

# 医療とAI・ビッグデータ入門（縮小版①）

## Pythonの基礎

~データの扱い、numpy、pandasについて~

## Pythonによる線形回帰

本教材を使用した際にはお手数ですが、  
下記アンケートフォームにご協力下さい。

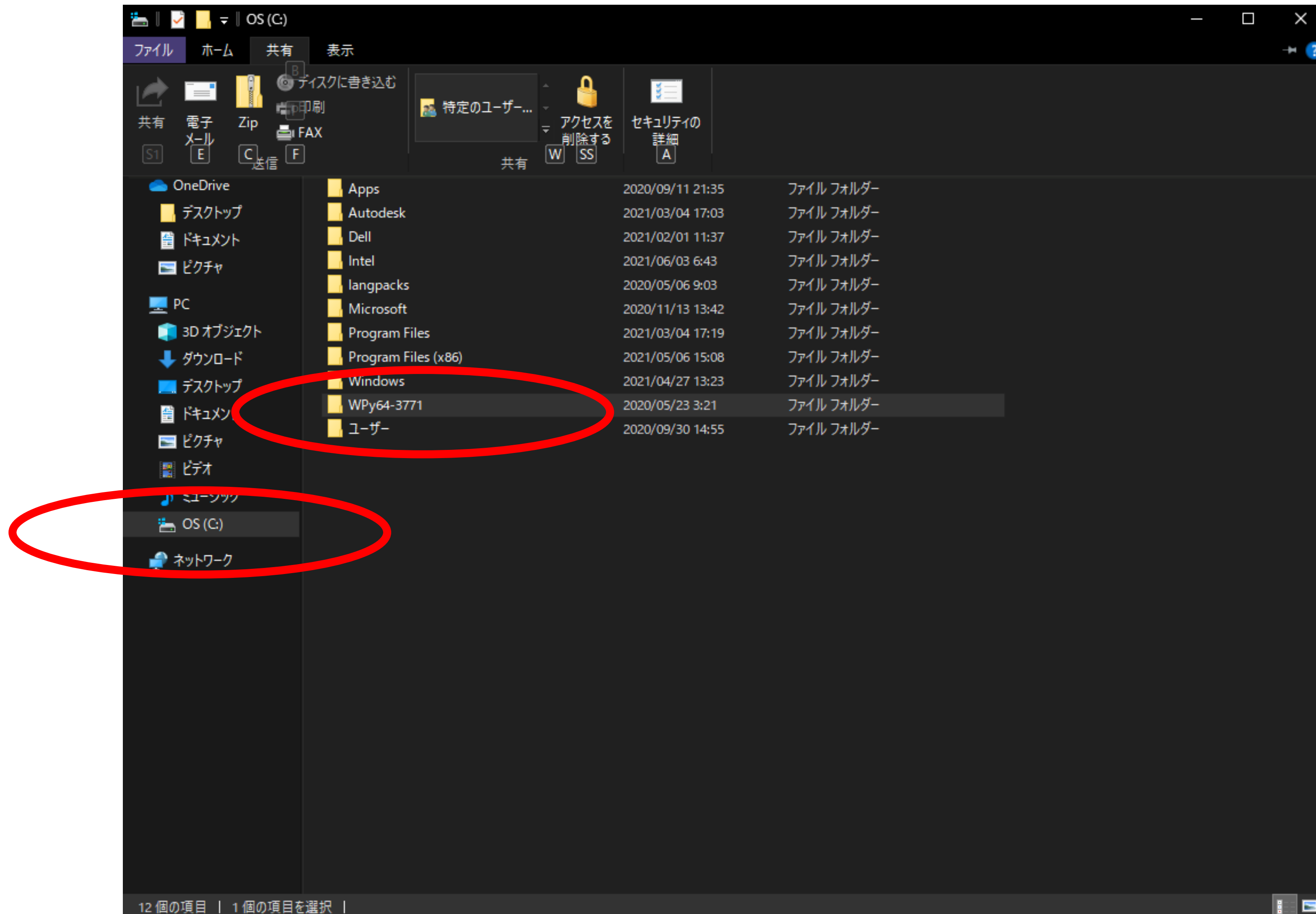
<https://forms.gle/cgej2DL5PvneRhCp8>

東京医科歯科大学  
統合教育機構  
須藤毅顕

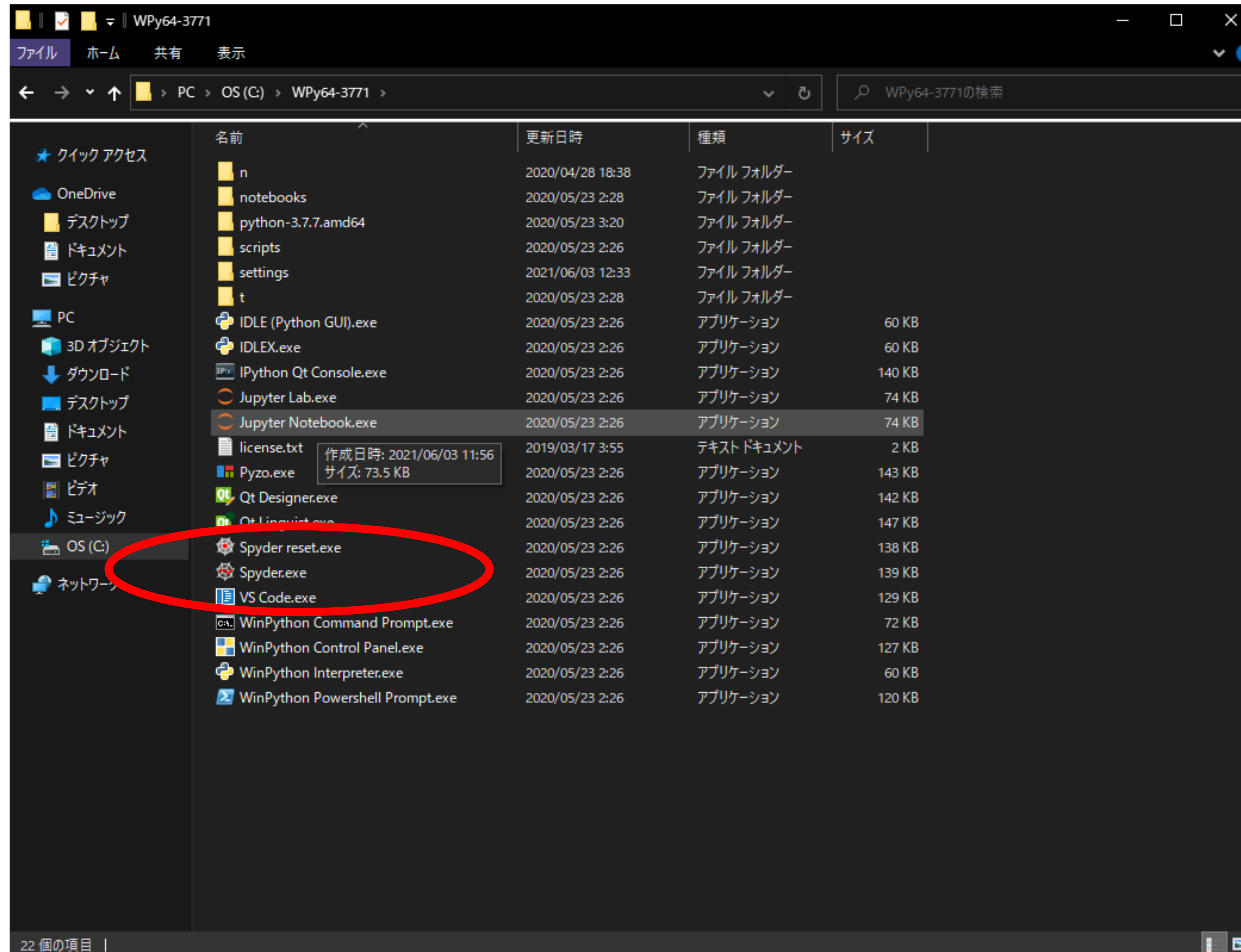
# データの扱いとライブラリの使い方

まず機械学習、深層学習で用いる  
pythonの基本知識とデータ取り扱い方について解説します。

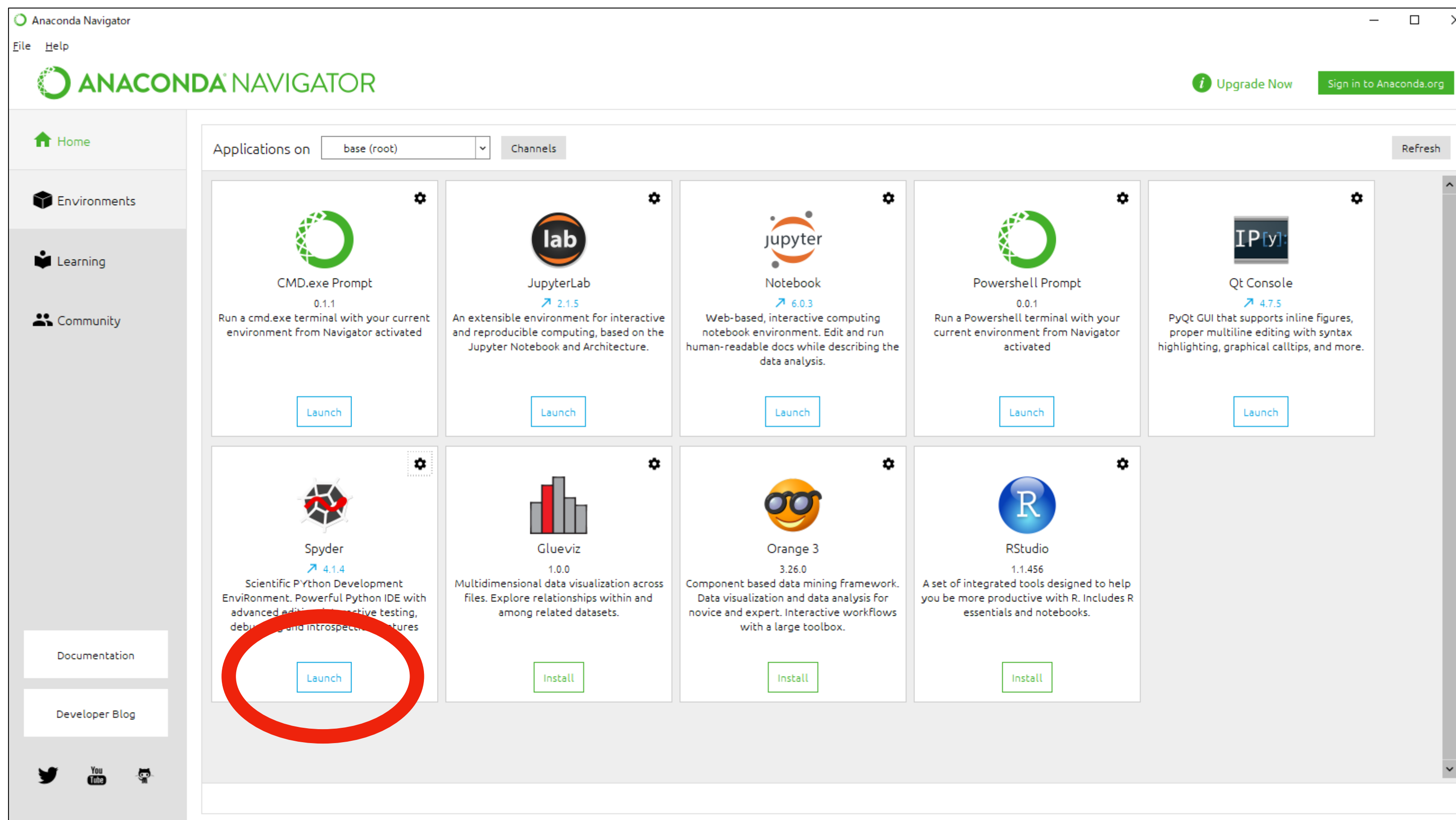
# Winpythonを起動しましょう (Windows)



# Winpythonを起動しましょう (Windows)



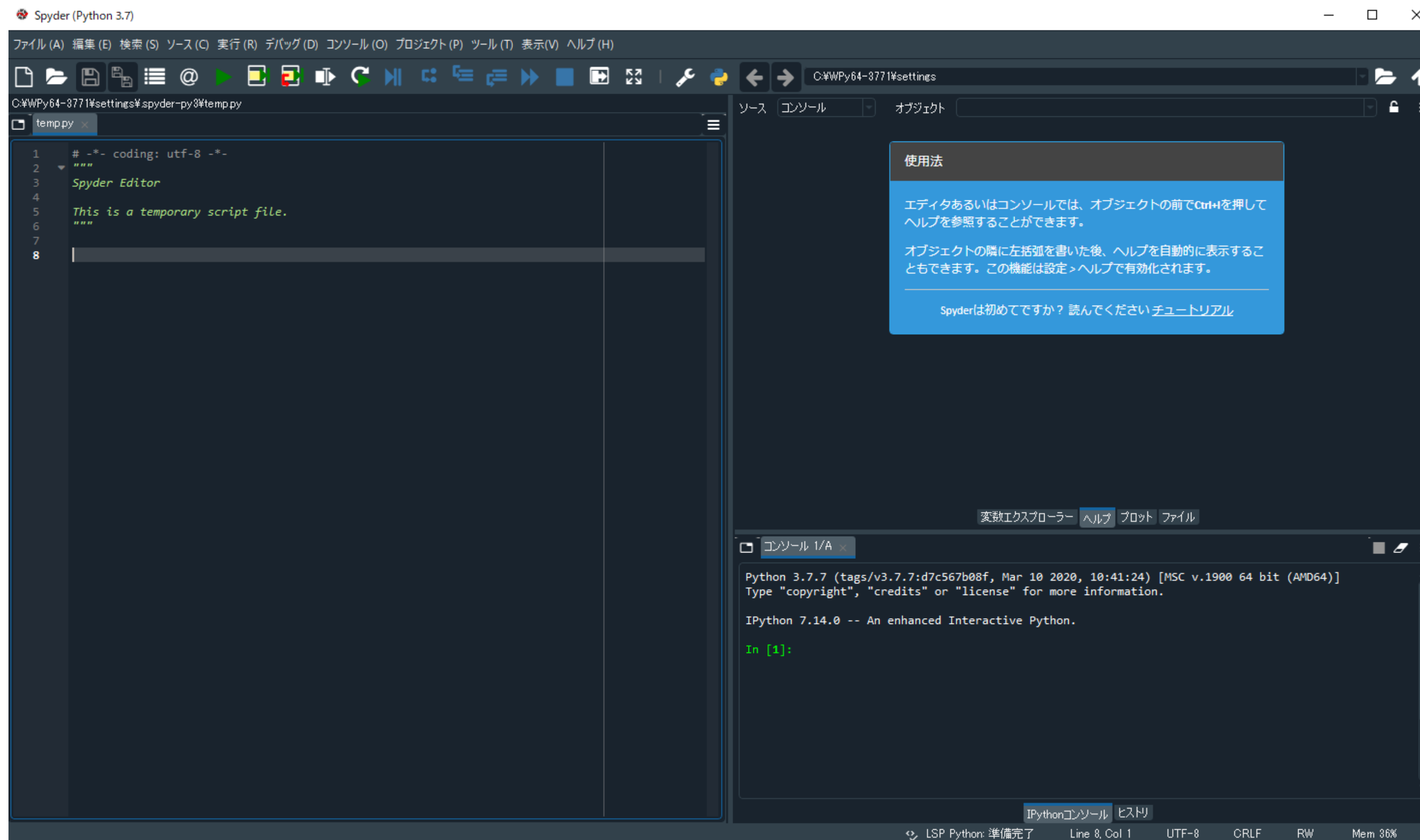
# Anaconda NavigatorでSpyderを立ち上げる(Mac)



“Install”となっている人は、クリックしてしばらく待つと  
“Launch”になるので、再度クリックします。

# Pythonの実行方法

Spyderのホーム画面が表示されます。(spyderのバージョン4)

