12 – TIPOS PROPIOS

Jorge Muñoz

IIC1103 – Introducción a la Programación

DICE (Dado)

```
El dado d tiene 6 caras
|3|
|5|
|6|
|2|
El dado d esta en el 2
```

```
# INSTRUCCIONES

#Crear un dado (de 6 caras)
d = Dice(6)

#Consultar numero de caras (faces)
print("El dado d tiene", d.faces, "caras")

#Lanzarlo
d.roll()
print(d)

d.roll()
print(d)

d.roll()
print(d)

d.roll()
print(d)

#Consultar numero
print("El dado d esta en el", d.num)
```

```
Lanzando ...
Dado Profe 1 |14|
Dado Profe 2 |8|
Dado Clase 1 |5|
Dado Clase 2 |4|
Gana el Profe 22 a 9
```

```
Lanzando ...
Dado Profe 1 |8|
Dado Profe 2 |2|
Dado Clase 1 |7|
Dado Clase 2 |20|
Gana la Clase 27 a 10
```

```
Lanzando ...
Dado Profe 1 |19|
Dado Profe 2 |14|
Dado Clase 1 |20|
Dado Clase 2 |13|
Empatan a 33
```

```python
dp1 = Dice(20)
dp2 = Dice(20)

dc1 = Dice(20)
dc2 = Dice(20)

print("Lanzando ...")
dp1.roll()
dp2.roll()
dc1.roll()
dc2.roll()

print("Dado Profe 1", dp1)
print("Dado Profe 2", dp2)
print("Dado Clase 1", dc1)
print("Dado Clase 2", dc2)

puntosp = dp1.num + dp2.num
puntosc = dc1.num + dc2.num

if puntosp > puntosc:
    print("Gana el Profe", puntosp, "a", puntosc)
elif puntosp < puntosc:
    print("Gana la Clase", puntosc, "a", puntosp)
elif puntosp == puntosc:
    print("Empatan a", puntosc)
```

WHAT IF I TOLD YOU



THAT IT'S ALL ONE GREAT BIG LIE.

imgflip.com



- Crearlos nosotros mismos
- Técnica: Programación Orientada a **Objetos** (OO)

int

bool

str

list









Usar TIPOS PROPIOS

Definir TIPOS PROPIOS

USAR TIPOS PROPIOS



```
############################
# QUE TE TIENEN QUE DAR
############################

# NOMBRE: Dice

# ATRIBUTOS:
# - faces       -> numero de caras del dado
# - num         -> numero en que esta el dado


# CONSTRUCTOR (1 PARAMETROS):
# - un int: numero de caras del dado a crear

# METODOS
# roll()    -> No parametros. No retorna nada.
#                Cambia (al azar) el num del dado
```

```
# Para CREAR: nombre de la clase y parametros
# E.g., Crear un dado (de 6 caras)
d = Dice(6)

# Para consultar un ATRIBUTO: .atributo
# E.g., Consultar numero de caras (faces)
print("El dado d tiene", d.faces, "caras")

# Para usar METODO: .metodo(params)
#E.g., Lanzarlo (no params, no retorna nada)
d.roll()
print(d)

d.roll()
print(d)

d.roll()
print(d)

d.roll()
print(d)

# Acceder a un ATRIBUTO: .atributo
#E.g., Consultar numero
print("El dado d esta en el", d.num)
```

INTROMONS (Usar)

LukaBat

Pokémon Chileno

[GBA-HACK]





Intro



```
# NOMBRE: Intromon

# ATRIBUTOS:
# - nom         -> su nombre
# - maxat       -> su ataque maximo
# - maxdef      -> su defensa maxima
# - vida        -> su vida

# CONSTRUCTOR (3 PARAMETROS):
# - un str: nombre
# - un int: ataque maximo
# - un int: ataque maximo

# METODOS
# ataque()  -> retorna int con cuanto ataca (de 0 a maxat)
# defensa() -> retorna int con cuanto se defiende (de 0 a maxdef)
# vivo()    -> retorna bool si esta vivo (vida > 0)
# damage(d) -> recibe int con dano a inflingir (restar de vida)
#                y retorna si murio (vida < 0) por ese dano
```



```
Nombre J1? NachoJuan
Ataque Max J1? 70
Defensa Max J1? 30
Nombre J2? ManoloX
Ataque Max J2? 50
Defensa Max J2? 50
QUE EMPIECE EL COMBATE!
NachoJuan ataca con 49 y ManoloX se defiende con 24
ManoloX recibe 25 de danyo ...
... pero sigue vivo con 75 de vida
ManoloX ataca con 8 y NachoJuan se defiende con 19
NachoJuan recibe 0 de danyo ...
... pero sigue vivo con 100 de vida
NachoJuan ataca con 3 y ManoloX se defiende con 21
ManoloX recibe 0 de danyo ...
... pero sigue vivo con 75 de vida
ManoloX ataca con 26 y NachoJuan se defiende con 14
NachoJuan recibe 12 de danyo ...
... pero sigue vivo con 88 de vida
NachoJuan ataca con 7 y ManoloX se defiende con 28
ManoloX recibe 0 de danyo ...
... pero sigue vivo con 75 de vida
ManoloX ataca con 4 y NachoJuan se defiende con 13
NachoJuan recibe 0 de danyo ...
... pero sigue vivo con 88 de vida
NachoJuan ataca con 64 y ManoloX se defiende con 23
ManoloX recibe 41 de danyo ...
... pero sigue vivo con 34 de vida
ManoloX ataca con 17 y NachoJuan se defiende con 8
NachoJuan recibe 9 de danyo ...
... pero sigue vivo con 79 de vida
NachoJuan ataca con 53 y ManoloX se defiende con 8
ManoloX recibe 45 de danyo ...
... y murio :(
Ganaste NachoJuan quedandote 79 de vida
```

