wikipedia.org/wiki/Piedra,_papel_o_tijera#C.C3.B3mo_se_juega

## Diseño

El programa debe implementar las siguientes clases:

- Juego
- Tijera
- Papel
- Piedra

La clase juego debe implementar un método “jugar” el cual acepta dos argumentos cuales son de tipo listas de Python.
Las listas pasadas como argumentos no pueden contener otro elemento que no sean las siguientes cadenas de texto: “piedra”, “papel”, “tijera”.
El método “jugar” debe comparar en orden las listas de movimientos de ambos jugadores, incrementando un contador interno que representa el puntaje acumulado por cada jugador en caso de ganar una partida.
Sean m1 y m2 movimientos pertenecientes al jugador 1 y jugador 2, el método “jugar” define cual es el movimiento ganador realizando las siguientes acciones:
Se crea una instancia de clases de entre las clases Piedra, Papel, Tijera que represente al movimiento de la partida. El mapa se define de la siguiente manera:

- Si el movimiento es “piedra” se debe generar una instancia de la clase Piedra.
- Si el movimiento es “tijera” se debe generar una instancia de la clase Tijera.
- Si el movimiento es “papel” se debe generar una instancia de la clase Papel.

Se comparan las instancias de clases m1, m2. Al compararse utilizando los operadores **<**, **>**, **==** (por ejemplo: m1 > m2) se ejecuta el método ***__cmp__***[7] de la instancia de clase evaluada.
Al momento de ejecutarse __cmp__ por comparar dos movimientos, m1 y m2 se definen las siguientes reglas para saber cual de los dos movimientos evaluados gana la partida:

- Si se obtiene como resultado un número negativo, m1 pierde contra m2.
- Si se obtiene como resultado el número cero, m1 empata contra m2.
- Si se obtiene como resultado un número positivo, m1 gana contra m2.

Es de suma importancia que las implementaciones de ***__cmp__*** en las clases Piedra, Tijera y Papel definan de forma correcta como se comporta la instancia de clase cuando es comparada con otra instancia, puesto que el resultado de esta comparación decide cuál es el movimiento ganador. Puede verse un ejemplo completo en Piedra, Papel o Tijeras (Github de Universidad del Este).

[7] https://docs.python.org/2/reference/datamodel.html#object.__cmp__

## Implementación

La implementación del juego es la siguiente[8]:

```
#!/usr/bin/python2
import unittest


__author__ = "Emiliano Dalla Verde Marcozzi <edvm@fedoraproject.org>"


class Elemento(object):

    @property
    def name(self):
        """Nombre del elemento"""
        return self._name


class Tijera(Elemento):

    def __init__(self):
        self._name = 'tijera'

    def __cmp__(self, obj):
        """Reglas:
            - Tijera empata con Tijera
            - Tijera pirde contra piedra
            - Tijera gana contra papel
        """
        if obj.name == 'tijera':
            return 0
        if obj.name == 'piedra':
            return -1
        if obj.name == 'papel':
            return 1


class Piedra(Elemento):

    def __init__(self):
        self._name = 'piedra'

    def __cmp__(self, obj):
        """Reglas:
            - Piedra empata con piedra
            - Piedra pirde contra papel
            - Piedra gana contra tijera
```

[8] https://github.com/UniversidadDelEste/PyZombis/commit/86352c96179fe12163e408a7a829cd9415a876fc

```
        """
        if obj.name == 'piedra':
            return 0
        if obj.name == 'papel':
            return -1
        if obj.name == 'tijera':
            return 1


class Papel(Elemento):

    def __init__(self):
        self._name = 'papel'

    def __cmp__(self, obj):
        """Reglas:
            - Papel empata con papel
            - Papel pierde contra tijera
            - Papel gana contra piedra
        """
        if obj.name == 'papel':
            return 0
        if obj.name == 'tijera':
            return -1
        if obj.name == 'piedra':
            return 1


class Juego(object):

    def __init__(self, player1, player2, number_of_games=3):
        """Parametros:
            - player1(str):     Movimientos del jugador 1
            - player2(str):     Movimientos del jugador 2
        """
        # Validamos que los parametros sean como los esperamos
        for moves in [player1, player2]:
            if not isinstance(moves, list):
                raise TypeError('El argumento debe ser de tipo lista')
            assert len(moves) == number_of_games, 'Cantidad de movimientos insuficiente'
            for elem in moves:
                assert elem in ['piedra', 'papel', 'tijera'], 'Elemento de lista no valido'
        self.m_player1 = player1
        self.m_player2 = player2
        self.score_player1 = 0  # puntaje del player1
        self.score_player2 = 0  # puntaje del player2
        self.number_of_games = number_of_games  # cantidad de partidas que se van a jugar

    def _get_elem(self, elem):
        elementos = dict(
            piedra=Piedra,
```

```
            papel=Papel,
            tijera=Tijera
        )

        return elementos[elem]()

    def jugar(self):
        """Retorna un entero:
            - 0 si la partida fue un empate
            - 1 si gano el jugador 1
            - 2 si gano el jugador 2
        """
        for x in range(self.number_of_games):
            mov1, mov2 = self._get_elem(self.m_player1[x]), self._get_elem(self.m_player2[x])
            if mov1 == mov2:       # empate
                self.score_player1 += 1
                self.score_player2 += 1
            if mov1 > mov2:
                self.score_player1 += 1
            if mov1 < mov2:
                self.score_player2 += 1

        winner = 0
        if self.score_player1 > self.score_player2:
            winner = 1
        if self.score_player1 < self.score_player2:
            winner = 2

        return winner
```

