АКАДЕМИЯ АЙТИ

День 16

academyit.ru

* Строки. Методы и функции. Использование срезов.
* Кортеж. Основные операции с кортежем. Распаковка кортежа.
* Список. Основные операции со списком.
* Словарь. Основные операции со словарем.
* Множества. Основные операции с множеством

Турашова Анна Николаевна
Преподаватель
anna1turashova@gmail.com
Telegram: @anna1tur

Поверка домашнего задания

Задача №3760. Словарь синонимов

Вам дан словарь, состоящий из пар слов. Каждое слово является синонимом к парному ему слову. Все слова в словаре различны. Для одного данного слова определите его синоним.

Входные данные

Программа получает на вход количество пар синонимов N. Далее следует N строк, каждая строка содержит ровно два слова-синонима. После этого следует одно слово.

Выходные данные

Программа должна вывести синоним к данному слову.

Примечание

Эту задачу можно решить и без словарей (сохранив все входные данные в списке), но решение со словарем будет более простым.

Примеры

входные данные
3 Hello Hi Bye Goodbye List Array Goodbye
выходные данные
Bye

Задача №3764. Частотный анализ

Дан текст. Выведите все слова, встречающиеся в тексте, по одному на каждую строку. Слова должны быть отсортированы по убыванию их количества появления в тексте, а при одинаковой частоте появления — в лексикографическом порядке.

Указание. После того, как вы создадите словарь всех слов, вам захочется отсортировать его по частоте встречаемости слова. Желаемого можно добиться, если создать список, элементами которого будут кортежи из двух элементов: частота встречаемости слова и само слово. Например, `[(2, 'hi'), (1, 'what'), (3, 'is')]`. Тогда стандартная сортировка будет сортировать список кортежей, при этом кортежи сравниваются по первому элементу, а если они равны — то по второму. Это **почти** то, что требуется в задаче.

Входные данные

Вводится текст.

Выходные данные

Выведите ответ на задачу.

Примеры

входные данные
hi hi what is your name my name is bond james bond my name is damme van damme claude van damme jean claude van damme
выходные данные
damme is name van bond claude hi my james jean what your

Задание 3

Напишите функцию **def f7(list1)**, которая будет выполняться рекурсивно и определять максимальное число из списка.

Ввод: list1 = [400, 1300, 700, 561, 123]
Вывод: 1300

Задание 4

Напишите функцию **def f3(list1)**, которая будет выполняться рекурсивно и вычислять сумму чисел из списка (с поддержкой вложенных списков).

Ввод: list1 = [-3, 8, 4, -7, [-1, 8, [2, [-4, 3]]]]
Вывод: 10

Продвинутые функции

Что такое генератор и как он работает?

- Генератор — это объект, который сразу при создании не вычисляет значения всех своих элементов.
- Он хранит в памяти только последний вычисленный элемент, правило перехода к следующему и условие, при котором выполнение прерывается.
- Вычисление следующего значения происходит лишь при выполнении метода `next()`. Предыдущее значение при этом теряется.

Этим генераторы отличаются от списков — те хранят в памяти все свои элементы, и удалить их можно только программно. Вычисления с помощью генераторов называются ленивыми, они экономят память.

Рассмотрим пример: создадим объект-генератор gen с помощью так называемого генераторного выражения. Он будет считать квадраты чисел от 1 до 4 — такую последовательность создаёт функция `range(1,5)`.

При четырёх вызовах метода `next(a)` будут по одному рассчитываться и выводиться на консоль значения генератора: `1`, `4`, `9`, `16`. Причём в памяти будет сохраняться только последнее значение, а предыдущие сотрутся.

Функции.

Lambda.

