



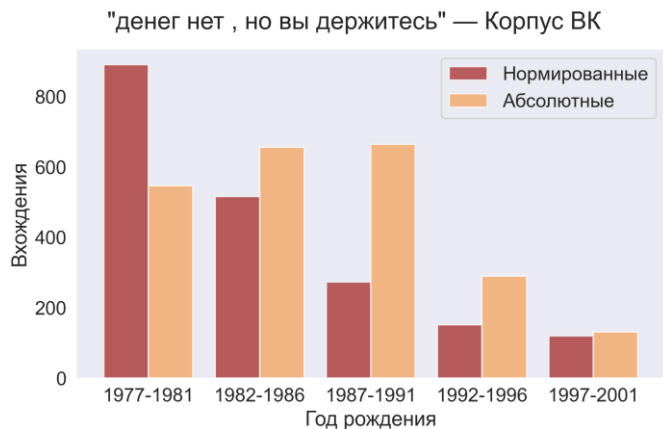
Программирование В лингвистике

Matplotlib, Seaborn, ipywidgets

Matplotlib

Matplotlib — популярная Python-библиотека для визуализации данных. Она используется для создания **любых видов графиков**: линейных, круговых диаграмм, гистограмм и других — в зависимости от задач.

Это огромная библиотека, и изучать ее возможности можно неделями, но для нас главное — разобраться в основных ее понятиях.



<- График на основе выдачи из корпуса ГИКРЯ с использованием возможностей **pandas, matplotlib и seaborn**

Структура Matplotlib

Matplotlib имеет три основных слоя: бэкенд-слой (backend), слой рисунков (artist) и слой скриптов (script).

Бэкенд-слой содержит три интерфейсных класса: **холст** (figure canvas), определяющий область рисунка, **прорисовщик** (renderer), умеющий рисовать на этом холсте, и **событие** (event), обрабатывающее ввод пользователя вроде щелчков мыши.

Слой рисунков создает изображения с помощью **прорисовщика** на холсте. Все, что находится на рисунке Matplotlib, является экземпляром **слоя рисунка** (artist). Засечки, заголовок, метки – все это индивидуальные объекты слоя рисунков.

Слой скриптов — это, по сути, интерфейс **matplotlib.pyplot**, который автоматизирует процесс определения холста, определения экземпляра рисунка и их соединения.

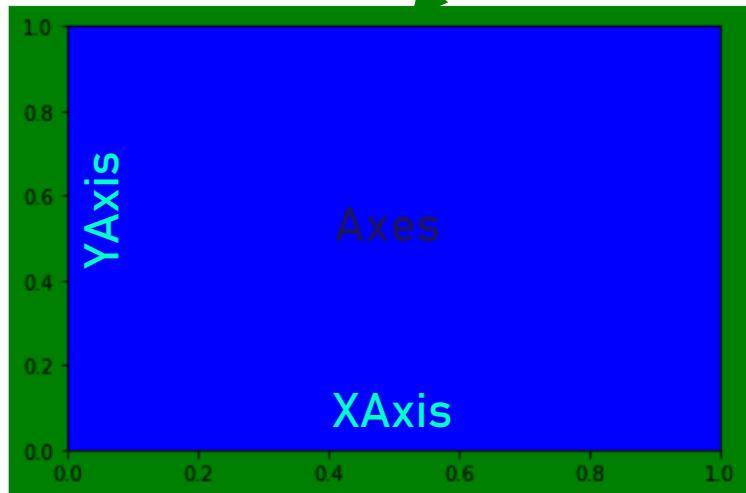
По сути, мы будем использовать лишь **слой скриптов**, а тот, в свою очередь, будет задействовать бэкенд и слой рисунков.

Figure и Axes

```
import matplotlib.pyplot as plt
```

```
fig = plt.figure()  
ax = fig.add_subplot(111)  
fig.set(facecolor='green')  
ax.set(facecolor='blue')  
plt.show()
```

Figure



Два объекта, о которых необходимо знать для работы с графиками – это **Figure** и **Axes**.

