# 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。それには関数ポインタも含まれる。

・変数はmutable（変更可能）とimmutable（変更不可）が存在する。

・動的型付け。

・ガベージコレクション対応

・パッケージ、モジュール、クラス、関数型　などマルチパラダイム。

・インデント型（ {} などの区切り文字が不要）

補足

・誰が書いても同じコードになる　が理念

・「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。

## インストール

### Linuxの場合

### Windowsの場合

C:\User\[ユーザー名]\AppData\Local\Programs\Python にある

C:\Users\[ユーザー名]\AppData\Roaming\Python にある事も

### 標準のpythonを変更

例えばpython3.9を新しくインストールしたとする

Linux

sudo unlink /usr/bin/python3 # 不要かも 2023-03-24

sudo rm /usr/bin/python3 # 不要かも 2023-03-24

sudo ln -sf /usr/bin/python3.9 /usr/bin/python3

補足

ln 書式： target\_path ← link 引数は対象が先、リンクが後

オプション：-s シンボリックリンクを作成；-f既存のファイルを強制的に上書き

こんな方法がある？

# update-alternativesを更新す

sudo update-alternatives --config python

sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.9 1

# update-alternatives: --install needs <link> <name> <path> <priority>

### コンパイルしてインストール

pythonのソースコードは以下のWebストレージでダウンロード可能

<https://www.python.org/ftp/python/>

#### インストール

[ terminal ]

# 必要パッケージのインストール

sudo apt update

sudo apt install python3-dev python3-pip zlib1g-dev libssl-dev

# python3.9 のダウンロードと展開

wget https://www.python.org/ftp/python/3.9.2/Python-3.9.2.tar.xz

tar xJvf Python-3.9.2.tar.xz

# コンパイルとインストール

cd Python-3.9.2 # ダウンロードしたディレクトリへ移動

./configure # 環境の確認とmakefileの作成

make –j4 # コンパイル　–jオプション；ジョブの数を指定

sudo make altinstall # インストール

/usr/local/bin/python3.9 にインストールされる

#### pip3.9のインストール

get-pip.pyというスクリプトを使う。見つからない場合は以下のどちらかで入手

# bootstrapから入手

Wget https://bootstrap.pypa.io/get-pip.py

# PypaのGithubリポジトリで入手

wget https://github.com/pypa/get-pip/raw/master/get-pip.py

# pip3.9のインストール

sudo python3.9 get-pip.py

# 使い方例：python-venvのインストール

sudo pip3.9 install python3.9-venv

## 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immutableかmutableかによってコピー変数での挙動が変わってくる

### 関数での変数の扱い

関数の中で代入しても、スコープが外れると値はもとに戻る（immutable的に動作する）

spam = "hoge"

def do\_local():

spam = "local"

do\_local()

print("After local assignment:", spam)

> hoge

関数の外で変数を生成している場合、関数より上で生成した変数のみ有効

val=7

def mymethod(arg=val):

print(arg)

val=10

mymethod ()

> 7

## 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

インターフェイスオプション

-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？

- 標準入力からコマンドを読み込む

一般オプション

-i スクリプトや-cオプションの実行後にインタラクティブモードに入る

-I 隔離モードで実行

公式：[https://docs.python.org/ja/3/using/cmdline.html#interface-options](https://docs.python.org/ja/3/using/cmdline.html" \l "interface-options)

### python-config

--cflag c言語ライブラリを再認識させる

python3.8-config --cflag

## インタラクティブモード

### プライマリプロンプト

入力を促す >>> の状態の事。逆にif文のように複数行にまたがる処理の場合は・・・と表記されるが、これをセカンダリプロンプトと呼ぶ。

プライマリプロンプト表示時の対話モードの終了

Ctrl+D Unix系

Ctrl+Z Windows

## 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](https://docs.conda.io/en/latest/miniconda.html" \l "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

### Docker

### 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。

VSCodeで使う例

source venv/bin/activate # 仮想環境のアクティベート

pip3 install jupyter # pipを用いてインストール

→ VSCodeの"Select Interpreter"でjupyterが入った仮想環境を選択。

## 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：

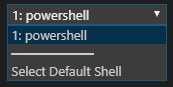
PSコマンドプロンプト

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。

}

}

## コンパイル

### compileall

python3 -m compileall main.py

importしているモジュールを格納しているディレクトリ内のソースコードもコンパイル可能

### py\_compile

python3 -m pip install py\_compile

python3 -m py\_compile main.py

### pyinstaller

python3 -m pip install pyinstaller

pyinstaller tmp.py --onefile

### py2exe

Windowsの実行ファイルを作成するツール

### Codon

codon run fib.py

公式Git：<https://github.com/exaloop/codon>

### Cython

C言語へトランスコンパイル

### Numba

## その他基本情報

### 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。 例：\_\_my\_private\_member

・mangling（ClassName\_\_my\_membernの形でモジュール内の変数にアクセスする事）

・\_\_ で始まる特別な変数

アンダーバー１つ。\_

・関数の返りがtupleの内使用しない戻り値がある場合、それをアンダーバー１つで示す。

（pythonにおけるアンダーバー１つはこの使われ方がメイン。）

・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

### isinstance

if isinstance(numeric\_level, int):

pass

# プログラム情報

## 基本

### 特殊文字

#### ＼（スラッシュ）

行内改行 ※＼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:

エスケープ

例：

\t タブ \n改行 \’シングルクォーテーション　\”ダブルクォーテーション　 \\ バックスラッシュ

#### バックスラッシュ（/）

位置専用引数

#### 〜（チルダ）

ビット単位のNOT演算子を表す

#### 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](https://docs.python.org/ja/3/library/constants.html" \l "Ellipsis) あまり情報がない

#### \_（アンダーバー）

・対話モードの時、最後に評価した戻り値へのポインタ。

hoge = lambda: 1 # def hoge(): return 1 と同等

hoge() # 関数 hogeを実行　> 結果:1

\_ #　結果:1

・関数の戻り値で「使わない部分」を暗に意味する変数名として使う

x, \_ = (1, 2, 3)

#### まとめ

|  |  |  |
| --- | --- | --- |
| 種類 | 説明 | 演算子 |
| 代入演算子 | 代入を行う | = |
| 累算代入演算子 | 左辺と右辺の値を演算して得られる結果を左辺に代入する | +=、-=、\*=、/=、//=、%=、>>=、<<=、&=、^=、|=、単項+、単項- |
| 算術演算子 | 加減乗除などを行う演算子 | +、-、\*、/、//、%、\*\* |
| 比較演算子 | 値が等しいかやそれらの大小を比較したり、複数要素を格納するオブジェクトに対して、要素の存在確認をしたりする演算子 | ==、!=、>、>=、<、<=、is、is not、in、not in |
| ブール演算子 | 真偽値のブール演算を行う | or、and、単項not |
| ビット演算子 | ビット演算（数値の2進数表現のビットごとの演算）を行う演算子 | &、^、|、単項~ |
| シフト演算子 | ビットシフト（数値の2進数表現を左右にずらす）を行う演算子 | <<、>> |
| 三項演算子 | ある条件が成否に応じて演算結果を決定する演算子（if文とは異なり、これは演算子なので、式中にそのまま記述できる）。条件式とも呼ぶ | x if 条件 else y |
|  |  | （条件が真ならx、偽ならy） |
| 代入式（セイウチ演算子） | Python 3.8で追加された、式しか記述できない場所で変数に何らかの値を代入できるようにする仕組み。「:=」がセイウチの目（:）と牙（=）のように見えることからセイウチ演算子などと呼ばれることもある | := |

### 四則演算

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

補足

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

優先順位　累乗⇒ 掛け算割り算⇒足し算引き算

論理和 bit\_or = 0b0100 | 0b1010

論理積 bit\_and = 0b1111 & 0b0101

### 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。 |  |

### 変数

a=b=1 # aもbも1になる。右から処理される？

### 予約語

公式：[https://docs.python.org/ja/3/reference/lexical\_analysis.html#keywords](https://docs.python.org/ja/3/reference/lexical_analysis.html" \l "keywords)

#### del

listやdictから要素を削除する。

tuppleはimmutableなので削除できない点に注意

用法

del my\_list[インデックス]

del my\_list[開始インデックス:終了インデックス]

del my\_dict['キー']

#### global

上位レベルで関数が実行された場合、スコープ外でも値の変更が適用される（mutableとして扱う）

nonlocal

同レベルで宣言された変数に関しては、関数のスコープ外でも値の変更が適用される（mutableとして扱う）

#### in

シークエンス内の値有無判定。 補足　シークエンスはstr、listなど

in シークエンス　でbool値が返る

例

a ='python code example'

if 'python' in a:

print('あったよ')

#### None

null値を示す。

null判別

str\_val = ''

if str\_val: #　str\_val is not None　と同等

print(str)

if not str\_val: # str\_val == None と同等

#### pass

なにもしない。classやmethodの定義だけを行う際などに使う。

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。

#### 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。

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:

詳細

・より厳密にいうと、withは「クラス内部でtry/finallyを用いた処理を行う」が正解。その為、何をfinallyで行うかはクラスにより異なり、「with使えば絶対安全」といった発想はやや危険。

・withで呼び出されるのは\_\_enter\_\_と\_\_exit\_\_の2つのメソッド。

### データ型

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 概要 | 備考 | 変更 | 反復 | シーケンス |
| bool |  | True (0以外はすべてTrue)  Flase(0 でも可) | 不可 |  |  |
| int | 数値 | 上限値（≒ビット数）はCPUに依存  sys.maxsizeで確認できる。 | 不可 |  |  |
| float | 浮動小数点。 | 64bit　C系のdouble相当  eについて：1e3 =1×10-3の事 | 不可 |  |  |
| complex | 複素数（実数と虚数） |  | 不可 |  |  |
| str | 文字列 | ダブル（"）シングル（'）どちらでも可 | 不可 | ✓ | ✓ |
| bytes |  |  | 不可 | ✓ | ✓ |
| bytearray |  |  | mutable | ✓ | ✓ |
| list |  |  | mutable | ✓ | ✓ |
| tuple |  |  | 不可 | ✓ | ✓ |
| dict |  |  | mutable | ✓ | map |
| char | python にはcharの概念はない。  str型をスライスして取り出す　e.g. str\_val[1] | | 存在しない | | |
|  |  |  |  |  |  |

#### 数値

|  |  |  |  |
| --- | --- | --- | --- |
| 種類 | やりかた | 具体例 | 備考 |
| 16進数 | 最初に0x | 0x10 |  |
| 8進数 | 最初に0o | 0o10 |  |
| 2進数 | 最初に0b | 0b10 |  |

isdigit

### 配列系

|  |  |  |  |
| --- | --- | --- | --- |
| 対象 | 概要 | 例 | 備考 |
| list | pythonの配列系の標準 | ["A", "B", "C"] |  |
| tuple | listより高速。値変更できない。 | ('A','B','C') | 変数を宣言するときhoge = 1, と最後をコンマにするとtuppleになってしまう点に注意 |
| set | 集合演算をする為の型 | {'A','B','C' } | データの重複ができない |
| dict | 「キー」でアクセスする配列 | {'A': 1, 'B': 2, 'C': 3} |  |

dict

値の取得方法の違い

|  |  |  |  |
| --- | --- | --- | --- |
| 手法 | 概要 | 値が無い場合 | 備考 |
| my\_dict['my\_key'] | 通常の方法 | 例外がraise |  |
| my\_dict.get('my\_key') | getメソッド | None |  |
| my\_dict.get('my\_key', None) | 初期値指定のget | 初期値になる |  |

# dict 型で index 付きのループを作成。さらにkeyとvalueに同時アクセス

for index, (key, value) in enumerate(my\_dict.items()):

append

arr.append(6,7)

arr.append((6,7))

### 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.　dicti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\_val in {‘’a’, ’b’}　→ 結果：True or False

### ループ

pythonのループにはwhileとforがある。

### 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。

str型のメソッド（抜粋）

capitalize 最初の文字だけ大文字にする

swapcase 大文字は小文字、小文字は大文字に

### 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

・indexは0から始まる

・負の値は末尾から数えた文字。-1は最終文字。

#### mutableとimmutable

概要

|  |  |  |
| --- | --- | --- |
|  | 説明 | 備考 |
| immutable | 値変更が不可 | int, str など |
| mutable | 値変更が可能 | list, 自作クラスなど |

例えばintは値の変更ができるからmutable、と誤解されがちだが、実は再代入時には**別のインスタンスが作成**されている。これはオブジェクトのidを調べると良く分かる

a = 100 # 変数の生成と代入

print(id(a)) # > 140732862243696（例）

a = 101 # 再代入。ここで別インスタンスが作成される

print(id(a)) # > 140732862243728（例）　idが変更されている

b = a # 参照コピー

print(id(b)) # > 140732862243728（例）　idはそのまま

#### 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ではこのように記述する

#### 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。

### ファイル関連

#### open

改行コードの指定 2023-09-01意味なかったが？？

with open('sample.txt', 'w', newline='\r\n') as f:

パーミションの指定

with open('sample.txt', 'w') as f:

pass

os.chmod # こんな手もあるらしい

#### csv

##### 概要

import csv

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

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, lineterminator='\n')

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 |
| module | 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とも言う  例外（Exception）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。  但し、mixedCaseが使われているモジュールがあれば、それに合わせるようにする。  module内部で使う変数⇒アンダースコアで始める。特に公開を想定しない内部変数の場合。  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。（この使われ方のほうが圧倒的に良く見かける。）  予約語と衝突する際に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。 例 arg = 2 みたいなのがダメ

