Pythonの歩き方

リリース ***0.1.2***

Noboru Yamamoto [1]**,**
J-PARC/KEK, Tsukuba, Japapn

2021 年 08 月 07 日

[1]mail-to:noboru.yamamoto at kek.jp

目 次

第 1 章 Python の歩き方 (Beginner’s guidebook on Python3) 3

1.1 旅の準備 4
1.2 Python のインストール確認 9
1.3 最新情報を見逃すな 11
1.4 ライブラリ／モジュールの入手方法 13
1.5 開発環境: 18
1.6 仮想環境：venv 21
1.7 バージョン管理 24
1.8 その他 25

第 2 章 Python プログラムの実例 29

2.1 数字の出現数を数える 30
2.2 今年の「13 日の金曜日」は何回？ 37
2.3 From Web to Plot 48
2.4 日本の休日 62
2.5 jupyter-execute の見本 68

第 3 章 Python にポインターはないの？ 71

第 4 章 章題名のアイデア 73

4.1 最新情報を見逃すな 74
4.2 繰り返しを任せよう。 75
4.3 小さくまとめて、何度も使う (def) 76
4.4 (Error 処理： try-except-else-finally) 77
4.5 しまっておいたり、取り出したり (File 入出力） 78
4.6 （Web からデータを入手する。:Beautiful Soup) 79
4.7 （数式処理： sympy, SageMath) 80
4.8 (jupyter lab) 81
4.9 (matplotlib.pyplot) 82
4.10 (matplotlib.splot) 83
4.11 (matplotlib.animation) 84
4.12 (read excel file:xlrd, openpyxl) 85
4.13 (write excel file:xlwt, openpyxl) 86

ProjectInfo Python の歩き方, version 0.1.2 by Noboru Yamamoto

Author Noboru Yamamoto

Version 0.1.2

Copyright 2021-, Noboru Yamamoto(KEK),

第1章　Pythonの歩き方 (Beginner’s guidebook on Python3)

1.1 旅の準備

これから Python を学ぶ旅に出る前に、 Python の概要を掴んでおきましょう。初めて聴く言葉もあるかと思いますが、それらの詳しい意味は徐々に説明していきます。

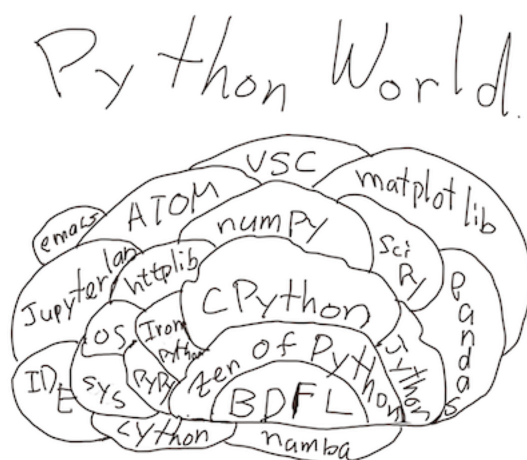

図-1.1: Python Map Image

1.1.1 python とは

まずは目的地である Python の概要を説明します。

Python はプログラミング言語の規格であり、またそのインタプリター形の*実行系の名前でもあります。(この実行系は `CPython` と呼ばれることもあります。)

2021 年現在において、Python はさまざまな局面で使われています。何よりも、Python はあなたが電子計算機を使う上でとても有用なツールになるでしょう。

ここでは、Python をあなたの PC で使い始めるに当たって、ちょっと知っておくと役に立つであろう事を紹介しておきます。ここで紹介した事を全てマスタする必要はありませんが、Python プログラム開発を始めた時、思い出していただければ役にたつこともあるのではないかと期待しています。

BDFL：Benevolent Dictator For Life 慈悲深き終身の独裁官, Python の著者 Guido van Rossum のこと。彼はオランダ出身。

1.1.2 Python の歴史

図-1.2: Python_Logo_Image

---

1. Python は Guido van Rossum(BDFL) 氏がクリスマス休暇のプロジェクトとして作り出した。
2. Python ロゴの蛇は O’Reilly の表紙から。
3. Python の名前は TV 番組のタイトルから

Mark Lutz 氏の「Programming Python」（第一版）の序文によると、 Python は Guido van Rossum 氏が 1989 年のクリスマス休暇中の「趣味」のプログラミングプロジェクトとして始めたプログラミング言語です。　van Rossum 氏は大ファンであった「Monty Python’s Flying Circus」に因んで、この言語を「深く意味も考えずに」　Python と名付けたとのことです。つまり、Python の始まりは蛇の Python とは (直接には）関係が無かったということです。

