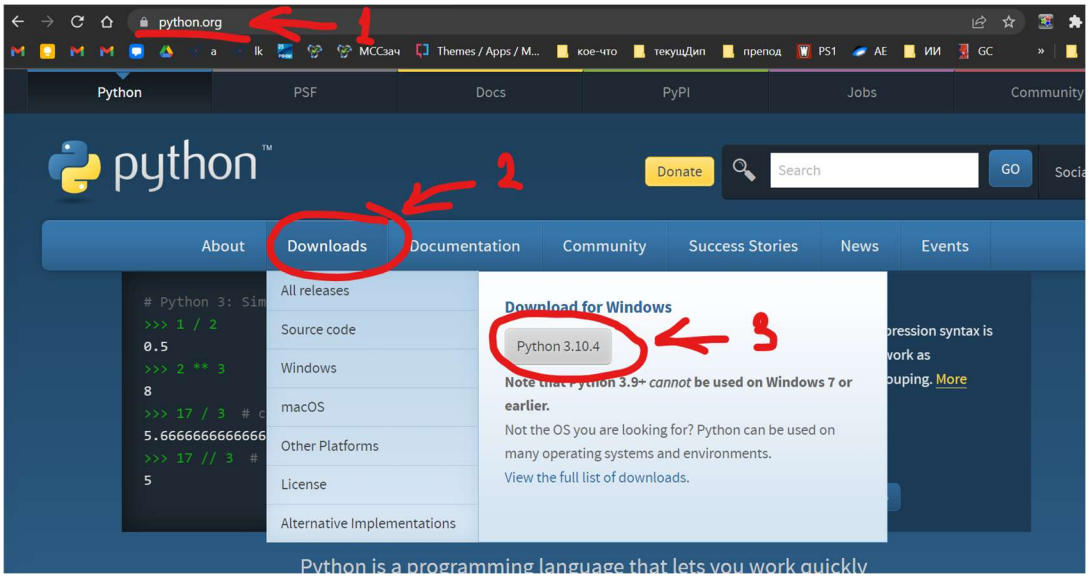
Вводная работа

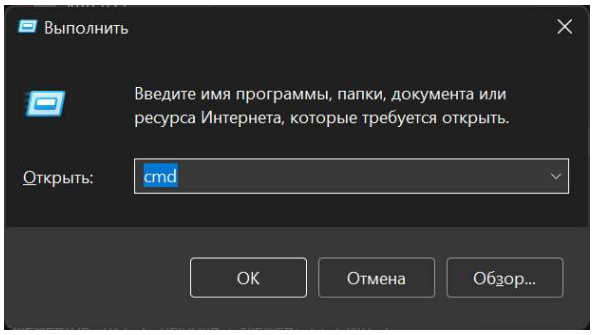
Подготовка к работе

Для начала следует загрузить последнюю версию python с официального сайта *python.org*.



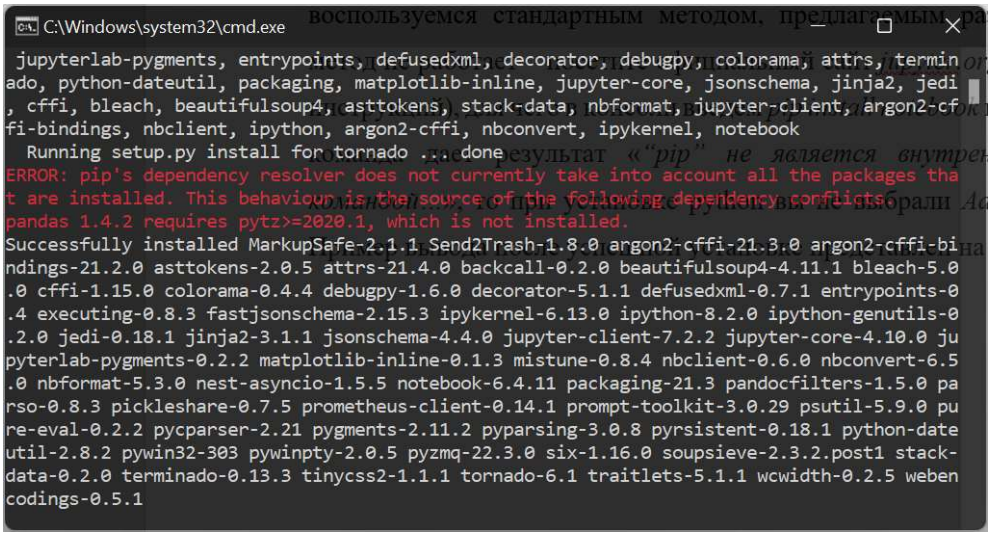
После загрузки запустите скачанный файл, выберите пункт *Add Python to PATH* (добавит возможность упрощенного вызова python и компонентов через консоль) и выберите *Install Now* (установит все необходимые компоненты, используйте *Customize* только если знаете, зачем вам менять набор устанавливаемых компонентов). По окончании закройте установщик.

Теперь запустите консоль Windows (командную строку). Самый простой способ это сделать – нажать на клавиатуре *Win+R*, ввести в поле *cmd* и нажать OK.

