# 目次

# Web スクリプティング／Python 導入

## はじめに

Web サービスの普及でシステム連携が簡単にできるようになりました。J2EE 普及期には基幹系システムはサーブレットの先に CORBA／SOAP、EJB(Enterprise JavaBeans)等が繋がる重厚長大な形態以外に選択肢がありませんでしたが、より手軽な Web サービス（REST）の普及によりこれでコンポーネントを疎結合する構成が増えています。Web サービスは Srpring Framework 等の利用で比較的簡単に作れますが、複数のサービスを利用するアプリケーションを試験する際はより簡易にテストデータを返す機能が欲しくなります。現在では Python、JavaScript(NODE.js)、Ruby 等の多くのスクリプト言語が http/https のサーバ機能を備えており、個人用のテスト環境を手軽に作ることができます。

スクリプト言語は Java 以上にオブジェクト指向が進み（関数オブジェクト等）それに伴って言語仕様も複雑になってきています。この資料ではスクリプトを使う利便性を活かせるように Python の標準ライブラリを使い、http レスポンスを返す例を中心に手軽に使える機能に絞って紹介します。

## 1. Python とは

JavaScript と並んで人気が高い[1]スクリプト言語です。C/C++用の拡張 API を持っているため、色々な組織が数値計算や AI、ビッグデータ処理等の拡張ライブラリを公開して利用者が増えています。本体と標準ライブラリは改変と商用利用が可能なオープンソース（Ver.2.1 以降は PSF ライセンス[2]）で、Linux の主要なディストリビューションにはデフォルトでインストールされています。初期の頃から簡易な開発環境（1990 年代 Ver. 1.5.2 から IDLE）が同梱されて日本での初期の紹介文書に C 言語の教育用環境という表記があったように記憶していますが、コードの書き方は全く独特で処理ブロックをインデントで表しポインターは使えません。現在は教育用という表現は見当たらず、教育用言語の ABC に影響を受けたとだけ書かれています。

### 1.1. インストール

以下は、Windows 10 に最新版の開発用 Python(Ver. 3.10.5)をインストールする前提で記述します。他にも CI 運用や埋め込み利用のためのサブセットがありますが、こちらは開発に使えません。また、Microsoft Store からインストールすることもできますが、「Microsoft Store アプリの制限により、Python スクリプトには `TEMP` フォルダやレジストリのような共有の場所への完全な書き込み権限は無いでしょう」と但し書きがついています。

（1）ダウンロード

以下の URL の Downloads タブに各種の OS 向けのダウンロード・リンクが書かれています。

python™ https://www.python.org/

Files から Description に Recommended と書かれたインストーラをダウンロードします。GUI で起動した場合、セットアップに必要な資材はオプション選択後、インストール中にダウンロードが行われます。

---

[1] GitHub 言語ランキング https://madnight.github.io/githut/#/pull_requests/2022/1
GooleTrend https://trends.google.co.jp/trends/explore?cat=5&q=python,javascript,ruby,php,perl

[2] PSF ライセンス https://docs.python.org/3/license.html

（2）インストールオプション

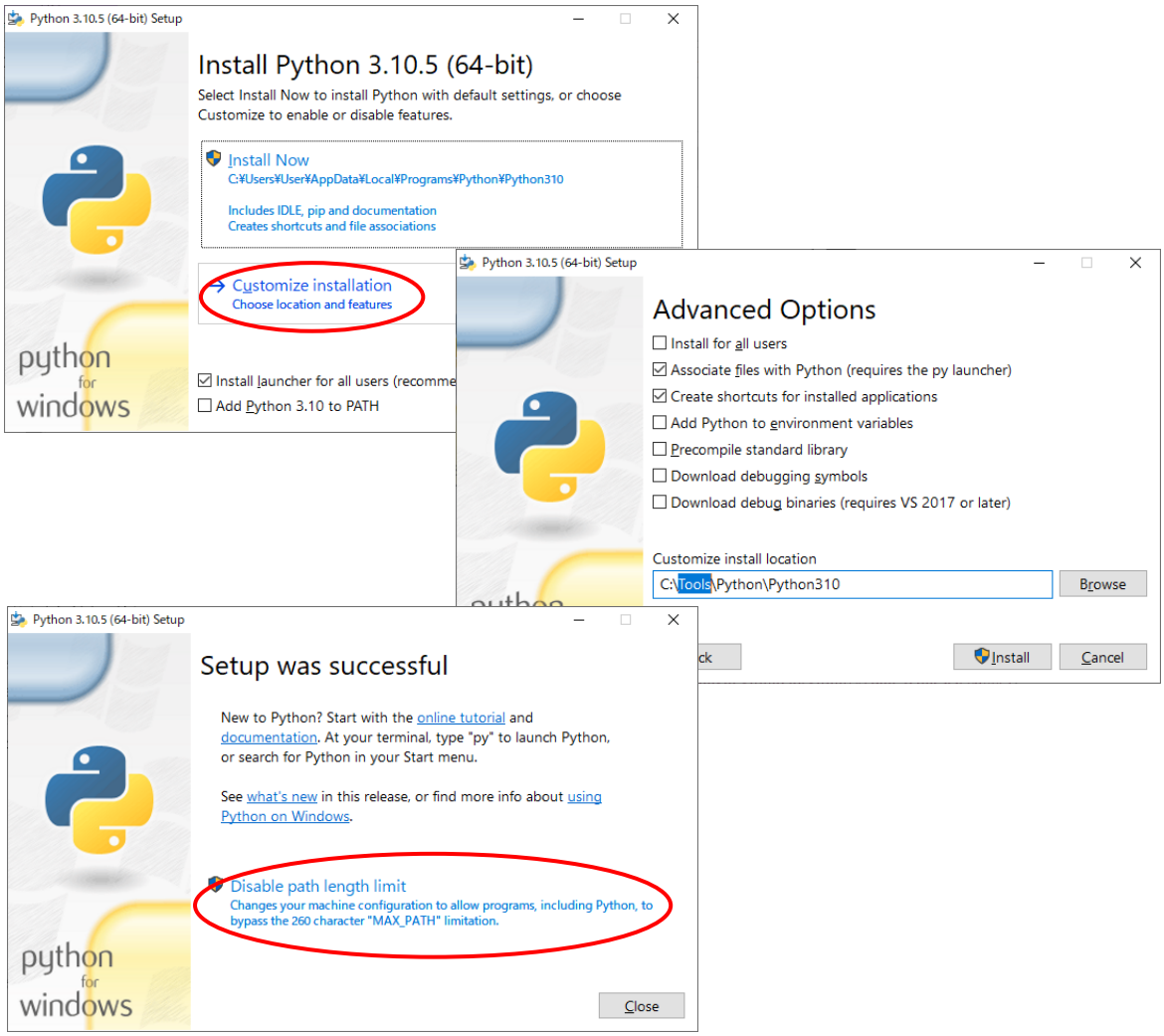
以下のオプションは変更するか、意識しておいた方が良い項目です。

① インストール先の初期値はインストール利用者のローカルパスになっています

… Install launcher for all users が初期値でチェックされているので、共用環境なら変更する

② インストール完了の画面で Disable path length limit を押下すると Python が 260 文字以上のファイルパス名を受け入れるようになります

… Windows 10 バージョン 1607 以降では、一般的な Win32 ファイルおよびディレクトリ関数から MAX_PATH 制限が削除されました（最大パスは約 32,767 文字です）[3]。




## 1.2. 日本語の情報

前項で紹介した Python 公式サイトに上でリンクされている日本語サイトは停止したものが多いですが、下記サイトは活動していて Windows 以外についても環境構築の手順を説明しています

PyJUG - Python Japan Users Group https://www.python.jp/

[3] パスの最大長の制限
https://docs.microsoft.com/ja-jp/windows/win32/fileio/maximum-file-path-limitation?tabs=cmd

