

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Ю.А. Иванова, В.Ю. Полищук, Е.С. Попова

Программирование на Python

Методические указания для выполнения лабораторных работ

Рекомендовано в качестве учебного пособия

Редакционно-издательским советом

Томского политехнического университета

Томский политехнический университет

2020

Инструментом программиста является компьютер, рассмотрим его устройство. Все вычисления в компьютере производятся центральным процессором. Файлы с программами хранятся в постоянной памяти (на жестком диске), а в момент выполнения загружаются во временную (оперативную) память. Ввод информации в компьютер осуществляется с помощью клавиатуры (устройства ввода), а вывод – с помощью монитора



(устройства вывода).

Компьютер хранит и обрабатывает данные в бинарном (машинном) коде, представленном 1 и 0. Первые программы писались на перфокартах – прямоугольниках из тонкого картона, представляющих информацию наличием или отсутствием отверстий в определённых позициях карты.









Писать программы вида 1010101010010101010 для человека является трудоемкой задачей, поэтому со временем появились программы-трансляторы с языка программирования, понятного человеку, на машинный язык, понятный компьютеру.

Языки программирования, которые приближены к машинному уровню, называют языками низкого уровня (например, язык ассемблера). Другой вид языков – языки высокого уровня (например, Python, Java, C#) – больше приближены к мышлению человека.



Как правило, языки программирования создавались под конкретную задачу. На сегодняшний день существует множество различных языков программирования, наиболее популярные из которых представлены в таблице.

	Язык ассемблера	Микрокомпьютеры с очень ограниченными ресурсами.		50-ые
	Fortran	Математические расчеты.		
<hr/>				
	Basic	Языки для обучения программированию.		60-70-ые
	Pascal			
	C	Системное программирование (драйвера и пр.)		ОС UNIX
<hr/>				
	C++	Включает все возможности языка C, реализует ООП подход.	Потребность в больших программах - появился подход ООП.	80-ые
<hr/>				
	Java	Крупные программы для бизнеса (ООП). Сложно написать плохую программу.	Потребность в программистах и переносимости. Автоматизировать кофемашину.	90-ые
	Python	Автоматизация рутинной деятельности (быстро), обучение программированию.		Персональные ПК, Интернет
	PHP	Разработка динамических сайтов.		
<hr/>				
	C# (.NET)	Крупные программы для бизнеса (ООП). Много общего с Java. Зависимость от продуктов Microsoft.	Обобщение и объединение: собрать всё лучшее, что было до этого.	2000-ые

Началом общения с компьютером послужил машинный код. Затем в 50-ые годы двадцатого века появился низкоуровневый язык ассемблера, наиболее приближенный к машинному уровню. Он привязан к процессору, поэтому его изучение равносильно изучению архитектуры процессора. На языке ассемблера пишут программы и сегодня, он незаменим в случае небольших устройств (микроконтроллеров), обладающих очень ограниченными ресурсами памяти.

Следующий этап (80-ые годы) характеризуется появлением объектно-ориентированного программирования (ООП), которое должно было упростить создание крупных промышленных программ. Появляется ученый – Б. Страуструп, которому недостаточно было возможностей языка C, поэтому он расширяет этот язык путем добавления ООП. Новый язык получил название C++.

В 90-ые годы появляются персональные компьютеры и сеть Интернет, потому требуются новые технологии и языки программирования. Язык Java создавался с оглядкой на C++ и с перспективой развития сети Интернет.

Примерно в одно время с Java появляется Python. Разработчик языка – математик Гвидо ван Россум занимался долгое время разработкой языка ABC, предназначенного для обучения программированию. С ростом сети Интернет потребовалось создавать динамические сайты – появился серверный язык программирования PHP.

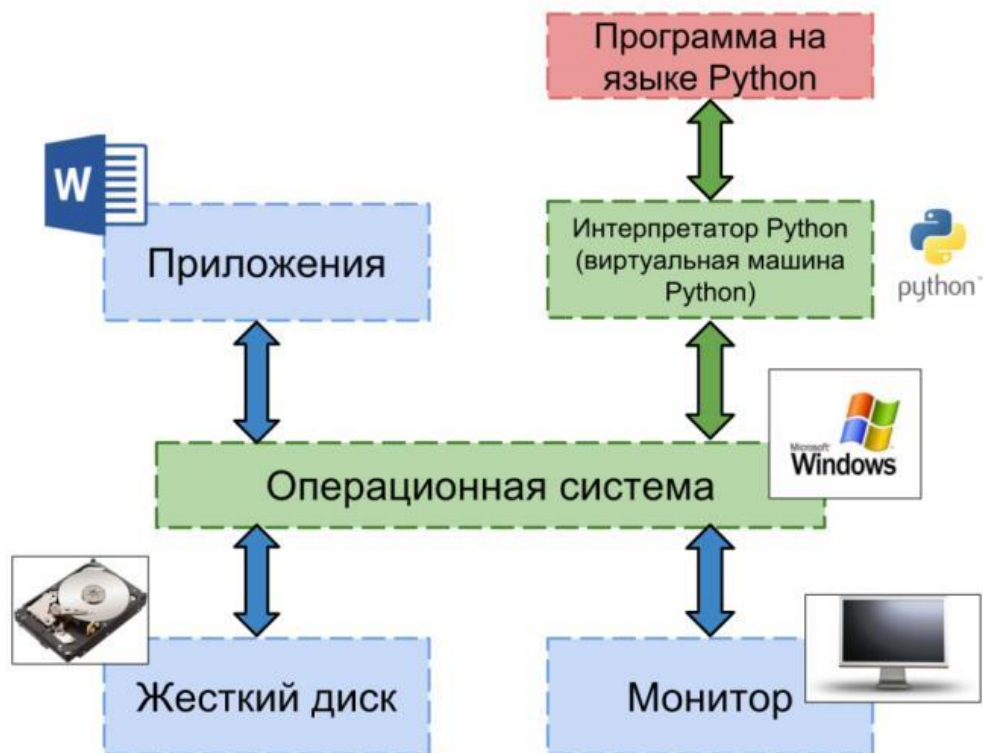
В 2000-ые годы наблюдается тенденция объединения технологий вокруг крупных корпораций. В это время получает развитие язык C# на платформе .NET.

Язык программирования Python

Питон является одним из наиболее широко используемых языков программирования в следующих областях:

1. Системное программирование.
2. Разработка программ с графическим интерфейсом.
3. Разработка динамических веб-сайтов.
4. Интеграция компонентов.
5. Разработка программ для работы с базами данных.
6. Быстрое создание прототипов.
7. Разработка программ для анализа данных.
8. Разработка программ для научных вычислений.
9. Разработка игр.

