Google Colaboratoryを使ってみよう

統合教育機構
須藤毅顕

これからのpythonのプログラミング演習はgoogle colaboratoryを使います

google colaboratoryとは

特徴	Googleが提供するPythonの実行環境 無料で利用出来る 導入が容易 GPUも使用できる
注意点	Googleのアカウントが必要 一度の使用時間(ランタイム)が連続最長12時間まで (=書いたプログラム(コード)は保存されますが、実行内容は 消えるため実行し直す必要がある)

CPUとGPU

CPU

Central Processing Unit。コンピュータにおける中心的な演算装置。

GPU

Graphics Processing Unit。画像処理に特化した演算装置。並列演算処理に優れ、行列演算が得意なため、画像処理以外でも機械学習の領域でも利用される。

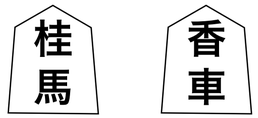
Google ColaboratoryはプログラムCPUとGPUとTPU(Tensor Processing Unit)を選んで実行することが出来ます(デフォルトはCPU)。TPUはGoogle社が開発した機械学習に特化した演算装置で特定の条件においてはGPUよりも高速になります。

CPUとGPUのイメージ

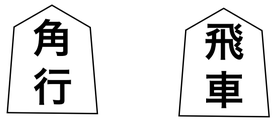
CPUが優れていると色んな高度な処理が可能

CPU 低

CPU 中



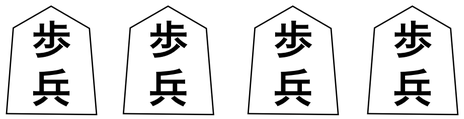
CPU 高



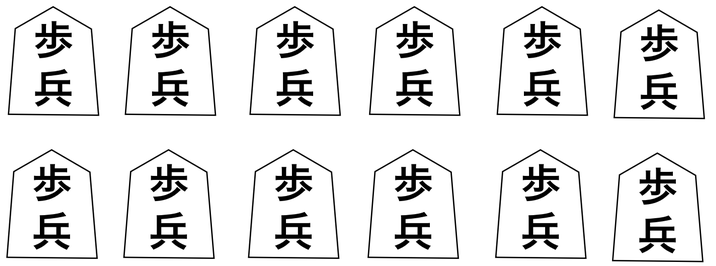
GPUが優れていると並列した単純作業を高速に行うことが出来る

GPU 低

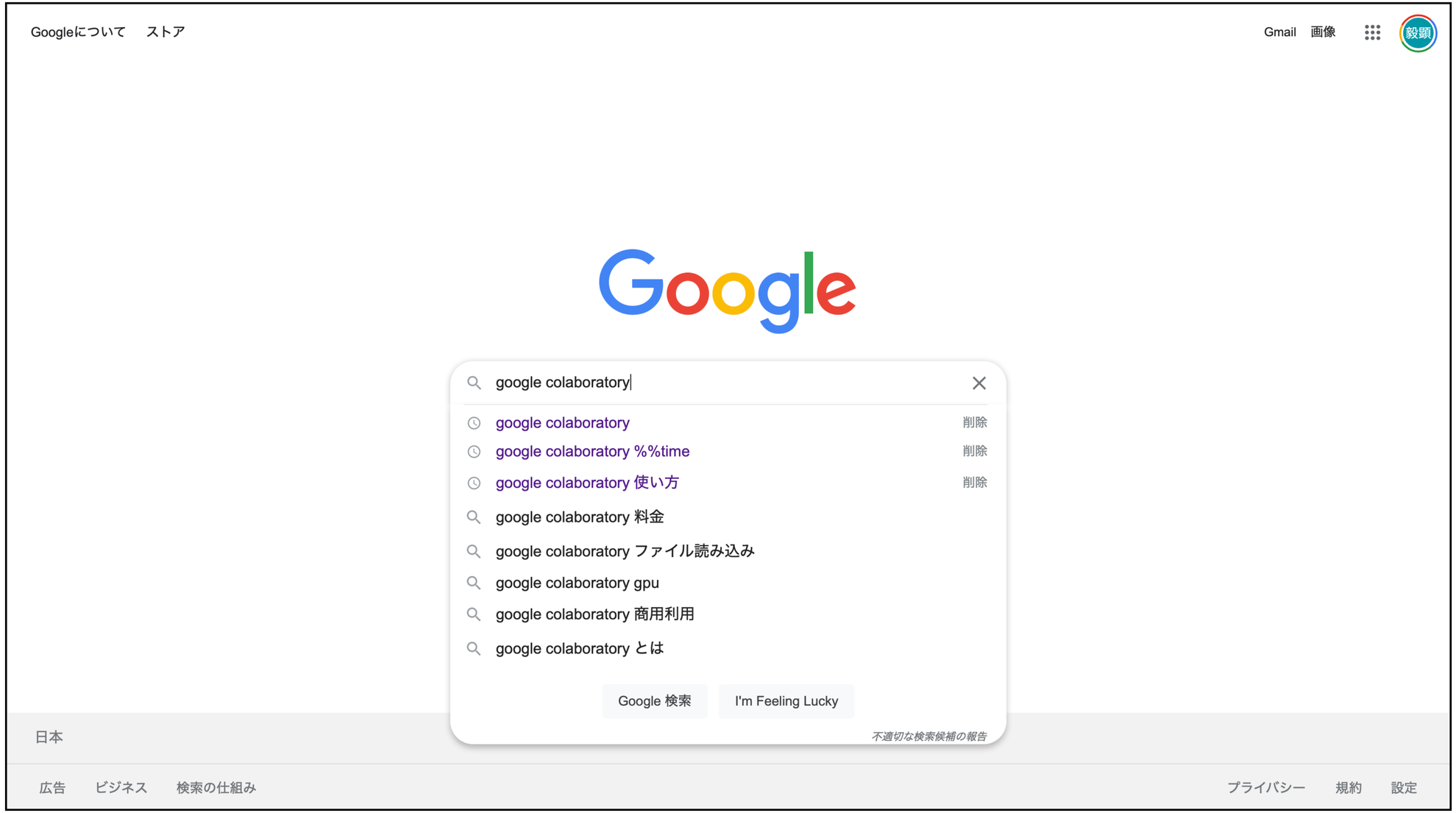
GPU 中



GPU 高



google colaboratoryで検索する



一番上をクリック

もしくは https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja

Colaboratoryのメイン画面

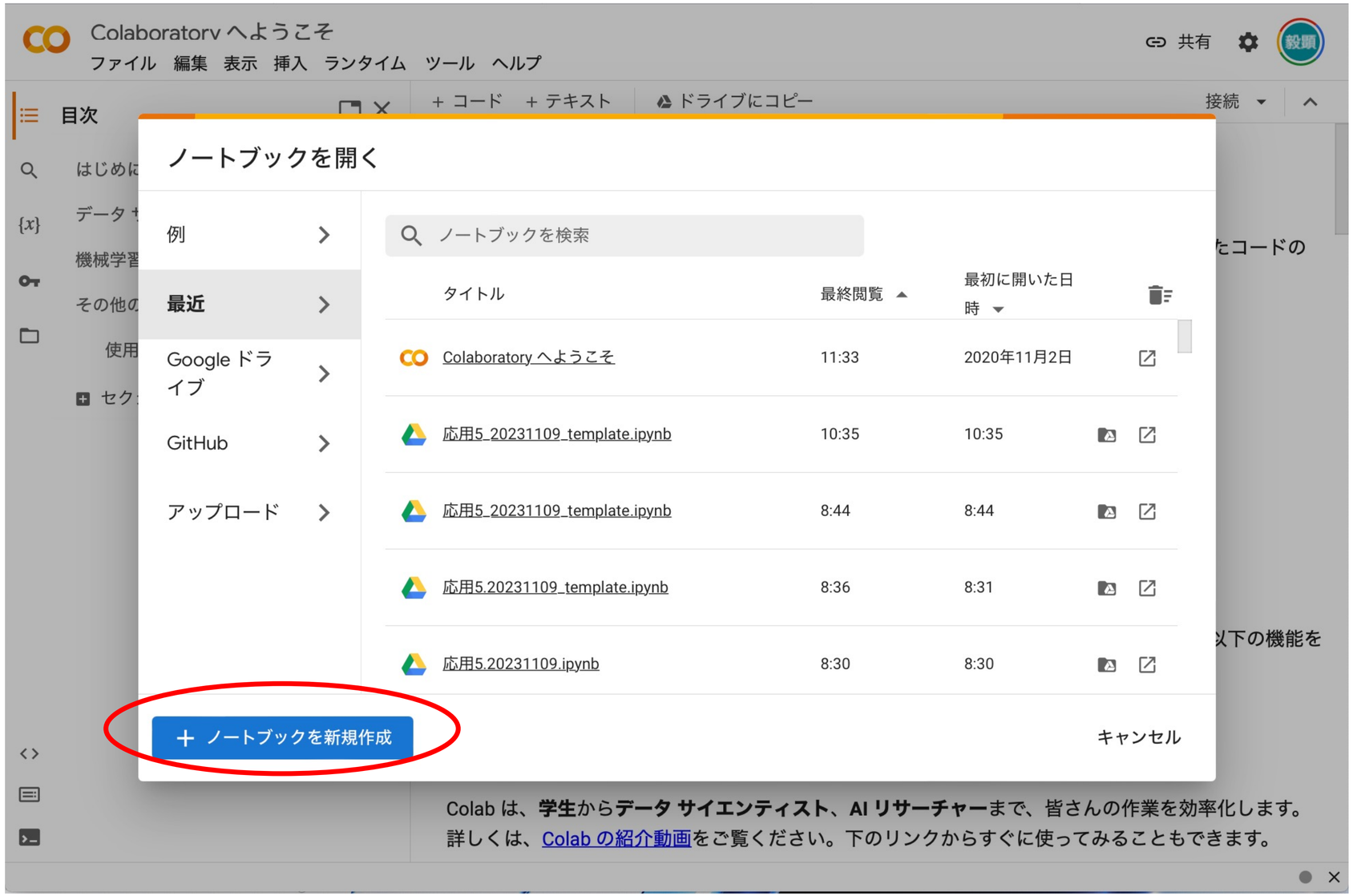
この画面はColaboratoryの説明画面になります。
時間のある時に読んでみてください。

2023/02/20の初期画面

Googleのログインが聞かれた場合

2回目以降

“ノートブックを新規作成”をクリックするとすぐに開始画面に移動します(スライド12)

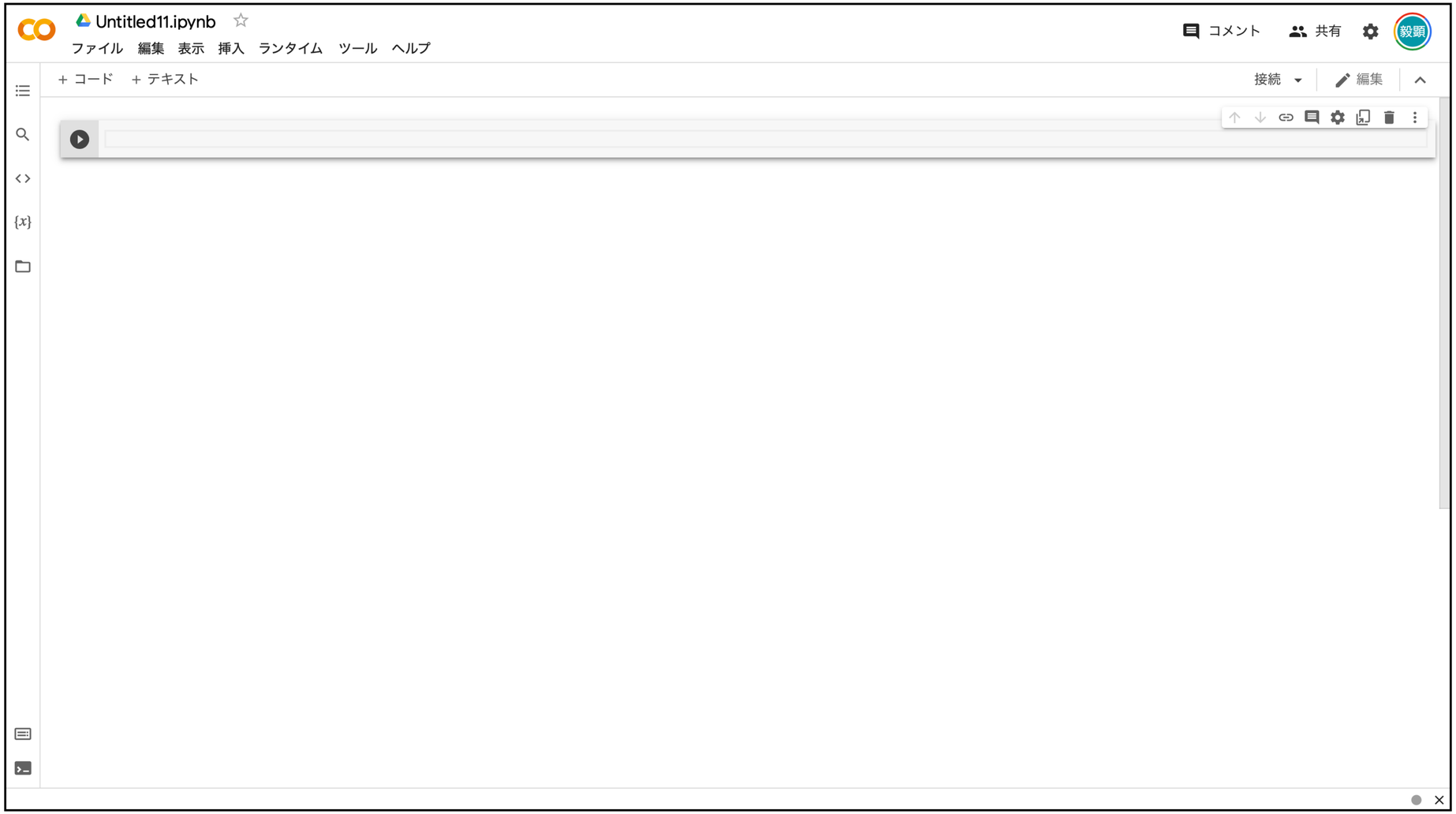


キャンセルを押すとメイン画面に戻る

2023/02/20の初期画面

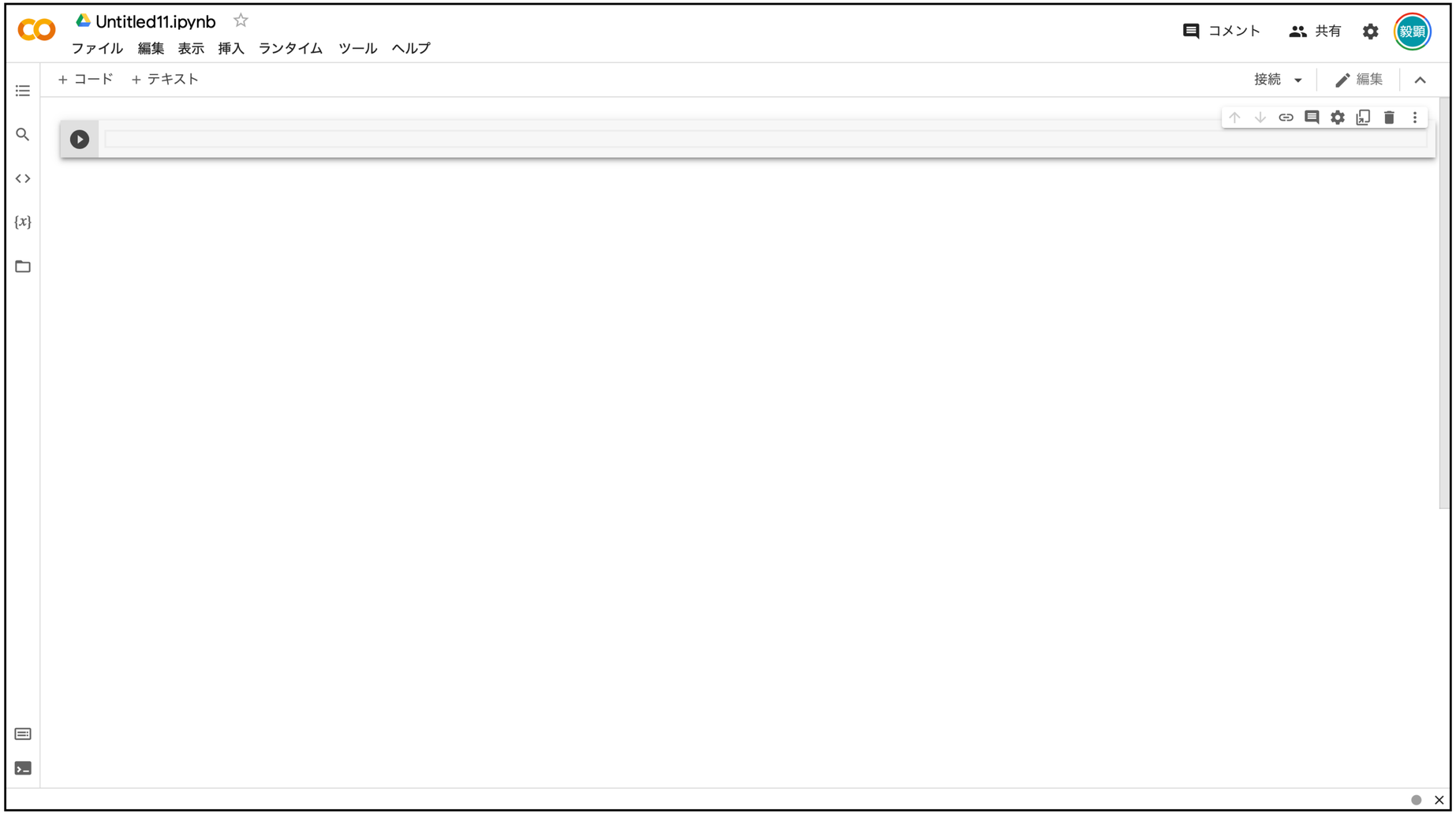
「ファイル」→「ノートブックを新規作成」をクリック

新規のノートブックが表示される



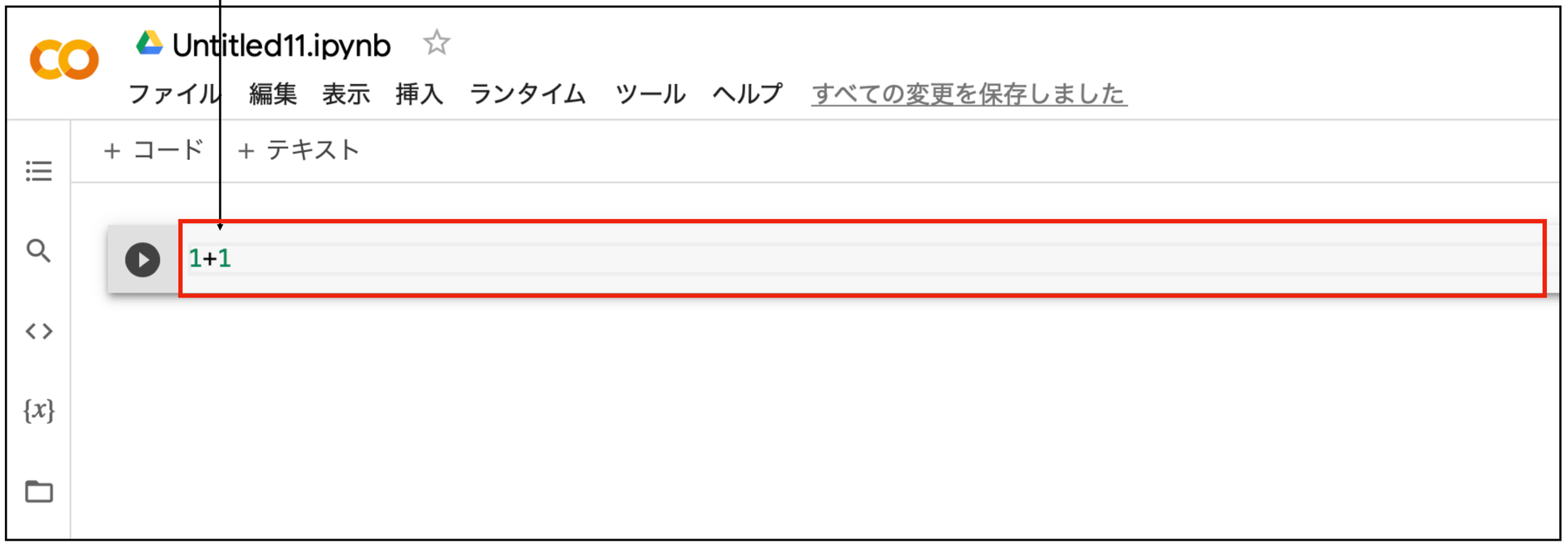
ノートブックはColaboratoryでpythonを記述するファイルです。

新規のノートブックが表示される



ノートブックは拡張子が.pyではなく、.ipynbになります。
(最初はUntitled○○.ipynbとなってます。)

ここにカーソルを合わせて1+1と入力してみましょう



実行するには横の ▶ か、shift + enterを押します。

2が実行結果として表示されました

(一番最初は実行に少し時間が掛かります。)

