

【Python】応用

目次

- パッケージ全版編
 - パッケージ（ライブラリ）について
- 文字列編
 - 正規表現
 - 形態素解析
- 数学編
 - 時間
 - 乱数
 - NumPy——一般、配列全般
 - NumPy——生成
 - NumPy——変換
 - グラフ（プロット）
- システム・ファイル・ディレクトリ編
 - パス
 - ファイル・ディレクトリ操作
 - システム
 - プログラムとプロセス
 - 並行処理
 - ログ出力
- ファイル編集（詳解）編
 - Excel
 - Word
 - CSVファイル
 - JSONファイル
 - PDF編集
 - XMLファイル
- データ分析編
 - Pandas——一般、シリーズ・データフレーム共通
 - Pandas——シリーズ
 - Pandas——データフレーム
 - ORM

【Python】応用

目次

- パッケージ全版編
 - パッケージ（ライブラリ）について
- 文字列編
 - 正規表現
 - 形態素解析
- 数学編
 - 時間
 - 乱数
 - NumPy——一般、配列全般
 - NumPy——生成
 - NumPy——変換
 - グラフ（プロット）
- システム・ファイル・ディレクトリ編
 - パス
 - ファイル・ディレクトリ操作
 - システム
 - プログラムとプロセス
 - 並行処理
 - ログ出力
- ファイル編集（詳解）編
 - Excel
 - Word
 - CSVファイル
 - JSONファイル
 - PDF編集
 - XMLファイル
- データ分析編
 - Pandas——一般、シリーズ・データフレーム共通
 - Pandas——シリーズ
 - Pandas——データフレーム
 - ORM

- 通信編
 - メール送信
 - URL
 - サーバへのアクセス
 - スクレイピング
 - Webアプリ作成（Flask）
- 画像・音声・動画編
 - 簡単な画像処理
 - 画像認識
 - 音声処理
 - 音楽解析
 - MIDI
 - 動画編集
 - YouTube
- GUI編
 - 仮想ディスプレイ
 - マウス操作、キーボード操作など

- 通信編
 - メール送信
 - URL
 - サーバへのアクセス
 - スクレイピング
 - Webアプリ作成（Flask）
- 画像・音声・動画編
 - 簡単な画像処理
 - 画像認識
 - 音声処理
 - 音楽解析
 - MIDI
 - 動画編集
 - YouTube
- GUI編
 - 仮想ディスプレイ
 - マウス操作、キーボード操作など

モジュール、パッケージ（ライブラリ）全般編

■モジュールについて

- ▶ モジュールを直接実行

■パッケージ（ライブラリ）について

- ▶ ※ pip とはPythonのパッケージを管理するためのツールである。
- ▶ ※ ネットで検索していて `pip3` とか出てきたら → `pip` で構わない
- ▶ ※ 実は `pip` ではなく `python3 -m pip` とするのが推奨されている。
- ▶ pip (conda) 自体を更新
- ▶ パッケのインストール
- ▶ バージョン指定してインストール
- ▶ パッケのアンインストール
- ▶ 全パッケのリスト（ver付）
- ▶ アプデが必要なパッケのリスト

モジュール、パッケージ（ライブラリ）全般編

■モジュールについて

- ▶ モジュールを直接実行 `$ python -m モ※` ※パスではないので `.py` も不要

■パッケージ（ライブラリ）について

- ▶ ※ pip とはPythonのパッケージを管理するためのツールである。
- ▶ ※ ネットで検索していて `pip3` とか出てきたら → `pip` で構わない
- ▶ ※ 実は `pip` ではなく `python3 -m pip` とするのが推奨されている。
- ▶ pip (conda) 自体を更新 `$ pip install -U pip` `$ conda update conda`
- ▶ パッケのインストール `$ pip install pack1 ...` `$ conda install pack1 ...`
- ▶ バージョン指定してインストール `$ pip install pack==ver` `$ conda install pack==ver`
- ▶ パッケのアンインストール `$ pip uninstall package` `$ conda uninstall package`
- ▶ 全パッケのリスト（ver付） `$ pip list` か `$ pip freeze` `$ conda list`
- ▶ アプデが必要なパッケのリスト `$ pip list -o`

▶ あるパッケージをアプデ	
▶ バージョン指定してアプデ	
▶ すべてのパッケージをアプデ	
▶ パッケージの場所を確認	
▶ パッケージ検索対象フォルダを確認	
▶ ☆ 標準ライブラリに含まれる代表的なモジュール	
▶ ☆ 代表的なサードパーティーライブラリ	
▶ ※ モジュールやパッケージ（ライブラリ） と同名の .py ファイルを作ってはならない！	
▶ モジュールの取り込み	
▶ PYファの一段上の階層のモジュールの取り込み	<code>import ..モジュール from .. import モジュール</code>
▶ 文字列でモジュールを取り込む	
▶ モジュールで利用できる関数やクラスの一覧	
▶ ☆ インポート元のPYファのパス	

■自作パッケージ（自作ライブラリ）について

- ▶ ☆ 自作ライブラリ的设计
- ▶ ☆ 自作パッケージ的设计
- ▶ ☆ 自作ライブラリの**パスを通す**（パッケージ検索対象フォルダとして新規追加）

文字列編

■正規表現

▶ ☆ 主なメタ文字	
▶ ※ Pythonの正規表現の規格は「Perlの正規表現」とほぼ同じ。	
▶ ※ 正規表現のパターン文字列 (<i>regex</i>) で <code>r</code> を冒頭に付けておくのは、 <code>\n</code> や <code>\t</code> など Python 標準によって勝手に置き換えられるものを置き換えさせないため。	
▶ reモジュール	
▶ Matchオブ	
▶ パターン全体にマッチした値	

▶ あるパッケージをアプデ	<code>\$ pip install -U package</code> <code>\$ conda update package</code>
▶ バージョン指定してアプデ	<code>\$ pip install -U pack==ver</code> ※ Anaconda には ない
▶ すべてのパッケージをアプデ	<code>☆☆☆</code> <code>\$ conda update --all</code>
▶ パッケージの場所を確認	<code>\$ pip show package</code> ※ Anaconda には ない
▶ パッケージ検索対象フォルダを確認	<code>import sys</code> <code>print(sys.path)</code>
▶ ☆ 標準ライブラリに含まれる代表的なモジュール	
▶ ☆ 代表的なサードパーティーライブラリ	
▶ ※ モジュールやパッケージ（ライブラリ） と同名の .py ファイルを作ってはならない！	
▶ モジュールの取り込み	<code>import モジュール from モジュール import 関数や変数</code> (as 別名)
▶ PYファの一段上の階層のモジュールの取り込み	<code>import ..モジュール from .. import モジュール</code>
▶ 文字列でモジュールを取り込む	<code>from importlib import import_module</code> <code>module1 = import_module('モジュール名')</code> <code>module1.関数()</code> や変数
▶ モジュールで利用できる関数やクラスの一覧	<code>import モジュール</code> <code>dir(モジュール)</code> ※文字列型でない
▶ ☆ インポート元のPYファのパス	

■自作パッケージ（自作ライブラリ）について

- ▶ ☆ 自作ライブラリ的设计
- ▶ ☆ 自作パッケージ的设计
- ▶ ☆ 自作ライブラリの**パスを通す**（パッケージ検索対象フォルダとして新規追加）

文字列編

■正規表現

▶ ☆ 主なメタ文字	
▶ ※ Pythonの正規表現の規格は「Perlの正規表現」とほぼ同じ。	
▶ ※ 正規表現のパターン文字列 (<i>regex</i>) で <code>r</code> を冒頭に付けておくのは、 <code>\n</code> や <code>\t</code> など Python 標準によって勝手に置き換えられるものを置き換えさせないため。	
▶ reモジュール	<code>import re</code>
▶ Matchオブ	<code>m = re.search(regex, str)</code> ※マッチしないとNoneが返る
▶ パターン全体にマッチした値	<code>m.group(0)</code>

▶ <i>n</i> 番目のキャプチャにマッチした値	
▶ マッチした値の開始・終了位置	
▶ <i>regex</i> が <i>str</i> にマッチするか	
▶ <i>regex</i> が <i>str</i>全体 にマッチするか	
▶ <i>regex</i> にマッチする全箇所	
▶ <i>regex</i> とマッチ箇所を置換	
▶ ☆ マッチした部分を使って置換	
▶ 何個置換したかを取得	

■形態素解析

▶ MeCab エンジン	
▶ 準備	
▶ 形態素のイテレータ(似)	
▶ 形態素の文字列じたい	
▶ 形態素の情報(品詞など)	
▶ ☆ (クラス化で便利に)	

数学編

■時間

time、datetime

▶ time モ	
▶ ある時間だけ待つ	
▶ UNIX時間を取得	
▶ datetime モ	
▶ 今日の日付オブを取得	
▶ 現在の日時オブを取得	
▶ 現在の年、...、秒を取得	
▶ 文字列を日付オブに	
▶ 日時や日付を比較	

▶ <i>n</i> 番目のキャプチャにマッチした値	m.group(<i>n</i>) ※ <i>n</i> ≥ 1
▶ マッチした値の開始・終了位置	m.start() ・ m.end() ※ m.span() で (開始位置, 終了位置)
▶ <i>regex</i> が <i>str</i> にマッチするか	bool(re.search(<i>regex</i> , <i>str</i>))
▶ <i>regex</i> が <i>str</i>全体 にマッチするか	bool(re.compile(<i>regex</i>).fullmatch(<i>str</i>))
▶ <i>regex</i> にマッチする全箇所	re.findall(<i>regex</i> , <i>str</i>) ※: リスト
▶ <i>regex</i> とマッチ箇所を置換	re.sub(<i>regex</i> , <i>replacement</i> , <i>str</i>)
▶ ☆ マッチした部分を使って置換	
▶ 何個置換したかを取得	re.subn(<i>regex</i> , <i>replacement</i> , <i>str</i>)

■形態素解析

▶ MeCab エンジン	☆☆☆ import Mecab
▶ 準備	tagger = Mecab.Tagger()
▶ 形態素のイテレータ(似)	tagger.parse("") # ※ parseToNode() の不具合回避の為に必須 node = tagger.parseToNode(文章) (node = node.next で次へ)
▶ 形態素の文字列じたい	node.surface
▶ 形態素の情報(品詞など)	node.feature
▶ ☆ (クラス化で便利に)	

数学編

■時間

time、datetime

▶ time モ	import time
▶ ある時間だけ待つ	time.sleep(0.3)
▶ UNIX時間を取得	ut = time.time()
▶ datetime モ	from datetime import datetime, date, timedelta
▶ 今日の日付オブを取得	dd = date.today()
▶ 現在の日時オブを取得	dt = datetime.now()
▶ 現在の年、...、秒を取得	dt.year .month .day .hour .minute .second .microsecond
▶ 文字列を日付オブに	datetime.strptime('0000-00-00', "%Y-%m-%d")
▶ 日時や日付を比較	ddやdtどうしを比較演算子 (> など) で結ぶ

