Python3 応用
（ユーザーインターフェース）

＜ 目次 ＞

セイ・コンサルティング・グループ株式会社

Self-Actualization for You SAY

セイ・コンサルティング・グループ

1章 GUIアプリケーション（tkinterモジュール）

「tkinter」は「Tk」というGUIを構築するためのライブラリをPythonから使うためのフロントエンドモジュールです。tkinterはPythonの標準モジュールとして提供されていてimportすればすぐに使えます。
tkinterを使うと、ボタンやリストのような部品のついたGUIアプリケーションを簡単に作成することができます。

tkinterのTkクラスは引数なしでインスタンス化します。これは通常アプリケーションのメインウィンドウになります。

このメインウィンドウにボタンなどの部品を配置してGUIアプリケーションを組み立てていきます。この部品のことを一般的にウィジェット（Widget）と呼びます。

ウィジェットにはボタンやチェックボックス、リストボックス、テキスト入力フィールドなどいろいろな部品がクラスとして定義されています。
代表的なウィジェット

ウィジェットのクラス名	説明
Label	テキストのラベル
Button	ボタン
Entry	１行テキスト入力フィールド
Checkbutton	チェックボタン
Listbox	リストボックス
Menu	メニューバー
Radiobutton	ラジオボタン
Scale	スケール（スライダ）
Scrollbar	スクロールバー
Text	複数行テキスト入力
Canvas	画像などを表示する部品

ウィジェットは次の形式で生成します。

```
widget = widgetClass(parent, option = value, .... )
```

widgetClassには生成するウィジェットのクラス名を、parentにはメインウィンドウやほかのウィジェットのオブジェクトを指定します。返り値は生成したウィジェットのオブジェクトです。

主なウィジットのオプションには下表のものがあります。

オプション	説明	備考
foreground または fg	文字や線を描くのに使用する色を指定	共通
background または bg	背景色の指定	共通
text	ウィジェット内に表示されるテキスト	多く共通
textvariable	テキストを格納するオブジェクトを指定	多く共通
height	ウィジェットの高さ	共通
width	ウィジェットの幅	共通
relief	枠のスタイルを指定 tk.FLAT（デフォルト）,tk.RAISED,tk.SUNKEN, tk.GROOVE,tk.RIDGE	共通
command	ボタンがクリックされたときの関数を指定	Button Checkbutton Radiobutton など
variable	ウィジェットの値を格納するオブジェクトを指定	Checkbutton Radiobutton Scale など

Tkinter のクラスの詳細は「Tkinter 8.5 reference: a GUI for Python」（https://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html）を参照のこと

サンプル

```
import tkinter as tk

root = tk.Tk()
l = tk.Label(root,text='Hello Python!')
b = tk.Button(root,text="Quit",command = root.destroy)
l.pack()
b.pack()
root.mainloop()
```

メインウィンドウを生成します。

```
root = tk.Tk()
```

メインウィンドウに「Hello Python!」という文字のラベルを生成します。

```
l = tk.Label(root,text='Hello Python!')
```

メインウィンドウにボタンを生成します。ボタンには「Quit」という文字を表示し、押されたときにウィンドウを閉じる処理を定義します。

```
b = tk.Button(root,text="Quit",command = root.destroy)
```

ウィジェットをを整列させるために pack メソッドを呼び出します。

```
l.pack()
b.pack()
```

メインウィンドウの mainloop メソッドにより処理をウィンドウに渡します。

```
root.mainloop()
```

１．１ ウィジェットの配置

ウィジェットを配置するには pack()、grid()、place()の３つのメソッドがあります。

pack メソッドは縦または横に一列に並べるのに使い、grid メソッドは縦横の格子（グリッド）を想定して行（row）と列（column）を指定して２次元的に配置するのに使います。place メソッドは x と y のピクセル座標を使って配置するのに使います。

pack メソッドの主なオプション

属性	説明
anchor	配置可能なスペースに余裕がある場合にウィジェットをどこに配置するかを指定 「tk.C」中央寄せ（デフォルト）、「tk.W」左寄せ、「tk.E」右寄せ、「tk.N」上寄せ、 「tk.S」下寄せ、「tk.NW」左上寄せ、「tk.SW」左下寄せ、「tk.NE」右上寄せ、 「tk.SE」右下寄せ
expand	親のウィジェットが大きくなったとき、ウィジェットが連動して大きくなるかどうかを指定 「１」を指定すると連動し、「０」を指定すると連動しない。デフォルトは「０」
fill	ウィジェットが空いているスペースを埋めるかどうかの指定 「tk.NONE」元のサイズを保持、「tk.X」横に広がる、「tk.Y」縦に広がる、 「tk.BOTH」縦横に広がる
padx	ウィジェット外側の横の隙間を指定
pady	ウィジェット外側の縦の隙間を指定
ipadx	ウィジェット内側の横の隙間を指定
ipady	ウィジェット内側の縦の隙間を指定
side	どの方向から詰めていくかを指定 「tk.TOP」上から詰める（デフォルト）、「tk.LEFT」左から詰める、 「tk.RIGHT」右から詰める、「tk.BOTTOM」下から詰める

