

Taller Python

Teoría de Lenguajes de Programación

Tomás Escobar Rivera - 1036448606

Leonard David Vivas Dallos - 1070947936

Oscar Mauricio Salazar Ospina

23 de Septiembre de 2024

Universidad Nacional de Colombia

Punto 1

La oficina de pasaportes requiere la generación de códigos de identificación utilizando las siguientes restricciones caracteres especiales dadas las siguientes restricciones:

- Letras: A,B,C,D,E
- Dígitos: 4,5,6,7,8

Las identificaciones constan de: palabra “PAS-” seguido de 2 letras, 6 letras, “-” y la hora actual, ej: PAS-AO234587-03:55. Encuentre todos los posibles códigos y retorna el tamaño de la lista.

Solución:

```
import itertools

from datetime import datetime

# Defino las constantes
letras = 'ABCDE'
digitos = '45678'

# Generar todas las combinaciones posibles
comb_letras = itertools.product(letras, repeat=2) # 2 letras
comb_digitos = itertools.product(digitos, repeat=6) # 6 digitos

# Calcula todas las combinaciones posibles
combinaciones = []

for letra, digito in itertools.product(comb_letras, comb_digitos):
    pasaporte = 'PAS-{}{}{}'.format(''.join(letra),
    ''.join(digito), ''.join(datetime.now().strftime('%H%M')))
    combinaciones.append(pasaporte)
```

```
# Cuenta el total de combinaciones

total = len(combinaciones)

print(total)
```

Tests Ejecutados en Python

```
El total de combinaciones posibles es: 390625

Process finished with exit code 0
```

Punto 2

Utilice TDD: imagina y crea una aplicación de tu gusto (carro de compras, juego, sistema de calificaciones, viajes, etc):

- Selecciona al menos 3 casos de uso en tu aplicativo (carga de puntos, redención de productos, creación nuevo usuario, etc).
- Define los requisitos del sistema y el detalle de las funcionalidades (breve descripción).
- Crea los casos y las pruebas unitarias para dichos casos de uso.
- Crea la implementación para atender los casos planteados.

```
# APLICATIVO: Juego FC Mobile

# CASOS DE USO:

# 1. Carga de FC Points y/o Monedas
# 2. Redención de Sobre
# 3. Compra de Jugadores

# REQUERIMIENTOS: En la creación de nuestro juego, cada jugador
inicia con 500 FC Points, 1000 Monedas y 5 Jugadores en su equipo,
cada uno con 70 de valoración. Se lleva la valoración del equipo.
La valoración del equipo es el promedio de la valoración de los
```