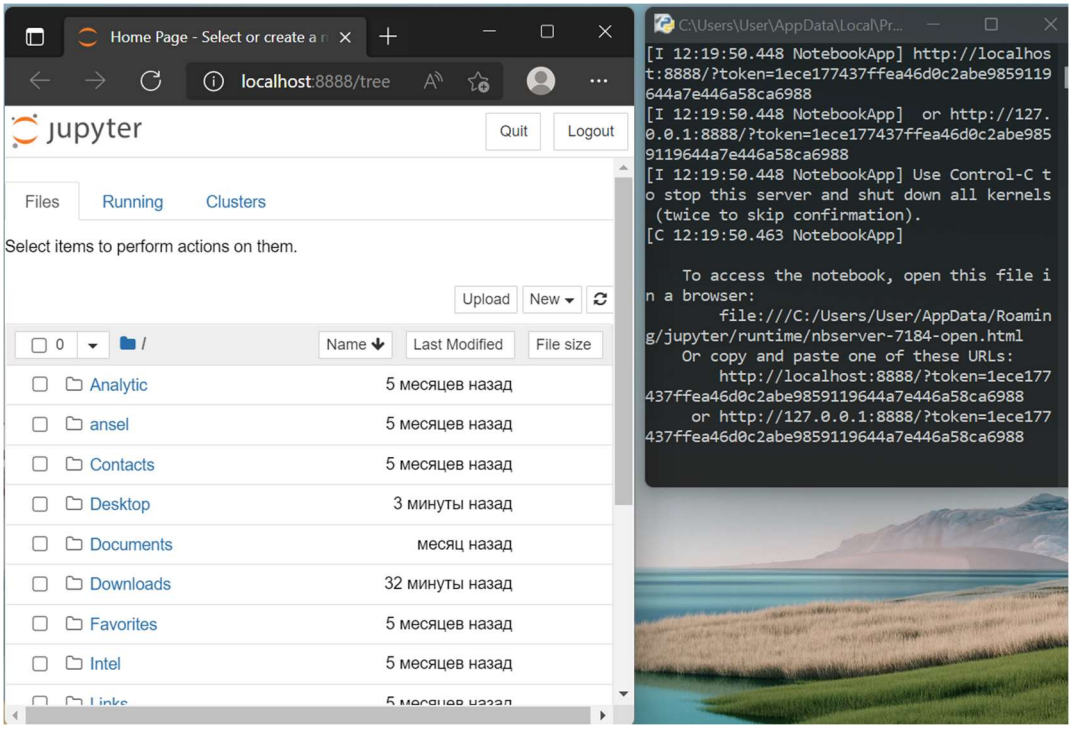


Теперь необходимо установить сам пакет *Jupyter Notebook*, для чего воспользуемся стандартным методом, предлагаемым разработчиком (если метод не работает – посетите официальный сайт *jupyter.org* для обновленных инструкций), для чего в консоли введем *pip install notebook* и жмем Enter[1]. Если команда дает результат «*“pip” не является внутренней или внешней командой…*», то при установке python вы не выбрали *Add Python to PATH.* Пример вывода после успешной установке представлен на скриншоте ниже. Наличие некоторых ошибок возможно (из-за конфликтов зависимостей), важно лишь чтобы работал установленный Jupyter Notebook.

[1] pip – менеджер пакетов python, работает аналогично менеджеру пакетов apt во многих дистрибутивах GNU/Linux. Менеджер пакетов – программа для установки программ, хранящихся на сервере разработчика или сообщества (репозиторий). Google Play и AppStore – менеджеры пакетов, имеющие графический интерфейс и прочие функции, у pip же графического интерфейса нет, поэтому установка производится с помощью консольных команд. Синтаксис команд для pip выглядит так: «pip действие имя-пакета», то есть можно установить/удалить пакет (библиотека python, программа на языке python из репозитория) введя вместо «действие» install/uninstall, также необходимо знать имя требуемого пакета.

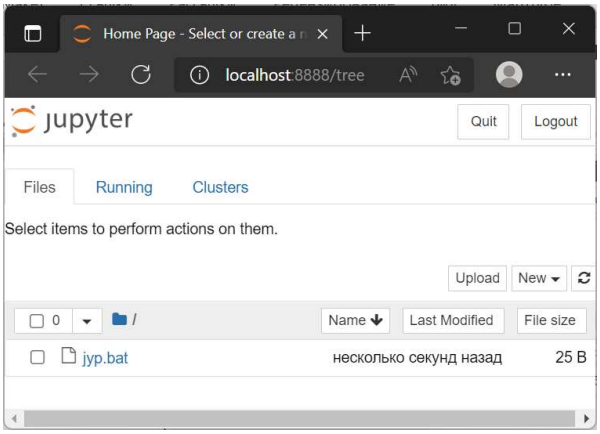


Запустите из консоль Jupyter Notebook посредством команды *jupyter notebook*. Откроется окно браузера, при этом консоль останется открытой – она необходима для работы программы, не закрывайте ее до конца работы с Jupyter Notebook (сворачивать можно). Также запустить программу можно из окна запуска (Win+R) той же командой *jupyter notebook*.



По умолчанию программа открывается в месте запуска консоли – папке пользователя (или system32, если консоль запущена с правами администратора). Чтобы запускать программу из другой папки, создайте в папке, где будете хранить документы Jupyter текстовый документ, откройте его, впишите в нем строку «*cmd /c " jupyter notebook"*». После сохраните,

закройте, и смените расширение файла с .txt на .bat (если расширения файла не видно – найдите в интернете инструкция, как включить его в проводнике).



Для Linux установка выполняется через менеджер пакетов, запуск Jupyter Notebook осуществляется через консоль.

Основы python и Jupyter Notebook

Главное отличие Jupyter Notebook от обычных IDE для python – работа в документах, разделенных на ячейки (Cells). Если в обычной IDE после написания кода в документе он исполняется целиком, то в Jupyter Notebook ячейки исполняются независимо, тем самым можно исполнять только те части кода, которые вам нужны в данный момент. Такая концепция построения документа с кодом названа REPL (read–eval–print loop – цикл чтения-вычисления-вывода), она удобна при математических вычислениях из-за чего используется, например, в системе компьютерной алгебры Wolfram Mathematica.

