

Google Colaboratory でのプログラミング

2024年 A1 木曜2限

担当：地引

TA：董/石井

Google Colaboratory の基本操作

最近の情報通信環境を取り巻く状況を考慮し、ここではクラウド環境上で提供される処理系の利用について説明します。

教育用計算機や個人のPCにインストールした処理系を利用してもよいです。

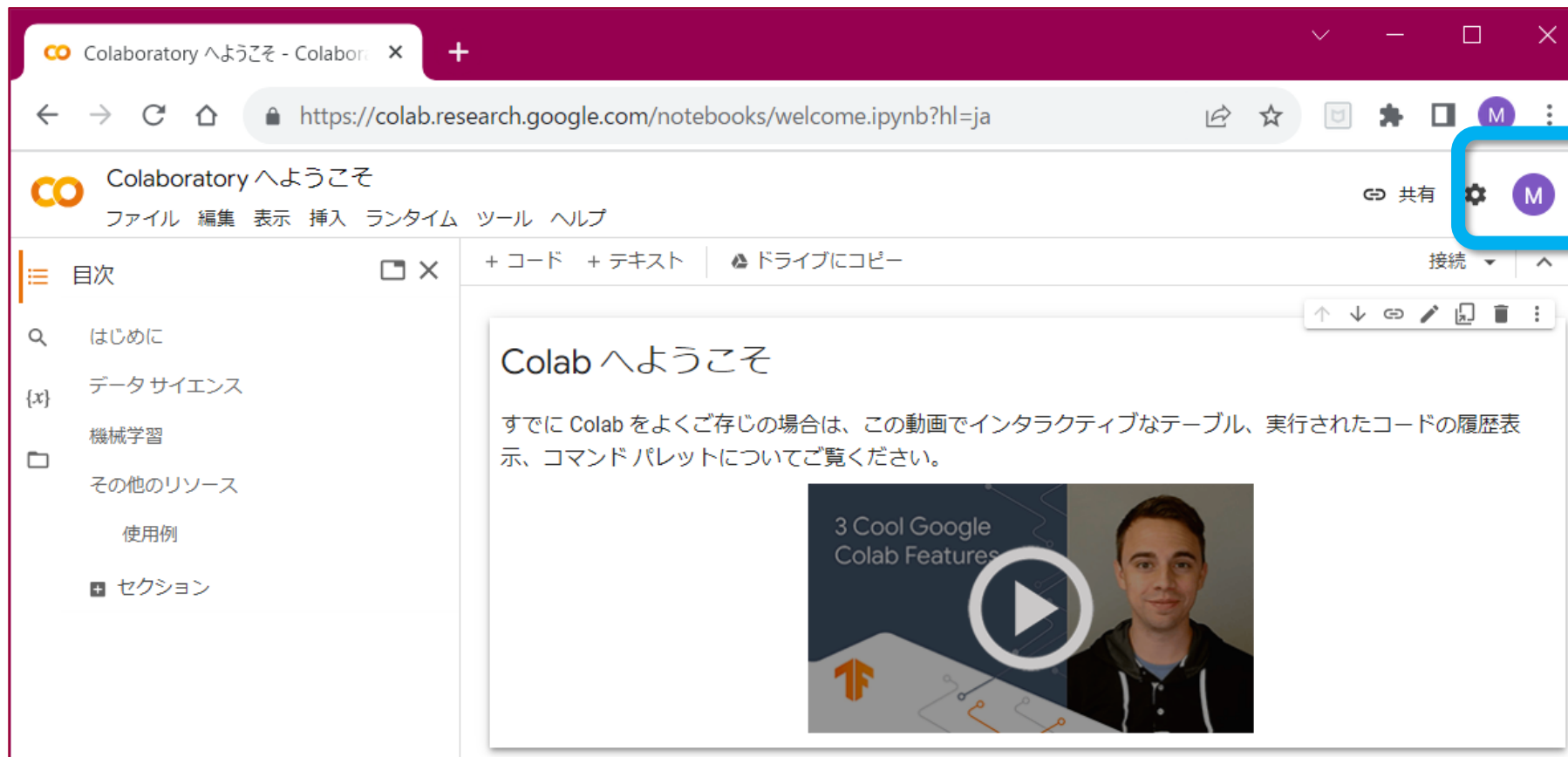
Google Colaboratory へのログイン (1)

- Google Colaboratory は、下記の URL からログインできます。
 - <https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja>
 - 右端上に表示される **ログイン** をクリックし、ECCS クラウド メール のアカウント「????@g.ecc.u-tokyo.ac.jp」でログインして下さい。



Google Colaboratory へのログイン (2)

ログイン後は、以下のようなウィンドウが表示されます (変化に気付きにくい…)。



確認

Python プログラム ファイルの操作

The screenshot shows the Google Colaboratory interface. The 'File' menu is open, displaying various options. A pink box highlights the top three items: 'ノートブックを新規作成', 'ノートブックを開く', and 'ノートブックをアップロード'. A blue box highlights the bottom two items: '.ipynb をダウンロード' and '.py をダウンロード'. A pink callout box points to the top three items, and a blue callout box points to the bottom two items.

まずは [ファイル] タブをクリックします。
この辺りのファイル操作メニューは意味が分かるかと思いますが
（“ノートブックを開く” については、後ほど補足します）。

手元の（個人所有のPC上にある）Jupyter Notebook を
利用したい人は、この辺りのファイル操作メニューを使います。

Jupyter Notebook(Google Colaboratory)の基本的な操作方法

- Returnキーを押す：入力ボックス内で改行
- Shift + Return：そのマスを実行しつ次のマスへ進む
- Ctrl + Return：そのマスを実行する
- 知っているとちょっと便利な機能

Python プログラムの記述・実行

The screenshot shows a Google Colab notebook titled "Untitled0.ipynb". The interface includes a top navigation bar with "ファイル" (File), "編集" (Edit), "表示" (View), "挿入" (Insert), "ランタイム" (Runtime), "ツール" (Tools), and "ヘルプ" (Help). A sidebar on the left shows a table of contents with "目次" (Table of Contents) and "セクション" (Section). The main area contains a code cell with the code `3+5` and its output `8`. A red box highlights the "+ コード" button, a green box highlights the play button, and a blue box highlights the output area. Arrows point from these elements to explanatory text boxes.

ファイル名を変更したい場合は、[ファイル] タブ以下にメニューがあります。

プログラムの記述と実行は、次の手順で行ないます。

- ① “+ コード” をクリックすると、プログラムを記述するセルが用意されます。
- ② このボタンを押すことで、プログラムが実行されます。
- ③ コード セルの下に結果が表示されます。

プログラムをさらに記述したい場合は、下記のどちらの方法でも可能です。

- 既存のコードセルに追記する。
- 再度 “+ ファイル” を実行し、新たに用意したコードセルに記述する。

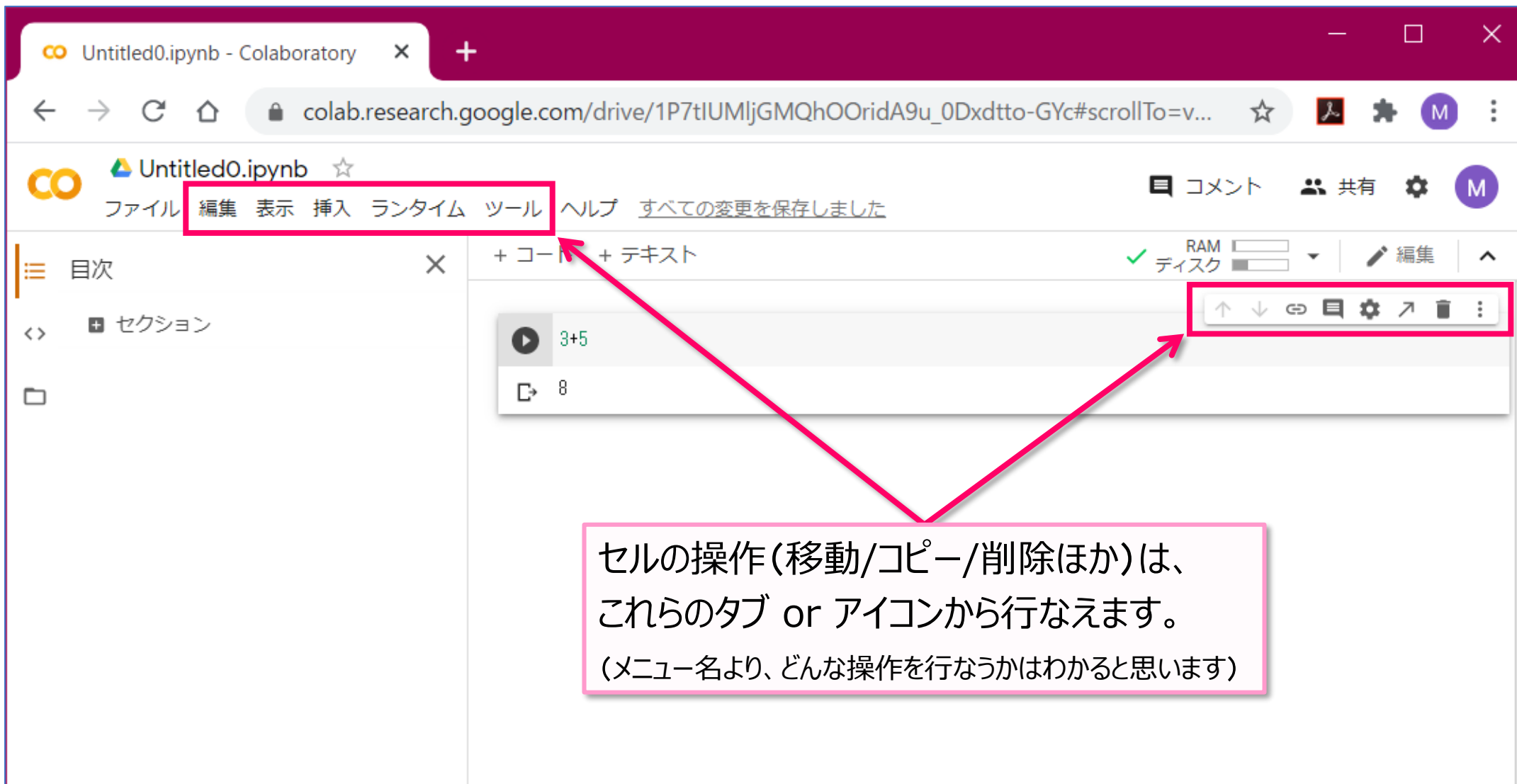
プログラムのコピー&ペースト

The screenshot shows the Jupyter Notebook interface for a file named 'Untitled0.ipynb'. The 'Edit' menu is open, and the '貼り付け' (Paste) option is highlighted. A red box highlights the 'Edit' menu item. A red arrow labeled '1' points to the 'Edit' menu, and another red arrow labeled '2' points to the '貼り付け' option. The interface also shows a file explorer on the left, a toolbar with 'RAM' and 'ディスク' indicators, and a 'コメント' (Comment) button.

