Pythonプログラミング　中級

2024年12月7日（土）第１回
講師：渡邉 貴之

概要

- Pythonについて・・・
  - Pythonはコードが簡潔に記述できて読みやすいなどの特徴があり、様々な分野で人気を得ています。
  - 最近は機械学習やディープラーニングといった人工知能やデータサイエンスの分野においても、豊富な追加モジュールが揃っていることから、急速に人気が高まっています。
- 本講座では・・・
  - 本講座では、Pythonの初歩を学習済みの方を対象として、matplotlibやpandasなどのライブラリを用いたデータの集計や分析といった内容を座学だけでなく実際にコードを打ち込んで実行する演習を行います。
  - 各種アンケートデータの集計や分析をPythonで実行する方法について扱います。
- 参考図書
  - 下山ら, "Python実践データ分析100本ノック", 秀和システム

講座の内容

- タイムテーブル

第1回 （12月7日）	**9:30 ～ 10:15**
	Google Colaboratory環境の確認
第2回 （12月7日）	**10:25 ～ 11:10**
	pandasによるデータの集計と可視化1
第3回 （12月7日）	**11:15 ～ 12:00**
	pandasによるデータの集計と可視化2
第4回 （12月14日）	**9:30 ～ 10:15**
	scikit-learnによる回帰分析
第5回 （12月14日）	**10:25 ～ 11:10**
	MeCabによる自由回答の分析と可視化1
第6回 （12月14日）	**11:15 ～ 12:00**
	MeCabによる自由回答の分析と可視化2

※途中、休憩時間を適宜取ります

実習室利用ガイダンス

- おねがい
  - 室内での飲食は禁止されています。
- コンピュータの起動とログオン
  - 別紙「講座用ユーザIDとパスワード」をお手元にご用意下さい。
  - コンピュータの電源を入れ、ログオン画面にて「他のユーザー」をクリックします。
  - パソコン用ユーザIDとパスワードを入力し、矢印ボタンをクリックします。
  - ログイン後、中央モニタに講師のプロジェクタと同じ画面が映ります

その他

- 自販機
  - １階の階段横にあります。
- トイレ
  - 各階のエレベータ脇にあります。
- 食堂
  - 土日は休業となります。
- 喫煙場所
  - 構内は全面禁煙です。