Python Logo

このことから、Python のアイコンとして当初は、
なども使われていました。 しかし、Mark Lutz 氏の「Programming Python」が蛇本と呼ばれることがあるように、大蛇 (Python) がその表紙に描かれていたことから (？)、Python のアイ

コンは蛇のアイコンへと変化していきました。

現在の Python の公式ロゴは　です。

改めて考えると、このロゴは Python2 の時代に定められました。Python3 が主流になった今日、ロゴも三匹の蛇 (Python) に変わるのかもしれません。

蛇足ですが、迷惑メールを spam メールと呼ぶのは、「Monty Python’s Flying Circus」の一つのエピソードにちなんでいるということは良く知られています。

PEP (Python Enhancement Proposal)

Python 本体の開発は、PEP（Python Enhancement Proposal）を提出して新しい機能を提案することから始まります。試験実装と議論を通じて PEP は改定されてゆき、合意が得られたところで、公式な言語仕様として採用されます。インターネットの仕様における RFC(Request For Comments) に似た仕組みです。

これまでに提案された PEP は、PEP　インデックス (https://www.python.org/dev/peps/)　で知ることができます。

1.1.3 Python の地図

モジュール (Python のライブラリ）

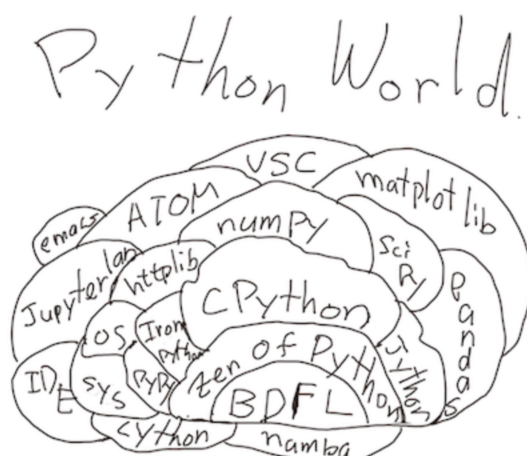

図-1.3: Python Map Image

Python の特徴のひとつは豊富なライブラリ（モジュール）です。

Python 自体は汎用のプログラム言語ですので、電子計算機で実現できることは、Python 言語を使ってできるといってもいいのですが、豊富に用意されたライブラリ／アプリケーションを利用することで、目的を手早く、簡単に実現できるでしょう。

Python の配布パッケージには　*Python* **標準ライブラリ** (Python ドキュメント ページ (https://docs.python.org/ja/3/library/index.html) を参照）が含まれています。 Python 標準ライブラリには、

1. Python がもつ基本的なデータ型を効率よく利用するためのライブラリ（`string`,`re`,`datetime`,…)、
2. 数学関数 (`math`, `statistics`)、

3. OS のもつ様々な機能（os)、
4. GUI 作成用ライブラリ (tkinter)

などが用意されています。

標準ライブラリ以外にもサードパーティから豊富なライブラリが提供されています。これらの拡張ライブラリは PyPI (https://pypi.org/) あるいは GitHub (https://github.com/) などで公開されています。PyPI(Python Package Index) は python ライブラリを配布する de facto スタンダードとなっており、pip コマンド あるいは pip モジュールを使うことで、簡単にモジュールを入手し、インストールすることができます。

よく使われるサードパーティモジュールには、matplotlib, scipy, numpy,pandas などがあります。 J-PARC の制御システムでは、ca,ROOT, cVXI などのモジュールが使われています。 (matplotlib をベースにした seaborn など世の中には様々なモジュールが存在しています。全てをカバーすることは難しいこと、また長期にわたってそれらのモジュールが利用できるかどうかは不明であることなどから、ここでは評価がある程度確立したモジュールだけをご紹介します。)

Python のモジュールはそれ自体だけが Python で書かれたモジュールと、C/C++などのコンパイル言語を使って書かれたモジュールが存在します。 後者のモジュールは、python のプログラムで実行速度が特に問題になるような場合に、実行速度を改善するために使われることがあります。 Cython, f2py, numba などのツールを使うことで、python モジュールを高速化する手法も存在します。

OS などが提供する C/C++などで開発されたライブラリを python から呼び出す方法も存在します (ctypes)。この手法は強力ではありますが、OS 間の移植性が失われますし、最悪の場合システムのシャットダウンを引き起こす可能性すらあります。これらの手法は**最後の武器** と考えておくのが良いでしょう。

いろいろな Python 処理系 (実行系）

多くの場合、python.org が配布している Python 処理系 (CPython) が使われますが、目的に応じてそれ以外の Python 処理系 (https://www.python.org/download/alternatives/) が有用である場合があります。