作成するプログラムの種類によっては、お手本となるプログラムをコピーして、それを修正した方が効率的な場合があります。セルに対するプログラムのコピーは、以下の手順で行ないます。

- ① 手元の PC 上などにあるプログラムをマウス操作でコピー
- ① コピー先となるセルを選択
- ② “編集” → “貼り付け” を選択
- ③ もし、「Ctrl-v を押せ」との表示が一瞬出たら、従う。経験的には、思うように動かないことが多い気がします。。

セルの操作



The screenshot shows the Google Colaboratory interface. At the top, there is a browser window with the address bar showing 'colab.research.google.com'. Below the browser, the Colaboratory header includes 'Untitled0.ipynb' and a menu with options like 'ファイル', '編集', '表示', '挿入', 'ランタイム', 'ツール', and 'ヘルプ'. The main workspace contains a code cell with the text '3+5' and its output '8'. A red box highlights the menu options '編集', '表示', and '挿入'. Another red box highlights the cell toolbar icons: up arrow, down arrow, link, comment, settings, refresh, and delete. Two red arrows point from the text box below to these two red boxes.

セルの操作(移動/コピー/削除ほか)は、これらのタブ or アイコンから行なえます。(メニュー名より、どんな操作を行なうかはわかると思います)

注意：Python プログラムの実行結果（1）

- 教材「Python によるプログラミングの初歩的な解説」でも説明していますが、Python プログラムの実行結果は、基本的に次の方針に沿って表示されます。
 1. print 文などにより明示的に表示を指定された場合は、それを表示する。
 2. コードセルにある一番最後のコードが結果を表示できる場合は、それを表示する（それより前にあるコードは、上記 1 以外表示されません）。
 - › 計算式やクォーテーションで囲った文字列は、上記 2 に該当します。
 - › 呼び出した関数が値を返す場合は、上記 2 に該当します。
 - › 関数定義は、上記 2 に該当しません。
- Google Colaboratory での実行例を見てみましょう。

注意：Python プログラムの実行結果（2）

The screenshot shows a Google Colaboratory notebook with three code cells. The first cell contains arithmetic operations, the second defines a function, and the third calls the function and prints a message. Callouts explain that only the final result of multiple calculations is shown, function definitions are not displayed, and print statements are shown regardless of their position in the code.

複数の計算式がありますが、一番最後の計算結果だけが表示されます。

関数の定義では、何も表示されません。

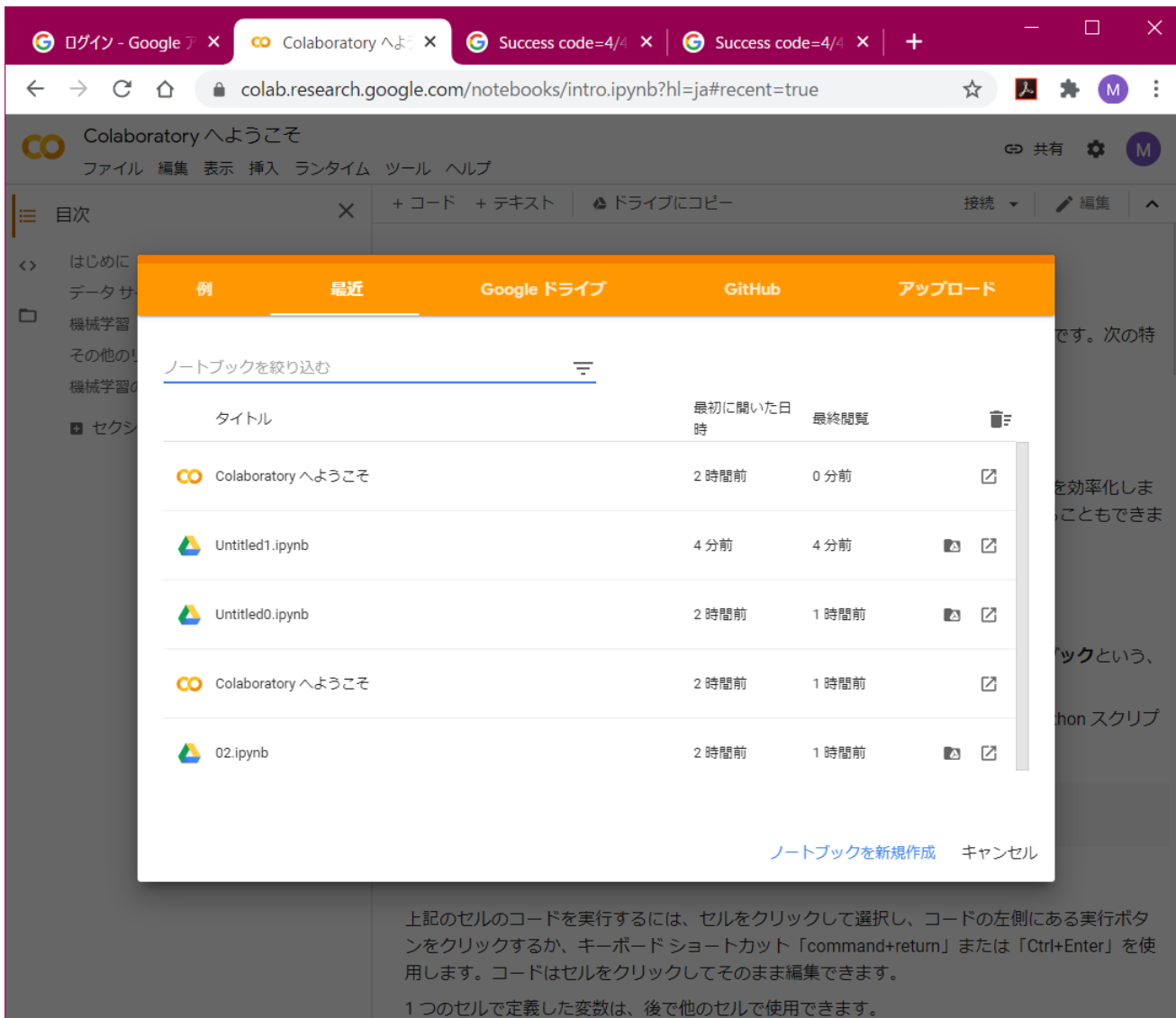
print文は、コードの最後でなくても表示されます。また、複数の関数呼び出しがありますが、コードの最後に位置するものだけが結果を表示されます。

教材「Pythonによるプログラミングの初歩的な解説」にも目を通して置いて下さい。

ノートブックを開く

Google Colaboratory や Jupyter Notebook では、プログラム ファイルをノートブックと呼びます。

ノートブックを開く（1）



The screenshot shows the Google Colaboratory interface with a file selection dialog open. The dialog has tabs for '例', '最近', 'Google ドライブ', 'GitHub', and 'アップロード'. The '最近' tab is selected, showing a list of notebooks. The list has columns for 'タイトル', '最初に開いた日時', and '最終閲覧'. The top notebook is 'Colaboratory へようこそ', opened 2 hours ago and viewed 0 minutes ago. Below it are 'Untitled1.ipynb' (opened 4 minutes ago, viewed 4 minutes ago), 'Untitled0.ipynb' (opened 2 hours ago, viewed 1 hour ago), another 'Colaboratory へようこそ' (opened 2 hours ago, viewed 1 hour ago), and '02.ipynb' (opened 2 hours ago, viewed 1 hour ago). At the bottom of the dialog are links for 'ノートブックを新規作成' and 'キャンセル'.

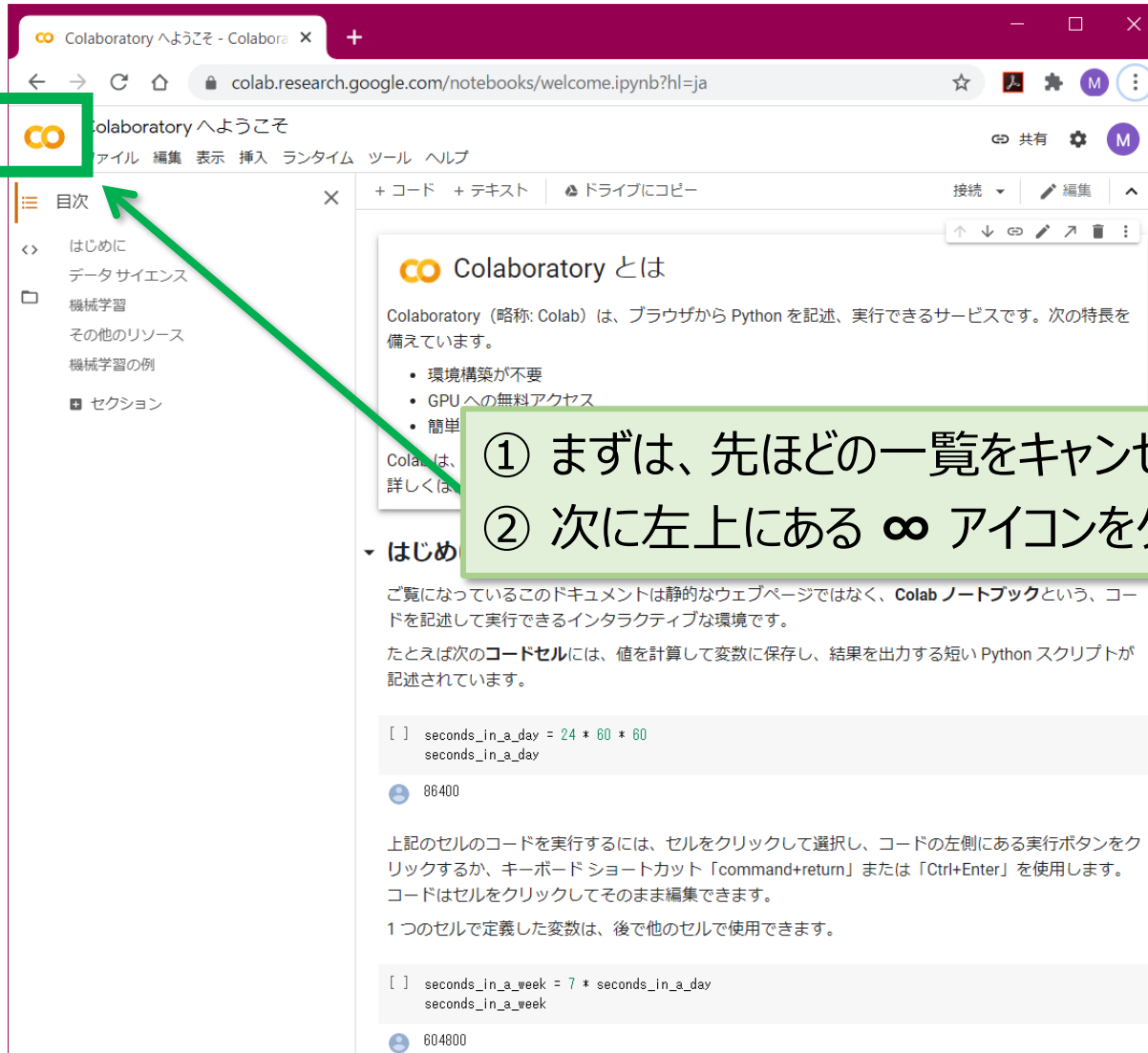
例	最近	Google ドライブ	GitHub	アップロード
ノートブックを絞り込む				
タイトル		最初に開いた日時	最終閲覧	
Colaboratory へようこそ		2 時間前	0 分前	
Untitled1.ipynb		4 分前	4 分前	
Untitled0.ipynb		2 時間前	1 時間前	
Colaboratory へようこそ		2 時間前	1 時間前	
02.ipynb		2 時間前	1 時間前	

Google Coraboratory にログインする or [ファイル] → [ノートブックを開く] を選択すると、左のようなファイル一覧が表示されます。

しかし、これは最近開いたファイルの一覧であり、自分が開きたいファイルはこの中にもないかも知れません(履歴を消すなど)。

そのような場合は、これから説明する方法でファイルを探して下さい。

ノートブックを開く (2)



Colaboratory へようこそ

Colaboratory とは

Colaboratory (略称: Colab) は、ブラウザから Python を記述、実行できるサービスです。次の特長を備えています。

- 環境構築が不要
- GPU への無料アクセス
- 簡単

Colab は、詳しくは...

