筋電図データで学ぶデータ処理入門

Python編

セミナーの構成

1. PythonとColaboratory
2. プログラムの制御
3. NumPyと数値計算
4. データの可視化
5. SciPyと信号処理
6. バッチ処理

第１回 PythonとColaboratory

本日のメニュー

- プログラミングとは
- Python
- Google Colaboratory
- print関数
- 間違い探し
- 算術演算
- 誤差
- データの種類
- 変数
- コメント
- コードの良い書き方
- 参考

プログラミング

- コンピュータへの命令を書くこと
- パズルゲーム的に

Python

- 1991年 Guido van Rossumが開発
- インタプリタ型スクリプト言語
- 拡張可能
- 機械学習ブームで人気
  - 科学技術計算ツールで特に人気
- 読みやすい・学習しやすい

PEP 8

PEP: Python Enhancement Proposal
Pythonicなコードの書き方

PEP 8: Pythonコードのスタイルガイド

- Pythonプログラムの書き方のルール
  - 変数の名前の付け方
  - インデント（字下げ）はスペース4個
  - スペースの入れ方 etc.

Google Colaboratory

- Jupyter NotebookのGoogle版
- 環境構築不要
- Webブラウザで記述、実行できる
- Markdownテキストも利用
- 共有できる
- 基本的に無料

演習上の注意点

- メンタル
  - 他人が決めたルールなので受けいる
  - パズルやプラモデルのように考える
  - エラー等は英語
    - 翻訳ツールを活用 → DeepL
- 作業面
  - プログラムのことを（ソース）コードという
  - プログラムは**半角英数**で
  - 大文字・小文字の区別がある
  - 行頭はあけない（スペースを入れない）
    - 行頭のスペースには意味がある
- スライド中の背景がグレーの部分はコードまたは入出力

やってみよう！

1. 準備

- ツール -> 設定 -> エディタ
  - インデント幅（スペース）を4に設定
  - 行番号を表示をチェック
    （1度行えばOK）
- 配付ファイルを自分のものに
  - 「ドライブにコピー」をクリック
- 右上の「接続」をクリック

2. Colaboratoryの使い方

- https://colab.research.google.com
- セル
  - コード
    - プログラムを記述
    - 再生ボタンで実行
  - テキスト
    - Markdownテキストを記述

3. Markdown

- 簡単な記号で書式を指定する文書作成法

```
# 見出し1
## 見出し2
### 見出し3
#### 見出し4
##### 見出し5
###### 見出し6

本文テキスト。ただ文章を入力する。
改行は無視される（スペースと同じ）。

段落を変えるには空行を入れる。

- 箇条書き1
- 箇条書き2
  - インデントあり 箇条書き3

1. 順序付きリスト1
2. 順序付きリスト2

**強調 セミナー中のメモ等に活用してください。**
```

4. Hello World!

(何故かプログラミング言語入門の定番)

画面に「Hello World!」と表示する命令

```
print('Hello World!')
```

ここでのルール等

```
print('Hello World!')
```

- 「'Hello World!'」はPythonで扱えるデータの一種で**文字列 String**という
- 文字列は「`'`」か「`"`」（引用符 quote）で囲ったもの
  - 2種類あるのはもう一方を文字として扱うため
- 文字列には日本語も使える

  `'こんにちは'`
- この形を**関数 function**という
  - 画面に出力しなさい、という命令の関数
  - printが関数名
  - カッコの中に関数に渡すデータ（**引数 argument**という）
  - 標準で使える関数を**組み込み関数 Built-in function**
- 行の始めから書く
  - 行頭にスペースを入れてならない

5. Hello World again!

```
print('Hello World!')
    print('Hello World!')
```

エラー Error

プログラムのなんらかの間違いのこと

```
File "<ipython-input-4-7a69e3c1a937>", line 2
    print('Hello World!')
    ^
IndentationError: unexpected indent
```

- `IndentationError`
  - エラーの種類 インデントの間違い
- `unexpected indent`
  - エラーの説明 不要なインデント
- Pythonでは行頭の字下げ（インデント）に意味がある
- 必要のないインデントをしてはならない

6 エラー Error

```
print('Hello World!)
print('こんにちは')
```

```
  File "<ipython-input-3-6d22d03c5544>", line 1
    print('Hello World)
                       ^
SyntaxError: EOL while scanning string literal
```