・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；わざとスペルミスさせた単語）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。

・\_\_ で始まる特別なメンバ（特殊メソッドなど）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

#### flake8

#### 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（attribute）」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

Pythonにおける構造化プログラムの区分単位をまとめる。

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。

|  |  |  |
| --- | --- | --- |
| 区分 | 概要 | 備考 |
| package | 複数のmoduleで構成 |  |
| module | 複数のclassで構成 |  |
| class | 複数のmethod, attributeで構成 | pythonでは体積を少な目に |
| method | 処理の最小単位 |  |
| attribute | クラス内変数やプロパティなど |  |

### 注意事項

#### interface

pythonにinterfaceは存在しない。

抽象クラス（abstract class）は存在するのでそれで対応。

### method

#### 概要

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

#### self

class 内メソッドの第一引数の変数名は必ず selfにする。

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。

#### \*argsと\*\*kwargs

argはargument，引数の事。

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。

|  |  |  |
| --- | --- | --- |
| 対象 | 概要 | 例 |
| \*args | tupple型（の変数）を格納 | (1, 2, 3, 4, 5) |
| \*\*kwargs | dict型（の変数）を格納 | {"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

#### 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

#### @overloadデコレータ

#### 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。

### class

#### 概要

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

#### metaクラス

classの設定をする機能。

abstract

抽象クラスとして定義する

class Meta:

abstract = True

### module

#### 概要

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。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++など他の構造化言語に慣れている人にとっては，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

#### 概要

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。会社で例えると１つの事業所。

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の使用目的によるが，packageを作る機会は稀。

補足：python3.3以降は、\_\_init\_\_.pyが無くても１つのディレクトリを１つのpackageとみなすように変更された。

（参考）[package化 - ファイル構成](#_ファイル構成)

### enum

#### Literal

python3.8から使える。入力値を制限できるenumと言った感じ。

Fruit = Literal['apple', 'banana']

### 特殊メソッド

公式：[https://docs.python.org/ja/3/reference/datamodel.html#special-method-names](https://docs.python.org/ja/3/reference/datamodel.html" \l "special-method-names)

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

クラスのコンストラクタ（近年ではイニシャライザという呼び方が好まれる）

#### \_\_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？

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に使用（関数ポインタ化）する際に使用される。

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 # このdictを用いた方法を取らないとエラーになる

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

### 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キャッシュ。Gitなどではignoreの対象とする。

│ 　├ \_\_init\_\_.py # 特殊モジュール \_\_init\_\_.py（近年のpythonでは任意）

│　 └ main.py #　本処理を記述する。ファイル名は慣例としてmain、app、runなどが多い。

│

├─ 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1つのモジュールとして認識させる。

（python3.3以降は\_\_init\_\_.pyが無くてもディレクトリを１つのモジュールとして認識するようになった）

・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。

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。

参考：[https://docs.python.org/ja/3/tutorial/modules.html#packages](https://docs.python.org/ja/3/tutorial/modules.html" \l "packages)

###### 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 : [ # この様な使い方もできるが、するべきではない

print('hoge'),

print('fuga')

]

### generic

たとえばint, floatのみを許容するといったように、特定の型のみを型ヒントとして提示したい場合に使う。

from typing import TypeVar, Generic

# 型変数を宣言。　変数名は何でも良いが、TypeVar()の引数文字列と変数名は一致させる必要がある

T = TypeVar('T') # 何か１種類の型をなんでも受け付ける、という感じ

T = TypeVar('T', int, float) # 特定の型のみに制限したい場合にはこのようにする

def get\_t(arg: T): # メソッドで使用する場合

pass

class MyContainer(Generic[T]): # 型引数を取るクラスを作成。TypeVarで型引数の条件を指定

pass

# インスタンス生成共有

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

比較

|  |  |
| --- | --- |
| Any | 制限が無い。例えばメソッドの引数がint, str, 独自型なんでもござれ |
| TypeVar | 型の種類は問わないが、一度決めたらその後はその型１つに制限される |
| Union | 「２つの型のどちらか」を意味する。悪く言えばどちらか分からない。 |

djangoの例（django.db.models.manager.py）

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

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

class BaseManager(Generic[\_T]):

公式：[https://docs.python.org/ja/3/library/typing.html#generics](https://docs.python.org/ja/3/library/typing.html" \l "generics)

### 非同期処理

#### 概要

|  |  |  |
| --- | --- | --- |
| ライブラリ名 | 概要 | 備考 |
| asyncio | シングルスレッドで**非同期処理** | async/await 構文を使い 並行処理を実行 |
| threading | **単一**コアで**並行**処理 | ・ストリーミング処理、バックグラウンド処理に向く。 |
| multiprocessing | **複数**コアで**並列**処理 | 以下の場合に向く  ・シビアなタイミング制御が必要  ・大量の演算処理 |
| ThreadPoolExecutor | 高度なマルチスレッド | concurrent.futuresモジュール内  通信 |
| ProcessPoolExecutor | 高度な複数コア処理 | concurrent.futuresモジュール内  演算 |
|  |  |  |

Lock

参考１：https://3pysci.com/compare\_threading\_multiprocessing-1/

参考２：https://qiita.com/Toyo\_m/items/992b0dcf765ad3082d0b

#### concurrent.futures

##### ThreadPoolExecutorとProcessPoolExecutorの違い

|  |  |  |
| --- | --- | --- |
|  | ThreadPoolExecutor | ProcessPoolExecutor |
| 実行のされ方 | 同プロセス、他タスクとして実行 | 別プロセスとして実行 |
| 同時実行タスク数 | 上限がある | 制限なし |
| 用途 | 簡単な通信など | 大量タスクの実行 |

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

max\_workers def=CPUの数

initializer 最初に実行するメソッドを指定

mp\_context

max\_tasks\_per\_child

thread\_name\_prefix スレッド名に接頭辞 例：hoge\_\*\*\* みたいな

##### submit

１つの処理を行う

def task(value):

# do something

with ThreadPoolExecutor(max\_workers=2, thread\_name\_prefix="thread") as executor:

executor.submit(task, 100)

##### map

複数の処理を行う

def task1(value):

# do something

def task2(value):

# do something

with ThreadPoolExecutor(max\_workers=2, thread\_name\_prefix="thread") as executor:

executor.submit(task, 100)

公式：<https://docs.python.org/ja/3/library/concurrent.futures.html>

#### multiprocessing

##### Value

書式：Value(typecode\_or\_type: Type[\_CT@Value], \*args: Any, lock:)

typecode\_or\_type valueの型を指定

'b' bool

'i' int

概要

\_\_main\_\_ の場合、別プロセス間で変数を共有できない問題がある。

mode\_enabled=False

def on\_event(enabled:bool):

mode\_enabled=enabled # これだと常にFalseになってしまう

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

some\_class.event\_evoked = on\_event

↓

Valueクラスを使う事で、共有メモリ上に保持できる

from multiprocessing import Value

mode\_enabled=Value('b', False) # 初期値の設定方法

def on\_event(arg:bool):

mode\_enabled.value=arg # .value プロパティを用いる

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

some\_class.event\_evoked = on\_event

公式：[https://docs.python.org/ja/3/library/multiprocessing.html#sharing-state-between-processes](https://docs.python.org/ja/3/library/multiprocessing.html" \l "sharing-state-between-processes)

##### Array

##### dummy

multiprocessing.dummy

#### Thread

方法１

thread = threading.Thread(target=my\_method)

thread1.start()

thread1.join()

方法２

class MyCustomListener(threading.Thread):

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

supuer().\_\_init\_\_():

self.running = True

# run, stop メソッドをオーバーライド

def run(self)

while self.running:

pass # 実際には処理を記述

def stop(self)

self.running = False

if \_\_name\_\_ == "\_\_main\_\_":

listener = MyCustomListener()

listener.start()

### デコレータ

import time

# 時間を測るデコレータ関数

def method\_timer(func):

def wrapper(\*args, \*\*kwargs):

start\_time = time.perf\_counter() # 時間を測り始める

result = func(\*args, \*\*kwargs)

duration = time.perf\_counter() - start\_time # 終了時間 - 開始時間でかかった時間を計測

print(f"func:{func.\_\_name\_\_}\ttime: {duration: .5f} sec")

return result

return wrapper

@method\_timer

def some\_heavy\_task():

time.sleep(5)

some\_heavy\_task()

## ユニットテスト

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

条件が満たされないときに例外をraiseする。

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |  |

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

# 具体例

winswを用いたサービス化

補足：LinuxであればuWSGI, gunicornを用いれば良い。

## Windows環境での定周期処理

Windows環境ではsignal.SIGALRMが使えない。

その代案

import time

import threading

def worker():

print(time.time())

time.sleep(8)

def scheduler(interval:int, func, wait = True):

base\_time = time.time()

next\_time = 0

while True:

t = threading.Thread(target = func)

t.start()

if wait:

t.join()

next\_time = ((base\_time - time.time()) % interval) or interval

time.sleep(next\_time)

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

scheduler(1, worker, False)

## 環境変数

os.environ

python標準の環境変数をあつかうモジュール

import os

os.environ['MY\_VARIABLE'] = 'hoge'

print(os.environ['MY\_VARIABLE'])

python-dotenv

.envファイルからの読み込みをする

インストール

[ terminal ]

pip install python-dotenv

[ python ]（例）

from dotenv import load\_dotenv

import os

# .env ファイルから環境変数を読み込む

load\_dotenv()

# .env ファイルの内容にアクセス

print(os.environ.get('MY\_VARIABLE'))

補足

この.envファイルは公開レポジトリに含まれないようにする必要がある

gitの場合

[ .gitignore ]（例）

.env

os.getenv

環境変数のdictにget()メソッドでのアクセスを強制した感じ。

補足：該当のキーが存在しなかった場合の例外エラーを避ける

以下の二つの方法は同等：

os.getenv('MY\_VARIABLE', None)

os.environ.get('MY\_VARIABLE', None)

→ あまり使わないと思われる

# 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='')

### open

## python2特有

### commands

python3にて廃止。代わりにsubprocessを使う

## 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

### 概要

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

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

### 基本データ型

[https://docs.python.org/ja/3/library/ctypes.html#fundamental-data-types](https://docs.python.org/ja/3/library/ctypes.html" \l "fundamental-data-types)

## calendar

HTML形式のカレンダーなどを扱う

## functools

### deprecatedデコレータ

今後廃止予定のメソッドなどに付ける

@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。

### cacheデコレータ

メソッドをキャッシュで保持し、２回目以降の実行を高速にする

@cache

def func(num):

# なんらかの重い処理

for \_ in range(1000000):

num += 1

return num

### lru\_cacheデコレータ

フィボナッチ数列というらしい

## exception

### 概要

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

### 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

特定の

socket モジュール

socket.timeout: ソケット操作がタイムアウトしたときに発生します。

socket.error: ソケット関連のエラーで、基底の例外クラスです。具体的なエラー内容は、この例外の引数や属性から取得できます。

http.client モジュール（HTTP通信に関連するエラー）

http.client.HTTPException: すべてのHTTPエラーの基底クラス。

http.client.RemoteDisconnected: リモートサーバーが接続を閉じた場合に発生します。

urllib.request モジュール

urllib.error.URLError: URLの開き方に関連するエラーの基底クラス。

urllib.error.HTTPError: HTTPリクエストに関連するエラー。HTTPステータスコードなどの詳細が含まれます。

### 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](https://docs.python.org/ja/3/library/warnings.html" \l "the-warnings-filter)

（公式資料）[https://docs.python.org/ja/3/library/exceptions.html#warnings](https://docs.python.org/ja/3/library/exceptions.html" \l "warnings)

#### 警告文の表示

import warnings

warnings.simplefilter('always', DeprecationWarning)

def foo():

warnings.warn("警告文")

print("hoge")

こちらはあくまで「例外」なので、"hoge"は表示されない

import warnings

warnings.simplefilter('always', DeprecationWarning)

def foo():

raise DeprecationWarning("警告文")

print("hoge")

## os

### chmod

os.chown (file\_path, 1000, 1000)

ファイル名、ユーザーID、グループID

-1 にすると変更なし

補足：IDはidコマンドで確認できる

### chown

os.chmod(file\_path, 0o777)

### getuid

### 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。

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'config.settings')

### system

Linuxの場合

import os

os.system('sudo shutdown now')

Windowsの場合

import os

os.system('shutdown -s')

## 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](https://docs.python.org/ja/3.10/howto/logging.html" \l "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](https://docs.python.org/ja/3/library/logging.html" \l "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](https://docs.python.org/ja/3/library/datetime.html" \l "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](https://docs.python.org/ja/3/library/datetime.html" \l "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。

## io

### io.BytesIO

seek seek(0)とするとファイルの読み取り、書き込み位置をファイルの先頭に戻す

### io.StringIO

基本

# CSVデータ

csv\_data = [

['Name', 'Age', 'Country'],

['John', '25', 'USA'],

]

# StringIOを使用してCSVファイルを作成する

csv\_buffer = io.StringIO()

writer = csv.writer(csv\_buffer)

# 書き込み

for row in csv\_data:

writer.writerow(row)

csv\_content = csv\_buffer.getvalue()

f = StringIO()

#以下の行でCSVファイルをメモリ上に書き出し

#よくある解説では、書き出し先のパスを指定したりするが、今回はStringIO()を指定

writer = csv.writer(f, quotechar='"', quoting=csv.QUOTE\_ALL, lineterminator="\n")