- ▶ 日時や日付を足し引き
- ▶ ☆ main()関数の実行時間を調べる

dateutil、pytz（datetimeを併用）

- ▶ dateutil モ
- ▶ pytz モ
- ▶ 日付を足し引き
- ▶ 日付の一部を書き換え
- ▶ 月末にする

■乱数

- ▶ random モジュール
- ▶ $0 \leq \text{乱数} < 1$ の生成
- ▶ $\min \leq \text{乱数} < \max \in \mathbb{R}$
- ▶ $\min \leq \text{乱数} \leq \max \in \mathbb{N}$

■NumPy——一般、配列全般

- ▶ NumPy ライブラリ
- ▶ ※ NumPy は文字列も扱えるが、均一な数値データを扱うのに最も適している。
- ▶ ※ `IndexError: index 8 is out of bounds for axis 0 with size 8` などと出るのは、配列の範囲外を切り抜いたとき。

- ▶ ※ 配列（numpy.ndarray 型）はリスト（list 型）とは違う。
- ▶ ※ 配列はミュータブル。
- ▶ ※ 配列のスライスはビュー（覗き）となる。

- ▶ リストを配列に変換
- ▶ 配列をリストに変換
- ▶ 配列の形状
- ▶ 配列の次元
- ▶ 行・要素を参照
- ▶ 行・要素を任意順で取得
- ▶ 配列がビューか否か
- ▶ 配列のコピー

- ▶ 日時や日付を足し引き
- ▶ ☆ main()関数の実行時間を調べる

dateutil、pytz（datetimeを併用）

- ▶ dateutil モ
- ▶ pytz モ
- ▶ 日付を足し引き
- ▶ 日付の一部を書き換え
- ▶ 月末にする

■乱数

- ▶ random モジュール
- ▶ $0 \leq \text{乱数} < 1$ の生成
- ▶ $\min \leq \text{乱数} < \max \in \mathbb{R}$
- ▶ $\min \leq \text{乱数} \leq \max \in \mathbb{N}$

■NumPy——一般、配列全般

- ▶ NumPy ライブラリ
- ▶ ※ NumPy は文字列も扱えるが、均一な数値データを扱うのに最も適している。
- ▶ ※ `IndexError: index 8 is out of bounds for axis 0 with size 8` などと出るのは、配列の範囲外を切り抜いたとき。

- ▶ ※ 配列（numpy.ndarray 型）はリスト（list 型）とは違う。
- ▶ ※ 配列はミュータブル。
- ▶ ※ 配列のスライスはビュー（覗き）となる。
- ▶ リストを配列に変換
- ▶ 配列をリストに変換
- ▶ 配列の形状
- ▶ 配列の次元
- ▶ 行・要素を参照
- ▶ 行・要素を任意順で取得
- ▶ 配列がビューか否か
- ▶ 配列のコピー

▶ 条件を満たす要素の座標	
---------------	--

■NumPy——生成

スカラーを生成

▶ 円周率	
▶ ネイピア数	

1次元配列を生成

▶ range()的な	
▶ bytesオブを1次元配列に	
▶ 等差数列	

配列を生成

▶ 乱数 (0～1, 一様分布)	
▶ 乱数 (標準正規分布)	
▶ 乱数 (一般の正規分布)	
▶ 乱数 (二項・ベータ・ガンマ・カイ二乗分布)	
▶ 空の配列	
▶ 全要素が同じ値の配列	

■NumPy——変換

配列 → スカラー

▶ 最大値・最小値	
-----------	--

配列 → 1次元配列

▶ 条件に合う要素の値たち	
▶ 平坦化	

配列 → 同形の配列

▶ 値を丸める	
▶ 要素どうしの加減算	
▶ 要素どうしで掛け算	

▶ 条件を満たす要素の座標	list(zip(*np.where(a < 7))	※: 座標タプルのリスト
---------------	----------------------------	--------------

■NumPy——生成

スカラーを生成

▶ 円周率	np.pi
▶ ネイピア数	np.e

1次元配列を生成

▶ range()的な	np.arange(○)
▶ bytesオブを1次元配列に	np.frombuffer(<i>bytes</i> , dtype=np.uint8)
▶ 等差数列	np.linspace(<i>start</i> , <i>stop</i> , <i>num</i>) ※デフォだと <i>stop</i> 含む np.arange(<i>start</i> , <i>stop</i> , <i>step</i>) ※ " 含まない

配列を生成

▶ 乱数 (0～1, 一様分布)	np.random.rand(* <i>shape</i>) か np.random.uniform(size= <i>shape</i>)
▶ 乱数 (標準正規分布)	np.random.randn(* <i>shape</i>)
▶ 乱数 (一般の正規分布)	np.random.normal(<i>mean</i> , <i>std</i> , <i>shape</i>)
▶ 乱数 (二項・ベータ・ガンマ・カイ二乗分布)	np.random.binomial(<i>n</i> , <i>p</i> , <i>shape</i>) ・ .beta(<i>a</i> , <i>b</i> , <i>m</i> <i>shape</i>) ・ .gamma(<i>k</i> , <i>θ</i> , <i>shape</i>) ・ .chisquare(<i>k</i> , <i>shape</i>)
▶ 空の配列	np.empty(<i>shape</i>) .empty_like(<i>a</i>)
▶ 全要素が同じ値の配列	np.zeros(<i>shape</i>) .ones(<i>shape</i>) .full(<i>shape</i> , <i>num</i>) np.zeros_like(<i>a</i>) .ones_like(<i>a</i>) .full_like(<i>a</i> , <i>num</i>)

■NumPy——変換

配列 → スカラー

▶ 最大値・最小値	a.max() ・ a.min()
-----------	-------------------

配列 → 1次元配列

▶ 条件に合う要素の値たち	a[a < 7] ※ビュー
▶ 平坦化	a.ravel() か a.flatten() ※1個目はビューの時アリ

配列 → 同形の配列

▶ 値を丸める	np.round(a, 小数点以下の桁数)
▶ 要素どうしの加減算	a1+ a2 a1 - a2
▶ 要素どうしで掛け算	np.multiply(a1, a2)

▶ スカラー倍	
▶ 平方根	
▶ 累乗	
▶ 指数関数	
▶ 常用・自然・二進対数	
▶ radを度に・度をradに	
▶ 三角関数	
▶ 逆三角関数	
▶ 比較演算して真偽値に	

配列 → 異形かもしれない配列

▶ 転置	
▶ 内積	
▶ ある次元の要素を逆順に	
▶ 隣接要素で差をとる	
▶ 和をとる	
▶ 平均をとる	
▶ 最大値・最小値をとる	
▶ 標準偏差・分散をとる	
▶ 要素数を保持した変形	
▶ 各要素を配列にし次元増	
▶ 配列を結合	

■グラフ（プロット）

▶ matplotlib ラ pyplot モ	
▶ 折れ線グラフ	
▶ 散布図	

設定

▶ フォントの種類（全体）	
▶ フォントの種類（数式）	

▶ スカラー倍	a * 数値
▶ 平方根	np.sqrt(a) か a ** 0.5
▶ 累乗	np.power(底, 指数) か 底 ** 指数 ※底, 指数には配列も可
▶ 指数関数	np.exp(a)
▶ 常用・自然・二進対数	np.log10(a) ・ .log(a) ・ .log2(a)
▶ radを度に・度をradに	np.rad2deg(a) ・ .deg2rad(a)
▶ 三角関数	np.sin(a) .cos(a) .tan(a) ※radで渡すこと
▶ 逆三角関数	np.arcsin(a) .arccos(a) .arctan(a) .arctan2(a1, a2) ※: rad
▶ 比較演算して真偽値に	a < 7 a != 0 (a < 7) & (a != 0) a1 < a2

配列 → 異形かもしれない配列

▶ 転置	a.T か np.array(list(zip(*a))) ※1個目のはビュー
▶ 内積	np.dot(a1, a2)
▶ ある次元の要素を逆順に	a[::-1] a[:, ::-1] のようにスライスを使う
▶ 隣接要素で差をとる	np.diff(a, axis= <i>axis</i>) ※ n=n とすると n 回繰り返してくれる
▶ 和をとる	a.sum(axis= <i>axis</i>)
▶ 平均をとる	a.mean(axis= <i>axis</i>)
▶ 最大値・最小値をとる	a.max(axis= <i>axis</i>) ・ a.min(axis= <i>axis</i>)
▶ 標準偏差・分散をとる	a.std(axis= <i>axis</i>) ・ a.var(axis= <i>axis</i>)
▶ 要素数を保持した変形	a = a.reshape(<i>shape</i>) ※ビュー
▶ 各要素を配列にし次元増	a[:, :, np.newaxis] ※例えば [1 2 3] → [[1] [2] [3]] になる
▶ 配列を結合	np.concatenate([a1, ...], axis= <i>axis</i>) ※ axis は 0 から ※第 axis 次元の要素数が増加する ※それ以外の次元の要素数は a1 , a2 , ... で同一の必要

■グラフ（プロット）

▶ matplotlib ラ pyplot モ	\$ pip install matplotlib import matplotlib.pyplot as plt
▶ 折れ線グラフ	plt.plot(<i>x_sequence</i> , <i>y_sequence</i>) #
▶ 散布図	plt.scattar(<i>x_sequence</i> , <i>y_sequence</i>) #

設定

▶ フォントの種類（全体）	plt.rcParams['font.family'] = 'Times New Roman'
▶ フォントの種類（数式）	plt.rcParams['mathtext.fontset'] = 'stix'

▶ フォントサイズ（全体）	
▶ ※ ネットで調べてて ax と出てきたら	<code>ax = plt.gca()</code> で対応。
▶ ☆ 軸の設定のサマリーチャート	
▶ 軸ラベル	
▶ フォントサイズ（目盛）	
▶ 目盛を内向きに	
▶ 目盛を非表示	
▶ 目盛を指数表記に	
▶ グラフのタイトル	
▶ 凡例を表示	

数式

▶ ※ 公式ドキュメントはここ。	
▶ 数式を書く	

数式

▶ ※ 公式ドキュメントはここ。	
▶ 数式を書く	<code>r'\$ここに数式の書式文字列を書きます\$'</code>

▶ フォントサイズ（全体）	<code>plt.rcParams['font.size'] = 15</code>
▶ ※ ネットで調べてて ax と出てきたら	<code>ax = plt.gca()</code> で対応。
▶ ☆ 軸の設定のサマリーチャート	
▶ 軸ラベル	<code>plt.xlabel('時間 [s]')</code>
▶ フォントサイズ（目盛）	<code>plt.rcParams['xtick.labelsize'] = 9</code>
▶ 目盛を内向きに	<code>plt.rcParams['xtick.direction'] = 'in'</code>
▶ 目盛を非表示	<code>plt.xticks([])</code>
▶ 目盛を指数表記に	<code>ax.ticklabel_format(style='sci', axis='x', scilimits=(0,0))</code>
▶ グラフのタイトル	<code>plt.title('タイトル')</code>
▶ 凡例を表示	<code>plt.legend()</code>