Python の処理系としては、次のようなものがよく知られています。

1. CPython： 公式の Python 処理系. C/C++による実装
2. Jython: java による実装。 Java object が python object に自動的にマップされます。同じ作者による、.Net 向けの Iron Python もあります。
3. micropython: マイクロコントローラ向けの小型の実装
4. pypy: RPython(制限された python,Python のサブセット）で作られた python 実装系。JIT(Just in Time) コンパイラを内蔵している。処理系自体は RPython から C/C++な

どに変換された後、処理系が作成される。

5. Brython: (https://brython.info/index.html) Javascript による Python の実装。web ページに Python プログラムを埋め込んで、実行できる。

1.2 Python のインストール確認

Python プログラムを実行するためには、Python プログラムを実行するための処理系 (アプリケーション) が必要です。幸いなことには、お使いの計算機には既に python3 がインストールされているかもしれません。　確かめてみましょう

コマンドラインツールで

```
python3 --version
```

あるいは

```
python --version

py --version
```

のいずれかを実行してみましょう。　Linux/macOS などのシステムでは前者 2 つのどちらか、Windows の環境では最後のコマンド (py --version) が使えるかもしれません。
以下に実行例を示します。

```
%%sh
python --version # on most of Unixen including macOS
#py --version # windows only
```

```
Python 2.7.16
```

```
%%bash
python3 --version
```

```
Python 3.8.2
```

```
import sys
print (sys.version)
```

```
3.9.6 (v3.9.6:db3ff76da1, Jun 28 2021, 11:49:53)
[Clang 6.0 (clang-600.0.57)]
```

これらのコマンドを実行した結果が

```
Python 3.9.6
```

であれば python3 のバージョン 3.9.6 がインストールされています。(2021.7.21 現在の python3 最新版は、3.9.6 です。)

注) 実行例の欄にある%%bash は jupyterlab でこの入力を bash で実行することを指示する、directive です。

1.2.1 python がインストールされていなかったら？

不幸にして、

- これらのコマンドが全く動作しない、
- あるいはインストールされた python がまだ python2 に止まっている

と言った場合には、Python　ダウンロードサイト (https://www.python.org/downloads) からご使用の環境にあったインストーラーをダウンロードして、python3 をインストールしましょう。

1.3 最新情報を見逃すな

1.3.1 Python ドキュメント

公式の Python3 ドキュメント (日本語訳）は Python ドキュメント　サイト (https://docs.python.org/ja/3/)　から閲覧可能です。　チュートリアルも用意されています。最新版以外のドキュメントも選択して表示させることができます。

1.3.2 PyPI サイト

問題を解決するのに必要なライブラリが見つからない時は、 PyPI Site (https://pypi.org/) を探してみましょう。

(`pip` コマンドの `search` サブコマンドはこの文書執筆時の状況では**使えない**状態です。　PyPI ホームページで検索しましょう。）

1.3.3 The Hitchhiker’s Guide to Python!

Python ドキュメントには公式の チュートリアルも掲載されています。此他にも色々な Python 関連の資料が Internet から入手可能です。

この文書とよく似た名前の、The Hitchhiker’s Guide to Python! (https://docs.python-guide.org/) は分かりやすい online の Python 入門書です。　github (https://github.com/realpython/python-guide) で最新版が提供されています。

日本語版 (https://python-guideja.readthedocs.io/ja/latest/) もありますが、残念なことに最新版への追随が遅れている様です。

この文書を書き始めた後に、このサイトを **再発見** しました。以前にサイトを訪れたことはあったものの、すっかりその存在を忘れていました。　とはいえ、どこかで影響を受けているかもしれません。

1.3.4 `help` 関数

python インタプリタには `help` 関数が用意されています。`help` の引数には関数名、クラス名、モジュール名などを与えることができます。関数を定義する際に `help` のための文字列も定義しておくことができます。

```
def foo(arg:int):
    """
    docstring と呼ばれる関数定義のこの部分が
    ヘルプメッセージとして使われます。
    """
    return arg+2

help(foo)
foo(100)
```

```
Help on function foo in module __main__:

foo(arg: int)
    docstring と呼ばれる関数定義のこの部分が
    ヘルプメッセージとして使われます。
```

```
102
```

1.4 ライブラリ／モジュールの入手方法

python の強みの一つは公開されている豊富なモジュール（ライブラリ）です。あなたの問題を解決するのに役立つモジュールをまず探してみましょう。

PyPI:The Python Package Index (https://pypi.org/) は Pyton のライブラリを探し始めるのに一番役に立つでしょう。 もちろん、`Google Seach` も頼りになるツールです。