jugadores. Puedo agregar FC Points o Monedas a mi cuenta. Por cada 100 FC Points que agregue, mejora el peor jugador de su plantilla en 1 punto (si hay dos o más jugadores con la misma valoración, mejora cualquiera de ellos. Para la mejora de jugadores se mejora tomando el piso de la cantidad de FC Points agregados. La valoración máxima a la que puede llegar dicha mejora es 90, el resto de mejoras disponibles por la recarga serán perdidas). Puedo redimir un sobre. Por cada sobre redimido, se añade un jugador aleatorio a mi plantilla con valoración entre 70 y 90. Cada sobre descuenta 50 FC Points o 100 Monedas. (La primera opción es descontar FC Points, si no se cuenta con FC Points, se descuentan Monedas). Se pueden comprar jugadores, por cada jugador comprado, se descuenta su valoración multiplicada por 10 Monedas. (Por ejemplo, si quiero comprar un jugador de 80 de valoración, se descuentan 800 Monedas). Cada jugador comprado se añade a mi plantilla y me da 10 FC Points.

```
import unittest
```

```
import random
```

```
class Jugador:
```

```
    def __init__(self, valoracion):
```

```
        self.valoracion = valoracion
```

```
class JuegoFCMobile:
```

```
def __init__(self):  
    self.fc_points = 500  
    self.monedas = 1000  
    self.plantilla = [Jugador(70) for _ in range(5)]  
  
    def valoracion_promedio(self):  
        total = sum(jugador.valoracion for jugador in  
self.plantilla)  
        return total / len(self.plantilla)  
  
    def agregar_fc_points(self, puntos):  
        self.fc_points += puntos  
  
        # Mejora el peor jugador  
        peor_jugador = min(self.plantilla, key=lambda jugador:  
jugador.valoracion)  
        peor_jugador.valoracion = min(90, peor_jugador.valoracion +  
puntos // 100)  
  
    def agregar_monedas(self, monedas):  
        self.monedas += monedas  
  
    def redimir_sobre(self):  
        costo_sobre_fc_points = 50  
        costo_sobre_monedas = 100  
        nuevo_jugador_valoracion = random.randint(70, 90)  
  
        if self.fc_points >= costo_sobre_fc_points:
```

```

        self.fc_points -= costo_sobre_fc_points

self.plantilla.append(Jugador(nuevo_jugador_valoracion))

        elif self.monedas >= costo_sobre_monedas:

            self.monedas -= costo_sobre_monedas

self.plantilla.append(Jugador(nuevo_jugador_valoracion))

        else:

            print("No hay suficientes FC Points ni Monedas para
redimir un sobre.")

def comprar_jugador(self, valoracion):

    costo = valoracion * 10

    if self.monedas >= costo:

        self.monedas -= costo

        self.fc_points += 10

        self.plantilla.append(Jugador(valoracion))

    else:

        print("No hay suficientes Monedas para comprar el
jugador.")

class TestJuegoFCMobile(unittest.TestCase):

    def setUp(self):

        # Inicialización del juego para cada prueba

        self.juego = JuegoFCMobile()

```

```
def test_creacion_juego(self):  
    # Entonces el juego inicia con 500 FC Points, 1000 Monedas  
y 5 jugadores en la plantilla  
  
    self.assertEqual(self.juego.fc_points, 500)  
    self.assertEqual(self.juego.monedas, 1000)  
    self.assertEqual(len(self.juego.plantilla), 5)  
    self.assertEqual(self.juego.valoracion_promedio(), 70)  
  
def test_valoracion_promedio(self):  
    # Dado un equipo con jugadores de valoración [70, 70, 70,  
80, 80, 80]  
  
    self.juego.plantilla = [Jugador(70), Jugador(70),  
Jugador(70), Jugador(80), Jugador(80), Jugador(80)]  
  
    # Cuando calculo la valoración promedio  
    promedio = self.juego.valoracion_promedio()  
  
    # Entonces el promedio debe ser 75  
    self.assertEqual(promedio, 75)  
  
def test_carga_fc_points_mejora_jugador(self):  
    # Dado un equipo con valoraciones [70, 70, 71, 72, 73] y  
100 FC Points  
  
    self.juego.plantilla = [Jugador(70), Jugador(70),  
Jugador(71), Jugador(72), Jugador(73)]  
  
    self.juego.fc_points = 100  
  
    # Cuando agrego 100 FC Points  
    self.juego.agregar_fc_points(100)  
  
    # Entonces el peor jugador (70) mejora en 1 punto
```

```

        valoraciones = [jugador.valoracion for jugador in
self.juego.plantilla]

        self.assertEqual(valoraciones, [71, 70, 71, 72, 73])

        # Los FC Points resultan en 200

        self.assertEqual(self.juego.fc_points, 200)

    def test_carga_fc_points_multiples_mejoras(self):

        # Dado un equipo con valoraciones [70, 71, 72, 73, 74] y
300 FC Points

        self.juego.plantilla = [Jugador(70), Jugador(71),
Jugador(72), Jugador(73), Jugador(74)]

        self.juego.fc_points = 300

        # Cuando agrego 300 FC Points (3 mejoras)

        self.juego.agregar_fc_points(300)

        # Entonces el equipo debe tener valoraciones [73, 71, 72,
73, 74]

        valoraciones = [jugador.valoracion for jugador in
self.juego.plantilla]

        self.assertEqual(valoraciones, [73, 71, 72, 73, 74])

        # Los FC Points resultan en 600

        self.assertEqual(self.juego.fc_points, 600)

    def test_carga_fc_points_mejora_maxima(self):

        # Dado un equipo con un jugador ya con valoración 90 y 200
FC Points

        self.juego.plantilla = [Jugador(90), Jugador(90),
Jugador(86), Jugador(88), Jugador(87)]

        self.juego.fc_points = 200

