matplotlibで矢印を描画したい! annotateを使おう!

Python　matplotlib

この記事は最終更新日から1年以上が経過しています。

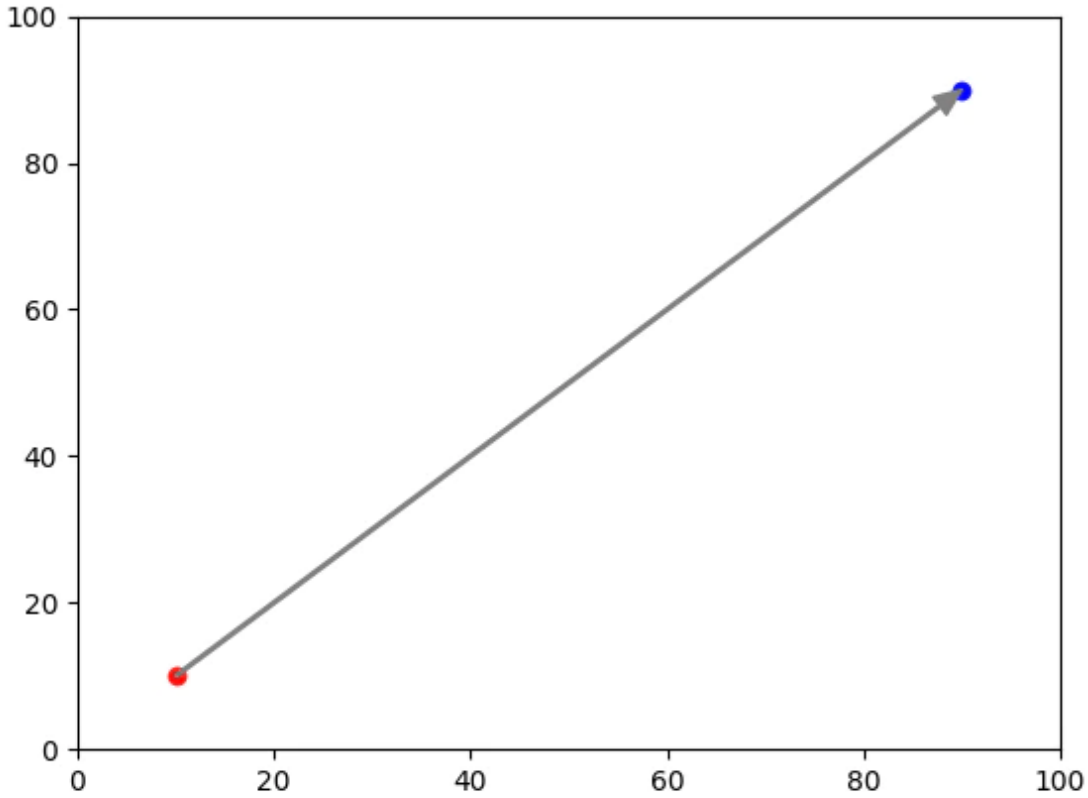
描画見本

とりあえず先に矢印を描画するプログラムを出しておきます.
一刻でも早く知りたい方用です.

```
import numpy as np
from matplotlib import pyplot

def arrow():
    fig = pyplot.figure()
    ax = fig.add_subplot(111)
```

```
    point = {
        'start': [10, 10],
        'end': [90, 90]
    }

    ax.plot(*point['start'], 'o', color="red")
    ax.plot(*point['end'], 'o', color="blue")

    ax.annotate('', xy=point['end'], xytext=point['start'],
                arrowprops=dict(shrink=0, width=1, headwidth=8
                                headlength=10, connectionstyle
                                facecolor='gray', edgecolor='g
                )

    ax.set_xlim([0, 100])
    ax.set_ylim([0, 100])

    pyplot.show()

if __name__ == "__main__":
    arrow()
```

出力結果はこんな感じになります.



矢印の描画について

以下で矢印の描画に関する部分のみ説明していきます.

今回は annotate を利用した矢印の描画方法について説明します.

annotateの主な引数

矢印の描画に関する引数だけ説明していきます.

引数名	役割
s	注釈文章の内容
xy	終点の位置
xytext	始点の位置(テキストの位置)
arrowprops	始点と終点間を結ぶ矢印の設定

もともとannotateは注釈を加えるためのもので, 注釈文を描画し, そこから任意のポイントまで矢印を伸ばすように描画することを想定されています.
ですが今回は注釈文を `s=""` のようにして文章を消すことで, 矢印のみを表示することができます.