grid メソッドの主なオプション

属性	説明
column	配置する列
columnspan	何列にわたって配置するかを指定（デフォルトは１）
row	配置する行
rowspan	何行にわたって配置するかを指定（デフォルトは１）
padx	ウィジェット外側の横の隙間を指定
pady	ウィジェット外側の縦の隙間を指定
ipadx	ウィジェット内側の横の隙間を指定
ipady	ウィジェット内側の縦の隙間を指定
sticky	pack の anchor と fill を合わせた属性 スペースに余裕がある場合にどこに配置するか、どのように引き伸ばすかを指定 「tk.C」中央寄せ（デフォルト）、「tk.W」左寄せ、「tk.E」右寄せ、「tk.N」上寄せ、 「tk.S」下寄せ、「tk.NW」左上寄せ、「tk.SW」左下寄せ、「tk.NE」右上寄せ、 「tk.SE」右下寄せ 位置決めには値を単独で、引き伸ばす場合には伸ばす方向に＋の後に引き伸ばす方向の値 例えば左寄せは「tk.W」、上寄せは「tk.N」、 左右に引き伸ばすには「tk.W+tk.E」、上下に引き伸ばすには「tk.N+tk.S」、 全体に引き伸ばすには「tk.W+tk.E+tk.N+tk.S」

place メソッドの主なオプション

属性	説明
x	ウィンドウ左端からの位置
y	ウィンドウ上端からの位置

1．2 ウィンドウのタイトルとサイズ

ウィンドウのサイズの指定は geometry メソッドを使います。サイズは引数で「横ピクセル数 x 縦ピクセル数」の文字列を渡します。縦ピクセル数と横ピクセル数の間に小文字の x（エックス）を挟みます。

例えば、横 300px、縦 200px のウィンドウサイズを指定するには以下のようにします。

```
root.geometry('300x200')
```

ウィンドウのタイトルは title メソッドを使って引数にタイトル名の文字列を渡して設定します。

例えば、タイトルに「Hello」の文字を指定する場合は以下のようにします。

```
root.title('Hello')
```

1．3 ウィジェットとの値のやりとり

ウィジェットの生成後にウィジェットからデータを取得したり、設定したりする場合は以下のクラスのオブジェクトを介して行います。

具体的にはウィジェットの生成時のオプションの textvariable や vriable に以下のクラスのオブジェクトを定義します。

クラス名	説明
IntVar	整数を保持するクラス
DoubleVar	倍精度の実数を保持するクラス
StringVar	文字列を保持するクラス
BooleanVar	真偽値を保持するクラス

このオブジェクトから値を取り出すには get()メソッド、値を設定するには set(値)メソッドを使用します。

```
import tkinter as tk

def setLabel():
    lstr.set(tstr.get())

root = tk.Tk()
lstr = tk.StringVar()
lstr.set('Hello Python!')
tstr = tk.StringVar()
l = tk.Label(root,textvariable=lstr)
t = tk.Entry(root,textvariable=tstr)
b = tk.Button(root,text="Push",command = setLabel)
l.pack()
t.pack()
b.pack()
root.mainloop()
```

１．４ Frame クラスを継承した GUI アプリケーション

Tkinter を使って GUI アプリケーションの作り方として、Frame クラスを継承したクラスを作り、メソッドを定義して独自の機能を追加する方法を紹介します。

```
import tkinter as tk

class MyFrame(tk.Frame):
    def __init__(self,master=None):
        super().__init__(master)
        self.lstr = tk.StringVar()
        self.lstr.set('Hello Python!')
        self.tstr = tk.StringVar()
        self.l = tk.Label(root,textvariable=self.lstr)
        self.t = tk.Entry(root,textvariable=self.tstr)
        self.b = tk.Button(root,text="Push",command = self.buttonClick)
        self.l.pack()
        self.t.pack()
        self.b.pack()
        self.pack()

    def buttonClick(self):
        self.lstr.set(self.tstr.get())

if __name__ == '__main__':
    root = tk.Tk()
    mf = MyFrame(root)
    mf.mainloop()
```

１．５ イベントの取り扱い

GUI はユーザーの操作などのイベントに応答する形でプログラムを書く必要があります。イベントと対応する処理を結びつけるには、Frame やウィジェットの bind()、bind_all()メソッドを使います。

クラス名	説明
`w.bind(event,response)`	このウィジェット（w）にフォーカスがあるときに、event で指定したイベントが発生した場合に、response で指定された関数が呼ばれる
`w.bind_all(event,response)`	このウィジェット（w）以下のどのウィジェットにフォーカスがあるときでも、event で指定したイベントが発生した場合に、response で指定された関数が呼ばれる

イベントの種類

イベント名	説明	備考
`KeyPress` または `Key`	キーボードのキーを押下したときに発生するイベント	前に「Shift-」「Control-」「Alt-」「Lock-」を付加することによりキーの同時押しのイベントを指定できる また後ろに「-」に続いてキー名を付加することで特定キーの押下のイベントを指定できる 例: <Control-Key-a>
`KeyRelease`	キーボードのキーを離したときに発生するイベント	
`ButtonPress` または `Button`	マウスのボタンを押下したときに発生するイベント	前に「Shift-」「Control-」「Alt-」「Lock-」を付加することによりキーの同時押しのイベントを「Double-」「Triple-」を付加することによりダブルクリック、トリプルクリックのイベントを指定できる また、後ろに「-」に続いてマウスボタン番号を付加することで特定のマウスボタンの押下のイベントを指定できる 例: <Shift-Button-1>
`ButtonRelease`	マウスのボタンを離したときに発生するイベント	
`Enter`	マウスがウィジェットの領域に入ったときに発生するイベント	
`Leave`	マウスがウィジェットの領域から出たときに発生するイベント	
`Motion`	マウスがウィジェットの領域で動いたときに発生するイベント	

```
import tkinter as tk

class MyFrame(tk.Frame):
    def __init__(self,master=None):
        super().__init__(master)
        self.lstr1 = tk.StringVar()
        self.lstr1.set('Hello Python!')
        self.lstr2 = tk.StringVar()
        self.lstr2.set('Hello Python!')
        self.l1 = tk.Label(root,textvariable=self.lstr1)
        self.l2 = tk.Label(root,textvariable=self.lstr2)
        self.l1.pack()
        self.l2.pack()
        self.pack()
        for x in ('<Button-1>','<Button-2>'):
            self.bind_all(x,self.mouse_click)
        self.bind_all('<Enter>',self.mouse_in)
        self.bind_all('<Leave>',self.mouse_out)
        self.bind_all('<Motion>',self.mouse_move)
        self.bind_all('<Key>',self.key_press)

    def key_press(self,event):
        self.lstr1.set('KEY {0}'.format(event.char))

    def mouse_click(self,event):
        self.lstr1.set('MOUSE CLICK')

    def mouse_in(self,event):
        self.lstr1.set('MOUSE IN')

    def mouse_out(self,event):
        self.lstr1.set('MOUSE OUT')

    def mouse_move(self,event):
        self.lstr2.set('MOVE {0},{1}'.format(event.x,event.y))


if __name__ == '__main__':
    root = tk.Tk()
    root.geometry('200x100')
    mf = MyFrame(root)
    mf.mainloop()
```

