Soylent Documentation

Release 0.0

Adrien Beaucreux

May 01, 2013

CONTENTS

Quite a lot of people who have heard about soylent have toyed with the idea of mixing their own.

In my case, looking for the optimal mix of components, with so many components to choose from, became a ground for fiddling around.

What if I could just put a list of ingredients in a database, together with their nutritional values, and some kind of software could create a mix that has the right amount of every nutrient?

This is my attempt at creating such a software.

Contents:

CHAPTER

ONE

DEVELOPPER'S GUIDE

This section of the documentation is an in depth explanation of the structure of the code and the choices that led to it.

1.1 Data Structure

The whole sense of soylent is to provide all the nutrients needed by the body, in an easy to mix way.

This module contains the different classes we need to represent an alimentation

In the most basic form, we have a recipe, containing quantities of ingredients. An ingredient in turn contains certain quantities of nutrients.

In order to avoid creating a new file format, and to get all the goodies that comes with it, for example migration scripts, we use *SQLAlchemy* to manage the storage.

In order to simplify the management and calculations for the physical quantities, we use magnitude

1.1.1 Nutrients types

class `models.`**`Nutrient`**(*name*)

A Nutriment is pretty much anything that the body needs to be fed in order to work properly.

As for the relevant parts, all nutrients have a name, but some have more than one (think translation). Some have a minimal amount, other are optional. Some have a maximal amount, other can be consumed without problems. Then there are multiple recommended values, according to different guidelines, like the US recommended Dietary Reference Intake, or the EU-Directive 2008-100-EC.

given that most of these are optional, there is no reason to put them in the class itself.

They will be added in due time as needed.

A nitrient by itself doesn't do a lot, so we kept the methods down to a minimum.

class `models.`**`Macronutrient`**(*name*, *conversion_factor*)

A Macronutrient is the kind of nutrient that can be converted to energy by the body, as described in the Food energy article on wikipedia.

This class derives from `Nutrient`, using *concrete inheritance mapping* to build the relationship.

`conversion_factor`

`energy_per_weight`

per default, the energy per weight is a magnitude expressed in Joules per gram.

There are different sources of macro nutrients, represented as subclasses of `Macronutrient`. For the moment, only the `Carbohydrate`, `Fat` and `Protein` are represented.

class `models.`**`Carbohydrate`**(*name*)
This class is used to represent a type of Carbohydrate.

It has no specific attribute or method.

class `models.`**`Fat`**(*name*, *saturated*, *monounsaturated*, *polyunsaturated*)
This class is used to represent different types of Fat.

It has three attributes, all Booleans, to indicate kind of saturation of the fat:

`saturated`

`monounsaturated`

`polyunsaturated`

class `models.`**`Protein`**(*name*, *essential*)
This class is the result of a misunderstanding on my part. Instead of being used to represent Protein, it is used to represent the different Amino acids.

`essential`
`an amino acid is essential if it cannot be synthesized by the body.`

1.1.2 Ingredient

Now that we have the nutrients, we can pack them in ingredients.

This is done using two classes:

class `models.`**`Ingredient`**(*name*, *serving_size*, *serving_unit*)
An ingredient has a name, logically, it is also measured in a specific unit and contains some nutrients per servings.

`serving`
read/write property for the serving as a physical quantity.

The instances also contains different read only properties representing the different kind of macronutrients contained in the Ingredient as a list of `IngredientNutrient`.

`macronutrients`

`carbohydrates`

`fats`

`proteins`

`energy_per_serving`(*self*, *serving*)
returns the energy contained in a serving in Joules. This is calculated based on the macronutrients contained in the ingredient.

Interrestingly, it sometimes is lower than the energy per serving indicated on the package.

In the same way that there are four properties for the macronutrients, there are three different methods to return the energy provided by the different kinds of macronutrients for a given serving: .. method:: carbohydrates_per_serving(self, serving) .. method:: fats_per_serving(self, serving) .. method:: proteins_per_serving(self, serving)

class `models.`**`IngredientNutrient`**(*ingredient*, *nutrient*, *serving_amount*, *serving_size*)
This class is an *association object* to join the ingredients with their nutrients.

Both ingredient and nutrient can be references using the corresponding attributes `ingredient` and `nutrient`.

concentration
A property representing the concentration of the nutrient for a serving, e.g. '5 g/ml'.

weight_per_serving(*serving*)
The amount of nutrient of a given serving of the ingredient.

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

m