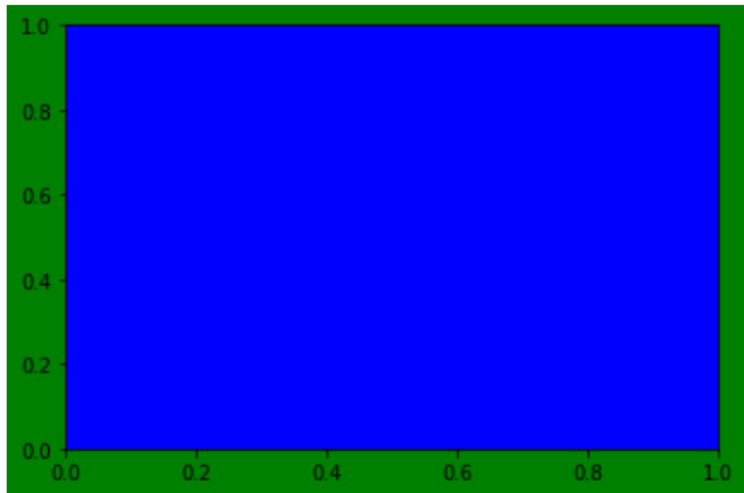
Рисунок (figure) – это та **область**, на которой **нарисованы графики**. Таких областей может быть несколько, каждая из которых может содержать несколько осей (axes).

Оси (Axes) – это, по сути, **сами графики**, а также все **вспомогательные элементы** (вроде сетки и подписей). Каждая область осей содержит ось абсцисс (XAxis) и ось ординат (YAxis), на которые мы можем прилепить подписи, деления и метки.

Подробнее

```
import matplotlib.pyplot as plt
```

```
fig = plt.figure()  
ax = fig.add_subplot(111)  
fig.set(facecolor='green')  
ax.set(facecolor='blue')  
plt.show()
```



> `import matplotlib.pyplot as plt` – стандартный способ импортировать слой скриптов из `matplotlib`.

> `fig = plt.figure()` – вызов конструктора экземпляра класса `Figure`. Может принимать много аргументов для настройки будущего рисунка.

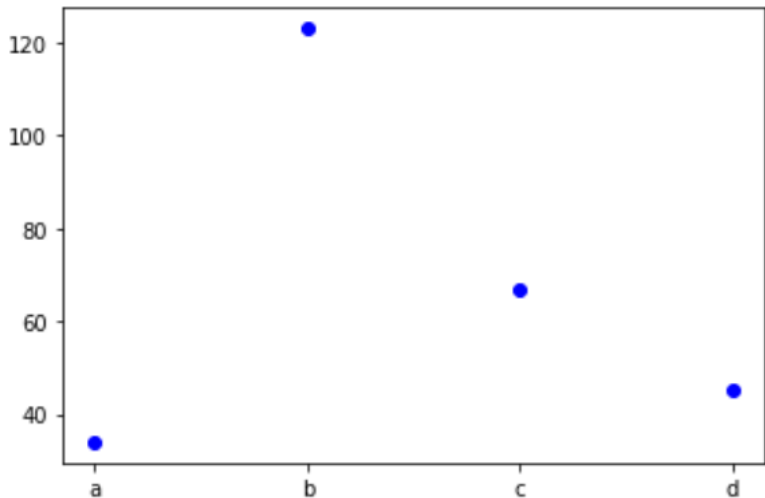
> `ax = fig.add_subplot(111)` – вызов метода класса `Figure`, который добавляет один экземпляр класса `Axes` на рисунок. `111` означает, что на нашем рисунке будет всего **один график**. Первая единица – сколько графиков по горизонтали, вторая – по вертикали, третья – номер графика на сетке.

> `fig.set(facecolor='green')` – метод настройки рисунка, меняет его цвет. У метода `set` есть много прочих параметров. Аналогично для `ax.set()`.

> `plt.show()` – функция, которая ищет последнюю активную фигуру и показывает ее.

Наш первый график

```
x = ['a', 'b', 'c', 'd']  
y = [34, 123, 67, 45]  
fig = plt.figure()  
ax = fig.add_subplot(111)  
ax.plot(x, y, 'bo')  
plt.show()
```



> **x** – список, который попадет в нашем графике в **ось абсцисс**.

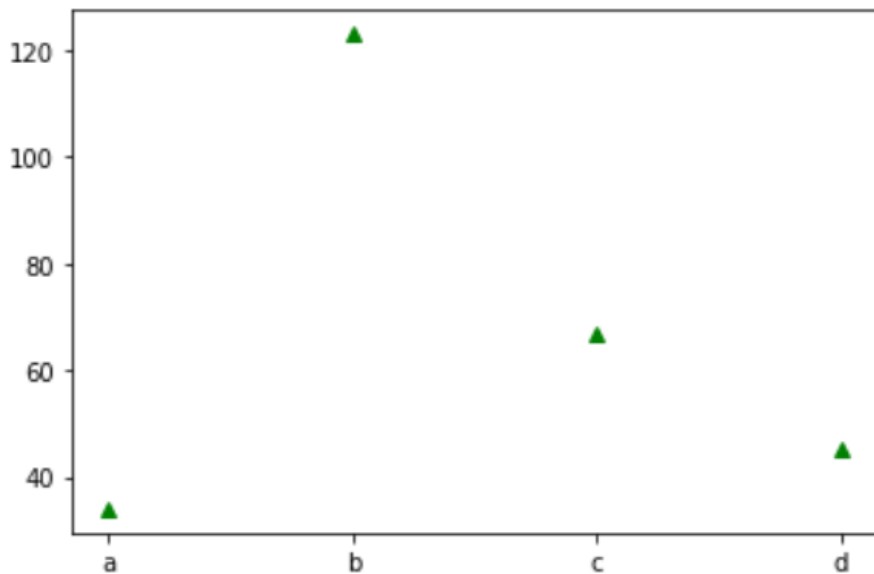
> **y** – список, который попадет в **ось ординат**.

> **ax.plot(x, y, 'bo')** – метод создания графика. Принимает ось абсцисс и ось ординат, а также кучу опциональных аргументов по настройке графика. **'bo'** – способ отображения графика, означает blue o, то есть **синие круглые точки**. Можно изменить стиль точек на графике, а также нарисовать непрерывный график (с линиями), используя этот параметр. Если его не указывать, то будет нарисован **непрерывный график**.

Подробнее про параметры метода plot можно найти [по этой ссылке](#).

plt.plot

```
plt.plot(x, y, '^g')  
plt.show()
```

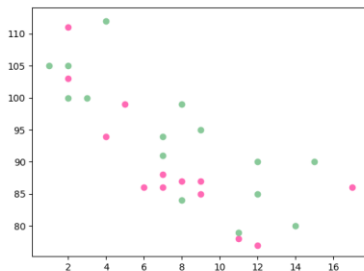


Самый быстрый способ нарисовать подобный график – просто использовать функцию `plot`, но тогда большинство настроек **не будет доступно**.