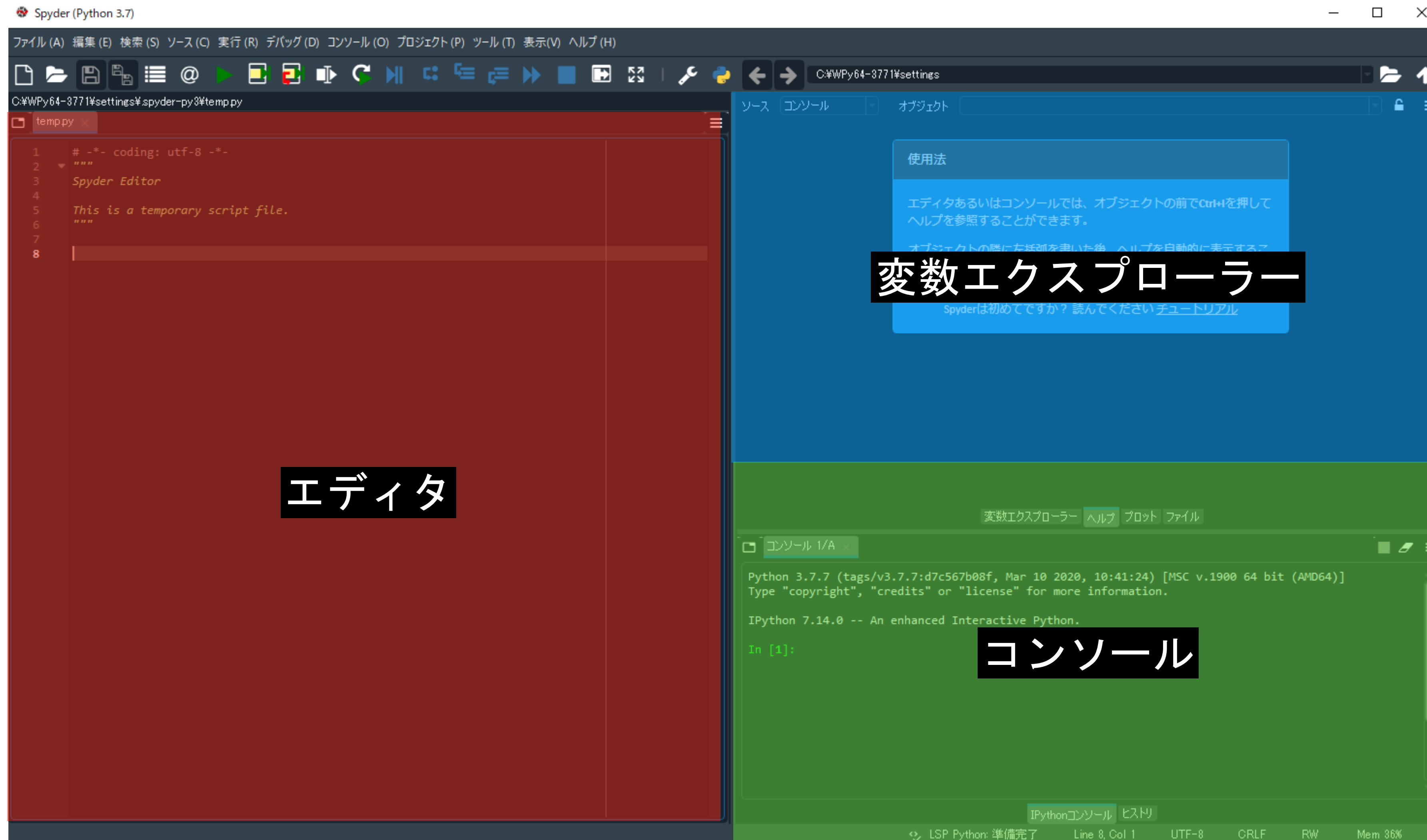


バージョンが違っていると表示されるアイコンが少し違う可能性があります。そのままでも大丈夫ですし、アップデートしても良いです。



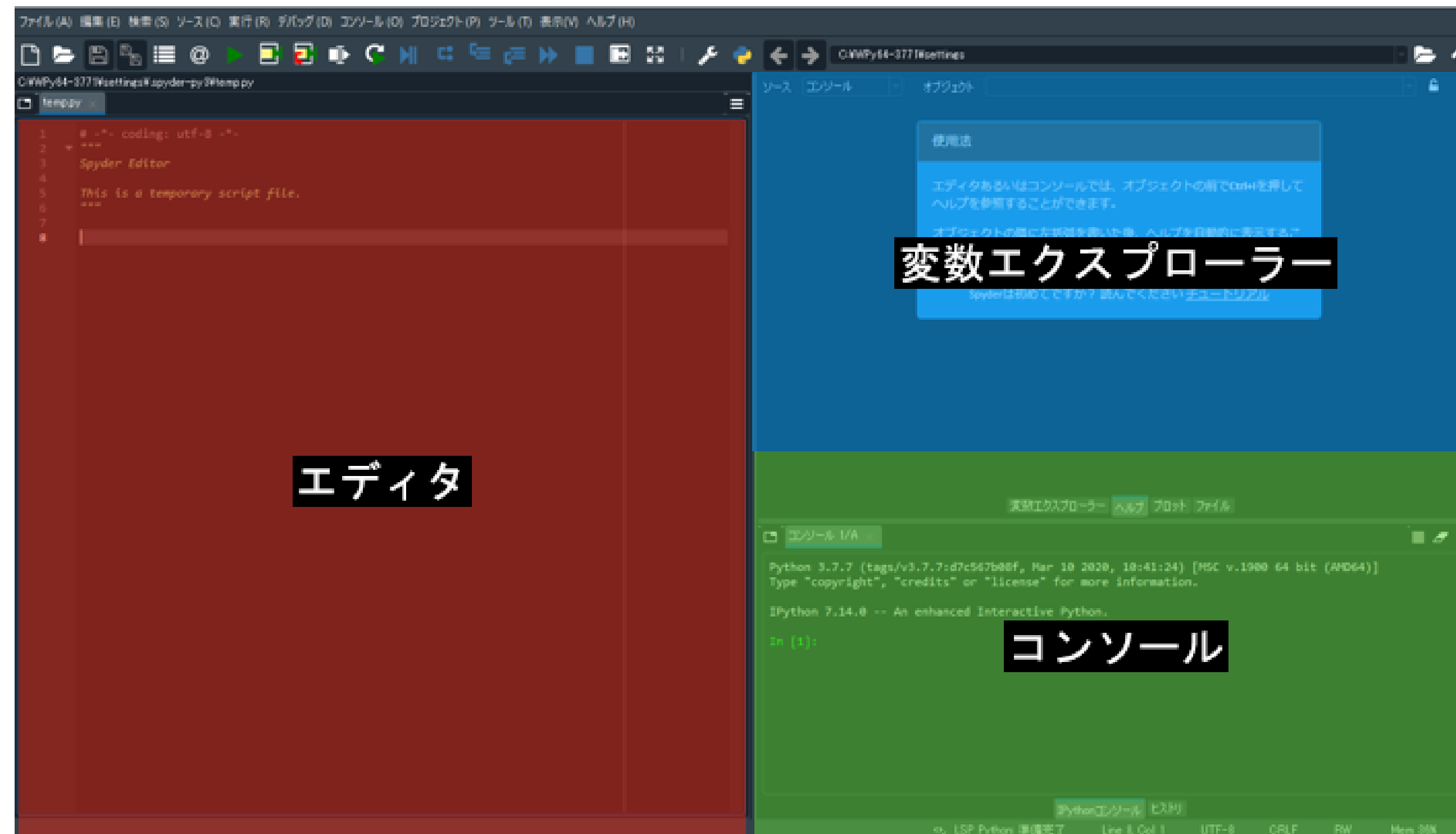
# Spyderを起動する

大きく3つの画面に分かれております。左側のエディタ、右上の変数エクスプローラー、右下のコンソールです。



# Spyderの実行方法

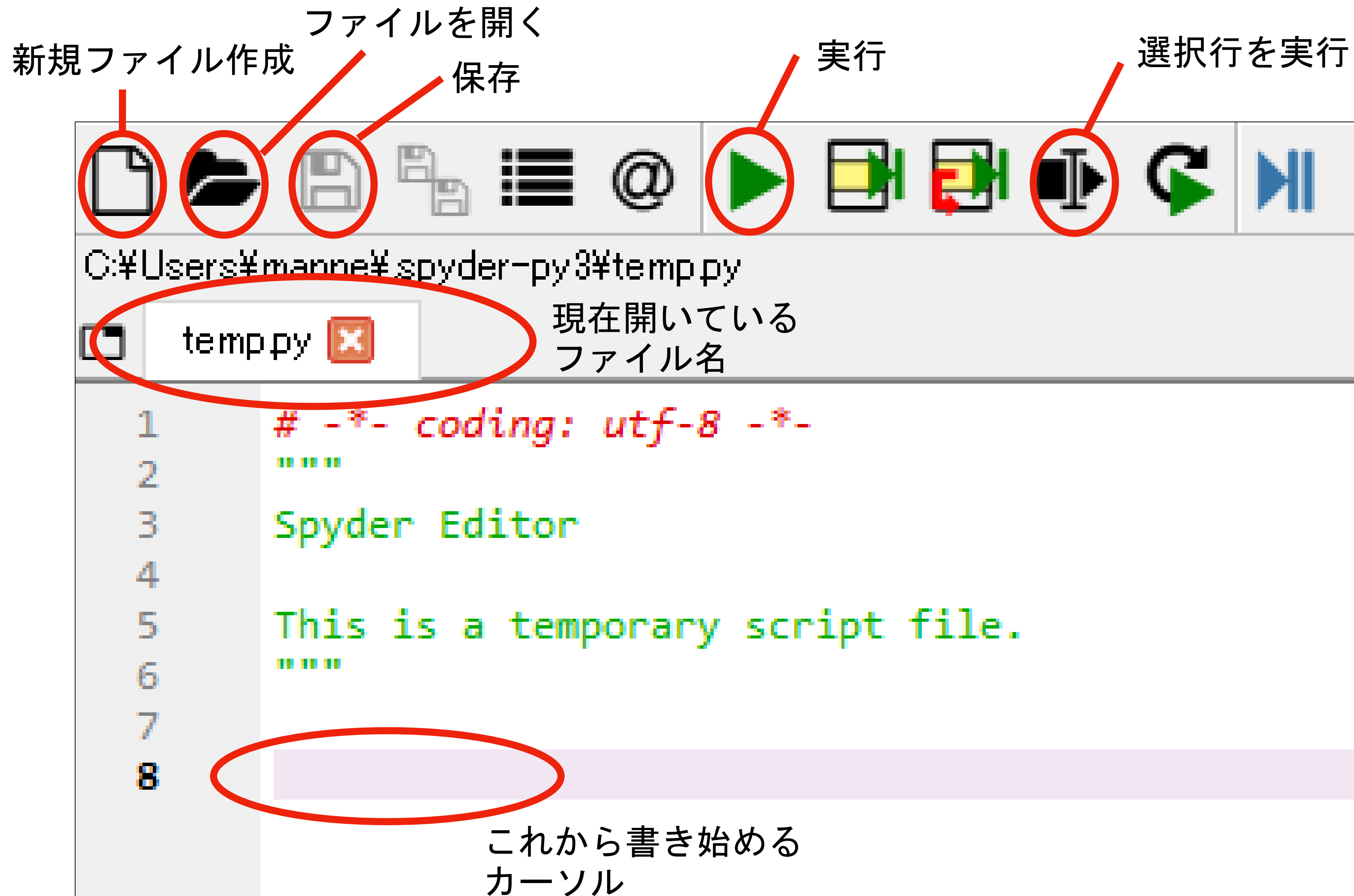
- ① エディタに文を書き、"実行"で(上から順に)全ての行を実行する
- ② エディタに文を書き、"選択行を実行"で選んだ行を実行する
- ③ コンソールに直接書き込み、enterで実行する  
→結果は全てコンソールに表示される



③は対話的(チャットのように)すぐ結果が表示されるのですが、長い文を書くときや文を修正、保存するにはエディタの使用が適していますので今回は①②で行います



# Spyderを実行する（確認事項）



# Spyderを実行する（確認事項）

新規ファイル作成      ファイルを開く      保存      実行      1行ずつ実行

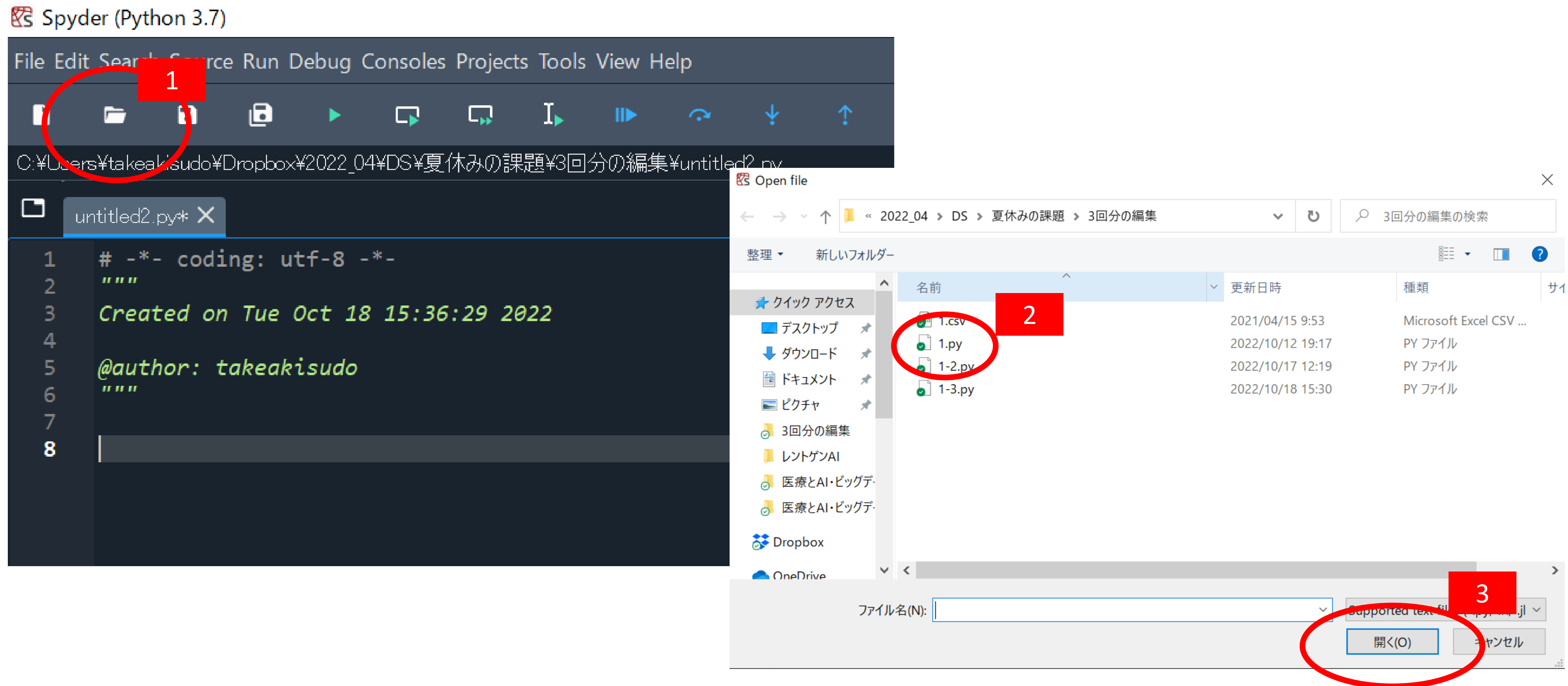
The screenshot shows the Spyder IDE interface. At the top is a toolbar with icons for creating a new file, opening a file, saving, running, and running line by line. Below the toolbar is a text area showing the file path: `/Users/takeakisudo/Desktop/タイトル無し1.py`. Underneath the path is a tab bar with two tabs: `temp.py` and `タイトル無し1.py`. The `タイトル無し1.py` tab is selected and highlighted with a red oval, with a label pointing to it that says "現在開いているファイル名". The main editor area shows the following code:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Tue May 25 06:50:31 2021
5
6  @author: takeakisudo
7  """
8  
```

The line number 8 is circled in red, with a label pointing to it that says "これから書き始めるカーソル".

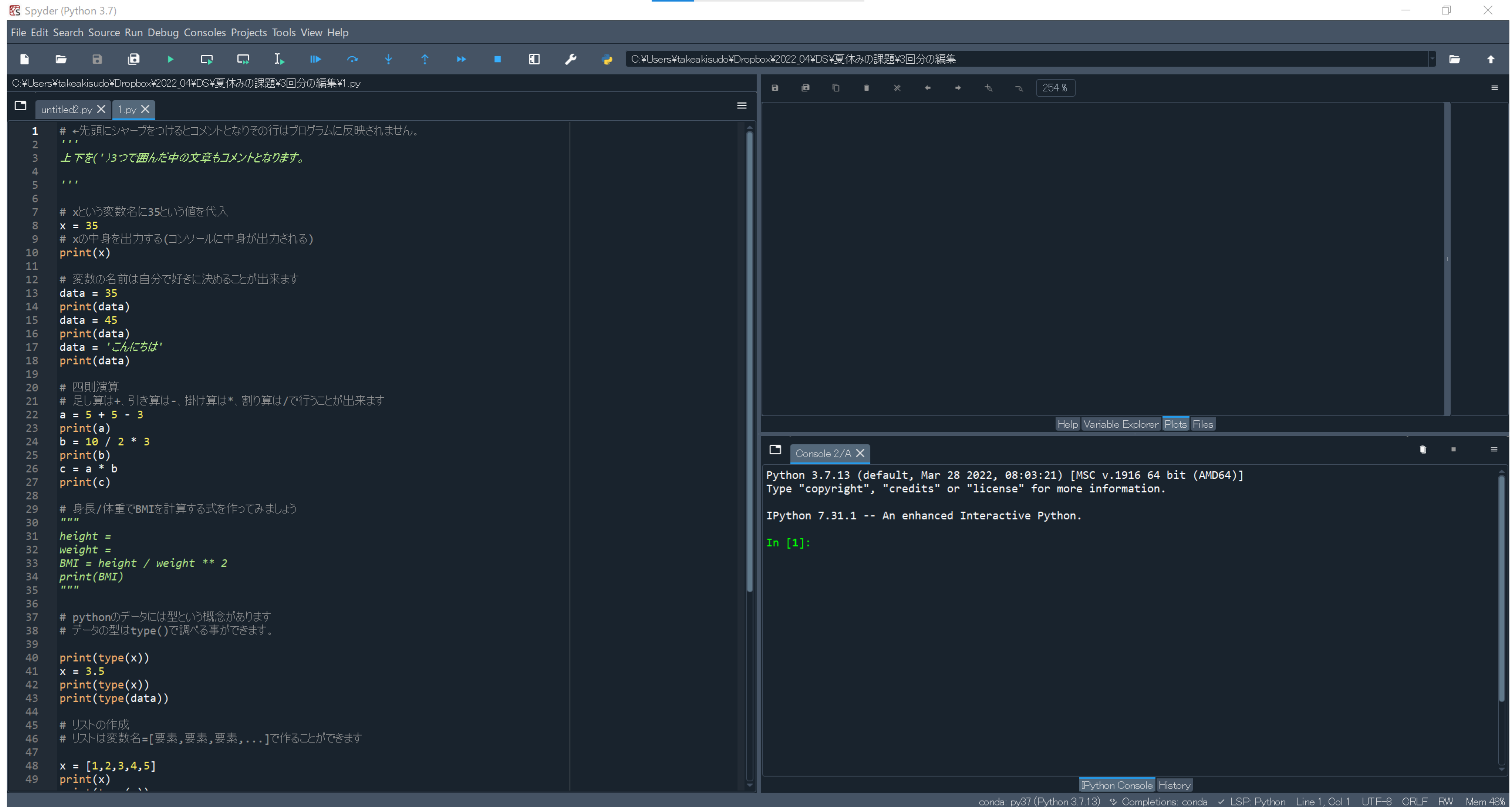
# ファイルを開く

医療とAI・ビッグデータ入門フォルダの1.pyを開いてみましょう



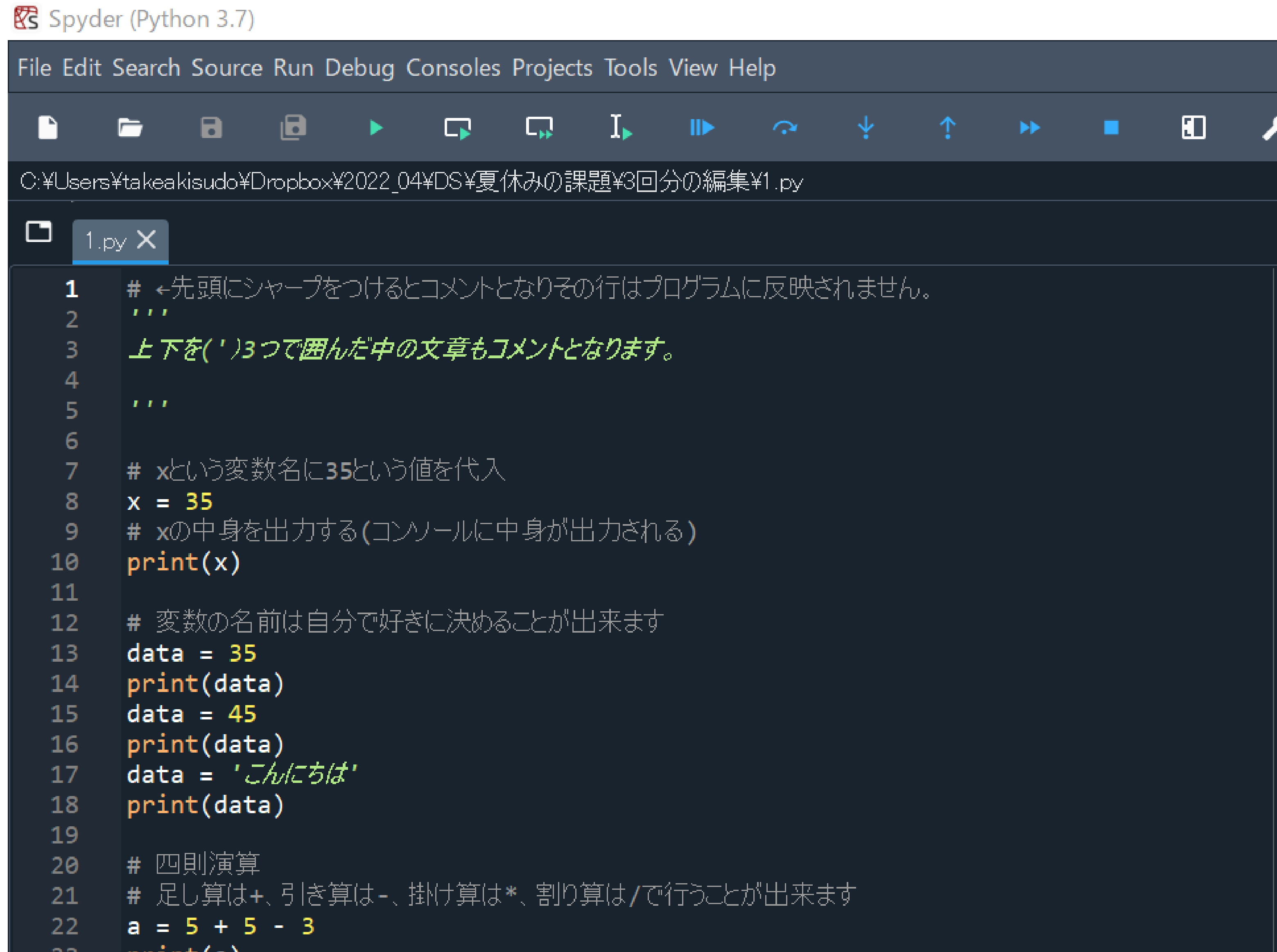
# 1.pyを開いたSpyderの画面

左側のエディタに1.pyの中身が表示されます



# データの操作の仕方について

拡大するとこのようになっております。

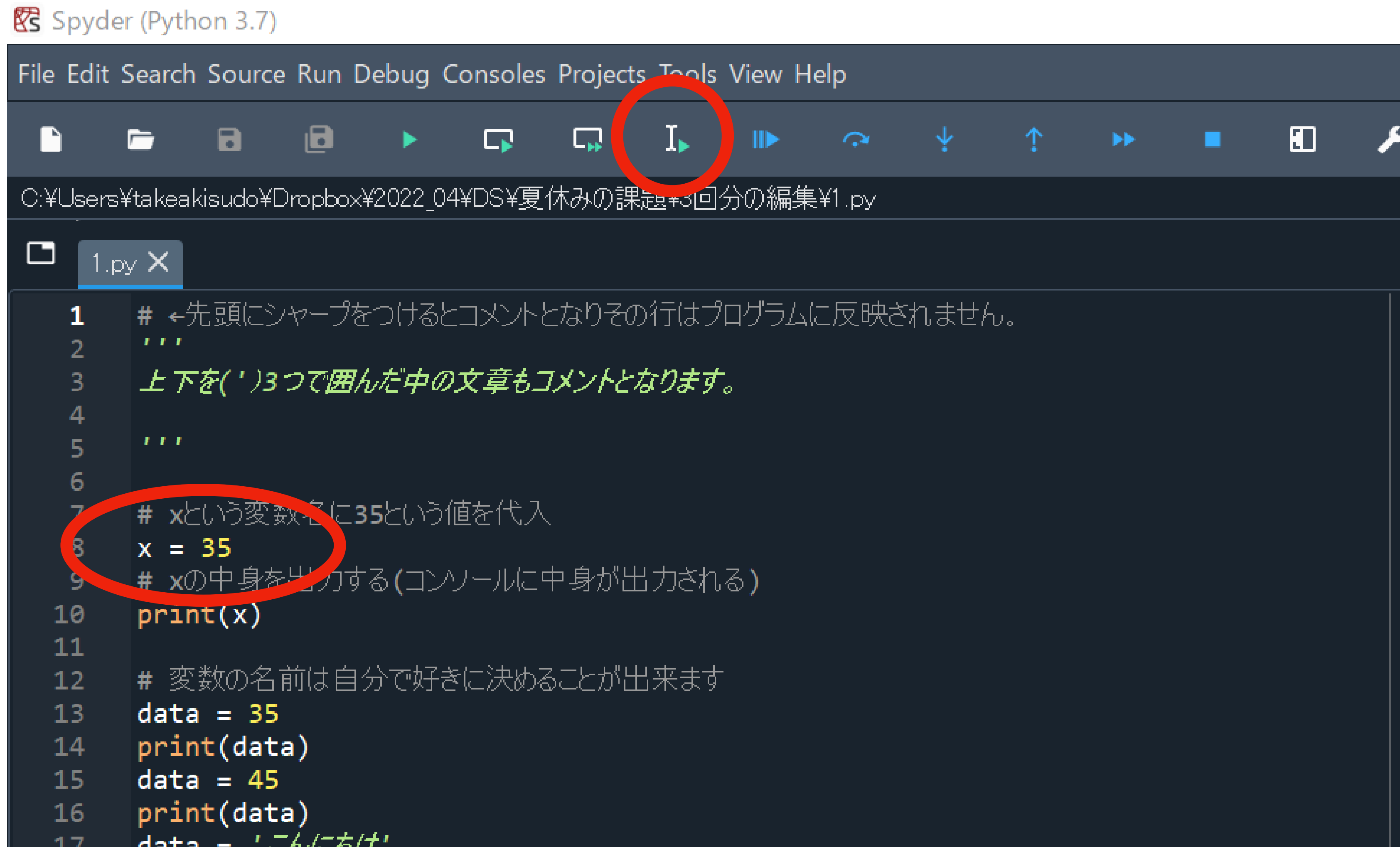


The image shows the Spyder Python IDE interface. The title bar indicates 'Spyder (Python 3.7)'. The menu bar includes 'File', 'Edit', 'Search', 'Source', 'Run', 'Debug', 'Consoles', 'Projects', 'Tools', 'View', and 'Help'. The toolbar contains icons for file operations (new, open, save, save as), execution (run, step through, step over, step under, break), and other functions (undo, redo, find, replace, zoom in, zoom out, full screen, settings). The file path is 'C:\Users\takeakisudo\Dropbox\2022\_04\DS\夏休みの課題\3回分の編集\1.py'. The editor shows a Python script with the following code:

```
1  # ←先頭にシャープをつけるとコメントとなりその行はプログラムに反映されません。
2  '''
3  上下を(')3つで囲んだ中の文章もコメントとなります。
4  '''
5
6
7  # xという変数名に35という値を代入
8  x = 35
9  # xの中身を出力する(コンソールに中身が出力される)
10 print(x)
11
12 # 変数の名前は自分で好きに決めることができます
13 data = 35
14 print(data)
15 data = 45
16 print(data)
17 data = 'こんにちは'
18 print(data)
19
20 # 四則演算
21 # 足し算は+, 引き算は-, 掛け算は*, 割り算は/で行うことができます
22 a = 5 + 5 - 3
23 print(a)
```

# データの操作の仕方について

カーソルを" x= 35"の行に合わせて(クリックして)、"選択行を実行"を押してみましょう  
(x = 35 の行 (この図では 8 行目)であればどの位置にカーソルがあっても問題ありません)



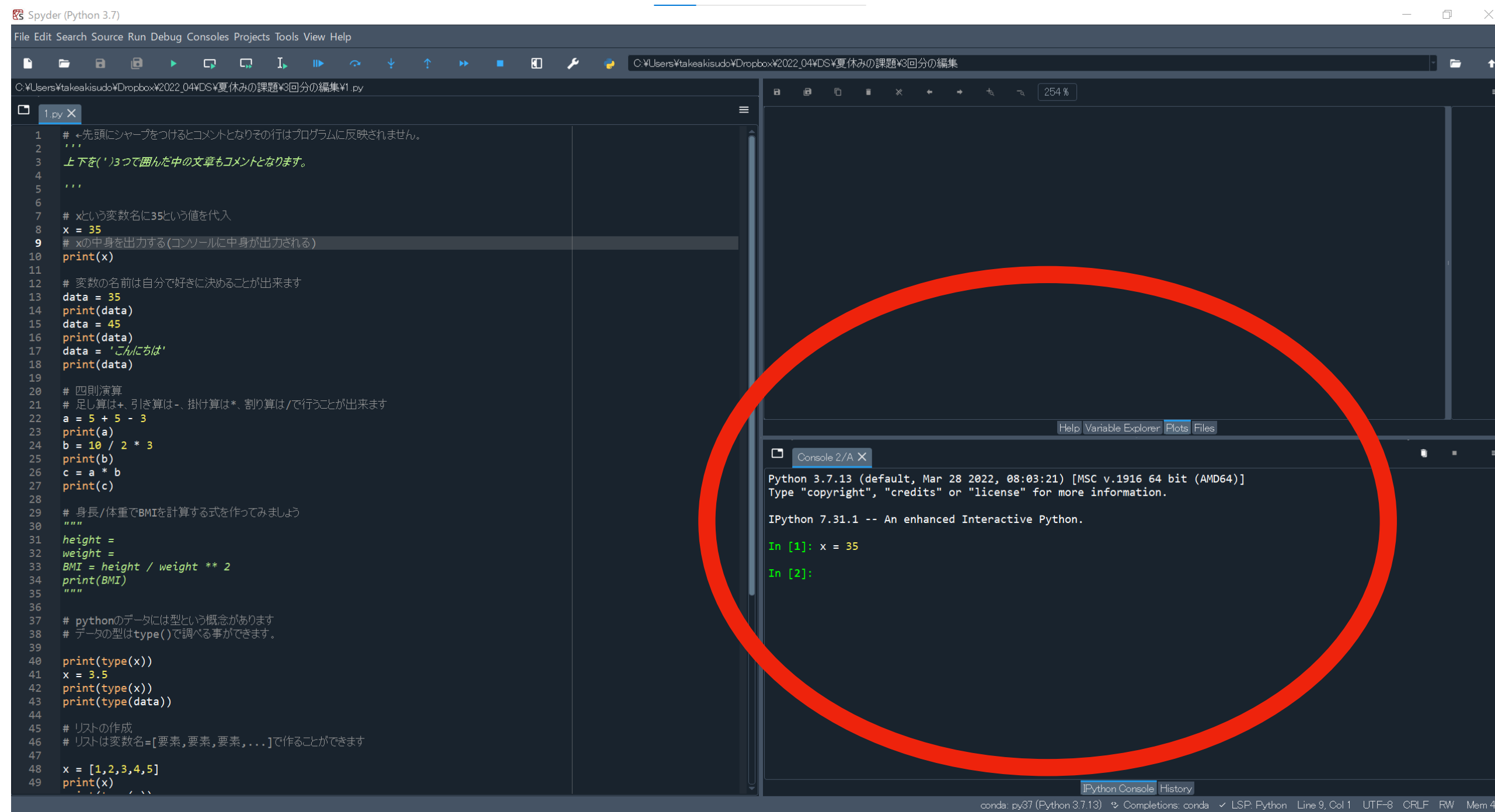
```
Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\takeakisudo\Dropbox\2022_04\DS\夏休みの課題\3回分の編集\1.py
1.py
1  # ←先頭にシャープをつけるとコメントとなりその行はプログラムに反映されません。
2  '''
3  上下を(')3つで囲んだ中の文章もコメントとなります。
4  '''
5
6
7  # xという変数名に35という値を代入
8  x = 35
9  # xの中身を表示する(コンソールに中身が出力される)
10 print(x)
11
12 # 変数の名前は自分で好きに決めることができます
13 data = 35
14 print(data)
15 data = 45
16 print(data)
17 data = 'こんばんは'
```



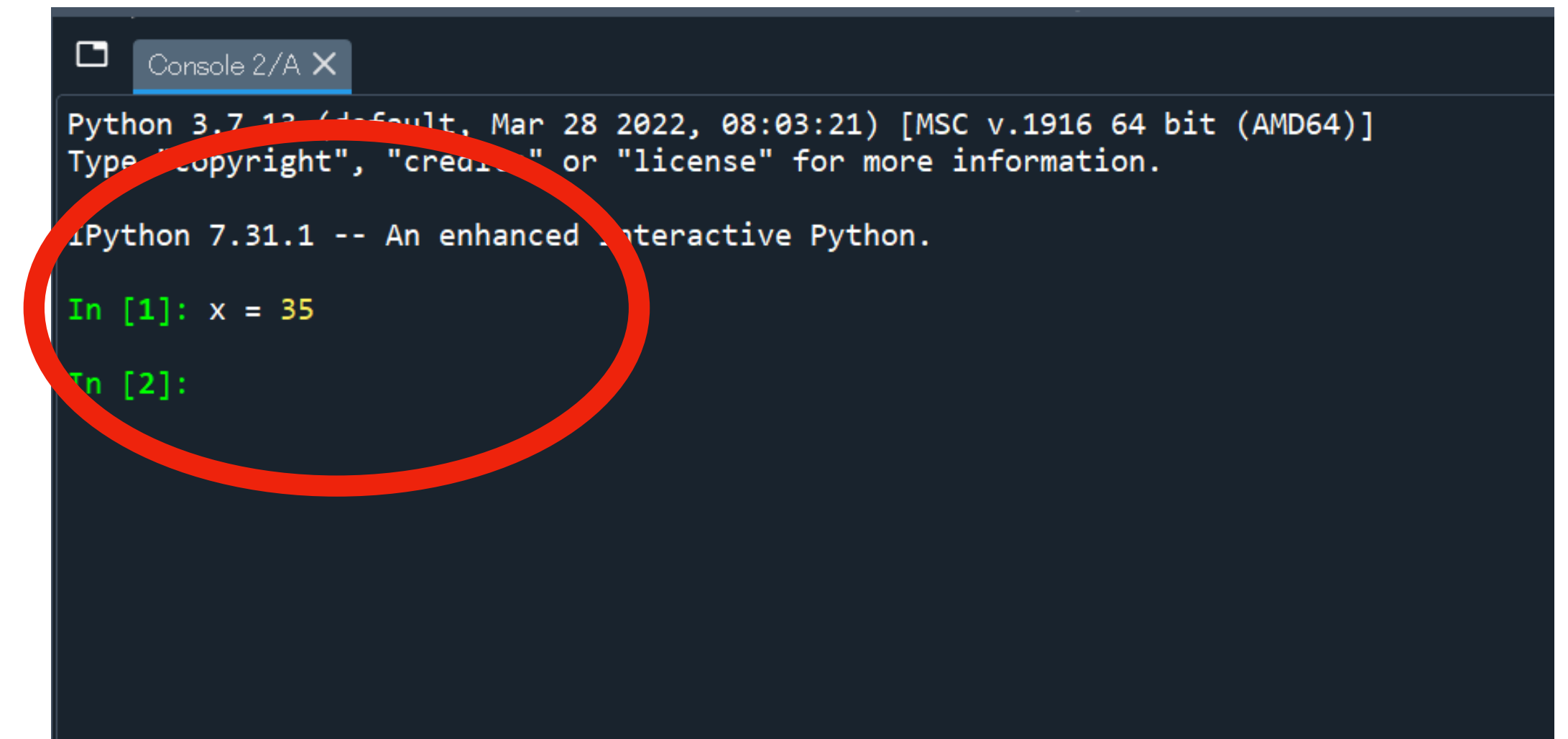
# データの操作の仕方について

右下のコンソール画面に実行結果が表示されます

## Spyder全体画面



## コンソール画面



このように、エディタで書いた内容(プログラム)の実行結果が右下のコンソール画面に表示されます

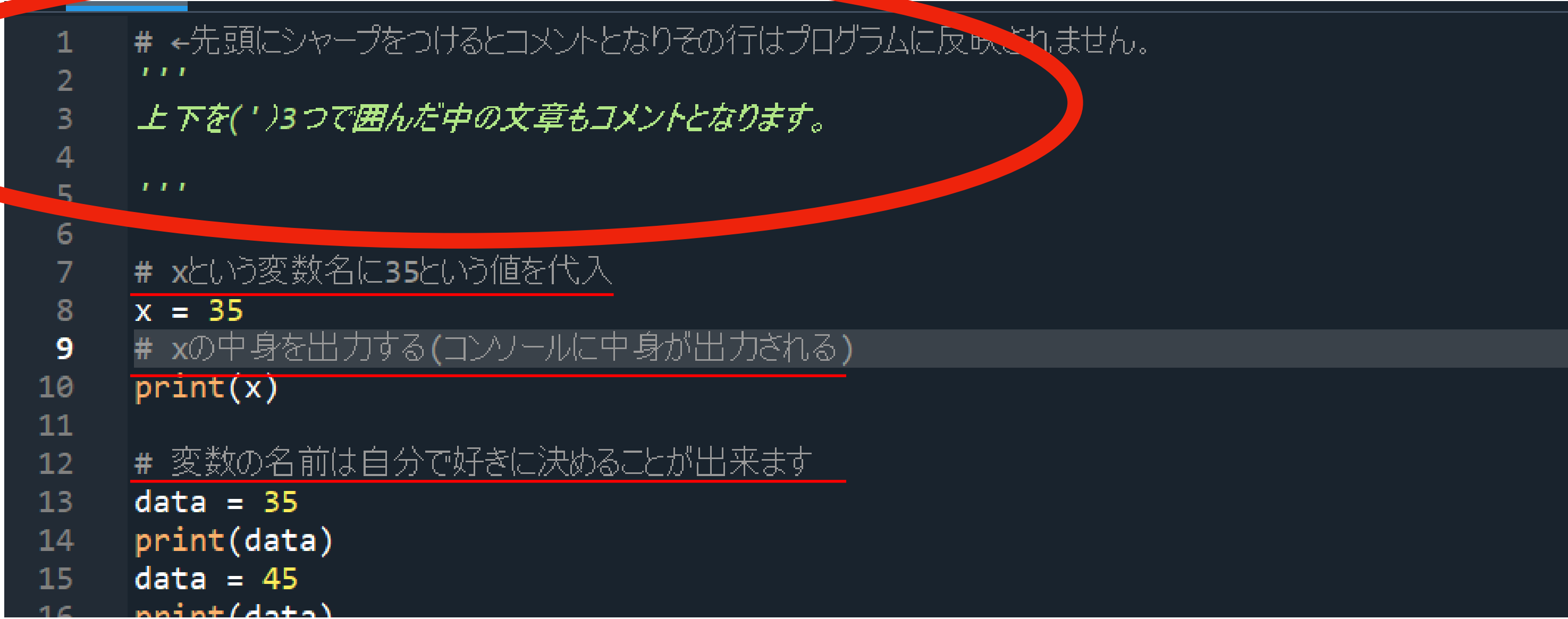
# データの操作の仕方について

一行ずつ実行のアイコンを押すと自動的に次の行に移動します

```
1  # ←先頭にシャープをつけるとコメントとなりその行はプログラムに反映されません。
2  '''
3  上下を('')3つで囲んだ中の文章もコメントとなります。
4  '''
5
6
7  # xという変数名に35という値を代入
8  x = 35
9  # xの中身を出力する(コンソールに中身が出力される)
10 print(x)
11
12 # 変数の名前は自分で好きに決めることができます
13 data = 33
14 print(data)
15 data = 45
16 print(data)
```

# データの操作の仕方について

一行ずつ実行のアイコンを押すと自動的に次の行に移動します

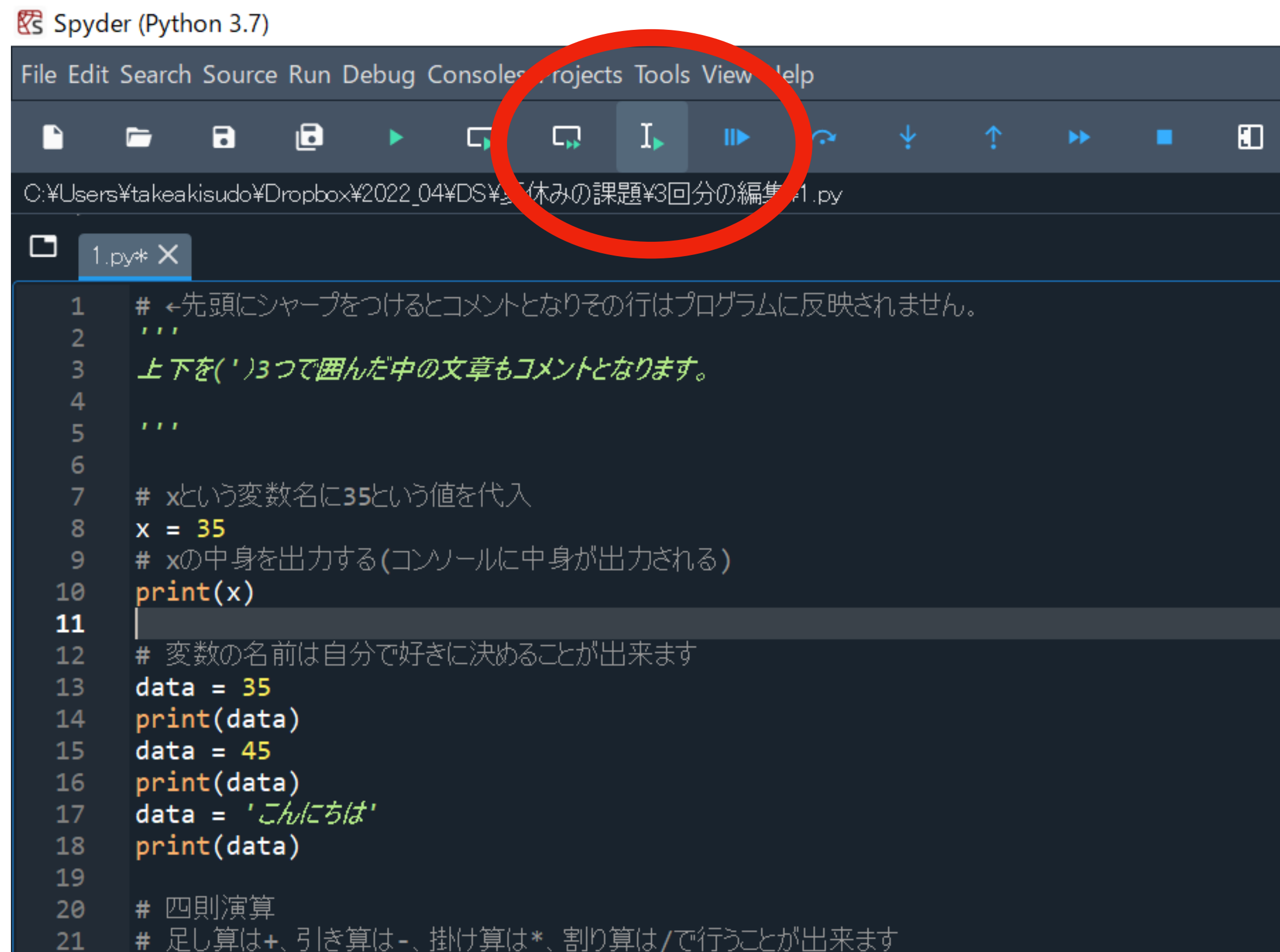


```
1  # ←先頭にシャープをつけるとコメントとなりその行はプログラムに反映されません。
2  '''
3  上下を(')3つで囲んだ中の文章もコメントとなります。
4  '''
5
6  # xという変数名に35という値を代入
7  x = 35
8  # xの中身を出力する(コンソールに中身が出力される)
9  print(x)
10
11 # 変数の名前は自分で好きに決めることができます
12 data = 35
13 print(data)
14 data = 45
15 print(data)
```

エディタ内の#で始まる行もしくは'''で囲まれた文章はコメントで  
実行してもコンソールに結果が反映されません。  
メモや書いている内容を整理する際に使用します。

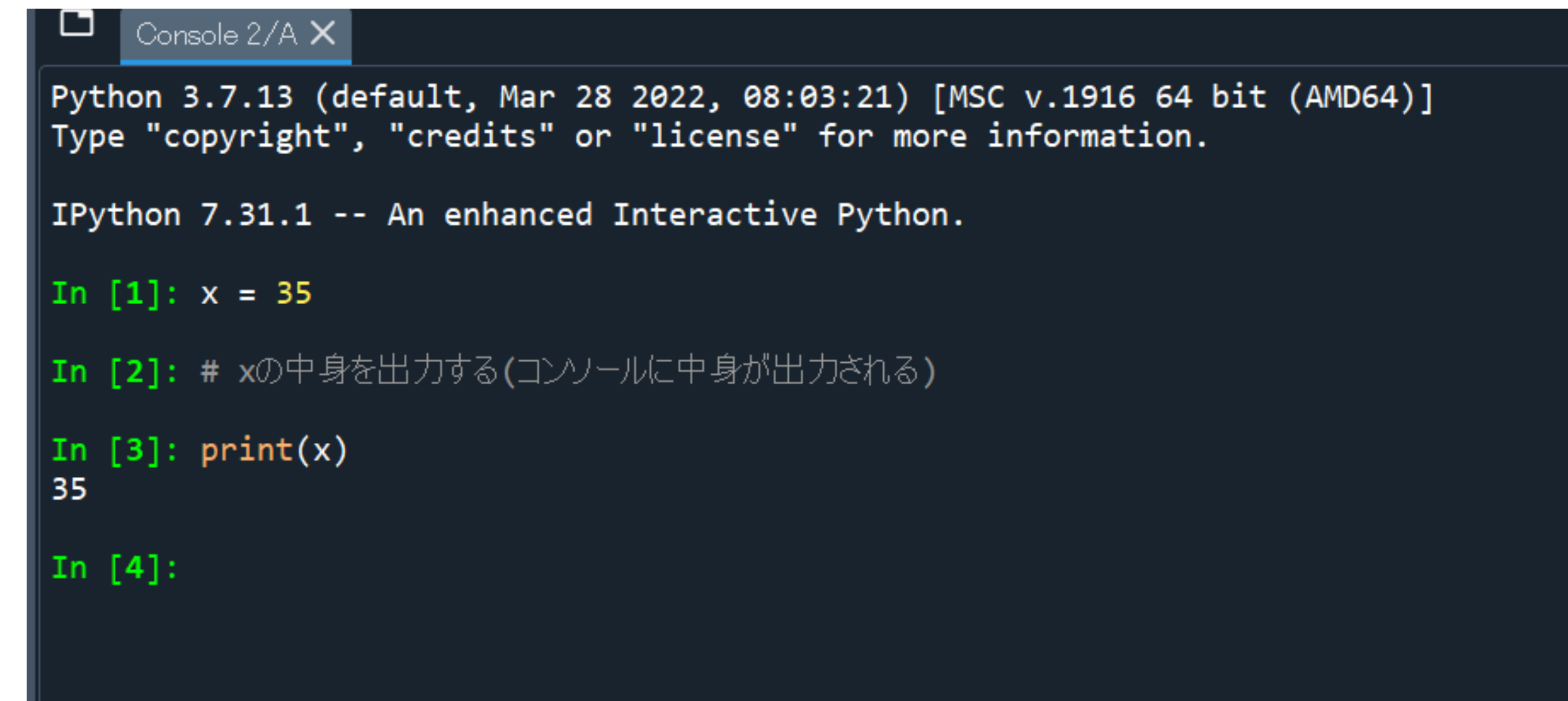
# データの操作の仕方について

試しに一度に実行のアイコンを2回押して2行追加で実行してみましょう



```
1 # ←先頭にシャープをつけるとコメントとなりその行はプログラムに反映されません。
2 '''
3 上下を(')3つで囲んだ中の文章もコメントとなります。
4 '''
5
6
7 # xという変数名に35という値を代入
8 x = 35
9 # xの中身を表示する(コンソールに中身が表示される)
10 print(x)
11
12 # 変数の名前は自分で好きに決めることができます
13 data = 35
14 print(data)
15 data = 45
16 print(data)
17 data = 'こんにちは'
18 print(data)
19
20 # 四則演算
21 # 足し算は+, 引き算は-, 掛け算は*, 割り算は/で行うことができます
```

## コンソール画面



```
Python 3.7.13 (default, Mar 28 2022, 08:03:21) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 7.31.1 -- An enhanced Interactive Python.