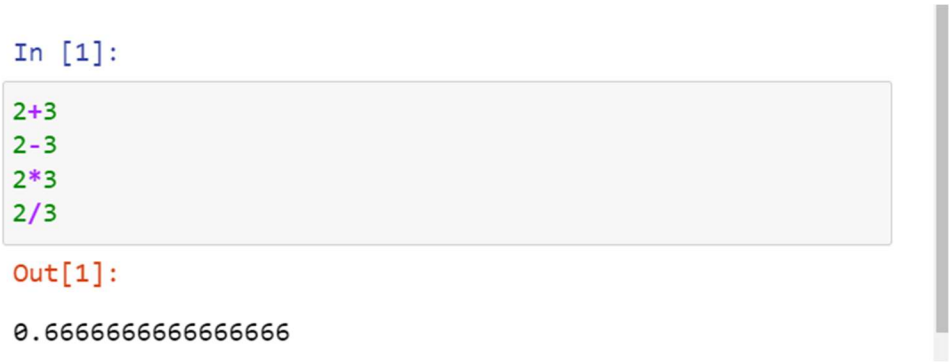
Откройте Jupyter Notebook и создайте новый файл, для чего в правом углу над списком файлов выберите New – Python 3. Откроется новая вкладка с одной ячейкой In (для ввода). Попробуйте ввести в ней несколько простейших алгебраических операций. В python строки кода не требуют фигурных скобок для ограничения функций и «;» для ограничения строки кода, поэтому в единственную ячейку просто введите следующие строки:

```
2+3
2-3
```

2*3

2/3

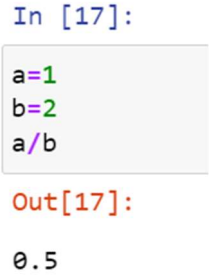
и исполните ячейку кнопкой Run сверху либо шорткатом (Shift+Enter или Ctrl+Enter).



Видим, что результат выводится только для 2/3, поскольку данная операция стоит последней, то есть *следующие друг за другом математические операции исполняются, но выводится только последняя из них*. Для вывода каждого из результатов необходимо воспользоваться стандартной функцией *print()*. Просто возьмите каждую заданную строку в скобки функции print() и повторите исполнение ячейки.

Переменным в python присваивается значение через «=». Зададим две переменных вместо прошлого кода в той же ячейке и исполним ее.

Видим, что результата (вывода) – нет, поскольку в ячейке только присваиваются значение, нет процедур с выводом. Если же добавить a/b в последнюю строку ячейки, то при исполнении результат появится.



Массивы в python задаются в квадратных скобках, элементы перечисляются через запятую. Зададим простой массив из трех элементов строчного типа (записываются в кавычках – либо в одинарных, либо в

двойных, **разницы нет**) и выведем их в цикле for. У этого цикла для нужной задачи будет следующая структура: «*for итератор in имя_массива:*», где итератор – и номер элемента в массиве, и ссылка на этот элемент. То есть python читает данную запись так «*для каждого элемента в массиве имя_массива исполни то, что записано после двоеточия*».

Двоеточие открывает внутреннее содержание (тело) какой-либо функции, то же самое, что в C++ и подобных языках делает «{». Если тело функции содержит лишь одну строку, то его можно записать в той же строке после двоеточия, иначе – с новой строки, *но обязательно с отступом от левого края (нажатием Tab на клавиатуре)*, все что на новых строках идет после двоеточия с отступом считается телом функции, после вызова которой стоит двоеточие.

```
In [3]:
words = ["я первое","а я второе","я последнее"]
for x in words:
    print(x)

я первое
а я второе
я последнее
```

В соответствии со сказанным выше, можно ту же процедуру реализовать и одной строкой.

```
In [6]:
words = ["я первое","а я второе","я последнее"]
for x in words: print(x)

я первое
а я второе
я последнее
```

Доступ к отдельным элементам заданного массива производится через «*имя_массива[индекс_в_массиве]*». Выведем первое (индекс массива начинается с нуля) значение заданного массива words в новой ячейке (для создания новой ячейки снизу нажать на панели сверху Insert – Insert Cell Below или нажать на ячейку вне поля ввода и нажать на клавиатуре латинскую B).

```
In [6]:
words = ["я первое","а я второе","я последнее"]
for x in words: print(x)

я первое
а я второе
я последнее
```

```
In [8]:
words[0]
Out[8]:
'я первое'
```

Функции в python задаются по конструкции «*def имя_функции(параметр1, параметр2, ...): тело функции*». Для примера ниже задана $f(x)=x^2+2$ и ее вызов для $x=0$ и $x=2$. Степень в python задается с помощью «**».

```
def f(x):
    return x**2+2

In [33]:
f(0)
Out[33]:
2

In [34]:
f(2)
Out[34]:
6
```

Перед дальнейшими действиями удалим ячейки с f(0) и f(2). Удалить ячейку можно из меню Edit-Delete Cells или нажать на ячейку вне поля ввода и двукратно нажать на клавиатуре D.

Функцию можно использовать для заполнения массивов. К примеру, зададим в массиве X значения от 1 до 10, а в массив Y запишем $f(X_i)$, то есть значение функции для каждого элемента X.

Для задания массива в виде диапазона значений в python используется *range(первый_элемент, последний_элемент, шаг)*. Если вызвать range без шага

– то шаг будет равен 1, если вызвать с одним аргументом – сформируется массив из значений от 1 до заданного в виде аргумента значения с шагом 1.

```
X=range(10)
```

Массив Y инициализируем как *Y=[]*, после чего в цикле for заполним его значениями $f(X_i)$, прикрепляя на каждой итерации $f(X_i)$ к массиву Y, для чего используется функция «*имя_массива.append(элемент)*», добавляющая в конец массива выбранный элемент.

Заполним указанным образом Y и выведем его значения. Так как в одной ячейке и объявление Y=[], и наполнение данного массива с помощью .append(), данный массив (при исполнении ячейки) будет сперва обнуляться, а затем заполняться, тем самым не нужно беспокоиться, что многократное исполнение ячейки прицепит к концу Y слишком много элементов.

```
In [13]:
```

```
def f(x):
    return x**2+2
X=range(10)
Y=[]
for x in X:
    Y.append(f(x))
```

```
In [14]:
```

```
Y
```

```
Out[14]:
```

```
[2, 3, 6, 11, 18, 27, 38, 51, 66, 83]
```

Как было сказано, в тело цикла можно поместить более одной операции. Зададим новый пустой массив *Y1* вне цикла и наполним его в том же цикле значениями f(x), деленными надвое.

```
Y1=[]
for x in X:
    Y.append(f(x))
    Y1.append(f(x)/2)
```

Для вывода обоих массивов воспользуемся упомянутой ранее print() для каждого массива.

```
[2, 3, 6, 11, 18, 27, 38, 51, 66, 83]
[1.0, 1.5, 3.0, 5.5, 9.0, 13.5, 19.0, 25.5, 33.0, 41.5]
```

Часто при программировании требуется использовать условие. В python оно задается по конструкции «*if условие: операция*». Аналогично for, у if оператор может располагаться в той же строке либо в строчке под if с отступом слева (TAB) относительно if. Jupyter Notebook расставит отступы за вас сразу после переноса строки после двоеточия.

