第4回
関数とオブジェクト（メソッド）
＋
モジュール
＋
簡単なグラフ描画

目次

関数とは

- これまでの演習で、いくつかの関数を使ってきました。
  - print関数、int関数、str関数、input関数、range関数など
- 関数は、プログラム内でよく利用する処理や作業を、手軽に呼び出せるように用意されているもので、Pythonで最初から用意されている関数を組み込み関数と呼びます。
- 関数の呼び出し（復習）
  - 構文 `戻り値を受け取る変数 = 関数名（引数1, 引数2, ・・・）`

関数を定義する

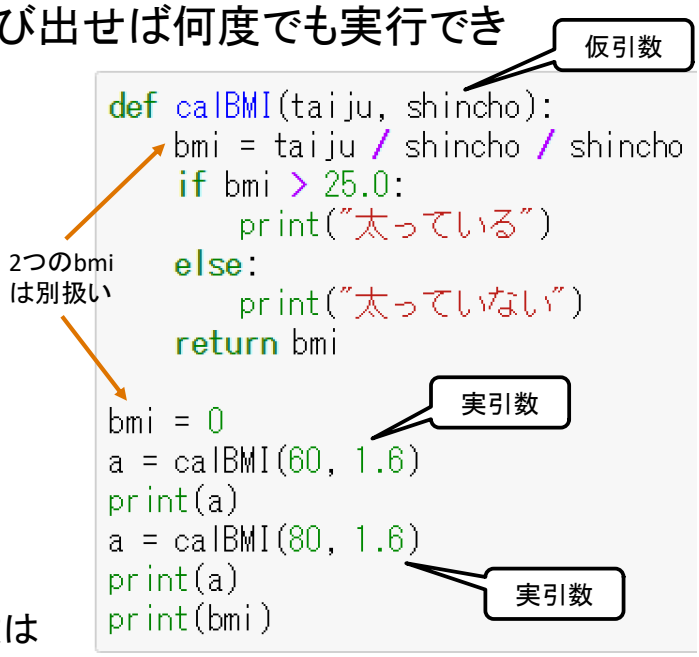
- Pythonでは新しい関数を自分で作る（定義する）こともできます。
  - 繰り返し利用するような処理や、他のプログラムでも再利用できそうな処理は、関数にまとめておくとプログラムを効率的に作ることができます。
  - 関数定義の構文
    - 関数はdef文で定義します。
    - defの後に関数名を置き、引数を受け取る変数のリストを丸括弧で囲みます。引数がない場合は丸括弧の中は空にします。
    - 関数内の処理はブロックに書きます。
    - 呼び出し元に結果を返すにはreturn文を使います。
  - Spyderのメニューから[ファイル]-[新規ファイル]を選択して、あたらしいソースファイルを作成します。

BMIを計算するcalBMI関数の定義例です。
- 引数として体重、身長（単位：m）を受け取ります。
- 最後にreturn 文で変数bmiの値を返します。

```
def calBMI(taiju, shincho):
    bmi = taiju / shincho / shincho
    if bmi > 25.0:
        print("太っている")
    else:
        print("太っていない")
    return bmi
```

関数を呼び出す

- いったん定義した関数は、呼び出せば何度でも実行できるので便利です。
- 実引数と仮引数
  - 関数呼び出しで指定する引数のことを実引数と呼びます。
  - 関数定義の引数のリストに記述した変数を仮引数と呼びます。「仮」とは、実際に関数が呼び出されるまでどのような値が入るかわからないからです。
- ローカル変数
  - 関数の中で使われている変数は関数の中でだけ有効です。
  - 関数の中と外で同じ名前の変数があったとしても、別のものとして扱われます。




```
def calBMI(taiju, shincho):
    bmi = taiju / shincho / shincho
    if bmi > 25.0:
        print("太っている")
    else:
        print("太っていない")
    return bmi

bmi = 0
a = calBMI(60, 1.6)
print(a)
a = calBMI(80, 1.6)
print(a)
print(bmi)
```

```
太っていない
23.4375
太っている
31.25
0
```

引数のキーワード指定とデフォルト値

- 引数のキーワード指定
  - 複数の引数をとる関数を呼び出す場合、引数の順序に注意する必要があります。
  - 実引数に対して仮引数名をキーワード指定することで、順番にかかわらず目的の仮引数に値を渡すことができます。
  - 方法：引数指定の際に、「仮引数名=実引数」の形式で渡す

```
a = calBMI(shincho=1.6, taiju=90)
print(a)
a = calBMI(taiju=40, shincho=1.5)
print(a)
```