Выполнение программ осуществляется операционной системой (Windows, Linux и пр.). В задачи операционной системы входит распределение ресурсов (оперативной памяти и пр.) для программы, запрет или разрешение на доступ к устройствам ввода/вывода и т. д.



Для запуска программ на языке Python необходима программа-интерпретатор (виртуальная машина) Python. Данная программа скрывает от Python-программиста все особенности операционной системы, поэтому, написав программу на Python в системе Windows, ее можно запустить, например, в GNU/Linux и получить такой же результат.

Начало работы с Google Colab

Лабораторные работы будут осуществляться в лаборатории Гугл (Google Colaboratory), в нее можно перейти по следующей ссылке:

<https://colab.research.google.com/notebooks/welcome> (рис.1).

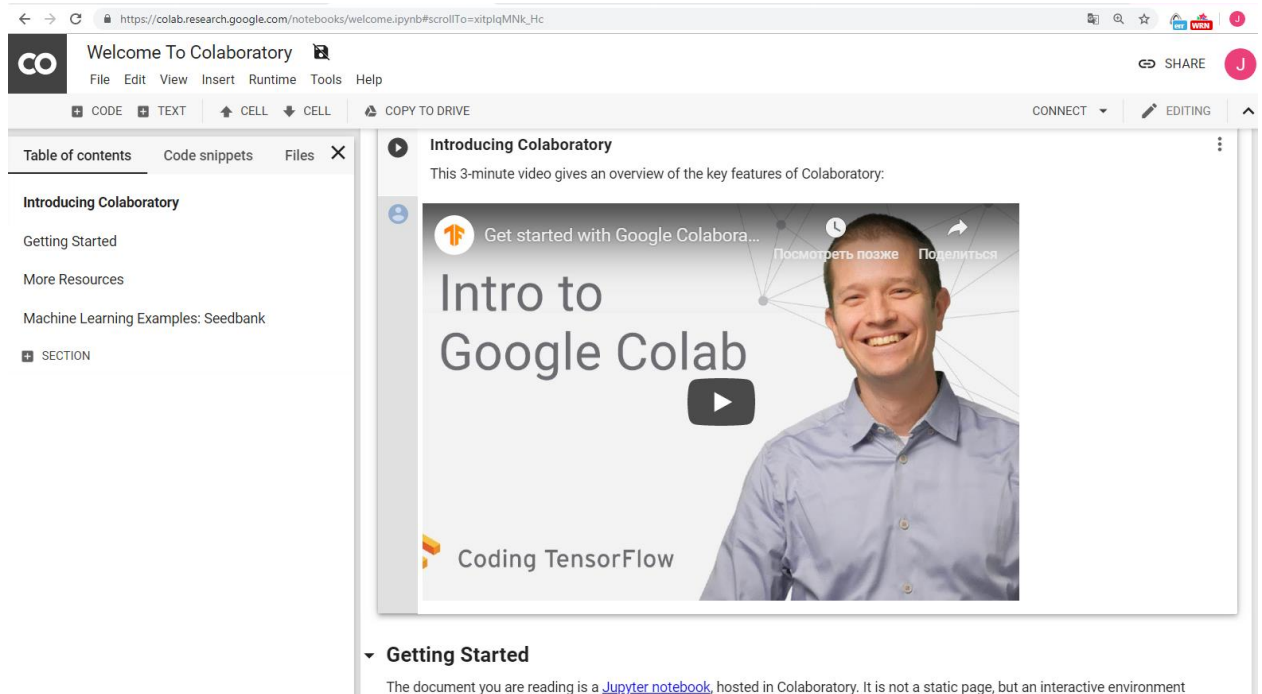


Рис. 1. Начальная страница лаборатории Гугл

Работа будет вестись в блокноте Jupyter (<https://jupyter.org/>), размещенным в лаборатории. Блокнот Jupyter представляет собой не статическую страницу, а интерактивную среду, которая позволяет писать и выполнять код на Python и других языках.

Отличительной особенностью написания кода в блокноте Jupyter является то, что код разбивается на ячейки (cell), каждая из которых может быть выполнена отдельно.

Файлы, создаваемые и редактируемые в блокноте Jupyter имеют расширения **.ipynb**.

Для начала работы можно запустить код, представленный на первой странице лаборатории (рис.2). Для этого нужно нажать на стрелку в соответствующей ячейке или выделить ячейку и нажать **ctrl+enter**. Для выполнения кода последовательно во всех ячейках можно воспользоваться сочетанием клавиш **ctrl+F9**.

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

86400

Рис. 2. Пример кода на языке Python в блокноте Jupyter

В данной программе производится подсчет количества секунд в сутках. Переменной **seconds_in_a_day** присваивается значение, равное $24(\text{часа}) * 60(\text{минут}) * 60(\text{секунд})$. Вторая строка предназначена для вывода значения, хранящегося в переменной **seconds_in_a_day**.

Для создания нового файла вашей программы нужно перейти в меню File->New Python 3 notebook.

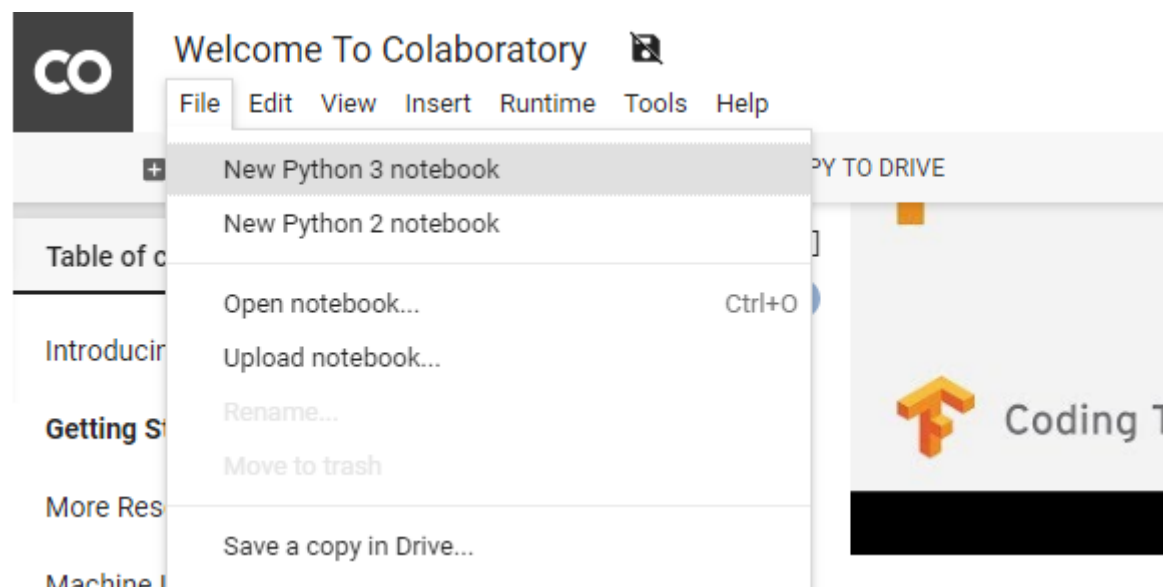


Рис. 3. Создание нового программного файла .ipynb

В новой вкладке откроется файл Untitled2.ipynb (рис. 4), с которым вы будете работать далее.