In [1]: x = 35

In [2]: # xの中身を表示する(コンソールに中身が表示される)

In [3]: print(x)
35

In [4]:
```

# の行はそのまま文章が表示されますが、何も気にしなくて大丈夫です。

print(x)を実行すると35という結果が表示されます。

# 変数の扱い方

データを保存したものを”変数”と言います

変数名	=	データ
-----	---	-----

独自につける変数名に対して、データを代入します

入力

# xという変数名に35という値を代入 x = 35
-------------------------------

“=”は数学的には等しいという意味ですが、多くのプログラミング言語では、左側の変数に右側の値を代入するという意味になります。

# 関数について

プログラミングでは“関数”というものを扱います

関数は、四則演算、繰り返し、比較演算など何かしらの処理を保存したもので、データを関数に与えることで処理を行ったデータを返してくれます。



関数は自作することも可能ですが、pythonではあらかじめ便利な関数が多数用意されています。(組み込み関数といいます)



# print関数

print(引数) : ( )の中身を出力する

入力

```
# xという変数名に35という値を代入  
x = 35  
# xの中身を出力する(コンソールに中身が出力される)  
print(x)
```

出力

35

print()関数は最もよく使う組み込み関数の1つです。関数の( )の中を引数(ひきすう)と言い、ここに変数やデータを入れるとコンソールに中身を出力してくれます。

ここではxに3を代入しているので、xをprint関数で出力し、35が表示されます。

# 1行ずつ実行してみましょう

```
data = 35  
print(data)
```

```
data = 45
```

```
print(data)
```

```
data = "こんにちは"
```

```
print(data)
```

←dataという変数に35を代入するという意味

←変数dataを出力しろという意味

→ 35

←dataという変数に35を代入するという意味  
(35から45に上書きされる)

←変数dataを出力しろという意味

→ 45

←dataという変数に"こんにちは"という文字列を代入するという意味。  
(45から"こんにちは"に上書きされる。)

←変数dataを出力しろという意味

→ こんにちは

```
In [4]: data = 35
```

```
In [5]: print(data)  
35
```

```
In [6]: data = 45
```

```
In [7]: print(data)  
45
```

```
In [8]: data = 'こんにちは'
```

```
In [9]: print(data)  
こんにちは
```

# 変数とprint()の演習

変数名ageに自分の年齢を代入して出力してみましょう

```
21
22 # 変数名ageに自分の年齢をいれて出力してみましょう
23 $$$ = $$$
24 print($$$)
25
```

# 変数とprint()の演習

変数名ageに自分の年齢を代入して出力してみましょう

```
21  
22 # 変数名ageに自分の年齢をいれて出力してみましょう  
× 23 $$$ = $$$  
24 print($$$)  
25
```

```
# 変数名ageに自分の年齢をいれて出力してみましょう  
age = 37  
print(age)
```

```
In [112]: age = 37  
  
In [113]: print(age)  
37
```

# 四則演算

足し算は+、引き算は-、掛け算は\*、割り算は/  
階乗は\*を連続して書きます

```
a = 5 + 5 - 3
print(a)
b = 10 / 2 * 3
print(b)
c = a * b
print(c)
print(100 * 100)
print(2**3)
```

半角スペースはあっても無くても認識されません

```
In [11]: a = 5 + 5 - 3
In [12]: print(a)
7
In [13]: b = 10 / 2 * 3
In [14]: print(b)
15.0
In [15]: c = a * b
In [16]: print(c)
105.0
In [17]: print(100 * 100)
10000
```

## 四則演算の演習

身長と体重を変数に代入してBMIを計算する式を作ってみましょう

$$\text{BMI} = \text{体重(kg)} / (\text{身長(m)})^2$$

#を消して実行してみましょう



## 四則演算の演習

身長と体重を変数に代入してBMIを計算する式を作ってみましょう

$$\text{BMI} = \text{体重(kg)} / (\text{身長(m)})^2$$

```
weight = 62
height = 1.68
BMI = weight / (height ** 2)
print(BMI)
```

```
In [114]: weight = 62
In [115]: height = 1.68
In [116]: BMI = weight / (height ** 2)
In [117]: print(BMI)
21.9671201814059
```

# Pythonの値には「型」という概念が存在する

変数に入れるデータは、整数なのか、小数なのか、文字のデータ(文字列)なのかを区別する必要があります。これを型といい、型には「文字列(str)」、「整数型(int)」、「小数型(float)」、「リスト型(list)」、「タプル型(tuple)」などがあります。  
組み込み関数である`type(引数)`で調べることが出来ます。

```
x = 35
print(type(x))
x = 3.5
print(type(x))
data = '文字だよ'
print(type(data))
```

←xには35(整数)を代入

←type(x)を出力しろという意味

←xに3.5(小数)を代入

←xに'文字だよ'(文字列)を代入

```
In [118]: x = 35
```

```
In [119]: print(type(x))
<class 'int'>
```

```
In [120]: x = 3.5
```

```
In [121]: print(type(x))
<class 'float'>
```

```
In [122]: data = '文字だよ'
```

```
In [123]: print(type(data))
<class 'str'>
```

# リストについて

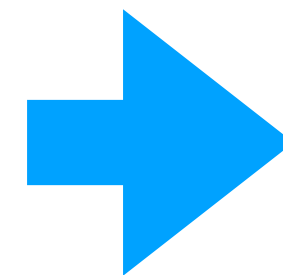
リスト型は、文字列や数値を複数まとめて格納する型です。

データが数多くあった場合、全てを変数に代入して扱うのは大変です。  
リストでは、沢山のデータを1つの変数の中に代入することが出来ます。

変数 = [要素1, 要素2, 要素3, ...]
---------------------------

(リストではそれぞれのデータのことを要素と言います)

a = 1  
b = 1  
c = 2  
d = 4  
e = 6  
f = 8



a = [1, 1, 2, 4, 6, 8]

## #2) リストの作成

```
x = [1,2,3,4,5]
print(x)
print(type(x))
print(x[0])
print(x[4])
x[0] = 10
print(x)
```

← 変数xに要素1に1、要素2に2、要素3に3、要素4に4、要素5に5をリストとして代入

→ [1, 2, 3, 4, 5]

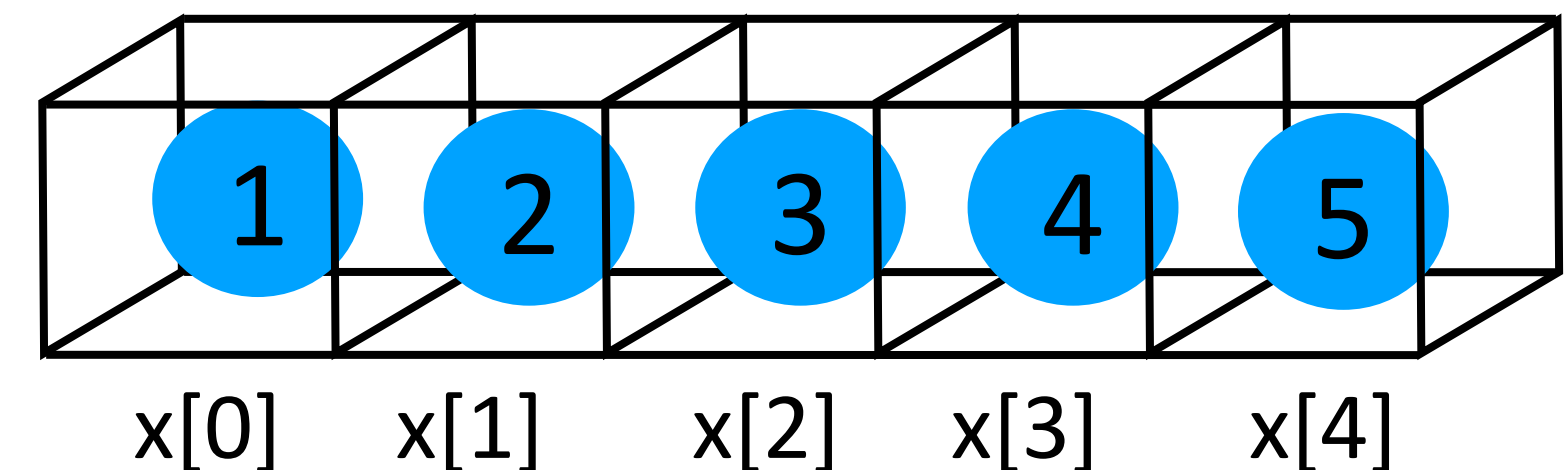
→ list

← xの1番目の要素を表示する

← xの5番目の要素を表示する

→ xの1番目に10を代入する

このリストのように、複数の値を変数に入れて操作できる形のものを配列と言います。



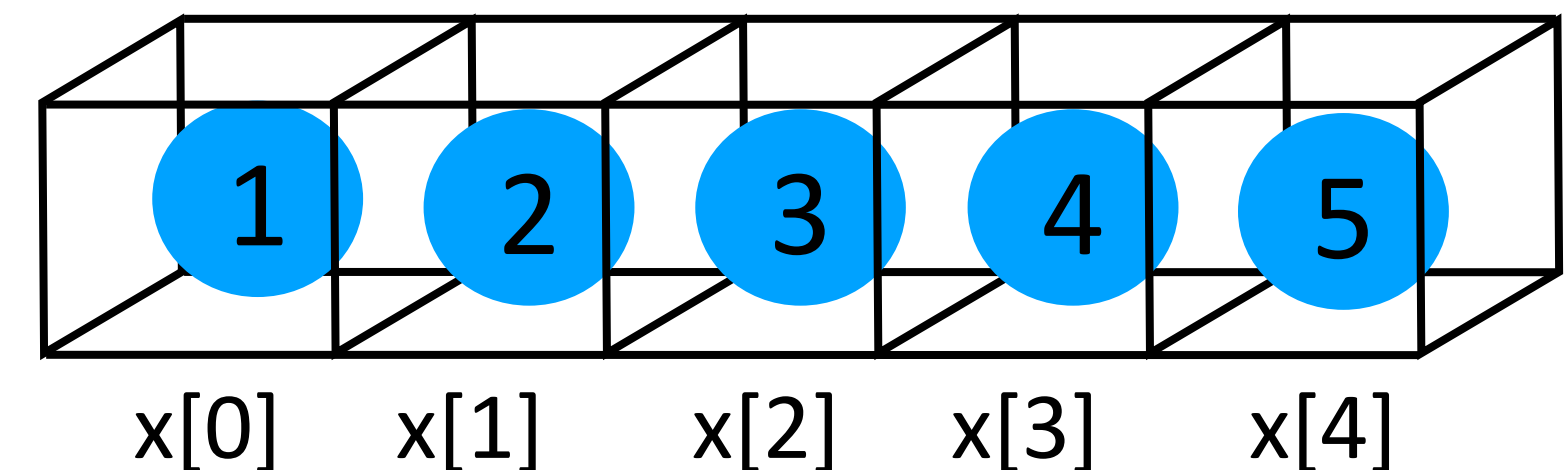
## #2) リストの作成

```
x = [1,2,3,4,5]
print(x)
print(type(x))
print(x[0])
print(x[4])
x[0] = 10
print(x)
```

- ← 変数xに要素1に1、要素2に2、要素3に3、要素4に4、要素5に5をリストとして代入
- [1, 2, 3, 4, 5]
- list
- ← xの1番目の要素を表示する
- ← xの5番目の要素を表示する
- xの1番目に10を代入する

```
In [124]: x = [1,2,3,4,5]
In [125]: print(x)
[1, 2, 3, 4, 5]
In [126]: print(type(x))
<class 'list'>
In [127]: print(x[0])
1
In [128]: print(x[4])
5
In [129]: x[0] = 10
In [130]: print(x)
[10, 2, 3, 4, 5]
```

このリストのように、複数の値を変数に入れて操作できる形のものを配列と言います。



## #2) リストの作成

リストはlen()で要素の個数、max()で要素の最大値、min()で要素の最小値、sum()で要素の合計値を計算してくれます

```
In [54]: print(x)
[10, 2, 3, 4, 5]
```

```
print(len(x))
print(max(x))
print(min(x))
print(sum(x))
```

```
In [55]: print(len(x))
5
```

```
In [56]: print(max(x))
10
```

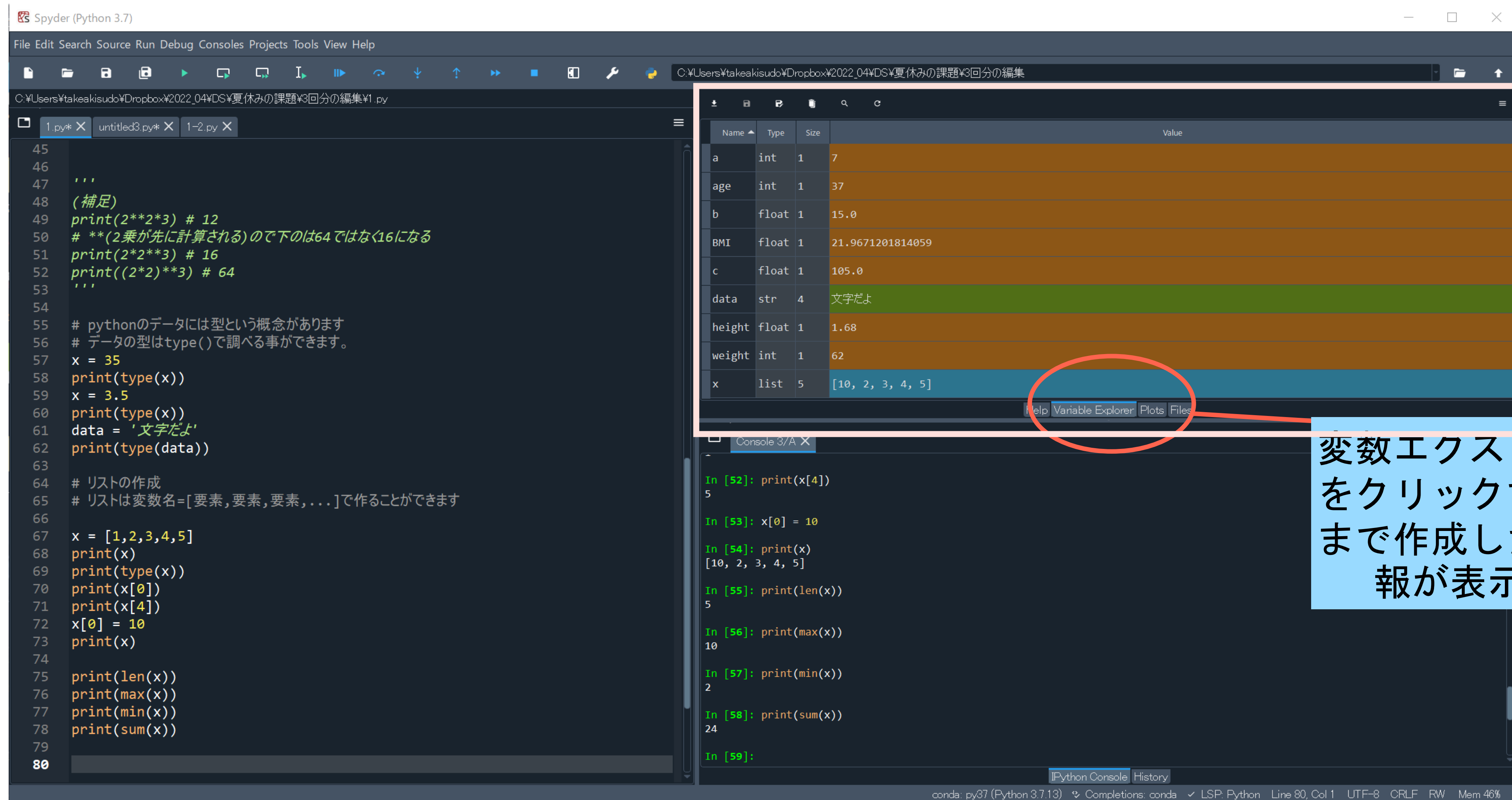
```
In [57]: print(min(x))
2
```

```
In [58]: print(sum(x))
24
```



# 変数エクスプローラーについて

spyderでは変数エクスプローラーで変数の名前、型、値などの情報を得ることができます



The screenshot shows the Spyder Python IDE interface. The left pane contains a Python script with the following code:

```
45
46
47 '''
48 (補足)
49 print(2**2*3) # 12
50 # **(2乗が先に計算される) ので下のは64ではなく16になる
51 print(2*2**3) # 16
52 print((2*2)**3) # 64
53 '''
54
55 # pythonのデータには型という概念があります
56 # データの型はtype()で調べる事ができます。
57 x = 35
58 print(type(x))
59 x = 3.5
60 print(type(x))
61 data = '文字だよ'
62 print(type(data))
63
64 # リストの作成
65 # リストは変数名=[要素,要素,要素,...]で作ることができます
66
67 x = [1,2,3,4,5]
68 print(x)
69 print(type(x))
70 print(x[0])
71 print(x[4])
72 x[0] = 10
73 print(x)
74
75 print(len(x))
76 print(max(x))
77 print(min(x))
78 print(sum(x))
79
80
```

The right pane shows the Variable Explorer panel, which displays a table of variables:

Name	Type	Size	Value
a	int	1	7
age	int	1	37
b	float	1	15.0
BMI	float	1	21.9671201814059
c	float	1	105.0
data	str	4	文字だよ
height	float	1	1.68
weight	int	1	62
x	list	5	[10, 2, 3, 4, 5]

The 'x' variable is highlighted with a red circle. A red arrow points from the 'Variable Explorer' tab in the bottom right to the text box on the right.

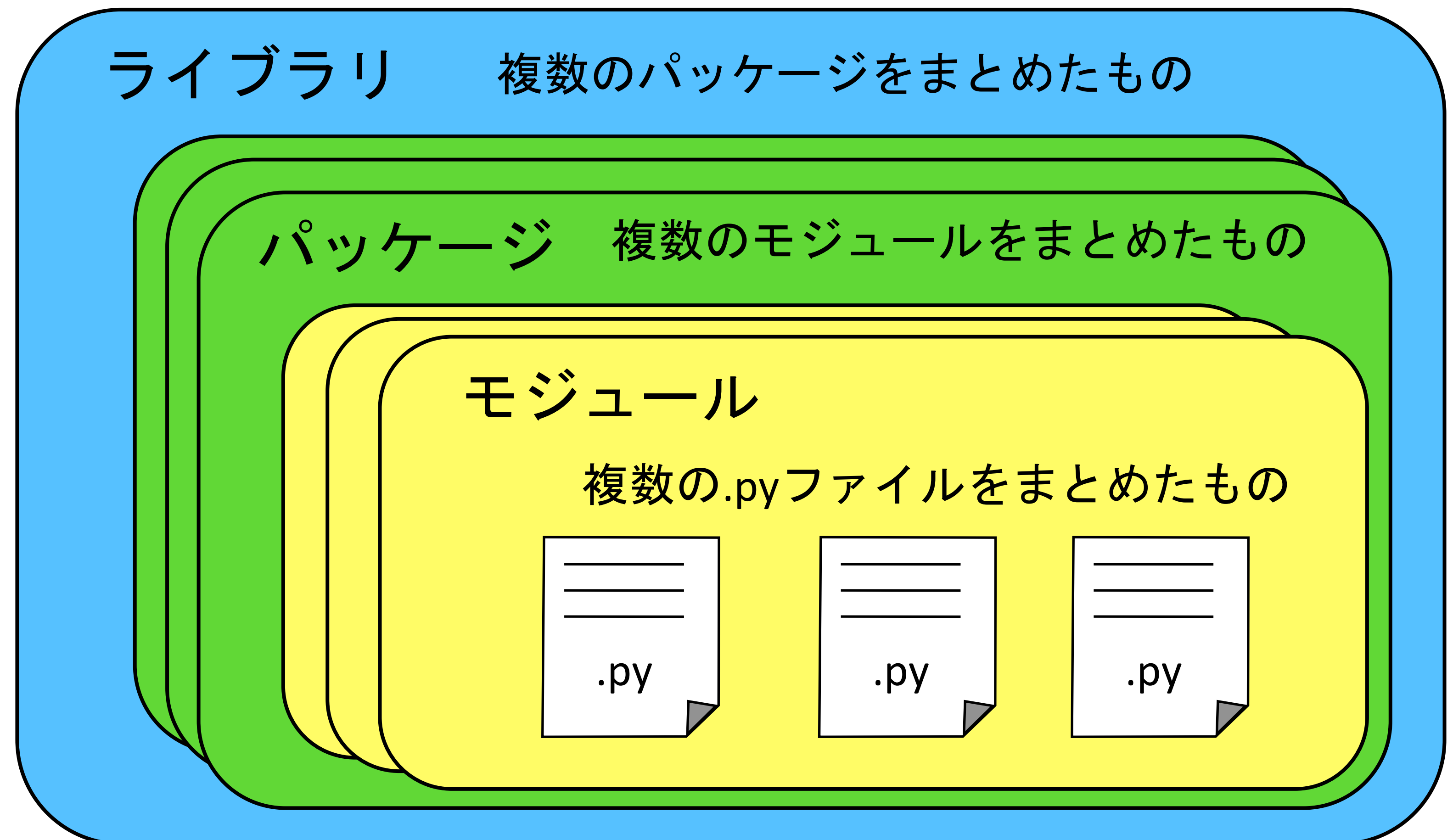
変数エクスプローラーをクリックするとこれまで作成した変数の情報が表示される

# ライブラリについて

Pythonの機能を拡張する機能の1つ

ライブラリを読み込むと、ライブラリに含まれる多くの関数などの機能を追加できる

ここでは次回から多用する  
numpy、matplotlib  
のライブラリを使用してみます



# numpy

numpyは数値計算に優れたライブラリ

```
import numpy as np
```

この1文で、"np"という別名をつけてnumpyを読み込む"という意味になります。

使うときは"np.処理の名前(クラスと言います)"のように、頭にnp.をつけます  
これでそれぞれのライブラリが持つクラスという機能を使用することが出来ます。

1-2.pyのファイルを開いてみましょう

# numpyのarrayについて

numpyのarrayというクラスを使ってみましょう  
リストを用いてarrayを作ります。

変数 = np.array(リスト)

入力

出力

```
# numpyの配列を作るにはnp.arrayと書きます
import numpy as np
test = np.array([1,2,3,4,5])
test2 = [1,2,3,4,5]
print(test)
print(type(test))
```

# numpyのarrayについて

numpyのarrayというクラスを使ってみましょう  
リストを用いてarrayを作ります。

変数 = np.array(リスト)

test	Array of int32	(5,)	[1 2 3 4 5]
test2	list	5	[1, 2, 3, 4, 5]

入力

```
# numpyの配列を作るにはnp.arrayと書きます
import numpy as np
test = np.array([1,2,3,4,5])
test2 = [1,2,3,4,5]
print(test)
print(type(test))
```

出力

```
In [59]: import numpy as np

In [60]: test = np.array([1,2,3,4,5])

In [61]: test2 = [1,2,3,4,5]

In [62]: print(test)
[1 2 3 4 5]

In [63]: print(type(test))
<class 'numpy.ndarray'>
```

numpyが作るデータの型をnumpy配列(ndarray)と言います

# numpyのarrayについて

numpy配列に足し算やかけ算を行うと、配列の全ての要素に対して演算が行われます。

```
test = np.array([1,2,3,4,5])
```

```
test2 = [1,2,3,4,5]
```

入力

```
test = test * 2
test2 = test2 * 2
print(test)
print(test2)
test = test + 2
print(test)
test2 = test2 + 2
```

# numpyのarrayについて

numpy配列に足し算やかけ算を行うと、配列の全ての要素に対して演算が行われます。

```
test = np.array([1,2,3,4,5])  
test2 = [1,2,3,4,5]
```

入力

```
test = test * 2  
test2 = test2 * 2  
print(test)  
print(test2)  
test = test + 2  
print(test)  
test2 = test2 + 2
```

出力

```
In [74]: test = test * 2  
In [75]: test2 = test2 * 2  
  
In [76]: print(test)  
[ 2  4  6  8 10]  
  
In [77]: print(test2)  
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]  
  
In [78]: test = test + 2  
  
In [79]: print(test)  
[ 4  6  8 10 12]  
  
In [80]: test2 = test2 + 2  
Traceback (most recent call last):  
  