#1- Crear las variables de tipo Intromon



#2- Atacarse mutuamente hasta que uno ya no este vivo

```
if turno == 1: #Ataca el J1, se defiende J2
```



```
elif turno == 2: #Ataca el J2, se defiende J1
```



#3- Felicitar al ganador (el que quedo vivo)



```
#1- Crear las variables de tipo Intromon
nom1 = input("Nombre J1? ")
at1 = int(input("Ataque Max J1? "))
def1 = int(input("Defensa Max J1? "))
j1 = Intromon(nom1,at1,def1)

nom2 = input("Nombre J2? ")
at2 = int(input("Ataque Max J2? "))
def2 = int(input("Defensa Max J2? "))
j2 = Intromon(nom2,at2,def2)


#2- Atacarse mutuamente hasta que uno ya no este vivo
print("QUE EMPIECE EL COMBATE!")

turno = 1
ambos_vivos = True
while ambos_vivos == True:

        if turno == 1: #Ataca el J1, se defiende J2

            if j1.vivo() == True:
                print("Ganaste", j1.nom, "quedandote", j1.vida, "de vida")
            elif j2.vivo() == True:
                print("Ganaste", j2.nom, "quedandote", j2.vida, "de vida")

        elif turno == 2: #Ataca el J2, se defiende J1

            if j1.vivo() == True:
                print("Ganaste", j1.nom, "quedandote", j1.vida, "de vida")
            elif j2.vivo() == True:
                print("Ganaste", j2.nom, "quedandote", j2.vida, "de vida")

#3- Felicitar al ganador (el que quedo vivo)
if j1.vivo() == True:
    print("Ganaste", j1.nom, "quedandote", j1.vida, "de vida")
elif j2.vivo() == True:
    print("Ganaste", j2.nom, "quedandote", j2.vida, "de vida")
```

```python
turno = 1
ambos_vivos = True
while ambos_vivos == True:

    if turno == 1: #Ataca el J1, se defiende J2

        #Ataque
        a = j1.ataque()
        d = j2.defensa()
        print(j1.nom,"ataca con",a,"y",j2.nom,"se defiende con",d)

        #Danyo
        if a >= d:
            danyo = a - d
        else:
            danyo = 0
        vivo = j2.damage(danyo)
        print(j2.nom,"recibe",danyo,"de danyo ...")

        if vivo == True:
            print("... pero sigue vivo con",j2.vida,"de vida")
            turno = 2 #Cambiar turno
        else:
            print("... y murio :( ")
            ambos_vivos = False #Acabo el combate


    elif turno == 2: #Ataca el J2, se defiende J1

        #Ataque
        a = j2.ataque()
        d = j1.defensa()
        print(j2.nom,"ataca con",a,"y",j1.nom,"se defiende con",d)

        #Danyo
        if a >= d:
            danyo = a - d
        else:
            danyo = 0
        vivo = j1.damage(danyo)
        print(j1.nom,"recibe",danyo,"de danyo ...")
        if vivo == True:
            print("... pero sigue vivo con",j1.vida,"de vida")
            turno = 1 #Cambiar turno
        else:
            print("... y murio :( ")
            ambos_vivos = False #Acabo el combate
```