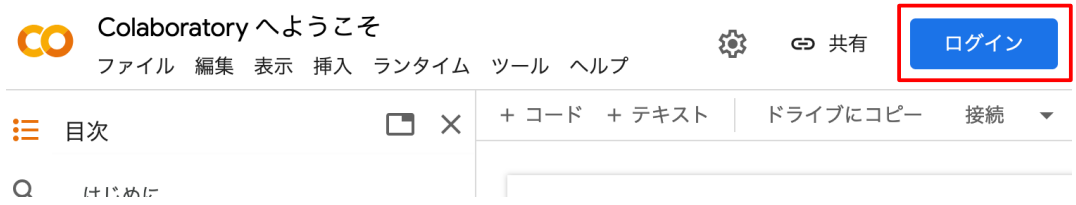
第１回
Google Colaboratory
環境の確認

目次

Google Colaboratory（1）

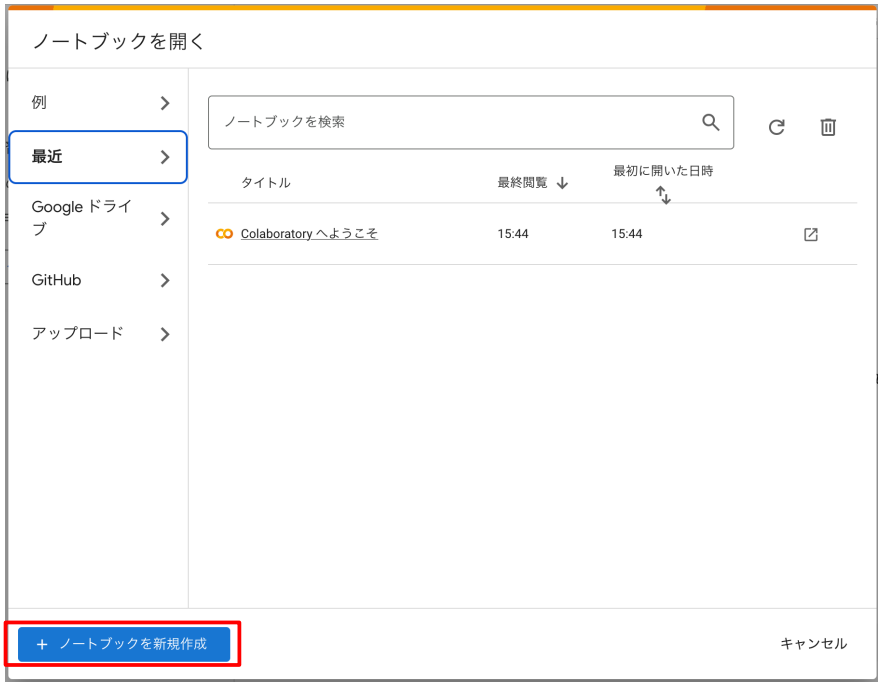
- 今回の実習では、インストール不要のPython環境である Google Colaboratory（以下、Google Colab）を使用します。
  - Googleアカウントをご用意ください
  - 手順
    - Webブラウザ（Edge/Chrome等）を起動する
    - google colaboratory で検索する
    - 下記リンクを開く

