Занятие 9

Вложенные словари

Работа с текстовыми файлами

Начнем с ...

- 1. Разминка (Что напечатает?)
- 2. Обсуждение домашнего задания

Что напечатает?

```
print(sorted([1, -5, 2, -4, 3, float('inf')], key = lambda x: abs(x)))

print(sorted([1, 2, 3, 111, 222, 333], key = lambda x: str(x)))

print(sorted([1, 0, [], [0], (), (0,), (0)], key = lambda x: bool(x)))

print(sorted(['Hello', 'This', 'Crazy', 'World']))

print(sorted(['Hello', 'This', 'Crazy', 'World'], key = lambda x: x[::-1]))
```

Задача 8-1

На вход программе подается строка генетического кода, состоящая из букв А (аденин), Г (гуанин), Ц (цитозин), Т (тимин).
Подкорректируйте код.
Если рядом стоят А и Г, то поменяйте их местами.
Если рядом стоят Ц и Т, то поместите АГ между ними.

Задача 8-2

На вход подается список, состоящий из списков чисел, например:

lst = [[1,5,3], [2,44,1, 4], [3,3]]

Отсортируйте этот список по возрастанию общего количества цифр в каждом списочке.

Каждый списочек отсортируйте по убыванию.

Такие словари называются вложенными. И для указания элементов необходимо указывать несколько индексов, например:

lst[0][0] = 1, lst[0][1] = 5, lst[1][0] = 2, lst[1][3] = 4

Что напечатает print(lst[1][1])?

Задача 8-3

Дан список слов. Отсортируйте его по количеству уникальных букв в каждом слове в обратном порядке.

Например: [‘abab’, ‘xx’, ‘aaaaaaa’, ‘abcbab’].

Результат: [‘abcbab’, ‘abab’, ‘aaaaaaa’, ‘xx’]

Если число уникальных букв одинаково, то порядок алфавитный.

Пример

```
def fun1(x): return x % 10
def fun2(x): return (x % 10, x)

spi = [222, 21, 1, 111, 12, 322]

print(sorted(spi, key = fun1)) #   print(sorted(spi, key = lambda x: x % 10))

print(sorted(spi, key = fun2)) #   print(sorted(spi, key = lambda x: (x % 10, x)))
```

Что напечатает?

```
print(sorted(spi, key = lambda x: (x%10, (x // 10) % 10)))
```

Lambda в sort, sorted, max, min

- max(lst, key = abs)
- max(lst, key = lambda x: abs(x))

Но можно и более сложные функции:

- Отсортировать список целых чисел по возрастанию, но сначала четные числа, а потом нечетные.
- sorted(lst, key = lambda x: (x%2, x))

- Что надо изменить, чтобы сначала были нечетные числа, а потом четные?

- Отсортируйте список слов не зависимо от регистра, например:

Вход: [‘b’, ‘A’, ‘Z’, ‘x’] Выход: [‘A’, ‘b’, ‘x’, ‘Z’]

functools.**reduce**(*function*, *iterable*[, *initializer*])

- функция reduce() принимает два обязательных параметра — функцию и список.
- Сперва она применяет стоящую первым аргументом функцию для двух начальных элементов списка, а затем использует в качестве аргументов этой функции полученное значение вместе со следующим элементом списка и так до тех пор, пока весь список не будет пройден, а итоговое значение не будет возвращено.
- Например: reduce(lambda x, y: x + y, [1,2,3,4,5]) вычисляет ((((1+2)+3)+4)+5)
- Для того, чтобы использовать reduce(), вы должны сначала импортировать ее из модуля functools.
- Что напечатает reduce(lambda x, y: x + y, [1,2,3,4,5], 100)?

Задание

Дан список из кортежей (Фамилия, премия).

Напечатать эти кортежи в порядке убывания премии.

Тех, у кого одинаковая премия, то печатать в алфавитном порядке.