はじめ

ご覧になっているこのドキュメントは静的なウェブページではなく、Colab ノートブックという、コードを記述して実行できるインタラクティブな環境です。

たとえば次のコードセルには、値を計算して変数に保存し、結果を出力する短い Python スクリプトが記述されています。

```
[ ] seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

86400

上記のセルのコードを実行するには、セルをクリックして選択し、コードの左側にある実行ボタンをクリックするか、キーボードショートカット「command+return」または「Ctrl+Enter」を使用します。コードはセルをクリックしてそのまま編集できます。

1つのセルで定義した変数は、後で他のセルで使用できます。

```
[ ] seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

604800

- ① まずは、先ほどの一覧をキャンセルし、適当なファイルを開きます。
- ② 次に左上にある ∞ アイコンをクリックします。

ノートブックを開く (3)

The screenshot shows the Google Drive web interface. The left sidebar contains navigation options: 新規 (New), 候補 (Suggestions), マイドライブ (My Drive), 共有ドライブ (Shared Drives), 共有アイテム (Shared Items), 最近使用したアイテム (Recently Used Items), スター付き (Starred), ゴミ箱 (Trash), and 保存容量 (Storage). The main area displays search results for notebooks, showing two files: 'Untitled0.ipynb' and 'Untitled1.ipynb'. The 'Untitled0.ipynb' file is highlighted with a blue box, and a blue arrow points from a text box to the 'ドライブ' (Drive) icon in the top left. A green box at the bottom contains the instruction: ③ ノートブックの一覧が表示されるので、該当するファイルをWクリックします。 (Since the notebook list is displayed, double-click the corresponding file.)

Google Colaboratory では、作成したノートブックを自動的に Google Drive へ保存します。これを見ると、Google Drive へ移ったことが分かりますね。

③ ノートブックの一覧が表示されるので、該当するファイルをWクリックします。

ノートブックを開く（４）

検索結果 - Google ドライブ

Google ドライブ: ログイン

drive.google.com/drive/search?q=owner:me%20(type:application/vnd.google.colab...

Untitled0.ipynb Google Colaboratory で開く

新規

候補

マイドライブ

共有ドライブ

共有アイテム

最近使用したアイテム

スター付き

ゴミ箱

保存容量

22.5 KB 使用

```
3+5
2+5

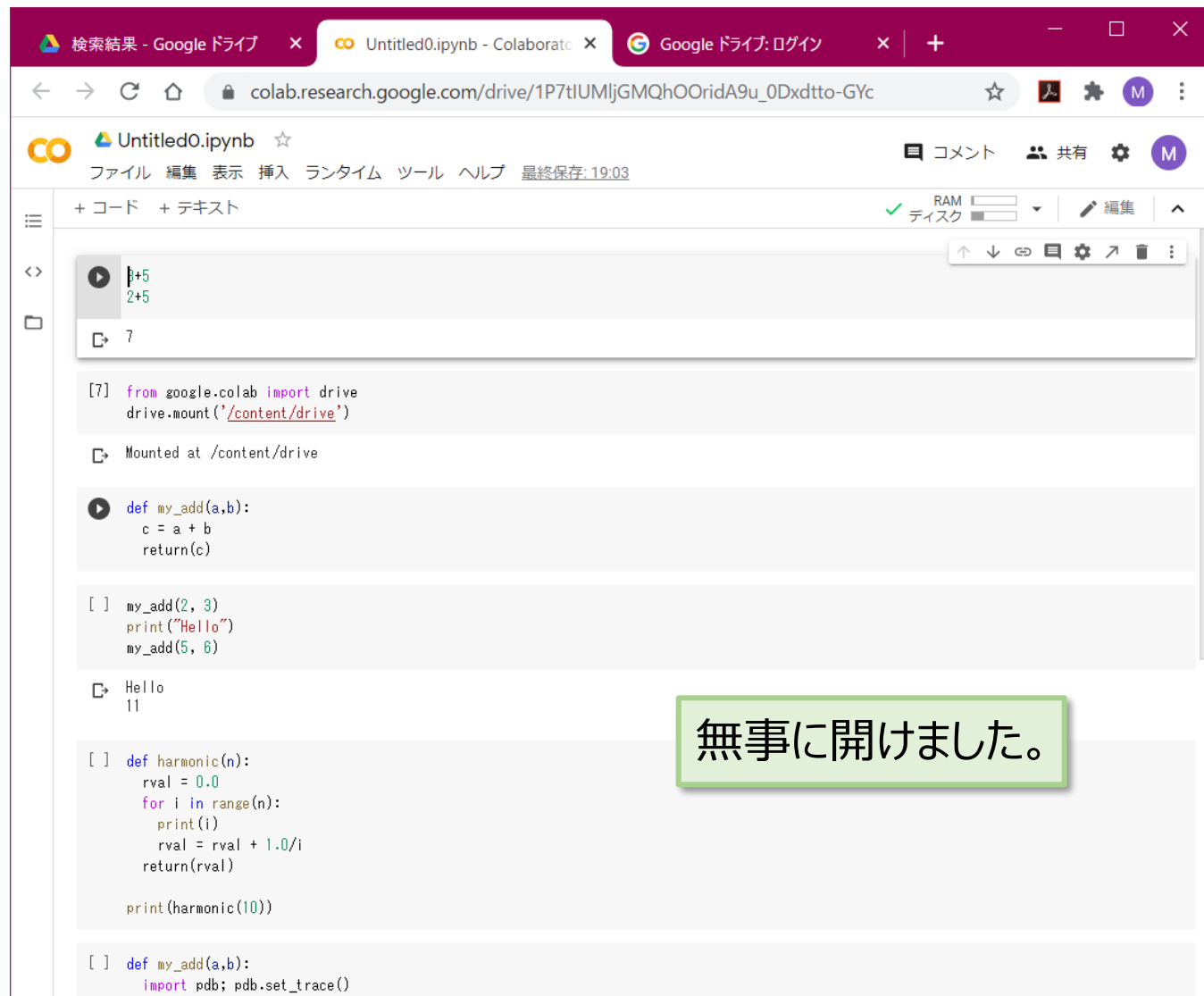
def my_add(a,b):
    c = a + b
    return(c)
import pdb;pdb.set_trace()
print("debug 1")
def my_test(a,b):
    import pdb;pdb.set_trace()
    print("debug 2")
    c = my_add(a,b)
    return(c)
my_test(2,3)

my_add(2, 3)
print("Hello")
my_add(5, 6)
```

この画面が表示されない場合もあります。

④ “Google Colaboratory” で開くを選択(クリック)します。

ノートブックを開く (5)



The screenshot shows a Google Colab notebook titled "Untitled0.ipynb". The interface includes a browser window with the URL `colab.research.google.com/drive/1P7tIUmljGMQhOOridA9u_0Dxdtto-GYc`. The notebook content is as follows:

```
+ コード + テキスト
▶ 1+5
  2+5
↳ 7

[7] from google.colab import drive
     drive.mount('/content/drive')
↳ Mounted at /content/drive

▶ def my_add(a,b):
    c = a + b
    return(c)

[ ] my_add(2, 3)
     print("Hello")
     my_add(5, 6)
↳ Hello
  11

[ ] def harmonic(n):
     rval = 0.0
     for i in range(n):
         print(i)
         rval = rval + 1.0/i
     return(rval)

     print(harmonic(10))

[ ] def my_add(a,b):
     import pdb; pdb.set_trace()
```

A green callout box with the text "無事に開けました。" (Successfully opened.) is overlaid on the right side of the notebook interface.

重要!! (1)

とはいえ、ここに一覧されるノートブックは、Google Drive 内にある全ノートブックの検索結果です。当然、ノートブックが増えて来ると、ズラズラと一覧され、お目当てのノートブックを探すのも大変です。

The screenshot shows the Google Drive search results page. The search bar contains the query: `owner:me type:application/vnd.google.colaboratory || type:application/vnd.`. The search results are displayed in a table with columns for Name, Owner, Last Updated, and File Size. The results are sorted by '最終更新' (Last Updated) in descending order. The first result is 'Untitled0.ipynb' (1 KB, updated 2023/06/16). Other results include 'Model0.ipynb' (225 KB, updated 2022/10/01), 'Untitled5.ipynb' (123 KB, updated 2022/07/01), 'Untitled2.ipynb' (123 KB, updated 2022/06/30), 'Untitled3.ipynb' (52 KB, updated 2021/07/01), and 'Untitled1.ipynb' (2 KB, updated 2021/03/26). A green box highlights the search results table, and a green arrow points from the text above to the search bar and the table.

名前	オーナー	最終更新	ファイルサイ
Untitled0.ipynb	自分	2023/06/16	1 KB
Model0.ipynb	自分	2022/10/01	225 KB
Untitled5.ipynb	自分	2022/07/01	123 KB
Untitled2.ipynb	自分	2022/06/30	123 KB
Untitled3.ipynb	自分	2021/07/01	52 KB
Untitled1.ipynb	自分	2021/03/26	2 KB

重要!! (2)

保存先は Google Drive なので、別途 Google Drive にログインして、“Colab Notebooks” フォルダ以下にサブ フォルダを作成し、適宜ノートブックを分類して整理しましょう。

Colab からノートブックを開く時は、前スライドにある検索結果一覧ではなく、左側に表示されるファイル システムを辿ってお目当てのノートブックを指定できるようになりましょう (重要なスキルです)。

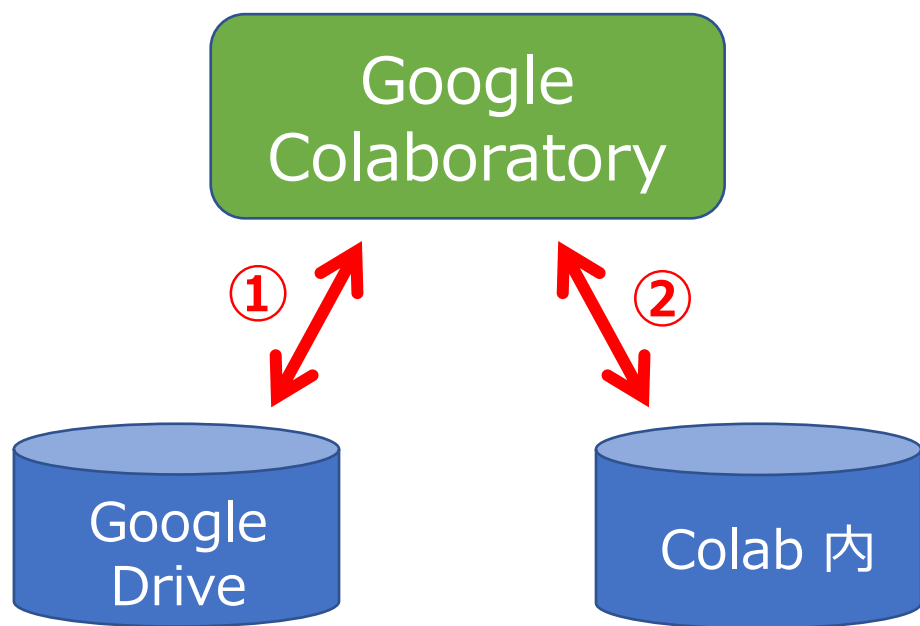
Colab のメニューは、Google Drive をブラウザで開いた時と同じなので、操作も難しくないはずです。