2 章 CUI アプリケーション（Curses モジュール）

curses モジュールは、可搬性のある高度な端末操作のデファクトスタンダードである curses ライブラリへのインタフェースを提供します。

curses が最も広く用いられているのは Unix 環境ですので Linux 環境や MacOS 環境の Python には標準で入っています。
しかし、Windows 環境では標準ライブラリに curses モジュールは入っていないので、PyPI から Windows 用の curses モジュールをダウンロードしてインストールする必要があります。

```
pip install windows-curses
```

Windows 用の curses モジュールの利用方法は、標準ライブラリの curses と同じです。

それでは、以下にサンプルプログラムを示します

```
#! /usr/bin/env python
# coding: utf-8
import curses

def curses_test(stdscr):
  (wh,ww) = stdscr.getmaxyx()
  while True:
    stdscr.addstr(wh//2-4,ww//2-6, "Welcome to Curses")
    stdscr.addstr(wh//2-3,ww//2-6, "Type q to exit")
    stdscr.addstr(wh//2-2,ww//2-6, "INPUT>")
    c = stdscr.getch()
    if c == ord("q"):
      break
    stdscr.addstr(wh//2,ww//2, "[" + chr(c) + "]")
    stdscr.refresh()

#curses の起動（wrapper を使うことで自動的に初期化される）
curses.wrapper(curses_test)
```

最下行で curses.wrapper()関数を呼び出しています。ここで curses を起動しています。
wrapper 関数を使うことで煩雑な初期化処理が簡素化できます。
wrapper 関数の引数に与えられた関数の中で curses の制御を行います。

wrapper の指定された関数には stdscr という引数が定義されています。
この stdscr が curses の提供する window オブジェクトで、この window オブジェクトを使って画面制御を行います。

window オブジェクトの主なメソッドには以下のものがあります。

getmaxyx()	ウィンドウの高さおよび幅を表すタプル(y,x)を返します
addstr(y,x,str)	(y,x)に文字列 str を描画します。以前ディスプレイにあった内容はすべて上書きされます。
getch()	文字を 1 個取得します。返される整数は ASCII の範囲の値となるわけではないので注意してください。ファンクションキー、キーパッドのキー等は 256 よりも大きな数字を返します。
refresh()	ディスプレイを即時更新し現実のウィンドウとこれまでの描画/削除メソッドの内容との同期をとります。
erase()	window をクリアします。
move(y,x)	カーソルを(y,x)の位置に移動します。

より詳細な情報については Python3 ドキュメントの標準ライブラリの curses（https://docs.python.org/ja/3/library/curses.html）の Window オブジェクトを参照してください。

3章 Webアプリケーション（flaskモジュール）

Pythonの Flaskモジュールを使って Webアプリケーションの実行環境を構築します。

Flaskとは Python用の Webアプリケーションフレームワークで、簡易的は Webサーバーの機能も提供するモジュールです。Pythonの標準モジュールではありません。

3．1 Flaskモジュールのインストール

Anacondaには Flaskモジュールはインストールされています。
Anaconda以外の環境の場合は pipでインストールしてください。

```
pip install Flask
```

以下のようなメッセージが出ればインストール完了です。

```
Installing collected packages: itsdangerous, click, MarkupSafe, Jinja2, Werkzeug, Flask
  Running setup.py install for itsdangerous ... done
  Running setup.py install for MarkupSafe ... done
Successfully installed Flask-1.0.2 Jinja2-2.10 MarkupSafe-1.0 Werkzeug-0.14.1 click-6.7 itsdangerous-0.24
```

> インターネット接続を Proxy経由で行うネットワーク環境では pip3コマンドがエラーになる場合があります。
> コマンドプロンプトで以下の Proxy関係の環境変数の設定コマンドを実行してから、再度 pip3コマンドを実行してください。
>
> ```
> set HTTP_PROXY=http://<proxy_server>:<port>
> set HTTPS_PROXY=http://<proxy_server>:<port>
> ```

3．2 サンプルプログラムの作成

はじめに動作確認のために以下のプログラムを「hello.py」という名前で作成します。

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return "Hello, World!"

if __name__ == '__main__':
    app.run(host="0.0.0.0")
```

ブラウザの URLで'/'へのアクセスがあった場合に、後続の関数を呼び出す

ブラウザに HTMLデータ（text/html）として"Hello,World!"を返す

importしての利用ではなく、このファイル自身を実行した場合には変数__name__に'__main__'が入っている

Webサーバーとして Webアプリケーションを実行する
hostで指定した IPアドレスで接続を待つ
”0.0.0.0”は全ての IPを意味する

3．3 サンプルプログラムの実行と動作確認

Windows では作成したファイルをダブルクリックで実行するか、コマンドプロンプトで作成したファイルのあるディレクトリに移動して以下のコマンドを実行します。