Например: [(Иванов, 100), (Петров, 200), (Сидоров, 200), (Воробьев, 100), (Лунин, 200)]

Результат:

Лунин 200

Петров 200

Сидоров 200

Воробьев 100

Иванов 100

print(sorted(lst, lambda x: (-x[1], x[0])))

Что напечатает?

print(sorted(lst, lambda x: (x[1], x[0])))

Методы словарей

- **dict.clear**() - очищает словарь.
- **dict.copy**() - возвращает копию словаря.
- classmethod **dict.fromkeys**(seq[, value]) - создает словарь с ключами из seq и значением value (по умолчанию None).
- **dict.get**(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).
- **dict.items**() - возвращает пары (ключ, значение).
- **dict.keys**() - возвращает ключи в словаре.
- **dict.pop**(key[, default]) - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).
- **dict.popitem**() - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.
- **dict.setdefault**(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).
- **dict.update**([other]) - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).
- **dict.values**() - возвращает значения в словаре.

Вложенные словари

Ключами (keys) словаря могут быть только неизменяемые объекты: числа, строки, кортежи, True, False и некоторые другие.

{True:1, False:2, 1:2, '1':2, frozenset((1,2)):[1,2]}

Значениями (values) может быть все, что угодно: числа, строки, кортежи, True, False, а также: списки, множества, функции, лямбда функции…

a = {True:abs, False:(lambda x:x * x), 3:{1,2,3}, '1':[1,2], 'Город':'Санкт-Петербург’}

Что напечатает?

a[True](-123)

a[False](5)

А могут быть и словари!!!

Что напечатает students[0]?

students[1]?

students[0]['name']?

students[1]['age']?

Для обращения к элементам внутренних словарей нужны 2 ключа!

Напечатайте все пары всех значений всех словарей (словарь имеет два уровня вложенности)

Задание

Дан словарь dct с **двумя** уровнями вложенности.

Введите ключ x и напечатайте все значения всех словарей, у которых ключ совпадает с x.

Например:

dct = {1: 123, 2:234, 3:{1:111, 2:222}, 4:{1:’abc’, 2: ‘def’}}

x = 1

Результат: 123 111 abc

Подсказка: функция type() определяет тип аргумента: int, str, list, dict и др.

import collections

collections.Counter - вид словаря, который позволяет нам считать количество неизменяемых объектов (в большинстве случаев, строк).

collections.deque(iterable, [maxlen]) - создаёт очередь из итерируемого объекта с максимальной длиной maxlen. Очереди очень похожи на списки, за исключением того, что добавлять и удалять элементы можно либо справа, либо слева.

collections.defaultdict ничем не отличается от обычного словаря за исключением того, что по умолчанию всегда вызывается функция, возвращающая значение

collections.OrderedDict - ещё один похожий на словарь объект, но он помнит порядок, в котором ему были даны ключи.

collections.namedtuple позволяет создать тип данных, ведущий себя как кортеж, с тем дополнением, что каждому элементу присваивается имя, по которому можно в дальнейшем получать доступ

collections.Counter

```
import collections
a = collections.Counter('aabbbcccddddeeeeeabcdef')
print(a)
b = dict(a)
print(b)
print(a.keys())
```

```
Counter({'c': 5, 'd': 5, 'e': 5, 'b': 4, 'a': 3, 'f': 1})
{'a': 3, 'b': 4, 'c': 5, 'd': 5, 'e': 5, 'f': 1}
dict_keys(['a', 'b', 'c', 'd', 'e', 'f'])
```

Работа с файлами

TXT

- TXT — это формат файлов, который содержит текст, упорядоченный по строкам.
- Текстовые файлы отличаются от двоичных файлов, содержащих данные, не предназначенные для интерпретирования в качестве текста (закодированный звук или изображение).

- Что мы делаем с файлом?
  - Открыть
  - Прочитать
  - Дописать
  - Переписать
  - Закрыть!!!

Открытие файла

- Прежде, чем работать с файлом, его надо открыть.

