ae-2.Python プログラミングとデータ分析の基礎：Trinket を活用した実践的入門

（AI演習）（全１５回）

https://www.kkaneko.jp/ai/ae/index.html

金子邦彦

CC BY NC SA

人工知能（AI）の学習のための指針

1. **実践重視**： AIツールを実際に使用し、機能に慣れる

2. **エラーを恐れない**： 実行においては、エラーの発生の可能性がある．エラーを恐れず，むしろ学習の一部として捉えるポジティブさが大切.

3. **段階的学習**：基礎から応用へと段階的に学習を進め，AIの可能性を前向きに捉える

アウトライン

1. Python プログラミングの基礎
2. 変数，代入
3. オブジェクト，メソッド
4. プログラミングの楽しさと達成感
5. Python の外部ライブラリの概要と活用
6. 実データを用いた散布図の作成
7. 人工知能でのプログラミングの重要性

Python

- **Python** は多くの人々に利用されている**プログラミング言語**の１つ
- **読みやすさ，書きやすさ，幅広い応用範囲**が特徴

```
   from keras.models import Sequentia
   model = Sequential()
   from keras.layers import Dense, Ac

   model.add(Dense(units=64, input_di
   model.add(Activation('relu'))
   model.add(Dense(units=max(set(y_tr
   model.add(Activation('softmax'))
   model.compile(loss='sparse_categor
                 optimizer='sgd',
                 metrics=['accuracy']
   model.fit(x_train, y_train, epochs
   score=model.evaluate(x_test, y_tes
   print(score)
   model.predict(x_test)
   model.summary()
och 1/200
3 [==============================] - 0s
3200
och 2/200
[==============================] - 0s
200
ch 3/200
[==============================] - 0s
0
```

Python 言語が広く使用されている理由

文法のシンプルさ

- **Python** は，**直感的で読みやすい**文法
- 例えば、**print** で簡単に**出力**できる、**if** や **else** で**条件分岐**、**for** や **while** で**繰り返し（ループ）**

拡張性

- **多岐にわたる分野で利用**が可能
- 例えば、**関数やクラスを定義**するための **def** や **class**、**継承**や**オブジェクトの属性名と値**を操作するための **super** や **vars** などがある。

柔軟性

- シンプルなスクリプトも、高度なプログラムも作成可能
- **オブジェクト指向**の機能を持ち、**__init__** や **self** のようなキーワードを使用して**クラス**を利用できる。

trinket

- **Trinket** は**オンライン**の **Python**、**HTML** 等の**学習サイト**
- ブラウザで動作
- 有料の機能と無料の機能がある
- **自分が作成した Python プログラムを公開し、他の人に実行してもらうことが可能**（そのとき、書き替えて実行も可能）

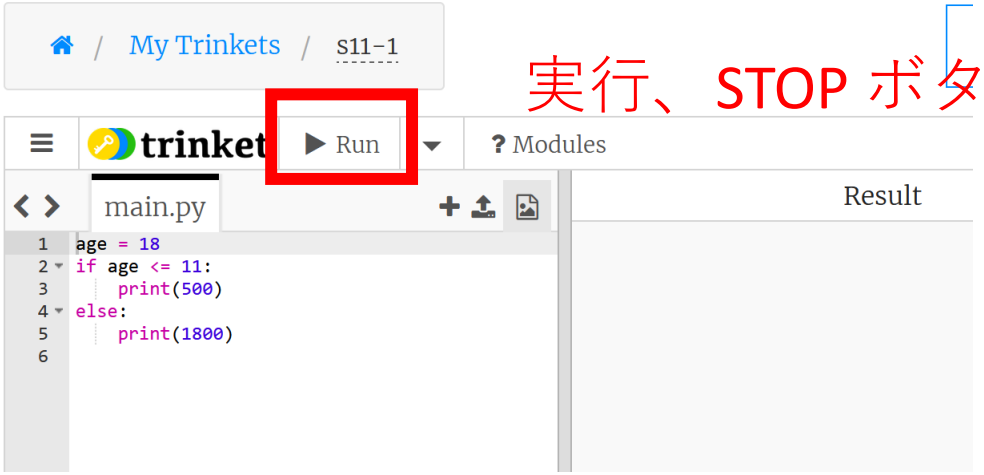
- **Python の標準機能**を登載、その他、次の外部ライブラリがインストール済み

matplotlib.pyplot, numpy, processing, pygal

trinket でのプログラム実行

- **trinket は Python, HTML などのプログラムを書き実行できる**サイト
- https://trinket.io/python/0fd59392c8

のように、違うプログラムには違う URL が割り当てられる



ソースコードの
メイン画面

実行結果

- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

2-1. Python プログラミングの基礎

変数，代入

- **変数**：プログラム内で名前を付けて利用する**オブジェクト**で，**値を保存**し，後から**参照**できる

- **代入**：「**x = 100**」のように書くことで，**x という名前の変数に、値 100 が保存**される

例

```
x = 100
```

オブジェクトとメソッド

- **オブジェクト**：コンピュータでの**操作や処理の対象となるもの**

t.goto(0,100)	**t　オブジェクト** **goto(0,100)　メソッド** 間を「.」で**区切って**いる

- **メソッド: オブジェクト**に属する機能や操作．オブジェクトがもつ能力に相当する
- **引数：メソッド**が行う操作の詳細に関する情報．**メソッド**呼び出しのときに、引数を指定できる

t.goto(0,100)

引数

Python プログラムの書き方（コーディングスタイル）

- **インデント（字下げ）によるブロックの区切り**
  （通常「タブ」または「４つの半角スペース」）
- **コメント**：行の先頭に「#」
- **空白行**：プログラムを読みやすくするために挿入可能

例

```
import turtle

for i in range(4):
    turtle.forward(100)
    turtle.right(90)

turtle.done()
```

空白行

インデント（字下げ）

空白行

2-2. 変数，代入

変数，代入

- **変数**：プログラム内で名前を付けて利用する**オブジェクト**で，**値を保存**し，後から**参照**できる

- **代入**：「**x = 100**」のように書くことで，**x という名前の変数に、値 100 が保存**される

例

```
x = 100
```

trinket

- **Trinket** は**オンライン**の **Python**、**HTML** 等の**学習サイト**
- ブラウザで動作
- 有料の機能と無料の機能がある
- **自分が作成した Python プログラムを公開し、他の人に実行してもらうことが可能**（そのとき、書き替えて実行も可能）

- **Python の標準機能**を登載、その他、次の外部ライブラリがインストール済み

matplotlib.pyplot, numpy, processing, pygal

trinket でのプログラム実行

- **trinket** は **Python, HTML などのプログラムを書き実行できる**サイト
- https://trinket.io/python/0fd59392c8

