# ソースコードの色

class 予約語

import ライブラリ関連？ from, importなど

TestClassクラス

'string' 文字列

コメント：コメント

# 基本情報

## 特徴

・インタプリタ型（コンパイルしない）

・変数がすべて参照型object。それには関数ポインタも含まれる。

・インデント型（ C言語などと異なり，{} などで区切らない）

・「実行可能な擬似コード」と呼ばれる。これは「人間とコンピュータの双方が理解できる表現」という意味。

## オブジェクト指向的な特徴

### オーバーロード

基本的にオーバーロードはできない。どうしてもやりたい場合は

・パラメータに対するデフォルト値を用意

・classmethodを用いる

### operator

演算子に対応したメソッドの記述。Pythonでもできる（らしい）

（公式）<https://docs.python.org/ja/3/library/operator.html>

### 型変換

#### 継承関係にある場合

#### 継承ではない場合

できない？

### 参照渡し

pythonでは変数は基本的に「参照渡し」

但し、引数のオブジェクトが「変更不可な型（Immutable）」（str, int値など）か「変更可能な型（Mutable）」かによってコピー変数での挙動が変わってくる

## python自体のコマンド

pythonの基本の起動方法

python # python2を対話形式で起動

python-is-python2 # 近年(2021.07月現在)のpython2 。「python2はもう使うな」という意思を感じる

python3 # python3の場合

python3 some\_cord.py # スクリプトの実行

オプション

-m モジュールをスクリプトとして実行 mod

-c コマンドの実行？

## 定義済関数

isinstance

if isinstance(numeric\_level, int):

pass

## 実行環境の構築

### 概要

pythonの場合、以下の様な用途で本環境とは異なる環境を用意する事が多い。

・実験の為に新しいライブラリを導入したいが，本環境は汚したくない場合。

・サーバーや客先環境など，特定の環境をそっくりそのままクローンしたい場合。

・あるライブラリの，特定バージョンが入った環境が欲しい場合。

参考https://qiita.com/po3rin/items/5c853bc172e5de330148

### venv

インストール

sudo dpkg -l \*venv # インストール有無を確認(dpkg)

sudo apt list --installed \*venv # インストール有無を確認(venv)

sudo apt install python3-venv # 仮想環境を作るパッケージのインストール

例

mkdir **myproj** # 実行環境用のディレクトリを作成

python -m venv **myenv** # 仮想環境を作成

**.myenv**¥Scripts¥activate # 仮想環境を有効化(Win) ※psの実行ポリシーをRemoteSigned

source ./**myenv**/bin/activate # 仮想環境を有効化(Linux, Mac)

(myenv) python -m pip install --upgrade pip # 試しにpip アップグレード

(myenv) deactivate # 終了

補足

python2の場合はvirtualenvを用いる。

sudo pip install virtualenv

（参考）

venvの使い方 <https://virtualenv.pypa.io/en/16.7.9/userguide.html>

ソースファイルなどはmyvenv\Lib\sites-packages\{package\_name}に格納される

### pipenv

pip install pipenv

### miniconda

#### 概要

軽量なanaconda。自身でパッケージ管理したい時などに用いる。また、minicondaは無料のまま。

miniconda+conda-forgeで環境を構築する人が多い（らしい）

#### チュートリアル

インストール

Linux 公式のshellスクリプトを用いる。（注意；ファイルに実行権限を加える必要あり）

sudo chmod a+x Miniconda3-latest-Linux-x86\_64.sh

bash Miniconda3-latest-Linux-x86\_64.sh

# > デフォルトだと /home/username/miniconda にインストールされる

Windows 公式のインストーラを用いる

<https://docs.conda.io/en/latest/miniconda.html#miniconda>

使い方

[ conda prompt ]

(base) ~$ conda create -n myenv # /home/username/miniconda3/envs/myenv が作成される

(base) ~$ conda activate myenv # myenvをアクティベート

(env) ~$

(env) ~$ conda deactivate # myenvを終了

アンインストール

Linux

conda install anaconda-clean # 先にファイルを削除？

rm -rf /home/username/miniconda3 # cleanしたら、ディレクトリごと削除すれば良い

強いて言うなら、環境変数pathからminicondaの情報を削除しておく

<https://docs.anaconda.com/anaconda/install/uninstall/>

#### conda-forge

conda config --add channels conda-forge

#### レファレンス

##### コマンド

conda-clean

create

conda create -n <仮想環境名>

conda create -n <仮想環境名> python=3.6

install

conda install anaconda-clean #

conda install anaconda-clean --yes #

### anaconda

#### 概要

200名以上の営利企業で公式レポジトリ（repo.anaconda.com）を利用する場合は有料。

### WSL

Windowsの場合

## Jupyter Notebook

コードや実行結果をそのままの形で保存できる。

実験的にコードを実行する場合や、教育などで使われる。

## 環境変数

PYTHONUNBUFFERED

空でない文字列を設定すると準入出力及び標準エラー出力がすべてバッファリングなしで行う（？）

python -u と同等。

・

・

## VS Codeの利用

### 概要

デバッグやインテリセンス機能が強力なので、pythonの開発にはVS Codeを強く推奨。

但し、VS Codeで仮想環境を構築してpythonを動かしているようで、実際に使う際にはPCの方にもpythonをインストールする必要がある。

### 標準設定

注意：

※ここには全体的な設定方法を書き残すが、ワークスペースと仮想環境を利用するケースの方が圧倒的に多いと思われる。

File → Preferences → Settings または→ Settingsでも良い

作業ディレクトリ User：Features → Terminal → Integrated: Cwd

python pipの使い方：

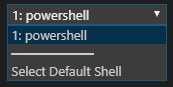
PSコマンドプロンプト

python -m pip install --upgrade pip

インタープリタ選択

コマンドパレットを開く（Ctrl+Shift+P）→ Python: Select Interpreter （タイプしても良い）

PC側の管理

powershell を使う

VSCode上でCtrl+@ でTerminal を開く → インタープリタにpowershell を選択する。（試しに下の --upgrade をやってみる）

### ワークスペース

複数のファイル、ディレクトリを１まとめにしたもの。.code-workspace というjson形式のファイルで管理する。

保存：File → OpenFolder から Save Workspace As が一番楽。

閉じる：File → Close Workspace 開く：File → Open Workspace

各ワークスペースごとの設定はSettingを開いて Workspaceタブを選ぶ。あとはUser設定と同じ。

まずは仮想環境を作成する。（参考）[仮想環境](#_仮想環境)

mkdir **myworkspace** # 仮想環境用のディレクトリを新たに作る

python -m venv **myworkspace**/**myvenv** # 仮想環境を作成

補足

同ディレクトリに.pyファイルが存在し、VSCodeにpython拡張が入っている場合は、「仮想環境の作成を検知しました」といったメッセージが表示される。

次にワークスペースで使うインタープリタを指定する。上の様にSettingを開いても良いが、

.code-workspace の”settings” の部分を直接編集する。

{

（中略）

"settings": {

"python.pythonPath": "**myvenv**/bin/python", # python3の場合もある。ディレクトリを直に見て確認。

}

}

## その他基本情報

### Pythonにおける構造化

Pythonでは構造化プログラムも作成可能。以下にその区分単位をまとめる。

但し，Pythonでは構造化しなくても十分に実用に耐えうる場合もあり，構造化するかどうかは状況による。

#### method

ある最小単位の処理をまとめたもの。

例）「ただ1を加算するだけ」のmethod

def add\_one(arg):

arg += 1

#### class

複数のmethodなどを集め，１つの機能を管理しやすくしたもの。

例）「簡単な計算機能」をまとめたclass

class Clac():

def add\_one(arg): # 1加算するmethod

arg += 1

def minus\_one(arg): # 1減算するmethod

arg -= 1

... # 似たようなmethodが複数個ある

#### module

通常，１つの .py ファイルの事を指し，１つの機能を管理する為にの複数のclassなどを記述したもの。

他の構造化言語にはあまり見られない管理区分で，会社で例えるとclassが「課」でmoduleは「部」といった所。ただ，必ずしもclassよりもmoduleの方が大きな機能になる訳では無い。

（蛇足）C++などの他の構造化言語に慣れていると，しばしばclassでまとめるべきか，moduleでまとめるべきか迷う場面がある。

#### package

複数の同系のmoduleを集め，大きな機能を管理しやすくしたもの。

具体的には\_\_init\_\_.py を内部に持つディレクトリはpackageとみなされる。

Pythonの使用目的によっては，packageを作る機会は稀な事もある。

補足：python3.3以降は、\_\_init\_\_.pyが無くてもpackageとみなす

（参考）[package化 - ファイル構成](#_ファイル構成)

### アンダーバーについて

Pythonにおいてはアンダーバーが大きな意味を持つ点に留意する。

アンダーバー２つ。def \_\_init\_\_(): や\_\_some\_member など

module内のprivateなmemberを示す。

厳密にはClassName\_\_int\_valueといった形でアクセスできない事はない？？。（manglingと呼ばれる）

アンダーバー１つ。\_

1. 関数の返りがtupleの時，使用しない方の戻り値をアンダーバー１つで示す。こちらがメイン。

2. 慣例的にprivateメンバを指す。但し外部からの参照は可能。

詳細は[変数命名規則](#_変数命名規則)を参照。

### python Launcher

Windows向きのアプリで、複数のバージョンのpythonインタプリタの中から、適切なバージョンを選択してくれる。

実行ファイル名：py.exe

使い方

Command Prompt

py --help

（補足）仮想環境への理解があれば，python Launcherの出番はほとんど無い

### .pyiファイル

インターフェイスらしい。

（参考）<https://wingware.com/doc/edit/analysis-helping-pyi-files>

### .pycファイル

コンパイル済pythonファイル。.pyファイル実行の際，次回の実行の為に残されるバイナリファイル。

marshal と呼ばれるより原始的な直列化モジュールを用いているらしい

# プログラム情報

## 基本

### コーディング基本

・行内改行

＼（バックスラッシュ） ※＼が無くても問題なく動く改行も多数存在する

if self.\_\_baz is None \

or self.\_\_foo is None \

or self.\_\_bar is None:

### shegang

スクリプトファイルの１行目に記述され、インタプリタや文字コードの指定を行うもの。

例）

#!/usr/bin/env python

# coding: utf-8

インタプリタの指定方法は２通りある

|  |  |  |
| --- | --- | --- |
| 書き方 | !/usr/bin/env python! | !/usr/bin/ python |
| 意味 | PATHから実行パスを導く。 | pythonが実際にインストールされているディレクトリを明示的に指定している |
| メリット | 異なるシステム上でも動く |  |
| デメリット | コマンドライン引数としてオプションを渡せない。 |  |

### 四則演算

加算+　減算-　乗算\*　除算/　余%　累乗 \*\* [ e.g. 2\*\*3 ；結果：8 ]

文字列にも掛け算が使える e.g. ‘hoge’ \* 3 ；結果：’hogehogehoge’

（補足）\*\*はdict型の引数の明示にも用いられる　例：\*\*kwargs

### データ型

bool ：True 真(1 でも可；0以外はすべてTrue) Flase　偽(0 でも可)

string ： シングル、ダブル　どちらのクオーテーションでも可能

char ：python にはcharの概念はない。stringを配列としてとらえる。e.g. hoge\_str[1]

#### 数値

16進数 0x [e.g. 0x10] 8進数 0o [e.g. 0o10] 2進数　0b [e.g.0b10]

### 条件分岐

#### if

# 条件に応じた値の変更

val = 100 if django.VERSION < (3, 1) else 0

not キーワードに注意！

DEBUG = False

if not DEBUG:

print('True')

> True # Trueと表示される

#### switch

python にはswitchがない。elif で実装できるので、運営側に実装する気がない。

似たようなことをするには：

1.　elifを用いる

2.　Dictionary型をつかう　e,g, num\_to\_kanji = {1:"一", 2:"二", 3:"三", 4:"四"}

3.　in を用いる。 e.g. if str in {‘’a’, ’b’}　→ 結果：True or False

### ループ

pythonのループにはwhileとforがある。

### List系

### 文字列関連

#### エスケープ

raw\_string　=　r'docs\test.txt' # rでraw stringsになる。.NETの「@」のようなもの

raw\_string　=　'docs\\test.txt' # 「\」で「\」をエスケープする。

### プログラム一般

#### 関数（def）

関数を宣言するような形で書けば良い

def add(x,y):

print x+y

return 0

（詳細）

#### コマンドライン引数

[ test.py ]

import sys

args = sys.argv

print(args[0])

print(args[1])

terminal

python test.py hoge fuga

> hoge

> fuga

#### decorator

メソッドに追加の処理や変数などを付け加えるもの。

一番有名なものは@classmethod

class MyClass:

@classmethod

def count(cls):

return csl.items.count()