1.4.1 pip/PyPI

PIP : PyPI = yum(dnf) : RedHat/CentOS repository = apt : Ubuntu

---

pip と PyPI の関係は yum/dnf と RedHat/CentOS などのリポジトリとのに関係に似ています。

PyPI サイト (https://pypi.org/) に登録されたさまざまな Python のライブラリを pip コマンドを使って手元の環境にインストールすることができます。

pip の動作確認

pip は Python 3.4 以降には、標準で付属しています。 pip が見つからなければ、python3 のバージョンを 3.4 以上に更新した方が良いでしょう。

`pip` がインストールされているかどうかを確認するために、コマンドラインで、

```
python3 -m pip --version
```

を実行してみましょう。

```
pip 21.0.1 from /Library/Frameworks/Python.framework/Versions/3.9/lib/
↪python3.9/site-packages/pip (python 3.9)
```

などと端末に出力されれば、`pip` はインストール済みです。

```
%%bash
python3 -m pip --version
```

```
pip 21.1.3 from /Users/noboru/Library/Python/3.8/lib/python/site-
↪packages/pip (python 3.8)
```

python3 -m pip <subcommand> <parameters>

以前は、コマンド pip3 が使われていましたが、次の例の様にこの形式は将来的に使えなくなります。python3 -m pip の形式を使いましょう。

```
%%bash
pip3 --version
```

```
pip 21.1.3 from /Users/noboru/Library/Python/3.8/lib/python/site-
↪packages/pip (python 3.8)
```

```
WARNING: pip is being invoked by an old script wrapper. This will fail␣
↪in a future version of pip.
Please see https://github.com/pypa/pip/issues/5599 for advice on␣
↪fixing the underlying issue.
To avoid this problem you can invoke Python with '-m pip' instead of␣
↪running pip directly.
```

pyhton3 -m <module name>は python モジュールをメインプログラム (スクリプト) として実行することを指示します。

pip のインストール

不幸にして pip がインストールされていない場合は、上に述べたように python3.4 以降の python3 をインストールすることを検討しましょう。

どうしても python3.4 以降の python3 をインストールできない場合には、

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

から get-pip.py プログラムをダウンロードした上、

```
python3 get-pip.py
```

あるいは windows では、

```
py  get-pip.py
```

を実行します。 pip のインストールの詳細は pip のインストール (https://pip.pypa.io/en/stable/installation/) に詳しく紹介されています。

pip 本体の更新

pip は python3 の標準配布の中に含まれていますが、python3 本体の更新とは独立にバージョンアップされます。必要に応じて

```
python3 -m pip install -U pip
```

を実行して、インストールされている pip のバージョンを更新しておきましょう。

pip の使い方 （-h/--help)

```
python3 -m pip --help
```

サブコマンドの詳細は、サブコマンド (以下の例では install) の後に-h または--help オプションを指定します。

```
python3 -m pip install --help
```

モジュールのインストール （install サブコマンド)

```
python3 -m pip install <module 名>
```

search サブコマンドは現状使えないので、PyPI の web ページでモジュールを検索しましょう。

モジュールの更新 (-U/--upgrade)

```
python3 -m pip install -U <module 名>
```

ユーザースペースへのインストール (--user オプション)

```
python3 -m pip install --user  <module 名>
```

M1-mac での注意点

---

x86-64 向けしかないバイナリライブラリを含むモジュール (例えば matplotlib) を使う際には、

```
arch -x86_64 python3
```

を使う必要があります。

M1-mac と呼ばれる arm ベースの CPU を持つ macitosh では、バイナリファイルとして arm64 向け、x86_64 向け、それらふたつのバイナリを一つのファイルとしてもつ fat バイナリの三つの形式があります。

残念ながらこのノートの執筆時点 (2021/7/2 現在) では、pip を経由して macos にインストール可能なパッケージでありながら、x86_64 向けのバイナリファイルだけが提供されているものがあります。　例えば、よく使われる matplotlib　がその一つです。このような場合、pip でパッケージを install するばあいに、arch コマンドを用いて、

```
arch -x86_64 python3 -m pip install -U matplotlib
```

などとして、インストールすることができます。パッケージが x86_64 向けのバイナリしか持っていませんので、利用する場合にも、

```
arch -x86_64 python3
```

などとして、python3 本体も x86_64 バイナリを使って起動することが必要です。　いずれ M1 mac 向けの配布パッケージがバイナリを含む場合には、Fat binary 形式が配布されるようになると思われますが、それまで、少しの間このトリックが必要になります。

（Python.org で配布されている macosx 向けの python3 本体の配布も実験的に Fatbinary の Installer が提供されています。あと少しの我慢でしょう。）