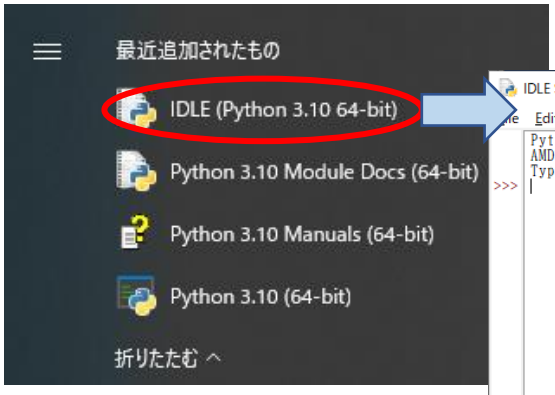
 

## 2. 開発環境

以下の説明では、Python と一緒にデフォルトでインストールされる IDLE を使います。
インストールが完了すると、スタートメニューに Python と一緒に表示されます。

（1）IDLE 起動／Shell ウィンドウ表示

IDLE は Shell と Editor の 2 種類のウィンドウがあり、初期設定のままであれば起動時に Shell ウィンドウが開き、>>>の出ている行で Python の式や文を実行することができます。



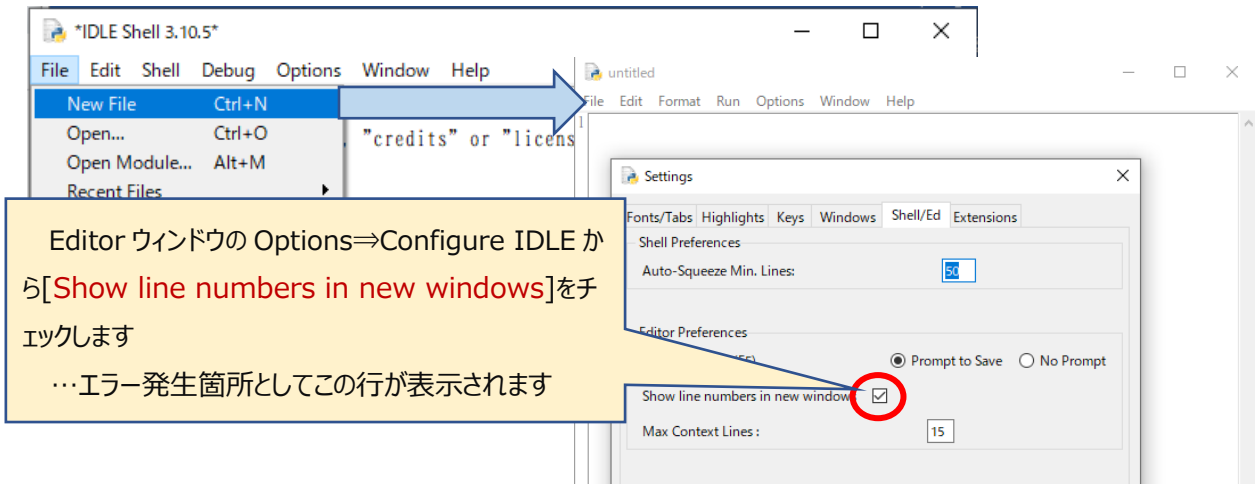

※下記例はフォントを BIZ UD 明朝に変えてます




入力した式は Enter 押下で実行され、値が下の行に表示されます。また、組み込み関数をタイプするとガイドが表示されます。

（2）Editor ウィンドウ表示

スクリプトの作成は Shell から Editor ウィンドウを開いて行います。




Editor ウィンドウの Options⇒Configure IDLE から[Show line numbers in new windows]をチェックします
…エラー発生箇所としてこの行が表示されます

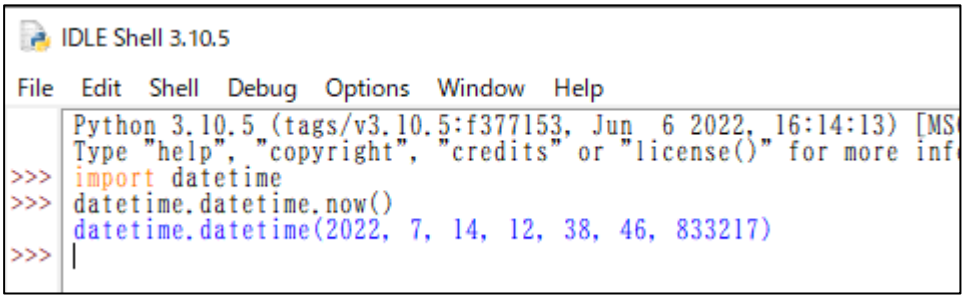
## 3. Python の導入知識

本資料は本格的な開発ではなく Python の便利な部分を手軽に使うのが目的なのでオブジェクト指向に纏わる面倒には立ち入らないようにして、サンプルを理解するのに必要な知識や用語を説明します。細かい点は公式サイト[4]を参照してください。

### 3.1. スクリプトとモジュール

Python はインタープリターで、Shell を使って対話的に実行するかスクリプト（プログラム[5]）をファイルから読み込んで実行します。このファイルをモジュールと呼び拡張子 .py のテキストファイルです。組み込み関数以外の大部分はモジュールの形で Lib ディレクトリの中に格納されており（標準ライブラリ）、import して利用することができます。

以下は、datetime モジュール内に定義されている datetime クラスの now メソッドで現在の日時を表示させた例です。datetime.datetime が冗長にみえますが、構造を見ると理由が分かります。

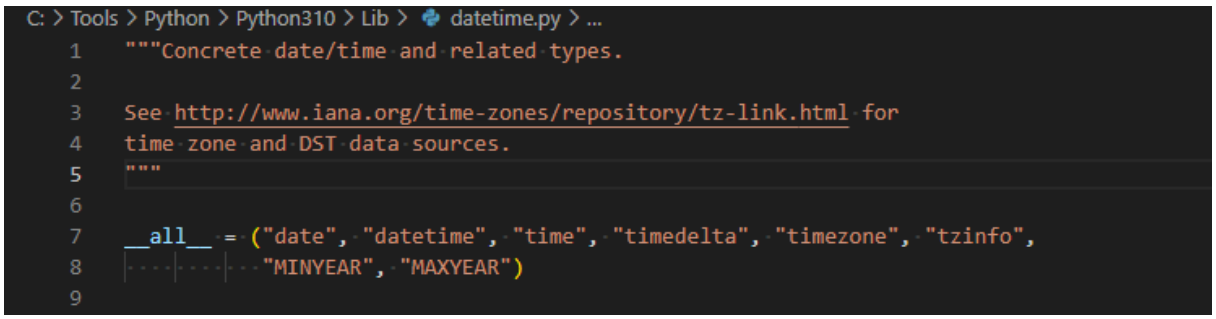



### 3.2. モジュールの構成と要素

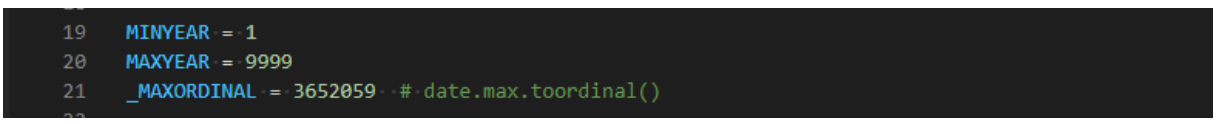
Python のインストール先のディレクトリにある Lib/datetime.py を例にすると、内容は以下のようになっています（一部抜粋）。

