

Corrección refactorización Yahtzee

Enunciado

Crear un **repositorio en GitHub** para los **katas** y **subir correcciones** que le hagamos a los compañeros **sobre la refactorización de yahtzee.py y test_yahtzee.py**

Introducción

Se ha creado un repositorio con todo lo referente a los katas de refactorización para Entornos. En el además de **mis ejercicios** hay un **directorio dedicado a la corrección de los compañeros**. Donde hay y un **directorio Benito con su código** y un **directorio correcciones dentro de Benito con el código corregido por mi**. Ambos llevan **comentado las modificaciones** que se han realizado además **se ha modificado el archivo de test y todo funciona correctamente**.

Yahtzee.py

Organización del código

El **primer fallo** sin ser realmente un fallo que he visto es que la estructura del código **no estaba demasiado bien organizada**.

Modifiqué la misma para **que la función init y los métodos de la clase** estuvieran en **primer lugar, y no en el medio del archivo**.

```
26
27 class Yahtzee:
28
29     @staticmethod # podria ser un metodo de la clase
30     def chance(d1, d2, d3, d4, d5):
31         return d1 + d2 + d3 + d4 + d5
32
33     @staticmethod
34     def yathzee_scores_50(lista):
35         if lista.count(lista[0]) == 5:
36             return 50
37         else:
38             return 0
39
40     ### def ones, twos, threes in the same funcion
41     @staticmethod
42     def ones(d1, d2, d3, d4, d5):
43         lista = (d1, d2, d3, d4, d5)
44         return lista.count(1)
45
46     @staticmethod
47     def twos(d1, d2, d3, d4, d5):
48         lista = (d1, d2, d3, d4, d5)
49         return lista.count(2)*2
50
51     @staticmethod
52     def threes(d1, d2, d3, d4, d5):
53         lista = (d1, d2, d3, d4, d5)
54         return lista.count(3)*3
```

Organización anterior a las modificaciones

```
class Yahtzee:

    def __init__(self, d1, d2, d3, d4, d5): # init y metodos
                                                # al inicio
        self.dice = [d1, d2, d3, d4, d5]

    def fours(self):
        return self.dice.count(4)*4

    def fives(self):
        return self.dice.count(5)*5

    def sixes(self):
        return self.dice.count(6)*6

    def chance(self, d1, d2, d3, d4, d5): #hago el metodo de
        self.chance = d1 + d2 + d3 + d4 + d5
        return self.chance

    def yathzee_scores_50(self, lista): # agrego el metodo a i
        self.lista = lista
        if self.lista.count(self.lista[0]) == 5:
            return 50
        else:
            return 0
```

Organización posterior a las modificaciones

Métodos de la clase

Este es un problema que **también tuvimos José Rosendo y yo**. Tras haber dedicado las últimas semanas a programación y orientación a objetos y teniendo claro las ideas de refactorización he llegado a la conclusión de que **en este caso es necesario suprimir los métodos estáticos** y se **han hecho métodos de la clase**. Esto se verá a continuación con los siguientes puntos.

No hay ningún comentario en el código

Es necesario documentar las funciones del programa. Y aquellas que pueden ser más complejas

Función chance()

Además de ser un **método estático** la función **recibe los cinco dados y los suma**, Se ha agregado el atributo self y creado un atributo de la clase para sumar los dados

```
26
27 class Yahtzee:
28
29     @staticmethod # podria ser un metodo de la clase
30     def chance(d1, d2, d3, d4, d5):
31         return d1 + d2 + d3 + d4 + d5
32
```

Método chance() antes de modificar

```
def chance(self, d1, d2, d3, d4, d5): #hago el metodo de
    self.chance = d1 + d2 + d3 + d4 + d5
    return self.chance
```

Método chance() después de modificar

Función Yathzee_score_50()

Era una función estática y se ha hecho la misma propia de la clase. Añadiendo el atributo listo a la misma, este atributo se utiliza más adelante en otras funciones

```
@staticmethod
def yathzee_scores_50(lista):
    if lista.count(lista[0]) == 5:
        return 50
    else:
        return 0
```

Método yathzee_score_50() antes de modificar

```
def yathzee_scores_50(self, lista): # agrego el metodo a
    self.lista = lista
    if self.lista.count(self.lista[0]) == 5:
        return 50
    else:
        return 0
```

Método Yahtzee_score_50() después de modificar

Funciones ones() twos() threes()