DEFINIR TIPOS PROPIOS

```
############################
# QUE TE TIENEN QUE DAR
############################

# NOMBRE: Dice

# ATRIBUTOS:
# - faces       -> numero de caras del dado
# - num         -> numero en que esta el dado


# CONSTRUCTOR (1 PARAMETROS):
# - un int: numero de caras del dado a crear

# METODOS
# roll()    -> No parametros. No retorna nada.
#                Cambia (al azar) el num del dado
```



```
class Dice:
    def __init__(self,n):
        self.faces = n
        self.num   = 1
```

```
d = Dice(6)
print(d)
```

```
<__main__.Dice object at 0x104aff580>
```



```
############################
# QUE TE TIENEN QUE DAR
############################

# NOMBRE: Dice

# ATRIBUTOS:
# - faces       -> numero de caras del dado
# - num         -> numero en que esta el dado


# CONSTRUCTOR (1 PARAMETROS):
# - un int: numero de caras del dado a crear

# METODOS
# roll()    -> No parametros. No retorna nada.
#                Cambia (al azar) el num del dado
```



```
class Dice:
    def __init__(self,n):
        self.faces = n
        self.num = 1
    def __str__(self):
        numstr = str(self.num)
        txt = "|"+numstr+"|"
        return txt
```

```
|3|
|5|
|6|
|2|
```

```
d.roll()
print(d)
d.roll()
print(d)
d.roll()
print(d)
d.roll()
print(d)
```



```
############################
# QUE TE TIENEN QUE DAR
############################

# NOMBRE: Dice

# ATRIBUTOS:
# - faces       -> numero de caras del dado
# - num         -> numero en que esta el dado


# CONSTRUCTOR (1 PARAMETROS):
# - un int: numero de caras del dado a crear

# METODOS
# roll()    -> No parametros. No retorna nada.
#                Cambia (al azar) el num del dado
```

```
class Dice:
    def __init__(self,n):
        self.faces = n
        self.num = 1

    def __str__(self):
        numstr = str(self.num)
        txt = "|"+numstr+"|"
        return txt

    def roll(self):
        self.num = random.randint(1,self.faces)
```

INTROMONS (Definir)

```
# NOMBRE: Intromon

# ATRIBUTOS:
# - nom         -> su nombre
# - maxat       -> su ataque maximo
# - maxdef      -> su defensa maxima
# - vida        -> su vida

# CONSTRUCTOR (3 PARAMETROS):
# - un str: nombre
# - un int: ataque maximo
# - un int: ataque maximo

# METODOS
# ataque()  -> retorna int con cuanto ataca (de 0 a maxat)
# defensa() -> retorna int con cuanto se defiende (de 0 a maxdef)
# vivo()    -> retorna bool si esta vivo (vida > 0)
# damage(d) -> recibe int con dano a inflingir (restar de vida)
#              y retorna si murio (vida < 0) por ese dano
```



```
print(j1)
NachoJuan(70/30)
```

```
class Intromon:

    def __init__(self,n,maxa,maxd):
        self.nom = n
        self.maxat = maxa
        self.maxdef = maxd
        self.vida = 100

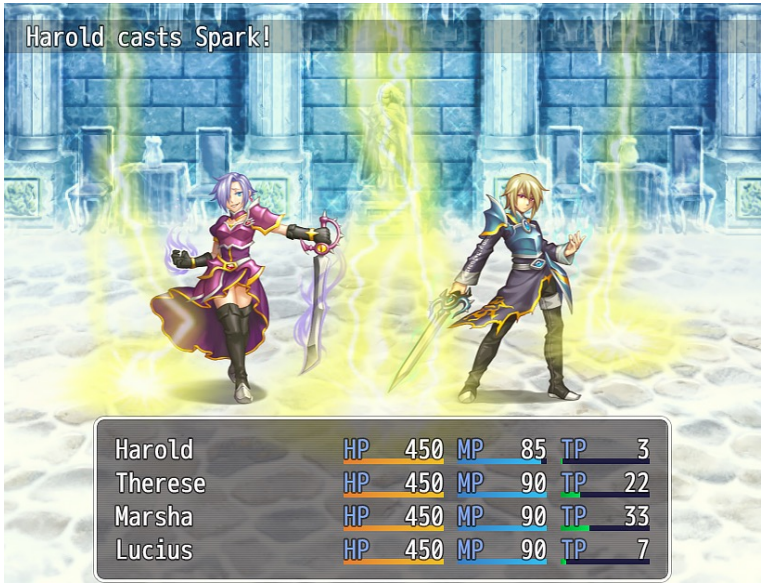

    def __str__(self):
        astr = str(self.maxat)
        dstr = str(self.maxdef)
        t = self.nom + "(" + astr + "/" + dstr +")"
        return t

    def ataque(self):
        return random.randint(1,self.maxat)

    def defensa(self):
        return random.randint(1,self.maxdef)

    def vivo(self):
        return self.vida > 0

    def damage(self,d):
        self.vida = self.vida - d
        res = self.vivo()
        return res
```