  File "C:\Users\takeakisudo\AppData\Local\Temp\ipykernel_12992\589282275.py",  
    line 1, in <module>  
        test2 = test2 + 2  
TypeError: can only concatenate list (not "int") to list
```

# matplotlib

matplotlibは図を書く機能を持ったライブラリです

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

matplotlibの中のpyplotというモジュールをpltと省略してインポートします

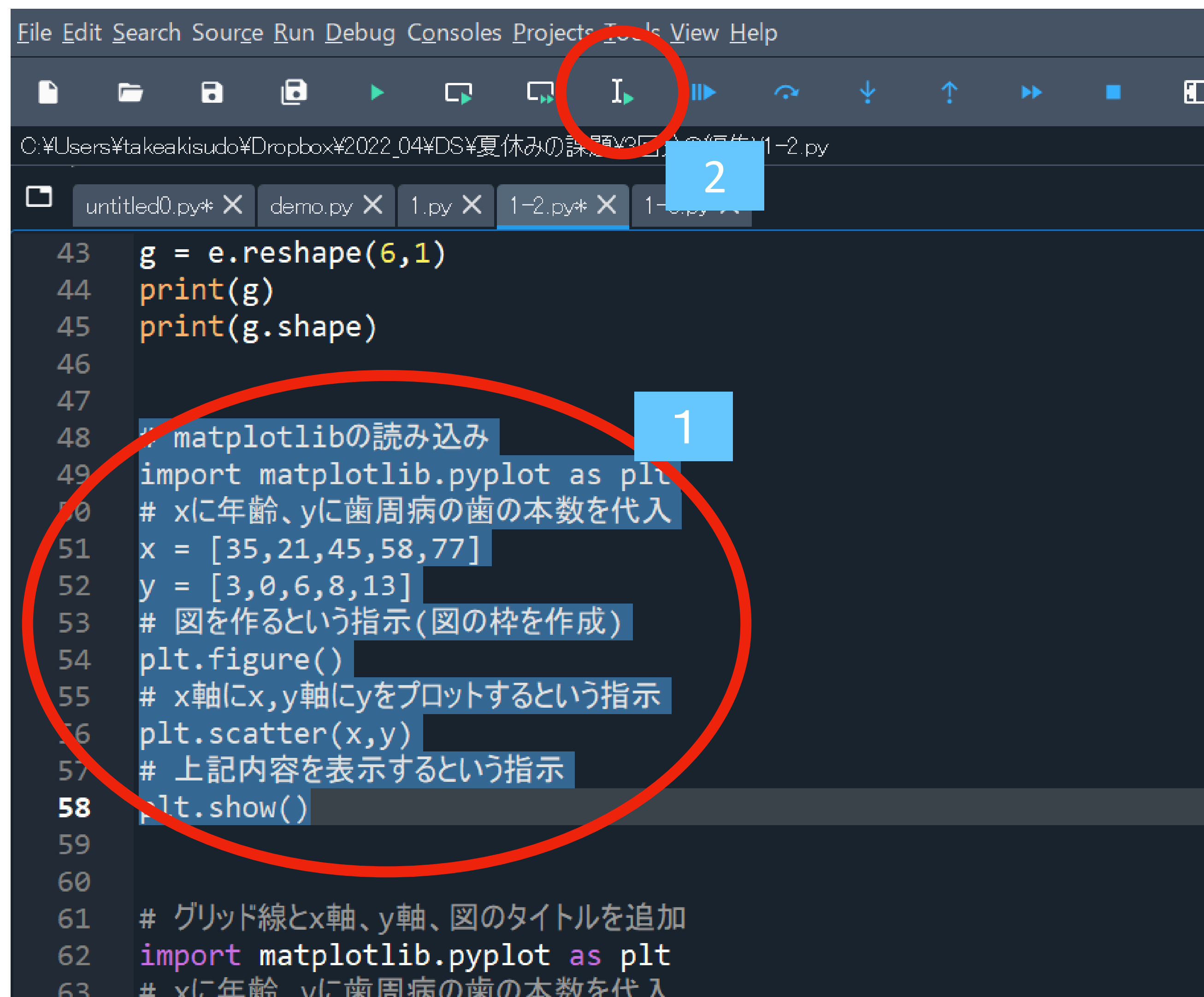
被験者	年齢	歯周病の歯の本数
1	35	3
2	21	0
3	45	6
4	58	8
5	77	13

この様なデータを使って作図をしてみます



# matplotlibの実行

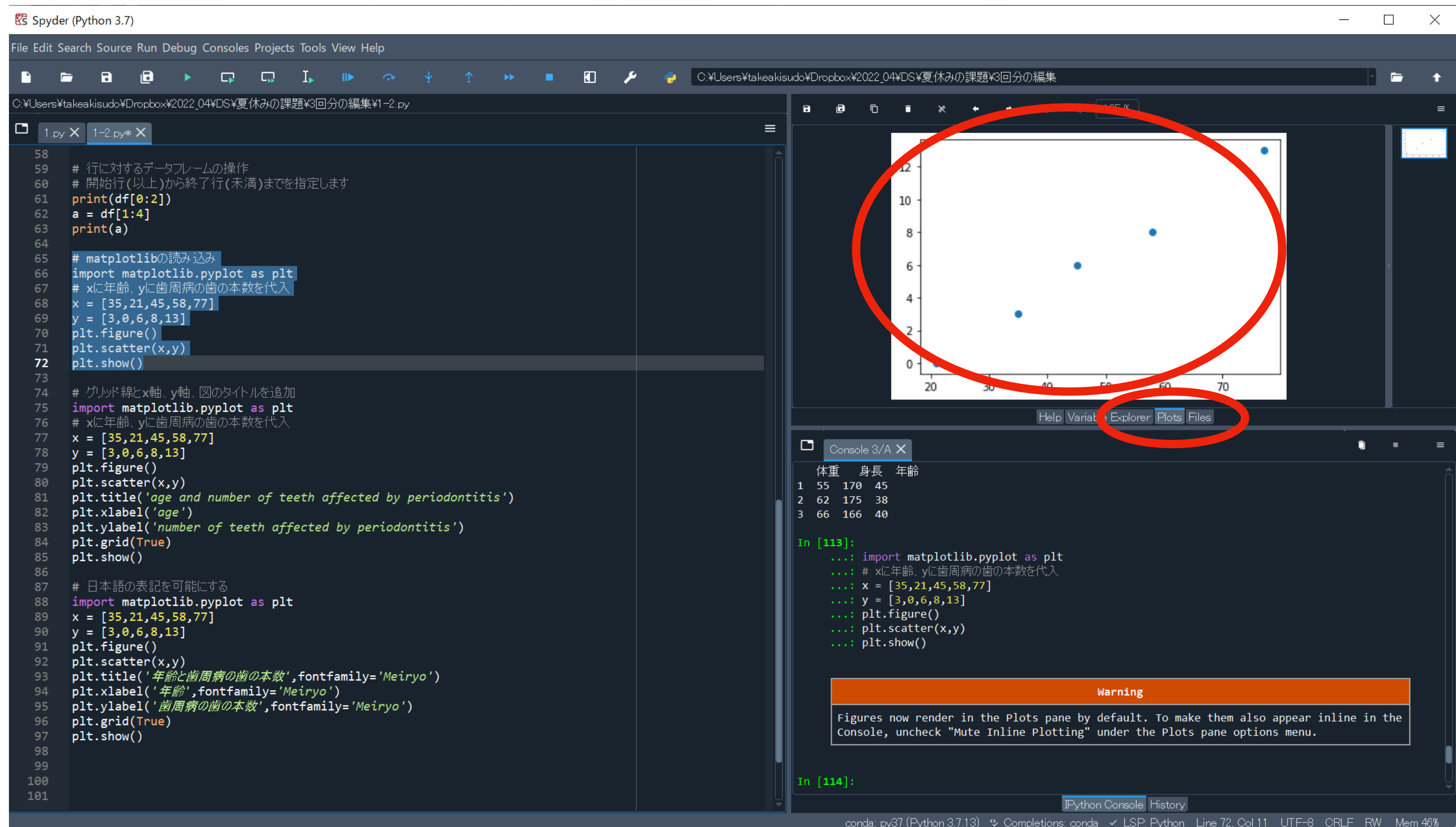
matplotlibは複数行を選んでから(今回は48行目から58行目)実行します



The screenshot shows a code editor with a dark theme. The menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains various icons, with the 'Run' icon (a green play button) circled in red and labeled with a blue box containing the number '2'. The code editor displays a Python script with line numbers 43 to 63. Lines 48 to 58 are highlighted in blue and circled in red, with a blue box containing the number '1' next to them. The code includes comments in Japanese explaining the steps: importing matplotlib.pyplot as plt, defining x and y arrays, creating a figure, plotting a scatter plot, and displaying it.

```
43 g = e.reshape(6,1)
44 print(g)
45 print(g.shape)
46
47
48 # matplotlibの読み込み
49 import matplotlib.pyplot as plt
50 # xに年齢、yに歯周病の歯の本数を代入
51 x = [35,21,45,58,77]
52 y = [3,0,6,8,13]
53 # 図を作るとい指示(図の枠を作成)
54 plt.figure()
55 # x軸にx,y軸にyをプロットするという指示
56 plt.scatter(x,y)
57 # 上記内容を表示するという指示
58 plt.show()
59
60
61 # グリッド線とx軸、y軸、図のタイトルを追加
62 import matplotlib.pyplot as plt
63 # xに年齢、yに歯周病の歯の本数を代入
```

# 実行してみる



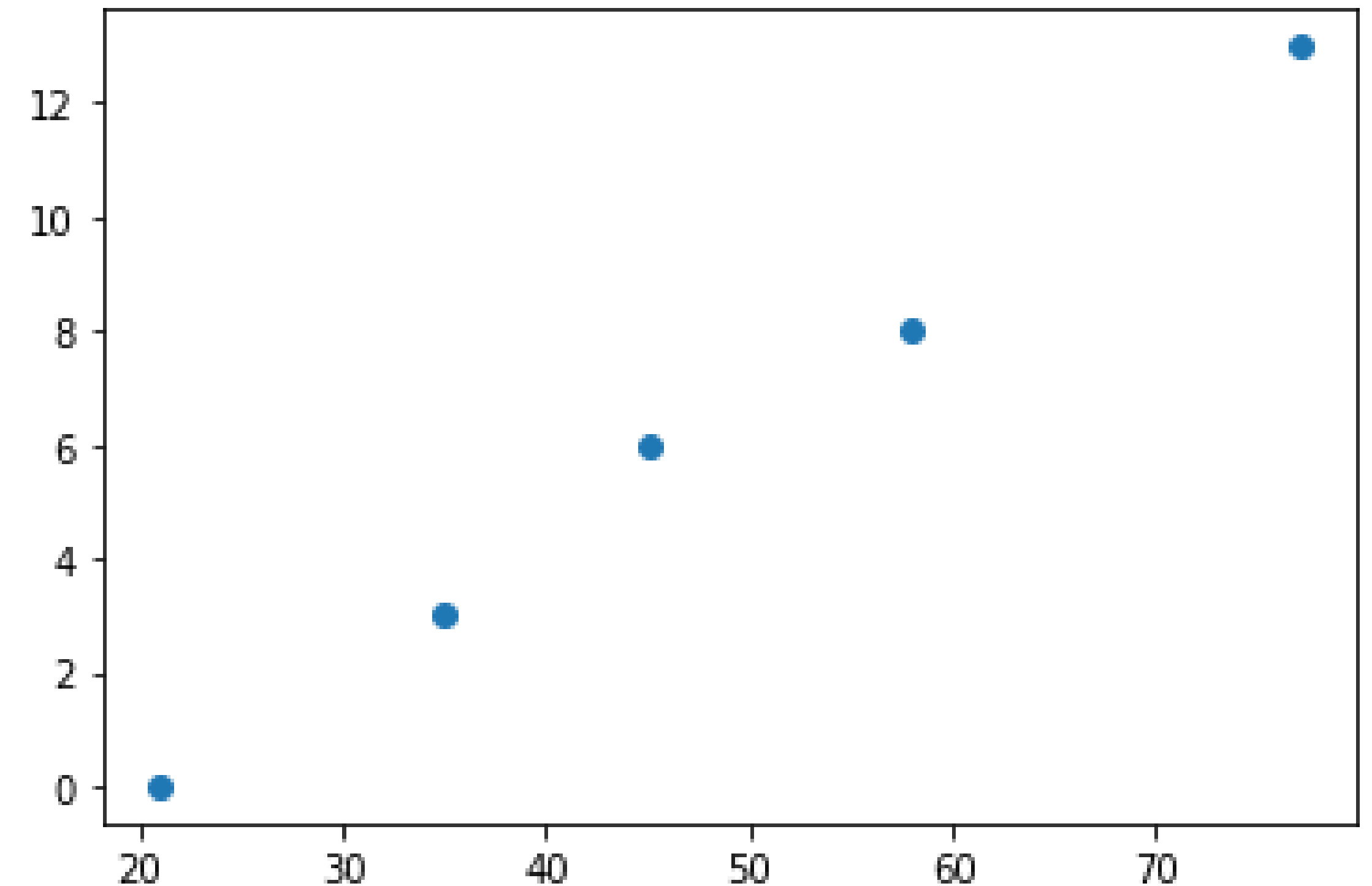
右上のプロットをクリックすると図が表示されます。  
(コンソールにも図が表示出来ますが初期設定では出てきません)

# 点をプロットしてみる(散布図)

## プログラムの中身

```
# matplotlibの読み込み
import matplotlib.pyplot as plt
# xに年齢、yに歯周病の歯の本数を代入
x = [35,21,45,58,77]
y = [3,0,6,8,13]
# 図を作るという指示(図の枠を作成)
plt.figure()
# x軸にx,y軸にyをプロットするという指示
plt.scatter(x,y)
# 上記内容を表示するという指示
plt.show()
```

## 作図の結果



実際に指示しているプログラムは6行  
(他はコメント)

# 点をプロットしてみる(散布図)

## プログラムの中身

```
# matplotlibの読み込み
import matplotlib.pyplot as plt

# xに年齢、yに歯周病の歯の本数を代入
x = [35,21,45,58,77]
y = [3,0,6,8,13]
```

# 点をプロットしてみる(散布図)

## プログラムの中身

```
# matplotlibの読み込み
```

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

```
# xに年齢、yに歯周病の歯の本数を代入
```

```
x = [35,21,45,58,77]
```

```
y = [3,0,6,8,13]
```

被験者	年齢	歯周病の歯の本数
1	35	3
2	21	0
3	45	6
4	58	8
5	77	13

リストとして各変数の値をxとyに代入している

x[0]は35、y[0]は3になる

# 点をプロットしてみる(散布図)

matplotlib(.pyplot)の機能

作図の結果

# 図を作るという指示(図の枠を作成)

```
plt.figure()
```

# x軸にx,y軸にyをプロットするという指示

```
plt.scatter(x,y)
```

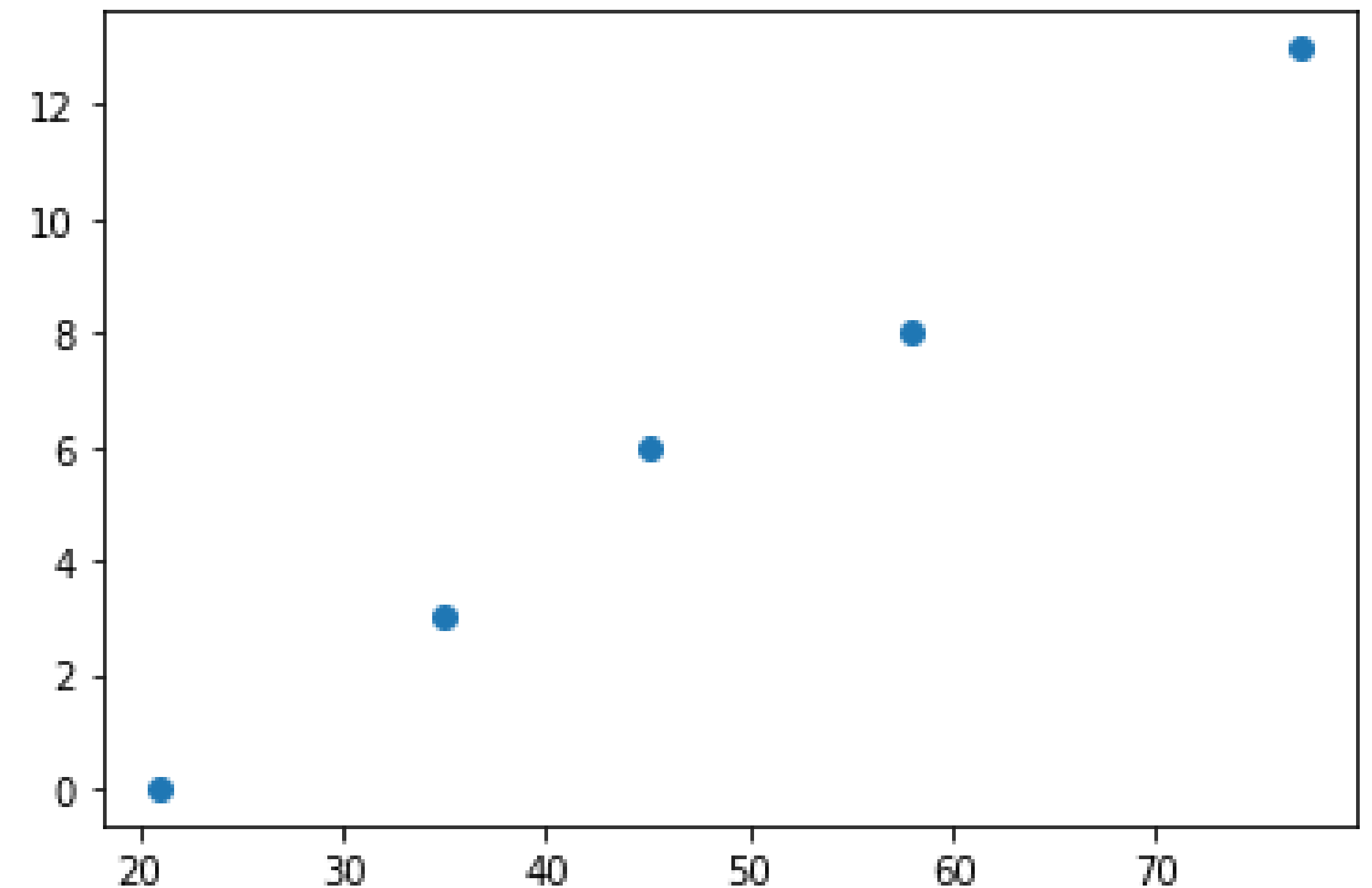
# 上記内容を表示するという指示

```
plt.show()
```

仮に、x=ではなく、

x2 = [35,21,45,58,77]としていたら

plt.scatter(x2, y)とすれば良い



# グリッド線とx軸、y軸、図のタイトルを追加

グリッド線、軸の名前、タイトルを追加します  
まずはすべて選んで実行してみましょう

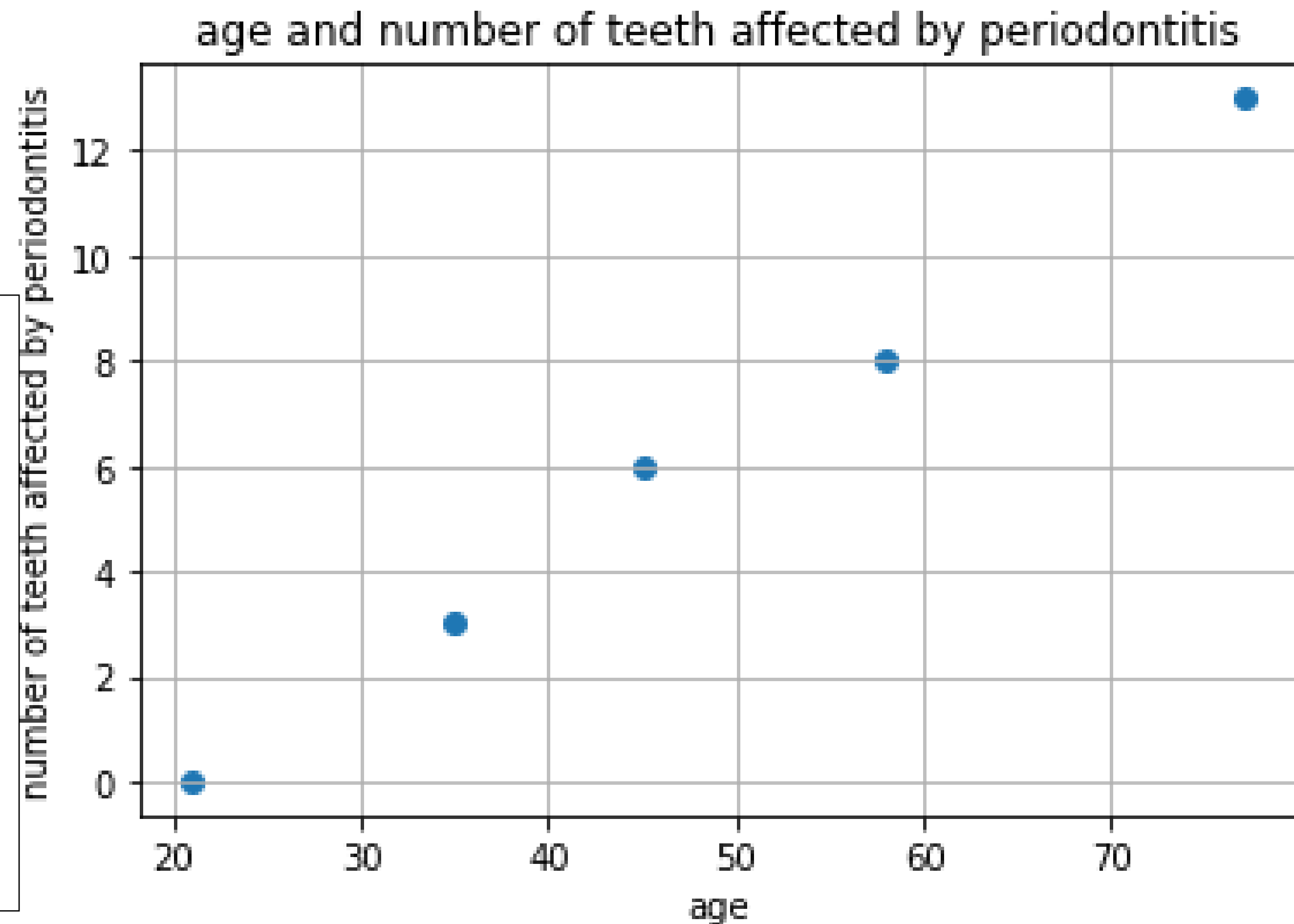
```
# グリッド線とx軸、y軸、図のタイトルを追加
import matplotlib.pyplot as plt
# xに年齢、yに歯周病の歯の本数を代入
x = [35, 21, 45, 58, 77]
y = [3, 0, 6, 8, 13]
plt.figure()
plt.scatter(x, y)
plt.title('age and number of teeth affected by periodontitis')
plt.xlabel('age')
plt.ylabel('number of teeth affected by periodontitis')
plt.grid(True)
plt.show()
```

# 点をプロットしてみる(散布図)

実行すると少し違った図が出てきます

```
plt.title("タイトル名")  
plt.xlabel("x軸の名前")  
plt.ylabel("y軸の名前")  
plt.grid(True)
```