En primer lugar, de nuevo el método es estático. Y en segundo lugar vamos a reutilizar self. Lista que la dejamos como atributo de la clase en el método anterior, reduciendo así código

```
### def ones, twos, threes in the same funcion
@staticmethod
def ones(d1, d2, d3, d4, d5):
    lista = (d1, d2, d3, d4, d5)
    return lista.count(1)

@staticmethod
def twos(d1, d2, d3, d4, d5):
    lista = (d1, d2, d3, d4, d5)
    return lista.count(2)*2

@staticmethod
def threes(d1, d2, d3, d4, d5):
    lista = (d1, d2, d3, d4, d5)
    return lista.count(3)*3
```

Funciones ones() twos() threes() antes de modificar

```
def ones_twos_threes(self, lista, opcion): # Agrupación de
    self.lista = lista
    if opcion.lower() == "ones":
        return self.lista.count(1)
    elif opcion.lower() == "twos":
        return self.lista.count(2)*2
    elif opcion.lower() == "threes":
        return self.lista.count(3)*3
    else:
        return False
```

Funciones unificadas en una sola ones_twos_threes()

Función score_pair() two_pair()

Estas funciones además **de ser estáticas** y de que **las vamos ha hacer parte de la clase**. Tenemos un bloque en el que se repite exactamente lo mismo y es la creación de la lista de **counts**. Por lo que directamente crearé una función que devuelva **self.counts** para utilizarla directamente en **score_pair()** y **two_pair()**

```
@staticmethod
def score_pair(d1, d2, d3, d4, d5):
    counts = [0]*6
    counts[d1-1] += 1
    counts[d2-1] += 1
    counts[d3-1] += 1
    counts[d4-1] += 1
    counts[d5-1] += 1
    at = 0
    for at in range(6):
        if (counts[6-at-1] == 2):
            return (6-at)*2
    return 0

@staticmethod
def two_pair(d1, d2, d3, d4, d5):
    counts = [0]*6
    counts[d1-1] += 1
    counts[d2-1] += 1
    counts[d3-1] += 1
    counts[d4-1] += 1
    counts[d5-1] += 1
    n = 0
    score = 0
    for i in range(6):
        if (counts[6-i-1] >= 2):
            n = n+1
            score += (6-i)
    if (n == 2):
```

Funciones score_pair() two_pair() antes de modificar

```
def make_list_counts(self, d1, d2, d3, d4, d5): # funcion comu
    # además de que i
    self.counts = [0]*6
    self.counts[d1-1] += 1
    self.counts[d2-1] += 1
    self.counts[d3-1] += 1
    self.counts[d4-1] += 1
    self.counts[d5-1] += 1
    return self.counts

def score_pair(self): # función self recube counts
    at = 0
    for at in range(6):
        if (self.counts[6-at-1] == 2):
            return (6-at)*2
    return 0

def two_pair(self): # funcion self recibe counts
    n = 0
    score = 0
    for i in range(6):
        if (self.counts[6-i-1] >= 2):
            n = n+1
            score += (6-i)

    if (n == 2):
        return score * 2
    else:
        return 0
```

Nueva función make_list_counts() y la modificación score_pair()t wo_pair()

Funciones four_of_a_kind() y three_of_a_kind():

De nuevo las hacemos de la clase, reutilizamos el atributo de la clase self.lista y unimos ambas funciones.

```
09 # Three_of_a_kind_four_of_a_kinf pueden ir en una sola fun
10 @staticmethod
11 def four_of_a_kind(d1, d2, d3, d4, d5):
12     lista = [d1, d2, d3, d4, d5]
13     if lista.count(lista[0]) >= 4:
14         return lista[0]*4
15     elif lista.count(lista[1]) >= 4:
16         return lista[1]*4
17     else:
18         return 0
19
20 @staticmethod
21 def three_of_a_kind(d1, d2, d3, d4, d5):
22     lista = [d1, d2, d3, d4, d5]
23     if lista.count(lista[0]) >= 3:
24         return lista[0]*3
25     elif lista.count(lista[1]) >= 3:
26         return lista[1]*3
27     else:
28         return 0
29
30 ###
```

Funciones four_of_a_kind() three_of_a_kind() antes de modificar

```
def four_of_a_kind_three_of_a_kind(self, lista, opcion): #  
    # y three of  
    self.lista = lista  
    if opcion == "three":  
        if self.lista.count(self.lista[0]) >= 3:  
            return self.lista[0]*3  
        elif self.lista.count(self.lista[1]) >= 3:  
            return self.lista[1]*3  
        else:  
            return 0  
  
    if opcion == "four":  
        if self.lista.count(self.lista[0]) >= 4:  
            return self.lista[0]*4  
        elif self.lista.count(self.lista[1]) >= 4:  
            return self.lista[1]*4  
        else:  
            return 0  
  
    else:  
        return 0
```