- 引数のデフォルト値
  - 関数定義の仮引数の後ろに「=値」として、デフォルト値を設定しておくことができます。デフォルト値が設定された引数は、呼び出す際に省略することができます。

```
def calBMI(taiju, shincho=1.5):
    bmi = taiju / shincho / shincho
     ・・・以下省略・・・
```

複数の戻り値

- 戻り値はreturn文で呼び出し元に返すことができますが、複数の戻り値を戻すこともできます。
  - 構文 `return 戻り値1, 戻り値2, ・・・`

- 呼び出し側での受け取り
  - 複数の戻り値を返す関数を呼び出した場合、結果はタプル型で受け取れます。
  - タプルでなく単独の変数として受け取るには、=の前に戻り値の数だけ変数をカンマで区切って並べます。

タプルで受け取る場合

```
def calBMI(taiju, shincho=1.5):
    bmi = taiju / shincho / shincho
    if bmi > 25.0:
        txt = "太っている"
    else:
        txt = "太っていない"
    return bmi, txt
a = calBMI(60, 1.6)
print(a)
```

```
(23.4375, '太っていない')
```

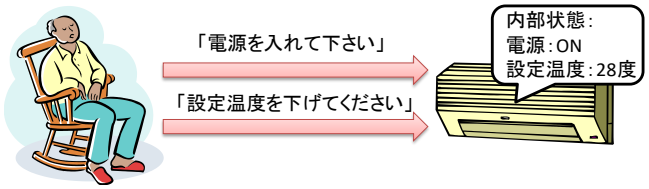
単独の変数で受け取る場合

```
a, b = calBMI(60, 1.6)
print(a)
print(b)
```

```
23.4375
太っていない
```


オブジェクト

- 「人間」は「エアコン」の電源を入れるために、リモコンの電源スイッチを押す
  - これは、「人間」というモノ（オブジェクト）が、「エアコン」というモノに対して「電源を入れて下さい」というメッセージを送信したことになります
- 「人間」と「エアコン」の関係は、何の変哲もないもの
  - しかし、驚くべき利便性を秘めています
  - 「人間」は、「エアコン」内部の機械（コンプレッサーやモーターなど）のしくみや、リモコンとエアコン本体の間の通信技術などについて全く知らなくても、リモコンのボタンを押しさえすれば「エアコン」の電源を入れたり設定温度を変更することができます
  - また、「エアコン」の設定温度は「エアコン」自身が記憶・管理しているため、「人間」が設定温度を忘れてしまっても支障が無いし、99度などの異常な温度に設定できない仕組みとなっています



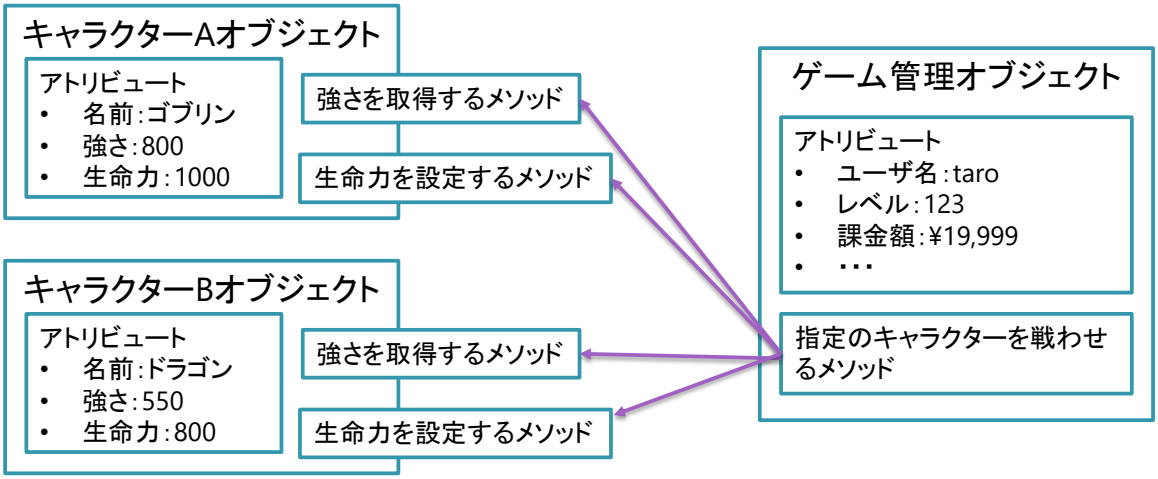
オブジェクト指向とは

- 現実世界の「モノ」の利便性を、プログラミング開発に取り入れた考え方
- オブジェクト指向による利便性
  - 例えば、オブジェクトAからオブジェクトBの機能を利用することを考えます