のように、違うプログラムには違う URL が割り当てられる

- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

演習．変数と代入

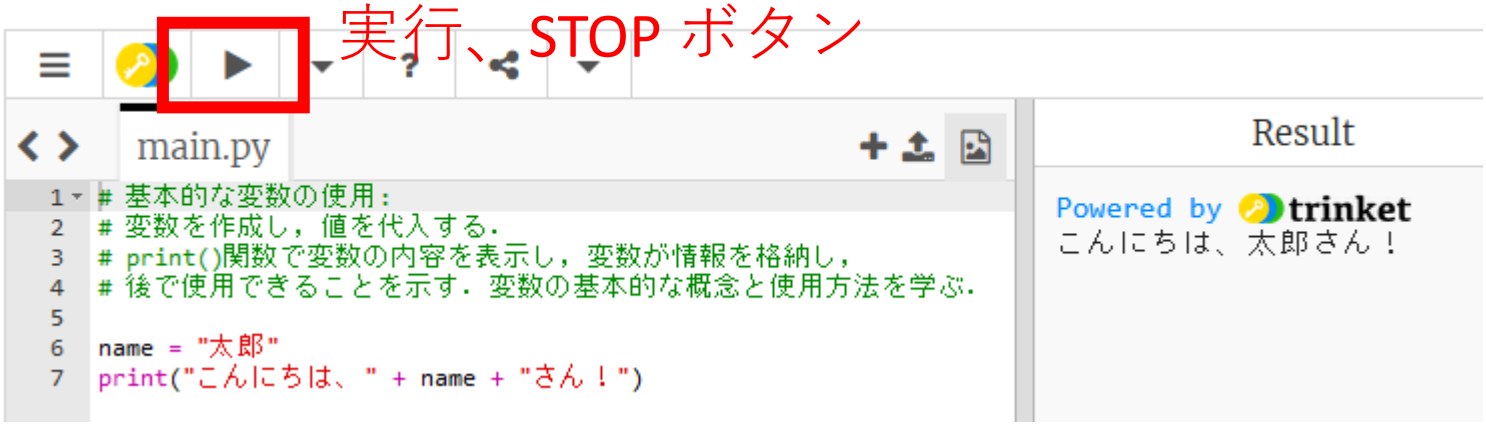
1．基本的な変数の使用

① trinket の次のページを開く

https://trinket.io/python/abafd851480a

- 変数
- 値を代入する方法
- print() 関数を使って変数の内容を表示する方法

② 実行結果が，次のように表示されることを確認



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

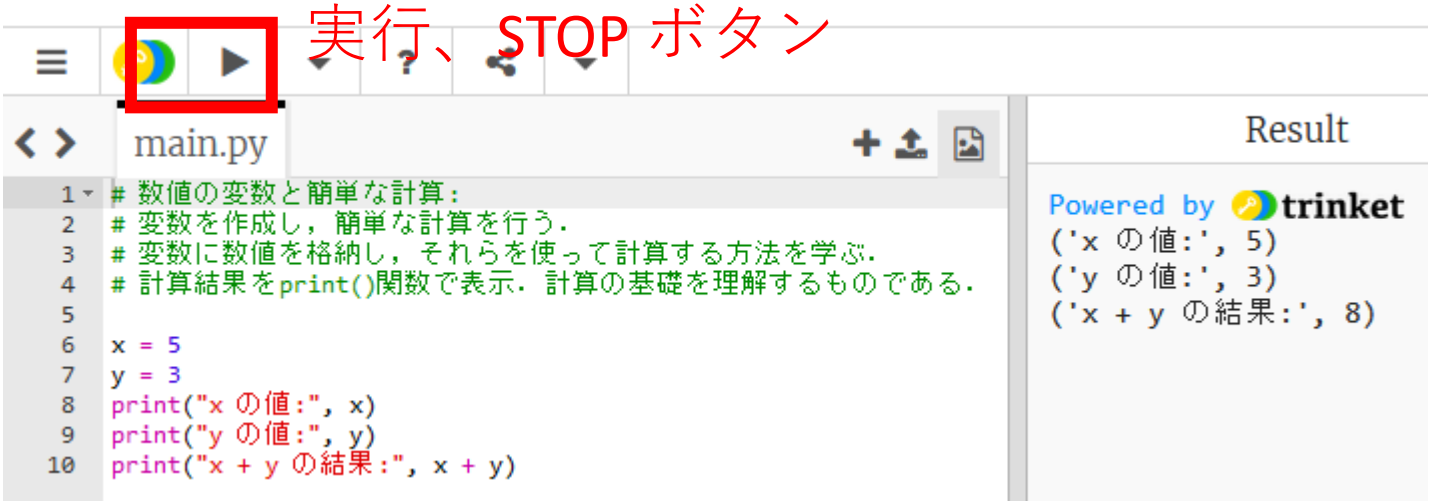
2．基本的な変数の使用

① trinket の次のページを開く

https://trinket.io/python/9870e86d63b9

- 複数の変数を使って簡単な計算を行う
- print() 関数を使って変数の内容を表示する

② 実行結果が，次のように表示されることを確認



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

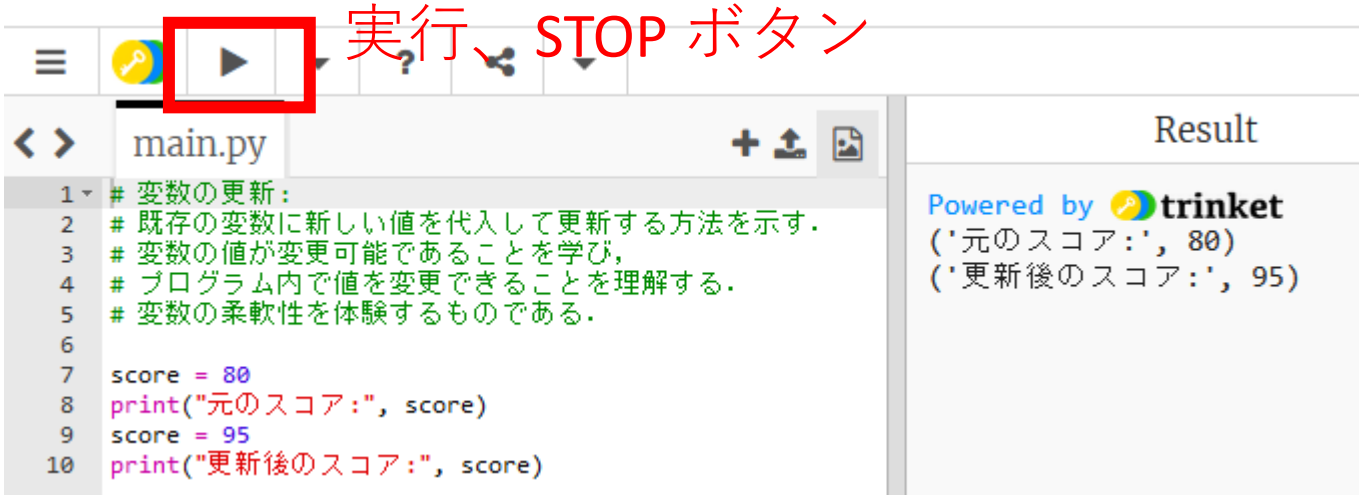
3．変数の更新

① trinket の次のページを開く

https://trinket.io/python/b869619b0874

- 既存の変数に新しい値を代入して更新
- 変数の値が変更可能

② 実行結果が，次のように表示されることを確認



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

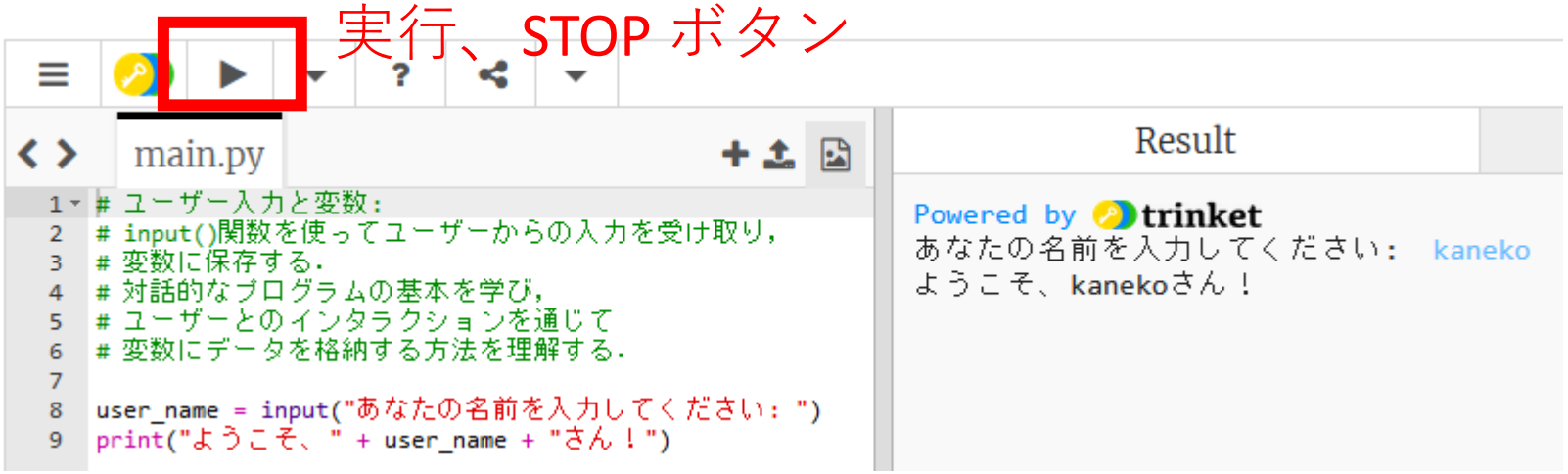
4．ユーザー入力と変数

① trinket の次のページを開く

https://trinket.io/python/45f0bed92360

- input() 関数を使ってユーザーからの入力を受け取る
- ユーザー入力を変数に保存
- 対話的なプログラムの基本

② 実行結果が，次のように表示されることを確認



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

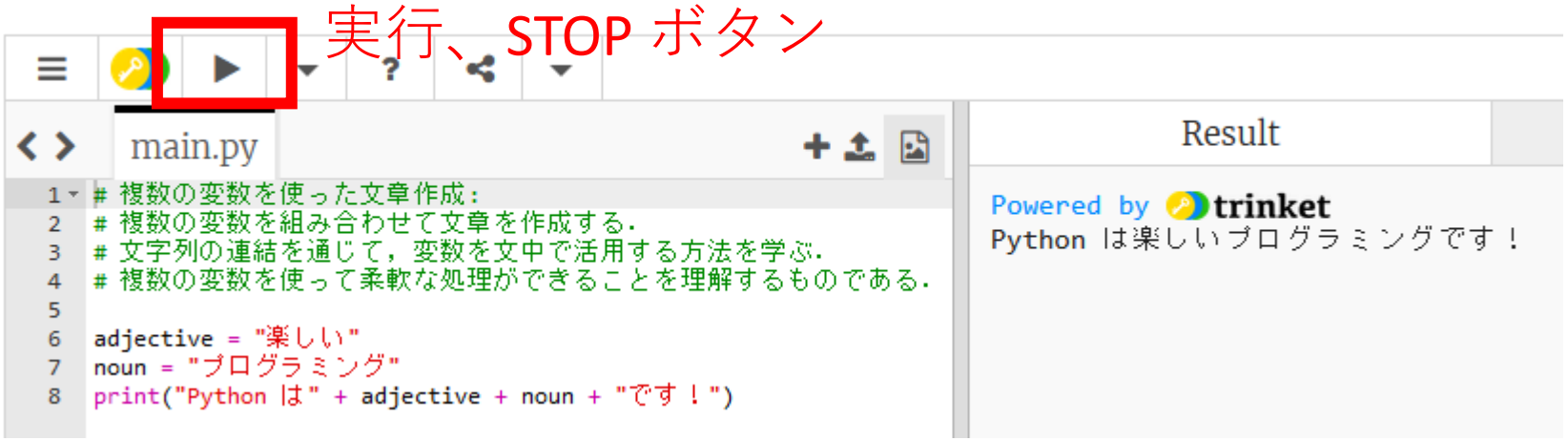
5．複数の変数

① trinket の次のページを開く

https://trinket.io/python/abafd851480a

- 複数の変数を組み合わせて使用
- 表示の際に，２つの変数の値を表示

② 実行結果が，次のように表示されることを確認



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

2-3. オブジェクト，メソッド

オブジェクトとメソッド

- **オブジェクト**：コンピュータでの**操作や処理の対象となるもの**

t.goto(0,100)	**t　オブジェクト** **goto(0,100)　メソッド** 間を「.」で**区切って**いる

- **メソッド: オブジェクト**に属する機能や操作．オブジェクトがもつ能力に相当する
- **引数：メソッド**が行う操作の詳細に関する情報．**メソッド**呼び出しのときに、引数を指定できる

t.goto(0,100)

引数

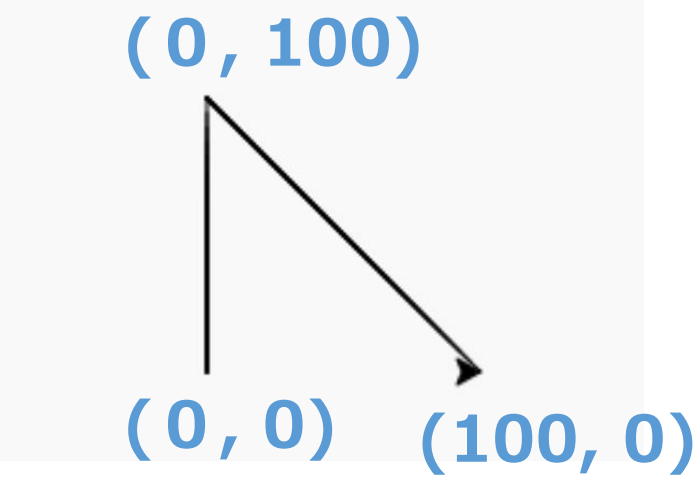
タートルグラフィックス

カーソルを使って絵を描く

- **タートルグラフィックス**を用いた演習により、**プログラムによって図形を描画**する．それを通して、プログラムの**動作を視覚的に理解**
- 論理的思考力や課題解決力の向上にもつながる

```
1  import turtle
2  t = turtle.Turtle()
3  t.goto(0,100)
4  t.goto(100,0)
```

タートルグラフィックスの機能をインポートする「import turtle」が必要



タートルグラフィックス入門

```
1  import turtle
2  t = turtle.Turtle()
3  t.goto(0,100)
4  t.goto(100,0)
```

オブジェクト　メソッド

- **メソッド**は、オブジェクトが持つ機能を呼び出すためのもの
- 「**goto**」は**指定した座標への移動**

主なメソッド

- **goto**（＜横方向の値＞，<縦方向の値>）　**移動**
- **forward**(＜移動量＞)　**前進**
- **backword**(＜移動量＞)　**後退**
- **right**(＜角度＞)　**右回りに回転**
- **left**(＜角度＞)　**左回りに回転**

オブジェクトとメソッド

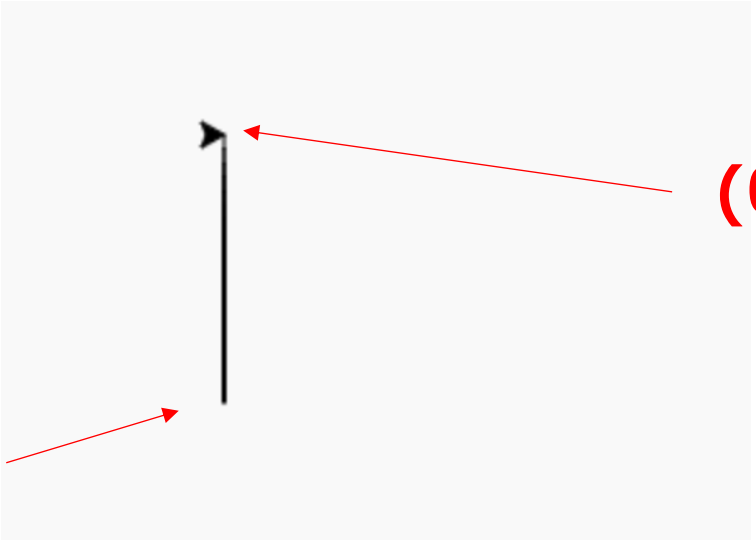
```
import turtle

t = turtle.Turtle()

t.goto(0,100)
```