Funciones four_of_a_kind() three_of_a_kind() despues de modificar

Funciones smallStraight() y largeStraigh()

Al ser tan parecidas las unificaré en una las haré métodos de la clase y se reutilizará el atributo self.lista

```
###  
# small y large se pueden agrupar en una sosla funcion  
# y hacer que formen parte de la clase  
@staticmethod  
def smallStraight(d1, d2, d3, d4, d5):  
    if sorted((d1, d2, d3, d4, d5)) == [1, 2, 3, 4, 5]:  
        return 15  
    else:  
        return 0  
  
@staticmethod  
def largeStraight(d1, d2, d3, d4, d5):  
    if sorted((d1, d2, d3, d4, d5 )) == [2,3,4,5,6]:  
        return 20  
    else:  
        return 0
```

Funciones smallStraight() y largeStraigh() antes de modificar

```
# añadidas a la clase
def small_large_list(self, lista):
    self.lista = lista
    if sorted(self.lista) == [1, 2, 3, 4, 5]:
        return 14
    if sorted(self.lista) == [2, 3, 4, 5, 6]:
        return 20
    else:
        return 0
```

Funcion `small_large_list()` de las dos anteriores

Función `fullHouse()`

Esta funciona igual salvo por que la hemos hecho más sencilla quitando tails y reutilizar el atributo `self.counts()` y su método `make_list_counts()` además de hacerla método de la clase

```
@staticmethod
def fullHouse(d1, d2, d3, d4, d5):
    tallies = []
    _2 = False
    i = 0
    _2_at = 0
    _3 = False
    _3_at = 0

    tallies = [0]*6
    tallies[d1-1] += 1
    tallies[d2-1] += 1
    tallies[d3-1] += 1
    tallies[d4-1] += 1
    tallies[d5-1] += 1

    for i in range(6):
        if (tallies[i] == 2):
            _2 = True
            _2_at = i+1

    for i in range(6):
        if (tallies[i] == 3):
            _3 = True
            _3_at = i+1

    if (_2 and _3):
        return _2_at * 2 + _3_at * 3
    else:
        return 0
```

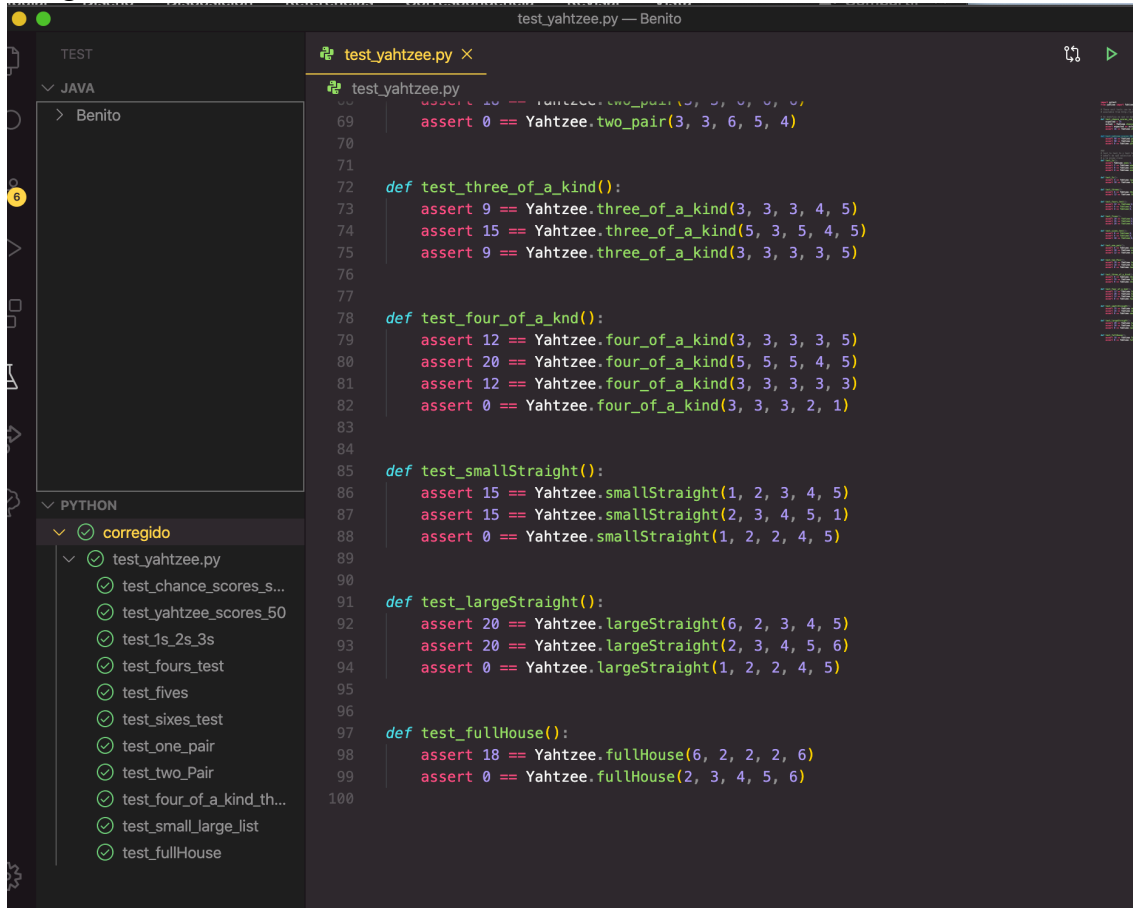
Función `Fullhouse()` antes de modificar

```
# se ha agregado a la clase ademas de recibir e  
# de la clase counts  
def fullHouse(self):  
    _2 = False  
    i = 0  
    _2_at = 0  
    _3 = False  
    _3_at = 0  
    for i in range(6):  
        if (self.counts[i] == 2):  
            _2 = True  
            _2_at = i+1  
  
    for i in range(6):  
        if (self.counts[i] == 3):  
            _3 = True  
            _3_at = i+1  
  
    if (_2 and _3):  
        return _2_at * 2 + _3_at * 3  
    else:  
        return 0
```

