

演習授業中の質問対応について

Zoom ミーティング

表示

ミーティング チャット

演習授業中の質問をチューターの先生方が対応させていただきます。

演習にエラーが出たなど問題があったらリアクションの**挙手**を押してください。

質問内容を入力して、「**全員**」宛てに送信してください。

Miho Ishimaru

ここにメッセージを入力します...

宛先: 全員

ミュート解除

ビデオの開始

セキュリティ

参加者 2

画面共有

リアクション

アプリ

ホワイトボード

ノート

詳細

終了

演習4

Python基礎 モジュール、パッケージ、ライブラリ

本スライドは、自由にお使いください。
使用した場合は、このQRコードからアンケート
に回答をお願いします。



統合教育機構 曹 日丹

医療とAI・ビッグデータ入門

演習2-7の構成

Python基礎を学びましょう

演習2

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 12/7 10:40-11:35

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

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

モジュール Pythonのコード（**変数、関数、クラス**など）を含むファイルです

パッケージ

ライブラリ

Python基礎 モジュール、パッケージ、ライブラリ

モジュール Pythonのコード（**変数、関数、クラス**など）を含むファイルです

パッケージ 複数のモジュールをグループ化します。

モジュール

モジュール

モジュール

モジュール

モジュール

ライブラリ

Python基礎 モジュール、パッケージ、ライブラリ

モジュール Pythonのコード（**変数、関数、クラス**など）を含むファイルです

パッケージ 複数のモジュールをグループ化します。

モジュール

モジュール

モジュール

モジュール

モジュール

ライブラリ 複数のモジュールやパッケージで構成されています。

モジュール

モジュール

モジュール

モジュール

モジュール

モジュール

モジュール

モジュール

モジュール

モジュール

モジュール Pythonのコード（**変数、関数、クラス**など）を含むファイルです。

拡張子： .py

クラス：関数や変数などを含む構造です。
クラスの中の関数は、メソッドと言います。

モジュールを作ってみましょう

年末年始挨拶の言葉をモジュールにしたいです。

Merry Christmas!

関数

モジュール

Happy New Year!

関数

モジュールを作ってみましょう

年末年始挨拶の言葉をモジュールにしたいです。

コード：

```
def dec():  
    return "Merry Christmas!"
```

関数

コード：

```
def jan():  
    return "Happy New Year!"
```

関数

モジュール

モジュールを作ってみましょう

年末年始挨拶の言葉をモジュールにしたいです。

コード：

コマンド

ファイル名

```
%%writefile greetingM.py
```

```
def dec():  
    return "Merry Christmas!"
```

```
def jan():  
    return "Happy New Year!"
```

%%writefileとは：

新しいモジュールを、名前を指定して作成するコマンドです。

モジュール


モジュールを作ってみましょう

年末年始挨拶の言葉をモジュールにしたいです。

```
%%writefile greetingM.py
```

```
def dec():  
    return "Merry Christmas!"
```

```
def jan():  
    return "Happy New Year!"
```

 Writing greetingM.py



モジュールを作ってみましょう

年末年始挨拶の言葉をモジュールにしたいです。

関数を呼び出します

コード：

```
import greetingM
```

モジュール名：greetingM

モジュールを作ってみましょう

年末年始挨拶の言葉をモジュールにしたいです。

関数を呼び出します

モジュール名.関数名 ()

コード：

```
print(greetingM.dec())
```



'Merry Christmas!'

greetingのdecを呼び出しました。

コード：

```
print(greetingM.jan())
```



'Happy New Year!'

greetingのjanを呼び出しました。

Colab

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

 Colaboratory へようこそ
ファイル 編集 表示

目次

はじめに

データサイエンス

機械学習

その他のリソース

使用例

セクション

ノートブックを開く

例

最近

Google ドライブ

GitHub

アップロード

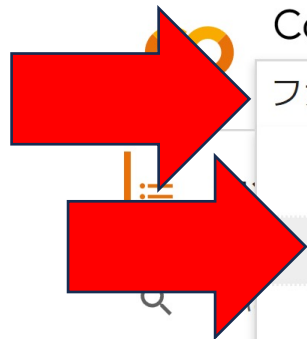
ノートブックを検索

タイトル	最終閲覧	最初に開いた日時	
 演習2コード	9:43	10月25日	 
 Colaboratory へようこそ	9:21	2022年12月23日	
 演習3コード	9:21	9:21	 
 演習4コード	11月2日	11月1日	 
 演習5コード	11月1日	10月13日	 