データ ファイルの扱い

Google Colaboratory のファイル管理イメージ

Google Colaboratory では、扱うファイルを下記①,②のどちらかに保存します。プログラム ファイルは、特に準備をしなくても①に保存されますが、データ用のファイルを保存する場合は、どちらに保存しても事前の準備や保存後の扱いが少々面倒です。



①にあるデータ ファイル(プログラム ファイルではない)を Colab からアクセスするには、Google Drive をマウント (一つのファイル システムとして接続) する必要があります。以降のスライドで説明しますが、現状ではマウント処理を簡便に行なえない場合があるという欠点があります。②の方はマウントが不要で、より直感的に扱えますが、保存したファイルは、一定時間後に削除されてしまうという欠点があります。

Google Drive のマウント (1)

- 以下では、Colab 上の Python コードがアクセスするデータ ファイルの保存先として、Google Drive を利用する手順を説明します。
- 次スライド以降の手順により、Colab に Drive をマウントすることで、Python コードから Drive 内のデータ ファイルへアクセスできます。
- 但し、Python コード内にファイルのパス名を正しく記載しないと入力/出力ができないことに注意して下さい。
 - 詳細は、“【重要】データ ファイルへのアクセス” スライドを参照
 - パス名の利用に慣れましょう → これができないと今後何もできない…

Google Drive のマウント (2)

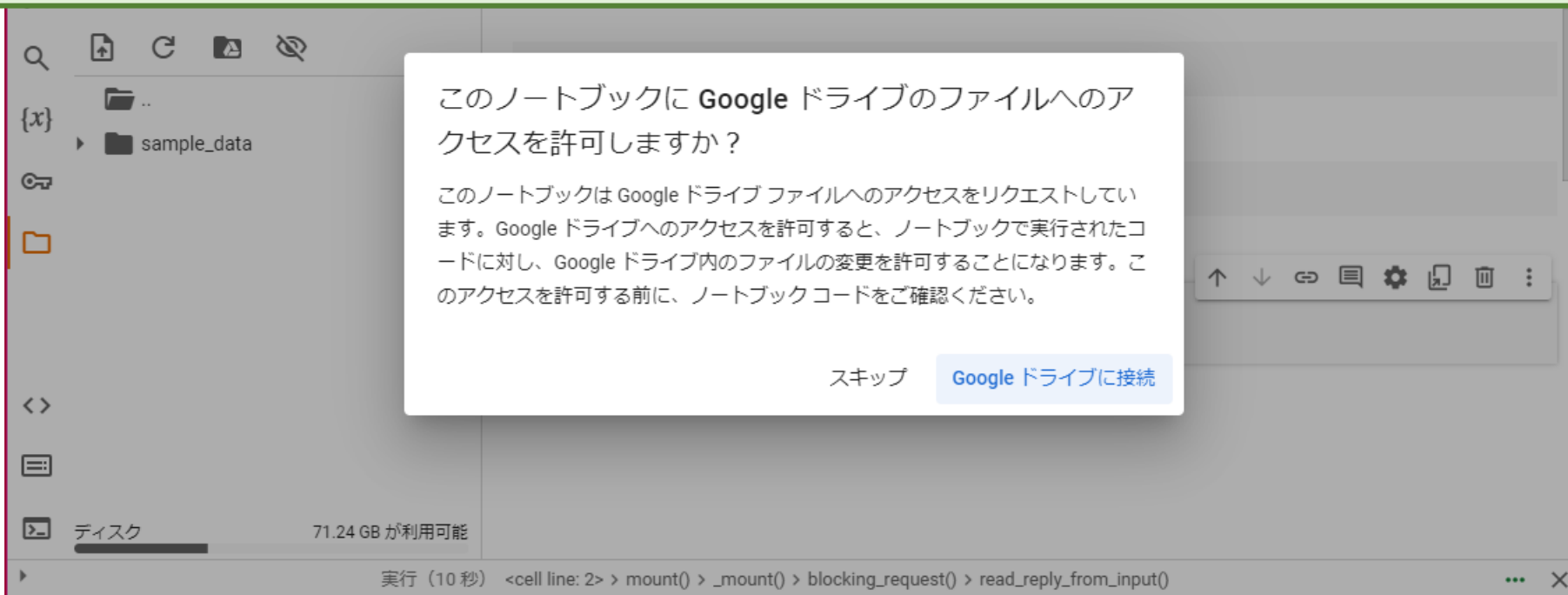
② ファイル操作のアイコンが表示されるので、マウント操作のアイコンを選択します。

① 左端にあるフォルダのようなアイコンをクリックします。

The screenshot shows a web browser window with the URL `https://colab.research.google.com/drive/1A6RbVgkRY1pKnH_Ggxzl18v9fGwU...`. The interface includes a file manager on the left with a folder icon highlighted by a green box and an arrow. The top toolbar contains various icons, with the mount icon (a folder with a plus sign) also highlighted by a green box and an arrow. A terminal window in the center shows the command `!pwd` and the output `/content`, and another terminal window below it shows `sample_data`.

Google Drive のマウント (3)

このような感じでGoogle Drive のマウント処理が自動的に始まった場合は、“Google Drive のマウント (6)” スライド以降を参照して下さい。
自動的に始まらなかった場合は次スライドへ。



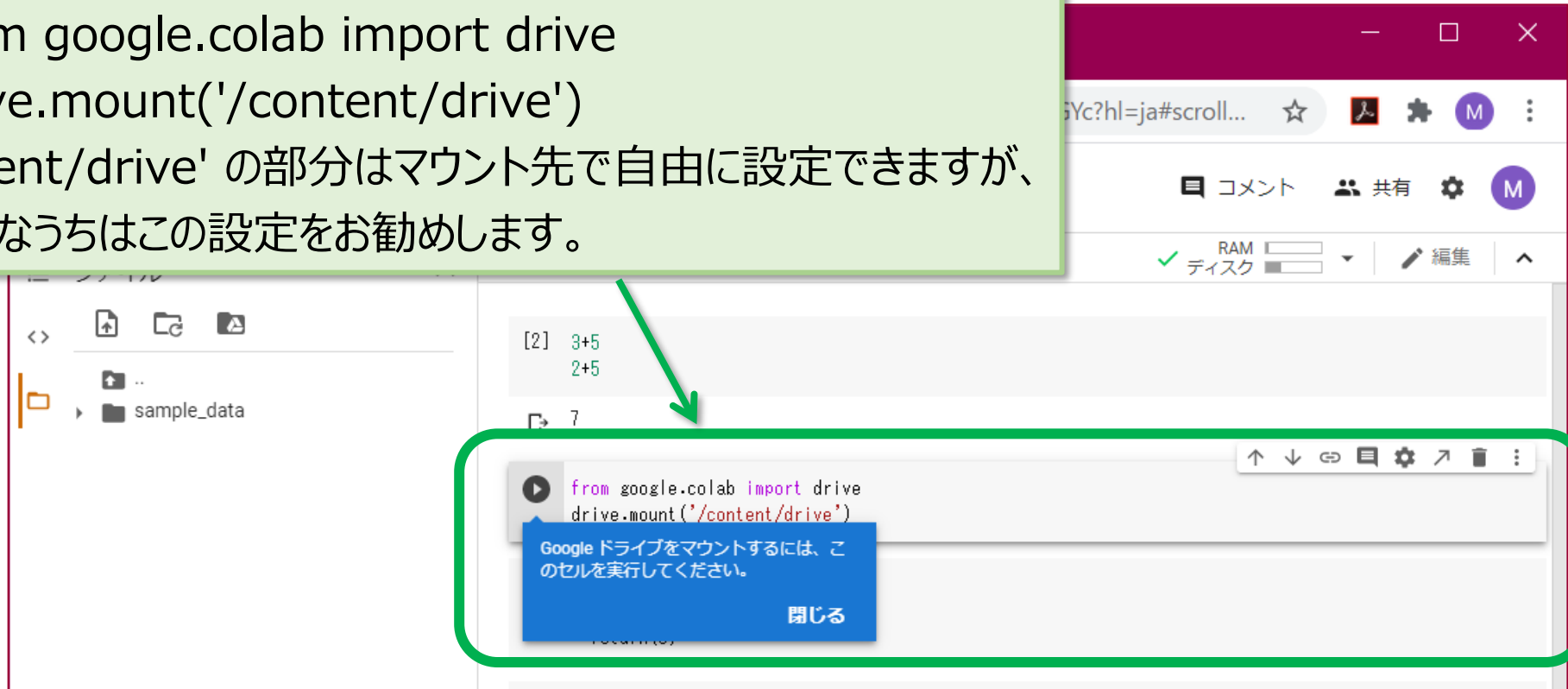
Google Drive のマウント (4)

- ③ 前スライドのようにマウント処理が自動的に開始されず、マウントに必要なプログラムの実行が表示された場合は、そのセルを実行します。

参考: マウントに必要なプログラム

```
from google.colab import drive
drive.mount('/content/drive')
```