xchange.com/questions/190946/text-based-python-rpg-game

Stack Exchange NEW Search on Code Review…

CODE REVIEW

Home
Questions
Tags
Users
Unanswered

Text-based Python RPG game

2

1

This is a little text-based Python adventure game I found ir
battle system, a shop, a save option and more. You can fig

```
import sys
import os
import random
import pickle

weapons = {"Great Sword":40}

class Player:
    def __init__(self, name):
        self.name = name
        self.maxhealth = 100
        self.health = self.maxhealth
        self.base_attack = 10
        self.gold = 40
        self.pots = 0
        self.weap = ["Rusty Sword"]
        self.curweap = ["Rusty Sword"]

    @property
    def attack(self):
        attack = self.base_attack
        if self.curweap == "Rusty Sword":
            attack += 5

        if self.curweap == "Great Sword":
            attack += 15

        return attack


class Goblin:
    def __init__(self, name):
        self.name = name
        self.maxhealth = 50
        self.health = self.maxhealth
        self.attack = 5
```

justinmeister / The-Stolen-Crown-RPG

<> Code Issues 0 Pull requests 0

GitHub is hom
and review

Branch: master ▾ The-Stolen-Crown-RPG / data

justinmeister Almost finished tmx map of town, a few m

1 contributor

14 lines (11 sloc) 369 Bytes

```
__author__ = 'justinarmstrong'
import pygame as pg
from .. import constants as c


class Portal(pg.sprite.Sprite):
    """Used to change level state"""
    def __init__(self, x, y, name):
        super(Portal, self).__init__()
        self.image = pg.Surface((32, 32))
        self.image.fill(c.BLACK)
        self.rect = pg.Rect(x, y, 32, 32)
        self.name = name
```

Cubilete



```python
import random

class Dice:
    def __init__(self,n):
        self.faces = n
        self.num = 1

    def __str__(self):
        numstr = str(self.num)
        txt = "|"+numstr+"|"
        return txt

    def roll(self):
        self.num = random.randint(1,self.faces)


# 'sumacubilete(c)'
# recibe un parametro 'c' list de Dice
# y retorna int con la suma de todos los dados
def sumacubilete(c):
    suma = 0
    for d in c:
        suma += d.num
    return suma
```

```python
#TESTCASE
d1 = Dice(6)
d2 = Dice(6)
d3 = Dice(6)
d4 = Dice(6)
d5 = Dice(6)
#Dados en num que quiero
d1.num = 6
d2.num = 5
d3.num = 6
d4.num = 5
d5.num = 3
#Cubilete
cubilete = [d1,d2,d3,d4,d5]
suma = sumacubilete(cubilete)
print("Correcto:",25,"Obtenido:", suma)
```

Correcto: 25 Obtenido: 25

Wok


NUESTRO MENU
WOK SIMPLE:
BASE* + 1 TOPPING A + 1 TOPPING B + SALSA
$4.190
WOK SUPER:
BASE* + 1 TOPPING A + 2 TOPPING B + SALSA
$4.490
WOK DELUXE:
BASE* + 2 TOPPING A + 2 TOPPING B + SALSA
$4.990
BASE
TOPPING A
TOPPING B
SALSA
Noodle de Huevo
Noodle de Arroz
Noodle Integral
Arroz Jazmín
Arroz Salvaje
Mix Vegano
Carne
Camarón
Pollo
Cerdo
Tofu
Brócoli
Mini choclo
Champiñón
Cebollín
Pimentón
Zapallo italiano
Cebolla morada
Dientes de dragón
Coco-Curry
Take a wok
Teriyaki
Mani Thai
Tomate Sweet & Sour
Soya
* Todas las bases tienen verduras y huevo
** Agrega maní o almendras por $250
Diario de un Auto-Exilio
BEBIDAS*

```python
class Ingrediente:
    def __init__(self,n,pic,pre):
        self.nom = n
        self.picante = pic
        self.precio = pre

    def __str__(self):
        txt = self.nom
        if self.picante == True:
            txt = txt + "*"
        return txt


class Wok:
    def __init__(self,c,f,t,s):
        self.cliente = c
        self.fondo = f
        self.topping = t
        self.salsa = s

    def calcular_precio(self):
        total = ( self.fondo.precio +
                  self.topping.precio +
                  self.salsa.precio)
        return total
```

```python
#TESTCASE
noodles = Ingrediente("Noodles", False, 10)
jazmin = Ingrediente("Arroz jazmin", False,10)
merken = Ingrediente("Arroz merken",True,20)

pollo = Ingrediente("Pollo", False, 10)
tofu = Ingrediente("Tofu", False, 20)
cerdo = Ingrediente("Cerdo Picante", True, 30)

curry = Ingrediente("Salsa Curry",True,10)
soya = Ingrediente("Salsa Soya", False, 10)
teriyaki = Ingrediente("Salsa Teriyaki", True, 10)

w = Wok("Jorge",noodles,cerdo,curry)

print("El Wok de", w.cliente, "cuesta", w.calcular_precio(), "...")
print("... y lleva", w.fondo, w.topping, w.salsa)
```