```
python hello.py
```

正しく実行できれば以下のようなメッセージが表示され、以後はブラウザからのアクセスがある度にログが出力されます。

```
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

ブラウザから「http://<IP アドレス>:5000/」にアクセスして「Hello,World!」と表示されれば正しく動作しています。

また、コマンドラインから HTTP の操作ができる curl コマンドがあると便利です。
CURL のサイトから Windows 用のバイナリをダウンロードできます。
（https://curl.haxx.se/download.html）
以下のようにコマンド入力すると、HTTP レスポンスデータが表示されます。

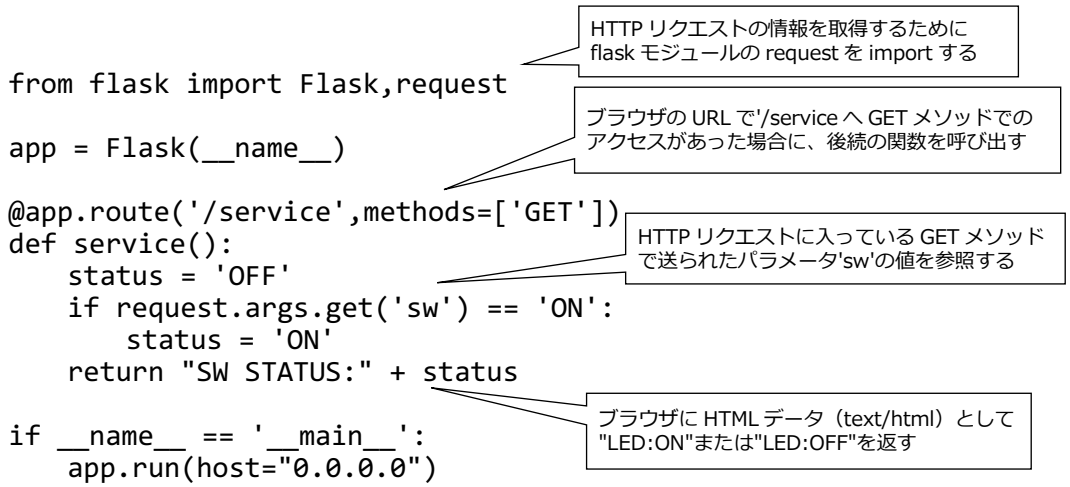
```
curl -i http://<IP アドレス>:5000/
```

ステータスコードが 200 で、HTTP ボディ部に「Hello,World!」と表示されれば正しく動作しています。

Web アプリケーションを終了するにはキーボードから「Ctrl + c」を入力します。

3．4 GET メソッドでデータを受け取る

@app.route で method を GET メソッドの受け取りを指定します。
GET メソッドでデータを受け取るには flask モジュールの request をインポートして、request.args.get()メソッドでパラメータを指定して受け取ります。




```
from flask import Flask,request

app = Flask(__name__)

@app.route('/service',methods=['GET'])
def service():
    status = 'OFF'
    if request.args.get('sw') == 'ON':
        status = 'ON'
    return "SW STATUS:" + status

if __name__ == '__main__':
    app.run(host="0.0.0.0")
```

3．5 GET メソッドでデータを受け取る動作確認

前の動作確認の要領でプログラムを起動し、ブラウザで動作確認をする。

- URL に「http://<IP アドレス>:5000/service?sw=ON」と入力して「SW STATUS:ON」と表示されること。
- URL に「http://<IP アドレス>:5000/service」と入力して「SW STATUS:OFF」と表示されること。

また、curl コマンドを使って通信し

```
curl -i http://<IP アドレス>:5000/service?sw=ON
```

ステータスコードが 200 で、HTTP ボディ部に「SW STATUS:ON」と表示されれば正しく動作しています。

3．6 HTML のテンプレートを使って WebUI をつくる

右のような HTML の画面から操作するように修正します。

まずは、テンプレートディレクトリとして
プログラムファイルの置いてある場所に「templates」というディレクトリを作成します。
そのディレクトリの中に次の内容の index.html というファイルを作成します。

```
<html>
  <head>
    <title>SW CONTROL</title>
  </head>
  <body>
    <h1>SW CONTROL</h1>
    {% if status %}
      <h2>SW STATUS:{{status}}</h2>
    {% endif %}
    <a href="/service?sw=ON">
      <input type="button" value="SW ON"></a>
    <a href="/service?sw=OFF">
      <input type="button" value="SW OFF"></a>
  </body>
</html>
```

次にプログラムを以下のように修正します。

HTML のテンプレートファイルを使用するために flask モジュールの render_template を import する

```
from flask import Flask,request,render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template("index.html")

@app.route('/service',methods=['GET'])
def led():
    status = 'OFF'
    if request.args.get('sw') == 'ON':
        status = 'ON'
    return render_template("index.html", status=status)

if __name__ == '__main__':
    app.run(host="0.0.0.0")
```