+ ノートブックを新規作成

キャンセル

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



Colaboratory へようこそ

ファイル 編集 表示 挿入 ランタイム ツール ヘルプ

ノートブックを新規作成

ノートブックを開く

Ctrl+O

ノートブックをアップロード

名前の変更

ドライブにコピーを保存

コピーを GitHub Gist として保存

GitHub にコピーを保存

保存

Ctrl+S

変更履歴

ダウンロード

印刷

Ctrl+P

ド + テキスト

ドライブにコピー

Colab へようこそ

に Colab をよくご存じの場合は、この動画でインタラクティブなラドの履歴表示、コマンドパレットについてご覧ください。



Colab とは

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

ノートブックを開く

例 >

最近 >

Google ドライブ >

GitHub >

アップロード >

タイトル	所有者	最終閲覧 ▲	最終更新 ▼		
演習4コード.ipynb					
演習準備資料.ipynb	曹日丹	11月1日	11月1日		
演習1116確認.ipynb のコピー	曹日丹	10月31日	10月27日		
2023入門dataframe.ipynb	曹日丹	10月31日	10月27日		
演習1116確認.ipynb	曹日丹	10月25日	10月25日		

+ ノートブックを新規作成

キャンセル

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

ノートブックを開く

例 >

最近 >

Google ドラ
イブ >

GitHub >

アップロード >



参照

または、ここにファイルをドラッグしてください

演習4コード.ipynb

演習授業中の質問対応について

Zoom ミーティング

表示

ミーティング チャット

演習授業中の質問をチューターの先生方が対応させていただきます。

演習にエラーが出たなど問題があったらリアクションの**挙手**を押してください。

質問内容を入力して、「**全員**」宛てに送信してください。

Miho Ishimaru

ここにメッセージを入力します...

宛先: 全員

ミュート解除

ビデオの開始

セキュリティ

参加者 2

画面共有

リアクション

アプリ

ホワイトボード

ノート

詳細

終了

18

モジュール Pythonのコード（**変数、関数、クラス**など）を含むファイルです

パッケージ 複数のモジュールから構成されています。

モジュール

モジュール

モジュール

モジュール

モジュール

パッケージを作ってみましょう

パッケージ

パッケージは階層構造を持っています。

📁 giftP

📄 __init__.py

📄 greetingM.py

モジュール1: greetingM.py

📄 presentM.py

モジュール2: presentM .py

パッケージ

パッケージは階層構造を持っています。

```
└─ giftP
   ├── __init__.py
   ├── greetingM.py
   └── presentM.py
```

コード：

```
!mkdir giftP          #ディレクトリを作成します
!touch giftP/__init__.py # 初期化ファイル
!touch giftP/greetingM.py # モジュール1
!touch giftP/presentM.py # モジュール2
```

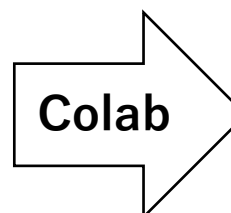
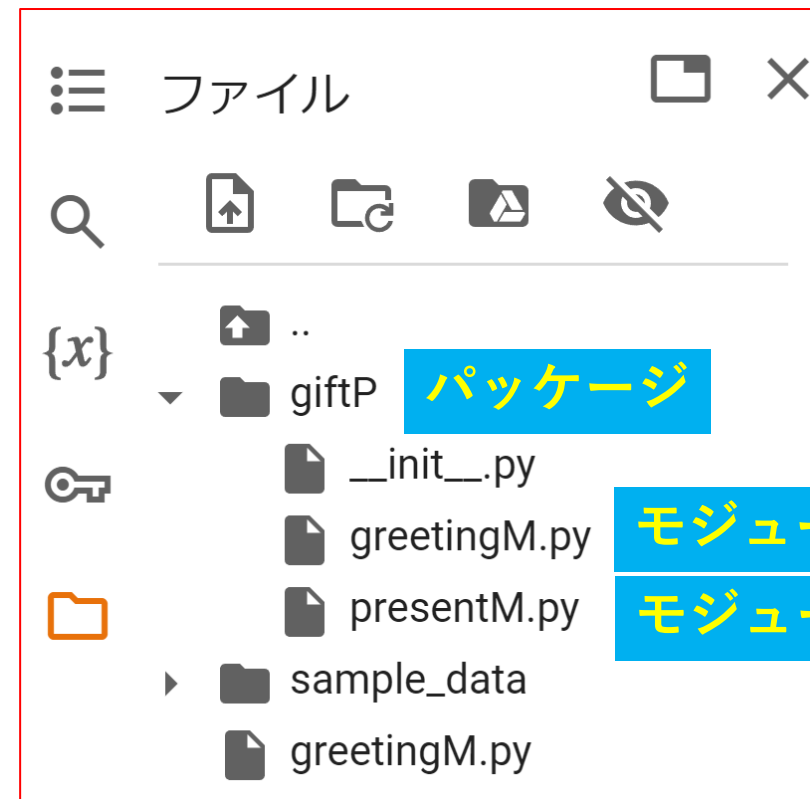
パッケージ