```



```
# Cuando agrego 500 FC Points (una mejora)

self.juego.agregar_fc_points(500)

# Entonces el jugador con valoración 86 mejora a 90 y el
resto se pierde

valoraciones = [jugador.valoracion for jugador in
self.juego.plantilla]

self.assertEqual(valoraciones, [90, 90, 90, 88, 87])

# Los FC Points resultan en 700

self.assertEqual(self.juego.fc_points, 700)

def test_carga_fc_points_sin_mejora_excedida(self):

# Dado un equipo con todos los jugadores en valoración 90 y
200 FC Points

self.juego.plantilla = [Jugador(90), Jugador(90),
Jugador(90), Jugador(90), Jugador(90)]

self.juego.fc_points = 200

# Cuando agrego 200 FC Points (2 mejoras)

self.juego.agregar_fc_points(200)

# Entonces ningún jugador debe mejorar, ya que todos tienen
90

valoraciones = [jugador.valoracion for jugador in
self.juego.plantilla]

self.assertEqual(valoraciones, [90, 90, 90, 90, 90])

# Los FC Points resultan en 400

self.assertEqual(self.juego.fc_points, 400)

def test_agregar_monedas(self):

# Dado que el jugador tiene 1000 Monedas
```

```

        self.juego.monedas = 1000

        # Cuando agrega 500 Monedas

        self.juego.agregar_monedas(500)

        # Entonces las Monedas resultan en 1500

        self.assertEqual(self.juego.monedas, 1500)

def test_redimir_sobre_fc_points(self):

    # Dado que el jugador tiene 500 FC Points y 1000 Monedas

    self.juego.fc_points = 500

    self.juego.monedas = 1000

    num_jugadores_inicial = len(self.juego.plantilla)

    # Cuando redime un sobre

    self.juego.redimir_sobre()

    # Entonces se descuentan 50 FC Points

    self.assertEqual(self.juego.fc_points, 450)

    # Y se añade un nuevo jugador a la plantilla

    self.assertEqual(len(self.juego.plantilla),
num_jugadores_inicial + 1)

def test_redimir_sobre_con_monedas(self):

    # Dado que el jugador tiene 40 FC Points y 1000 Monedas

    self.juego.fc_points = 40

    self.juego.monedas = 1000

    num_jugadores_inicial = len(self.juego.plantilla)

    # Cuando redime un sobre

    self.juego.redimir_sobre()

    # Entonces se descuentan 100 Monedas (FC Points
insuficientes)

```

```
self.assertEqual(self.juego.monedas, 900)

# Y se añade un nuevo jugador a la plantilla

self.assertEqual(len(self.juego.plantilla),
num_jugadores_inicial + 1)


def test_redimir_sobre_sin_recursos(self):

    # Dado que el jugador tiene 40 FC Points y 50 Monedas

    self.juego.fc_points = 40

    self.juego.monedas = 50

    num_jugadores_inicial = len(self.juego.plantilla)

    # Cuando intenta redimir un sobre

    self.juego.redimir_sobre()

    # Entonces no se realiza la redención

    self.assertEqual(self.juego.fc_points, 40)

    self.assertEqual(self.juego.monedas, 50)

    # Y no se añade un nuevo jugador a la plantilla

    self.assertEqual(len(self.juego.plantilla),
num_jugadores_inicial)


def test_comprar_jugador(self):

    # Dado que el jugador tiene 1000 Monedas y 500 FC Points

    self.juego.monedas = 1000

    self.juego.fc_points = 500

    num_jugadores_inicial = len(self.juego.plantilla)

    # Cuando compra un jugador con valoración 80

    self.juego.comprar_jugador(80)

    # Entonces se descuentan 800 Monedas

    self.assertEqual(self.juego.monedas, 200)
```

```

        # Y se añaden 10 FC Points

        self.assertEqual(self.juego.fc_points, 510)

        # Además, la plantilla debe tener un jugador adicional

        self.assertEqual(len(self.juego.plantilla),
num_jugadores_inicial + 1)

    def test_comprar_jugador_sin_monedas(self):

        # Dado que el jugador tiene 500 Monedas y 500 FC Points y 5
jugadores

        self.juego.monedas = 500

        self.juego.fc_points = 500

        # Cuando intenta comprar un jugador con valoración 80

        self.juego.comprar_jugador(80)

        # Entonces no se realiza la compra

        self.assertEqual(self.juego.monedas, 500)

        # Y no se añaden FC Points

        self.assertEqual(self.juego.fc_points, 500)

        # Además, la plantilla no debe tener un jugador adicional

        self.assertEqual(len(self.juego.plantilla), 5)

if __name__ == '__main__':

    unittest.main()