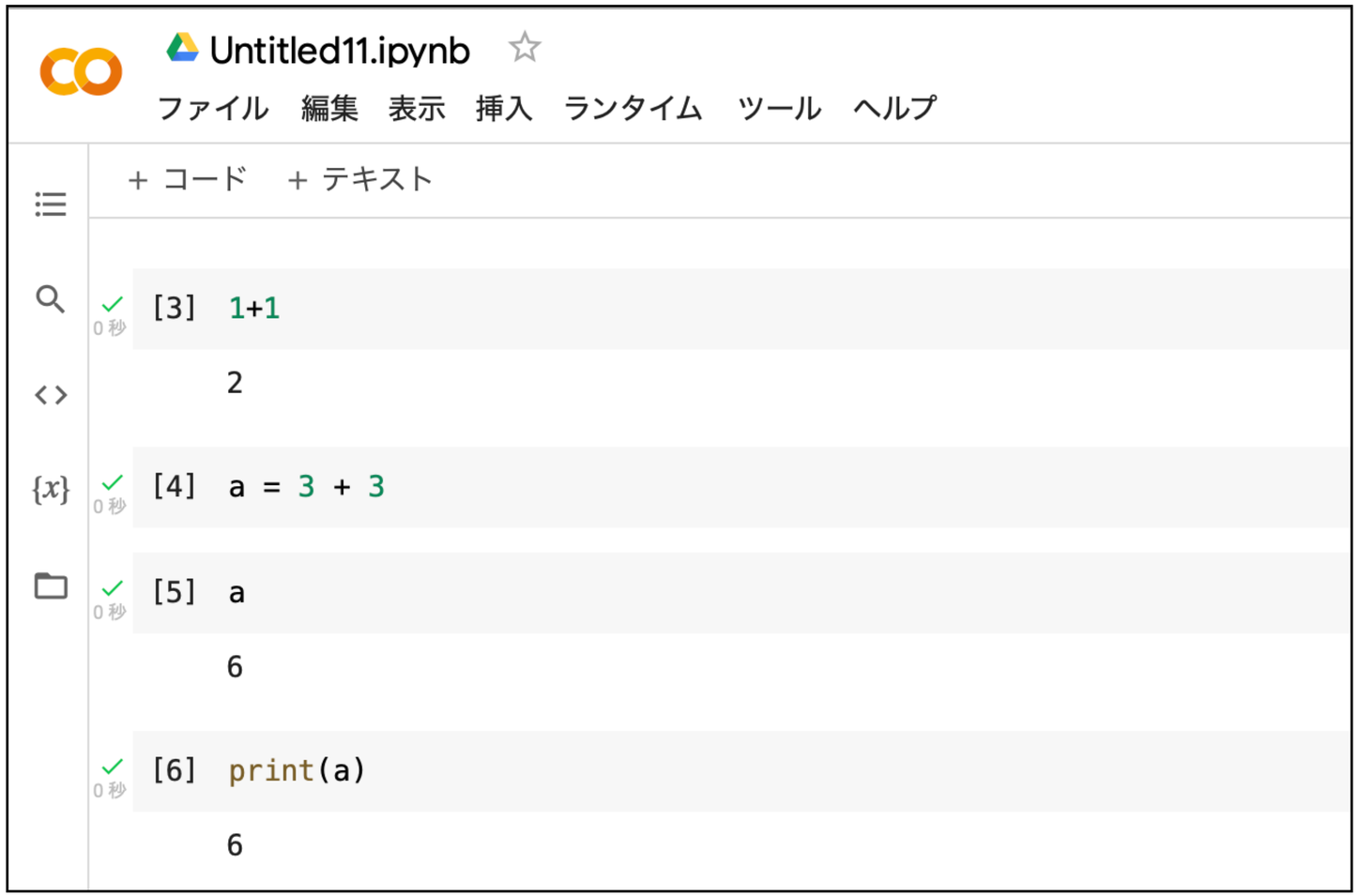
a = 3 + 3 と入力して実行してみましょう

実行結果は何も表示されません

aと入力して実行してみましょう

aには3+3が代入されているので6が表示されます

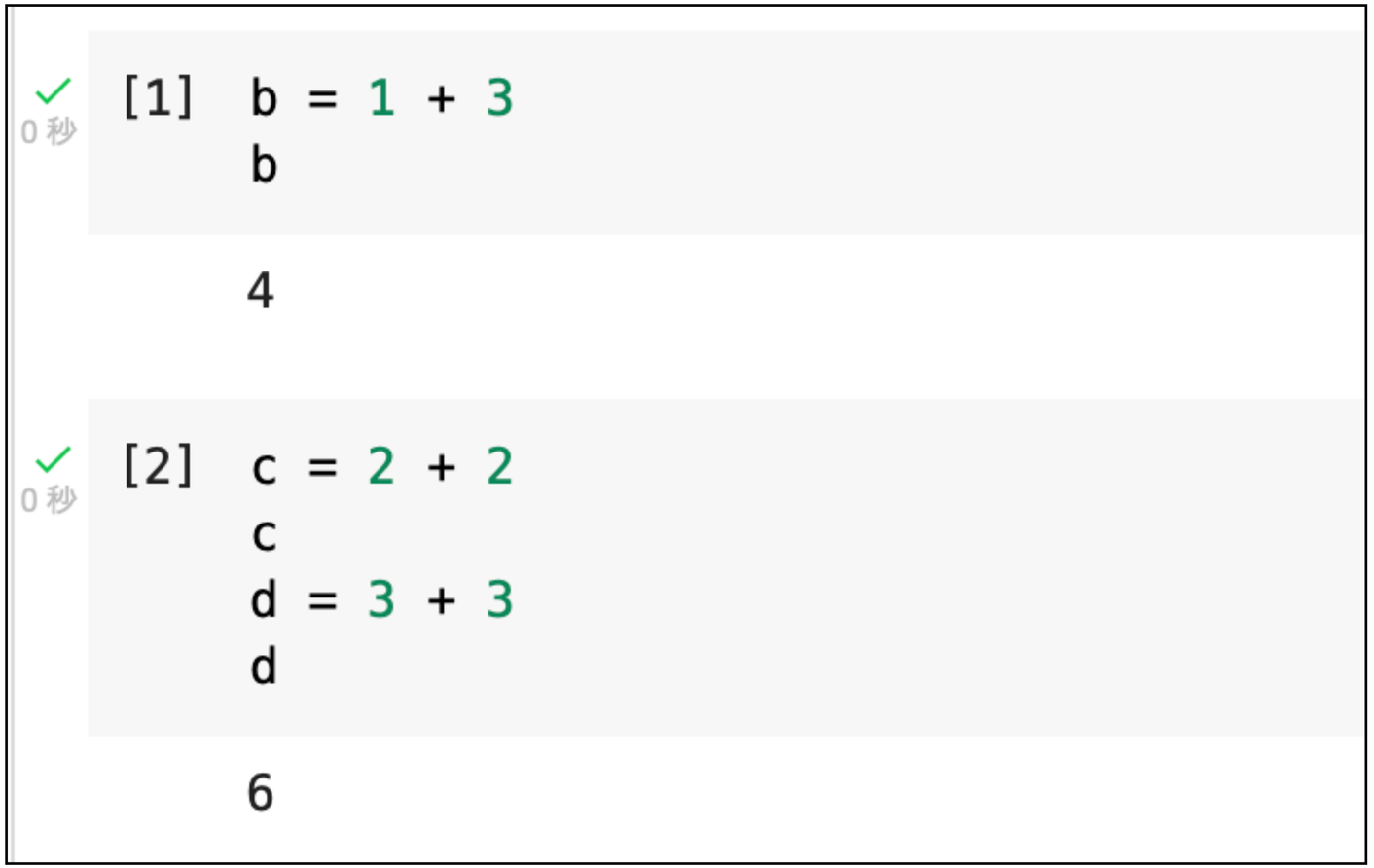
print(a)でも同じ結果が表示されます



Colaboratoryではセルの中の最後の行にある変数はそのままでも出力(print)されます

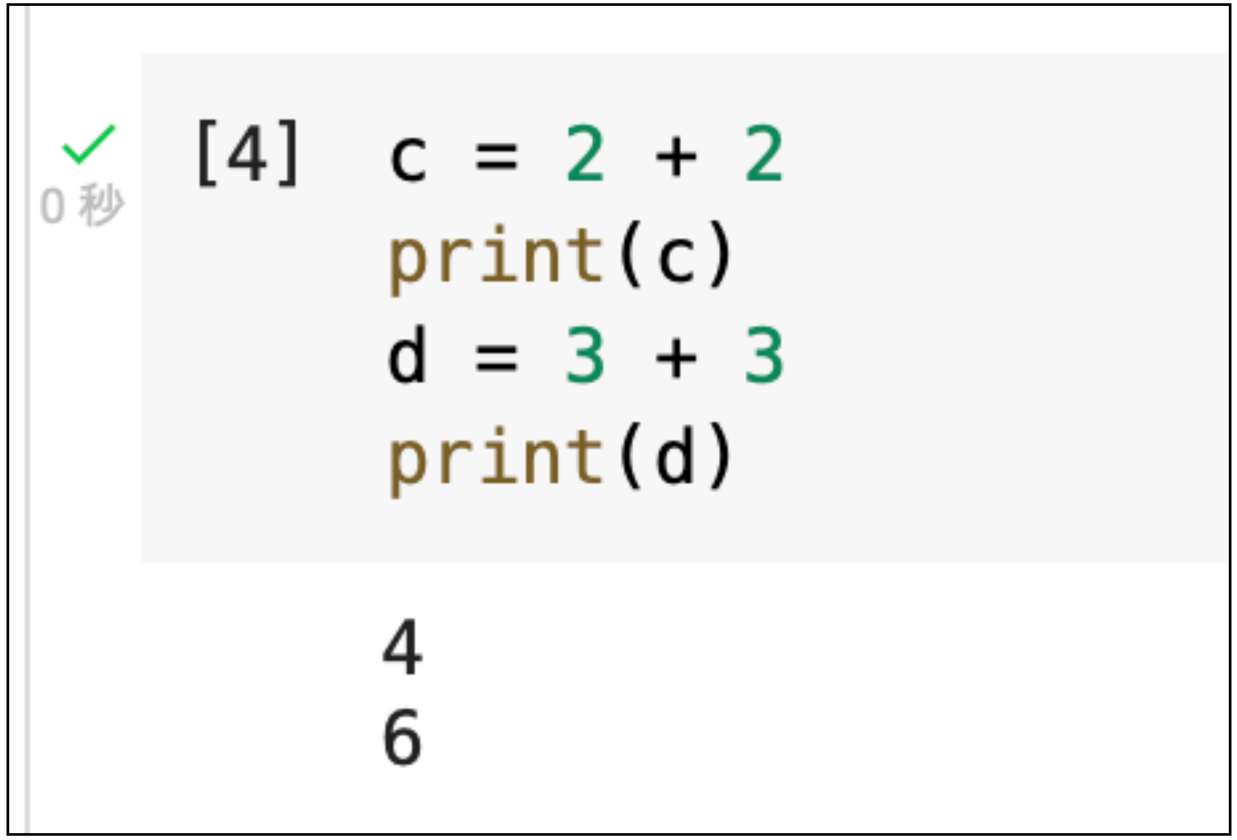
(これはconsoleやanacondaのJupyter notebookでも同様です。)

1つのセルの中には何行でも書けます。

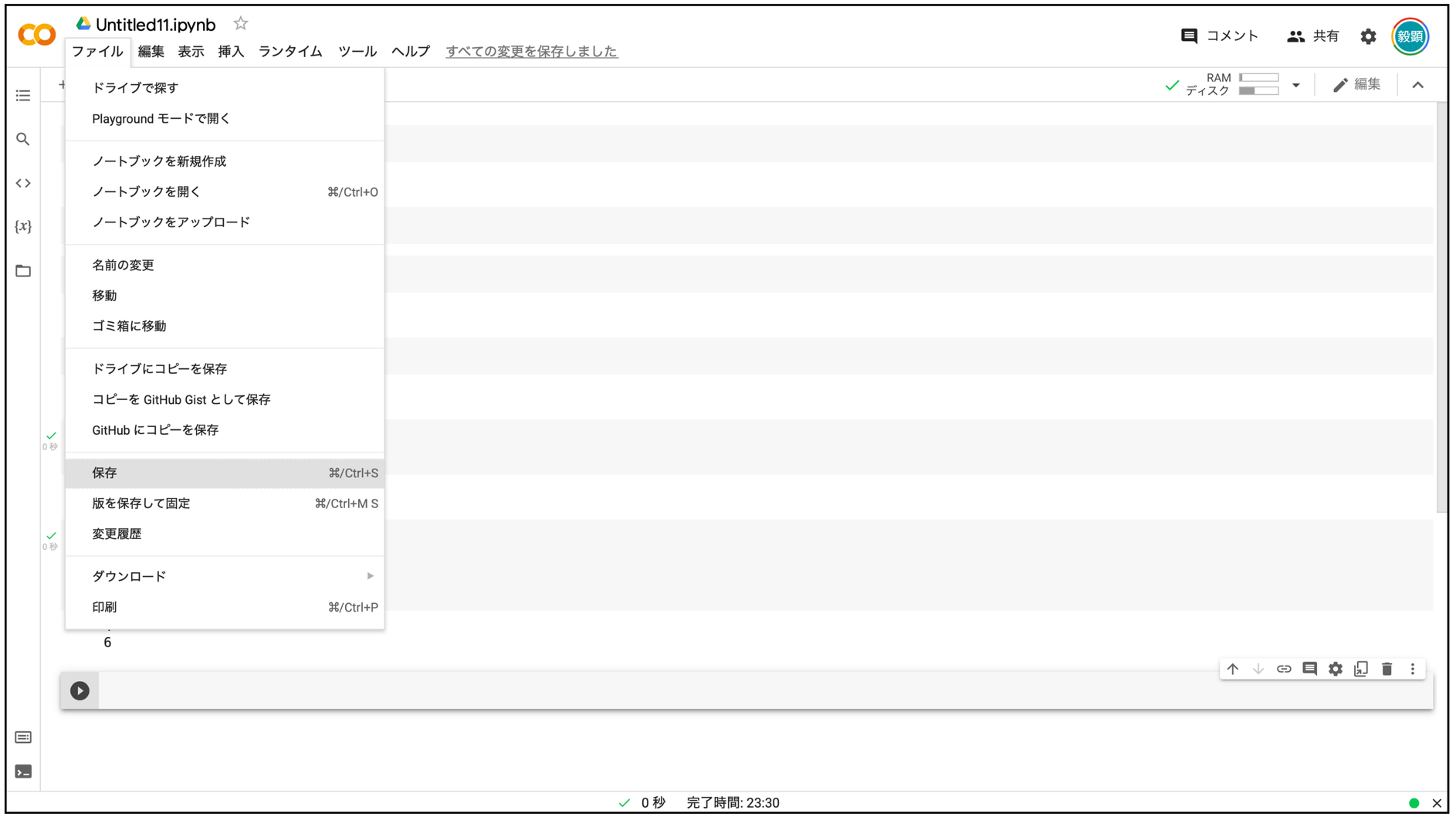


変数名のみで実行結果されるのはセルの最後の行の変数だけです

print関数を使えばどちらも変数の中身が表示されます

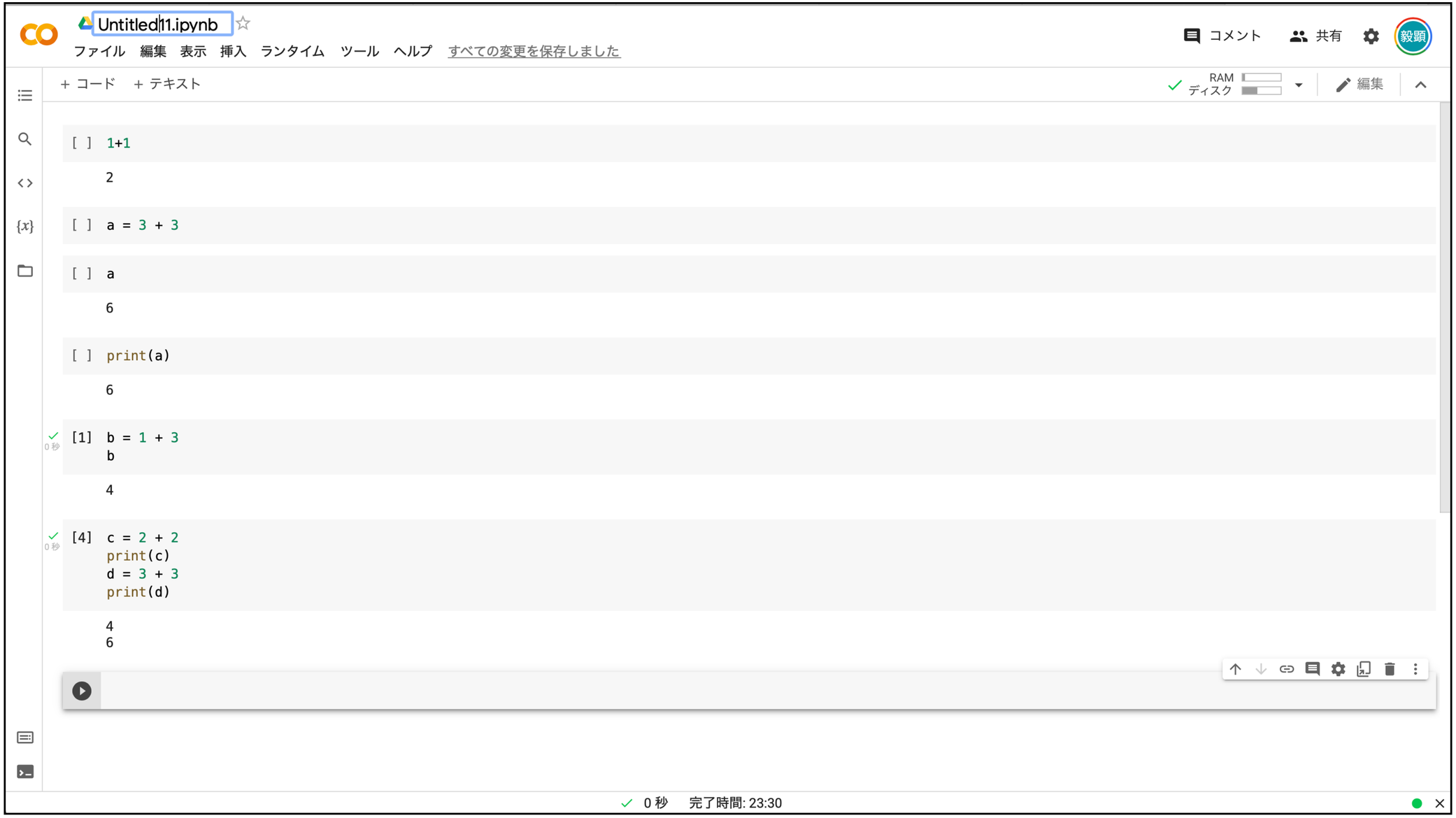


「ファイル」→「保存」でノートブックを保存することが出来ます

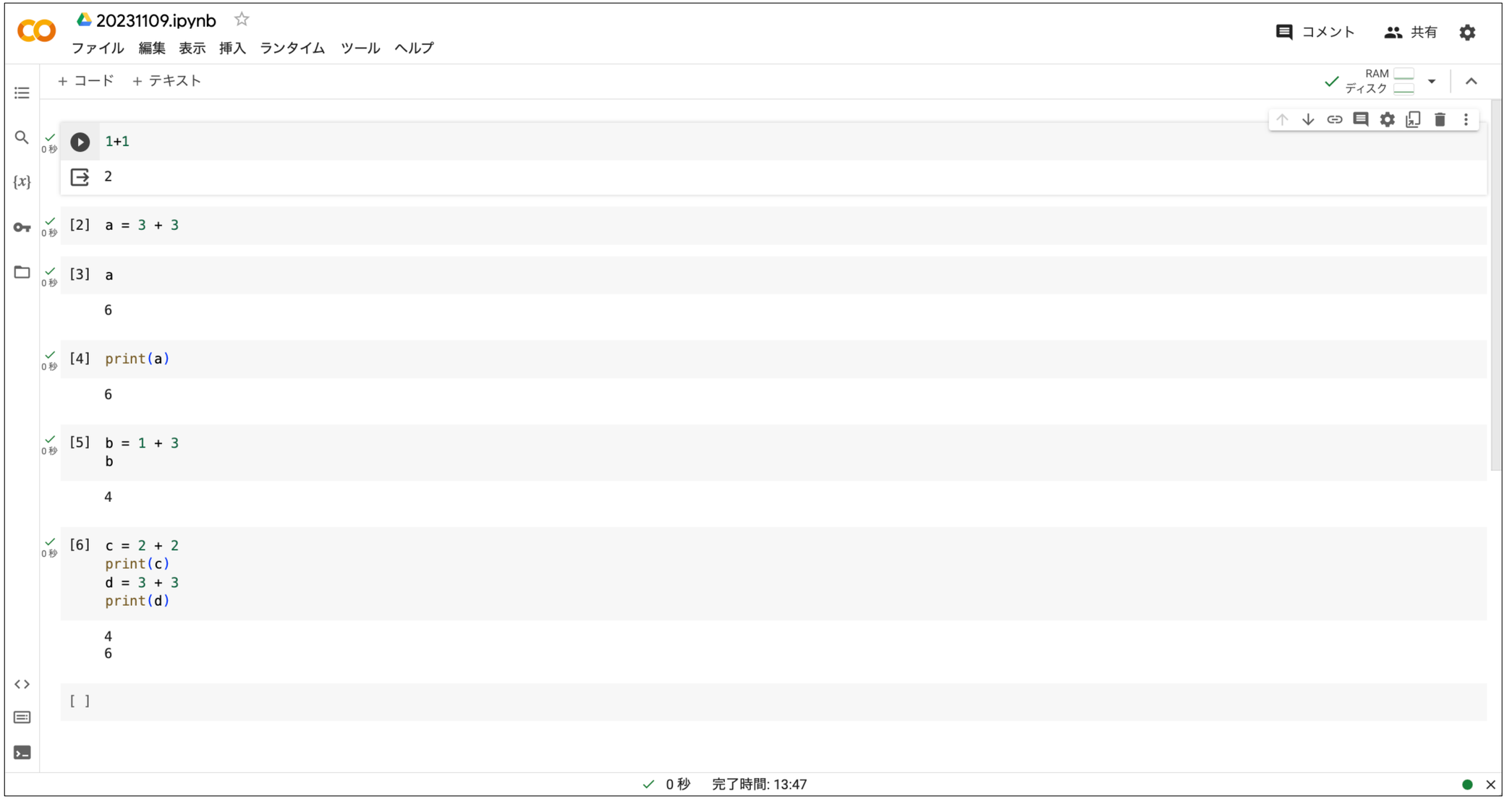


CTRL + s でも保存出来ます (Macの場合はcommand + s)

ファイル名をクリックすると青枠になりファイル名を変更できます



20231109.ipynbに変更出来ました



型はtypeで調べます(復習)

```
hello = "こんにちは"
print(type(hello))
num1 = 3
print(type(num1))
num2 = 4.5
print(type(num2))
```

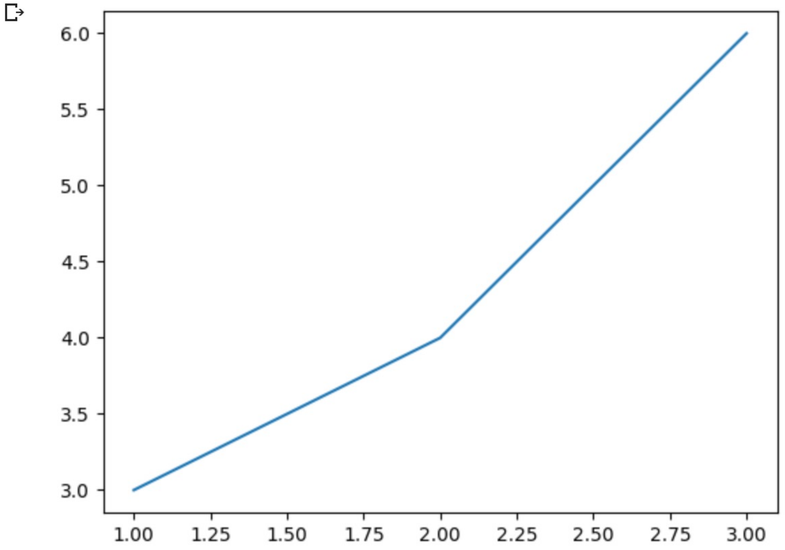
```
<class 'str'>
<class 'int'>
<class 'float'>
```

spyderの変数エクスプローラーがないため、
変数の型はtypeで調べます(似た機能はあります)

ライブラリを使う(matplotlib)

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

```
x = [1,2,3]
y = [3,4,6]
plt.plot(x,y)
plt.show()
```




```
x = [1,2,3]
y = [3,4,6]
plt.plot(x,y)
plt.show()
```

matplotlibはspyderでは1行ずつでなくまとめて実行していました。
colaboratoryでは1つのセル単位で実行します

ライブラリを使う

ライブラリは標準ライブラリと外部ライブラリがあります

標準ライブラリ：最初からpythonに備わっているライブラリ
外部ライブラリ：外からインストールするライブラリ

- matplotlibは外部ライブラリなので本来はインストールする必要があります。
- WinPythonでは授業で扱うライブラリがインストール済みのものをダウンロードしてました。
- Anacondaでは仮想環境にライブラリをインストールしてました

colaboratoryはあらかじめいくつもの外部ライブラリがインストールされた状態で始めることが出来ます

ライブラリを使う

!pip listと入力すると使用出来るライブラリが一覧として表示されます。

```
!pip list
```

```
Package                       Version
----------------------------- ---------------------
absl-py                       1.4.0
alabaster                     0.7.13
albumentations                1.2.1
altair                        4.2.2
anyio                         3.6.2
appdirs                       1.4.4
argon2-cffi                   21.3.0
argon2-cffi-bindings          21.2.0
array-record                  0.2.0
arviz                         0.15.1
astropy                       5.2.2
astunparse                    1.6.3
                              0.4.3
LunarCalendar                 0.0.9
lxml                          4.9.2
Markdown                      3.4.3
markdown-it-py                2.2.0
MarkupSafe                    2.1.2
matplotlib                    3.7.1
matplotlib-inline             0.1.6
matplotlib-venn               0.11.9
mdurl                         0.1.2
miniKanren                    1.0.3
missingno                     0.5.2
mistune                       0.8.4
mizani                        0.8.1
mkl                           2019.0
```