  - オブジェクAは、オブジェクトBの詳細な中身を知らなくても、どのようなメソッドがあるかだけを知っていれば、オブジェクトBの機能を呼び出すことが出来ます
  - オブジェクトBの状態はオブジェクトBの内部に記憶されているので、オブジェクトAで気に掛ける必要がありません
  - オブジェクトAとオブジェクトBは独立した部品となっているので、それぞれ個別に開発するか、すでに開発されたものを再利用することができます

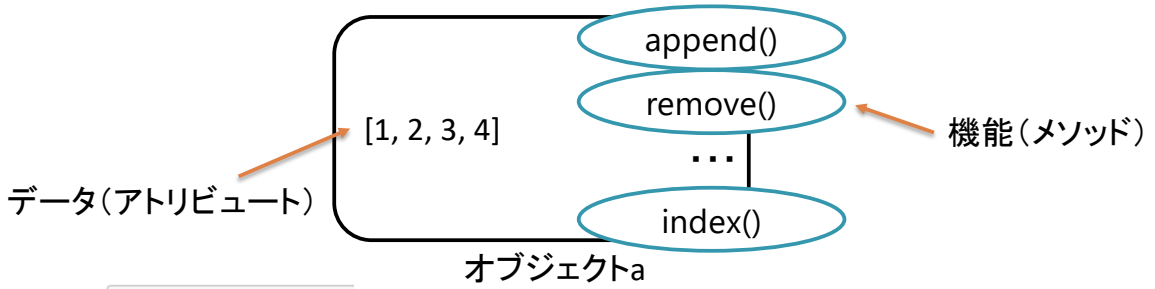
Pythonにおけるオブジェクト（1）

- Pythonなどのオブジェクト指向型のプログラミング言語では、「オブジェクト（Object）」と呼ばれる機能部品を組み合わせてプログラムを開発できます
- オブジェクトには、オブジェクトの特性や状態を記憶しておくためのデータである複数の「アトリビュート(Attribute)」と、アトリビュートに対する処理を実行する複数の「メソッド(Method)」が含まれています
- プログラムを実行すると、オブジェクト同士はお互いに相手のメソッドを呼び出しながら動作します



Pythonにおけるオブジェクト(2)

- Pythonでは、すべての組み込みデータ型(数値型、文字列型、リスト等)はオブジェクトの特徴を持っています。
  - オブジェクトは、データ(アトリビュート)と、データに対する機能(メソッド)を併せ持った存在です。
  - 例:リスト　a = [1, 2, 3, 4]




```
a = [1,2,3,4]
print(a)
del a[2]
print(a)
a.remove(1)
print(a)
```

del文を使ってインデックス2を削除

removeメソッドを使って値1を削除

```
[1, 2, 3, 4]
[1, 2, 4]
[2, 4]
```


クラス

- Pythonでは組み込み型以外の様々なオブジェクトが利用できます。
  - クラスはオブジェクトの設計図であり、組み込み型以外のオブジェクトは、クラスをもとに生成します。
  - クラスをもとにしたオブジェクトを生成する構文

インスタンス変数名 = クラス名(引数)

  - 演習:日付を管理するためのdateクラス
    - dateクラスは、Pythonの標準モジュールであるdatetimeモジュールに備わっていますが、そのままでは使用できないので、事前にimportする(読み込む)必要があります。
    - Spyderで[ファイル]-[新規ファイル]として、下記のコードを試しましょう。

```
from datetime import date
```

この文法については後ほど解説します

```
a = date(2021, 3, 6)
print(a)
```

2021-03-06

  - インスタンスとは?
    - クラスをもとに生成した個々のオブジェクトを、クラスを実体化した存在という意味を明確化するためにインスタンス(実体)と呼ぶ場合があります。

dateオブジェクトの使用例

- dateクラスのオブジェクトには、便利なメソッドや演算が備わっています。
  - 曜日を数字で取得するweekdayメソッド
    - 月曜日を0、日曜日を6として、曜日を整数で取得できます。
  - 日数計算が可能
    - インスタンスの差(ー)で日数が計算可能です。

```
from datetime import date

a = date(2021, 3, 6)
print(a)
b = date(2021, 2, 20)
print(b)
print(a.weekday())
print(a - b)
```