（1）モジュールの構成

① モジュールの先頭部分：3 連続の引用符で囲んだ説明文 docstring と変数の定義／代入式




（中略）　※__all__も変数で、タプル（変更不能な複数データの集まり）型で代入しています




[4] Python 言語リファレンス https://docs.python.org/ja/3/reference/index.html

[5] Python チュートリアル-モジュール https://docs.python.org/ja/3/tutorial/modules.html

② datetime クラス定義（➡が定義の階層）定義行の**行末に“：”を書き、続く桁落としの行でブロック**を構成します。桁落としの桁数は厳密に揃っている必要があります。

C: > Tools > Python > Python310 > Lib > datetime.py > datetime

```
1563  class datetime(date):
1564      """datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])
1565
1566      The year, month and day arguments are required. tzinfo may be None, or an
1567      instance of a tzinfo subclass. The remaining arguments may be ints.
1568      """
1569      __slots__ = date.__slots__ + time.__slots__
1570
1571  >   def __new__(cls, year, month=None, day=None, hour=0, minute=0, second=0,
```

③ now メソッド定義（行頭のインデントで datetime との包含関係を表す）

C: > Tools > Python > Python310 > Lib > datetime.py > datetime > now

```
1695      @classmethod
1696      def now(cls, tz=None):
1697          "Construct a datetime from time.time() and optional time zone info."
1698          t = _time.time()
1699          return cls.fromtimestamp(t, tz)
1700
```

④ datetime クラス定義の終わり（2168 行:インデントを戻して datetime が終わったことを表す）

C: > Tools > Python > Python310 > Lib > datetime.py > ...

```
2164      def __reduce__(self):
2165          return self.__reduce_ex__(2)
2166
2167
2168  datetime.min = datetime(1, 1, 1)
2169  datetime.max = datetime(9999, 12, 31, 23, 59, 59, 999999)
2170  datetime.resolution = timedelta(microseconds=1)
```

⑤ モジュールの終端

C: > Tools > Python > Python310 > Lib > datetime.py > ...

```
2503  # pretty bizarre, and a tzinfo subclass can override fromutc() if it is.
2504
2505  try:
2506      from _datetime import *
2507  except ImportError:
2508      pass
2509  else:
2510      # Clean up unused names
2511      del (_DAYNAMES, _DAYS_BEFORE_MONTH, _DAYS_IN_MONTH, _DI100Y, _DI400Y,
2512           _DI4Y, _EPOCH, _MAXORDINAL, _MONTHNAMES, _build_struct_time,
2513           _check_date_fields, _check_time_fields,
2514           _check_tzinfo_arg, _check_tzname, _check_utc_offset, _cmp, _cmperror,
2515           _date_class, _days_before_month, _days_before_year, _days_in_month,
2516           _format_time, _format_offset, _index, _is_leap, _isoweek1monday, _math,
2517           _ord2ymd, _time, _time_class, _tzinfo_class, _wrap_strftime, _ymd2ord,
2518           _divide_and_round, _parse_isoformat_date, _parse_isoformat_time,
2519           _parse_hh_mm_ss_ff, _IsoCalendarDate)
2520      # XXX Since import * above excludes names that start with _,
2521      # docstring does not get overwritten. In the future, it may be
2522      # appropriate to maintain a single module level docstring and
2523      # remove the following line.
2524      from _datetime import __doc__
2525
```

（2）モジュールの要素

① 実行文

関数やクラスに含まれない（インデント無の最上位）実行可能な文は、モジュールをコマンドラインや import で参照すると即時に実行されます。

② 変数と名前

変数は代入と同時に確保（バインディング）されます。datetime.py の 7 行目にある__all__=は外部に提供するクラスや定数の名前を代入すると同時に変数の宣言にもなっています。

ブロックの中と外で同一の変数名に代入があるとスコープが異なる別の変数とみなします。変数がブロック内のローカル変数と認識されたくないときは global（最上位で定義した変数）や nonlocal（上位の関数で定義した変数）という指定が必要になります。

変数の前後のアンダースコア( _ )は慣用的な名前付けで、特別な機能はありません。アンダースコアに関しては PEP 8「Python コードのスタイルガイド」[6]という文書があり、標準ライブラリはこれに従って作られています。よく目にするのは以下のものです。

・内部用で非公開にしたい変数の名前は先頭にアンダースコアを一つ付ける

・予約済キーワードと重複する場合は末尾に一つアンダースコアを付ける

・__all__、__name__ 他、前後に二つ付いているのは Python が意味付けした特殊変数名

【注意】

Python には Java のアクセス修飾子 private にあたるものがなく、変数名の先頭にアンダースコアを付けても import した側に見えてしまいます（関数、メソッド等も同様です）。対策として datetime.py の 2511 行目で del( )により名前の抹消（アンバインド）を行っています。

③ 関数

関数定義も非公開の関数名は先頭にアンダースコアをつけ、必要なら最後の実行文で del します。関数名の後に（）を付けなかった場合、関数は実行されずに関数自体がオブジェクト（コードオブジェクト／関数オブジェクト）として代入されます。

④ クラス

クラス名の後ろに( )で 0 個以上のパラメータを付けて実行すると、クラスのインスタンスが返されます。() を付けなかった場合、実行されずにクラス自体がオブジェクト（コードオブジェクト／クラスオブジェクト）として代入されます。

クラス内に関数様に定義されているのはメソッドです。datetime.datetime.now の定義の直前に書かれている@classmethod は、インスタンス化せずに呼び出すことを可能にするためのデコレータと呼ぶ記述です。これがついている now()は datetime クラスをインスタンス化せずにメソッドとして呼び出すことができます。類似のデコレータに@staticmethod があります。

また、メソッド名には予約された特殊メソッド名があります（宣言は任意です）。

＜特殊メソッドの例＞

__new__(cls[, ...]) クラス cls の新しいインスタンスを作るために呼び出されます。

__init__(self[, ...]) インスタンス self が生成された後、retun 実行の前に呼び出されます

[6] **Python Enhancement Proposals** https://peps.python.org/pep-0008/

### 3.3. モジュールの実行

モジュールに書かれた処理を実行する方法に以下の 3 つがあります（シバンや拡張子関係付け等の OS 固有の方法は除く）。

① python コマンドのパラメータとして py ファイルのパスを渡す

python *モジュール名*.**py** <任意のパラメータ群>

※ py ファイルに記述された、関数定義の外にある実行文がスクリプトとして実行されます

② python コマンドのパラメータにモジュール名を指定する

python **-m** *モジュール名* <任意のパラメータ群>

※ Python モジュール検索パスからモジュール名に合致するものを探し、①同様実行します

③ スクリプトの中からインポートする

import *モジュール名*

※ import されると関数定義の外の実行文が一度だけ実行されます。実行文を迂回したい場合は __name__を判定して分岐させます

### 3.4. オブジェクト（型）

Python 言語リファレンス-第 3 章 データモデル では次のようにオブジェクトを定義しています。『…データを抽象的に表したものです。Python プログラムにおけるデータは全て、オブジェクトまたはオブジェクト間の関係として表されます。(ある意味では、プログラムコードもまたオブジェクトとして表されます。…』具体的には、Python/C API リファレンスマニュアル-[共通のオブジェクト構造体]に、『全てのオブジェクトは構造体 PyObject を拡張した型で表現し、PyObject には参照カウント（0 になるとオブジェクトは解放されます）と型（PyTypeObject）を持っている』と説明されており、主要な型の種類は以下のように分類されます。

【組込み型】

- ├数値…int（整数）—boolean（True, False)、float（浮動小数点数)、 complex（複素数）
- ├変更不能シーケンス…string（文字列)、tuple（タプル：”( )”で囲んだ要素集合）
- ├変更可能シーケンス…list（”[ ]”で囲んだ要素集合）
- ├集合型…set（”{ }”で囲んだ要素集合。要素に順がなく重複を許さない。set()で空({})が返る）
- ├マッピング（辞書）…dict（”{ }”で囲んだキー：値のペアを要素とする集合…{*キー:値*, …}）
- ├クラス、クラスインスタンス
- ├モジュール
- ├関数
- ├メソッド
- ├コードオブジェクト
- ├ヌルオブジェクト（None）
- ┆