数式

▶ ※ 公式ドキュメントはここ。	
▶ 数式を書く	<code>r'\$ここに数式の書式文字列を書きます\$'</code>

数式

▶ ※ 公式ドキュメントはここ。	
▶ 数式を書く	<code>r'\$ここに数式の書式文字列を書きます\$'</code>

システム・ファイル・ディレクトリ編

▶ pathlib モ Path ク	
▶ 文字列をPath型に	
▶ パスを文字列にする	
▶ 属する（親の）ディのパス	
▶ 実行中.pyのディのパス	
▶ パス結合	
▶ 作業ディのパス	
▶ ホームディのパス	
▶ 相対パスを絶対パスに	<code>p = p.resolve()</code>
▶ ベースネーム	
▶ 拡張子なしのベースネーム	
▶ 拡張子	

システム・ファイル・ディレクトリ編

▶ pathlib モ Path ク	<code>from pathlib import Path</code>
▶ 文字列をPath型に	<code>p = Path('パスの文字列')</code>
▶ パスを文字列にする	<code>p.as_posix()</code>
▶ 属する（親の）ディのパス	<code>p.parent</code>
▶ 実行中.pyのディのパス	<code>Path(__file__).parent</code>
▶ パス結合	<code>p / 'dir1/sample.txt'</code>
▶ 作業ディのパス	<code>Path.cwd()</code> ※ Unix でいう <code>.</code> ※クラスメソッド
▶ ホームディのパス	<code>Path.home()</code> ※ Unix でいう <code>~</code> ※クラスメソッド
▶ 相対パスを絶対パスに	<code>p = p.resolve()</code>
▶ ベースネーム	<code>p.name</code>
▶ 拡張子なしのベースネーム	<code>p.stem</code>
▶ 拡張子	<code>p.suffix</code> ※ <code>.</code> 付きで返る。


▶ ベースネームを置換	
▶ 拡張子を置換	

パスの指す実際のファやディの情報を取得・変更

▶ 有するファディのパスの一覧	
▶ 有する 特定の拡張子の ファのパスの一覧	
▶ パス（のもの）が存在するか	
▶ パスが存在するファ・ディか	
▶ パスがWin上で予約済みか	
▶ パスのファイルサイズ	
▶ パスの（指す）ディを削除	
▶ パスのファを削除	
▶ パスのディを作成	
▶ 深いパスのディまで一気に	
▶ ☆ 新規の一時ファイルを作成し、用が済めば削除	

■ファイル・ディレクトリ操作

ファイル操作

▶ OS モジュール	
▶ ファが存在するか	
▶ ファ作成	
▶ ファ削除	
▶ 属するディの絶対パス	<code>os.path.dirname(<i>path</i>)</code>
▶ 実行中.pyのディの絶対パス	<code>os.path.dirname(__file__)</code>
▶ ベースネーム	
▶ 拡張子なしのベースネーム	<code>os.path.splitext(os.path.basename(<i>path</i>))[0]</code>
▶ 拡張子	<code>os.path.splitext(os.path.basename(<i>path</i>))[1]</code> ※  付き
▶ パス結合	<code>os.path.join(<i>dirpath1</i>, <i>dirpath2</i>, ..., <i>path</i>)</code>
▶ 名称変更	
▶ shutil モジュール	


▶ ベースネームを置換	<code>p = p.with_name('sample2.txt')</code>
▶ 拡張子を置換	<code>p = p.with_suffix('.txt')</code>

パスの指す実際のファやディの情報を取得・変更

▶ 有するファディのパスの一覧	<code>p.iterdir()</code> ※ list() か sorted() でくくるべき。
▶ 有する 特定の拡張子の ファのパスの一覧	直属のファのみ対象なら <code>p.glob('*.py')</code> ※ " より下のファも対象なら <code>p.glob('**/*.py')</code> か <code>p.rglob('*.py')</code>
▶ パス（のもの）が存在するか	<code>p.exists()</code>
▶ パスが存在するファ・ディか	<code>Path.is_file(p)</code> ・ <code>Path.is_dir(p)</code>
▶ パスがWin上で予約済みか	<code>p.is_reserved()</code> ※: Boolean
▶ パスのファイルサイズ	<code>p.stat().st_size</code>
▶ パスの（指す）ディを削除	<code>p.rmdir()</code> # ※ そのディは空である必要。
▶ パスのファを削除	<code>p.unlink()</code> #
▶ パスのディを作成	<code>p.mkdir()</code> # ※ 既存ならエラー
▶ 深いパスのディまで一気に	<code>p.mkdir(parents=True, exist_ok=True)</code> ※ " <code>/"</code> でもスルー
▶ ☆ 新規の一時ファイルを作成し、用が済めば削除	

■ファイル・ディレクトリ操作

ファイル操作

▶ OS モジュール	<code>import os</code>
▶ ファが存在するか	<code>os.path.exists(<i>path</i>)</code> か <code>os.path.isfile(<i>path</i>)</code>
▶ ファ作成	<code>f = open(<i>path</i>, 'w')</code> <code>f.write("")</code> <code>f.close</code>
▶ ファ削除	<code>os.remove(<i>path</i>)</code> ※ ファが存在しないとエラー
▶ 属するディの絶対パス	<code>os.path.dirname(<i>path</i>)</code>
▶ 実行中.pyのディの絶対パス	<code>os.path.dirname(__file__)</code>
▶ ベースネーム	<code>os.path.basename(<i>path</i>)</code>
▶ 拡張子なしのベースネーム	<code>os.path.splitext(os.path.basename(<i>path</i>))[0]</code>
▶ 拡張子	<code>os.path.splitext(os.path.basename(<i>path</i>))[1]</code> ※  付き
▶ パス結合	<code>os.path.join(<i>dirpath1</i>, <i>dirpath2</i>, ..., <i>path</i>)</code>
▶ 名称変更	<code>os.rename(<i>path</i>, <i>newPath</i>)</code> #
▶ shutil モジュール	<code>import shutil</code>

▶ 移動	
▶ 名前を変更して移動	
▶ メタデータを除いてコピー	
▶ メタデータを含めてコピー	
▶ ☆ 新規の一時ファイルとして複製し、用が済めば削除	
▶ ☆ mp3ファのプロパティ（タグ情報）の変更	
▶ ☆ あるディが有するファすべてを処理してゆく	
▶ ☆ あるディが有する 特定の拡張子の ファすべてを処理してゆく	

ディレクトリ操作

▶ 作業ディの絶対パス	
▶ 作業ディを変更	
▶ ディが存在するか	
▶ ディ作成	
▶ 深いディまで一気に作成	
▶ 中身ごとディを削除	
▶ （空の）ディを削除	
▶ 親のディの絶対パス	<code>os.path.dirname(path)</code>
▶ 中身を空にする	
▶ 中のファおよびディのリスト	
▶ 中のファのリスト	
▶ 中のディのリスト	

■システム

▶ 環境変数の値を取得	
▶ 環境変数の値を一時的に追加や上書き	
▶ ※ 環境変数をプログラムごとに管理したい場合は、 <code>.env</code> の仕組みを使おう	
▶ PYファ中でシステムコマンドを実行	
▶ PCの名前	

■プログラムとプロセス

▶ 移動	<code>shutil.move('filePath', 'newPath') #</code>
▶ 名前を変更して移動	<code>shutil.move('filePath', 'dirPath/newFileName') #</code>
▶ メタデータを除いてコピー	<code>shutil.copy(moveと同じ引数) #</code>
▶ メタデータを含めてコピー	<code>shutil.copy2(moveと同じ引数) #</code>
▶ ☆ 新規の一時ファイルとして複製し、用が済めば削除	
▶ ☆ mp3ファのプロパティ（タグ情報）の変更	
▶ ☆ あるディが有するファすべてを処理してゆく	
▶ ☆ あるディが有する 特定の拡張子の ファすべてを処理してゆく	

ディレクトリ操作

▶ 作業ディの絶対パス	<code>os.getcwd()</code>
▶ 作業ディを変更	<code>os.chdir(path) #</code>
▶ ディが存在するか	<code>os.path.exists(path)</code> か <code>os.path.isdir(path)</code>
▶ ディ作成	<code>os.mkdir(path)</code> ※ 既存ならエラー
▶ 深いディまで一気に作成	<code>os.makedirs(path, exist_ok=True)</code> ※ 既存でもスルー
▶ 中身ごとディを削除	<code>shutil.rmtree(path)</code>
▶ （空の）ディを削除	<code>shutil.rmdir(path)</code> ※ 空じゃなければエラー
▶ 親のディの絶対パス	<code>os.path.dirname(path)</code>
▶ 中身を空にする	<code>shutil.rmtree(ノッ)</code> <code>os.mkdir(ノッ)</code>
▶ 中のファおよびディのリスト	<code>os.listdir(path)</code>
▶ 中のファのリスト	<code>[f for f in os.listdir(ノッ) if os.path.isfile(os.path.join(ノッ, f))]</code>
▶ 中のディのリスト	<code>[f for f in os.listdir(ノッ) if os.path.isdir(os.path.join(ノッ, f))]</code>

■システム

▶ 環境変数の値を取得	・ <code>os.environ['環境変数']</code> ・ <code>os.environ.get('環境変数', デフォルト値)</code> ※ (代) <code>os.getenv</code>
▶ 環境変数の値を一時的に追加や上書き	<code>import os</code> <code>os.environ['環境変数'] = '値'</code>
▶ ※ 環境変数をプログラムごとに管理したい場合は、 <code>.env</code> の仕組みを使おう	
▶ PYファ中でシステムコマンドを実行	<code>!</code> をコマンドの前につける
▶ PCの名前	<code>import socket</code> <code>socket.gethostname()</code>

■プログラムとプロセス

▶ 実行中のPythonプログラムのプロセスID `os.getpid()`

プロセスの生成（ほかのプログラムの実行）

▶ ※ Python以外のプログラムファイルやシェルコマンド（Windowsならcmdコマンド）を実行できる。

▶ subprocess モジュール `import subprocess`

▶ (ふつうに) 同期で命令文実行 `proc = subprocess.run(コマンドのリスト, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)`

▶ エラー時の出力を取得 `if proc.returncode: proc.stdout.decode('Shift-JIS')`

▶ (ふつうに) 非同期で命令文実行 `proc = subprocess.Popen(コマンドのリスト, ...)`

▶ プロセスを強制終了 `proc.kill()`

▶ 何らかのアプリを起動 `subprocess.run(アプリのパス, ...)`

▶ ファイルをあるアプリで開く `subprocess.run([アプリのパス, ファイルパス], ...)`

▶ ファイルを規定のアプリで開く `subprocess.run(['start', ファイルパス], ...)`

▶ ☆ 印刷（プリンターの選択可能）

■並行処理

マルチスレッド

▶ threading モジュール `import threading`

▶ ※ マルチスレッドを行うときにグローバルなデータを書き換えるのは避けよう。

▶ ※ マルチスレッドはI/Oバウンド問題の解決には有効だが、CPUバウンド問題には不適。

▶ ※ マルチスレッドを行う場合、プロセスは1個に限定するのがよい。

マルチプロセス（multiprocessing : 推奨）

▶ multiprocessing モジュール `import multiprocessing as mp`

▶ プロセスプールをつくる `pool = mp.Pool(maxProcesses)`

▶ ある関数の実行を開始する `results = pool.map(関数, [p1, p2, ...])` や `results = pool.map(関数, [(p1, p2, ...), (p1, p2, ...), ...])`