'/content/drive' の部分はマウント先で自由に設定できますが、不慣れなうちはこの設定をお勧めします。

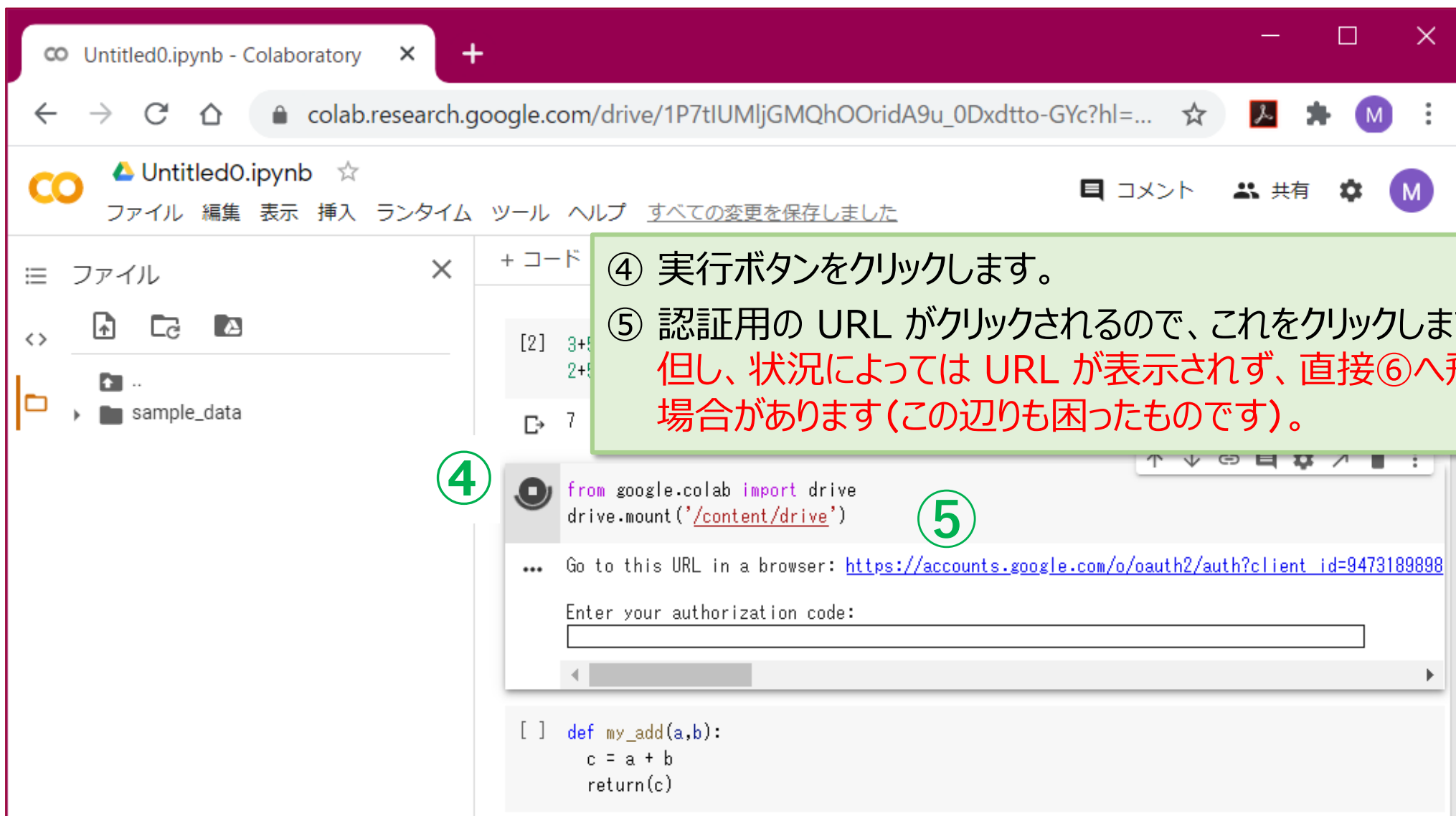


The screenshot shows a Google Colab notebook interface. On the left, there is a file explorer showing a folder named 'sample_data'. The main area displays a code cell with the following code:

```
from google.colab import drive
drive.mount('/content/drive')
```

Below the code, a blue warning message box is displayed with the text: "Google ドライブをマウントするには、このセルを実行してください。" (To mount Google Drive, please run this cell.) and a "閉じる" (Close) button. A green arrow points from the text in the slide to the code cell. The entire code cell and warning box are enclosed in a green rounded rectangle.

Google Drive のマウント (5)



④ 実行ボタンをクリックします。

⑤ 認証用の URL がクリックされるので、これをクリックします。
但し、状況によっては URL が表示されず、直接⑥へ飛ぶ場合があります(この辺りも困ったものです)。

```
from google.colab import drive
drive.mount('/content/drive')
```

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189898

Enter your authorization code:

Google Drive のマウント (6)

⑥ ECC クラウドメールのアドレスをクリックします。

Google にログイン

アカウントの選択

「Google Drive File Stream」に移動

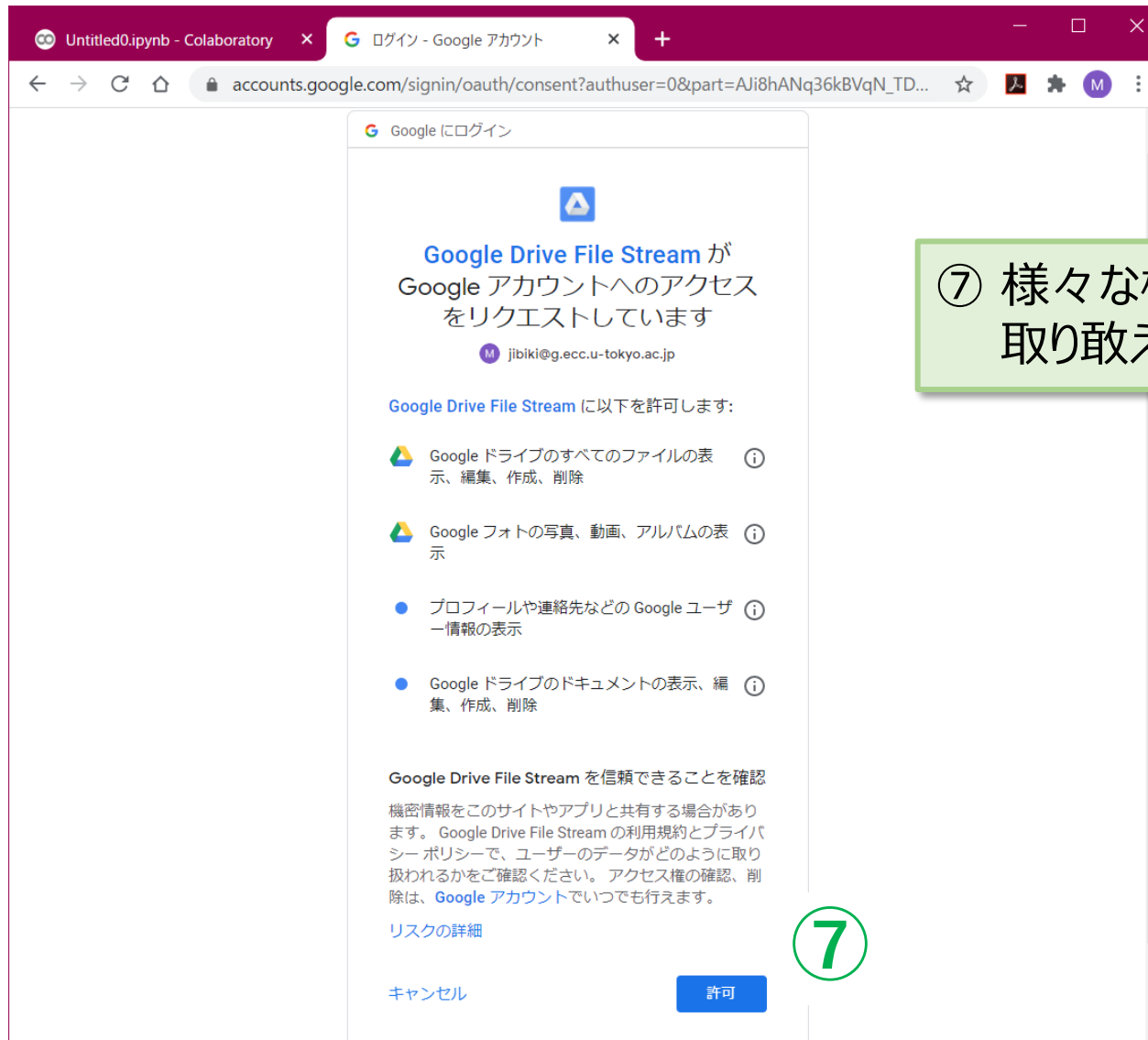
M Masahiro JIBIKI
jibiki@g.ecc.u-tokyo.ac.jp

別のアカウントを使用

続行するにあたり、Google はあなたの名前、メールアドレス、言語設定、プロフィール写真を Google Drive File Stream と共有します。

日本語 ヘルプ プライバシー 規約

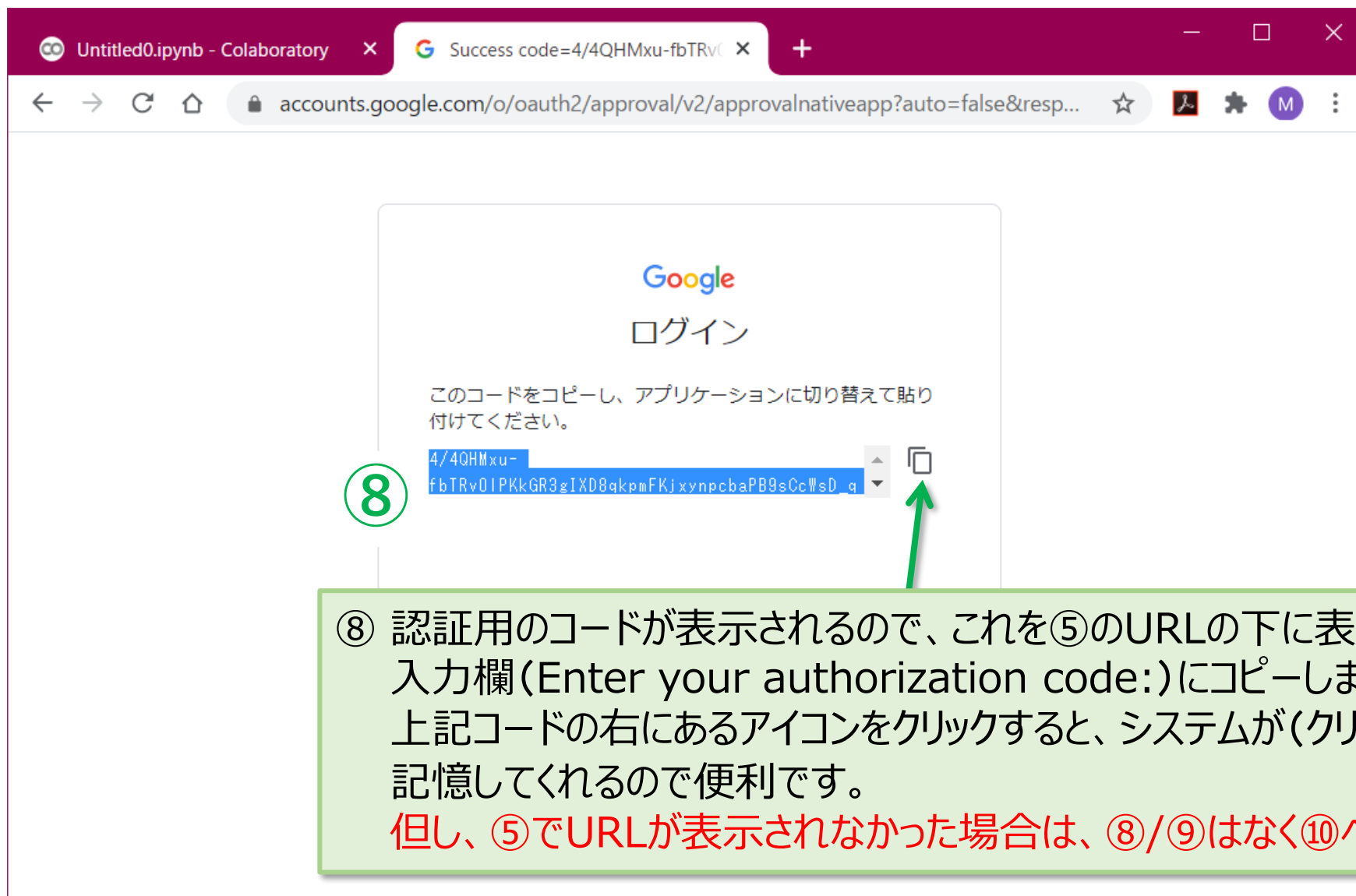
Google Drive のマウント (7)