矢印の設定を行う `arrowprops` について書いていきます.

`arrowprops` について

`arrowprops` は辞書形式で矢印の描画に関する設定が行えます.

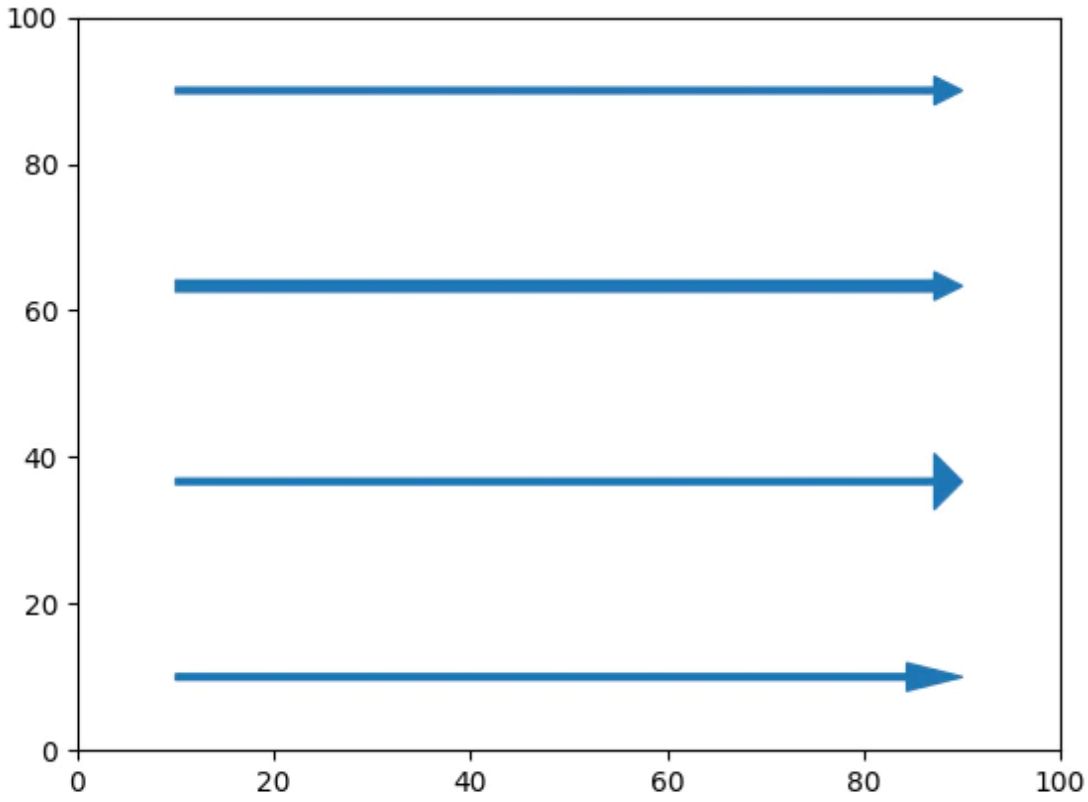
少し特殊でキー arrowstyle を**設定する場合と設定しない場合**で大きく変わっていきます.
arrowstyle は矢印のテンプレを利用できるみたいなものです.

- キー arrowstyle を設定しない場合

arrowstyle を利用しない場合, 以下のキーに関して設定を行うことができます.

キー	説明
width	矢印の線の幅
headwidth	矢印自体の横幅
headlength	矢印自体の長さ
shrink	なんだろう…？

矢印は一方向で, 特に変更はできない(はず)です.
デフォルト, width 大, headwidth 大, headlength 大にして比較したもの以下に貼っておきます.



- キー arrowstyle を設定する場合

arrowstyle に値を設定し, 用意されている設定を利用することができます.