#以下なんやかんやCSVファイルにデータを追記

names = []

for field in in\_db.header.fields:

names.append(field.name)

writer.writerow(names)

for rec in in\_db:

writer.writerow(rec.fieldData)

#StringIO()の中身（CSVファイル）を取得する場合、必ずgetvalue()

return f.getvalue()

https://pypi.org/project/streamio/

サンプル

import csv

import io

csv\_data = [

['Name', 'Age', 'Contry'],

['John', '25', 'USA'],

]

with io.String() as f:

writer = csv.writer(f)

for row in csv\_data:

writer.writerow(row)

csv\_content = f.getvalue()

## 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

## 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>

## multiprocess

### Value

書式：Value(typecode\_or\_type, \*args, lock=True)

例

from multiprocessing import Value

shared\_boolean = Value('b', True) # 共有メモリ上に配置する変数

if shared\_boolean.value: # 使う時

pass

typecode\_or\_type

型コード（下表参照）かctype（[参考](https://docs.python.org/ja/3/library/ctypes.html?highlight=ctypes" \l "fundamental-data-types)）の型を指定する

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 型  コード | Cの型 | Pythonの型 | 最小サイズ  (バイト単位) | 備考 |
| b | signed char? | int | 1 |  |
| B | unsigned char | int | 1 |  |
| u | wchar\_t | Unicode文字(unicode型)? | 2 |  |
| h | signed short? | int | 2 |  |
| H | unsigned short? | int | 2 |  |
| i | signed int? | int | 2 |  |
| I | unsigned int? | int | 2 |  |
| l | signed long? | int | 4 |  |
| L | unsigned long? | int | 4 |  |
| q | signed long long? | int | 8 |  |
| Q | unsigned long long? | int | 8 |  |
| f | 浮動小数点数 | 浮動小数点数 | 4 |  |
| d | double | 浮動小数点数 | 8 |  |

ctypesを使う場合

from ctypes import c\_bool

shared\_boolean = Value(c\_bool, True)

ndarrayなどを使う場合

from ctypes import c\_char\_p # uint(=byte)型ポインタ

array = np.array([0, 1, 2, 3], dtype=)

shared\_boolean = Value(c\_char\_p, True)

公式：[https://docs.python.org/ja/3/library/multiprocessing.html#multiprocessing.Value](https://docs.python.org/ja/3/library/multiprocessing.html" \l "multiprocessing.Value)

公式（プロセス間メモリ）：[https://docs.python.org/ja/3/library/multiprocessing.html#sharing-state-between-processes](https://docs.python.org/ja/3/library/multiprocessing.html" \l "sharing-state-between-processes)

## pathlib

iterdir

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

## pickle

基本

import pickle

sample\_list = [1,2,3]

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

pickle.dump(sample\_list,f)

f.close

## queue

## re

正規表現を扱う

result = re.search()

### まとめ

|  |  |  |
| --- | --- | --- |
| メソッド | 概要 |  |
| match | 先頭が一致するか |  |
| fullmatch |  |  |
| search | 一致するかどうか |  |
| findall | 一致したものをlistで返す |  |
|  |  |  |
| sub | 置き換え |  |
|  |  |  |

### comple

何回も使いまわす時に便利、高速。

repatter = re.compile(pattern)

result = repatter.match(content)

### sub

result = re.sub('^E', '', arg) # 先頭文字のEを排除する例

### 資料

参考：https://note.nkmk.me/python-re-match-search-findall-etc/

## signal

### signal による非同期処理

import signal

import time

def handler (arg1, args2):#パラメータは必ず1つらしい

print(time.time())

signal.signal(signal.SIGALRM, handler)

signal.setitimer(signal.ITIMER\_REAL, 1, 1)

time.sleep(1000)

signal.signal(signal.SIGALRM, 呼び出したい関数)

signal.setitimer(signal.ITIMER\_REAL, 1回目の実行までの時間, 2回目以降の実行間隔)

参考：[https://qiita.com/montblanc18/items/05715730d99d450fd0d3#ok-](https://qiita.com/montblanc18/items/05715730d99d450fd0d3" \l "ok-シグナルハンドラを利用する)シグナルハンドラを利用する

### 本来の使い方

システム（OS）の出す割り込み信号を処理する。

その為、OSによって扱える信号が異なる。

SignalListener

### 扱うことのできる割り込み信号リスト

[https://docs.python.org/ja/3/library/signal.html#module-contents](https://docs.python.org/ja/3/library/signal.html" \l "module-contents)

## signal\_listener

from signal\_listener import SignalListener, SignalInterrupt

## 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)

### 補足

python2ではcommandsという同等モジュールがあったらしいが、python3にて廃止。

## shutil

# 以下のどれでも良い

shutil.copy(file\_copy\_from, 'dir/copy/to')

shutil.copy(file\_copy\_from, 'dir/copy/to/')

shutil.copy(file\_copy\_from, 'dir/copy/to/sample.txt')

## 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

### 具体例

#### threading.Threadを用いたスケジューラの作成

def worker():

ret, frame = cap.read()

if ret:

print(time.time())

def scheduler(interval:int, func:Callable[[None], None], wait = True):

base\_time = time.time()

next\_time = 0

while True:

t = threading.Thread(target = func)

t.start()

if wait:

t.join()

next\_time = ((base\_time - time.time()) % interval) or interval

time.sleep(next\_time)

## typing

### 概要

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

### Optional

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:

### Callable

書式：Callable[[引数1の型, 引数2の型… ], 戻り値]

from collections.abc import Callable, Awaitable

async def on\_update(value: str) -> None:

# Body

callback\_func: Callable[[str], Awaitable[None]] = on\_update

公式：[https://docs.python.org/ja/3/library/typing.html#callable](https://docs.python.org/ja/3/library/typing.html" \l "callable)

### Awaitable

公式：[https://docs.python.org/ja/3/library/collections.abc.html#collections.abc.Awaitable](https://docs.python.org/ja/3/library/collections.abc.html" \l "collections.abc.Awaitable)

### 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)

## xml

ElementTree

root.findall

# 子要素(複数の場合最初の１つ)

root.find('person') # Element

# 検索して、そのイテレータを返す

root.iter('folder')

## warnings

warningの無視の仕方

import warnings

warnings.filterwarnings("ignore")

# パッケージ

## 代表的パッケージ

### numpy

#### 概要

内部はC言語（又はFortrun）で実装されている為、通常のpythonの処理よりは高速。

特徴

ブロードキャスト

これを使うと

おもしろいのはboolもブロードキャストできる点

scores = np.array([0.9,0.8,0.7,0.6,0.5,0.4])

print([scores >= 0.5])

> [True, True, True, True, True, False]

print(scores[scores >=0.5])

> [0.9,0.8,0.7,0.6,0.5]

arrayとndarrayの違い

numpy.ndarray クラス

numpy.array numpy.ndarrayを作成する為の関数

#### チュートリアル

##### スライス

pythonと同じで [start:stop:step] となる

例

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

series = np.arange(10, 15) # [10, 11, 12, 13, 14] となる

print(array[0]) # 行を選択　インデックスは**0から始まる**為、１行目を選択

print(array[:,2] # 列を選択　**「:」は「すべて」の意**なのですべての行の3列目を選択

print(array[0,2] # すべての行の1列目と3列目を選択

print(series[::-1]) # 反転 [14 13 12 11 10]

print(array[::-1]) # 行の反転 [[4 5 6], [1 2 3]]

print(array[:,::-1]) # 列の反転 [[3 2 1], [6 5 4]]

注意点

通常pythonではスライスの結果は値渡しだが、numpyの場合は参照渡しとなる。

#### レファレンス（基本中の基本）

##### arrange

連番を作成する

##### array

配列を作成する。numpyの最も基本的なメソッドの１つ。

array = np.array([1,2], dtype=np.int16)

1 + np.arrange(4) # 少し分かりにくいが、ブロードキャストの効果で 1,2,3,4 となる。

補足

・戻り値はndarray型。

・何も型指定をしない場合、np.int64が選択される。（windowsではnp.in32）

##### reshape

a = np.arange(8).reshape(4, 2)

print(a)

# [[ 0 1 2 3]

# [ 4 5 6 7]]

関連：ravel

#### レファレンス

##### ndarray

###### copy

コピーを作成する。（値渡しでndarrayを返す）

補足：通常は参照渡し

###### tobytes

参考：numpy.frombuffer

##### astype

型変換する

array = np.array([1, 2], dtype=np.int16)

print(type(array.dtype)) # dtype('int16')

print(d.astype(array.float16)) # array([1., 2.], dtype=float16) 配列の中身も表示されるらしい

##### argmax

配列の最大要素のインデックスを返す

##### average

各種平均

array = np.array([[0.1, 0.2, 0.8, 0.9],

[0.05, 0.15, 0.75, 0.85]])

# 平均

print('axis=None:%s' % np.average(array)) # 全部の平均

print('axis=0:%s' % np.average(array, axis=0)) # 列ごとの平均

print('axis=1:%s' % np.average(array, axis=1)) # 行ごとの平均

##### 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]

##### diag

構文：numpy.diag(v, k=0)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 型 |  |  |
| v | array\_like |  |  |
| k | int, optional |  |  |

##### 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

##### eye

Z = np.eye(2)

[[1. 0.]

[0. 1.]

Z = np.eye(3)

[[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]]

##### 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)

##### frombuffer

注意点

連続しない配列の場合 np.ascontiguousarray()で配列を連続にしてから渡す

空のバッファの場合 numpy.empty()で空の配列を作成してから渡す

公式：<https://numpy.org/doc/stable/reference/generated/numpy.frombuffer.html>

##### hstack

関連：vstack

##### hsplit

b3=np.array()

関連：vsplit

##### info

関数の使い方などを調べる

np.info(np.add)

##### isnan

欠損値NaN判別を行う

補足

欠損値：値が格納されていない変数。本来得られるはずだったデータが無い。

公式：<https://numpy.org/doc/stable/reference/generated/numpy.isnan.html>

##### linalg

###### 概要

線型変換を扱う

.norm()ベクトルの長さ

###### eig

固有値（eigenvalues）と固有ベクトル（eigenvectors）を返す

公式：<https://numpy.org/doc/stable/reference/generated/numpy.linalg.eig.html>

##### 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)

> [-6, -5.87, -5.75, ... 5.75, 5.87, 6.]

##### nonzero

0でない要素だけを取り出す

np.nonzero([1,2,0,0,4,0])

[0, 1, 4]

##### max

Z = np.random.random(10)

Z.max()

こんな方法もある

Z = np.random.random(10)

print(np.max(Z))

関連：min

##### meshgrid

格子点を作る

import numpy as np

x = np.arange(0, 6, 1)

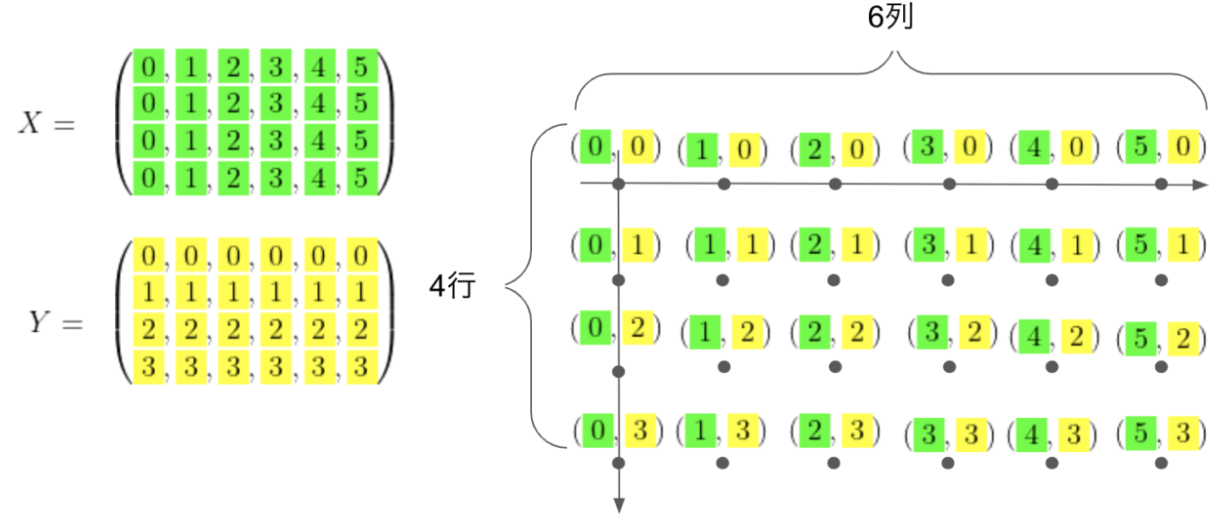
y = np.arange(0, 4, 1)

X, Y = np.meshgrid(x, y)

XY = np.vstack([X.ravel(), Y.ravel()])

print(XY)