インポート

オブジェクト生成。t へのセット。

(0, 100) への移動

実行結果
最初の位置は
(0, 0)



オブジェクトが動く

オブジェクトとメソッド

```
import turtle

t = turtle.Turtle()

t.goto(0,100)

t.goto(100,0)
```

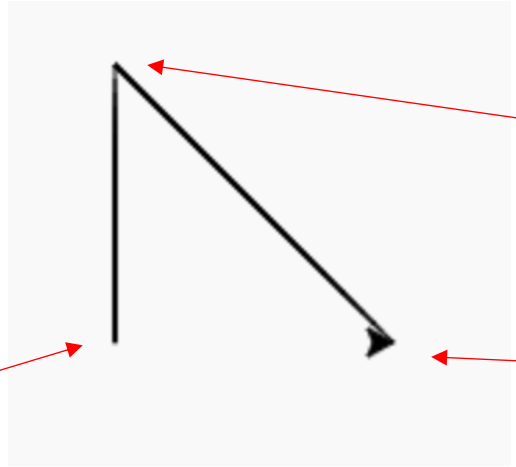
インポート

オブジェクト生成。t へのセット。

(0, 100) への移動

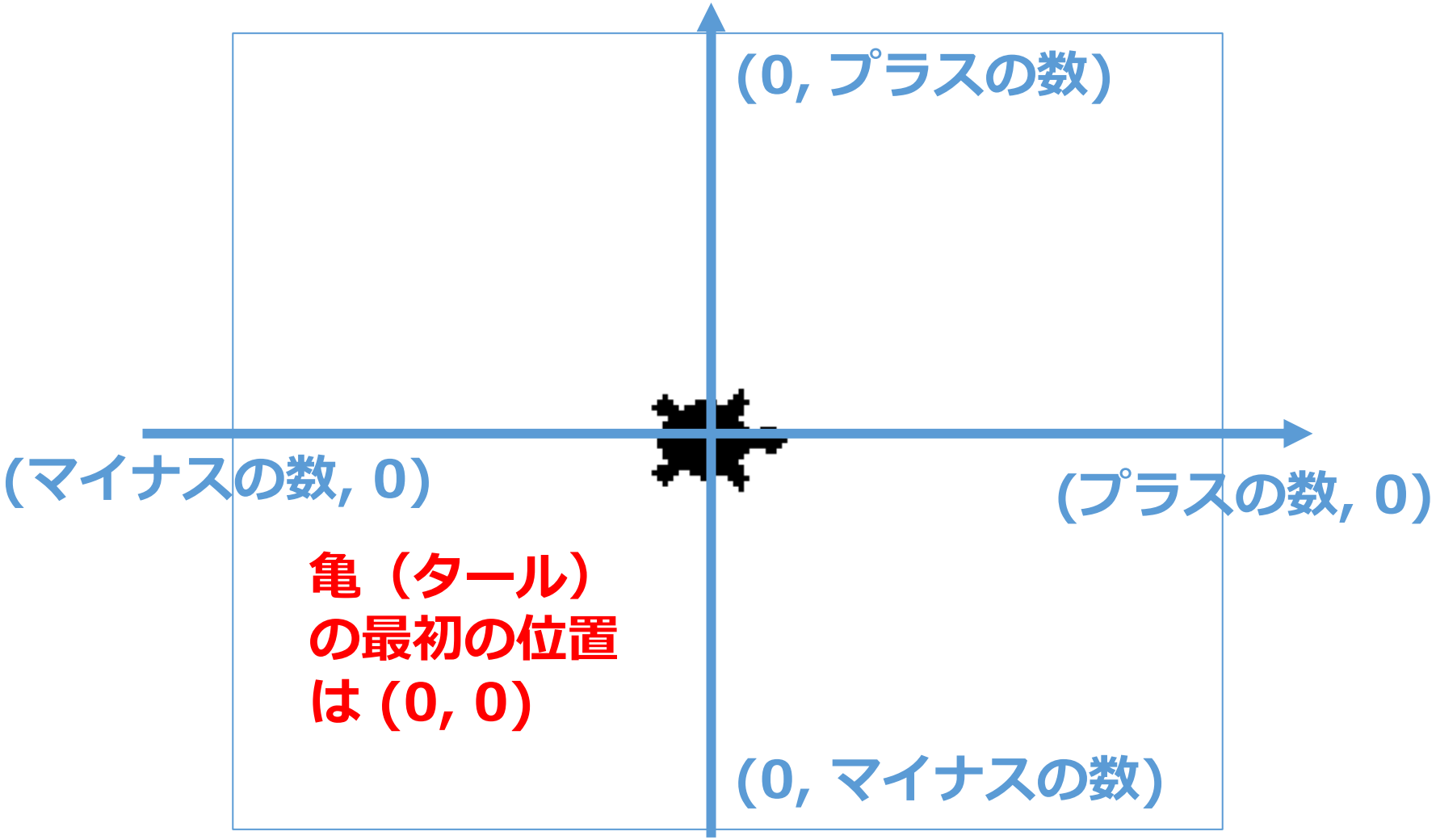
(100, 0) への移動

実行結果
最初の位置は
(0, 0)



オブジェクトが動く

メソッド goto の引数となる
縦方向の値と，横方向の値



演習．タートルグラフィックス

```
import turtle

t = turtle.Turtle()

t.goto(0,100)

t.goto(100,0)
```

ライブラリのインポート

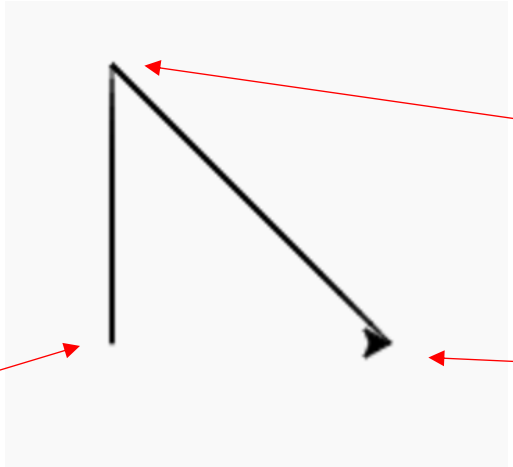
オブジェクト生成。t へのセット。

(0, 100) への移動

(100, 0) への移動

実行結果

最初の位置は
(0, 0)



(0, 100)

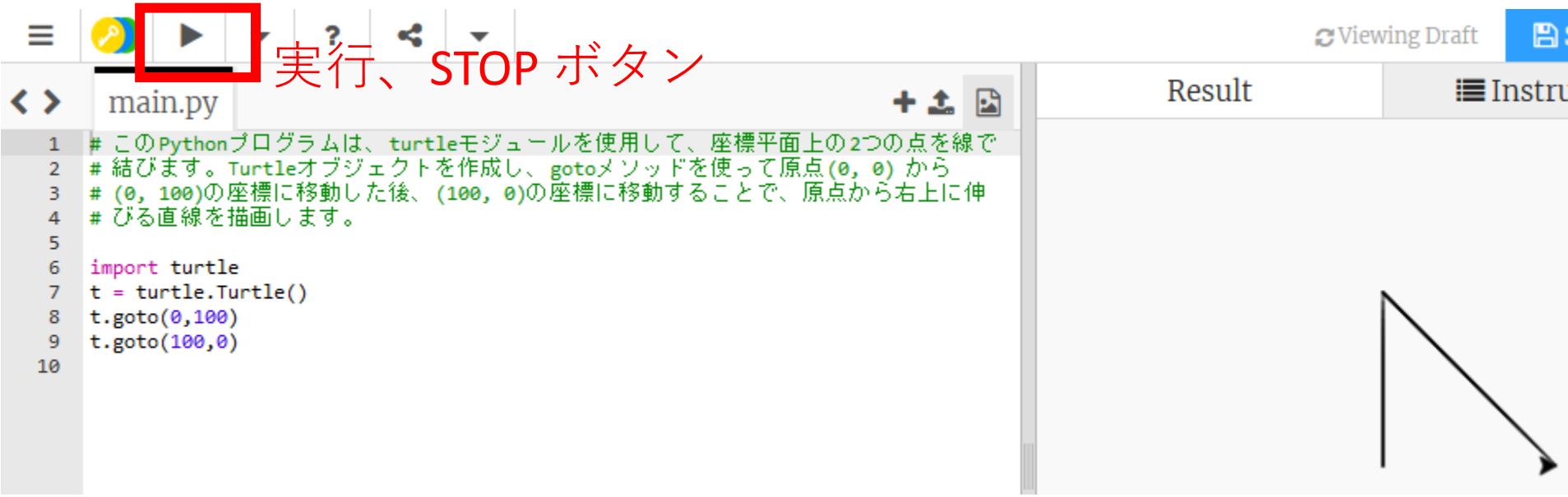
(100, 0)

(0, 0)