▶ 先の関数の実行の結果を得る `for result in results: result` を使った処理

▶ ※ 別モジュールでマルチプロセスを実装したのを読み込む場合、`if __name__ == '__main__':` を忘れずに。そしてできれば `mp.freeze_support()` も行う。

▶ ☆ 用例

▶ 実行中のPythonプログラムのプロセスID `os.getpid()` ※当然 `import os` のもと

プロセスの生成（ほかのプログラムの実行）

▶ ※ Python以外のプログラムファイルやシェルコマンド（Windowsならcmdコマンド）を実行できる。

▶ subprocess モジュール `import subprocess`

▶ (ふつうに) 同期で命令文実行 `proc = subprocess.run(コマンドのリスト, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)`

▶ エラー時の出力を取得 `if proc.returncode: proc.stdout.decode('Shift-JIS')`

▶ (ふつうに) 非同期で命令文実行 `proc = subprocess.Popen(コマンドのリスト, ...)`

▶ プロセスを強制終了 `proc.kill()`

▶ 何らかのアプリを起動 `subprocess.run(アプリのパス, ...)`

▶ ファイルをあるアプリで開く `subprocess.run([アプリのパス, ファイルパス], ...)`

▶ ファイルを規定のアプリで開く `subprocess.run(['start', ファイルパス], ...)`

▶ ☆ 印刷（プリンターの選択可能）

■並行処理

マルチスレッド

▶ threading モジュール `import threading`

▶ ※ マルチスレッドを行うときにグローバルなデータを書き換えるのは避けよう。

▶ ※ マルチスレッドはI/Oバウンド問題の解決には有効だが、CPUバウンド問題には不適。

▶ ※ マルチスレッドを行う場合、プロセスは1個に限定するのがよい。

マルチプロセス（multiprocessing : 推奨）

▶ multiprocessing モジュール `import multiprocessing as mp`

▶ プロセスプールをつくる `pool = mp.Pool(maxProcesses)`

▶ ある関数の実行を開始する `results = pool.map(関数, [p1, p2, ...])` や `results = pool.map(関数, [(p1, p2, ...), (p1, p2, ...), ...])`

▶ 先の関数の実行の結果を得る `for result in results: result` を使った処理

▶ ※ 別モジュールでマルチプロセスを実装したのを読み込む場合、`if __name__ == '__main__':` を忘れずに。そしてできれば `mp.freeze_support()` も行う。

▶ ☆ 用例

マルチプロセス (concurrent.futures)

- ▶ concurrent.futures モジュール
- ▶ ※ concurrent.futures モジュールは threading モジュール、 multiprocessing モジュールをラップした高レベルのAPI。
- ▶ Executorオブジェクト
- ▶ ある関数の実行を開始する
- ▶ 先の関数の実行の結果を得る
- ▶ (futureが複数ある状況で) 終了したものから順に結果を得る

非同期I/O

- ▶ asyncio モジュール
- ▶ ※ asyncioでは必ず、司令塔の役割をもった（main的な）コルーチン関数を用意し、その外側のスコープでその関数を実行する処理をする。
- ▶ コルーチン関数を定義
- ▶ ※ コルーチン関数は呼び出してもすぐには実行されず、いったんコルーチンオブジェクトが返る。コルーチンオブジェクトは関数の中身の処理したいを表していると考えてよい（たまたに関数の中身と完全に同じではない処理のこともあるが）。コルーチンオブジェクトの実行はイベントループ内でのみ可能である。
- ▶ 最上位のコルーチンオブジェクトを実行
- ▶ ※ （コルーチン関数の定義内で）コルーチンオブジェクトを実行するには、前に await を付ける。

ログ出力

loggingパッケージを使うやり方（PSCDGと公式による）

- ▶ logging パッケージ
- ▶ ログを記録
- ▶ ログ出力レベルを変更する
- ▶ 1行1行の表示形式を変更
- ▶ 出力先をファイルにする
- ▶ ※ logging.basicConfig は先に実行されたものが有効になる。

loggingパッケージLoggerインスタを使うやり方（未検証）

- ▶ ☆ Python がログ出力する仕組み

マルチプロセス (concurrent.futures)

- ▶ concurrent.futures モジュール import concurrent.futures as cf ※ cf は自己流
- ▶ ※ concurrent.futures モジュールは threading モジュール、 multiprocessing モジュールをラップした高レベルのAPI。
- ▶ Executorオブジェクト executor = cf.ProcessPoolExecutor(max_workers=n)
- ▶ ある関数の実行を開始する future = executor.submit(function, p1, p2, ...)
- ▶ 先の関数の実行の結果を得る future.result()
- ▶ (futureが複数ある状況で) 終了したものから順に結果を得る for fut in cf.as_completed([future1, future2, ...]): fut.result() を使った処理※ ※返り値なくても必須

非同期I/O

- ▶ asyncio モジュール import asyncio
- ▶ ※ asyncioでは必ず、司令塔の役割をもった（main的な）コルーチン関数を用意し、その外側のスコープでその関数を実行する処理をする。
- ▶ コルーチン関数を定義 async def hoge(p1, ...): ..
- ▶ ※ コルーチン関数は呼び出してもすぐには実行されず、いったんコルーチンオブジェクトが返る。コルーチンオブジェクトは関数の中身の処理したいを表していると考えてよい（たまたに関数の中身と完全に同じではない処理のこともあるが）。コルーチンオブジェクトの実行はイベントループ内でのみ可能である。
- ▶ 最上位のコルーチンオブジェクトを実行 asyncio.run(mainCoroutine)
- ▶ ※ （コルーチン関数の定義内で）コルーチンオブジェクトを実行するには、前に await を付ける。

ログ出力

loggingパッケージを使うやり方（PSCDGと公式による）

- ▶ logging パッケージ import logging (Divider以下は)
- ▶ ログを記録 logging.info('インフォですよ。') # ※←INFOの場合
- ▶ ログ出力レベルを変更する logging.basicConfig(level=logging.INFO) # ※← "
- ▶ 1行1行の表示形式を変更 logging.basicConfig(format=※※) # ※ 独特な文字列
- ▶ 出力先をファイルにする logging.basicConfig(filename=パス) #
- ▶ ※ logging.basicConfig は先に実行されたものが有効になる。

loggingパッケージLoggerインスタを使うやり方（未検証）

- ▶ ☆ Python がログ出力する仕組み

- ▶ ☆ ログが出力されるまでの流れ
- ▶ loggingバLoggerインストール `from logging import getLogger logger = getLogger(__name__)`
- ▶ ☆ 準備
- ▶ ログを記録 (DEBUGレベル) `logger.debug("デバッグですよ。") #`
- ▶ ログを記録 (INFOレベル) `logger.info("インフォですよ。") #`
- ▶ ログを記録 (WARNINGレベル) `logger.warning("ワーニングですよ。") #`
- ▶ ログを記録 (ERRORレベル) `logger.error("エラーですよ。") #`

ファイル操作（詳解）編

■Excel

- ▶ ※ セル番地や `row=` , `column=` に数値を指定するときは `1, 2, 3, ...` で。
- ▶ openpyxl モジュール
- ▶ ※ `ValueError: Row or column values must be at least 1` と出たら → 1からにしよう

ブック

- ▶ 読み込み
- ▶ 新規作成
- ▶ 保存

シート

- ▶ 取得
- ▶ アクティブシート
- ▶ 名前を取得・変更
- ▶ 追加
- ▶ コピー
- ▶ 削除

行・列

- ▶ 行・列の挿入
- ▶ 行・列の削除
- ▶ データの限界行,列番号

- ▶ ☆ ログが出力されるまでの流れ
- ▶ loggingバLoggerインストール `from logging import getLogger logger = getLogger(__name__)`
- ▶ ☆ 準備
- ▶ ログを記録 (DEBUGレベル) `logger.debug("デバッグですよ。") #`
- ▶ ログを記録 (INFOレベル) `logger.info("インフォですよ。") #`
- ▶ ログを記録 (WARNINGレベル) `logger.warning("ワーニングですよ。") #`
- ▶ ログを記録 (ERRORレベル) `logger.error("エラーですよ。") #`

ファイル操作（詳解）編

■Excel

- ▶ ※ セル番地や `row=` , `column=` に数値を指定するときは `1, 2, 3, ...` で。
- ▶ openpyxl モジュール `$ pip install openpyxl import openpyxl`
- ▶ ※ `ValueError: Row or column values must be at least 1` と出たら → 1からにしよう

ブック

- ▶ 読み込み `wb = openpyxl.load_workbook(path)`
- ▶ 新規作成 `wb = openpyxl.Workbook()`
- ▶ 保存 `wb.save(filepath)`

シート

- ▶ 取得 `ws = wb['name']` や `ws = wb.worksheets[idx]`
- ▶ アクティブシート `ws = wb.active`
- ▶ 名前を取得・変更 `ws.title` ・ `ws.title = 'newName'`
- ▶ 追加 `wb.create_sheet(title='name', index=idx)` ※indexは省略可
- ▶ コピー `ws_copy = wb.copy_worksheet(ws)`
- ▶ 削除 `wb.remove(ws) #`

行・列

- ▶ 行・列の挿入 `ws.insert_rows(idx, cnt)` ・ `ws.insert_cols(idx, cnt)` ※cntは省略可
- ▶ 行・列の削除 `ws.delete_rows(idx, cnt)` ・ `ws.delete_cols(idx, cnt)` ※cntは省略可
- ▶ データの限界行,列番号 `ws.max_row` ・ `ws.max_col`

セル範囲

- ▶ 取得
- ▶ シートの全セルの2次元リスト
(データが存在する行・列まで)

セル

- ▶ 取得
- ▶ 行番号・列番号
- ▶ アドレス
- ▶ 値を取得・変更
- ▶ セルが空白か

■Word

- ▶ python-docx モジュール

■CSVファイル

CSV モジュールによる

- ▶ csv モジュール
- ▶ ファイルを開く、閉じる

- ▶ 全データの2次元リスト
- ▶ ☆ 検索

- ▶ 書き込みの準備
- ▶ 1行新たに書き込む

- ▶ ☆ 特定の行、列の値を変更

■JSONファイル

JSONモジュールによる

- ▶ ファイルを開く、閉じる
- ▶ json モジュール
- ▶ 辞書として読み込み
- ▶ 辞書を書き出し

■PDF編集

セル範囲

- ▶ 取得
- ▶ シートの全セルの2次元リスト
(データが存在する行・列まで)