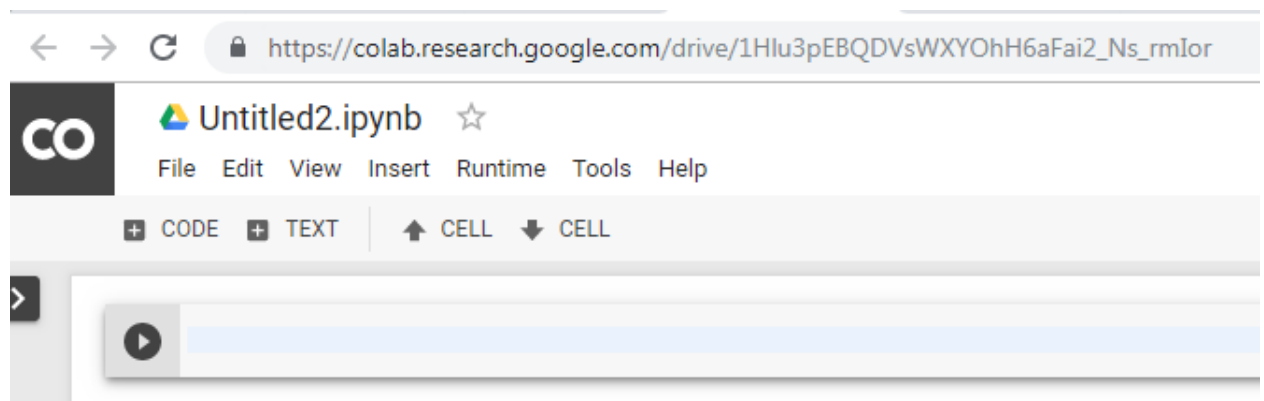
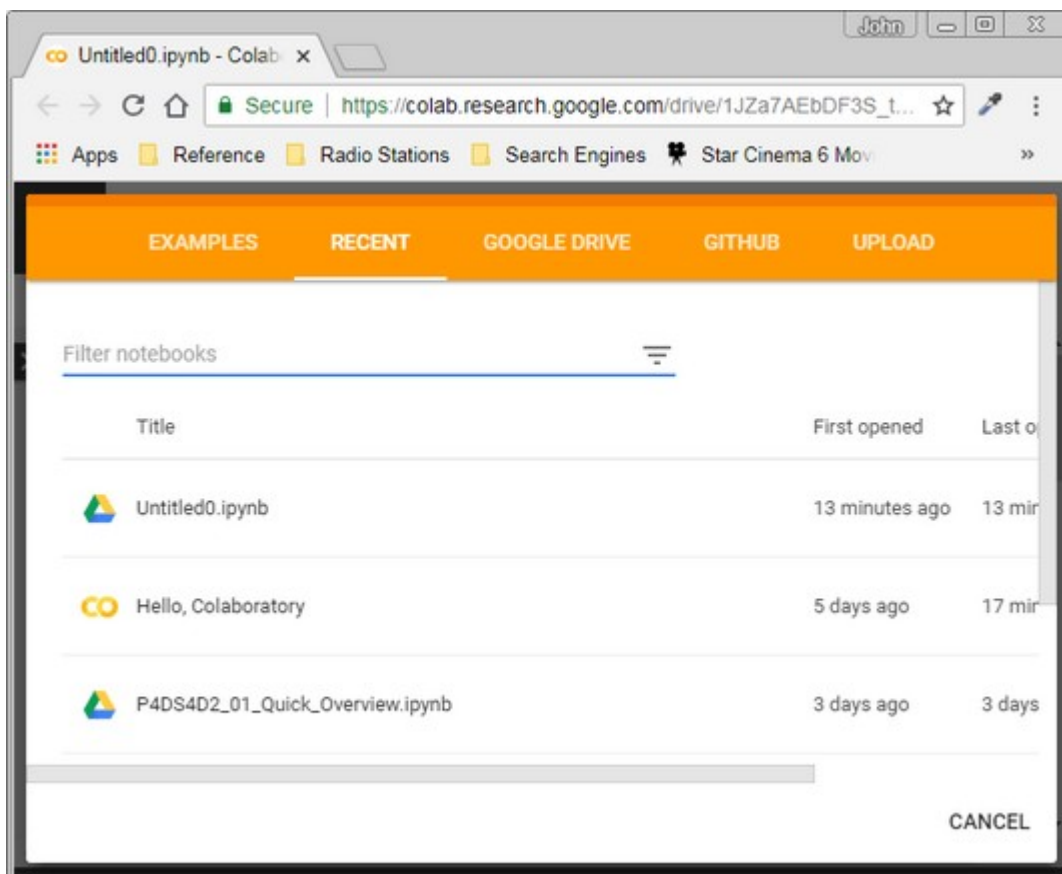


Рис. 4. Пример нового программного файла

Переименовать файл можно выбрав в меню File→Rename.

Открытие ранее созданных проектов осуществляется выбором File → Open Notebook. Появится диалоговое окно, отображающее недавно открытые файлы.



Для того чтобы сохранить файл с новым именем на жесткий диск необходимо в меню выбрать File->download .ipynb. После выполнения лабораторной работы нужно будет сохранять файлы на диске.

Для добавления новой ячейки выберите Insert->Code cell (рис.5).

