Pràctica 2: Gestió de Receptes de Cuina

Tecnologia de la Programació — iTIC

Aleix Llusà-Serra Sebastià Vila-Marta

25 de febrer de 2020

Índex

1 Organització

1.1 Objectius

L'objectiu d'aquesta pràctica és prendre contacte amb els programes estructurats en base a classes d'objectes. El context del projecte se situa en una fundació dedicada a les arts gastronòmiques, en el marc d'un projecte de recerca sobre la interacció entre productes i tècniques de cocció en la cuina catalana.

1.2 Condicions

- La pràctica està calibrada per a ésser treballada en equip.
- El model de desenvolupament que es demana que utilitzeu és «test driven programming». Si no en recordeu els detalls del curs passat, pregunteu!

1.3 Lliurables

- Caldrà lliurar el codi resultant del projecte i una documentació escrita amb `Sphinx` que ha d'incloure també una taula de dedicacions de cada persona del grup a les diferents tasques de la pràctica.

- La durada de la pràctica és de 2 setmanes.

2 Introducció

La Fundació Delícia es dedica a la investigació en l'àmbit de l'alta gastronomia. Unes investigadores d'aquesta fundació estan desenvolupant un projecte en el que volen estudiar quines tècniques de cocció s'apliquen a quins productes en l'àmbit de la cuina catalana. A tal efecte es dediquen a recopilar receptaris de cuina catalana que abasten des dels llibres de cuina medieval com el *Llibre de Sent Soví*, fins als llibres actuals sobre cuina tecnoemocional d'Adrià.

Les investigadores de Delícia necessiten un aplicatiu informàtic que els permeti emmagatzemar informació sobre aquests receptaris per tal de poder-la tractar convenientment més endavant. Aquesta pràctica ha de resoldre aquest problema.

2.1 Arquitectura

L'aplicació consistirà en un conjunt de classes i un programa principal. Les classes són:

Producte Representa un producte constitutiu d'una o més receptes. Cada instància de Producte està identificada per un nom.

Recepta Representa una recepta de cuina. Una instància de Recepta:

- Té un nom que la identifica.
- Està associada a n instàncies de Producte.
- Per a cada producte la recepta coneix el pes en grams que se'n requereix.

Receptari Un Receptari representa una col·lecció d'instàncies de Recepta i Producte interrelacionades. És una *Façana*.

Interpret Un Interpret representa un intèrpret de comandes simple. Usarem una instància d'aquesta classe com a interfície d'usuari entre el nostre programa i l'usuari.

Més a més, l'aplicació tindrà un mòdul `main.py` que encabirà el programa principal.

Tasca 1 Feu un diagrama de classes del programa.

Tasca 2 Prepareu l'entorn de treball on desenvolupareu el projecte. A tal efecte:

1. Creeu un directori anomenat `receptari` i dins d'aquest un subdirectori anomenat `src` en que hi anireu escrivint els fitxers font del projecte.

2. Dins de `receptari` i usant les comandes de `Sphinx` creeu un directori anomenat `doc` en el que escriureu la documentació usant les eines que ja coneixeu.

3. Creeu un capítol de la documentació en el que començareu a anotar el temps que dediqueu individualment a cada tasca. Feu servir una taula amb un format similar al següent:

Tasca	Persona1 (h)	Persona 2 (h)	Total (h)
T1	1.20	2.10	3.30
T2	—	0.80	0.80
T3	4.13	2.00	6.13
⋮	⋮	⋮	⋮
TOTAL	5.33	4.90	10.23

3 Implementació i test

3.1 Classe Producte

El mòdul `producte.py` conté la classe Producte. La classe ha de tenir els següents atributs:

nom [**str**, Públic] És el nom del producte.

La classe disposarà d'un constructor amb el nom com a paràmetre.

TASCA 3 Implementeu el mòdul `producte.py`: primer definiu els doctests corresponents, assegureu-vos que són complets. Després implementeu els mètodes i comproveu que funcionen correctament usant els doctests. Finalment documenteu el codi correctament i annexeu aquesta documentació a la documentació del projecte usant `Sphinx`.