パッケージは階層構造を持っています。



コード：パッケージを作成します。

```
!mkdir giftP          #ディレクトリを作成します
!touch giftP/__init__.py # 初期化ファイル
!touch giftP/greetingM.py # モジュール1
!touch giftP/presentM.py # モジュール2
```



パッケージ

階層構造を持っています。

📁 giftP

📄 __init__.py

📄 greetingM.py

📄 presentM.py

コマンド

`%%writefile`

コマンド

`%%writefile`

パッケージ

階層構造を持っています。

📁 giftP

📄 __init__.py

📄 greetingM.py

📄 presentM.py

パッケージ名/モジュール名.py

%%writefile giftP/greetingM.py

パッケージ名/モジュール名.py

%%writefile giftP/presentM.py

パッケージ

階層構造を持っています。

📁 giftP

📄 __init__.py

📄 greetingM.py

📄 presentM.py

モジュール内容

モジュール内容

コード：

```
%%writefile giftP/greetingM.py
def dec():
    return "Merry Christmas!"
def jan():
    return "Happy New Year!"
```

コード：

```
%%writefile giftP/presentM.py
def dec():
    return "a strawberry cake!"
def jan():
    return "otoshidama!"
```

パッケージ

📁 giftP

📄 __init__.py

📄 greetingM.py

📄 presentM.py

```
from パッケージ名 import モジュール名
```

パッケージのモジュール1をインポートします

コード：

```
from giftP import greetingM
```

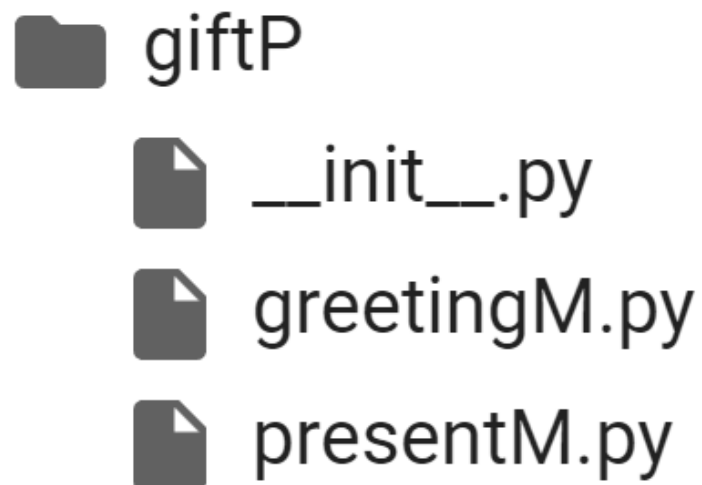
パッケージのモジュール2をインポートします

コード：

```
from giftP import presentM
```

パッケージ

階層構造を持っています。



コード：

モジュールを呼び出します

```
from giftP import greetingM
```

```
print(greetingM.dec())
```

```
➞ 'Merry Christmas!'
```

```
print(greetingM.jan())
```

```
➞ 'Happy New Year!'
```

パッケージ

階層構造を持っています。



コード：

モジュールを呼び出します

```
from giftP import greetingM
```

```
print(greetingM.dec())
```

```
↳ 'Merry Christmas!'
```

```
print(greetingM.jan())
```

```
↳ 'Happy New Year!'
```

```
from giftP import presentM
```

```
print(presentM.dec())
```

```
↳ 'a strawberry cake'
```

```
print(presentM.jan())
```

```
↳ 'an otoshidama'
```