※型は type(*オブジェクト*)を使って、クラスの親子関係は isinstance(*オブジェクト*, *型*)で継承関係の有無（互換性）を調べることができます

・ 関数、メソッドもオブジェクトで型を持っており、以下のような記述ができます。

<例１ メソッドオブジェクト>

```
f = datetime.datetime.now      # now()の処理結果ではなく now のコード自体を代入
f()                            # f に代入されているコードの呼出し
# ---上のコードは下記と同一の結果になります---
datetime.datetime.now()
```

<例２ コードオブジェクトと関数オブジェクトの違い>

```
def outerFunc(x):
    def innerFunc(y):
        print(x - y)

    return innerFunc

a=outerFunc(100)  # ①
a(10) # ①-1
a(99) # ①-2
b=outerFunc(1000) # ②
b(10) # ②-1
b(99) # ②-2
```

```
=== 以下の出力になります ===
90
1
990
901
```

※outerFunc は①と②で innerFunc のコードを返してきて、このコードを呼び出す①-1 と②-1 のパラメータも同値です。違いは①と②で outerFunc に渡しているパラメータで、これは innerFunc の呼出し時には渡していません。それでも結果に outerFunc へのパラメータが反映されているのは①の関数オブジェクトと②の関数オブジェクトの各々にパラメータが保存（セルオブジェクト）されているためです。①、②の関数オブジェクトが参照しているコードオブジェクトは同一です

・ 関数にはオブジェクト的な取り扱いを行うデコレータという機能があります

*@デコレータ関数名* を関数定義の前に書くと *デコレータ関数名（定義した関数）* にコンパイラが置き換えます。下記の③'で行われる処理は x=( deco_func(add_func) )(1, 5) となります。

```
def deco_func(para_func):   # ①
    def exec_func(*args):   # ② *args の * は可変長引数を表し、複数の引数をタプルとして受け取ります
        print('func=', para_func.__name__, '*args=', args)
        val = para_func(*args)
        return val

    return exec_func

# 関数デコレータ
@deco_func                  # ①'
def add_func(n, m):         # ③
    """ n から m までの合計を返す関数 """
    ret = 0
    for i in range(n, m + 1):
        ret += i
```

```
        print(i, '¥t', 'calc=', ret)
    return ret

@deco_func                  # ①'
def multi_func(n, m):       # ④
    """ n から m までの乗算計を返す関数 """
    ret = 1
    for i in range(n, m + 1):
        ret *= i
        print(i, '¥t', 'calc=', ret)
    return ret


x = add_func(1, 5)      # ③'
print(x)
y = multi_func(1, 5)    # ④'
print(y)
=== 以下の出力になります ===
func= add_func *args= (1, 5)
1    calc= 1
2    calc= 3
3    calc= 6
4    calc= 10
5    calc= 15
15
func= multi_func *args= (1, 5)
1    calc= 1
2    calc= 2
3    calc= 6
4    calc= 24
5    calc= 120
120
```

※①'は deco_func のパラメータとして関数オブジェクト add_func を渡し、deco_func の内部関数 exec_func の中から呼ばれるようにし、③'で add_func が呼ばれたらその exec_func を実行するようになります。④'も同様に y=( deco_func(multi_func) )(1, 10)と等価になります。このデコレートの処理はコンパイラーが行っているので解り辛いですが、上のコードを IDLE の Editor ウインドに貼り付けて実行すると同一の結果が得られることが確認できます。

## 3.5. 名前と id（名前空間、スコープ）

Python のスクリプト（関数、クラスに含まれない実行文）以外のコードとデータはオブジェクトで、名前を使って操作します。

（1）名前の定義と id

モジュールはファイル名、関数、クラス、メソッドはそれぞれの定義で付けた名前で識別し、変数は代入式で定義（バインディング）されます。オブジェクトは全て固有の id（組み込み関数 id(オブジェクト)で確認できます）と型を持ち、変数へは id が代入されます。変数は定義後に全く別の型（整数から文字列等）のオブジェクト（の id）を代入することで書き換えることができます。

（2）名前空間

オブジェクトにアクセスするとき名前の階層を使って対象を特定します。ファイルに保存されている関数を実行する場合は、このファイルを import *モジュール名* し、*モジュール名.関数名*()で実行することができます。モジュールに含まれている関数や変数群は先頭に *モジュール名.* を付けることで他のモジュールを import しても名前が重複することがありません。このようにオブジェクトの集合を区分するのが名前空間です。名前が重複する場合は import as で別の名前空間を作ることができます。メソッドにアクセスする場合は、*モジュール名. クラス名. メソッド名* のようにします。

最上位であるスクリプトは__main__という名前で、Python のシェルに__name__と打鍵して確認することができます（__name__は Python で定義済の特殊属性です)。

※オブジェクトへのアクセスは id を代入した変数をとおしても行えます。名前空間を指定しなければブロック内で名前の定義を探します

（3）スコープ（ブロック）

デフォルトでは変数はブロックのローカル変数として定義（名前の束縛／バインディング）されます。ブロックはモジュール、関数、クラス、対話的に実行されたコマンド等が該当します。

包含する（関数内に定義した関数等）ブロックで宣言した変数を更新したいのであれば nonlocal や global の宣言が必要です。何も指定をしなければ変数名は代入処理を囲む最小範囲のブロックの中からのみ探され外部の同一の名前の変数を更新することはありません。

モジュールの外に開示したくない変数名や関数名は del 文で名前の解放（アンバインド）を行うことができます。

## 3.6. クラス／インスタンスオブジェクト／属性

前項の関数オブジェクトでオブジェクト指向的なことは大概できてしまいそうですが、クラスでなければできないまたはクラスを使うと容易になる内容がいくつかあります。

① 異なるデータ属性値を持つインスタンスオブジェクトを並列で生成することができます
② 関連するデータや処理を関連付けて開発することができます
③ 既存クラスに対して継承や属性の上書きによる拡張ができます
④ __new__(cls[, ...])、__init__(self[, ...]) 等、インスタンス生成時に初期化する機構を持ちます

（1）クラス

クラスオブジェクトに対しては属性（attribute）の参照とインスタンスの生成ができます。属性はクラスに定義された変数とメソッドで、クラス名と属性をドット”.”で繋ぎ *クラス名. 属性* で参照します。

（2）インスタンスオブジェクト

var=*クラス名*() と書くとクラスをインスタンス化したオブジェクトを var に代入します。インスタンスに対して後から属性の追加もできます。

（3）属性（アトリビュート）

属性にはデータ属性とメソッドがあります。インスタンスオブジェクトの属性はインスタンス毎に個別のオブジェクトになります。

<クラスの定義、使用例>

```
class classA():
    a = 'This class is classA' # クラス変数（クラス定義の直下に定義）
    def __init__(self, a):     # インスタンス初期化（self にインスタンスが入り自動で呼び出し）
        print('①__inint__ para=', a)
        self.a = a             # インスタンス変数（インスタンス毎に確保される）

    @classmethod               # クラスメソッドのデコレータ（直下のメソッドがクラスメソッドになる）
    def classA_classm(cls):    # クラスメソッド（cls にはクラスが入る）
        print('④classA_classm cls.a=', cls.a)

    def classA_insm(self):     # デコレータが無いメソッドは、インスタンスでのみ呼び出される
        print('②-1 classA.a=', classA.a)
        print('②-2 self.a  =', self.a)

a=classA(100)          # ①classA のインスタンス生成(__init__)の引数 a にパラメータ(100)が入ります
a.classA_insm()        # ②classA のインスタンスメソッドを呼び出し
print(classA.a)        # ③classA のクラス変数を表示
classA.classA_classm()# ④classA のクラスメソッドを呼び出し
print(a.__dict__)      # ⑤インスタンスが持っている属性を全て表示
```

```
=== 以下の出力になります ===
①__inint__ para= 100
②-1 classA.a= This class is classA
②-2 self.a  = 100
This class is classA
④classA_classm cls.a= This class is classA
{'a': 100}
```

