Baking Cookies!

Group Names and Roles

- Jiaxin Luo (Driver)
- Jenny Lee (Proposer)
- Partner 3 (Role)

Introduction

In this worksheet, you'll implement a `recipe` class that stores data on how to create a tasty dish--like cookies!--in a structured way. We'll build on this `recipe` class in a future Discussion activity.

Schematically, a recipe has three primary pieces of data:

1. A title (e.g. "Cookies")
2. A list of ingredients, with quantities. Example:
```
Flour (grams)  : 400
Butter (grams) : 200
Salt (grams)   : 10
Sugar (grams)  : 100
```
3. A list of directions. Example:

> 1. In a large mixing bowl, cut the chilled butter into the flour and sugar.
> 2. Add the salt and sugar, and combine.
> 3. Roll into a log, and freeze.
> 4. Preheat the oven to 200 C°.
> 5. Cut the log of dough into thin disks and place on baking sheet.
> 6. Bake for 12 minutes, flipping after 7 minutes.

Our `recipe` class will store such data.

In this activity, it is not necessary to copy/paste the code that corresponds to your class. Rather, you can keep all the code for your class in Part A, and then just run the tests in the subsequent parts to verify that your code is working.

Part A

Start by creating a class called `Recipe`. Give this class an `__init__()` method that allows the user to set the `title`, `ingredients`, and `directions` as instance variables. That is, after having defined your class, you should be able to run the following code and receive the printed result.

```
cookies = Recipe("cookies", {"cookie jar" : 1}, ["take a cookie out of the
jar"])
print(cookies.title)
print(cookies.ingredients)
print(cookies.directions)
```

```
cookies
{'cookie jar': 1}
['take a cookie out of the jar']
```

In [1]:

```
# write class here
class Recipe:
    def __init__(self, title, ingredients, directions):
        self.title = title
        self.ingredients = ingredients
        self.directions = directions
```

In [2]:

```
# test code here
cookies = Recipe("cookies", {"cookie jar" : 1}, ["take a cookie out of the jar"])
print(cookies.title)
print(cookies.ingredients)
print(cookies.directions)
```

```
cookies
{'cookie jar': 1}
['take a cookie out of the jar']
```

Part B

Now add **input checking.** Modify the `__init__()` method to enforce the following conditions:

1. `title` must be a string. If not, raise an informative `TypeError`.
2. `ingredients` must be a `dict`. If not, raise an informative `TypeError`.
3. The keys of `ingredients` must all be strings. If not, raise an informative `TypeError`.
    - ***Hint***: `all([x == "cookies" for x in container])` will check whether `x` has value "cookies" for all `x` in `container`. You can modify this idea to perform this check without writing a `for`-`loop`, although such a loop is also a fine approach.
4. The `directions` must be a `list`. If not, raise an informative `TypeError`.

In this and future parts, you can modify your code in Part A -- no need to copy/paste your class.

Write a simple test case for each of these four conditions to show that the corresponding error is raised. The first one is written for you. Each of these test cases can be completed in a single line.

If you finish early, you can come back and add the following additional checks to your class:

1. The entries of `directions` must be strings. If not, raise an informative `TypeError` .
2. The values of `ingredients` must all be `int` s or `float` s. If not, raise an informative `TypeError` .
3. The values of `ingredients` must all be nonnegative. If not, raise an informative `ValueError` .

In [11]:
```
# first test
class Recipe:
    def __init__(self, title, ingredients, directions):
        if type(title) == str:
            self.title = title
        else:
            print("TypeError: title must be a string!")
        self.ingredients = ingredients
        self.directions = directions
```

In [19]:
```
# second test
class Recipe:
    def __init__(self, title, ingredients, directions):
        self.title = title
        if type(ingredients) == dict:
            self.ingredients = ingredients
        else:
            print("TypeError: ingredients must be a dictionary!")
        self.directions = directions
```

In [33]:
```
# third test
class Recipe:
    def __init__(self, title, ingredients, directions):
        self.title = title
        if all([type(x)==str for x in ingredients.keys()]) :
            self.ingredients = ingredients
        else:
            print("TypeError: keys of ingredients must be all strings!")
        self.directions = directions
```

In [38]:
```
# fourth test
class Recipe:
    def __init__(self, title, ingredients, directions):
        self.title = title
        self.ingredients = ingredients
        if type(directions) == list:
            self.directions = directions
        else:
            print("TypeError: directions must be a list!")
```