### Console（コマンドライン）系

#### print

コマンドラインに出力

#### input

コマンドラインからのデータ入力

### 比較的特殊なもの

#### 実行タイミング

module (.pyファイル) 内でclassもmethodも利用せず，直接書き込んだソースコードは，外部からの実行，またはimport した時に実行される。

また，インタプリタ言語である為にifなどの条件文も条件が一致した時のみ実行され，それ以外の部分で文法エラーなどを起こしていても，条件が変わるまでは問題なく実行される。

#### スライス

シーケンス（リスト、文字列、タプル、バイト列）の一部分を切り取ってコピーを返す。

[start:end:step] # [::-2] とstepだけ用いる事も可能

#### 属性（attribute)

\_\_main\_\_ 関数などの事

#### in [ リスト内判定]

bool 判定値 in list となる。文字列でも使える

a =‘python code example’

x = ‘python’ in a

#### with [ 境界内変数？]

with クラス.メソッド as 変数名 ※VBに似たような記述があったと思う。

C#のusingの様に、例外が発生しても閉じてくれるので、積極的に使用する。

with open("...") as f:

print(f.read())

asは必須ではない。以下は上と一緒になる

f = open("...")

with f:

print(f.read())

複数のwith を使う事もできる

with open("...") as f1, open("...") as f2:

※withに渡るのは\_\_enter\_\_と\_\_exit\_\_の2つのメソッドを持ったクラスのインスタンス。

#### インクリメンタルについて

python ではインクリメンタルの演算子が無い。

i++ # pythonではエラーとなる（javaやc#などでは使える）

i += 1 # pythonの場合

#### その他箇条書き

メソッドの呼び出しの再、引数が無い時は DoSomething() ではなく、DoSomething となる。

### その他

#### None

null値を示す。

null判別

str\_val = ‘’

if str\_val is not None:

print(str)

# if not str\_val: # こんな書き方を見たが何だったんだろう。20.07.21

#### del

リストから要素を削除

del リスト[インデックス]

del リスト[開始インデックス:終了インデックス]

#### yield

関数の処理を 一旦停止 し，値を返す。

呼び出し元で処理が終わると、また関数側に戻され、続きの処理が実行される。

for d in data: # 巨大なデータがあるとする

　　yield d #　一行ずつ返す。

#### Ellipsis

３点リーダ（…）の事。

「何らかの定義がなされているが、省略されている」の意。（比較）None

例：

import numpy

a = numpy.array([[1, 2],　[3, 4]]) # 2x2の多次元配列

a[0, **...**] # 0行目の任意の列の要素からなる配列

> array([1,2])

「：」との違いは多次元配列ではっきりとする。

import numpy

b = numpy.array([

[[1, 2], [3, 4]],

[[5, 6], [7, 8]]]) # 2x2x2の多次元配列

b[..., 0] # 第1, 2次元は任意、第3次元のインデックスは0 ---(1)

> array([[1, 3], [5, 7]])

（参考）<https://qiita.com/yubessy/items/cc1ca4dbc3161f84285e>

（公式）<https://docs.python.org/ja/3/library/constants.html#Ellipsis> あまり情報がない

### ファイル関連

#### csv

##### 概要

import csv

with open('csv\_test.csv', 'w', encoding='utf-8') as f: # with：C# で言うところの using

writer = csv.writer(f, lineterminator='\n') # 改行コード（\n）を指定しておく

writer.writerow([1,'スパム','500円'])

writer.writerow([2,'卵','168円'])

writer.writerow([3,'ベーコン','1,250円'])

with open('/usr/path/to/file.csv', encoding="utf-8") as f:

reader = csv.reader(f)

for row in reader:

print(row) # 行全体を取得

print(row[0]) # １列目の値を取得

##### reader メソッド

パラメータ

delimiter :str 改行コードを指定

doublequote :bool

lineterminator :str

lineterminator="\r\n"

quotechar :str

quotechar='"'

skipinitialspace :bool

skipinitialspace=True

encoding :str

encoding="utf-8"

names :List[str] ヘッダとなる文字列を指定する

reader = csv.reader(filename, names=['user\_id', 'name'])

次ページに続く

csv　続き

##### DictReader メソッド

辞書形式で読む。ヘッダのあるcsvだと特に便利。

（パラメータはreaderとほぼ同じ）

with open(full\_path, encoding="utf-8") as f:

reader = csv.DictReader(f)

for row in reader:

print(row['user\_id'])

##### その他の例

###### tryを介して返す例

def csvreader(self):

if not file\_path:

raise Exception(' file\_pathが指定されていません')

try:

with open(file\_path, encoding="utf-8") as f:

reader = csv.DictReader(f)

yield from reader # ここが理解しにくい部分

except:

raise

tuppleの場合

with open('test.csv', 'w', encoding='utf-8') as f:

writer = csv.writer(f, line)

for tup in tupple\_dict:

writer.writerow(tup)

## コーディング規則

### PEP8

PythonにはPEP8と呼ばれるコーディング規則が存在する。以下はそのまとめ。

（参考）<https://pep8-ja.readthedocs.io/ja/latest/> (日本語)

<https://legacy.python.org/dev/peps/pep-0008/> (Eng)

コーディング規約：<https://www.python.org/dev/peps/>

### 変数命名規則

|  |  |  |
| --- | --- | --- |
| 対象 | 規則 | 例 |
| package | 全小文字の短い名前　アンダースコア非推奨 | mypackage |
| モジュール | 全小文字の短い名前 アンダースコア可  C/C++で作られたpython用モジュールはアンダーバー１つで始まる。 | my\_module.py |
| クラス名  （例外含む） | camel case（アンダースコア無し）cap wordsとも  例外はサフィックスに -Errorをつける | MyFavoriteClass  CostomError（例外） |
| メソッド名 | lowercase にアンダースコア  propertyもメソッドとみなされる | my\_favorite\_method |
| 変数名 | 全小文字。メソッド名に準じる。  mixedCaseが使われているモジュールがあれば、それに合わせる。  module内部で使う変数アンダースコアで始める。特に公開メンバに対する内部変数の場合。  また，サブクラスでの名前衝突回避の為には先頭にアンダースコアを２つ付ける。（こちらの方が圧倒的に良く見かける。）  予約語と衝突する際には後ろにアンダースコアをつける | my\_favorite\_instance  \_my\_private\_instance  (python にprivateの概念はないが)  \_\_my\_cls\_member  base\_ |
| 定数 | 全大文字  但しpythonでは定数は嫌われる。settings.pyを用いる。 | MY\_FAVORITE\_CONST |
|  |  |  |

#### 簡単な命名規則の箇条書き：

・'l' (小文字のエル)、'O' (大文字のオー)、'I'(大文字のアイ)を１文字で使わない

・括弧の前後に空白は入れない。

・「基本として」カンマ/セミコロン/コロンの**直後**に空白を入れる。直前の方は入れない。

・代入演算子（= の事）を使う時に，値の整列の為でも余計な空白を入れない。

・二項演算子の場合には両側にスペースを１つ入れる。

・改行の際，二項演算子（+, - など）は行頭に置く。

・予約語などと衝突する場合には後ろにアンダーバーを用いる。勝手な造語やtypo（打ち間違い→わざとスペルミスした単語）などを使わない。

・インスタンスメソッドの第一引数：self

・classmethodの第一引数：cls ※クラス内のメソッドの意ではなく，@classmethodの事

・methodのデフォルトパラメータで=を使う際は，前後に空白を入れない。

・\_\_ で始まる特別な変数はdocstringの直後に置くべき。

#### pylint

pythonのコードチェッカーの1つ。後述するPEP8コーディング規則に則ってチェックをしてくれる。

PSコマンドプロンプト

pip install pylint

#### import

import math # 単体モジュールのimport。基本中の基本。

# この方法で読み込めるのは

# ・python の標準モジュール

# ・同階層のディレクトリにある .py ファイル（モジュール）やディレクトリ化されたpackage

#　・sys.pathに存在するもの

# また、基本的に１モジュールにつき１行使う事が推奨されている。

from math import pi, sqrt # 特定のメンバだけのimport

# math.pi でなくpiやsqrt　だけで使えるようになる

# また，同一モジュールからのimportは同じ行での記述が推奨されている

from my\_module import pi # 同名でのimportは上書きされ、後者のみが有効になる。注意！

import my\_modude as mod # アクセスする為の変数名を指定する。

# 以下はその他の方法

from my\_module import \* # 同ディレクトリにあるmy\_module.py から\_で始まるもの以外を読み込む。

# 未知の変数名が乱立し、管理しきれないので普通は行わない。

from . import file1 # package内参照。相対インポートと呼ばれる。あまり推奨されない。

from .. import file1 # 上階層のpackage参照1

# 親package参照2

import sys

sys.path.append('../') # 実行ディレクトリをルートとする という仕様がある為，あまり推奨されない。

import dir1.module1

複雑に見えるが、[ import ]でモジュールやpackageへのポインタ変数を作ると考えれば良い。

例えばfrom math import pi, sqrt ではpi 変数とsqrt変数を生成したととらえる。その為math.pi といったアクセス方法にはならない。

注意事項：

・各モジュールは、インタープリタの１セッションにつき１回だけimportされる。実行中にモジュールを修正した場合は、インタープリタの再起動が必要。（そんな例は稀だと思うが）

・python my\_module args と言った形で、モジュールをcmdなどから直接起動する事もできる。また、その際には\_\_name\_\_ に”\_\_main\_\_” が代入される。モジュール(.py) 側では、sys.argv[1] などとすると引数(args) を取得できる。sys.argv[0] は自身のファイルフルパスが格納されている。

また，importは、基本的に１行ずつ書く。但し同じモジュールからのimportはその限りではない。

import os # 良い例

import sys

import sys, os # 悪い例

importは，python標準ライブラリ，サードパーティ関連，自作またはアプリケーション特有

の順番にグループ化して記述する。

補足：

Pythonインタープリタはsys.module、標準ライブラリ、sys.pathの順にimportするらしい

attempted relative import with no known parent package

#### コメントアウト

# でコメントアウトできる。面倒だが，複数行のコメントアウトでも#を使う。

**「複数行はダブルクオーテーション3つ」 などと言う情報があるが大嘘**！。これはdocstring と呼ばれ、意味合いが全く異なる。

#### インデント

pythonは近年の言語同様，インデント位置が非常に重要な言語の内の１つ。

半角スペース４つを用いる。**TABを使わない！**

()などで折り返す時の例

# 開き括弧に揃える

foo = long\_function\_name(var\_one, var\_two,

var\_three, var\_four)

# 引数とそれ以外を区別するため、スペースを4つ(インデントをさらに)加える

def long\_function\_name(

var\_one, var\_two, var\_three,

var\_four):

print(var\_one)

# 突き出しインデントはインデントのレベルを深くする

foo = long\_function\_name(

var\_one, var\_two,

var\_three, var\_four)

命名規則の参考資料

<https://www.yoheim.net/blog.php?q=20160612>

・\_\_all\_\_, \_\_author\_\_のような二重アンダーバー変数は、from \_\_future\_\_ 以外の あらゆるimport文の前に置くべき。（？？？）

・docstringはダブルクォーテーションを用いる。（シングルでも可能だが、と言う意味）

・すべての行の長さを、最大79文字までに制限する。

・演算子は左端　※長年，演算子の直後に改行するスタイルが流行ったが、見にくいので使わない事！

良い例：

income = (gross\_wages

+ taxable\_interest

+ (dividends - qualified\_dividends)

・空白行

class 内のmethodは最低１行の空白行を入れる。グループの境目には２行入れても良い。

method内ではロジックの境目を示すために，空白行を控えめに使う。

・型ヒントのスタイル

コロンの後にスペースをひとつ入れる

良い例：

int\_val: int

### その他

・pythonにおいては本当の意味でprivateなものは存在しない為，non-publicとでも言うべき

・member の事をpythonでは「属性」と呼んでいる。

・raw文字列

path = r"C:\documents\nori\tama" # raw文字列 C系の @と同じ

・path

./ 現在のディレクトリ

../ 一つ上のディレクトリ

•命名規則という訳ではないが，importしたmoduleやmoduleやクラスと全く同じ名前の変数名を定義してしまうと，その名前はすべて変数名とみなされimport したmoduleとしてはみなされなくなる。

例）

import math

math = 'hoge'

print(math.pi) # ここで例外エラーが発生する

> **Exception has occurred: AttributeError**

> 'str' object has no attribute 'pi'

上の例は非常に分かりやすいかもしれないが，下の例，ソースの上の方で定義したクラス名と同じ名前のクラスのメンバ変数を使ってしまった場合などは分かり難いかもしれない。

例）

class TestClass:

str\_member = "hoge"

class TestClass2:

def \_\_init\_\_(**self**):

self.\_\_test\_class1 = TestClass.str\_member # ここで例外エラーが発生する

TestClass = 1 # このメンバを宣言した時点で「TestClass」はclassとは見做されなくなる。

test = TestClass2()