※クラス変数とインスタンス変数、クラスメソッドとインスタンスメソッド

・インスタンスメソッドの一つ目の引数に Python がインスタンスオブジェクトを付加します

・クラスメソッドの一つ目の引数には Python がクラスオブジェクトを付加します

・__init__が定義されていればインスタンス化の際に自動的に呼び出されます
（インスタンスメソッドとして明示的に、または*クラス名*.__init__(インスタンス,..)でも呼び出し可能です）

<クラスの継承の例>

```
class classB(classA):    # classA を継承して classB を定義
    a = 'it classB'

b=classB('bb')
print('classB a=', b.a)
classB.classA_classm()
```

```
=== 以下の出力になります ===
classB a= bb
④classA_classm cls.a= it classB
```

※継承しなくても、classB.meth=classA.classA_classm のように属性をコピーすることができます

## 3.7. 定義済みの属性（特殊属性）

Python が予め定義している属性があります。

__name__ モジュール、クラス、関数、メソッド、デスクリプタ、ジェネレータインスタンスの名前が設定されています

＜使用例＞

if __name__ == '__main__'

モジュールがスクリプトとして実行（python コマンドのパラメータで渡す）したとき、__name__には'__main__'という文字列が入ります

*object*.__dict__ オブジェクトに定義されている属性が設定されている属性です

＜使用例＞

```
class test():
    def __init__(self):
        self.var1 = 100
        self.var2 = 'abc'

t=test()
t.var3 = 'xyz' # インスタンスに追加した属性も登録されます
print(t.__dict__)
---------
{'var1': 100, 'var2': 'abc', 'var3': 'xyz'}
```

※モジュールやインスタンスの定義内容は__dir__() 関数を使ってみることもできます

＜実行例①＞

```
datetime.__dir__()
['__name__', '__doc__', '__package__', '__loader__', '__spec__', '__file__', '__cached__',
'__builtins__', '__all__', 'sys', 'MINYEAR', 'MAXYEAR', 'timedelta', 'date', 'tzinfo',
'time', 'datetime', 'timezone', 'datetime_CAPI']
```

＜実行例②＞

```
d=datetime.datetime(2030, 1, 1)
d.__dir__()
['__new__', '__repr__', '__hash__', '__str__', '__getattribute__', '__lt__', '__le__',
'__eq__', '__ne__', '__gt__', '__ge__', '__add__', '__radd__', '__sub__', '__rsub__', 'now',
'utcnow', 'fromtimestamp', 'utcfromtimestamp', 'strptime', 'combine', 'fromisoformat',
'date', 'time', 'timetz', 'ctime', 'timetuple', 'timestamp', 'utctimetuple', 'isoformat',
'utcoffset', 'tzname', 'dst', 'replace', 'astimezone', '__reduce_ex__', '__reduce__',
'hour', 'minute', 'second', 'microsecond', 'tzinfo', 'fold', '__doc__', 'min', 'max',
'resolution', 'fromordinal', 'fromisocalendar', 'today', 'strftime', '__format__',
'isocalendar', 'isoweekday', 'toordinal', 'weekday', 'year', 'month', 'day', '__setattr__',
'__delattr__', '__init__', '__subclasshook__', '__init_subclass__', '__sizeof__',
'__dir__', '__class__']
```

## 4. Web サービスの実装

Python にも Web アプリケーション開発のフレームワークが幾つかありますが、テストパターンに従った少数のバリエーションの応答を返すのであれば標準ライブラリで事足ります。

以下に Python 組み込みの wsgiref.simple_server を使った作り方とコードを説明します。

### （1）要件

以下の要件に基づいた実装例です。

① リクエスト毎にレスポンスの内容を変える（ローテーション）

② レスポンスは JSON 形式で行う

③ http ステータスとレスポンスの内容は予めファイルに保存しておく

### （2）ファイルの構成

```
＜開発フォルダ　¥WSGI＞
｜  simple_server.py     …サーバの本体となるモジュール
｜  exmple_service.py    …レスポンスを編集します
｜  file_control.py      …ファイルからレスポンス 1 件分のデータを切り出します
｜
├──data
｜      conf.csv        …テストデータのファイル名、データ区切文字列、n 件目を指定します
｜      data.txt        …テストデータを書いたファイル
｜
└──__pycache__          …Python が実行時につくるコンパイル結果（消しても差し支えありません）
       exmple_service.cpython-310.pyc
       file_control.cpython-310.pyc
```

### （3）テストデータの内容（data ディレクトリ）

① conf.csv

見出しとデータの 2 レコードで構成し、内容は以下のとおり。

＜テストデータファイル名＞、＜区切り文字列＞、＜次に使う順番…AP で更新します＞

```
data_file,delmiter,start_pos
data.txt,@record,1
```

② data.txt（①のテストデータファイル名）

conf.csv で指定した区切文字列でレスポンスを区切り、http ステータス（key=”status”）とレスポンスデータ（key=”resdata”）で 1 件の JSON データとして書きます。

```
@record
{”status” : ”200 OK”,
  ”resdata” : {
 (略)
  }
}
@record
{”status” : ”201 OK”
,”resdata” :{
 (略)
  }
}
```

（1 件）

## 4.1. コードの内容

（1）simple_server.py

```
'''
wsgiref --- WSGI ユーティリティとリファレンス実装...を改変
https://docs.python.org/ja/3/library/wsgiref.html
【改変内容】
Pyhon 公式ページのサンプルに json の取扱い（request/response）を追加
Windows の curl コマンドでリクエストを出す場合は、以下のように json の
キー、値を囲む引用符を¥でエスケープしないと json の解析でエラーになります。
curl -d {¥"key¥":¥"value¥"} -i -H "Content-Type: application/json; charset=Shift-JIS" http://localhost:8000
'''
from wsgiref.simple_server import make_server

import exmple_service
import logging

logging.basicConfig(level=logging.DEBUG) # CRITICAL,ERROR,WARNING,INFO,DEBUG,NOTSET

# A relatively simple WSGI application.
def simple_app(environ, start_response):
    logging.info('env path=%s,method=%s,CONTENT_TYPE=%s'
                 , environ['PATH_INFO']
                 , environ['REQUEST_METHOD']
                 , environ['CONTENT_TYPE']
                 )

    # リクエストの URL パスから、対象サービスを決定します
    match environ['PATH_INFO']:
        case "/":
            ret = exmple_service.accept_req(environ, start_response)

        case "/test1":
            ret = exmple_service.accept_req(environ, start_response)

        case _:
            raise Exception('指定されたパスは、サービス未開通です')


    return ret

with make_server('', 8000, simple_app) as httpd:
    print("Serving on port 8000...")
    httpd.serve_forever()
```