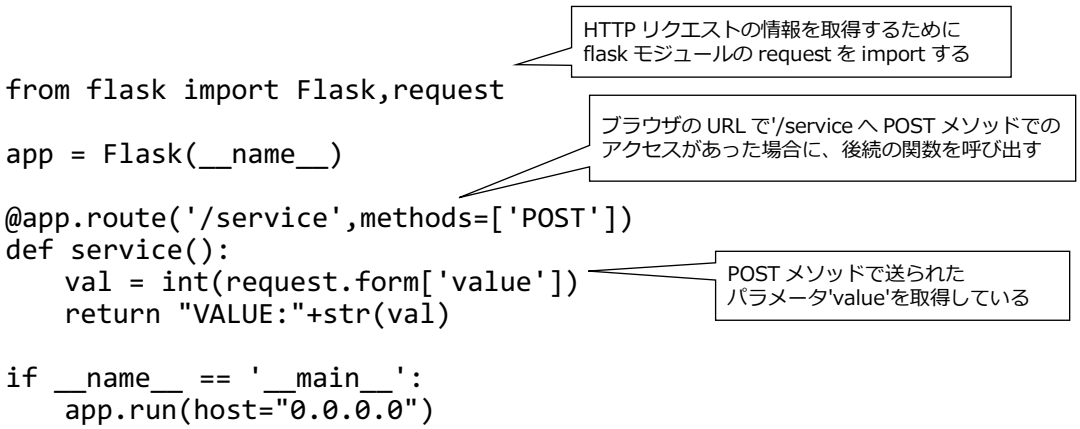
テンプレートファイルとして index.html を指定する

テンプレートファイルとして index.html を指定し、status の値を渡す

3．7 POST メソッドでデータを受け取る

@app.route で method を POST メソッドの受け取りを指定します。

POST メソッドでデータを受け取るには flask モジュールの request をインポートして、request.form[]でパラメータをキーで指定して受け取ります。




```
from flask import Flask,request

app = Flask(__name__)

@app.route('/service',methods=['POST'])
def service():
    val = int(request.form['value'])
    return "VALUE:"+str(val)

if __name__ == '__main__':
    app.run(host="0.0.0.0")
```

3．8 POST メソッドでデータを受け取る動作確認

前の動作確認の要領でプログラムを起動し、curl コマンドで動作確認をする。

```
curl -X POST -i http://<IP>:5000/service -d "value=1000"
```

ステータスコード 200 でボディ部に「VALUE:1000」と表示されることを確認する。

以下のように GET メソッドでアクセスした場合にはステータスコード 405 が返ってくることを確認しておいてください。

```
curl -i http:// <IP アドレス>:5000/service?value=1000
```

3．9 JSON データの操作

JSON とは、JavaScript から派生した、データ交換を行うためのデータ記述形式の一種です。2006 年 7 月に RFC 4627 として策定されました。

JSON は、JavaScript のサブセットでありデータの受信が軽快であるという特徴を持ちます。

また、JSON は特定のプログラミング言語に依存しない独立したテキスト形式であるため、JavaScript だけでなく C や C++、C#、Java、Perl、Python、といったプログラミング言語を使用するプログラマーにとって JSON は扱いやすい形式であるといわれています。

Python には JSON を扱うための json という標準モジュールが提供されています。

json モジュールを使うと Python のリスト型データや辞書型と JSON データを相互に変換することができます。

Python の変数を JSON 形式の文字列に変換するには json.dumps()関数を使用します。

```
import json

x = {'name':'yamada','tel':['052-000-0000',
      '090-1111-2222','070-3333-4444'],
      'fax':'052-555-6666','address':'JAPAN'}
jstr = json.dumps(x)
print(jstr)
```

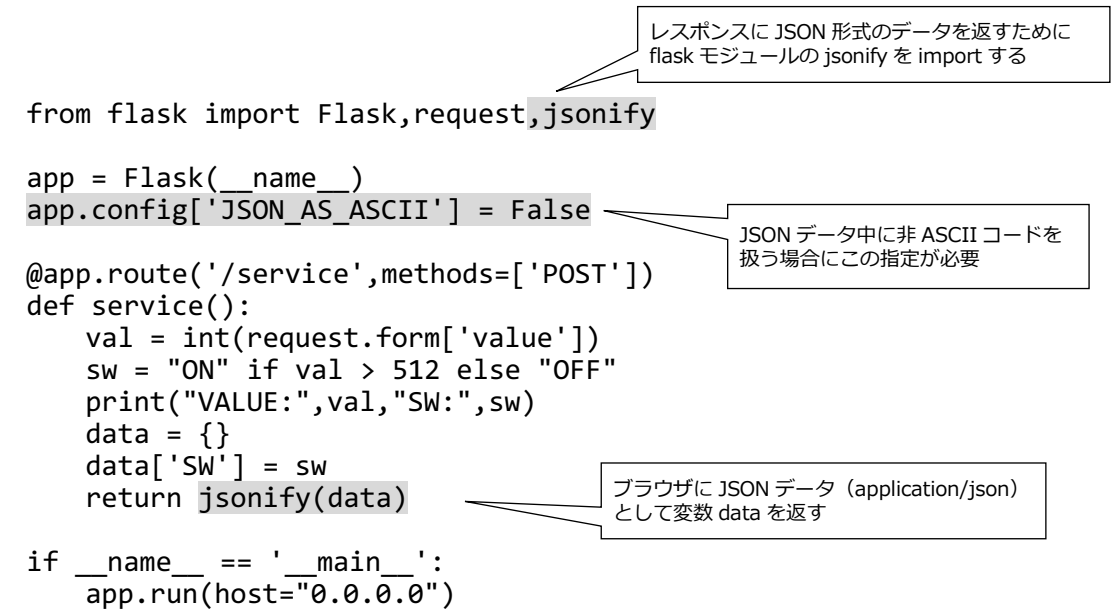
JSON 形式の文字列を Python の変数に変換するには json.loads()関数を使用します。

```
import json

jstr = ' {"name": "yamada", "tel": ["052-000-0000", "090-1111-2222", "070-
3333-4444"], "fax": "052-555-6666", "address": "JAPAN"}'

data = json.loads(jstr)
print(data['name'])
```