> **Exception has occurred: UnboundLocalError**

> local variable 'TestClass' referenced before assignment

⇒ こういった事が起こりうるので，きちんと命名規則を守る必要がある。

## 構造化

### 概要

他のオブジェクト指向の言語と少しイメージが異なる

package

複数のmoduleを持つ

module

複数のクラスを持つ。他の言語のclassの認識に近いかも。

class

DjangoのModelの様に、pythonではかなり体積の少ない区分となる？

### 注意事項

#### interface

pythonにinterfaceは存在しない。

抽象クラスは存在するのでそれで対応。

### class

#### Metaクラス

classの設定をする機能。

#### 特殊メソッド

##### \_\_init\_\_

クラスのコンストラクタ

##### \_\_str\_\_

.NETのtoString() の様な物。printなどで文字列に変換する場合に呼び出される。

##### \_\_repr\_\_

\_\_str\_\_ に似ているが、repr() を使った時の結果

##### \_\_hash\_\_

Hash値を返す。オブジェクトを示す、唯一無二、一意な値。

##### \_\_name\_\_

（呼び出し元）メソッドの名前

↓

ライブラリのモジュールの名を返す

？？試してみると、ファイルパスを名前空間にした文字列が得られるのだが？2022-05-09

例：some.directory.path.file\_name

##### \_\_dir\_\_

本インスタンスに対して dir() された時に呼び出される

##### \_\_bytes\_\_

文字通り。バイナリのバイトデータを返す。

##### \_\_bool\_\_

使う機会があるのだろうか？

##### \_\_enter\_\_

with 使用時によびだされる

（関連）\_\_exit\_\_ 終了時によびだされる

##### \_\_iter\_\_

イテレータ。これを実装しておくとfor ループで回す事ができる。

.NETのコレクション的な使用？

（関連）\_\_next\_\_

class MyIterator(object):

def \_\_init\_\_(self, \*numbers):

self.\_numbers = numbers # numberはintのtuppleまたはlist

self.\_i = 0 # ループカウンタ

def \_\_iter\_\_(self):

return self # \_\_next\_\_()はselfが実装してるのでそのままselfを返す

def \_\_next\_\_(self): # Python2だと next(self) で定義

if self.\_i == len(self.\_numbers):

raise StopIteration()

value = self.\_numbers[self.\_i]

self.\_i += 1

return value

my\_iterator = MyIterator(10, 20, 30)

for num in my\_iterator:

print('hello %d' % num)

##### \_\_call\_\_

インスタンスをメソッドのよう使用する際に実行される。

例

class CallTest:

def \_\_init\_\_(self, coefficient):

self.coefficient = coefficient

def \_\_call\_\_(self, variant):

return self.coefficient \* variant

test\_call\_5 = CallTest(5)

print(test\_call\_5(9))

> 45

##### \_\_add\_\_

+ で足し算された際に呼び出される。.NETのoperator+

def \_\_setattr\_\_(self, name, value): # = が使われた時に呼び出される。operator= みたいな

self.\_\_dict\_\_[name] = value

pass

def \_\_getattr\_\_(self, value): # \_\_setattr\_\_ のget版

pass

補足

setattrは以下のように使えるらしい。匿名クラスみたいな感じ？2022-08-08

class User():

pass

obj = User()

setattr(obj, "name", "Kuro") #　(インスタンス, メンバ名, 値)

print(obj.name)

.memberなどで追加しても良いが、setattrはdict型のループと相性が良い。

class User():

def \_\_init\_\_(self, attr\_dict):

for k, v in attr\_dict.items():

setattr(self, k, v)

### 継承

### 抽象クラス

from abc import ABCMeta, abstractmethod

class AbstractMyBase(metaclass=ABCMeta):

@abstractmethod

some\_method(self): # このメソッドは継承先で実装必須となる。

pass

not\_abstract\_method(self): # このメソッドは継承先で実装する必要がない。

pass # 但し、実装すると基クラスのメソッドがオーバーライドされる

### method

#### 基本

グローバル（モジュールに直接）書き込んだmethodの引数は自由だが、class 内メソッドの第一引数は必ず self

#### \*argsと\*\*kwargs

argはargument，ようするに引数の英単語の略。

また，引数名のまえに「\*」を追加する事で任意の数のパラメータを渡すことができる。

\*argsにはtupple型（の変数）を格納する (1, 2, 3, 4, 5)

\*\*kwargsにはdict型（の変数）を格納する {"a": 1, "b":2, "c":3}

両方設定する場合には「\*args」→「\*\*kwargs」の順番で定義しないとエラーが出るらしい。

補足：C言語では以下のような形が好まれる

int main(int argc, char \*argv[]){

}

// a.out 123 abc と実行する事で，argc=123，argv[2]=”abc” となる

#### 型ヒント

以下の様にする。注意点としては，型指定でなくただのヒント（情報）という点

def greeting(name: str) -> str:

return ’Hello’ + name

また、Listなどの特殊な型の場合はtypingモジュールを用いる

from typing import List

def some\_list () -> List[int]:

return [1,2,3,4]

自分で定義もできる

Vector = List[float]

def scale(scalar: float, vector: Vector) -> Vector:

##### typingモジュールの説明（抜粋）

Union：２つ以上の型が代入される可能性がある事を明示

arg:Union [int, None] # null 許容型のint と言う事

Optional：指定した型と、Noneの場合 ≒Union[type, None]

arg:Optional [int] # 同じく、null 許容型のint

Dict

decks: Dict[str, Any]

List

Any ：すべての型

返り値や，型指定のない引数はすべてAnyになる

特定の型に代入する時に型検査が行われないことに注意。

objectとの違い

#### static とclass method

|  |  |
| --- | --- |
| Staticmethodデコレータ | classmethodデコレータ |
| ・clsもselfも利用しない  ⇒ クラスメンバに依存しない処理である事の強調？  ・classmethodよりも処理が少し早い | ・第一引数にクラス本体としてclsを受け取る |

@staticmethod

def static\_method(): # static method

print('this is static method!') # インスタンスを使用しない（使用できない）メソッド

@classmethod

def \_\_mungled\_class\_method(cls): # マングリング化されたclass method

print('this is mungled class method...') #

return cls

#### docstring

クラスやメソッドの説明書き

class MyClass():

"""

このクラスの使い方を記述する

"""

def my\_method(arg1):

“””

このメソッドの説明

Parameters

--------------

arg：引数argの説明

Returns

--------------

戻り値の説明

“””

pass

以下の様にすると表示される

help(MyClass) # 全ての\_\_doc\_\_が表示される。

print(MyClass.\_\_doc\_\_) #

print(MyClass.my\_method\_\_doc\_\_) #

注意：Web資料では「複数行コメント」などと表現しているものがあるが、厳密には間違い。

### パッケージ化

#### 概要

##### ファイル構成

parent\_dir

├─ docs

├─ tests

│　 ├ \_\_init\_\_.py # ここに \_\_init\_\_.pyは要らない気がするが？

│　 ├ test\_advanced.py # 各種テストを1つの.pyファイルにまとめる。

│　 ├ test\_basic.py # 他人に使ってもらう時のサンプルソースにもなりうる。

│　 └ test\_core.py

├─ my\_package

│ 　├ \_\_pychache\_\_ # 実行時に自動的に作成される。次回の為のキャッシュ。無視して構わない。

│ 　├ \_\_init\_\_.py # 特殊モジュール \_\_init\_\_.py

│　 └ main.py #　本処理を記述するが，名前は別になんでも良い。

│

├─ requirements.txt # 必要package等の情報を記述する。

└─ setup.py # packageの依存情報、バージョン情報、package名を書く。

ポイント

・package名はPEP8に準じ、すべて小文字にしてアンダーバーはできるだけ使わない。

・ディレクトリは名前空間とほぼ同じ。実際に使うときは my\_package.main として使う

###### 特殊モジュール

\_\_init\_\_.py

このファイルが存在するディレクトリをPythonにpackageとして認識させる。

import [ディレクトリ名] でpackageを使用できる。なお、このファイルは空でも良いが、

使用先でimport された時に\_\_init\_\_.pyに記述されたスクリプトが実行される。

なお，Python3.3以降ではディレクトリであれば自動的にpackageとして認識するようになった。

但し，モジュールをimportした時に何か処理が必要な場合は使っても良い。

（補足）一部情報で「\_\_init\_\_,pyマクロ」と呼ばれているが，公式の情報では\_\_init\_\_,py file とされている。

###### tests

testディレクトリ

自分で考えたテストケースを必ず書くようにする。モジュールと対になるようにファイルを作成。

unittest と言う、標準のフレームワークがある。

<https://futurismo.biz/archives/4395/>　；　<https://pymotw.com/2/unittest/>

通常pythonで自身より上（親）のディレクトリを参照する事は無いが，packageのディレクトリとtestsディレクトリは同レベルの深さにあるので，一度親のパスを経由する必要がある。

ディレクトリ構成：

parent\_dir

├─ tests

├─ my\_package

…

以下略

その時のやり方：

import sys

import os

sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..')))

最後の行は少し複雑なので，分けて考える：

os.path.dirname(\_\_file\_\_) 今のファイルのディレクトリ名　\_\_file\_\_ は現ファイルのフルパス。

os.path.join はpath表現の文字列を少し簡単に作れる，と言うだけ

⇒ つまり，親ディレクトリを表す文字列 ’C:\current\_dir\_path\..’ を作っているに過ぎない。

更に sys.path　はインタープリターにモジュール置き場のディレクトリを追加指定する為のもので，insertは文字通り，そのモジュール置き場情報を追加する。

#### \_\_init\_\_.pyの書き方

├ children/

│ ├ \_\_init\_\_.py

│ ├ a.py

├ main.py

a.py

def hello():

print("hello from a.py")

パターン１

children/\_\_init\_\_.py

from . Import a

main.py

import children

children.a.hello()

パターン２（非推奨）

children/\_\_init\_\_.py

from .a Import hello as a\_hello

main.py

from children import \*

a\_hello

##### クラス

###### 継承

※詳細は書ききれないので，概要だけ：

class BaseTestClass(): #基底クラス（親クラス，スーパークラス）

\_\_str\_val = 'base!'

class InheritTestClass(BaseTestClass): # 継承クラス（子クラス，サブクラス）

def \_\_init\_\_(self, int\_val, str\_val):

super().\_\_init\_\_(int\_val, str\_val) # スーパークラスのinitialiser を呼び忘れないように

self.\_\_str\_val = 'super!' # これは本クラスの値を変更するだけ

参考になった資料：

<https://qiita.com/Kensuke-Mitsuzawa/items/7717f823df5a30c27077>

## 応用系

### lambda（ラムダ）

lambda\_simple = lambda n： n \* 2 # 引数有り

lambda\_no\_arg = lambda： print(‘fuga’) # 引数無し

lambda\_2args = lambda x,y： x \* y # 複数の引数

複数行書く事も可能だが、pythonのlambdaは基本的に短文を書くように設計されている。

lambda\_multi\_rows = lambda : [ # この様な使い方はするべきではない

print('hoge'),

print('fuga')

]

### generic

from typing import TypeVar, Generic

T = TypeVar('T') # 型変数を宣言する（変数名は何でも良い？）

class MyContainer(Generic[T]): # 編集中2022-03-15　結局Genericは使わなかった

pass

# インスタンス生成共有

container = MyContainer[int]() # int型のMyContainerを生成

djangoの例（django.db.models.manager.py）

\_T = TypeVar("\_T", bound=Model, covariant=True)

\_M = TypeVar("\_M", bound="BaseManager")

class BaseManager(Generic[\_T]):

### unittest

#### 基本

テストの自動化や報告機能などを持ったクラス。これを用いる事でテストを簡易化できる。

ファイル構成

parent\_dir

├ target\_module.py テスト対象のメソッドが書かれたmodule

├ test\_module.py 実際にテストを行うmodule

target.py

def add(arg1, arg2):

return arg1 + arg2

test\_module.py

import unittest

import target\_module

class TestAdd(unittest.TestCase): # unitest.TestCaseクラスを継承

def test\_add(self): # 実際のテストランナーとなるmethod

actual = target\_module.add(1, 1)

self.assertEqual(actual, 2) # methodの実行結果が2と等しいか確認

実行方法１：コマンドライン

python -m unittest test\_module.py # module 指定

python -m unittest test\_module.TestAdd.test\_add # method 指定

python -m unittest -v test\_module.py # 詳細表示オプション -v (verbose)

実効方法２：unittest.main()

# TestAdd クラスのどこかに以下を追加

if \_\_name\_\_ == '\_\_main\_\_': # 直接実行された場合は　という意味

unittest.main()