Для этой задачи есть встроенная функция open:

- f = open("test.txt", **encoding=**"utf-8")

- Результатом работы функция open возвращает специальный объект, который позволяет работать с файлом (файловый дескриптор)

Создайте в PyCharm текстовый файл test.txt, введите туда 4-5 строк:

First string

Second string

Третья строка

Четвертая строка

Синтаксис функции open()

fp = open(**file, mode='r',** buffering=-1, **encoding**=None, errors=None, newline=None, closefd=True, opener=None)

Параметры:

- **file** - абсолютное или относительное значение пути к файлу или файловый дескриптор открываемого файла.
- **mode** - необязательно, строка, которая указывает режим, в котором открывается файл. По умолчанию 'r'.
- **buffering** - необязательно, целое число, используемое для установки политики буферизации.
- **encoding** - необязательно, кодировка, используемая для декодирования или кодирования файла.
- **errors** - необязательно, строка, которая указывает, как должны обрабатываться ошибки кодирования и декодирования. Не используется в бинарном режиме
- **newline** - необязательно, режим перевода строк. Варианты: None, '\n', '\r' и '\r\n'. Следует использовать только для текстовых файлов.
- **closefd** - необязательно, bool, флаг закрытия файлового дескриптора.
- **opener** - необязательно, пользовательский объект, возвращающий открытый дескриптор файла.

Имя файла, какой файл, что делать

- У функции **open()** много параметров, нам пока важны 3 аргумента:

Первый, это имя файла.

Путь к файлу может быть относительным или абсолютным.

- **Второй** аргумент - это режим, mode, в котором мы будем открывать файл. Режим обычно состоит из двух букв, первой является тип файла - текстовый или бинарный, в котором мы хотим открыть файл, а второй указывает, что именно мы хотим сделать с файлом.
- **Третий** аргумент – кодировка файла

Первая буква режима:

"b" - открытие в двоичном режиме.
"t" - открытие в текстовом режиме (является значением по умолчанию).

Второй буква режима:

"r" - открытие на чтение (является значением по умолчанию).
"w" - открытие на запись, содержимое файла удаляется, если файла не существует, создается новый.
"x" - эксклюзивное создание(открытие на запись), бросается исключение FileExistsError, если файл уже существует.
"a" - открытие на дозапись, информация добавляется в конец файла.
"+" - открытие на чтение и запись

Примеры

- # Режим "w" открывает файл только для записи.
- Перезаписывает файл, если файл существует.
- Если файл не существует, создает новый файл для записи.
- f = open("test.txt", mode="w" **encoding=**"utf-8")

- # Открывает файл в бинарном режиме для записи и чтения.
- Перезаписывает существующий файл, если файл существует.
- Если файл не существует, создается новый файл для чтения и записи.
- f = open("music.mp3", mode="wb+")
- По всем режимам см. документацию open()

Закрыть файл

- После того как вы сделали всю необходимую работу с файлом - его следует закрыть.

- f = open("text.txt", encoding="utf-8")
- # какие-то действия
- f.close()