３．１０ REST サービス（Web サービス）の作成

Python の Flask モジュールを使って REST サービス環境を構築します。

サンプルプログラムを以下のように修正します。
修正内容は/service にアクセスしたときに GET メソッドと POST メソッドで情報を受け取れるようにし、そのレスポンスで JSON 形式のデータを送り返しています。




```
from flask import Flask,request,jsonify

app = Flask(__name__)
app.config['JSON_AS_ASCII'] = False

@app.route('/service',methods=['POST'])
def service():
    val = int(request.form['value'])
    sw = "ON" if val > 512 else "OFF"
    print("VALUE:",val,"SW:",sw)
    data = {}
    data['SW'] = sw
    return jsonify(data)

if __name__ == '__main__':
    app.run(host="0.0.0.0")
```

３．１１ REST サービスプログラムの実行と動作確認

前の動作確認の要領でプログラムを起動し、curl コマンドで動作確認をする。

```
curl -X POST -i http://<IP>:5000/service -d "value=1000"
```

JSON 形式のデータが返ってくれば POST メソッドの処理は正しく動作しています。

３．１２ Web サービスを利用する

```
from urllib.request import urlopen
from urllib.parse import urlencode
import json

server = "192.168.1.1"

while True:
    try:
        value = input('Input Value:')
        params = {'value':int(value) }
        enc_params = urlencode(params).encode("utf-8")
        url="http://"+server+":5000/service"
        with urlopen(url,enc_params) as res:
            data = json.loads(res.read().decode("utf-8"))
        print("SW:" + data['SW'])
    except:
        break
```

Appendix. TensorFlow によるディープラーニング

A. 1 TensorFlow とは

TensorFlow（テンソルフロー）は、Google が 2015 年にオープンソースとして公開した大規模な数値計算を行うライブラリです。機械学習や深層学習だけでなく、さまざまな数値演算ができる汎用的なライブラリです。

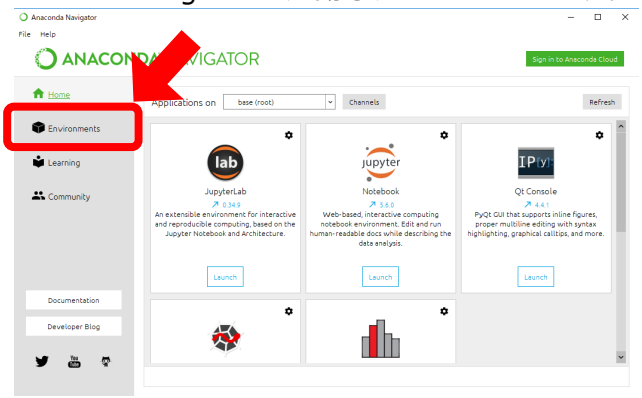
Tensor（テンソル）というのは、多次元行列計算のことです。

ライセンスは商用利用可能のオープンソース（Apache2.0）となっており、企業や個人、研究機関を問わず自由に利用できます。

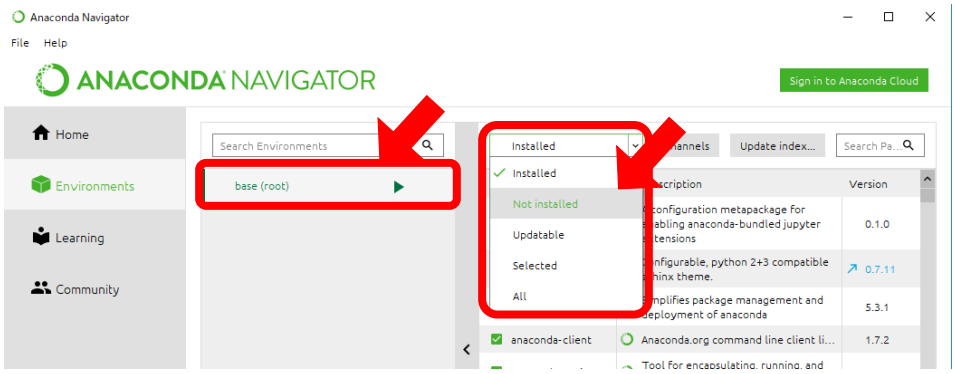
機械学習のライブラリとして人気があり、資料も充実しているのが特徴です。TensorFlow 自体は C++のライブラリで作られていますが、Python からできるだけオーバーヘッドなく実行できる仕組みとなっています。

A. 2 TensorFlow インストール

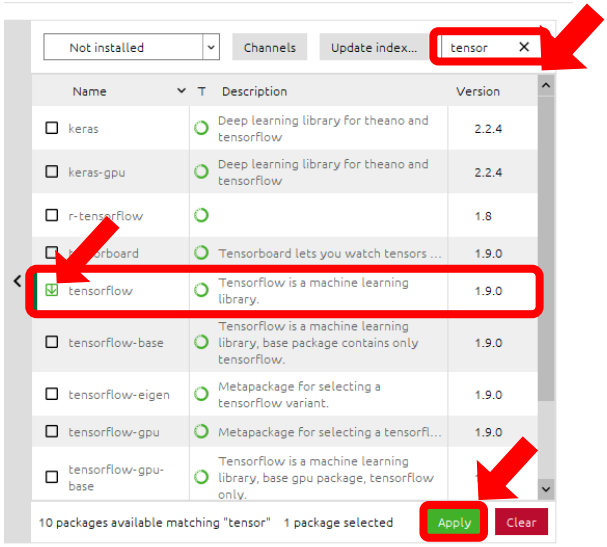
Anaconda Navigator を起動し、Environment タブをクリックします。



TensorFlow を導入したい仮想環境を選択し（本研修では base に）
メニューから[Not installed]を選択し



右上の[Search Packages]のところに「tensor」と入力し、パッケージを絞り込みます。
リストの中から[tensorflow]を探してチェックを入れ、[Apply]ボタンをクリックします。