Зададим в цикле условие для наполнения значениями больше 7 в массив Y, меньше или равные 7 – в массив Y1. Ниже слева скриншот задания условия, справа – результат вывода Y и Y1.

```
for x in X:
    if f(x)>7:
        Y.append(f(x))
    if f(x)<=7:
        Y1.append(f(x))
```

```
[11, 18, 27, 38, 51, 66, 83]
[2, 3, 6]
```

То же действие можно совершить с помощью только одного условного оператора if, для чего оператор, который требуется исполнить при невыполнении заданного условия, помещается после конструкции «*else:*». При этом результат (Y и Y1) не будут отличаться от рассмотренных выше.

```
for x in X:
    if f(x)>7:
        Y.append(f(x))
    else:
        Y1.append(f(x))
```

До конца 2021 года в Python не было аналога функции *switch* для условий с несколькими вариантами, поэтому в сообществе принято использовать конструкцию «*if-elif-else*». В этом случае внутри одного условного оператора if после каждого elif (сокращение от else if) можно задавать новое условие, конструкция аналогична последовательному исполнению множества операторов if.

Рассмотрим данную конструкцию на примере заполнения Y и Y1. Предположим, что требуется записать в Y все значения f(x) меньше 50, а также значение f(x), если оно равно 27, остальные же значения нужно записать в Y1. Для этого после if зададим условие «меньше 50» для заполнения Y, под ним

после elif «равно 27» для той же операции, и после else (для всех оставшихся значений) будем заполнять Y1. Сравнение (равенство) в условии на языке python задается «= =»

```
for x in X:
    if f(x)<50:
        Y.append(f(x))
    elif f(x)==27:
        Y.append(f(x))
    else:
        Y1.append(f(x))
```

```
[2, 3, 6, 11, 18, 27, 38]
[51, 66, 83]
```

Ту же процедуру можно выполнить без elif, если в if записать оба условия, для чего их необходимо взять в скобку и поставить между ними: *or* – если необходимо, чтобы выполнилось хотя бы одно условие (ИЛИ); *and* – чтобы выполнялись оба условия (И). Y1 и Y после данных изменений будут иметь те же значения, так как формально условие не поменялось.

```
for x in X:
    if (f(x)<50 or f(x)==27):
        Y.append(f(x))
    else:
        Y1.append(f(x))
```

Вернем исходное состояние цикла for для наполнения значениями f(x) массив Y, при этом объявление Y1 можно удалить. *На данном этапе замените f(x) на ту, что задается вашим вариантом в таблице ниже.*

Варианты функций для f(x)

1	**2**	**3**	**4**	**5**	**6**
x^3-2	x^2-7	x^3+2x	x^2+2x	$x^3-0,5x^2$	$0,1x^3+0,4x$
7	**8**	**9**	**10**	**11**	**12**
$0,1x-0,7x^3$	$-x^2+7$	$-x^3-0,4x$	$x^2-0,2x$	$x^3+0,5x^2$	$0,1x^3-0,4x$
13	**14**	**15**	**16**	**17**	**18**
$-x^3+2x^2$	$0,1x^2-7$	$0,6x^3+2,2x$	$-x^2+0,1x^3$	$0,3x^2-0,2x$	$0,7x+0,4x^2$

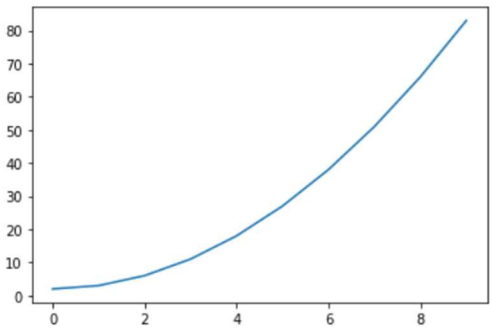
Теперь построим график Y(X), для чего потребуется подключить библиотеку matplotlib.pyplot (точнее библиотека matplotlib, но из нее мы берем

только класс pyplot, отвечающий за графики). Для этого создадим ячейку в начале документа (импорт библиотек принято производить в начале кода), в которой импортируем нужную библиотеку. Импорт в python осуществляется по конструкции «*import имя_библиотеки as краткое_название*», где краткого названия может (и as перед ним) и не быть, но с ним куда удобнее обращаться к классам и функциям библиотеки с длинным названием, как в данном случае.

```
In [16]:
import matplotlib.pyplot as plt
```

Не забываем исполнить ячейку с библиотекой для осуществления импорта и задаем параметры для построения графика Y(X). При кратком названии библиотеки plt для построения графика используется конструкция «*plt.plot(массив_для_x, массив_для_y)*».