La implementación de los tests es la siguiente:

```
class TestJuego(unittest.TestCase):

    def test_piedra_vs_piedra(self):
        piedra_1 = Piedra()
        piedra_2 = Piedra()
        assert piedra_1 == piedra_2

    def test_piedra_vs_tijera(self):
        piedra = Piedra()
        tijera = Tijera()
        assert piedra > tijera

    def test_piedra_vs_papel(self):
        piedra = Piedra()
        papel = Papel()
        assert piedra < papel

    def test_tijera_vs_tijera(self):
        tijera_1 = Tijera()
```

```
    tijera_2 = Tijera()
    assert tijera_1 == tijera_2

def test_tijera_vs_papel(self):
    tijera = Tijera()
    papel = Papel()
    assert papel < tijera

def test_tijera_vs_piedra(self):
    tijera = Tijera()
    piedra = Piedra()
    assert piedra > tijera

def test_papel_vs_papel(self):
    papel_1 = Papel()
    papel_2 = Papel()
    assert papel_1 == papel_2

def test_papel_vs_piedra(self):
    papel = Papel()
    piedra = Piedra()
    assert papel > piedra

def test_papel_vs_tijera(self):
    papel = Papel()
    tijera = Tijera()
    assert tijera > papel

def test_juego_empate(self):
    m1 = ['piedra', 'papel', 'tijera']
    m2 = ['tijera', 'papel', 'piedra']
    juego = Juego(m1, m2)
    assert juego.jugar() == 0

def test_juego_gana_jugador_1(self):
    m1 = ['piedra', 'tijera', 'papel']
    m2 = ['piedra', 'papel', 'papel']
    juego = Juego(m1, m2)
    assert juego.jugar() == 1

def test_juego_gana_jugador_2(self):
    m1 = ['piedra', 'tijera', 'papel']
    m2 = ['papel', 'piedra', 'tijera']
    juego = Juego(m1, m2)
    assert juego.jugar() == 2
```

## Prueba

Se puede probar el juego corriendo los tests de la siguiente forma:

```
[edvm@mitsuki piedra_papel_tijera (master)]$ python2 jug.py
............
----------------------------------------------------------------------
Ran 12 tests in 0.001s

OK
[edvm@mitsuki piedra_papel_tijera (master)]$
```

## Instrucciones paso a paso de la configuración de la actividad

La actividad cuenta principalmente de dos archivos:

- **`ej7.py`**
- **`evaluate.py`**

El archivo **`ej7.py`** es una versión del juego incompleta diseñada para que el alumno programe el código que falta. El archivo **`evaluate.py`** importa y testea el código programado en **`ej7.py`**. Tanto estos dos archivos, como una implementación del juego completa con sus tests ejecutándose correctamente puede encontrarse en el [siguiente link.]

Utilizando el módulo VPL de Moodle, los pasos a seguir para configurar esta actividad son:

1. Loguearse a moodle como instructor al curso donde se agregará la prueba.
2. Activar edición y añadir una nueva actividad. Completar los campos del formulario según corresponda. Al momento de solicitarse una **Descripción Completa** puede utilizar el siguiente texto:

```
Bienvenido al juego Piedra, Papel o Tijera!. Desafortunadamente el juego no funciona puesto que se han perdido algunas partes del código fuente. Si eres tan amable, por favor te pido me ayudes a ponerlo en funcionamiento nuevamente. Para ello deberás como primer paso, leer y entender el código existente. Luego, notaras que falta completar la implementación de las clases Piedra y Papel, puede que leyendo el Ćodigo de la clase Tijera te sirva de guia. También creo haber notado que en la clase Juego, en el método jugar, pareciera que falta definir cuando gana un jugador una partida, te he dejado un comentario en el código fuente para indicarte en qué parte deberías escribir el código faltante, espero te sea de ayuda. Muchas gracias por tu ayuda!
```

3. Configurar en la opcion `Ficheros Requeridos` el fichero **`ej7.py`**
4. Configurar en `Ajustes Avanzados` -> `Ficheros para la Ejecucion` el fichero **`evaluate.py`**

A continuación una captura de pantalla para guiarle en el proceso: