

## Практическое занятие 1.4.2

### Размещение текстовых элементов на рисунке в matplotlib

На прошлом занятии мы строили двумерные линейные графики с помощью функции `plot()` и задавали различные стили его отображения: цвет, тип, толщину линии. На этом занятии продолжим тему оформления графиков.

Текст является одним из базовых графических элементов рисунка в `matplotlib`. Подписи координатных осей и их делений, заголовки, пояснительные подписи на графиках и диаграммах – это всё текст. В `Matplotlib` возможна поддержка кириллицы для создания научной графики с подписями на русском языке. Одним из весомых преимуществ `matplotlib` при работе с текстом является простая поддержка математических формул с помощью `LaTeX`.

`Matplotlib` имеет обширную текстовую поддержку, включая поддержку математических выражений, поддержку для растровых и векторных выходных данных, текст с произвольными поворотами. `Matplotlib` обеспечивает создание красивых, сглаженных шрифтов и предоставляет пользователю возможность управлять свойствами текста (изменять размер шрифта, его толщину, расположение и цвет текста и т.д.)

Рассмотрим типовые элементы оформления графиков.

#### 1. Создание надписей и подписей

Одними из самых базовых графических команд являются команды, отображающие текст. Такой командой, не привязанной к какому-либо объекту вроде координатной оси или делений координатной оси, является команда `plt.text()`.

В качестве входных данных она принимает координаты положения будущей строки и сам текст в виде строки. По умолчанию координаты положения строки будут приурочены к области изменения данных. Можно задать положение текста в относительных координатах, когда вся область рисования изменяется по обеим координатным осям от 0 до 1 включительно. Таким образом, координата (0.5, 0.5) в относительных координатах означает центр области рисования.

Текст можно выровнять с помощью параметров `horizontalalignment` и `verticalalignment`, а также заключать его в рамку с цветным фоном, передав параметр `bbox`. `Bbox` – это словарь, работающий со свойствами прямоугольника (объектом `Rectangle`).

Помимо метода `text()` существуют и другие методы отображения текста в `pyplot`:

Список текстовых команд в `pyplot`:

`plt.xlabel()` – добавляет подпись оси абсцисс  $Ox$ ;  
`plt.ylabel()` – добавляет подпись оси ординат  $Oy$ ;  
`plt.title()` – добавляет заголовок для области рисования `Axes`;  
`plt.figtext()` – добавляет текст на рисунок `Figure`;  
`plt.suptitle()` – добавляет заголовок для рисунка `Figure`;

`plt.annotate()` – добавляет примечание, которое состоит из текста и необязательной стрелки в указанную область на рисунке.

Посмотрим, как работают все эти элементы на графике:

Для начала работы в блокноте выполним:

```
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

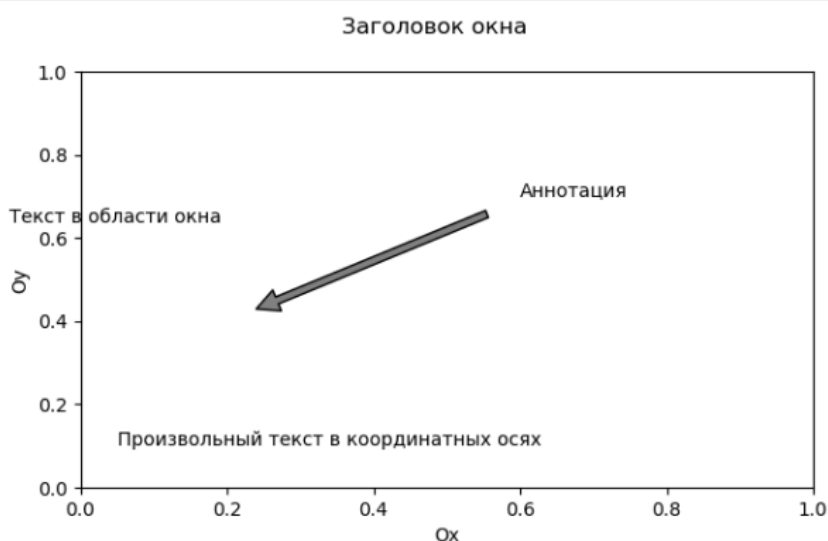
В некоторых случаях можно воспользоваться *jupyter notebook online*, а лучше выполнять работы в <https://colab.research.google.com/>.

### Пример 1.

```
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot()
plt.figtext(0.05, 0.6, 'Текст в области окна')
fig.suptitle('Заголовок окна')
ax.set_xlabel('Ox')
ax.set_ylabel('Oy')

ax.text(0.05, 0.1, 'Произвольный текст в координатных
осях')
ax.annotate('Аннотация', xy=(0.2, 0.4), xytext=(0.6, 0.7),
            arrowprops={'facecolor': 'gray', 'shrink':
0.1})

plt.show()
```



**Пример 2.** Если необходимо сфокусировать внимание на определенном участке графика – то это проще всего сделать с помощью стрелок:

Программный код Python:

```

x = np.linspace(-5, 5, 100)
y = x*(x - 4)*(x + 4)

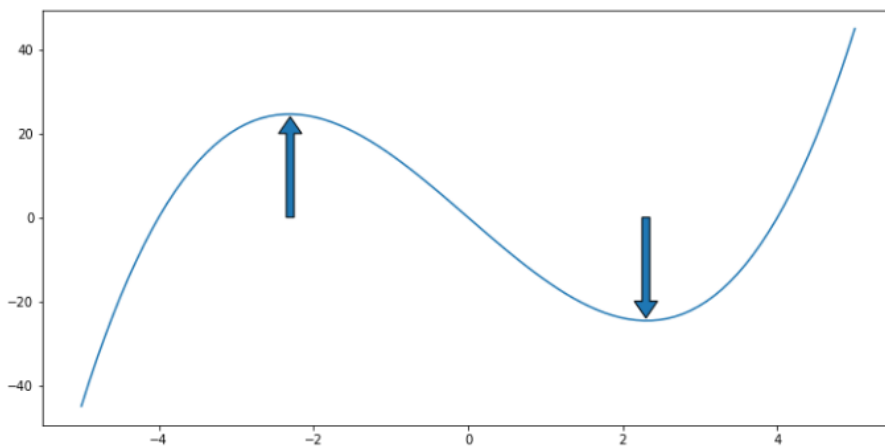
fig, ax = plt.subplots()

ax.plot(x, y)
ax.arrow(-2.3, 0, 0, 20,
        width = 0.1,
        head_length = 4)
ax.arrow(2.3, 0, 0, -20,
        width = 0.1,
        head_length = 4)

fig.set_figwidth(12)    # ширина и
fig.set_figheight(6)    # высота "Figure"

plt.show()

```



Arrow – стрелка – Всё о стрелке!!! – [https://pyprog.pro/mpl/mpl\\_arrow.html](https://pyprog.pro/mpl/mpl_arrow.html)

## 2. Оформление текстовых элементов

Как правило, все текстовые элементы в matplotlib – это объекты класса Text. И у них имеется стандартный набор свойств для их оформления. Полный их перечень можно посмотреть на странице официальной документации:

[https://matplotlib.org/stable/api/text\\_api.html](https://matplotlib.org/stable/api/text_api.html)

Рассмотрим наиболее употребительные:

Свойство	Описание
alpha	степень прозрачности (число в диапазоне [0; 1])
backgroundcolor	цвет фона
color или c	цвет текста

fontfamily или family	тип шрифта
fontsize или size	размер шрифта
fontstyle или style	стиль шрифта: {'normal', 'italic', 'oblique'}
fontweight или weight	степень утолщения – число от 0 до 1000 или константы: 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'
horizontalalignment или ha	выравнивание по горизонтали: {'center', 'right', 'left'}
label	текст заголовка
position	координаты текста (x, y)
rotation	поворот текста: вещественное число [0; 1] или константы {'vertical', 'horizontal'}
verticalalignment или va	выравнивание по вертикали: {'center', 'top', 'bottom', 'baseline', 'center_baseline'}
visible	отображение текста: True/False
x	координата x – вещественное число [0; 1]
y	координата y – вещественное число [0; 1]

Применим эти свойства в **Примере 1**.

**Пример 3.** В функции `figtext()` пропишем именованный параметр `fontsize`:

```
plt.figtext(0.05, 0.6, 'Текст в области окна', fontsize=24)
```

Теперь текст отображается с новым размером шрифта.

**Пример 4.** Укажем цвет:

```
plt.figtext(0.05, 0.6, 'Текст в области окна',  
fontsize=24, color='r')
```

Получим текст размером 24 и красным цветом.

По аналогии задаются все другие свойства.








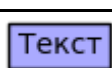
### 3. Параметр `bbox`

С помощью параметра `bbox` можно устанавливать дополнительные элементы оформления для текстовых блоков. Ему следует указать словарь, ключами которого являются аргументы класса `FancyBboxPatch`:

Свойство	Описание
boxstyle	вид рамки вокруг текста

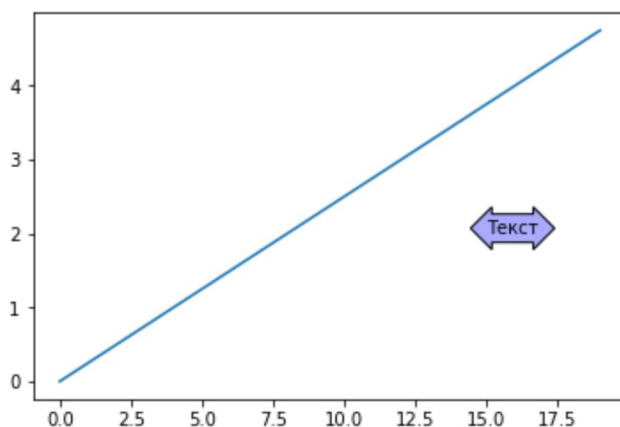
alpha	степень прозрачности фона
color	цвет фона с рамкой
edgecolor или ec	цвет рамки
facecolor или fc	цвет заливки
fill	использовать ли заливку: True/False
hatch	тип штриховки: {'/', '\', ' ', '-', '+', 'x', 'o', 'O', '.', '*'}
linestyle или ls	стиль линии границы: {'-', '--', '-.', ':', ' ', (offset, on-offseq), ...}
linewidth или lw	толщина рамки

В качестве параметра `boxstyle` можно выбирать следующие значения:

Атрибут	Описание	Вид
circle	имеет параметр <code>pad</code> (отступ от границы)	
darrow	имеет параметр <code>pad</code>	
larrow	имеет параметр <code>pad</code>	
rarrow	имеет параметр <code>pad</code>	
round	имеет параметры: <code>pad</code> , <code>rounding_size</code>	
roundtooth	имеет параметры: <code>pad</code> , <code>tooth_size</code>	
sawtooth	имеет параметры: <code>pad</code> , <code>tooth_size</code>	
square	имеет параметр <code>pad</code>	

**Пример 5.** Использование параметра `bbox`:

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.text(15, 2, 'Текст', bbox={'boxstyle':'darrow',
'facecolor': '#AAAAFF'})
ax.plot(np.arange(0, 5, 0.25))
plt.show()
```



#### 4. Цвет фона фигуры и координатных осей

Для выбора цвета и его названия можно пользоваться таблицей

black	linen	forestgreen	slategray
k	bisque	limegreen	lightsteelblue
dimgrey	darkorange	darkgreen	cornflowerblue
dimgray	burlywood	green	royalblue
grey	antiquewhite	g	ghostwhite
gray	tan	lime	lavender
darkgray	navajowhite	seagreen	midnightblue
darkgrey	blanchedalmond	mediumseagreen	navy
silver	papayawhip	springgreen	darkblue
lightgrey	moccasin	mintcream	mediumblue
lightgray	orange	mediumspringgreen	blue
gainsboro	wheat	mediumaquamarine	b
whitesmoke	oldlace	aquamarine	slateblue
white	floralwhite	turquoise	darkslateblue
w	darkgoldenrod	lightseagreen	mediumslateblue
snow	goldenrod	mediumturquoise	mediumpurple
rosybrown	cornsilk	azure	rebeccapurple
lightcoral	gold	lightcyan	blueviolet
indianred	lemonchiffon	paleturquoise	indigo
brown	khaki	darkslategray	darkorchid
firebrick	palegoldenrod	darkslategrey	darkviolet
maroon	darkkhaki	teal	mediumorchid
darkred	ivory	darkcyan	thistle
red	beige	c	plum
r	lightyellow	cyan	violet
mistyrose	lightgoldenrodyellow	aqua	purple
salmon	olive	darkturquoise	darkmagenta
tomato	y	cadetblue	m
darksalmon	yellow	powderblue	magenta
coral	olivedrab	lightblue	fuchsia
orangered	yellowgreen	deepskyblue	orchid
lightsalmon	darkolivegreen	skyblue	mediumvioletred
sienna	greenyellow	lightskyblue	deeppink
seashell	chartreuse	steelblue	hotpink
chocolate	lawngreen	aliceblue	lavenderblush
saddlebrown	honeydew	dodgerblue	palevioletred
sandybrown	darkseagreen	lightslategrey	crimson
peachpuff	palegreen	lightslategray	pink
peru	lightgreen	slategrey	lightpink

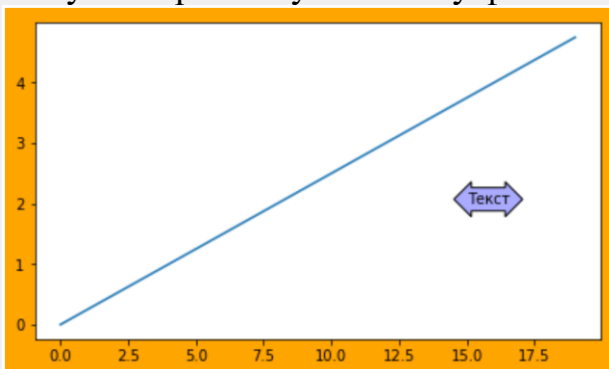
Используя параметр `facecolor` можно задавать цвет фона для всей формы и отдельно для координатных осей. Например, если при создании фигуры указать этот параметр в виде:

```
fig = plt.figure(figsize=(7, 4), facecolor='orange')
```

#### Пример 6.

```
fig = plt.figure(figsize=(7, 4), facecolor='orange')
ax = fig.add_subplot(111)
ax.text(15, 2, 'Текст', bbox={'boxstyle':'darrow',
'facecolor': '#AAAAFF'})
ax.plot(np.arange(0, 5, 0.25))
plt.show()
```

Получим оранжевую заливку фона окна.

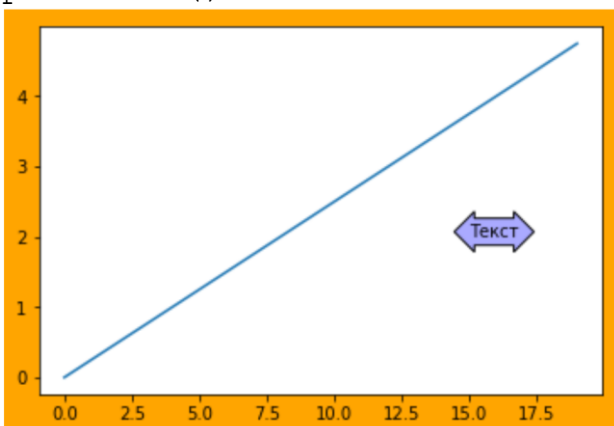


Это же можно сделать с помощью метода `set()` после создания фигуры:

```
fig.set(facecolor='orange')
```

#### Пример 7.

```
fig = plt.figure()
ax = fig.add_subplot(111)
fig.set(facecolor='orange')
ax.text(15, 2, 'Текст', bbox={'boxstyle':'darrow',
'facecolor': '#AAAAFF'})
ax.plot(np.arange(0, 5, 0.25))
plt.show()
```



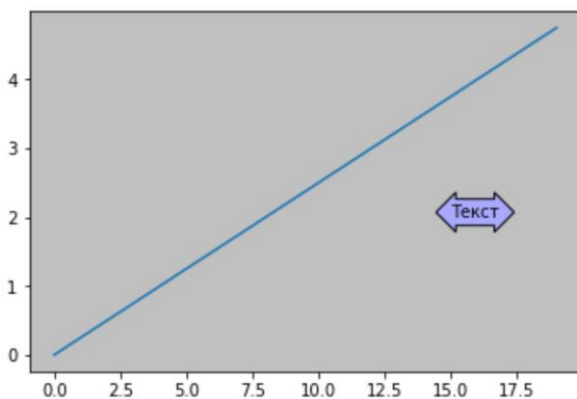
**Пример 8.** Изменим цвет фона области на серый (silver). Для этого вставим инструкцию

```
ax.set(facecolor='silver')
```

Программный код Python:

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set(facecolor='silver')
ax.text(15, 2, 'Текст', bbox={'boxstyle':'darrow',
'facecolor': '#AAAAFF'})
ax.plot(np.arange(0, 5, 0.25))
```

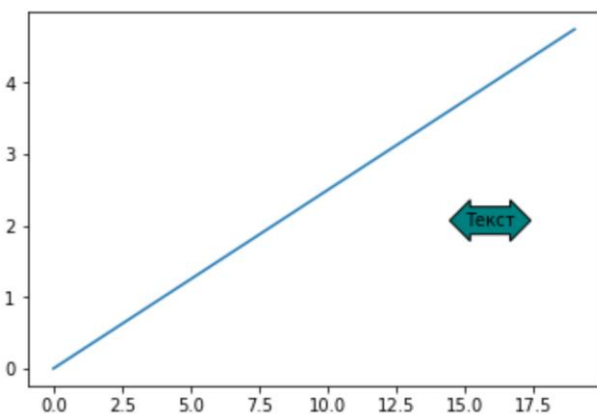
```
plt.show()
```



В программе цвет фигуры задан через код – '#AAAAFF'.

**Пример 9.** Заменяем цвет его названием, согласно приведенной выше палитре цветов, например, 'teal':

```
fig = plt.figure()
ax = fig.add_subplot(111)
ax.text(15, 2, 'Текст', bbox={'boxstyle':'darrow',
'facecolor': 'teal'})
ax.plot(np.arange(0, 5, 0.25))
plt.show()
```





На этом мы завершим знакомство с типовыми элементами оформления графиков. Поговорим об отображении легенды – краткой информации по каждому графику.

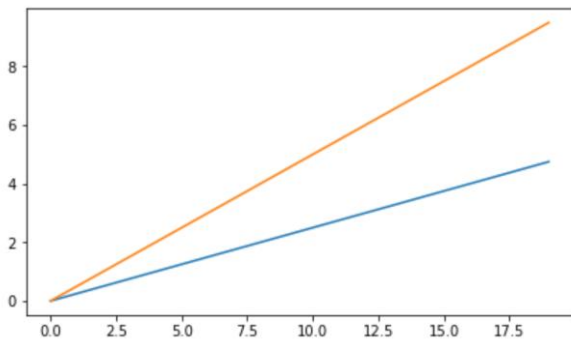
## 5. Легенда

Легенда – краткая информация по каждому графику, позволяющая определить что соответствует определенному цвету линии или маркера.

Пусть в координатных осях представлены два графика:

### Пример 10.

```
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)
ax.plot(np.arange(0, 5, 0.25))
ax.plot(np.arange(0, 10, 0.5))
plt.show()
```

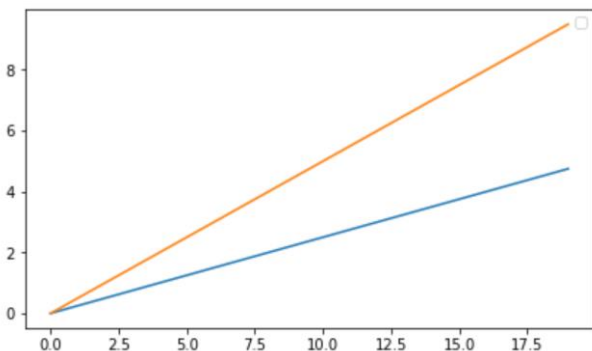


### Пример 11. Отобразим легенду с помощью метода `ax.legend()`

Программный код Python:

```
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)
ax.plot(np.arange(0, 5, 0.25))
ax.plot(np.arange(0, 10, 0.5))
ax.legend()
plt.show()
```

Появится сообщение: `No handles with labels found to put in legend` – *не найдено элементов, которые можно было вставить в легенду*. На рисунке увидим пустой квадратик в верхнем левом углу.



Почему в нем ничего нет? Дело в том, что нужно для каждой линии (графика) добавить символьное имя. Это делается с помощью именованного параметра `label`:

```
ax.plot(np.arange(0, 5, 0.25), label='line1')
ax.plot(np.arange(0, 10, 0.5), label='line2')
```

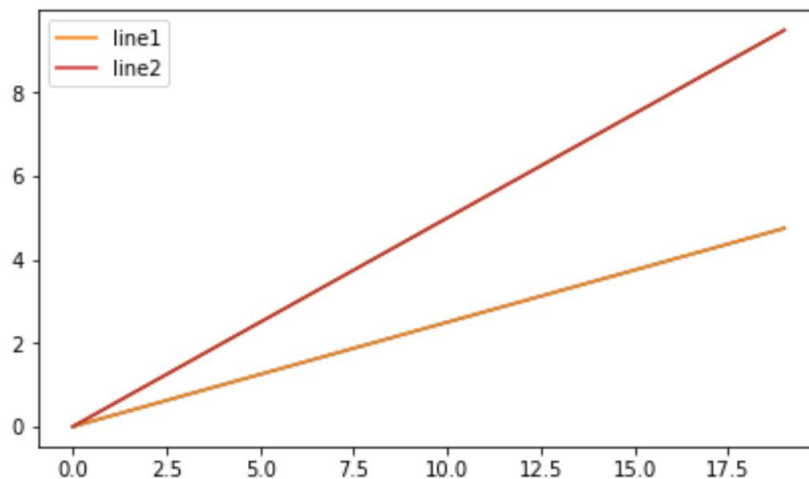
Окончательный программный код Python приведен в примере 12.

### Пример 12.

```
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)
ax.plot(np.arange(0, 5, 0.25))
ax.plot(np.arange(0, 5, 0.25), label='line1')
ax.plot(np.arange(0, 10, 0.5))
ax.plot(np.arange(0, 10, 0.5), label='line2')
ax.legend()

plt.show()
```

Теперь при запуске программы мы увидим следующее окно:



**Пример 13.** Легенда использует цвет и тип линии графика и, если его изменить, например, так:

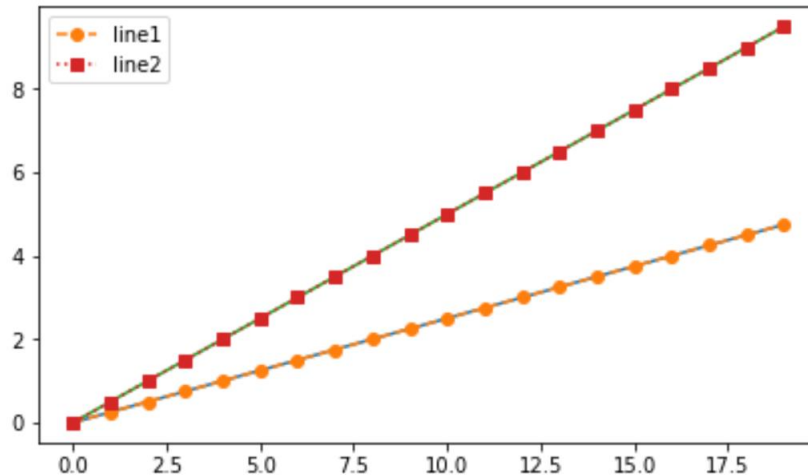
```
ax.plot(np.arange(0, 5, 0.25), '--o', label='line1')
ax.plot(np.arange(0, 10, 0.5), ':s', label='line2')
```

Программный код Python:

```
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)
ax.plot(np.arange(0, 5, 0.25))
ax.plot(np.arange(0, 5, 0.25), '--o', label='line1')
ax.plot(np.arange(0, 10, 0.5))
ax.plot(np.arange(0, 10, 0.5), ':s', label='line2')
ax.legend()

plt.show()
```

то это автоматически приведет к изменению и в окне легенды.



**Пример 14.** Мы можем самостоятельно указывать метки линий при отображении легенды. Для этого достаточно передать список меток в метод `legend()`:

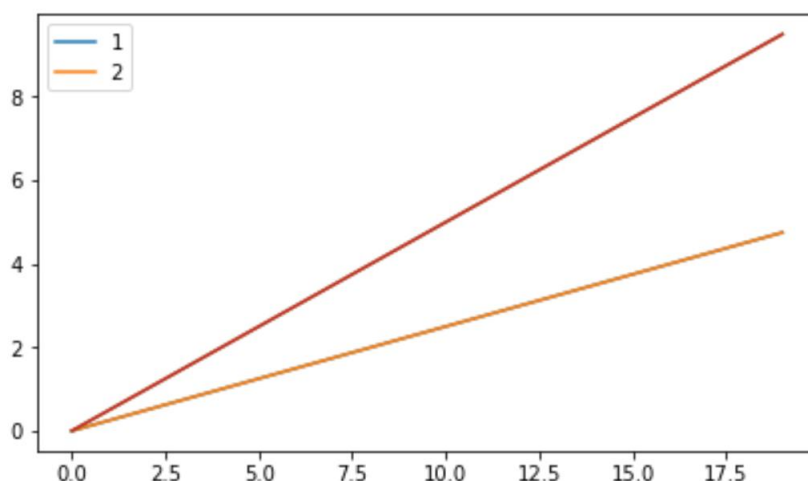
```
ax.legend(['1', '2'])
```

Параметры `label` в этом случае можно уже не прописывать.

Программный код Python:

```
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)
ax.plot(np.arange(0, 5, 0.25))
ax.plot(np.arange(0, 5, 0.25))
ax.plot(np.arange(0, 10, 0.5))
ax.plot(np.arange(0, 10, 0.5))
ax.legend(['1', '2'])
```

```
plt.show()
```



**Пример 15.** Третий вариант оформить легенду: мы можем указать список линий и меток при отображении информации в легенде:

```
line1, = ax.plot(np.arange(0, 5, 0.25), '--o',
label='line1')
```

```

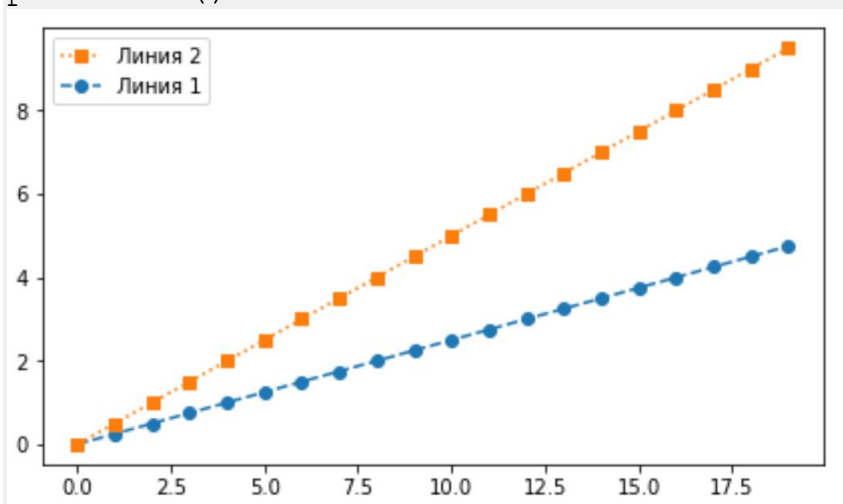
line2, = ax.plot(np.arange(0, 10, 0.5), ':s',
label='line2')
ax.legend((line2, line1), ['Линия 2', 'Линия 1'])
=====
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)

line1, = ax.plot(np.arange(0, 5, 0.25), '--o',
label='line1')
line2, = ax.plot(np.arange(0, 10, 0.5), ':s',
label='line2')

ax.legend((line2, line1), ['Линия 2', 'Линия 1'])

plt.show()

```



Далее, с помощью параметра `loc` можно указывать расположение окна легенды в пределах координатных осей. Этот параметр может принимать следующие значения:

```

['best', 'upper right', 'upper left', 'lower left', 'lower
right', 'right', 'center left', 'center right', 'lower
center', 'upper center', 'center']

```

Названия здесь говорят сами за себя.

**Пример 16.** Запись:

```

ax.legend((line2, line1), ['Линия 2', 'Линия 1'], loc='
lower right')

```

располагает легенду в нижнем правом углу.

Программный код Python:

```

fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)

```

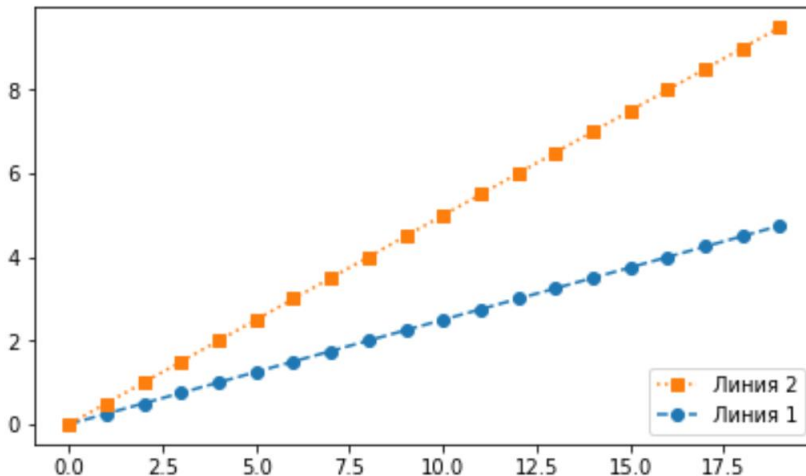
```

line1, = ax.plot(np.arange(0, 5, 0.25), '--o',
label='line1')
line2, = ax.plot(np.arange(0, 10, 0.5), ':s',
label='line2')

ax.legend((line2, line1), ['Линия 2', 'Линия 1'],
loc='lower right')

plt.show()

```



Аналогично используются и другие значения.

**Пример 17.** Отобразим в цикле все возможные расположения легенды, используя все 10 перечисленных выше значений параметра `loc`:

```

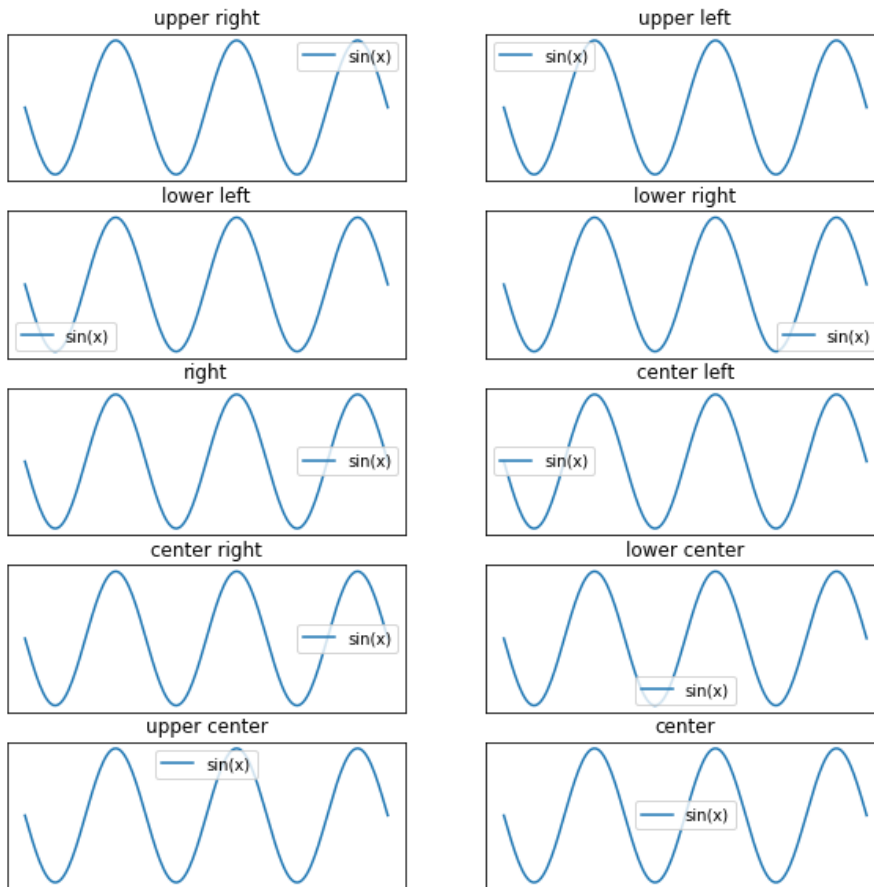
x = np.linspace(-3*np.pi, 3*np.pi, 200)
y1 = np.sin(x)
fig, axes = plt.subplots(5, 2)

location = ['upper right', 'upper left', 'lower left',
            'lower right', 'right', 'center left',
            'center right', 'lower center', 'upper center',
            'center']
i = 0

for ax in axes.ravel():
    ax.plot(x, y1, label = 'sin(x)')
    ax.legend(loc = location[i])
    ax.set_title(location[i])
    ax.set_xticks([])
    ax.set_yticks([])
    i += 1

fig.set_figheight(10)
fig.set_figwidth(10)
plt.show()

```



**Пример 18.** Если требуется более точное указание местоположения легенды, то можно воспользоваться параметром `bbox_to_anchor` и указать координаты информационного окна:

```
ax.legend((line2, line1), ['Линия 2', 'Линия 1'],
bbox_to_anchor=(0.5, 0.7))
```

Здесь значения прописываются в диапазоне от 0 до 1 как доли от размеров координатных осей.

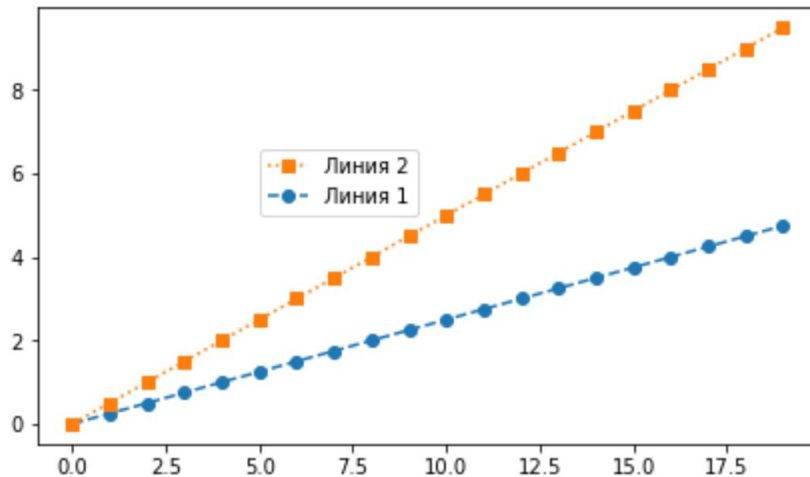
Программный код Python:

```
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)

line1, = ax.plot(np.arange(0, 5, 0.25), '--o',
label='line1')
line2, = ax.plot(np.arange(0, 10, 0.5), ':s',
label='line2')

ax.legend((line2, line1), ['Линия 2', 'Линия 1'],
bbox_to_anchor=(0.5, 0.7))

plt.show()
```



**Пример 19.** Настройка параметров содержащей текст области с помощью параметра `bbox`, который в свою очередь принимает словарь из ключей параметров и их значений:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

box_1 = {'facecolor': 'teal', # цвет области
        'edgecolor': 'red',   # цвет крайней линии
        'boxstyle': 'round'}  # стиль области

box_2 = {'facecolor': 'k',
        'edgecolor': 'r',
        'boxstyle': 'circle',
        'linestyle': ':',     # начертание линии
        'linewidth': '3'}     # толщина линии

box_3 = {'facecolor': 'teal', # цвет области
        'edgecolor': 'red',   # цвет крайней линии
        'boxstyle': 'arrow',  # стиль области
        'pad': 0.9}          # отступы

ax.text(0.05, 0.5, 'Какой-то текст',
        bbox = box_1,
        color = 'white',      # цвет шрифта
        fontsize = 20)

ax.text(0.5, 0.35, 'Какой-то\n текст',
        bbox = box_2,
        horizontalalignment = 'center', # горизонтальное
выравнивание
        color = 'white',
        fontsize = 20)

ax.text(0.7, 0.5, 'Какой-то текст',
        bbox = box_3,
        color = 'white',
```

```

        fontsize = 20)

fig.set_figwidth(12)
fig.set_figheight(2)

plt.show()

```

**Был рисунок черный**



**Мой рисунок**



**Пример 20.** Рассмотрим еще один интересный параметр легенды `shadow`, который устанавливает тень легенды и убирает ее прозрачность:

```

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-3*np.pi, 3*np.pi, 200)
y = np.sin(x)

fig, axes = plt.subplots(1, 2)
axes[0].legend(shadow = True,

axes[0].plot(x, y, label = 'sin(x)')

        fontsize = 15)
axes[0].set_title('shadow = True')

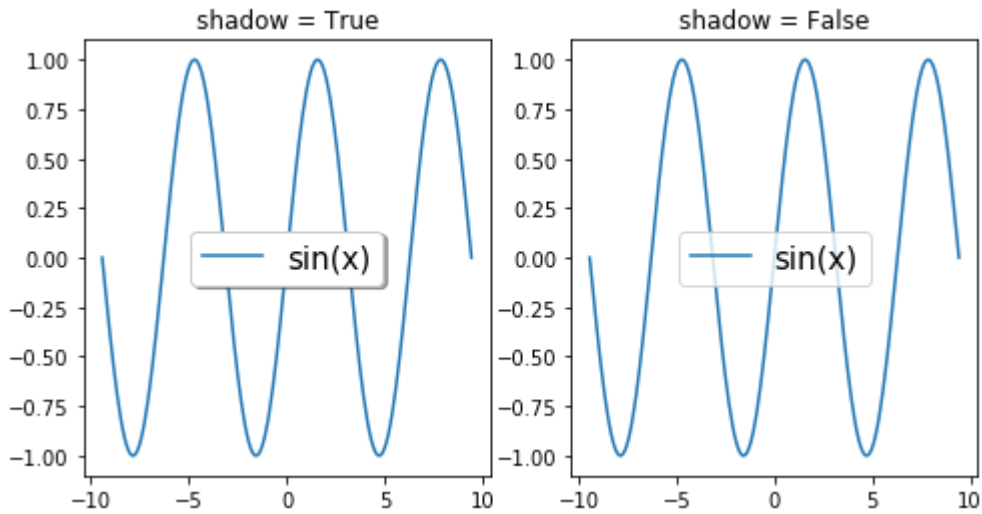
axes[1].plot(x, y, label = 'sin(x)')
axes[1].legend(shadow = False,
        fontsize = 15)
axes[1].set_title('shadow = False')

fig.set_figwidth(8)
fig.set_figheight(4)

```



```
plt.show()
```



**Пример 21.** Вариант настройки внешнего вида легенды:

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

x = np.linspace(-5, 5, 10)

y1 = 2*x + 5
y2 = -x + 8
y3 = 0.5*x + 5
y4 = np.full(10, 5)