Note: matplotlib 本体ではなく、その中に含まれる scipy モジュールが問題となります。scipy をソースコードから arm64 向け（あるいは Fatbinary 向けに）にコンパイルするなどの方法でインストールすれば、 arch コマンド無しでも利用可能になります。（吉本さん、情報提供有難うございました。）

　macos で Python で SSL を使う note: macos 上の python プログラムで SSL を使う際には次のおまじないが必要です。

```
import os, certifi
os.environ["SSL_CERT_FILE"]=certifi.where()
```

1.4.2 conda/anaconda

conda はデータサイエンス分野でよく使われるパッケージ配布用プログラムです。この conda をベースにデータサイエンスでよく使われさまざまなパッケージを管理／配布しているシステムが anaconda と呼ばれています。

anaconda では Excel の表や RDB のテーブルと親和性のある `dataframe` というデータ構造を使います。この `dataframe` は pandas モジュールをインポートすることで、conda/anaconda と離れて利用することも可能です。

尤も、これは順序が逆でデータ解析の Python モジュールをまとめたパッケージを, 蛇のアナコンダに因んで、anaconda と名づけたと思われます。その anaconda のパッケージプログラムを conda と名づけたということと類推します。

それではなぜアナコンダなのかといえば、Python 自体のアイコンに蛇が使われるように、python=大蛇ということで、大蛇の一種であるアナコンダにかけたということでしょう。　もっともプログラム言語の Python の名前は BBC で放送されたコメディ番組”Monty Python’s Flying Circus“から採ったと作者 Guido van Rossum が述べています。(Foreward for”Programming Python",Mark Lutz, 1996, O’Reilly & Associates, Inc. ）　[fn_python][]

[fn_python]: 関係ないですが、　Wikipedia（日本語版）の Python の解説に、KEKB control home page (http://www-acc.kek.jp/WWW-ACC-exp/KEKB/control/KEKB-Control-home.html)　が脚注で引用されている (リンク切れ）ことに気が付きました。(2021.4.21)

1.4.3 その他のパッケージ配布システム

Linux であれば、dnf(yum) や apt といったパッケージ配布プログラムで python 処理系や様々なモジュールをインストール／更新管理することができます。これらの配布パッケージに含まれる Python やそのライブラリのバージョンは、上記の pip や conda のリポジトリに比べやや古いことがあります。

一方で、利用中のシステムとの互換性が確認されているなど、運用上の安定性には優れています。

macos では homebrew や Fink, mac ports　もよく使われるパッケージ配布システムです。

これらのパッケージ配布システムを複数同時に使うと、ライブラリバージョンの衝突などで動作不良になることがあります。注意してお使いください。

1.5 開発環境:

Python プログラムは様々な方法で開発することができます。簡単なテキストエディタから、プログラムの作成から、保存／管理／テスト／リリースまでの全てのステップを助けてくれる統合プログラミング環境 (IDE:Integrated Development Environment) まで多数の選択肢が存在します。　ここでは、現在 Python プログラム開発環境としてよく知られている Jupyter(IPython, Jupyter Lab) を紹介します。