```

Tests Ejecutados en Python

```
Tests passed: 12 of 12 tests - 0ms

C:\Users\David\anaconda3\python.exe "C:/Program Files/JetBrains/PyCharm 2023.2.1/plugins/python/helpers/pycharm/_jb_pytest_runner.py" --path "D
Testing started at 9:45 p. m. ...
Launching pytest with arguments D:\Users\David\Documents\UNAL\Semestre 2024-1\Teoría de Lenguajes de Programación 2024-01\Módulo 4 - Programaci

===== test session starts =====
collecting ... collected 12 items

Punto 2.py::TestJuegoFCMobile::test_agregar_monedas PASSED [ 8%]
Punto 2.py::TestJuegoFCMobile::test_carga_fc_points_mejora_jugador PASSED [ 16%]
Punto 2.py::TestJuegoFCMobile::test_carga_fc_points_mejora_maxima PASSED [ 25%]
Punto 2.py::TestJuegoFCMobile::test_carga_fc_points_multiples_mejoras PASSED [ 33%]
Punto 2.py::TestJuegoFCMobile::test_carga_fc_points_sin_mejora_excedida PASSED [ 41%]
Punto 2.py::TestJuegoFCMobile::test_comprar_jugador PASSED [ 50%]
Punto 2.py::TestJuegoFCMobile::test_comprar_jugador_sin_monedas PASSED [ 58%]No hay suficientes Monedas para comprar el jugador.

Punto 2.py::TestJuegoFCMobile::test_creacion_juego PASSED [ 66%]
Punto 2.py::TestJuegoFCMobile::test_redimir_sobre_con_monedas PASSED [ 75%]
Punto 2.py::TestJuegoFCMobile::test_redimir_sobre_fc_points PASSED [ 83%]
Punto 2.py::TestJuegoFCMobile::test_redimir_sobre_sin_recursos PASSED [ 91%]No hay suficientes FC Points ni Monedas para redimir un sobre.

Punto 2.py::TestJuegoFCMobile::test_valoracion_promedio PASSED [100%]

===== 12 passed in 0.07s =====

Process finished with exit code 0
```

Punto 3

Utilizar el siguiente dataset:

<https://www.kaggle.com/datasets/shreyanshverma27/imdb-horror-chilling-movie-dataset?resource=download>

- Importar el dataset

```
import pandas as pd

df = pd.read_csv("Horror Movies IMDb.csv", sep=",") # Creacion
de un nuevo dataframe a partir de un dataset en un csv, tiene mas
argumentos como header, nrow, na_filter, sep
```

- Imprimir los primeros 5 datos

```
df.head(5)
```

Ejecutado en Python

Movie Title	Movie Year	Runtime	Genre	Rating	Director	Votes	Gross
0 Alien	1979	117	Horror, Sci-Fi	8.5	Ridley Scott	9,05,275	\$78.90M
1 Psycho	1960	109	Horror, Mystery, Thriller	8.5	Alfred Hitchcock	6,89,068	\$32.00M
2 The Shining	1980	146	Drama, Horror	8.4	Stanley Kubrick	10,51,582	\$44.02M
3 The Thing	1982	109	Horror, Mystery, Sci-Fi	8.2	John Carpenter	4,39,793	\$13.78M
4 Tumbbad	2018	104	Drama, Fantasy, Horror	8.2	Rahi Anil Barve	53,297	NaN

