работа на уроке (18.02)

1. `with open(…) as …` `.read()`
   `.split()` `.lower()` `string.punctuation`
   Напишите функцию `parsing()`, которая получает на вход путь к файлу, а возвращает список слов из этого файла:
   a. разделенных по пробелу;
   b. с маленькой буквы;
   c. без знаков пунктуации;
   d. без пустых строк.
2. (делать после 1-й)
   `*args` `flag` `len()`
   Напишите функцию `cyrillic()`, которой может подаваться на вход одно или несколько слов в качестве аргументов. Функция должна проверить каждое слово, что оно содержит только кириллические буквы (никаких латиницы, пунктуации или цифер), и, если это так, запомнить его. По итогу выполнения функции нужно вернуть:
   a. одно подходящее слово строкой, если на вход было подано одно слово;
   b. список `right_words` подходящих слов, если на вход было подано несколько слов;
   c. `None`, если ни одного подходящего слова не оказалось (назависимо от числа аргументов)
3. (делать после 2-й)
   `len()`
   Напишите функцию `filter_words()`, у которой было бы два аргумента: путь к текстовому файлу и целое число `n`. Она должна вызвать поочередно функции `parsing()` (аргумент — путь к файлу) и `cyrillic()` (аргумент — набор предобработанных слов) внутри своего тела и, получив список кириллических слов, вернуть список тех слов из текста, длина которых больше какого-то `n`.

4. `list()` `.append()` `.extend()` `end=True` `type()`
   Напишите функцию `tuple_end()`, которая будет добавлять к кортежу элемент_ы в начале или в конце. Ей подаются три аргумента:

   a. сам кортеж (`start_tuple`)

   b. кортеж элементов или один элемент, который надо добавить к исходному кортежу (`addition`)

   c. булевый аргумент `end`, который по умолчанию равен `True` и отвечает за то, с какой стороны нужно добавлять (`end=True` — в конец, `end=False` — в начало)

   Вернуть функция должна получившийся после добавления кортеж.

   Примеры работы функции:

```
>>> tuple_end((1, 2, 3), 4)
(1, 2, 3, 4)

>>> tuple_end((1, 2, 3), 0, False)
(0, 1 ,2, 3)

>>> tuple_end((1, 2, 3), (4, 5, 6), True)
(1, 2, 3, 4, 5, 6)

>>> tuple_end((1, 2, 3), (4, 5, 6), False)
(4, 5, 6, 1, 2, 3)
```

5. Напишите функцию `c_align()`, которая принимает на вход любой текст и выводит его на экран, но не просто так, а с выравниванием по центру. Она сохраняет изначальное деление на абзацы, но на следующую строку могут переноситься части слова (пример: *r-ules*, *b-ad*).
   Сравните, если на вход подается строка (ее можно получить, если запустить `import this`):

```
s = '''The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
```

```
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!'''
```

```
>>> c_align(s)
              The Zen of Python, by Tim Peters

               Beautiful is better than ugly.
             Explicit is better than implicit.
               Simple is better than complex.
            Complex is better than complicated.
                Flat is better than nested.
                Sparse is better than dense.
                    Readability counts.
     Special cases aren't special enough to break the r
                          ules.
            Although practicality beats purity.
             Errors should never pass silently.
                Unless explicitly silenced.
     In the face of ambiguity, refuse the temptation to
                          guess.
     There should be one-- and preferably only one --ob
                    vious way to do it.
     Although that way may not be obvious at first unle
                      ss you're Dutch.
                 Now is better than never.
      Although never is often better than *right* now.
     If the implementation is hard to explain, it's a b
                         ad idea.
     If the implementation is easy to explain, it may b
                       e a good idea.
     Namespaces are one honking great idea -- let's do
                       more of those!
```

6. Напишите функцию `is_balanced()`, которая вычислит все ли открытые скобки правильно закрыты.

   Примеры запуска:

```
is_balanced('(()')
False
```

```
is_balanced('))((')
False
```

```
is_balanced('(()())')
True
```

---

1. `def log_in()`

   `input()` `.keys()`

   Дан словарь с парами имя пользователя-пароль:

   ```
   passwords = {'Alex': '123', 'Batman': '5uP3rm4n_5t1Nk5', 'little_cat': 'meow111', 'Mi
   chael': 'qwerty123'}
   ```

   Напишите функцию `log_in`, которая запрашивает от пользователя имя пользователя и пароль и сверяет их с данными в словаре. Если имя пользователя есть в словаре и пароль совпадает с соответствующим значением в словаре, то выводим на экран *добро пожаловать*. Если пароль не совпадает, пишем *неверный пароль*. Если введённого имени пользователя нет в словаре, пишем *неизвестное имя пользователя*.