3.2 Classe Recepta

El mòdul `recepta.py` conté la classe Recepta. La classe ha de tenir els següents atributs:

nom [**str**, Públic] És el nom de la recepta.

ingredients [**list**, Privat] És una llista de tuples que enregistra els ingredients de cada recepta. Cada tupla (p,q) codifica un producte p i la corresponent quantitat en grams q.

La classe disposarà dels següents mètodes:

1. __init__(self, nom)

   Constructor. La llista d'ingredients d'una Recepta acabada de crear és la llista buida.

2. afegeix_ingredient(self,p,q)

   Modificador. Afegeix q grams del producte p a la recepta. Si la recepta ja contenia el Producte p, incrementa la seva quantitat en q grams.

3. conte_ingredient(self,p)

   Retorna **True** ssi la recepta conté l'ingredient p.

4. quantitat_ingredient(self,p)

   Retorna la quantitat de producte p en grams o 0 si no en conté.

5. pes_total(self)

   Retorna el pes base total de la recepta en grams.

6. ingredients(self)

   Retorna la llista d'ingredients de la recepta.

TASCA 4 Definiu l'esquelet de la classe i afegiu els doctests corresponents. Assegureu-vos que són complets. Aneu implementant els mètodes i comproveu que passen els doctests correctament. Finalment documenteu el codi correctament i annexeu aquesta documentació a la documentació del projecte usant `Sphinx`.

3.3 Classe Receptari

El mòdul `receptari.py` conté la classe Receptari. La classe ha de tenir els següents atributs:

receptes [**dict**, Privat] És un diccionari en el que la clau correspon al nom de totes les receptes del receptari i el valor correspon a l'objecte Recepta corresponent.

productes [**dict**, Privat] És un diccionari en el que la clau correspon al nom de tots els productes que intervenen en el receptari i el valor correspon a l'objecte Producte corresponent.

La classe disposarà dels següents mètodes:

1. __init__(self)

   Constructor. Crea un receptari buit, sense productes ni receptes.

2. afegeix_recepta(self,n)

   Modificador. Afegeix una recepta de nom n al receptari. Si existia una recepta del mateix nom es queixa. La recepta, inicialment no té ingredients.

3. afegeix_producte(self, nomp)

   Afegeix un nou Producte de nom nomp. Si el producte ja existeix es queixa.

4. afegeix_ingredient_recepta(self,nomr, nomp, q)

   Afegeix q grams de l'ingredient de nom nomp a la recepta de nom nomr. Si la recepta o l'ingredient no existeix es queixa.

5. receptes_ingredient(self,nomp)

   Retorna la llista de noms de receptes en que participa l'ingredient de nom nomp.

6. receptes(self)

   Retorna la llista de noms de receptes del receptari.

7. ingredients(self)

   Retorna la llista de noms de productes del receptari.

8. recepta(self,nomr)

   Retorna una llista de parells (nomp, q) cadascun dels quals representa un ingredient de la recepta nomr. Cada parell agrega el nom del producte nomr i el pes necessari en grams q.

TASCA 5 Implementeu el mòdul `receptari.py` seguint la mateixa pauta que en la tasca anterior. Documenteu-lo usant `Sphinx`.

3.4 Classe Intèrpret

Aquesta classe d'objectes abstreu un intèrpret d'ordres senzill. Recordeu que un intèrpret d'ordres és un programa que, de forma interactiva, llegeix ordres de l'usuari i les va executant una a una. Ja coneixeu alguns intèrprets d'ordres, en particular:

1. La shell de UNIX, que us aten en la terminal d'ordres del sistema operatiu.

2. L'intèrpret interactiu de Python, que us permet executar ordres Python interactivament i usar-lo com una calculadora.

Un objecte de la classe Interpret és un interpret d'ordres configurable. Això significa que per usar-lo primer cal configurar-lo. Configurar-lo consisteix a dir-li quines ordres ha de conèixer i què han de fer.

