

# 【Python】応用

## 目次

- パッケージ全版編
  - モジュールについて
  - パッケージ（ライブラリ）について
  - 自作パッケージ（自作ライブラリ）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（【Python】並行処理）
  - ログ出力
- ファイル編集（詳解）編
  - 音楽ファイル
  - Excelファイル
  - Wordファイル
  - CSVファイル
  - JSONファイル
  - YAMLファイル
  - PDFファイル
  - XMLファイル
- データ分析編

# 【Python】応用

## 目次

- パッケージ全版編
  - モジュールについて
  - パッケージ（ライブラリ）について
  - 自作パッケージ（自作ライブラリ）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（【Python】並行処理）
  - ログ出力
- ファイル編集（詳解）編
  - 音楽ファイル
  - Excelファイル
  - Wordファイル
  - CSVファイル
  - JSONファイル
  - YAMLファイル
  - PDFファイル
  - XMLファイル
- データ分析編

- Pandasによるデータ分析（【Python】Pandas）
- 通信編
  - ソケット通信
  - URL
  - サーバへのアクセス
  - スクレイピング
  - データベース接続
  - ORM
  - APサーバ
  - FlaskによるWebアプリ作成（【Python】Flask）
  - DjangoによるWebアプリ作成
  - StreamlitによるWebアプリ作成
  - 静的サイトジェネレーション
  - メール送信
- 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

- Pandasによるデータ分析（【Python】Pandas）
- 通信編
  - ソケット通信
  - URL
  - サーバへのアクセス
  - スクレイピング
  - データベース接続
  - ORM
  - APサーバ
  - FlaskによるWebアプリ作成（【Python】Flask）
  - DjangoによるWebアプリ作成
  - StreamlitによるWebアプリ作成
  - 静的サイトジェネレーション
  - メール送信
- 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 はPythonのパッケージを管理するためのツール（パッケージ管理システム、パッケージマネージャ）の1つで、標準のものである。
- ▶ ※ ネットで検索していて `pip3` とか出てきたら → `pip` で構わない

# 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 はPythonのパッケージを管理するためのツール（パッケージ管理システム、パッケージマネージャ）の1つで、標準のものである。
- ▶ ※ 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) のバージョン確認

▶ pip (conda) を更新

▶ パッケージのインストール

▶ バージョン指定してインストール

▶ 一括インストール

▶ パッケージのアンインストール

▶ 全パッケージのリスト (ver付)

▶ アプデが必要なパッケージのリスト

▶ 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——import ..モ——from .. import モ

▶ 文字列でモを取り込む

▶ モで利用できる関数やクラスのリスト

▶ ☆ 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)

▶ ※ 実は `pip` ではなく `python3 -m pip` とするのが推奨されている。

▶ pip (conda) のバージョン確認 \$ pip --version \$ conda info

▶ pip (conda) を更新 \$ pip install -U pip \$ conda update conda

▶ パッケージのインストール \$ 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

▶ 一括インストール \$ pip install -r requirements.txt ※condaにはない

▶ パッケージのアンインストール \$ pip uninstall package \$ conda uninstall package

▶ 全パッケージのリスト (ver付) \$ pip list か \$ pip freeze \$ conda list

▶ アプデが必要なパッケージのリスト \$ pip list -o

▶ あるパッケージをアプデ \$ pip install -U package \$ conda update package

▶ バージョン指定してアプデ \$ pip install -U pack==ver ※condaにはない

▶ すべてのパッケージをアプデ ☆☆☆ \$ conda update --all

▶ パッケージの場所を確認 \$ pip show package ※condaにはない

▶ パッケージ検索対象フォルダを確認 import sys print(sys.path)

▶ ☆ 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 import モ from モ import 関数や変数 (as 別名)

▶ PYファの一段上の階層のモジュールの取り込み——import ..モ——from .. import モ

▶ 文字列でモを取り込む from importlib import import\_module  
module1 = import\_module('モ') module1.関数()や変数

▶ モで利用できる関数やクラスのリスト import モ dir(モ※) ※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 (*regex*p) で `r` を冒頭につけておくのは、`\n` や `\t` 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
- ▶ パターンに重なった部分
- ▶ " の開始・終了位置
- ▶ キャプチャに重なった部分
- ▶ *regex*pが*str*の一部に重なるか
- ▶ *regex*pが*str*全体に重なるか
- ▶ *regex*pにマッチする全箇所
- ▶ *regex*p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

# 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 (*regex*p) で `r` を冒頭につけておくのは、`\n` や `\t` 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 `import re`
- ▶ Matchオブ `m = re.search(regex, str)` ※マッチしないとNoneが返る
- ▶ パターンに重なった部分 `m.group()` ※先頭から見て初めて重なった部分
- ▶ " の開始・終了位置 `m.start()・m.end()` ※ `m.span()` で (始, 終) に。
- ▶ キャプチャに重なった部分 `m.group(n※)` や `m.group(name)` ※≥1
- ▶ *regex*pが*str*の一部に重なるか `bool(re.search(regex, str))`
- ▶ *regex*pが*str*全体に重なるか `bool(re.fullmatch(regex, str))`
- ▶ *regex*pにマッチする全箇所 `re.findall(regex, str)` ※: リスト
- ▶ *regex*pとマッチ箇所を置換 `re.sub(regex, replacement, str)`
- ▶ ☆ 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(regex, replacement, str)`