1.5.1 IPython/Jupyter/Jupyter lab

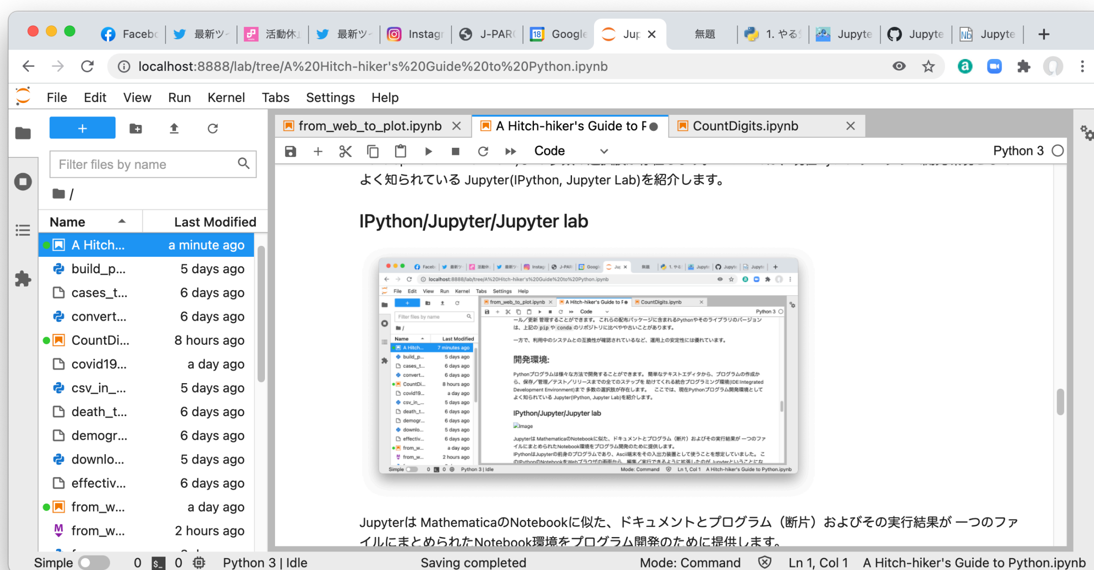

図-1.4: Jupyterlab screen Images

Notebook 形式で入力と出力 (結果）を保存しておくことができる。

Jupyter は Mathematica の Notebook に似た、ドキュメントとプログラム（断片）およびその実行結果が一つのファイルにまとめられた Notebook 環境をプログラム開発のために提供します。

IPython は Jupyter の前身のプログラムであり、Ascii 端末をその入出力装置として使うことを想定していました。この IPython の Notebook を Web ブラウザの画面から、編集／実行できるように拡張したのが Jupyter ということになります。 さらに、この web ブラウザ上の編集／実行環境を進化させ、次の世代の Jupyter になると考えられているのが、Jupyter-lab です。jupyterlab のドキュメントは https://jupyterlab.readthedocs.io/en/stable/ を探してみましょう。

Jupyter/ Jupyter-lab は `pip` を使って簡単にインストールすることができます。

pip install jupyter pip install jupyterlab

IPython/Jupyter/Jupyter-lab は python 言語専用の開発支援プログラムというわけではなく、C/C++など様々プログラミング言語の開発支援パッケージが存在しています。例えば、Open Source の数式処理システムである SageMath は、Jupyter を標準の開発環境としています。

Jupyter で利用可能なプログラム言語 (Kernel) は

List of Kernels (https://github.com/jupyter/jupyter/wiki/Jupyter-kernels)

のリストをご覧ください。

1.5.2 Python3 をサポートする エディター

python3 プログラムの開発にはどのようなテキストエディターでも使うことができます。しかしながら、 **emacs/ atom /VSC(Visual Stduio Code)** など多くのプログラム 開発用エディタは `python` プログラム開発をサポートする機能を有しています。 使い慣れたエディタに `python` サポート機能を追加することで、開発の効率も上がるでしょう（多分）。

1.5.3 IDLE: Python 標準配布に含まれる IDE

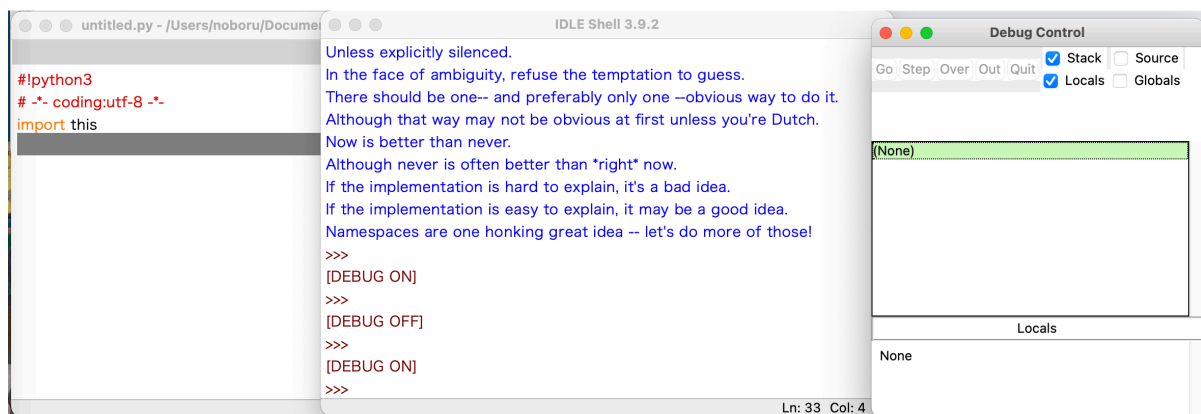

図-1.5: Image

IDLE は多くの（もし全てでなければ）Python 処理系の配布パッケージに含まれている、Python 専用のプログラム開発環境です。 IDLE 自体も Python を使って制作されています。IDE として期待される 基本的な機能はサポートされています。

Python 処理系がインストールされている環境であれば、ほとんどの場合利用可能という意味では、入門の説明にちょうど良いのですが、それほど使われているわけではなさそうです。Python と Tk でここまでできるというサンプルではあります。