>ts = TestAdd() # これだけで実行される。

（公式）<https://docs.python.org/3/library/unittest.html>

#### メモ

こんな書き方もある

def test\_even(self):

for i in range(0, 6):

with self.subTest(i=i):

self.assertEqual(i % 2, 0)

#### ライブラリ（unittest）

setUp

テストメソッドの直前に呼び出される。

subTest

subTest(msg=None,\*\*params)

msg サブテストが失敗した際の表示メッセージ

params サブテストが失敗の際に，使われたパラメータを結果に表示させる為に与える引数。

tearDown

テストメソッドが実行され、結果が記録された直後に呼び出される。

setUpClass()

個々のクラス内のテストメソッドが実行される前に呼び出される

@classmethod

def setUp(cls):

cls=MyClass() # インスタンス生成する必要があるらしい

...

tearDownClass

個々のクラス内のテストメソッドが実行された後に呼び出される。使い方はsetUpClassに同じ。

##### assertメソッド

|  |  |  |  |
| --- | --- | --- | --- |
| メソッド | 確認事項 | ver. | 備考 |
| assertEqual(a, b) | a == b |  |  |
| assertNotEqual(a, b) | a != b |  |  |
| assertTrue(x) | bool(x) is True |  |  |
| assertFalse(x) | bool(x) is False |  |  |
| assertIs(a, b) | a is b | 3.1 |  |
| assertIsNot(a, b) | a is not b | 3.1 |  |
| assertIsNone(x) | x is None | 3.1 |  |
| assertIsNotNone(x) | x is not None | 3.1 |  |
| assertIn(a, b) | a in b | 3.1 |  |
| assertNotIn(a, b) | a not in b | 3.1 |  |
| assertIsInstance(a, b) | isinstance(a, b) | 3.2 |  |
| assertNotIsInstance(a, b) | not isinstance(a, b) | 3.2 |  |

### assert

条件が満たされないときに例外をraiseする。

用法

assert 条件式, 条件式がFalseの場合に出力するメッセージ

具体例

assert tf.\_\_version\_\_.startswith('2') # tensorflowより

# 標準ライブラリ

## 概要

公式資料：<https://docs.python.org/ja/3/library/index.html>

## importなしで使えるもの

### print

#### 同じ行に表示

例：

list = ['Alice', 'Bob', 'Charlie']

for i, name in enumerate(list):

# 重い処理があるとする

print('\rprocess-done, %i - %s' % (i, name), end='')

## argparse

### 概要

pythonコマンド実行時の引数を管理する。

### 基本

[ main.py ]

import argparse

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser()

parser.add\_argument('model') # 最初のコマンドライン引数を追加

parser.add\_argument('label') # ２つ目のコマンドライン引数を追加

parser.add\_argument('--number', type=int, default=3) # オプション引数を追加

args = parser.parse\_args()

print('model:%s label:%s number:%s' % (args.model, args.label, args.number))

[ terminal ]

python3 main.py hoge fuga --number 2

>> model:hoge label:fuga number:2

### 詳細

parser = argparse.ArgumentParser(description='説明')

parser.add\_argument('arg', help='説明') # 説明の文字列を指定

>> python3 main.py –h # このようにすると表示される

parser.add\_argument('arg', type=str) # データ型の指定

parser.add\_argument('arg', nargs='\*') # 複数の引数の指定

parser.add\_argument('arg', nargs='+') # 複数の引数の指定（１つ以上強制）

parser.add\_argument('--option') # オプション引数

parser.add\_argument('-o', '--option') #　省略形オプション

## ctypes

PythonからC/C++の共有ライブラリ（いわゆる.dllとか.soとか）を呼び出すための方法の一つ

（比較）SWIG

## calendar

HTML形式のカレンダーなどを扱う

## functools

deprecatedデコレータ

@deprecated

def some\_old\_function(x, y):

return x + y

2022-06-23現在、使い方が良く分からない。

## exception

### 概要

### 多用されるもの

KeyboardInterrupt Ctrl+C によるブレイクの事。よく下の様に用いられる。

while:

try:

# 処理

except KeyboardInterrupt:

break

（参考）<https://docs.python.org/ja/3/library/exceptions.html>

NotImplementedError 未実装メソッド。実際にはpassを使うかこちらを使うか悩ましいところ。

FileNotFoundError 要求されたファイルやディレクトリが存在しない

OSError システム関連のエラー

TypeError 適切ではない型で演算を行った。

ValueError 引数の型はあっているけれど誤った値を取っている場合

def read\_analog(ch):

if ch < 0 or ch > 3:

raise ValueError("range of ch must be in [0-3]")

ZeroDivisionError ゼロによる除算。ArithmeticErrorを基底クラスにしている

ImportError

例（Djangoより抜粋）

try:

from django.core.management import execute\_from\_command\_line

except ImportError as exc:

raise ImportError(

"Couldn't import Django. Are you sure it's installed and "

"available on your PYTHONPATH environment variable? Did you "

"forget to activate a virtual environment?"

) from exc

### 継承マップ

BaseException

├─ SystemExit

├─ KeyboardInterrupt

├─ GeneratorExit

├┬ Exception

│├─ StopIteration

│├─ StopAsyncIteration

│├─ ArithmeticError

| ├─ FloatingPointError

| ├─ OverflowError

| ├─ ZeroDivisionError

├─ AssertionError

├─ AttributeError

├─ BufferError

├─ EOFError

├─ ImportError

| ├─ ModuleNotFoundError

├─ LookupError

| ├─ IndexError

| ├─ KeyError

├─ MemoryError

├─ NameError

| ├─ UnboundLocalError

+-- SystemError

+-- TypeError

|

次ページに続く

+-- OSError

| +-- BlockingIOError

| +-- ChildProcessError

| +-- ConnectionError

| | +-- BrokenPipeError

| | +-- ConnectionAbortedError

| | +-- ConnectionRefusedError

| | +-- ConnectionResetError

| +-- FileExistsError

| +-- FileNotFoundError

| +-- InterruptedError

| +-- IsADirectoryError

| +-- NotADirectoryError

| +-- PermissionError

| +-- ProcessLookupError

| +-- TimeoutError

+-- ReferenceError

+-- RuntimeError

| +-- NotImplementedError

| +-- RecursionError

+-- SyntaxError

| +-- IndentationError

| +-- TabError

+-- ValueError

| +-- UnicodeError

| +-- UnicodeDecodeError

| +-- UnicodeEncodeError

| +-- UnicodeTranslateError

+-- Warning

+-- DeprecationWarning

+-- PendingDeprecationWarning

+-- RuntimeWarning

+-- SyntaxWarning

+-- UserWarning

+-- FutureWarning

+-- ImportWarning

+-- UnicodeWarning

+-- BytesWarning

+-- ResourceWarning

## os

### environ

#### 概要

環境変数を表すマップ型オブジェクト

#### setdefault

フルネーム：os.environ.setdefault

環境変数に値が存在しない場合のみ値を入れる。

## logging

### 概要

文字通りログを表示，記録する機能を提供する。

print()などもログを残す手段としては有効だが，積極的にloggerを使うべき。

理由：

明確にログ文字列であるとコード作成者の意図を明示できる為。

python標準の方式に倣う事で他の人にも理解してもらいやすくする。

### 基本情報

dictConfig形式とfileConfig形式がある。dictConfigの方が新しい。

ログレベル

|  |  |  |  |
| --- | --- | --- | --- |
| 列挙型名 | レベル | メソッド名 | 詳細 |
| NOTSET | 0 | ? | 設定値などの記録（全ての記録） |
| DEBUG | 10 | debug | 動作確認などデバッグの記録 |
| INFO | 20 | info | 想定通りの正常動作の記録 |
| WARNING | 30 | warning | 想定外で，問題に関連する可能性がある現象の記録。(def.) |
| ERROR | 40 | error | エラーなど重大な問題 |
| CRITICAL | 50 | critical | プログラムが実行を続けられない致命的な問題 |

### チュートリアル

import logging

# WARNING以上を含むログをテキストファイルに出力する（デフォルトではコンソール表示）

logging.basicConfig(filename='test.log', level=logging.WARNING)

# ログの出力

logging.warning('Watch out!')

logging.info('I told you so')

情報

公式チュートリアル：<https://docs.python.org/ja/3.10/howto/logging.html#logging-basic-tutorial>

### レファレンス

#### ロガー設定

##### 概要

|  |  |
| --- | --- |
|  |  |
| logging.config.fileConfig | 比較的古い |
| logging.basicConfig | 簡単な設定方式？ |
| logging.config.dictConfig | ver3.2で追加。高度だが、汎用性の高い設定方式？ |

##### fileConfig

[ logging.config.fileConfig ]

# logger 用の設定ファイル。iniファイル形式

[loggers]

keys=root

[handlers]

keys=consoleHandler,fileHandler

[formatters]

keys=f1

[logger\_root]

level=WARNING

handlers=fileHandler,consoleHandler

[handler\_consoleHandler]

class=StreamHandler

formatter=f1

args=(sys.stdout,)

[handler\_fileHandler]

class=FileHandler

formatter=f1

args=('/home/keisukeo/test.log',)

[formatter\_f1]

format=[%(asctime)s]%(filename)s(%(lineno)d): %(message)s

datefmt=%Y-%m-%d %H:%M:%S

##### basicConfig

構文：

引数

filename

level

filemode python標準のopenメソッドと同じ方式。’r’、 ’w’ など。

format 実際の文字列の記述書式を指定する。

注意点

debug() や info() などの書込みメソッドよりも先にbasicConfigメソッドを呼び出す必要がある。

logging.basicConfig(filename='test.log', level=logging.DEBUG)

補足

・Pythonの昔のバージョン(e.g. ver3.11など)ではbasicConfigメソッドにencodingという引数が存在していた。

・

##### dictConfig

例：（Djangoより）

DEFAULT\_LOGGING = { # このDEFAULT\_LOGGINGという変数はdict形式の文字列になっている

'version': 1,

# ...略

}

logging.config.dictConfig(DEFAULT\_LOGGING)

version 常に1

disable\_existing\_loggers 既存のログ設定を無効化するかどうか

formatters ログフォーマットを記述。以下のように'handlers'内で文字列指定する

'formatters': {

'basic': {

'format': '[{server\_time}] {message}'

},

},

'handlers': {

'file': {

#...中略

'formatter' : 'basic'

}

}

handlers

class logging.Handlerクラスを継承したクラス名を文字列指定

level

filters

filename 保存するファイル名を相対パスで指定（ルートはどこに？2022-05-09）

formatter 先に定義したformatterの名前を文字列指定

loggers

handlers

level

propagate ログ出力イベントを上位のロガーハンドラにも渡すかどうか。def=True

次ページに続く

ファイルに保存する場合

例：Djangoから

[ settings.py ]

LOG\_DIR = os.path.join(BASE\_DIR, 'log') # ログ保存ディレクトリ

LOGGING = {

'version': 1, # 1 固定

'disable\_existing\_loggers': False, # 既存のログ設定を無効化しない

'formatters': {

'production': {

'format': '%(asctime)s %(message)s'

},

},

'handlers': {

'file': {

'level': 'INFO',

'class': 'logging.**FileHandler**',

'filename': os.path.join(BASE\_DIR, 'log' ,"info.log"),

'formatter': 'production',

},

},

}

補足：年、月でログのディレクトリを作成する場合

LOG\_DIR = os.path.join(BASE\_DIR, 'log',

today.strftime('%Y'),

today.strftime('%m')) # ログ保存ディレクトリ

if not os.path.exists(LOG\_DIR): # ディレクトリが無いとエラーになる為作成。

os.mkdir(LOG\_DIR) # （月ごとに作成する必要がある為、自動化が望ましい）

[ views.py ]

#### getLoggerメソッド

loggerオブジェクトを返すコンストラクタのような役割。getLogger(name)のnameが変わらない限り、１度だけ呼び出すのが基本となるため、各モジュールファイル（.py）の先頭に、以下の様に記述するのが基本。

import logging

logger = logging.getLogger(\_\_name\_\_)