## ■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	
▶ ある時間だけ待つ	
▶ UNIX時間を取得	
▶ datetime モ	
▶ 今日の日付オブジェクトを取得	
▶ 現在の日時オブジェクトを取得	
▶ 現在日時のタイムスタンプ文字列	
▶ 現在の年、...、秒を取得	
▶ 文字列を日付オブジェクトに	
▶ 日時や日付を比較	
▶ 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	

▶ random モジュール	
▶ 0≤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。	
▶ ※ <code>IndexError: index 8 is out of bounds for axis 0 with size 8</code> 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。

▶ 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()      ※UNIXエポックからの経過秒数
▶ datetime モ	from datetime import datetime, date, timedelta
▶ 今日の日付オブジェクトを取得	dd = date.today()
▶ 現在の日時オブジェクトを取得	dt = datetime.now()
▶ 現在日時のタイムスタンプ文字列	datetime.now().strftime(r'%Y%m%d%H%M%S%f')
▶ 現在の年、...、秒を取得	dt.year .month .day .hour .minute .second .microsecond
▶ 文字列を日付オブジェクトに	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 ( <code>&gt;</code> など) で結ぶ
▶ 日時や日付を足し引き	dd + timedelta(days=1)      など
▶ ☆ main()関数の実行時間を調べる	

## dateutil、pytz（datetimeを併用）

▶ dateutil モ	\$ pip install dateutil from dateutil.relativedelta import relativedelta
▶ pytz モ	\$ pip install pytz      import pytz
▶ 日付を足し引き	dd + relativedelta(years=-3, months=+1, days=+9)    など
▶ 日付の一部を書き換え	dd + relativedelta(year=3, month=1, day=9)    など
▶ 月末にする	dd + relativedelta(months=+1, day=1, days=-1)

▶ random モジュール	import random
▶ 0≤乱数<1の生成	random.random()      ※: float型
▶ $min \leq \text{乱数} < max \in \mathbb{R}$	random.uniform( <i>min</i> , <i>max</i> )      ※: float型
▶ $min \leq \text{乱数} \leq max \in \mathbb{N}$	random.randint( <i>min</i> , <i>max</i> )    か    .randrange( <i>min</i> , <i>max</i> +1)

## NumPy——一般、配列全般

▶ NumPy ライブラリ	\$ pip install numpy      import numpy as np
▶ ※ 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。	
▶ ※ <code>IndexError: index 8 is out of bounds for axis 0 with size 8</code> 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	a = np.array(l)      ※ 1次元リスト→ベクトル、2次元リスト→行列
▶ 配列をリストに変換	l = a.tolist()
▶ 配列の形状	a.shape    ※ 配列が <i>n</i> 次元ならの要素数 <i>n</i> のタプルが返る
▶ 配列の次元	a.ndim
▶ 行・要素を参照	a[4]      ・      a[0][2]    a[0, 2]      ※スライスも可能！
▶ 行・要素を任意順で取得	a[[4, 3, 0, 6]]      ・      a[[1, 5, 7, 2], [0, 3, 1, 2]]      ※ "      ※コピー
▶ 配列がビューか否か	a.base      ※: 元配列; <span>None</span>
▶ 配列のコピー	np.array(a)      か      a.copy()      か      np.copy(a)
▶ 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 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(a)
▶ 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(a)      .ones_like(a)      .full_like(a, <i>num</i> )

■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	
▶ 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 → 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	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> )    ※ <span style="background-color: #f8d7da;">n=n</span> とすると <span style="background-color: #f8d7da;">n</span> 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	

■グラフ（プロット）

▶ matplotlib ラ pyplot モ	
▶ ※ pyplotでグラフを描くには「Pyplotインターフェイス」「オブジェクト指向」のどちらかのやり方で行うことになる。（ <code>ax</code> を使うのは後者）	
▶ 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）	
▶ フォントサイズ（全体）	
▶ ※ ネットで調べて <code>ax</code> 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	

システム・ファイル・ディレクトリ編

■パス

▶ 配列を結合	<code>np.concatenate([a1, ...], axis=<i>axis</i>)</code> ※ <code>axis</code> は <code>0</code> から ※第 <code>axis</code> 次元の要素数が増加する ※それ以外の次元の要素数は <code>a1</code> , <code>a2</code> , ... で同一の必要
---------	--

■グラフ（プロット）

▶ matplotlib ラ pyplot モ	<code>\$ pip install matplotlib</code> <code>import matplotlib.pyplot as plt</code>
▶ ※ pyplotでグラフを描くには「Pyplotインターフェイス」「オブジェクト指向」のどちらかのやり方で行うことになる。（ <code>ax</code> を使うのは後者）	
▶ 折れ線グラフ	<code>plt.plot(x_sequence, y_sequence) #</code>
▶ 散布図	<code>plt.scattar(x_sequence, y_sequence) #</code>

設定

▶ フォントの種類（全体）	<code>plt.rcParams['font.family'] = 'Times New Roman'</code>
▶ フォントの種類（数式）	<code>plt.rcParams['mathtext.fontset'] = 'stix'</code>
▶ フォントサイズ（全体）	<code>plt.rcParams['font.size'] = 15</code>
▶ ※ ネットで調べて <code>ax</code> 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>