```
# グリッド線とx軸、y軸、図のタイトルを追加  
import matplotlib.pyplot as plt  
# xに年齢、yに歯周病の歯の本数を代入  
x = [35,21,45,58,77]  
y = [3,0,6,8,13]  
plt.figure()  
plt.scatter(x,y)  
plt.title('age and number of teeth affected by periodontitis')  
plt.xlabel('age')  
plt.ylabel('number of teeth affected by periodontitis')  
plt.grid(True)  
plt.show()
```



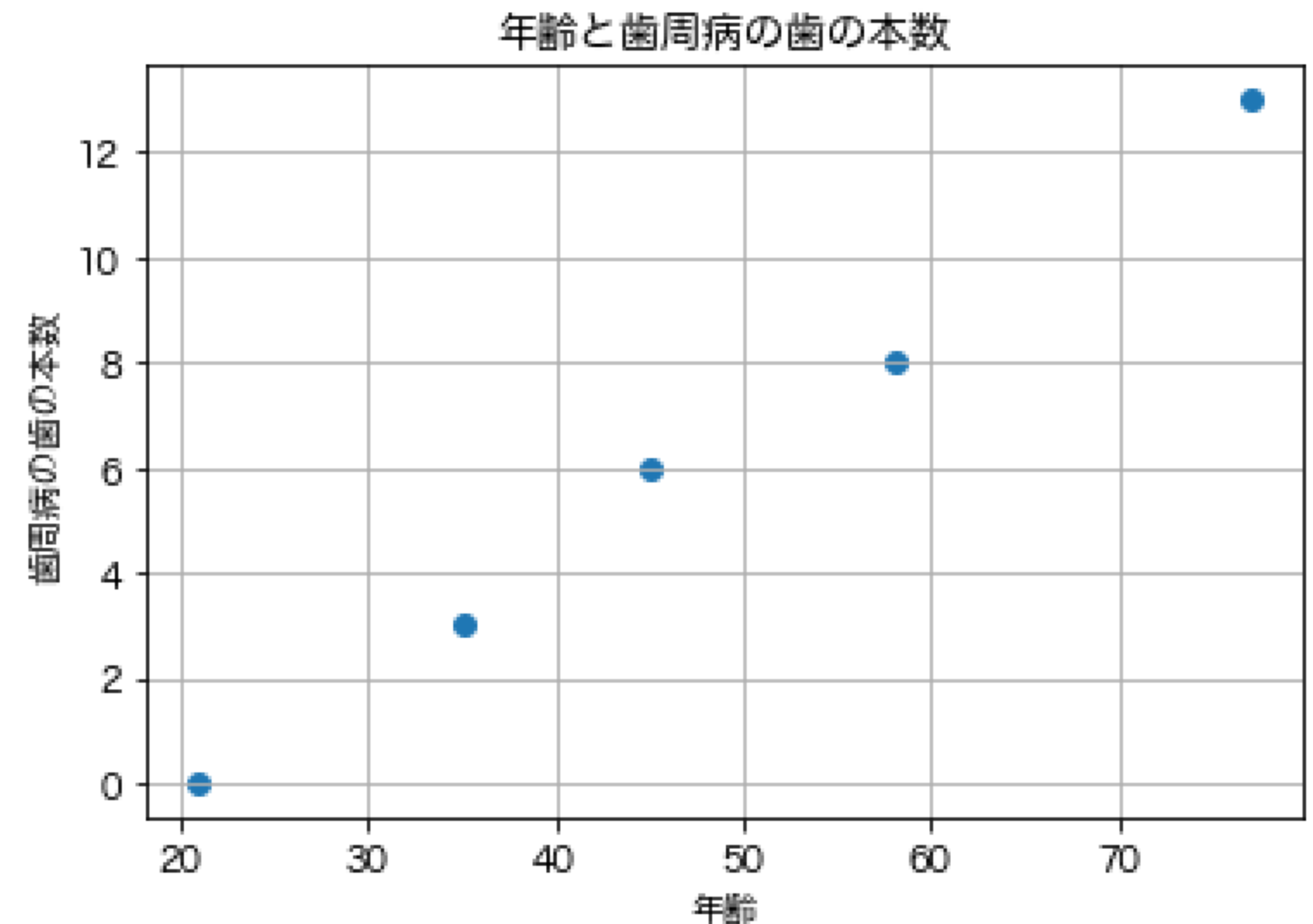


# 日本語表示にする

matplotlibはそのままと日本語表示が出来ずエラーになります  
日本語のフォントのメイリオなどを指定することで日本語で表示することが出来ます

```
# 日本語の表記を可能にする
import matplotlib.pyplot as plt
x = [35,21,45,58,77]
y = [3,0,6,8,13]
plt.figure()
plt.scatter(x,y)
plt.title('年齢と歯周病の歯の本数',fontfamily='Hiragino Maru Gothic Pro')
plt.xlabel('年齢',fontfamily='Hiragino Maru Gothic Pro')
plt.ylabel('歯周病の歯の本数',fontfamily='Hiragino Maru Gothic Pro')
plt.grid(True)
plt.show()
```

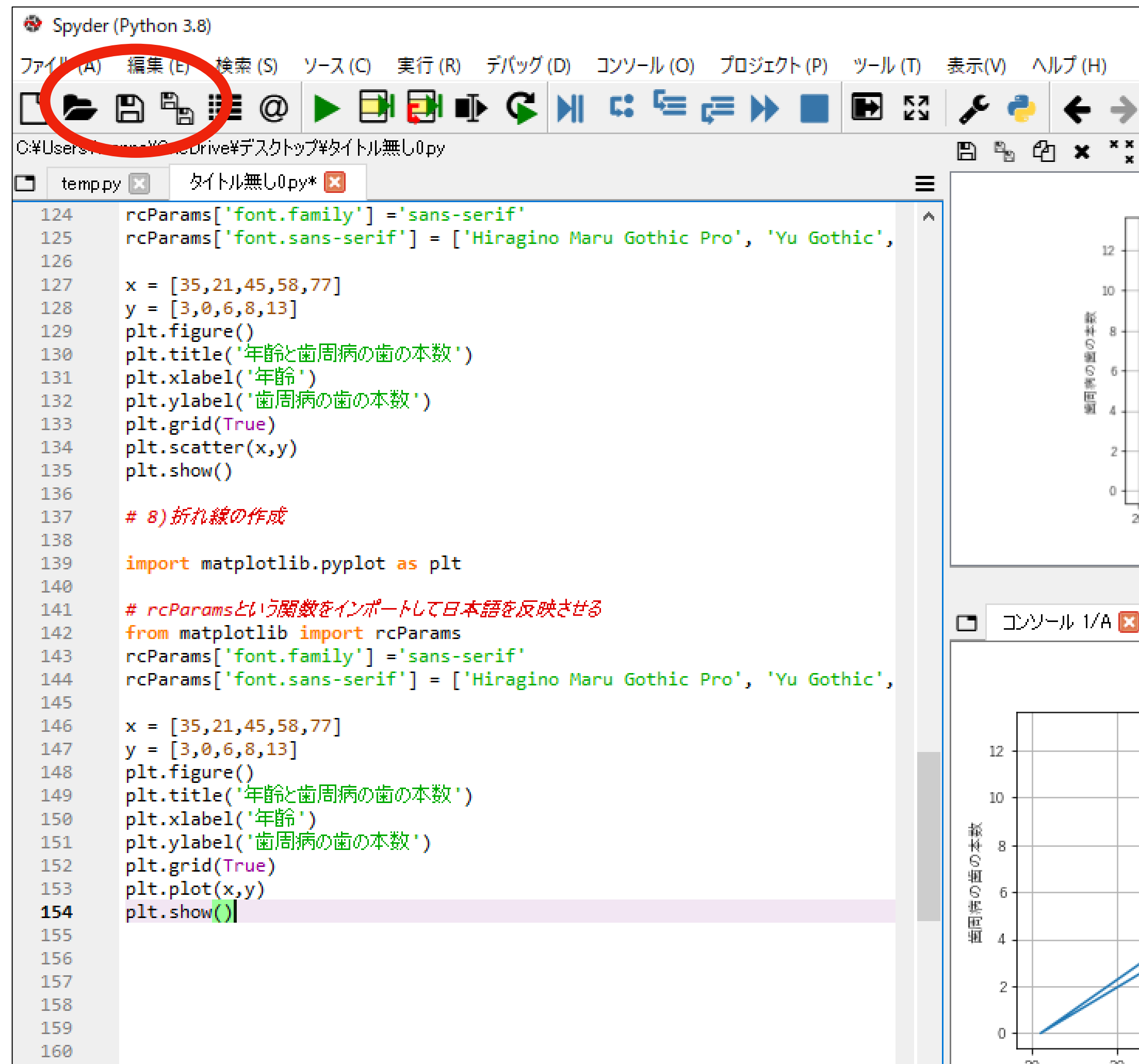
このように色々オプションを加えることで  
色々な図を作ることができます



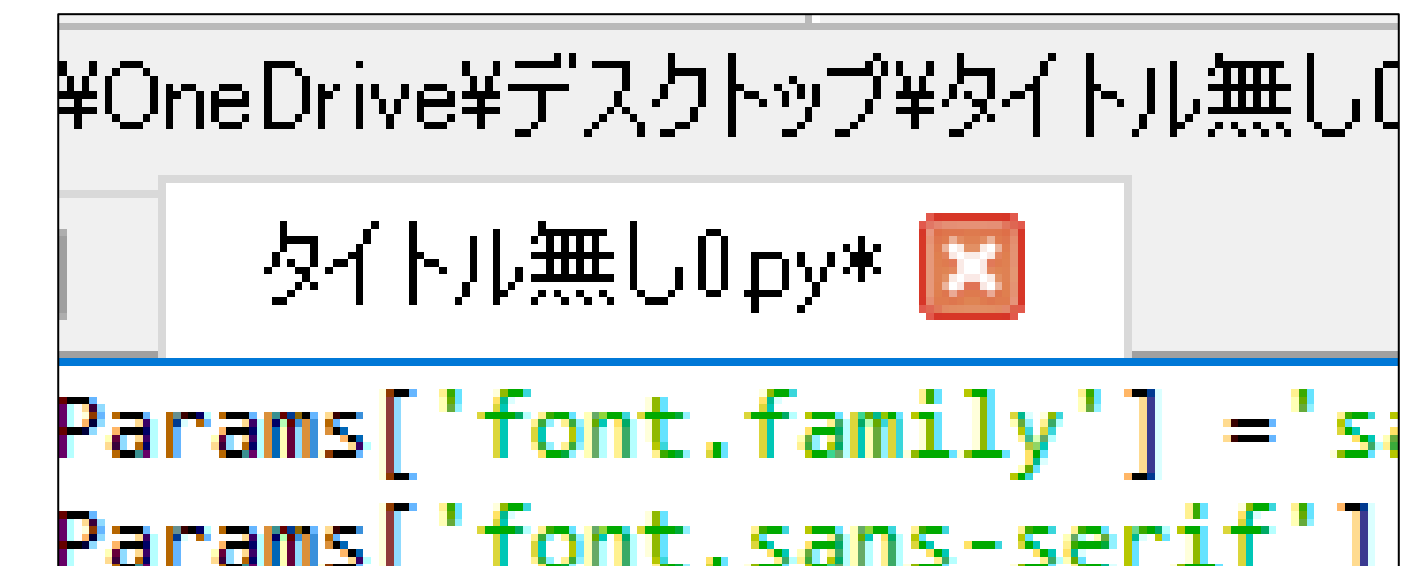
日本語が正しく表記されました

# データの保存

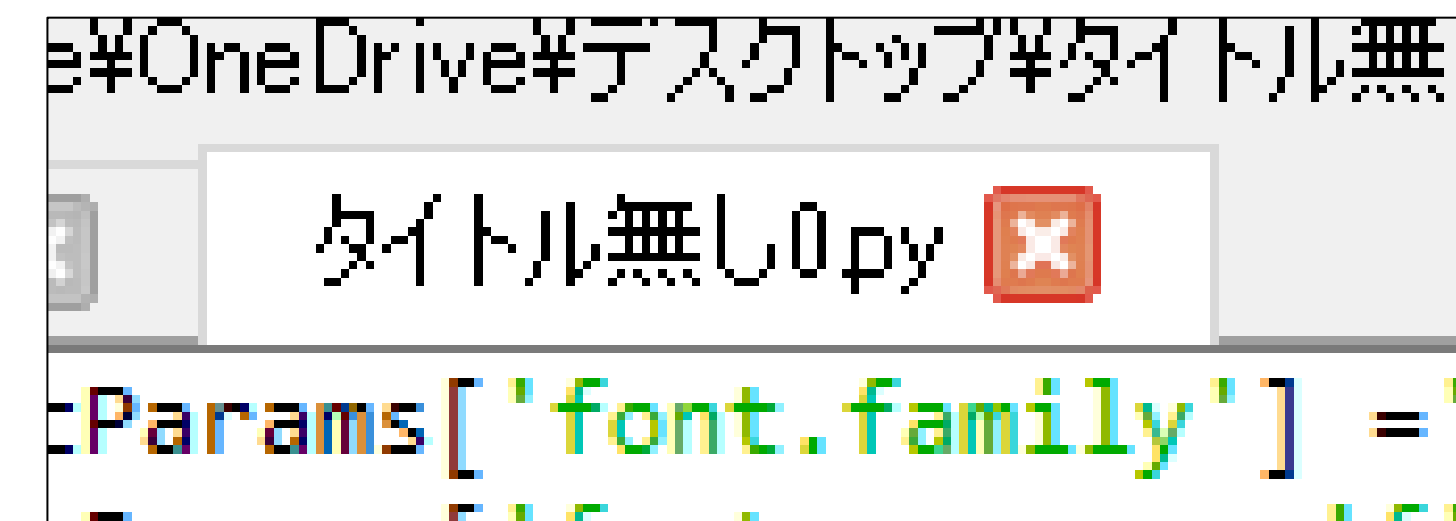
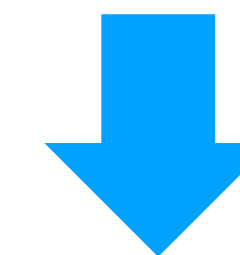
保存するとファイル名の“\*”が消えます。（何か変更すると\*が付きます）



```
124 rcParams['font.family'] = 'sans-serif'
125 rcParams['font.sans-serif'] = ['Hiragino Maru Gothic Pro', 'Yu Gothic',
126
127 x = [35, 21, 45, 58, 77]
128 y = [3, 0, 6, 8, 13]
129 plt.figure()
130 plt.title('年齢と歯周病の歯の本数')
131 plt.xlabel('年齢')
132 plt.ylabel('歯周病の歯の本数')
133 plt.grid(True)
134 plt.scatter(x, y)
135 plt.show()
136
137 # 8) 折れ線の作成
138
139 import matplotlib.pyplot as plt
140
141 # rcParamsという関数をインポートして日本語を反映させる
142 from matplotlib import rcParams
143 rcParams['font.family'] = 'sans-serif'
144 rcParams['font.sans-serif'] = ['Hiragino Maru Gothic Pro', 'Yu Gothic',
145
146 x = [35, 21, 45, 58, 77]
147 y = [3, 0, 6, 8, 13]
148 plt.figure()
149 plt.title('年齢と歯周病の歯の本数')
150 plt.xlabel('年齢')
151 plt.ylabel('歯周病の歯の本数')
152 plt.grid(True)
153 plt.plot(x, y)
154 plt.show()
```



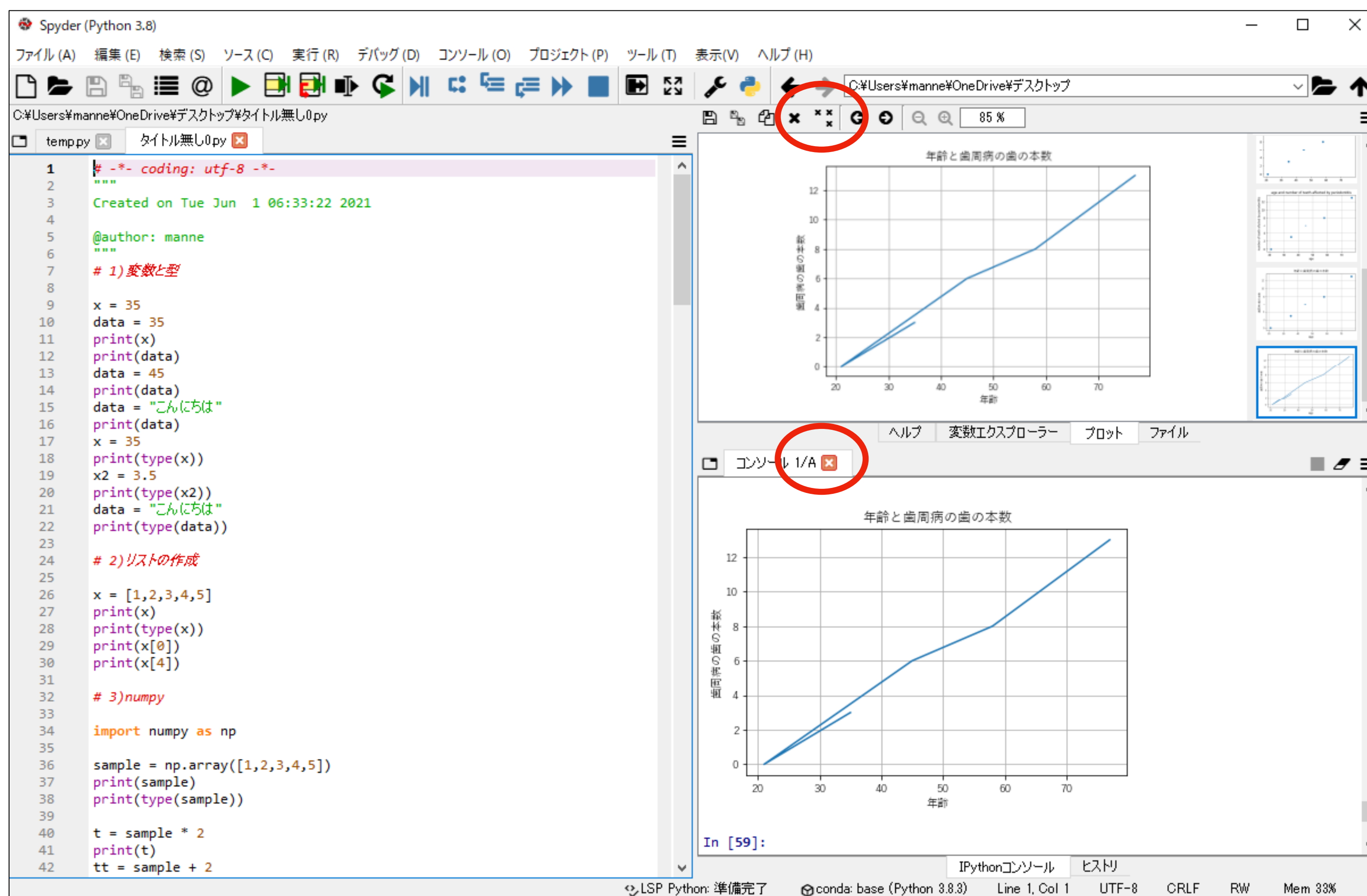
```
#OneDrive#デスクトップ#タイトル無し0
タイトル無し0py*
Params['font.family'] = 's
Params['font.sans-serif']
```