ファイルを読み込む

Google Colaboratoryではファイルを読み込むことができます

手順

1. Google Driveをマウントする
2. Google Driveにファイルをアップロードする
3. Google Colaboratoryで読み込む

メニュー画面からマウントする

画面左のフォルダのタブをクリックします

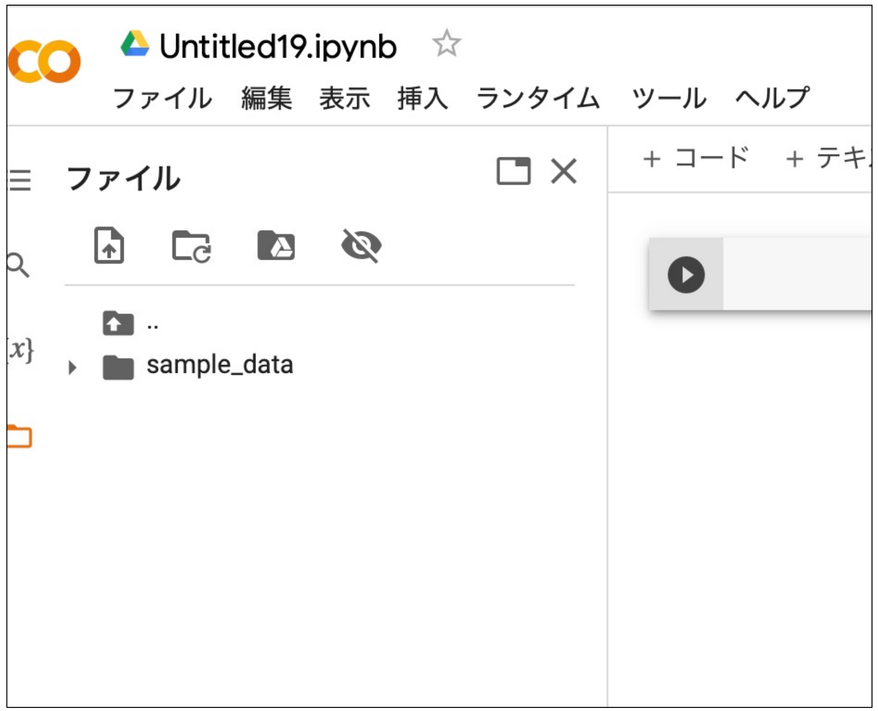
メニュー画面からマウントする

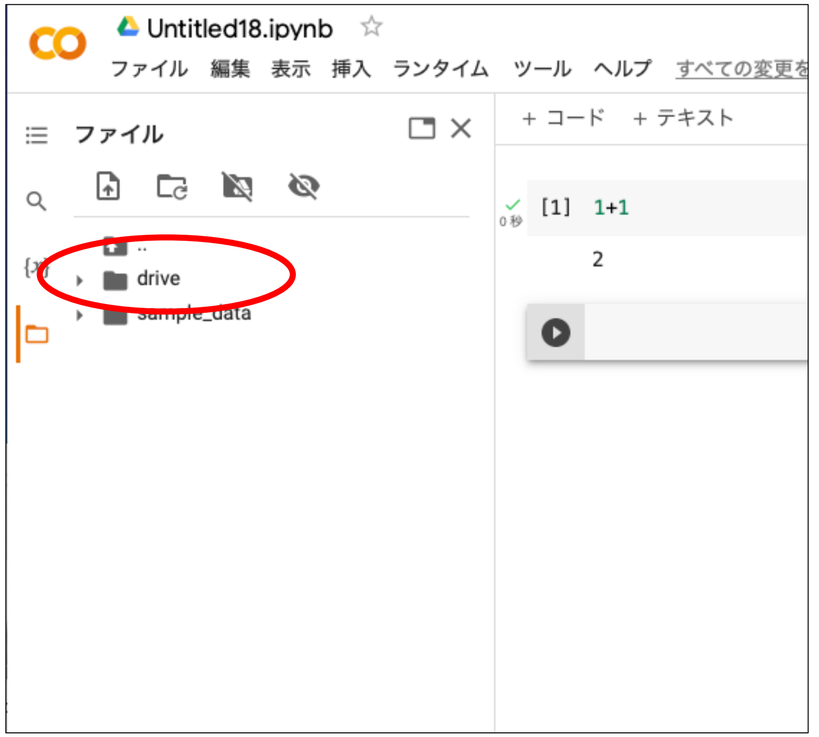
Driveをマウントするアイコン をクリックします

メニュー画面からマウントする

“Googleドライブに接続”をクリックします

メニュー画面からマウントする

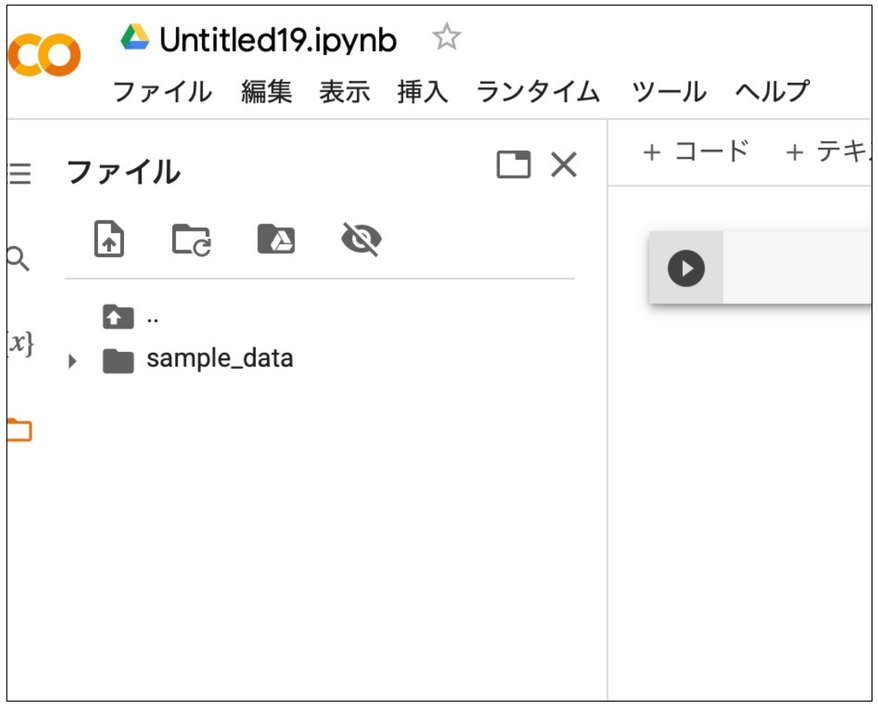


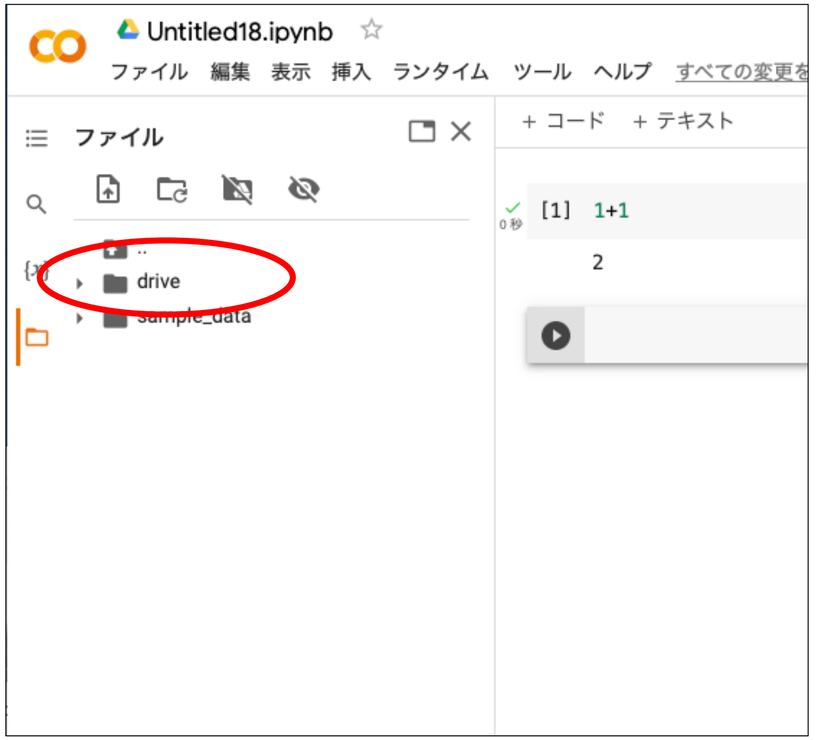


Google Driveがマウントされるとdriveというアイコンが作成されます

(補足) 最初に別の確認画面が表示された場合

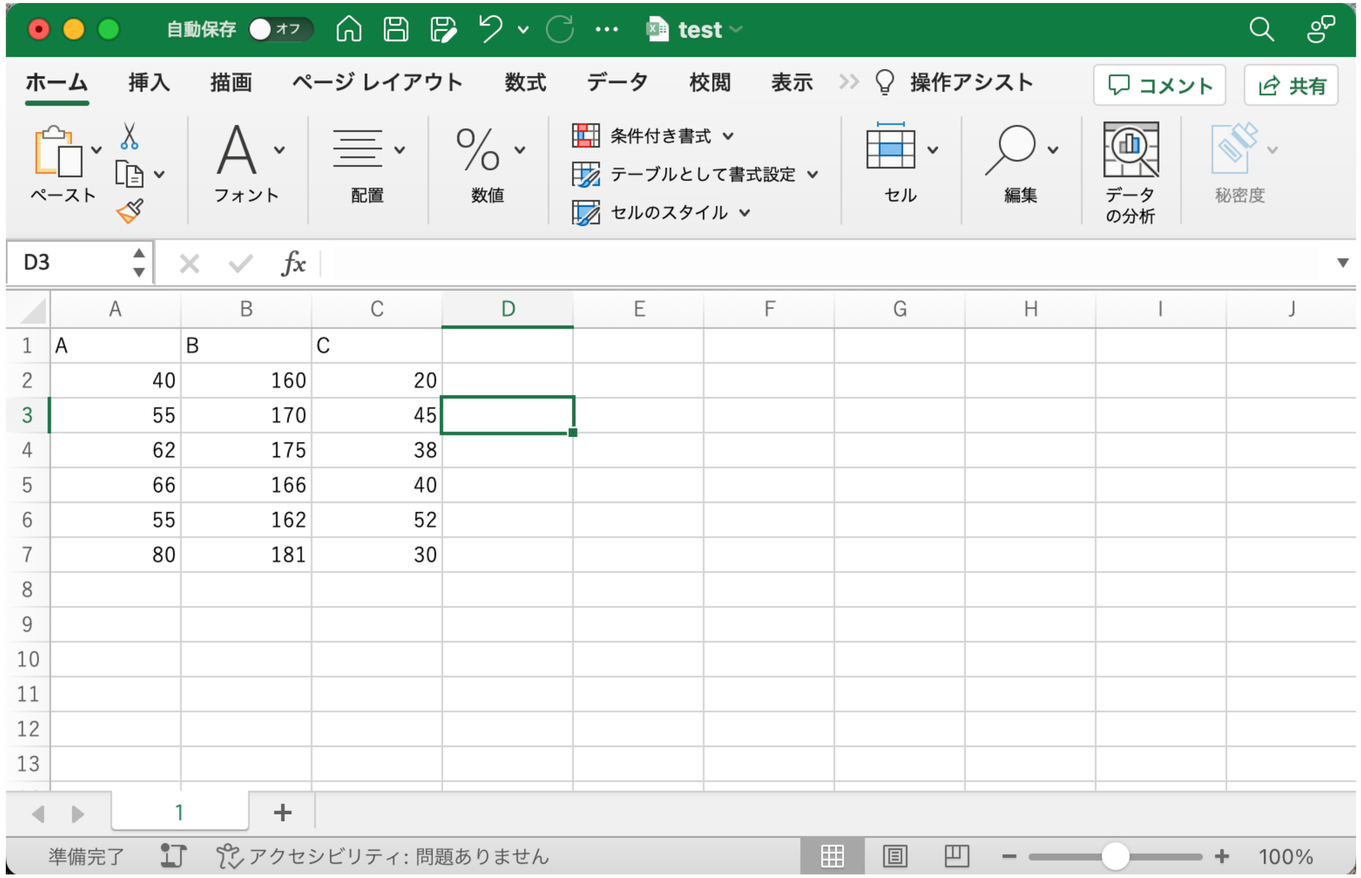
メニュー画面からマウントする





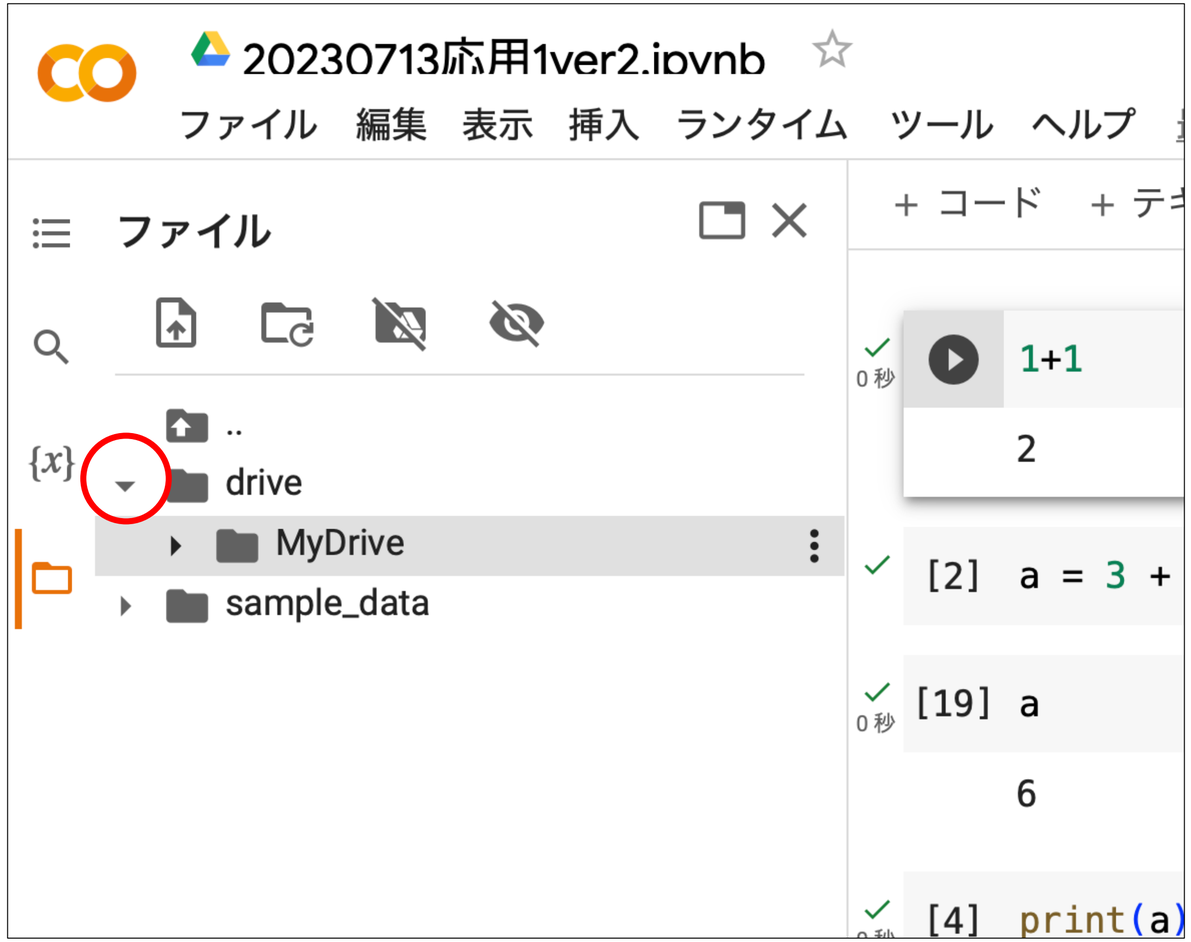
Google Driveがマウントされるとdriveというアイコンが作成されます

Google Driveにファイルをアップロードする



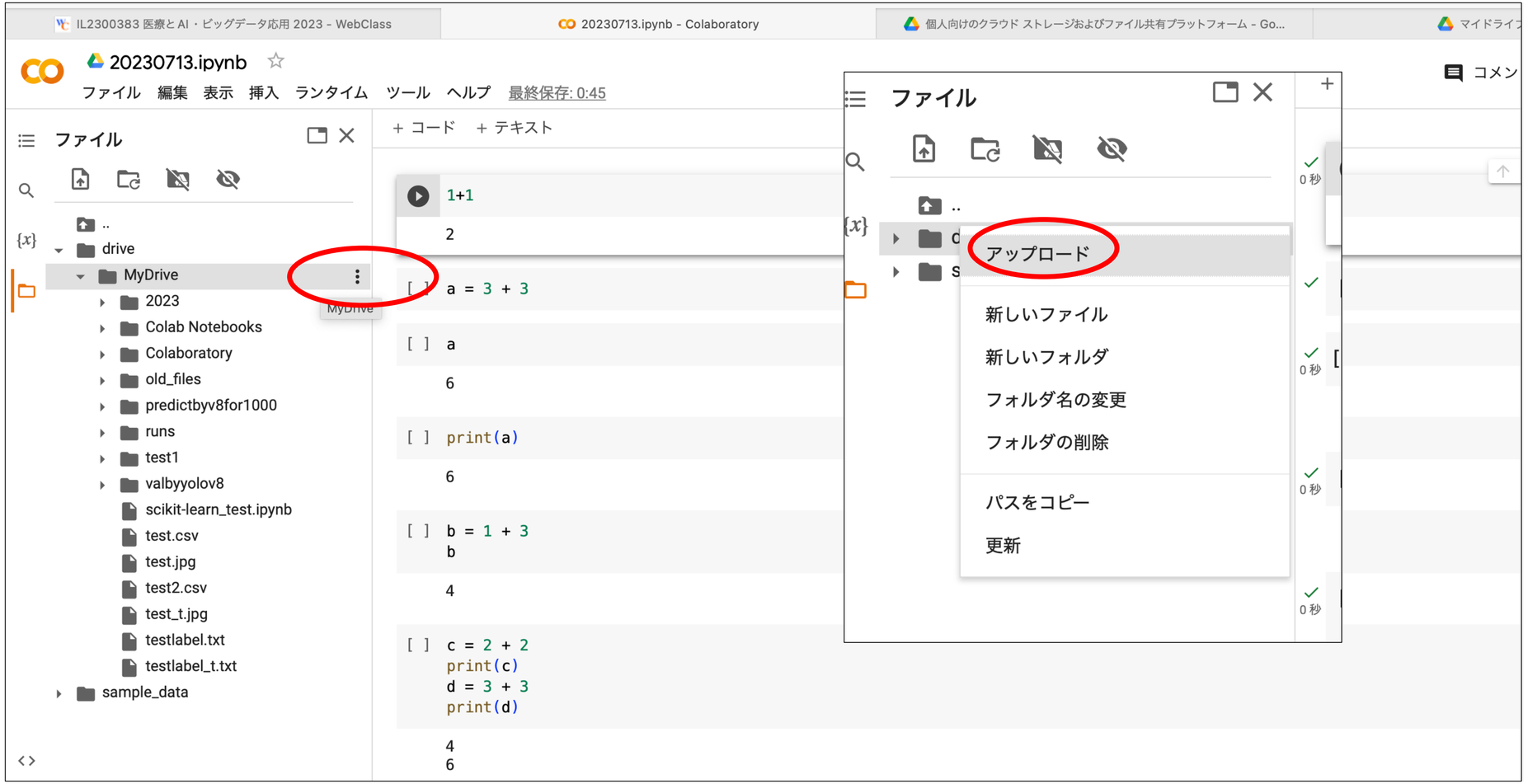
試しにWebclassにあるtest.csvをアップロードしてみます
(ダウンロードして分かりやすい場所に保存して下さい)

Google Driveにファイルをアップロードする



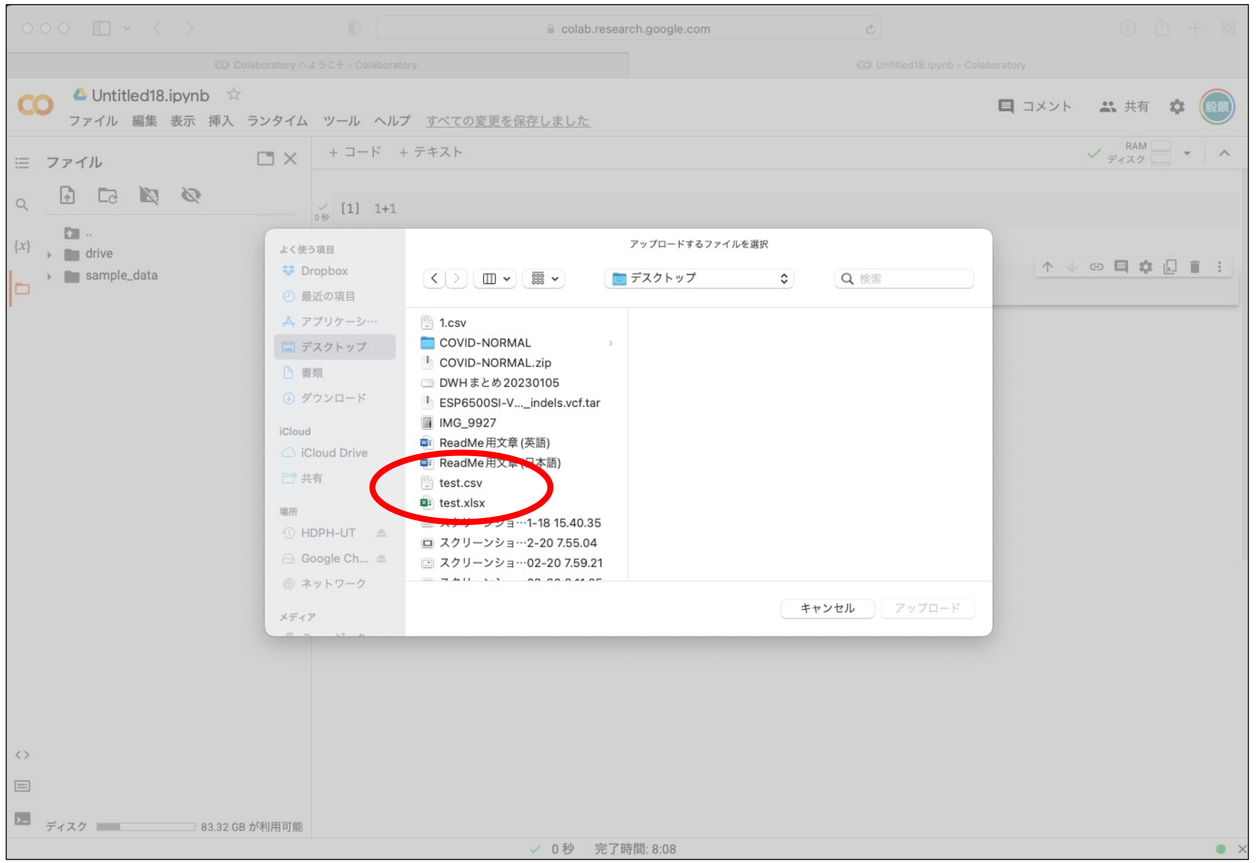
driveのアイコンの▸を押すと ▾ になり、その中身が表示されます。
MyDriveが出てきます。