システム・ファイル・ディレクトリ編

■パス



▶ 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	
▶ パスを分解する	
▶ 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	
▶ ベースネームを置換	
▶ 拡張子を置換	
▶ 絶対パスを相対パスにする	

## 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 <b>特定の拡張子の</b> 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	
▶ パスのファを作成	
▶ パスのファを削除	
▶ パスのディを作成	
▶ パスのディを削除	
▶ 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	


▶ 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> ※ <code>p.parents [1]</code> とすると親の親になる
▶ パスを分解する	<code>p.parts</code> ※ <code>('C:\\', 'Program Files', 'PSF')</code> 等になる
▶ 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	<code>p = p.with_name('sample2.txt')</code>
▶ 拡張子を置換	<code>p = p.with_suffix('.txt')</code>
▶ 絶対パスを相対パスにする	<code>p = p.relative_to(子孫ディレクトリのパス)</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> ※ <code>list()</code> か <code>sorted()</code> でくくるべき。
▶ 有する <b>特定の拡張子の</b> 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>
▶ パスのファを作成	<code>p.touch() #</code> ※ <code>exist_ok=True</code> 可能。
▶ パスのファを削除	<code>p.unlink() #</code> ※ <code>missing_ok=True</code> 可能。
▶ パスのディを作成	<code>p.mkdir() #</code> ※ <code>parents=True</code> <code>exist_ok=True</code> 可能。
▶ パスのディを削除	<code>p.rmdir() #</code> ※ そのディは空である必要。
▶ 深いパスのディまで一気に	<code>p.mkdir(parents=True, exist_ok=True)</code> ※ " <code>でもスルー</code>
▶ ☆ 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(path)</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(path))[0]</code>
▶ 拡張子	<code>os.path.splitext(os.path.basename(path))[1]</code> ※  付き
▶ パス結合	<code>os.path.join(dirpath1, dirpath2, ..., path)</code>
▶ 名称変更	
▶ shutil モジュール	
▶ 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	
▶ ☆ 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 <b>特定の拡張子の</b> 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	
▶ 中身が空のディを削除	
▶ 中身ごとディを削除	

■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(path)</code> か <code>os.path.isfile(path)</code>
▶ ファ作成	<code>f = open(path, 'w')</code> <code>f.write('')</code> <code>f.close</code>
▶ ファ削除	<code>os.remove(path)</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(path)</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(path)</code>
▶ 拡張子なしのベースネーム	<code>os.path.splitext(os.path.basename(path))[0]</code>
▶ 拡張子	<code>os.path.splitext(os.path.basename(path))[1]</code> ※  付き
▶ パス結合	<code>os.path.join(dirpath1, dirpath2, ..., path)</code>
▶ 名称変更	<code>os.rename(path, newPath) #</code>
▶ shutil モジュール	<code>import shutil</code>
▶ 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	
▶ ☆ 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 <b>特定の拡張子の</b> 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
▶ 中身が空のディを削除	<code>shutil.rmdir(path)</code> ※ 空じゃなければエラー
▶ 中身ごとディを削除	<code>shutil.rmtree(path)</code>

▶ 親のディレクトリの絶対パス	<code>os.path.dirname(<i>path</i>)</code>
▶ 中身を空にする	
▶ 中のファイルおよびディレクトリのリスト	
▶ 中のファイルのリスト	
▶ 中のディレクトリのリスト	

## ■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ファア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

▶ 実行中のPythonプログラムのプロセスID	
--------------------------	--

## 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	
▶ (ふつうに) 同期で命令文実行	
▶ エラー時の出力を取得	
▶ (ふつうに) 非同期で命令文実行	
▶ プロセスを強制終了	
▶ 何らかのアプリを起動	
▶ ファイルのあるアプリで開く	
▶ ファイルを規定のアプリで開く	
▶ ☆ 印刷（プリンターの選択可能）	

## ■ログ出力

▶ ☆ loggingライブラリの大まかな理解
-------------------------

## loggingを直接使うやり方（非推奨）

▶ 親のディレクトリの絶対パス	<code>os.path.dirname(<i>path</i>)</code>
▶ 中身を空にする	<code>shutil.rmtree(<i>ℓ</i>)</code> <code>os.mkdir(<i>ℓ</i>)</code>
▶ 中のファイルおよびディレクトリのリスト	<code>os.listdir(<i>path</i>)</code>
▶ 中のファイルのリスト	<code>[f for f in os.listdir(<i>ℓ</i>) if os.path.isfile(os.path.join(<i>ℓ</i>, f))]</code>
▶ 中のディレクトリのリスト	<code>[f for f in os.listdir(<i>ℓ</i>) if os.path.isdir(os.path.join(<i>ℓ</i>, 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ファア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	<code>os.getpid</code> ※当然 <code>import os</code> 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	<code>import subprocess</code>
▶ (ふつうに) 同期で命令文実行	<code>proc = subprocess.run(コマンドのリスト, shell=True, stdout=subprocess.PIPE, stderr=subprocess.STDOUT)</code>
▶ エラー時の出力を取得	<code>if proc.returncode:ℓ    proc.<b>stdout</b>.decode('Shift-JIS')</code>
▶ (ふつうに) 非同期で命令文実行	<code>proc = subprocess.Popen(コマンドのリスト, …)</code>
▶ プロセスを強制終了	<code>proc.kill()</code> #
▶ 何らかのアプリを起動	<code>subprocess.※(アプリのパス, " )</code> ※Popenがrunか
▶ ファイルのあるアプリで開く	<code>subprocess.※([アプリのパス, ファイルパス], " )</code>
▶ ファイルを規定のアプリで開く	<code>subprocess.※(['start', ファイルパス], " )</code>
▶ ☆ 印刷（プリンターの選択可能）	