（2）exmple_service.py

```
1  import logging
2  import json
3  import re
4  import file_control
5
6  def accept_req(environ, start_response):
7      '''
8      HTTP リクエストを 1 件処理します。
9      【前提条件】REQUEST__METHOD=POST、CONTENT_TYPE=JSON
10
11     '''
12     # ヘッダーから文字コードを取り出します
13     # header の charset=～から文字コードを取得。設定がなかったら utf-8 を拾うように加工
14     encode = re.search('charset=(?P<encode>[a-zO-9¥-]+)'
15                        , environ['CONTENT_TYPE'] + '; charset=utf-8'
16                        , flags=re.IGNORECASE).group('encode')
17     if encode == None: # 上でデフォルト値を設定しているので、ここは真にならないはず
18         encode = 'UTF-8'
19     logging.info('encode=%s', encode)
20
21     # ヘッダーに入っているコンテンツ長の POST 内容を取り出し、JSON に変えます
22     content_length = int(environ.get('CONTENT_LENGTH', 0)) # コンテンツ長さ
23     req_body = environ['wsgi.input'].read(content_length).decode(encode) # 長さ分だけ読込
24     logging.info('request=%s' , req_body)
25     req_json = json.loads(req_body)
26     logging.info('json=%s' , req_json)
27
28     # レスポンスを作る為のファイルを読み書きするモジュールを呼び出します
29     data = file_control.FileCtrl('data').get_content()
30     ret = json.loads(data)
31
32     # http ヘッダーの内容を再設定します
33     status = ret['status']
34     headers = [('Content-type', 'application/json; charset=utf-8')]
35     start_response(status, headers)
36     # wsgi の出力は wite()1 回分を 1 bytes オブジェクトで、n 行分の list にする
37     # ↓は str 型の encode メソッドで json 全体を bytes にし、[角括弧]で囲んで list にしている
38     return [json.dumps(ret['resdata'], ensure_ascii=False).encode('utf-8')]
```

（3）file_control.py

```
import logging
import csv
import mmap

class FileCtrl:
    def __init__(self, path):
        '''
        パラメータで受け取った path のディレクトリにある conf.csv ファイルを
        csv.DictReader を使って dict（辞書）型のインスタンス変数に読み込みます
        '''
        logging.debug('get_config')
        self.conf_file = path + r'/conf.csv'
        logging.info('confifile=%s', self.conf_file)
        with open(self.conf_file, newline='') as conf:
            reader = csv.DictReader(conf)
            self.confdic = [row for row in reader][0]
            self.path = path

        logging.debug(self.confdic)

    def update_config(self):
        '''
        インスタンス変数：confdic の内容で conf.csv ファイルを書き換えます
        '''
        logging.debug('update_config')
        with open(self.conf_file, 'w', newline='') as conf:
            writer = csv.writer(conf)
            writer.writerow(self.confdic.keys())
            writer.writerow(self.confdic.values())

    def get_content(self):
        '''
        区切文字を探してデータを切り出します（最後まで行ったら最初に戻ります）
        大きなファイルも扱えるように mmap を使って区切文字を探しバイナリで切り出していますが、小さなファイルであれば readlines().find で見つけたほうが早いかもしれません。
        '''
        logging.debug('get_content conf=%s', self.confdic)
        with open(self.path + '/' + self.confdic['data_file'], 'rb', 0) as data:
            s = mmap.mmap(data.fileno(), 0, access=mmap.ACCESS_READ)
            i = 0
            sPos = int(self.confdic['start_pos'])
            fstr = self.confdic['delmiter'].encode()
            # テストデータの切り出し開始場所を探す
            while i < sPos:
                fByte = s.find(fstr)
                logging.debug('i=%s, sPos=%s', i, sPos)
                if fByte == -1:
                    if i > 0:
                        sPos = 1
                        i = 0
```

```
51                          s.seek(0)
52                      else:
53                          logging.error('%s にデータが見つかりません', self.path + '/' + self.confdic['data_file'])
54                          raise Exception('テストデータがみつかりません')
55                  else:
56                      sByte = fByte
57                      s.seek(fByte + len(fstr))
58                      i += 1
59
60              # テストデータの切り出し終端を探す
61              s.seek(fByte + len(fstr))
62              fByte = s.find(fstr)
63              ret = s.read(None if fByte == -1 else fByte - sByte - len(fstr)).decode()
64              # 読み出し位置のカウントアップ
65              self.confdic['start_pos'] = 1 if fByte == -1 else sPos + 1
66              logging.info('fByte=%s,sPos=%s', fByte, self.confdic['start_pos'])
67
68          # conf ファイルの書き戻し
69          self.update_config()
70          logging.info('ret=%s', ret)
71          return ret
```

### 4.2. コード説明

この実装で使っている、汎用的に使えそうな標準ライブラリの機能について説明します。

#### 4.2.1. ロギング（logging）

logging を使ってコンソールやファイルに実行ログを出力することができます。複数の logger を生成してログ毎に出力先を変えたりフォーマットを変えることができますが、デフォルトで logging を使う場合は、以下のようになります。

・出力のレベルは CRITICAL、ERROR、WARNING、INFO、DEBUG、NOTSET

・ログ出力は logging.*xxx*()で行い、xxx は critical(),error()…のように設定するレベルを使います（simple_server.py:19 行目、他）

・複数のモジュールから呼び出す場合は、呼び出し階層の最上位で logging.basicConfig を使い環境設定をしておきます（simple_server.py:15 行目）

※例えば logging.basicConfig(level=logging.DEBUG)を実行しておくと各モジュールに書かれた DEBUG 以上のレベルのログが出力されます

#### 4.2.2. http ヘッダーの処理（正規表現：re）

クライアント／サーバ間で通信を行う場合、使う文字コード等を予め決めておく必要があります。また、公開していない情報を扱う場合は最初に認証手続きが必要になります。Web の場合はそれらの情報を http ヘッダーに格納した文字列として交換するので、文字列を抜き出すために正規表現（標準ライブラリの re）を使います。(exmple_service.py:14~16)

```
encode = re.search('charset=(?P<encode>[a-z0-9¥-]+)'
                   , environ['CONTENT_TYPE'] + '; charset=utf-8'
                   , flags=re.IGNORECASE).group('encode')
```

このコードは以下のように動作します。

・http ヘッダーの Content_type から charset=という文字列に続く英数字と - の連続を切り出し、encode という名前を付けます(?P<encode>*保存対象を表す正規表現*)
・文字列は大小文字どちらでも合致するようにパラメータに flags=re.IGNORCASE を付加します
・( )で囲んだ正規表現に合致すると、re.search()の実行結果 group に該当文字列が保存されます
※この後、environ['wsgi.input'].read(content_length).decode(**encode**)のように使って指定された文字コードでリクエストボディーを読み取ります

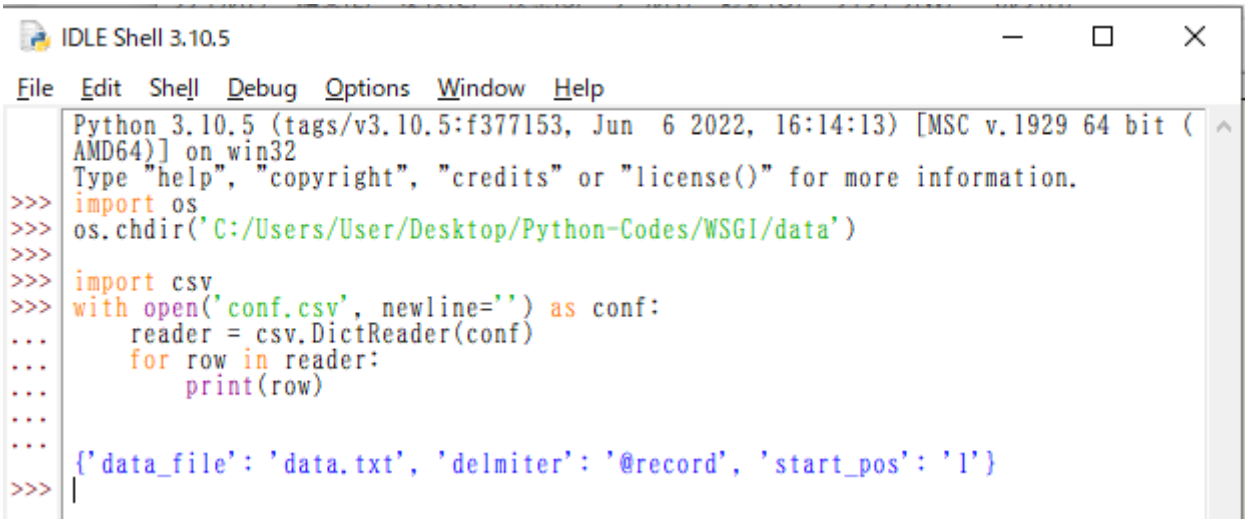
### 4.2.3. with 文と CSV ファイル読込み

CSV ファイルは csv モジュールの DictReader で見出し行を反映した形で dict 型に読み込むことができます（file_control.py:15～16 行め）。

```
with open(self.conf_file, newline='') as conf: # ①
    reader = csv.DictReader(conf)              # ②
    self.confdic = [row for row in reader][0]  # ③
```