Part C

Implement *scalar multiplication*. If `cookies` is a `recipe` , then `2*cookies` is a new `recipe` in which all the values of the `ingredients` have been doubled. For example:

```
title = "cookies"

ingredients = {
    "Flour (grams)"  : 400,
    "Butter (grams)" : 200,
    "Salt (grams)"   : 10,
    "Sugar (grams)"  : 100
}

directions = [
    "In a large mixing bowl, cut the chilled butter into the flour and
sugar." ,
    "Add the salt and sugar, and combine." ,
    "Roll into a log, and freeze." ,
    "Preheat the oven to 200 C°." ,
    "Cut the log of dough into thin disks and place on baking sheet." ,
    "Bake for 12 minutes, flipping after 7 minutes."
]

cookies = Recipe(title, ingredients, directions)
```

Then,

```
doubled_cookies = 2*cookies
doubled_cookies.ingredients
```

```
   {'Flour (grams)': 800,
    'Butter (grams)': 400,
    'Salt (grams)': 20,
    'Sugar (grams)': 200}
```

Hints:

- The required magic method is called `__rmul__(self, multiplier)` . Please make sure that the return value of this magic method is a new instance of class `Recipe` .
- *Dictionary comprehensions* provide a convenient way to make new dictionaries from old ones. Their syntax is related to list comprehensions. For example:

  ```
  d = {"shortbread cookie" : 2, "chocolate chip cookie" : 1}
  {"tasty " + key : val for key, val in d.items()}
  ```

In [46]:

```
class Recipe:
    def __init__(self, title, ingredients, directions):
        self.title = title
        self.ingredients = ingredients
        self.directions = directions

    def __rmul__(self, multiplier):
        return Recipe(self.ingredients*multiplier.ingredients)
```

In [49]:

```
r= Recipe("ds",1,[])
e = Recipe("ds",5,[])
t=r*e
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12732/3779814067.py in <module>
      1 r= Recipe("ds",1,[])
      2 e = Recipe("ds",5,[])
```

```
----> 3 t=r*e

TypeError: unsupported operand type(s) for *: 'Recipe' and 'Recipe'
```

In [41]:

```
# test case -- run this code, no need to modify.
title = "cookies"

ingredients = {
    "Flour (grams)"  : 400,
    "Butter (grams)" : 200,
    "Salt (grams)"   : 10,
    "Sugar (grams)"  : 100
}

directions = [
    "In a large mixing bowl, cut the chilled butter into the flour and sugar." ,
    "Add the salt and combine." ,
    "Roll into a log, and freeze." ,
    "Preheat the oven to 200 C° ." ,
    "Cut the log of dough into thin disks and place on baking sheet." ,
    "Bake for 12 minutes, flipping after 7 minutes."
]
cookies = Recipe(title, ingredients, directions)

doubled_cookies = 2*cookies
doubled_cookies.ingredients
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12732/2660864.py in <module>
     19 cookies = Recipe(title, ingredients, directions)
     20
---> 21 doubled_cookies = 2*cookies
     22 doubled_cookies.ingredients

TypeError: unsupported operand type(s) for *: 'int' and 'Recipe'
```

In [ ]:

Part D

Implement attractive printing, such that, if `cookies` is a `Recipe` , then calling

```
print(cookies)
```

will print out the title, ingredients, and directions in an attractive and readable format. Feel free to be creative! Here's one illustration. Using the same recipe for `cookies` from Part C,

```
print(cookies)
```

```
# printed output

How To Make Cookies
-------------------------------

Ingredients

   Flour (grams) : 400
   Butter (grams) : 200
```

```
    Salt (grams) : 10
    Sugar (grams) : 100

Directions

    1. In a large mixing bowl, cut the chilled butter into the flour
and sugar.
    2. Add the salt and sugar, and combine.
    3. Roll into a log, and freeze.
    4. Preheat the oven to 200 C°.
    5. Cut the log of dough into thin disks and place on baking sheet.
    6. Bake for 12 minutes, flipping after 7 minutes.
```

Hint: printing is controlled by the `__str__()` magic method. The `__str__()` method should **return** the string you desire to print. Actual printing should happen separately.

Note: feel free to use any tools that you can think of to solve this problem.

In [ ]:

```python
# demonstration of printing
print(cookies)
```

Part E (Optional)

Create an object of class `Recipe` called `our_recipe` and instantiate it with the Reviewer's favorite recipe. Paste the output of `print(our_recipe)` in a post on Campuswire!