<https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html>



##### newaxis

新しい次元を追加する

##### pad

配列の端に数値を加える。これをpadding処理と言う。

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ名 | データ型 |  | 初期値 |
| array | array\_like  of rank N |  |  |
| pad\_width | sequence, array\_like,  int |  |  |
| mode | str,  func |  | ‘constant’ |

mode

|  |  |  |
| --- | --- | --- |
| 値 | 意味 | 関連 |
| "constant" | 定数でパディングを行う。 | constant\_values |
| "edge" | Pads with the edge values of array. |  |
| "linear\_ramp" | Pads with the linear ramp between end\_value and the array edge value. |  |
| "maximum" | Pads with the maximum value of all or part of the vector along each axis. |  |
| "mean" | Pads with the mean value of all or part of the vector along each axis. |  |
| "median" | Pads with the median value of all or part of the vector along each axis. |  |
| "minimum" | Pads with the minimum value of all or part of the vector along each axis. |  |
| "reflect" | Pads with the reflection of the vector mirrored on the first and last values of the vector along each axis. |  |
| "symmetric" | Pads with the reflection of the vector mirrored along the edge of the array. |  |
| "wrap" | Pads with the wrap of the vector along the axis. The first values are used to pad the end and the end values are used to pad the beginning. |  |
| "empty" | Pads with undefined values. |  |

公式：<https://numpy.org/doc/stable/reference/generated/numpy.pad.html>

##### percentile

例えば50パーセンタイルなら中央値が返る。

中途半端な値に対しては、隣接の2つの値を線形補完して求める

##### random

###### 概要

###### random

np.random.random(4)

([0.18909088, 0.81784749, 0.42602287, 0.06341262])

np.random.random((2,3))

([[0.38954721, 0.22049524, 0.50618259], [0.94510682, 0.89374439, 0.08642851]])

###### permutation

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

###### default\_rng

default\_rng(seed=None)-> Generator

seed {int, array\_like[ints], SeedSequence, BitGenerator, Generator} 乱数生成のシード値

[https://numpy.org/doc/stable/reference/random/generator.html#numpy.random.default\_rng](https://numpy.org/doc/stable/reference/random/generator.html" \l "numpy.random.default_rng)

###### Generator

Generatorインスタンスを返す各メソッドのまとめ

|  |  |  |
| --- | --- | --- |
|  | 概要 | 詳細、備考 |
| standard\_normal |  | 平均0、分散1 |
| normal | 正規分布（ガウス分布） |  |

rng = np.random.default\_rng()

rnd\_val = rng.standard\_normal() # １つの乱数値を生成

array = rng.standard\_normal(10) # n個の乱数値を生成

※以下、特筆すべきもののみ個別で記す

###### Generator.uniform

乱数を生成する

書式：uniform(low=0.0, high=1.0, size=None)

size =(10,2) の時は10行、2列の行列が作成される

<https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.uniform.html>

##### ravel

１次元の配列を返す。（参照渡し）

関連：flatten（値渡し）、reshape（戻り値の次元数は選択可能）

Z = np.arange(8).reshape(2, 4)

Z\_flatten = Z.flatten() # 1次元の新たな変数を生成する（immutable; 値渡し）

Z\_flatten[3] = 251

print(Z)

# [[0 1 2 3], [4 5 6 7]] 値渡しなので、元の変数に変化は無い

Z\_raval = Z.ravel() # 次元にほどく(ravel)関数へのポインタというイメージ（mutable; 参照渡し）

Z\_ravel[3] = 251

print(Z)

# [[0 1 2 251], [4 5 6 7]] 値渡しなので、元の変数に変化は無い

##### repeat

array = np.repeat([1, 2, 3], 2)

> [1 1 1 2 2 2 3 3 3]

##### show\_config()

numpyパッケージの設定関連を表示する

##### sort

##### 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>

##### split

注意点：あくまで作成されるのはndarray

box = np.array([0.41775188,0.03537531,0.99609039,0.86868081])

ymin, xmin, ymax, xmax = np.split(trc.box, 4)

print(ymin)

# [0.41775188]

##### transpose

転置行列を返す。

関連：T（まったく同じ）

##### unique

配列の要素をユニーク化して返す。

≒SQLのDISTINCT

##### unravel\_index

##### zeros

|  |  |  |
| --- | --- | --- |
| パラメータ名 | データ型 |  |
| shape | intまたはtuple(int) |  |
|  |  |  |

公式：https://numpy.org/doc/stable/reference/generated/numpy.zeros.html

np.zeros((2,3))

> [[0. 0. 0.]

[0. 0. 0.]]

関連

ones すべて1

補足：numpy100本ノックでは0の事をnullとも呼んでいた

#### リファレンス（プロパティ）

##### sizeプロパティ

zeros = np.zeros((2,3))

zeros.size

> 6

#### 注意点

要素数の異なるndarrayを作成するのは非推奨。但し、作成自体は可能。

import numpy as np

a\_object = np.array([[1], [1,2], ['one']])

#### 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)

## Python用

### six

python2系と3系の互換性の為のライブラリ

## ドキュメント関連

### Excel

エクセル

#### 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） |  |

補足

Pandasからでもエクセルを読む事はできる。

ただし、どちらかと言うとpandasはデータ分析用途に向く。

#### 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】

###### load\_workbook

load\_workbook('xxxfile.xlsm',read\_only=False,　keep\_vba=True)

###### border

for row in range(1, sheet.max\_row+1):

for col in range(1, sheet.max\_column+1):

cell = sheet.cell(row=row, column=col)

print('[枠線] r:%s c:%s %s %s %s %s' % \

(row, col, cell.border.top.style, cell.border.bottom.style,

cell.border.left.style, cell.border.right.style) )

##### 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

###### Paragraph,

###### PageBreak,

###### FrameBreak

###### Flowable

##### 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)

###### TableStyle

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.

LINEBELOW

LINEABOVE

LINEBEFORE 左側に罫線を引く

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

|  |  |
| --- | --- |
|  | 備考 |
|  |  |
| 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。 |
|  |  |

-

次ページに続く

～reportlab　表　つづき

注意点

範囲を指定する場合は1つ目の引数の値が2つ目の引数の値以下でないといけない。

これはbottomup=Falseの場合でも適用される。

例：

pdf=canvas.Canvas('temp\_purchaseitem.pdf', bottomup=False)

table\_items.setStyle(TableStyle([

('BACKGROUND', (-2, -1), (-2,-2), colors.lightpink), # NG

('BACKGROUND', (-2, -2), (-2,-1), colors.lightpink), # OK

]))

また、**エラーとはならないのが却って厄介**。指定方法がNGの場合、設定値が無効化される。

資料

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

#### PyPDF2

pdfの文字列化

・画像の抽出

・PDFファイルの結合や分割

・しおり（目次）の追加

#### PyMuPDF

#### PDFMiner

Pdfからテキストの抽出ができる

<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

### forever

特定のスクリプトを永続起動させる。

構文：python -m forever.run [OPTIONS] SCRIPT [SCRIPT\_ARGS]

例：

python -m forever.run -t 3 -i 3 python -u -m forever.example

公式：<https://pypi.org/project/forever/>

### 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/

### pydotplus

DOT言語へのPythonインターフェースを提供

Graphvizの存在ありき

### テスト

#### Factory

### 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

import mojimoji

text = "ｐｙｔｈｏｎ パイソン １０００"

print(mojimoji.zen\_to\_han(text)) # 結果：python ﾊﾟｲｿﾝ 1000

print(mojimoji.zen\_to\_han(text, kana=False)) # 結果：python パイソン 1000

kana bool型。かなの変換を行うかどうか。def.=True

digit bool型。数値の変換を行うかどうか。def.=True

ascii bool型。英字の変換を行うかどうか。def.=True

<https://pypi.org/project/mojimoji/>

## OS関連

### 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

指定したタイミングで指定した処理（ジョブ）を自動的に行う

インストール

pip install apscheduler

サンプルソース：<https://github.com/agronholm/apscheduler/tree/3.x/examples>

from apscheduler.schedulers.blocking import BlockingScheduler

from apscheduler.schedulers.background import BackgroundScheduler

### 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

## Web系

### Webサーバー

#### 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](https://docs.gunicorn.org/en/stable/deploy.html" \l "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

#### waitress

##### 概要

WSGIWeb準拠のWebサーバ。Windows環境でのwsgiサーバーはこれしか選択肢が無いらしい

こんな感じで使用

[ powershell ]

waitress-serve --port=8080 my\_module:my\_method

もしくは

[ run.py ]

import waitress import serve

def main():

pass

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

serve(main, host='0.0.0.0', port=8080 )

[ powershell ]

python run.py

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

<https://msiz07-flask-docs-ja.readthedocs.io/ja/latest/deploying/waitress.html>

参考

Windows環境でのwsgiサーバー：<https://matome.eternalcollegest.com/post-2144189743674290701>

##### トラブルシューティング

###### WARNING:waitress.queue:Task queue depth is N

アプリケーション内で待機中の処理がn個ある事を示している。

対策

演算能力を上げる

タスクキューの値を増やす

非同期処理を加える

Webアプリ系

### Webアプリ系

#### 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/

### WebAPI系

#### BeautifulSoup

厳密にはWebのスクレイピングだが、これを使ってWebAPIっぽい事もできる。

#### 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。

（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/>

#### 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。

### その他Web系

#### ping3

pip install ping3

#### pings

pip install pings

#### smtplib

メール用。

#### Scrapy

複数ページにアクセスするクローリング

robots.txtに対応

Webサイトへのアクセス間隔の設定

## データ系

### キャッシュ

#### 概要

|  |  |  |  |
| --- | --- | --- | --- |
|  | 保存場所 | 対象 | 備考 |
| cache | メモリ | 関数の戻り値 | python標準（functools） |
| lru\_cache | メモリ | 関数の戻り値 | python標準（functools） |
| cachetools |  |  |  |
| joblib | ストレージ | 関数の戻り値 |  |
| pickle | ストレージ | オブジェクト |  |
| redis-py | DB | オブジェクト | OSSのredisサーバーと併用 |
| pymemcache | メモリ |  | pure-Python |
| python-memcached | DB |  | OSSのMemcachedサーバーと併用 |

補足

・pymemcache はmemcachedとは違うので注意！

・関数の戻り値をキャッシュする場合は、引数が同じ場合は同じ値を返す場合に向く。

その他

|  |  |
| --- | --- |
|  |  |
| UWSGICache |  |
| Flaskのflask\_caching | FlaskでViewをはじめとした関数のキャッシュ化。Redisを併用。 |
| streamlitのcache | 関数のキャッシュ化 |

#### cachetools

from cachetools import TTLCache, cached

|  |  |
| --- | --- |
| 名前 | 内容 |
| TTLCache | キャッシュの存続可能時間指定。  存続可能時間を超えたものにはアクセス不可。  最も使用頻度の低いアイテムから破棄される。 |
| LFUCache | 最小使用頻度。アイテム取得頻度を計測して、最も頻繁に使用されていないものから破棄 |
| LRUCache | 最長未使用頻度。最も長く使用されていないものから破棄 |
| RRCache | ランダム。アイテムをランダムに選択して、そこから破棄 |

#### joblib

from joblib import Memory

memcache

import memcache

from functools import lru\_cache

@lru\_cache(maxsize = None)

#### 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>

#### redis-py

### データ変換

#### 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>

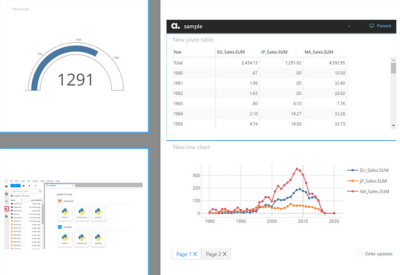
### データ分析

#### atoti

pythonで簡単にデータ分析用ダッシュボードを作成する

[https://www.atoti.io](https://www.atoti.io/)

pip install atoti[jupyterlab]



かなり便利そうなんだけど、Web上での資料があまり無いような？

#### mglearn

グラフ描画やデータ読込でソースコードがぐちゃぐちゃにならないようにするためのライブラリ

anacondaなどには含まれていなことが多いので注意

pip install mglearn

#### 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。

|  |  |  |  |
| --- | --- | --- | --- |
| クラス | 配列次元 |  | 備考 |
| Series | 1次元 |  |  |
| DataFrame | 2次元 |  |  |
| Panel | 3次元 |  |  |

##### レファレンス

###### Series

###### DataFrame

概要

行数が1になると自動的にSeriesになってしまう。

明示的にDataFrameを使いたい場合には以下のようにする

import pandas as pd

s1 = pd.Series([1,2,3], index=['col1', 'col2', 'col3',])

df = pd.DataFrame([s1])

列の抜粋

df\_col = df['col1'] # 列１つの場合は文字列

df\_cols = df[['col1', 'col2']] # 列２つ以上の場合はlist で指定する（tupleではダメ）

Seriesからの結合

s1 = pd.Series([1, 2, 3])

s2 = pd.Series([4, 5, 6])

df\_clac = pd.concat([

s1,

s2.rename('col2'), # 列名の指定

], axis=1)

\_\_init\_\_

DataFrame() 列名などの指定は後で行う

DataFrame({}) dict方式で列名と値を指定

DataFrame(columns={'col1':[], 'col2':[], 'col3':[], ])

DataFrame(columns)

DataFrame(columns=[ 'col1', 'col2', 'col3', ])

names 行ラベル（ヘッダーを指定）

columns 列名を指定

apply

astype

df.astype({'uuid': str}, {'dt': datetime.datetime})

df.index.astype(str)

describe()

列ごとのcount, mean, std などの結果をテーブル形式で表示する。

df = pd.read\_csv('sample.csv')

print(df.describe())

filter()

書式：DataFrame.filter(items=None, like=None, regex=None, axis=None)

items 完全一致

like 部分一致

regex 正規表現で選択

公式：<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.filter.html>

isin

indexではない値を検索する。

書式 -> List[bool]

mask = df['sex'].isin([male,female])

if mask.any():

print('あるよ')