① with 文はブロックに初期化処理と終了処理を付加したクラスのように動作し、open と同時に使った場合は、ブロックの処理を抜ける前にファイルの close を終了処理として実行します
∴ with open ~と書いた場合はブロックの終了方法に関わりなく close が不要になります
② csv モジュールの DictReader を使うと csv ファイを辞書(dict)型に読み込むことができます
パラメータにファイルオブジェクト（例の中では conf）以外を指定しなかった場合、ファイルの先頭行を列名として使います。(列名がファイル中に書かれていない場合は、パラメータに fieldnames=*列名* を与えます)
③ reader(conf ファイルを読み込む DictReader のオブジェクト)は、for 文で参照されるたびに 1 件の csv を返します。更に[ ]の中に文や式が書かれているのはリストの内包表記で、この実行結果がリストの要素になります。この行は細かくは、以下のように動作します。
・[]内の最左端の row がリストに登録されます。この row は右側の for 文の実行結果です
・reader が eof を返すまで for row in reader が実行されて row（辞書型）が書き換わります
・リストができた後に行の末尾にある[0]がインデックスとして評価され、リストの 1 件目が self.confdic に代入されます

※実際にどのように動作するかは、IDLE の Shell からコマンドを逐次実行して確認できます



#### 4.2.4. dict 型から CSV ファイル書込み

DictReader に対応するクラスとして csv.DiceWriter がありますが、こちらは fieldnames パラメータが必須なので列名を指定する必要があります。dict 型は{キー:値 , …}なので csv.writer で見出し行としてキーを、データ行として値を書きだすことで csv.DictReader で読み込んだファイルのフォーマットを再現することができます（file_control.py:27～29 行め）。

```
writer = csv.writer(conf)                    # ①
writer.writerow(self.confdic.keys())         # ②
writer.writerow(self.confdic.values())       # ③
```

※with 文のブロック内に書くのは他のファイルアクセスと同様です

① csv.writer のオブジェクトを作ります

② dict 型の全てのキーを取り出して書き出します

③ dict 型のデータだけ書きだします

#### 4.2.5. 三項演算子

Python の三項演算子は処理の後ろに if 文を書き、以下のようにします。

*＜条件が真の時の処理＞* if *条件* else *＜条件が偽の時の処理＞*

（file_control.py:63 行め）

```
ret = s.read(None if fByte == -1 else fByte - sByte - len(fstr)).decode()
```

下線部が三項演算子になっていて、read メソッドのパラメータを fByte の値で決めています。

・真の場合は、None（オブジェクトが存在しないことを示す、None 型のオブジェクトです）

・偽の場合（fByte が-1 でないとき）、fByte から sByte と fstr のバイト数を減じた値を s.read()のパラメータとします

#### 4.2.6. 大きなファイルから文字列を探す（mmap）

mmap（メモリマップドファイル）はファイルをメモリに割り当てて、メモリアクセスを通してファイルにアクセスするものです。一般ファイルは AP からの I/O 命令の都度レコードやセクター単位でデバイス⇔キャッシュ⇔ユーザ領域とデータ転送を行いますが、mmap はファイル全体がメモリにあるものとしてアクセスを行う OS の API です。仮想記憶を扱うように動作し、OS レベルで必要または必要になりそうな部分をディスクからメモリに読み込みます。

mmap に関する処理は、file_contorol.py の 38 行目以降

① `s = mmap.mmap(data.fileno(), 0, access=mmap.ACCESS_READ)`

ファイルナンバーとファイルサイズ（0 を指定するとファイル全体の大きさと同)、アクセス制御を指定します。Windows に関しては、固有なパラメータや 0 バイトのファイルが扱えない等の制約があります。

② 他のメソッド

文字列を探す　…s.find(*検索文字列*)

find/read 等の開始位置を設定　…s.seek(*現在位置を設定するバイト数*)

現在位置からファイルを読込む　…s.read(*バイト数*)

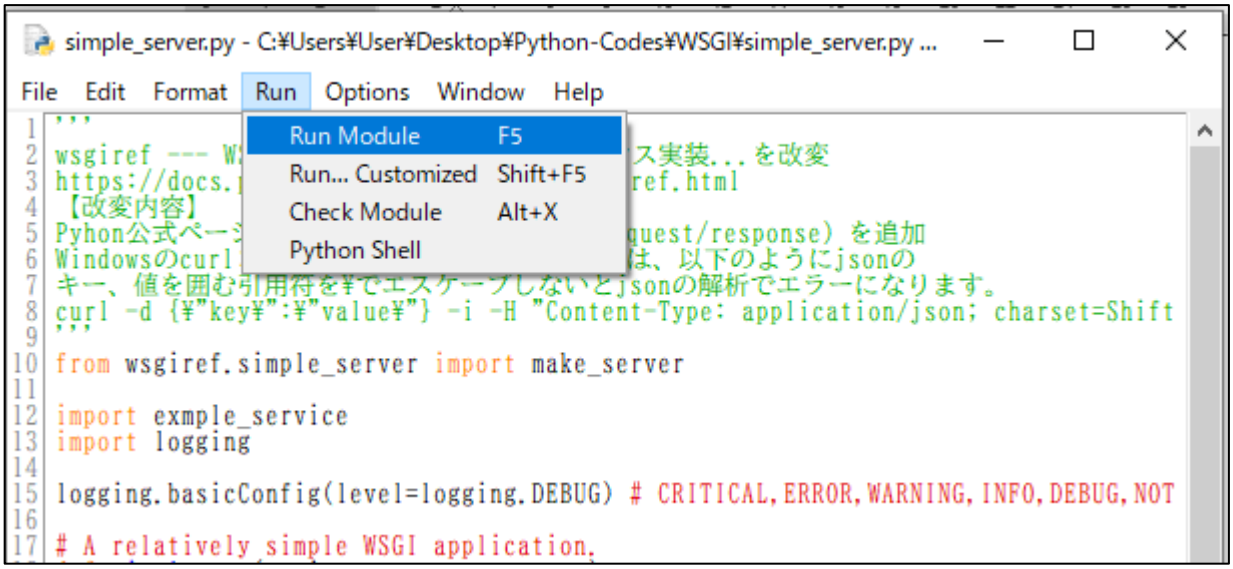
## 5. サービスの起動

simple_server.py を起動すると 39 行目の with make_server('', 8000, simple_app) as httpd: のブロックが実行され、ポート番号 8000 でサービスが常駐します。