- Encontrar la longitud del dataset

```
len(df) # Cantidad de columnas o registros del dataset
```

Ejecutado en Python

```
836
```

- Imprimir los encabezados

```
df.columns
```

Ejecutado en Python

```
Index(['Movie Title', 'Movie Year', 'Runtime', 'Genre', 'Rating', 'Director',
      'Votes', 'Gross'],
      dtype='object')
```

- Obtener un sub dataFrame con películas desde 1980 hasta la actualidad

```
df_1980_present = df[df['Movie Year'] >= 1980]
df_1980_present
```

Ejecutado en Python

<< 1-10 776 rows x 8 columns pd.DataFrame

	Movie Title	Movie Year	Runtime	Genre	Rating	Director	Votes	Gross
2	The Shining	1980	146	Drama, Horror	8.4	Stanley Kubrick	10,51,582	\$44.02M
3	The Thing	1982	109	Horror, Mystery, Sci-Fi	8.2	John Carpenter	4,39,793	\$13.78M
4	Tumbbad	2018	104	Drama, Fantasy, Horror	8.2	Rahi Anil Barve	53,297	NaN
10	The Blue Elephant	2014	170	Drama, Horror, Mystery	8.0	Marwan Hamed	29,151	NaN
11	Shaun of the Dead	2004	99	Comedy, Horror	7.9	Edgar Wright	5,72,237	\$13.54M
13	Let the Right One In	2008	114	Drama, Fantasy, Horror	7.9	Tomas Alfredson	2,21,604	\$2.12M
15	Get Out I	2017	104	Horror, Mystery, Thriller	7.8	Jordan Peele	6,42,604	\$176.04M
16	Predator	1987	107	Action, Adventure, Horror	7.8	John McTiernan	4,35,400	\$59.74M
17	I Saw the Devil	2010	144	Action, Crime, Horror	7.8	Jee-woon Kim	1,37,559	\$0.13M
24	Evil Dead II	1987	84	Comedy, Horror	7.7	Sam Raimi	1,73,914	\$5.92M

- Dado el sub dataFrame anterior: encontrar la película más corta, la menos rentable y el promedio de duración

```
# Encontrar la película más corta en el sub dataFrame
pelicula_mas_corta =
df_1980_present.loc[df_1980_present['Runtime'].idxmin()]
print("Película más corta:")
```

```

print(pelicula_mas_corta)

print("")

# Encontrar la película menos rentable en el sub dataframe
df_1980_present_con_gross =
df_1980_present.dropna(subset=['Gross']).copy()
df_1980_present_con_gross.loc[:, 'Gross'] =
df_1980_present_con_gross['Gross'].replace('[\$,M]', '',
regex=True).astype(float)

# Filtrar las películas con Gross mayor a 0
df_1980_present_con_gross =
df_1980_present_con_gross[df_1980_present_con_gross['Gross'] >= 0]

# Encontrar la película menos rentable
pelicula_menos_rentable =
df_1980_present_con_gross.loc[df_1980_present_con_gross['Gross'].i
dxmin()]

print("Película menos rentable:")
print(pelicula_menos_rentable)

print("")

# Calcular el promedio de duración en el sub dataframe
promedio_duracion = df_1980_present['Runtime'].mean()
print("Promedio de duración:")
print(promedio_duracion)

```

Ejecutado en Python

```
Película más corta:  
Movie Title      Host II  
Movie Year       2020  
Runtime          57  
Genre            Horror, Mystery  
Rating           6.5  
Director         Rob Savage  
Votes            34,122  
Gross            NaN  
Name: 297, dtype: object
```

```
Película menos rentable:  
Movie Title      Ginger Snaps  
Movie Year       2000  
Runtime          108  
Genre            Drama, Fantasy, Horror  
Rating           6.8  
Director         John Fawcett  
Votes            49,563  
Gross            0.0  
Name: 202, dtype: object
```

```
Promedio de duración:  
101.10695876288659
```

