HALMSTAD UNIVERSITY

Programming

Lab Sessions

HALMSTAD UNIVERSITY

Assignment V - Presentations

- Zoom room for presentations:

  Meeting ID: 641 7002 9853

  Passcode: 034035

- **SECOND LECTURE OF THE WEEK. Be on time**
- Martin will be available to solve doubts at the usual meeting room with an exercise list.

HALMSTAD UNIVERSITY

Resources

- Slides for this lecture and the solutions for the third exercise list are available at marcomarinho.com.
- Videos for Martins solutions of the exercises are up.

HALMSTAD UNIVERSITY

DET ÄR MATEMATIKFRI FREDAG MINA BEKANTA

Classes

- Abstractions that allow us to deal better with the problem domain at hand.
- Easier to rationalize as humans, classes are things that have meaning to us.
- Classes can have attributes (information, something it is), and methods (action, something it does)
- Are the basis of modern programming languages.

Bakery

The problem

- Create a bakery program that can store recipes for different types of cakes and breads. The program should store and return the name of the cake, and the ingredients, the calories, and the price based on ingredients.
- The user can choose a profit margin to set the price of the cakes and breads.
- All information will be taken from a file.

Example

- The most basic component of the problem at hand is an ingredient. An ingredient has a name, a price, and a caloric content. Let's create a class for the ingredient.

Example

```
class ingredient:

    #Method to initalize the ingredient with name, price, and calories
    def __init__ (self, name, price, calories):
        self.name = name
        self.price = price
        self.calories = calories

    #String representation of our class
    def __str__ (self):
        return 'Name: '+self.name+ ', Price: '+str(self.price)+ ', Calories: ' +str(self.calories)

#Create an ingredient
flour = ingredient('Flour', 100, 100)

#Print it
print(flour)
```

Example

- Now we can build a class for a cake, a cake has a name a list of ingredients and a profit margin.

Example

```
class cake:

    #Initialize cake object
    def __init__ (self, name):
        self.name = name
        self.ingredients = []
        self.profit_margin = 0

    #Add an ingredient to the cake recipe
    def addIngredient(self, ingredient):
        self.ingredients.append(ingredient)

    #Set a profit margin for the cake
    def setProfitMargin(self, profit_margin):
        self.profit_margin = profit_margin

    #Calculates and return the price of the cake
    def getPrice(self):
        total = 0
        for ingredient in self.ingredients:
            total = total + ingredient.price
        return total * (1 + self.profit_margin/100)

    #Calculates and returns the total calories of the cake
    def getCalories(self):
        total = 0
        for ingredient in self.ingredients:
            total = total + ingredient.calories
        return total

    #Define a pretty representation of the cake
    def __repr__(self):
        string = self.name + '\nIngredients: '
        for ingredient in self.ingredients:
            string = string + ingredient.name+', '
        string = string[:-2] + '\nPrice: '+ str(self.getPrice())
        string = string + '\nCalories: '+ str(self.getCalories())
        return string
```

Example

- Now, lets read a list of ingredients from a file, and create a list of ingredient objects in our program.

Example

```
#Import our classes from a different module
import bakery

#Open and clean file
source = open('ingredients.txt')
raw_data = source.readlines()
data = []
for element in raw_data:
    data.append(element.strip())

#Split the lines of data at the common character and create the respective ingredient objects
ingredients = []
for element in data:
    split_element = element.split(' - ')
    ingredient = bakery.ingredient(split_element[0], int(split_element[1]), int(split_element[2]))
    ingredients.append(ingredient)

#Print the ingredients
print(ingredients)
```

Example

- Now, lets adapt our program so that we can create the Gigacake, a cake that's is made from cakes

Example

```
class ingredient:

    #Method to initalize the ingredient with name, price, and calories
    def __init__ (self, name, price, calories):
        self.name = name
        self.price = price
        self.calories = calories

    def getCalories(self):
        return self.calories

    def getPrice(self):
        return self.price

    #String representation of our class
    def __repr__ (self):
        return 'Name: '+self.name+ ', Price: '+str(self.price)+ ', Calories: '
+str(self.calories)
```

