**Money and Calories Calculator**

You've reached the final project of the sprint. When you complete this task, you'll receive two great tools for controlling your expenses and income.

Let's turn to the technical details before proceeding with the task.

**Pip package manager**

Pip package manager was installed on your computer along with Python, and it will help you manage modules, libraries, and frameworks in the Python environment to install, delete, and update them. M*o*st of the Python packages are publicly available on this site:<https://pypi.org/> (*Python Package Index*) - by default, that's where pip will address by command

|  |
| --- |
| $ pip install any-package |

If you need to install a strictly defined version of the package, just specify it:

|  |
| --- |
| $ pip install django==2.02 |

If for some reason the commands for pip do not work, try contacting pip3 instead of pip:

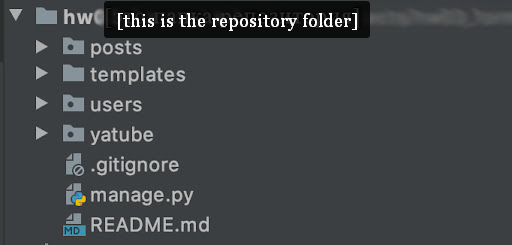
|  |
| --- |
| $ pip3 install any-package |

Other package managers have been also written for Python, but so far none has become more popular than pip.

**Working on the task**

1. Now you need to link your GitHub account to the Practicum account.
2. After linking your accounts, you will notice a new hw\_python\_oop private repository automatically appear on your GitHub.
3. Copy this repository to your computer. Currently, the hw\_python\_oop repository contains an empty code blank and a set of automatic tests.
4. Install the automatic code testing program - pytest. Before submitting the solution for review, make sure that the automated tests are successful.
5. When all the tests are passed, push your code (run git push) to your hw\_python\_oop repository. After this, click the "Submit for verification" button.

Be careful: the files and folders of your project should be placed in the main folder of the repository that you downloaded.



If you can't send the project in the usual way, and the platform offers to upload the project as an archive, follow two rules:

https://pictures.s3.yandex.net:443/resources/Untitled\_1626686630.png

* Your archive with the project should be in ZIP format. The archive format is fundamentally important.
* The archive size should not exceed 4 MB

1. Автоматические тесты будут ещё раз запущены на платформе, и если ваше решение их проходит, оно отправится на *code review*, его проверят специалисты.

In case of difficulties, go back to the lesson "Getting to know Git."

**Task requirements**

Create two calculators: for counting money and calories.

You don't need to write the user interface part of the calculators, write only the logic — a separate class for each of the calculators.

The money calculator should:

1. Save a new expense record using the add\_record()method
2. Count how much money was spent today using the get\_today\_stats() method
3. Determine how much more money can be spent today in rubles, dollars, or euros with the  get\_today\_cash\_remained(currency) method
4. Calculate how much money has been spent in the last 7 days with the get\_week\_stats() method

The calorie calculator should:

1. Save a new meal record with the add\_record() method
2. Count how many calories have already been eaten today with the get\_today\_stats() method
3. Determine how many more calories you can/should get today with the get\_calories\_remained() method
4. Count the calorie intake in the last 7 days with the get\_week\_stats() method

The calculators have many overlapping functions: they must be able to store records (about food or money: in fact, all numbers and dates), know the daily limit (how much money can be spent per day or how many calories can be consumed), and summarize records for specific dates. Insert all this general functionality into the parent **Calculator** class, and inherit the **CaloriesCalculator** and **CashCalculator** classes from it.

The constructor of the **Calculator** class must accept one argument — the limit number (the daily spending/calories limit set by the user). In the constructor, create an empty list where records will be stored later (call it records).

To make creating records easier, create a separate **Record** class for them. Then, save the following variables to it:

* amount number (monetary amount or number of calories),
* date of record creation date (it is passed explicitly to the constructor, or a default value - the current date - assigned to it),
* a comment explaining what the money was spent on or where the calories came from.

Examples of these records:

|  |
| --- |
| *# for CashCalculator*  r1 = Record(amount=145, comment="Unrestrained shopping", date="08.03.2019") r2 = Record(amount=1568, comment="Filling the consumer basket", date="09.03.2019") r3 = Record(amount=691, comment="Taxi ride", date="08.03.2019")  *# for CaloriesCalculator* r4 = Record(amount=1186, comment="A piece of cake. And another one.", date="24.02.2019") r5 = Record(amount=84, comment="Yogurt.", date="23.02.2019") r6 = Record(amount=1140, comment="A bag of chips.", date="24.02.2019") |

**Learn more about the output format**

1. The get\_calories\_remained() method of the calorie calculator should return

* *"Today you can eat some more, but with a total calorie content of no more than N kcal"*, if the limitlimitis not reached,
* or *"Stop eating!"* if the limit has been reached or exceeded.

2. The get\_today\_cash\_remained(currency) method of the money calculator should accept the currency code as input: one of the lines "rub", "usd" or "eur".

It returns a message about the status of the daily balance in the chosen currency, rounding the amount to two decimal places (to hundredths):

* *"There are N Rubles/USD/Euro left for today"* — in case the limit limit is not reached,
* or *"There is no money, stay strong"* if the limit is reached,
* or *"There is no money, stay strong: your debt is N Rubles/USD/Euro"*, if the limit is exceeded.

Specify the exchange rate with the constants USD\_RATE and EURO\_RATE, right in the body of the ***CashCalculator*** class. The rate you specify is not so important, so choose any that looks like correct. The exchange rate values can be viewed, for example, on this website: [https://www.xe.com/](https://yandex.ru/). We will learn how to get the current exchange rate from the market, but a little later.

**Recommendations and tips**

1. How to make a new record

The add\_record() method takes an object of the *Record* class as an argument and stores it in the records list.

1. Working with dates and time

You will need a function to convert dates from strings to the datetime.date format and the function dt.datetime.now() which returns the current date.

This is a good time to review the lesson about datetime from the free Python course :)

|  |
| --- |
| *import datetime as dt          date\_format = '%d.%m.%Y' moment = dt.datetime.strptime('16.12.2019', date\_format) print(moment) # prints something like 2019-12-16 00:00:00          day = moment.date() print(day) # prints the date: 2019-12-16          now = dt.datetime.now() print(now) # prints the current time in the format: 2019-01-31 13:33:27.506227          print(now.date())  # prints the current date: 2019-01-31* |

1. How to check that everything is working

To see for yourself that your classes are working correctly, write a scenario for their use.

Some examples:

|  |
| --- |
| *import datetime as dt          class Record:      ...          class Calculator:      ...          # let's create a money calculator with a daily limit of 1000 cash\_calculator = CashCalculator(1000)          # the date is not specified in the parameters, # so by default, today's date should be automatically added to the record cash\_calculator.add\_record(Record(amount=145, comment="coffee"))  # and the date should also be added to this record automatically cash\_calculator.add\_record(Record(amount=300, comment="To Mark for lunch")) # here, the user specified the date, and we saved it cash\_calculator.add\_record(Record(amount=3000, comment="At the bar for Tanya's birthday", date="08.11.2019"))                  print(cash\_calculator.get\_today\_cash\_remained("rub")) # prints # There are 555 rubles left for today* |