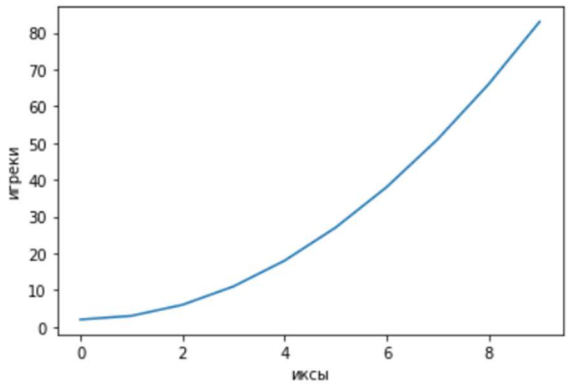
```
In [17]:
import matplotlib.pyplot as plt
plt.plot(X,Y)
Out[17]:
[<matplotlib.lines.Line2D at 0x25377babc40
>]
```



Чтобы задать обозначения осей нужно воспользоваться функциями *.ylabel()* и *.xlabel()* библиотеки, в скобках которых соответственно вписываем название осей в апострофах. Чтобы отображался только график, без текстовой информации о данных над графиком, также следует добавить в конец ячейки *plt.show()* (в консольной версии python эта команда выводит график, в Jupyter же график, как запрошенные к выводу данные, выводится и без данной команды).

In [30]:

```
import matplotlib.pyplot as plt
plt.plot(X,Y)
plt.xlabel('иксы')
plt.ylabel('игреки')
plt.show()
```



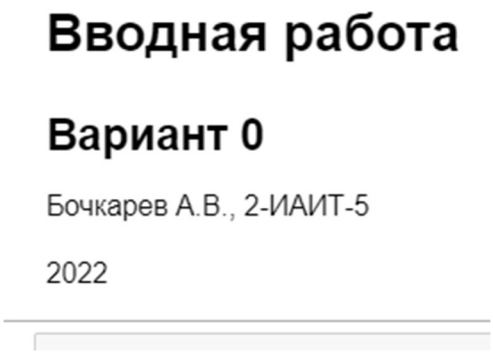
Оформление документа Jupyter и подготовка отчета

Тип ячеек в документе можно менять с исполняемого кода (*Code*) на простой текст (*Markdown*). Это дает возможность оформлять отчет по работе из документа Jupyter без переноса материалов в текстовый редактор, а также удобно для добавления комментариев, деления документа на озаглавленные блоки и т.п.

В документе по вводной работе создайте вверху ячейку, над импортом библиотек. Для смены типа ячейки нужно либо выбрать тип в выпадающем списке меню (между иконками перемотки и клавиатуры), либо щелкнуть по ячейке вне поля ввода и нажать на клавиатуре *M* (для смены на текст) или *Y* (для смены на код). Смените тип ячейки на текст. Неформатированный формат ячейки (Raw NBConvert) использоваться не будет, а заголовочный тип (Heading) заменен на текстовый с указанием уровня заголовка. В Jupyter доступно 6 уровней заголовков, все они отличаются размером (с увеличением уровня уменьшается размер), а последние два уровня имеют начертание курсивом.

Для задания заголовка в текстовой ячейке используется символ «#» в начале строки, причем количество идущих подряд таких символов задает уровень заголовка, а между текстом заголовка и последним таким символом

должен стоять пробел. Задайте первой строкой название работы (заголовок уровня 1), второй – слово Вариант и номер варианта (уровень 2), третьей – ваше ФИО, курс-факультет-группу (простым текстом), четвертой – год выполнения. Обратите внимание, что перенос (Enter) между двумя строками текста не отображается после исполнения текстовой ячейки (две строки слипаются в одну, но после заголовка любого уровня перенос ставится автоматически), для отображения переноса требуется при редактировании текста делать два переноса между строками или добавить в конце строки HTML-тег переноса <*br*>. Чтобы отобразить текст после редактирования, ячейку с ним требуется исполнить. Для редактирования текстовой ячейки дважды щелкните по ней.



Чтобы сохранить оформленный отчет как PDF, нужно либо воспользоваться конвертером (File - Download as – PDF via LaTeX или PDF via HTML), который загружается отдельно, либо открыть режим печати (на клавиатуре Ctrl+P), в котором возможно сохранить файл как PDF или XPS (требуется выбрать такую опцию в поле принтеров).

Работа 1. Аппроксимация данных полиномами в Jupyter Notebook

В настоящей работе рассматривается аппроксимация некоторыми встроенными в библиотеки python методами. Перед началом работы создайте шапку отчета, как указано во вводной работе, в разделе оформления.

Зададим функцию *f(x)* согласно варианту ниже.

Варианты функций для f(x)

1	**2**	**3**	**4**	**5**	**6**
x^3-2	x^2-7	x^3+2x	x^2+2x	$x^3-0,5x^2$	$0,1x^3+0,4x$
7	**8**	**9**	**10**	**11**	**12**
$0,1x-0,7x^3$	$-x^2+7$	$-x^3-0,4x$	$x^2-0,2x$	$x^3+0,5x^2$	$0,1x^3-0,4x$
13	**14**	**15**	**16**	**17**	**18**
$-x^3+2x^2$	$0,1x^2-7$	$0,6x^3+2,2x$	$-x^2+0,1x^3$	$0,3x^2-0,2x$	$0,7x+0,4x^2$

Сформируем, аналогично вводной работе, массив *X* с значениями от 0 до $N\cdot 100$ с шагом $N\cdot 2$, где N – номер варианта (*в данном примере используется та же функция, что и во вводной работе, а N=75*). В массив *Y* запишем значения f(x) для всех элементов массива X с наложенной помехой.

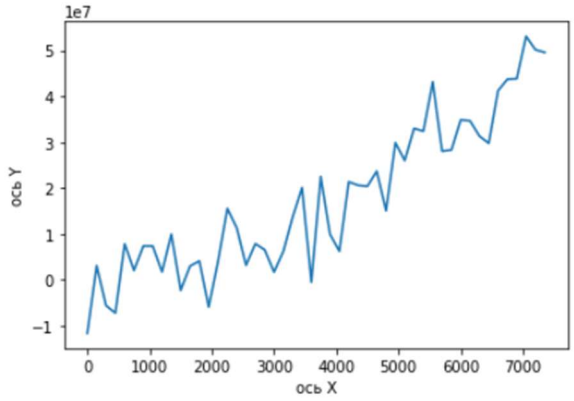
Для задания помехи создадим переменную *noise* с вызовом метода случайной величины из библиотеки *NumPy*, которую требуется импортировать: `import numpy as np`. Из np помеха формируется следующим образом: «*np.random.тип_распределения(среднее, СКО, число_элементов)*». В данной работе тип распределения задан вариантом, среднее равно 0, СКО равно 0.1 от максимума f(x) по модулю, число элементов равно длине массива X. Для вычисления максимума массива используется функция *max()*, чтобы получить модуль массива – *np.absolute()*, таким образом, СКО для noise можно записать как «*0.1*f(max(np.absolute()))*».