- Encontrar la desviación estándar de las calificaciones de la película en el sub dataframe

```
# Encontrar la desviación estándar de las calificaciones de la  
película en el sub dataframe  
desviacion_estandar_rating = df_1980_present['Rating'].std()  
desviacion_estandar_rating
```

Ejecutado en Python

```
0.8838861854481449
```

- Encontrar el promedio de votos agrupados género

```
# Encontrar el promedio de votos agrupados por género
```



```
df_genero = df.copy()

df_genero['Votes'] = df_genero['Votes'].str.replace(',', '',
    '').astype(int)

promedio_votos_genero = df_genero.groupby('Genre')['Votes'].mean()

promedio_votos_genero
```

Ejecutado en Python

« ← 1-10 ▾ ➡ » Length: 79, dtype: float64 [pd.Series](#)

Genre	Votes
Action, Adventure, Comedy	70572.666667
Action, Adventure, Crime	81017.000000
Action, Adventure, Drama	75749.166667
Action, Adventure, Fantasy	179800.000000
Action, Adventure, Horror	161026.047619
Action, Comedy, Drama	55923.000000
Action, Comedy, Fantasy	49718.600000
Action, Comedy, Horror	118984.076923
Action, Crime, Drama	133878.666667
Action, Crime, Horror	112437.333333

- Encontrar los directores y el número de ocurrencias en el dataset, ordenados descendientemente

```
# Encontrar los directores y el número de ocurrencias en el
dataset, ordenados descendientemente

directores_ocurrencias = df['Director'].value_counts() # Está
ordenado descendientemente por defecto

directores_ocurrencias
```

Ejecutado en Python

<< 1-10 >> Length: 544, dtype: int64 pd.Series

	Director
John Carpenter	11
Wes Craven	11
David Cronenberg	8
James Wan	7
Guillermo del Toro	7
Paul W.S. Anderson	6
George A. Romero	6
Alexandre Aja	6
Sam Raimi	6
Christopher Landon	6

- Encontrar la películas mejor calificadas y que pertenezcan incluyan estos 3 géneros (Horror, Mystery, Sci-Fi)

```
# Encontrar las películas mejor calificadas que incluyan los
# géneros Horror, Mystery y Sci-Fi

peliculas_mejor_calificadas = df[
    df['Genre'].str.contains('Horror') &
    df['Genre'].str.contains('Mystery') &
    df['Genre'].str.contains('Sci-Fi')
].sort_values(by='Rating', ascending=False)

peliculas_mejor_calificadas
```

Ejecutado en Python

<< 1-10 >> 20 rows x 8 columns pd.DataFrame CSV

	Movie Title	Movie Year	Runtime	Genre	Rating	Director	Votes	Gross
3	The Thing	1982	109	Horror, Mystery, Sci-Fi	8.2	John Carpenter	4,39,793	\$13.78M
136	Timecrimes	2007	92	Horror, Mystery, Sci-Fi	7.1	Nacho Vigalondo	66,167	\$0.04M
184	Nope	2022	130	Horror, Mystery, Sci-Fi	6.8	Jordan Peele	2,32,097	\$123.28M
204	eXistenZ	1999	97	Horror, Mystery, Sci-Fi	6.8	David Cronenberg	1,03,653	\$2.84M
253	Bird Box	2018	124	Horror, Mystery, Sci-Fi	6.6	Susanne Bier	3,60,116	NaN
260	The Faculty	1998	104	Horror, Mystery, Sci-Fi	6.6	Robert Rodriguez	1,28,887	\$40.06M
307	Possessor	2020	103	Horror, Mystery, Sci-Fi	6.5	Brandon Cronenberg	41,017	NaN
323	Time Lapse	2014	104	Horror, Mystery, Sci-Fi	6.5	Bradley King	47,416	NaN
433	Color Out of Space	2019	111	Horror, Mystery, Sci-Fi	6.2	Richard Stanley	52,020	NaN
459	Open Grave	2013	102	Horror, Mystery, Sci-Fi	6.2	Gonzalo López-Gallego	30,633	NaN