Google Driveにファイルをアップロードする



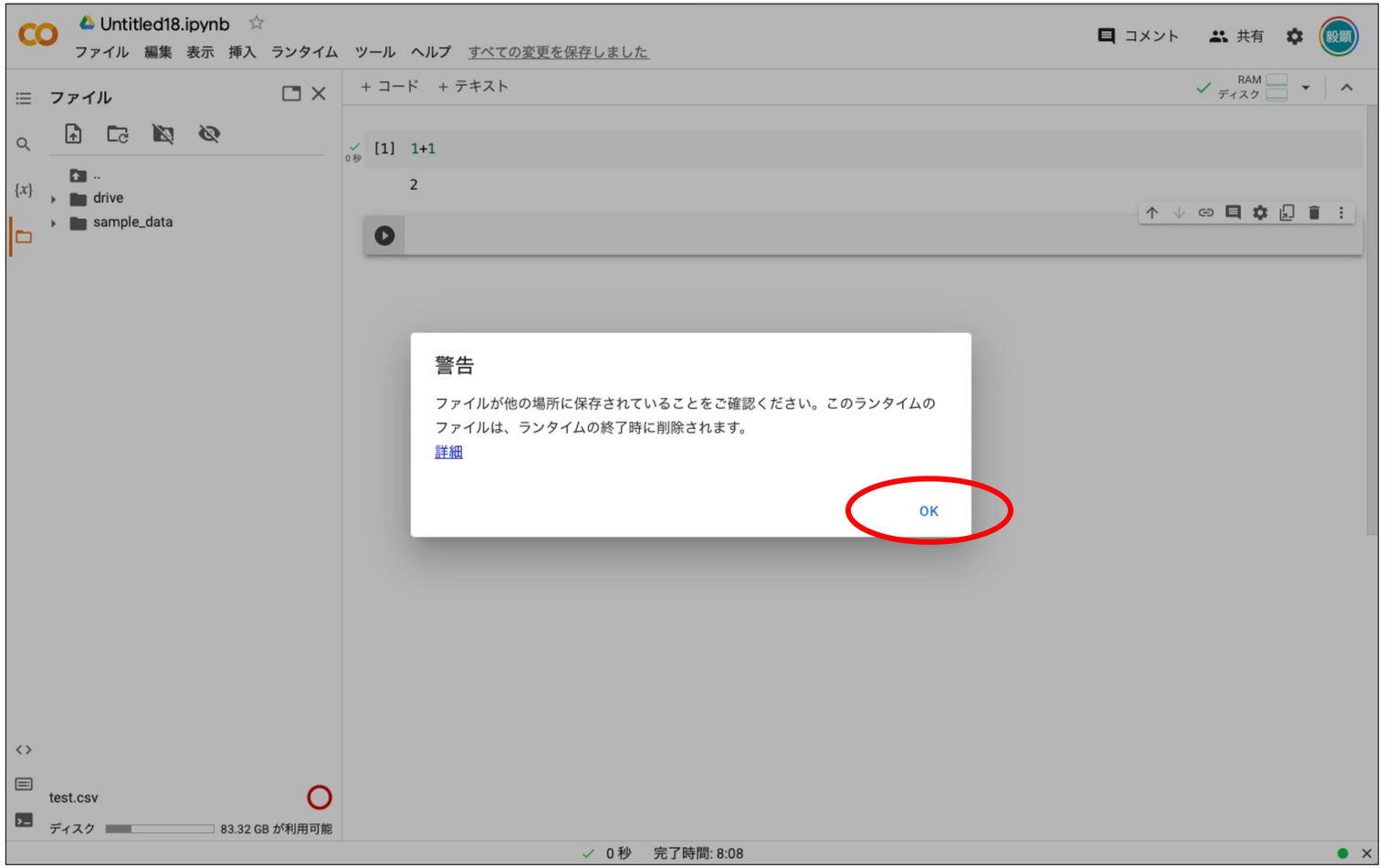
MyDriveのフォルダにカーソルを合わせると右に ⋮ が表示されるのでクリックします
表示される「アップロード」をクリックします

Google Driveにファイルをアップロードする



フォルダ選択画面が表示されるので、
今回はデスクトップの”test.csv”を選びます

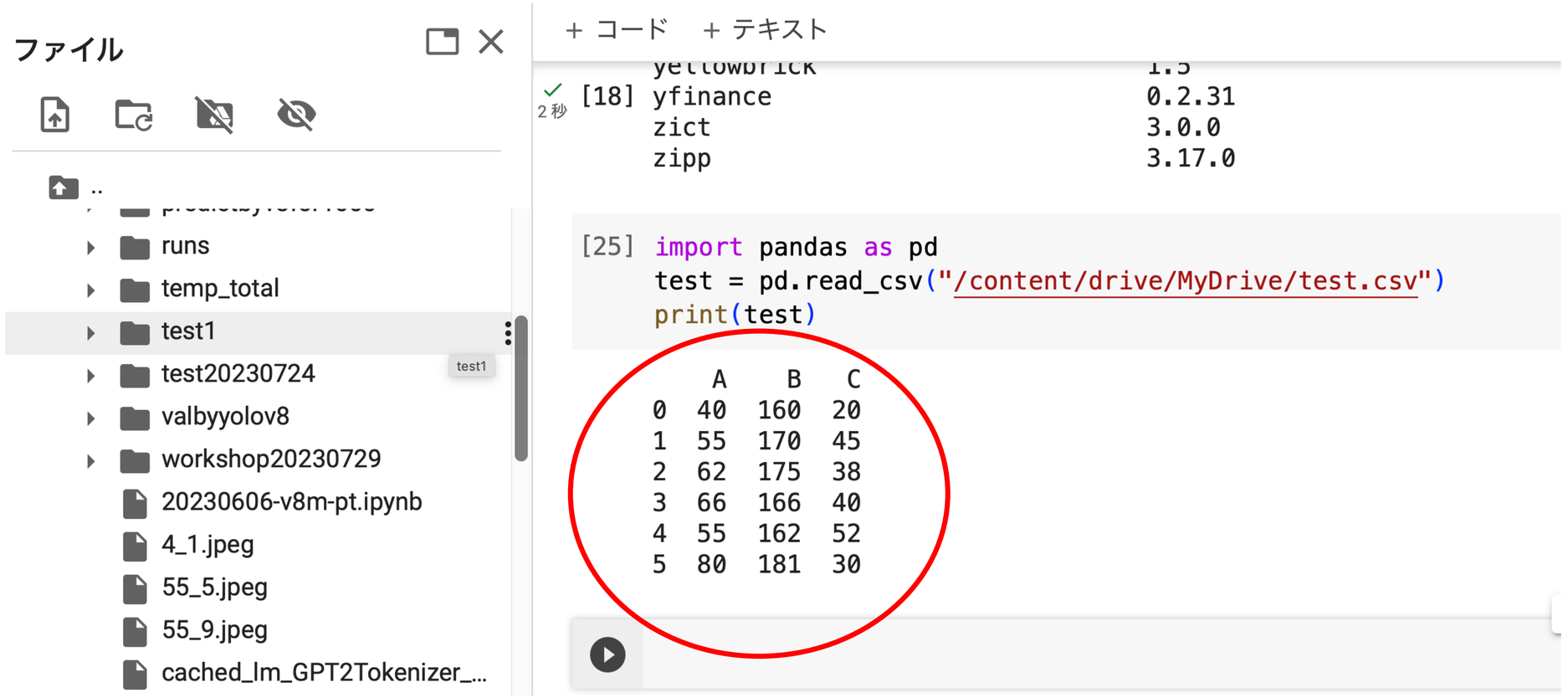
Google Driveにファイルをアップロードする



Google Driveにファイルをアップロードする

Mydriveの中にtest.csvがアップロード出来ました
(これはGoogle Driveの中にアップロード出来ていることになります)

ファイルを読み込む




```
import pandas as pd
test = pd.read_csv("/content/drive/MyDrive/test.csv")
print(test)
```

ファイルの場所(パスといいます)

色んな読み込み方がありますが、ここでは左の書き方で
(pandasというライブラリを用いて)
ファイルを読み込む方法を使っています

保存して再度読み込む

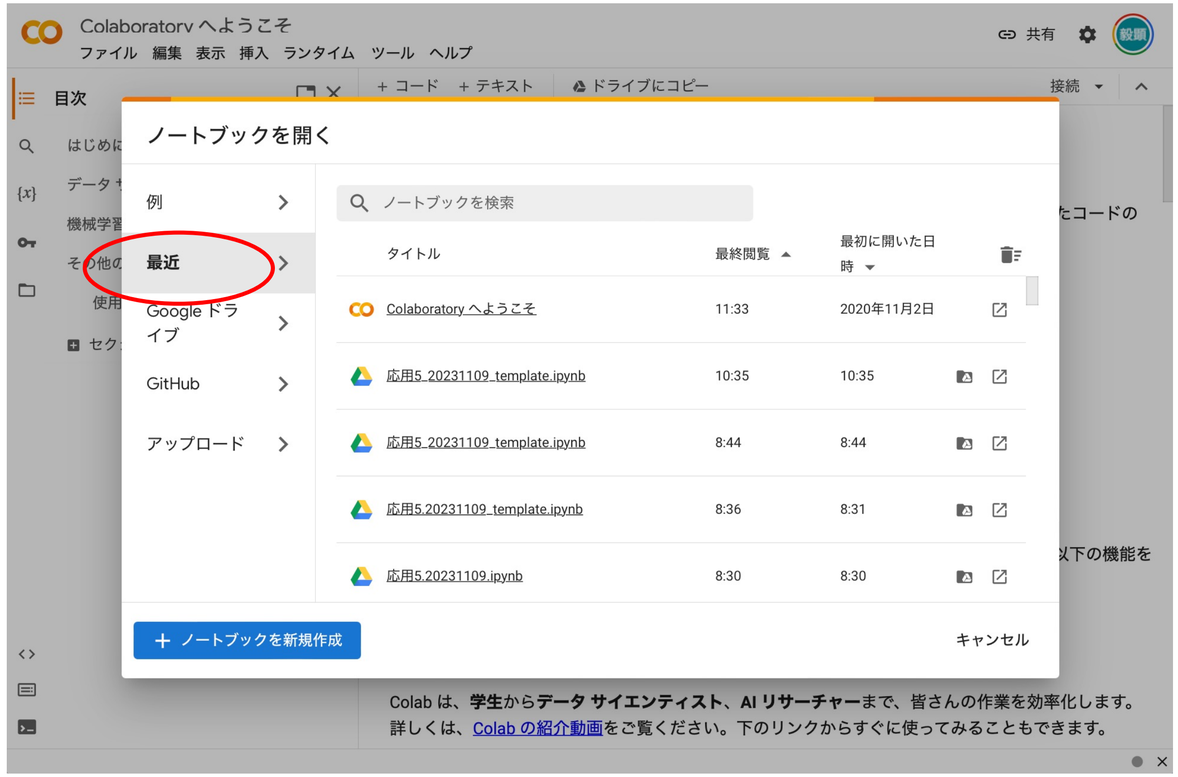
保存して再度同じファイルを読み込んでみます
(ファイル名を再度20231109.ipynbにしています)

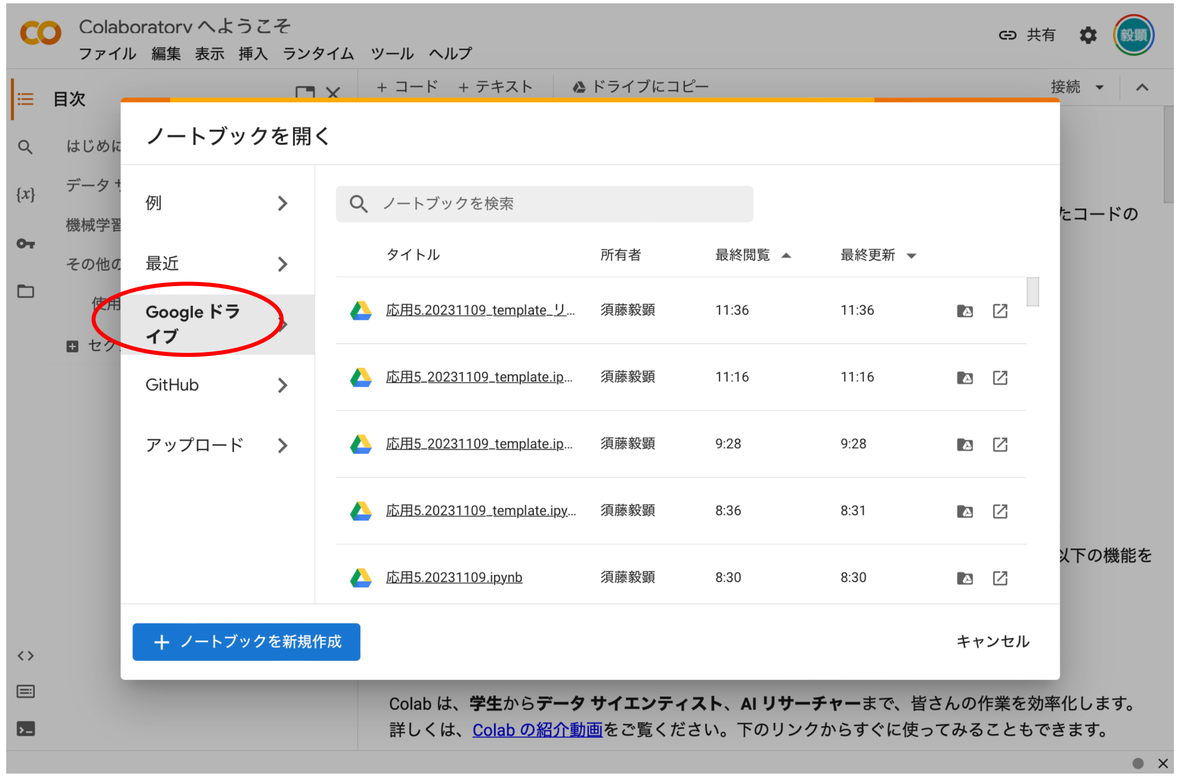
保存して再度読み込む

一度閉じて再度Google Colabを開いてみましょう

保存して再度読み込む

「最近」の中か、Googleドライブの中にファイルを見つけることが出来ます。





保存して再度読み込む

前回の作成したコードと実行結果が表示されたファイルを開くことが出来ます。

長時間空いているとリセットされて変数の情報が消えます。
その場合はいきなりprint(a)などを実行してもエラーになります。
再度実行しなおす必要があります。

作ったファイルをダウンロードする

「ファイル」→「ダウンロード」→「ipynb」でファイルをダウンロードできます。

課題

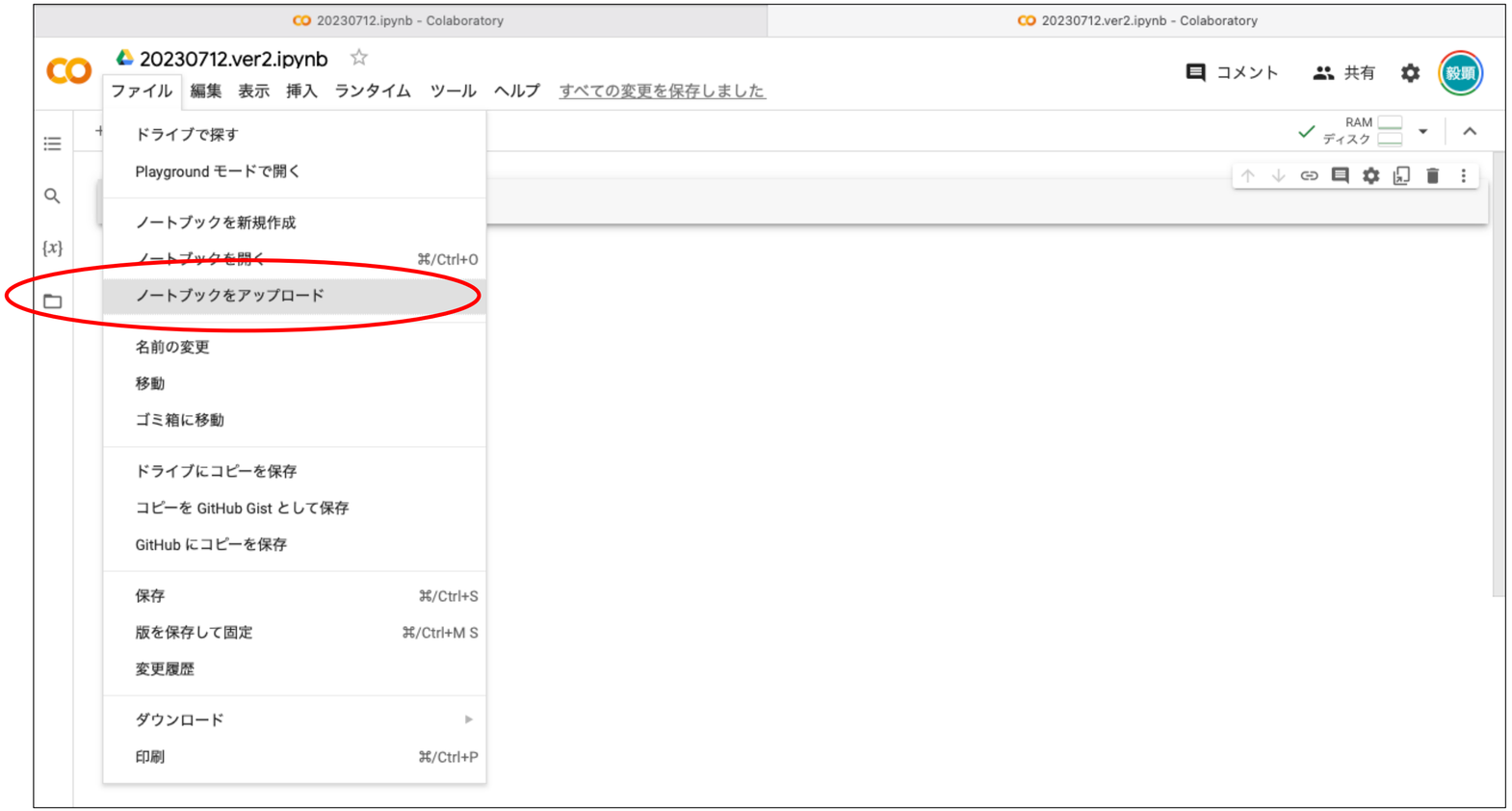
- WebClassにある”演習１課題(入門).ipynb”をやってみましょう
- 実行したら”学籍番号_名前_1.ipynb”という名前で保存
  して提出して下さい。(参考資料もwebclassにあります)

締め切りは2週間後の11/23の23:59です。
締め切りを過ぎた課題は受け取らないので注意して下さい

ipynbのファイルの開き方は次ページ参照

ipynbのファイルを開く

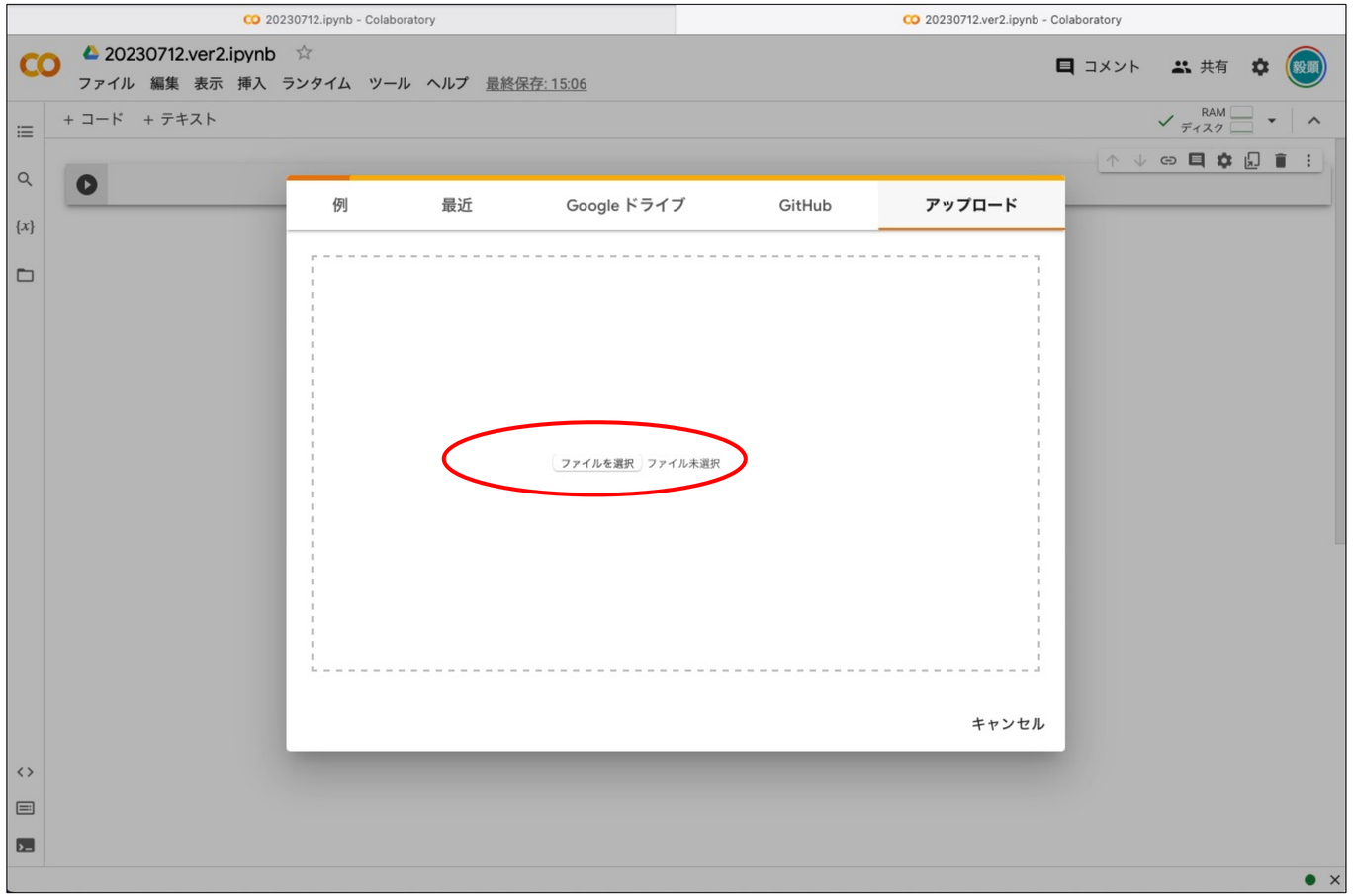
ipynb(ノートブック)を開くには、いくつか方法がありますが、ここでは
「ノートブックのアップロード」からファイルを開いてみます