例：（Djangoより）

DEFAULT\_LOGGING = {

'version': 1,

# ...中略

'handlers': {

'console': { 略 }

'mail\_admins': { 略 }

'file': { 略 }

'loggers': {

'': { # 下のdjango以外が対象（Djangoの自作アプリなど）

'handlers': ['file'],

'level': 'INFO',

'propagate': False,

},

'django': { # djangoフレームワークが残すログ（デバッグ時）

'handlers': ['console', 'mail\_admins'],

'level': 'INFO',

},

},

}

この時、django.pyが残すloggingはコンソール画面に表示され、それ以外のloggingはファイルに保存される事になる。業務

#### LogRecord属性

|  |  |  |
| --- | --- | --- |
| 属性名 | 書式 | 説明 |
| asctime | %(asctime)s | LogRecord が生成された時刻を人間が読める書式で表したもの。デフォルトでは "2003-07-08 16:49:45,896" 形式 (コンマ以降の数字は時刻のミリ秒部分) です。 |
| created | %(created)f | LogRecord が生成された時刻 (time.time() によって返される形式で)。 |
| ファイル名 | %(filename)s | pathname のファイル名部分。 |
| funcName | %(funcName)s | ロギングの呼び出しを含む関数の名前。 |
| levelname | %(levelname)s | メッセージのための文字のロギングレベル ('DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL')。 |
| levelno | %(levelno)s | メッセージのための数値のロギングレベル (DEBUG, INFO, WARNING, ERROR, CRITICAL)。 |
| lineno | %(lineno)d | ロギングの呼び出しが発せられたソース行番号 (利用できる場合のみ)。 |
| message | %(message)s | msg % args として求められた、ログメッセージ。 Formatter.format() が呼び出されたときに設定されます。 |
| module | %(module)s | モジュール (filename の名前部分)。 |
| msecs | %(msecs)d | LogRecord が生成された時刻のミリ秒部分。 |
| name | %(name)s | ロギングに使われたロガーの名前。 |
| pathname | %(pathname)s | ロギングの呼び出しが発せられたファイルの完全なパス名 (利用できる場合のみ)。 |
| process | %(process)d | プロセス ID (利用可能な場合のみ)。 |
| processName | %(processName)s | プロセス名 (利用可能な場合のみ)。 |
| relativeCreated | %(relativeCreated)d | logging モジュールが読み込まれた時刻に対する、LogRecord が生成された時刻を、ミリ秒で表したもの。 |
| thread | %(thread)d | スレッド ID (利用可能な場合のみ)。 |
| threadName | %(threadName)s | スレッド名 (利用可能な場合のみ)。 |

例

logging.basicConfig(

format="%(asctime)s:LINE[%(lineno)s] %(levelname)s %(message)s"

)

公式資料：<https://docs.python.org/ja/3/library/logging.html#logrecord-objects>

#### Handlerクラス

##### 概要

継承する事で、独自の

設定ファイルの'handlers'で'class'として指定する。

例：（Djangoのdjango\util\log.pyより）

DEFAULT\_LOGGING = {

#...中略

'handlers': {

'mail\_admins': {

'level': 'ERROR',

'filters': ['require\_debug\_false'],

'class': 'django.utils.log.AdminEmailHandler' # 補足：このファイルはdjango\util\log.py

},

#...中略

}

class AdminEmailHandler(logging.Handler):

#...略

##### StreamHandler

logging コアパッケージに含まれるpythonのデフォルトのログハンドラ。

ログ出力を sys.stdout（標準出力）, sys.stderr（標準エラー）,ファイルなどのストリームに送信。

（公式資料）<https://docs.python.org/ja/3/library/logging.handlers.html>

## list

test.list()

引数が無いとnull（空白）を削除する？

## datetime

### medelta

+ datetime.timedelta(days=1)

### strftime

オブジェクトを与えられた書式に従って文字列に変換。

※date型やtime型にも存在する。

### strptime

指定された対応する書式で文字列を構文解析して datetime オブジェクトにする

※datetime型にしか存在しない。

（参考）<https://docs.python.org/ja/3/library/datetime.html#datetime.datetime.strptime>

### strftimeとstrptimeで用いるフォーマット文字

%Y 年

%m 月

%d 0埋めした日にち

%H 時間

%M 分

%S 秒

（参考）<https://docs.python.org/ja/3/library/datetime.html#strftime-and-strptime-format-codes>

### date()

datetime型からdate型へ変換する。

## decimal

### getcontextメソッド

# Decimalクラスのコンストラクタにfloatが入力された際はFloatOperationをraiseする

getcontext().traps[FloatOperation] = True

### Decimalクラス

from decimal import Decimal

val1 = Decimal('0.1')

val2 = Decimal('0.1')

print('val1:%s\tval2:%s\tresult:%s' % (val1, val2, val1 \* val2))

注意点

・Decimalのコンストラクタにfloatを入れてしまうと下記のような結果になり、意味が無い。

例：0.1000000000000000055511151231257827021181583404541015625

⇒ 対策： **str（文字列）で代入**する。

・Decimal同士の掛け算は桁数まで乗算されてしまう。

例：0.10×0.10=0.100

⇒ 対策： strの書式で桁数を指定する

from decimal import Decimal

val = Decimal('0.100')

print('custom digit:%.2f' % (val)) # 小数点第二位まで表示

⇒ 対策：またはDecimal.quantize（推奨）やroundを使う。

from decimal import Decimal, ROUND\_HALF\_UP

Decimal('0.0100').quantize(Decimal('0.01'), ROUND\_HALF\_UP)

## hashlib

パスワードなどのハッシュ化を提供する

## http

### server

ローカルサーバーを立ち上げる

基本

ディレクトリ構成

/ PWD（現在のディレクトリ）

├ index.html 同レベルにindex.htmlを配置する

[ python ]

python -m http.server # デフォルトでは8000番ポートが使用される

python -m http.server 8080 # 8080番ポートを使用

補足

python2系ではSimpleHTTPServerというクラス名だった。

## json

### 概要

文字通り、json文字列を扱う。基本的にはdict型に変換してくれる。

import json

json\_file = open('static\json\project\_number\_category.json', 'r', encoding="utf-8")

json\_data = json.load(json\_file)

### バイナリデータの取り扱い

import json

import base64

DataFrame？

## math

.e 自然対数の底（ネイピア数）を返すプロパティ（関連）numpy.e

exp() 指数関数（自然指数関数）:

factorial() x の階乗を整数で返します

（公式）<https://docs.python.org/ja/3.7/library/math.html>

## pickle

基本

import pickle

sample\_list = [1,2,3]

f = open('sample.textfile','w')

pickle.dump(sample\_list,f)

f.close

## re

正規表現を扱う

result = re.search()

## socket

### PC名の取得

import socket

host = socket.gethostname()

print(host)

## subprocess

### run

### PIPE

以下のようにして使う（らしい）

proc = subprocess.run(".test.sh", shell=True, stdout=PIPE, stderr=PIPE, text=True)

## sys

### modules

import によって読み込まれたモジュールの情報を格納する（dict型？）

# 今まで読み込んだモジュールのリストを表示する

print('\n'.join([repr(n) for n in sys.modules.items()]))

（情報もと）<https://qiita.com/katsuko0303/items/6e1cbf78e7fa9a2b668c>

### threading

## typing

### Optional

### cast

型変換を行う

例：Optional型からintへの変換

a: Optional[int] = 0

a = cast(int,a)

## unittest

### mock

from unittest import mock

#### patchデコレータ

from unittest.mock import patch

@patch('module.ClassName2')

def test(MockClass1, MockClass2)

## 例外

（公式）<https://docs.python.org/ja/3/library/exceptions.html>

警告

DeprecationWarningとPendingDeprecationWarning、およびImportWarningは無視される点に注意。

表示させるには以下を実行

warnings.simplefilter('always', DeprecationWarning)

（補足資料）<https://docs.python.org/ja/3/library/warnings.html#the-warnings-filter>

（公式資料）<https://docs.python.org/ja/3/library/exceptions.html#warnings>

# パッケージ

## Python用

### six

python2系と3系の互換性の為のライブラリ

## 計算

### numpy

概要

内部はC言語（又はFortrun）で実装されている為、通常の

連番を作成

import numpy as np

np.arange(5)

#### 特徴

ブロードキャスト

#### ndarray

#### array

#### argmax

配列の最大要素のインデックスを返す

#### cov

分散・共分散（2つの変数の関係の強さを表す指標）行列を返す。

となり、この行列は [ Xの（標本）分散，XYの共分散，XYの共分散，Yの（標本）分散 ] となる

構文

np.cov(

m,

y=None,

rowvar=True,

bias=False,

ddof=None, =0：標本分散共分散行列；=1：不偏分散共分散行列

fweights=None,

aweights=None,

dtype=None)[source]

#### dot

行列の内積（行列同士の積）を返す。列数と行数（≒要素数）が一致していないとエラー。

import numpy as np

a = np.array([1,2,3], dtype=np.float64)

b = np.array([10,100,1000], dtype=np.float64)

np.dot(a, b) # 1×10+2×100+3×1000

> 3210.0

#### expand\_dims

新たな次元を追加する

公式：[https://numpy.org/doc/stable/reference/generated/numpy.expand\_dims.html#](https://numpy.org/doc/stable/reference/generated/numpy.expand_dims.html)

#### hstack

#### linalg

.norm()ベクトルの長さ

#### linspace

要素数を指定して等差数列を生成する（比較）arrange

t = np.linspace(-6, 6, 100)

#### random

permutation

https://numpy.org/doc/stable/reference/random/generated/numpy.random.permutation.html

#### std

標準偏差を求める

構文：np.std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>)

#### subplot

サブ領域を作成＋指定。これにより複数のグラフを描写する事ができる。

引数

args（数値引数）

行、列、index：指定する領域(plot)番号。

def.=(1,1,1)

indexは

例

plt.subplot(1,2,1) # １行、２列に分けた領域の内、1番目のサブ領域を指定

plt.subplot(121) # 3桁の数値で表現する事も出来る

（公式）<https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html>

transpose

転置行列を返す。

#### 具体例

縦400.横400、3色（3ch）の画像配列（画像データ）を作成

import numpy as np

img = np.zeros((400,400,3), np.uint8)

### scipy

stats

gmean 相乗平均を求める

#### special

from scipy import special

##### erf

誤差関数

##### comb

二項係数

#### integrate

積分関連を扱う？

## 文字列系

unicodedata

例

import unicodedata

text = '①：１２３㌔バイトのﾃｷｽﾄﾃﾞｰﾀを，ＣＬＥＡＮＳingする。'

print("Before:", text)

text = unicodedata.normalize('NFKC', text)

print("After: ", text)

# => Before: ①：１２３㌔バイトのﾃｷｽﾄﾃﾞｰﾀを，ＣＬＥＡＮＳingする。

# => After: 1:123キロバイトのテキストデータを,CLEANSingする。

（公式）<https://docs.python.org/ja/3/library/unicodedata.html>

（関連）jaconv（ジャパニーズコンバーター）、mojimoji（モジモジ）、str.maketrans() と str.translate()

### normalize

構文：unicodedata.normalize(form, unistr)

form の有効な値は、'NFC'、'NFKC'、'NFD'、'NFKD'

NFKD 正規形 KD

NFKC 正規形 KC

D = Decomposition （分解）、C = Composition （合成）、K = Compatibility（互換性）

（参考）<http://nomenclator.la.coocan.jp/unicode/normalization.htm>

### mojimoji

全角　半角など

## ドキュメント関連

### Excel

#### 概要

xlwings Python 3.6以上で動作。ExcelがインストールされたOS上で実行する必要がある

|  |  |  |  |
| --- | --- | --- | --- |
|  | openpyxl | xlwings |  |
| Python ver | 3.6以上 | 3.6以上 |  |
| マクロ | 不可 | 可能 |  |
| Excel | 不要 | 必要 |  |
| xlsファイル |  |  |  |
| 処理速度 | 遅め | 早い |  |
|  |  |  |  |
|  |  |  |  |
| 備考 |  | 内部でpywin32を使用  （Windows APIを呼び出す為、winOSである必要がある） |  |

#### openpyxl

##### 概要

Excelファイルの書式を保ったまま編集ができる。情報も一番多い。

注意点

・図形や写真のデータを扱う事ができず、openpyxlでブックを保存すると図形や写真が消えてしまう。

・画像データを扱う際はPilllowをインストールする必要がある。

##### チュートリアル

単純に指定のセルに値を入れて保存するまで

# -\*- coding: utf-8 -\*-

import openpyxl

FILE\_PATH=r'docs\test.xlsx'

workbook = openpyxl.load\_workbook(FILE\_PATH) # 補足 openpyxlは\_\_enter\_\_が無い→with不可

sheet = workbook.worksheets[0]

sheet['A3'] = 789

sheet.cell(row=1, column=1, value="'123") # セル番号は1から始まる

sheet.cell(row=1, column=2).value='456' # これでも良い

workbook.save(FILE\_PATH)

workbook.close()

Web上では文字列でシートを取得している例が見られるが、workbook.worksheetsはdictでなくlistなので、文字列で指定する事はできないはず。その為、以下の様にする：

index = workbook.sheetnames.index('Sheet1')

sheet = workbook.worksheets[index]

印刷範囲

<https://openpyxl.readthedocs.io/en/stable/print_settings.html>

Value must be one of {‘thin’, ‘dashed’, ‘mediumDashDot’, ‘dashDotDot’, ‘hair’, ‘dotted’, ‘mediumDashDotDot’, ‘medium’, ‘double’, ‘dashDot’, ‘slantDashDot’, ‘thick’, ‘mediumDashed’}