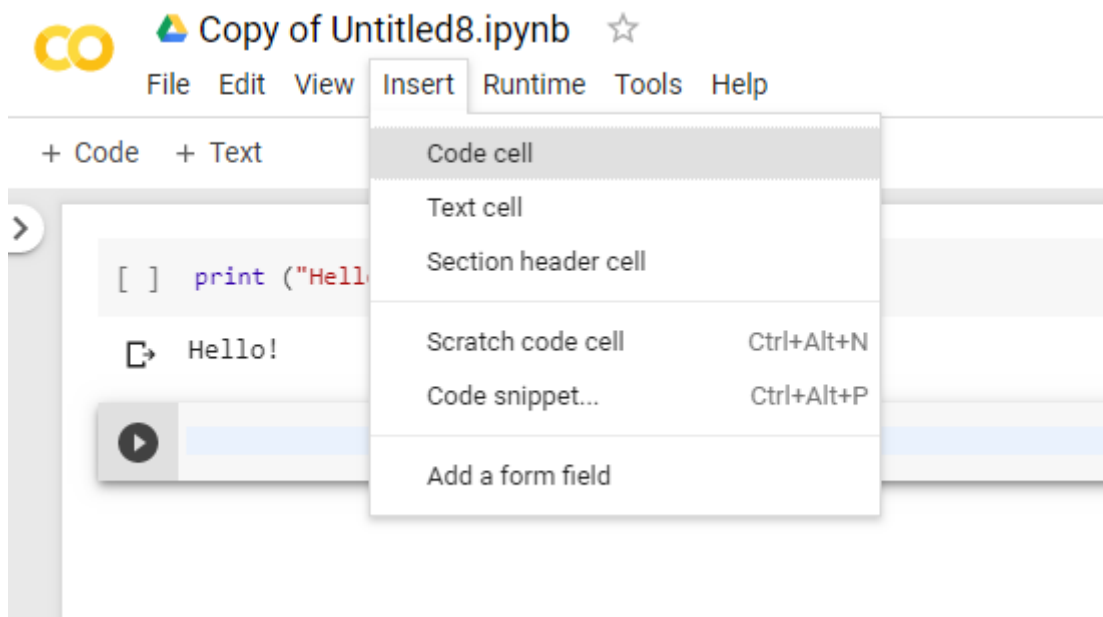


Рис. 5. Добавление новой ячейки кода

При наведении курсора мыши на ячейку кода, справа вверху появляется всплывающее меню (рис. 6). Нажатие на значок «сообщение» позволяет добавить комментарий к ячейке. Нажатие на корзину – удалить ячейку. Нажатие на «шестиренку» даст возможность редактирования настроек данной ячейки.

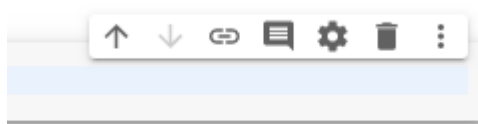


Рис. 6. Всплывающее окно ячейки кода

Автоматическая подсказка синтаксиса кода

Дополнения кода и подсказки из документации происходят автоматически при вводе. Для этого используйте следующие сочетания клавиш:

Ctrl-пробел, чтобы заново открыть завершение кода.

Ctrl-shift-пробел, чтобы заново открыть подсказки параметров.

```
1 import pandas as pd
2
3 pd.D
```

DataFrame	pd.DataFrame(data=None, index=None, columns =None, dtype=None, copy=False)
DateOffset	
DatetimeIndex	
DatetimeTZDtype	

Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

Parameters

data : ndarray (structured or homogeneous), Iterable, dict, or DataFrame

Для ознакомления с функциями языка Python можно воспользоваться документацией по языку Python: <https://docs.python.org/3/tutorial/>

Задания к лабораторным работам

Лабораторная работа №1. Введение в Python

Задание 1

Напишите программу для решения примера (по вариантам).

Предусмотрите проверку деления на ноль. Все необходимые переменные пользователь вводит через консоль. Запись $|$ пример $|$ означает «взять по модулю», т.е. если значение получится отрицательным, необходимо сменить знак с минуса на плюс.

Для вычисления примеров вам понадобится библиотека `math`. Подключить ее можно, записав в ячейке кода: `import math`.

Вариант 1.

$$k = \ln \left| (y - \sqrt{|x|}) \cdot \left(x - \frac{y}{z + x^2 / 4} \right) \right|$$

Вариант 2.

$$d = \frac{2 \cos(x - \pi / 6)}{1/2 + \sin^2(y)} + \frac{|y - x|}{3}$$

Вариант 3.

$$w = \frac{(x / y) \cdot (z + x) \cdot e^{|x-y|} + \ln(1 + e)}{\sin^2(y) - (\sin(x) \cdot \sin(y))^2}$$

Вариант 4.

$$b = \frac{3 + e^{y-1}}{1 + x^2 \cdot |y - \operatorname{tg}(z)|}$$

Задание 2

Разработать программу для вычисления выражения и вывода полученного результата. Соответствующие исходные данные ввести с клавиатуры.

Вариант 1:

$$p = \begin{cases} \sqrt{|a \cdot b|} + 2 \cdot c, a \cdot b < -2 \\ a^3 + b^2 - c^2, -2 \leq a \cdot b \leq 2 \\ a^c - b, a \cdot b > 2 \end{cases}$$

Исходные данные: a, b, c.

Вариант 2:

$$h = \begin{cases} \arctg(x + |y|), x < y \\ \arctg(|x| + y), x > y \\ (x + y)^2, x = y \end{cases}$$

Исходные данные: x, y.

Вариант 3:

$$b = \begin{cases} \ln(x / y) + (x^2 + y)^3, x / y > 0 \\ \ln |x / y| + (x^2 + y)^3, x / y < 0 \\ (x^2 + y)^3, y \neq 0, x = 0 \\ 0, y = 0 \end{cases}$$

Исходные данные: x, y.

Вариант 4:

$$b = \begin{cases} \sin(x + y) + 2 \cdot (x + y)^2, x - y > 0 \\ \sin(x - y) + (x - y)^3, x - y < 0 \\ |x^2 + \sqrt{y}|, y \neq 0, x = 0 \\ 0, y = 0 \end{cases}$$

Исходные данные: x, y.

Лабораторная работа №2. Работа с файлами. Списки

Задание. Программа должна создавать файл *.xls, записать в него сгенерированный случайным образом массив чисел. Затем, с помощью реализованного алгоритма сортировки, одного из предложенных преподавателем, записать отсортированную последовательность чисел в ранее созданный файл *.xls.

Алгоритмы сортировки:

- Сортировка выбором
- Сортировка вставками
- Сортировка “Методом пузырька”
- Сортировка Шелла
- Быстрая сортировка

Процесс создания, сохранения и работы с файлами на диске

Для создания файла необходимо выполнить команду:

```
f = open ("file1.txt", "w +")
```

Мы объявили переменную `f`, чтобы открыть файл с именем "file1.txt". Функция `open` принимает 2 аргумента: путь к файлу, который мы хотим открыть, и строку, представляющую виды разрешений или операций, которые мы хотим выполнить над файлом.

Возможные варианты второго аргумента:

" r " Открыть текстовый файл для чтения. Поток расположен в начале файла.

" r + " Открыт для чтения и письма. Поток расположен в начале файла.

" w " Обрезать файл до нулевой длины или создать текстовый файл для записи. Поток располагается в начале файла.

" w + " Открыт для чтения и письма. Файл создается, если он не существует, в противном случае он затирается. Поток расположен в начале файла.

" a " Открыта для письма. Файл создается, если он не существует. Поток расположен в конце файла. Последующие записи к файлу всегда будет в конце текущего конца файла.

