Pauta_Auxiliar_11_Objetos_y_Clases

December 10, 2020

1 Auxiliar 11 - Objetos y Clases

1.1 P1 Amigo secreto

Creamos las siguientes clases que nos permitirán modelar el juego amigo secreto.

1.1.1 A)

Defina la clase **Persona**. Esta debe tener los campos nombre y regala_a. El primero será un string y el segundo una Persona.

[4]:
```
# A
# Campos :
# nombre : str
# regala_a : Persona
class Persona():
    def __init__(self, nombre = "", regala_a=None):
        self.nombre = nombre
        self.regala_a = regala_a
```

1.1.2 B)

Defina la clase **Regalo**. Esta debe tener los campos nombre y destinatario. El primero será un string y el segundo una Persona.

[5]:
```
# B
# Campos :
# nombre : str
# destinatario : Regalo
class Regalo():
    def __init__(self, nombre = "", destinatario=None):
        self.nombre = nombre
        self.destinatario = destinatario
```

1.1.3 C)

Defina la clase **Juego**. Esta tendrá los campos participantes y regalos. El primero será una lista de personas y el segundo una lista de regalos.

1.1.4 D)

Defina el método sorteo para la clase **Juego**. Este debe devolver una lista que represente a quién le entregará un regalo la persona que tiene la misma posición en la lista participantes.

Utilice la función **shuffle del módulo random** de python en su implementación. Además, recuerde que una persona no puede ser amiga secreta de si misma y que todos deben recibir un regalo.

1.1.5 E)

Defina el método **asignarAmigos** para la clase Juego. Esta función recibe una lista de personas, que representa a quienes se les hará un regalo a partir de su posición en la lista, es decir, la persona con indice i recibirá un regalo de la persona en la lista de participantes que tenga la posición i.

El método deberá setear el valor regala_a para todas las personas de la lista de participantes. Además, para el caso en que las listas tienen un tamaño diferente se debe imprimir un error.

1.1.6 F)

Defina el método **entregaRegalos** para la clase Juego. Este debe mostrar en pantalla a todos los participantes, con el regalo que hicieron y a quién se lo entregaron.

[6]:
```python
import random

# C
# Campos :
# participantes : list(Persona)
# regalos : list(Regalo)
class Juego():
    def __init__(self, participantes = [], regalos = []):
        self.participantes = participantes
        self.regalos = regalos

    # D
    # None -> list(Persona)
    # Retorna una lista que representa a quién le entregará un regalo la␣
 ↪persona
    # que tiene la misma posición en la lista participantes.
    def sorteo(self):
        copy_list = self.participantes.copy()
        while True:
            random.shuffle(copy_list)
            auto_regalo = False
            for indice in range(len(copy_list)):
                if copy_list[indice] == self.participantes[indice]:
                    auto_regalo = True
                    break
            if not auto_regalo:
                return copy_list
    #E
```

```
    # list(Persona) -> None
    # Efecto: Cambia el valor regala_a para todas las personas de la lista de␣
 ↪participantes,
    # a partir de la lista que se recibe.
    # En caso que las listas tengan tamaño difrente se imprime un error y no se␣
 ↪cambia la lista.
    def asignarAmigos(self, lista_sorteo):
        if len(self.participantes) != len(lista_sorteo):
            print("Listas deben ser del mismo tamaño")
        else:
            for i in range(len(lista_sorteo)):
                self.participantes[i].regala_a = lista_sorteo[i]

    #F
    # Imprime en pantalla a todos los participantes junto con el regalo que␣
 ↪hicieron y a quiénes se lo entregaron.
    def entregaRegalos(self):
        for regalo in self.regalos:
            destinatario = regalo.destinatario
            remitente = None
            for partipante in self.participantes:
                if(destinatario == partipante.regala_a):
                    remitente = partipante
                    break
            print(remitente.nombre + " regaló " + regalo.nombre + " a " +␣
 ↪destinatario.nombre)

### Test
p1 = Persona("Patana")
p2 = Persona("Juanin")
p3 = Persona("Tulio")

r1 = Regalo("Traje de detective", p1)
r2 = Regalo("Cámara", p2)
r3 = Regalo("Reloj Relox", p3)

lista_participantes = [p1, p2, p3]
regalos = [r1,r2,r3]
j = Juego(lista_participantes, regalos)
## Test sorteo
lista_destinatarios=j.sorteo()
participantes1=j.participantes
assert p1 in lista_destinatarios and p2 in lista_destinatarios and p3 in␣
 ↪lista_destinatarios
assert participantes1[0]!=lista_destinatarios[0] and participantes1[1]!
 ↪=lista_destinatarios[1] and participantes1[2]!=lista_destinatarios[2]
```

```
## Test asignarAmigos
participantes2=j.participantes
assert participantes2[0].regala_a == None and participantes2[1].regala_a ==␣
 ↪None and participantes2[2].regala_a == None
nueva_lista_destinatarios=[p3,p1,p2]
j.asignarAmigos(nueva_lista_destinatarios)
assert participantes2[0].regala_a == p3 and participantes2[1].regala_a == p1␣
 ↪and participantes2[2].regala_a == p2

## Correr entregaRegalos
j.entregaRegalos()
```

```
Juanin regaló Traje de detective a Patana
Tulio regaló Cámara a Juanin
Patana regaló Reloj Relox a Tulio
```