El Wok de Jorge cuesta 50 ...
... y lleva Noodles Cerdo Picante* Salsa Curry*

NERDFLIX

1. Nerdflix (Ex 2020-1 P3a)

Objetivo Parte A

Para esta parte deberás modelar la clase `Pelicula`, que tiene los siguientes atributos:

- **`nombre`**: corresponde al nombre de la película, y es único para cada una.
- **`ano`**: corresponde al año de lanzamiento de la película

Deberás definir la clase con los métodos **`__init__`** y **`posterior:`**

- El método **`__init__`** recibe como argumento un titulo de tipo *str*, que representa el `nombre` de la película, y un año de lanzamiento de tipo *int* , en este orden.

- El método **`posterior`** recibe como argumento un año de tipo *int* y retorna **`True`** si la película fue lanzada ese año o en un año posterior. En caso contrario deberá retornar **`False`**. Por ejemplo si el método recibe el año 2010 y el año de la película es 2000 entonces éste debe retornar **`False`** dado que el año de la película no es ni igual ni posterior al año entregado

Lo único que tienes que hacer es implementar la clase. Si respetas el orden de los atributos, el nombre de la clase y de su método, entonces no tendrás problemas con el testeo de tu ejercicio. De todas maneras, a continuación se encuentra la explicación del formato de input y output que debe seguir tu código:

```
class Pelicula:
    def __init__(self,t,y):
        self.nombre = t
        self.ano = y

    def posterior(self,y):
        res = self.ano >= y
        return res
```

Introducción



La aplicación Nerdflix busca ofrecer las mejores películas de computación. Tu objetivo es modelar este programa usando python para poder llevarlo al mercado y hacerte rico con el dinero de los nerds.

Objetivo Parte B

Para esta parte deberás modelar la clase `Usuario`, que tiene los siguientes atributos:

- **`nombre`**: corresponde al nombre (*str*) del usuario, y es único para cada uno.
- **`contrasena`**: corresponde a la contraseña (*str*) del usuario.

Deberás definir la clase con los métodos **`__init__`** y **`pass_fragil`**.

- El método **`__init__`** recibe como argumento un nombre de tipo *str* y una contraseña de tipo *str* , en este orden.

- El método **`pass_fragil`** no recibe argumentos y retorna **`True`** si la contraseña del usuario es frágil, y **`False`** si no. Una contraseña se considera frágil si tiene menos de 8 caracteres o si no tienen ningún número. Por ejemplo si el método recibe la contraseña ´holi´, ésta debe retornar **`True`** dado que se cumplen ambas condiciones de una contraseña frágil, es decir tiene menos de 8 caracteres y no contiene números.

Lo único que tienes que hacer es implementar la clase. Si respetas el orden de los atributos, el nombre de la clase y de su método, entonces no tendrás problemas con el testeo de tu ejercicio. De todas maneras, a continuación se encuentra la explicación del formato de input y output que debe seguir tu código:

```python
class Usuario:
    def __init__(self,n,c):
        self.nombre = n
        self.contrasena = c

    def pass_fragil(self):
        corta = len(self.contrasena) < 8

        tienenums = False
        #Mirar si tiene 0, 1, 2, ...
        for i in range(0,10):
            si = str(i)
            if si in self.contrasena:
                tienenums = True

        fragil = corta or (not tienenums)
        return fragil
```

Objetivo Parte C

Situación Real

Para esta parte deberás modelar la clase `Nerdflix`, la cual debe tener los siguietes atributos:

- **`peliculas`**: lista de películas que guarda instancias de la clase `Pelicula` definida en la Parte A. Inicialmente comienza vacía y no se recibe como parámetro.
- **`usuarios`**: lista de usuarios que guarda instancias de la clase `Usuario` definida en la Parte B. Inicialmente comienza vacía y no se recibe como parámetro.
- **`visualizaciones`**: lista de listas donde cada elemento está en el formato `[usuario, pelicula]`, las cuales representan que el usuario almacenado en la instancia `usuario` ha visto la película almacenada en la instancia `pelicula`. Inicialmente comienza vacía y no se recibe como parámetro.

Dado lo anterior, la clase `Nerdflix` no recibe parametros.