   Примеры запуска:

   ```
   >>> log_in()
   Имя пользователя: little_cat
   Пароль: bow-wow
   неверный пароль
   ```

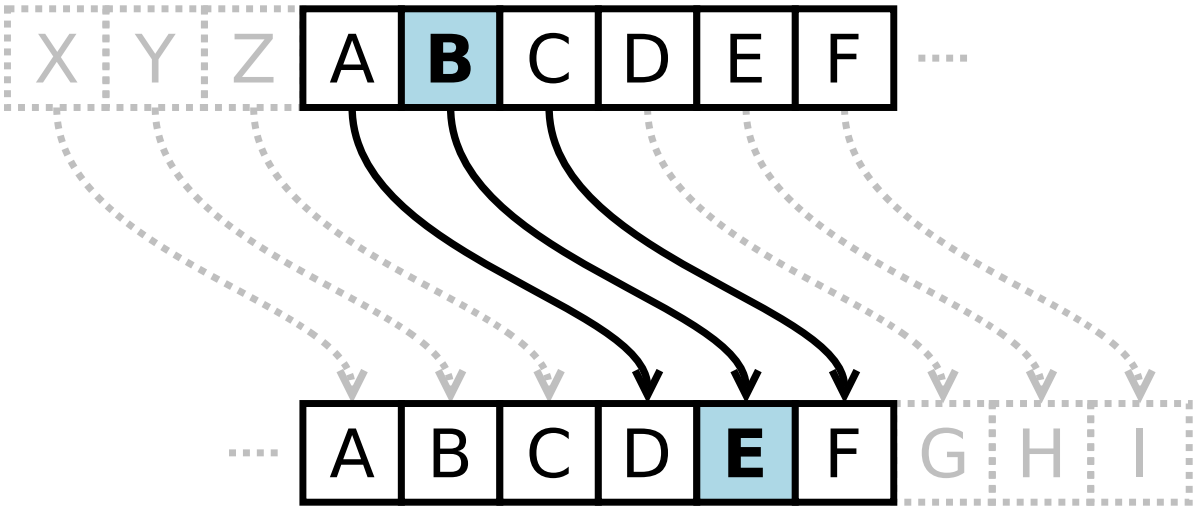
   ```
   >>> log_in()
   Имя пользователя: little_dog
   Пароль: bow-wow
   неизвестное имя пользователя
   ```

```
>>> log_in()
Имя пользователя: little_cat
Пароль: meow111
добро пожаловать
```

2. `def cypher()`

   `ord()` `chr()` `for-loop`

   Думаю, все знают, что такое Шифр Цезаря: это тот самый, где мы сдвигаем буквы в одну из сторон, например так:



   Напишите функцию `cypher`, которая принимала бы на вход строку, а возвращала бы ее в зашифрованном виде со сдвигом 2 (a → c, b → d…).
   P. S. После этого можете написать функцию `decypher`, которая помогла бы вам расшифровать эту строчку:

```
"g fmnc wms bgblr rpylqjyrc gr zw fylb. rfyrq ufyr ammnsrcpq ypc dmp. bmgle gr gl zw
 fylb gq glcddgagclr ylb rfyr'q ufw rfgq rcvr gq qm jmle. sqgle qrpgle.myicrpylq() gq
pcammmclbcb))"
```

   P. P. S. Напишите функцию `cypher_upgrade`, которая принимала бы на вход не только строку на шифровку, но и число, равное тому, на сколько надо сдвигать все буквы.

3. `def vowels()`

   `with open(…) as …:` `'r'` `.read()` `for-loop`

Напишите функцию `vowels`, которая получает название файла (он должен находиться в той же папке, что и запускаемый код, поэтому будет достаточно просто прочитать файл по передаваемому названию).

Верните количество гласных и согласных букв в этом файле. Предполагается, что файл будет либо на русском, либо на английском языке.

4. `def find_key()` `for-loop` `tuple()`
   Все постоянно жалуются, что в словаре можно найти значение по ключу, но не ключ по значению (что логично, так как это соответствие может не быть уникальным). Напишите функцию `find_key`, которая принимает на вход словарь и значение из этого словаря и возвращает ключ для данного значения. Если их несколько, вернуть кортеж ключей (`tuple`).

3. `def xxx()` `for-loop`
   `re.sub()`
   У всех иногда западают клавиши на ноутбуке и получается что-то такое: `автоооооооомат`. Давайте напишем функцию `xxx`, которая исправит эту оплошность. Считаем, что не бывает слов, где 2 подряд идущие буквы - это правильно. Для нас это всегда неверно. Ваша функция должна принимать одно слово и возвращать одно слово с исправлением.
   P. S. Эта задачка легко решается через регулярки, но не обязательно (я не настаиваю) …

   Примеры работы функции:

```
'a' -> 'a'
'ax' -> 'ax'
'axx' -> 'ax'
'axxxb' -> 'axb'
'axxxbxxxc' -> 'axbxc'
```