セル

- ▶ 取得
- ▶ 行番号・列番号
- ▶ アドレス
- ▶ 値を取得・変更
- ▶ セルが空白か

■Word

- ▶ python-docx モジュール

■CSVファイル

CSV モジュールによる

- ▶ csv モジュール
- ▶ ファイルを開く、閉じる

- ▶ 全データの2次元リスト
- ▶ ☆ 検索

- ▶ 書き込みの準備
- ▶ 1行新たに書き込む

- ▶ ☆ 特定の行、列の値を変更

■JSONファイル

JSONモジュールによる

- ▶ ファイルを開く、閉じる
- ▶ json モジュール
- ▶ 辞書として読み込み
- ▶ 辞書を書き出し

■PDF編集

PyPDF2

▶ PyPDF2 ラ	
▶ ※ ページ番号は 0 から。	
▶ ※ PyPDF2にはコンテンツをゼロから全て作ることはできない。	
▶ PdfFileReaderオブ	
▶ ページ数	
▶ PdfFileWriterオブ	
▶ readerを一気に追加	
▶ ページを追加	
▶ ページを途中に挿入	
▶ 空白のページを追加	
▶ PDFとして書き出し	
▶ PageObjectオブ	
▶ 空白の "	
▶ ページのサイズ	
▶ ページの回転角度	
▶ ページを回転する	
▶ ページを拡大縮小する	
▶ 別ページを上から貼る	
▶ ☆ 回転に左右されず、見かけ上の幅, 高を得る。	
▶ PdfFileMerger インスタ	
▶ 結合（：後ろに挿入）	
▶ 途中に挿入	
▶ 一部のページを結合	
▶ 一部のページを挿入	
▶ PDFとして書き出し	
▶ インスタを後始末	

PyPDF2

▶ PyPDF2 ラ	\$ pip install PyPDF2 import PyPDF2
▶ ※ ページ番号は 0 から。	
▶ ※ PyPDF2にはコンテンツをゼロから全て作ることはできない。	
▶ PdfFileReaderオブ	reader = PyPDF2.PdfFileReader(パス)
▶ ページ数	reader.getNumPages()
▶ PdfFileWriterオブ	writer = PyPDF2.PdfFileWriter()
▶ readerを一気に追加	writer.appendPagesFromReader(reader) #
▶ ページを追加	writer.addPage(pg) #
▶ ページを途中に挿入	writer.insertPage(pg, ページ番号) #
▶ 空白のページを追加	writer.addBlankPage(width, height) ※ 角度は指定不可
▶ PDFとして書き出し	with open(パス, 'wb') as f: writer.write(f) #
▶ PageObjectオブ	pg = reader.getPage(pageNumber) ※ writer でも可能！
▶ 空白の "	PyPDF2.PageObject.createBlankPage(width=幅, height=高さ)
▶ ページのサイズ	pg.mediaBox.upperRight ※: (幅float, 高float) ※ 後述のrotのために実際の見た目の幅, 高でない可能性アリ
▶ ページの回転角度	pg.get('/Rotate') ※: 0, 90, 180, 270 （なぜかNoneの時も）
▶ ページを回転する	pg.rotateClockwise(angle) #
▶ ページを拡大縮小する	pg.scale(sx, sy) #
▶ 別ページを上から貼る	pg.mergeTranslatedPage(page2=pg_another, tx=x, ty=y) #
▶ ☆ 回転に左右されず、見かけ上の幅, 高を得る。	
▶ PdfFileMerger インスタ	merger = PyPDF2.PdfFileMerger()
▶ 結合（：後ろに挿入）	merger.append(パス) #
▶ 途中に挿入	merger.merge(ページ番号, パス) #
▶ 一部のページを結合	merger.append('パス', pages=(start, stop)) #
▶ 一部のページを挿入	merger.merge(ページ番号, パス, pages=(start, stop)) #
▶ PDFとして書き出し	merger.write(新パス) #
▶ インスタを後始末	merger.close() #

- ▶ ※ PdfReadWarning: Superfluous whitespace found in object header と出てきて目障り → .PdfFileReader(f) .PdfFileMerger() .PdfFileWriter() の引数に strict=False を設定。

Pdf2Image (・Poppler)

- ▶ Pdf2Image パ
- ▶ Poppler パ
- ▶ PDFを画像化

■XMLファイル

lxml

- ▶ lxml ラ
- ▶ cssselect ラ

Beautiful Soup

- ▶ Beautiful Soup 4 ラ
- ▶ ファイルを読み込む
- ▶ 文字列を読み込む

pyquery

- ▶ pyquery ラ

通信編

■メール送信

- ▶ smtplib ライブラリ
- ▶ email ライブラリ
- ▶ ☆ Gmail で送信

■URL

- ▶ urllib.parse モ
- ▶ URLをParseResultオブに
- ▶ ※ urlparse() に似たものとして urlsplit() があるが、これはURLからパラメータを分割しない。RFC 2396に従うような、各セグメントにパラメータを指定できるURLに便利。

- ▶ ※ PdfReadWarning: Superfluous whitespace found in object header と出てきて目障り → .PdfFileReader(f) .PdfFileMerger() .PdfFileWriter() の引数に strict=False を設定。

Pdf2Image (・Poppler)

- ▶ Pdf2Image パ \$ pip install pdf2image import pdf2image
- ▶ Poppler パ \$ pip install poppler (今回は import しなくていい)
- ▶ PDFを画像化 images = pdf2image.convert_from_path(fpath_src.as_posix()) images[n].save(path, 'png')

■XMLファイル

lxml

- ▶ lxml ラ \$ pip install lxml
- ▶ cssselect ラ \$ pip install cssselect

Beautiful Soup

- ▶ Beautiful Soup 4 ラ \$ pip install beautifulsoup4 from bs4 import BeautifulSoup
- ▶ ファイルを読み込む with open(・・) as f: soup = BeautifulSoup(f, 'html.parser')
- ▶ 文字列を読み込む soup = BeautifulSoup(文字列, 'html.parser')

pyquery

- ▶ pyquery ラ \$ pip install pyquery

通信編

■メール送信

- ▶ smtplib ライブラリ import smtplib
- ▶ email ライブラリ import email
- ▶ ☆ Gmail で送信

■URL

- ▶ urllib.parse モ import urllib.parse ※ import urlliblib じゃ無理
- ▶ URLをParseResultオブに o = urllib.parse.urlparse(URL)
- ▶ ※ urlparse() に似たものとして urlsplit() があるが、これはURLからパラメータを分割しない。RFC 2396に従うような、各セグメントにパラメータを指定できるURLに便利。

▶ スキーム	
▶ ネットワークロケーション	
▶ パス	
▶ パラメータ	
▶ クエリ	
▶ フラグメント	
▶ ユーザ名	
▶ パスワード	
▶ ホスト名	
▶ ポート番号	
▶ クエリを辞書に変換	

■サーバへのアクセス

▶ requests ラ	
▶ GETメソでアクセス	
▶ POSTメソでアクセス	
▶ PUT・DELETE・HEAD・PATCHメソでアクセス	
▶ ステータスコード	
▶ レスポンスヘッダ	
▶ エンコード	
▶ レスポンスボディ (バイナリ)	
▶ " (HTML様文字列)	
▶ 文字化け時のエンコード修正	

■スクレイピング

JS不要サイトを操作

▶ MechanicalSoup ライブラリ	
▶ ☆ 準備と始末	
▶ ページ遷移	
▶ リロード	
▶ 現ページのHTML	

▶ スキーム	o.scheme	
▶ ネットワークロケーション	o.netloc	※; 空文字列
▶ パス	o.path	※; 空文字列
▶ パラメータ	o.params	※; 空文字列
▶ クエリ	qs = o.query	※: ? 含まない文字列; 空文字列
▶ フラグメント	o.fragment	※; None
▶ ユーザ名	o.username	※; None
▶ パスワード	o.password	※; None
▶ ホスト名	o.hostname	※: 小文字の文字列; None
▶ ポート番号	o.port	※: int型; None
▶ クエリを辞書に変換	urllib.parse.parse_qs(qs)	※辞書のvalueはリスト型！

■サーバへのアクセス

▶ requests ラ	\$ pip install requests import requests
▶ GETメソでアクセス	res = requests.get(URL)
▶ POSTメソでアクセス	res = requests.post(URL, data={'key1': 'value1', ...})
▶ PUT・DELETE・HEAD・PATCHメソでアクセス	res = requests.put(URL) res = requests.delete(URL) res = requests.head(URL) res = requests.patch(URL)
▶ ステータスコード	res.status_code
▶ レスポンスヘッダ	res.headers ※: 大小文字区別しない特殊な辞書
▶ エンコード	res.encoding ※ヘッダの情報をもとに取得
▶ レスポンスボディ (バイナリ)	res.content
▶ " (HTML様文字列)	res.text ※文字化けするときはエンコードを修正
▶ 文字化け時のエンコード修正	res.encoding = res.apparent_encoding ※遅くなるかも

■スクレイピング

JS不要サイトを操作

▶ MechanicalSoup ライブラリ	\$ pip install MechanicalSoup import mechanicalsoup
▶ ☆ 準備と始末	
▶ ページ遷移	browser.open(URL) #
▶ リロード	browser.refresh() #
▶ 現ページのHTML	browser.page ※下の page を用いて str(page) も可

▶ 〃のMechanicalSoupオブジェクト	
▶ 要素を取得（CSSセレクターで）	
▶ 要素の中身	
▶ フォーム入力～送信	
▶ ☆ (クラス化で便利に)	

JSサイトを操作

▶ Selenium ラ webdriver モジュール	
▶ ☆ ドライバーをインストール	
▶ ☆ Chrome起動オプションを設定（変数 <code>options</code> の定義）	
▶ Chrome新規ウィンドウ起動	
▶ ウィンドウを閉じる	
▶ ☆ 準備と始末のまとめ	
▶ ページ遷移	
▶ リロード	
▶ 空のタブを開く	
▶ ウィンドウサイズ変更	
▶ サイトをスクショ	
▶ Byクラス	
▶ 要素を取得（IDで）	
▶ 要素を取得（CSSセレクターで）	
▶ 要素にたいして文字列を入力	
▶ 要素をクリック	
▶ 要素のコンテンツ	
▶ 要素に付帯するある属性の値	
▶ disabled属性がないか	
▶ ☆ ページ全体をスクショ	

▶ 〃のMechanicalSoupオブジェクト	<code>page = browser.get_current_page()</code>
▶ 要素を取得（CSSセレクターで）	<code>page.select("セレクター")</code> ※: 該当要素のリスト (1要素でも)
▶ 要素の中身	要素.text か 要素.contents ※(<code>.contents</code> なら): リスト
▶ フォーム入力～送信	0. フォームを選択 <code>browser.select_form('form[～～]') #</code> 1. ある部品に入力 <code>browser['※'] = 値</code> ※部品のname属性の値 2. 送信 <code>browser.submit_selected() #</code>
▶ ☆ (クラス化で便利に)	

JSサイトを操作

▶ Selenium ラ webdriver モジュール	<code>\$ pip install selenium</code> <code>from selenium import webdriver</code>
▶ ☆ ドライバーをインストール	
▶ ☆ Chrome起動オプションを設定（変数 <code>options</code> の定義）	
▶ Chrome新規ウィンドウ起動	<code>driver = webdriver.Chrome(options=options)</code>
▶ ウィンドウを閉じる	<code>driver.close()</code> 全ウィンドウを閉じるなら <code>driver.quit()</code>
▶ ☆ 準備と始末のまとめ	
▶ ページ遷移	<code>driver.get(URL) #</code>
▶ リロード	<code>driver.refresh() #</code>
▶ 空のタブを開く	<code>driver.execute_script("window.open();")</code>
▶ ウィンドウサイズ変更	<code>driver.set_window_size(x, y) #</code>
▶ サイトをスクショ	<code>driver.screenshot(filename) #</code>
▶ Byクラス	<code>from selenium.webdriver.common.by import By</code>
▶ 要素を取得（IDで）	<code>element = driver.find_element(By.ID, "hoge")</code> や <code>element2 = element1...</code>
▶ 要素を取得（CSSセレクターで）	<code>element = driver.find_element(By.CSS_SELECTOR, "セレクター")</code> や <code>element2 = element1...</code>
▶ 要素にたいして文字列を入力	<code>element.send_keys("Text") #</code>
▶ 要素をクリック	<code>element.click() #</code>
▶ 要素のコンテンツ	<code>element.text</code>
▶ 要素に付帯するある属性の値	<code>element.get_attribute('属性名')</code>
▶ disabled属性がないか	<code>element.is_enabled()</code>
▶ ☆ ページ全体をスクショ	