① trinket の次のページを開く

https://trinket.io/python/f29bfe71cd

② 実行結果が，次のように表示されることを確認

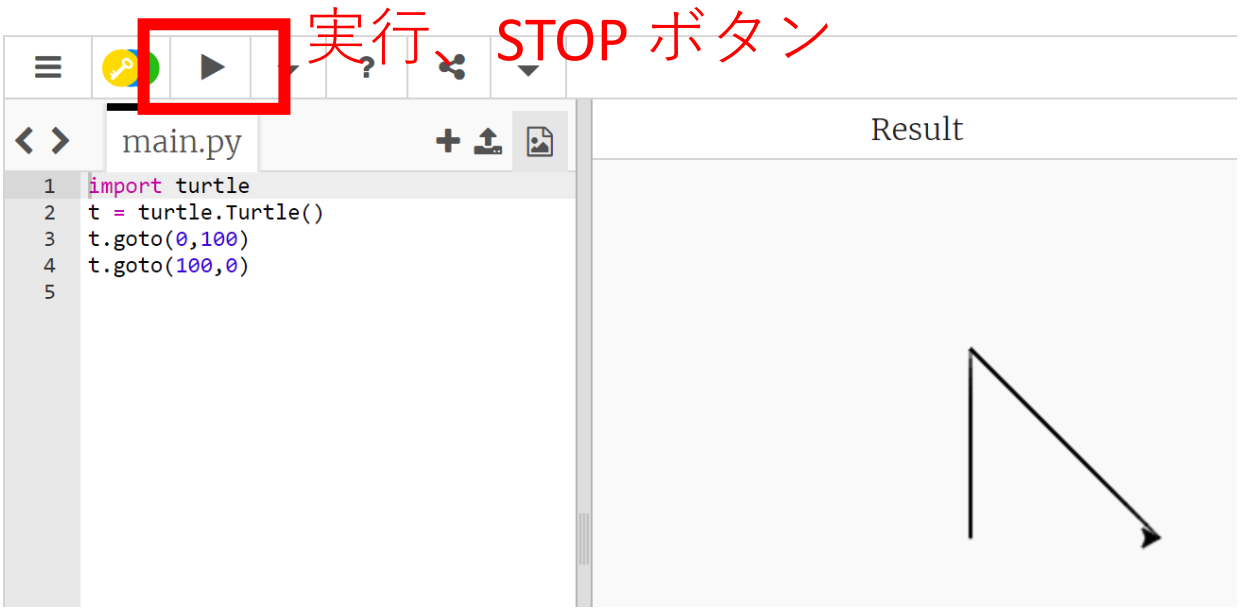


- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

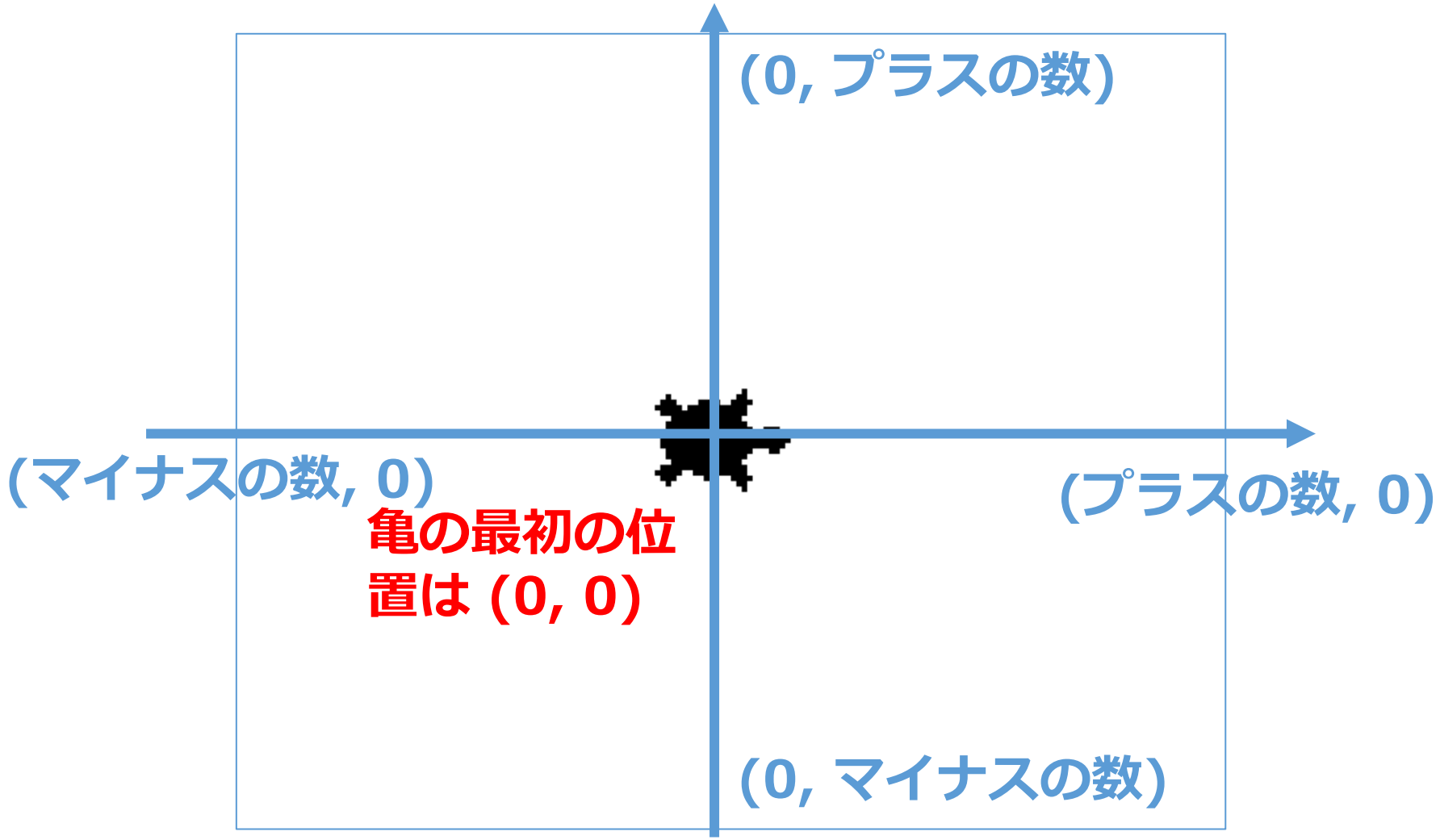
③ 次のように**書き換えて**、**再実行**し、**結果が変わる**ことを確認

```
import turtle
t = turtle.Turtle()
t.goto(0,100)
t.goto(200,0)
```

書き換え



④ 次を参考に、**自分で引数を書き替えたり**、プログラム内に**「t.goto(<引数>)」を増やして**、**思い通りの図形**を目指す。



ここまでのまとめ

- **変数**：プログラム内で**名前を付けて値を保存し参照できるオブジェクト**
- **オブジェクト**：コンピュータでの**操作や処理の対象となるもの**である.
- **メソッド**：**オブジェクトが持つ機能や操作**を表すもの.

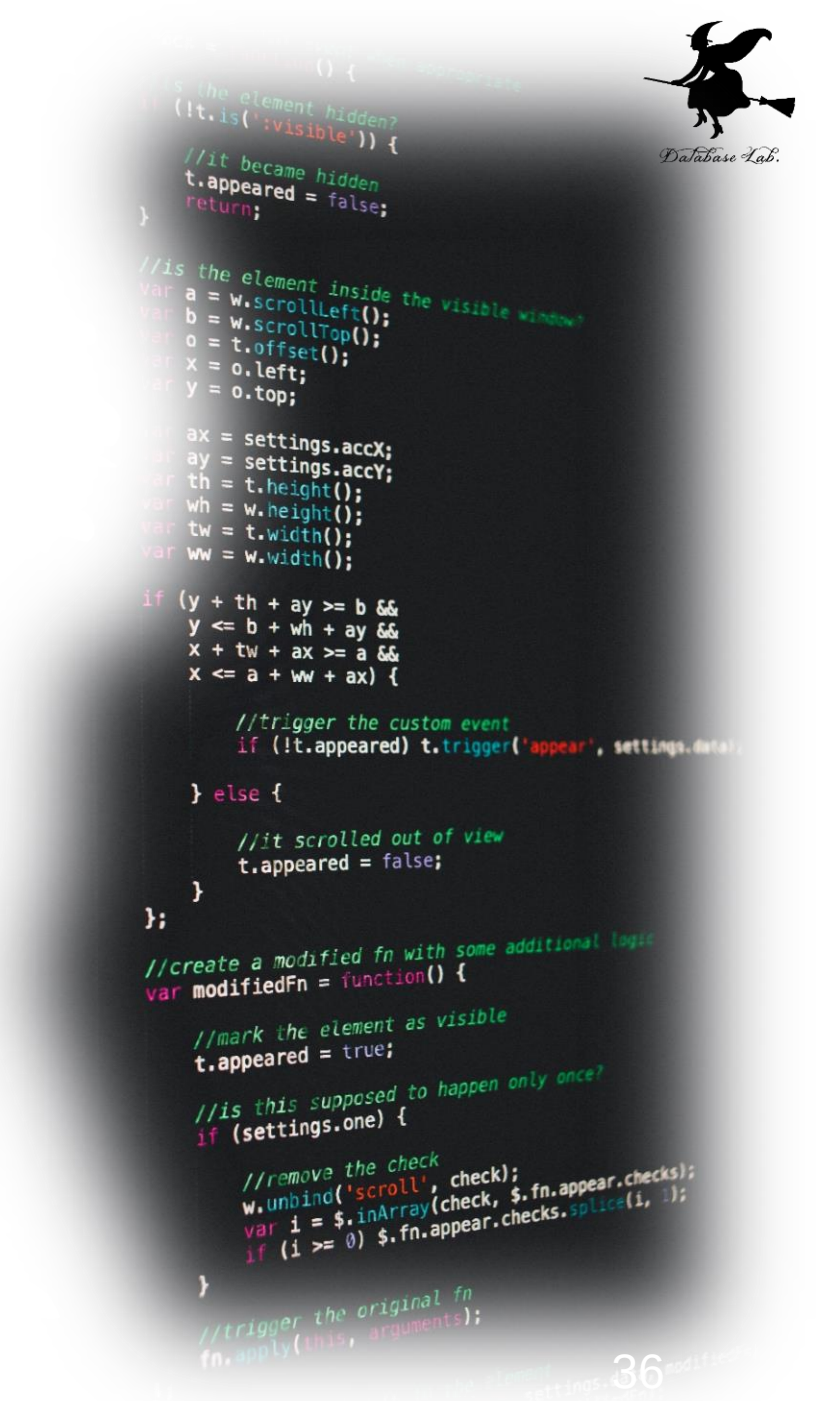
  例: t.goto(100,0)では **t はオブジェクト**、**goto はメソッド**。
- **引数**：メソッドには，**操作の詳細を指定**するための引数を設定できる.

  例：t.goto(100,0) では、引数は「100,0」
- Python では，**インデント**を適切に使用する.
- **プログラムの修正や改良を繰り返し行う**ことで，**目的の動作を実現**できる.

2-4. プログラミングの楽しさと達成感

ソースコード

- **ソースコード**は，**プログラミング言語**で書かれた**プログラム**のもの
- 人間も**読み書き**，**編集**できる
- **ソースコード**により，**プログラムの動作を理解**し，必要に応じて**改変**できる