補足：indexに設定してある列は検索に使用できないらしい

公式：<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.isin.html>

items

def items() -> Iterable[tuple[Hashable, Series]]

loc

loc.apply

行ラベルと列ラベルを指定して、DataFrameの特定の部分集合を取得する。

plot()

kind

'hist' ヒストグラム

'scatter' 散布図

pivot

公式：<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pivot.html>

query()

あるDataFrameのから条件に当てはまるDataFrameを作成する

df.query('int\_val >= 150')

df.query('int\_val >= 150 and str\_val=="hoge"')

df.query('date\_val < "2000-01-30 12:34:56"')

他：!=、or、

具体例

df = pd.DataFrame({

'A': range(1, 6), # 1, 2, 3, 4, 5

'B': range(10, 0, -2), # 10,8 ,6 ,4, 2

'C C': range(10, 5, -1) # 10,9 ,8 ,7, 6

})

df.query('B == `C C`')

# 結果

# A B C C

# 0 1 10 10

公式：<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.query.html>

set\_index

特徴

loc, atで要素を選択（抽出）するとき分かりやすくなる

指定した行の情報をユニークに指定する（？）

df = pd.DataFrame({

'month': [1, 4, 7, 10],

'sale': [55, 40, 84, 31]

})

df.set\_index('month')

パラメータ引数

inplace

verify\_integrity 新しい行が追加される時に重複チェックを行うかどうかdef.=False

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.set_index.html>

sum()

書式：DataFrame.sum(axis=None, skipna=True, level=None, numeric\_only=None, min\_count=0, \*\*kwargs)

axis

numeric\_only int,float,boolのみ；近いうちに必須オプションになるらしい

補足：sumの結果１行だけになるとSeries型になる。

公式：<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sum.html>

transpose（メソッド）

列と行を入れ替える

T（プロパティ）

transposeメソッドと同じ結果を返すプロパティ。

<https://pandas.pydata.org/pandas-docs/version/0.25.0/reference/api/pandas.DataFrame.T.html>

###### 入出力系

単にprintする

print(df)

to\_csv

DataFrame.to\_csv(path\_or\_buf=None, header=True, index=True) 【抜粋】

path\_or\_buf

header

index インデックス列を書き込むかどうか

mode python標準のopenメソッドと同様の方法で書き込みモードを選択

補足：パーミション、所有者の指定はできない。（python標準ではパーミションは対応可能）

公式：<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html>

to\_markdown()

pip install tabulateが必要

to\_pickle

import pandas as pd

df.to\_pickle('sample.pickle')

# 読み込み

df = pd.read\_pickle('sample.pickle')

補足：pickleはバイト配列にシリアライズする為のパッケージ

シリアライズするとファイルサイズはむしろ増える点に注意。アクセススピードは速くなる。

read\_csv

pd.read\_csv(DATA\_URL)

パラメータ

nrows

names ヘッダー

dtype 使用する列とそのデータ型

json\_normalize

read\_sql

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_sql.html>

read\_sql\_query

SQL 文をそのまま使う

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_sql_query.html>

read\_html()

html内のテーブルをさがしてdfのリスト（つまりPanelになる？）に変換する

BeautifulSoup4、html5lib、lxmlが必要

データ操作

###### データ操作系

apply

対象範囲のすべての値を加工する

補足：applyメソッドはDataframe, Seriesにそれぞれ存在する。

Dataframe

書式：DataFrame.apply(func, axis=0, raw=False, result\_type=None, args=(), \*\*kwargs)[source]

例：

distinct\_df = df.apply(lambda x: len(x.unique()))

公式：<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.apply.html>

Series：

書式：Series.apply(func, convert\_dtype=True, args=(), \*\*kwargs)[source]

公式：<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.apply.html>

get\_dummies

ダミー変数を作成する。

補足：ダミー変数

カテゴリ分けされた変数を0と1だけで構成されているデータ（one-hot表現）に変換したもの。

例：

パラメータ

|  |  |  |
| --- | --- | --- |
| パラメータ名 | 型 | 概要 |
| data | 配列,Series/DataFrame | ダミー変数を作成したいデータを選択します。 |
| prefix | str/list[str] /dict[str], | DataFrameのカラムラベルに付け加える文字列を指定します。  (省略可能)初期値None |
| prefix\_sep | str | prefixで指定された文字列を付け加える際に橋渡しとなる文字列を指定します。  (省略可能)初期値’\_‘ |
|  |  |  |
| dummy\_na | bool値 | (省略可能)初期値False |
|  |  | 欠損値もダミー変数として処理するかどうかを指定します。 |
| columns | リスト | (省略可能)初期値None |
|  |  | ダミー変数に変換したい列データを指定します。指定されない場合は全てのデータを対象にダミー変数を作成します。 |
| sparse | bool値 | (省略可能)初期値False |
|  |  | SparseDataFrameとして返り値を返すかどうかを指定します。 |
| drop\_first | bool値 | (省略可能)初期値False |
|  |  | カテゴリー数がk個あった時、k-1個のダミー変数を作成するかどうかを指定します。 |
| dtype | データ型 | (省略可能)初期値np.uint8 |
|  |  | 新たなカラムのデータ型を指定します。 |

<https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html>

groupby

公式：<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.core.groupby.GroupBy.sum.html>

###### 統計系

mean

df.mean() # 列ごとの平均１

df.mean(axis=0) # 列ごとの平均２

df.mean(axis=1) # 行ごとの平均

count

value\_count

データの出現回数を返す

###### plotting（可視化系）

scatter\_matrix

scatter matrix（スカッター行列；各データ間の相関関係を見る為の図）の描画を行う。

公式：<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.plotting.scatter_matrix.html>

###### html系

set\_properties

各セルに一律に適用する CSS

set\_table\_styles

各セルに個別に適用する CSS

table\_css = [

# ヘッダの CSS

{

"selector": "th",

"props": [

("background-color", "#00557F"),

("color", "white"),

("border", "1px solid #006699"),

],

},

# ボディの CSS

{

"selector": "td",

"props": [

("background-color", "#E1EEF4"),

("color", "#00557F"),

("border", "1px solid #006699"),

],

},

]

style = df.head().style

style = style.set\_precision(2).set\_table\_styles(table\_css)

style

###### concat

対象：DataFrame型、Series型

ignore\_index=False, インデックスを指定しなくても、新しい行を追加できる。（自動インデックス）

verify\_integrity=False,

axis Literal型；0:index, 1:column,

sort=None

関連：appendメソッド

具体例

cols=['datetime', 'col1', 'col2']

df = pd.DataFrame(columns= cols)

new\_df = pd.DataFrame(

data=[[datetime.datetime.now(),1, 'hoge' ]], # Datafremeなので二重リスト

columns= cols,

)

df = pd.concat(

[df, new\_df],

ignore\_index=True,

axis=0,

)

print(df)

メモ：書き込みを非同期でやる場合

with ThreadPoolExecutor(thread\_name\_prefix="thread\_csv\_write") as executor:

executor.submit(write\_csv\_2)

補足

2023-02-07 「行」としてSeriesをDataFrameに追加する事を思いついたが、できないのかもしれない

###### date\_range

###### 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

.shape DataFrameサイズを確認

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

df.shape

>(1309, 14)

##### 具体例

###### 時間ごとにまとめる

データ

2000-01-01 01:45:00,1,0,0

2000-01-01 01:50:00,1,0,1

2000-01-01 01:55:00,1,1,0

2000-01-01 02:00:00,0,0,1

2000-01-01 02:05:00,1,0,0

方法１：集計用の列を追加する

df = pd.DataFrame(

columns=['datetime', 'col1', 'col2', 'col3'],

)

# pandasのdatetime型に変換

df["datetime"] = pd.to\_datetime(df["datetime"])

# hour列を作成

df["hour"] = df["datetime"].dt.hour

# 時間ごとの合計を出す

df = df.groupby("hour").sum()

方法２：set\_indexを使う

（調査中）

df = df.set\_index([df.index,df.index.hour])

df.index.names = [“date”,”hour”]

df.groupby(level=[“hour”]).mean()

pandas.Index.names

###### cssを編集する

（効果不明）

table\_css=[

{

"selector" : "td",

"props" : [

("color" : "#005577"),

]

}

]

df.head().style.set\_properties(table\_css)

##### 資料

データ検索のまとめ：<https://qiita.com/k_maki/items/812967c8cbf26331e418>

都道府県データとpandas：<https://www.procrasist.com/entry/4-pandas>

#### Polars

超高速版のPandas

pip install polars

RustとPythonで使える

<https://www.pola.rs/>

#### scikit-learn

概要

機械学習のアルゴリズムを提供するライブラリ。

##### リファレンス

###### 全体

公式：<https://scikit-learn.org/stable/modules/classes.html>

###### Cluster

全体

[https://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster](https://scikit-learn.org/stable/modules/classes.html" \l "module-sklearn.cluster)

KMeans

AgglomerativeClustering

<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html>

affinity

|  |  |  |  |
| --- | --- | --- | --- |
| euclidean | ユークリッド距離 |  |  |

linkage complete

###### decomposition.PCA

主成分分析を行う

model = sklearn.decomposition.PCA(arg)

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

パラメータ

|  |  |  |  |
| --- | --- | --- | --- |
| 名前 | データ型 | 概要 | 初期値 |
| n\_components | float or int | 圧縮後の次元数 | None |
| copy | bool |  | True |
| whiten | bool | 白色化処理を行う | False |
| svd\_solver |  |  | 'auto' |

公式：<https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>

n\_components とsvd\_solverの組み合わせで色々な方式を選択できるらしい

|  |  |  |
| --- | --- | --- |
| n\_components | svd\_solver |  |
| 'mle' | 'full' |  |
|  |  |  |

###### impute

SimpleImputer

欠損値(NaN)を統計的な値で穴埋めする

公式：<https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html>

###### linear\_model

LogisticRegression

###### naive\_bayes

ナイーブベイズクラス分類器を提供する

線形モデルよりも訓練が高速。

汎化性能はLogisticRegressionやLinearSVCよりもわずかに劣る

MultinomialNB

<https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html>

GaussianNB

<https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html>

BernoulliNB

<https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html>

###### neighbors

KNeighborsRegressor

k-最近傍回帰によるクラス分類を提供する

from sklearn.neighbors import KNeighborsClassifier

alpha

KNeighborsClassifier

k-最近傍法によるクラス分類を提供する

###### metrics

confusion\_matrix

混同行列(Confusion Matrix)を作成する

書式：confusion\_matrix(y\_true, y\_pred, \*, labels=None, sample\_weight=None, normalize=None)

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ名 | データ型 | 概要 | 初期値 |
| y\_true | array-like |  |  |
| y\_pred | array-like |  |  |
| labels | array-like |  | None |
| sample\_weight | array-like |  | None |
| normalize | str | ‘true’  ‘pred’,  ‘all’ | None |

classification\_report

適合率、再現率、F値を出力する

例

from sklearn.datasets import load\_iris

from sklearn.svm import SVC

from sklearn.metrics import classification\_report

from sklearn.model\_selection import train\_test\_split

iris = load\_iris()

# 学習データセット、テストデータセットに分割(7:3)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

iris.data, iris.target, test\_size=0.3, random\_state=123)

# Support Vector Machineで学習

svc = SVC()

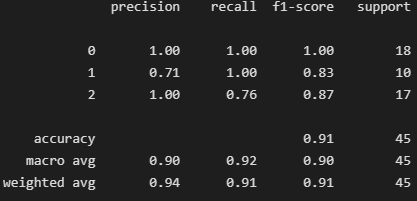
svc.fit(X\_train, y\_train)

# テストデータセットの予測

y\_pred = svc.predict(X\_test)

# 適合率、再現率、F値を出力

print(classification\_report(y\_test, y\_pred))

結果

クラス

marcro avg 横方向の各指標に対してのクラス平均

weighted avg 重み付き平均

|  |  |  |  |
| --- | --- | --- | --- |
| 列名 | 意味 | 概要 |  |
| 無し | クラス（ラベル） |  |  |
| precision | 適合率 |  |  |
| recall | 再現率 |  |  |
| f1-score | F値 |  |  |
| support | データ数 |  |  |

F値 二値分類のタスク（問題）に対する評価指標の一つで、適合率（Precision）と再現率（Recall）のトレードオフ関係に着目し、2つの値を調和平均（詳細後述）した値のこと

roc\_curve

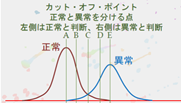
補足

ROC曲線（Receiver Operating Characteristic；受信者動作特性）

カットオフポイントごとに真陽性率（=TPF）と偽陽性率（=FPF）を計算し、

縦軸にTPF、横軸にFPFにしたグラフにする？

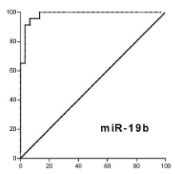
縦軸に感度、横軸に偽陽性率 (False Positive Rate) (1.0 – 特異度)？

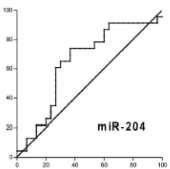
カットオフポイント（Cutoff Point）

正常と異常を分ける点（右図参照）

roc\_auc\_score

補足

AUC；Area Under the Curve

ROC曲線は左上側のスペースが少なければ少ないほど「良い判定」となり、それを数値化したのがAUC。具体的にはROC曲線で作られる面積を差す。右図では左側の方が「良い判定」となる。