Чтение файла

- Теперь мы хотим прочитать из него информацию.
- Для этого есть несколько способов, но большого интереса заслуживают лишь два из них.
- Первый - метод **read,**
  - читающий весь файл целиком, если был вызван без аргументов, и
  - n символов, если был вызван с аргументом (целым числом n).

- ```
  f = open("test.txt", "r")

  print(f.read(5))
  print(f.read(5))
  print(f.read(4))
  print(f.read())

  f.close()
  ```
  ```

Фунция readlines()

- Файлы можно читать не только целиком или посимвольно, но и построчно.
- Для этого у объекта файла есть метод readlines, который возвращает список из строк файла.

- f = open("test.txt", "rt")
- print(f.readlines())
- f.close()

- Обратите внимания, что каждая строка в списке имеет в конце символ `\n`.

Задание

Прочитайте содержимое файла с помощью функции readlines()

Присвойте ее результат переменной lst

Напечатайте пронумерованный список строк.

Постарайтесь избавиться от лишних пустых строк.

Фунция readline()

- Функция readlines() загружает все строки целиком и хранит их в оперативной памяти,
- что может быть очень накладно, если файл занимает много места на жёстком диске.
- Можно читать файл построчно с помощью функции readline()

- f = **open**("text.txt", "rt")
- print(f.**readline**())
- print(f.**readline**())
- f.**close**()
- Также обратите внимание, что возвращённые строки имеют в конце символ `\n`.

Задание

Прочитайте содержимое файла с помощью функции readline()

Присвойте результат переменной lst

Напечатайте пронумерованный список строк.

Постарайтесь избавиться от лишних пустых строк.

Итерирование файла

- Ещё один способ прочитать файл построчно - использовать файл как итератор. Такой вариант считается самым оптимизированным

- f = **open**("text.txt")
- for line in f:
- print(line)

- f.close()

Запись

- Теперь рассмотрим запись в файл.
- Для того чтобы можно было записывать информацию в файл, нужно открыть файл в режиме записи.
- Для записи в файл используется функция write.
- При открытии файла на запись из него полностью удаляется предыдущая информация.

- fout = open("test.txt", "wt")
- fout.write("New string")
- fout.write("Another string")
- fout.close()

- Если вы откроете файл в текстовом редакторе, то увидите, что строки "New string" и "Another string" склеились.
- Так произошло, потому что между ними нет символа перевода строки.

writelines() print()

- Также в файлах, открытых на запись, есть метод writelines, который позволяет записать несколько строк в файл

- f = open("text.txt", "wt")
- lines = [
- "New string\n",
- "Another string\n",
-]
- f.**writelines**(lines)
- f.close()

Можно использовать print(), если указать имя файла:

print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

По умолчанию стандартный вывод на экран, а можно file = ‘test_out.txt’

Напишите программу, которая печатает в текстовый файл строчки из числа и его квадрата, т.е.

0 0

1 1

2 4

3 9 и т.д.

Задание

- Прочитать информацию из файла test.txt.
- Записать в файл test1.txt, только те строки, которые содержат цифры.
- Например:

- Hello!
- This is the 1st letter.
- Bye

- Результат: This is the 1st letter.

Дозапись

- Если нужно записать в конец файла какую-то информацию, то можно сделать это, открыв файл в режиме дозаписи.
- Все методы, доступные в режиме записи также доступны в режиме дозаписи.

- f = open("text.txt", "at")

```
f.write("First string\n")
lines = [
    "Second string\n",
    "Third string\n",
]
f.writelines(lines)

f.close()
```

- Измените в последней программе w на a. Запустите и посмотрите, что получилось.

Запись с возможностью чтения

- Иногда нужно открыть файл с возможностью и записи, и чтения.
- В Python есть два режима:

  * Запись с возможностью чтения ("w+")
  * Чтение с возможностью записи ("r+")

  На первый взгляд кажется, что они ничем не отличаются, но это не так.

- При открытии файла на запись (w+) с возможностью чтения из файла полностью удаляется вся информация.

Пример

```
f = open("test.txt", "w+t")

print(f.read())
f.write("Hello\n")
print(f.read())

f.close()


# Теперь измените  w+ на r+
```

Указатель позиции

- При чтении файла функция `read` читает символы друг за другом, а при записи в файл все строки (строки байт) записываются последовательно друг за другом.
- Это поведение объясняется тем, что python хранит специальный указатель, позиция этого указателя говорит, с какого места читать из файла или писать в файл.

  Независимо от того в каком режиме открыт файл у каждого объекта файла есть методы `**tell**` и `**seek**`.
- Метод `**tell**` возвращает целое число - позицию, где сейчас находится указатель.
- Метод `**seek**` принимает целое число и переносит указатель в указанную позицию.
- Например, передвинуть указатель на две позиции вперёд можно следующим образом

- ```
  position = f.tell()
  f.seek(position + 2)
  ```
  ```

Пример

- `# Начать чтение с 3 символа строки`
- `f = open('testFile.txt', 'r')`
- `f.seek(3)`
- `print(f.read())`

Задание

- Откройте текстовый файл.
- Каждый второй знак этого файла перенесите в другой файл.

Задание

Прочитать строки текста из одного файла,
отсортировать слова внутри строки по возрастанию и
записать обновленные строки в другой файл.