曜日の取得

日数計算

```
2021-03-06
2021-02-20
5
14 days, 0:00:00
```


モジュール

- Pythonには、標準モジュールと呼ばれるプログラムを構成する多数の部品が備わっています。
- 各モジュールには、関連する関数やクラスが用意されており、組み込みのデータ型や関数では実現できないような複雑な機能を手軽に実行することができます。
- モジュールを利用するには、import文を使ってあらかじめモジュールを読み込んでおく必要があります。
  - 構文

    import モジュール名

  - 演習:math(数学)モジュールのインポート
    - Spyderで[ファイル]-[新規ファイル]として、下記のコードを試しましょう。

```
import math
a = math.sqrt(2)
print(a)
```

インポート

平方根を計算する関数

  - モジュールに含まれる関数やクラスを呼び出すには、頭に「モジュール名.」を付けます。

from文

- import文に合わせてfrom文を使うと、関数をモジュール名を省略して呼び出すことができるようになります。
  - 構文

    from モジュール名 import 関数名

  - 例

```
from math import sqrt
a = sqrt(2)
print(a)
```

sqrt関数をモジュール名を省略して呼び出し

- インポートする関数名として*を指定すると、モジュール内のすべての関数を呼び出せるようになります。
  - 例

```
from math import *
a = sqrt(2)
print(a)
```

＊はワイルドカードとして機能

as文

- import文に合わせてas文を使うと、読み込んだモジュールや関数に別名を割り当てることができます。
  - 構文

    import モジュール名 as 別名
    from モジュール名 import 関数名 as 別名

  - 例

```
import math as m
a = m.sqrt(2)
print(a)
```

mathモジュールをmとしてインポート

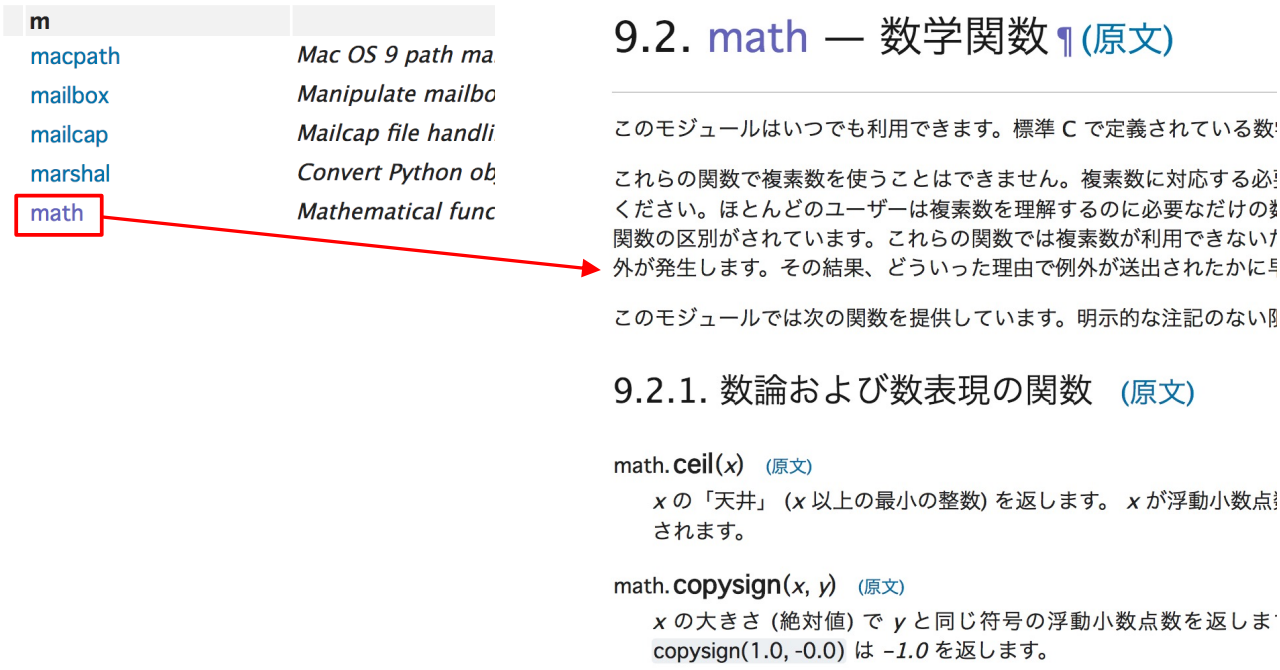
  - 例

```
from math import sqrt as sq
a = sq(2)
print(a)
```