    - 「ログイン」ボタンをクリックし、Googleアカウントでログインする



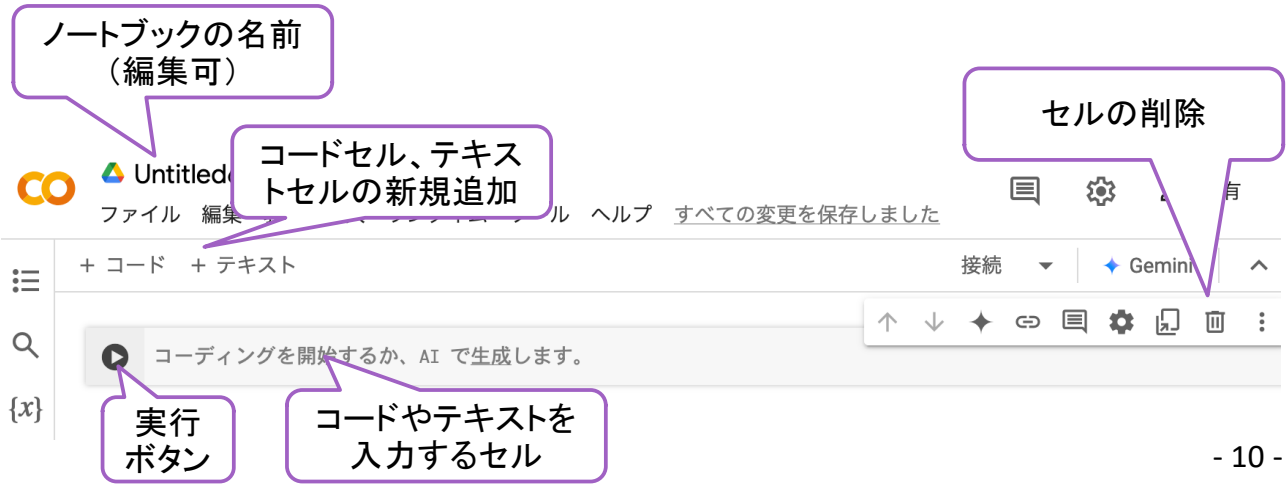
Google Colaboratory（2）

- Google Colaboratoryでは、ノートブックという単位でソースコードを記述します。
  - 「ノートブックを新規作成」ボタンをクリックします



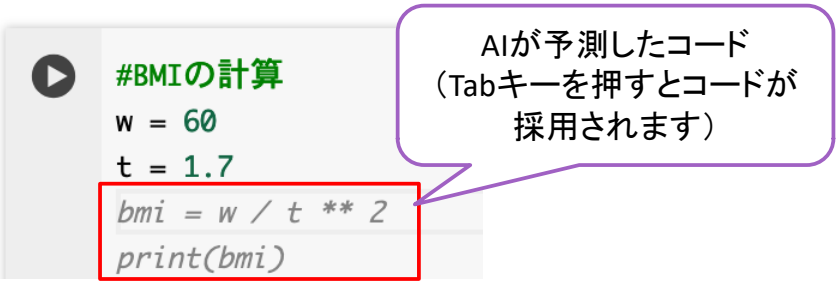
Google Colaboratory（3）

- コードやテキストを入力する枠をセルと呼びます。
- セルには、Pythonのコードを入力するコードセルと、説明書きを入力するテキストセルがあり、メニューからセルを追加することができます。
- コードは実行ボタンで実行できるほか、キーボードのCtrl＋Enterキーでも実行できます。また、Shift＋Enterキーで実行とセルの追加が同時にできます。