Además del constructor, deberás definir los siguiente métodos de la clase `Nerdflix`:

- **`agregar_usuario`**`(self, u)`: recibe como parámetro `(u)` una instancia de la clase `Usuario` y la agrega a la lista de usuarios en la aplicación.
- **`agregar_pelicula`**`(self, p)`: recibe como parámetro `(p)` una instancia de la clase `Pelicula` y la agrega a la lista de películas en la aplicación.
- **`peli_disponible`**`(self, n)`: recibe como parámetro `(n)` un *string* que representa el título de una película y retorna `True` si la película está disponible en la aplicación (es decir, si el título se encuentra en la base de datos de Nerdflix) y `False` en caso contrario.
- **`users_fragiles`**`(self)`: no recibe parámetros y retorna una lista con todos los nombres *string* de los usuarios en la lista de usuarios de `Nerdflix` que poseen una contraseña frágil. Para esto, considera que puedes llamar al método `pass_fragil` de la clase `Usuario` definida en la Parte B.
- **`nuevo_visionado`**`(self, u, p)`: recibe como parámetro una instancia de la clase `Usuario` `(u)` y una instancia de la clase `Pelicula` `(p)`. El método debe guardar en la lista de visualizaciones que el usuario `u` vió la película `p`.



El formato que utilices para crear la clase queda a tu criterio, pero deberá correr con el código entregado como input. **Lo único que tienes que hacer es implementar la clase.**

No te preocupes si no lograste implementar correctamente las clases `Pelicula` y `Usuario`. Estas ya estan correctamente implementadas e importadas

Resolución Problema

```python
class Nerdflix:
    def __init__(self):
        self.peliculas = []
        self.usuarios = []
        self.visualizaciones = []

    def agregar_usuario(self,u):
        self.usuarios.append(u)

    def agregar_pelicula(self,p):
        self.peliculas.append(p)

    def peli_disponible(self,n):
        disponible = False
        for p in self.peliculas:
            if p.name == n:
                disponible = True
        return disponible

    def users_fragiles(self):
        fragiles = []
        for u in self.usuarios:
            if u.pass_fragil():
                fragiles.append(u.nombre)
        return fragiles

    def nuevo_visionado(self,u,p):
        v = [u,p]
        self.visualizaciones.append(v)
```



Objetivo Parte C

Para esta parte deberás modelar la clase `Nerdflix`, la cual debe tener los siguietes atributos:

- **`peliculas`**: lista de películas que guarda instancias de la clase `Pelicula` definida en la Parte A. Inicialmente comienza vacía y no se recibe como parámetro.
- **`usuarios`**: lista de usuarios que guarda instancias de la clase `Usuario` definida en la Parte B. Inicialmente comienza vacía y no se recibe como parámetro.
- **`visualizaciones`**: lista de listas donde cada elemento está en el formato `[usuario, pelicula]`, las cuales representan que el usuario almacenado en la instancia `usuario` ha visto la película almacenada en la instancia `pelicula`. Inicialmente comienza vacía y no se recibe como parámetro.

Dado lo anterior, la clase `Nerdflix` no recibe parametros.

Además del constructor, deberás definir los siguiente métodos de la clase `Nerdflix`:

- **`agregar_usuario`**`(self, u)`: recibe como parámetro (`u`) una instancia de la clase `Usuario` y la agrega a la lista de usuarios en la aplicación.
- **`agregar_pelicula`**`(self, p)`: recibe como parámetro (`p`) una instancia de la clase `Pelicula` y la agrega a la lista de películas en la aplicación.
- **`peli_disponible`**`(self, n)`: recibe como parámetro (`n`) un *string* que representa el título de una película y retorna `True` si la película está disponible en la aplicación (es decir, si el título se encuentra en la base de datos de Nerdflix) y `False` en caso contrario.
- **`users_fragiles`**`(self)`: no recibe parámetros y retorna una lista con todos los nombres *string* de los usuarios en la lista de usuarios de `Nerdflix` que poseen una contraseña frágil. Para esto, considera que puedes llamar al método `pass_fragil` de la clase `Usuario` definida en la Parte B.
- **`nuevo_visionado`**`(self, u, p)`: recibe como parámetro una instancia de la clase `Usuario` (`u`) y una instancia de la clase `Pelicula` (`p`). El método debe guardar en la lista de visualizaciones que el usuario `u` vió la película `p`.
- **`max_visionador`**`(self, n)`: recibe como parámetro (`n`) un *string* que representa el título de una película y retorna el nombre *string* del usuario que más visualizaciones tiene de la película `n`. Si ningún usuario tiene visualizaciones de la película retorna un *string* vacío. En caso de haber más de un usuario con la mayor cantidad de vistas puede retornar cualquiera. Puedes asumir que la película `n` siempre estará disponible en la aplicación.