ネットワーク環境によっては時間がかかります。

A． 3 TensorFlow の基本

TensorFlow は高速に処理できるように分散性、並列・分散実行が効率よく行えるデータフローグラフという独特の概念で処理を記述します。

まずは、データフローグラフについて理解するために TensorFlow に簡単な計算をさせてみましょう。以下は、TensorFlow を利用して簡単な足し算を行うプログラムです。

```
import tensorflow as tf
a = tf.constant(1234)
b = tf.constant(5000)
add_op = a + b

sess = tf.Session()
res = sess.run(add_op)
print(res)
```

このプログラムは最初の行で TensorFlow モジュール読み込みます。

次の行で TensorFlow 内に定数を宣言しています。tf.constant によって定数 a には 1234、定数 b には 5000 という値を宣言しています。

【書式】tensorflow.constant(value, dtype=None, shape=None, name='Const, verify_shape=False)

パラメタ名	概要
value	定数値または定数値のリストを指定します。定数値のリストの場合は shape でサイズを指定します。
dtype	型を指定します。指定されていない場合は value の値から推測された型がセットされます。
shape	オプション。データのサイズを指定します。
name	定数を内部的に識別するための名前。
verify_shape	value を検証可能にするかどうか

続く行で足し算を行う計算を定義します。

重要な点として、この時点で変数 add_op に代入されているのは、データフローグラフと呼ばれるオブジェクトです。

そして、その後に TensorFlow で計算を実行するための tf.Session()によってセッションを開始します。

【書式】tensorflow.Session(target='', graph=None, config=None)

パラメタ名	概要
target	オプション。接続先の実行エンジンを指定します。
graph	オプション。使用するデータフローグラフを指定する場合に使用します。
config	オプション。セッションの構成オプションを指定します。

セッションを開始したら、データフローグラフを run()メソッドに与えます。
ここではじめて計算が行われます。

【書式】Session.run(fetches, feed_dict=None, options=None, run_metadata=None)

パラメタ名	概要
fetches	計算グラフの op ノード（Tensor オブジェクト）を指定します。
feed_dict	型を指定します。指定されていない場合は feches から推測された型がセットされます。
options	オプション。このセッションを使って特定のステップの振る舞いを制御できます。
run_metadata	オプション。非テンソルからの出力を収集します。

簡単なプログラムですが、最初に計算処理をコンパイルして「グラフ」と呼ばれるオブジェクトを構築しておいて、コンパイル後にグラフを実行するという、TensorFlow の流れを把握してください。

TensorFlow のプレースホルダ

プレースホルダの機能を紹介します。これは、テンプレートに値をあてはめるための仕組みです。

TensorFlow ではデータフローグラフを作成しておいて、それを実行するという仕組みでした。
プレースホルダはデーターフローグラフの作成時には値を入れず容れ物だけ用意しておいて、セッションを実行する段階で容れ物の部分に値を差し込んで実行するという仕組みです。

実際の利用例をみてみましょう。
次のプログラムはリスト a に指定した各要素を 2 倍にして表示するというプログラムです。

```
import tensorflow as tf
a = tf.placeholder(tf.int32,[3])
b = tf.constant(2)
x2_op = a * b

sess = tf.Session()
r1 = sess.run(x2_op,feed_dict={a:[1,2,3]})
print(r1)
r2 = sess.run(x2_op,feed_dict= {a:[10,20,30]})
print(r2)
```

tf.placeholder によってプレースホルダの使用を宣言します。
ここでは tf.int32、つまり 32 ビットの整数型で、要素が 3 つある配列を宣言します。

【書式】tensorflow.placeholder(dtype, shape=None, name=None)

パラメタ名	概要
dtype	要素の型
shape	オプション。要素の形状を指定します。
name	オプション。オペレーションの名前。

「x2_op = a * b」の部分では定数 2 とプレースホルダ a を掛け算する演算を宣言します。
セッションを開始して、その後の「sess.run」の部分でプレースホルダに Python のリストを与えます。これにより、a の部分に[1,2,3]だったり[10,20,30]といった値が埋め込まれて実行されます。

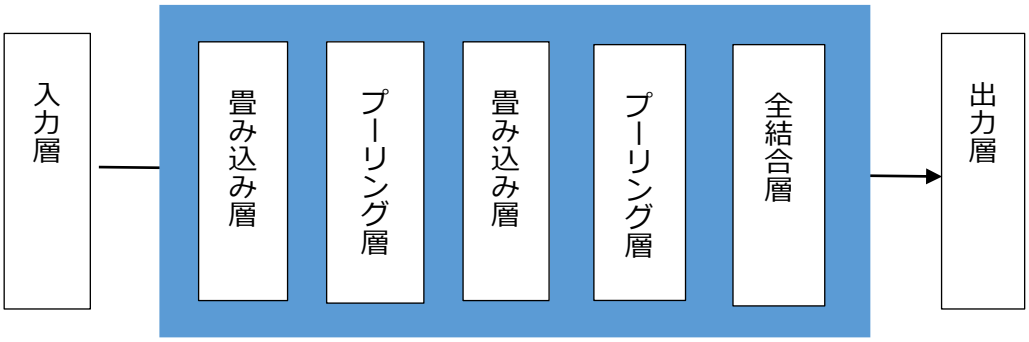
上記のプロブラムではプレースホルダの宣言時に、整数型で要素が 3 個の配列を指定しました。しかし、要素が固定では不便なこともあります。
そこで、要素数を指定する部分に None を指定することで、任意のサイズを指定できるようになります。

```
import tensorflow as tf
a = tf.placeholder(tf.int32,[None])
b = tf.constant(2)
x2_op = a*b
sess = tf.Session()

r1 = sess.run(x2_op,feed_dict={a:[1,2,3,4,5]})
print(r1)
r2 = sess.run(x2_op,feed_dict= {a:[10,20]})
print(r2)
```