## ■ログ出力

▶ ☆ loggingライブラリの大まかな理解
-------------------------

## loggingを直接使うやり方（非推奨）

- ▶ logging ラ
- ▶ ログを記録
- ▶ ※ 次の `logging.basicConfig` はふつう先に実行されたものが有効になる。が、Python 3.8 から導入された `force=` 引数を `True` にすると設定が上書きされる。
- ▶ 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.Loggerを使うやり方

- ▶ ☆ ログレコード、ロガー、ハンドラ、フォーマッタ、フィルタ
- ▶ ☆ ログが出力されるまでの流れ
- ▶ ※ 基本的な使う流れは、
  - (1)ログを管理するloggerを作成。
  - (2)ログ出力を管理するhandlerを作成。
  - (3)任意のhandlerをloggerにセット。(Qiita記事より)
- ▶ getLoggerクラス
- ▶ ※ 環境設定ファイルを別で作って読み込むのがよい。
- ▶ ロガーを取得
- ▶ ログを記録

- ▶ logging ラ `import logging` (Divider以下は)
- ▶ ログを記録 `logging.info('インフォですよ。') # ※←INFOの場合`
- ▶ ※ 次の `logging.basicConfig` はふつう先に実行されたものが有効になる。が、Python 3.8 から導入された `force=` 引数を `True` にすると設定が上書きされる。
- ▶ 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=formatString) #`
- ▶ 出力先をファイルにする `logging.basicConfig(filename=パス) #`

## logging.Loggerを使うやり方

- ▶ ☆ ログレコード、ロガー、ハンドラ、フォーマッタ、フィルタ
- ▶ ☆ ログが出力されるまでの流れ
- ▶ ※ 基本的な使う流れは、
  - (1)ログを管理するloggerを作成。
  - (2)ログ出力を管理するhandlerを作成。
  - (3)任意のhandlerをloggerにセット。(Qiita記事より)
- ▶ getLoggerクラス `from logging import getLogger`
- ▶ ※ 環境設定ファイルを別で作って読み込むのがよい。
- ▶ ロガーを取得 `logger = getLogger(loggerName)`  
※ `LoggerName` が同じロガーは**オブとして同一**。
- ▶ ログを記録 `logger.info('インフォですよ。') # ※←INFOの場合`

## ファイル操作（詳解）編

### ■音楽ファイル

- ▶ mutagen ラ

### mp3 ファイル

- ▶ ☆ mp3ファのプロパティ（タグ情報）の変更

### m4a (AAC) ファイル

- ▶ mutagen.mp4.MP4クラス
- ▶ MP4オブ
- ▶ タグ情報の1つを参照

## ファイル操作（詳解）編

### ■音楽ファイル

- ▶ mutagen ラ `$ pip install mutagen` (直接importしないのが普通)

### mp3 ファイル

- ▶ ☆ mp3ファのプロパティ（タグ情報）の変更

### m4a (AAC) ファイル

- ▶ mutagen.mp4.MP4クラス `from mutagen.mp4 import MP4` ※ `m4a` `M4A` でない！
- ▶ MP4オブ `mp4 = MP4(filePath)`
- ▶ タグ情報の1つを参照 `mp4.tags['propertySymbol※']` ※曲名なら `\xa9nam` 。

▶ タグを追加、上書き	
▶ 変更を保存する	
▶ MP4Coverクラス	
▶ MP4Coverオブ	

■Excelファイル

- ▶ ※ 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	

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)	

▶ タグを追加、上書き	<code>mp4.tags['propertySymbol※'] = value</code>
▶ 変更を保存する	<code>mp4.save(filePath)</code> # ※別ファにするなら元のファをコピーしておく必要あり
▶ MP4Coverクラス	<code>from mutagen.mp4 import MP4Cover</code>
▶ MP4Coverオブ	<code>covr = MP4Cover(binary, MP4Cover.FORMAT_JPEG)</code>

■Excelファイル

- ▶ ※ 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	<code>wb = openpyxl.load_workbook(path)</code>
▶ 新規作成	<code>wb = openpyxl.Workbook()</code>
▶ 保存	<code>wb.save(filePath)</code>

シート

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

行・列

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

セル範囲

▶ 取得	<code>r = ws['A2:E5']</code>
▶ 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)	<code>cells = [[cell for cell in row] for row in ws]</code> ※ これ積極的に行おう