El formato que utilices para crear la clase queda a tu criterio, pero deberá correr con el código entregado como input. **Lo único que tienes que hacer es implementar la clase.**

No te preocupes si no lograste implementar correctamente las clases `Pelicula` y `Usuario`. Estas ya estan correctamente implementadas e importadas




I'LL WAIT
FOR YOU HERE



Objetivo Parte C

Para esta parte deberás modelar la clase `Nerdflix`, la cual debe tener los siguietes atributos:

- **`peliculas`**: lista de películas que guarda instancias de la clase `Pelicula` definida en la Parte A. Inicialmente comienza vacía y no se recibe como parámetro.
- **`usuarios`**: lista de usuarios que guarda instancias de la clase `Usuario` definida en la Parte B. Inicialmente comienza vacía y no se recibe como parámetro.
- **`visualizaciones`**: lista de listas donde cada elemento está en el formato `[usuario, pelicula]`, las cuales representan que el usuario almacenado en la instancia `usuario` ha visto la película almacenada en la instancia `pelicula`. Inicialmente comienza vacía y no se recibe como parámetro.

Dado lo anterior, la clase `Nerdflix` no recibe parametros.

Además del constructor, deberás definir los siguiente métodos de la clase `Nerdflix`:

- **`agregar_usuario`**`(self, u)`: recibe como parámetro (`u`) una instancia de la clase `Usuario` y la agrega a la lista de usuarios en la aplicación.
- **`agregar_pelicula`**`(self, p)`: recibe como parámetro (`p`) una instancia de la clase `Pelicula` y la agrega a la lista de películas en la aplicación.
- **`peli_disponible`**`(self, n)`: recibe como parámetro (`n`) un *string* que representa el título de una película y retorna `True` si la película está disponible en la aplicación (es decir, si el título se encuentra en la base de datos de Nerdflix) y `False` en caso contrario.
- **`users_fragiles`**`(self)`: no recibe parámetros y retorna una lista con todos los nombres *string* de los usuarios en la lista de usuarios de `Nerdflix` que poseen una contraseña frágil. Para esto, considera que puedes llamar al método `pass_fragil` de la clase `Usuario` definida en la Parte B.
- **`nuevo_visionado`**`(self, u, p)`: recibe como parámetro una instancia de la clase `Usuario` (`u`) y una instancia de la clase `Pelicula` (`p`). El método debe guardar en la lista de visualizaciones que el usuario `u` vió la película `p`.
- **`max_visionador`**`(self, n)`: recibe como parámetro (`n`) un *string* que representa el título de una película y retorna el nombre *string* del usuario que más visualizaciones tiene de la película `n`. Si ningún usuario tiene visualizaciones de la película retorna un *string* vacío. En caso de haber más de un usuario con la mayor cantidad de vistas puede retornar cualquiera. Puedes asumir que la película `n` siempre estará disponible en la aplicación.

El formato que utilices para crear la clase queda a tu criterio, pero deberá correr con el código entregado como input. **Lo único que tienes que hacer es implementar la clase.**

No te preocupes si no lograste implementar correctamente las clases `Pelicula` y `Usuario`. Estas ya estan correctamente implementadas e importadas

Resolución Problema

```
def max_visionador(self,n):
    #Crear 2 listas con nombres que visualizaron
    #Una sin repetidos y otra con repetidos
    sinrep = []
    conrep = []
    for v in self.visualizaciones:
        u = v[0]
        p = v[1]
        if p.name == n:
            conrep.append(u.nombre)
            if u.nombre not in sinrep:
                sinrep.append(u.nombre)

    #Para cada usuario (sin repetidos) mirar
    #cuantas veces ha visto la peli
    #y quedarse con el maximo
    maxnom = ""
    maxnum = -1
    for nom in sinrep:
        #Cuantas veces la ha visto
        num = 0
        for x in conrep:
            if nom == x:
                num += 1

        #Max
        if num > maxnum:
            maxnum = num
            maxnom = nom

    return maxnom
```