1.5.4 emacs

EMACS editor

図-1.6: Image

GNU が配布しているテキストエディタ。著者の愛用です。python-mode/cython-mode などを実現する　拡張モジュール（Emacs Lisp のプログラム）が配布されており、これらのモジュールを導入することで、python プログラムの開発が効率化されます。

atom /VSC(Visual Stduio Code)

ATOM editor

atom/VSC はいずれも Javascript で開発された、近年流行りのエディターです。いずれも無料で入手できますが、atom は open source, VSC はマイクロソフトが開発提供しています。VSC には Python をサポートする機能拡張がマイクロソフトから提供されていますので、　これをインストールして利用します。

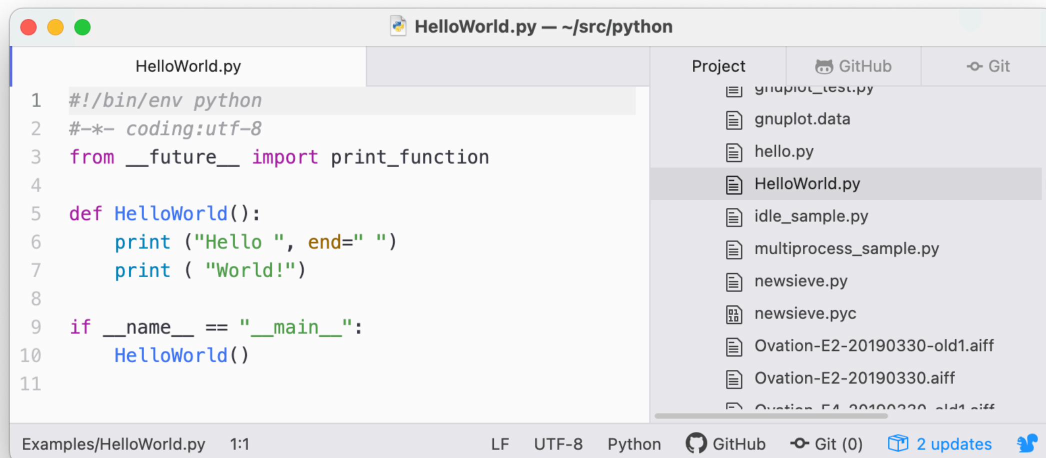

図-1.7: Image

1.6 仮想環境：venv

モジュールのコンフリクトを避けるには， 仮想環境を使いましょう。

色々なプログラムを同時に開発していると、インストールしている Python モジュールのコンフリクトが起きてしまうことがあります。また、新しい Python モジュールをテストしたいけれど、システムにはまだインストールしたくないということもあります。このような場合に役に立つのが venv などの Python 実行用仮想環境作成ツールです。venv は python3 の標準配布に含まれているので、これから始めるのが良いでしょう。

virtualenv は python2 の時代によく使われていたツールです。venv は virtualenv の機能のサブセットを python3 の標準機能として取り込んだものとされています。 フルセットの `virtualenv <https://virtualenv.pypa.io/en/latest/>`__ や `pyenv <https://github.com/pyenv/pyenv>`__ などの機能強化されたツールも 3rd パーティから登場しています。

1.6.1 venv の使い方

venv 環境の作成

python の独立した環境を作るために、venv を使ってみましょう。

まずこの環境をセットアップするディレクトリ（ここでは、/path/to/new/virtual/environment）とします。

```
python3 -m venv /path/to/new/virtual/environment
```

を実行することで、/path/to/new/virtual/environment の中に、独立した環境を設定するために必要なファイルを作成します。

```
% ls /path/to/new/virtual/environment
 bin         include     lib      pyvenv.cfg
```

pytenv.cfg には venv の設定が書き込まれています。

```
% cat pyvenv.cfg
home = /path/to/PythonBinary/3.9/bin
include-system-site-packages = false
version = 3.9.6
```

venv 環境に入る。

使用する Shell に応じた方法で venv 環境を `activate` します。

作成した venv 環境を activate することで、この独立した python 環境の利用が始まります。activate は venv の必要とする環境変数等を操作します。したがって、お使いになる SHELL に合わせた activate ファイルを利用することが必要です。

bash/shell では、

```
source <venv>/bin/activate
```

あるいは

```
. <venv>/bin/activate
```

csh/tcsh では、

```
source <venv>/bin/activate.csh
```

fish では、

```
source <venv>/bin/activate.fish
```

PowerShell Core

```
$ <venv>/bin/Activate.ps1
```

Windows の cmd.exe では、

```
C:\> <venv>\Scripts\activate.bat
```

PowerShell では

```
PS C:\> <venv>\Scripts\Activate.ps1
```

をそれぞれ実行します。 `<venv>` は作成した venv 環境の Path 名 (作成例では、/path/to/new/virtual/environment ）を指定します。