Лямбда-функция — это как обычная функция, но без имени:

```
type(lambda x: x ** 2)
```

```
function
```

`Lambda` можно применять с `filter`:

```
list(filter(lambda x: x > 0, range(-2, 3)))
```

```
[1, 2]
```

Упражнение: написать функцию, которая превращает список чисел в список строк.

```
def stringify_list(num_list):
    return list(map(str, num_list))

stringify_list(range(10))
```

```
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

Функции.

Map.

Иногда бывает необходимо применить какую-то функцию к набору элементов. Для этих целей существует несколько стандартных функций. Одна из таких функций — это `map`, которая принимает функцию и какой-то итерабельный объект (например, список) и применяет полученную функцию ко всем элементам объекта.

```
def squarify(a):
    return a ** 2

list(map(squarify, range(5)))
```

```
[0, 1, 4, 9, 16]
```

Обратите внимание на вызов функции `list` вокруг `map`'а, потому что `map` по умолчанию возвращает `map object` (некий итерабельный объект).

То же самое можно сделать и без функции `map`, но более длинно:

```
squared_list = []
for number in range(5):
    squared_list.append(squarify(number))

print(squared_list)
```

```
[0, 1, 4, 9, 16]
```

Функции.

`Filter`.

Ещё одна функция, которая часто используется в контексте функционального программирования, это функция `filter`. Функция `filter` позволяет фильтровать по какому-то предикату итерабельный объект. Она принимает на вход функцию-условие и сам итерабельный объект.

```
def is_positive(a):
    return a > 0

list(filter(is_positive, range(-2, 3)))
```

```
[1, 2]
```

Заметим, что несмотря на то, что map и filter очень мощны, не стоит злоупотреблять ими, т.к. это ухудшает читаемость кода.
Если мы хотим передать в map небольшую функцию, которая нам больше не понадобится, можно использовать анонимные функции (или lambda-функции). Lambda позволяет вам определить функцию in place, то есть без литерала def. Сделаем то же самое, что и в предыдущем примере, с помощью lambda:

```
list(map(lambda x: x ** 2, range(5))
```

```
[0, 1, 4, 9, 16]
```

Функции.

List comprehensions (generator).

До этого момента мы с вами определяли списки стандартным способом, однако в питоне существует более красивая и лаконичная конструкция для создания списков и других коллекций. Раньше мы делали:

```
square_list = []
for number in range(10):
    square_list.append(number ** 2)

print(square_list)
```

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Лучше использовать списочные выражения (`list comprehensions`), то есть писать цикл прямо в квадратных скобках:

```
square_list = [number ** 2 for number in range(10)]
print(square_list)
```

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Со списочными выражениями код работает немного быстрее.

Функции.

List comprehensions (generator).

Точно так же можно написать списочное выражение с некоторым условием:

```
even_list = [num for num in range(10) if num % 2 == 0]
print(even_list)
```

```
[0, 2, 4, 6, 8]
```

С помощью `list comprehensions` можно определять словари таким образом:

```
square_map = {number: number ** 2 for number in range(5)}
print(square_map)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Если применять `list comprehensions` с фигурными скобками, но без двоеточий, мы получим `set`:

```
reminders_set = {num % 10 for num in range(100)}
print(reminders_set)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Списочные выражения позволяют вам делать вложенные списки `for` и другие сложные выражения. Тем не менее, делать это не рекомендуется, т.к. это снижает читаемость кода.
Без скобок списочное выражение возвращает генератор — объект, по которому можно итерироваться (подробнее про генераторы будет рассказано позже).

```
print(type(number ** 2 for number in range(5)))
```

```
<class 'generator'>
```

Функции.

Zip.

Точно так же можно написать списочное выражение с некоторым условием:

```
even_list = [num for num in range(10) if num % 2 == 0]
print(even_list)
```

```
[0, 2, 4, 6, 8]
```

С помощью `list comprehensions` можно определять словари таким образом:

```
square_map = {number: number ** 2 for number in range(5)}
print(square_map)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

Если применять `list comprehensions` с фигурными скобками, но без двоеточий, мы получим `set`:

```
reminders_set = {num % 10 for num in range(100)}
print(reminders_set)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Списочные выражения позволяют вам делать вложенные списки `for` и другие сложные выражения. Тем не менее, делать это не рекомендуется, т.к. это снижает читаемость кода.

Функции.
Генераторы.

Простейший генератор — это функция в которой есть оператор `yield`. Этот оператор возвращает результат, но не прерывает функцию. Пример:

```
def even_range(start, end):
    current = start
    while current < end:
        yield current
        current += 2

for number in even_range(0, 10):
    print(number)
```

```
0
2
4
6
8
```

Функции.
Генераторы.

Генератор even_range прибавляет к числу двойку и делает с ним операцию yield, пока current < end. Каждый раз, когда выполняется yield, возвращается значение current, и каждый раз, когда мы просим следующий элемент, выполнение функции возвращается к последнему моменту, после чего она продолжает исполняться. Чтобы посмотреть, как это происходит на самом деле, можно воспользоваться функцией next, которая действительно применяется каждый раз при итерации.

```
ranger = even_range(0, 4)
next(ranger)
```

0

```
next(ranger)
```

2

```
next(ranger)
```

StopIteration Traceback (most recent call last)
<ipython-input-6-9065b0f81b55> in <module>() —-> 1 next(ranger)

StopIteration:

Мы получили ошибку, т.к. у генератора больше нет значений, которые он может выдать.

Функции.
Генераторы.

Когда применяются генераторы? Они нужны, например, тогда, когда мы хотим итерироваться по большому количеству значений, но не хотим загружать ими память. Именно поэтому стандартная функция `range()` реализована как генератор (впрочем, так было не всегда).

Приведём классический пример про числа Фибоначчи:

```
def fibonacci(number):
    a = b = 1
    for _ in range(number):
        yield a
        a, b = b, a + b

for num in fibonacci(7):
    print(num)
```

```
1
1
2
3
5
8
13
```

С таким генератором нам не нужно помнить много чисел Фибоначчи, которые быстро растут — достаточно помнить два последних числа.

Функции.
Генераторы.

Еще одна важная особенность генераторов — это возможность передавать генератору какие-то значения. Эта особенность активно используется в асинхронном программировании, о котором будет речь позднее. Пока определим генератор `accumulator`, который хранит общее количество данных и в бесконечном цикле получает с помощью оператора `yield` значение. На первой итерации генератор возвращает начально значение `total`. После этого мы можем послать данные в генератор с помощью метода генератора `send`. Поскольку генератор остановил исполнение в некоторой точке, мы можем послать в эту точку значение, которое запишется в `value`. Далее, если `value` не было передано, генератор выходит из цикла, иначе прибавляем его к `total`.

```
def accumulator():
    total = 0 while True:
        value = yield total
        print('Got: {}'.format(value))
        if not value:
            break
        total += value

generator = accumulator()
next(generator)
print('Accumulated: {}'.format(generator.send(1)))
print('Accumulated: {}'.format(generator.send(1)))
```

```
Got: 1 Accumulated: 1
Got: 1 Accumulated: 2
```

Подсистема pip и venv

Pip.

pip — система управления пакетами, которая используется для установки и управления программными пакетами, написанными на Python. Много пакетов можно найти в Python Package Index (PyPI).

Одно из главных преимуществ pip — это простота интерфейса командной строки, которая позволяет установить пакеты Python простой командой

```
pip install some-package-name
```

Важно, что `pip` предоставляет возможность управлять всеми пакетами и их версиями с помощью файла `requirements.txt`. Это позволяет эффективно воспроизводить весь необходимый список пакетов в отдельном окружении (например, на другом компьютере) или в виртуальном окружении. Это достигается с помощью правильно составленного файла requirements.txt и следующих команд:
Для создания `requirements.txt`:

```
pip freeze > requirements.txt
```

Для установки из `requirements.txt`:

```
pip install some-package-name
```

Venv.

Venv - Модуль обеспечивающий поддержку создания облегченных “виртуальных сред” с собственными каталогами, необязательно изолированными от системных каталогов. Каждая виртуальная среда имеет свой собственный двоичный файл Python (который соответствует версии двоичного файла, использованного для создания этой среды) и может иметь свой собственный независимый набор установленных пакетов Python.

Создание виртуальные среды делается путем выполнения команды venv:

```
python -m venv /путь/для/новой/виртуальной/среды
```

Активация виртуальной среды делается путем запуска скрипта activate:

```
каталог/с/пректом/venv/Scripts/activate
```

Очень важно иметь представление об этом, так как по умолчанию, каждый объект вашей системы будет использовать одинаковые каталоги для хранения и разрешения пакетов (сторонних библиотек. На первый взгляд это не выглядит чем-то значительным. Это так, но только в отношении системных пакетов, являющихся частью стандартной библиотеки Python – но сторонние пакеты – это другое дело.

Представим следующий сценарий, где у вас есть два проекта: проект А и проект Б, которые оба имеют зависимость от одной и той же библиотеки – проект В. Проблема становится явной, когда мы начинаем запрашивать разные версии проекта В. Может быть так, что проект А запрашивает версию 1.0.0, в то время как проект Б запрашивает более новую версию 2.0.0, к примеру. Это большая проблема Python, поскольку он не может различать версии в каталоге «site-packages». Так что обе версии 1.0.0 и 2.0.0 будут находиться с тем же именем в одном каталоге.

И так как проекты хранятся в соответствии с их названиями, то нет различий между версиями. Таким образом, проекты А и Б должны будут использовать одну и ту же версию, что во многих случаях неприемлемо.

Тут-то и вступает в игру виртуальная среда.

Работа с файлами

Файлы

- Файл (англ. file) –именованная область данных на носителе информации (например, диске)
- Физическая организация хранения файлов и папок зависит от операционной и файловой системы
- Путь файла (или путь к файлу) — последовательное указание имен папок, через которые надо пройти, чтобы добраться до объекта. Папки (каталоги) в записи пути разделяются слешем.
- Типичная схема работы с файлом:
  - открыть файл для чтения или записи
  - выполнить чтение или запись
  - закрыть файл

Linux(Mac Os X) и Windows

- В Linux, как правило, файлы не имеют расширения. Все устройства и диски включены в общее дерево, корень которого обозначается "/"
- Для отделения имен папок в Linux используется прямой слеш "/", а не обратный, как в Windows
- Для путей к файлам в Windows используется обратный слэш и его нужно либо удваивать

```
f = open('C:\\users\\user\\1.txt')
```

либо все равно писать прямой слэш

```
f = open('C:/users/user/1.txt')
```

- Если мы указываем относительные пути, например,

```
f = open("1.txt")
```

то путь отсчитывает от основного файла с питон-программой

Открытие файла

- Для открытия файла используется функция open

  f = open(file_name, mode)

- Первый аргумент – имя файла, второй – режим открытия. Для текстового режима можно указать кодировку
  encoding = 'utf8' или encoding = 'cp1251 '

Режим	Обозначение
'r'	открытие на чтение (является значением по умолчанию).
'w'	открытие на запись, содержимое файла удаляется, если файла не существует, создается новый.
'x'	открытие на запись, если файла не существует, иначе исключение.
'a'	открытие на дозапись, информация добавляется в конец файла.
'b'	открытие в двоичном режиме.
't'	открытие в текстовом режиме (является значением по умолчанию).
'+'	открытие на чтение и запись

Чтение из файла

- Для чтения используется метод read

```
f = open("Толстой.txt", encoding="utf8")
text = f.read() # весь текст
print(len(text))
first = f.read(100) # первые 100 символов
print(first)
```

- Для чтения одной строки есть метод readline()

```
f = open("Толстой.txt", encoding="utf8")
for i in range(10):
      print(f.readline())
```

- Для чтения файла в список строк readlines()

```
lines = f.readlines()
```

Запись в файл

- Для записи в файл есть два режима: w (если файл существовал, его содержимое будет потеряно) и a — запись идет в конец файла. После выбора режима можно также ввести и символ "+", который позволяет делать и операции чтения, и операции записи
- Метод write используется для записи, при этом перевод строки не добавляется автоматом, возвращает количество записанных символов