##### リファレンス

sheetnames

指定のワークブックに含まれるシート名のlistを返す。

（関連）get\_sheet\_names()【廃止予定】

##### 資料

（公式）<https://openpyxl.readthedocs.io/en/stable/>

#### pywin32

#### xlsxwriter

pandasに含まれる。

#### xlwt／xlrd

##### 概要

xlsファイルを扱う事ができる。（openpyxlでは扱えない）

xlwt 書き込み

xlrd 読み込み

### PDF

#### 概要

以下のライブラリが有名

pdf作成

・reportlab

・pandasを使う

pdf読み込み

・PyPDF2

#### reportlab

##### 概要

ReportLab PLUS というものがあり、そちらは有料

年間約2.2万円（1375GBP）

日本語表示にはフォント

無料のフォントを探してくるのが良い。<http://jikasei.me/font/genshin/>

from reportlab.pdfbase.pdfmetrics import registerFont

from reportlab.pdfbase.ttfonts import TTFont

pdf=canvas.Canvas('docs/pdf/test2.pdf')

registerFont(TTFont('梅明朝',　'./ume-tmo3.ttf'))

pdf.setFont('梅明朝', 12)

pdf.drawString(0.5\*cm, 25\*cm, 'あああ')

公式：<https://docs.reportlab.com/>

公式チュートリアル： <https://docs.reportlab.com/reportlab/userguide/ch1_intro/>

##### コンストラクタ

例

from reportlab.pdfgen import canvas

pdf=canvas.Canvas('./pdf/test2.pdf')

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ名 | 概要 | 初期値 | 備考 |
| filename |  | 入力必須 |  |
| pagesize= |  | None |  |
| bottomup | 下から上へ描写 | 1(True) |  |
| pageCompression |  | None |  |
| invariant |  | None |  |
| verbosity |  | 0 |  |
| encrypt |  | None |  |
| cropMarks |  | None |  |
| pdfVersion |  | None |  |
| enforceColorSpace |  | None |  |
| initialFontName |  | None |  |
| initialFontSize |  | None |  |
| initialLeading |  | None |  |
| cropBox |  | None |  |
| artBox |  | None |  |
| trimBox |  | None |  |
| bleedBox |  | None |  |
| lang |  | None |  |

##### 一般

showPage 改ページを挿入する

##### 図形

rect 矩形を描写

pdf.rect(2\*cm, 2\*cm, 6\*cm, 6\*cm, stroke=1, fill=1)

pdf.rect(1\*cm, 1\*cm, 4\*cm, 4\*cm, stroke=1, fill=1)

setFillColorRGB

pdf.setFillColorRGB(54, 54, 0)

setFillColorRGB

pdf.setFillColorRGB(0, 0, 0)

##### platypus

from reportlab.platypus import Paragraph, PageBreak, FrameBreak

##### 表

setStyle

wrapOn

drawOn

幅の指定

table=Table(colWidths=(19\*mm, 24\*mm)) # Tableのコンストラクタで指定する

セルの設定

**C1 R1**形式。（列：横方向，行：縦方向）。エクセルと違うので注意！

**(0,0)は一番左上のセル**を示す。(-1,-1)は一番右下のセルを示す（bottom\_up=True時[def.]）

⇒ 描写座標は(0,0)が左下なのに、セル座標では(0,0)は左上を示す。非常に紛らわしい。

番号は0から始まる。

マイナス値は**最後からn行目（n列目）**を示す。(pythonと同じ考え方)

FONT - takes fontname, optional fontsize and optional leading.

FONTNAME (or FACE) - takes fontname.

FONTSIZE (or SIZE) - takes fontsize in points; leading may get out of sync.

LEADING - takes leading in points.

TEXTCOLOR - takes a color name or (R,G,B) tuple.

LEFTPADDING - takes an integer, defaults to 6.

RIGHTPADDING - takes an integer, defaults to 6.

BOTTOMPADDING - takes an integer, defaults to 3.

TOPPADDING - takes an integer, defaults to 3.

BACKGROUND - takes a color defined by an object, string name or numeric tuple/list,

or takes a list/tuple describing a desired gradient fill which should

contain three elements of the form [DIRECTION, startColor, endColor]

where DIRECTION is either VERTICAL or HORIZONTAL.

ROWBACKGROUNDS - takes a list of colors to be used cyclically.

COLBACKGROUNDS - takes a list of colors to be used cyclically.

ALIGNMENT (or ALIGN) - 横位置。値=[LEFT, RIGHT, CENTRE (or CENTER)] 又はdecimal値

VALIGN 縦位置。値=[TOP, MIDDLE, BOTTOM(def.)]注意： rowHeightsの指定が必要。

資料

https://qiita.com/kokardy/items/92e8f3b65c965e20de34

#### PyPDF2

pdfの文字列化

#### PyMuPDF

#### pdfminer

pdfの文字列化

<https://github.com/pdfminer/pdfminer.six>

## プログラミング

### Pydantic

型アノテーションを利用して、実行時における型ヒントを提供したりする。

json 等で記述されたファイルを型安全に読み書きする為に用いられる。

補足：FastAPIで用いられるが、FastAPI専用という事は無い。

[ models.py ]

from datetime import datetime

from typing import Optional

from pydantic import BaseModel, Field

class Order(BaseModel):

name: str

created\_at: datetime

price: float = Field(..., gt=0)

note: Optional[str] = None

ファイルからの読み取り（python）

from pathlib import Path

fpath = Path(...)

model = Order.parse\_file(fpath)

### cookiecutter

Pythonプロジェクトの雛形を作成するツール

### tox

複数pythonパッケージのバージョン管理ツール。

とくに開発テストで用いられる。

pip install tox

https://tox.wiki/en/latest/

## サービス（デーモン）系

### celery

非同期処理をあつかう？

pip install celery

補足

別にメッセージブローカーのサービスが必要。Redisが使われる事が多い。

### supervisor

Pythonスクリプトのデーモン化

sudo apt install supervisor

pip install supervisor # こちらでもインストールできるが、格納場所が変わってくる

### schedule

import schedule

from time import sleep

def task():

print("タスク実行中") # 実行する関数

schedule.every(10).seconds.do(task) # スケジュール登録

while True: # イベント実行

schedule.run\_pending()

sleep(1)

### apscheduler

pip install apscheduler

## 画像関連

### Pillow

(PIL)

画像処理のライブラリ

### opencv

詳細は別ドキュメントAI関連　に記述。

## テスト

### Factory

## Web系

### gunicorn

#### 概要

pythonで動くアプリケーションサーバー。Django，Flaskなどで使われる。

FastAPIではデフォルトのアプリケーションサーバーになっている。

#### 設定ファイル

仮想環境の場合

/home/myapp/env/lib/python\*.\*/site-packages/gunicorn/config.py

#### 自動起動

ExecStart=/home/kimisyo/minicona3/envs/django/bin/gunicorn --config /home/kimisyo/helloworld/gunicorn.conf.py helloworld.wsgi:application

デフォルトのポート番号は8000らしい？

<https://docs.gunicorn.org/en/stable/deploy.html#systemd>

詳細

./env/lib/python3.7/site-packages/gunicorn/sock.py

トラブルシューティング

これが有力っぽい：https://teratail.com/questions/362399

資料

[ /etc/systemd/system/gunicorn.socket ]

[Unit]

Description=gunicorn socket

[Socket]

ListenStream=/run/gunicorn.sock

SocketUser=www-data # nginxがアクセスできさえすればよい（らしい）

# SocketMode=600 # socket権限で制限する場合？？

[Install]

WantedBy=sockets.target

→うまくいかなかった

### pings

pip install pings

### ping3

pip install ping3

### smtplib

メール用。

### selenium

Webブラウザの自動化ツール

（補足）pythonだけのパッケージではない　対応言語：Java、Python、C#、Ruby、JavaScript、Kotlin

（公式）<https://www.selenium.dev/ja/documentation/>

### waitress

WSGIWeb準拠のWebサーバ

こんな感じで使用

waitress-serve --port=8080 config.wsgi:application

（公式）<https://docs.pylonsproject.org/projects/waitress/en/stable/>

## Webアプリ関連

### jinja

Python用のhtmlテンプレートエンジン。

テンプレートエンジン：テンプレートと呼ばれる雛形と、あるデータモデルで表現される入力データを合成し、成果ドキュメントを出力するソフトウェア

#### %タグ

##### if条件分岐

{% if 条件式1 %}

<!-- 条件式1がtrueの場合の処理 -->

{% elif 条件式2 %}

<!-- 条件式2がtrueの場合の処理 -->

{% else %}

<!-- 条件式1、2のいずれも満たさない場合の処理 -->

{% endif %}

##### forループ

{% for i in 配列 %}

<!-- ループの中身 -->

{% endfor %}

#### 資料

公式資料：https://jinja2docs.readthedocs.io/en/stable/

## 通信系

#### paho

##### 概要

MQTTのクライアント（publisher、subscriber）のパッケージ。

他にbrokerが必要な点に注意。

##### 例

import paho.mqtt.client as mqtt

# The callback for when the client receives a CONNACK response from the server.

def on\_connect(client, userdata, flags, rc):

print("Connected with result code "+str(rc))

client.subscribe('ir/test')

def on\_message(client, userdata, msg):

print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect(host='localhost', port=1883, keepalive=60)

client.loop\_forever()

## データベース系

### sqlalchemy

#### 概要

Python用のObject Relational Mapper。

データベースのテーブルとpythonのクラスを対応させてデータ管理を行う。

#### チュートリアル

##### インストール

python -m pip install SQLAlchemy

python -m pip install greenlet # 補足情報：内部で必要になるパッケージ

##### 接続文字列

postgresqlの場合

engine = create\_engine(

"postgresql+psycopg2://user:password@127.0.0.1/db\_name",

)

MySQL(MariaDB)の場合

engine = create\_engine(

"mysql+pymysql://user:password@127.0.0.1/db\_name?charset=utf8mb4"

)

SQLiteの場合

engine = create\_engine(

"sqlite+pysqlcipher://user:password@127.0.0.1/db\_name.db",

connect\_args={"check\_same\_thread": False}

)

# ローカルファイルの場合

engine = create\_engine(

'sqlite:///C:\\path\\to\\your\\db.sqlite', echo = True)

SQLServerの場合

2017. 2012+, 2005にしか対応していない様子

2019にも接続できた2022-02-04W

import urllib

server = '127.0.0.1\SQLEXPRESS'

database = 'seigi\_app'

username = 'sa'

password = 'sa'

odbc\_connect = urllib.parse.quote\_plus('DRIVER={ODBC Driver 17 for SQL Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD=' + password)

engine = create\_engine('mssql+pyodbc:///?odbc\_connect=' + odbc\_connect)

他にも方法があるかも？

公式：<https://docs.sqlalchemy.org/en/14/dialects/mssql.html#module-sqlalchemy.dialects.mssql.pyodbc>

##### 対応するクラスの作成

from sqlalchemy.schema import Column

from sqlalchemy.types import Integer, String

class User(Base):

\_\_tablename\_\_ = "user" # テーブル名を指定

user\_id = Column(Integer, primary\_key=True) # 列名=変数名とする

name = Column('名前', String(255)) # または左の方法で列名を指定する

補足：データテーブルに存在するすべての列に対応する必要は無い。

参考資料：<https://docs.sqlalchemy.org/en/14/dialects/>

##### SQL操作

DELETE

count = session.query(User).filter(User.id==1).delete()

session.commit() # delete実行

公式：[https://docs.sqlalchemy.org/en/14/orm/query.html?highlight=filter#](https://docs.sqlalchemy.org/en/14/orm/query.html?highlight=filter)

参考：<https://qiita.com/tomo0/items/a762b1bc0f192a55eae8>

##### トランザクション

実際に接続できたもの

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

from sqlalchemy.schema import Column

from sqlalchemy.types import Integer, Text, VARCHAR

Base = declarative\_base()

class User(Base):

\_\_tablename\_\_ = 'user' # テーブル名を指定

id = Column(Integer, primary\_key=True)

name = Column(VARCHAR)

if \_\_name\_\_== 'main':

connect\_string = 'postgresql+psycopg2://usi:kaikusai134@192.168.11.97:5432/usi'

engine = create\_engine(connect\_string, isolation\_level="SERIALIZABLE")

SessionClass = sessionmaker(engine) # セッションを管理するクラスを作成

session = SessionClass()

records = session.query(User).all()

for rec in records:

print(rec.name)

users = session.query(User.name, User.age).all() # =select

users = session.query(User).filter(User.name=='hoge') # =where

参考：<https://qiita.com/ariku/items/75799665acd09520bed2>

公式：<https://docs.sqlalchemy.org/en/14/tutorial/index.html>

#### 詳細

##### データ型

抜粋

|  |  |  |
| --- | --- | --- |
|  | 概要 | 詳細／備考 |
| Integer | a |  |
| String | 文字数指定可能な文字列 |  |
| Text | 文字数を指定しない文字列 |  |
|  |  |  |