Colab

既に存在しているモジュール

Pythonには多くの標準モジュールがあります。
一般的なモジュールの例：

- **math** - 数学的な関数や定数を提供します。
- **datetime** - 日付と時刻を操作するためと関数などを提供します。

既存しているモジュール math

モジュールをインポートします。

コード：

```
import math
```

コード：

```
print(math.pi)
```

mathの属性

円周率 π の値を取得します。

コード：

```
print(math.e)
```

mathの属性

自然対数の底を取得します。

既存しているモジュール math

モジュールをインポートします。

コード：


```
import math
```

コード：

```
print(math.pi)
```

mathの属性

円周率 π の値を取得します。


 3.141592653589793

コード：

```
print(math.e)
```

mathの属性

自然対数の底を取得します。

 2.718281828459045

Colab

既存しているモジュール math

mathの関数： `math.sqrt()`

平方根を計算します。 `math.sqrt(引数)`

コード：

```
print(math.sqrt(16))
```


既存しているモジュール math

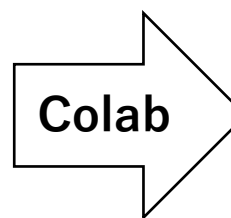
mathの関数： `math.sqrt()`

平方根を計算します。 `math.sqrt(引数)`

コード：

```
print(math.sqrt(16))
```

 4.0



既存しているモジュール math

mathの関数： `math.log(x,base)`

底が base である x の対数を計算します。

`math.log(x,base)`

コード：

```
x = 1000.0
base = 10.0

print(math.log(x,base))
```

既存しているモジュール math

mathの関数： $\log(x, \text{base})$

底が base である x の対数を計算します。

$\log(x, \text{base})$

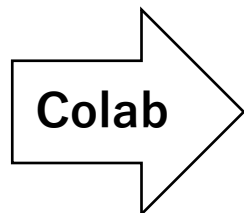
コード：

```
x = 1000.0
base = 10.0

print(math.log(x, base))
```

 2.9999999999999996

計算結果は 2.99999999999999962.9999999999999996 となりました。これは理論値である 33 に非常に近い値です。このわずかな差異は、コンピュータが浮動小数点数を扱う際に生じる誤差によるものです。



Pythonは豊富なライブラリを持っています。
いくつかの主要なPythonライブラリとその概要を示しています。

NumPy:

数値計算ライブラリ。
効率的な数値計算のための多次元配列と、それを操作するツールを提供しています。

Pandas:

データ分析ライブラリ。
DataFrameという強力なデータ構造を用いて、データの操作や分析を行うことができます。

Matplotlib:

データの可視化ライブラリ。
グラフの作成・描画ができます。

Scikit-learn:

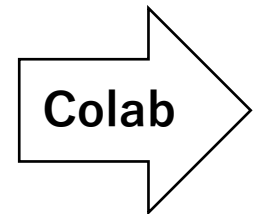
機械学習ライブラリ。
分類、回帰、クラスタリングなど、様々な機械学習アルゴリズムの実装が含まれています。

Numpy-ライブラリ

numpyをインポートする必要があります。

```
import numpy as np
```

npはnumpyの略称です。



numpyのarray関数

1次元配列の作成

1 2 3 4 5

2次元配列の作成

1 2 3
4 5 6
7 8 9
10 11 12

4 列、3行

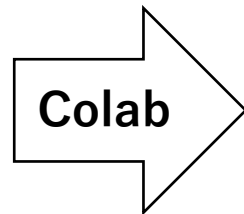
numpyのarray関数

```
np.array(リスト)  
print(np.array(リスト))
```

1次元配列の作成：リストから作成します

コード：

```
list1 = [1, 2, 3, 4, 5]  
  
print(list1)  
print(np.array([1, 2, 3, 4, 5]))
```



numpyのarray関数

```
np.array(リスト)  
print(np.array(リスト))
```

1次元配列の作成：リストから作成します

コード：

```
list1 = [1, 2, 3, 4, 5]  
  
print(list1)  
print(np.array([1, 2, 3, 4, 5]))
```



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

Colab

numpyのarray関数

```
np.array(リスト)  
print(np.array(リスト))
```

2次元配列(行列)の作成： リストから作成します

コード：

```
list2 = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]  
         [リスト1, リスト2, リスト3, リスト4]  
         [1行目, 2行目, 3行目, 4行目]
```

```
print(list2)  
print(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]))
```

numpyのarray関数

```
np.array(リスト)  
print(np.array(リスト))
```

2次元配列(行列)の作成： リストから作成します

コード：

```
list2 = [[1, 2, 3], [4, 5, 6, ], [7, 8, 9], [10, 11, 12]]  
print(list1)  
print(np.array([[1, 2, 3], [4, 5, 6, ], [7, 8, 9], [10, 11, 12]]))
```

```
➞ [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]  
[[ 1  2  3]  
 [ 4  5  6]  
 [ 7  8  9]  
 [10 11 12]]
```