- ▶ ☆ (実例) スクレイピング
- ▶ ☆ 2023年1月9日

■Webアプリ作成

Flask

- ▶ Flask ラ
- ▶ Flask-SQLAlchemy ラ
- ▶ ☆ 準備
- ▶ ☆ 表（モデル）を定義
- ▶ 表を作成

Django

- ▶ Django パ

データ分析編

■Pandas——一般、シリーズ・データフレーム共通

- ▶ Pandas ラ

スカラーに変換

- ▶ オブの行数

同形の変換

- ▶ ビューからコピーへ
- ▶ 各値が欠測値か・否か
- ▶ 欠損値を置換
- ▶ ある値をもつセルを置換
- ▶ 行名を一括で変更
- ▶ 行名で並び替え
- ▶ 比較演算して真偽値に

列を増減する変換

- ▶ ☆ (実例) スクレイピング
- ▶ ☆ 2023年1月9日

■Webアプリ作成

Flask

- ▶ Flask ラ \$ pip install Flask from flask import Flask
- ▶ Flask-SQLAlchemy ラ ☆☆☆ from flask_sqlalchemy import SQLAlchemy
- ▶ ☆ 準備
- ▶ ☆ 表（モデル）を定義
- ▶ 表を作成 db.create_all()

Django

- ▶ Django パ \$ pip install django

データ分析編

■Pandas——一般、シリーズ・データフレーム共通

- ▶ Pandas ラ \$ pip install pandas import pandas as pd

スカラーに変換

- ▶ オブの行数 len(obj) か obj.shape[0]

同形の変換

- ▶ ビューからコピーへ obj.copy() ※ このobjはビューの状態であることを想定
- ▶ 各値が欠測値か・否か obj.isnull() ・ obj.notnull()
- ▶ 欠損値を置換 obj.fillna(値) ※デフなら列ごとに違う値にする方法もある
- ▶ ある値をもつセルを置換 obj.replace(what, repl) や obj.replace([what1, ...], repl)
- ▶ 行名を一括で変更 obj.index = ['r0', 'r1', 'r2', ...]
- ▶ 行名で並び替え obj.sort_index() ※降順なら ascending=False に。
- ▶ 比較演算して真偽値に obj < 7 obj != 0 (obj < 7) & (obj != 0) obj1 < obj2

列を増減する変換

▶ 左右で 連結	
▶ 行名を連番に戻す	

行を増減する変換

▶ 最初の5行だけ抜き出し	
▶ 行を削除	
▶ 上下で 連結	
▶ 空の1行を新規追加	
▶ 1行を新規追加	
▶ 無作為に(1・複)行を抽出	
▶ 無作為に(1・複)列を抽出	

その他

▶ 2つのオブを 連結結合	
▶ CSVとして書き出し	
▶ ※ ほとんどのメソッドは仮引数 <code>inplace</code> をもち、 <code>True</code> にするとその <code>obj</code> じたいを更新する（破壊的変更を行う）。	
▶ ※ 以降、削減のために <code>idx</code> は <code>['r0', ...]</code> などを、 <code>col</code> は <code>['c0', ...]</code> などを指すこととする。	

■Pandas——シリーズ

▶ リストからシリに	
▶ 辞書からシリに	
▶ ※ 仮引数 <code>dtype</code> で型を指定できる。文字列なら <code>pd.StringDtype()</code> 。	
▶ デフの列からシリに	
▶ デフの行からシリに	
▶ ☆ NumPy と重複する文法	

スカラーに変換

▶ ある行名が存在するか	
▶ 値 (1つ) を参照	

同形の変換

▶ 左右で 連結	<code>df = pd.concat([obj0, ...], axis=1)</code> ※ 共通した行名のみにしたいなら <code>join='inner'</code> に。
▶ 行名を連番に戻す	<code>obj = obj.reset_index()</code>

行を増減する変換

▶ 最初の5行だけ抜き出し	<code>obj.head()</code>
▶ 行を削除	<code>obj = obj.drop(['r2', ...])</code> ※1行なら <code>[]</code> 不要
▶ 上下で 連結	<code>df = pd.concat([obj0, ...])</code>
▶ 空の1行を新規追加	<code>df.loc[df.index.max()+1] = None</code> ※行名が連番のときに限る
▶ 1行を新規追加	<code>concat()</code> を使おう
▶ 無作為に(1・複)行を抽出	<code>obj.sample()</code> ・ <code>obj.sample(n=<i>num</i>)</code>
▶ 無作為に(1・複)列を抽出	<code>obj.sample(axis=1)</code> ・ <code>obj.sample(n=<i>num</i>)</code>

その他

▶ 2つのオブを 連結結合	<code>pd.merge(obj0, obj1, on='キー列の名前')</code>
▶ CSVとして書き出し	<code>obj.to_csv(パス, header=列名含めるか, index=行名含めるか)</code>
▶ ※ ほとんどのメソッドは仮引数 <code>inplace</code> をもち、 <code>True</code> にするとその <code>obj</code> じたいを更新する（破壊的変更を行う）。	
▶ ※ 以降、削減のために <code>idx</code> は <code>['r0', ...]</code> などを、 <code>col</code> は <code>['c0', ...]</code> などを指すこととする。	

■Pandas——シリーズ

▶ リストからシリに	<code>ser = pd.Series(l, index=idx)</code> ※ <code>name=</code> で名前を設定可能
▶ 辞書からシリに	<code>(d = {'r0': v0, ...} 等で)</code> <code>ser = pd.Series(l)</code> ※ "
▶ ※ 仮引数 <code>dtype</code> で型を指定できる。文字列なら <code>pd.StringDtype()</code> 。	
▶ デフの列からシリに	<code>df['r2']</code> か <code>df.loc[:, 'r2']</code> か <code>df.iloc[:, 2]</code>
▶ デフの行からシリに	<code>df.loc['r2']</code> か <code>df.iloc[2]</code>
▶ ☆ NumPy と重複する文法	

スカラーに変換

▶ ある行名が存在するか	<code>'r2' in ser</code>
▶ 値 (1つ) を参照	<code>ser['r2']</code> ※: <code>dtype</code> の型

同形の変換

▶ シリの名前を変更	
▶ 置換	
▶ 正規表現への 前方 マッチ	
▶ 関数を適用して変換	
▶ 文字列型に変換	

行を増減する変換

▶ 値 (複数) を参照	
▶ 欠損値の行を削除	
▶ フィルター	
▶ 値の重複をなくす	

■Pandas——データフレーム

▶ ※ 1列のデータフレームをつくることも可能。ただ、複数列あるデフを1列に変換するという行為をすれば往々にして、シリーズ型に還元されるので注意。	
▶ 空のデータフレーム	
▶ リストの辞書からデフに	
▶ 2次元辞書からデフに	
▶ 2次元リスト,行列からデフに	
▶ シリの辞書からデフに	
▶ 辞,シリのリストからデフに	
▶ CSVからデフに	
▶ Excelファからデフに	
▶ ※ 以上は仮引数 <code>dtype</code> で型を指定できる。文字列なら <code>pd.StringDtype()</code> 。	
▶ HTML中の表たちから	
デフのリストに	

スカラーに変換

▶ ある列名が存在するか	
▶ デフの列数	

▶ シリの名前を変更	<code>ser.name = '名前'</code>
▶ 置換	<code>ser.str.replace(<i>what</i>, <i>replacement</i>)</code>
▶ 正規表現への 前方 マッチ	<code>ser.str.match(<i>regex</i>)</code> ※: bool のシリーズ
▶ 関数を適用して変換	<code>ser.map(関数名)</code> ※実は辞書を渡すのもOK
▶ 文字列型に変換	<code>ser.astype(str)</code>

行を増減する変換

▶ 値 (複数) を参照	<code>ser[['r2', ...]]</code> ※: シリ
▶ 欠損値の行を削除	<code>ser.dropna()</code>
▶ フィルター	<code>ser[※]</code> ※serに処理を施して要素がboolになったシリ
▶ 値の重複をなくす	<code>ser.unique()</code>

■Pandas——データフレーム

▶ ※ 1列のデータフレームをつくることも可能。ただ、複数列あるデフを1列に変換するという行為をすれば往々にして、シリーズ型に還元されるので注意。	
▶ 空のデータフレーム	<code>df = pd.DataFrame()</code>
▶ リストの辞書からデフに	(<code>d = {'c0': [v00, ...], ...}</code> 等で) <code>df = pd.DataFrame(d, index=idx)</code>
▶ 2次元辞書からデフに	(<code>d = {'c0': {'r0': v00, ...}, ...}</code> 等で) <code>df = pd.DataFrame(d)</code>
▶ 2次元リスト,行列からデフに	<code>df = pd.DataFrame(l, index=idx, columns=col)</code>
▶ シリの辞書からデフに	<code>df = pd.DataFrame(d)</code>
▶ 辞,シリのリストからデフに	(<code>l = [{'c0': v00, ...}, ...]</code> 等で) <code>df = pd.DataFrame(l, index=idx)</code>
▶ CSVからデフに	(列名, 行名のないCSVから) <code>pd.read_csv(パス, header=None)</code> (列名のみあるCSVから) <code>pd.read_csv(パス)</code> (列名, 行名があるCSVから) <code>pd.read_csv(パス, index_col=0)</code>
▶ Excelファからデフに	<code>pd.read_excel(パス, sheet_name=シート名か番号※, header=..., index_col=...)</code> ※番号なら <code>0</code> 始まりの整数
▶ ※ 以上は仮引数 <code>dtype</code> で型を指定できる。文字列なら <code>pd.StringDtype()</code> 。	
▶ HTML中の表たちから	<code>pd.read_html(パス※)</code>
デフのリストに	※Path型, HTML様文字列, URL, バッファも可。

スカラーに変換

▶ ある列名が存在するか	<code>'c2' in df</code>
▶ デフの列数	<code>len(df.columns)</code> か <code>df.shape[1]</code>

- ▶ 1つの値を参照
- ▶ ある列名が存在するか

同形の変換

- ▶ 特定の1列を編集
- ▶ ある列の名前を変更
- ▶ 列名を一括で変更
- ▶ ある列の値で並び替え
- ▶ 列名で並び替え
- ▶ 行名で並び替え

列を増減する変換

- ▶ 列 (1つ・複数) を参照
- ▶ シリを列として新規追加
- ▶ 列を削除
- ▶ 欠損値を含む列を削除
- ▶ ある列を行名に抜擢

行を増減する変換

- ▶ 行 (1つ・複数) を参照
- ▶ 欠損値を含む行を削除
- ▶ 全てが欠損値の行を削除
- ▶ いくつかの行に絞る
- ▶ 特定の1列でフィルター
- ▶ ある列で重複をなくす