Чтобы сложить каждое значение f(x) со значением помехи при наполнении массива Y добавим перед циклом for наполнения массива

переменную «*c=0*», а в теле массива к f(x) внутри append() прибавим *noise[c]*, не забыв при этом в теле цикла в конце увеличить данный индекс на 1.

```
c=0
for x in X:
    Y.append(f(x)+noise[c])
    c+=1
```

Построим график данной функции, как и в вводной работе (импортировав `import matplotlib.pyplot as plt`), заменив обозначения осей на те, что видите на рисунке ниже.



Перейдем непосредственно к аппроксимации. Наложенная помеха, как видим по графику, существенна, что зачастую соответствует реально собранным данным. Для аппроксимации прямой в пакете *sklearn* содержится класс *linear_model* и интересующий нас метод *LinearRegression*, которую необходимо импортировать: `from sklearn.linear_model import LinearRegression as lappr`. Если импорт не удается – установите библиотеку с помощью команды *pip install sklearn.* Здесь производится импорт только одной функции LinearRegression за счет конструкции «*from библиотека.класс import метод as имя_в_документе*».

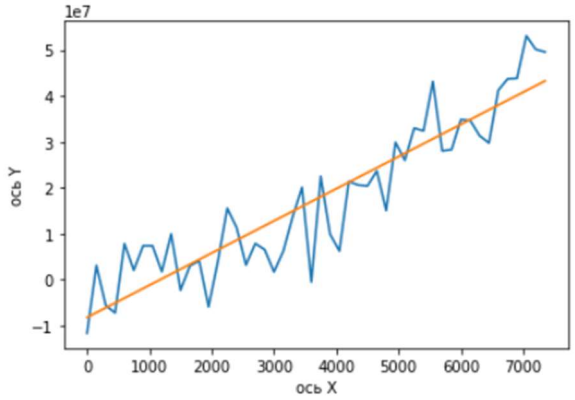
Данная функция принимает в качестве Y – столбец, в качестве X – строку, в данной же работе оба массива заданы как столбцы. Чтобы преобразовать X из строки в столбец (транспонировать, повернуть) используется метод *reshape((-1,1))* из np, но для его применения требуется преобразовать X в *np.array* (сейчас и X и Y по типу данных – list). Чтобы типы данных, подаваемых в *lappr* совпадали, также преобразуем в np.array и Y. В python в большинстве случаев для преобразования между типами данных

используется конструкция «*требуемый_тип(переменная_др_типа)*». Тем самым, для формирования np.array из X и Y (обозначим их в новом типе данных как *X1* и *Y1*) воспользуемся выражениями на скриншоте ниже (тут же сразу при преобразовании происходит «поворот» X).

```
X1=np.array(X).reshape((-1, 1))
Y1=np.array(Y)
```

Аппроксимируем прямой полученные значения, для чего зададим переменную с вызовом функции аппроксимации `model = lappr()` и ниже инициализируем аппроксимацию строкой вида `model.fit(X1, Y1)`. Остается построить графики – исходные данные и наложенный результат аппроксимации. Для графика аппроксимации используются те же значения X1 и аппроксимированные значения по оси ординат, которые вызываются из модели командой *model.predict(X1)*. В python графики накладываются друг на друга, если команды для них исполняются в одном коде, как здесь и происходит.

```
plt.plot(X1,Y1)
plt.plot(X1,model.predict(X1))
plt.xlabel('ось X')
plt.ylabel('ось Y')
plt.show()
```



Оценка точности модели производится коэффициентом детерминации (коэффициент «схожести» исходных и аппроксимированных значений, принимает значение 1 при полном совпадении аппроксимации и исходных данных). Вызывается данный коэффициент командой *model.score(X1, Y1)*.

```
print('Коэффициент детерминации:', model.score(X1, Y1))
```

```
Коэффициент детерминации: 0.8307089873739635
```

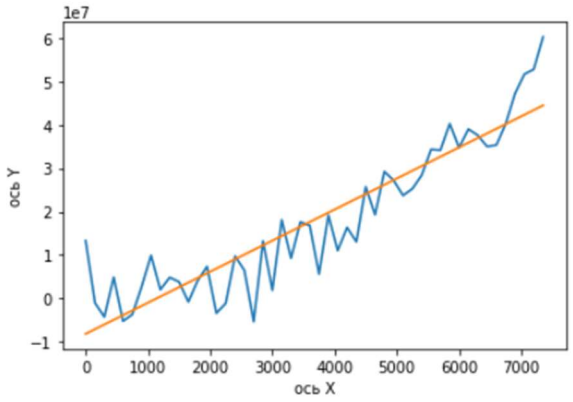
Также метод аппроксимации полиномом любой степени имеется в NumPy и именуется *polyfit.* Для его использования не требуется переворачивать X, но данный метод выдает коэффициенты полинома, а не сам массив. Также у метода нет функции оценки точности (коэффициента детерминации). Для получения коэффициентов полинома используется конструкция «*np.polyfit(X,Y,порядок_полинома)*», а для получения функции по коэффициентам полинома можно воспользоваться «*np.poly1d(коэффициенты_полинома)*». То есть следует подставить первую конструкцию во вторую для получения функции (выражение для полинома) в том же формате, что и f(x). При этом, для расчета значения этой функции в конкретной точке *x0* требуется в конце после функции дописать *(x0).*

Аппроксимируем той же прямой по данному методу Y. Для этого зададим в новой ячейке ниже *Y2=[]* и в цикле заполним данный массив значениями полинома 1 степени.

```
Y2=[]

for x in X:
    Y2.append(np.poly1d(np.polyfit(X,Y,1))(x))
```

Строим график Y2 от X вместе с графиком Y от X.