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ファイル

▶ 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

▶ 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	

■YAMLファイル

▶ PyYaml ライブラリ	
▶ 辞書として読み込み	
▶ 辞書を書き出し	

■PDFファイル

PyPDF2

セル

▶ 取得	<code>c = ws["A1"]</code> や <code>c = ws.cell(1, 1)</code>
▶ 行番号・列番号	<code>c.row</code> ・ <code>c.column</code>
▶ アドレス	<code>c.coordinate</code> ※これで <code>(1, 1)</code> → <code>'A1'</code> ってのも可能に
▶ 値を取得・変更	<code>c.value</code> ・ <code>c.value = 'value'</code>
▶ セルが空白か	<code>c.value is None</code> ※ <code>== ''</code> ではないことに注意。

■Wordファイル

▶ python-docx モジュール	<code>\$ pip install python-docx</code> <b>import docx</b>
---------------------	--

■CSVファイル

CSV モジュールによる

▶ csv モジュール	<code>import csv</code>
▶ ファイルを開く、閉じる	<code>open()</code> <code>close()</code> あるいは <code>with</code> を使う

▶ 全データの2次元リスト	<code>data = [row for row in csv.reader(f)]</code>
▶ ☆ 検索	

▶ 書き込みの準備	(Winでは <code>open()</code> の引数 <code>newline=""</code> のもと) <code>writer = csv.writer(f)</code>
▶ 1行新たに書き込む	(Winでは <code>open()</code> の引数 <code>newline=""</code> のもと) <code>writer.writerow(1次元リスト)</code> か <code>.writerows(2次元リスト)</code>

▶ ☆ 特定の行、列の値を変更

■JSONファイル

▶ ファイルを開く、閉じる	<code>open()</code> <code>close()</code> あるいは <code>with</code> を使う
▶ json モジュール	<code>import json</code>
▶ 辞書として読み込み	<code>json.load(f)</code>
▶ 辞書を書き出し	<code>json.dump(d, f) #</code> ※ <code>indent=2</code> とすると見やすい

■YAMLファイル

▶ PyYaml ライブラリ	<code>\$ pip install pyyaml</code> <b>import yaml</b>
▶ 辞書として読み込み	<code>yaml.safe_load(f)</code>
▶ 辞書を書き出し	<code>yaml.dump(d, f)</code> ※日本語を出力するなら要工夫

■PDFファイル

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というのはできない。	
▶ 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	

▶ ※ 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。

▶ 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というのはできない。	
▶ 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	
▶ CSSセレクターで要素を取得	
▶ "（最初の要素のみ）	
▶ 要素の中身	
▶ 要素の属性値を参照	
▶ 属性と属性値の一覧	
▶ 要素全体（外側のタグを含む）の文字列	

pyquery

▶ pyquery ラ	
-------------	--

通信編

■ソケット通信

▶ socket モ	
▶ ☆ (サクッと) TCP通信を行う	
▶ ☆ (サクッと) UDP通信を行う	

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')
▶ CSSセレクターで要素を取得	soup.select("セレクター")     ※: 要素(=Tagオブ)のリスト
▶ "（最初の要素のみ）	soup.select_one("セレクター")     ※: 要素(=Tagオブ)
▶ 要素の中身	要素.text か 要素.contents ※( .contents なら): リスト
▶ 要素の属性値を参照	要素['属性']     ※ tag['id'] のように
▶ 属性と属性値の一覧	要素.attrs     ※: 辞書
▶ 要素全体（外側のタグを含む）の文字列	・ 普通のフォーマットなら     str(要素※ <sup>1</sup> )     ※ <sup>1</sup> soup も可 ・ 綺麗なフォーマットにするなら     要素※ <sup>1</sup> .prettyfy()

pyquery

▶ pyquery ラ	\$ pip install pyquery
-------------	------------------------

通信編

■ソケット通信

▶ socket モ	import socket
▶ ☆ (サクッと) TCP通信を行う	
▶ ☆ (サクッと) UDP通信を行う	



▶ ストリーム指向とは	
（コネクション指向）	
▶ データグラム指向とは	
（コネクションレス指向）	
▶ ソケットオブ	
▶ サバにてIP, ポートをソケに紐づけ	
▶ ソケを閉じる	

ストリーム指向の場合

▶ サバにて接続を受け付ける状態に	
▶ サバにて接続を受信する	
▶ ※ conn はその接続で送受信するために新しく作られたソケットオブジェクト。あとで必ず conn.close() で閉じよう。	
▶ サバからデータを返す	

---

▶ クラからリクエストを送る	
▶ クラにてデータを受信する	

データグラム指向の場合

▶ データを送る	
▶ データを受信する	

■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。	
▶ 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	

▶ ストリーム指向とは	通信に先立って通信路を確保しておいたうえで、データを送受信し、のちに通信路の解放処理を行う通信形式
▶ データグラム指向とは	データを送信する場合に、あらかじめ通信路を確保する必要がなく、いきなりデータを送ることができる通信形式
▶ ソケットオブ	・ストリーム指向： s = socket.socket(addressFamily, socket.SOCK_STREAM) ・データグラム： s = socket.socket(addressFamily, socket.SOCK_DGRAM)
▶ サバにてIP, ポートをソケに紐づけ	s.bind((IPAddress, port)) # ※1度のみ可能
▶ ソケを閉じる	s.close() #

ストリーム指向の場合

▶ サバにて接続を受け付ける状態に	s.listen(並列的に処理できるリクエスト数) #
▶ サバにて接続を受信する	conn, addr = s.accept()
▶ ※ conn はその接続で送受信するために新しく作られたソケットオブジェクト。あとで必ず conn.close() で閉じよう。	
▶ サバからデータを返す	conn.send(bytes) #