参考資料：<https://docs.sqlalchemy.org/en/14/core/type_basics.html>

##### SQL

COUNT

session.query(User).\

filter(User.id==1).\

print(production\_logs.count())

#### レファレンス

##### create\_engine

構文；

create\_engine(

"{dialect}+{driver}://{username}:{password}@{host}:{port}/{database}?charset={charset\_type})

|  |  |
| --- | --- |
| 要素 | 説明 |
| dialect | DBの種類を指定する．sqlite, mysql, postgresql, oracle, mssqlとか． |
| driver | DBに接続するためのドライバーの指定をする．指定しなければ，"default" DBAPIになる． |
| username | DBに接続することができるユーザ名を指定する． |
| password | DBに接続するためのパスワードを指定する． |
| host | ホスト名を指定する．localhostとかIPアドレスとか． |
| port | ポート番号を指定する．指定しなければ，defaultのポート番号になるっぽい？ |
| database | 接続するデータベース名を指定する． |
| charset\_type | 文字コードを指定する．utf8とか． |

パラメータ

connect\_args

# SQLiteの時必須のパラメータらしい

engine = create\_engine(

SQLALCHEMY\_DATABASE\_URL, connect\_args={"check\_same\_thread": False}

)

convert\_unicode 全ての文字列のString.convert\_unicodeがTrueの時と同じような処理を行う

###### excute

SQLの実行をする。

engine.execute('SELECT \* FROM person')

##### ext.declarative

###### declarative\_base

metadata.create\_all(database.engine)

checkfirst １度でも実行済かどうかの確認（二回目以降はテーブル作成は無い）def.=True

Base.metadata.create\_all(bind=engine, checkfirst=False)

tables （配列）作成するテーブルの指定

Base.metadata.create\_all(bind=engine, tables=[User.\_\_table\_\_])

##### ext.declarative

###### declarative\_base

metadata.create\_all(database.engine)

##### orm

###### sessionmaker

データベースとの接続セッションを提供する

構文

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

engine = create\_engine('postgresql://scott:tiger@localhost/')

Session = sessionmaker(engine)

with Session() as session:

session.add(some\_object)

session.add(some\_other\_object)

session.commit()

公式：<https://docs.sqlalchemy.org/en/14/orm/session_api.html>

参考：<https://qiita.com/ariku/items/14f49fe24395f1dd9412>

autocommit = False,

autoflush = False,

bind 対象のエンジンを指定する

engine = create\_engine( SQLALCHEMY\_DATABASE\_URL )

SessionLocal = sessionmaker(bind=engine)

##### orm.schema

###### Column

パラメータ

default

class User(Base):

\_\_tablename\_\_ = "users"

is\_active = Column(Boolean, default=True)

index bool

unique bool

primary\_key bool

autoincrement=True, 連番にするかどうか

##### orm.scoping

###### scoped\_session

session = scoped\_session(

# ORM実行時の設定。自動コミットするか、自動反映するなど。

　　　　sessionmaker(

　　　　　　　　autocommit = False,

　　　　　　　　autoflush = False,

　　　　　　　　bind = ENGINE

　　　　)

)

##### pydantic

###### @validator

入力値のチェックを行う？

@validator('username')

def username\_alphanumeric(cls, v):

assert v.isalpha(), 'must be alphanumeric'

return v

参考：https://qiita.com/0622okakyo/items/d1dcb896621907f9002b

公式：https://pydantic-docs.helpmanual.io/

##### security

###### HTTPBasic

###### HTTPBasicCredentials

##### sql.functions

###### current\_timestamp

##### filter

公式：<https://docs.sqlalchemy.org/en/14/orm/query.html#sqlalchemy.orm.query.Query.join>

###### one

user=session.query(User).filter(User.id==user\_id).one()

（参考）all

###### limit

指定する上位n件のレコードを取得する

user\_name = session.query(User).limit(10).all() # all()をつけないと重くなるらしい

###### in

names = ['taro', 'jiro', 'ichiro']

brothers = session.query(User).filter(User.name.in\_(names)).all()

そのまま使うと遅くなるらしい。工夫が必要。

###### orderby

users = session.query(User).order\_by(desc(User.created\_at)).all()

###### distinct

重複レコードの排除

user\_name = session.query(User).distinct(User.name).all()

注意：PostgreSQLのみで有効

###### join

user\_name = session.query(User, UserSocial).\

join(UserSocial, User.id==UserSocial.user\_id).all()

outerjoin LEFT JOINの事

##### dialects

###### mysql

MySQLの場合、このパッケージ下にINTEGER, BOOLEANなどのクラスがあるので、sqlalchemy.types の代わりに使う？

unsigned

id = Column (‘id’, INTEGER(unsigned=True), # こんな感じで

#### 資料

公式情報：<https://docs.sqlalchemy.org/en/14/index.html>

FastAPI入門（無料Kindle）：<https://zenn.dev/sh0nk/books/537bb028709ab9>

#### メモ

行先不明の情報をとりあえず残す

def get\_db(): # リクエスト時にSessionLocalを作成し完了したら終了する

try:

db = SessionLocal()

yield db # yield を用いる事でセッションを確実に終了する

finally:

db.close()

#### 具体例

product\_names\_of\_today = session.query(Log) \

.filter(Log.line\_name=='CHボデー', cast(Log.date\_time, Date)=='2022-07-29').all()

## データ可視化

### グラフ作成：matplotlib

#### 概要

python -m pip install -U matplotlib

python -m pip install -U japanize\_matplotlib # 日本語表示する為のパッケージ

情報元　<https://matplotlib.org/index.html>

%matplotlib inline

Jupyter Notebookでmatplotlibを使用する際に記述する。

matplotlibではバックエンドを「inline」に指定している。

（参考）https://www.yutaka-note.com/entry/matplotlib\_inline

#### レファレンス

##### axhline

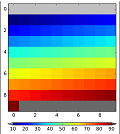
水平線を描写する

plt.axhline(y=3.5, color='r', linestyle='--')

axhline と axvline

hlines と vlines

##### colorbar()

表にカラーバーを追加する。（右図参照）

散布図などでも使用可能らしい

構文：matplotlib.pyplot.colorbar(mappable=None, cax=None, ax=None, \*\*kw)[source]

（参考）<http://hydro.iis.u-tokyo.ac.jp/~akira/page/Python/contents/plot/general/colorbar.html>

（公式；英語）<https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.colorbar.html>

##### figure()

##### grid()

格子線（グリッド）の設定を行う。

構文：matplotlib.pyplot.grid(visible=None, which='major', axis='both', \*\*kwargs)

##### legend

凡例の表示

##### imshow()

##### show()

subplot()

##### xticks()

X軸の目盛り（Tick）位置と、メモリの値を指定

（例）

xticks(np.arange(0, 1, step=0.2)) # 初期値0、最終値1、増幅0.2

（公式；英語）<https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.xticks.html>

（関連）yticks()

##### xlabel()

横（x）軸のラベル名を指定する

（関連）ylabel

### pandas

#### 概要

データ分析（データサイエンス）のライブラリ。大きな表データ，行列を扱う事ができる。

#### レファレンス

##### plot

kind

'hist' ヒストグラム

'scatter' 散布図

#### ライブラリ

##### head

最初の5行を表示

##### tail

最後の5行を表示

df = pd.read\_csv('../input/titanic3.csv', encoding='utf-8')

df.head

df.tail

.shape DataFrameサイズを確認

df = pd.read\_csv('../input/titanic3.csv')

df.shape

>(1309, 14)

##### json\_normalize

##### transpose()

### vega

## データ分析

### scikit-learn

#### リファレンス

##### decomposition.PCA

model = sklearn.decomposition.PCA(arg)

trans\_data = model.fit\_transform(data)

## データ管理／変換

### pickle

pythonで複数のオブジェクトを1つのまとまりに保存する

import pickle

pickle.dump(オブジェクト, ファイル):

pickle.load(ファイル):

import pickle

d = {"name":"ndj", "age":25, "hobby": "YouTube"}

with open("sample.pickle", mode="wb") as f:

pickle.dump(d, f)

（公式資料）<https://docs.python.org/ja/3/library/pickle.html>

### d5py

HDF5ファイル（.h5または.hdf5）を扱う。

HDF5

Hierarchical Data Formatバージョン5の略。

階層化された形でデータを保存することができるファイル形式

（参考）https://qiita.com/simonritchie/items/23db8b4cb5c590924d95

## Windows関連

### pywin32

#### 概要

Windows環境でWin32 APIへアクセスする。

サービス（デーモン）を作成する事もできる。

#### インストール

pip install pywin32

#### チュートリアル

import win32serviceutil

import win32service

import win32event

import servicemanager

import socket

class MySvc (win32serviceutil.ServiceFramework):

def \_\_init\_\_(self,args):

win32serviceutil.ServiceFramework.\_\_init\_\_(self,args)

self.stop\_event = win32event.CreateEvent(None,0,0,None)

self.stop\_requested = False

# サービス停止

def SvcStop(self):

self.ReportServiceStatus(win32service.SERVICE\_STOP\_PENDING)

win32event.SetEvent(self.stop\_event)

self.stop\_requested = True

# サービス開始

def SvcDoRun(self):

servicemanager.LogMsg(

servicemanager.EVENTLOG\_INFORMATION\_TYPE,

servicemanager.PYS\_SERVICE\_STARTED,

(self.\_svc\_name\_,''))

# メインループ関数呼び出し

self.main\_loop()

pass

if \_\_name\_\_ == '\_\_main\_\_':

win32serviceutil.HandleCommandLine(MySvc)

サービス登録

python test.py --startup delayed install # サービス登録

python test.py start # サービス実行

python test.py stop # サービス停止

python test.py remove # サービス削除

### asyncio.windows\_events

asyncio.windows\_events import NULL

2022-05-20何のこと？

## 深層学習関連

### 概要

TensorFlow

PyTorch

Chainerからのフォーク。

Chainer

日本発の深層学習ライブラリ。

自然言語処理の分野で重宝。

Keras

最も優しい深層学習ライブラリ。

近年ではTensorFlowの一部として組み込まれている

### TensorFlow

#### 概要

テンソルフロー、テンサーフロー

機械学習ライブラリで

|  |  |  |  |
| --- | --- | --- | --- |
| 開発 | Google Brain | 対応OS | Linux、macOS、Windows、Android、iOS |
| 運営 |  | 対応言語 | C言語、C++、Python、Java、Go |
| ライセンス | Apache License 2.0 |  |  |

以下のような用途に使用可能

顔認識、音声認識、被写体認識（コンピュータビジョン）

画像検索、リアルタイム翻訳

ウェブ検索最適化

メール分別、メール自動返信文作成

自動運転車[25]、マリオカートをプレイ[26][27]

#### チュートリアル

pip3 install tensorflow

#### 用語

ML；Machine Learning 機械学習

### pytorch

自然言語処理で利用されている、Pythonのオープンソースの機械学習ライブラリ。

### gym

正式名はOpenAI Gym。強化学習用シミュレーションライブラリ。

Windowsは公式にはサポートしていない。（いちおう動作はする）

インストール

pip install gym

## その他AI系

### Mecab

和布蕪（めかぶら）

オープンソースの日本語形態解析システム

## html関連

### lxml

Web上のhtml文書から情報をスクレイピングする為のツール。

### selenium

ブラウザ操作専用のRPAツール。使用にあたり、WebDriver、Selenium２つのアプリケーションをインストールする必要がある。

補足：ソフトウェア名はSeleniumで、python専用ではない。Java, JavaScript, C#, Ruby, Kotlinなどに対応。

公式のDockerイメージもある為、比較的簡単に試す事ができる。

## その他

### pyautogui

キーボード操作やマウス操作をプログラムで実行する（RPA）。

### Notebook

### Jupyter

psycopg2 postgresqlのドライバ

pypostgresql postgresqlのドライバ

pytz

## メモ

パッケージのバージョンによるクラスの変更ができる？

class JsonModel(models.Model):

value = JSONField() if django.VERSION < (3, 1) else models.JSONField()