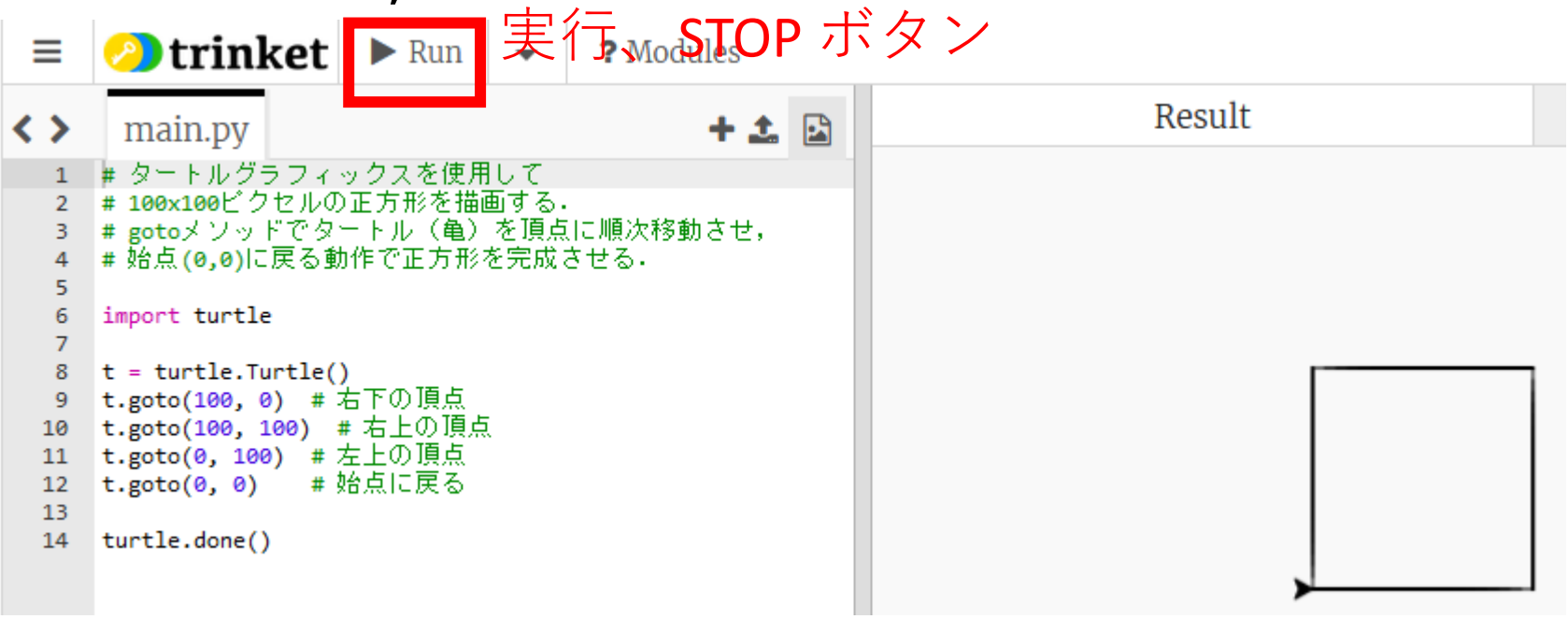


演習
図形のバリエーション

① 別のプログラムを試す．trinket の次のページを開く

https://trinket.io/python/035810ce8d49

② 実行結果が，次のように表示されることを確認

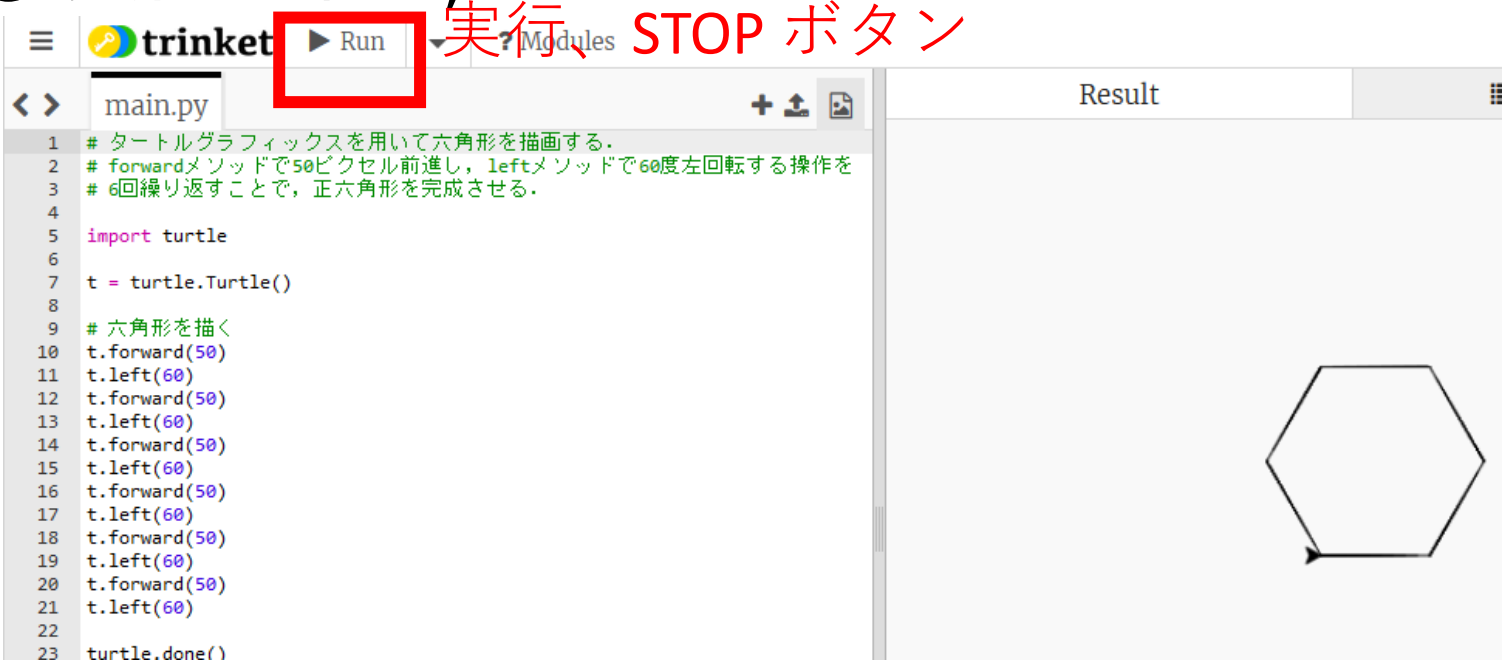


- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

③ 別のプログラムを試す．trinket の次のページを開く

https://trinket.io/python/ddb861147133

④ 実行結果が，次のように表示されることを確認



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

演習
プログラミングの楽しさと達成感

各自の自発的な演習、自己研鑽の時間

① Pythonでグラフィックスを描く

資料のプログラムを動かし理解を深める

② Pythonの基本を押さえる

オブジェクト、メソッド、引数

③ 発想力、創造力

turtleライブラリを使用して、**あなた自身がデザインした図形を描く**。

④ 自主性、自己研鑽力、自分なりに工夫したことを振り返る

説明されなかった機能（他の図形の書き方）などを自主的に**調べ、理解し、自分で試してみる**。そして、**自分なりに工夫したこと**を振り返り、省察することで、さらに実力アップ。

①複雑な図形

https://trinket.io/python/5366def2f4

②色，円

https://trinket.io/python/f8cd554693

① https://trinket.io/python/5366def2f4

```
import turtle
t = turtle.Turtle()
t.goto(0, 100)
t.goto(58, -80)
t.goto(-95, 30)
t.goto(95, 30)
t.goto(-58, -80)
t.goto(0, 100)
```

ライブラリのインポート

オブジェクト生成。t へのセット。

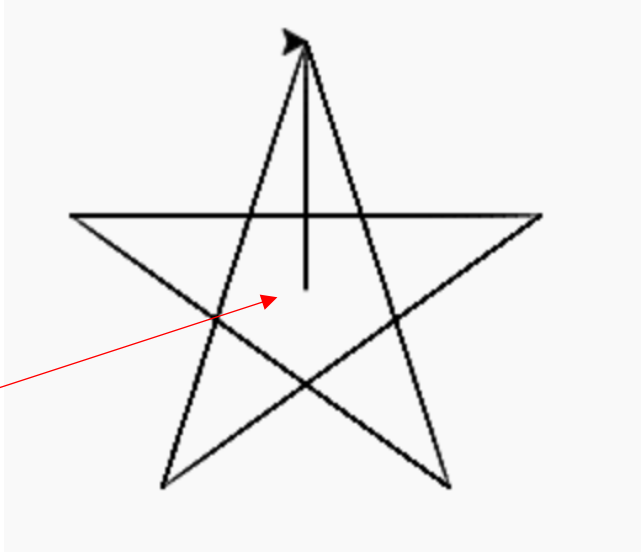
移動

実行結果

最初の位置は

(0, 0)

(0, 0)



② https://trinket.io/python/f8cd554693

```
import turtle

t = turtle.Turtle()

colors = ["red", "green", "blue"]

for i in range(3):

    t.color(colors[i])

    t.circle(30)

    t.forward(50)
```

ライブラリのインポート

オブジェクト生成。t へのセット。

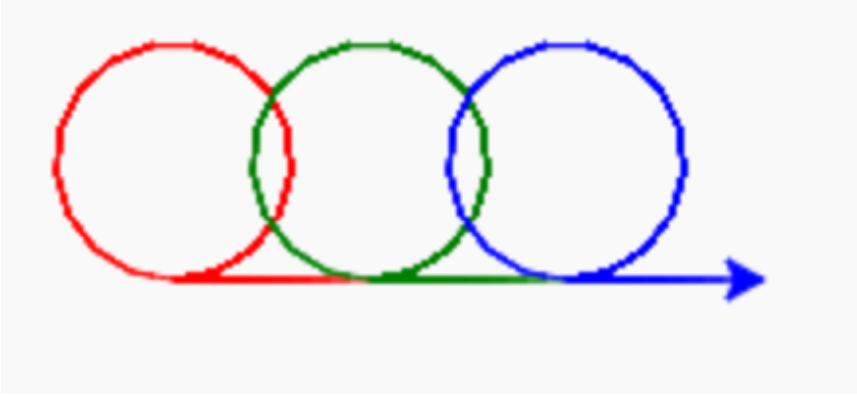
色は、赤、緑、青

色を変える

半径３０の円

前に50進む

実行結果



2-5. Python の外部ライブラリの概要と活用

外部ライブラリとは

- プログラミング言語の**標準機能以外**の**追加機能を提供するプログラム**集.　（外部）
- さまざまな目的のために**何度も利用できる**　（ライブラリ）

外部ライブラリの利用

- **インストール**（使えるようにするための操作）が必要

  操作コマンド例： pip install numpy

- Trinket では，**いくつかの外部ライブラリがインストール済み**

  matplotlib.pyplot, numpy, processing, pygal

- 外部ライブラリの利用には，**プログラム内でのインポートが必要**

  Python では import を使用

  例：import numpy as np

外部ライブラリ活用のメリット

- プログラム作成の手間を削減
- プログラムの量を削減
- 高度な機能の利用
- 効率的に動作する外部プログラムの利用

外部ライブラリの活用はソフトウェア開発に不可欠．効率的で高機能なプログラム作成を可能にする．

代表的な外部ライブラリ例

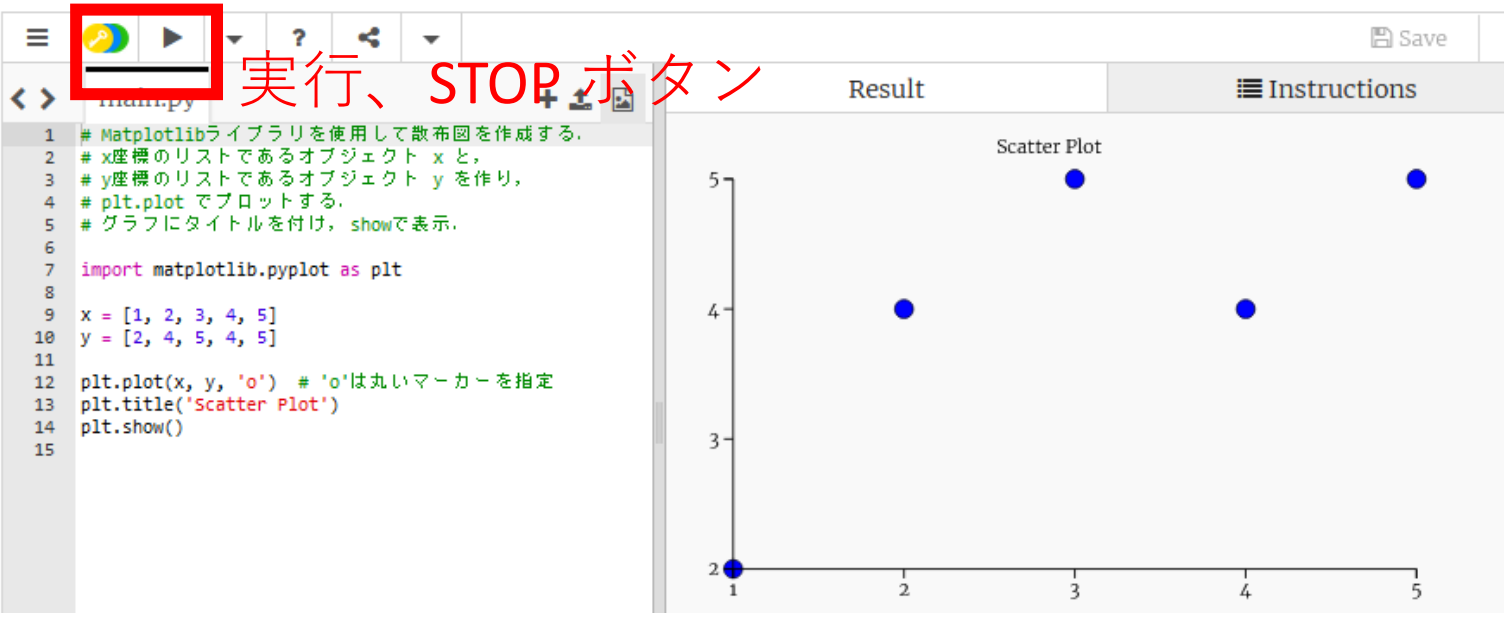
- NumPy：数値計算、配列
- Matplotlib.Pyplot：グラフ、図表作成