sqrt関数をsqとしてインポート

標準モジュール

- 標準モジュールの一覧
  https://docs.python.jp/3/py-modindex.html



サードパーティー製のモジュールの一例

- 標準モジュール以外にも、Pythonで利用できる様々なモジュールがインターネット上で公開されています。

モジュール名	概要
matplotlib	グラフなどの可視化モジュール
seaborn	matplotlibの見栄えをより綺麗にするモジュール
NumPy	計算を効率的に行うためのモジュール
SymPy	数式・記号計算用モジュール
pandas	データ解析支援モジュール（Excelファイルの読み書きが可能）
openpyxl	Excelファイルの読み書きに特化したモジュール
xlwings	Excelアプリを直接制御できるモジュール
scipy	NumPyを利用した数値解析モジュール
scikit-learn	機械学習モジュール
NetworkX	グラフ・ネットワーク計算と可視化モジュール
Basemap	地図描画モジュール

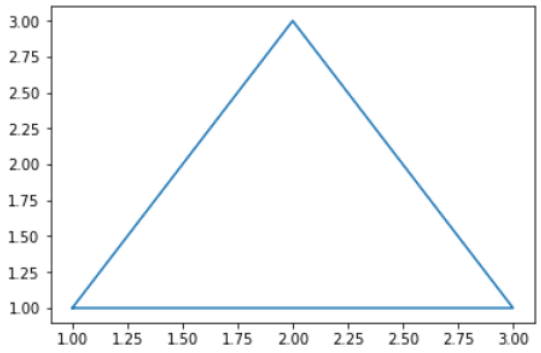
matplotlib

- matplotlibはPythonでグラフを描画するための可視化モジュールで、複数のモジュールから構成されています
  - matplotlib.pyplot･･･グラフ描画モジュール
  - matplotlib.patches･･･図形描画モジュール
  - matplotlib.animation･･･アニメーション描画モジュール
  - ･･･

- ファイルを新規作成し、折線グラフを描画するmatplotlib.pyplotのplot関数を使ってみる

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

折れ線グラフを描画するplot関数　x座標　y座標



マーカーや軸ラベルの設定

- マーカーや軸ラベルを設定してみます

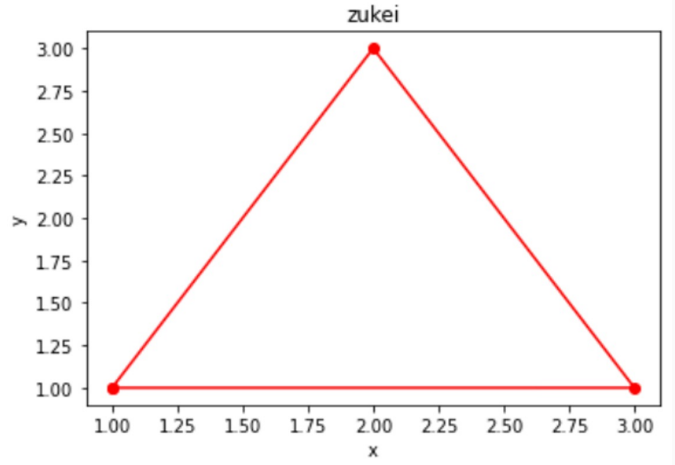
```
import matplotlib.pyplot as plt

plt.plot([1,2,3,1], [1,3,1,1], color="red", marker="o")
plt.title("zukei")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

線の色　マーカー

タイトルや軸ラベルの設定

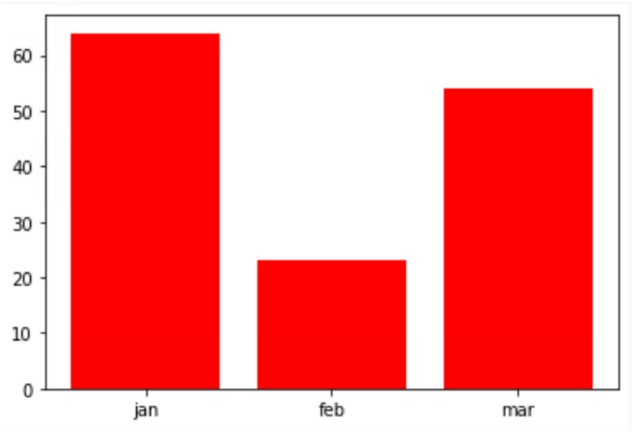
描画を確定させる



棒グラフ

- matplotlib.pyplotのbar関数で縦棒グラフが描画できます