```
#OneDrive#デスクトップ#タイトル無
タイトル無し0py
Params['font.family'] =
```

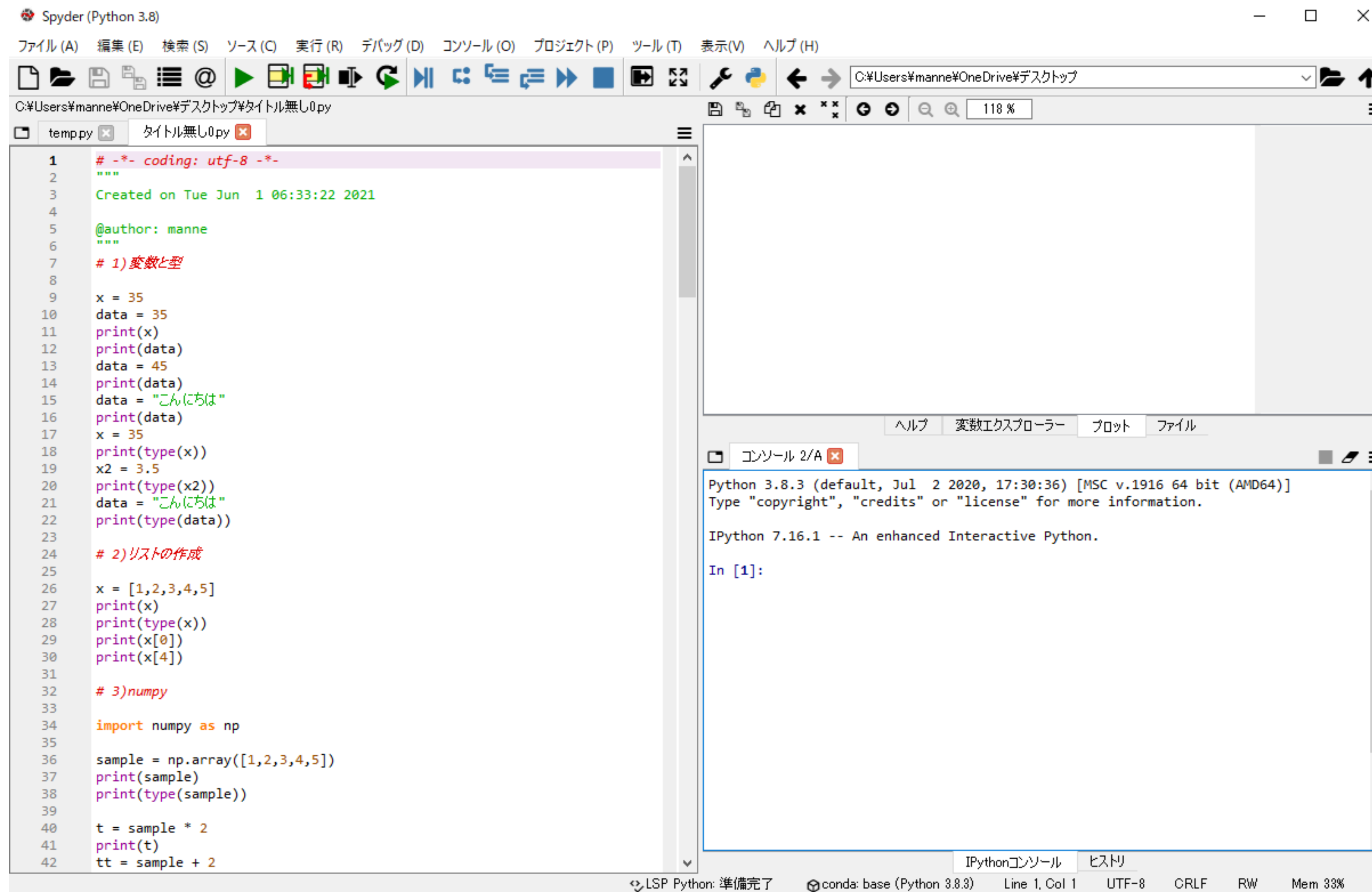
# コンソールのリセット

プロットとコンソールも下のアイコンを押すとリセットすることができます。



# コンソールのリセット

プロットとコンソールも下のアイコンを押すとリセットすることができます。



# 作図を試みよう

xに2, 5, 8, 11, 14, 16, 20

yに1, 4, 8, 11, 15, 30, 55

タイトルを”名前\_学籍番号”

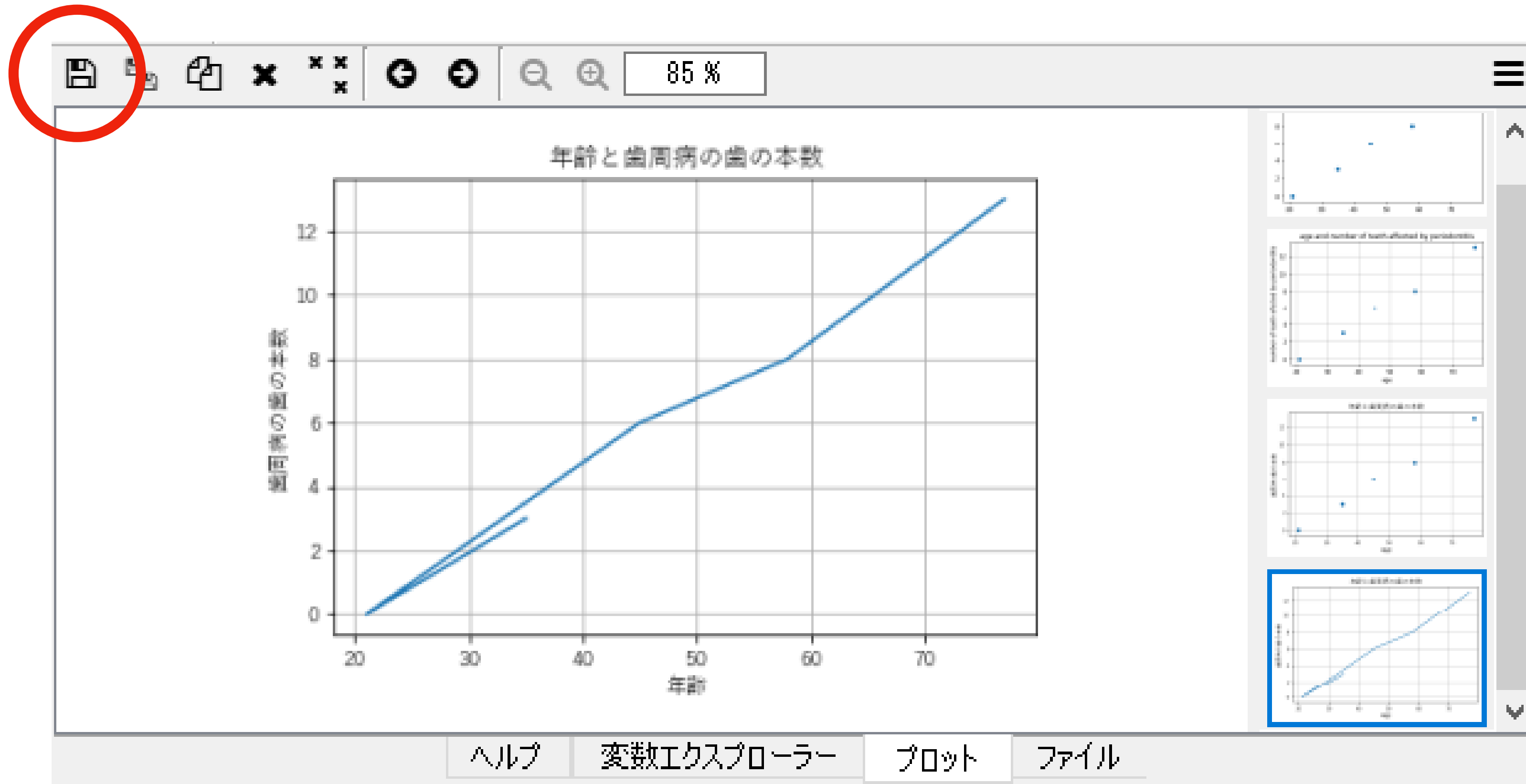
x軸の名前を”X軸”

y軸の名前を”Y軸”

として、散布図と折れ線グラフをそれぞれ作成してみよう

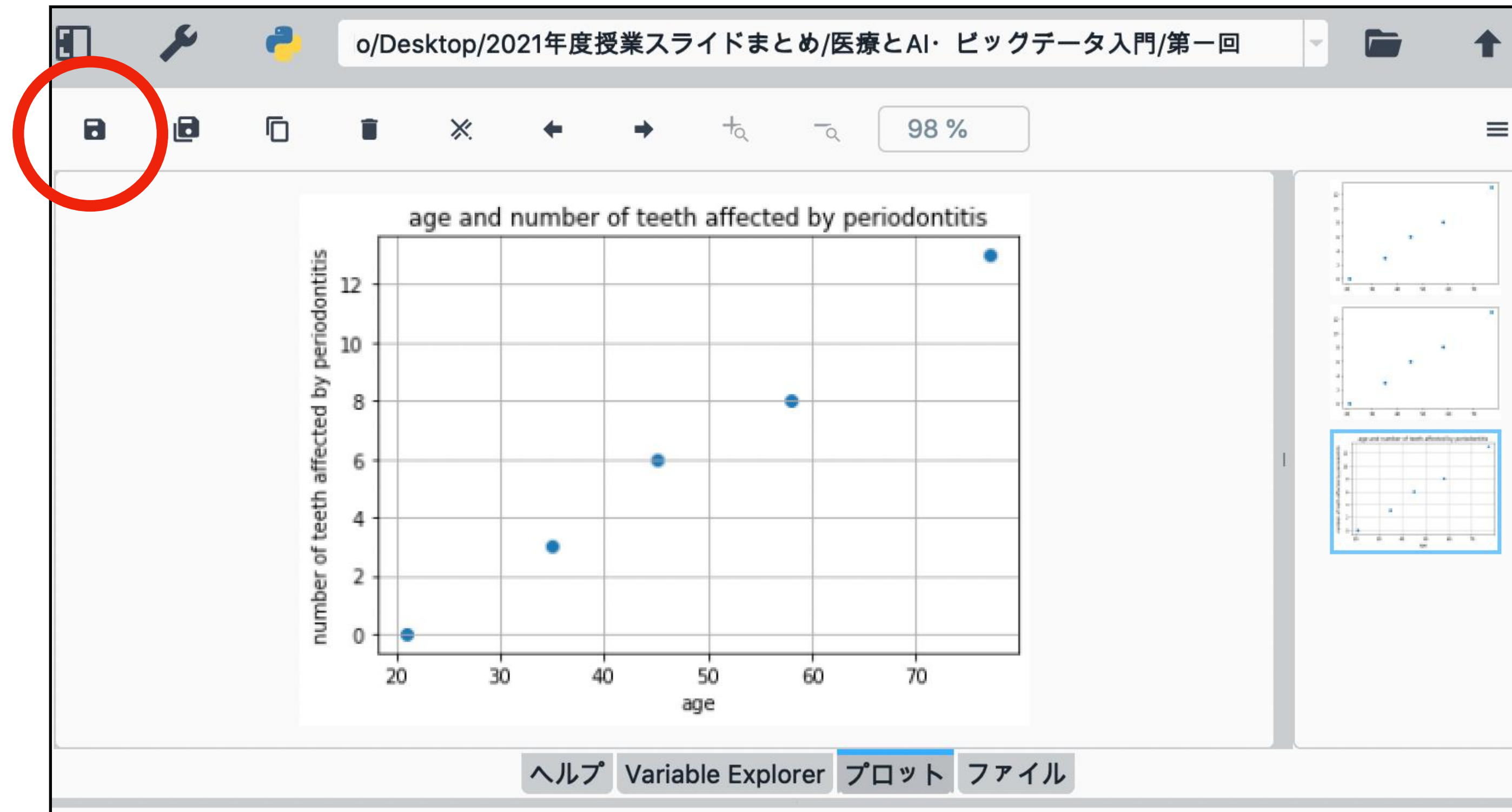
画面を保存して、それぞれ「学籍番号\_名前\_散布図」、  
「学籍番号\_名前\_折れ線」で提出して下さい

# 画像の保存方法



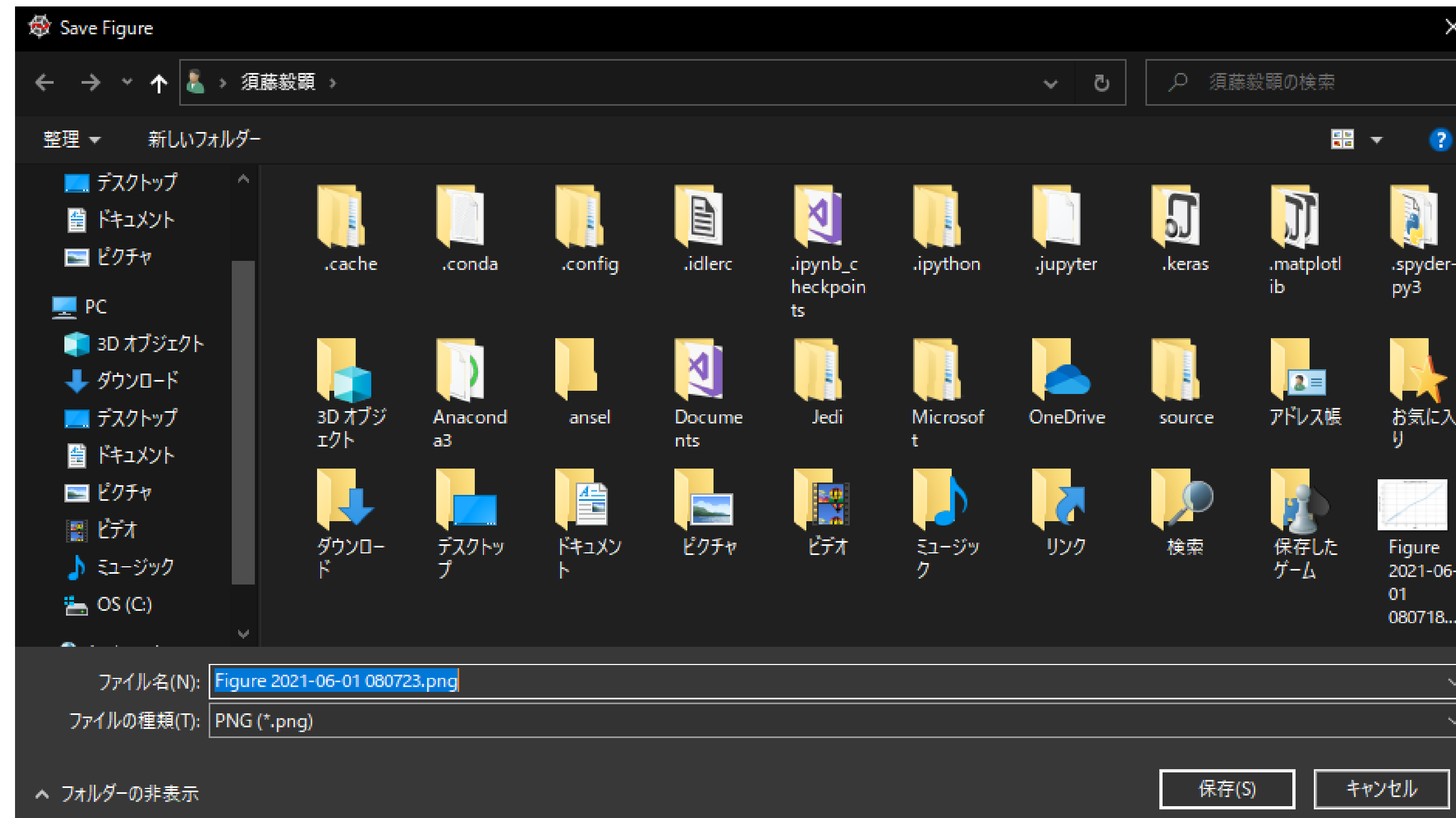
左上の保存アイコンをクリック

# 画像の保存方法



バージョン5も同様

# 画像の保存方法



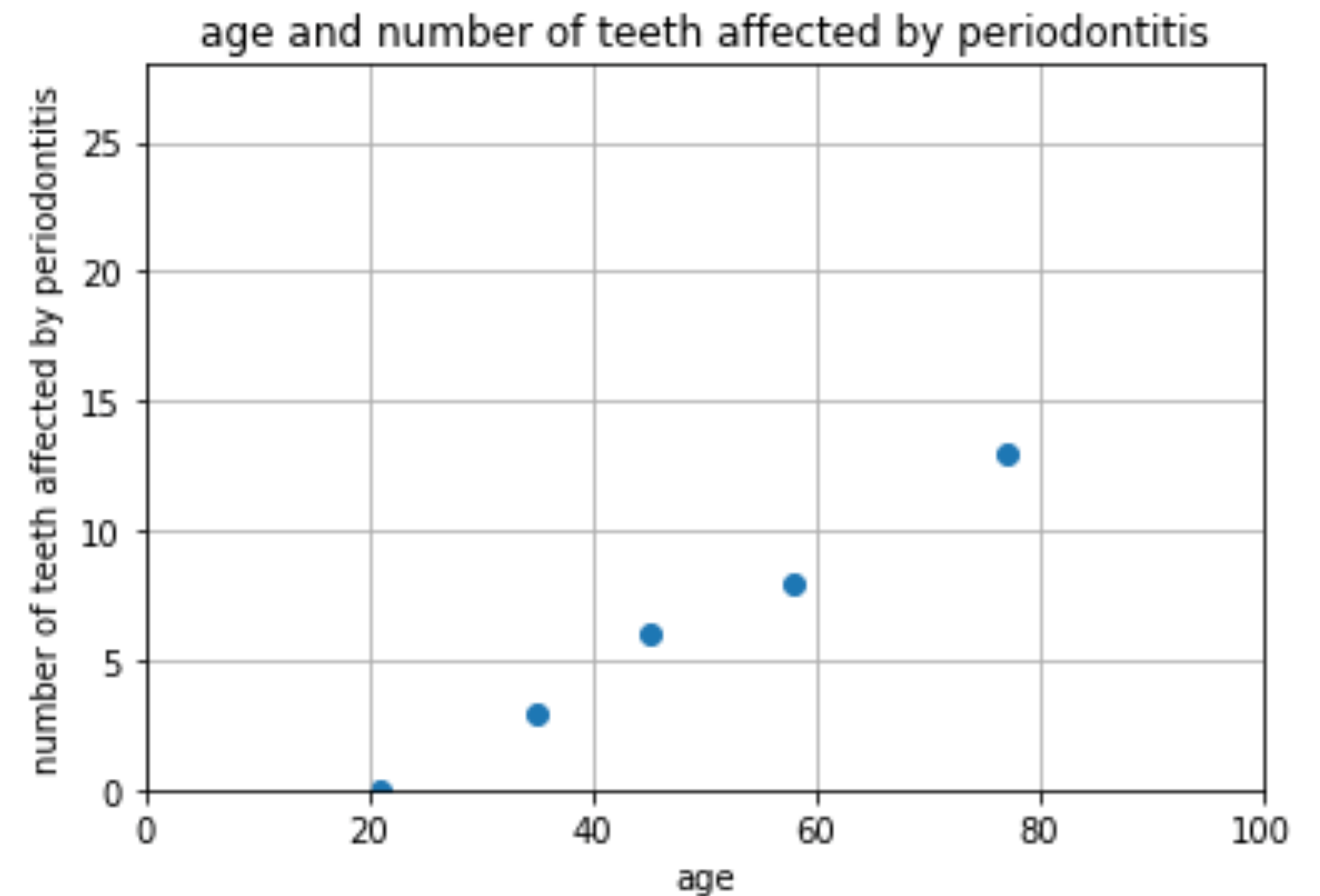
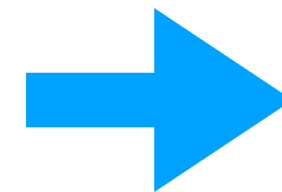
場所を指定して保存



# 前回のデータ

5人の年齢と歯周病の歯の本数を作図するところまで行いました

被験者	年齢	歯周病の歯の本数
1	35	3
2	21	0
3	45	6
4	58	8
5	77	13

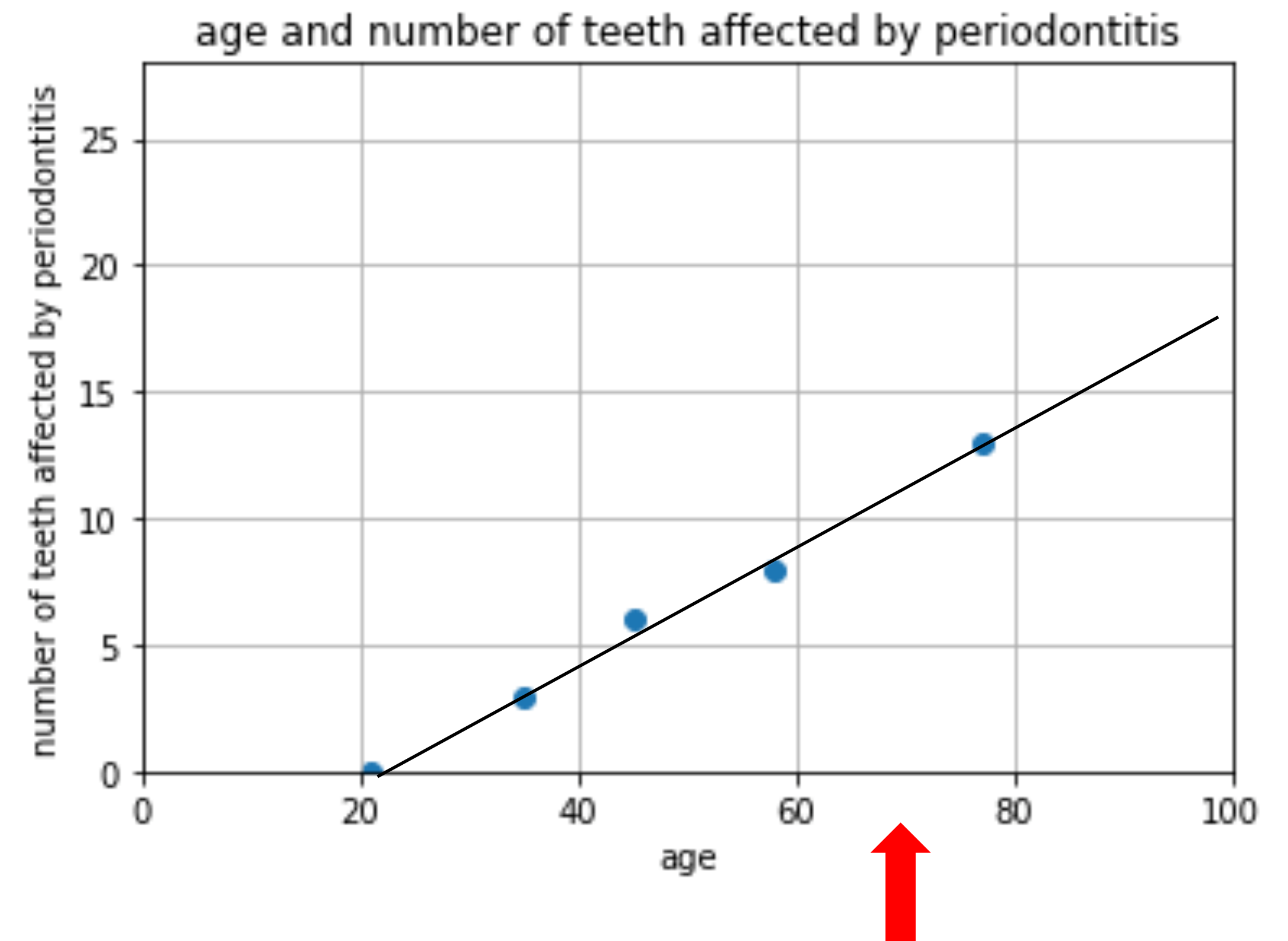
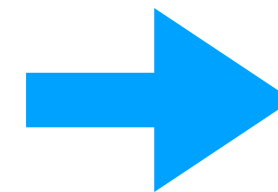


# 前回のデータ

5人の年齢と歯周病の歯の本数を作図するところまで行いました

このデータは直線に近似出来そう！？

被験者	年齢	歯周病の歯の本数
1	35	3
2	21	0
3	45	6
4	58	8
5	77	13



線形回帰で70歳の歯周病の歯の本数を予測する

# 回帰分析

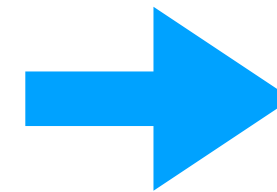
回帰分析は与えられたデータが当てはまるような関数を考える

単回帰分析は直線で表すことが出来る(線形単回帰分析) → 回帰直線

特徴 1 つ

特徴＝説明変数

年齢



正解＝目的変数

歯周病の歯の本数

回帰式

$$y = b_0 + b_1x \quad (y : \text{目的変数、} x : \text{説明変数、} b_0 : \text{切片、} b_1 : \text{傾き})$$

# 線形回帰で88歳の歯周病の歯の本数を予測する

## scikit-learnを用いた線形回帰(機械学習)の書き方

- ①学習モデルの選択
- ②データを入れて学習させる
- ③傾き(偏回帰係数)と切片(定数項)を求める
- ④予測を行う

# 線形回帰で88歳の歯周病の歯の本数を予測する

## scikit-learnを用いた線形回帰(機械学習)の書き方

- ① 学習モデルの選択(今回は線形回帰)  
(モデル名) = `LinearRegression()`
- ② データを入れて学習させる  
(モデル名).`fit`(説明変数, 目的変数)
- ③ 傾き(偏回帰係数)と切片(定数項)を求める  
(モデル名).`coef_`            #傾き  
(モデル名).`intercept_`    #切片
- ④ 予測を行う  
(モデル名).`predict`(新たな説明変数)

# scikit-learnを用いた線形回帰分析

## 1-3.pyを開きましょう