⑦ 様々な権限を求められますが、取り敢えずこれらを全て許可しておきます。

7

Google Drive のマウント (8)



⑧ 認証用のコードが表示されるので、これを⑤のURLの下に表示された入力欄(Enter your authorization code:)にコピーします。上記コードの右にあるアイコンをクリックすると、システムが(クリップボードに)記憶してくれるので便利です。
但し、⑤でURLが表示されなかった場合は、⑧/⑨はなく⑩へ飛びます。

Google Drive のマウント (9)

⑨ Google Coraboratory のページへ戻り(タブを切り替え)、
⑧でコピーした認証用コードを入力します(入力後は Enter キーを押す)。
但し、⑤でURLが表示されなかった場合は、⑨はなく⑩へ飛びます。

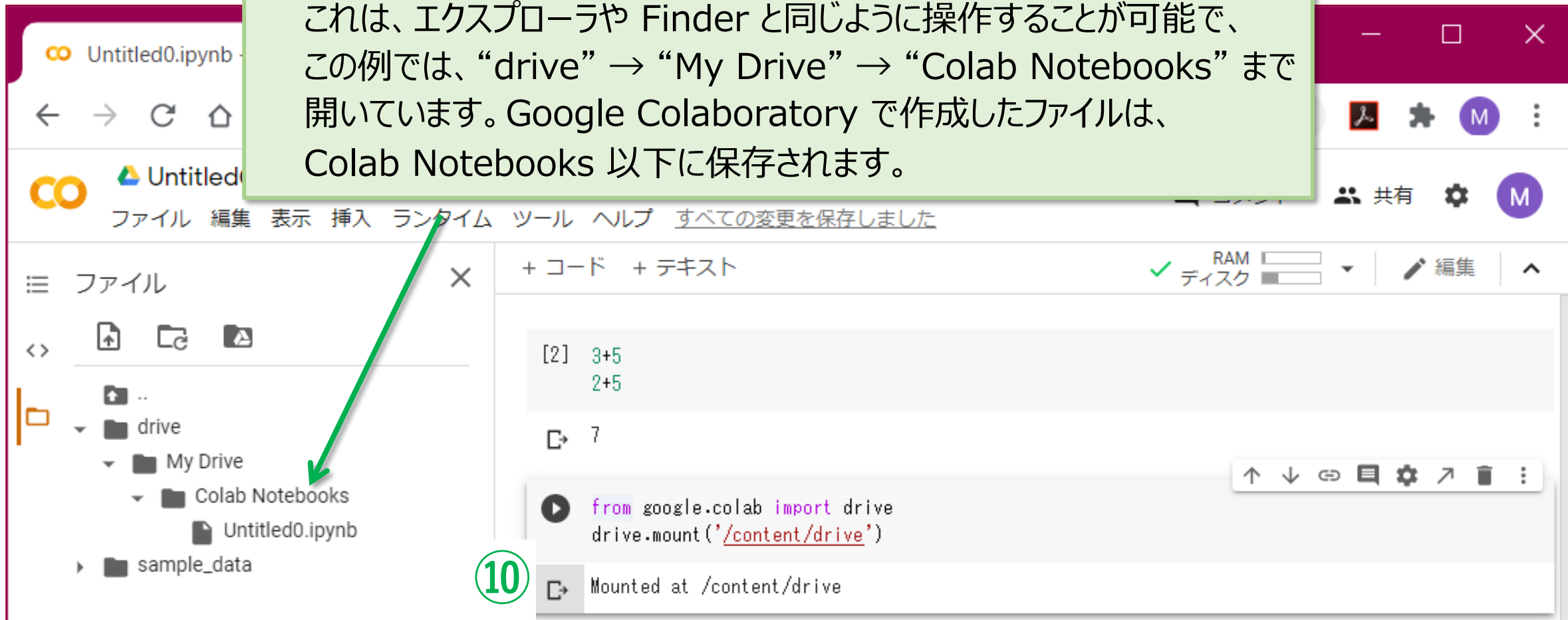
```
from google.colab import drive
drive.mount('/content/drive')
```

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6t

Enter your authorization code:

Google Drive のマウント (10)

⑩ 認証に成功すると、`/content/drive` にマウントしたと表示されます。左側にあるファイル システム ビューに、“drive”が追加されています。これは、エクスプローラや Finder と同じように操作することが可能で、この例では、“drive” → “My Drive” → “Colab Notebooks” まで開いています。Google Colaboratory で作成したファイルは、Colab Notebooks 以下に保存されます。



The screenshot shows the Google Colaboratory interface. On the left, a file explorer sidebar is open, showing a tree view: 'drive' (expanded) -> 'My Drive' (expanded) -> 'Colab Notebooks' (expanded) -> 'Untitled0.ipynb'. A green arrow points from the text box to the 'Colab Notebooks' folder. The main area shows a code editor with the following code and output:

```
+ コード + テキスト
```

```
[2] 3+5  
     2+5
```

```
7
```

```
▶ from google.colab import drive  
   drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

At the bottom left of the code editor area, there is a circled number '10' with a green border, which corresponds to the step number in the text box above.

【重要】データ ファイルへのアクセス（1）

- 前のスライドでも述べましたが、ファイル システムやファイルのパス名を理解できていないと、プログラムを作成して実行し、その結果を利用することがほとんどできません。
- パス名の利用に慣れましょう → これができないと今後何もできない…
- これは、本講義だけの問題ではありません。

【重要】データ ファイルへのアクセス（2）



Google Colab を起動した直後の状態です。
pwd コマンド (ファイル システムにおけるカレント フォルダの確認) と
ls コマンド (カレント フォルダ内にあるファイルおよびフォルダの確認) を実行し、
Google Colab のカレント フォルダと其中身を確認します。
カレント フォルダは `"/content"` で、`/content` 内には `sample_data` フォルダが存在しています。

【重要】データ ファイルへのアクセス (3)

1 [3] `from google.colab import drive`
`drive.mount('/content/drive')`
Mounted at /content/drive

2 [4] `!ls /content`
drive sample_data

2 [5] `!ls`
drive sample_data

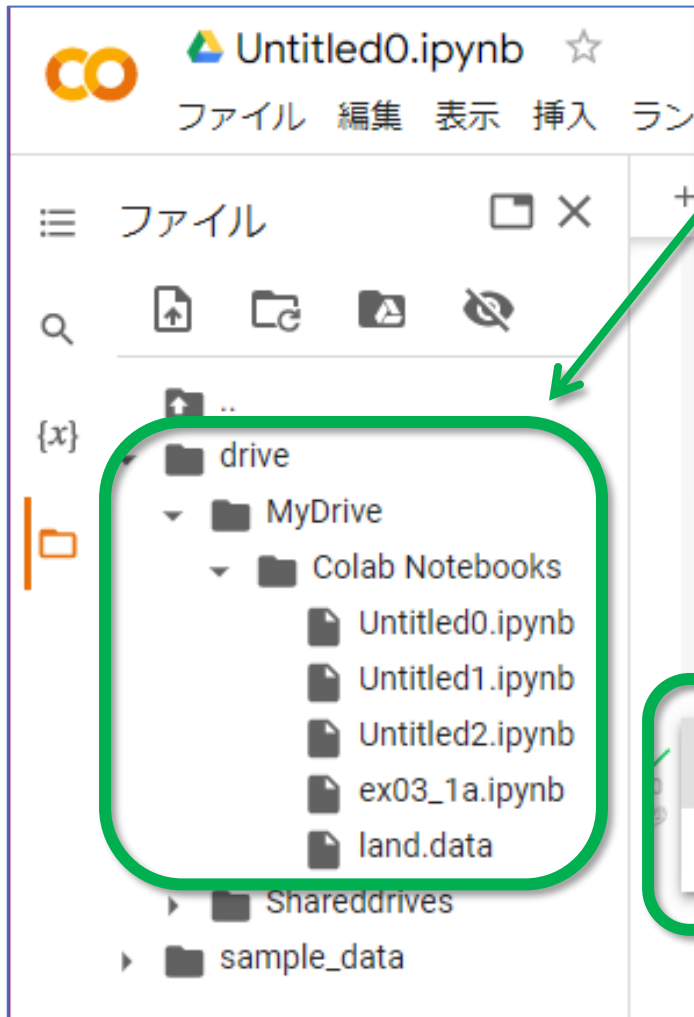
2 [6] `!ls drive/MyDrive/Data`
land.data

3 [7] `!cat drive/MyDrive/Data/land.data`
001101110111
011110111010
111110101001
100011111011

- ① Google drive を `/content/drive` にマウントした状態です。左側には Google Drive の階層構造が表示されています。これは、エクスプローラーや Finder のように操作できます。
- ② Colab のカレント フォルダは `/content` なので、カレント フォルダ以下に Google Drive がマウントされていることが分かります。
- ③ この例では、Google Drive のマイドライブにある Data フォルダに存在する `land.data` を開いています。`land.data` には 0/1 の数値が格納されています。

Python コードからデータ ファイル開いたり結果を保存するには、このようにパス名を使ってファイルを指定します。まずは、この例を十分に理解しておいて下さい。

【重要】データ ファイルへのアクセス（4）



Drive のマウント後、Google Drive へ「普通に」アップロードしたファイルは、Python コードから `/content/drive/MyDrive/` 以下のパス名で開くことができます(同様に、Python コードが作成したファイルも Google Drive 上で「普通に」操作できます)。

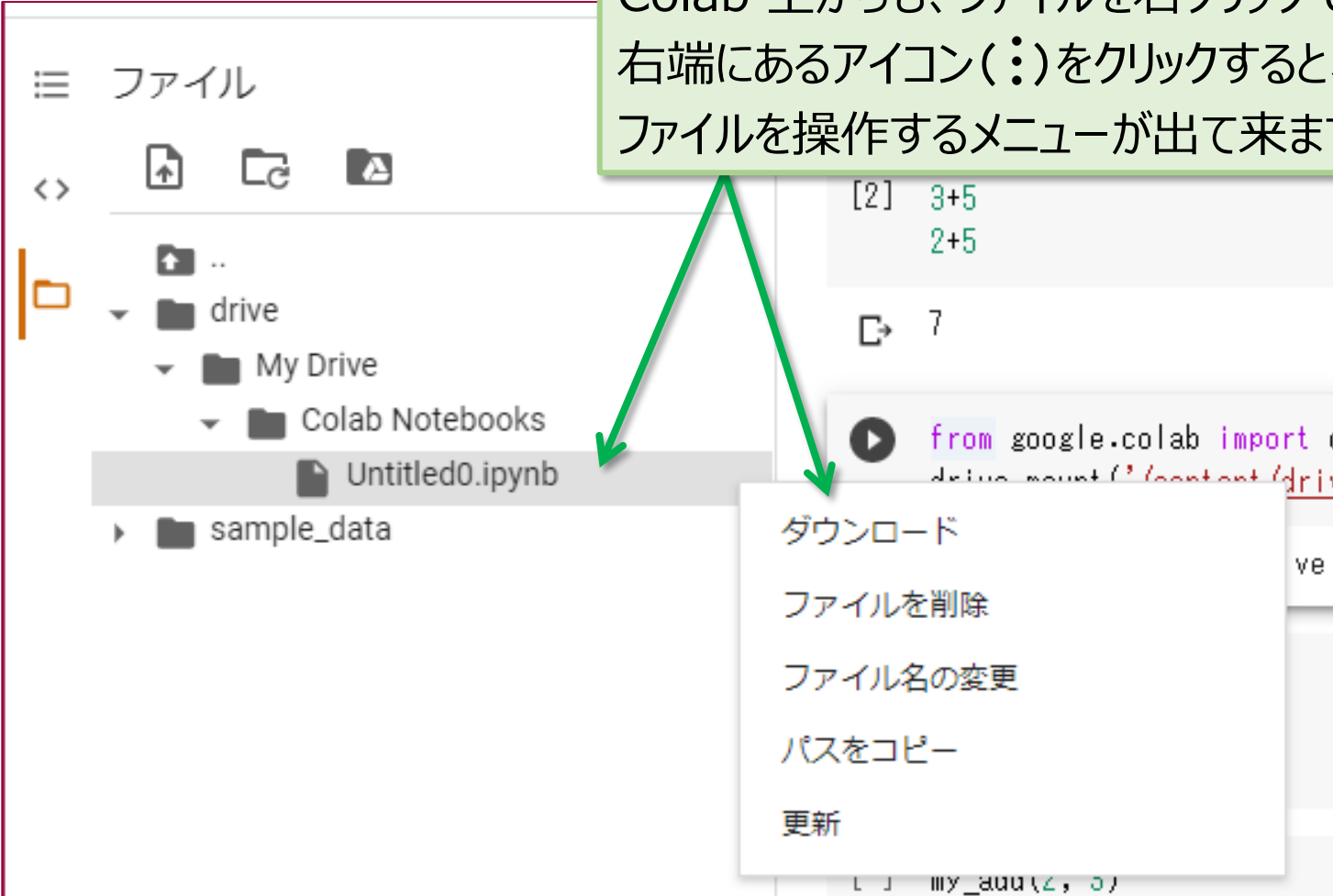
また、Colab で作成した Python プログラムは(空白の前にある`¥`に注意)、`/content/drive/MyDrive/Colab¥ Notebooks/` 以下のパス名で、アクセスできます(パス名の利用に慣れましょう)。