Визуальных отличий от результата lappr не наблюдается. Определим точность аппроксимации polyfit с помощью встроенного в sklearn метода, который импортируем через `from sklearn.metrics import r2_score as det` . Вызовем данный метод для сопоставления Y2 и Y.

```
]: det(Y,Y2)
]: 0.8299140331162999
```

Как видим, коэффициент детерминации практически равен полученному ранее.

Самостоятельное задание:

1) увеличивая порядок (от 2 до 15) аппроксимации polyfit добейтесь максимального значения det (наивысшая точность), если точность не повышается после некоторого порядка – останавливайтесь;

2) задайте в новой ячейке снизу массив Y3=[] и наполните его в цикле без суммирования с помехой;

3) в той же ячейке задайте массив Y4=[] и наполните его в цикле аппроксимацией зависимости X от Y3 с помощью polyfit;

4) добейтесь, аналогично пункту 1, максимальной точности det(Y3,Y4), постройте график (функции для вывода коэффициента det и графика можете задать в той же ячейке или создать новые);

5) создайте в конце документа ячейку, смените ее тип на текст и запишите в ней вывод, отвечающий на вопрос: **как влияет наложенная помеха на точность аппроксимации?**

Работа 2. Применение аппроксимации к датасету

В данной работе требуется аппроксимировать стандартные массивы данных. В качестве примера рассмотрим встроенные в библиотеку *seaborn* данные об измерениях параметров цветов, известные как «ирисы Фишера». Это один из популярных наборов данных (также именуют датасет или сет), содержащий четыре различных размера для трех различных сортов цветка ириса. Для начала импортируем библиотеку seaborn, содержащую данный сет.

```
In [1]: import seaborn as sns

---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Input In [1], in <cell line: 1>()
----> 1 import seaborn as sns

File ~\AppData\Roaming\Python\Python310\site-packages\seaborn\__init__.py:2, in <module>
      1 # Import seaborn objects
----> 2 from .rcmod import *  # noqa: F401,F403
      3 from .utils import *  # noqa: F401,F403
      4 from .palettes import *  # noqa: F401,F403
```

При попытке импорта возникает ошибка – библиотека не найдена. Очевидно, библиотека не была установлена, что требуется исправить. Как было сказано в начале пособия, для установки библиотек также можно использовать pip, причем не обязательно вызывать его в консоли – можно установить библиотеку прямо из документа Jupyter.

```
In [2]:
pip install seaborn

Requirement already satisfied: seaborn in
c:\users\user\appdata\roaming\python\python
310\site-packages (0.11.2)
Requirement already satisfied: numpy>=1.15
in c:\users\user\appdata\roaming\python\pyt
hon310\site-packages (from seaborn) (1.22.
3)
```

После установки попытка импорта не выдает ошибок. После установки строку с pip install seaborn можно удалить. Импортируем из библиотеки нужный нам сет в переменную *iris*.

```
In [3]: import seaborn as sns

In [ ]: iris = sns.load_dataset("iris")
```

Вывод данной переменной дает представление о содержимом сета. В нем 150 строк и 6 столбцов – 1) номер образца; 2) длина чашелистика (sepal length); 3) ширина чашелистика (sepal width); 4) длина лепестка (petal length); 5) ширина лепестка (petal width); 6) сорт ириса (setosa – щетинистый, virginica – виргинский, versicolor – разноцветный).

Out[5]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...

Для доступа к одному из столбцов в многомерном массиве с именованными столбцами достаточно вызвать конструкцию «*имя_массива.имя_столбца*». Выведем значения ширины чашелистика.

```
Out[8]: 0      3.5
        1      3.0
        2      3.2
        3      3.1
        4      3.6
              ...
        145    3.0
        146    2.5
        147    3.0
        148    3.4
        149    3.0
        Name: sepal_width, Length: 150, dtype: float64
```

Чтобы построить зависимость между двумя параметрами одного сорта ириса, допустим, между шириной чашелистика и длиной лепестка у Ирисов разноцветных, необходимо извлечь из массива данные, касающиеся только данного сорта. Стандартных функций у python для этого нет, поэтому воспользуемся средствами библиотеки *pandas*, импорт осуществим там же, где импортировали seaborn.

```
In [9]: import seaborn as sns
        import pandas as pd
```

В последней ячейке (если она не пуста, создайте снижу новую) зададим два пустых массив – X=[] для точек по оси абсцисс и Y=[] для точек по оси ординат. В рамках примера примем для значений по оси X ширину чашелистиков, а для Y – длину лепестков, причем оба массива значений возьмем только для одного сорта, разноцветного (*versicolor*). Чтобы извлечь из стандартного сета данных значения из некоторого *столбца* нужно задать «*переменная_с_датасетом['имя_столбца'].values*». Согласно выбранным для примера параметрам, в три переменные запишем: 1) типы цветов (*species*); 2) значения ширины чашелистиков (*sepal_width*); 3) значения длины лепестков (*petal_width*). Все скриншоты в работе используют данные параметры, но вам требуется использовать тот сорт и те значения для X и Y, что которые указаны в таблице ниже для вашего варианта.