公式：<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html>

###### model\_selection

概要

[https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model\_selection](https://scikit-learn.org/stable/modules/classes.html" \l "module-sklearn.model_selection)

cross\_val\_score

交差検証を提供する。

公式：<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html>

|  |  |  |  |
| --- | --- | --- | --- |
|  | データ型 |  | 初期値 |
| estimator |  | 学習器を持つ(fit関数を持つ)オブジェクト |  |
| X | array-like | 学習データ（array, listでも良い） |  |
| y | array-like | 教師あり学習を行う場合の正解データ |  |
| groups | array-like | GroupKFoldを行う場合の層化指標 |  |
| scoring | str /  callable | 評価指標 | None |
| cv |  | Cross-ValidationのFold数  None=5-fold | None |
| n\_jobs |  | 利用するCPU数  None=1個，-1=全てのCPU | None |
| verbose |  | 何ステップおきに学習状況を出力するか | 0 |
| fit\_params |  | estimatorに渡すパラメータ | None |
| pre\_dispatch |  | 並列実行時の起動ジョブ数（少ない方がメモリ消費は抑えられる） |  |
| error\_score |  | ：エラー発生時にエラーをraiseするか（’raise’を設定） |  |

交差検証；cross-validation

個々のモデルの汎化性能を評価する手法で、データをK個に分割しその内１つをテストデータにして、更にそれをK回繰り返してすべてのグループを１回ずつテストデータになるようにする手法

K-fold

cross\_validate

cross\_val\_scoreとほぼ同じだが、scoringに指定できるものがcross\_val\_scoreとは異なる。（らしい）

GridSearchCV

ハイパーパラメータの候補を指定してベストな値を探す。

公式：<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html>

RandomizedSearchCV

ハイパーパラメータをランダムに指定してベストな値を探す？

公式：<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html>

train\_test\_split

汎化性能の評価の為、訓練用とテスト用に分割する（ホールドアウト法；hold-out validation）

array = range(10)

train, test = train\_test\_split(array)

print(train, test)

> [5, 0, 8, 4, 9, 3, 7] [2, 6, 1]

書式：train\_test\_split(\*arrays, test\_size, train\_size, random\_state=None, shuffle=True, stratify=None)[source]

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ名 | データ型 | 概要 | 初期値 |
| train\_size | float or int | int分割データ数を指定  float:0<n<1 で分割割合を指定 | None |
| test\_size | float or int | 同上 | None |
| random\_state |  | 乱数のシード値を指定する |  |

[https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.train\_test\_split.html#sklearn.model\_selection.train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html" \l "sklearn.model_selection.train_test_split)

StratifiedKFold

LeaveOneOut

モデルの評価方法としてLeaveOneOut交差検証（Leave-One-Out Cross Validation）を提供する

from sklearn.model\_selection import LeaveOneOut

rng = np.random.default\_rng()

array = rng.standard\_normal(5)

loo = LeaveOneOut() # Generatorが返る

for train\_indices, test\_indices in loo.split(array): # Generatorなのでloopで使うしかない？

train = array[train\_indices]

test = array[test\_indices]

print(train, test, train\_indices, test\_indices)

例えば上の例では以下のようになる

|  |  |  |  |
| --- | --- | --- | --- |
| ループ回数 | train\_indices | test\_indices |  |
| 1 | [1, 2, 3, 4] | [0] |  |
| 2 | [0, 2, 3, 4] | [1] |  |
| .. |  |  |  |
| 5 | [0, 1, 2, 3] | [4] |  |

loo.get\_n\_splits(array)

<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.LeaveOneOut.html>

###### preprocessing

StandardScaler

データを標準化する

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(data) # std（標準偏差）とmean（平均値）を算出

data\_scaled = scaler.transform(data)

# これでも良い

data\_scaled = scaler.fit\_transform(data)

公式：

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

###### svm.SVC

サポートベクタマシン

書式：svm.SVC(\*, C=1.0, kernel='rbf')

|  |  |  |  |
| --- | --- | --- | --- |
|  | データ型 | 概要 | 初期値 |
| C |  | どれだけマージンを広くするか  値が小さいほどマージンが小さい |  |
| kernel |  | {‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’} or callable | ’rbf’ |
|  |  |  |  |

rbf radial basis function

公式：[https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html" \l "sklearn.svm.SVC)

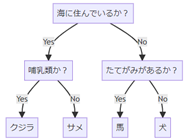
公式：[https://scikit-learn.org/stable/modules/svm.html#svc](https://scikit-learn.org/stable/modules/svm.html" \l "svc)

fit

support\_vectors

###### svm. LinearSVC

###### tree.DecisionTreeClassifier

概要

決定木によるクラス予測機能を提供する。

公式：<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

具体例

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

# Irisデータセットを読み込む

iris = load\_iris()

X, y = iris.data, iris.target

# 学習データセットとテストデータセットに分割する

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=123)

# 決定木をインスタンス化する (木の最大の深さ=3)

tree = DecisionTreeClassifier(max\_depth=3, random\_state=123)

# 学習

tree.fit(X\_train, y\_train)

# 予測

y\_pred = tree.predict(X\_test)

score = accuracy\_score(test\_y, predicted\_y)

詳細

fit

#### scipy

stats

gmean 相乗平均を求める

##### cluster.hierarchy

###### dendrogram

##### special

from scipy import special

###### erf

誤差関数

###### comb

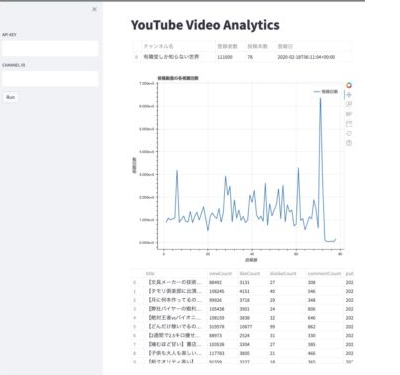
二項係数

##### integrate

積分関連を扱う？

#### streamlit

##### 概要

pandas と併用する事で、分析結果や可視化ツールのWebアプリを簡単に作成できる。

pip install streamlit

公式：<https://streamlit.io/>

pip：https://pypi.org/project/streamlit/

Streamlitは各ウィジェットが押下・変更される度に、都度コード全体が再実行される仕様

##### チュートリアル

###### ディレクトリ構成

myproj/

├ venv/

├ main.py

[ main.py ]

import streamlit as st

import pandas as pd

st.title('My first app')

df = pd.DataFrame({

'first column': [1, 2, 3, 4],

'second column': [10, 20, 30, 40]

})

st.write(df)

[ terminal ]

streamlit run main.py

補足：初回起動時のみ、emailアドレスの入力を求められる。空白にしておけば入力無しで先に進める。

Webブラウザで下記へアクセス（通常自動的に行われる）

http://localhost:8501/

たった、これだけ。

##### 設定ファイル

以下の場所に「グローバル設定ファイル」が存在する

~/.streamlit/config.toml # Linux/Mac

%userprofile%/.streamlit/config.toml # Windows

作業ディレクトリ直下にプロジェクトの設定ファイルの作成も可能

myproj/

├ venv /

├ .streamlit/config.toml # これ

├ main.py

試しにポート番号を変えてみる

[ .streamlit/config.toml ]

[server]

port = 8080

###### [server]

|  |  |  |  |
| --- | --- | --- | --- |
| 名前 | データ型 | 概要 | 初期値 |
| allowHosts | list | チェックボックスをチェックしたときやスライダーの点の色 |  |

###### [theme]

|  |  |
| --- | --- |
|  |  |
| primaryColor | チェックボックスをチェックしたときやスライダーの点の色 |
| backgroundColor | メインの背景 |
| secndaryBackgroundColor | インプットボックスやサイドバーなどの背景 |
| textColor | テキストの色 |
| font | フォント |

##### 具体例

###### csvファイルを自動読み込みして表示更新する

[ main.py ] (streamlit run .\main.pyで実行)

import streamlit as st

import pandas as pd

import time

filename = 'data/sample.csv'

def load\_data():

return pd.read\_csv(

filename,

names=['uuid', 'datetime', 'result'], # ヘッダー

)

# 棒グラフとテーブルを使う場合の例

table = st.empty()

bar = st.empty()

while True:

df=load\_data()

table.dataframe(df)

table.bar\_chart(df.filter(items=['col1', 'col2']))

time.sleep(1)

###### 表示される文字などのスタイルを変更する

方法１：markdownを使って<h5>タグなどを適用する

text\_number.markdown('<h5>hoge </h5>', unsafe\_allow\_html=True)

##### 詳細

###### 設定ファイル

グローバル設定ファイル

~/.streamlit/config.toml # Linux/Mac

%userprofile%/.streamlit/config.toml # Windows

プロジェクトの設定ファイル（作業ディレクトリ直下）

.streamlit/config.tom

公式：<https://docs.streamlit.io/library/advanced-features/configuration>

[theme]

表示テーマの設定

<https://docs.streamlit.io/library/advanced-features/theming>

###### ブラウザでの設定

Theme

###### 外部公開

wsgi

2023年7月現在ではwsgiに対応していない。（その為gunicornには非対応）

公式：<https://docs.streamlit.io/knowledge-base/deploy/does-streamlit-support-wsgi-protocol>

Streamlit Cloud

https://streamlit.io/cloud

Nginx

<https://sig9.org/archives/4928>

##### レファレンス（streamlit自体）

###### cache

インメモリの KVS（key-value store）によるキャッシングで動作速度を改善

###### @cache

再実行時にキャッシュした結果を利用できるようになる

例：

@st.cache

def load\_csv():

return pd.read\_csv(filename)

###### empty()

空のオブジェクトを生成する。

###### set\_page\_config

st.set\_page\_config(layout="wide")

###### session\_state

セッション変数を管理する。

##### レファレンス（一般的な表示）

###### image

画像を表示する

opencvで静止画を表示する例

import os

import streamlit as st

import cv2

img = cv2.imread(os.path.join('img', 'sample.png'))

st.image(img)

st.number\_input('my\_number')

###### spinner

###### subheader

st.subheader('サブヘッダを表示')

###### title

st.title('タイトル')

###### write

なんでも表示できる

st.write('<span style="color:red;background:pink">該当するデータがありません・・・・</span>',

unsafe\_allow\_html=True)

参考：<https://docs.streamlit.io/library/api-reference/text>

##### レファレンス（グラフ系）

###### line\_chart

折れ線グラフの表示

st.line\_chart(chart\_data)

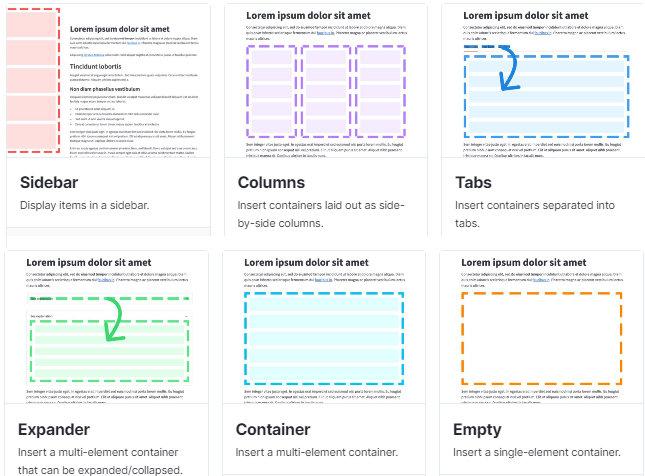
棒グラフの表示

###### bar\_chart

##### レファレンス（レイアウト系）

###### レイアウト概要

<https://docs.streamlit.io/library/api-reference/layout>



###### columns

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ名 |  | 概要 | 詳細 |
| spec | array-like | 幅 |  |

###### sidebar

##### レファレンス（html系）

###### CSSを直接記述

markdownメソッドを利用

st.markdown("""

<style>

.big-font {

font-size:300px !important;

}

</style>

""", unsafe\_allow\_html=True)

st.markdown('<p class="big-font">Hello World !!</p>', unsafe\_allow\_html=True)

###### style

df.style

###### caption

st.caption('キャプションを表示')

##### レファレンス（入力系）

###### button

import streamlit as st

btn = st.button()

if btn:

value+=1

公式：<https://docs.streamlit.io/library/api-reference/widgets/st.button>

###### checkbox

公式：<https://docs.streamlit.io/library/api-reference/widgets/st.checkbox>

###### download\_button

ダウンロードボタンを表示する

###### map

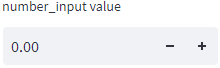
地図の表示

公式：<https://docs.streamlit.io/library/api-reference/charts/st.map>

###### multiselect

input type=multiselect（コンボボックス）を表示

###### number\_input



number1 = st.number\_input('number\_input value')

公式：<https://docs.streamlit.io/library/api-reference/widgets/st.number_input>

###### tabs

tab1, tab2 = st.tabs(["CHART", "MAP"])

# 折れ線グラフ

with tab1:

"""

グラフ描画のコード

"""

# 階級区分図

with tab2:

"""

グラフ描画のコード

"""

###### slider

slide\_value = st.slider("slider value", min\_value=0, max\_value=1000, step=1, value=100)



##### レファレンス（有名ライブラリ）

###### plotly\_chart

plotlyのグラフ（Figure）を使う

st.plotly\_chart

import streamlit as st

import pandas as pd

import plotly.graph\_objects as go

plot\_pie = go.Figure(data=[go.Pie(

labels=df\_pie.index,

values=df\_pie.values, hole=.3)]