その他

- ▶ 転置
- ▶ 1行ずつ処理

■ORM

- ▶ SQLAlchemy ラ
- ▶ MySQLConnector ラ

- ▶ 1つの値を参照
- ▶ ある列名が存在するか

同形の変換

- ▶ 特定の1列を編集
- ▶ ある列の名前を変更
- ▶ 列名を一括で変更
- ▶ ある列の値で並び替え
- ▶ 列名で並び替え
- ▶ 行名で並び替え

列を増減する変換

- ▶ 列 (1つ・複数) を参照
- ▶ シリを列として新規追加
- ▶ 列を削除
- ▶ 欠損値を含む列を削除
- ▶ ある列を行名に抜擢

行を増減する変換

- ▶ 行 (1つ・複数) を参照
- ▶ 欠損値を含む行を削除
- ▶ 全てが欠損値の行を削除
- ▶ いくつかの行に絞る
- ▶ 特定の1列でフィルター
- ▶ ある列で重複をなくす

その他

- ▶ 転置
- ▶ 1行ずつ処理

■ORM

- ▶ SQLAlchemy ラ
- ▶ MySQLConnector ラ

- ▶ Mysqlclient パ
- ▶ PyMySQL パ
- ▶ ☆ 準備 1. DBエンジンとsessionの作成 ※別ファ (`./setting.py` とする) に書かれない
- ▶ ☆ 準備 2. モデル（表）を定義 ※別ファ (`./model.py` とする) に書かれない
- ▶ ☆ 準備 3 ※実際の実行ファイルに書かれる
- ▶ ※ `_mysql_connector.MySQLInterfaceError: Can't connect to MySQL server on '192.168.2.108:3306' (10061)` などと出てきたら → サーバのMySQLにたいする外部ホストからの接続を許可する
- ▶ ※ 1対多で、`user` の `files` 属性に `file` を追加したい場合は、`user.files.append(file)` ではなく `file.user = user` とせよ。(そしてこれだけでちゃんと互いに参照可能になる)

表fooの録を取得

- ▶ ※ SQLAlchemyでは クエリオブジェクト (sqlalchemy.orm.Query) を生成するたびにSQL文を発行する（メソッドチェーンにすれば1回で済む）。以下の `△` はクエリオブジェクトを表すこととする。
- ▶ 表fooの全列抽出するクエリ
- ▶ 特定の列だけ抽出するクエリ
- ▶ 条件に合うものに絞るクエリ
- ▶ ある列で並び替えるクエリ
- ▶ 最大 n 件に絞るクエリ
- ▶ m 件目以降で "
- ▶ `△`をFooオブのリストに
- ▶ `△`の最初の1件をFooオブに

表fooを更新

- ▶ 録を追加
- ▶ 録の値を変える
- ▶ 録を削除

- ▶ Mysqlclient パ `$ pip install mysqlclient`
- ▶ PyMySQL パ `$ pip install PyMySQL`
- ▶ ☆ 準備 1. DBエンジンとsessionの作成 ※別ファ (`./setting.py` とする) に書かれない
- ▶ ☆ 準備 2. モデル（表）を定義 ※別ファ (`./model.py` とする) に書かれない
- ▶ ☆ 準備 3 ※実際の実行ファイルに書かれる
- ▶ ※ `_mysql_connector.MySQLInterfaceError: Can't connect to MySQL server on '192.168.2.108:3306' (10061)` などと出てきたら → サーバのMySQLにたいする外部ホストからの接続を許可する
- ▶ ※ 1対多で、`user` の `files` 属性に `file` を追加したい場合は、`user.files.append(file)` ではなく `file.user = user` とせよ。(そしてこれだけでちゃんと互いに参照可能になる)

表fooの録を取得

- ▶ ※ SQLAlchemyでは クエリオブジェクト (sqlalchemy.orm.Query) を生成するたびにSQL文を発行する（メソッドチェーンにすれば1回で済む）。以下の `△` はクエリオブジェクトを表すこととする。
- ▶ 表fooの全列抽出するクエリ `ses.query(Foo)` ※: `△`
- ▶ 特定の列だけ抽出するクエリ `ses.query(Foo.列1, Foo.列2, ...)` ※: `△`
- ▶ 条件に合うものに絞るクエリ `△.filter(Foo.列1 == 値1, Foo.列2 > 値2, ...)` ※: `△`
- ▶ ある列で並び替えるクエリ `△.order_by(Foo.列1, Foo.列2, ...)` ※: `△`
- ▶ 最大 n 件に絞るクエリ `△.limit(n)` ※厳密には最大 n 件にする ※: `△`
- ▶ m 件目以降で " `△.limit(n).offset(m)` `:limit(m, n)` ※: `△`
- ▶ `△`をFooオブのリストに `△.all()` ※: Fooオブのリスト
- ▶ `△`の最初の1件をFooオブに `△.first()` ※: Fooオブ; `None`

表fooを更新

- ▶ 録を追加 `foo = Foo() foo.列1 = 値1 ... ses.add(foo) # ses.commit() #`
- ▶ 録の値を変える `△.all()` や `△.first()` で既存のFooオブを取得したあと、`foo.列1 = 値1 ... ses.add(foo) # ses.commit() #`
- ▶ 録を削除 `△.delete()`

■簡単な画像処理

- ▶ Pillow ラ
- ▶ ※ 座標は左上が原点。
- ▶ ☆ 対応しているファイル形式
- ▶ 画像を読み込む
- ▶ 画像を新規作成
- ▶ グレースケールに変更
- ▶ 画像のサイズ
- ▶ 拡大や縮小
- ▶ 図形描画の準備
- ▶ 四角形を描画
- ▶ 線分を描画
- ▶ テキストを配置
- ▶ 画像を保存
- ▶ ☆ OpenCV形式の画像をPillow形式の画像に

■画像認識

- ▶ ※ NumPy ライブラリをインストールしておかないとOpenCVはインストールできない。
- ▶ OpenCV ラ
- ▶ 画像を読み込む
- ▶ グレースケールで読み込む
- ▶ 透過画像を読み込む
- ▶ ☆ Pillow形式の画像をOpenCV形式の画像に
- ▶ 色空間をBGR→Grayに
- ▶ 色空間をBGR→HSVに
- ▶ 色空間をGray→BGRに
- ▶ ※ 色空間が何であれ、要素の値の範囲は 0 ～ 255 。
- ▶ ※ OpenCVでは BGR の順なので注意（RGBではない）。

■簡単な画像処理

- ▶ Pillow ラ \$ pip install Pillow from PIL import Image
- ▶ ※ 座標は左上が原点。
- ▶ ☆ 対応しているファイル形式
- ▶ 画像を読み込む im = Image.open(imgPath)
- ▶ 画像を新規作成 im = Image.new("RGB", (width, height), (R, G, B))
- ▶ グレースケールに変更 im = im.convert("L")
- ▶ 画像のサイズ im.width im.height
- ▶ 拡大や縮小 im = im.resize((width, height))
- ▶ 図形描画の準備 from PIL import ImageDraw draw = ImageDraw.Draw(im)
- ▶ 四角形を描画 draw.rectangle((x1, y1, x2, y2), fill=(R, G, B), outline=(R, G, B)) #
- ▶ 線分を描画 draw.line((x1, y1, x2, y2), fill=(R, G, B), width=width) #
- ▶ テキストを配置 from PIL import ImageFont draw.text((x, y), txt, font=ImageFont.truetype("〇〇.ttf", size=size), fill=(R, G, B)) #
- ▶ 画像を保存 im.save(filePath) # ※ .png がいいっぽい。
- ▶ ☆ OpenCV形式の画像をPillow形式の画像に

■画像認識

- ▶ ※ NumPy ライブラリをインストールしておかないとOpenCVはインストールできない。
- ▶ OpenCV ラ ☆☆☆ import cv2
- ▶ 画像を読み込む with open(imgPath, 'rb') as f: buf = f.read() img = cv2.imdecode(np.frombuffer(buf, dtype=np.uint8)) ※パスが全角含まぬなら img = cv2.imread(imgPath) で可 ※ファが存在しない場合エラーにならず None を返す
- ▶ グレースケールで読み込む . . . img = cv2.imdecode(·, 0)
- ▶ 透過画像を読み込む . . . img = cv2.imdecode(·, -1)
- ▶ ☆ Pillow形式の画像をOpenCV形式の画像に
- ▶ 色空間をBGR→Grayに img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
- ▶ 色空間をBGR→HSVに img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
- ▶ 色空間をGray→BGRに img_color = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
- ▶ ※ 色空間が何であれ、要素の値の範囲は 0 ～ 255 。
- ▶ ※ OpenCVでは BGR の順なので注意（RGBではない）。

▶ 色反転	
▶ ぼかし処理	
▶ 画像の縦と横の長さ	
▶ 画像を表示	
▶ トリミング	
▶ 画像を回転	
▶ 画像を保存	
▶ オブジェクト検出 （ラベリング）	
▶ 領域ベースマッチング	
▶ 線分を引く	
▶ 四角形枠線を描画	
▶ テキストを配置	
▶ ※ OpenCV (cv2) での数値は厳密に整数型でないとエラーになるので注意。	

■音声処理

▶ pydub ラ AudioSegment ク	
▶ ※ ffmpeg-python もインストールする必要がある。	
▶ ※ 対応しているファイル形式はffmpegと同じ。	
▶ 音源ファを読み込み	
▶ ☆ librosa用のNumPy配列から読み込み	
▶ 音を連結する	
▶ 音量を変える	
▶ 切り取り	
▶ 別の音声を重ねる	
▶ 編集した音源を保存	
▶ ☆ librosa用のNumPy配列に変換	
▶ 音源の長さ	
▶ チャンネル数	

▶ 色反転	<code>img_inv = cv2.bitwise_not(img)</code>
▶ ぼかし処理	<code>img_blur = cv2.GaussianBlur(img, 加-初サハ", 標準偏差)</code>
▶ 画像の縦と横の長さ	<code>h, w = img.shape[:2]</code> （グレーなら <code>h, w = img.shape</code> ）
▶ 画像を表示	<code>cv2.imshow("ウィンドウ名", img)</code> <code>cv2.waitKey(〇〇秒)</code>
▶ トリミング	<code>img_trim = img[y:y+h, x:x+w]</code> ※ <i>x</i> と <i>y</i> の順番に注意
▶ 画像を回転	<code>img_rtt = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)</code> など
▶ 画像を保存	<code>buf = cv2.imencode(filePath, img)[1]</code> <code>with open(path, 'wb') as f:</code> <code>f.write(buf)</code> # ※パスが全角含まぬなら <code>cv2.imwrite(filePath, img)</code> # で可
▶ オブジェクト検出 （ラベリング）	<code>cv2.connectedComponents(img)</code> か <code>cv2.connectedComponentsWithStats(img)</code>
▶ 領域ベースマッチング	<code>cv2.matchTemplate(imgSrc, tempSrc, cv2.TM_CCOEFF_NORMED)</code>
▶ 線分を引く	<code>cv2.line(img, (x1, y1), (x2, y2), (B, G, R))</code> #
▶ 四角形枠線を描画	<code>cv2.rectangle(img, (x1, y1), (x2, y2), (B, G, R))</code> #
▶ テキストを配置	<code>cv2.putText(img, txt, loc, fontStyle, size, color, thickness, lineTipe)</code> #
▶ ※ OpenCV (cv2) での数値は厳密に整数型でないとエラーになるので注意。	