- `SyntaxError` 構文（文法）間違い
- `EOL while scanning string literal`
  文字列を調べている間に行末に達した
  → 引用符が閉じていない
- 文字の色を参考に

7 エラー Error

```
print(’こんにちは’)
print('こんにちは’)
```

```
  File "<ipython-input-8-2d7fd5f03d5a>", line 1
    print(’こんにちは’)
          ^
SyntaxError: invalid character ’’’ (U+2019)
```

- `SyntaxError` 構文（文法）間違い
- `invalid character ’’’ (U+2019)`
  無効な文字
  → 全角文字の使用

8. エラー Error

```
Print('Hello World!')
print('Hello World!')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-983c3189a5cf> in <cell line: 1>()
----> 1 Print('Hello World!')

NameError: name 'Print' is not defined
```

- `NameError` 名前の間違い
- `name 'Print' is not defined`
  'Print'は定義されていません
- 大文字と小文字の間違い、スペルミス

9. 文字列の連結

```
print('Hello' + ' ' + 'World!')
print('Hello', 'World!')
```

- 文字列は足し算するとつながる
- print関数は引数をカンマ区切りで複数あたえるとスペース区切りで出力

10. 算術演算

```
3 + 1
```

- 実行すると計算結果を表示
- 「+」のような記号を **演算子 operator** という
- 算術演算を行うので特に算術演算子という

10. 算術演算

```
3 + 1
3 - 1
```

- 最後の結果しか出力されない
- 基本的にprint関数を使いましょう

11. 算術演算

```
print(3 * 2)
print(4 ** 2)
print(4 / 2)
print(4.0 * 2)
```

- 計算結果
  - 割り算以外の整数同士の結果は整数 integer
  - 割り算の結果は必ず浮動小数点数 float
  - 浮動小数点数が含まれていたら浮動小数点数

12. 算術演算

```
print(4 / 2)
print(4 // 2)
print(5 // 2)
print(5 % 2)
```

- 割り算の結果
  - 浮動小数点数
  - 商 整数に丸める
    小数点以下切り捨て
  - 剰余 整数

13. 算術演算

```
print(3 + 2 * 10)
print((3 + 2) * 10)
```

- 計算の順序は数学と同じ
  - 乗除算が優先
  - 左から順に
  - 順番を変えるには括弧を仕様

PEP8

- スペースの入れ方
  - カッコのすぐ内側はスペースなし
  - カンマの後にスペース
  - 演算記号の前後にスペース
  - 見にくくなる場合は入れなくても良い

    ex) `(8*2 + 3*4)`

算術演算子

演算子	意味	優先順位
+	足し算	3
-	引き算	3
*	かけ算	2
/	割り算（結果は浮動小数点数）	2
//	割り算（結果は整数、 小数点以下切り捨て）	2
%	剰余（割り算の余り）	2
**	べき乗	1

- ()カッコで優先順位を変更できる

データ型 type

文字列 String

```
'Hello World!'
'こんにちは'
"Hello Werktisch!"
"That's mine"
'3.14'
```

- プログラムで文字を扱いたいとき使用
- 引用符（'または"）に囲まれた文字
- 「'」を文字列に含めたいときは引用符は「"」
- 「"」を文字列に含めたいときは引用符は「'」
- **数字**を引用符で囲むと文字列、**数値ではない**

データ型

整数 Integer

```
10
```

- 序数、インデックス等に使用

浮動小数点数 Float

```
10.0
3.14
```

- 通常の計算に使用

14 データ型のエラー

```
print(3 + '2')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-4f605c4b9ee9> in <cell line: 1>()
----> 1 print(3 + '2')

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- `TypeError` 型の間違い
- `unsupported operand type(s) for ...`
  +演算子の**被演算子 operand**に整数と文字列はサポートされていない
- 文字はコンピュータにとってただの記号で意味はない

15. 浮動小数点数の誤差

```
print(0.1 + 0.1 + 0.1)

0.30000000000000004
```

- 二進数では正確に小数を表現できない
- 精度としては充分実用的
- 条件判断等で注意が必要
  - 0.3以下だったXXXをしなさい

16. 変数 variable

- データの入れ物（少し古い考え）
  - 識別のための名前
  - 名札に近い
- 保存や再利用のために
- 変数名 データに名前をつける
  - 使える文字
    英数字とアンダーバー `_`
  - 先頭の文字
    英文字とアンダーバー `_`（数字は使えない）
- **予約後 keyword** は使えない