過去に(自分のGoogleアカウントで)開いたノートブックやGoogle Driveに
アップロード済みのノートブックであれば、「ノートブックを開く」から選びます

ipynbのファイルを開く

「ファイルを選択」から選ぶか、四角の点線枠内にファイルをドラッグ&ドロップします



今回は、webclass上の応用1_課題の”kadai1.pynb”をダウンロードしてアップロードします

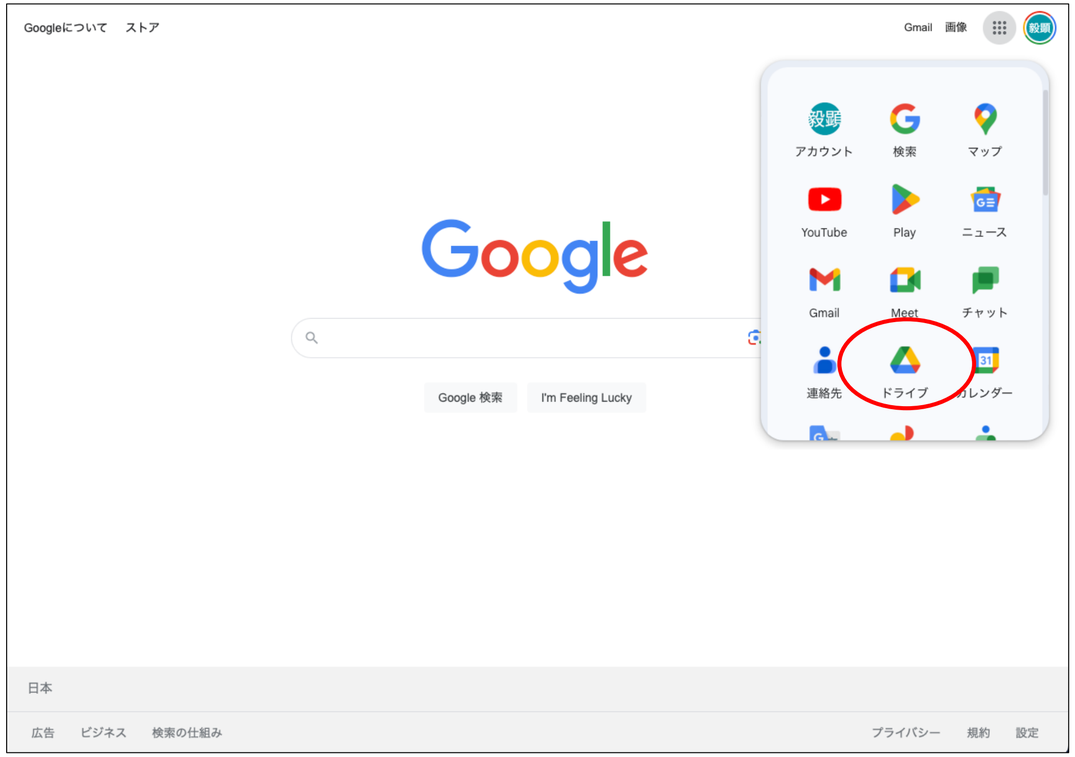
ipynbのファイルを開く

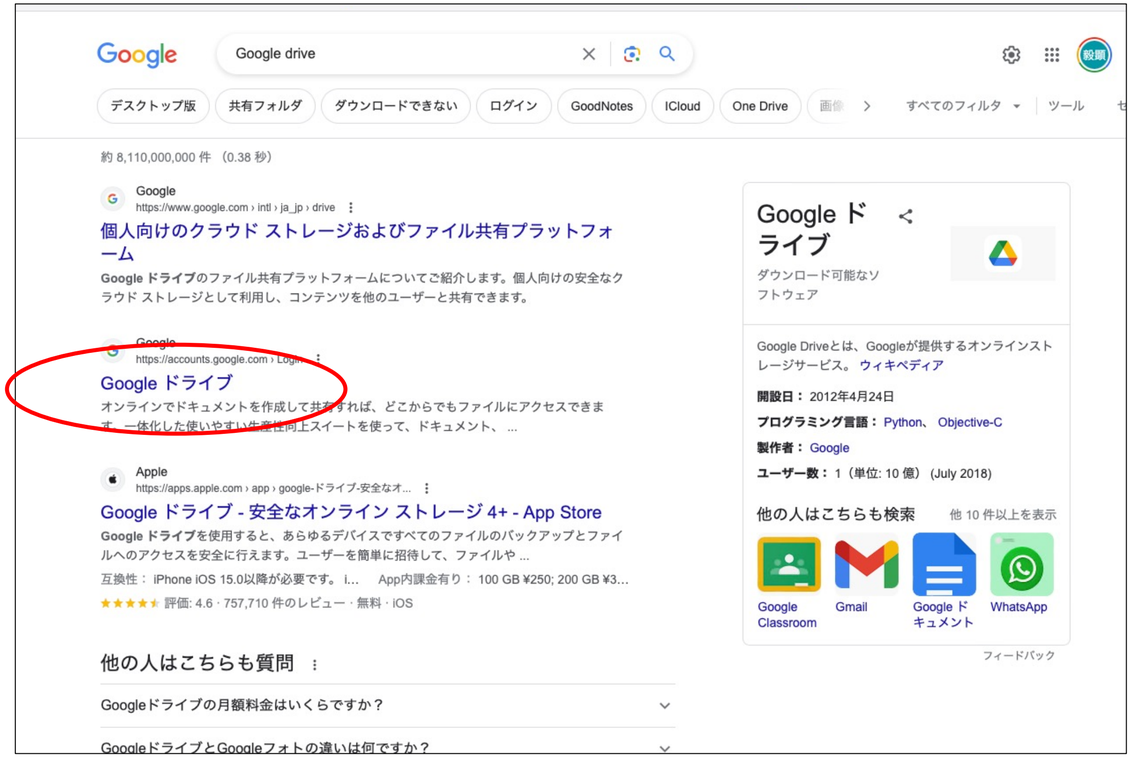
うまく開けた場合は、ファイルの内容が表示されます

アップロードがうまく行かない人は、次のページにある別の方法でアップロードします

ipynbのファイルを開く(2)

(アップロードがうまく行かない人用)
Google.comからアプリを選択もしくはGoogle Driveを検索します



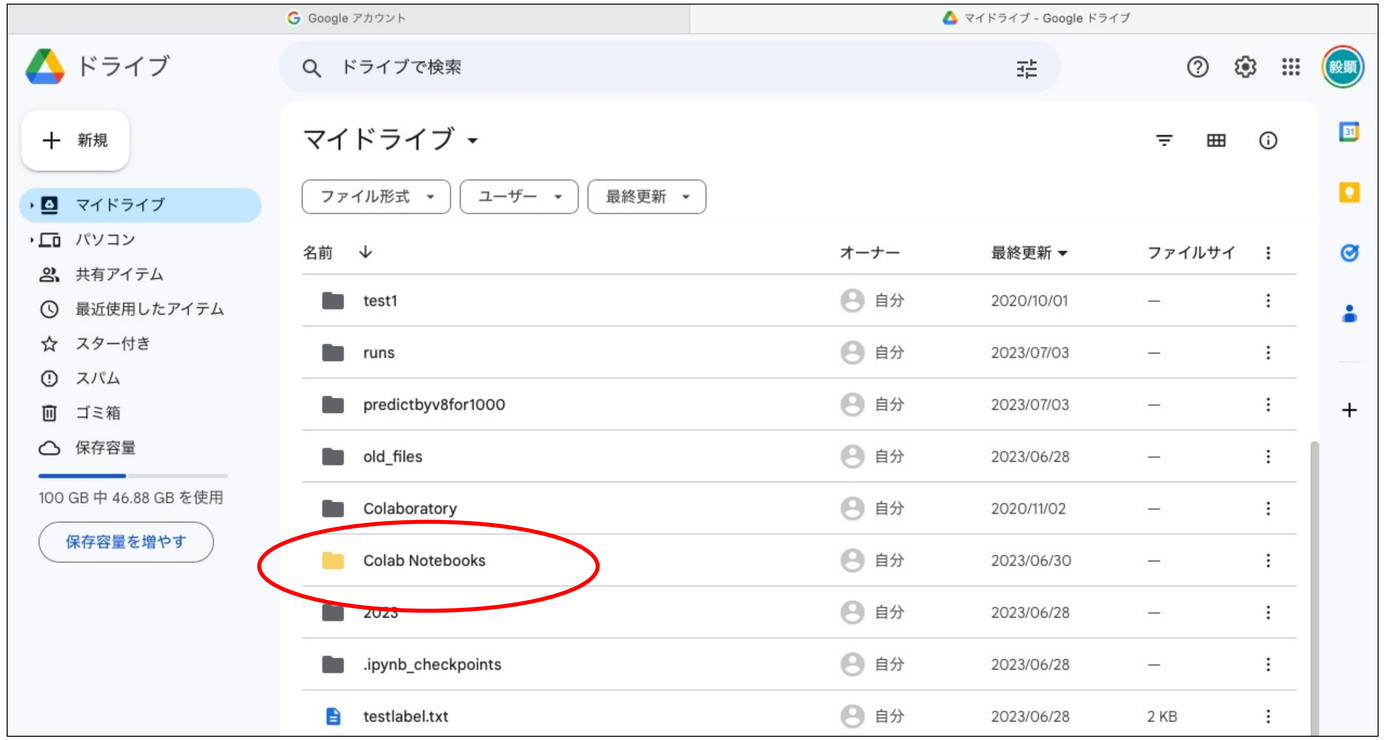


Google Driveを開きます

ipynbのファイルを開く(2)

(アップロードがうまく行かない人用)

「Colab Notebooks」をクリックしてその中にファイルをアップロード(ドラック&ドロップ)します

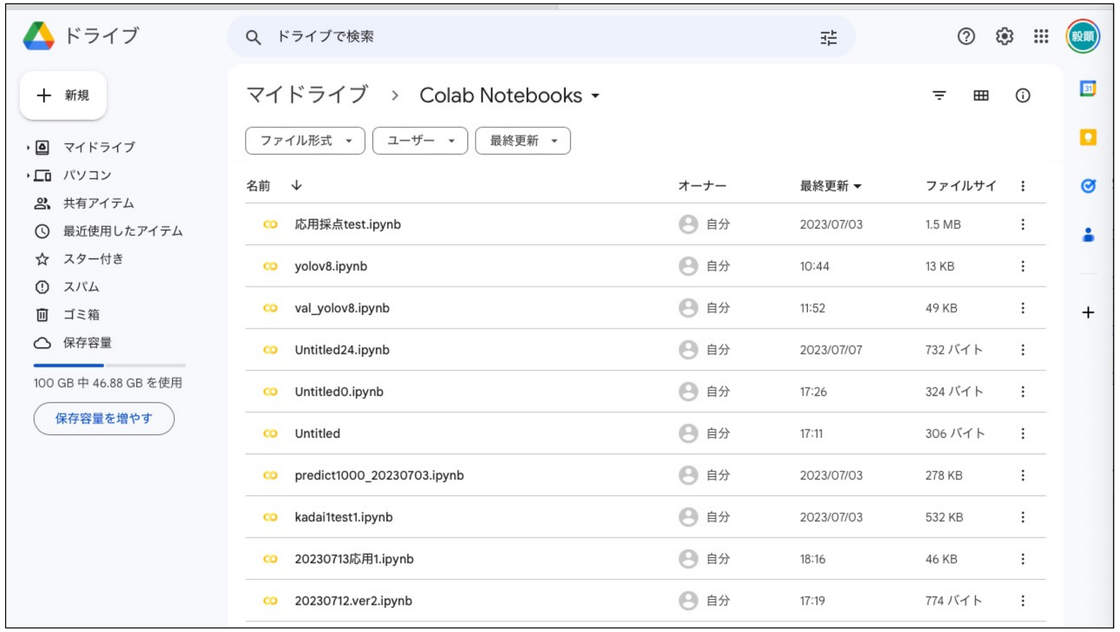


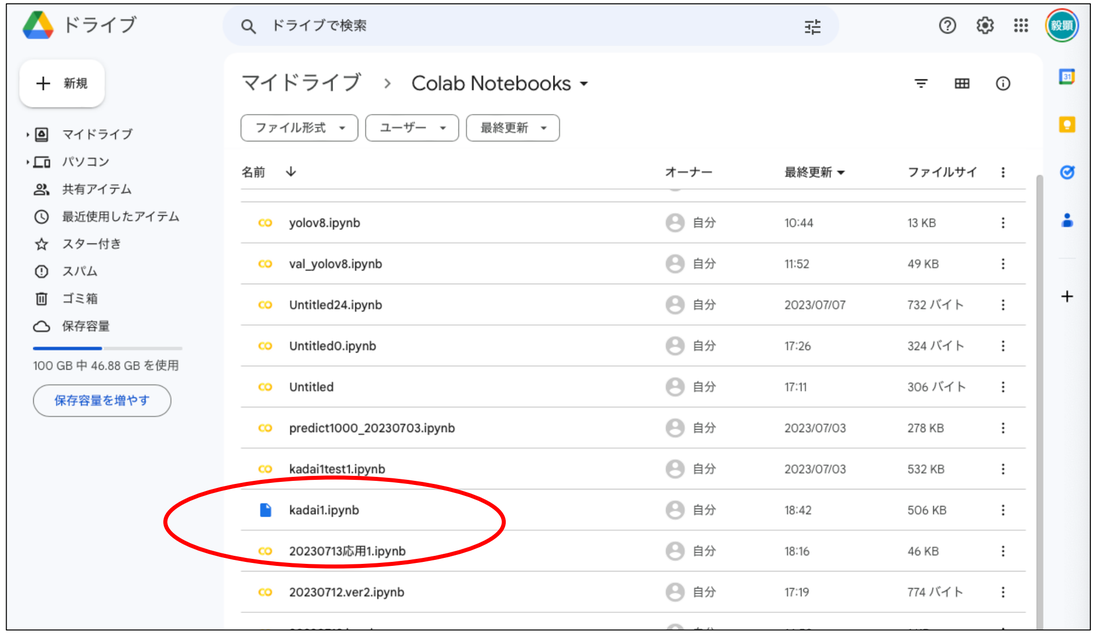
他の場所でも問題ないはずですが、Google Colabで作成したファイルはここに保存されます

ipynbのファイルを開く(2)

(アップロードがうまく行かない人用)

「Colab Notebooks」をクリックしてその中にファイルをアップロード(ドラック&ドロップ)します



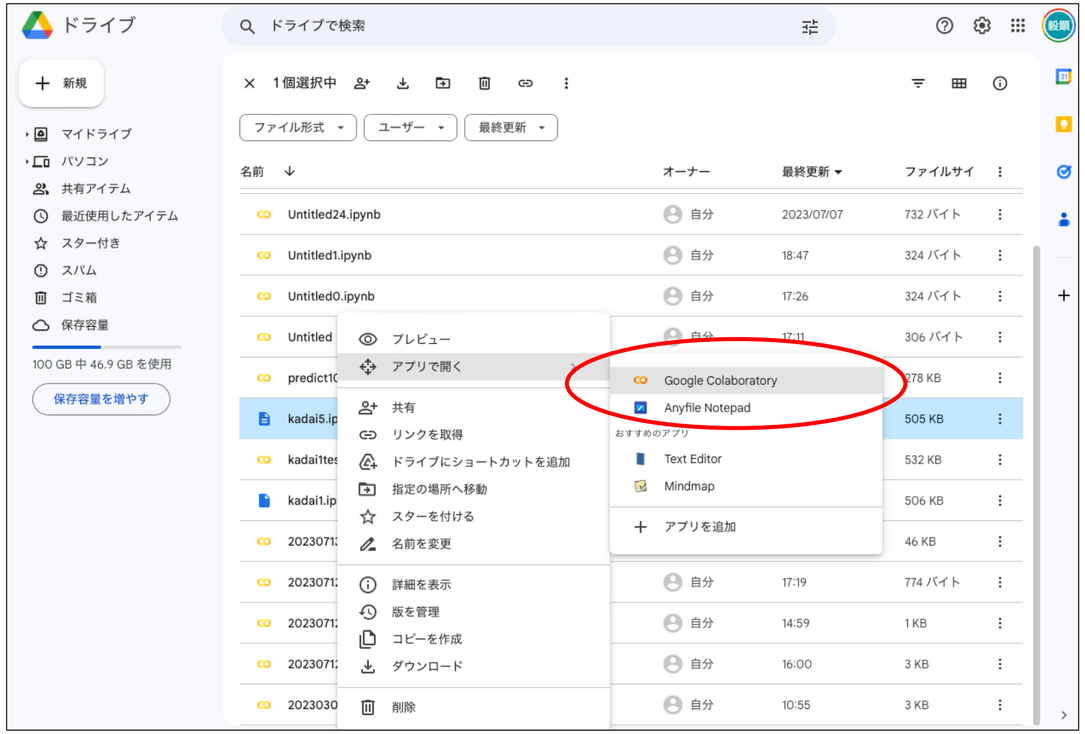


他の場所でも問題ないはずですが、Google Colabで作成したファイルはここに保存されます

ipynbのファイルを開く(2)

(アップロードがうまく行かない人用)

アップロードしたらファイルを右クリックして「アプリで開く」→「Google Colaboratory」を選びます



それでも上手く行かない人は聞いて下さい
(来週の金曜日17:00にzoomの質問コーナーを設けます)

演習2

Python基礎　Pythonの変数とデータの型

統合教育機構　曹　日丹

演習2-7の構成

Python基礎を学びましょう

演習2　11/16 11:35-12:20

Pythonの変数とデータの型

演習3

プログラミング基礎

演習4

モジュール、パッケージ、ライブラリ

Pythonを使ってみましょう

演習5

患者の歯に関する病院のリアルワールドデータの説明

架空データ

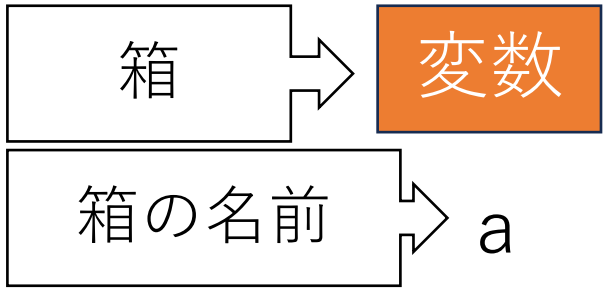
演習6

データクレンジングに必要なライブラリ（Pandas）の応用

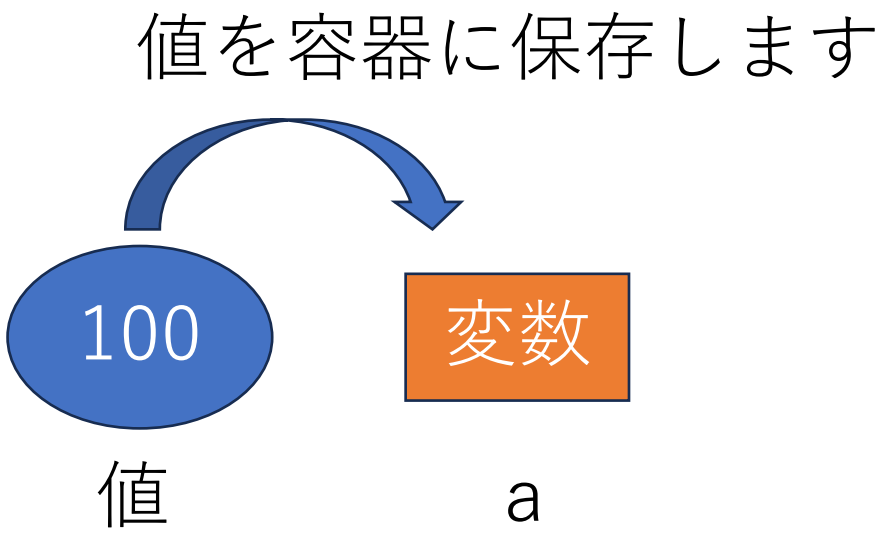
演習7

データクレンジングとデータの可視化

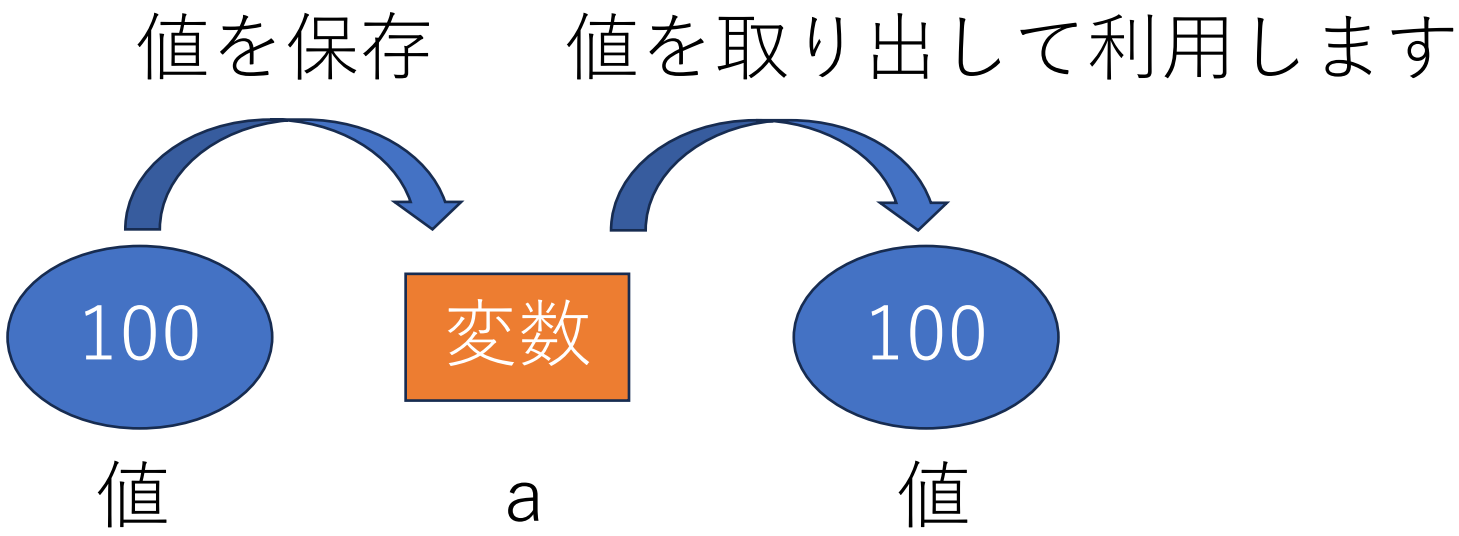
変数とは、データや値を一時的に保持するための名前付きの「箱」のようなものです。
この変数を使用することで、プログラム中でその値を繰り返し利用することができます。



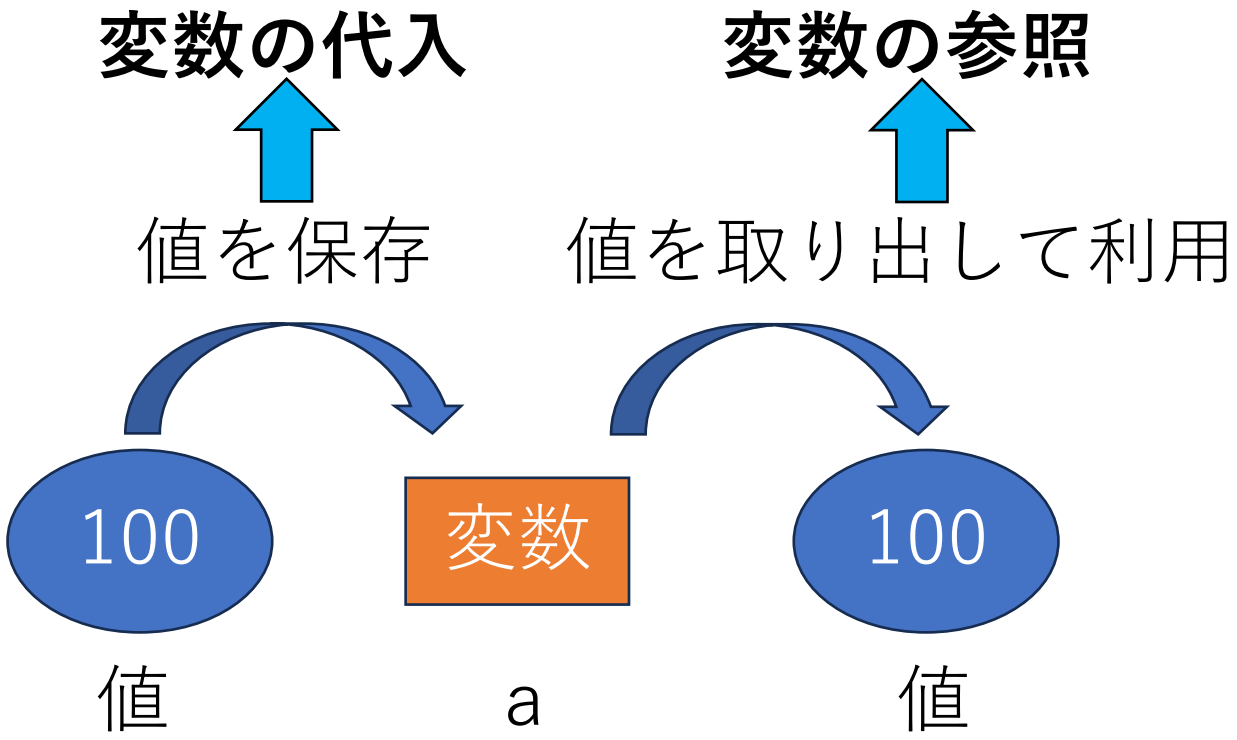
変数とは、データや値を一時的に保持するための名前付きの「容器」または「箱」のようなものです。
この変数を使用することで、プログラム中でその値を繰り返し利用することができます。



