Introducción a la Programación

Prof. Agustín Gravano

Primer semestre de 2022

Clase teórica 21: Clases y objetos (parte II)

Frutería Online

Queremos programar una frutería.

De cada fruta nos interesa guardar:

- Nombre (un string)
- Precio por kilo (un float)
- Estaciones en las cuales está disponible (un conjunto de strings)

Ejemplo:

- Banana de Ecuador
- $ 150.0 por kilo
- disponible en primavera, otoño e invierno

Frutería Online

```
1  class Fruta:     ## Definición de la clase Fruta.
2      def __init__(self, n:str, p:float, es:Set[str]):
3          ''' Inicializa una fruta con nombre n, precio p, estaciones es. '''
4          self.nombre:str = n
5          self.precio:float = p
6          self.estaciones:Set[str] = es
7
8      def disponible_en(self, estacion:str) -> bool:
9          '''
10         Requiere: estacion es 'primavera', 'verano', 'otoño' o 'invierno'.
11         Devuelve: True estacion está en las estaciones de la fruta;
12                   False en caso contrario
13         '''
14         return estacion in self.estaciones
15
16     def __repr__(self) -> str:
17         ''' Devuelve una representación string de la fruta. '''
18         return self.nombre + ' ($' + str(self.precio) + '/kg)'
19
20     def __lt__(self, other) -> bool:
21         ''' Devuelve True si self < other; False en caso contrario. '''
22         return self.precio < other.precio
```

Testing de unidad de la clase **Fruta**

```
 1  import unittest
 2  from fruta import Fruta
 3
 4  PRI:str = 'primavera'
 5  VER:str = 'verano'
 6  OTO:str = 'otoño'
 7  INV:str = 'invierno'
 8
 9  class TestFruta(unittest.TestCase):
10      def test_disponible_en(self):
11          f1:Fruta = Fruta('Banana de Ecuador', 150.0, {PRI, OTO, INV})
12          self.assertTrue(f1.disponible_en(PRI))
13          self.assertFalse(f1.disponible_en(VER))
14          self.assertTrue(f1.disponible_en(OTO))
15          self.assertTrue(f1.disponible_en(INV))
16
17          f2:Fruta = Fruta('Pera Williams', 70.0, {VER})
18          self.assertFalse(f2.disponible_en(PRI))
19          self.assertTrue(f2.disponible_en(VER))
20          self.assertFalse(f2.disponible_en(OTO))
21          self.assertFalse(f2.disponible_en(INV))
22
23      def test_menor(self):
24          f1:Fruta = Fruta('Banana de Ecuador', 150.0, {PRI, OTO, INV})
25          f2:Fruta = Fruta('Pera Williams', 70.0, {VER})
26          self.assertTrue(f2 < f1)
27          self.assertFalse(f1 < f2)
28          self.assertFalse(f1 < f1)
29          self.assertFalse(f2 < f2)
30
31      def test_repr(self):
32          f1:Fruta = Fruta('Banana de Ecuador', 150.0, {PRI, OTO, INV})
33          f2:Fruta = Fruta('Pera Williams', 70.0, {VER})
34          self.assertEqual(str(f1), 'Banana de Ecuador ($150.0/kg)')
35          self.assertEqual(str(f2), 'Pera Williams ($70.0/kg)')
36
37  unittest.main()
```

Frutería Online

Para nuestra frutería, queremos modelar:

- Fruta ✓
- Catálogo de frutas, a cargar de un archivo CSV
- Carrito de compras, con operaciones para agregar/sacar kg de fruta del carrito y para consultar el importe total

Además, nos darán ya implementada una interfaz de usuario que usará nuestras clases `Fruta`, `Catalogo` y `Carrito`.

Lectura de un archivo CSV

Archivo ''frutas.csv'':

```
nombre,precio_por_kg,estaciones
Banana de Ecuador,150,"primavera,otoño,invierno"
Pera Williams,70,verano
Uva verde,60,verano
Frutillas,197.5,"primavera,verano"
Manzana,99.99,"verano,primavera,otoño,invierno"
...
```

```
1  import csv
2  from typing import TextIO
3
4  filename:str = 'frutas.csv'    # nombre del archivo con los datos
5  f:TextIO = open(filename)
6
7  frutas:List[Fruta] = []
8  for linea in csv.DictReader(f):
9      # linea es un Dict[str,str]
10     nombre:str   = linea['nombre']
11     precio:float = float(linea['precio_por_kg'])
12     est_str:str  = linea['estaciones']
13     estaciones:Set[str] = set(est_str.split(','))
14     fru:Fruta = Fruta(nombre, precio, estaciones)
15     frutas.append(fru)
```

En la línea 5 abrimos un archivo en modo lectura.