" a+ " файл открыт для чтения и письма, если файл не существует, то он создается. Поток располагается в конце файла.

	r	r+	w	w+	a	a+
Read(чтение из файла)	+	+		+		+
Write(запись в файл)		+	+	+	+	+
write after seek (возможно начала записи с произвольного участка файла)		+	+	+		
Create(создание файла)			+	+	+	+
Truncate (при открытии файл становится пустым)			+	+		
position at start (позиция при открытии устанавливается в начало файла)	+	+	+	+		
position at end (позиция при открытии устанавливается в конец файла)					+	+

В следующем примере в цикле в файл записываются строки «This is line 1,2..., n»:

```
for i in range(10):
    f.write("This is line %d\r\n" % (i+1))
f.close()
```

В цикле for задается диапазон 10 чисел. Для записи данных в файл используется функция write. В каждой строке в файл записывается строка: «This is a line», затем %d (предупреждает, что будет введено целое число), символ перевода каретки на новую строку(\r\n) и само значения номера строки %(i+1). Таким образом, в основном мы вводим номер строки, которую пишем, затем помещаем ее в символ возврата каретки и символ новой строки. В конце файл необходимо закрыть (функция f.close()).

Если вы попытаетесь записать в файл просто созданный список, то он запишется как строковые данные со скобками. Впоследствии это приведет к затруднению считывания данных. Поэтому лучше использовать указания типов при записи данных в файл. Уточним, как работает типизированный ввод данных.

```
>>> pupil = "Ben"
>>> old = 16
>>> grade = 9.2
>>> f.write ("It's %s, %d. Level: %f" % (pupil, old, grade))
It's Ben, 16. Level: 9.200000
```

В вышеприведенном коде создаются 3 переменные строкового, целого и вещественного типа. Для записи их в файл необходимо предварительно указать, в каком порядке и какого типа переменные будут сохраняться "It's %s, %d. Level: %f", а затем перечислить сами переменные. Таким образом буквы s, d, f обозначают типы данных – строку, целое число, вещественное число.

Чтение файла

Для открытия файла в режиме чтения наберите команду:

```
f = open ("file1.txt", "r")
```

Можно использовать функцию mode в коде, чтобы проверить, находится ли файл в открытом режиме. Если да, мы продолжаем

```
if f.mode == 'r':
```

Используйте f.read, чтобы прочитать данные файла и сохранить их в переменной content.

1. Способ вывода – выводим все содержимое файла в переменную content

```
#Open the file back and read the contents
```

```
f=open("file1.txt", "r")
```

```
if f.mode == 'r':
```

```
    contents =f.read()
```

```
    print (contents)
```

```
f.close()
```

Для того, чтобы получить из строки список отдельных элементов , можно воспользоваться функцией split("arg"), где arg – символ разбиения:

```
with open('foo.txt','r')as f:
```

```
    c=f.read().split(" ")
```

```
    print(type(c[0]))
```


Символом разбиения может быть пробел (как в примере), и любой символьный знак, например «,», «/» и др.

2. Способ вывода – поэлементное задание и вывод переменной *x*:

```
f=open("file1.txt", "r")  
#or, readlines reads the individual line into a list  
fl =f.readlines()  
for x in fl:  
    print(x)  
f.close()
```

Загрузка файла на google-диск

```
from google.colab import files // из библиотеки google.colab загружается  
                                раздел files  
with open('example.txt', 'w') as f: // открытие файла «example.txt» с правом  
                                    чтения в переменную f  
    f.write('some content') // запись в файл f текста «some content»  
files.download('example.txt') // сохранение файла на диске как «example.txt»
```

2. Способ. Для того чтобы подгрузить файл в colab вам потребуется обратиться к своему google диску. В приведенном ниже примере показано, как подключить диск Google Drive во время выполнения с помощью кода авторизации и как записывать и читать файлы там. После выполнения вы увидите новый файл (foo.txt) по адресу <https://drive.google.com/>.

После монтирование google-диска необходимо перейти по предложенной ссылке URL (рис. 7).

```
[ ] from google.colab import drive
drive.mount('/content/drive')
```

Go to this URL in a browser: <https://accounts.google.com>
Enter your authorization code:
.....
Mounted at /content/drive

Рис. 7. Процесс подключение google-диска

Самая нижняя строка на рис. 7. Оповещает о том, что ваш google-диск подключен.

Файлы вашего google-диска находятся по адресу: «/content/drive/My Drive/»

Для просмотра существующих документов на вашем google-диске воспользуемся командой `!ls` (рис. 8),

```
[113] !ls /content/drive/My\ Drive
```

↳	'Colab Notebooks'	foo.txt	'Диссер-Выделение лиц.СНС.pdf'
	file1.txt	timeline	

Рис. 8. Просмотр существующих документов на google-диске

Создадим файл `foo.txt` (рис. 9), записав в нем фразу «Hello Google Drive!».

```
with open('/content/drive/My Drive/foo.txt', 'w') as f:
    f.write('Hello Google Drive!')
!cat /content/drive/My\ Drive/foo.txt
```

↳ Hello Google Drive!

Рис. 9. Изменение содержимого файла `foo.txt` и его вывод

Последняя строка кода на рис. 9. Позволяет вывести содержимое файла «`foo.txt`», где

- `!cat` – команда вывода содержимого файла,
- `/content/drive/My\ Drive/` – путь к файлу,
- `foo.txt` – название файла.

Для дополнительной информации можно обратиться:
<https://colab.research.google.com/notebooks/io.ipynb#scrollTo=7taylj9wpsA2>

Лабораторная работа №3.

Описание объектно-ориентированного подхода можно посмотреть в пособии «book.pdf»: Глава 15. «Объектно-ориентированное программирование в Python».

Классы предоставляют средства объединения данных и функциональности вместе. Создание нового класса создает новый тип объекта, позволяя создавать новые экземпляры этого типа. Каждый экземпляр класса может иметь атрибуты, описывающие его состояние. Экземпляры класса также могут иметь методы (определяемые его классом) для изменения его состояния.

Классы Python предоставляют все стандартные возможности объектно-ориентированного программирования: механизм наследования классов позволяет использовать несколько базовых классов, производный класс может переопределять любые методы своего базового класса или классов, а метод производного класса может вызывать метод базового класса с тем же именем. Объекты могут содержать произвольное количество и типы данных. Как и в случае с модулями, классы обладают динамической природой Python: они создаются во время выполнения и могут быть изменены в дальнейшем после создания.

По умолчанию члены класса (включая элементы данных) являются общедоступными, а все функции-члены являются виртуальными. Функция метода объявляется с явным первым аргументом, представляющим объект, который неявно предоставляется вызовом. Сами классы являются объектами. Это обеспечивает семантику для импорта и переименования. Встроенные типы могут использоваться пользователем как базовые классы для расширения.