```
x = ["jan", "feb", "mar"]
y = [64, 23, 54]
plt.bar(x, y, color="red")
plt.show()
```



- bar関数を、barh関数に変更することで、横棒グラフが描画できます

円グラフ

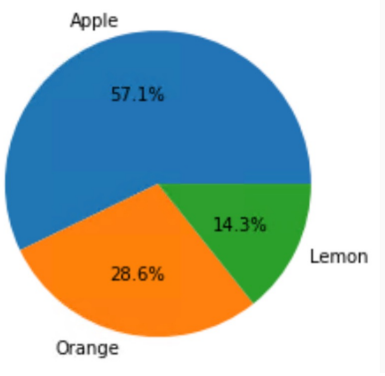
- matplotlib.pyplotのpie関数で円グラフが描画できます

```
x = ["Apple", "Orange", "Lemon"]
y = [200, 100, 50]
plt.pie(y)
plt.show()
```

- labels引数でラベルを表示できます
- autopct引数で%を表示できます

```
x = ["Apple", "Orange", "Lemon"]
y = [200, 100, 50]
plt.pie(y, labels=x, autopct="%1.1f%%")
plt.show()
```

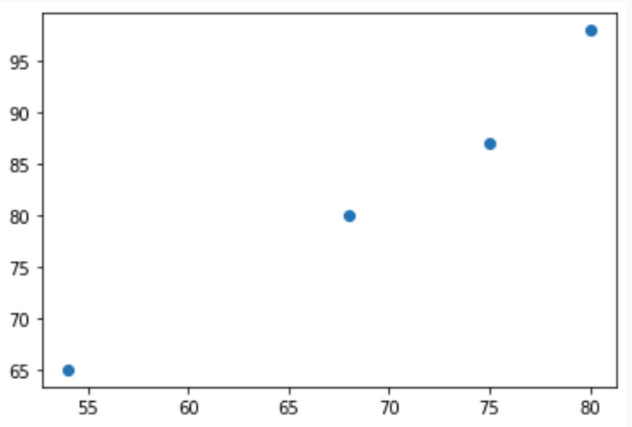
.1fは小数点以下1桁を表している



散布図(1)

- matplotlib.pyplotのscatter関数で散布図が描画できます
  - 散布図とは、x軸とy軸に異なる量を割り当てて、データがあてはまる座標に点(サンプルポイント)を打って表したグラフで、2つの量の関係を図示することができます

```
eigo = [80, 54, 75, 68]
sugaku = [98, 65, 87, 80]
plt.scatter(eigo, sugaku)
plt.show()
```

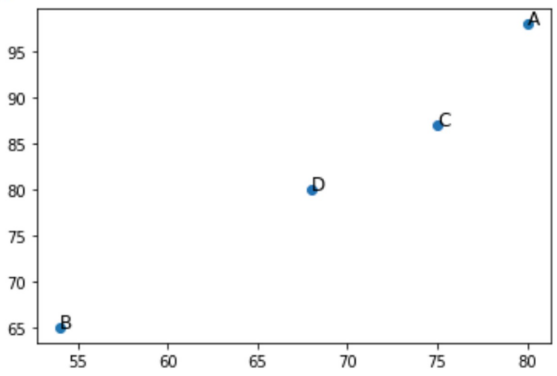


散布図(2)

- 散布図の各サンプルポイントにラベルを表示するには？
  - matplotlib.pyplotの、text関数を使用します
    例:
    - plt.text(x座標, y座標, ラベル, fontsize=フォントサイズ)

```
eigo = [80, 54, 75, 68]
sugaku = [98, 65, 87, 80]
label = ["A", "B", "C", "D"]
plt.scatter(eigo, sugaku)
for data in zip(eigo, sugaku, label):
    plt.text(data[0], data[1], data[2], fontsize=12)
plt.show()
```

zip関数とは
複数のリストを結合する関数
eigo, sugaku, labelの先頭から[80, 98, "A"][54, 65, "B"]・・・と順に取り出すことができる

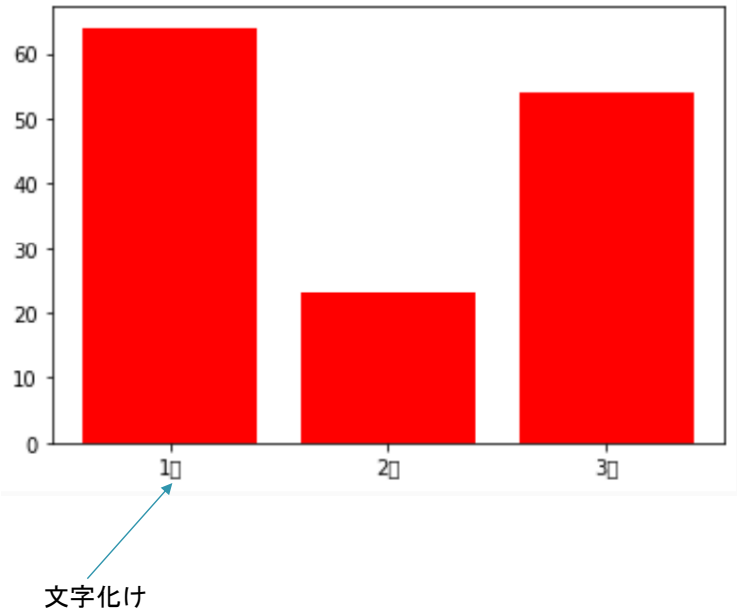


(補足)グラフにおける日本語の文字化け

- Spyderでは、グラフのラベルなどに日本語が含まれると日本語フォントの設定が無いため文字化けしてしまいます。