Colab

属性

配列名.shape 形状を返します

配列名.size 総要素数を返します

配列名.ndim 次元数を返します

```
print(arr.shape) # 形状を表示
print(arr.size) # 総要素数を表示
print(arr.ndim) # 次元数を表示
```

コード：

```
arr1 = np.array([1, 2, 3, 4, 5])

print(arr1)
print(arr1.shape) # 形状を表示
print(arr1.size) # 総要素数を表示
print(arr1.ndim) # 次元数を表示
```

属性

配列名.shape 形状を返します

配列名.size 総要素数を返します

配列名.ndim 次元数を返します

```
print(arr.shape) # 形状を表示
print(arr.size)  # 総要素数を表示
print(arr.ndim)  # 次元数を表示
```

コード：

```
arr1 = np.array([1, 2, 3, 4, 5])
```

```
print(arr1)
```

```
print(arr1.shape) # 形状を表示
```

```
print(arr1.size) # 総要素数を表示
```

```
print(arr1.ndim) # 次元数を表示
```



[1 2 3 4 5]

(5,)

5

1

属性

配列名.shape 形状を返します

配列名.size 総要素数を返します

配列名.ndim 次元数を返します

```
print(arr.shape) # 形状を表示
print(arr.size) # 総要素数を表示
print(arr.ndim) # 次元数を表示
```

コード：

```
arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)
print(arr1.shape) # 形状を表示
print(arr1.size) # 総要素数を表示
print(arr1.ndim) # 次元数を表示
```

NumPyなどの配列操作ライブラリでの次元の表現方法に基づいています。

形状が (5,) であるということは、配列が5つの要素を持つ一次元配列であることを意味します。これは、2次元以上の配列とは異なり、行や列の概念が適用されないため、ただ一つの数値で要素数を示しています。



[1 2 3 4 5]

(5,)

5

1

Colab

属性

配列名.shape 形状を返します

配列名.size 総要素数を返します

配列名.ndim 次元数を返します

コード：

```
arr2 = np.array([[1, 2, 3], [4, 5, 6]])

print(arr2)
print(arr2.shape)      # 形状を表示
print(arr2.size)       # 総要素数を表示
print(arr2.ndim)       # 次元数を表示
```

属性

配列名.shape 形状を返します

配列名.size 総要素数を返します

配列名.ndim 次元数を返します

コード：

```
arr2 = np.array([[1, 2, 3], [4, 5, 6]])

print(arr2)
print(arr2.shape)      # 形状を表示
print(arr2.size)       # 総要素数を表示
print(arr2.ndim)       # 次元数を表示
```



```
[[1 2 3]
 [4 5 6]]
```

(2, 3)

6

2

Colab

配列操作メソッド 配列名.reshape(): 配列の形状を変更します。

1次元配列を2次元配列に変換します。

配列名.reshape(行の数, 列の数)

2次元配列を1次元配列に変換します。

配列名.reshape(-1)

1次元配列を2次元配列に変換します。

コード：

```
arr2 = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])  
  
print(arr2)  
print(arr2.reshape(3, 3))
```

```
➞ [1 2 3 4 5 6 7 8 9]  
   [[1 2 3]  
    [4 5 6]  
    [7 8 9]]
```

Colab

2次元配列を1次元配列に変換します。

コード：

```
arr3 = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])  
  
print(arr3)  
print(arr3.reshape(-1))
```



```
[[1 2]  
 [3 4]  
 [5 6]  
 [7 8]]  
[1 2 3 4 5 6 7 8]
```

Colab

統計的計算メソッド:

.sum() : 配列内の要素の合計を計算します。

.mean() : 配列の平均値を計算します。

```
arr3 = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
```

コード :

```
print(arr3.sum())
```

```
print(arr3.mean())
```

統計的計算メソッド:

.sum() : 配列内の要素の合計を計算します。

.mean() : 配列の平均値を計算します。

```
arr3 = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
```

コード :

```
print(arr3.sum())
```

➡ 36

```
print(arr3.mean())
```

➡ 4.5

Colab

WebClassで課題を提出してください、締め切りは12月21日23:59までです。

課題1：

Numpyライブラリをインポートするコードを書いてください。

Numpyの略称は「np」とします。

課題2：

print関数を使って、下記2つのリストから2次元配列を作成するコードを書いてください。

[1,2,3,4,5]

[5,4,3,2,1]