Графики matplotlib

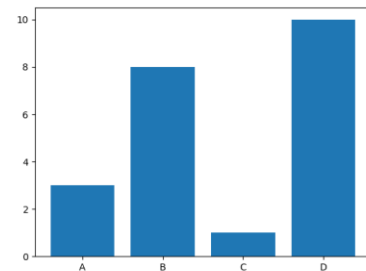
Точечный

`ax.scatter(x, y)`



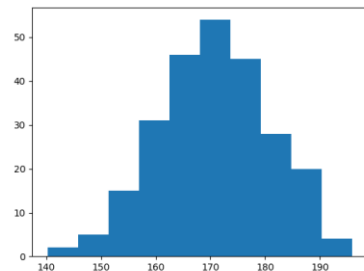
Столбиковый

`ax.bar(x, height(y), width, bottom, align)`



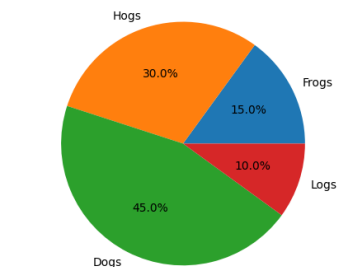
Гистограмма

`ax.hist(x, bins)`



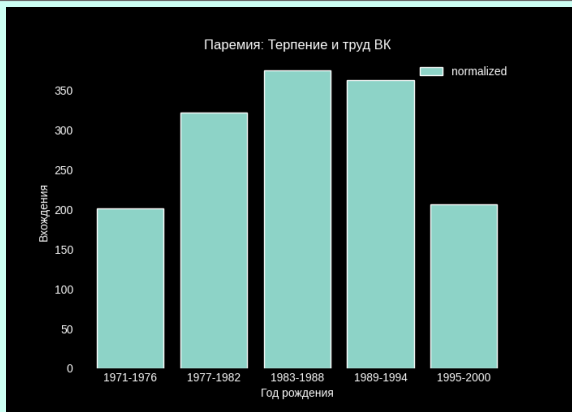
Пай-чарт

`ax.pie(sizes, labels, autopct)`



Возможности matplotlib

```
fig = plt.figure(figsize=(7, 5))
ax = fig.add_subplot(111)
ax.bar(final_data['year_cohorts'], final_data['normalized'])
ax.set_title(f'Паремиа: {paraemia} {corpus}')
ax.set_ylabel("Вхождения")
ax.legend('normalized')
plt.xlabel("Год рождения")
plt.style.use('dark_background')
plt.savefig(f'{paraemia}.png')
```



Базовые **настройки**, которые стоит знать:

У Figure есть size: `fig = plt.figure(figsize=(x, y))`

У ax можно установить заголовок: `ax.set_title('Title')`

У графиков можно менять **цвет, форму, линии, размер шрифтов**;

Можно устанавливать **xticks** (значения по оси x) и **yticks**;

Можно добавить легенду (каким цветом что обозначено): `ax.legend()`

Можно сохранять график с помощью `savefig(path)`;

Можно сразу устанавливать весь стиль для графика с помощью `plt.style.use(['...'])` (посмотреть, какие доступны, можно с помощью `plt.style.available`)

ax.bar vs. DataFrame.plot.bar

На данных примерах выполнен **один и тот же график** – первый с использованием метода bar в **Pandas**, второй – с использованием только возможностей **Matplotlib**.

```
def simple_bar(df):
    df.plot.bar(x="year_cohorts",
                y=['normalized', "result"],
                ylabel="Вхождения",
                alpha=0.75, rot=0, width=0.8,
                color=["brown", "sandybrown"])
    plt.title("Паремия: " + paraemia + " " + corpus)
    plt.xlabel("Год рождения")
```