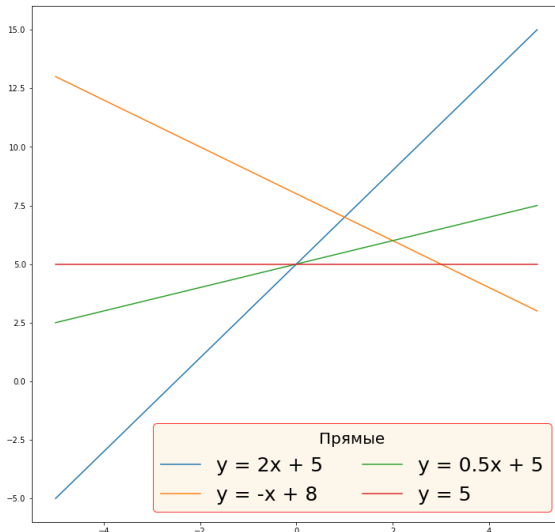
fig, ax = plt.subplots()

ax.plot(x, y1, label = 'y = 2x + 5')
ax.plot(x, y2, label = 'y = -x + 8')
ax.plot(x, y3, label = 'y = 0.5x + 5')
ax.plot(x, y4, label = 'y = 5')

ax.legend(fontsize = 25,
          ncol = 2,      # количество столбцов
          facecolor = 'oldlace', # цвет области
          edgecolor = 'r',    # цвет крайней линии
          title = 'Прямые',   # заголовок
          title_fontsize = '20' # размер шрифта заголовка
        )

fig.set_figwidth(12)
fig.set_figheight(12)

plt.show()
```



**Пример 22.** Пакет `matplotlib` позволяет отображать формулы, записанные в формате TeX. Для этого описание нужно заключить между символами `$` и использовать режим записи `r` (raw – «сырая» строка без экранирования символов). Например, можно сформировать следующие подписи у графиков:

```
ax.legend((line2, line1), [r'$f(x) = a \cdot b + c$',
r'$f(x) = k \cdot x + b$'])
```

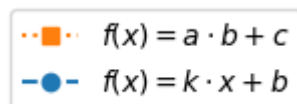
```
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)
```

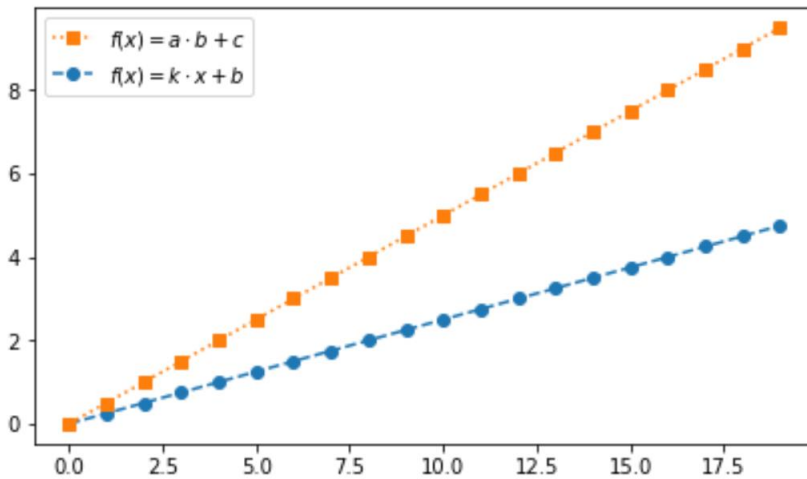
```
line1, = ax.plot(np.arange(0, 5, 0.25), '--o',
label='line1')
line2, = ax.plot(np.arange(0, 10, 0.5), ':s',
label='line2')
```

```
ax.legend((line2, line1), [r'$f(x) = a \cdot b + c$',
r'$f(x) = k \cdot x + b$'])
```

```
plt.show()
```

В результате, увидим такое окно легенды:





Причем, такую TeX-нотацию можно применять для любых текстовых элементов пакета `matplotlib`.

Для оформления информации в окне легенды можно использовать следующие параметры:

Параметр	Описание
<code>fontsize</code>	Размер шрифта (число или строка: {'xxsmall', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'})
<code>frameon</code>	Отображать ли рамку у легенды (True/False)
<code>framealpha</code>	Прозрачность фона (вещественное число или None)
<code>facecolor</code>	Цвет заливки
<code>edgecolor</code>	Цвет рамки
<code>title</code>	Текст заголовка, либо значение None
<code>title_fontsize</code>	Размер шрифта для заголовка

**Пример 23.** Пропишем следующее оформление:

```
ax.legend((line2, line1), ['Линия 2', 'Линия 1'],
bbox_to_anchor=(0.5, 0.6), facecolor='darkviolet',
framealpha=0.5)
```

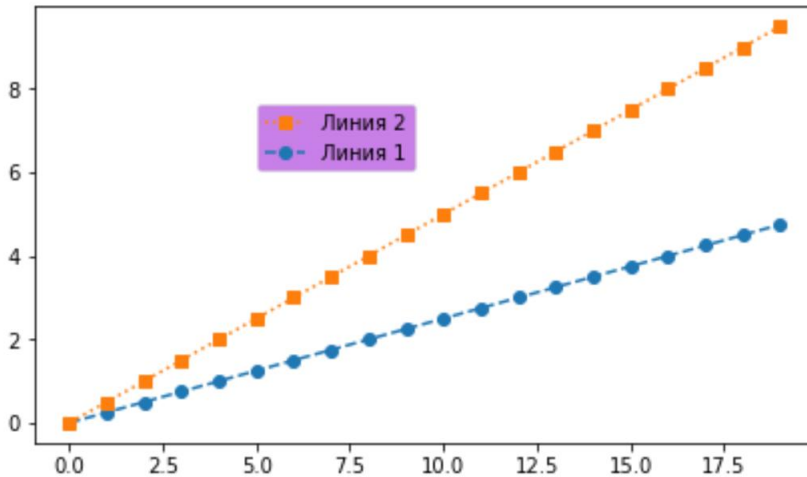
Программный код Python:

```
fig = plt.figure(figsize=(7, 4))
ax = fig.add_subplot(111)

line1, = ax.plot(np.arange(0, 5, 0.25), '--o',
label='line1')
line2, = ax.plot(np.arange(0, 10, 0.5), ':s',
label='line2')
```

```
ax.legend((line2, line1), ['Линия 2', 'Линия 1'],
bbox_to_anchor=(0.5, 0.6), facecolor='darkviolet',
framealpha=0.5)

plt.show()
```



## 6. Шрифты. Стили и форматы

В настройках `matplotlibrc` или `rcParams` существует такой параметр как `fonts`, то есть шрифты. Всего существует 5 наборов шрифтов:

- `cursive`;
- `fantasy`;
- `monospace`;
- `sans-serif`;
- `serif`.

Один из этих пяти наборов является текущим. За это отвечает параметр `font.family`. Каждый набор может состоять из одного или более шрифтов. Данная настройка определяет шрифт для всех подписей и текста на рисунке. Если конкретную подпись необходимо сделать другим шрифтом, можно указать шрифт из текущего стиля прямо в команде, передав в качестве соответствующего параметра словарь:

```
{'fontname': 'название_шрифта'}.
```

Помимо семейств, текст также может иметь стиль. Атрибут стиля `style` может быть либо `'italic'`, либо `'oblique'`, либо `'normal'` (по умолчанию).

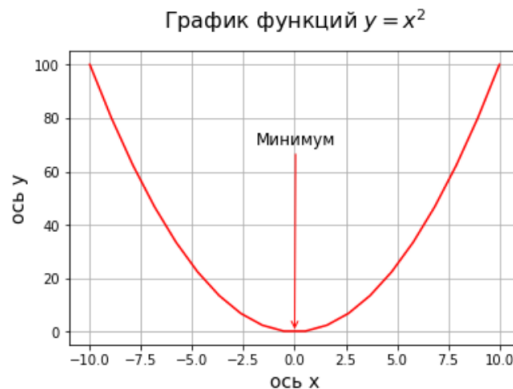
Толщина или "жирность" шрифта, может быть задана через атрибут `fontweight`, который принимает значения `'bold'`, `'light'` или `'normal'` (по умолчанию). Стили и форматы можно комбинировать.

### Пример 24.

```
x = np.linspace(-10, 10, 20)
y = x**2
plt.plot(x, y, color='red', label='Линия 1')
plt.grid() # наносится сетка
```

```
plt.xlabel('ось x', fontsize=14) # добавляем подпись к оси абцисс
"ось x"
plt.ylabel('ось y', fontsize=14) # добавляем подпись к оси ординат
"ось y"
plt.title(r'График функций  $y = x^2$ ', fontsize=16, y=1.05) #
добавляем заголовок к графику "График функции  $y=x^2$ "

plt.annotate('Минимум', xy=(0.,0.), xytext=(-1.9, 70.), fontsize=12,
arrowprops = dict(arrowstyle = '->',color = 'red'))
```



Лучший вариант изменить шрифт:  
программный код Python:

```
x = np.linspace(-10, 10, 20)
y = x**2
plt.plot(x, y, color='red', label='Линия 1')
plt.grid() # наносится сетка
plt.xlabel('ось x', fontsize=14) # добавляем подпись к оси
абцисс "ось x"
plt.ylabel('ось y', fontsize=14) # добавляем подпись к оси
ординат "ось y"
ssfont = {'fontname':'cursive'}
#ssfont = {'fontname':'fantasy'}
plt.title(r'График функций  $y = x^2$ ', fontsize=16, y=1.05,
**ssfont) # добавляем заголовок к графику "График функции
 $y=x^2$ "
#plt.text(-2., 10., 'Минимум',fontsize=12); # добавляем
подпись к графику в точке (-2.5, 10)
plt.annotate('Минимум', xy=(0.,0.), xytext=(-1.9, 70.),
fontsize=12, arrowprops = dict(arrowstyle = '->',color =
'red'))
# Параметр xy определяет координаты точки, которую
необходимо подписать (конечная точка стрелки)
# Параметр xytext определяет координаты точки, в которой
будет располагаться текст

plt.show()
```

Самостоятельно можно рассмотреть *gif*-анимацию. Задание необязательное для выполнения.

## 7. GIF анимация

Иногда нам необходимо наблюдать за изменением графика в зависимости от изменения определенного параметра. Такую возможность лучше (проще) всего реализовать в виде *gif*-анимации:

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

# Импортируем модуль для работы с анимацией:
import matplotlib.animation as animation

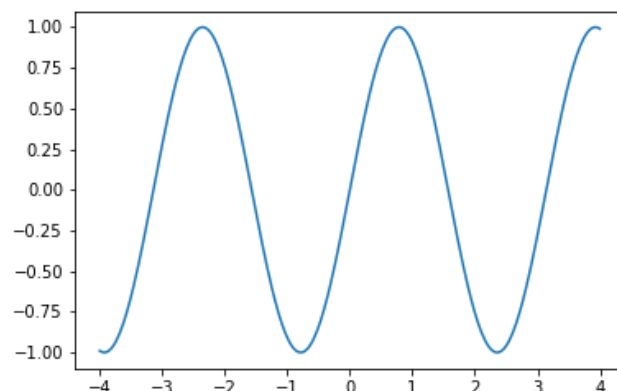
t = np.linspace(-4, 4, 300)

fig, ax = plt.subplots()

# Создаем функцию, генерирующую картинки
# для последующей "склейки":
def animate(i):
    ax.clear()
    line = ax.plot(t, np.sin(i*t))
    return line

# Создаем объект анимации:
sin_animation = animation.FuncAnimation(fig,
                                       animate,
                                       frames=np.linspace(2, 4, 30),
                                       interval = 10,
                                       repeat = False)

# Сохраняем анимацию в виде gif файла:
sin_animation.save('моя анимация.gif',
                  writer='imagemagick',
                  fps=30)
```



«Работающий» рисунок приведен в файле *gif*-анимация.

Данный пример демонстрирует самый простой способ создания анимации – использование класса `FuncAnimation`. Данный класс позволяет создавать экземпляры анимации и сохранять. При создании указываются следующие атрибуты: `fig` – объект области *Figure*, который используется для получения рисунков анимации. `func` – функция которая генерирует кадры анимации, в нашем случае это функция `animate(i)`. Первый аргумент в данной функции должен определять получение кадров анимации, т.е. как то влиять на изменение картинки, в примере выше параметр `i` используется при построении функции `np.sin(i*t)`. Следующий атрибут `frames` по сути это может быть любой итерируемый объект, длина которого определяет количество кадров анимации, в примере выше, это `np.linspace(2, 4, 30)`, т.е. 30 кадров. `Interval` задает задержку кадров в миллисекундах (по умолчанию 200). И, наконец, `repeat` – управляет повторением последовательности кадров после завершения их показа (по умолчанию `True`).

После того, как был создан экземпляр анимации можно переходить к его сохранению с помощью метода `save()`. Данный метод принимает в качестве строки имя выходного файла. Далее следует параметр `writer` – "создатель" анимации, т.е. модуль который способен склеить и сохранить картинки графика в файл, в нашем случае используется `imagemagick` так как мы хотим создать *gif*-анимацию. И последний используемый параметр – `fps` – количество кадров в секунду. Помимо этого может быть указан параметр `dpi` – разрешение (количество точек на единицу длины) получаемых кадров. Данный параметр позволяет повысить качество анимации, но в то же время возрастает и объем конечного файла.