```
from sklearn.linear_model import LinearRegression
```

```
x = [[35],[21],[45],[58],[77]]  
y = [3,0,6,8,13]
```

```
model = LinearRegression()
```

```
model.fit(x,y)
```

```
print(model.coef_)  
print(model.intercept_)
```

```
test = [[70]]  
num_teeth = model.predict(test)  
print("70歳の時の本数は",num_teeth,"本")
```

① 学習モデルの選択(今回は線形回帰)  
(モデル名) = `LinearRegression()`

② データを入れて学習させる  
(モデル名).`fit`(説明変数,目的変数)

③ 傾き(偏回帰係数)と切片(定数項)を求める  
(モデル名).`coef_`            #傾き  
(モデル名).`intercept_`    #切片

④ 予測を行う  
(モデル名).`predict`(新たな説明変数)

# scikit-learnを用いた線形回帰分析

```
from sklearn.linear_model import LinearRegression
```

```
x = [[35],[21],[45],[58],[77]]  
y = [3,0,6,8,13]
```

```
model = LinearRegression()
```

```
model.fit(x,y)
```

```
print(model.coef_)  
print(model.intercept_)
```

```
test = [[70]]  
num_teeth = model.predict(test)  
print("700歳の時の本数は",num_teeth,"本")
```

① 学習モデルの選択(今回は線形回帰)  
(モデル名) = `LinearRegression()`

② データを入れて学習させる  
(モデル名).`fit`(説明変数,目的変数)

③ 傾き(偏回帰係数)と切片(定数項)を求める  
(モデル名).`coef_` #傾き  
(モデル名).`intercept_` #切片

④ 予測を行う  
(モデル名).`predict`(新たな説明変数)

$x = [[35],[21],[45],[58],[77]]$ ,  $y = [3,0,6,8,13]$ として、  
説明変数を $x$ （年齢）、目的変数を $y$ (歯の本数) に代入

(Scikit-learnを使うときは説明変数のデータを2次元配列にする)

# 線形回帰で88歳の歯周病の歯の本数を予測する

## scikit-learnを用いた機械学習の書き方

①学習モデルの選択(今回は線形回帰)  
(モデル名) = `LinearRegression()`

`LinearRegression()`をモデル名(変数)に代入することで  
scikit-learnの`LinearRegression()`という  
機能を使うことが出来る

モデル名は何でも良い

```
model = LinearRegression()
```



# 線形回帰で88歳の歯周病の歯の本数を予測する

## scikit-learnを用いた機械学習の書き方

② データを入れて学習させる  
(モデル名).fit(説明変数, 目的変数)

今回は、モデル名をmodel、説明変数をx(年齢)、  
目的変数をy(歯周病の歯の本数)としたい

```
model.fit(x,y)
```

modelは線形回帰を選んでいるので、  
これでxとyを用いて線形回帰による学習を行う

# 線形回帰で88歳の歯周病の歯の本数を予測する

## scikit-learnを用いた機械学習の書き方

③傾き(偏回帰係数)と切片(定数項)を求める  
(モデル名).coef\_            #傾き  
(モデル名).intercept\_      #切片

線形回帰での傾きと切片を求める。  
中身を出力したいので、print()を用いる

```
print(model.coef_)
```

```
print(model.intercept_)
```

# 線形回帰で88歳の歯周病の歯の本数を予測する

## scikit-learnを用いた機械学習の書き方

④ 予測を行う  
(モデル名).**predict**(新たな説明変数)

70才の時の歯の本数を知りたいので、

```
test = [[70]]  
num_teeth = model.predict(test)  
print("70才の時の本数は", num_teeth, "本")
```

# 線形回帰で88歳の歯周病の歯の本数を予測する

## scikit-learnを用いた機械学習の書き方

④ 予測を行う  
(モデル名).**predict**(新たな説明変数)

70才の時の歯の本数を知りたいので、

```
test = [[70]]  
num_teeth = model.predict([[70]])  
print("70才の時の本数は", model.predict([[70]]), "本")
```

# 線形回帰で88歳の歯周病の歯の本数を予測する

```
from sklearn.linear_model import LinearRegression
```

```
x = [[35],[21],[45],[58],[77]]
```

```
y = [3,0,6,8,13]
```

```
model = LinearRegression()
```

```
model.fit(x,y)
```

```
print(model.coef_)
```

```
print(model.intercept_)
```

```
test = [[70]]
```

```
num_teeth = model.predict(test)
```

```
print("70歳の時の本数は",num_teeth,"本")
```

① 学習モデルの選択(今回は線形回帰)  
(モデル名) = `LinearRegression()`

② データを入れて学習させる  
(モデル名).`fit`(説明変数,目的変数)

③ 傾き(偏回帰係数)と切片(定数項)を求める  
(モデル名).`coef_`            #傾き  
(モデル名).`intercept_`    #切片

④ 予測を行う  
(モデル名).`predict`(新たな説明変数)

# 線形回帰で88歳の歯周病の歯の本数を予測する

```
from sklearn.linear_model import LinearRegression
```

```
x = [[35],[21],[45],[58],[77]]
```

```
y = [3,0,6,8,13]
```

```
model = LinearRegression()
```

```
model.fit(x,y)
```

```
print(model.coef_)
```

```
print(model.intercept_)
```

```
test = [[70]]
```

```
num_teeth = model.predict(test)
```

```
print("70歳の時の本数は",num_teeth,"本")
```

```
[[0.22983521]]
```

```
[-4.84822203]
```

```
70歳の時の本数は [[11.24024284]] 本
```

①学習モデルの選択(今回は線形回帰)  
(モデル名) = LinearRegression()

②データを入れて学習させる  
(モデル名).fit(説明変数,目的変数)

③傾き(偏回帰係数)と切片(定数項)を求める  
(モデル名).coef\_            #傾き  
(モデル名).intercept\_    #切片

④予測を行う  
(モデル名).predict(新たな説明変数)

# 補足 Python3.7以降での一般的な書き方  
print(f"70歳の時の本数は{num\_teeth}本")  
(フォーマット済み文字リテラルと言います。)

## 線形回帰で88歳の歯周病の歯の本数を予測する

```
[[0.22983521]]  
[-4.84822203]  
88歳の時の本数は [[15.37727667]] 本
```

$$y = b_0 + b_1x \quad (y : \text{目的変数、} x : \text{説明変数、} b_0 : \text{切片、} b_1 : \text{傾き})$$

$$y = (-4.84822203) + (0.22983521) x$$

線形回帰分析を行い、学習によってこの式が算出された

この式をもとに、`model.predict()`で70歳の時は11.24本と予測された

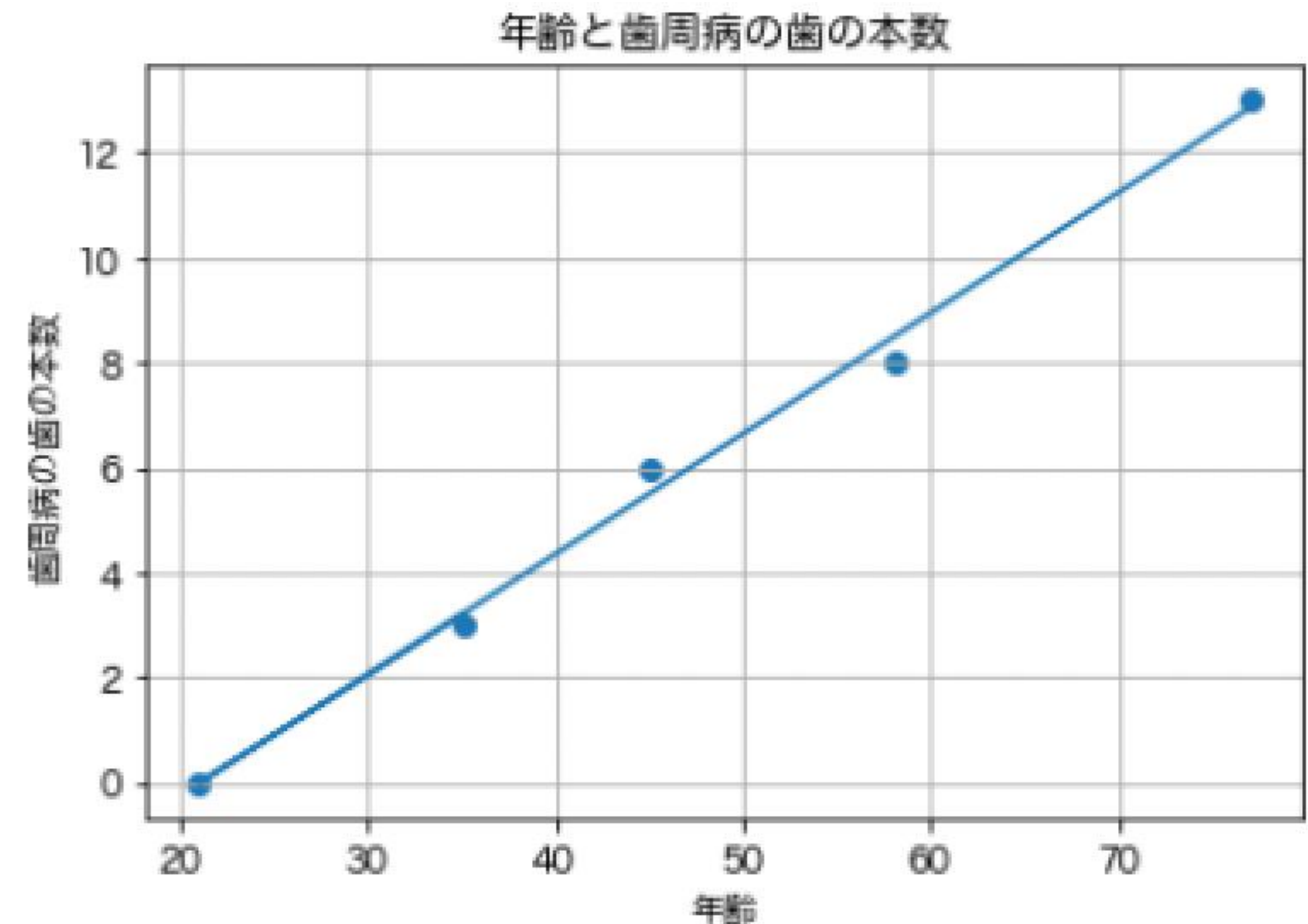


# 回帰直線を作図する

`plt.scatter(x,y)`が実際の(x,y)の点  
`plt.plot(x,model.predict(x))`がモデルが予測した点をつないだ回帰直線となります。

```
import matplotlib.pyplot as plt

plt.figure()
plt.title('年齢と歯周病の歯の本数',fontfamily='Hiragino Maru Gothic Pro')
plt.xlabel('年齢',fontfamily='Hiragino Maru Gothic Pro')
plt.ylabel('歯周病の歯の本数',fontfamily='Hiragino Maru Gothic Pro')
plt.grid(True)
plt.scatter(x,y)
plt.plot(x,model.predict(x))
# plt.scatter(test,num_teeth)
plt.show()
```





# 回帰直線を作図する

```
x = [[35],[21],[45],[58],[77]]  
y = [3,0,6,8,13]
```

```
plt.scatter(x,y)
```

x軸y軸に

35と3、21と0、45と6、58と8、77と13  
の点を打つ(散布図といいます)

```
plt.plot(x,model.predict(x))
```

x軸y軸に

35とmodel.predict([[35]])、  
21とmode.predict([[21]])、  
45とmodel.predict([[45]])、  
58とmodel.predict([[58]])、  
77とmodel.predict([[77]])

を通る直線(or曲線)を書く

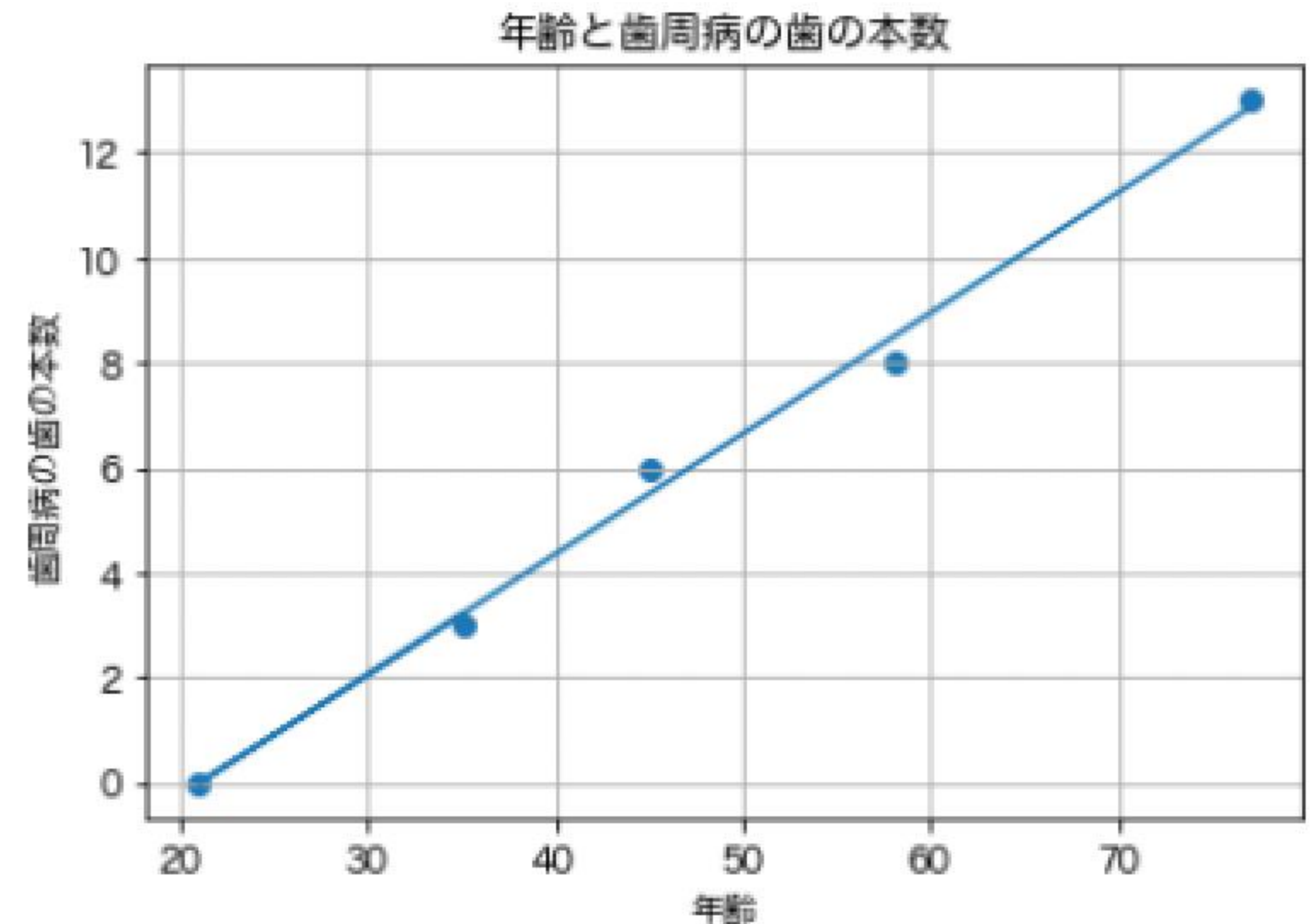
← x=[[35]]の時のmodelが予測した値  
← x=[[21]]の時のmodelが予測した値  
← x=[[45]] ・ ・ ・  
← x=[[58]] ・ ・ ・  
← x=[[77]] ・ ・ ・

# 回帰直線を作図する

plt.scatter(x,y)が実際の(x,y)の点  
plt.plot(x,model.predict(x))がモデルが予測した点をつないだ回帰直線となります。

```
import matplotlib.pyplot as plt

plt.figure()
plt.title('年齢と歯周病の歯の本数',fontfamily='Hiragino Maru Gothic Pro')
plt.xlabel('年齢',fontfamily='Hiragino Maru Gothic Pro')
plt.ylabel('歯周病の歯の本数',fontfamily='Hiragino Maru Gothic Pro')
plt.grid(True)
plt.scatter(x,y)
plt.plot(x,model.predict(x))
# plt.scatter(test,num_teeth)
plt.show()
```



# 回帰直線を作図する

もう一つのアスタリスクも消してみましよう  
plt.scatter(test,num\_teeth)は何の点になるでしょうか？

```
import matplotlib.pyplot as plt

plt.figure()
plt.title('年齢と歯周病の歯の本数',fontfamily='Hiragino Maru Gothic Pro')
plt.xlabel('年齢',fontfamily='Hiragino Maru Gothic Pro')
plt.ylabel('歯周病の歯の本数',fontfamily='Hiragino Maru Gothic Pro')
plt.grid(True)
plt.scatter(x,y)
plt.plot(x,model.predict(x))
plt.scatter(test,num_teeth)
plt.show()
```

# 回帰直線と予測した値を作図する

```
plt.scatter(test,num_teeth)
```

x軸にtest (= [[70]]), y軸にnum\_teeth (=model.predict([[70]]) の点を打つ

```
import matplotlib.pyplot as plt

plt.figure()
plt.title('年齢と歯周病の歯の本数',fontfamily='Hiragino Maru Gothic Pro')
plt.xlabel('年齢',fontfamily='Hiragino Maru Gothic Pro')
plt.ylabel('歯周病の歯の本数',fontfamily='Hiragino Maru Gothic Pro')
plt.grid(True)
plt.scatter(x,y)
plt.plot(x,model.predict(x))
plt.scatter(test,num_teeth)
plt.show()
```

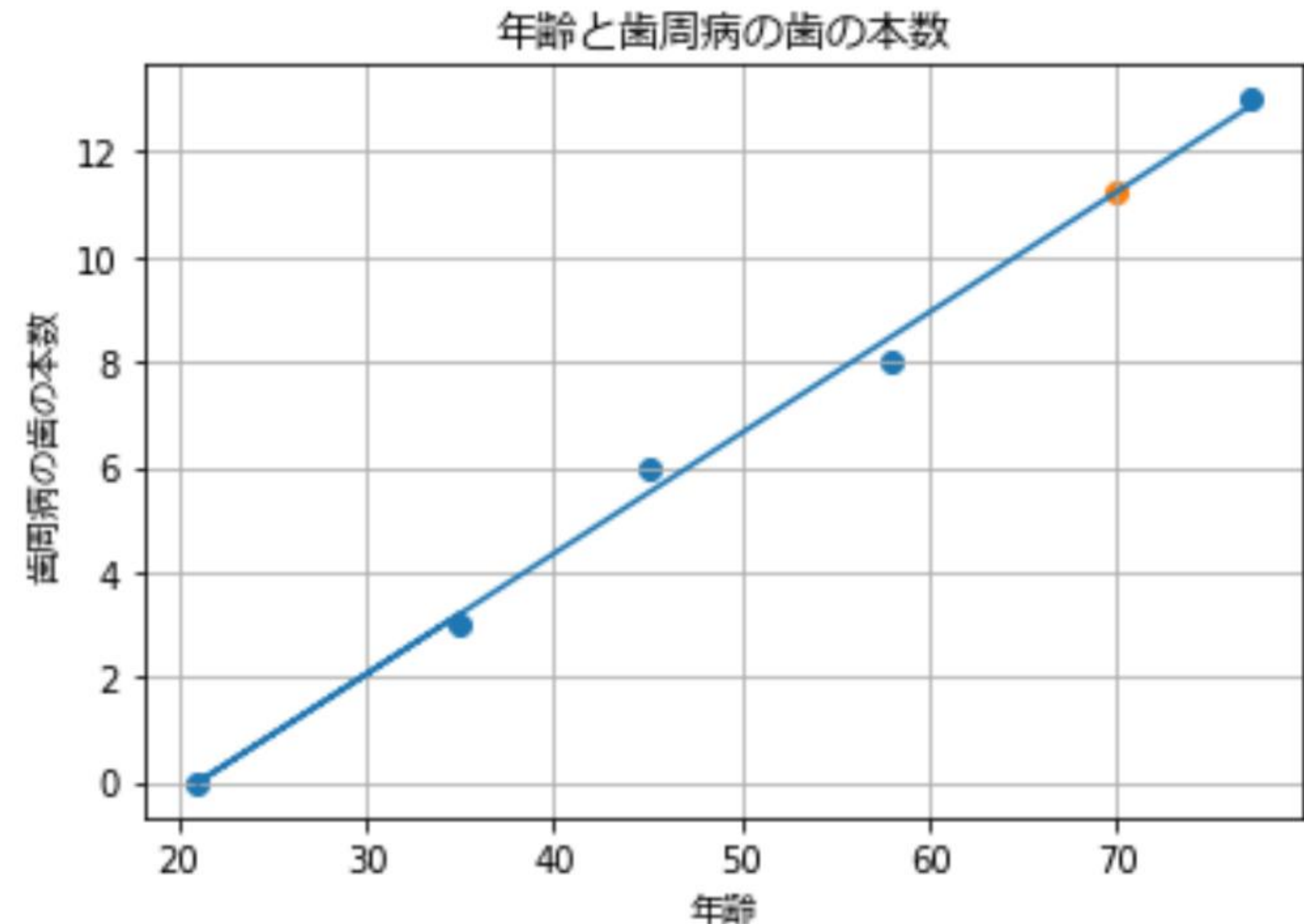
# 回帰直線と予測した値を作図する

```
plt.scatter(test,num_teeth)
```

x軸にtest (= [[70]]), y軸にnum\_teeth (=model.predict([[70]]))の点を打つ

```
import matplotlib.pyplot as plt

plt.figure()
plt.title('年齢と歯周病の歯の本数',fontfamily="Hiragino Maru Gothic Pro")
plt.xlabel('年齢',fontfamily="Hiragino Maru Gothic Pro")
plt.ylabel('歯周病の歯の本数',fontfamily="Hiragino Maru Gothic Pro")
plt.grid(True)
plt.scatter(x,y)
plt.plot(x,model.predict(x))
plt.scatter(test,num_teeth)
plt.show()
```



# まとめ

実際の授業では機械学習の一步目として線形回帰と合わせて  
ロジスティック回帰も体験してもらっております

①学習モデルの選択(今回は線形回帰)  
(モデル名) = `LinearRegression()`

②データを入れて学習させる  
(モデル名).`fit`(説明変数,目的変数)

③傾き(偏回帰係数)と切片(定数項)を求める  
(モデル名).`coef_`           #傾き  
(モデル名).`intercept_`   #切片

④予測を行う  
(モデル名).`predict`(新たな説明変数)

①学習モデルの選択(今回はロジスティック回帰)  
(モデル名) = `LogisticRegression()`

②データを入れて学習させる  
(モデル名).`fit`(説明変数,目的変数)

③傾き(偏回帰係数)と切片(定数項)を求める  
(モデル名).`coef_`           #傾き  
(モデル名).`intercept_`   #切片

④予測を行う  
(モデル名).`predict`(新たな説明変数)  
(モデル名).`predict_proba`(新たな説明変数)

お時間ある方はwebclassをご確認頂ければ幸いです