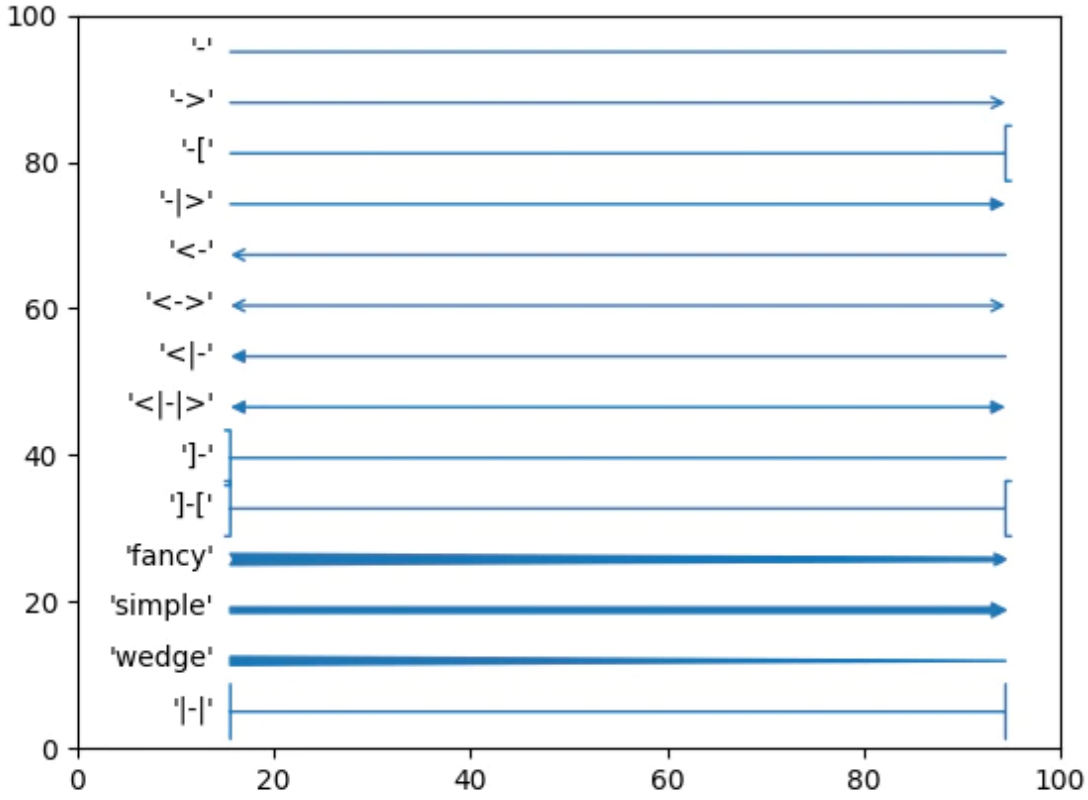
例

```
ax.annotate('', xy=start, xytext=end,
            arrowprops=dict(arrowstyle='-|>',
                            connectionstyle='arc3',
                            facecolor='C0',
```

```
                edgecolor='C0')
        )
```

名前	属性
-	None
->	head_length=0.4, head_width=0.2
-[	widthB=1.0, lengthB=0.2, angleB=None
-|>	head_length=0.4, head_width=0.2
<-	head_length=0.4, head_width=0.2
<->	head_length=0.4, head_width=0.2
<|-	head_length=0.4, head_width=0.2
<|-|>	head_length=0.4, head_width=0.2
]-	widthA=1.0, lengthA=0.2, angleA=None
]-[	widthA=1.0, lengthA=0.2, angleA=None, widthB=1.0, lengthB=0.2, angleB=None
fancy	head_length=0.4, head_width=0.4, tail_width=0.4
simple	head_length=0.5, head_width=0.5, tail_width=0.2
wedge	tail_width=0.3, shrink_factor=0.5
|-|	widthA=1.0, angleA=None, widthB=1.0, angleB=None

それぞれを利用して描画したものが以下になります.



線の太さとかはいじれないっぽいので, 細かく設定したい場合は向きません.

- 他のキー

`arrowstyle` を利用しても/しなくても以下のキーに対して値を設定することができます.

Key	Description
connectionstyle	the connection style
relpos	default is (0.5, 0.5)

Key	Description
patchA	default is bounding box of the text
patchB	default is None
shrinkA	default is 2 points
shrinkB	default is 2 points
mutation_scale	default is text size (in points)
mutation_aspect	default is 1.
?	any key for matplotlib.patches.PathPatch