変数とは、データや値を一時的に保持するための名前付きの「容器」または「箱」のようなものです。この変数を使用することで、プログラム中でその値を繰り返し利用することができます。



変数とは、データや値を一時的に保持するための名前付きの「容器」または「箱」のようなものです。この変数を使用することで、プログラム中でその値を繰り返し利用することができます。

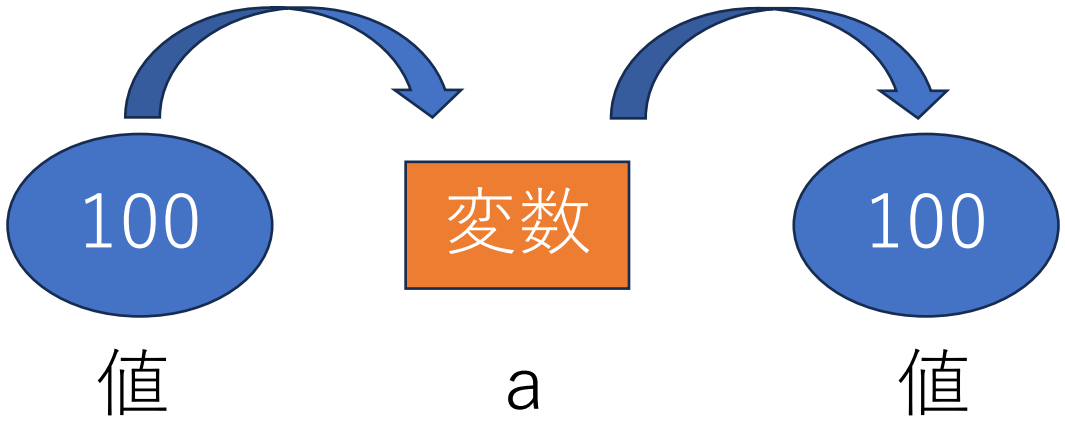


変数とは、データや値を一時的に保持するための名前付きの「容器」または「箱」のようなものです。この変数を使用することで、プログラム中でその値を繰り返し利用することができます。

コード01

プログラミング言語で表現します

変数の代入	変数の参照
a= 100	print(a)
値を保存	値を取り出して利用

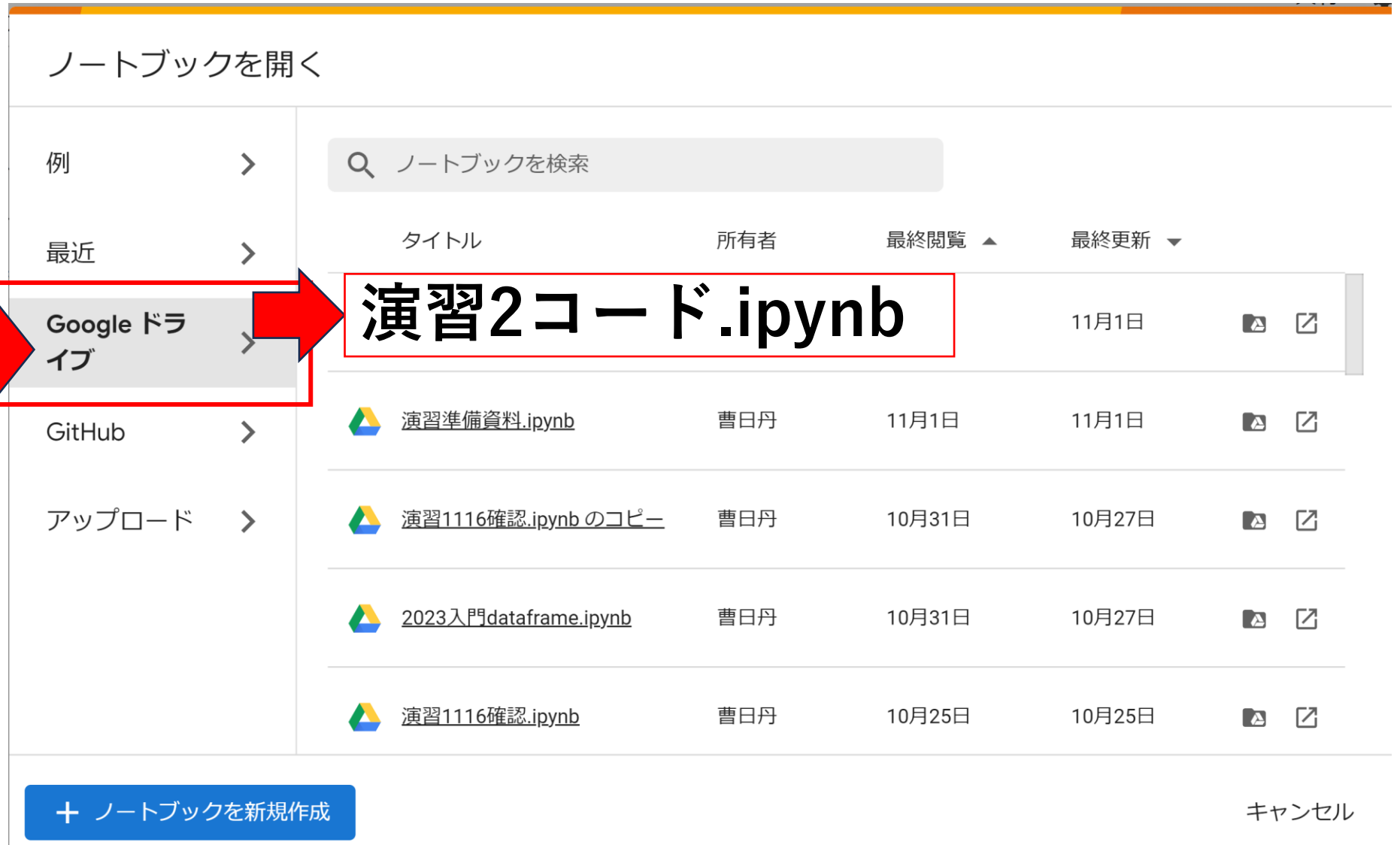


Colab

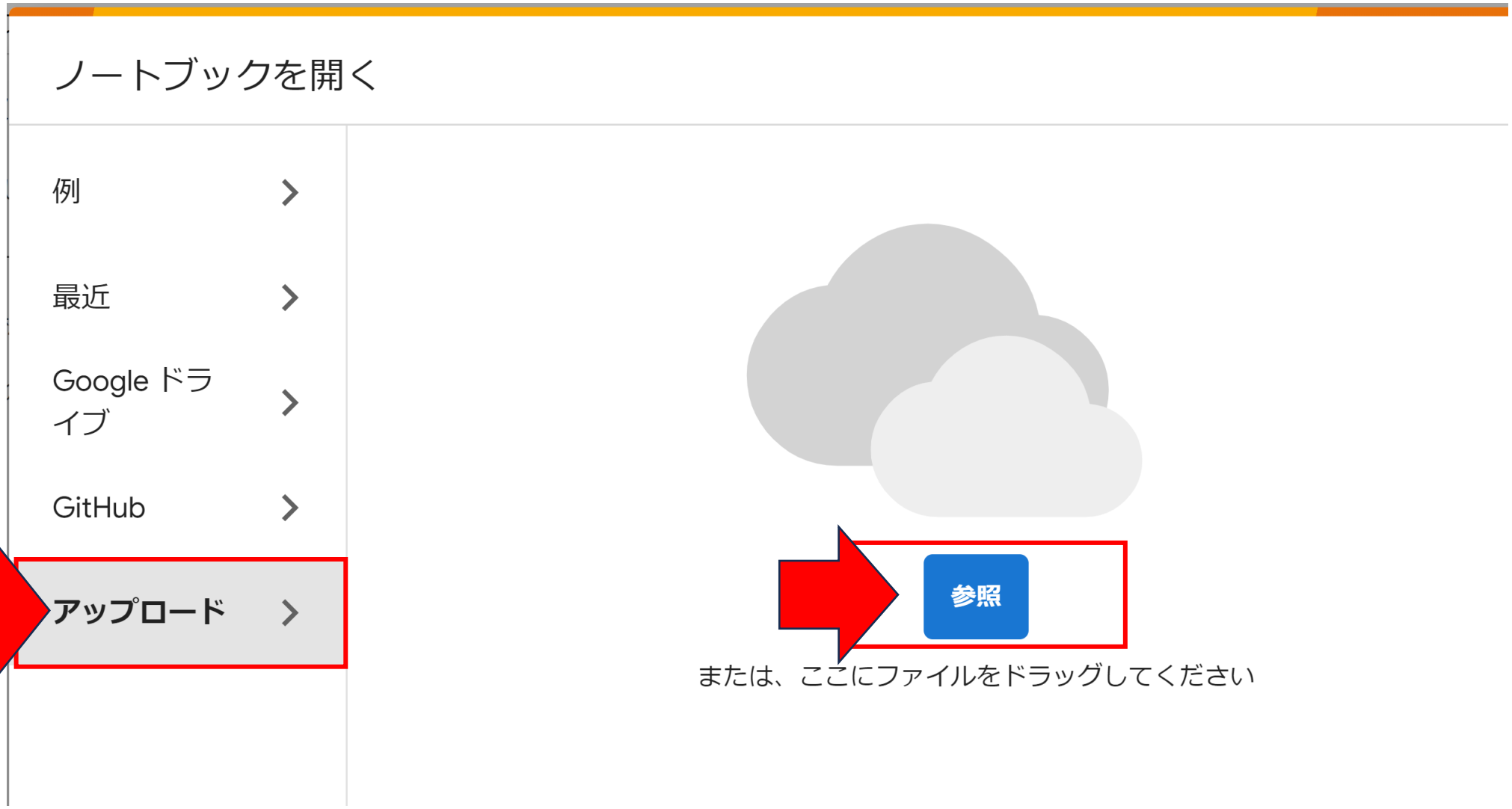
検索google colab　Colaboratory へようこそ - Colaboratory - Google

検索google colab　Colaboratory へようこそ - Colaboratory - Google

検索google colab　Colaboratory へようこそ - Colaboratory - Google



検索google colab　Colaboratory へようこそ - Colaboratory - Google



演習：コード01を書いてみましょう

GoogleColabでノートブックを開きましょう

[ ]

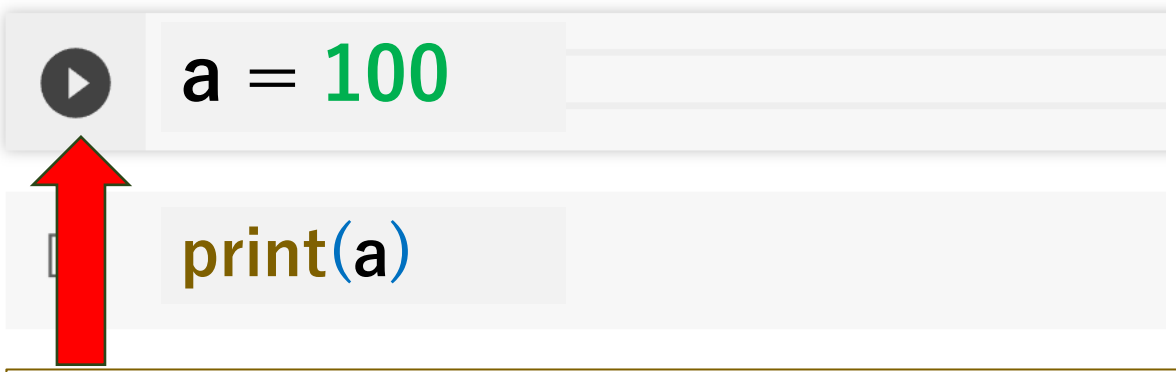
[ ]

[ ]

演習：コード01を書いてみましょう

コード01

```
a = 100
print(a)
```

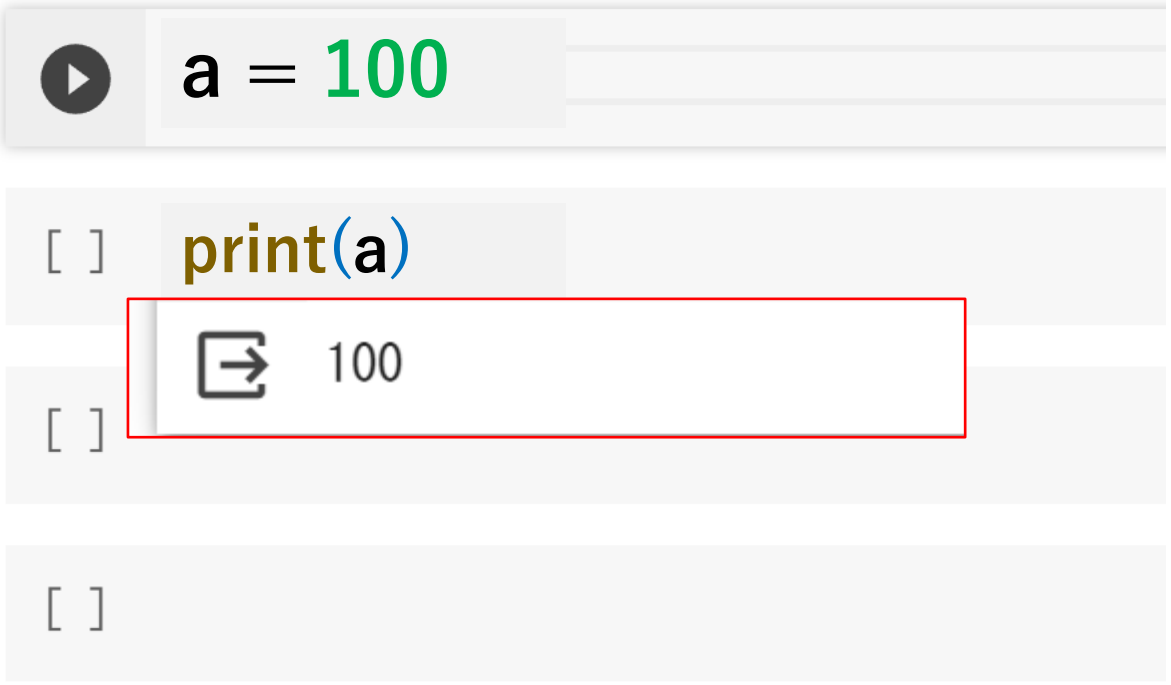


実行ボタンを押す、またはShift+Enterを押すと現在のセルのコードを実行できます。

演習：コード01を書いてみましょう

コード01

```
a = 100
print(a)
```

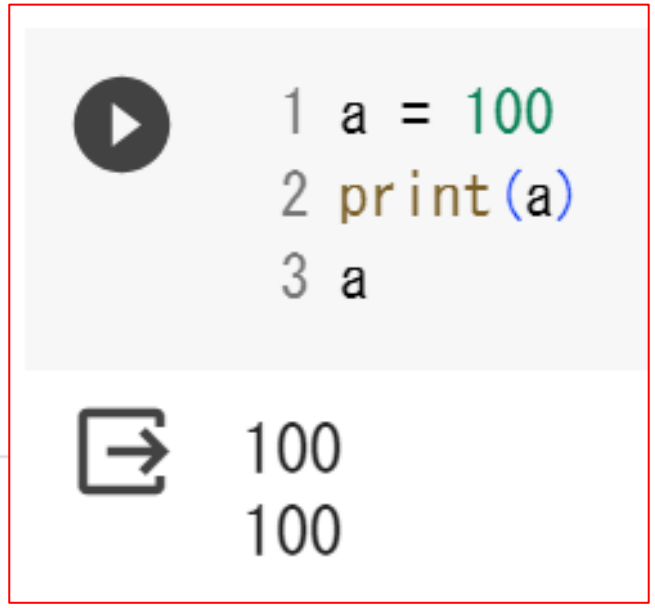


Colab

演習：コード01を書いてみましょう

コード01

```
a = 100
print(a)
```



演習：コード02を書いてみましょう

コード02

```
a= "hello world "
print(a)
a= [11, 16]
print(a)
```

文字を入力するため""が必要です。

リストを作成するため[]が必要です。

```
a= "hello world"
```

```
print(a)
```

```
a= [11, 16]
```

```
print(a)
```

演習：コード02を書いてみましょう

```
a= "hello world"  # ""（ダブルクォーテーション）の中に文字を入れてみましょう

print(a)

a= [11, 16]       # []角括弧の中にリストを作りましょう

print(a)
```

```
hello world
[11, 16]
```

記号は、コメントを示すために使われます。
コメントは、コードを実行する際には無視されるテキスト

1回目に実行した結果　文字

2回目に実行した結果　リスト

Colab

データの型：

Pythonにはさまざまなデータ型が存在します。以下はPythonの基本的なデータ型です：

基本データ型

コンテナデータ型

データの型：

Pythonにはさまざまなデータ型が存在します。以下はPythonの基本的なデータ型です：

基本データ型

- int(整数)
- float(浮動小数点数)

コンテナデータ型

- str (文字列)
- list (リスト)
- tuple (タプル)
- set (セット)
- dict (辞書)

データの型：

Pythonにはさまざまなデータ型が存在します。以下はPythonの基本的なデータ型です：

基本データ型

int(整数)
a = 5

float(浮動小数点数)
b = 5.0

コンテナデータ型

str (文字列)
greeting = "Hello, World!"

list (リスト)
numbers = [1, 2, 3, 4, 5]

tuple (タプル)
colors = ("red", "green", "blue")

set (セット)
fruits = {"apple", "banana", "cherry"}

dict (辞書)
person = {"name": "John", "age": 30, "city": "New York"}

デモ：データの型を確認しましょう

基本データ型

```
[ ]  a = 5
     b = 5.0
     c = 3+4*b
```

コンテナデータ型

```
     greeting = "Hello, World!"
     numbers = [1,2,3,4,5]
     colors = ("red","green","blue")
     person = {"name":"John","age":30, "city":"New York"}
     fruits = {"apple", "banana","cherry"}
```

デモ：データの型を確認しましょう

type関数で型を確認できます

```
type(a)
```

```
type(c)
```

```
type(numbers)
```

```
```

変数エクスプローラで型を確認できます

変数

名前	型	形状	値
a	int		5
b	float		5.0
c	float		23.0
colors	tuple	3 items	('red', 'green', 'blue')
fruits	set	3 items	{'apple', 'banana', 'cherry'}
greeting	str	13 chars	'Hello, World!'
numbers	list	5 items	[1, 2, 3, 4, 5]
person	dict		{'name': 'John', 'age': 30, 'c

Colab

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

リスト（list）の特徴：

順序付けられた要素の集まり
要素の**重複が可能**
異なるデータ型の要素を含むことができる
要素の追加、削除、**変更が可能**
各要素は**インデックス**（位置）によってアクセスされる。

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

リスト（list）の特徴：

順序付けられた要素の集まり
要素の**重複が可能**
異なるデータ型の要素を含むことができる
要素の追加、削除、**変更が可能**
各要素は**インデックス**（位置）によってアクセスされる。

セット（set)特徴:

重複しない要素の集まり。
要素の追加や削除は可能。
順序付けられない。

タプル（Tuple）特徴:

順序付けられた要素の集まり。
変更不可能です。

辞書 (Dictionary)特徴:

順序付けられたキーと値の集まり。
要素の追加、削除、変更することが可能。

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード03を書いてみましょう

リストを作成します。

```
fruits = [      ,      ,      ,      ]
```

文字の要素で構成されています。

```
numbers = [ ,  ,  ,  ]
```

数字の要素で構成されています。

```
mixed = [数字, 文字, リスト]
```

複数の要素で構成されています。

角括弧で定義されます、要素の間にカンマで区切られます。

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード03を書いてみましょう

リストを作成します。

```
fruits = ["apple" , "banana" , "cherry",  “melon"]
```

文字の要素で構成されています。

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

数字の要素で構成されています。

```
[ ]  mixed = [1, "apple", 3.14, [5, 6, 7]]
```

複数の要素で構成されています。

```
[ ]
```

リストでは、異なるデータ型の要素を含むことができます。

Colab

Python基礎　Pythonの変数とデータの型

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード03を書いてみましょう

```
fruits = ["apple" , "banana" , "cherry", "melon"]
```

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

```
mixed = [1, "apple", 3.14, [5, 6, 7]]
```

```
type(mixed)
```

type()関数を使って型を確認できます。

Colab

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

リスト（list）の特徴：

順序付けられた要素の集まり
要素の**重複が可能**
異なるデータ型の要素を含むことができる
要素の追加、削除、**変更が可能**
各要素は**インデックス**（位置）によってアクセスされる。

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード04を書いてみましょう

インデックスを使用して要素にアクセスします

```
fruits = ["apple" , "banana" , "cherry", "melon"]
```

```
[ ]  print(fruits[    ])
```

角括弧にインデックス数字を入力します。

```
[ ]  print(fruits[    ])
```

角括弧にインデックス数字を入力します。

```
[ ]
```

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード04を書いてみましょう

インデックスを使用して要素にアクセスします

```
fruits = ["apple" , "banana" , "cherry", "melon"]
```

```
print(fruits[0])
```

```
[→] apple
```

“apple”を取り出しました

```
print(fruits[-1])
```

```
[→] melon
```

"melon"を取り出しました

```
fruits = ["apple" , "banana" , "cherry", “melon"]
```

fruits[0]　[1]　[2]　[3]

[-3]　[-2]　[-1]

インデックスを使用してリストの要素にアクセスします

```
fruits = ["apple" , "banana" , "cherry", "melon"]
```

fruits[0]	fruits[1]	fruits[2]	fruits[3]
fruits[-4]	fruits[-3]	fruits[-2]	fruits[-1]

リスト内の各要素は、0から始まるインデックス番号を持ちます。
最初の要素はインデックス0にあり、次の要素は1、2、3と続きます。
このインデックスを使用して、特定の要素をアクセスできます。

コード04 Colab

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード05を書いてみましょう

スライスを使用して部分リストを取得する:

```
fruits = ["apple" , "banana" , "cherry", "melon"]
```

```
[ ]  print(fruits[ : ])
```

角括弧に範囲を表す数字を入力します。

```
[ ]  print(fruits[ : ])
```

角括弧に範囲を表す数字を入力します。

```
[ ]
```

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード05を書いてみましょう

スライスを使用して部分リストを取得する:

```
fruits = ["apple" , "banana" , "cherry", "melon"]
```

```
print(fruits[1:3])
```

```
[→] ["banana", "cherry"]
```

```
print(fruits[:2])
```

```
[→] ["apple", "banana"]
```

```
fruits = ["apple" , "banana" , "cherry",  “melon"]
```

fruits[0]　[1]　[2]　[3]

fruits[0]　[1]　[2]　[3]

スライスを使用することで、リスト内の特定の範囲の要素を取り出すことができます。

```
fruits = ["apple" , "banana" , "cherry", "melon"]
```

fruits[0]　fruits[1]　fruits[2]　fruits[3]

fruits[1]　fruits[2]　fruits[1:3]

fruits[0]　fruits[1]　fruits[0:2]