演習．外部ライブラリの活用例

① trinket の次のページを開く

https://trinket.io/python/a563124a187c

② 実行結果が，次のように表示されることを確認

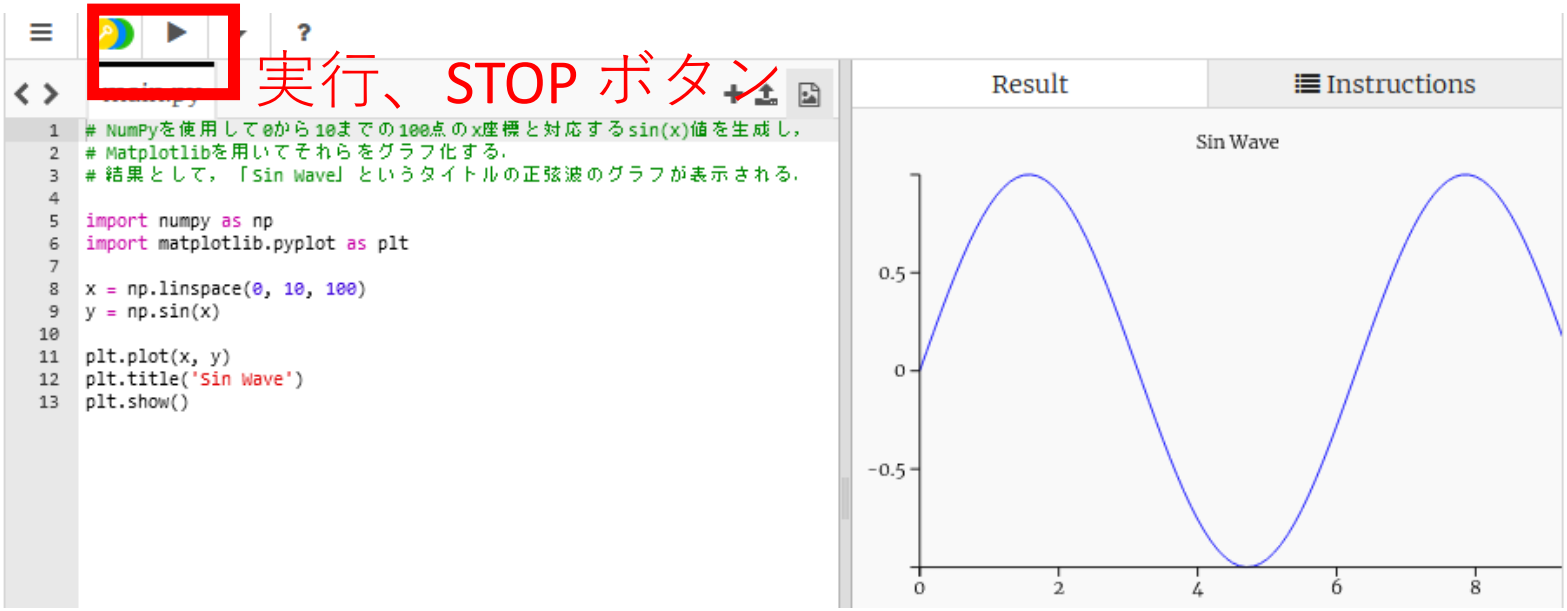


- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

③ trinket の次のページを開く

https://trinket.io/python/5830b6d18e9c

④ 実行結果が，次のように表示されることを確認



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

⑤ trinket の次のページを開く

https://trinket.io/python/9fe4ad1bb348

- 乱数は，予測不可能な数値のこと。
- 様々な用途（シミュレーション、ゲーム、暗号化など）で使用される。

⑥ 実行結果が，次のように表示されることを確認

- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

プログラム内でのインポート

- 例：import numpy as np

代表的な外部ライブラリ

- NumPy：数値計算、配列処理
- Matplotlib.Pyplot：グラフ、図表作成

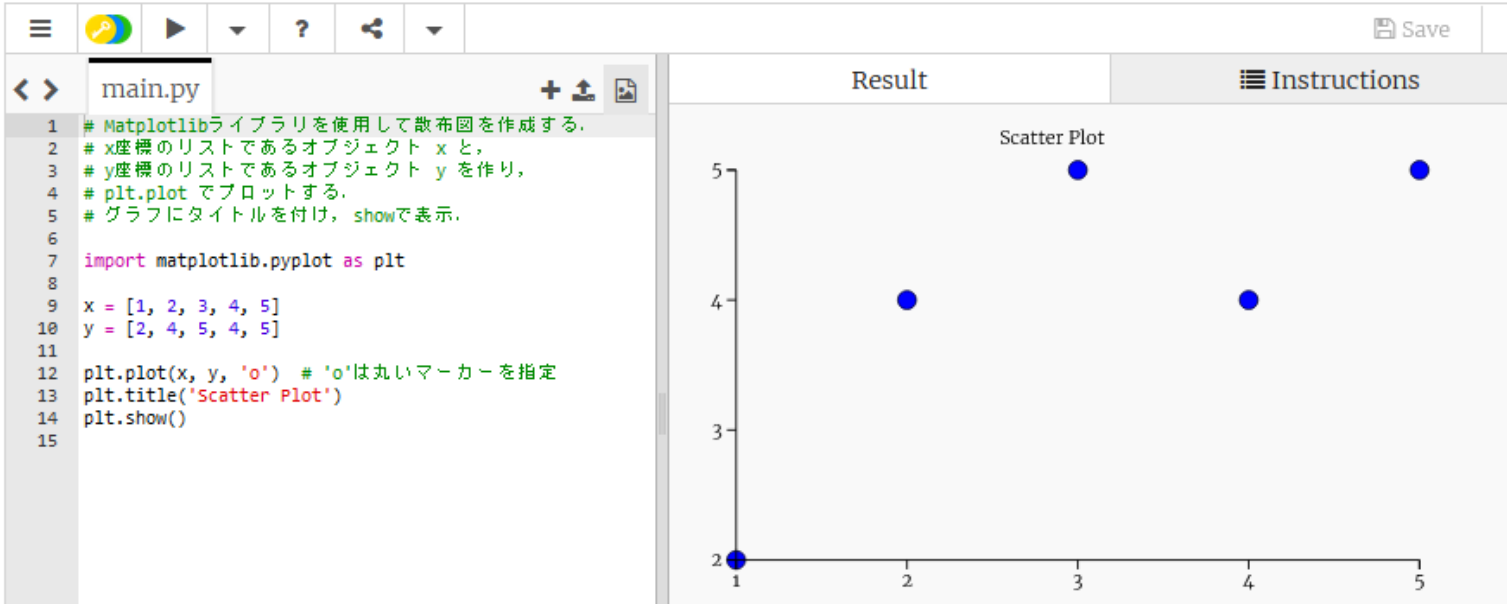
ここまでのまとめ

- **外部ライブラリ**は，プログラミング言語の標準機能以外の**追加機能を提供するプログラム集**である.
- 外部ライブラリの利用には，**インストール**と**プログラム内でのインポート**が必要である.
- 外部ライブラリを活用することで，**プログラム作成の手間と量を削減**できる.
- 外部ライブラリにより，**高度な機能**を持ち，**効率的に動作するプログラム**を利用できる.
- 外部ライブラリの活用は，**効率的で高機能なプログラム開発に不可欠**である.

2-6. 実データを用いた散布図の作成

散布図作成プログラムの例

import matplotlib.pyplot as plt	インポート
x = [1, 2, 3, 4, 5]	データの準備（xとy）
y = [2, 4, 5, 4, 5]	
plt.plot(x, y, 'o’)	プロット
plt.title('Scatter Plot’)	グラフのタイトル設定
plt.show()	グラフの表示



- import matplotlib.pyplot as pltmatplotlib.pyplot
  matplotlib.pyplot ライブラリを plt という別名でインポートする.
- x = [1, 2, 3, 4, 5]
  x座標のデータを含むリストを作成
- y = [2, 4, 5, 4, 5]
  y座標のデータを含むリストを作成
- plt.plot(x, y, 'o’)
  xとyのデータポイントをプロット。‘o’は丸いマーカー（点）を指定
- plt.title('Scatter Plot’)
  グラフのタイトルを「Scatter Plot」に設定
- plt.show()
  作成したグラフを表示

プログラムを用いて散布図を作成することのメリット

- **データ処理の自動化**：

大量のデータセットを自動的に処理し、散布図を生成できる.

- **再現性の向上**：

同じ散布図を簡単に再現できる.

- **カスタマイズの柔軟性**：

グラフの細かな部分までカスタマイズできる.