```
import pdb; pdb.set_trace()
print("debug 1")
my_add(2,3)
```

```
!ls /content/drive/MyDrive/Colab¥ Notebooks/
```

```
ex03_1a.ipynb  land.data  Untitled0.ipynb  Untitled1.ipynb  Untitled2.ipynb
```

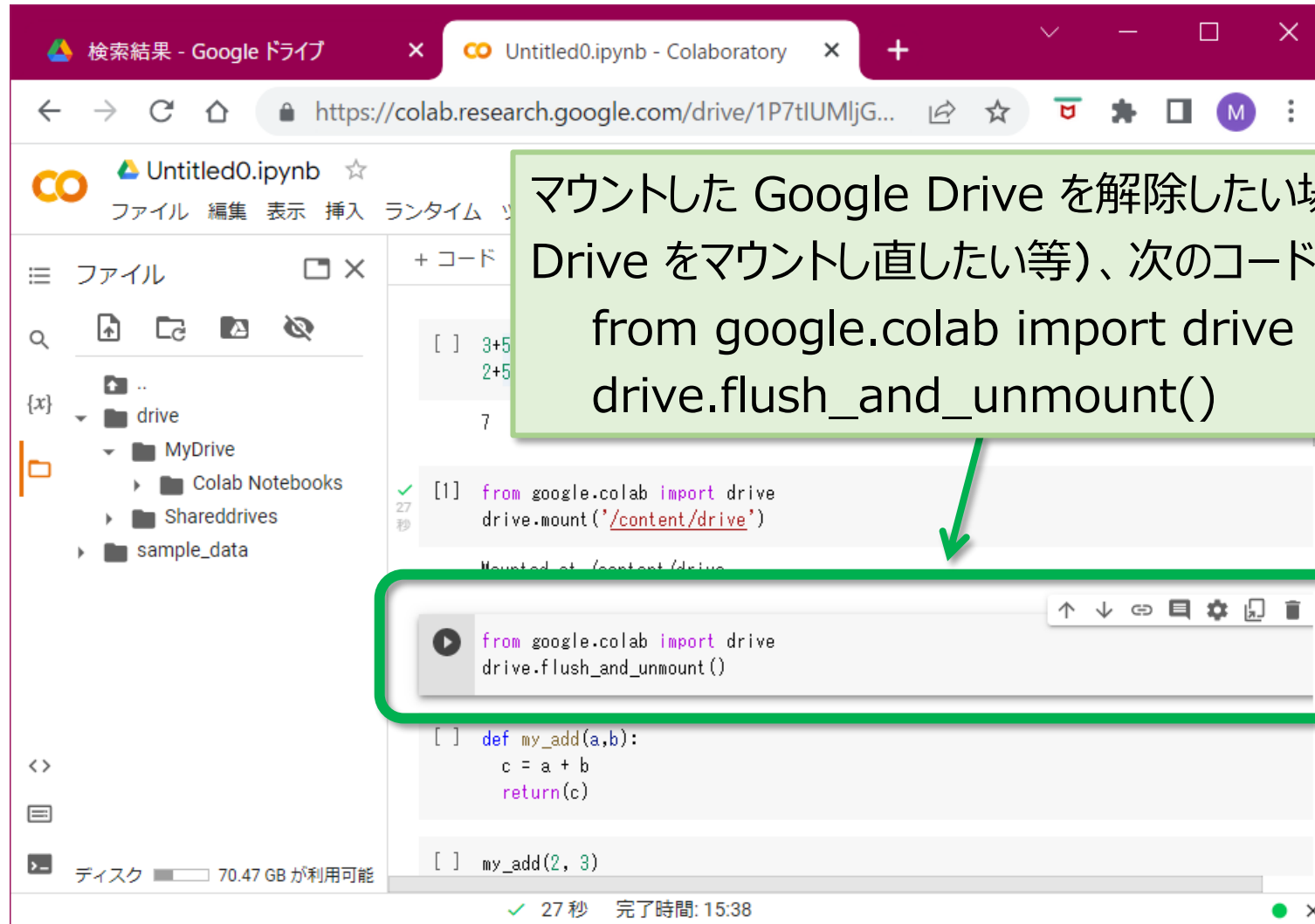
【重要】データ ファイルへのアクセス（5）



Colab 上からも、ファイルを右クリック or ファイル名の右端にあるアイコン(⋮)をクリックすると、Google Drive のファイル进行操作するメニューが出て来ます。

The screenshot shows the Google Colab file explorer interface. On the left, a sidebar lists folders like 'drive', 'My Drive', and 'Colab Notebooks', with a file named 'Untitled0.ipynb' selected. A context menu is open over this file, listing options: 'ダウンロード', 'ファイルを削除', 'ファイル名の変更', 'パスをコピー', and '更新'. A green callout box with arrows points to the file name and the context menu, containing the text: 'Colab 上からも、ファイルを右クリック or ファイル名の右端にあるアイコン(⋮)をクリックすると、Google Drive のファイル进行操作するメニューが出て来ます。'

Google Drive のマウントを解除 (1)



マウントした Google Drive を解除したい場合は(例えば、別アカウントの Drive をマウントし直したい等)、次のコードを新規セルで実行します。

```
from google.colab import drive
drive.flush_and_unmount()
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
from google.colab import drive
drive.flush_and_unmount()
```

```
def my_add(a,b):
    c = a + b
    return(c)
```

```
my_add(2, 3)
```

ディスク 70.47 GB が利用可能

Google Drive のマウントを解除 (2)

検索結果 - Google ドライブ × Untitled0.ipynb - Colaboratory ×

← → ↻ 🏠 🔒 https://colab.research.google.com/drive/1P7tIUmljG... ☆ 🗑️ ⚙️ 📄 M ⋮

CO Untitled0.ipynb ☆
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ [すべての変更を保存しました](#) コメント 共有 ⚙️ M

☰ ファイル 🗑️ X

🔍 📁 📄 🗑️

{x} ..
▼ drive
▶ sample_data

[] 3+5
2+5
7

✓ 27 秒 [1] `from google.colab import drive`
`drive.mount('/content/drive')`
Mounted at /content/drive

✓ 1 秒 ▶ `from google.colab import drive`
`drive.flush_and_unmount()`

↑ ↓ 🔗 💬 ⚙️ 📄 🗑️

Google Drive のマウントが解除されました。
“drive”の左側▼以下に何も表示されていない点に注意
 (“sample_data”は、“drive”以下にあるわけではない)。

困った時の対応

エラー メッセージを確認しよう (1)

```
def harmonic(n):  
    rval = 0.0  
    for i in range(n):  
        rval = rval + 1.0/i  
    return(rval)  
  
print(harmonic(10))
```

例えば、上のようなプログラムを作成したとします。各コードの意味は、講義が進むにつれ説明をしますが、取り敢えず上のプログラムでは、次の漸化式を計算すると考えて下さい。

$$a_{n+1} = a_n + 1.0/n \quad (a_0 = 0, n = 0, 1, \dots, 9)$$

この場合、 $n = 0$ の時に 0 による割り算が発生し、エラーとなることが予想されます。

実際に動かしてみましよう。

エラー メッセージを確認しよう (2)

```
def harmonic(n):
    rval = 0.0
    for i in range(n):
        rval = rval + 1.0/i
    return(rval)

print(harmonic(10))
```

```
ZeroDivisionError                                Traceback
<ipython-input-3-f4409d6c49f5> in <module>()
      5     return(rval)
      6
----> 7 print(harmonic(10))

<ipython-input-3-f4409d6c49f5> in harmonic(n)
      2     rval = 0.0
      3     for i in range(n):
----> 4         rval = rval + 1.0/i
      5     return(rval)
      6