```
import matplotlib.pyplot as plt

x = ["1月", "2月", "3月"]
y = [64, 23, 54]
plt.bar(x, y, color="red")
plt.show()
```



(補足)グラフにおける日本語の文字化け

- matplotlibの日本語フォントの文字化け対策
  - 独立行政法人情報処理推進機構が配布している日本語フォントをインストールする(IPAフォント)
    - https://moji.or.jp/ipafont/ipaex00301/ からIPAフォント(ipaexg00301.zip)をダウンロードする。
    - ダウンロードしたファイルを展開し、ipaexg.ttfをエクスプローラを使って C:¥ProgramData¥Anaconda3¥Lib¥site-packages¥matplotlib¥mpl-data¥fonts¥ttf に入れる。上記フォルダが無い場合は、 C:¥Users¥自分のユーザアカウント名¥anaconda3¥Lib¥site-packages¥matplotlib ¥mpl-data¥fonts¥ttf に入れる。
    - spyderを終了する。
    - [スタート]-[コンピュータ]を開き、C:¥Users¥自分のユーザアカウント名 の .matplotlibフォルダ を削除する
    - spyderを再起動する。

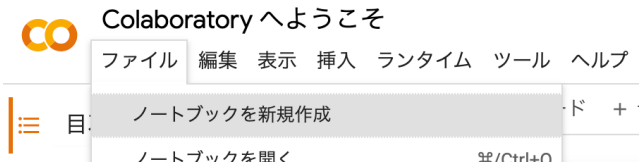
インストールレスのPython環境の紹介（Google Colaboratory）

- Webブラウザを使ってPythonプログラムの作成と実行ができます
  - 要：Googleアカウント
- Edge/Chrome/Safariを起動する
  - google claboratory で検索する
  - 下記リンクを開く

https://colab.research.google.com › notebooks › intro

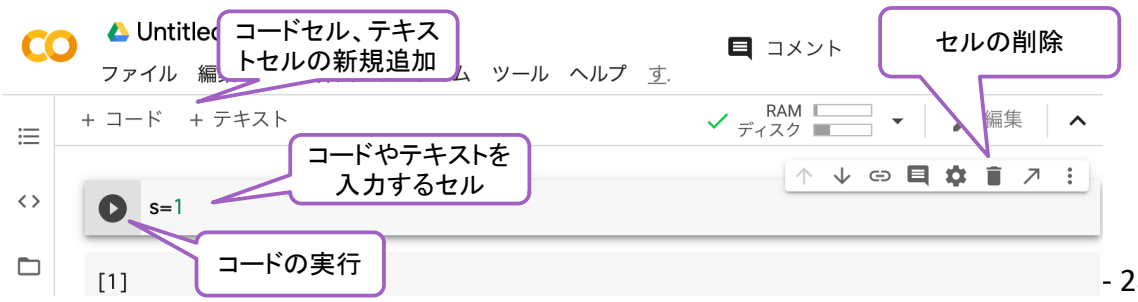
Colaboratory - Google Colab

  - 「ファイル」「ノートブックを新規作成」



ノートブック

- Google Colaboratoryでは、ノートブックという単位でプログラムを記述します。
- セルという枠にコードを入力します。
  - コード以外にもテキスト（メモ書き）を入力するセルを追加できます。
- コードを実行するには、セル左の ▶ ボタンか、下記のキーを押します。
  - 実行：Ctrl+Enter
  - 実行と新たなセルの追加：Shift+Enter



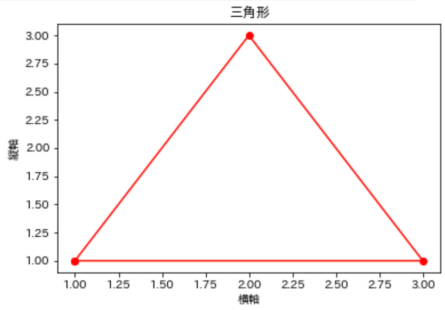
Google Colaboratoryの文字化け対策

- matplotlibで日本語を使えるようにするモジュール「japanize-matplotlib」をインストールする。

japanize-matplotlibのインストール