A．４ TensorFlow で深層学習

深層学習のしくみ

深層学習は、ニューラルネットワークにいくつかの層を重ねたものです。
ここでは「畳み込みニューラルネットワーク」を利用して画像処理を行います。
畳み込みニューラルネットワーク（CNN:Convolutional Neural Network）とは、入力層と出力層の間の中間層とプーリング層を用意するものです。



画像認識では、よく画像をぼかしたりエッジを強調したりするのですが、これと同じような操作をニューラルネットワークで行います。
畳み込みとプーリングの処理では、解像度を粗くするリサンプリングするような処理を繰り返し行います。

畳み込み層は、画像の局所的な特徴抽出を行う層です。
プーリング層は、畳み込み層で得た特徴マップを縮小する層です。局所的な特徴を維持しつつ、縮小を行うことにより、位置変更に対する結果の変化を抑えることができます。
全結合層は、畳み込み層やプーリング層の結果である二次元の特徴マップを一次元に展開します。

深層学習の実践

それでは実際のプログラムで確認してみましょう。ここでは手書き数字の MNIST の画像を 0 から 9 に分類するプログラムを試してみます。

TensorFlow のライブラリで、MNIST の画像をダウンロードしデータを抽出するには、以下のコードを記述して、実行すると mnist というディレクトリにデータベースをダウンロードします。

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("mnist/",one_hot=True)
```

ここで取得した mnist オブジェクトがどのような構造になっているのか、簡単に確認しておきましょう。

```
mnist.train.images
mnist.train.labels
```

このように mnist.train に訓練データが、mnist.test にテストデータが入っています。それぞれのオブジェクトは画像データの images と、ラベルデータの labels というプロパティを持っています。

深層学習のプログラム

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

# MINST の手書き画像データを読み込む
mnist = input_data.read_data_sets("mnist/", one_hot=True)

pixels = 28 * 28 # 28x28 ピクセル
nums = 10 # 0-9 の 10 クラスに分ける

# プレースホルダを定義
x  = tf.placeholder(tf.float32, shape=(None, pixels), name="x") # 画像データ
y_ = tf.placeholder(tf.float32, shape=(None, nums), name="y_")  # 正解ラベル

# 重みとバイアスを初期化する関数
def weight_variable(name, shape):
    W_init = tf.truncated_normal(shape, stddev=0.1)
    W = tf.Variable(W_init, name="W_"+name)
    return W
def bias_variable(name, size):
    b_init = tf.constant(0.1, shape=[size])
    b = tf.Variable(b_init, name="b_"+name)
    return b

# 畳み込みを行う関数
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1,1,1,1], padding='SAME')
# 最大プーリングを行う関数
def max_pool(x):
    return tf.nn.max_pool(x, ksize=[1,2,2,1],
        strides=[1,2,2,1], padding='SAME')

# 畳み込み層 1
with tf.name_scope('conv1') as scope:
    W_conv1 = weight_variable('conv1', [5, 5, 1, 32])
    b_conv1 = bias_variable('conv1', 32)
    x_image = tf.reshape(x, [-1, 28, 28, 1])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

# プーリング層 1
with tf.name_scope('pool1') as scope:
    h_pool1 = max_pool(h_conv1)

# 畳み込み層 2
with tf.name_scope('conv2') as scope:
    W_conv2 = weight_variable('conv2', [5, 5, 32, 64])
    b_conv2 = bias_variable('conv2', 64)
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
```

```
# プーリング層 2
with tf.name_scope('pool2') as scope:
    h_pool2 = max_pool(h_conv2)

# 全結合レイヤー
with tf.name_scope('fully_connected') as scope:
    n = 7 * 7 * 64
    W_fc = weight_variable('fc', [n, 1024])
    b_fc = bias_variable('fc', 1024)
    h_pool2_flat = tf.reshape(h_pool2, [-1, n])
    h_fc = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc) + b_fc)

# ドロップアウト(過剰適合)を排除
with tf.name_scope('dropout') as scope:
    keep_prob = tf.placeholder(tf.float32)
    h_fc_drop = tf.nn.dropout(h_fc, keep_prob)

# 読み出し層
with tf.name_scope('readout') as scope:
    W_fc2 = weight_variable('fc2', [1024, 10])
    b_fc2 = bias_variable('fc2', 10)
    y_conv = tf.nn.softmax(tf.matmul(h_fc_drop, W_fc2) + b_fc2)

# モデルの学習
with tf.name_scope('loss') as scope:
    cross_entoropy = -tf.reduce_sum(y_ * tf.log(y_conv))
with tf.name_scope('training') as scope:
    optimizer = tf.train.AdamOptimizer(1e-4)
    train_step = optimizer.minimize(cross_entoropy)

# モデルの評価
with tf.name_scope('predict') as scope:
    predict_step = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
    accuracy_step = tf.reduce_mean(tf.cast(predict_step, tf.float32))

# feed_dict の設定
def set_feed(images, labels, prob):
    return {x: images, y_: labels, keep_prob: prob}

# セッションを開始
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # TensorBoard への書き込み準備
    tw = tf.summary.FileWriter('log_dir', graph=sess.graph)
    # テスト用のフィードを生成
    test_fd = set_feed(mnist.test.images, mnist.test.labels, 1)
    # 訓練を開始
    for step in range(10000):
        batch = mnist.train.next_batch(50)
        fd = set_feed(batch[0], batch[1], 0.5)
        _, loss = sess.run([train_step, cross_entoropy], feed_dict=fd)
        if step % 100 == 0:
            acc = sess.run(accuracy_step, feed_dict=test_fd)
            print("step=", step, "loss=", loss, "acc=", acc)
    # 最終結果を表示
    acc = sess.run(accuracy_step, feed_dict=test_fd)
    print("正解率=", acc)
```