ZeroDivisionError: float division by zero
```

SEARCH STACK OVERFLOW

ZeroDivisionError が発生したと表示されました。

左の表示にある ----> の部分に注目して下さい。

これらの表示は上から見て行きます。意味は下記の通りです。

- まずは 7 行目にある print(harmonic(10)) を実行した。
- そこから harmonic 関数へ入り、4 行目の rval = rval + 1.0/i で止まった。

その下に、止まった理由を表示しています。

ZeroDivisionError: float division by zero

講義が進むにつれ説明をしますが、float は実数を意味します。

これらのメッセージを参考にして、プログラムを修正しましょう。

エラー メッセージを確認しよう (3)

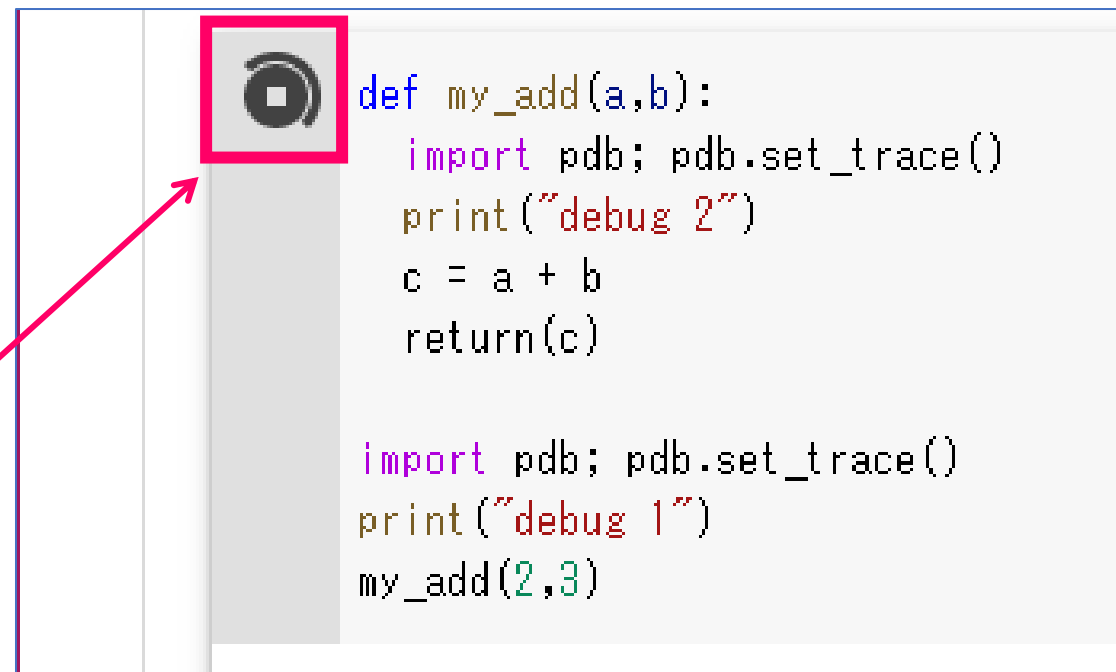
- 単純な誤りの原因として多いのは、以下の種類です。
 - インデントの整合が取れていない。
 - 複合文の “:” を忘れている。
 - 計数ループの範囲が合っていない。
 - 配列の上限/下限を超えている。
- 特に最後の二つについては、以降のスライド(デバッグの技法)を参考に、誤りを修正できるようにしましょう。

暴走してしまったら（１）

プログラムを実行したものの、いつまで経っても結果が出ない/終了しない場合は、プログラムが暴走している可能性があります。このような場合は、プログラムを強制終了し、後述する方法（デバッグの技法）を用いてプログラムを修正します。

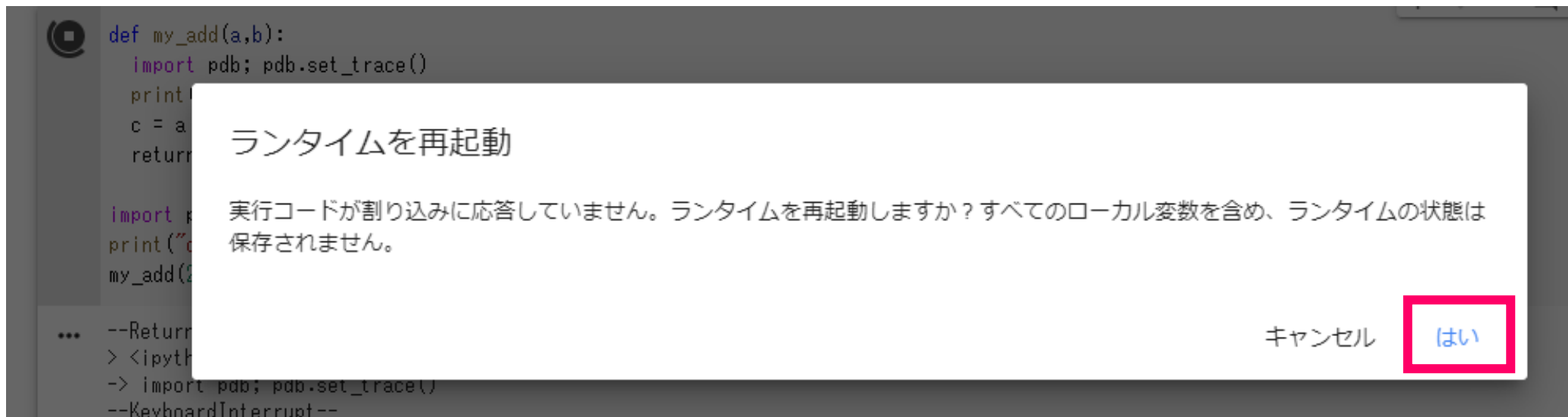
プログラムを強制終了するには、このアイコンをクリックします（実行前の形と実行中の形との違いに注意して下さい）。

強制終了されない場合は、何回かクリックしてみてください（どうしても終了しない場合は、“ランタイム” タブより “実行を中断” を選択）。



暴走してしまったら (2)

プログラムの強制停止を行なうと、以下のようなメッセージが表示されることがあります。



ランタイムとは、Google Colaboratory 上で Python プログラムの実行を行なう存在です。例えば、プログラムの実行箇所や、内部で利用するデータなどを管理します(これらは一例です)。上記メッセージの意味は、プログラムの強制停止の要求に対してランタイムが応答しないので、プログラムではなく、(と言うか、プログラムごと)ランタイムを強制再起動するという意味です。まあ、こうなってしまうたら仕方がないので、最終手段として「はい」を選びましょう。

デバッグの技法 (1)



```
def harmonic(n):  
    rval = 0.0  
    for i in range(n):  
        print(i)  
        rval = rval + 1.0/i  
    return(rval)  
  
print(harmonic(10))
```

0

```
ZeroDivisionError                                Traceback (most recent call last)  
<ipython-input-4-11a967b1c677> in <module>()   
      6     return(rval)  
      7  
----> 8     print(harmonic(10))  
  
<ipython-input-4-11a967b1c677> in harmonic(n)   
      3     for i in range(n):  
      4         print(i)  
----> 5         rval = rval + 1.0/i  
      6     return(rval)  
      7
```

ZeroDivisionError: float division by zero

SEARCH STACK OVERFLOW

プログラムのエラーを修正することをデバッグと呼びます。デバッグを行なうには、まずはそのエラー原因を突き止める必要があります。エラー原因の多くは、変数に予期しない値が入ってしまうことです。そのため、**怪しげな部分で変数の値を確認**することは、デバッグの基本と言えます。

この例では、計算をする直前で、**print 文**により計算に使う i の値を表示させています (print 文のより高度な使い方は、講義が進むにつれ説明します)。

エラーが検出される前に、 i の値 (0) が表示されていることを確認して下さい。

この技法は、基本中の基本なので (達人もこの技法を使います)、皆さんも**必ず習得**して下さい。

デバッグの技法 (2)

```
▶ def my_add(a,b):  
    c = a + b  
    return(c)  
  
my_add(2,3)
```

次は、左のようなプログラムを考えてみます。このプログラムは、足し算をする関数 `my_add` を定義し、それを外部から `my_add(2, 3)` として呼び出しています (つまり、`2 + 3` を計算しているわけです)。
`my_add(2, 3)` は、関数定義の外側にある点に注意して下さい。

```
▶ def my_add(a,b):  
    import pdb; pdb.set_trace()  
    print("debug 2")  
    c = a + b  
    return(c)  
  
import pdb; pdb.set_trace()  
print("debug 1")  
my_add(2,3)
```

このプログラムに対して、以下のコードを入れます (print 文は場所の確認用です)。
`import pdb; pdb.set_trace()`
その後、このセルを実行すると、上記コードを入れた場所で実行が停止します。この技法は、停止時点での変数を確認するだけではなく、停止した場所から 1 行ずつ再開できるので、自分が意図した順番でコードが実行されているかどうか確認できます (勿論、1 行進めた後に変数を確認することもできます)。但し現時点では、**上記のコードを入れる場所は、関数外 (左図で言えば `print("debug 1")`) よりも関数内 (同、`print("debug 2")`) の方が、混乱は少ないです (慣れてしまえば、どこでも可です)。ちょっと実行してみましょう。**

デバッグの技法 (3)

```
... --Return--
> <ipython-input-7-4879edcff19c>(7)<module>
-> import pdb; pdb.set_trace()
(Pdb) |
2     import pdb; pdb.set_trace()
3     print("debug 2")
4     c = a + b
5     return(c)
6
7 -> import pdb; pdb.set_trace()
8     print("debug 1")
9     my_add(2,3)
[EOF]
(Pdb) n
> /usr/local/lib/python3.6/dist-packages/IPython/core/in
-> sys.excepthook = old_excepthook
(Pdb) |
2880         self.hooks.pre_run_code_hook()
2881         #rprint('Running code', repr(cod
2882         exec(code_obj, self.user_globals)
2883         finally:
2884             # Reset our crash handler in pla
2885 ->         sys.excepthook = old_excepthook
2886     except SystemExit as e:
2887         if result is not None:
2888             result.error_in_exec = e
2889         self.showtraceback(exception_only=True)
2890         warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
(Pdb) |
```

前スライドのプログラムを実行した結果を表示します。
実行が停止したので、l コマンドで停止場所を確認しました。
確かに、import pdb; pdb.set_trace() の部分で止まっていますね。
但し、これは、関数外(左図で言えば print("debug 1"))にある
import pdb; pdb.set_trace() であることに注意して下さい。

次に、n コマンドで 1 行実行し、l コマンドで場所を確認してみました。
なんと、予想も付かない場所に行ってしまったようです。
これは、プログラムの実行を関数外で止めてしまったため、
利用者と対話的にやり取りするコードの中へ入ってしまったようです。
関数外では、Python の処理系は、利用者から入力があるかどうかを
確認しながら処理を進めるため、このようなコードが介在します。

デバッグの技法 (4)

```
def my_add(a,b):  
    import pdb; pdb.set_trace()  
    print("debug 2")  
    c = a + b  
    return(c)  
  
# import pdb; pdb.set_trace()  
# print("debug 1")  
my_add(2,3)
```

```
> <ipython-input-2-f13f7690921f>(3)my_add()  
-> print("debug 2")  
(Pdb) l  
1     def my_add(a,b):  
2     import pdb; pdb.set_trace()  
3 -> print("debug 2")  
4     c = a + b  
5     return(c)  
6  
7     # import pdb; pdb.set_trace()  
8     # print("debug 1")  
9     my_add(2,3)  
[EOF]  
(Pdb) n  
debug 2  
> <ipython-input-2-f13f7690921f>(4)my_add()  
-> c = a + b  
(Pdb) print(a, b)  
2 3  
(Pdb) c  
5
```

関数外にある `import pdb; pdb.set_trace()` を無効化して (先頭に `#` を付ける) 再実行してみました。
I コマンドより、関数内の `import pdb; pdb.set_trace()` で止まっていることが分かります。

続いて、`n` コマンドで 1 行実行すると、予想した通り `c = a + b` の行に移動したことが分かります。この時、`print(a, b)` コマンドより、`a` と `b` には 2 と 3 が入っていることが確認できました。
その後、`c` コマンドで、次の `import pdb; pdb.set_trace()` に出会うまで実行させます。今回は、`import pdb; pdb.set_trace()` に一度しか出会わない作りのため、プログラムが最後まで実行されて結果 (5) が表示されました。講義が進むにつれ説明しますが、プログラム内に繰り返しの構文が入っている場合、例えば `my_add` 関数は複数回呼び出される可能性があり、同じ `import pdb; pdb.set_trace()` に何度も出会うことがある点に注意して下さい。