```
!pip install japanize-matplotlib
import matplotlib.pyplot as plt
import japanize_matplotlib

plt.plot([1, 2, 3, 1], [1, 3, 1, 1], color="red", marker="o")
plt.title("三角形")
plt.xlabel("横軸")
plt.ylabel("縦軸")
```



課題4-1

- p7のcalBMI関数では、身長の単位はm（メートル）で指定する仕様になっていました。
- 私たちの普段の生活では、身長はcmで扱ったほうがわかりやすいので、shinchoをcmで指定できるようにcalBMI関数を修正してみてください。

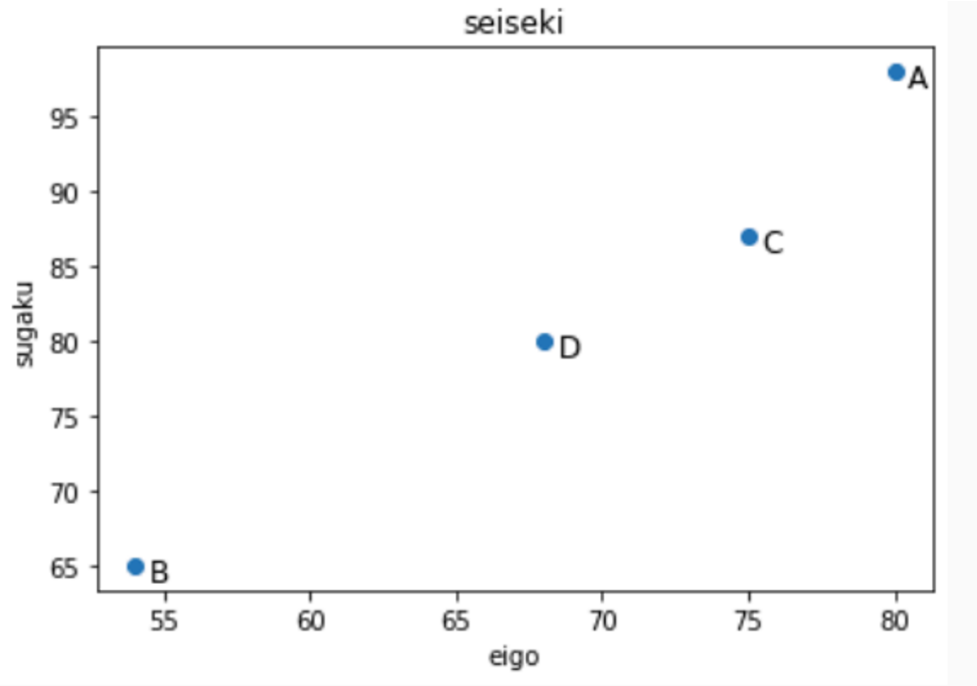
```
a, b = calBMI(60, 1.6)
```

身長をmでなくcm で指定できるように！

```
a, b = calBMI(60, 160)
```

課題4-2

- 散布図が、下記の表示となるようにp24のコードを修正してください



課題4-3

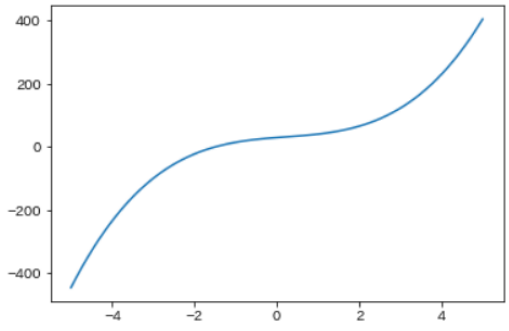
- matplotlibを用いて、

$$-5 \leq x \leq 5$$

において、

$$y = 3x^3 - 2x^2 + 10x + 30$$

のグラフを描いてください。




```
import matplotlib.pyplot as plt
xp = []
yp = []
for x in range(-50, 50):
    xp.append(x * 0.1)
    yp.append(ここを考える)

plt.plot(xp, yp)
plt.show()
```