)

st.plotly\_chart(plot\_pie, use\_container\_width=True)

公式：<https://docs.streamlit.io/library/api-reference/charts/st.plotly_chart>

##### レファレンス（データ系）

###### dataframe

公式：<https://docs.streamlit.io/library/api-reference/data/st.dataframe>

st.dataframe(df.style.highlight\_max(axis=0))

ハイライト表示

それ以外の表示方法

df = pd.DataFrame({

'first column': [1, 2, 3, 4],

'second column': [10, 20, 30, 40]

})

st.write(df)

##### レファレンス（動作）Performance

###### cacheデコレータ

インメモリの KVS（key-value store）によるキャッシングで動作速度を改善する

但し、2023年7月現在deprecated（廃止予定）なので使わないようにする（正確にはv0.88以降で非推奨）

代わりにcache\_dataかcache\_resourceを使用する。

例：

@st.cache

def load\_csv():

return pd.read\_csv(filename)

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ名 | 型 |  | 初期値 |
| func | callable |  | None |
| persist | boolean |  |  |
| allow\_output\_mutation | boolean |  |  |
|  |  |  |  |
|  |  |  |  |

##### 関連するパッケージ

###### streamlit\_autorefresh

自動更新を楽に実行するためのパッケージ？

##### 資料

###### 公式リファレンス

<https://docs.streamlit.io/library/api-reference>

###### グラフの種類

<https://docs.streamlit.io/library/api-reference/charts>

###### アプリのギャラリー

<https://streamlit.io/gallery>

###### opencvの例

<https://zenn.dev/whitphx/articles/streamlit-realtime-cv-app>

### データ可視化

#### pythonのデータ可視化パッケージ比較

|  |  |  |
| --- | --- | --- |
|  | 特徴 | 備考 |
| matplotlib | pythonグラフの老舗 |  |
| seaborn | matplotlibを見やすくした | ちょっとグラフが真面目すぎる。 |
| altair(vega) | グラフツールの定番 | vega-liteのPythonバインディング |
| plotly | 対話形式のグラフ | 見た目も良く、お勧め |
| plotly dash | plotlyを使ったダッシュボード |  |
| atoti | ダッシュボード作成ツール |  |
| streamlit | 可視化Webアプリ作成 | plotly、altair などのグラフを埋込む事ができる |
| Pandas | テーブル状のデータを扱う | 大量データに強い。高速。 |
| Polars | pandasの更に高速版 |  |

#### 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>

import matplotlib.pyplot as plt

##### チュートリアル

最も簡単な例；数式を表示

import numpy as np

import matplotlib.pyplot as plt

rng = np.random.default\_rng(123)

x = rng.random(size=(100, 2))

y = 2 \* x + 10

plt.plot(x, y)

plt.show()

最も簡単な例２；csvを表示

import matplotlib.pyplot as plt

df = pd.read\_csv('data/sample.csv', header=0,)

df.plot()

plt.show()

# fig, ax =plt.subplots(1,1)

# ax.table(cellText=df.values, colLabels=df.head)

##### 詳細

###### inlineについて

概要

これの事

%matplotlib inline

matplotlibでバックエンドを「inline」に指定　の意　⇒ グラフが標準出力に出力される

逆にこれをしないと：

ポップアップウィンドウでグラフが表示される。

対話側実行環境やエディタでは、グラフの表示のためにplt.show()をする必要がある。

その他

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

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

##### グラフ種類

|  |  |  |  |
| --- | --- | --- | --- |
| メソッド名 | 概要 |  |  |
| scatter | 散布図 |  |  |
|  | 折れ線グラフ |  |  |
| bar | 棒グラフ |  |  |
| barh | 横向き棒グラフ |  |  |
| hist | ヒストグラム |  |  |
| pie | 円グラフ |  |  |
| boxplot | 箱ひげ図 |  |  |
| violinplot | バイオリンプロット |  |  |

##### レファレンス

###### pyplot.subplot

一つの描画キャンパスを複数の領域に分割して、それぞれのサブ領域でグラフを作成することができる

（公式）https://matplotlib.org/stable/api/\_as\_gen/matplotlib.pyplot.subplot.html

import matplotlib.pyplot as plt

fig, axes = plt.subplots(2) # 2行1列つのサブプロットを配置

fig, axes = plt.subplots(2, 2) # 2行2列のサブプロットを配置

fig, axes = plt.subplots(ncols=2) # 1行2列のサブプロットを配置

plt.show()

###### axhline

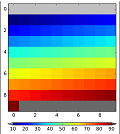
水平線を描写する

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

###### Figure

###### Axes

概要

fig, ax = plt.subplots()

ax.set\_xlabel('x label') # X軸にラベルを設定

# ２つ以上の場合

fig, axes = plt.subplots(ncols=2)

axes[0].set\_title('subplot title 0')

set\_ylabel

fig, ax = plt.subplots()

ax.set\_xlabel('x label') # X軸にラベルを設定

set\_title

fig, axes = plt.subplots(ncols=2)

# サブプロットにタイトルを設定

axes[0].set\_title('subplot title 0')

legend

凡例の表示

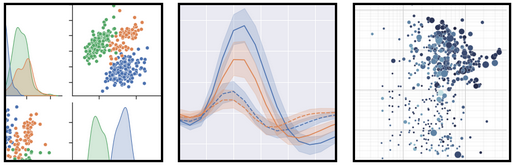
fig, ax = plt.subplots()

ax.plot([1, 2, 3], [2, 4, 9], label='legend label')

ax.legend(loc='lower right') # 凡例を右下に表示

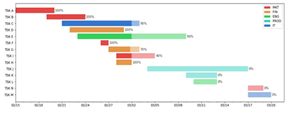
savefig

#### Seaborn

Matplotlibベースでつくられたデータ可視化ツール。

公式：https://seaborn.pydata.org/

#### python-gantt



#### plotly

##### 概要

https://plotly.com/python/

参考：[https://data-analytics.fun/2021/11/19/plotly-summary/#toc9](https://data-analytics.fun/2021/11/19/plotly-summary/" \l "toc9)

##### レファレンス

###### graph\_objects

import plotly.graph\_objects as go

公式：https://plotly.com/python/graph-objects/

テーブル関連

https://plotly.com/python/reference/#table

#### spotly dash

#### altair(vega-lite)

### 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

###### ディレクトリ構成

config

├ database.py

###### 接続文字列

共通情報

[ config/database.py ]

from sqlalchemy.engine.create import create\_engine

def get\_db\_session():

engine = create\_engine(url, pool\_recycle=10)

return sessionmaker(bind=engine)()

postgresqlの場合

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

pip install psycopg2-binaryが必要

MySQL(MariaDB)の場合

url = "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)

補足：sqlalchemy.engine.url.URLを用いた接続文字列作成方法もあるらしい

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](https://docs.sqlalchemy.org/en/14/dialects/mssql.html" \l "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

SELECT

query = session.query(Book).filter(Book.title==’’)

print(query.count()) # select実行

補足：変数queryを生成した時点ではまだSQLは実行されない（遅延実行）

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

AND

from sqlalchemy import and\_

session.query.filter(User.name == 'ed', User.fullname == 'Ed Jones')

session.query.filter(User.name == 'ed').filter(User.fullname == 'Ed Jones')

session.query.filter(and\_(User.name == 'ed', User.fullname == 'Ed Jones'))

公式：<https://docs.sqlalchemy.org/en/20/core/operators.html>

UPDATE

DELETE

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

session.commit() # delete実行

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

COUNT

session.query(User).\

filter(User.id==1).\

print(production\_logs.count())

###### 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() # all()メソッドでlistに変換

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

###### 基本的なテーブル定義

OneToMany

class Book (Base):

\_\_tablename\_\_ = 't\_book'

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

title = Column('title', String(255))

author = relationship("Author")

author\_id = Column(Integer, ForeignKey("t\_author.asset\_number") )

class Author(Base):

\_\_tablename\_\_ = 't\_author'

author\_id = Column(Integer, primary\_key=True)

author\_name = Column('asset\_name', String(255))

###### テーブルの作成

順序としては、Base（declarative\_base）を継承したモデルクラスを作成後、以下のスクリプトを実行

# セッションを管理するクラスを作成

SessionClass = sessionmaker(engine)

# テーブルの作成（ベースを継承した）

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

- checkfirst 作成済のテーブルに関しては更新を行わない　def=True

###### 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 | 引数なしでlength=maxになる |
| Text | 文字数を指定しない文字列 |  |
| Boolean |  |  |

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

###### pandasのdfへの変換

import pandas as pd

from config.database import get\_db\_session

from models import MyClass

session = get\_db\_session() # sessionはsqlalchemy.orm.session.Sessionクラス

conn = session.connection()

records = session.query(MyClass)

df=pd.read\_sql(records.statement, conn)

こんな情報もあるが？ 2023-06-08

df=pd.read\_sql(records.statement, session.bind)

###### Sessionの生成方法

Sessionクラスによる生成

sessionmakerによる生成

scoped\_sessionによる生成

scoped\_sessionによる生成（ORM編）

補足： <https://qiita.com/tosizo/items/86d3c60a4bb70eb1656e>

[ database.py ]

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

engine = create\_engine(～略～)

# この時点ではまだSessionインスタンスは生成されていない

Session = sessionmaker(autocommit = False,)

[ main.py ]

from database

session = Session()

##### データベース関連

共通に、以下のようなモデルを前提とする

from sqlalchemy.schema import Column, MetaData, Table

from sqlalchemy.types import Integer, String, DateTime

class Book (Base):

\_\_tablename\_\_ = 'book' # テーブル名を指定

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

title = Column('title', String(50))

###### INSERT

session.add(Book(

title = 'タイトル',

))

###### DELETE

session.delete(book)

###### EXISTS

ちょっとめんどくさい

query = session.query(Book).filter(Book.title == 'hoge')

if session.query(query.exists()).scalar():

print('exists!!')

###### ロールバック

try:

session.query(Location).delete()

session.commit()

except:

session.rollback()

raise

finally:

session.close()

補足：実はcommitしなければ自動的にrollbackされる（らしい）

##### レファレンス

###### 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

###### engin.create

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}

)

echo 型：bool

SQLAlchemyが発行するSQL文がコンソールに表示される

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

Engineクラス

excute

SQLの実行をする。

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

###### engine.url

URL

sqlalchemy.engine.url import URL

username = 'USER'

password = 'PASSWORD'

hostname = 'localhost'

dbname = 'DATABASE'

url = URL.create(

drivername='mysql+mysqldb',

username=username,

password=password,

host=hostname,

database=dbname,

query={"charset": "utf8"},

)

###### 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\_\_])

###### filter

公式：[https://docs.sqlalchemy.org/en/14/orm/query.html#sqlalchemy.orm.query.Query.join](https://docs.sqlalchemy.org/en/14/orm/query.html" \l "sqlalchemy.orm.query.Query.join)

one

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

（参考）all, one\_or\_none, first

first, one, one\_or\_noneの違い

|  |  |  |  |
| --- | --- | --- | --- |
|  | 検索結果:0件 | 検索結果:1件 | 検索結果:2件以上 |
| one() | NoResultFoundエラー発生 | １件を返す | NoResultFoundエラー発生 |
| one\_or\_none() | None | １件を返す | NoResultFoundエラー発生 |
| first() | None | １件を返す | 先頭１件を返す |

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

###### filter\_by

fileter\_by()の方がコードが短くなる

###### 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>

|  |  |  |
| --- | --- | --- |
| パラメータ名 | データ型 | 概要 |
| autocommit | bool |  |
| autoflush | bool |  |
|  |  |  |
|  |  |  |

bind 対象のエンジンを指定する

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

SessionLocal = sessionmaker(bind=engine)

relationship

OneToManyアクセスを可能にする。

back\_populates 逆引き（ManyToOne）アクセスを可能にする。

backref back\_populatesと同じく、逆引きを可能にする。どう違う？

Session

コンストラクタ

autocommit = False,

autoflush = True,

bind = engine

###### orm.schema

Column

パラメータ

default

class User(Base):

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

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

created\_at = Column(DateTime, default=datetime.now(), nullable=False)

|  |  |  |
| --- | --- | --- |
| パラメータ | データ型 | 意味 |
| index | bool |  |
| unique | bool |  |
| primary\_key | bool |  |
| autoincrement | bool | 連番にするかどうか |
| nullable | bool | Null許容（allow null） def.=True？ |
| default | any | 初期値 |

公式：[https://docs.sqlalchemy.org/en/20/core/metadata.html#sqlalchemy.schema.Column](https://docs.sqlalchemy.org/en/20/core/metadata.html" \l "sqlalchemy.schema.Column)

###### 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.elements

like, ilike などが格納されている

公式：<https://docs.sqlalchemy.org/en/20/core/sqlelement.html>

参考：<https://qiita.com/Bashi50/items/7f2214550e80b87fb922>

|  |  |  |
| --- | --- | --- |
|  |  |  |
| ilike |  |  |
| like |  |  |
| in\_ |  |  |

例：ilike

book = session.query(Book).filter(Book.title.ilike('Jekyll')).first()

例：in\_

book = session.query(Book).filter(Book.title.in\_(['Jekyll', 'Hyde'])

公式：[https://docs.sqlalchemy.org/en/20/core/sqlelement.html#sqlalchemy.sql.expression.ColumnElement](https://docs.sqlalchemy.org/en/20/core/sqlelement.html" \l "sqlalchemy.sql.expression.ColumnElement)