---

▶ クラからリクエストを送る	s.connect((IPAddress, port)) #
▶ クラにてデータを受信する	data = s.recv(受容する最大データ長)

データグラム指向の場合

▶ データを送る	s.sendto(bytes, (IPAddress, port)) #
▶ データを受信する	data, addr = s.recvfrom(受容する最大データ長)

■URL

▶ urllib.parse モ	import urllib.parse	※ import urllib 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	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	
▶ 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ponsヘッダ	
▶ エンコード	
▶ レスponsボディ (バイナリ)	
▶ " (文字列(HTML, JSON様))	
▶ " をJSONとしてデコード	
▶ 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	
▶ " の BeautifullSoup オブ	
▶ フォーム入力～送信	
▶ ☆ (クラス化で便利に)	

▶ ユーザ名	o.username	※; <span>None</span>
▶ パスワード	o.password	※; <span>None</span>
▶ ホスト名	o.hostname	※: 小文字の文字列; <span>None</span>
▶ ポート番号	o.port	※: int型; <span>None</span>
▶ クエリを辞書に変換	urlib.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pons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ponsボディ (バイナリ)	res.content
▶ " (文字列(HTML, JSON様))	res.text    ※文字化けするときはエンコードを修正
▶ " をJSONとしてデコード	res.json()    ※: 辞書か辞書のリスト; デコードエラー
▶ 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 <span>page</span> を用いて <span>str(page)</span> も可
▶ " の BeautifullSoup オブ	<b>page</b> = browser.page
▶ フォーム入力～送信	0. フォームを選択    browser.select_form('form[～～]') # 1. ある部品に入力    browser['※'] = 値    ※部品のname属性の値 2. 送信    browser.submit_selected() #
▶ ☆ (クラス化で便利に)	

JSサイトを操作

- ▶ Selenium ラ webdriver モ
- ▶ ☆ 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 options 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
- ▶ ☆ (実例) スクレイピング
- ▶ ☆ 2023年1月9日

■データベース接続

Mysqclient

- ▶ Mysqclient パ

MySQLConnector

- ▶ MySQLConnector ラ

JSサイトを操作

- ▶ Selenium ラ webdriver モ \$ pip install selenium from selenium import webdriver
- ▶ ☆ 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 options の定義）
- ▶ Chrome新規ウィンドウ起動 driver = webdriver.Chrome(options=options)
- ▶ ウィンドウを閉じる driver.close() 全ウィンドウを閉じるなら driver.quit()
- ▶ ☆ 準備と始末のまとめ
- ▶ ページ遷移 driver.get(URL) #
- ▶ リロード driver.refresh() #
- ▶ 空のタブを開く driver.execute\_script("window.open();")
- ▶ ウィンドウサイズ変更 driver.set\_window\_size(x, y) #
- ▶ サイトをスクショ driver.screenshot(filename) #
- ▶ Byクラス from selenium.webdriver.common.by import By
- ▶ 要素を取得（IDで） element = driver.find\_element(By.ID, "hoge")  
や element2 = element1...
- ▶ 要素を取得（CSSセレクターで） element = driver.find\_element(By.CSS\_SELECTOR, "セレクター")  
や element2 = element1...
- ▶ 要素にたいして文字列を入力 element.send\_keys("Text") #
- ▶ 要素をクリック element.click() #
- ▶ 要素のコンテンツ element.text
- ▶ 要素に付帯するある属性の値 element.get\_attribute('属性名')
- ▶ disabled属性がないか element.is\_enabled()
- ▶ ☆ 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

■データベース接続

Mysqclient

- ▶ Mysqclient パ \$ pip install mysqclient

MySQLConnector

- ▶ MySQLConnector ラ \$ pip install mysql-connector-python import mysql.connector

▶ DBに接続	
▶ 録を辞書として返すカーソルオブ	
▶ 録を名前付きタプルとして返す "	
▶ 1つの録を抽出	
▶ 複数の録を抽出	
▶ 抽出した録の数	
▶ DBとの接続を切断	

## PyMySQL

▶ PyMySQL パ	
-------------	--

### ■ORM

▶ SQLAlchemy ラ	
▶ ☆ 準備 1. DBエンジンとsessionの作成 ※別ファに書かれない ( <code>./setting.py</code> 等)	
▶ ☆ 準備 2. モデル (表) を定義 ※別ファに書かれない ( <code>./model.py</code> 等)	
▶ ☆ 準備 3 ※実際の実行ファイルに書かれる	
▶ ※ <code>_mysql_connector.MySQLInterfaceError: Can't connect to MySQL server on '192.168.2.108:3306' (10061)</code> 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、 <code>user</code> の <code>files</code> 属性に <code>file</code> を追加したい場合は、 <code>user.files.append(file)</code> ではなく <code>file.user = user</code> 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 <code>△</code> 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	
▶ ある列で並び替えするクエリ	
▶ 最大 $n$ 件に絞るクエリ	
▶ $m$ 件目以降で "	