Pandas.DataFrame.plot.bar

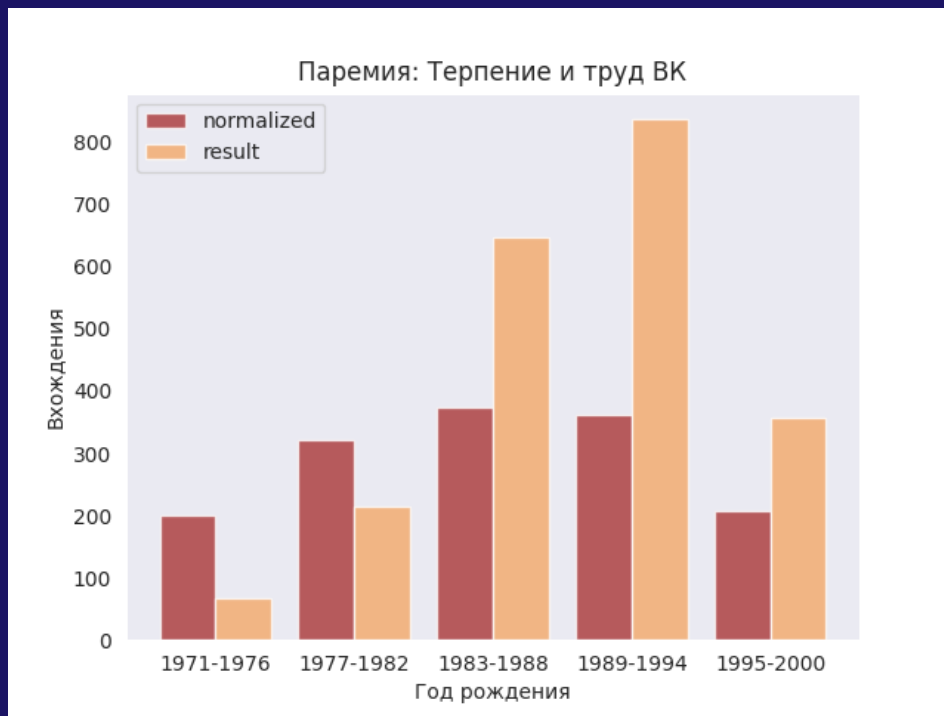
```
simple_bar(final_data)
plt.savefig(f'{paremia}.png')
```

Сохранение графика

```
def simple_bar(df):
    fig = plt.figure()
    ax = fig.add_subplot(111)
    X = df['year_cohorts']
    Xaxis = np.arange(len(X))
    ax.bar(Xaxis - 0.2, df['normalized'],
           alpha=0.75, width=0.4, color=["brown"])
    ax.bar(Xaxis + 0.2, df['result'],
           alpha=0.75, width=0.4, color=["sandybrown"])
    ax.set_ylabel("Вхождения")
    ax.set_xticks(Xaxis, X)
    ax.legend(['normalized', 'result'])
    plt.title("Паремия: " + paraemia + " " + corpus)
    plt.xlabel("Год рождения")
```

Matplotlib ax.bar

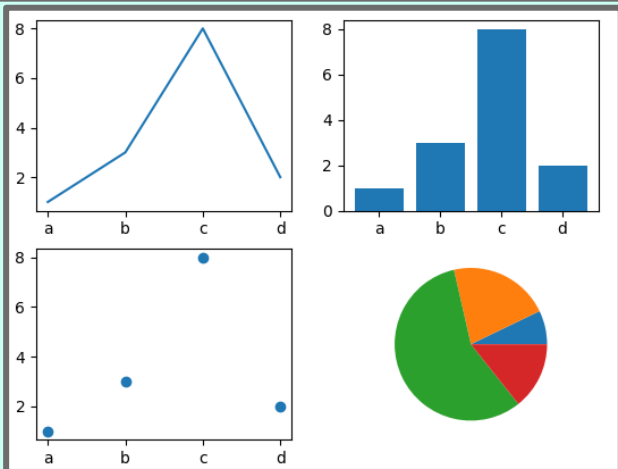
ax.bar vs. DataFrame.plot.bar



Результат

Подграфики

```
fig = plt.figure()  
ax_line = fig.add_subplot(2, 2, 1)  
ax_bar = fig.add_subplot(2, 2, 2)  
ax_scatter = fig.add_subplot(2, 2, 3)  
ax_pie = fig.add_subplot(2, 2, 4)
```