```
greeting = 'Hello'
trade_name = 'Werktisch'
print(greeting + ' ' + trade_name)
```

- 左辺の変数に右辺の**値**を入れる（**代入**する）
  - 同じものにする
- `=` は代入演算子

キーワードリスト

False	None	True	and	as
assert	async	await	break	case
class	continue	def	del	elif
else	except	finally	for	from
global	if	import	in	is
lambda	match	nonlocal	not	or
pass	raise	return	try	while
with	yield	_ (underbar)		

PEP8 変数名の付け方

```
this_year = 2023
PI = 3.14
```

- なるべく英単語
- 複数は単語をアンダーバーで区切る
- 全て小文字
- 定数（変わらない値）は全て大文字

17. 予期せぬエラー

```
word_length = len('python')
print(word_length)

len = 10
word_length = len('colaboratory')
print(word_length)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-b01b0cde5f5c> in <cell line: 1>()
----> 1 word_length = len('python')
      2 print(word_length)
      3
      4 len = 10
      5 word_length = len('colaboratory')

TypeError: 'int' object is not callable
```

- `'int' object is not callable` は呼び出せない
- 既存の関数名等を変数名にすると関数が使えなくなる
- 知らず知らずにやってしまうことがある

18. 変数

- 繰り返し使える
- ちょっとした変更にも便利

```
greeting = 'Hello'
name = 'Werktisch' # 自分の名前をいれて実行しましょう

print(greeting)
print(name)

print(greeting + ' ' + name)

message = greeting + ' ' + name
print(message)
```

19. 変数

- 再代入

```
number = 1
print(number)

number = 2
print(number)

number += 1
print(number)
```

- 新しい値をいれると、元の値は失われる

20. コメント

```
greeting = 'Hello'
name = 'Werktisch'

# print(greeting)
# print(name)

# greetingとnameを使って文章で出力
print(greeting + ' ' + name)
```

- プログラムとしては無視される
- 何をしているかメモする
- 一時的にコードを無効にしたいときに（**コメントアウト**）

21. モジュール

- プログラムを再利用可能なように保存したもの
  - ライブラリ
  - パッケージ（配付単位）
  - インストールが必要
    - Colabでは主要なものはインストール済み
- import文で読み込んで使用
- as で別名をつける
- ドット区切り構造（ディレクトリ構造に由来）
- from で一部だけ読み込む

```
import numpy as np
from bokeh.io import output_notebook, show

data = np.array([0, 1, 2, 3, 4])
print(data)
print(type(data))

# 出力
[0 1 2 3 4]
<class 'numpy.ndarray'>
```

22. 代表値の算出

- サンプルデータの最大値、平均値、積分値を求めます
- サンプルデータの型はnumpy.ndarrayという型
  - 主に行列演算をするための型

```
import numpy as np

# サンプルデータ
# 30から50の間の整数から、ランダムに10個選んだデータ
# 時刻はサンプリング周期0.1s
data = np.random.choice(np.arange(30, 51), 10)
t = np.arange(0, 1, 0.1)

print(f'時刻: dt = {t}')
print(f'振幅: data = {data}')

# 結果
時刻: dt = [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
振幅: data = [43 40 37 46 40 36 41 30 49 47]
```

注) ランダムな値なのでスライドとはことなることも

最大値

```
max_value = data.max()
print(max_value)

# 結果
49
```

平均値

- 時系列データの平均値は1sあたりの値
- 離散データの場合は算術平均と同じ

```
mean_value = data.mean()
print(mean_value)

# 結果
40.9
```

積分値

- 台形近似法
- `scipy.integrate.trapezoid` を使用

```
from scipy.integrate import trapezoid

integral_value = trapezoid(data, t)
print(integral_value)

# 結果
36.4
```

Pythonの型について

- Pythonの型はすべてオブジェクト
  - ex) 文字列 `'Hello'` は文字を表しているだけではない

オブジェクト

- モノ（オブジェクト）を中心に考える
  - モノを表す情報
  - そのモノに対して出来る操作

```
value = '3'
print(f'Type of value is {type(value)}')

isnumeric = value.isnumeric()
print(f'Variable value is numeric?\n\tIt is {isnumeric}')

# 結果
Type of value is <class 'str'>
Variable value is numeric?
        It is True
```

Google Colaboratory

https://colab.research.google.com/

翻訳ツール

https://www.deepl.com/

Marp

Markdown Presentation Ecosystem

https://marp.app/

This slide deck was rendered by Marp.