▶ DBに接続	<code>cnx = mysql.connector.connector(host=host, port=port, user=user, password=pw, database=db)</code>
▶ 録を辞書として返すカーソルオブ	<code>cur = cnx.cursor(dictionary=True)</code>
▶ 録を名前付きタプルとして返す "	<code>cur = cnx.cursor(named_tuple=True)</code>
▶ 1つの録を抽出	<code>cur.execute("selectSQLStatement") # cur.fetchone()</code>
▶ 複数の録を抽出	<code>cur.execute("selectStmt") # cur.fetchall() ※: リスト</code>
▶ 抽出した録の数	<code>cur.execute("selectStmt") # cur.fetchall() cur.rowcount</code>
▶ DBとの接続を切断	<code>cnx.close() #</code>

## PyMySQL

▶ PyMySQL パ	<code>\$ pip install PyMySQL</code>
-------------	-------------------------------------

### ■ORM

▶ SQLAlchemy ラ	<code>\$ pip install sqlalchemy</code>	<code>import sqlalchemy as sa</code>
▶ ☆ 準備 1. DBエンジンとsessionの作成 ※別ファに書かれない ( <code>./setting.py</code> 等)		
▶ ☆ 準備 2. モデル (表) を定義 ※別ファに書かれない ( <code>./model.py</code> 等)		
▶ ☆ 準備 3 ※実際の実行ファイルに書かれる		
▶ ※ <code>_mysql_connector.MySQLInterfaceError: Can't connect to MySQL server on '192.168.2.108:3306' (10061)</code> 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、 <code>user</code> の <code>files</code> 属性に <code>file</code> を追加したい場合は、 <code>user.files.append(file)</code> ではなく <code>file.user = user</code> 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 <code>△</code> 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	<code>ses.query(Foo)</code>	※: <code>△</code>
▶ 特定の列だけ抽出するクエリ	<code>ses.query(Foo.列1, Foo.列2, ...)</code>	※: <code>△</code>
▶ 条件に合うものに絞るクエリ	<code>△.filter(Foo.列1 == 値1, Foo.列2 &gt; 値2, ...)</code>	※: <code>△</code>
▶ ある列で並び替えするクエリ	<code>△.order_by(Foo.列1, Foo.列2, ...)</code>	※: <code>△</code>
▶ 最大 $n$ 件に絞るクエリ	<code>△.limit(n)</code>	※厳密には最大 $n$ 件にする ※: <code>△</code>
▶ $m$ 件目以降で "	<code>△.limit(n).offset(m)</code>	<code>△.limit(m, n)</code> ※: <code>△</code>

- ▶ △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

■APサーバ

gunicorn（グニコーン）

- ▶ ※ WSGIサーバのひとつである。
- ▶ インストール

uWSGI（ユーウiskiー）

- ▶ ※ WSGIサーバのひとつである。
- ▶ ※ INI形式設定ファイルの設定項目についての詳細は公式ドキュメントを参照されたい。
- ▶ インストール

■DjangoによるWebアプリ作成

- ▶ Django パ

■StreamlitによるWebアプリ作成

- ▶ Streamlit パ

■静的サイトジェネレーション

Pelican

- ▶ Pelican ラ
- ▶ 空のプロジェクトを作成
- ▶ サイトを生成
- ▶ サイトのプレビューを開始
- ▶ localhost以外のアクセスを許して "

■メール送信

- ▶ smtplib ライブラリ
- ▶ email ライブラリ

- ▶ △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()

■APサーバ

gunicorn（グニコーン）

- ▶ ※ WSGIサーバのひとつである。
- ▶ インストール \$ pip install gunicorn

uWSGI（ユーウiskiー）

- ▶ ※ WSGIサーバのひとつである。
- ▶ ※ INI形式設定ファイルの設定項目についての詳細は公式ドキュメントを参照されたい。
- ▶ インストール \$ pip install uwsgi

■DjangoによるWebアプリ作成

- ▶ Django パ \$ pip install django

■StreamlitによるWebアプリ作成

- ▶ Streamlit パ \$ pip install streamlit

■静的サイトジェネレーション

Pelican

- ▶ Pelican ラ \$ pip install pelican ※追加が推奨されるパッケあり
- ▶ 空のプロジェクトを作成 \$ pelican-quickstart
- ▶ サイトを生成 \$ pelican content
- ▶ サイトのプレビューを開始 \$ pelican --listen
- ▶ localhost以外のアクセスを許して " \$ pelican --listen -b 0.0.0.0 ※ --bind でも。

■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

## 画像・音声・動画編

### ■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

▶ ☆ Gmail で送信

## 画像・音声・動画編

### ■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(..., cv2.IMREAD_GRAYSCALE)`
- ▶ 透過画像を読み込む `... img = cv2.cvtColor(cv2.imdecode(..., cv2.IMREAD_UNCHANGED), cv2.COLOR_BGR2BGRA)`
- ▶ ☆ 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〃→HSVに	
▶ 色空間を〃→BGRA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）。	
▶ 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	