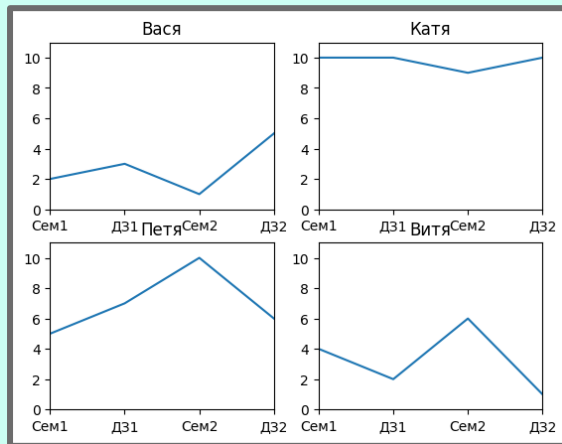
Как было сказано, можно поместить несколько графиков на одну фигуру. Первая цифра из трех – это **номер ряда**, вторая – **номер строки**, а третья – **индекс подграфика** (matplotlib воспринимает область графика как табличку).

* Наверху приведены лишь способы размещения графиков

Построение подграфиков в цикле

```
fig, axes = plt.subplots(2, 2)
for ax, col in zip(axes.flat, marks.columns):
    ax.set(xlim=(0, 3), ylim=(0, 11))
    ax.plot(marks.index, marks[col])
    ax.set_title(col)

plt.show()
```



> `fig, axes = plt.subplots(2, 2)` – распаковываем экземпляр класса `figure` и `axes` в соответствии с переданной сеткой (ряды, столбцы).

> `for ax, col in zip(...)` – итерируемся по двум массивам: по подграфикам и столбцам таблицы.

> `ax.set(xlim, ylim)` – устанавливаем минимальные и максимальные значения для осей `x` и `y`.

> `ax.plot(...)` – строим подграфик.

> `ax.set_title(col)` – называем наш подграфик.

Мы построили подграфики для **каждого студента** в зависимости от **заработанных им баллов** за семинары и домашние задания.

Трехмерные графики

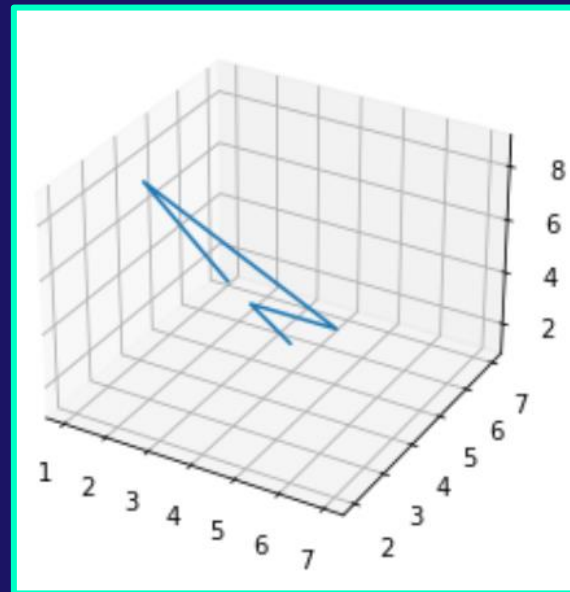
В matplotlib есть подбиблиотека, которая позволяет рисовать графики в трехмерном пространстве. Подробно рассматривать не будем, но самый простой пример кода выглядит так:

```
from mpl_toolkits.mplot3d import axes3d

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
X, Y, Z = [1, 2, 7, 3, 6], [7, 3, 2, 5, 2], [1, 9, 7, 3, 6]
ax.plot(X, Y, Z)
plt.show()
```

Для построения трехмерных графиков нужны **три оси** – ось абсцисс, ось ординат и ось аппликат.

Чтобы перейти в режим 3D, нужно лишь указать параметр `projection='3d'` в методе `fig.add_subplot` после импорта `axes3d`.