Варианты параметров

Вар.	**Сорт**	**X**	**Y**	**Вар.**	**Сорт**	**X**	**Y**
1	setosa	sepal_length	sepal_width	**10**	setosa	sepal_width	petal_length
2	versicolor	sepal_width	sepal_length	**11**	versicolor	petal_length	petal_width
3	virginica	petal_length	sepal_width	**12**	virginica	petal_width	petal_length
4	setosa	petal_width	sepal_width	**13**	setosa	sepal_length	petal_length
5	versicolor	sepal_length	sepal_width	**14**	versicolor	sepal_width	petal_width
6	virginica	sepal_width	sepal_length	**15**	virginica	petal_length	petal_width
7	setosa	petal_length	sepal_length	**16**	setosa	petal_width	sepal_length
8	versicolor	petal_width	sepal_length	**17**	versicolor	sepal_length	petal_width
9	virginica	sepal_length	petal_length	**18**	virginica	sepal_width	petal_length

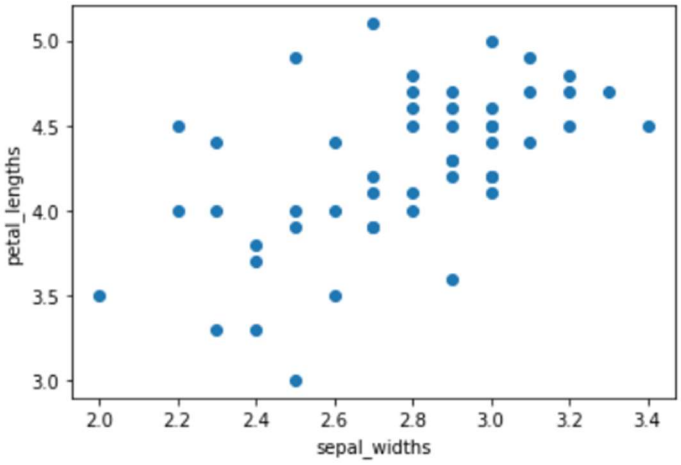
Примечание: здесь, для примера, названия столбцов указываются в одинарных кавычках, так как python допускает записать строковые данные и в одинарных, и в двойных кавычках.

```
In [90]: X=[]
         Y=[]
         types=iris['species'].values
         sepal_widths=iris['sepal_width'].values
         petal_lengths=iris['petal_length'].values
```

В той же ячейке в цикле запишем в X значения ширин чашелистиков, а в Y – значения длин лепестков. Для этого итератор (который опять обозначен в пособии *c*) будем менять в диапазоне от 1 до длины одного из трех заданных выше массивов с данными из датасета. Все они имеют один размер, поэтому не имеет значения, какой массив берется для итератора, в примере ниже используется длина *types*. В python длину массива (одномерного, то есть столбца или строки значений) можно определить командой «*len(массив)*».

Чтобы выбрать только относящиеся к разноцветному ирису значения длины и ширины, в тело цикла добавим условие if (с итератором, принимающим значения от 1 до длины массива types), внутри которого, при равенстве текущего типа цветка разноцветному, будут записываться в X и Y текущие значения требуемых параметров. В данном случае необходимо исполнить две операции (запись в X и Y), для этого нужно разместить обе операции с отступом под строкой с if.

Построим график извлеченных данных. Для этого сперва импортируем класс графопостроителя *matplotlib.pyplot*, после чего вернемся в последнюю ячейку и построим график Y от X с указанием заголовков по осям (в соответствии с именем параметров для этих осей). Так как данный график – набор разбросанных случайным образом точек, следует использовать метод *plt.scatter()* для его построения вместо рассмотренного ранее *plt.plot()*, который соединяет точки линией.



Самостоятельное задание:

1) аппроксимируйте с помощью polyfit заданный вариантом датасет, для чего увеличивая порядок (от 1 до 15) аппроксимации polyfit добейтесь максимального значения коэффициента детерминации (наивысшая точность), если точность не повышается после некоторого порядка – останавливайтесь;

2) постройте график результата аппроксимации, наложенный на точки датасета.

Работа 3. Аппроксимация логарифмической и экспоненциальной функцией с помощью polyfit

Рассмотренный метод polyfit позволяет аппроксимировать только полиномы, но подставляя в него преобразованные данные, возможно аппроксимировать и функции другого вида.

К примеру, **если вместо** массива **X** в него (при первом порядке полинома) **подставить натуральный логарифм** всех значений **X (*np.log(X)*),** то коэффициенты, выданные polyfit, будут относиться к функции

$$y = a_0 + a_1 \ln(x),$$

что является **логарифмической функцией**. **Если же** (также при первом порядке) **подставить в него вместо Y логарифм значений Y** – получим аппроксимацию для функции вида $\ln(y) = a_0 + a_1 x$ или, выразив y:

$$y = e^{a_0 + a_1 x} = \exp(a_0 + a_1 x),$$

то есть **экспоненциальную функцию**. Таким образом, получаемые при аппроксимации polyfit первого порядка коэффициенты могут относиться к функциям, не являющимся полиномами.

Единственной трудностью в аппроксимации является формирование массива аппроксимированных точек, так как использованный в прошлых работах np.poly1d() для восстановления полинома по коэффициентам аппроксимации, поддерживает только полиномы, то есть функции вида:

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n,$$

которые не позволяют восстанавливать экспоненциальную или логарифмическую функцию.

Для решения этой задачи создадим свои функции, начнем с логарифмической. Назовем ее logfit, в качестве аргументов она будет принимать два массива точек для аппроксимации и текущее значение x, для которого требуется вычислить логарифмическую функцию. В первой строчке в теле функции в произвольно названную переменную запишем вызов polyfit

для входных массивов, причем вместо X берем его логарифм. Эта переменная при аппроксимации станет массивом из двух чисел: первое – это коэффициент a_1, второе – коэффициент a_0 (polyfit выдает коэффициенты в обратном порядке). Во второй строке возвращаем значение требуемой функции, используя первый и второй элемент массива из первой строчки и np.log().

Аналогично задаем expfit для аппроксимации экспоненциальной функции, сменив возвращаемую функцию. Экспонента берется из библиотеки numpy с помощью *np.exp()*.

Самостоятельное задание:

1) задайте две функции – f1(x) – логарифмическую, f2(x) – экспоненциальную, коэффициенты a_1 и a_0 выбирайте по варианту – для f1(x) $a_0 = N,\ a_1 = N/10$, для f2(x) $a_0 = N/10,\ a_1 = N/10000$;

2) аппроксимируйте каждую из функций с помощью logfit и expfit, постройте два графика результатов аппроксимации вместе с исходными значениями, выведите коэффициенты детерминации для каждой аппроксимации.