プログラムを用いた散布図作成は、特に研究や大規模データ分析の場面で有効

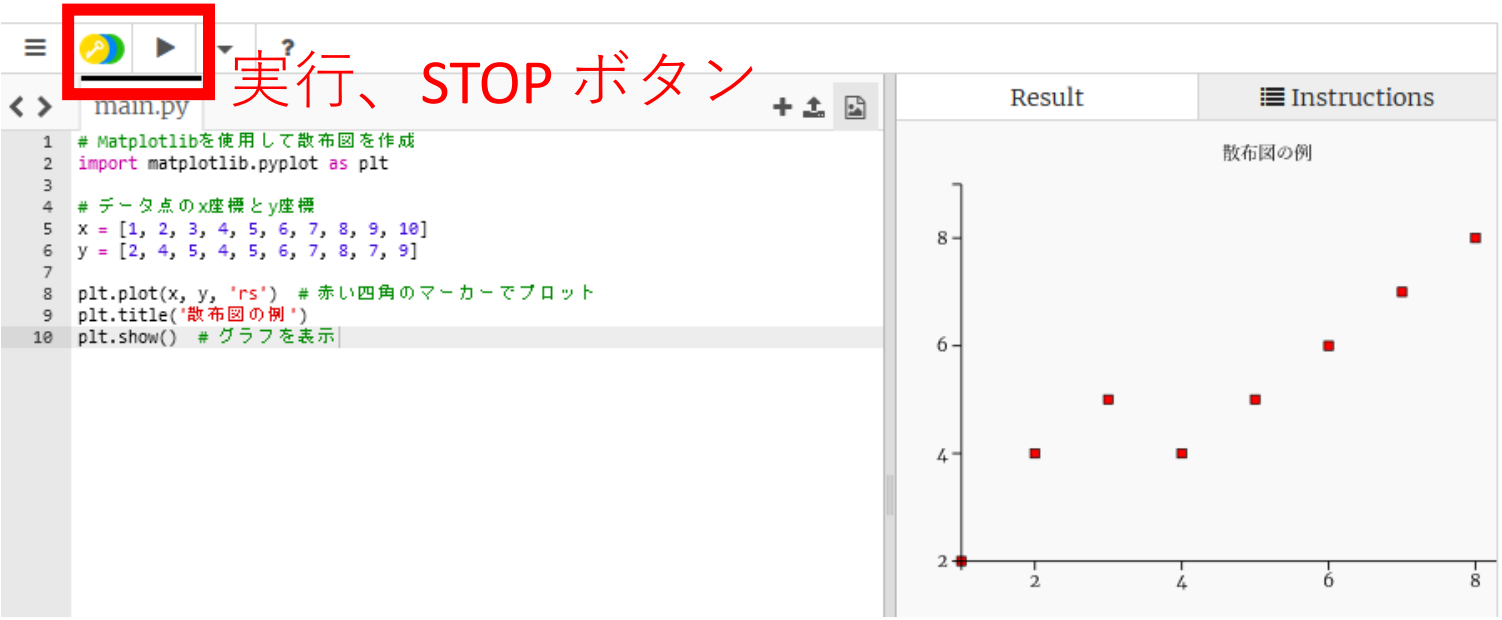
演習．散布図

```
# Matplotlibを使用して散布図を作成
import matplotlib.pyplot as plt

# データ点のx座標とy座標
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = [2, 4, 5, 4, 5, 6, 7, 8, 7, 9]

plt.plot(x, y, 'rs')  # 赤い四角のマーカーでプロット
plt.title('散布図の例')
plt.show()  # グラフを表示
```

① trinket の次のページを開く

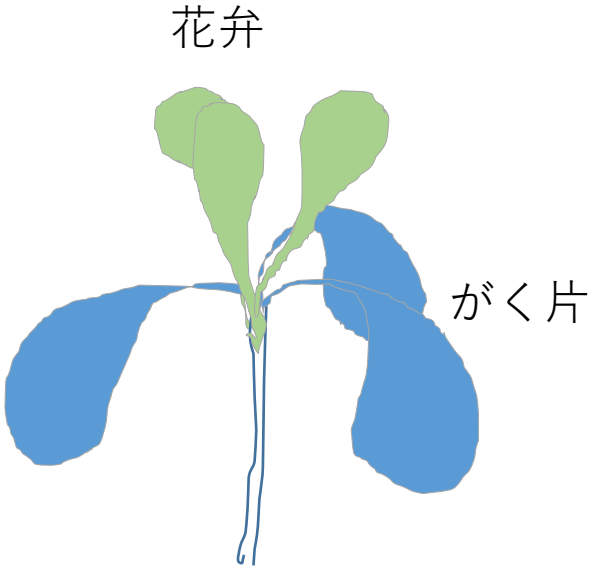
https://trinket.io/python/625038de4886

② 実行結果が，次のように表示されることを確認



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

アヤメ属 (Iris)



- 多年草
- 世界に 150種. 日本に 9種.
- **がく片**　Sepal

  3個（大型で下に垂れる）
- **花弁**　Petal

  3個（直立する）

Iris データセット

【概要】

- 有名なデータセット.
- 3種類のアヤメ（Iris）の花の特徴を記録したもの.

【データの内容】

- 150個のサンプル（各種類50個ずつ）.
- 4つの特徴（**がく片**の長さと幅、**花弁**の長さと幅）.
- 3つの種類（品種：setosa、versicolor、virginica）.

Iris データセット

Iris データセット

１５０サンプルのうち先頭１０

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa

がく片(Sepal)
の長さと幅

花弁(Petal)
の長さと幅

種類

演習５．実データの散布図

- Irisデータセットを散布図にする

  １つめ：がく片の長さと幅

  ２つめ：花弁の長さと幅
- Matplotlib ライブラリを使用
- 長さと幅の全体の分布を一目で把握できる

がく片の長さと幅

```
# Matplotlibを使用して散布図を作成
import matplotlib.pyplot as plt

# Irisデータセット
x = [5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.8, 4.8, 4.3, 5.8, 5.7, 5.4, 5.1, 5.7, 5.1, 5.4, 5.1, 4.6, 5.1, 4.8, 5.0,
5.0, 5.2, 5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5.0, 5.5, 4.9, 4.4, 5.1, 5.0, 4.5, 4.4, 5.0, 5.1, 4.8, 5.1, 4.6, 5.3, 5.0,
    7.0, 6.4, 6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5.0, 5.9, 6.0, 6.1, 5.6, 6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6, 6.8,
6.7, 6.0, 5.7, 5.5, 5.5, 5.8, 6.0, 5.4, 6.0, 6.7, 6.3, 5.6, 5.5, 5.5, 6.1, 5.8, 5.0, 5.6, 5.7, 5.7, 6.2, 5.1, 5.7,
    6.3, 5.8, 7.1, 6.3, 6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4, 6.8, 5.7, 5.8, 6.4, 6.5, 7.7, 7.7, 6.0, 6.9, 5.6, 7.7, 6.3, 6.7, 7.2, 6.2,
6.1, 6.4, 7.2, 7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6.0, 6.9, 6.7, 6.9, 5.8, 6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 5.9]

y = [3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4, 3.0, 3.0, 4.0, 4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3.0,
3.4, 3.5, 3.4, 3.2, 3.1, 3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.6, 3.0, 3.4, 3.5, 2.3, 3.2, 3.5, 3.8, 3.0, 3.8, 3.2, 3.7, 3.3,
    3.2, 3.2, 3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2.0, 3.0, 2.2, 2.9, 2.9, 3.1, 3.0, 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3.0, 2.8,
3.0, 2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3.0, 3.4, 3.1, 2.3, 3.0, 2.5, 2.6, 3.0, 2.6, 2.3, 2.7, 3.0, 2.9, 2.9, 2.5, 2.8,
    3.3, 2.7, 3.0, 2.9, 3.0, 3.0, 2.5, 2.9, 2.5, 3.6, 3.2, 2.7, 3.0, 2.5, 2.8, 3.2, 3.0, 3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2, 2.8,
3.0, 2.8, 3.0, 2.8, 3.8, 2.8, 2.8, 2.6, 3.0, 3.4, 3.1, 3.0, 3.1, 3.1, 3.1, 2.7, 3.2, 3.3, 3.0, 2.5, 3.0, 3.4, 3.0]

# すべてのデータポイントを同じ色とマーカーでプロット
plt.plot(x, y, color='blue', marker='o')

plt.xlabel('がく片の長さ (cm)')
plt.ylabel('がく片の幅 (cm)')
plt.title('アヤメのがく片の大きさ')

plt.show()  # グラフを表示
```

がく片の長さと幅

① trinket の次のページを開く

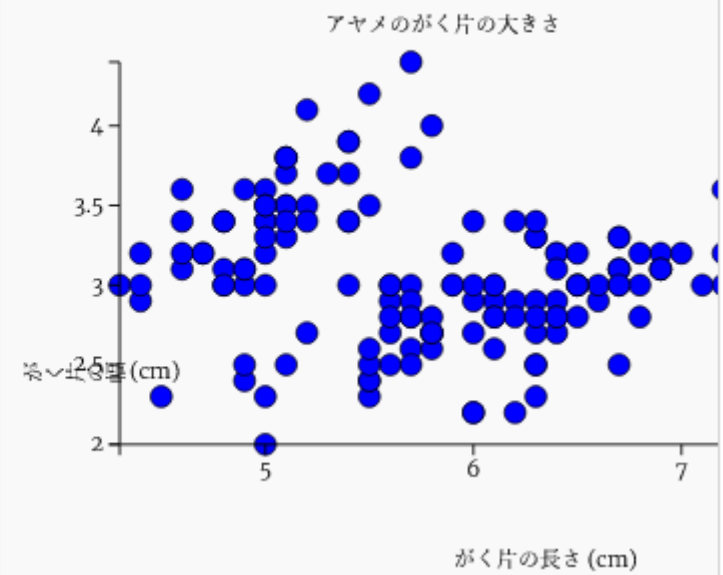
https://trinket.io/python/41dee25e267f

② 実行結果が，次のように表示されることを確認

main.py

```python
# Matplotlibを使用して散布図を作成
import matplotlib.pyplot as plt

# Irisデータセット
x = [5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.8, 4.8, 4.3, 5.8,
     7.0, 6.4, 6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5.0, 5.9, 6.0, 6.1, 5.6,
     6.3, 5.8, 7.1, 6.3, 6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4, 6.8, 5.7, 5.8,

y = [3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4, 3.0, 3.0, 4.0,
     3.2, 3.2, 3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2.0, 3.0, 2.2, 2.9, 2.9,
     3.3, 2.7, 3.0, 2.9, 3.0, 3.0, 2.5, 2.9, 2.5, 3.6, 3.2, 2.7, 3.0, 2.5, 2.8,

# すべてのデータポイントを同じ色とマーカーでプロット
plt.plot(x, y, color='blue', marker='o')

plt.xlabel('がく片の長さ (cm)')
plt.ylabel('がく片の幅 (cm)')
plt.title('アヤメのがく片の大きさ')

plt.show()  # グラフを表示
```

Result　Instructions



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

花弁の長さと幅

③ trinket の次のページを開く

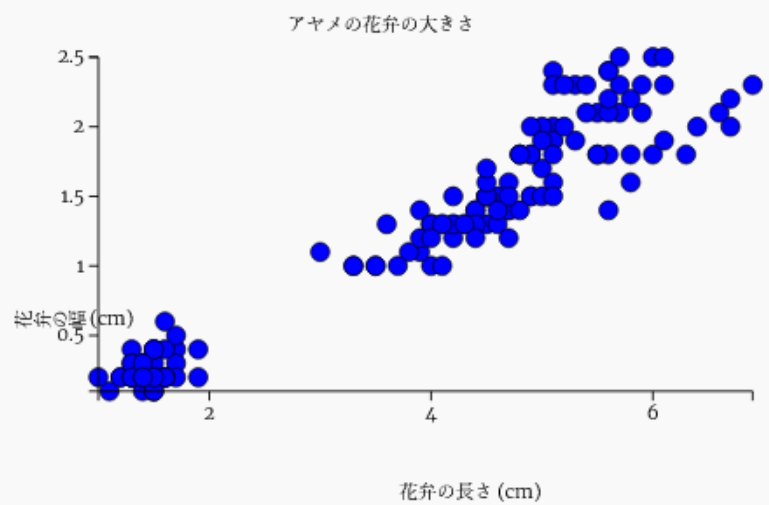
https://trinket.io/python/8f33f7399c2d

④ 実行結果が，次のように表示されることを確認

main.py

```python
# Matplotlibを使用して散布図を作成
import matplotlib.pyplot as plt

# Irisデータセット（花弁の長さと幅）
x = [1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.6, 1.4, 1.1
     4.7, 4.5, 4.9, 4.0, 4.6, 4.5, 4.7, 3.3, 4.6, 3.9, 3.5, 4.2, 4.0, 4.7
     6.0, 5.1, 5.9, 5.6, 5.8, 6.6, 4.5, 6.3, 5.8, 6.1, 5.1, 5.3, 5.5, 5.0

y = [0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.2, 0.1, 0.1
     1.4, 1.5, 1.5, 1.3, 1.5, 1.3, 1.6, 1.0, 1.3, 1.4, 1.0, 1.5, 1.0, 1.4
     2.5, 1.9, 2.1, 1.8, 2.2, 2.1, 1.7, 1.8, 1.8, 2.5, 2.0, 1.9, 2.1, 2.0

# すべてのデータポイントを同じ色とマーカーでプロット
plt.plot(x, y, color='blue', marker='o', linestyle='')

plt.xlabel('花弁の長さ (cm)')
plt.ylabel('花弁の幅 (cm)')
plt.title('アヤメの花弁の大きさ')

plt.show()  # グラフを表示
```

Result



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

ここまでのまとめ

- Matplotlib.pyplot は**グラフ作成**に使用できる外部ライブラリ.
- プログラム次第で，**異なる散布図の表示**も可能.
- **外部ライブラリ**の活用により，**プログラム作成の手間を削減**でき，**高度な機能**を利用することができる.