スライスは次の形式を取ります：start:end:step。
startはスライスの開始位置を示します、
endはスライスの終了位置を示します（この位置は含まれません）
stepはステップ数を指定します（指定しない場合はデフォルトで1となります）

コード05 Colab

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

リスト（ list ）の特徴：

順序付けられた要素の集まり
要素の**重複が可能**
異なるデータ型の要素を含むことができる
要素の**追加、削除、変更が可能**
各要素はインデックス（位置）によってアクセスされる。

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード06を書いてみましょう

リストの要素を変更します

```
fruits = ["apple" , "banana" , "cherry", "melon"]
```

```
fruits[ ] = "blueberry"
```

変更したい要素のインデックス数字を入力します。

```
print(fruits)
```

```
```

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード06を書いてみましょう

リストの要素を変更します

```
1 fruits = ["apple" , "banana" , "cherry", "melon"]
```

```
1 fruits[1] = "blueberry"
```

```
1 print(fruits)
```

```
[→] ["apple", "blueberry", "cherry", "melon"]
```

```
fruits = ["apple" , "banana", "cherry", "melon"]
```

fruits[0]　[1]　[2]　[3]

コード06 Colab

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード07を書いてみましょう

リストの要素を追加、削除します:

```
fruits.append("〇〇")
```

追加したい要素を入力します。

```
[ ] print(fruits)
```

新しいリストを表示します。

```
[ ] fruits.remove("〇〇")
```

削除したい要素を入力します。

```
[ ] print(fruits)
```

新しいリストを表示します。

Pythonにおける**set**、**dict（辞書）**、**tuple**、**list**は、いずれも複数の要素を格納できるデータ構造ですが、それぞれ異なる特性や用途を持っています。

演習：コード07を書いてみましょう

リストの要素を追加、削除します:

```
fruits.append("strawberry")
```

追加したい要素を入力します。

```
print(fruits)
```

[→] ["apple" , "banana" , "cherry", "melon", "strawberry"]

```
fruits.remove("cherry")
```

削除したい要素を入力します。

```
print(fruits)
```

[→] ["apple" , "banana" , "melon", "strawberry"]

コード07 Colab

さまざまな関数を使用してリストの計算ができます。

sum 関数：数値を含むリストの全要素の合計を計算する関数です。

①

```
total = sum([1,3,5,7,11])
print(total)
```

括弧の中数字を入れます。

②

```
リスト = [　　,　　,　　,　]
```

```
total = sum(リスト名)
print(total)
```

括弧の中数字から構成されている
リスト名を入れます。

さまざまな関数を使用してリストの計算ができます。

sum 関数：数値を含むリストの全要素の合計を計算する関数です。

①

```
total = sum([1,3,5,7,11])
print(total)
```

括弧の中数字を入れます。

```
27
```

②

```
リスト = [a1, a2, a3, a4]
```

括弧の中数字から構成されている
リスト名を入れます。

```
total = sum(リスト名)
print(total)
```

```
a1+ a2+ a3+ a4
```

さまざまな関数を使用してリストの計算ができます。

演習：コード08を書いてみましょう

リストの合計を計算します:

```
numbers = [10,20,30,40,50,60]
```

```
total = sum(numbers)

print( total )
```

```
[→] 210
```

括弧の中数字から構成されているリスト名を入れます。

コード08 Colab

さまざまな関数を使用してリストの計算ができます。

len関数：あるオブジェクトが持つ要素の数を返す組み込み関数です。

① 文字数が集計されます。

② リストの要素の数が集計されます。

さまざまな関数を使用してリストの計算ができます。

len関数：あるオブジェクトが持つ要素の数を返す組み込み関数です。

① 文字数が集計されます。

```
greeting = " Hello World"
```

② リストの要素の数が集計されます。

```
リスト = [　　,　　,　　,　]
```

さまざまな関数を使用してリストの計算ができます。

len関数：あるオブジェクトが持つ要素の数を返す組み込み関数です。

①

```
greeting = "Hello World"
```

```
count = len(greeting)
print(count)
```

括弧の中変数名を入れます。

②

```
リスト = [　　,　　,　　,　]
```

```
total = len(リスト名)
print(total)
```

括弧の中リスト名を入れます。

さまざまな関数を使用してリストの計算ができます。

演習：コード09を書いてみましょう

リストの要素の数を取得します:

```
numbers = [10,20,30,40,50,60]
```

```
count = len(numbers)
```

```
print( count )
```

```
[→] 6
```

コード09 Colab

さまざまな関数を使用してリストの計算ができます。

max 関数：与えられたリストの中から最大値を返す組み込み関数です。

min 関数：与えられたリストの中から最小値を返す組み込み関数です。

```
min_value = min(リスト名)
```

```
[ ] print(min_value)
```

```
[ ] max_value = max(リスト名)
```

```
[ ] print(max_value)
```

さまざまな関数を使用してリストの計算ができます。

演習：コード10を書いてみましょう

リスト内の最小値、最大値を取得します:

```
min_value = min(numbers)
```

```
print(min_value)
```

[→] 10

```
max_value = max(numbers)
```

```
print(max_value)
```

[→] 60

コード10 Colab

さまざまな関数を使用してリストの計算ができます。

演習：コード11を書いてみましょう

リスト内の要素の平均値を計算します:

```
numbers = [10,20,30,40,50,60]
```

```
average =sum(numbers) / len(numbers)
```

```
print(average)
```

```
[→] 35.0
```

Pythonは整数同士の割り算でも小数（浮動小数点数）の結果を返すように変更されました。

コード11Colab

演習：コード12を書いてみましょう

リストとリストを結合することもできます

```
newlist = リスト1名 + リスト2名
```

```
[ ] print(newlist)
```

```
[ ]
```

演習：コード12を書いてみましょう

リストとリストを結合することもできます

```
newlist =  fruits + numbers
```

```
numbers = [10,20,30,40,50,60]
```

```
fruits = ["apple", "banana", "cherry", "melon"]
```

```
print(newlist)
```

```
['apple', 'banana', 'cherry', 'melon',
10, 20, 30, 40, 50, 60]
```

コード12 Colab

課題：11月30日23:59までWebClassで提出してください。

課題1

下記5人の成績表リストを作成し、print一つを使って3番目と4番目の人の成績を表示するようにコードを書いてください。

成績表：97, 67, 89, 100, 87

課題2

a = "A Python list is an ordered collection of items. This means that each item in the list has a definite order, and that order will not change unless the list is explicitly modified."
Pythonの関数を使って変数aの文字数を答えてください。

演習2-7の構成

Python基礎を学びましょう

演習2　11/16 11:35-12:20

Pythonの変数とデータの型

演習3　11/30 11:35-12:20

プログラミング基礎

演習4

モジュール、パッケージ、ライブラリ

Pythonを使ってみましょう

演習5

患者の歯に関する病院のリアルデータの説明

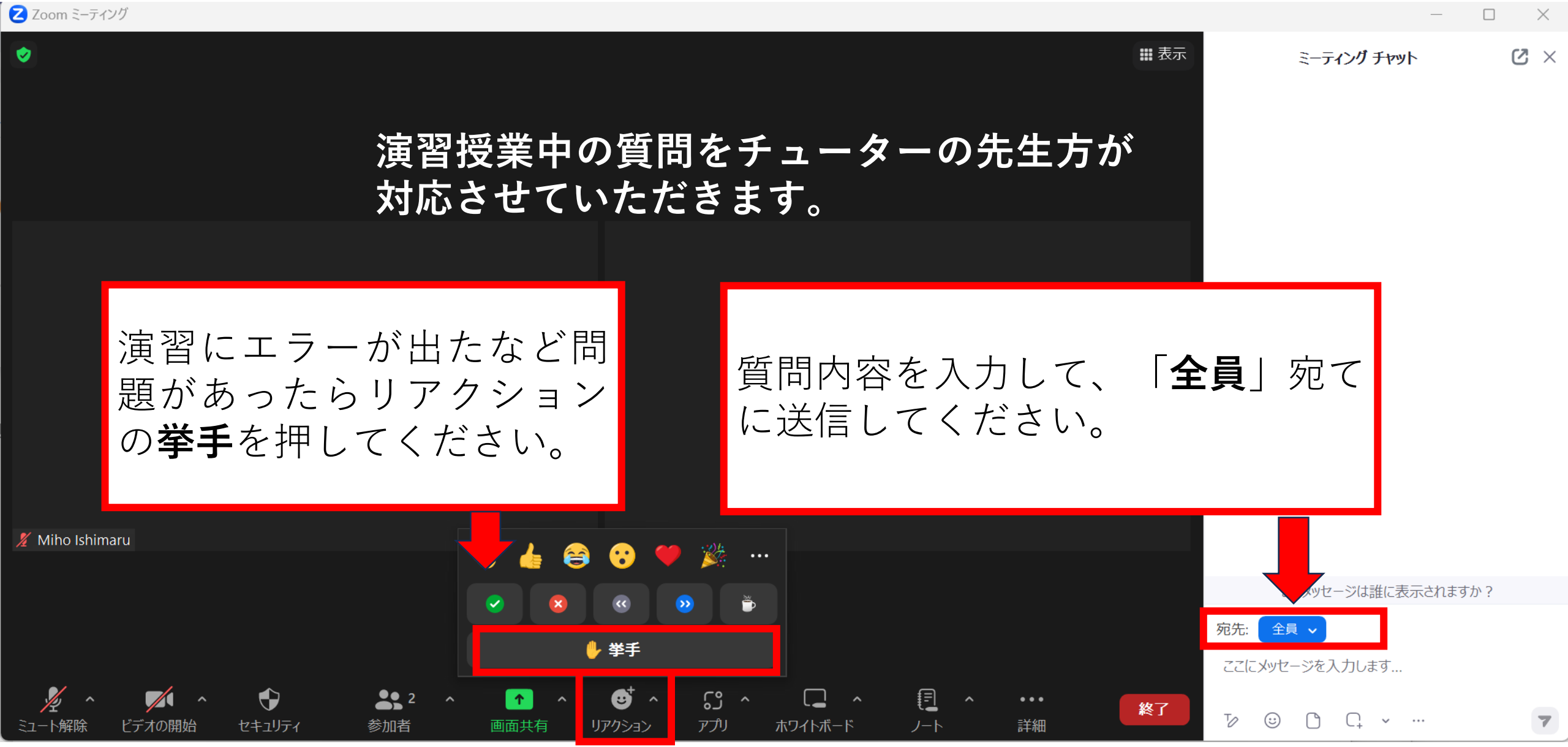
演習6

データクレンジングに必要なライブラリ（Pandas）の応用

演習7

データクレンジングとデータの可視化

演習授業中の質問対応について



演習3

Python基礎　プログラミング基礎

統合教育機構　曹　日丹

演習2-7の構成

Python基礎を学びましょう

演習2　11/16 11:35-12:20

Pythonの変数とデータの型

演習3　11/30 11:35-12:20

プログラミング基礎

演習4

モジュール、パッケージ、ライブラリ

Pythonを使ってみましょう

演習5

患者の歯に関する病院のリアルワールドデータの説明

演習6

データクレンジングに必要なライブラリ（Pandas）の応用

演習7

データクレンジングとデータの可視化

プログラミング（コーディングとも呼ばれます）は、
コンピュータに特定のタスクを実行させるための命令を書くプロセスです。プログラミング言語を使用して、コンピュータに対して**明確で順序だった命令のセット**を提供します。

コンピュータは人間とは異なり、曖昧な命令や指示を理解することができません。

コンピュータは命令を上から下へと一つずつ順番に実行します。

Python基礎　プログラミング基礎

プログラミング（コーディングとも呼ばれます）は、
コンピュータに特定のタスクを実行させるための命令を書くプロセスです。プログラミング言語を使用して、コンピュータに対して**明確で順序だった命令のセット**を提供します。

if と for　Pythonのプログラムで非常に頻繁に使用される構造です。

if文（条件分岐）:
特定の条件が真(True)か偽(False)かに基づいて**プログラムの実行経路**（**コードの実行フローを制御する**）を変更します。

for文（繰り返し）:
データの集まりの中、要素ごとに一連の操作を**繰り返し実行**します。繰り返す回数は、要素の数で決まります。

if と for

Pythonのプログラムで非常に頻繁に使用される構造です。

if文（条件分岐）:
if文は条件分岐を実現するためのもので、特定の条件が真(True)か偽(False)かに基づいて**プログラムの実行経路**を変更します。

日本語での**if文**に相当する部分は「**もし～なら**」や「もし条件が成り立つ場合は」などのように言えます。

for文（繰り返し）:
データの集まりの中、要素ごとに一連の操作を**繰り返し実行**します。繰り返す回数は、要素の数で決まります。

日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

if

日本語での**if文**に相当する部分は「**もし～なら**」や「もし条件が成り立つ場合は」などのように言えます。

```
if 条件:
    条件が真の場合に実行されるコード
else:
    条件が偽の場合に実行されるコード
```

if

日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

```
if 条件:
    条件が真の場合に実行されるコード
else:
    条件が偽の場合に実行されるコード
```

真(True)か**偽**(False)

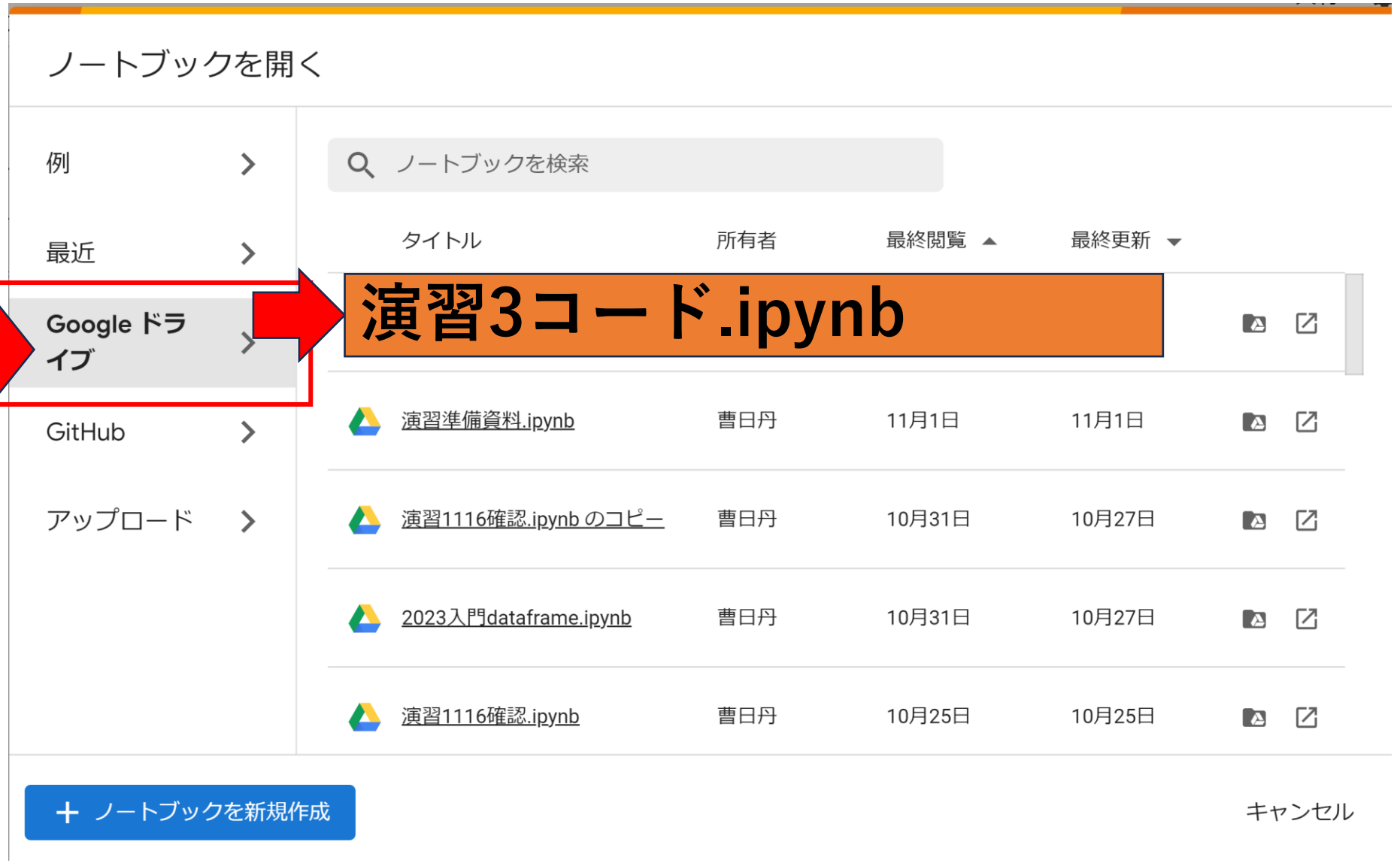
条件が真(True)か偽(False)かを判定するためには、**比較演算子**や**論理演算子**を使用します。

Colab

検索google colab　Colaboratory へようこそ - Colaboratory - Google

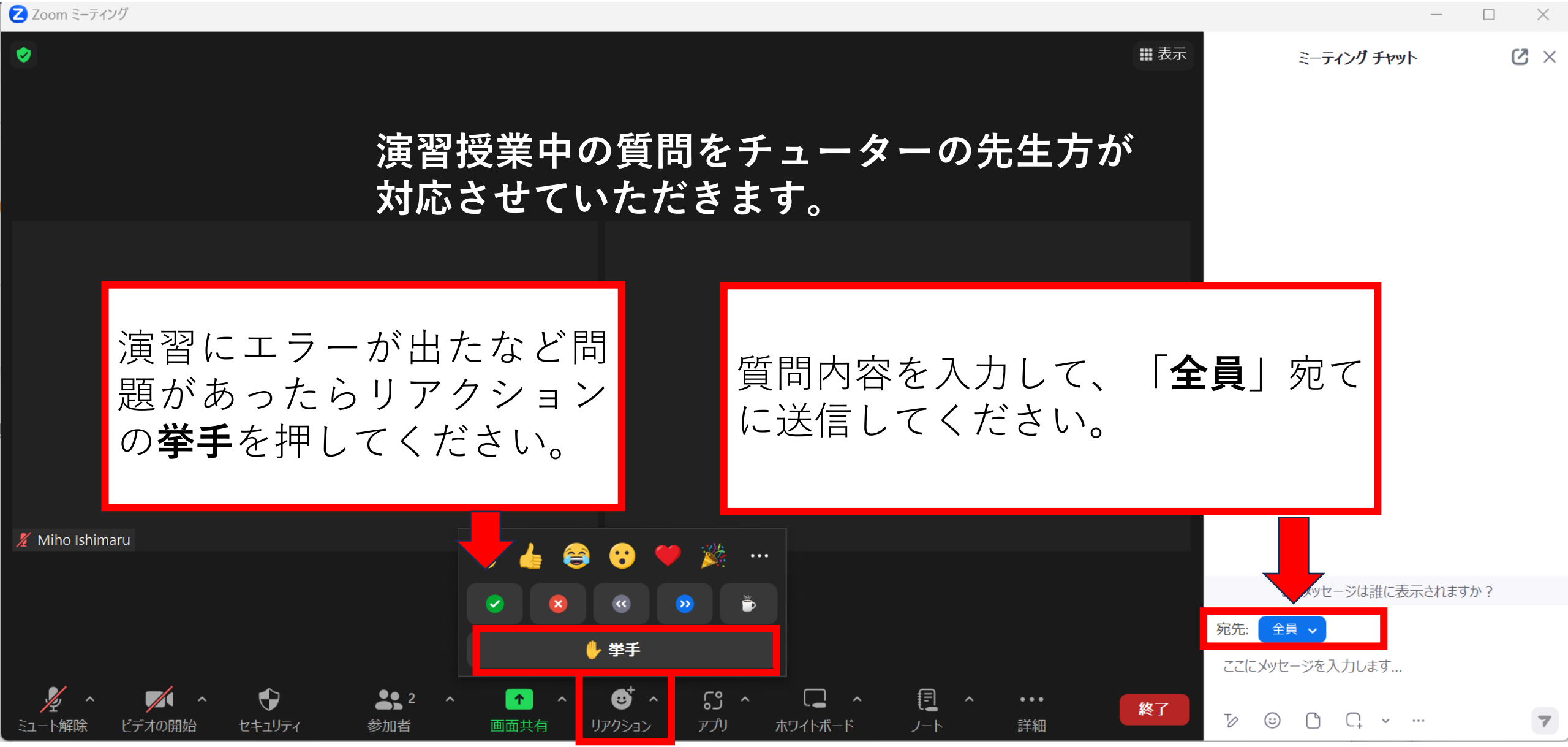
検索google colab　Colaboratory へようこそ - Colaboratory - Google

検索google colab　Colaboratory へようこそ - Colaboratory - Google



検索google colab　Colaboratory へようこそ - Colaboratory - Google

演習授業中の質問対応について



if

日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

デモ：ifを使って簡単なプログラムを書きましょう

age = ○○

```
if age < 18:                    条件を定義しました。
    print("未成年です。")        条件が真の場合に実行されるコード

else:
    print("成人です。")          条件が偽の場合に実行されるコード
```

if

日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

デモ：ifを使って簡単なプログラムを書きましょう

```
age = 18

if age < 18:
    print("未成年です。")

else:
    print("成人です。")
```

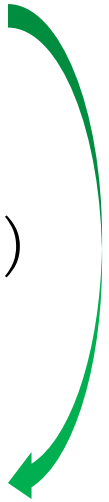
18歳は小なり18歳ではないため、条件が**偽**と判定しました。

if

日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

デモ：ifを使って簡単なプログラムを書きましょう

```
age = 18

if age < 18:
    print("未成年です。")

else:
    print("成人です。")
```



18歳は大なりイコール18歳であるため、実行されます

if

日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

デモ：ifを使って簡単なプログラムを書きましょう

```
age = 18

if age < 18:
    print("未成年です。")

else:
    print("成人です。")
```

18歳は大なりイコール18歳であるため、実行されます

成人です。

if

日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

デモ：ifを使って簡単なプログラムを書きましょう

```
age = 18

if age < 18:
    print("未成年です。")
else:
    print("成人です。")
```

Pythonのインデント（字下げ）：
・プログラムの構造を示すために使用スペースです
・インデントの位置によって、コードの開始と終了が判別されます。
・インデントは通常4つのスペースを使用することが推奨されています。

if

日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

デモ：ifを使って簡単なプログラムを書きましょう

```
age = 18
if age < 18:
    print("未成年です。")
else:
    print("成人です。")
```

比較演算子: 値や変数の比較を行います。
==：等しい
!=：等しくない
<：より小さい
>：より大きい
<=：以下
>=：以上

論理演算子: 2つ以上の条件を組み合わせて比較を行います。
and：かつ
or：または
not：でなければ

Colab

if

日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

```
if　条件1:
    #条件1がTrueの場合の処理
elif 条件2:
    #条件2がTrueの場合の処理
elif 条件3:
    #条件3がTrueの場合の処理
else: それ以外の場合
    #すべての条件がFalseの場合の処理
```

条件は2つ以上ある場合は、elifを使います。
elifは「else if」の短縮形です。日本語では
「**それともしも**」や「**そうではなくてもし**」
と言い換えることが一般的です。

条件1では**なくて**、
もし条件2に当てはまる**なら**

条件1でも**条件2**でも**なくて**、
もし条件3に当てはまる**なら**

デモ：if、elif、else文を使ってコードを書きましょう

if　日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

```
if 条件1:
  #条件2がTrueの場合の処理
elif 条件2:
  #条件2がTrueの場合の処理
elif 条件3:
  #条件3がTrueの場合の処理
elif 条件4:
  #条件4がTrueの場合の処理
else:
  #すべての条件がFalseの場合の処理
```