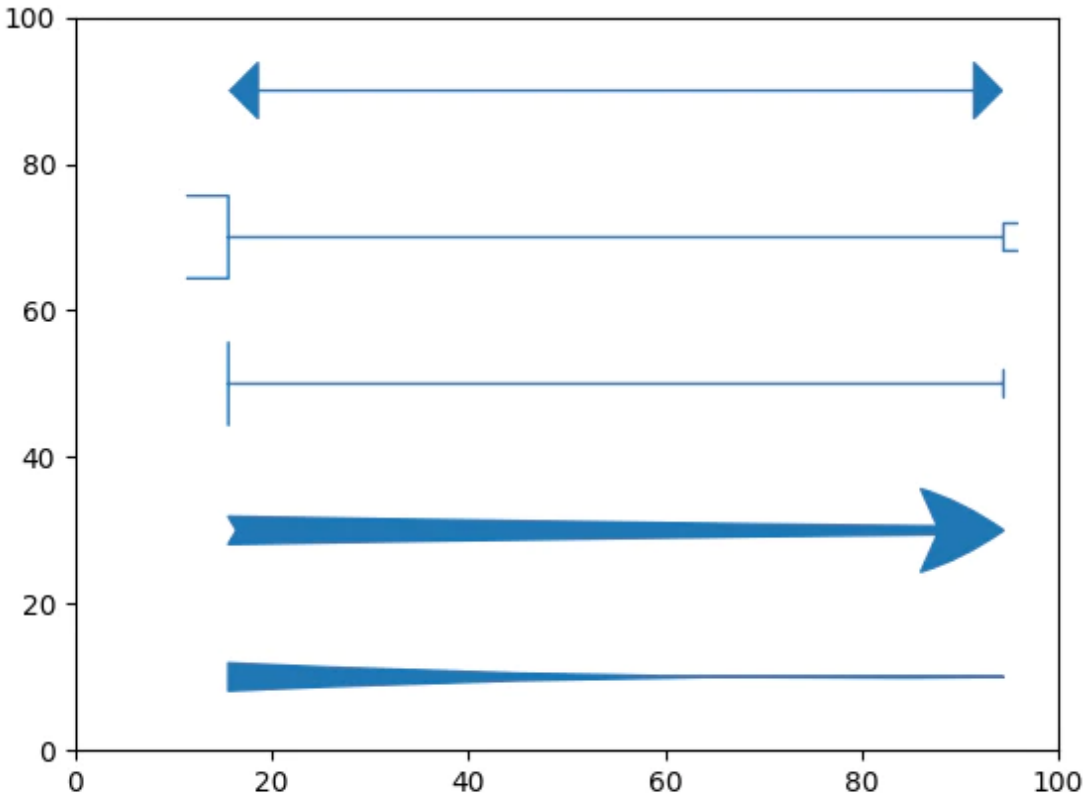
矢印をもっと細かく変更したい!

`arrowstyle` を設定すると矢印の太さなどが固定され, 設定しなければ矢印の形などが設定できない...
と思いきや,
`arrowstyle` には先ほどあげたような文字列以外に, `matplotlib.patches.ArrowStyle` クラスのインスタンスを渡すことができ, より細かく矢印自体の設定を行うことができます.

以下にソースコードとその出力をあげます.

```
from matplotlib import pyplot
from matplotlib.patches import ArrowStyle
import numpy as np

def arrow():
    fig = pyplot.figure()
    ax = fig.add_subplot(111)

    arrow_style_list = [
        ArrowStyle('<|-|>', head_length=1, head_width=1),
        ArrowStyle(']-[', lengthA=1.5, lengthB=0.5, widthA=1.5
        ArrowStyle('|-|', widthA=1.5, widthB=0.5),
        ArrowStyle('fancy', head_length=3, head_width=3, tail_
        ArrowStyle('wedge', tail_width=1, shrink_factor=0.1)
    ]
    x_start = 15
    x_end = 95
    y_list = np.linspace(10.0, 90.0, num=len(arrow_style_list)

    for y, arrow_style in zip(y_list[::-1], arrow_style_list):
        # 矢印の描画
        ax.annotate('', xy=(x_end, y), xytext=(x_start, y),
                    arrowprops=dict(arrowstyle=arrow_style,
                                    facecolor='C0',
                                    edgecolor='C0')
                    )

    ax.set_xlim([0, 100])
```

```
    ax.set_ylim([0, 100])

    pyplot.show()

if __name__ == '__main__':
    arrow()
```



ソースコードを見て解ると思いますが, 各スタイルで設定できる属性はスタイル毎に異なります.

どのような属性がいじれるかは先程の「 arrowstyle に設定できる値」の表の属性をご覧ください.