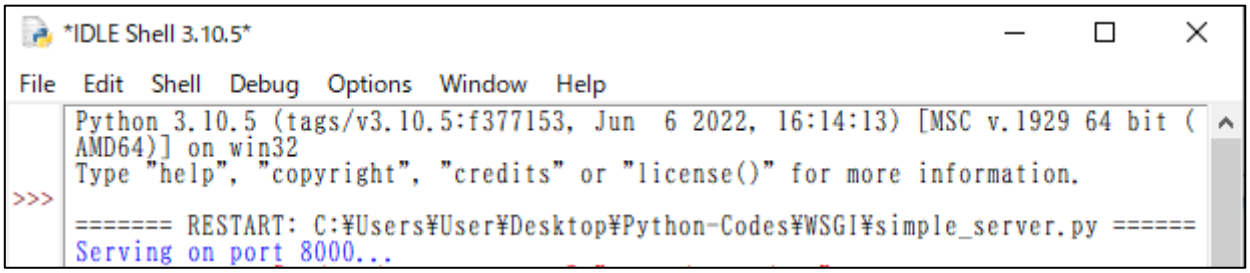
主要な起動の方法には 2 種類あります。

### 5.1. IDLE の Editor から起動

Simple_server の編集画面から F5 を押下します



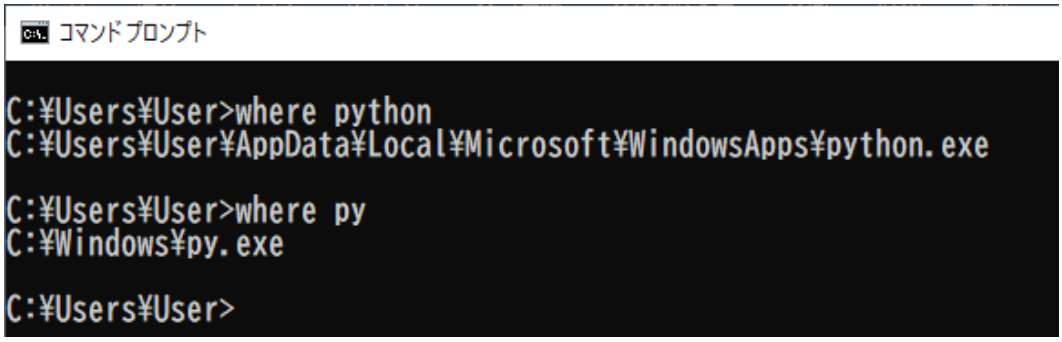

起動して Shell 画面に simple_server.py が出しているメッセージが表示されます。




修正をする都度、保存して simple_server.py の編集画面から F5 押下でサービスは再起動します。

### 5.2. コマンドプロンプトから py コマンド

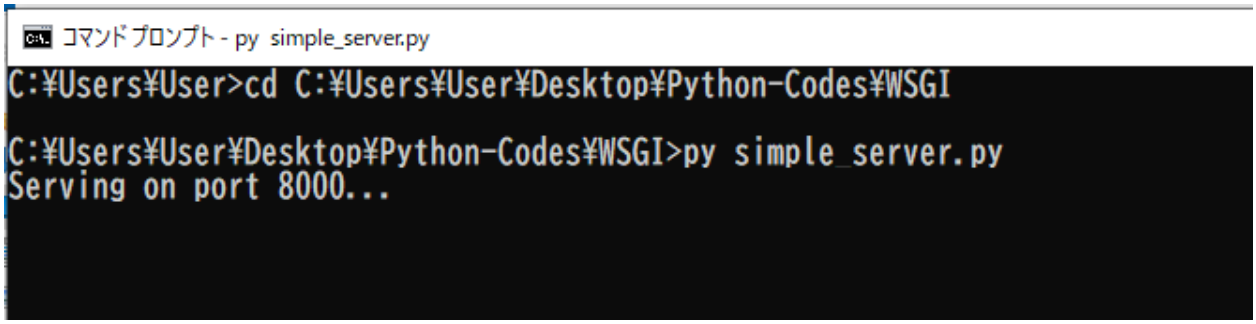
Windows には Microsoft フォルダに python.exe が入っていますが、これは Microsoft Store からインストールするためのもので、自分でインストールした Python の実行には py.exe を使います。



カレントディレクトリを simple_server.py のフォルダに移動し、py コマンドを実行します。

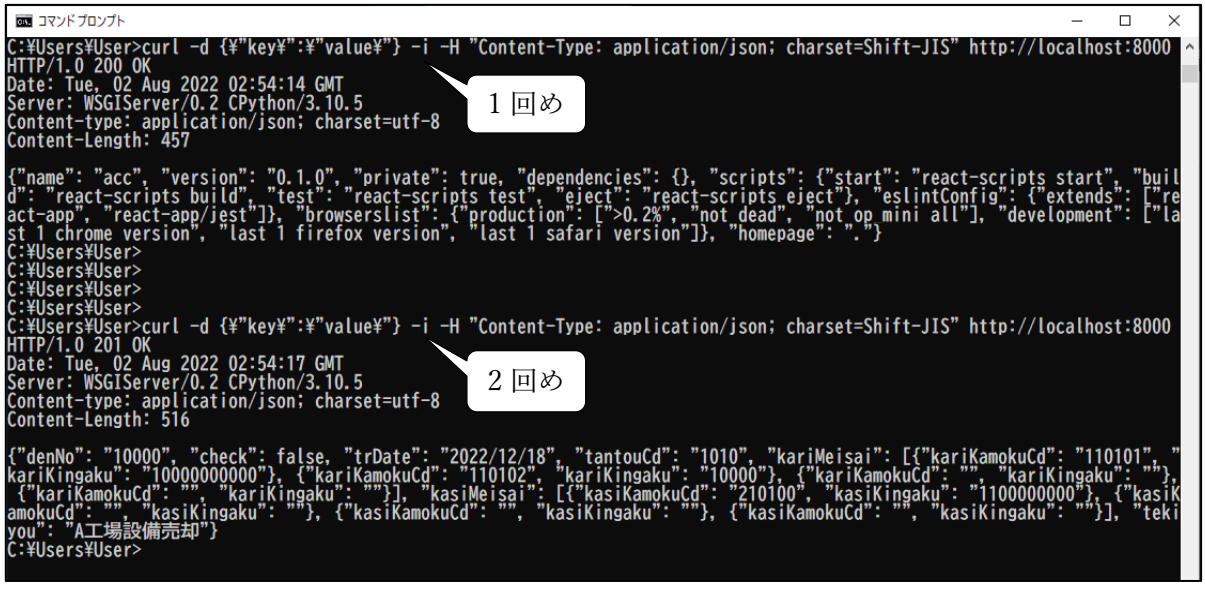
> py simple_server.py




## 6. クライアントからの呼出し

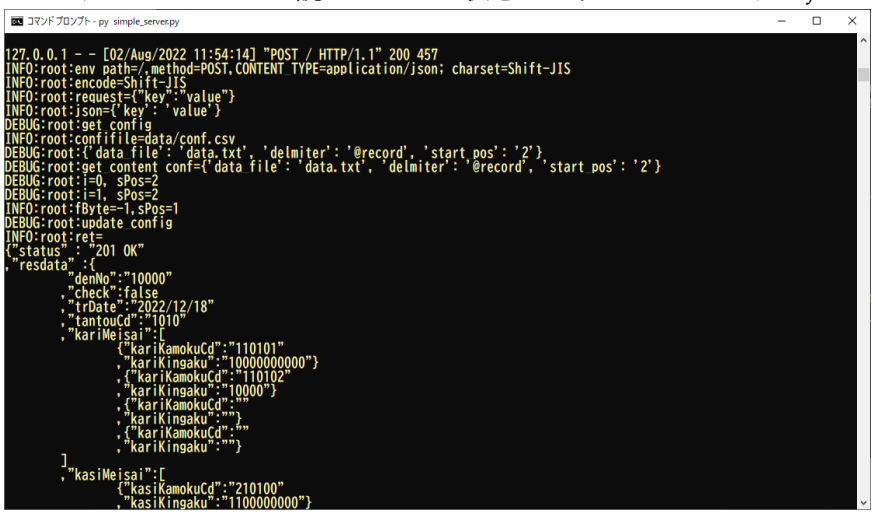
サービスが起動しているホスト:ポート（今回の例では、localhost:8000）に post リクエストを送ります。今回の例では、リクエストデータはログに出力するだけで他には使っていません。

以下が curl コマンドでリクエストを 2 回連続して送った例です。




起動したコマンドプロンプトまたは IDLE Shell に、下のようにログが出力されます。

ログ↓のファイルから読んだままの状態から、レスポンス↑は json-loads/dumps で整形します




以上