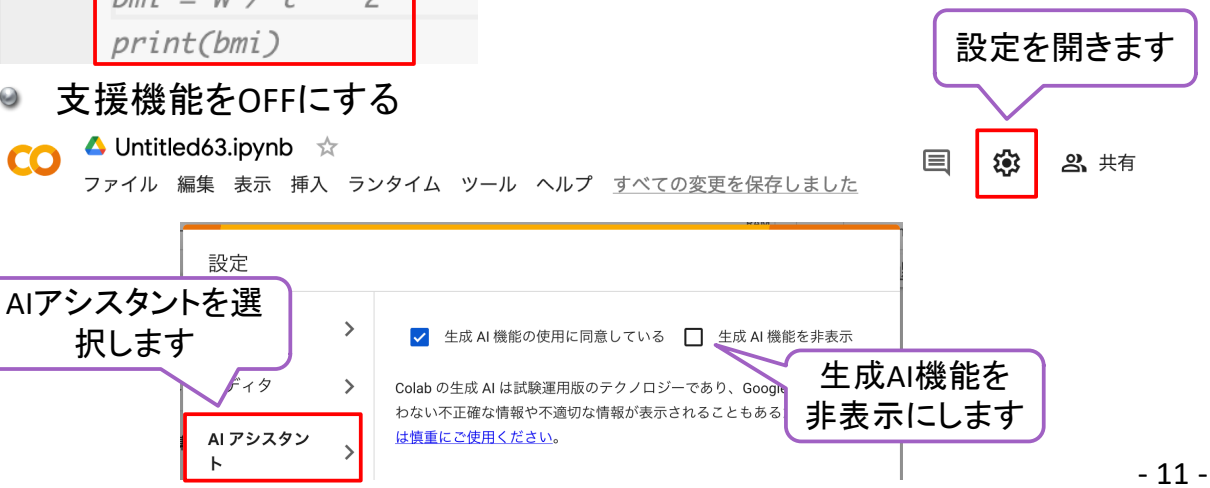


AIアシスタント機能

- 生成コード支援
  - Google Colabでは、コードの一部を入力すると、AIがその先のコードを予測して、候補を表示する機能が搭載されています。

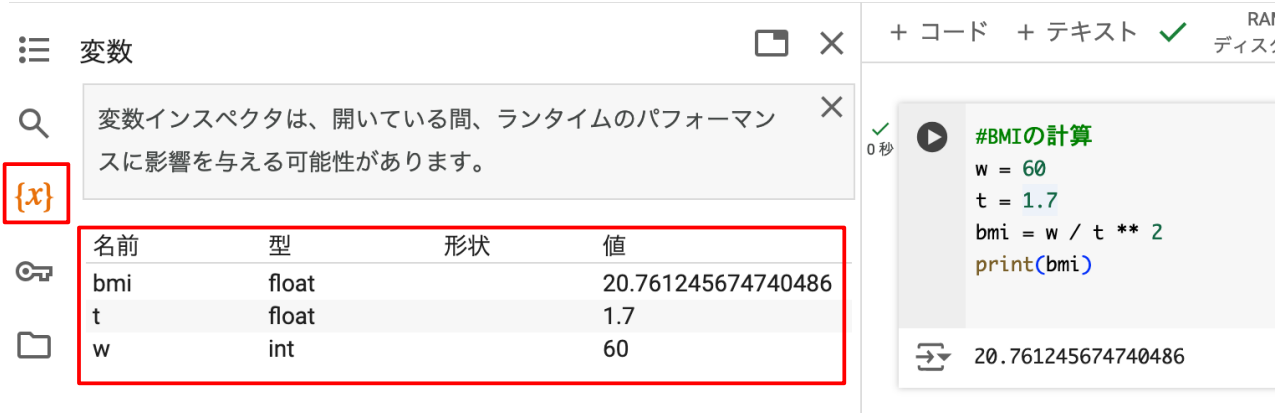


  - 支援機能をOFFにする



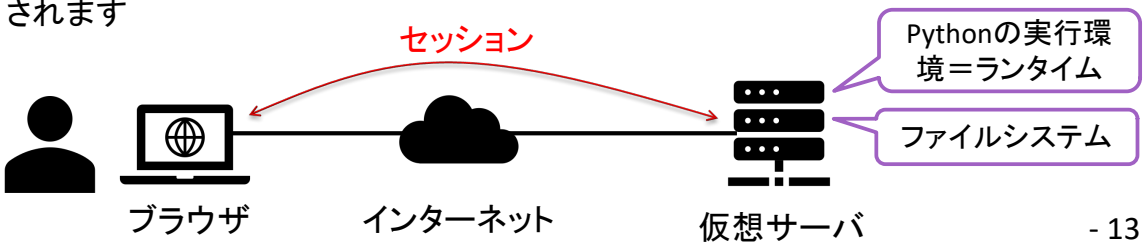
変数インスペクタ

- 現在、どのような変数に、どのような値が入っているかを確認することができます。
  - 画面左の {x} を押すと変数の一覧が表示されます



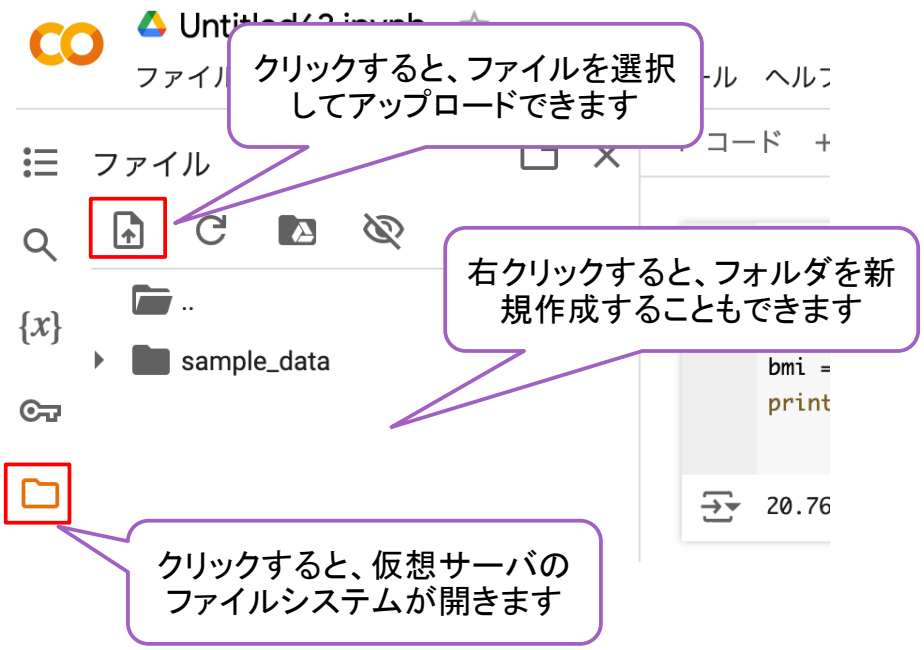
Google Colaboratoryの仕組み

- ランタイムとは
  - Google Colabでは、ノートブックを開いて最初にコードを実行すると仮想サーバがGoogle側で起動します（そのため、初回実行時には時間がかかります）
  - 仮想サーバが立ち上がると、Pythonの実行環境が用意され、これをランタイムと呼びます
  - 仮想サーバにはファイルシステムが用意され、ファイルをアップロードできます
- セッションとは
  - ブラウザとランタイムの接続のことをセッションと呼びます
  - 同時に幾つもノートブックを開こうとすると、「セッションが多すぎます」と新規セッションが制限されます
- 無料版の制限
  - 何も操作せずに90分経つとセッションが切断されます
  - ランタイムが起動して12時間経つと、ランタイムが初期化されます
  - 有料版（Google Colab Pro）では、90分ルールはなく、ランタイムは最長24時間維持されます