Example

```
class cake:

    #Initialize cake object
    def __init__ (self, name):
        self.name = name
        self.ingredients = []
        self.profit_margin = 0

    #Add an ingredient to the cake recipe
    def addIngredient(self, ingredient):
        self.ingredients.append(ingredient)

    #Set a profit margin for the cake
    def setProfitMargin(self, profit_margin):
        self.profit_margin = profit_margin

    #Calculates and return the price of the cake
    def getPrice(self):
        total = 0
        for ingredient in self.ingredients:
            total = total + ingredient.getPrice()
        return total * (1 + self.profit_margin/100)

    #Calculates and returns the total calories of the cake
    def getCalories(self):
        total = 0
        for ingredient in self.ingredients:
            total = total + ingredient.getCalories()
        return total

    #Define a pretty representation of the cake
    def __repr__(self):
        string = self.name + '\nIngredients: '
        for ingredient in self.ingredients:
            string = string + ingredient.name+', '
        string = string[:-2] + '\nPrice: '+ str(self.getPrice())
        string = string + '\nCalories: '+ str(self.getCalories())
        return string
```

Example

```
#Import our classes from a different module
import bakery

#Open and clean file
source = open('ingredients.txt')
raw_data = source.readlines()
data = []
for element in raw_data:
    data.append(element.strip())

#Split the lines of data at the common character and create the respective ingredient objects
ingredients = []
for element in data:
    split_element = element.split(' - ')
    ingredient = bakery.ingredient(split_element[0], int(split_element[1]), int(split_element[2]))
    ingredients.append(ingredient)

cake1 = bakery.cake('Chocolate')
cake1.addIngredient(ingredients[0])
cake1.addIngredient(ingredients[1])

cake2 = bakery.cake('Coconute')
cake2.addIngredient(ingredients[1])
cake2.addIngredient(ingredients[2])

cake3 = bakery.cake('Gigacake')
cake3.addIngredient(cake1)
cake3.addIngredient(cake2)

cake3.getCalories()

print(cake1.getCalories())
print(cake2.getCalories())
print(cake3.getCalories())

print(cake1.getPrice())
print(cake2.getPrice())
print(cake3.getPrice())
```

Example

- Now, lets update make it so that a user can create a cake from the list of ingredients present on the file:

Example

```
#Import our classes from a different module
import bakery

#Open and clean file
source = open('ingredients.txt')
raw_data = source.readlines()
data = []
for element in raw_data:
    data.append(element.strip())

#Split the lines of data at the common character and create the respective ingredient objects
ingredients = []
for element in data:
    split_element = element.split(' - ')
    ingredient = bakery.ingredient(split_element[0], int(split_element[1]), int(split_element[2]))
    ingredients.append(ingredient)

#Ask the user for the name of the cake
name = input('Please choose a name for the cake: ')
user_cake = bakery.cake(name)

#Print list of ingredients and add the user choice to the cake
while True:
    print('Please choose an ingredient from the list, type 0 to finish the cake: ')
    for i in range(len(ingredients)):
        print( i+1, ' - ', ingredients[i].name)
    choice = int(input('==>'))
    if choice == 0:
        break
    else:
        user_cake.addIngredient(ingredients[choice-1])

#Print final cake
print(user_cake)
```

Exercise

- **Person Class:** Create a class to represent a person:
  - Atributtes: name, age, weight,
  - Methods: getOlder, gainWeight, loseWeight, growHeight.
  - Each time the getOlder method is called, the person gains one year and gains 0.5cm of height until the are older than 21.

Exercise

```
class Person:

    def __init__(self, name, age, weight, height):
        self.name = name
        self.age = age
        self.weight = weight
        self.height = height

    def gainWeight(self, weight):
        self.weight += weight

    def loseWeight(self, weight):
        if (weight > self.weight):
            self.weight = 0
        else:
            self.weight -= weight

    def growHeight(self, height):
        self.height += height

    def getOlder(self, years):
        ageBefore = self.age
        self.age += years
        if (ageBefore < 25):
            if (self.age < 25):
                self.growHeight(years * 0.5)
            else:
                self.growHeight((25 - ageBefore) * 0.5)
```

End