2-7. 人工知能でのプログラミングの重要性

機械学習の特徴

機械学習は，**コンピュータ**が**データ**を使用して**学習**することにより**知的能力を向上**させる技術

- **情報の抽出**：データの中からパターンや関係性を自動で見つけ出す能力
- **簡潔さ**：人間が設定しなければならなかったルールを、自動で生成できるようになる
- **限界の超越**：他の方法では難しかった課題でも、機械学習を用いることで解決策や視点を得られる可能性がある

機械学習の概要と応用

1.　データの準備：

例： Irisデータセットの花弁の長さと幅のデータ

2. **機械学習の活用**：

- データを使って，コンピュータに学習させる.

3. **予測能力の獲得**：

- 学習の結果，新しいデータに対して予測ができるようになる.
- 例：花弁の長さが5.0cmの花に対して，その幅を1.7cmと予測

4. **データ内の規則性やパターンの自動抽出**：

- 従来のプログラミングは「花弁の長さがXcmなら幅はYcmである」といったルールを人間が記述する必要があった.
- 機械学習ではこれらの関係性を自動的に学習する.

データから価値ある情報を抽出し，新たな知見を得ることが可能になる.

演習６．データからのパターンの抽出

ページ６１～６４

- Irisデータセットを利用

  花弁の長さと幅のデータから，パターンを抽出
- 「線形回帰」の技術を使用
- 「花弁の長さが5.0cmの花に対して，その幅を1.7cmと予測」といった予測も可能になる

プログラム

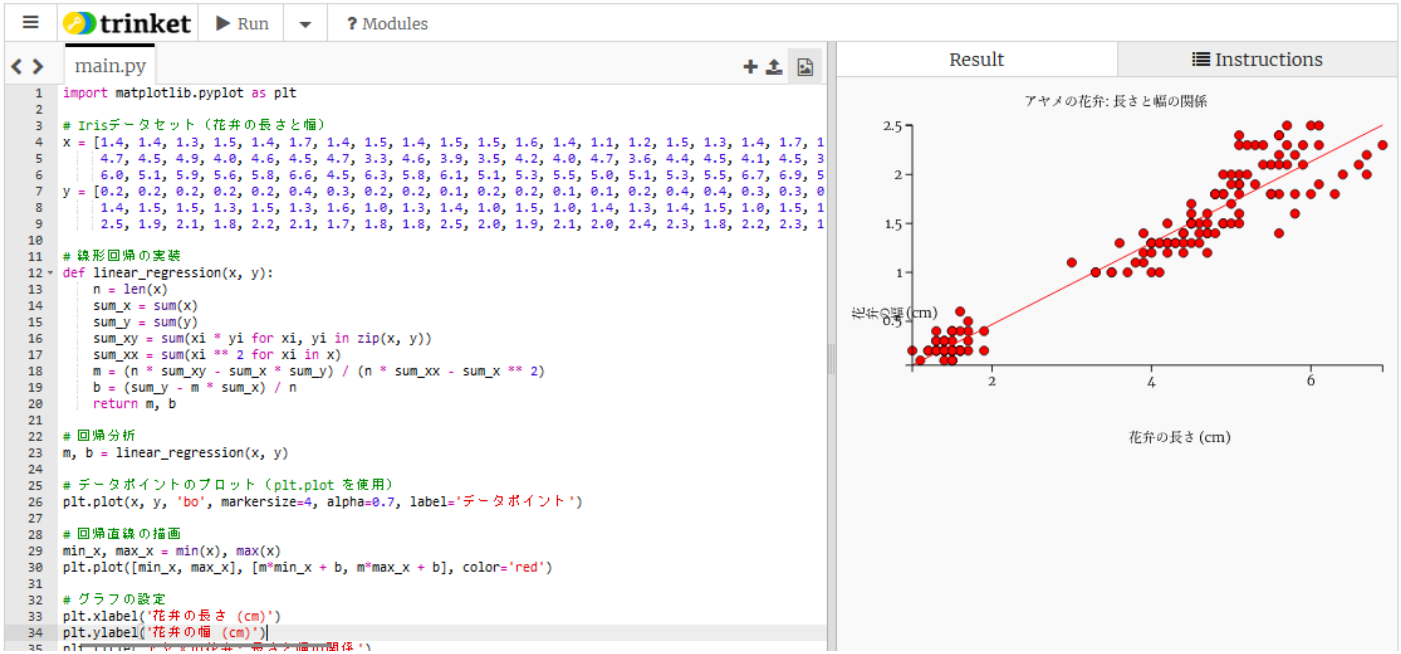
```
import matplotlib.pyplot as plt

# Irisデータセット（花弁の長さと幅）
x = [1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.6, 1.4, 1.1, 1.2, 1.5, 1.3, 1.4, 1.7, 1.5, 1.7, 1.5, 1.0, 1.7, 1.9, 1.6, 1.6, 1.5, 1.4, 1.6, 1.6, 1.5, 1.5, 1.4, 1.5, 1.2, 1.3, 1.5, 1.3, 1.5,
1.3, 1.3, 1.3, 1.6, 1.9, 1.4, 1.6, 1.4, 1.5, 1.4,
    4.7, 4.5, 4.9, 4.0, 4.6, 4.5, 4.7, 3.3, 4.6, 3.9, 3.5, 4.2, 4.0, 4.7, 3.6, 4.4, 4.5, 4.1, 4.5, 3.9, 4.8, 4.0, 4.9, 4.7, 4.3, 4.4, 4.8, 5.0, 4.5, 3.5, 3.8, 3.7, 3.9, 5.1, 4.5, 4.5, 4.7, 4.4, 4.1, 4.0,
4.4, 4.6, 4.0, 3.3, 4.2, 4.2, 4.2, 4.3, 3.0, 4.1,
    6.0, 5.1, 5.9, 5.6, 5.8, 6.6, 4.5, 6.3, 5.8, 6.1, 5.1, 5.3, 5.5, 5.0, 5.1, 5.3, 5.5, 6.7, 6.9, 5.0, 5.7, 4.9, 6.7, 4.9, 5.7, 6.0, 4.8, 4.9, 5.6, 5.8, 6.1, 6.4, 5.6, 5.1, 5.6, 6.1, 5.6, 5.5, 4.8, 5.4,
5.6, 5.1, 5.1, 5.9, 5.7, 5.2, 5.0, 5.2, 5.4, 5.1]
y = [0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.2, 0.1, 0.1, 0.2, 0.4, 0.4, 0.3, 0.3, 0.3, 0.2, 0.4, 0.2, 0.5, 0.2, 0.2, 0.4, 0.2, 0.2, 0.2, 0.2, 0.4, 0.1, 0.2, 0.2, 0.2, 0.2, 0.1, 0.2, 0.2,
0.3, 0.3, 0.2, 0.6, 0.4, 0.3, 0.2, 0.2, 0.2, 0.2,
    1.4, 1.5, 1.5, 1.3, 1.5, 1.3, 1.6, 1.0, 1.3, 1.4, 1.0, 1.5, 1.0, 1.4, 1.3, 1.4, 1.5, 1.0, 1.5, 1.1, 1.8, 1.3, 1.5, 1.2, 1.3, 1.4, 1.4, 1.7, 1.5, 1.0, 1.1, 1.0, 1.2, 1.6, 1.5, 1.6, 1.5, 1.3, 1.3, 1.3,
1.2, 1.4, 1.2, 1.0, 1.3, 1.2, 1.3, 1.3, 1.1, 1.3,
    2.5, 1.9, 2.1, 1.8, 2.2, 2.1, 1.7, 1.8, 1.8, 2.5, 2.0, 1.9, 2.1, 2.0, 2.4, 2.3, 1.8, 2.2, 2.3, 1.5, 2.3, 2.0, 2.0, 1.8, 2.1, 1.8, 1.8, 1.8, 2.1, 1.6, 1.9, 2.0, 2.2, 1.5, 1.4, 2.3, 2.4, 1.8, 1.8, 2.1,
2.4, 2.3, 1.9, 2.3, 2.5, 2.3, 1.9, 2.0, 2.3, 1.8]

# 線形回帰の実装
def linear_regression(x, y):
    n = len(x)
    sum_x = sum(x)
    sum_y = sum(y)
    sum_xy = sum(xi * yi for xi, yi in zip(x, y))
    sum_xx = sum(xi ** 2 for xi in x)
    m = (n * sum_xy - sum_x * sum_y) / (n * sum_xx - sum_x ** 2)
    b = (sum_y - m * sum_x) / n
    return m, b

# 回帰分析
m, b = linear_regression(x, y)

# データポイントのプロット（plt.plot を使用）
plt.plot(x, y, 'bo', markersize=4, alpha=0.7, label='データポイント')

# 回帰直線の描画
min_x, max_x = min(x), max(x)
plt.plot([min_x, max_x], [m*min_x + b, m*max_x + b], color='red')

# グラフの設定
plt.xlabel('花弁の長さ (cm)')
plt.ylabel('花弁の幅 (cm)')
plt.title('アヤメの花弁: 長さと幅の関係')

# グラフの表示
plt.show()
```

がく片の長さと幅

① trinket の次のページを開く

https://trinket.io/python/6bcf5472e3fd

② 実行結果が，次のように表示されることを確認



- 実行が開始しないときは、「**実行ボタン**」で**実行**
- ソースコードを**書き替えて再度実行**することも可能

プログラミングのさまざまな応用分野

- **Web開発**

フロントエンド（HTML, CSS, JavaScript），バックエンド（Python, Django, Flask）

- **データ分析**

ビッグデータ処理，統計分析，データビジュアライゼーション

- **人工知能**

自然言語処理，コンピュータビジョン，予測モデリング

- **ゲーム開発**

2Dゲーム，3Dゲーム，モバイルゲーム

- **IoT（Internet of Things）**

センサーデータの収集と分析，スマートホームシステム

- **サイバーセキュリティ**

ネットワークセキュリティ，暗号化技術

まとめ

- Python の基本的な文法と機能を学んだ
- Trinket を用いて実際にプログラムを実行した
- タートルグラフィックスを使用した
- 散布図（データ可視化）を体験した
- プログラミングの重要性と応用可能性を理解した

次のステップへ

- 人工知能のさらなる探求，学び

- 「AI演習」はプログラミングの初心者対象を前提

- AI の動く仕組みの理解（AI はプログラムで動く）
- AI プログラムは，人間が観察可能，編集による調整可能である
- そのために，プログラミングの入門知識が役立つ