Предусмотрите инкапсуляцию полей и при возможности методов разрабатываемого класса.

Вариант 1.

Напишите программу по следующему описанию. Есть класс "Воин". От него создаются два экземпляра-юнита. Каждому устанавливается здоровье в 100 очков. В случайном порядке они бьют друг друга. Тот, кто бьет, здоровья не теряет. У того, кого бьют, оно уменьшается на 20 очков от одного удара. После каждого удара надо выводить сообщение, какой юнит атаковал, и сколько у противника осталось здоровья. Как только у кого-то заканчивается ресурс здоровья, программа завершается сообщением о том, кто одержал победу.

Вариант 2.

Напишите программу по следующему описанию:

Есть класс Person, конструктор которого принимает три параметра (не учитывая self) – имя, фамилию и квалификацию специалиста. Квалификация имеет значение заданное по умолчанию, равное единице.

У класса Person есть метод, который возвращает строку, включающую в себя всю информацию о сотруднике.

Класс Person содержит деструктор, который выводит на экран фразу "До свидания, мистер ...". (вместо троеточия должны выводиться имя и фамилия объекта).

В основной ветке программы создайте три объекта класса Person. Посмотрите информацию о сотрудниках и увольте самое слабое звено.

В конце программы добавьте функцию input(), чтобы скрипт не завершился сам, пока не будет нажат Enter. Иначе вы сразу увидите, как удаляются все объекты при завершении работы программы.

Вариант 3.

Разработайте программу по следующему описанию.

В некой игре-стратегии есть солдаты и герои. У всех есть свойство, содержащее уникальный номер объекта, и свойство, в котором хранится принадлежность команде. У солдат есть метод "иду за героем", который в качестве аргумента принимает объект типа "герой". У героев есть метод увеличения собственного уровня.

В основной ветке программы создается по одному герою для каждой команды. В цикле генерируются объекты-солдаты. Их принадлежность команде определяется случайно. Солдаты разных команд добавляются в разные списки.

Измеряется длина списков солдат противоборствующих команд и выводится на экран. У героя, принадлежащего команде с более длинным списком, поднимается уровень.

Отправьте одного из солдат первого героя следовать за ним. Выведите на экран идентификационные номера этих двух юнитов.

Вариант 4.

В качестве практической работы попробуйте самостоятельно перегрузить оператор сложения. Для его перегрузки используется метод `__add__()`. Он вызывается, когда объекты класса, имеющего данный метод, фигурируют в операции сложения, причем с левой стороны. Это значит, что в выражении `a + b` у объекта должен быть метод `__add__()`. Объект `b` может быть чем угодно, но чаще всего он бывает объектом того же класса. Объект `b` будет автоматически передаваться в метод `__add__()` в качестве второго аргумента (первый – `self`).

Отметим, в Python также есть правосторонний метод перегрузки сложения - `__radd__()`.

Согласно полиморфизму ООП, возвращать метод `__add__()` может что угодно. Может вообще ничего не возвращать, а "молча" вносить изменения в какие-то уже существующие объекты. Допустим, в вашей программе метод перегрузки сложения будет возвращать новый объект того же класса.

Лабораторная работа №4

1. Создать программу для работы с базой данных, при этом добавив обработку исключительных ситуаций. База данных должна содержать не менее 5 связанных таблиц. У программы должен быть графический интерфейс (возможность отображать таблицы с помощью библиотеки Pandas, редактировать таблицы с помощью интерактивных форм) [<https://colab.research.google.com/notebooks/forms.ipynb>].
По сохраненным данным в таблицах построить минимум 3 различных аналитических графика [<https://colab.research.google.com/notebooks/charts.ipynb>, <https://colab.research.google.com/notebooks/widgets.ipynb>] с помощью библиотеки matplotlib.
2. Реализовать вычисление линейной аппроксимации методом наименьших квадратов (линия тренда).
 - а. Необходимо найти или сгенерировать последовательность данных $y(x)$ (например, прогноз погоды, курс валют... и др.), записать их в таблицу и вывести в виде графика на экран.
 - б. Реализовать метод наименьших квадратов для аппроксимации функции $y(x)$ (не используя сторонних библиотек).
 - в. Аппроксимировать данные и нарисовать линию тренда.

Реляционные базы данных

<http://citforum.ru/database/osbd/contents.shtml>

Создание базы данных с помощью SQLite

SQLite - это библиотека, написанная на языке C, которая предоставляет легкую дисковую базу данных, не требующую отдельного серверного процесса и позволяющую получить доступ к базе данных с использованием нестандартного варианта языка запросов SQL. Некоторые приложения могут использовать SQLite для внутреннего хранения данных. Также возможно создать прототип приложения с использованием SQLite, а затем перенести код в большую базу данных, такую как PostgreSQL или Oracle. Модуль sqlite3 был написан Герхардом Херингом. Он предоставляет интерфейс SQL, соответствующий спецификации DB-API 2.0, описанной в PEP 249 [<https://docs.python.org/2/library/sqlite3.html>].

Для создания новой базы данных необходимо импортировать библиотеку sqlite3 (она уже предварительно установлена в google-лаборатории) и вызвать функцию соединения с базой. При первом вызове будет создан файл «mydatabase.db»

```
conn = sqlite3.connect("mydatabase.db") # вызов соединения с базой
```

После этого в текущей папке появится новый файл «mydatabase.db», который можно увидеть, используя команду **!ls**:

```
[46] !ls
```

```
📁 mydatabase.db  sample_data
```

Можно создавать временные базы данных, хранящиеся в оперативной памяти, при этом файл сохраняться не будет, и доступность будет ограничена текущей сессией:

```
conn = sqlite3.connect(":memory:") # создание временной бд,  
# работающей только во время запуска скрипта
```

После получения соединения, вы можете обращаться к базе данных с помощью объекта типа `cursor`. Выполнение запросов к базе выполняется с помощью его метода `execute()`:

```
[60] cursor = conn.cursor()  
# Создание таблицы  
cursor.execute("""CREATE TABLE students  
                (id INTEGER PRIMARY KEY , FIO text, yearbirth text)  
                """)
```

```
📁 <sqlite3.Cursor at 0x7fb6d151f2d0>
```

В данном примере в базе данных «mydatabase» создается таблица «students» со следующими полями: «id» – первичный ключ, обозначающий номер студента, «FIO» – ФИО студента типа «текст» и «yearbirth» – год рождения типа «текст». Добавлять поля с первичным ключом в таблицу не обязательно, он автоматически сгенерируется во время создания таблицы. Но, если вы его задали при создании таблицы, то именно это поле будет первичным ключом.