■音声処理

▶ pydub ラ AudioSegment ク	<code>\$ pip install pydub</code> <code>from pydub import AudioSegment</code>
▶ ※ ffmpeg-python もインストールする必要がある。	
▶ ※ 対応しているファイル形式はffmpegと同じ。	
▶ 音源ファを読み込み	<code>sound = AudioSegment.from_file(path, format="format")</code>
▶ ☆ librosa用のNumPy配列から読み込み	
▶ 音を連結する	<code>sound + sound2</code>
▶ 音量を変える	<code>sound + dB</code>
▶ 切り取り	<code>sound[startMilisecond:stopMilisecond]</code>
▶ 別の音声を重ねる	<code>sound.overlay(sound2, position=startMilisecond)</code>
▶ 編集した音源を保存	<code>sound.export(path, format="format")</code> # ※ .m4aなら <code>format='mp4'</code> にして <code>bitrate="256K"</code> も追加
▶ ☆ librosa用のNumPy配列に変換	
▶ 音源の長さ	<code>sound.duration_seconds</code> ※単位は秒！
▶ チャンネル数	<code>sound.channels</code>

▶ サンプリングレート	
▶ ビット深度	
▶ サンプル数	

■音楽解析

- ▶ ※ 以下のライブラリやパッケージは、オーディオファイルをNumPy配列として扱うことができる。

soundfile または scipy.io.wavfile

▶ soundfile ラ	
▶ scipy.io.wavfile モ	
▶ ※ soundfile でサポートされているファイル形式は <code>sf.available_formats()</code> で確認できる。また、各形式でサポートされている「サブタイプ」を <code>sf.available_subtypes()</code> で確認できる。	
▶ 音源ファを読み込み	
▶ チャンネル分割	
▶ 編集した音源を保存	
▶ 編集した音源を保存	

librosa

▶ librosa パ	
▶ 音源ファを読み込み	
▶ BPM	
▶ ピッチを変える	
▶ 調波音と打楽器音に分解	
▶ チャンネル分割	
▶ モノラルに変換	
▶ 編集した音源を保存	

■MIDI

▶ サンプリングレート	<code>sound.frame_rate</code>
▶ ビット深度	<code>sound.sample_width * 8</code> ※ <code>.sample_width</code> 自体は byte 単位
▶ サンプル数	<code>int(sound.frame_count())</code>

■音楽解析

- ▶ ※ 以下のライブラリやパッケージは、オーディオファイルをNumPy配列として扱うことができる。

soundfile または scipy.io.wavfile

▶ soundfile ラ	<code>import soundfile as sf</code>
▶ scipy.io.wavfile モ	<code>from scipy.io import wavfile</code>
▶ ※ soundfile でサポートされているファイル形式は <code>sf.available_formats()</code> で確認できる。また、各形式でサポートされている「サブタイプ」を <code>sf.available_subtypes()</code> で確認できる。	
▶ 音源ファを読み込み	<code>data, samplerate = sf.read(path)</code> か <code>samplerate, data = wavfile.read(wavPath)</code> ※ステレオなら <code>data.shape</code> は <code>(サンプル数, 2)</code> 。
▶ チャンネル分割	<code>l_channel = data[:, 0]</code> <code>r_channel = data[:, 1]</code>
▶ 編集した音源を保存	<code>sf.write(path, data, samplerate)</code> # か <code>wavfile.write(path, samplerate, data)</code> #
▶ 編集した音源を保存	<code>wavfile.write(path, samplerate, data)</code> #

librosa

▶ librosa パ	<code>\$ pip install librosa</code> <code>import librosa as lr</code> ※ <code>lr</code> は自己流
▶ 音源ファを読み込み	モノラルなら <code>y, sr = lr.load(path, sr=None)</code> 多chなら <code>y, sr = lr.load(path, sr=None, mono=False)</code> ※ <code>y.shape</code> はそれぞれ <code>(サンプル数,)</code> <code>(ch数, サンプル数)</code>
▶ BPM	<code>lr.beat.beat_track(y=y*1, sr=sr)[0]</code> ※ ¹ モノラル ※低精度
▶ ピッチを変える	<code>y = lr.effects.pitch_shift(y, sr=sr, n_steps=何半音上げるか)</code>
▶ 調波音と打楽器音に分解	<code>y_harmonic, y_percussive = librosa.effects.hpss(y)</code> ※不明瞭
▶ チャンネル分割	<code>l_channel = y[0]</code> <code>r_channel = y[1]</code>
▶ モノラルに変換	<code>librosa.to_mono(y)</code> ※ <code>y.mean(axis=0)</code> と全く同じ
▶ 編集した音源を保存	<code>sf.write(path, y if len(y.shape)==1 else y.T, sr)</code> #

■MIDI

Pretty Midi

▶ Pretty Midi ラ	
▶ PrettyMidiオブ	
▶ BPM	

Mido

▶ Mido ラ	
▶ ファを読み込み	
▶ ☆ BPMを取得	

■動画編集

▶ ffmpeg-python パ	
▶ ※ ほどんどのファイル形式に対応している。	
▶ 読み込むファを指定	
▶ 特定の時間に限定して "	
▶ 音声だけに・映像だけに	
▶ ※ 以下のメソッドはどれも s をレシーバとして実行することも可能（その場合第1引数の s は書かない）。	
▶ 左右反転・上下反転	
▶ クリッピング	
▶ リサイズ	
▶ フレームレートを変更	
▶ 保存先を指定	
▶ 形式を変えつつ "	
▶ 読み込み～保存を実行	
▶ ☆ 動画のそれぞれのコマを NumPy.Array にする	
▶ ☆ 動画から静止画を抜き取る	

■YouTube

youtube-dl

▶ youtube-dl パッケージ	
▶ ☆ 動画ダウンロード（現在Anacondaでは実行に失敗している）	

Pretty Midi

▶ Pretty Midi ラ	\$ pip install pretty_midi import pretty_midi
▶ PrettyMidiオブ	midi = pretty_midi.PrettyMidi(filePath)
▶ BPM	midi.get_tempo_change()[1][0]

Mido

▶ Mido ラ	\$ pip install mido import mido
▶ ファを読み込み	mid = mido.MidiFile(パス)
▶ ☆ BPMを取得	

■動画編集

▶ ffmpeg-python パ	\$ pip install ffmpeg-python import ffmpeg
▶ ※ ほどんどのファイル形式に対応している。	
▶ 読み込むファを指定	s = ffmpeg.input(path) ※ s は stream が普通
▶ 特定の時間に限定して "	s = ffmpeg.input(path, ss=start※, t=length) ※ 3 '0:03' 等
▶ 音声だけに・映像だけに	s.audio ・ s.video ※ s['a'] ・ s['v'] でも。
▶ ※ 以下のメソッドはどれも s をレシーバとして実行することも可能（その場合第1引数の s は書かない）。	
▶ 左右反転・上下反転	ffmpeg.hflip(s) ・ ffmpeg.vflip(s)
▶ クリッピング	ffmpeg.crop(s, upper_left_x, upper_left_y, width, height)
▶ リサイズ	ffmpeg.filter(s, 'scale', width, -1) ※当然 .., -1, height でも
▶ フレームレートを変更	ffmpeg.filter(s, 'fps', fps=fps, round='down')
▶ 保存先を指定	ffmpeg.output(s, path)
▶ 形式を変えつつ "	.mp3なら format='mp3' を上記に追加 .m4aなら format='mp4', audio_bitrate='256K' を上記に追加
▶ 読み込み～保存を実行	stdout, stderr = ffmpeg.run(s) ※ overwrite_output=True 推奨
▶ ☆ 動画のそれぞれのコマを NumPy.Array にする	
▶ ☆ 動画から静止画を抜き取る	

■YouTube

youtube-dl

▶ youtube-dl パッケージ	\$ pip install youtube-dl import youtube_dl
▶ ☆ 動画ダウンロード（現在Anacondaでは実行に失敗している）	

PyTube

- ▶ PyTube パッケージ
- ▶ ☆ 動画ダウンロード

GUI編

■仮想ディスプレイ

- ▶ ☆ pyvirtualdisplay パッケージを導入
- ▶ ☆ 仮想ディスプレイを利用

■マウス操作、キーボード操作など

- ▶ PyAutoGui モ
- ▶ 操作不能に陥ったなら

キー操作

- ▶ キーを押下
- ▶ 同時に複数のキーを押下
- ▶ 文字列を入力
- ▶ ☆ 日本語入力

マウス操作

- ▶ 画面サイズ
- ▶ 現在のポインタ位置
- ▶ ポインタを移動
- ▶ ドラッグ
- ▶ 現在地で左クリック
- ▶ 特定座標で左クリ
- ▶ 特定のボタンを左クリ

画面遷移前に次の操作を実行させないために

- ▶ ある時間だけ待つ
- ▶ 画像を認識し座標を取得

PyTube

- ▶ PyTube パッケージ \$ pip install pytube import pytube
- ▶ ☆ 動画ダウンロード

GUI編

■仮想ディスプレイ

- ▶ ☆ pyvirtualdisplay パッケージを導入
- ▶ ☆ 仮想ディスプレイを利用

■マウス操作、キーボード操作など

- ▶ PyAutoGui モ ☆☆☆ import pyautogui
- ▶ 操作不能に陥ったなら プログラム実行中にマウスカーソルを一番左上に移動

キー操作

- ▶ キーを押下 pyautogui.press('キー名') か pyautogui.press(['キー名1', ...])
- ▶ 同時に複数のキーを押下 pyautogui.hotkey('キー名1', ...)
- ▶ 文字列を入力 pyautogui.write('文字列') ※ **非推奨**→日本語入力と同様にやれ
- ▶ ☆ 日本語入力

マウス操作

- ▶ 画面サイズ screen_x, screen_y = pyautogui.size()
- ▶ 現在のポインタ位置 pyautogui.position()
- ▶ ポインタを移動 pyautogui.moveTo(x, y) # か pyautogui.move(Δx, Δy) #
- ▶ ドラッグ pyautogui.dragTo(x, y) # か pyautogui.drag(Δx, Δy) #
- ▶ 現在地で左クリック pyautogui.click() #
- ▶ 特定座標で左クリ pyautogui.click(x, y) #
- ▶ 特定のボタンを左クリ pyautogui.click(ボタンの画像ファイルのパス) #

画面遷移前に次の操作を実行させないために

- ▶ ある時間だけ待つ import time time.sleep(0.3) #
- ▶ 画像を認識し座標を取得 pyautogui.locateOnScreen(imgFilePath, confidence=threshold)