Con el `for` de la línea 8, leemos el archivo, una línea por vez.

En cada iteración, `linea` es un diccionario que tiene como claves los nombres de las columnas (`nombre`, `precio_por_kg`, `estaciones`), asociados a los valores correspondientes para cada fruta.

Ejemplo: `linea['nombre']` $\rightarrow$ `'Banana de Ecuador'`

Catálogo de frutas

```python
from fruta import Fruta
from typing import List, TextIO, Set
import csv

class Catalogo:
    def __init__(self, archivo_csv:str):
        '''
        Inicializa el catálogo de frutas, cargando las frutas contenidas
        en el archivo archivo_csv.
        Requiere: archivo_csv es el nombre de un archivo en formato
                  CSV (valores separados por comas), con tres columnas:
                  'nombre' (str), precio_por_kg (float), estaciones (lista de
                  strings separados por comas, donde los strings posibles son
                  'primavera', 'verano', 'otoño' o 'invierno').
        '''
        self.frutas:List[Fruta] = []
        f:TextIO = open(archivo_csv)
        for linea in csv.DictReader(f):
            nombre:str   = linea['nombre']
            precio:float = float(linea['precio_por_kg'])
            estaciones:Set[str] = set(linea['estaciones'].split(','))
            fru:Fruta = Fruta(nombre, precio, estaciones)
            self.frutas.append(fru)
        f.close()

    def frutas_de_estacion(self, estacion:str) -> List[Fruta]:
        '''
        Requiere: estacion es 'primavera', 'verano', 'otoño' o 'invierno'
        Devuelve: las frutas disponibles en la estación dada (en algún orden).
        '''
        vr:List[Fruta] = []
        for fru in self.frutas:
            if fru.disponible_en(estacion):
                vr.append(fru)
        return vr
```

Testing de unidad de la clase `Catalogo`

Para testear esta clase, creamos archivos CSV con datos inventados: `frutas-para-testing1.csv` y `frutas-para-testing2.csv`.

Después armamos casos de test, en los cuales creamos objetos de la clase `Catalogo` con esos archivos, y vemos que todo funcione como esperamos.

(Ver `catalogo-test.py`.)

Siguiendo con la Frutería Online...

Ahora queremos programar un carrito de compras.

En concreto, necesitamos modelar carritos de compras,
a los cuales queremos poder agregar y sacar kilogramos de alguna fruta
(en cantidades enteras) y consultar el importe total (la suma de precios).

¿Cómo podemos lograr esto?

Idea: definir una clase Carrito, que tenga un diccionario que asocia:

$$\text{fruta} \rightarrow \text{cantidad de kg (número entero)}$$

Un objeto Carrito tendría (al menos) estos métodos:

- Inicializar con un diccionario vacío.
- Agregar `p` kilos de una fruta `f` (`p` entero mayor que 0).
- Sacar `p` kilos de una fruta `f` (`p` entero mayor que 0, tal que en el carrito haya al menos `p` kilos de la fruta `f`).
- Calcular el precio del contenido actual.

```python
class Carrito:
    def __init__(self):
        ''' Inicializa un carrito vacío. Requiere: nada. '''
        self.kg_fruta:Dict[Fruta, int] = dict()

    def __repr__(self) -> str:
        ''' Devuelve un string con los datos del carrito. Requiere: nada. '''
        return str(self.kg_fruta)

    def agregar(self, f:Fruta, p:int):
        ''' Modifica: Agrega p kg de la fruta f al carrito. Requiere: p>0.'''
        if f in self.kg_fruta:
            self.kg_fruta[f] = self.kg_fruta[f] + p
        else:
            self.kg_fruta[f] = p

    def sacar(self, f:Fruta, p:int):
        ''' Modifica: Saca p kg de la fruta f del carrito.
            Requiere: p>0, y hay al menos p kilos de f. '''
        self.kg_fruta[f] = self.kg_fruta[f] - p
        if self.kg_fruta[f] == 0:
            self.kg_fruta.pop(f)

    def calcular_precio_total(self) -> float:
        ''' Devuelve: el precio total de la fruta en el carrito. Req: nada.'''
        vr:float = 0.0
        for fruta in self.kg_fruta:
            peso:int = self.kg_fruta
            vr = vr + peso * fruta.precio
        return vr
```

Ver también el testing de unidad de esta clase en el archivo carrito-test.py.

(Continuamos la próxima clase.)

Bibliografía complementaria:

- APPP2, capítulos 12, 13 y 14 (excepto 14.9).
- HTCSP3, capítulos 15 y 16.

Con lo visto, ya pueden resolver la Guía de Ejercicios 9 completa.

(Por ahora, no prestar atención a los requerimientos de complejidad temporal pedidos en algunos ejercicios.)