Добавление данных в таблицу осуществляется с помощью SQL-команды «INSERT».

После добавления записи необходимо сохранить изменения с помощью функции `commit()`. SQLite не пишет в базу данных, пока вы не совершите транзакцию. Транзакция состоит из 1 или более запросов, которые вносят изменения в базу данных одновременно. Это сделано для того, чтобы

упростить восстановление после случайных изменений или ошибок. Транзакции позволяют выполнить несколько запросов, а затем, наконец, изменить базу данных с результатами всех из них. Это гарантирует, что в случае сбоя одного из запросов база данных не будет частично обновлена

```
[87] # Вставляем данные в таблицу
      cursor.execute("""INSERT INTO students
                        VALUES (2, 'Andreev Ivan', '2002')""")
      )

      # Сохраняем изменения
      conn.commit()

      # Вставляем множество данных в таблицу используя безопасный метод "?"
      stud = [(1, 'Sergeev Anton', '2002'),
              (3, 'Savelieva Anastasia', '2001'),
              (5, 'Plohotnik Maxim', '2000'),
              (4, 'Ignatieva Alisa', '2002')]

      cursor.executemany("INSERT INTO students VALUES (?,?,?)", stud)
      conn.commit()
```

Вывод записей таблицы на экран:

```
▶ print("\nHere's a listing of all the records in the table:")
   for row in cursor.execute("SELECT * FROM students ORDER BY id"):
     print(row)
```



```
Here's a listing of all the records in the table:
(1, 'Sergeev Anton', '2002')
(2, 'Andreev Ivan', '2002')
(3, 'Savelieva Anastasia', '2001')
(4, 'Ignatieva Alisa', '2002')
(5, 'Plohotnik Maxim', '2000')
```

Первый запрос, который мы выполнили, называется SELECT *, что означает, что мы хотим выбрать все записи из таблицы “students”, отсортированные по полю “id”.

Удаление записи из базы данных:

```
[7] import sqlite3

      conn = sqlite3.connect("mydatabase.db")
      cursor = conn.cursor()

      sql = "DELETE FROM students WHERE FIO = 'Savelieva Anastasia'"

      cursor.execute(sql)
      conn.commit()
```

В запросе из таблицы `students` удаляются данные, у которых в поле `FIO` записано «Savelieva Anastasia»

Вывод записей таблицы на экран после удаления строки:

```
[8] print("Here's a listing of all the records in the table:")
    for row in cursor.execute("SELECT rowid, * FROM students ORDER BY FIO"):
        print(row)
```

```
↳ Here's a listing of all the records in the table:
(1, '2', 'Andreev Ivan', '2002')
(5, '4', 'Ignatieva Alisa', '2002')
(4, '5', 'Plohotnik Maxim', '2000')
(2, '1', 'Sergeev Anton', '2002')
```

Здесь мы в запросе `SELECT` указали еще `rowid` – это первичный ключ таблицы по-умолчанию. Можно пользоваться им, можно создать свой собственный, как в нашем случае поле “`id`”. Видно, что строка с `FIO` «Savelieva Anastasia» была удалена из таблицы.

Запрос на выбор всех студентов, у которых год рождения 2002:

```
[11] sql = "SELECT * FROM students WHERE yearbirth=?"
      cursor.execute(sql, ['2002'])
      print(cursor.fetchall()) # or use fetchone()
```

```
↳ (('2', 'Andreev Ivan', '2002'), ('1', 'Sergeev Anton', '2002'), ('4', 'Ignatieva Alisa', '2002'))
```

Мы выполняем `SQL` и используем функцию `fetchall()` для получения результатов. Можно также использовать функцию `fetchone()` для получения только первого результата. Функция `execute()` выполняет `sql`-запрос на выбор данных из таблицы.

Как видите, результаты отформатированы в виде списка кортежей. Каждый кортеж соответствует строке в базе данных, к которой мы обращались. Работа с данными таким способом довольно болезненна. Нам нужно вручную добавить заголовки столбцов и вручную проанализировать данные. К счастью, у библиотеки `панд` есть более простой способ, который мы рассмотрим в следующем разделе.

Пример ниже показывает, как команда `LIKE` используется при поиске частичных фраз. В нашем случае, мы искали по всей таблице `FIO`, которые начинаются с “`S`”. Знак процента (%) является подстановочным оператором.

```
[13] print("\nResults from a LIKE query:")
      sql = "SELECT * FROM students WHERE FIO LIKE 'S%'"
      cursor.execute(sql)

      print(cursor.fetchall())
```

```
↳ Results from a LIKE query:
   [('1', 'Sergeev Anton', '2002')]
```

В результате данного запроса были выбраны все студенты, чья фамилия начиналась на “S”.

Прежде чем двигаться дальше, рекомендуется закрыть открытые объекты Connection и Cursor. Это предотвращает блокировку базы данных SQLite. Когда база данных SQLite заблокирована, возможно, вы не сможете обновить базу данных и получите ошибки. Мы можем закрыть курсор и соединение следующим образом:

```
▶ cursor.close()
  conn.close()
```

Работа с SQLite и Pandas DataFrame

Для чтения результатов SQL-запроса можно использовать функцию `read_sql_query` библиотеки `pandas DataFrame`. Приведенный ниже код выполнит тот же запрос, который мы только что сделали, но вернет `DataFrame`. Он имеет несколько преимуществ по сравнению с запросом, выполненным нами ранее, т.к.

- Не требуется создание объекта `Cursor` или вызова `fetchall` в конце.
- Имена заголовков из таблицы считываются автоматически.
- Создается `DataFrame`, позволяющий быстро отобразить и изучить данные.

```
import pandas as pd
import sqlite3
conn = sqlite3.connect("mydatabase.db") # вызов соединения с базой
df = pd.read_sql_query("select * from students limit 5;", conn)
df
```

	id	FIO	yearbirth
0	1	Sergeev Anton	2002
1	2	Andreev Ivan	2002
2	3	Savelieva Anastasia	2001
3	4	Ignatieva Alisa	2002
4	5	Plohotnik Maxim	2000

В результате получаем красиво отформатированный объект DataFrame, с помощью него можно легко обратиться к определенному столбцу таблицы и вывести все имена и фамилии студентов:

```
df["FIO"]
```

0	Sergeev Anton
1	Andreev Ivan
2	Savelieva Anastasia
3	Ignatieva Alisa
4	Plohotnik Maxim

Name: FIO, dtype: object

Создание таблиц с помощью Pandas

Пакет pandas дает нам гораздо более быстрый способ создания таблиц. Нам просто нужно сначала создать DataFrame, а затем экспортировать его в таблицу SQL. Сначала мы создадим DataFrame:

```
import pandas as pd
import sqlite3
from datetime import datetime
df = pd.DataFrame(
[[1, "Sergeev Ivan Andreevich", "Informatics", datetime(2016, 9, 29, 12, 20),
datetime(2016, 9, 29, 13, 55)]],
columns=["id", "TechersFIO", "subject", "lectureBegins", "lectureEnds"])
```