Función fullHouse() después de modificar

test_yahtzee.py

Para finalizar comentar que se ha modificado el archivo de test y todo funciona correctamente. Evidentemente al hacer los métodos estáticos, métodos de la clase es necesario modificarlo casi todo. Expondré en el documento una captura de los tests pasados correctamente. Estos están en el **directorio corregido** junto a el resto del código



The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer is divided into 'TEST' and 'PYTHON' sections. Under 'PYTHON', there is a folder named 'corregido' which contains several test files, all marked with a green checkmark. The code editor displays the contents of 'test_yahtzee.py', which includes several test functions for Yahtzee game logic. The tests are as follows:

```
69 assert 10 == Yahtzee.two_pair(3, 3, 6, 5, 4)
70
71
72 def test_three_of_a_kind():
73     assert 9 == Yahtzee.three_of_a_kind(3, 3, 4, 5)
74     assert 15 == Yahtzee.three_of_a_kind(5, 3, 5, 4, 5)
75     assert 9 == Yahtzee.three_of_a_kind(3, 3, 3, 3, 5)
76
77
78 def test_four_of_a_kind():
79     assert 12 == Yahtzee.four_of_a_kind(3, 3, 3, 3, 5)
80     assert 20 == Yahtzee.four_of_a_kind(5, 5, 5, 4, 5)
81     assert 12 == Yahtzee.four_of_a_kind(3, 3, 3, 3, 3)
82     assert 0 == Yahtzee.four_of_a_kind(3, 3, 3, 2, 1)
83
84
85 def test_smallStraight():
86     assert 15 == Yahtzee.smallStraight(1, 2, 3, 4, 5)
87     assert 15 == Yahtzee.smallStraight(2, 3, 4, 5, 1)
88     assert 0 == Yahtzee.smallStraight(1, 2, 2, 4, 5)
89
90
91 def test_largeStraight():
92     assert 20 == Yahtzee.largeStraight(6, 2, 3, 4, 5)
93     assert 20 == Yahtzee.largeStraight(2, 3, 4, 5, 6)
94     assert 0 == Yahtzee.largeStraight(1, 2, 2, 4, 5)
95
96
97 def test_fullHouse():
98     assert 18 == Yahtzee.fullHouse(6, 2, 2, 2, 6)
99     assert 0 == Yahtzee.fullHouse(2, 3, 4, 5, 6)
100
```

```
### 1º
# La ubicación del código no es la más adecuada
```

```
### 2º
# No hay ingun comentario en ninguna función

### 3º
# La función chance revive 5 dados y los suma si es la suma de 5 dados se
podfria
# simplement recibir self y sumar los dados de seff

### 4º
# las función yathzee_score_50 también puede ir dentro de una clase
# ya que lo recibe es una lista puede ser perfectamente self.dice

### 5º
# Las funciones ones, twos threes pueden ir en la misma función
# además de ir dentro de la clase

### 6º
# small y large se pueden agrupar en una sosla funcion
# y hacer que formen parte de la clase

### 7º
# se suprime tallies y trabaja sobre self.counts
# se hace metodo de la clase
```