```
f = open("example.txt", 'w')
print(f.write('123\n456'))
f.close()
f = open("example.txt", 'r')
print(f.read())
```

- В файл может писать функция print, если ей передать именованный параметр file

```
print("Hello", file = f)
```

Запись в файл

- Для записи в файл есть два режима: w (если файл существовал, его содержимое будет потеряно) и a — запись идет в конец файла. После выбора режима можно также ввести и символ "+", который позволяет делать и операции чтения, и операции записи
- Метод write используется для записи, при этом перевод строки не добавляется автоматом, возвращает количество записанных символов

```
f = open("example.txt", 'w')
print(f.write('123\n456'))
f.close()
f = open("example.txt", 'r')
print(f.read())
```

- В файл может писать функция print, если ей передать именованный параметр file

```
print("Hello", file = f)
```

Закрытие файла, блок with

- После работы с файлом он должен быть закрыт

```
f.close()
```

- После завершения программы файлы автоматически закрываются

Домашнее задание

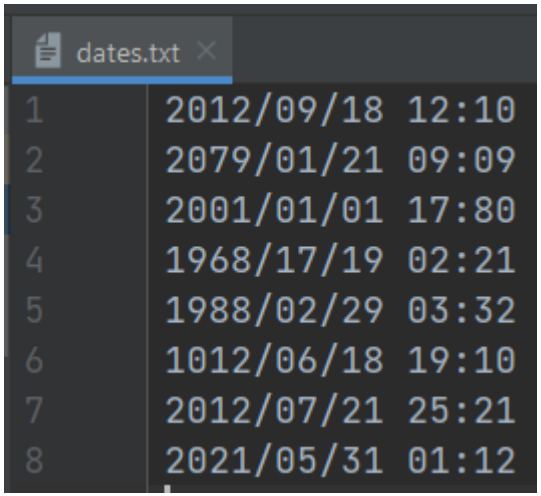
Задача 1

Создайте пустой файл pyramid.txt и напишите функцию, которая будет рисовать в этом файле пирамиду.

Функция принимает один аргумент – количество строчек в пирамиде.

Задача 2

Дан файл с датами:



2012/09/18 12:10
2079/01/21 09:09
2001/01/01 17:80
1968/17/19 02:21
1988/02/29 03:32
1012/06/18 19:10
2012/07/21 25:21
2021/05/31 01:12

Выбрать **существующие** даты в промежутке с 1950 по 2050 год.
Ответ вывести в любом формате.

Примеры:
2012/09/18 12:10 — Да
2079/01/21 09:09 — Нет (после 2050)
2001/01/01 17:30 — Да
1968/17/19 02:21 — Нет (17 месяца нет)

АПЛАНА
ГРУППА КОМПАНИЙ

Входит в ГК Аплана

АКАДЕМИЯ АЙТИ

Основана в 1995 г.

E-learning и очное обучение

Направления обучения:

- Информационные технологии
- Информационная безопасность
- ИТ-менеджмент и управление проектами
- Разработка и тестирование ПО
- Гос. и муниципальное управление

Филиалы:

Санкт-Петербург, Казань, Уфа, Челябинск, Хабаровск, Красноярск, Тюмень, Нижний Новгород, Краснодар, Волгоград, Ростов-на-Дону

Головной офис в Москве

Разработка программного обеспечения и информационных систем

Ежегодные награды Microsoft, Huawei, Cisco и другие

Программы по импортозамещению

Ресурсы более 400 высококлассных экспертов и преподавателей

Сеть региональных учебных центров по всей России

Крупные заказчики

СБЕР

РЖД

Билайн

TATNEFT

НОРНИКЕЛЬ

100+
сотрудников

АКАДЕМИЯ АЙТИ

Спасибо за внимание!

Центральный офис:
Москва, Варшавское шоссе 47, корп. 4, 7 этаж
Тел: +7 (495) 150-96-00
academy@it.ru
academyit.ru