Una sessió de treball típica amb aquesta classe podria ser:

```
>>> def c1(l): print "executo l'ordre 1: {0}".format(l[0])
>>>
>>> def c2(l): print "executo l'ordre 2: {0}".format(l[0])
>>>
>>> i = Interpret()
>>> i.set_prompt("**")
>>> i.afegeix_ordre("menja", c1)
>>> i.afegeix_ordre("beu", c2)
>>>
>>> i.run()
** menja caramel
executo l'ordre 1: caramel
** beu xocolata
executo l'ordre 2: xocolata
** surt
>>>
```

Fixeu-vos que una instància de l'intèrpret es configura amb el mètode afegeix_ordre. Afegir una ordre implica indicar el nom de la ordre i també la funció que implementa aquesta ordre. També haureu observat que l'intèrpret s'engega amb el mètode run i acaba quan l'usuari usa l'ordre especial `surt`.

En el mòdul `interpret.py`, la classe Interpret ha de tenir els següents atributs:

prompt [**str**, Privat] Emmagatzema el prompt que usarà l'intèrpret.

dcom [**dict**, Privat] És el diccionari que emmagatzema les ordres conegudes per l'intèrpret. El diccionari emmagatzema una entrada per a cada ordre. Per a una ordre específica, la clau correspon amb el nom de l'ordre i el valor és la funció que implementa l'ordre (vegeu el mètode afegeix_ordre).

La classe disposarà dels següents mètodes:

1. __init__(self)

   Constructor. Crea un interpret buit, sense ordres.

2. set_prompt(self,p)

   Modificador. Fixa l'string p com el prompt de l'intèrpret.

3. afegeix_ordre(self,nomc,ordre)

   Modificador. Afegeix a l'intèrpret una ordre de nom nomc associada a la funció ordre. Si ja existia una ordre amb aquest nom, es queixa. Noteu que el tercer paràmetre del mètode és una funció!

   La funció de nom ordre és una funció que té com a únic paràmetre una llista de strings.

4. run(self)

   Arrenca l'execució d'aquest intèrpret d'ordres. L'intèrpret s'executa indefinidament fins que l'usuari escriu l'ordre `surt`.

   A cada cicle d'interpretació, l'intèrpret escriu el prompt, llegeix un string del teclat, l'analitza separant els mots que el formen. El primer mot considera que és un nom d'ordre i la resta de mots els paràmetres d'aquesta ordre. Finalment executa la funció corresponent a l'ordre i li passa com a paràmetre la resta de mots en una llista.

TASCA 6 Implementeu el mòdul tot i definint els doctests corresponents, assegureu-vos que són complets i funcionen correctament. Finalment documenteu el codi correctament i annexeu aquesta documentació a la documentació del projecte usant `Sphinx`.

3.5 Mòdul main

El mòdul `main.py` conté el programa principal. Aquest programa crea una instància de Receptari i una instància d'Interpret juntament amb les funcions que implementen les ordres necessàries. A continuació, engega l'intèrpret. Aquest es va comunicant interactivament amb l'usuari fins acabar la sessió.

L'intèrpret ha de tenir les següents ordres:

`producte <nom>`

Afegeix un producte al receptari que té nom `<nom>`.

`recepta <nom>`

Afegeix una recepta al receptari que té nom `<nom>`.

`ingredient <nomp> <nomr> <qua>`

Afegeix `<qua>` grams de l'ingredient de nom `<nomp>` a la recepta de nom `<nomr>`.

`print <ent> [<nom>]`

Escriu per pantalla segons el valor de `<ent>`. Si `<ent>` és:

`receptes`

Escriu la llista de noms de receptes del receptari.

`productes`

Escriu la llista de noms de producte del receptari.

`recepta`

Escriu els ingredients i la quantitat que intervenen en la recepta de nom `<nom>`.

`receptes-ing`

Escriu la llista de noms de recepta en les que participa l'ingredient anomenat `<nom>`.

`surt`

Acaba l'execució del programa.

TASCA 7 Implementeu el mòdul principal i comproveu que funciona correctament.