実行すると、

```
bash-3.2$ source  /path/to/new/virtual/environment/bin/activate
(environment) bash-3.2$
```

の様にシェル　プロンプトに利用している venv 環境の名前が追加されます。この状態で起動される python の本体は、

```
(environment) bash-3.2$ which python3
 /path/to/new/virtual/environment/bin/python3
```

となっています。

この venv 環境で、pip3 や"python3 -m pip"を使って、python モジュールをインストールすると、 `<venv>/lib/python3.x/site-packages` に必要なファイルが追加されます。この仕組みによって、通常の環境と venv 環境を切り分けている訳です。

venv 環境を抜ける。

この独立した環境を抜け出して、元の状態に戻るには、deactivate コマンドを使います。

```
(environment) bash-3.2$ deactivate
bash-3.2$
```

1.7 バージョン管理

今時のプログラム開発ではソースコードをバージョン管理することが必須でしょう。

git(`git`)/ mercurial(`hg`) などが現場での主要なバージョン管理ソフトウェアでしょう。
これらのソフトを環境に応じてお使いください。

1.7.1 mercurial(`hg`)

mercurial は python で作られているバージョン管理ツールで、

```
python3 -m pip install mercurial
```

で簡単にインストール可能です。

多くの統合開発環境でもこれらのツールはサポートされています。

1.8 その他

1.8.1 cython

もっと速く！：C/C++のライブラリと Python の結合

プログラムの実行時間の 99%はプログラムの 1%の実行であると言われることがあるように、プログラムのごく一部を高速化することで、飛躍的に性能が向上します。

python 処理系はインタプリタ形のシステムなので、実行速度は C++/Fortran などと比べれば遅くなります。　cython は python 言語のサブセットに独自の拡張を加えた cython 言語で書かれたプログラムをコンパイルし、python から呼び出せるようにしてくれます。cython 言語は python 言語と一定の互換性がありますので、python 言語で書かれたプログラム（関数）をコンパイルすることも可能です。cython は既存の別言語 (C++とか）で書かれたライブラリを python から利用する場合にも使われます。

`numba` といった同様の機能をもったツールもあらわれています。

1.8.2 numpy/f2py/scipy

numpy は python で高速な数値計算を行う場合の実質的な基盤となっている、numpy.ndarray データ構造などを提供しています。`scipy` は科学計算向けのライブラリです。

1.8.3 numba

`numba` は python(のサブセット）で記述されたプログラムを、実行時に機械語に変換して実行する JIT(Just In Time) 型コンパイルツールです。一旦機械語に翻訳された結果は再利用されますので、大量のデータを処理する際には、翻訳 (コンパイル）の時間を考慮しても、大きな実行速度の改善が見込まれます。

1.8.4 Python のチューニング： cProfile/profile/timeit

1.8.5 py2app/py2exe

python で作成したプログラムを単独のアプリケーションとして配布可能な形にまとめあげてくれるツールです。作成した python プログラムを必要な python モジュールおよび python の処理系をパッケージ化してくれます。

1.8.6 python2 と python3 の違いについて

Python2 から Python3 へのバージョンアップでは、

- print が文から関数になった。
- Tkinter の名前が tkinter に変更された。
- 文字型データ (str) が unicode になった。

など Python 言語の文法の変更、モジュール名の変更／統合など広範な変更があります。幸いなことに、2to3 などのツールを使うことで、python2 向けのスクリプトを python3 向けのスクリプトに書き換えることや、python2/python3 両用のスクリプトの書き換えることなどをサポートしてくれるツールが存在します。詳細は、拙著のメモ Python2 から Python3 へ移行するときに知っておくといいかもしれない幾つかの事 (http://www.j-parc.jp/ctrl/documents/articles/python2to3/index.html) がご参考になれば幸いです。

1.8.7 python の哲学 (禅)

import this を実行してみましょう。

```python
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

“PEP 20 – The Zen of Python”は、 Python の先駆者の一人である Tim Perters 氏が拝察した Python の産みの親 Guido van Rossum 氏の Python 設計の指針を簡潔に表現したものです。

筆者の個人的見解ですが、これは”Zen(禅)”であって、 教条主義的に守るためのルールというわけではなく、迷った時の指針として使われるべきものと考えます。 （”禅宗は臨機応変”だそうです。）

“PEP 20”には、”Long time Pythoneer Tim Peters succinctly channels the BDFL’s guiding principles for Python’s design into 20 aphorisms, only 19 of which have been written down.”となっていて、あと一つ書き下されていない指針があることになります。 未完のまま残しておくのは、”Zen"らしいところですね。