# パッケージマネージャー

## pip

### 概要

pythonで使える外部ライブラリを管理する。

Python Installs Packagesの略らしい。

色々な使い方があり、多少紛らわしい：

pip install [*パッケージ名* ] # 新規インストール

pip3 install [*パッケージ名* ] # python3 の場合

python -m pip install [*パッケージ名* ] # 上と全く同じらしい。（他のpipコマンドも同様）

sudo pip install [*パッケージ名* ] # sudo を付けた方法（後述）

sudo apt install python-pip # aptを利用（後述）

補足：

sudo 有無の違い

sudoなし

~/.local/lib/python[*ver*]/site-packagesにインストール　※「~」はホームディレクトリの意

sudoあり

/usr/local/lib/python[*ver*]]/dist-packagesにインストール

/usr/localは、ローカル機（全体）で使用されるファイルを格納する。→他ユーザーにも影響。

pipとaptの違い

補足：aptからでもpipパッケージをインストールする事ができる。

pip

PyPI.PyPlによりホスト、提供される。様々なバージョンを選べる

apt

Canonicalによりホスト、提供される。基本的に最新バージョンのものしか選べない。

→venvなんかでバージョン管理したい時に不便。

pipとpython -m pip（pip3 とpython3 -m pip）の違い

pip

標準のpipが受け持つバージョンのpythonが対象となる。対象のpythonの確認方法は以下

pip -V

# 通常実行（例）

> pip 20.3.4 from /usr/lib/python3/dist-packages/pip (python 3.9)

# 仮想環境の場合（例） 補足：仮想環境の時はpipもpython -m pip も全く同じ結果になる？

> pip 22.2.1 from /home/usr1/my\_proj/.venv/lib/python3.9/site-packages/pip (python 3.9)

pip

標準pythonが対象となる。確認方法は以下

which python3

> /usr/bin/python3

### 基本情報

#### requirements.txt

複数のpipパッケージのバージョンを完全に一致させる為に使われるファイル

正式な書式に従う事で、下のようにインストール時に使う事ができる。

pip install -r requirements.txt # このような事が可能

pip freeze > requirements.txt # 現在の状況を保存する方法。

requirements.txt

Django>=1.8,<2.0 # 最低と最高のバージョン指定

補足：

setupのinstall\_requires　にとって代わるもの。（過去はそのような方法が取られていたらしい）

### 基本コマンド

pip install -U *パッケージ名*  # アップデート。 --updateでも良い。

pip install --update pip # 例　pip自身をアップデート。

pip install [-U] *./ファイル名*  # .whl などのファイルを使う場合（PyPIからダウンロード）

pip install パッケージ名==バージョン # インストール（バージョン指定）

pip install "cookiecutter==1.4.0" # 例

pip list # インストール済みパッケージ名とバージョン一覧

pip list --outdate # 最新版になっていないもののみ表示

pip freeze # インストール済みパッケージ名とバージョン一覧

pip -V # pipのバージョン情報を表示 --version

pip show *パッケージ名* # パッケージのバージョン情報などを表示

pip help # pipの主要コマンドとオプション一覧を表示

pip *コマンド* -h # コマンドの内容とオプションを表示

pip uninstall *パッケージ名* # アンインストール

pip download *パッケージ名*  # 最新ファイルをDL(インストールはしない)

### オプション

--proxy プロキシサーバーを利用する

pip install --proxy="https://192.168.0.1:8080" django

-q；--quiet 結果を出力しない

## Windows環境

現在のプロキシサーバーの調べ方

cmd

> netsh

netsh > winhttp

netsh winhttp > show proxy

> exit // 終了

### コマンド詳細

#### サブコマンド

##### list

python3 -m pip list | grep

#### オプション

-v pipのバージョン確認

pytnon –m pip -v

### トラブルシューティング(pip)

#### python setup.py egg\_info" failed with error code 1

原因が２つあり，１つめはpipバグ。

python -m pip install --upgrade pip setuptools # 既に修正済なので，更新すれば解決される。

２つ目。このエラーメッセージはpipのパッケージの新規インストールの際に発生しているはず。

そのパッケージインストールの時に”Error:” でエラーメッセージを表示しているはずなので，そこを確認する。

恐らく、何らかのパッケージが不足している，といった旨のメッセージであるはず。

#### not a supported wheel on this platform

原文：ERROR: パッケージ名.whl is not a supported wheel on this platform.

理由：pythonのバージョンに合わない.whlファイルをインストールしようとした。

対策：

恐らくオフラインで.whlファイルからpipパッケージをインストールしようとしているはず。

パッケージファイルの名前に「CP\*\*」という記述があるはずなので確認 例：CP37

pythonのバージョンに合った.whlファイルをダウンロードし直す　 例：python3.8 ⇒ CP38

## PyCharm

GUIでpythonのパッケージ管理ができる。

## poetry

poetry init コマンドを実行すると、最終的にpyproject.toml ファイルが作成される。

## Pipfile

## オフラインでpipを更新

### 基本

パッケージダウンロード先：<https://pypi.org/>

 ⇒ 検索 ⇒ 選択 ⇒ 

旧バージョンが欲しい場合は　　を探す

こんな感じでインストールできる

python -m pip install -U .lib\pip-19.1.1.zip

参考：<https://qiita.com/saten/items/d2ac85947583723246bf>

参考（英語）：<https://pip.pypa.io/en/stable/cli/pip_download/>

### CPの調べ方

pipパッケージによっては「CP」がいくつも用意されているものがある。

そのCPは特定のpythonバージョンに対応している。例python3.7⇒CP37，python3.8⇒CP38

補足：一応以下のようにすると具体的なCP名を確認できるらしい（未確認）

python

from setuptools import pep425tags

print(pep425tags.get\_supported())

参考：<https://greenhornprofessional.hatenablog.com/entry/2020/06/14/121944>

補足情報

.whl コンパイル済のpythonパッケージ（？）

# 資料

## 100本ノック

### num.py

https://github.com/rougier/numpy-100/blob/master/README.md

### pandas

GithubでJypter notebookがダウンロードできるので、簡単に始められる

https://github.com/kunishou/Pandas\_100\_knocks

（元ネタ）<https://www.dataschool.io/python-pandas-tips-and-tricks/>

※下準備として以下が必要。

・ jupyterのインストール

[Ctrl]+[Shift]+[p]でコマンドパレット→「select interprenter to start jupyter server」

・pipの必要パッケージ（必要であれば仮想環境を使用する）

pip3 install pandas

pip3 install matplotlib

・問題の読み込み

# Shift + Enterで題材データ、回答コードを読み込んで下さい

ipykernelもインストールする流れになる

（参考）<http://houdoukyokucho.com/2020/08/11/post-1484/>

baseball = pd.read\_csv(‘data.csv’)

len(baseball[(baseball[‘BMI’] < 200)] # このように、条件で絞り込みができるらしい

### 深層学習

<https://github.com/karaage0703/DeepLearningMugenKnock>

#### 準備

minicondaのインストール

公式のインストーラー（またはバッチ）を利用

注意！このスクリプトはなぜか実行権限が付いていない。chmodでx（実行）を付与する。

（公式）<https://docs.conda.io/en/latest/miniconda.html#latest-miniconda-installer-links>

資料

https://qiita.com/karaage0703/items/7b5d54223d06e4b6ef0f

### 言語処理

https://nlp100.github.io/ja/

github：https://github.com/nlp100/nlp100.github.io

### 画像処理

#### 概要

OpenCVなどを用いず、画像処理の基礎の部分を学習する。

github：https://github.com/yoyoyo-yo/Gasyori100knock

### その他

基本アルゴリズム

https://awesomeopensource.com/project/darkprinx/break-the-ice-with-python

## ipynbファイル

フォーマットとしてはJSON

python3 -m pip install -U ipykernel # Jupyter Notebookのpython用パッケージ

## トラブルシューティング（python全体）

### タブとスペースの混在

inconsistent use of tabs and spaces in indentation

理由：1つの行で（特にインデントに）TabとSpaceが両方混在している。

対策：どちらか片方にする。pythonではTabを極力使わない方が良いので，Spaceに統一すると良い。

### 循環参照

most likely due to a circular import

理由

循環参照の内、どちらか片方を\_\_main\_\_モジュールとして読み込む為の不完全な参照。

発生原因は非常に複雑な為、ここでは記述しきれない。参考資料を参照。

（参考）<https://qiita.com/katsuko0303/items/6e1cbf78e7fa9a2b668c>

対策：

・そもそも循環参照しないような設計を心掛ける。

・if \_\_name\_\_=’\_\_main\_\_’ 書き，他モジュールからimportされた時に実行されたく無い処理はそこに記述する。

・from 無しでimportだけで読み込むと良い？？

⇒ python2まではそれでよかったらしい

（解決法の参考資料）https://code-examples.net/ja/q/6ff362

python3以降

・annotationsとTYPE\_CHECKINGを利用？

from \_\_future\_\_ import annotations

from typing import TYPE\_CHECKING

if TYPE\_CHECKING:

from a import A

<https://zenn.dev/ganariya/articles/python-lazy-annotation>

### List, Tupleの読み込みエラー

TypeError: 'dict' object is not callable

dict = { 'hoge': 1, 'fuga': 2 }

hoge\_val = dict('hoge') # こうゆう事をすると発生。→ dict['hoge'] が正解。

### ’msvcp140\_1.dll'がない

Could not find the DLL(s) 'msvcp140\_1.dll'

理由

Microsoft Visual C++ 再頒布可能パッケージがインストールされていない。

（windows環境）

対策：

Microsoft Visual C++ 再頒布可能パッケージをインストール。

（公式）<https://docs.microsoft.com/ja-JP/cpp/windows/latest-supported-vc-redist?view=msvc-170>

# 用語

## Lazyという表現

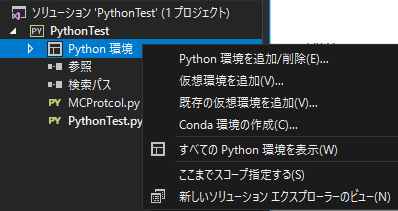
〇〇Lazyという表現のパッケージ名をたまに見かける。

これは、実際にそのパッケージ（のインスタンス）にアクセスが発生するまでは特に何もせず、アクセスされた時に初めて実行される、といった表現。

# その他

## .NET ライブラリ

※個人的にはpython で.NET を利用するメリットはほとんど無いと考える

**Python for .NET**：<http://pythonnet.github.io/>

Webで調べると「pip install pythonnet　でインストール」とあるが、VS上では「Python環境」を右クリック → 全てのPython環境を表示 →概要をクリック　PyI

を表示する。（右写真参考）

→検索ウィンドウにmatplotlib と入力→ pythonnet と入力 pip install pythonnet を実行する。→ VS再起動

import clr

clr.AddReference("System")

で.NET ライブラリは読み込み可能。（但し当然Windows環境でないと使えない）

dll をpython のコードと同じディレクトリに入れれは、自作dll も使用可能。

## C#（.NET）からの実行 [20.03.25]

System.Diagnostics.Processを使う事で，.NET環境からもpython を実行できる。

概要：

・pythonへの引数は　“module\_path arg1 arg2” と言った形で，空白で区切る。

・実行の時に使うmodule\_path に空白を含んではいけない。厳密には可能だが，引数とみなされてしまい上手く動作しない。また，クォーテーション( ’ や ” )で囲ってもダメらしい。

・python側で**一番最初にprint()した文字列**がpythonからの戻り値になる。

対象moduleとの間にjson文字列を返すラッパーを用意すると良い。（xmlとかでも良いが）

具体例：

string interpreter\_fullPath = “python.exe” //path が通ってる場合はこれで良い

string module\_fullPath = @"C:\src\excution.py" //空白を含んではいけない　‘ “ も使えない

var arguments = new List<string>

{

module\_fullPath, // args[0]：対象moduleのpath　※pathに空白を含んではいけない

"10", // args[1]：python 側で int(arg[1]) などとして，任意の型に変換する

"20" // args[2]：

};

using (var process = new System.Diagnostics.Process()

{

StartInfo = new System.Diagnostics.ProcessStartInfo(interpreter\_fullPath)

{

UseShellExecute = false,

RedirectStandardOutput = true,

Arguments = string.Join(" ", arguments),

CreateNoWindow = true,

},

})

{

process.Start();

var stream = process.StandardOutput;

var result = stream.ReadLine();

process.WaitForExit();

process.Close();

Console.WriteLine("value:" + result);

}

## Web資料

pythonチュートリアル

<https://docs.python.org/ja/3/tutorial/>

※なお、このチュートリアルは本にもなっている。

pythonライブラリ

<https://docs.python.org/ja/3/library/index.html>

## 勉強に最適なサンプル

### Python自身のライブラリ

Pythonはc++とかみたいに使用するライブラリのソースコードが見られる。これを分析する事は，使用するライブラリの理解にも繋がり、package構造も学べる為非常に有用。

但しPython中級レベル以上でないと挫折するかも。

ライブラリのパス

C:\Users\[user\_name] \AppData\Local\Programs\Python\Python37\Lib

### 一般アプリ

Anki

Gitで普通にダウンロードできる上に，万人が使えるアプリなので，実際に使用しながらライブラリ構造を学習できる。

## 疑問

covariant, contravariant？

VT\_co = TypeVar('VT\_co', covariant=True)

KT\_contra = TypeVar('KT\_contra', contravariant=True)

## コンパイラ

pyinstaller