1.2 P2. Polinomios

Un polinomio de grado n se puede representar como una lista de n+1 coeficientes. ### A) Diseñe una clase **Pol** que permita ser utilizada de la siguiente manera:

Pol([a]) crea el polinomio *a*, *Pol([b, a])* crea *ax+b*, *Pol([c, b, a])* crea *ax^2+bx+c* y así sucesivamente.

1.2.1 B)

Defina la función **p.evaluar(x)**, la cual evalúa el argumento x en el polinomio p.

1.2.2 C)

Defina la función **p.toString()**, la que devuelve un string que representa al polinomio p.

1.2.3 D)

Defina la función **p1.sumar(p2)**, la cual recibe 2 polinomios y devuelve un polinomio correspondiente a la suma de estos.

Para ello, cree y luego utilice la función auxiliar **max_lista**, la cual reciba 2 listas y devuelva la lista que tenga el mayor largo, en caso, de que ambas tengan el mismo largo devuelva cualquiera.

[8]:
```
# Campos :
# coeficientes : lista
class Pol():
### A)
    # Constructor
    def __init__(self, coeficientes=[]):
        self.coeficientes = coeficientes
### B)
### num-> num
### Evalua x en el polinomio
    def evaluar(self, x):
```

```
        largo = len(self.coeficientes)
        resultado = 0
        for indice in range(largo):
            coeficiente = self.coeficientes[indice]
            x_n = x**indice
            resultado = resultado+ coeficiente*x_n
        return resultado
### C)
### None -> str
### Devuelve un string que representa al polinomio
    def toString(self):
        largo = len(self.coeficientes)
        if largo == 0:
            return ""
        respuesta= str(self.coeficientes[0])
        for indice in range(1,largo):
            coeficiente = str(self.coeficientes[indice])
            if coeficiente != "0":
                x_n = "x^"+ str(indice)
                respuesta = coeficiente+x_n +" + "+respuesta
        return respuesta

### D)
### Pol -> Pol
### Devuelve la suma del polinomio con otro polinomio.
    def sumar(self, pol):
        min_largo = min(len(self.coeficientes), len(pol.coeficientes))
        max_lista_coef = max_lista(self.coeficientes, pol.coeficientes)
        coeficientes_sumados = []
        for i in range(min_largo):
            suma = self.coeficientes[i] + pol.coeficientes[i]
            coeficientes_sumados.append(suma)
        for i in range(min_largo, len(max_lista_coef)):
            coeficientes_sumados.append(max_lista_coef[i])
        return Pol(coeficientes_sumados)

    ### max_lista: list list -> list
    ### Devuelve la lista que contenga el mayor número de elemntos,
    ### y en caso de empate devuelve la segunda lista.
    ### Ej: max_lista([1,2,3,4], [1,2,3]) devuelve [1,2,3,4]
    ### max_lista([1,2,3],[1,4,5]) entrega [1,4,5]

    global max_lista
    def max_lista(pol1, pol2):
        if len(pol1) > len(pol2):
            return pol1
        else:
```

```
            return pol2
    assert max_lista([1,2,3,4], [1,2,3]) == [1,2,3,4]
    assert max_lista([1,2,3],[1,4,5]) == [1,4,5]


### Test
p1 = Pol([1,1,-3,0,1])
p2 = Pol([-2,5,-1,1])
## Test de método evaluar
assert p1.evaluar(1) == 0
assert p2.evaluar(1) == 3
## Test de método toString
assert p1.toString() == "1x^4 + -3x^2 + 1x^1 + 1"
assert p2.toString() == "1x^3 + -1x^2 + 5x^1 + -2"
## Test de método sumar
p3=p1.sumar(p2)
assert p3.coeficientes == [-1,6,-4,1,1]
```