成績の値に基づいて評価します

成績	評価
90以上	A+
80以上	A
70以上	B
60以上	C
60未満	D

デモ：if、elif、else文を使ってコードを書きましょう

if　日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

```
if score < 60
  #条件1がTrueの場合の処理
elif 60 <= score < 70
  #条件2がTrueの場合の処理
elif 70 <= score < 80
  #条件3がTrueの場合の処理
elif 80 <= score <90
  #条件4がTrueの場合の処理
else:
  #すべての条件がFalseの場合の処理
```

成績の値に基づいて評価します

成績	評価
90以上	A+
80以上	A
70以上	B
60以上	C
60未満	D

```
score >= 90
```

デモ：if、elif、else文を使ってコードを書きましょう

if　日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

```
if score < 60
  #条件1がTrueの場合 "D"
elif 60 <= score < 70
  #条件2がTrueの場合 "C"
elif 70 <= score < 80
  #条件3がTrueの場合 "B"
elif 80 <= score <90
  #条件4がTrueの場合 "A"
else:
  #すべての条件がFalseの場合 "A+"
```

成績の値に基づいて評価します

成績	評価
90以上	A+
80以上	A
70以上	B
60以上	C
60未満	D

デモ：if、elif、else文を使ってコードを書きましょう

if　日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

```
if score < 60
  print( "D" )
elif 60 <= score < 70
  print( "C" )
elif 70 <= score < 80
  print( "B" )
elif 80 <= score <90
  print( "A" )
else:
  print( "A+" )
```

成績の値に基づいて評価します

成績	評価
90以上	A+
80以上	A
70以上	B
60以上	C
60未満	D

Colab

デモ：if、elif、else文を使ってコードを書きましょう

if　日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

```
score = 95
if score < 60
  print( "D" )
elif 60 <= score < 70
  print( "C" )
elif 70 <= score < 80
  print( "B" )
elif 80 <= score <90
  print( "A" )
else:
  print( "A+" )
```

成績の値に基づいて評価します

成績	評価
90以上	A+
80以上	A
70以上	B
60以上	C
60未満	D

Colab

デモ：if、elif、else文を使ってコードを書きましょう

if　日本語での**if文**に相当する部分は「**もし～なら**」や「**もし**条件が成り立つ**場合は**」などのように言えます。

```
score = 95
if score < 60:
    print( "D" )
elif 60 <= score < 70:
    print( "C" )
elif 70 <= score < 80:
    print( "B" )
elif 80 <= score <90:
    print( "A" )
else:
    print( "A+" )
```

× × × × ○

実行されます

成績の値に基づいて評価します

成績	評価
90以上	A+
80以上	A
70以上	B
60以上	C
60未満	D

Colab

if と for

Pythonのプログラムで非常に頻繁に使用される構造です。

if文（条件分岐）:
if文は条件分岐を実現するためのもので、特定の条件が真(True)か偽(False)かに基づいて**プログラムの実行経路**（**コードの実行フローを制御する**）を変更します。

日本語での**if文**に相当する部分は「**もし～なら**」や「もし条件が成り立つ場合は」などのように言えます。

for文（繰り返し）:
データの集まりの中、要素ごとに一連の操作を**繰り返し実行**します。繰り返す回数は、要素の数で決まります。

日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

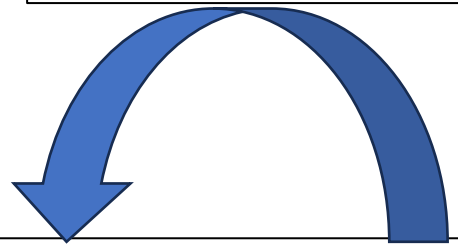
for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

```
for 変数 in 要素の集まり:
    繰り返し実行するコード
```

forブロック

for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

繰り返すたびにデータの集まりの要素を順番に変数に代入します。



```
for 変数 in 要素の集まり:
    繰り返し実行するコード
```

要素の集まり：

リスト　fruits= ["apple", "banana", "cherry"]
タプル　fruits = ("apple", "banana", "cherry")
辞書　　fruits = {"apple": "red", "banana": "yellow", "cherry": "red"}
セット　fruits = {"apple", "banana", "cherry"}

整数列：

range(n): 0以上n未満までの範囲の整数列を作成されます。

for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

デモ：forを使ってプログラムを書きましょう

1. リストの各要素を表示する

```
fruits = ["apple", "blueberry", "melon", "strawberry"]
```

リストの要素を一つずつ取り出します

```
print(fruits[0])
print(fruits[1])
print(fruits[2])
print(fruits[3])
```

```
apple
blueberry
melon
strawberry
```

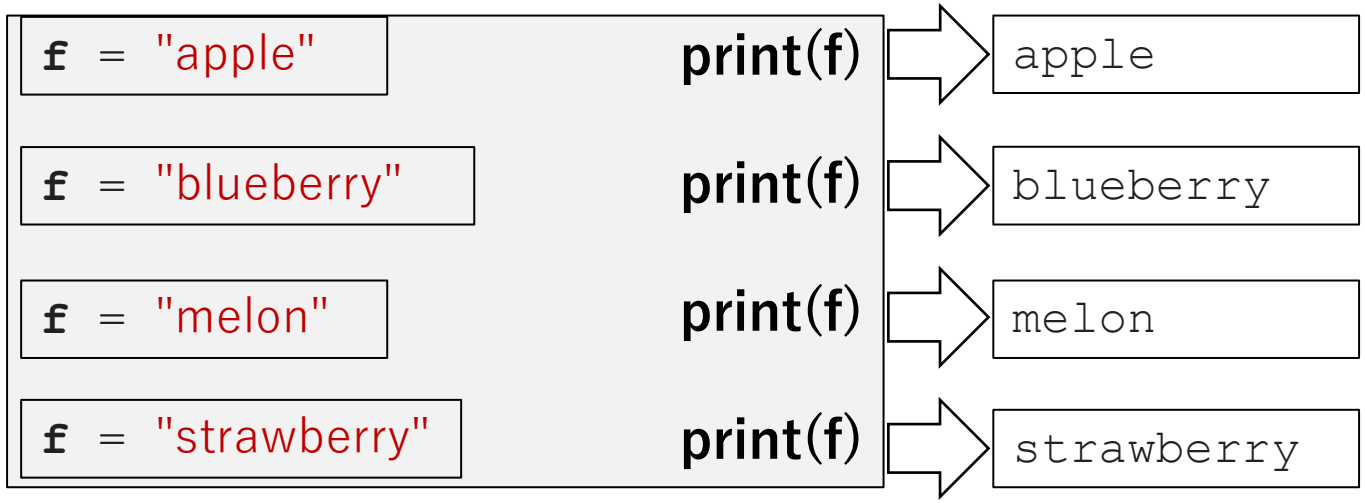
for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

デモ：forを使ってプログラムを書きましょう

1. リストの各要素を表示する

```
fruits = ["apple", "blueberry", "melon", "strawberry"]
for f in fruits:
    print(f)
```

リストの要素を一つずつ取り出します



リスト fruits 内の各要素が変数fに順番に代入され、print()関数によって画面に表示されます。結果として、リスト内の要素が順番に表示されることになりました。

Colab

for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

デモ：forを使ってプログラムを書きましょう

2. 0から3までの整数を順番に表示します

整数の集まり：
range(n): 0以上n未満までの範囲の整数列を作成されます。

```
for i in range(n):
    print(i)
```

range(n): a以上b未満までの範囲の整数列を作成されます。

```
for i in range(a, b):
    print(i)
```

for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

デモ：forを使ってプログラムを書きましょう

2. 0から3までの整数を順番に表示します

整数の集まり：
range(n): 0以上n未満までの範囲の整数列を作成されます。

```
for i in range(4):
    print(i)
```

range(n): a以上b未満までの範囲の整数列を作成されます。

```
for i in range(0, 4):
    print(i)
```

for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

デモ：forを使ってプログラムを書きましょう

2. 0から3までの整数を順番に表示します

整数の集まり：
range(n): 0以上n未満までの範囲の整数列を作成されます。

```
for i in range(4):
    print(i)
```

```
for i in range(0, 4):
    print(i)
```

```
i = 0
    print(i)
i = 1
    print(i)
i = 2
    print(i)
i = 3
    print(i)
```

```
0
1
2
3
```

```
0
1
2
3
```

range(n): a以上b未満までの範囲の整数列を作成されます。

Colab

for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

演習：コード02を書いてみましょう

5人の成績の値に基づいて評価します

```
scores = [99, 75, 59, 85, 60]

for s in scores:
    
    print("D")
    
    print("C")
    
    print("B")
    
    print("A")
  else:
    print("A+")
```

成績	評価	
90以上	A+	条件1
80以上	A	条件2
70以上	B	条件3
60以上	C	条件4
60未満	D	条件5

for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

演習：コード02を書いてみましょう

5人の成績の値に基づいて評価します

```
scores = [99, 75, 59, 85, 60]

for s in scores:
  if s < 60:
    print("D")
  elif 60 <= s <70 :
    print("C")
  elif 70 <= s <80:
    print("B")
  elif 80 <= s <90 :
    print("A")
  else:
    print("A+")
```

成績	評価	
90以上	A+	条件1
80以上	A	条件2
70以上	B	条件3
60以上	C	条件4
60未満	D	条件5

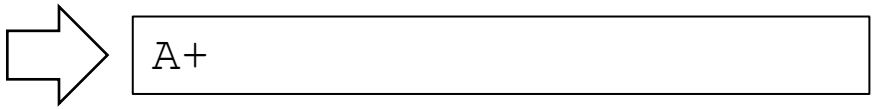
for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

演習：コード02を書いてみましょう

5人の成績の値に基づいて評価します

```
scores = [99, 75, 59, 85, 60]

s = 99
  if s < 60:      ×
    print("D")
  elif 60 <= s <70 :×
    print("C")
  elif 70 <= s <80: ×
    print("B")
  elif 80 <= s <90 :×
    print("A")
  else:           ○
    print("A+")
```

実行されます



for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

演習：コード02を書いてみましょう

5人の成績の値に基づいて評価します

```
scores = [99, 75, 59, 85, 60]

s = 75
  if s < 60:      ×
    print("D")
  elif 60 <= s <70 :×
    print("C")
  elif 70 <= s <80: ○
    print("B")
  elif 80 <= s <90 :
    print("A")
  else:
    print("A+")
```

実行されます

以下、実行されません

```
A+
```

```
B
```

for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

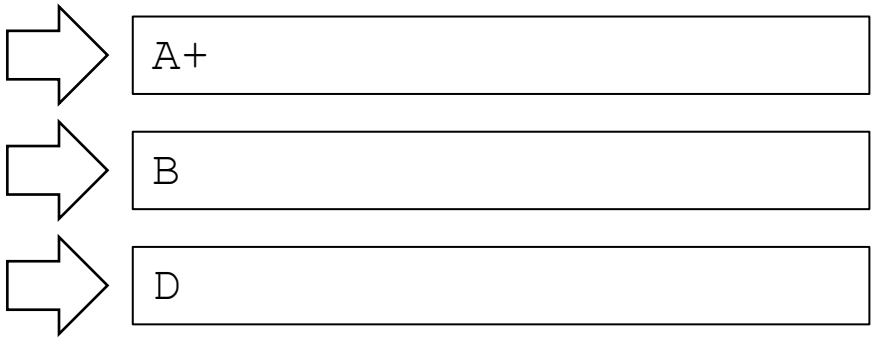
演習：コード02を書いてみましょう

5人の成績の値に基づいて評価します

```
scores = [99, 75, 59, 85, 60]

s = 59
  if s < 60:
    print("D")
  elif 60 <= s <70 :
    print("C")
  elif 70 <= s <80:
    print("B")
  elif 80 <= s <90 :
    print("A")
  else:
    print("A+")
```

○ 実行されます

以下、実行されません



for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

演習：コード02を書いてみましょう

5人の成績の値に基づいて評価します

```
scores = [99, 75, 59, 85, 60]

s = 85
  if s < 60:    ×
    print("D")
  elif 60 <= s <70 :×
    print("C")
  elif 70 <= s <80: ×
    print("B")
  elif 80 <= s <90 :○
    print("A")
  else:
    print("A+")
```

実行されます

以下、実行されません

A+

B

D

A

for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

演習：コード02を書いてみましょう

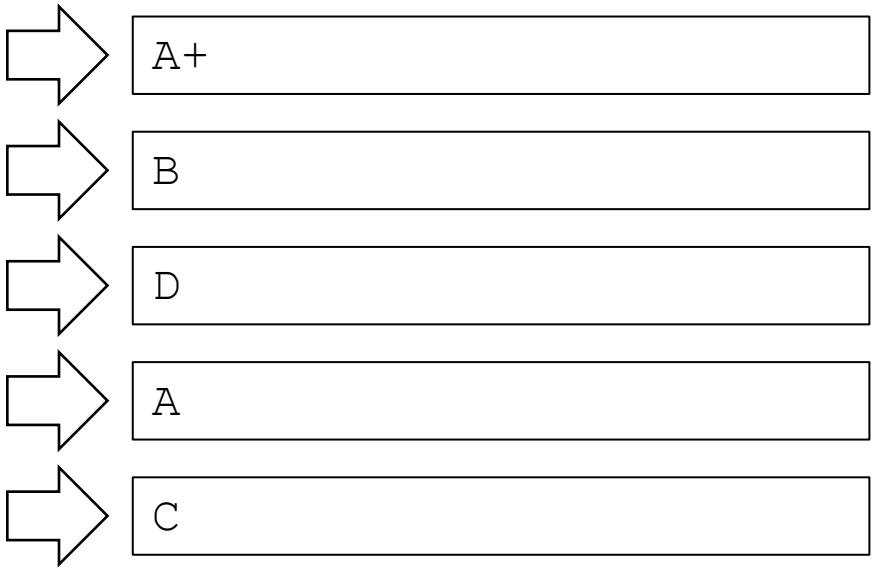
5人の成績の値に基づいて評価します

```
scores = [99, 75, 59, 85, 60]

s = 60
  if s < 60:
    print("D")
  elif 60 <= s <70 :
    print("C")
  elif 70 <= s <80:
    print("B")
  elif 80 <= s <90 :
    print("A")
  else:
    print("A+")
```

×

○ 実行されます

以下、実行されません



for　日本語での**for文**に相当する部分は「**～のために**繰り返す」や「**～ごとに**繰り返す」などのように言えます。

演習：コード02を書いてみましょう

5人の成績の値に基づいて評価します

```
scores = [99, 75, 59, 85, 60]

for s in scores:
  if s < 60:
    print("D")
  elif 60 <= s <70 :
    print("C")
  elif 70 <= s <80:
    print("B")
  elif 80 <= s <90 :
    print("A")
  else:
    print("A+")
```

繰り返す回数は、リストの要素の数で決まります。
要素を全て代入した後、繰り返しが終わります。

```
A+
B
D
A
C
```

Colab

関数

特定の処理を実行するためのコードです。

関数　特定の処理を実行するためのブロックをまとめたものです。

タスク：四則演算

$x = a + 4*b + 5*c$

関数　特定の処理を実行するためのブロックをまとめたものです。

タスク：四則演算

$x = a + 4*b + 5*c$

a = 1 b = 2 c = 3	➡	x = 1 + 4*2 +5*3	➡	24
a = 4 b = 5 c = 6	➡	x = 4 + 4*5 +5*6	➡	54
…	➡		➡	…

関数　特定の処理を実行するためのブロックをまとめたものです。

処理を関数のコードに変換しました。

関数

```
def q(a, b, c):

    x = a + 4*b +5*c

    return x
```

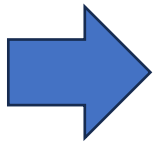
関数　特定の処理を実行するためのブロックをまとめたものです。

処理を関数のコードに変換しました。

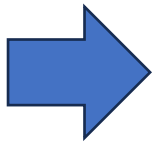
関数

```
def q(a, b, c):

    x = a + 4*b +5*c

    return x
```

```
a = 156
b = 243
c = 399
```



```
x = q(156, 243, 399)
```



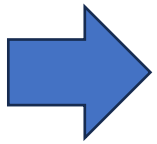
関数　特定の処理を実行するためのブロックをまとめたものです。

処理を関数のコードに変換しました。

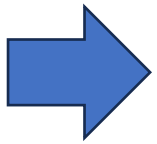
関数

```
def q(a, b, c):

    x = a + 4*b +5*c

    return x
```

a = 156
b = 243
c = 399



`x = q(156, 243, 399)`



3123

関数　特定の処理を実行するためのブロックをまとめたものです。

関数の中身を見てみます。

```
def    q(a, b, c):
    x = a + 4*b +5*c


    return x
```

キーワード　**関数名**　**引数**：関数にデータを渡すための変数

関数　特定の処理を実行するためのブロックをまとめたものです。

関数の中身を見てみます。

```
def q(a, b, c):
    x = a + 4*b +5*c

    return x
```

キーワード

関数名

引数：関数にデータを渡すための変数

関数の本体：関数の処理を記述します

関数　特定の処理を実行するためのブロックをまとめたものです。

コードの中身を見てみます。

```
def q(a, b, c):
    x = a + 4*b +5*c

    return x
```

キーワード

関数名

引数：関数にデータを渡すための変数

関数の本体：関数の処理を記述します

戻り値：関数が値を返す場合、return文を使用してその値を指定します

関数　特定の処理を実行するためのブロックをまとめたものです。

```
def 関数名(引数1, 引数2, ...):
    処理1
    処理2
    return 戻り値
```

インデント：
プログラムの構造を示すために使用される空白文字
処理1、処理2およびreturnは、関数の一部内容として認識されています。

関数　特定の処理を実行するためのブロックをまとめたものです。

```
def 関数名(引数1, 引数2, ...):

    処理1

    return 戻り値
```

```
def q(a, b, c):

    x = a + 4*b +5*c

    return x
```

関数qを作成しました。

関数qを**呼び出します**。

```
x = q(1, 2, 3)
print(x)
```

Colab

関数　**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを計算する関数を書きましょう**

BMI（Body Mass Index、ボディマス指数）は、体重と身長を基にして身体の肥満度を評価するための指数です。体重（kg）を身長（m）の2乗（身長×身長）で割った数値がBMI指数となります。

BMIは以下の数式で計算されます：
BMI = 体重（kg） / (身長（m） * 身長（m）)

関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを計算する関数を書きましょう**

bmi = weight / (height** 2)　処理方法：計算方法

引数a　weight = ____**m**

引数b　height = ____**Kg**

関数

** ：プログラミング言語では累乗（べき乗）の演算子です。
height 2**：身長の2乗の意味です。

関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを計算する関数を書きましょう**

bmi = weight / (height** 2)　処理方法：計算方法

引数a　weight = ____m

引数b　height = ____Kg

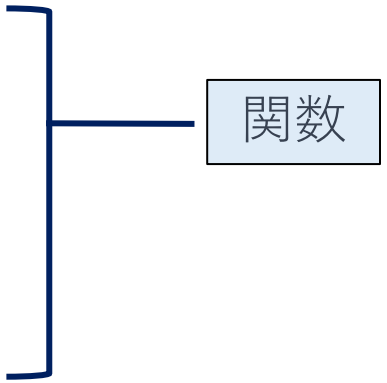
```
def calculate_bmi( 引数a 引数b ):
    処理方法：計算方法
    return
```

関数

関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを計算する関数を書きましょう**

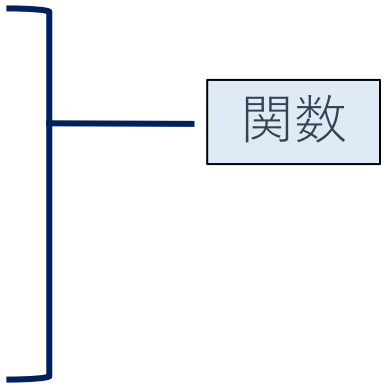
```
def calculate_bmi(  weight,  height  ):

    

    return 
```



関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを計算する関数を書きましょう**

```
def calculate_bmi(    weight,    height    ):

    bmi = weight / (height** 2)

    return bmi
```



関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを計算する関数を書きましょう**

```
def calculate_bmi(    weight,    height    ):
    bmi = weight / (height** 2)
    return bmi
```

関数calculate_bmiを作成しました。

関数calculate_bmiを呼び出します。

weight = 70
height = 1.75

関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを計算する関数を書きましょう**

```
def calculate_bmi(    weight,    height    ):
    bmi = weight / (height** 2)
    return bmi



mybmi = calculate_bmi(70, 1.75)
print(mybmi)
```

関数calculate_bmiを作成しました。

関数calculate_bmiを呼び出します。

weight = 70
height = 1.75

関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを計算する関数を書きましょう**

```
def calculate_bmi(    weight,    height    ):
    bmi = weight / (height** 2)
    return bmi



mybmi = calculate_bmi(70, 1.75)
print(mybmi)

22.86
```

関数**calculate_bmi**を作成しました。

関数**calculate_bmi**を呼び出します。

weight = **70**
height = **1.75**

Colab

関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード04を書いてみましょう

関数　**BMIの値に基づいて肥満度を評価します**

BMIの値を以下のように解釈することが一般的です：

BMI < 18.5　　: 低体重（Underweight）

18.5 ≤ BMI < 24.9: 正常体重（Normal weight）

25 ≤ BMI < 29.9　: 軽度の肥満（Overweight）

BMI ≥ 30　　: 肥満（Obesity）

引数：bmi

処理方法：if文

関数

関数　関数は、**特定のタスクや処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを評価します**

```
def bmi_category(bmi):
    処理方法：if、elif、else文
    return BMIの評価
```

（引数：bmi）

関数

関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを評価します**

```
def bmi_category(bmi):

    if bmi < 18.5:

    elif 18.5 <= bmi < 25:

    elif 25 <= bmi < 30:

    else:
```

引数：bmi

処理方法：if条件式

関数

関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード03を書いてみましょう　**BMIを評価します**

```
def bmi_category(bmi):

    if bmi < 18.5:
        return "低体重"

    elif 18.5 <= bmi < 25:
        return "正常体重"

    elif 25 <= bmi < 30:
        return "軽度の肥満"

    else:
        return "肥満"
```

if文
return
if文
return
if文
return
if文
return

関数

関数　関数は、**特定の処理を実行する**ためのブロックをまとめたものです。

演習：コード04を書いてみましょう　**BMIの値に基づいて肥満度を評価します**

```
def bmi_category(bmi):
    if bmi < 18.5:
        return "低体重"
    elif 18.5 <= bmi < 25:
        return "正常体重"
    elif 25 <= bmi < 30:
        return "軽度の肥満"
    else:
        return "肥満"
```

関数**bmi_category**を作成しました。

```
mybmi = bmi_category(20.5)
print(mybmi)
```

関数**bmi_category**を呼び出します。

```
正常体重
```

Colab

課題：WebClass上に回答を入力してください

課題1
下記挨拶文のリストを用意しました。
greeting = ["Good morning", "Good afternoon", "Good evening", "Good night"]
for文を使用して、順番に言葉を表示するようにコードを書いてください。
繰り返し変数はgとします。

課題2
if、elif、else文を使用して気温によって快適さを判断するコードを書いてください。
気温の変数は、tとします。現在の気温は、15度とします。
0度以下：寒すぎます
0-10度：寒いです
10-25度：快適です
25-35度：熱いです。
35度以上：暑すぎます。

演習2-7の構成

Python基礎を学びましょう

演習2　11/16 11:35-12:20

Pythonの変数とデータの型

演習3　11/30 11:35-12:20

プログラミング基礎

演習4　12/7 10:40-11:25

モジュール、パッケージ、ライブラリ

Pythonを使ってみましょう

演習5　12/7 11:35-12:20

患者の歯に関する病院のリアルデータの説明

演習6

データクレンジングに必要なライブラリ（Pandas）の応用

演習7

データクレンジングとデータの可視化