###### sql.expression

func

[https://docs.sqlalchemy.org/en/14/core/sqlelement.html#sqlalchemy.sql.expression.func](https://docs.sqlalchemy.org/en/14/core/sqlelement.html" \l "sqlalchemy.sql.expression.func)

###### sql.functions

current\_timestamp

###### types

代表的な型

|  |  |
| --- | --- |
|  |  |
| BigInteger | A type for bigger int integers. |
| Boolean | A bool datatype. |
| Date | A type for datetime.date() objects. |
| DateTime | A type for datetime.datetime() objects. |
| Enum | Generic Enum Type. |
| Float | Type representing floating point types, such as FLOAT or REAL. |
| Integer | A type for int integers. |
| Interval | A type for datetime.timedelta() objects. |
| LargeBinary | A type for large binary byte data. |
| MatchType | Refers to the return type of the MATCH operator. |
| Numeric | Base for non-integer numeric types, such as NUMERIC, FLOAT, DECIMAL, and other variants. |
| PickleType | Holds Python objects, which are serialized using pickle. |
| SchemaType | Mark a type as possibly requiring schema-level DDL for usage. |
| SmallInteger | A type for smaller int integers. |
| String | The base for all string and character types. |
| Text | A variably sized string type. |
| Time | A type for datetime.time() objects. |
| Unicode | A variable length Unicode string type. |
| UnicodeText | An unbounded-length Unicode string type. |

情報元：[https://docs.sqlalchemy.org/en/14/core/type\_basics.html#generic-camelcase-types](https://docs.sqlalchemy.org/en/14/core/type_basics.html" \l "generic-camelcase-types)

公式：[https://docs.sqlalchemy.org/en/14/core/type\_basics.html#module-sqlalchemy.types](https://docs.sqlalchemy.org/en/14/core/type_basics.html" \l "module-sqlalchemy.types)

##### 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()

情報元：https://www.sukerou.com/2018/12/sqlalchemy-onefirst.html

##### 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()

#### pyodbc

odbc接続を提供するライブラリ

例

def sql\_execute(con, slq):

cursor = con.cursor()

cursor.execute(slq)

rows = cursor.fetchall()

cursor.close()

return rows

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

con = login()

sql = '''select \*

from TEST'''

res = sql\_execute(con, sql)

for r in res:

print(r[0])

print(r[1])

## 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()

### pybleno

JavascriptでBLE通信を行うライブラリ（bleno）のpython版。

### pyserial

pip：<https://pypi.org/project/pyserial/>

公式：<https://pythonhosted.org/pyserial/index.html>

[ python ]

import serial

ser = serial.Serial( # シリアル通信の設定

port='/dev/ttyS0', # 使用するポート名の設定

baudrate= 9600,

bytesize= serial.EIGHTBITS, # SEVENBITS =7; EIGHTBITS =8

stopbits= serial.STOPBITS\_ONE,

parity= serial.PARITY\_NONE # なぜかNONE以外指定できない 19.07.10

)

インスタンスを生成するが、ポートを遅れて開く場合

import serial

ser = serial.Serial()

ser.port = '/dev/ttyUSB0'

ser.baudrate = 9600

print("Is the serial port open? ", ser.is\_open)

ser.open()

シリアルコンソールの処理も可能らしい

<https://pythonhosted.org/pyserial/shortintro.html>

Github：<https://github.com/pyserial/pyserial/blob/master/serial/tools/miniterm.py>

ser.write

ser

読み込み

方法１

while self.\_\_ser.in\_waiting < 2:

time.sleep(0.01) # 10msの遅延を追加（適宜調整可能）

data = ser.read(10) # 10byteを上限に読み込む(bytes型を返す)

data = self.\_\_ser.read(2)

方法2

bytes\_list = []

while self.\_\_ser.in\_waiting:

data = self.\_\_ser.read()

bytes\_list.append(data)

# ret\_str\_list = [("0x" + format(ret, "02x"))for ret in response]

return bytearray(b''.join(bytes\_list))

方法３

readline python 2.6以降では非推奨（サポート外）

## 制御系

### マイコン系

#### python-periphery

GPIO, LED, PWM, SPI, I2C, MMIO、シリアル通信などを使うためのライブラリ

GPIO

button=periphery.GPIO("/dev/gpiochip0", 6, "in", edge="both")

詳細：[https://python-periphery.readthedocs.io/en/latest/gpio.html#periphery.GPIO](https://python-periphery.readthedocs.io/en/latest/gpio.html" \l "periphery.GPIO)

SPI

# SPI1, SS0, Mode 0, 10MHz

spi1\_0 = SPI("/dev/spidev0.0", 0, 10000000)

# SPI1, SS1, Mode 0, 10MHz

spi1\_1 = SPI("/dev/spidev0.1", 0, 10000000)

##### リファレンス

###### read\_event()メソッド

EdgeEventを返す

edge

timestamp

###### GPIO. poll\_multiple

準備のできたデバイスを表示？？

gpios\_ready = periphery.GPIO.poll\_multiple([gpio\_in], 1)

情報元：https://github.com/vsergeev/python-periphery/blob/master/tests/test\_gpio.py

公式

<https://python-periphery.readthedocs.io/en/latest/gpio.html>

github issues <https://github.com/vsergeev/python-periphery/issues/21>

#### asyncio.windows\_events

asyncio.windows\_events import NULL

2022-05-20何のこと？

##### 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

### 信号系

keyboard

if keyboard.is\_pressed('n'):

print('n')

pynput

## AI関連

### 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。

詳細は別ドキュメント「AI関連」へ

#### 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

### 自然言語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

Janome

辞書を内包する形態素解析エンジン

SudachiPy

形態素解析ライブラリ

Gensim

文書のトピックモデルを実行するライブラリ

NLTK

## 画像・音声など

### 画像関連

#### ffmpeg

ffmpeg-pythonはffmpegを使う為のライブラリなので、動作環境にffmpegがインストールされている必要がある。

python3 -m pip install ffmpeg-python

公式：<https://kkroening.github.io/ffmpeg-python/>

#### Pillow

(PIL)

画像処理のライブラリ

##### 詳細情報

###### Exif情報の取得

from PIL.ExifTags import TAGS, GPSTAGS

##### リファレンス

###### ImageDraw

描画処理などを扱う

資料（ImageDraw）

公式：<https://pillow.readthedocs.io/en/stable/reference/ImageDraw.html>

#### PaddleOCR

かなり精度がよいらしい

#### PyOCR

#### EasyOCR

例：

ocr\_reader = easyocr.Reader(

['en'],

gpu = False,

model\_storage\_directory='models',

)

model\_storage\_directory

Windows：C:\Users\[ユーザー名]\.EasyOCR

補足：内部でtorchを使っているので、torchと同じ場所かも

公式：<http://www.jaided.ai/easyocr/documentation/>

#### tesseract

OCRのソフト。pipのパッケージではない為、厳密にはpythonではない。

性能はかなり悪いらしい

#### opencv

詳細は別ドキュメントAI関連　に記述。

### 動画系

#### MoviePy

動画編集を行う

内部でFFmpegを使っており、自動的にインストールされる

pip install moviepy

公式：<https://zulko.github.io/moviepy/index.html>

具体例：<https://zulko.github.io/moviepy/examples/examples.html>

Github：<https://github.com/Zulko/moviepy>

pypi：<https://pypi.org/project/moviepy/>

トリミング、サイズ変更と言った一般的な処理、他にも文字との合成、

例

from moviepy.editor import VideoFileClip

from moviepy.video.fx.resize import resize

from moviepy.video.fx.crop import crop

clip = VideoFileClip('example.mp4')

print(f'fps: {clip.fps} size:{clip.size}')

RATIO = 0.75

# 時間的な切り抜き

# clip = clip.subclip(0, 60)

# clip = clip.subclip('00:00:00', '00:00:10')

# トリミング

clip = crop(clip=clip, x1=5, y1=120, x2=655, y2=618)

# サイズ変更

clip = resize(clip, (clip.w \* RATIO, clip.h \* RATIO))

clip.write\_videofile(

"converted\_video.mp4",

fps=5

)

##### リファレンス

###### clip

cutout

crossfadein

crossfadeout

set\_starts

set\_duration

パラメータ

|  |  |  |  |
| --- | --- | --- | --- |
| 名前 | データ型 | 概要 | 初期値 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

subclip

具体例

clip = clip.subclip('00:01:00.000', -10)

clip.subclip(t\_start=10)

パラメータ

|  |  |  |  |
| --- | --- | --- | --- |
| 名前 | データ型 | 概要 | 初期値 |
| t\_start | int/str |  |  |
| t\_end | int/str | 負の値の場合は最後からn秒となる |  |
|  |  |  |  |

write\_videofile

パラメータ

|  |  |  |  |
| --- | --- | --- | --- |
| 名前 | データ型 | 概要 | 初期値 |
| fps |  | フレームレートを指定 |  |
| codec |  |  |  |
| autio\_codec |  |  |  |

bitrate

final\_clip.write\_videofile("final.mp4", bitrate="5000k")

###### video.compositing.concatenate

concatenate\_videoclips

複数の動画を結合する

from moviepy.editor import VideoFileClip

from moviepy.video.compositing.concatenate import concatenate\_videoclips

clip = VideoFileClip("sample.mp4")

sub1 = clip.subclip(3, 5)

sub2 = clip.subclip(t\_start='00:00:00.800', t\_end = '00:00:06.000')

clip = concatenate\_videoclips([sub1, sub2])

fade\_duration = 1 # 1-second fade-in for each clip

clips = [clip.crossfadein(fade\_duration) for clip in clips]

final\_clip = concatenate\_videoclips(clips, padding = -fade\_duration)

###### video.fx

resize

from moviepy.video.fx.resize import resize

crop

from moviepy.video.fx.crop import crop

# 始点座標 – 終点座標

clip = crop(clip=clip, x1=100, y1=100, x2=300, y2=300)

# 始点座標と幅、高さ

clip = crop(clip=clip, x1=100, y1=100, width=300, height=300)

パラメータ

|  |  |  |  |
| --- | --- | --- | --- |
| 名前 | データ型 | 概要 | 初期値 |
| x1 |  |  |  |
| y1 |  |  |  |
| x2 |  | 幅 |  |
|  |  |  |  |

###### その他

crossdissolve

## 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

### smopy

## 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：

|  |  |  |  |
| --- | --- | --- | --- |
|  | インストールされる場所 | 影響範囲 |  |
| python3 -m pip install |  | 現在のユーザー |  |
| pip3 install | ~/.local/lib/python[ver]/site-packages | 現在のユーザー |  |
| sudo pip3 install | /usr/local/lib/python[ver]]/dist-packages | システム全体 | /usr/localは、ローカル機（全体）で使用されるファイルを格納 |

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" numpy

補足

# powershell

$env:HTTPS\_PROXY="192.50.200.3:8080"

-q；--quiet 結果を出力しない

-H; 環境変数HOMEをrootユーザーのホームディレクトリに変更してコマンドを実行

-e；

--no-deps 依存関係を無視してインストール。.whファイルからインストールする時の標準

pip install --no-deps numpy-1.21.2-cp37-cp37m-win\_amd64.whl

-o； --outdate バージョンが最新バージョンでないパッケージを一覧表示

### 具体例

#### pipの初期化

pip freeze > uninstall.txt

pip uninstall -y -r uninstall.txt

Linuxの場合

pip freeze | xargs pip uninstall -y

#### tar.gzから.whファイルを作成

先に.tar.gzファイルをダウンロードしておく

#　必要パッケージのインストール

pip install wheel setuptools

# 展開して移動

tar xfvz python-dateutil-2.8.1.tar.gz

cd python-dateutil-2.8.1

# コンパイル

python setup.py bdist\_wheel

> この例の場合、python-dateutil-2.8.1/dist に.whファイルが出来上がる

補足：pip install python-dateutil-2.8.1.tar.gz を行う時もwheelとsetuptoolsが必要

### トラブルシューティング

#### ModuleNotFoundError: No module named 'pip.\_vendor.tenacity.retry'

pipが壊れている可能性がある

python -m ensurepip --upgrade

## 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

Pythonのパッケージマネージャの一つ

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

### データサイエンス／数学系

#### 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

#### Polars

<https://github.com/kunishou/Polars_100_knocks>

#### numpy

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

#### データサイエンス

データ前処理の100本ノック：<https://github.com/The-Japan-DataScientist-Society/100knocks-preprocess>

#### 機械学習

本だけど

<https://www.shuwasystem.co.jp/book/9784798063416.html>

### 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](https://docs.conda.io/en/latest/miniconda.html" \l "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

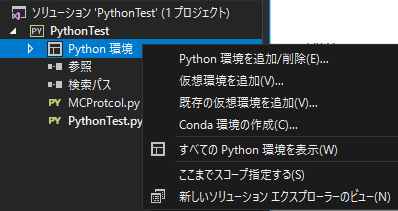
〇〇Lazyという表現のパッケージ名をたまに見かける。

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。

## .NETとの比較

### operator

==, >, < の類はoperatorという関数で

公式

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

## 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。

## python系雑多情報

### python Golf

補足：Golf – 出来るだけ少ない文字数でコードを記述する競技。またはその書き方。

<https://qiita.com/shibacho/items/37413a5d479d8bb57764>

## 疑問

covariant, contravariant？

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

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

## コンパイラ

pyinstaller

colabでpythonのバージョンを変更する方法

!sudo update-alternatives –config python3

補足

colabではpythonが常に起動しているような状態

!を付ける事でbashコマンドを実行する