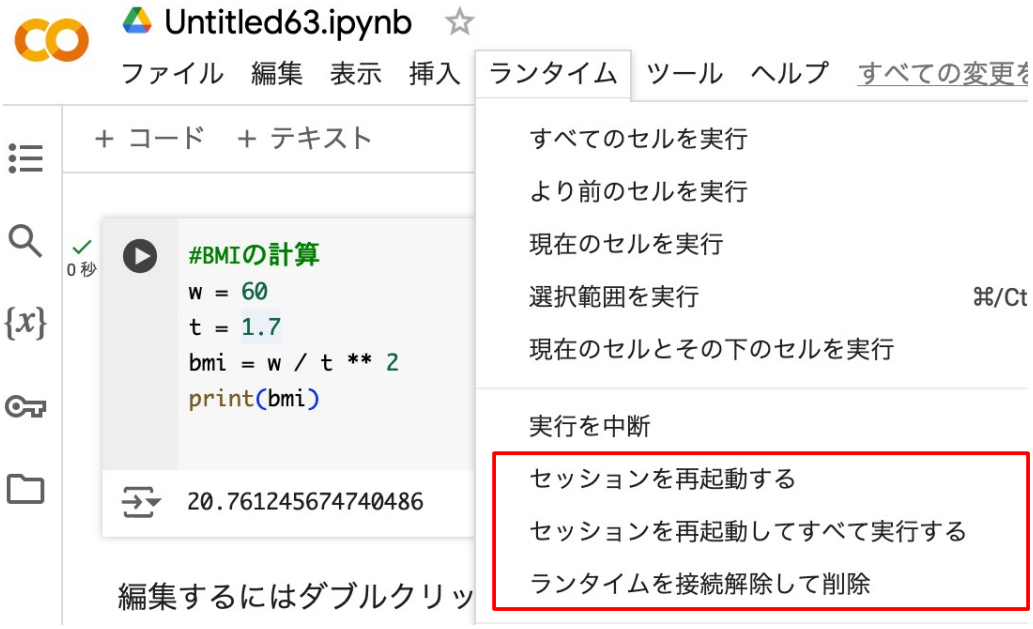
ファイルの操作

- ランタイムを実行している仮想サーバのファイルシステムに、ファイルをアップロードできます



ランタイムとセッションの操作

- 「セッションを再起動する」と、ランタイムは維持されますが、それまでに作成した変数はすべてクリアされます
- 「ランタイムを接続解除して削除」すると、セッションは切断され、ランタイムそのものが削除されます
  - アップロードしたファイルは削除されます



内包表記(コンプリヘンション)

- Pythonでは、リスト等に含まれる複数の要素に対して何か処理を行いたい場合はfor文を使うことができます。
- しかし、簡単な処理でもfor文の場合には複数行に渡るブロックの記述が必要です。

リストaの各要素を2乗したリストbを作成する例

```
a = [2, 4, 6, 8, 10]
b = []
for i in a:
    b.append(i*i)
print(b)
```

- 内包表記とは？
  - Pythonでは、内包表記という構文を使うと、リストやディクショナリを加工するような処理をブロックを使わずに1行で簡潔に書けるようになります。
  - しかも、実行速度も速くなります。

リスト内包表記(1)

- リスト内包表記は、あるシーケンスをもとにして、新しいリストを返す式として記述します。
  - 構文

リスト変数 = [一時変数による式 for 一時変数 in シーケンス]

  - 例:

```
a = [2, 4, 6, 8, 10]
b = []
for i in a:
    b.append(i*i)
print(b)
```

for文の場合

＝

```
a = [2, 4, 6, 8, 10]
b = [i * i for i in a]
print(b)
```

内包表記の場合

リストa内の要素をiに取り出してきて、iを基にリストbを作る

リスト内包表記(2)