Seaborn

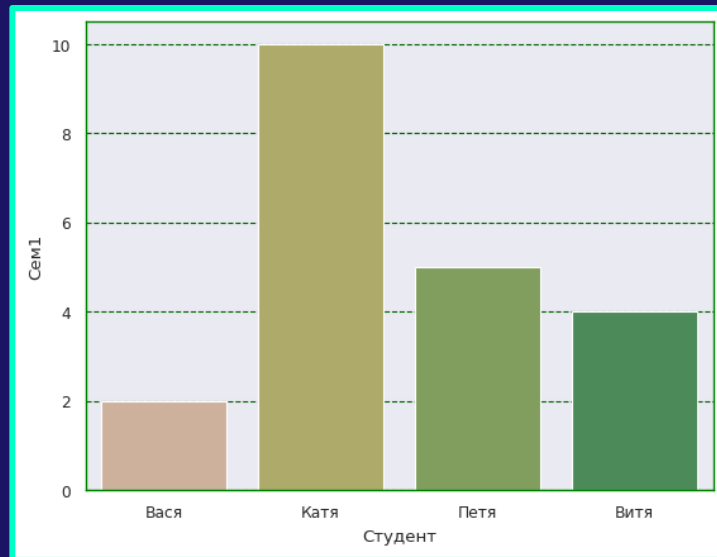
Seaborn – это библиотека, которая построена на базе matplotlib и очень хорошо с ней взаимодействует. Главная цель Seaborn – это **визуализация данных для машинного обучения**, поэтому seaborn хорошо работает с **Pandas.DataFrame**; также у seaborn очень красивые палитры.

Здесь приведен пример кода: можете ознакомиться с Seaborn [в этой документации](#).

```
import seaborn as sns

custom = {"axes.edgecolor": "green",
          "grid.linestyle": "dashed",
          "grid.color": "darkgreen"}
sns.set_style("darkgrid", rc = custom)
sns.set_context('paper')
sns.barplot(x='Студент', y="Сем1",
            data=df,
            palette=sns.color_palette('gist_earth_r'),
            hue='Студент')
```

Настройка графика



ipywidgets

ipywidgets – библиотека с виджетами для тетрадок.

Мы посмотрим лишь два из них, полный список виджетов можно посмотреть в [документации](#).

Test: 7

☐ Check me

Password:

Test:

Speed:

Slow

Regular

Pizza topping:

- ☒ pepperoni
- ☐ pineapple
- ☐ anchovies

✓Click me

Fruits

- Apples
- Oranges
- Pears

Click here

Text:

Number:

Time

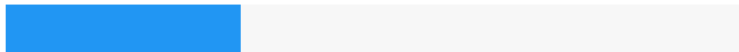
Progress bar



```
from ipywidgets import IntProgress
from IPython.display import display
import time
```

```
pb = IntProgress(min=0, max=100)
display(pb)
```

```
for i in range(100):
    pb.value += 1
    time.sleep(.1)
```



> `pb = IntProgress(min=0, max=100)` – создание экземпляра класса `IntProgress`. Необходимо задать диапазон, значения внутри которого мы будем увеличивать вручную.

> `display(pb)` – функция для отображения нашей строки прогресса.

> `for i in range(100)` – цикл, на каждой итерации которого наш `IntProgress` будет заполняться с помощью увеличения его атрибута.

> `pb.value += 1` – увеличение атрибута `IntProgress`.

> `time.sleep(.1)` – ‘зависание’ на 1 секунду. Благодаря нему мы будем наблюдать, как заполняется наш `IntProgress`.

Взаимодействие с графиком

```
from random import randrange
from ipywidgets import interact, IntSlider

def myfunc(x, n):
    x = [randrange(0, x + 1) for i in range(n)]
    y = [i ** 2 for i in x]
    sns.barplot(x=x, y=y, color='lime')
    plt.show()

interact(myfunc,
        x=IntSlider(min=1, max=30,
                    step=1, value=1),
        n=list(range(1, 11)))
```

> `x = [randrange(...)]` – получение списка случайных чисел в диапазоне от 0 до x в количестве n штук.

