

# Тема: Построение графиков с использованием библиотеки `Pyplot / plt` 2

## Задание.

1. Внимательно изучите материал ниже.
2. Напишите на Python программу анализа графиков функций.
3. Пользователь может выбрать одну из нескольких функций (не менее 5ти).
4. Для каждого графика вводятся параметры графика (минимум 2 параметра, плюс параметры координат «окна» вывода функции – (X1, X2, Y1, Y2).
5. Например, параметры функции - масштабирующие коэффициенты, A,B:  $y=A*\sin(B*x)$ .
6. Тогда в окне формы указываются поля ввода коэффициентов A и B, а также координат X1, X2, Y1, Y2.
7. При отображении графиков искать все точки пересечения.
8. Сделать графики масштабируемыми.
9. Выводить корни уравнения  $F1(x)=F2(x)$  для выбранной пары графиков.

## Краткое описание `Pyplot / plt` 2

Модуль `pyplot` — это коллекция функций в стиле команд, которая позволяет использовать `matplotlib` почти так же, как `MATLAB`. Каждая функция `pyplot` работает с объектами `Figure` и позволяет их изменять. Например, есть функции для создания объекта `Figure`, для создания

области построения, представления линии, добавления метки и так далее.

pyplot является зависимым от состояния (stateful). Он отслеживает статус объекта Figure и его области построения. Функции выполняются на текущем объекте.

## Простой интерактивный график

Для знакомства с [библиотекой matplotlib](#) и с самим pyplot начнем создавать простой интерактивный график. В matplotlib эта операция выполняется очень просто. Достаточно трех строчек кода.

Но сначала нужно импортировать пакет pyplot и обозначить его как plt.

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

В Python конструкторы обычно не нужны. Все определяется неявно. Так, при импорте пакета уже создается экземпляр plt со всеми его графическими возможностями, который готов к работе. Нужно всего лишь использовать функцию plot() для передачи функций, по которым требуется построить график.

Поэтому достаточно передать значения, которые нужно представить в виде последовательности целых чисел.

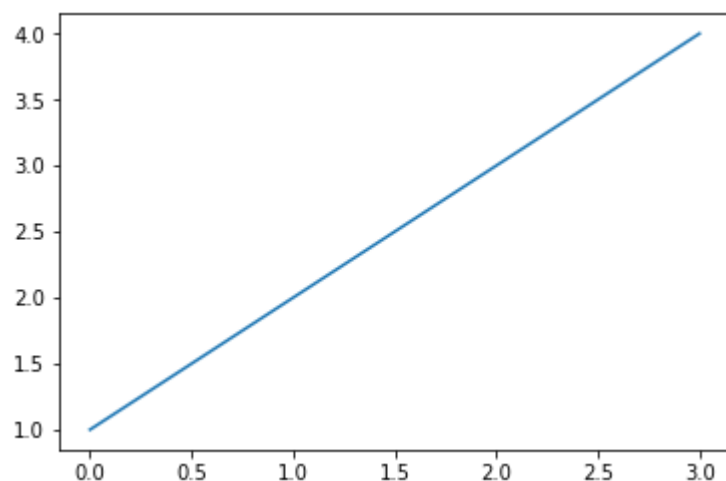
```
plt.plot([1,2,3,4])  
[<matplotlib.lines.Line2D at 0xa3eb438>]
```

В этом случае генерируется объект Line2D. Это линия, представляющая собой линейный тренд точек, нанесенных на график.

Теперь все настроено. Осталось лишь дать команду показать график с помощью функции `show()`.








```
plt.show()
```

Результат должен соответствовать показанному на изображении. Он будет отображаться в окне, которое называется `plotting window` с панелью инструментов. Прямо как в MATLAB.



## Окно графика

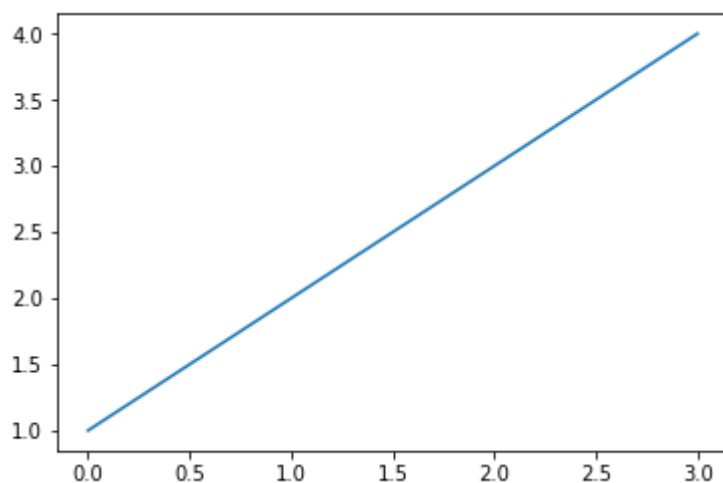
В этом окне есть панель управления, состоящая из нескольких кнопок.

-  Возвращает оригинальный вид
-  Следующий/предыдущий вид
-  Перемещать график левой кнопкой мыши,  
менять приближение — правой
-  Увеличить
-  Настроить подграфики
-  Сохранить/экспортировать объект
-  Отредактировать ось, кривую и  
параметры изображения

Код в консоли IPython передается в консоль Python в виде набора команд:

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.show()
```

Если же вы используете IPython QtConsole, то могли заметить, что после вызова `plot()` график сразу отображается без необходимости вызывать `show()`.



Если функции `plt.plot()` передать только список или массив чисел, `matplotlib` предположит, что это последовательность значений  $y$  на графике и свяжет ее с последовательностью натуральных чисел  $x$ : 0, 1, 2, 3, ....

Обычно график представляет собой пару значений  $(x, y)$ , поэтому, если нужно определить его правильно, требуется два массива: в первом будут значения для оси  $x$ , а втором — для  $y$ . Функция `plot()` принимает и третий аргумент, описывающий то, как нужно представить точку на графике.

## Свойства графика

На последнем изображении точки были представлены синей линией. Если не указывать явно, то график возьмет настройку функции `plt.plot()` по умолчанию:

- Размер осей соответствует диапазону введенных данных
- У осей нет ни меток, ни заголовков
- Легенды нет
- Соединяющая точки линия синяя

Для получения настоящего графика, где каждая пара значений  $(x, y)$  будет представлена в виде красной точки, нужно поменять это представление.

Если вы работаете в IPython, закройте окно, чтобы вернуться в консоль для введения новых команд. Затем нужно будет вызывать функцию `show()`, чтобы увидеть внесенные изменения.

```
plt.plot([1,2,3,4],[1,4,9,16], 'ro')  
plt.show()
```

Если же вы используете Jupyter QtConsole, то для каждой введенной команды будет появляться новый график.

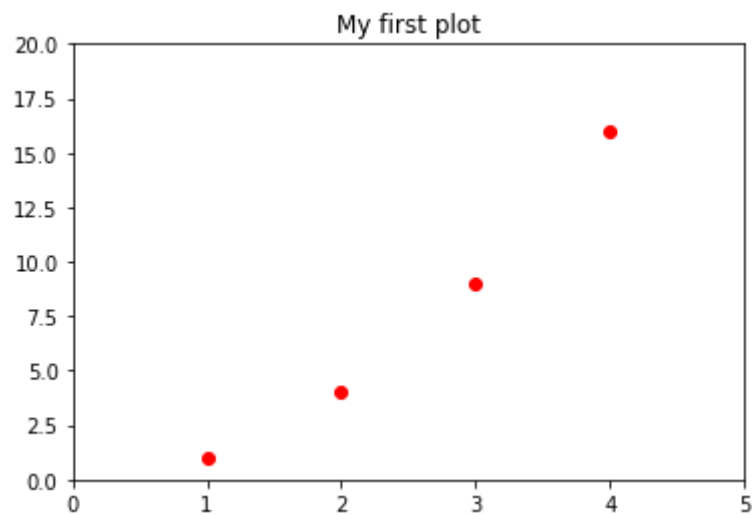
в будущем в примерах [средой разработки](#) будет выступать Google Colab.

Можно определить диапазон для осей x и y, задав значения в список `[xmin, xmax, ymin, ymax]` и передав его в качестве аргумента в функцию `axis()`.

Можно задать несколько свойств. Одно из них — заголовок, который задается через функцию `title()`.

```
plt.axis([0,5,0,20])  
plt.title('My first plot')  
plt.plot([1,2,3,4],[1,4,9,16], 'ro')  
plt.show()
```

На следующем изображении видно, как новые настройки делают график более читаемым. Так, конечные точки набора данных теперь распределены по графику, а не находятся на краях. А сверху есть заголовок.



## matplotlib и NumPy

Даже `matplotlib`, которая является полностью графической библиотекой, основана на `NumPy`. Вы видели на примерах, как передавать списки в качестве аргументов. Это нужно как для представления данных, так и для того, чтобы задавать границы осей. Внутри эти списки конвертируются в [массивы NumPy](#).

Таким образом можно прямо добавлять в качестве входящих данных массивы `NumPy`. Массив данных, обработанный `pandas`, может быть использован `matplotlib` без дальнейшей обработки.

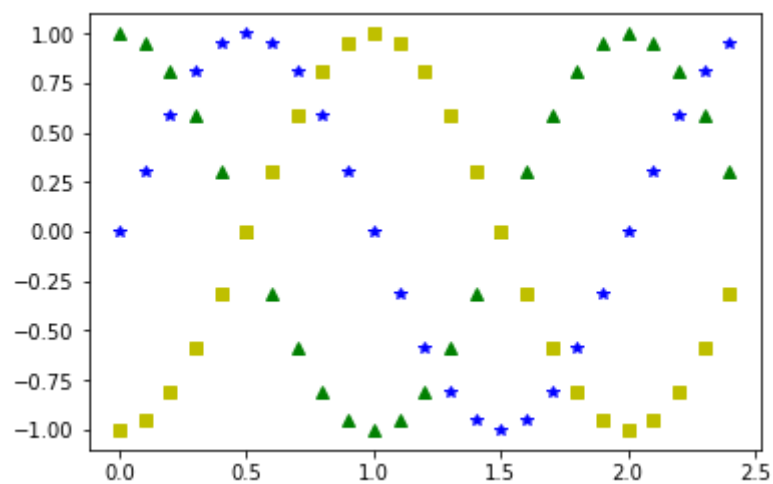
В качестве примера рассмотрим, как перенести на один график три тренда. Возьмем функцию `sin` из модуля `math`. Последний сперва нужно импортировать. Для генерации точек по синусоиде нужно [использовать библиотеку NumPy](#). Сгенерируем набор точек по оси `x` с помощью функции `arange()`, а для оси `y` воспользуемся функцией `map()`. С ее помощью применим `sin()` ко всем элементам массива (без цикла `for`).

```

import math
import numpy as np

t = np.arange(0, 2.5, 0.1)
y1 = np.sin(math.pi*t)
y2 = np.sin(math.pi*t+math.pi/2)
y3 = np.sin(math.pi*t-math.pi/2)
plt.plot(t, y1, 'b*', t, y2, 'g^', t, y3, 'ys')
plt.show()

```



Как видно на прошлом изображении, график представляет три разных тренда с помощью разных цветов и меток. В таких случаях когда тренд функции очевиден, график является не самой подходящим представлением — лучше использовать линии. Чтобы разделить их не только цветами, можно использовать паттерны, состоящие из комбинаций точек и дефисов.

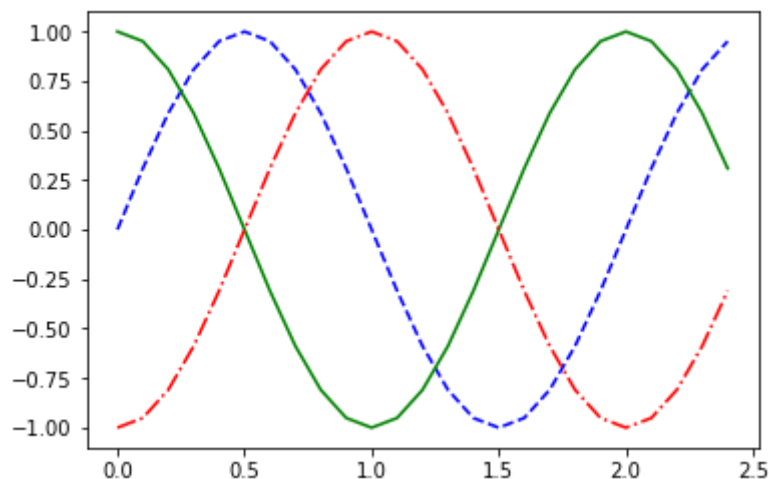
```

plt.plot(t, y1, 'b--', t, y2, 'g', t, y3, 'r-.')
plt.show()

```



Если вы не пользуетесь IPython QtConsole со встроенной matplotlib или работаете с этим кодом в обычной сессии Python, используйте команду `plt.show()` в конце кода для получения объекта графика со следующего изображения.



## Использование kwargs

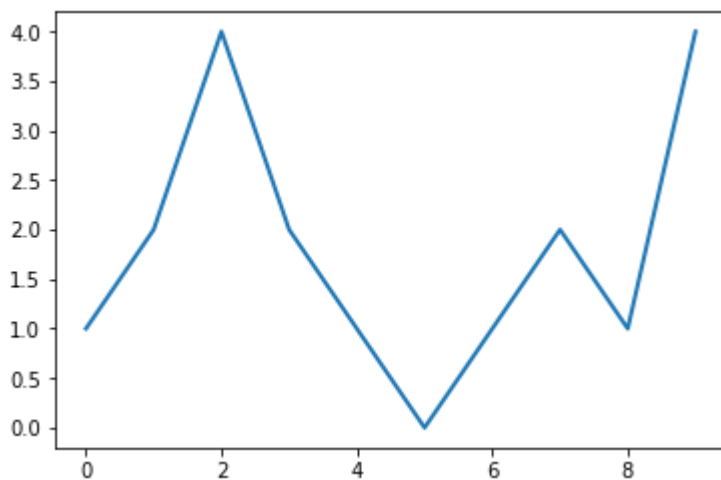
Объекты, которые создают график, имеют разные характеризующие их атрибуты. Все они являются значениями по умолчанию, но их можно настроить с помощью аргументов-ключевых слов — `kwargs`.

Эти ключевые слова передаются в качестве аргументов в функции. В документации по разным функциям библиотеки `matplotlib` они всегда упоминаются последними вместе с `kwargs`. Например, функция `plot()` описана следующим образом.

```
matplotlib.pyplot.plot(*args, **kwargs)
```

В качестве примера с помощью аргумента `linewidth` можно поменять толщину линии.

```
plt.plot([1,2,4,2,1,0,1,2,1,4], linewidth=2.0)  
plt.show()
```



## Работа с несколькими объектами Figure и осями

До сих пор во всех примерах команды `pyplot` были направлены на отображение в пределах одного объекта. Но `matplotlib` позволяет управлять несколькими Figure одновременно, а внутри одного объекта можно выводить подграфики.

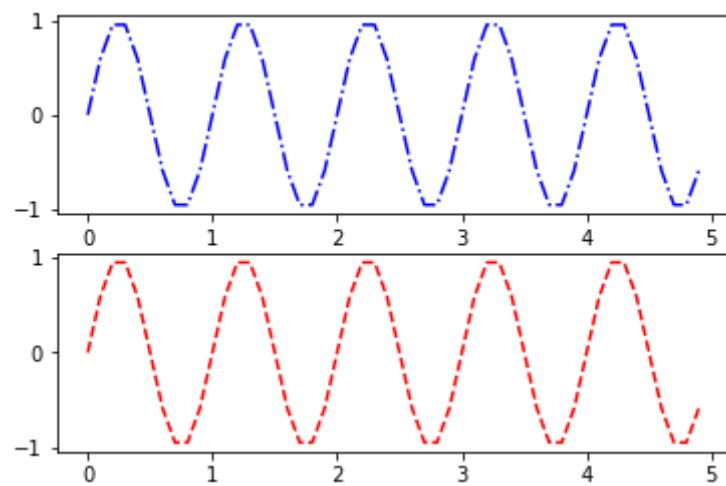
Работая с `pyplot`, нужно помнить о концепции текущего объекта Figure и текущих осей (графика на объекте).

Дальше будет пример с двумя подграфиками на одном Figure. Функция `subplot()`, помимо разделения объекта на разные зоны для рисования, используется для фокусировки команды на конкретном подграфике.

Аргументы, переданные `subplot()`, задают режим разделения и определяют текущий подграфик. Этот график будет единственным, на который воздействуют команды. Аргумент функции `subplot()` состоит из трех целых чисел. Первое определяет количество частей, на которое нужно разбить объект по вертикали. Второе — горизонтальное разделение. А третье число указывает на текущий подграфик, для которого будут актуальны команды.

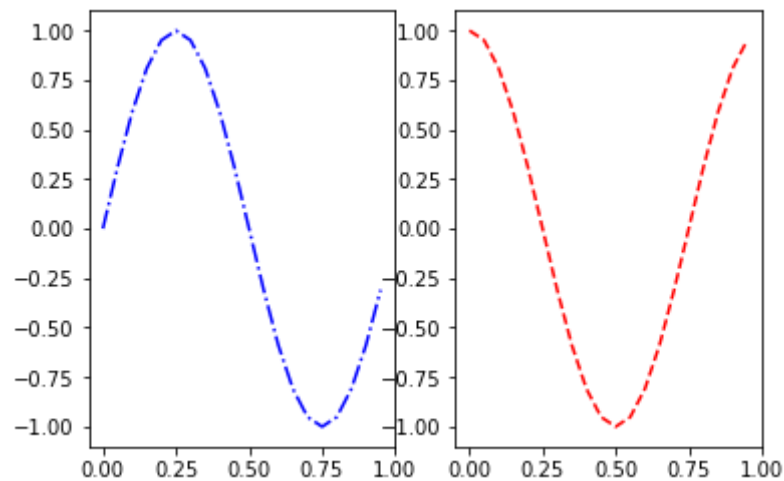
Дальше будут отображаться тренды синусоиды (синус и косинус), и лучше всего разделить полотно по вертикали на два горизонтальных подграфика. В график передают числа 211 и 212.

```
t = np.arange(0, 5, 0.1)
y1 = np.sin(2*np.pi*t)
y2 = np.sin(2*np.pi*t)
plt.subplot(211)
plt.plot(t, y1, 'b-.')
plt.subplot(212)
plt.plot(t, y2, 'r--')
plt.show()
```



Теперь — то же самое для двух вертикальных подграфиков. Передаем в качестве аргументов 121 и 122.

```
t = np.arange(0., 1., 0.05)
y1 = np.sin(2*np.pi*t)
y2 = np.cos(2*np.pi*t)
plt.subplot(121)
plt.plot(t, y1, 'b-.')
plt.subplot(122)
plt.plot(t, y2, 'r--')
plt.show()
```



## Добавление элементов на график

Чтобы сделать график более информативным, недостаточно просто представлять данные с помощью линий и маркеров и присваивать диапазон значений с помощью двух осей. Есть и множество других элементов, которые можно добавить на график, чтобы наполнить его дополнительной информацией.

В этом разделе добавим на график текстовые блоки, легенду и так далее.

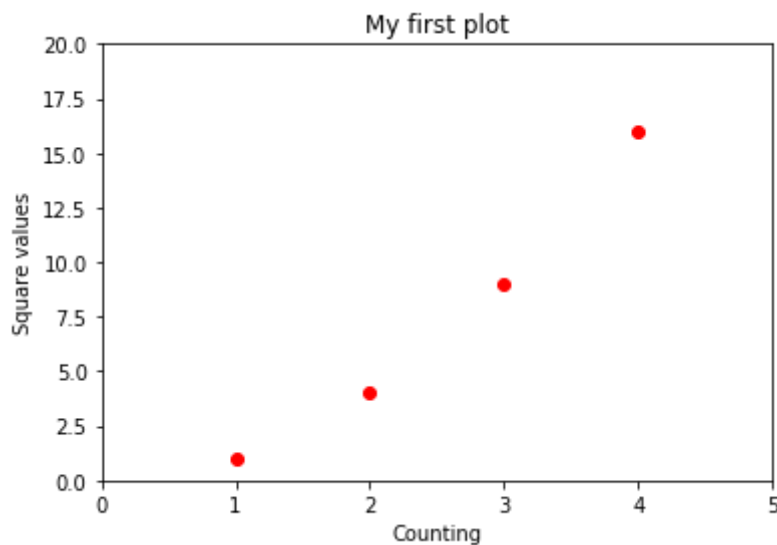
## Добавление текста

Вы уже видели, как добавить заголовок с помощью функции `title()`. Два других текстовых индикатора можно добавить с помощью меток осей. Для этого используются функции `xlabel()` и `ylabel()`. В качестве аргумента они принимают строку, которая будет выведена.

количество команд для представления графика постоянно растёт. Но их не нужно переписывать каждый раз. Достаточно использовать стрелки на клавиатуре, вызывая ранее введенные команды и редактируя их с помощью новых строк (в тексте они выделены жирным).

Теперь добавим две метки на график. Они будут описывать тип значений на каждой из осей.

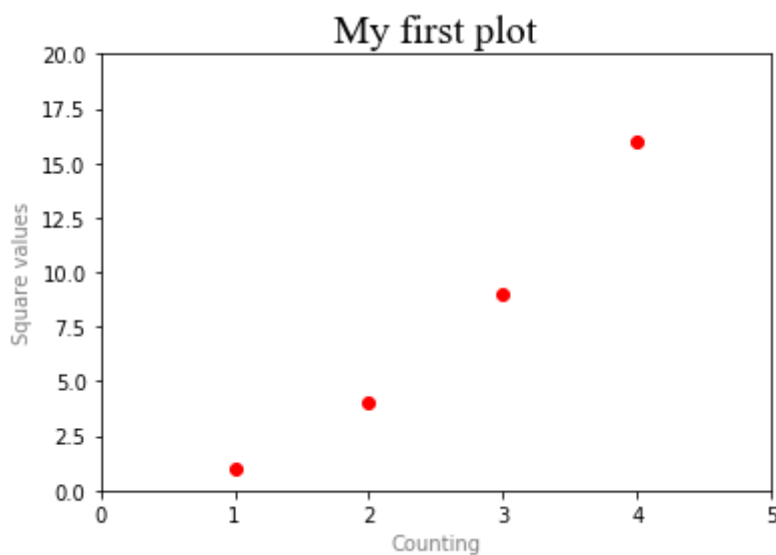
```
plt.axis([0, 5, 0, 20])  
plt.title('My first plot')  
plt.xlabel('Counting')  
plt.ylabel('Square values')  
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')  
plt.show()
```



Благодаря ключевым словам можно менять характеристики текста.

Например, можно поменять заголовок, выбрав другой шрифт и увеличив его размер. Также можно менять цвета меток осей, чтобы акцентировать внимание на заголовке всего графика.

```
plt.axis([0,5,0,20])  
plt.title('My first plot', fontsize=20, fontname='Times New Roman')  
plt.xlabel('Counting', color='gray')  
plt.ylabel('Square values',color='gray')  
plt.plot([1,2,3,4],[1,4,9,16], 'ro')  
plt.show()
```



Но в matplotlib можно делать даже больше: pyplot позволяет добавлять текст в любом месте графика. Это делается с помощью специальной функции `text()`.

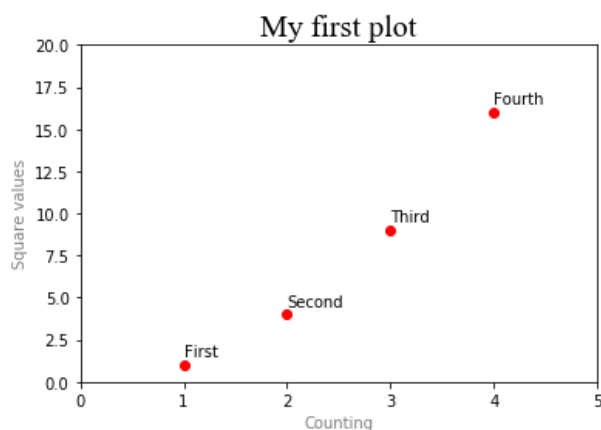
```
text(x,y,s, fontdict=None, **kwargs)
```

Первые два аргумента — это координаты, в которых нужно разметить текст. `s` — это строка с текстом, а `fontdict` (опционально) — желаемый шрифт. Разрешается использовать и ключевые слова.

Добавим метку для каждой точки графика. Поскольку первые два аргумента в функции являются координатами, координаты всех точек по оси `y` немного сдвинутся.

```
plt.axis([0, 5, 0, 20])
plt.title('My first plot', fontsize=20, fontname='Times New Roman')
plt.xlabel('Counting', color='gray')
plt.ylabel('Square values', color='gray')
plt.text(1, 1.5, 'First')
plt.text(2, 4.5, 'Second')
plt.text(3, 9.5, 'Third')
plt.text(4, 16.5, 'Fourth')
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.show()
```

Теперь у каждой точки есть своя метка.



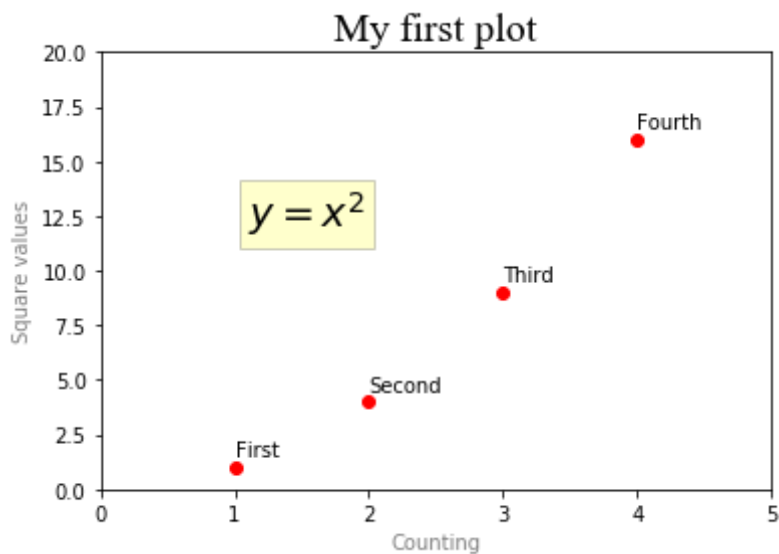


Поскольку matplotlib — это графическая библиотека, созданная для использования в научных кругах, она должна быть способна в полной мере использовать научный язык, включая математические выражения. matplotlib предоставляет возможность интегрировать выражения LaTeX, что позволяет добавлять выражения прямо на график.

Для этого их нужно заключить в два символа  $\$$ . Интерпретатор распознает их как выражения LaTeX и конвертирует соответствующий график. Это могут быть математические выражения, формулы, математические символы или греческие буквы. Перед LaTeX нужно добавлять `r`, что означает сырой текст. Это позволит избежать появления исключаящих последовательностей.

Также разрешается использовать ключевые слова, чтобы дополнить текст графика. Например, можно добавить формулу, описывающую тренд и цветную рамку.

```
plt.axis([0, 5, 0, 20])
plt.title('My first plot', fontsize=20, fontname='Times New Roman')
plt.xlabel('Counting', color='gray')
plt.ylabel('Square values', color='gray')
plt.text(1, 1.5, 'First')
plt.text(2, 4.5, 'Second')
plt.text(3, 9.5, 'Third')
plt.text(4, 16.5, 'Fourth')
plt.text(1.1, 12, r'$y = x^2$', fontsize=20,
bbox={'facecolor': 'yellow', 'alpha': 0.2})
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.show()
```



## Добавление сетки

Также на график можно добавить сетку. Часто это необходимо, чтобы лучше понимать положение каждой точки на графике.

Это простая операция. Достаточно воспользоваться функцией `grid()`, передав в качестве аргумента `True`.

```
plt.axis([0, 5, 0, 20])  
plt.title('My first plot', fontsize=20, fontname='Times New Roman')  
plt.xlabel('Counting', color='gray')  
plt.ylabel('Square values', color='gray')  
plt.text(1, 1.5, 'First')  
plt.text(2, 4.5, 'Second')  
plt.text(3, 9.5, 'Third')
```

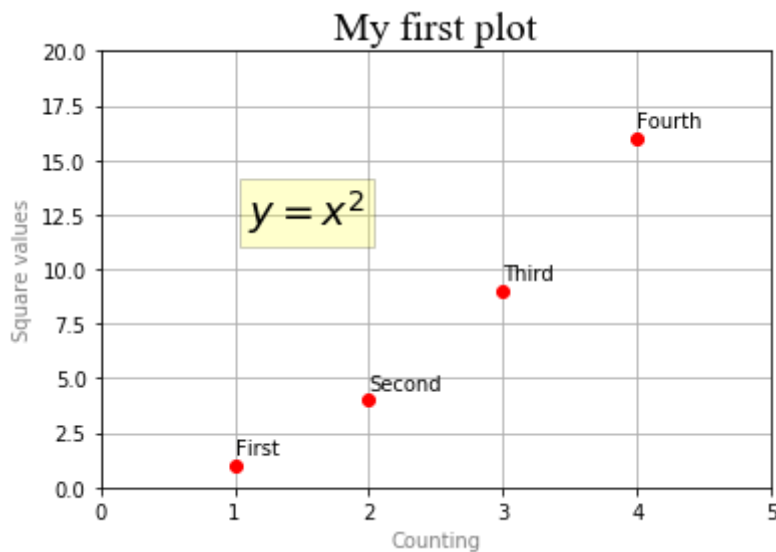
```
plt.text(4, 16.5, 'Fourth')

plt.text(1.1, 12, r'$y = x^2$', fontsize=20,
bbox={'facecolor': 'yellow', 'alpha': 0.2})

plt.grid(True)

plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')

plt.show()
```



## Добавление легенды

Также на графике должна присутствовать легенда. pyplot предлагает функцию `legend()` для добавления этого элемента.

В функцию нужно передать строку, которая будет отображаться в легенде. В этом примере текст `First series` характеризует входящий массив данных.

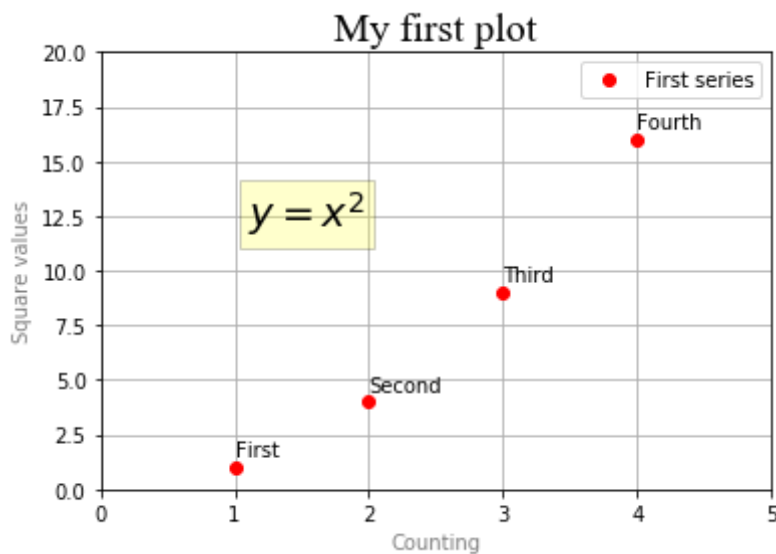
```
plt.axis([0, 5, 0, 20])

plt.title('My first plot', fontsize=20, fontname='Times New Roman')
```

```

plt.xlabel('Counting', color='gray')
plt.ylabel('Square values',color='gray')
plt.text(1,1.5,'First')
plt.text(2,4.5,'Second')
plt.text(3,9.5,'Third')
plt.text(4,16.5,'Fourth')
plt.text(1.1,12,r'$y = x^2$', fontsize=20,
bbox={'facecolor':'yellow','alpha':0.2})
plt.grid(True)
plt.plot([1,2,3,4],[1,4,9,16],'ro')
plt.legend(['First series'])
plt.show()

```



По умолчанию легенда добавляется в правый верхний угол. Чтобы поменять это поведение, нужно использовать несколько аргументов-ключевых слов. Так, для выбора положения достаточно передать аргумент `loc` со значением от 0 до 10. Каждая из цифр обозначает один из углов. Значение 1 — значение по умолчанию, то есть, верхний правый угол. В следующем примере переместим легенду в левый верхний угол, чтобы она не пересекалась с точками на графике.

Код положения	Положение
0	лучшее
1	Верхний правый угол
2	Верхний левый угол
3	Нижний левый угол
4	Нижний правый угол
5	Справа
6	Слева по центру
7	Справа по центру
8	Снизу по центру

9	Сверху по центру
10	По центру

Тут важно сделать одну ремарку. Легенды используются для указания определения набора данных с помощью меток, ассоциированных с определенным цветом и/или маркером. До сих пор в примерах использовался лишь один набор данных, переданный с помощью одной функции `plot()`. Но куда чаще один график может использоваться для нескольких наборов данных. С точки зрения кода каждый такой набор будет характеризоваться вызовом одной функции `plot()`, а порядок их определения будет соответствовать порядку текстовых меток, переданных в качестве аргумента функции `legend()`.

```
import matplotlib.pyplot as plt
plt.axis([0,5,0,20])
plt.title('My first plot', fontsize=20, fontname='Times New Roman')
plt.xlabel('Counting', color='gray')
plt.ylabel('Square values',color='gray')
plt.text(1,1.5,'First')
plt.text(2,4.5,'Second')
plt.text(3,9.5,'Third')
plt.text(4,16.5,'Fourth')
plt.text(1.1,12,r'$y = x^2$', fontsize=20,
bbox={'facecolor':'yellow', 'alpha':0.2})
plt.grid(True)
```

```
plt.plot([1,2,3,4],[1,4,9,16], 'ro')
plt.plot([1,2,3,4],[0.8,3.5,8,15], 'g^')
plt.plot([1,2,3,4],[0.5,2.5,4,12], 'b*')
plt.legend(['First series', 'Second series', 'Third series'],
loc=2)
plt.show()
```



## Сохранение графиков

В этом разделе разберемся, как сохранять график разными способами. Если в будущем потребуется использовать график в разных [Notebook](#) или сессиях Python, то лучший способ — сохранять графики в виде кода Python. С другой стороны, если они нужны в отчетах или презентациях, то подойдет сохранение в виде изображения. Можно даже сохранить график в виде HTML-страницы, что пригодится при работе в интернете.

## Сохранение кода

Как уже стало понятно, объем кода, отвечающего за представление одного графика, постоянно растет. Когда финальный результат удовлетворяет, его можно сохранить в файле `.py`, который затем вызывается в любой момент.

Также можно использовать команду `%save [имя файла] [количество строк кода]`, чтобы явно указать, сколько строк нужно сохранить. Если весь код написан одним запросом, тогда нужно добавить лишь номер его строки. Если же использовалось несколько команд, например, от 10 до 20, то эти числа и нужно записать, разделив их дефисом (10-20).

В этом примере сохранить весь код, отвечающий за формирование графика, можно с помощью ввода со строки 171.

```
In [171]: import matplotlib.pyplot as plt
```

Такую команду потребуется ввести для сохранения в файл `.py`.

```
%save my_first_chart 171
```

После запуска команды файл `my_first_chart.py` окажется в рабочей директории.

```
# %load my_first_chart.py
plt.axis([0, 5, 0, 20])
plt.title('My first plot', fontsize=20, fontname='Times New Roman')
plt.xlabel('Counting', color='gray')
```



```

plt.ylabel('Square values', color='gray')
plt.text(1, 1.5, 'First')
plt.text(2, 4.5, 'Second')
plt.text(3, 9.5, 'Third')
plt.text(4, 16.5, 'Fourth')
plt.text(1.1, 12, r'$y = x^2$', fontsize=20,
bbox={'facecolor': 'yellow', 'alpha': 0.2})
plt.grid(True)
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.plot([1, 2, 3, 4], [0.8, 3.5, 8, 15], 'g^')
plt.plot([1, 2, 3, 4], [0.5, 2.5, 4, 12], 'b*')
plt.legend(['First series', 'Second series', 'Third series'],
loc=2)
plt.show()

```

Позже, когда вы откроете сессию IPython, у вас уже будет готовый график и его можно редактировать с момента сохранения этой командой:

```
ipython qtconsole --matplotlib inline -m my_first_chart.py
```

Либо его можно загрузить заново в один запрос в QtConsole с помощью команды %load.

```
%load my_first_chart.py
```

Или запустить в уже активной сессии с помощью %run.

```
%run my_first_chart.py
```

в определенных случаях последняя команда будет работать только после ввода двух предыдущих.

## Сохранение сессии в HTML-файл

С помощью IPython QtConsole вы можете конвертировать весь код и графику, представленные в текущей сессии, в одну HTML-страницу. Просто выберите File → Save to HTML/XHTML в верхнем меню.

Будет предложено сохранить сессию в одном из двух форматов: HTML и XHTML. Разница между ними заключается в типе сжатия изображения. Если выбрать HTML, то все картинки конвертируются в PNG. В случае с XHTML будет выбран формат SVG.

В этом примере сохраним сессию в формате HTML в файле `my_session.html`.

Дальше программа спросит, сохранить ли изображения во внешней директории или прямо в тексте. В первом случае все картинки будут храниться в папке `my_session_files`, а во втором — будут встроены в HTML-код.

## Сохранение графика в виде изображения

График можно сохранить и виде файла-изображения, забыв обо всем написанном коде. Для этого используется функция `savefig()`. В аргументы нужно передать желаемое название будущего файла. Также важно, чтобы эта команда шла в конце, после всех остальных (иначе сохранится пустой PNG-файл).

```

plt.axis([0, 5, 0, 20])

plt.title('My first plot', fontsize=20, fontname='Times New Roman')

plt.xlabel('Counting', color='gray')
plt.ylabel('Square values', color='gray')

plt.text(1, 1.5, 'First')
plt.text(2, 4.5, 'Second')
plt.text(3, 9.5, 'Third')
plt.text(4, 16.5, 'Fourth')

plt.text(1.1, 12, r'$y = x^2$', fontsize=20,
bbox={'facecolor': 'yellow', 'alpha': 0.2})

plt.grid(True)

plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.plot([1, 2, 3, 4], [0.8, 3.5, 8, 15], 'g^')
plt.plot([1, 2, 3, 4], [0.5, 2.5, 4, 12], 'b*')

plt.legend(['First series', 'Second series', 'Third series'],
loc=2)

plt.savefig('my_chart.png')

```

Файл появится в рабочей директории. Он будет называться `my_chart.png` и включать изображение графика.

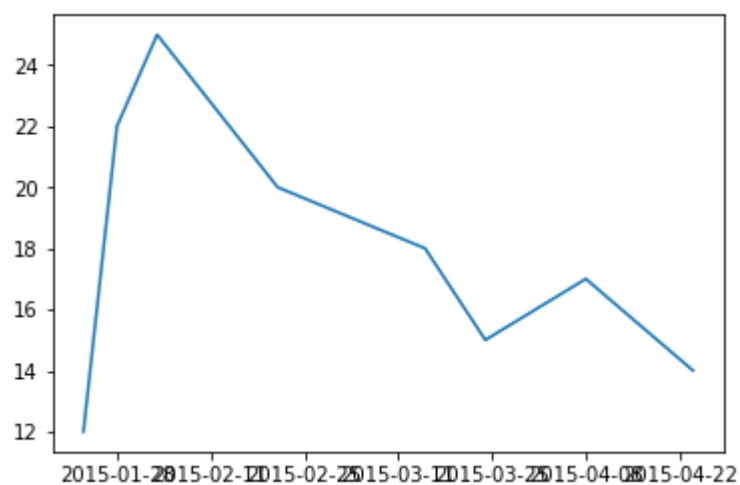
## Обработка значений дат

Одна из основных проблем при анализе данных — обработка значений дат. Отображение даты по оси (обычно это ось *x*) часто становится проблемой.

Возьмем в качестве примера линейный график с набором данных, который включает 8 точек, где каждая представляет точку даты на оси x в следующем формате: день-месяц-год.

```
import datetime
import numpy as np
import matplotlib.pyplot as plt

events = [datetime.date(2015, 1, 23),
          datetime.date(2015, 1, 28),
          datetime.date(2015, 2, 3),
          datetime.date(2015, 2, 21),
          datetime.date(2015, 3, 15),
          datetime.date(2015, 3, 24),
          datetime.date(2015, 4, 8),
          datetime.date(2015, 4, 24)]
readings = [12, 22, 25, 20, 18, 15, 17, 14]
plt.plot(events, readings)
plt.show()
```



Автоматическая расстановка отметок в этом случае — настоящая катастрофа. Даты сложно читать, ведь между ними нет интервалов, и они наслаиваются друг на друга.

Для управления датами нужно определить временную шкалу. Сперва необходимо импортировать `matplotlib.dates` — модуль, предназначенный для работы с этим типом дат. Затем указываются шкалы для дней и месяцев с помощью `MonthLocator()` и `DayLocator()`. В этом случае форматирование играет важную роль, и чтобы не получить наложение текста, нужно ограничить количество отметок, оставив только год-месяц. Такой формат передается в качестве аргумента функции `DateFormatter()`.

Когда шкалы определены (один — для дней, второй — для месяцев) можно определить два вида пометок на оси `x` с помощью `set_major_locator()` и `set_minor_locator()` для объекта `xaxis`. Для определения формата текста отметок месяцев используется `set_major_formatter`.

Задав все эти изменения, можно получить график как на следующем изображении.

```
import matplotlib.dates as mdates

months = mdates.MonthLocator()
days = mdates.DayLocator()
timeFmt = mdates.DateFormatter('%Y-%m')
events = [datetime.date(2015, 1, 23),
          datetime.date(2015, 1, 28),
```

```
datetime.date(2015, 2, 3),
datetime.date(2015, 2, 21),
datetime.date(2015, 3, 15),
datetime.date(2015, 3, 24),
datetime.date(2015, 4, 8),
datetime.date(2015, 4, 24)]
readings = [12, 22, 25, 20, 18, 15, 17, 14]
fig, ax = plt.subplots()
plt.plot(events, readings)
ax.xaxis.set_major_locator(months)
ax.xaxis.set_major_formatter(timeFmt)
ax.xaxis.set_minor_locator(days)
plt.show()
```