▶ 色空間をBGR→Grayに	<code>img_gray = cv2.cvtColor(<b>img</b>, cv2.COLOR_BGR2GRAY)</code>
▶ 色空間を〃→HSVに	<code>img_hsv = cv2.cvtColor(<b>img</b>, cv2.COLOR_BGR2HSV)</code>
▶ 色空間を〃→BGRAに	<code>img_bgra = cv2.cvtColor(<b>img</b>, cv2.COLOR_BGR2BGRA)</code>
▶ 色空間をGray→BGRに	<code>img_color = cv2.cvtColor(<b>img</b>, cv2.COLOR_GRAY2BGR)</code>
▶ ※ 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	<code>img_inv = cv2.bitwise_not(<b>img</b>)</code>
▶ ぼかし処理	<code>img_blur = cv2.GaussianBlur(img, 加減幅, 標準偏差)</code>
▶ 画像の縦と横の長さ	<code>h, w = <b>img</b>.shape[:2]</code> （グレーなら <code>h, w = <b>img</b>.shape</code> ）
▶ 画像を表示	<code>cv2.imshow("ウィンドウ名", <b>img</b>)</code> <code>cv2.waitKey(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, <b>img</b>)[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(<b>img</b>)</code> か <code>cv2.connectedComponentsWithStats(<b>img</b>)</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(<b>img</b>, (x1, y1), (x2, y2), (B, G, R))</code> #
▶ 四角形枠線を描画	<code>cv2.rectangle(<b>img</b>, (x1, y1), (x2, y2), (B, G, R))</code> #
▶ テキストを配置	<code>cv2.putText(<b>img</b>, txt, loc, fontStyle, size, color, thickness, lineType)</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	
▶ 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	
▶ 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	
▶ 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	
▶ BPM	
▶ ピッチを変える	

▶ 別の音声を重ねる	<code>sound.overlay(sound2, position=<i>startMilisecond</i>)</code>
▶ 編集した音源を保存	<code>sound.export(<i>path</i>, format="<i>format</i>") #</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	<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	<code>data, samplerate = sf.read(<i>path</i>)</code> か <b><code>samplerate, data = wavfile.read(<i>wavPath</i>)</code></b> ※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(<i>path</i>, data, samplerate) #</code> か <code>wavfile.write(<i>path</i>, <b>samplerate, data</b>) #</code>
▶ 編集した音源を保存	<code>wavfile.write(<i>path</i>, 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	モノラルなら <code>y, sr = lr.load(<i>path</i>, sr=None)</code> 多chなら <code>y, sr = lr.load(<i>path</i>, 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*<sup>1</sup>, sr=sr)[0]</code> ※ <sup>1</sup> 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	
▶ チャンネル分割	
▶ モノラルに変換	
▶ 編集した音源を保存	

## ■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

▶ 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 <code>s</code> 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 <code>s</code> 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 "	

▶ 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]      r_channel = y[1]</code>
▶ モノラルに変換	<code>lr.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	<code>\$ pip install pretty_midi      import pretty_midi</code>
▶ PrettyMidiオブ	<code><b>midi</b> = pretty_midi.PrettyMidi(filePath)</code>
▶ BPM	<code>midi.get_tempo_change()[1][0]</code>

### Mido

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

## ■動画編集

### ffmpeg

▶ ffmpeg-python パ	<code>\$ pip install ffmpeg-python      import ffmpeg</code>
▶ ※ 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	<code>s = ffmpeg.input(path)      ※ <code>s</code> は <code>stream</code> が普通</code>
▶ 特定の時間に限定して "	<code>s = ffmpeg.input(path, ss=start※, t=length)      ※ <code>3</code> <code>'0:03'</code> 等</code>
▶ 音声だけに・映像だけに	<code>s.audio      s.video      ※ <code>s['a']</code> ・ <code>s['v']</code> でも。</code>
▶ ※ 以下のメソッドはどれも <code>s</code> 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 <code>s</code> は書かない）。	
▶ 左右反転・上下反転	<code>ffmpeg.hflip(s)      ffmpeg.vflip(s)</code>
▶ クリッピング	<code>ffmpeg.crop(s, upper_left_x, upper_left_y, width, height)</code>
▶ リサイズ	<code>ffmpeg.filter(s, 'scale', width, -1)      ※当然 <code>.., -1, height</code> でも</code>
▶ フレームレートを変更	<code>ffmpeg.filter(s, 'fps', fps=fps, round='down')</code>
▶ 保存先を指定	<code>ffmpeg.output(s, path)      ※上書きするかは <code>run</code> メソ時に。</code>
▶ 形式を変えつつ "	<code>.mp3</code> なら <code>format='mp3'</code> を上記に追加 <code>.m4a</code> なら <code>format='mp4'</code> , <code>audio_bitrate='256K'</code> を上記に追加

- ▶ 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

Moviepy

- ▶ Moviepy パッケージをインストール
- ▶ 映像と音声を合成して保存

■YouTube

youtube-dl

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

PyTube

- ▶ PyTube パッケージをインストール
- ▶ ☆ 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 モジュールをインストール
- ▶ 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

- ▶ 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

Moviepy

- ▶ Moviepy パッケージをインストール
- ▶ 映像と音声を合成して保存

■YouTube

youtube-dl

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

PyTube

- ▶ PyTube パッケージをインストール
- ▶ ☆ 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 モジュールをインストール
- ▶ 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	

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

### 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	<code>import time     time.sleep(0.3) #</code>
▶ 画像を認識し座標を取得	<code>pyautogui.locateOnScreen(<i>imgFilePath</i>, confidence=<i>threshold</i>)</code>