- 複数リストの同じ順番の要素から新たなリストを作る
  - zip関数を使うことで、複数のリストから同じ順番に要素を取り出して、新たなリストを作成することができます。
  - 構文

リスト変数 = [一時変数による式 for 一時変数i, 一時変数j in zip(シーケンスA, シーケンスB)]

  - 例:3名の生徒の英語と数学の合計点を求める。

```
eigo = [80, 35, 65]
sugaku = [64, 76, 35]
goukei = [i + j for i, j in zip(eigo, sugaku)]
print(goukei)
```

```
[144, 111, 100]
```

(補足)リスト内包表記(3)

- 要素を追加する条件の指定
  - 内包表記にはifを追加することができ、条件がTrueの場合のみ要素が追加されます。
  - 構文

リスト変数 = [一時変数による式 for 一時変数 in シーケンス if 条件式]

  - 例:

```
a = [2, 4, 6, 8, 10]
b = []
for i in a:
    if i != 6:
        b.append(i*i)
print(b)
```

for文の場合

＝

```
a = [2, 4, 6, 8, 10]
b = [i * i for i in a if i != 6]
print(b)
```

内包表記の場合

iが6以外なら追加

(補足)リスト内包表記(4)

- 元となる要素の値によって追加する値の式を切り替えたい場合
  - 先ほどのifは要素を追加するかしないかを記述するための表記でしたが、追加する要素の式を切り替えたい場合には、forの前にifを書きます。
  - 構文

リスト変数 = [条件式が成り立つ場合の式 if 条件式 else 条件式が成り立たない場合の式 for 一時変数 in シーケンス]

  - 例:bmiが25.0以上の場合「太っている」それ以外「太っていない」という文字列に変換しなさい。

```
bmi = [25.4, 23.2, 26.7]
b = ["太っている" if i >= 25.0 else "太っていない" for i in bmi]
print(b)
```

```
['太っている', '太っていない', '太っている']
```

(補足)リスト内包表記(5)

- 元となる要素のインデックスも追加する要素の値や条件に加味したい場合
  - enumerate関数を使うことで、元となるリストのインデックス値を取り出すことができ、追加する要素の値や条件の材料にすることができます。
  - 構文

リスト変数 = [一時変数による式 for インデックス, 一時変数 in enumerate(シーケンス)]

  - 例:インデックス値をもとに、番号名簿を作る

```
a = ["鈴木", "田中", "加藤"]
b = [str(i+1) + "番:" + j for i, j in enumerate(a)]
print(b)
```

['1番:鈴木', '2番:田中', '3番:加藤']

(補足)リスト内包表記(6)

- 複数リストの要素全ての組合せから新たなリストを作る
  - 内包表記では、forを複数記述することで、複数のリストのすべての要素の組合せから新たなリストを作成することができます。
  - 構文

リスト変数 = [一時変数iとjによる式 for 一時変数i in シーケンスA for 一時変数j in シーケンスB]

    - この構文では、シーケンスAとシーケンスBのすべての要素の組合せが評価されます。
  - 例:ある大学にはAからCまでの棟があり、それぞれ3階建である。A棟1階からC棟3階までの要素を持つリストを作りなさい。

```
a = ["A棟", "B棟", "C棟"]
b = ["1階", "2階", "3階"]
c = [i + j for i in a for j in b]
print(c)
```

['A棟1階', 'A棟2階', 'A棟3階', 'B棟1階', 'B棟2階', 'B棟3階', 'C棟1階', 'C棟2階', 'C棟3階']

ラムダ（lambda）式

- ラムダ式とは
  - 一行で済むような簡単な関数を、簡易的に定義できるしくみ
  - 構文

    関数名 = lambda 仮引数: 仮引数による式

```
#2乗を計算する関数
def jijyo(x):
    return x * x

print(jijyo(4))
```

```
16
```

=

```
#２乗を計算するlambda式による関数
jijyo = lambda x: x * x

print(jijyo(4))
```

```
16
```