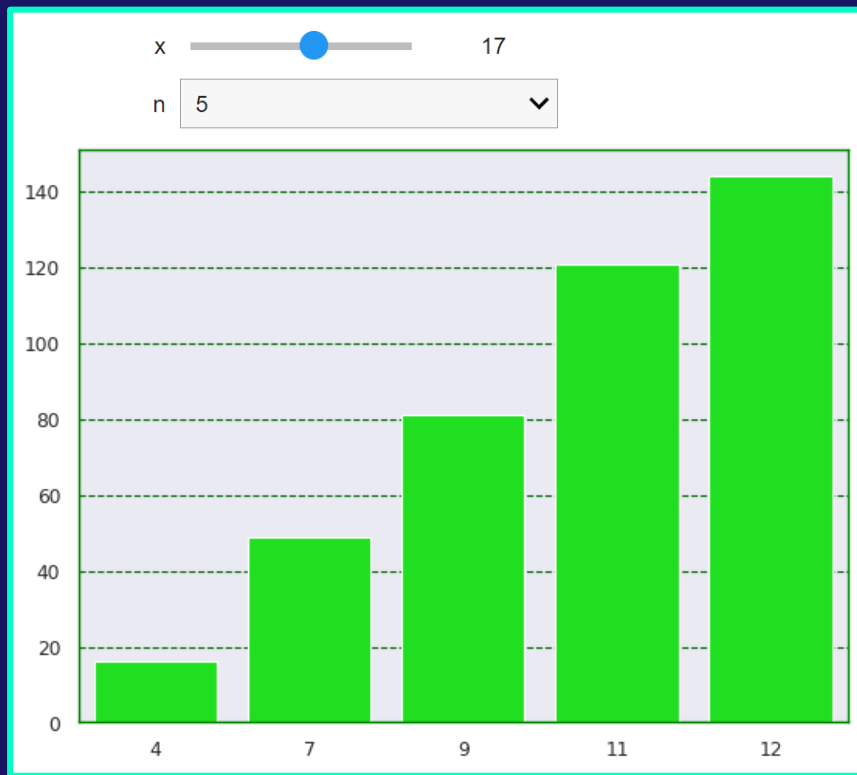
> `y = [i ** 2 for i in x]` – получение квадратов этих чисел.

> `sns.barplot(...)` – построение графика, где на оси абсцисс будут случайные числа, а на оси ординат – их квадраты.

> `plt.show()` – показ графика.

> `interact(...)` – функция для взаимодействия с графиком. Теперь x можно будет задать с помощью слайдера, а n – с помощью выпадающего списка.

Взаимодействие с графиком























Наш график будет перезагружаться **каждый раз**, когда мы будем **менять x и n** с помощью интерактивных виджетов `ipywidgets`.

Универсальные названия цветов

По [этой ссылке](#) Вы найдете все доступные ключевые слова-обозначения цветов для различных Python-библиотек.

Они помогут Вам в оформлении красивых графиков.

The following is the list of recognized color keywords that can be used as a keyword value for data type [<color>](#):

 aliceblue	rgb(240, 248, 255)	 lightpink	rgb(255, 182, 193)
 antiquewhite	rgb(250, 235, 215)	 lightsalmon	rgb(255, 160, 122)
 aqua	rgb(0, 255, 255)	 lightseagreen	rgb(32, 178, 170)
 aquamarine	rgb(127, 255, 212)	 lightskyblue	rgb(135, 206, 250)
 azure	rgb(240, 255, 255)	 lightslategray	rgb(119, 136, 153)
 beige	rgb(245, 245, 220)	 lightslategrey	rgb(119, 136, 153)
 bisque	rgb(255, 228, 196)	 lightsteelblue	rgb(176, 196, 222)
 black	rgb(0, 0, 0)	 lightyellow	rgb(255, 255, 224)
 blanchedalmond	rgb(255, 235, 205)	 lime	rgb(0, 255, 0)
 blue	rgb(0, 0, 255)	 limegreen	rgb(50, 205, 50)

...И т.д.