DCCHURRASCO



Pregunta 3

¡Bienvenido al restaurante *DCChurrasco*! El siguiente código usa la programación orientada a objetos y las clases de objetos `Mesa` y `Restaurante` para gestionar el restaurante. En concreto 1) Cuando llega un grupo de clientes, el sistema los sienta en una mesa libre, o les dice que no hay mesas disponibles, 2) Se puede pedir comida para una mesa, indicando el plato y su precio, 3) Al pedir la cuenta el sistema indica el número de platos consumidos por la mesa, el precio total y el precio por persona, en partes iguales, quedando la mesa libre en ese momento, 4) Da la opción de salir del programa. Considera que todas las mesas tienen 4 sillas y que siempre llegarán grupos de al menos 1 persona y a lo más 4 personas.

```
r = Restaurante('DCChurrasco', 2) # restaurante llamado 'DCChurrasco' con 2 mesas vacías

continuar = True
while continuar:
    opcion = int(input('1-Llegan clientes 2-Pedir comida 3-La cuenta 9-Salir: '))

    if opcion == 1:
        cantidad_clientes = int(input('Número de clientes: '))
        numero_mesa = r.llegan_clientes(cantidad_clientes)
        if numero_mesa == -1:
            print('No hay mesas disponibles :(')
        else:
            print('Siéntense en la mesa', numero_mesa)

    elif opcion == 2:
        numero_mesa = int(input('Número de mesa: '))
        plato = input('Plato: ')
        precio = int(input('Precio: '))
        r.mesas[numero_mesa].agregar_plato(plato, precio)

    elif opcion == 3:
        numero_mesa = int(input('Número de mesa: '))
        mesa = r.mesas[numero_mesa]
        print('Pidieron', len(mesa.obtener_platos()), 'platos')
        print('La cuenta son', mesa.obtener_cuenta())
        print('Eso hace', mesa.obtener_cuenta_por_persona(), 'por persona')
        mesa.vaciar_mesa()

    elif opcion == 9:
        continuar = False
        print('Saliendo')
```

IIC1103 – Introducción a la Programación
2018 - 1

Interrogación 2

Situación Real

Se te pide que, considerando el código anterior, implementes las clases `Mesa` y `Restaurante` con los siguientes métodos:

1) **`Mesa:`**

- **(7 puntos)** `__init__`: Inicializa una mesa vacía. No recibe ningún parámetro.
- **(7 puntos)** `ocupar_mesa`: Establece la mesa como ocupada. Recibe un parámetro: un `int` correspondiente al número de personas que se sientan en la mesa. No retorna nada.
- **(4 puntos)** `esta_vacia`: Retorna `False` si la mesa se encuentra ocupada, `True` en el caso contrario. No recibe ningún parámetro.
- **(4 puntos)** `agregar_plato`: Registra un nuevo plato a los platos pedidos por la mesa. Recibe como parámetros el nombre del plato como un `string` y el precio del plato como un `int`.
- **(4 puntos)** `obtener_cuenta`: Retorna un `int` con la suma de precios de todos los platos pedidos en una mesa. No recibe ningún parámetro.
- **(4 puntos)** `obtener_cuenta_por_persona`: Retorna un `float` con lo que debe pagar cada cliente. Se asume que dividen la cuenta en partes iguales. **Es necesario que se use el método `obtener_cuenta` para hacer este cálculo.** No recibe ningún parámetro.
- **(4 puntos)** `obtener_platos`: Retorna una `lista` de `string` con los platos pedidos por la mesa. No recibe ningún parámetro.
- **(7 puntos)** `vaciar_mesa`: Establece la mesa como vacía, cambiando los atributos necesarios para que pueda ser usada nuevamente por otro grupo de clientes.

2) **`Restaurante:`**

- **(12 puntos)** `__init__`: Inicializa un restaurante. Recibe como parámetro el nombre del restaurante como un `string` y un `int` con el número de mesas vacías (de la clase `Mesa`) que tiene el restaurante al crearse.
- **(7 puntos)** `llegan_clientes`: Recibe un `int` con la cantidad de clientes que llegaron. En caso de que exista una mesa vacía, la mesa queda ocupada con los clientes, y el método retorna un `int` con el número que identifica la mesa. Considera que las mesas de un restaurante se identifican con un número del 0 al número de mesas - 1. En caso de que haya más de una mesa vacía, el sistema puede elegir cualquiera. En caso de que no haya ninguna mesa vacía, el método simplemente retorna -1.

```python
class Mesa:
    def __init__(self):
        self.ocupantes = 0
        self.cuenta = 0
        self.platos = []

    def ocupar_mesa(self, num_ocupantes):
        self.ocupantes = num_ocupantes

    def esta_vacia(self):
        return self.ocupantes == 0

    def agregar_plato(self, plato, precio):
        self.cuenta += precio
        self.platos.append(plato)

    def obtener_cuenta(self):
        return self.cuenta

    '''Obligado usar obtener_cuenta()'''
    def obtener_cuenta_por_persona(self):
        return self.obtener_cuenta() / self.ocupantes

    def obtener_platos(self):
        return self.platos

    def vaciar_mesa(self):
        self.ocupantes = 0
        self.cuenta = 0
        self.platos = []
```

```
class Restaurante:

    def __init__(self, nombre, num_mesas):
        self.nombre = nombre
        self.mesas = []
        for i in range(0,num_mesas):
            self.mesas.append(Mesa())

    def llegan_clientes(self, num_comensales):
        ix = -1
        for i in range(0,len(self.mesas)):
            mesa = self.mesas[i]
            if ix == -1 and mesa.esta_vacia():
                ix = i
                mesa.ocupar_mesa(num_comensales)
        return ix
```