Затем мы сможем вызвать метод `to_sql` для преобразования `df` в таблицу в базе данных. Мы устанавливаем параметр `if_exists` для замены, чтобы удалить и заменить любые существующие таблицы с именем “`raspisanie`”:

```
df.to_sql("raspisanie", conn, if_exists="replace")
```

Затем мы можем проверить, что все работает, запросив базу данных:

```
pd.read_sql_query("select * from raspisanie;", conn)
```

	index	id	TechersFIO	subject	lectureBegins	lectureEnds
0	0	1	Sergeev Ivan Andreevich	Informatics	2016-09-29 12:20:00	2016-09-29 13:55:00

Редактирование таблиц с помощью Pandas

При работе с данными иногда приходится изменять конфигурацию таблиц. Допустим в ранее созданную таблицу “`raspisanie`” нам необходимо добавить поле аудитория («`room`»). Сначала сделаем это с помощью курсора `cur`:

```
cur.execute("alter table raspisanie add column room integer;")
```

```
pd.read_sql_query("select * from "+table_name+";", conn)
```

	index	id	TechersFIO	subject	lectureBegins	lectureEnds	room
0	0	1	Sergeev Ivan Andreevich	Informatics	2016-09-29 12:20:00	2016-09-29 13:55:00	None

Обратите внимание, что в SQLite все столбцы имеют значение `null` (что в Python переводится как `None`), поскольку для этого столбца еще нет никаких значений.

В библиотеке Pandas также предусмотрена подобная функция редактирования таблиц:

```
[13] df = pd.read_sql("select * from raspisanie", conn)
df["building"] = None
df.to_sql("raspisanie", conn, if_exists="replace")
```

```
pd.read_sql_query("select * from "+table_name+";", conn)
```

	level_0	index	id	TechersFIO	subject	lectureBegins	lectureEnds	room	building
0	0	0	1	Sergeev Ivan Andreevich	Informatics	2016-09-29 12:20:00	2016-09-29 13:55:00	None	None

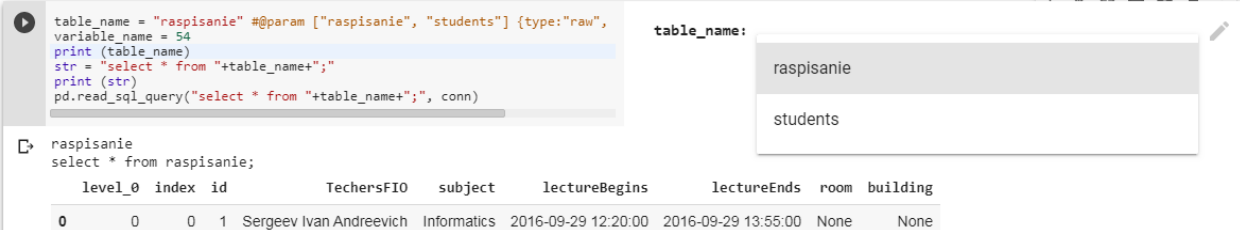
Дополнительные ссылки:

1. https://pandas.pydata.org/pandas-docs/stable/getting_started/comparison/comparison_with_sql.html
2. <https://docs.python.org/3/library/sqlite3.html>
3. http://sebastianraschka.com/Articles/2014_sqlite_in_python_tutorial.html

Создание и редактирование интерактивных форм

Формы позволяют удобно редактировать код. Выделив текущую ячейку кода необходимо выбрать раздел **Insert** → **Add form field**. При изменении значения в форме изменяется соответствующая строка кода [<https://colab.research.google.com/notebooks/forms.ipynb>].

Для сокрытия кода необходимо выбрать раздел **Edit** → **Show/hide code**. Для обращения таблицам и полям можно использовать интерактивные поля ввода:



```
table_name = "raspisanie" #@param ["raspisanie", "students"] (type:"raw",
variable_name = 54
print (table_name)
str = "select * from "+table_name+";"
print (str)
pd.read_sql_query("select * from "+table_name+";", conn)
```

table_name:

raspisanie

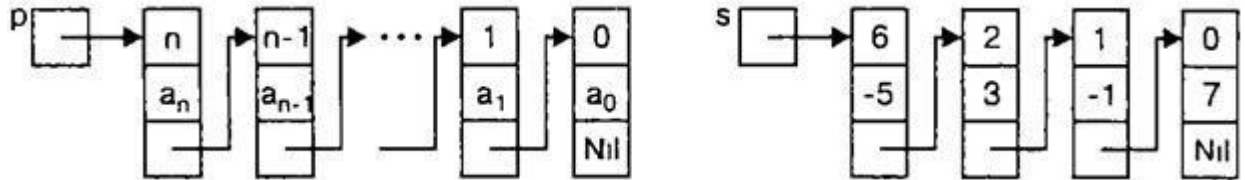
students

```
raspisanie
select * from raspisanie;
```

level_0	index	id	TechersFIO	subject	lectureBegins	lectureEnds	room	building	
0	0	0	1	Sergeev Ivan Andreevich	Informatics	2016-09-29 12:20:00	2016-09-29 13:55:00	None	None

Дополнительные лабораторные работы

1. Многочлен $P(x)=a_nx^n+a_{n-1}x^{n-1}+\dots+a_1x+a_0$ с целыми коэффициентами можно представить в виде списка. При этом, если $a_i=0$, то соответствующий элемент не включается в список. На рисунке показано общее представление многочлена и пример для $S(x)=-5x_6+3x_2-x+7$:



Необходимо описать тип данных, соответствующий предложенному представлению многочленов, а также разработать процедуру $Add(P,Q,R)$ вычисления суммы многочленов Q и R, результат – многочлен P.

2. Создать типизированный файл записей со сведениями о телефонах абонентов; каждая запись имеет поля: фамилия абонента, год установки телефона, номер телефона. По заданной фамилии абонента выдать номера его телефонов. Определить количество установленных телефонов с N-го года. Реализовать алгоритм бинарного поиска.

3. Реализовать алгоритм K средних, с графическим представлением его работы.

4. Создать приложение, которое позволяет загружать текстовый файл с html-кодом или css-кодом и выводить список всех ссылок, встречающихся в этом коде. Ссылки должны быть без повторов, однако с указанием количества повторений. Добавить возможность выбора ссылки и замены на другую по всему файлу. Добавить возможность создания списка запрещенных сайтов. Все ссылки, попадающие в черный список, можно автоматически заменить на [запрещенная ссылка].