第2章　始めの一歩： Hello World!

ブライアン・カーニハンとデニス・リッチーによる著書「プログラミング言語 C」（1978年）以来、プログラミング言語の最初の例として、文字列“Hello, World!”を画面に印刷するプログラムがよく使われています。この習慣に迎合して、ここでも Python 版の HelloWorld プログラムの一例を示します。

2.1 最初のバージョン

Python はインタプリタ型言語なので、端末からプログラムを一行づつ入力して動作を確認することができます。

```
% python3
Python 3.9.4 (v3.9.4:1f2e3088f3, Apr  4 2021, 12:32:44)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Hello, World!")
Hello, World!
```

端末で python3 コマンドを実行すると、python3 の入力プロンプト>>>が表示されます。

ここで print("Hello, World!") を入力して Enter キーを押せば、端末に文字列 “Hello, World!” が表示されます。

```
%%bash
python3
print ("Hello World!")
```

```
Hello World!
```

```
%%python3
print ("Hello, World!")
```

```
Hello, World!
```

…

が、ここでは繰り返し何度も使えるプログラムとしての HelloWorld プログラムをご覧いただきます。

2.2 Python プログラム：hello.py を用意する

作成したプログラムを繰り返し何度も使うために， プログラムはファイルに保存した形で用意します。

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# save this as hello.py
# 行中の#以降は行末までコメントとなる (Python プログラムの実行に影響しない)

def Hello():
    print("Hello, World!")
    return

if __name__ == "__main__":
    Hello()
```

```
Hello, World!
```

この様に Python の中で関数 hello() を定義することで、同じ動作を少ない手順で実行することができるようになります。(Python) プログラムの作成では、問題に応じて解決のための手順を分解／整理して関数やクラス （クラスについては後々説明します）の形でまとめておくということが大切です。

Jupyterlab, VSC, emacs などのプログラム開発支援機能のあるエディタでは、プログラムの実行結果をそれらの環境の中で確認することができます。具体的な手順についてはそれぞれの開発環境のマニュアルをご覧ください。近くに同じ環境をお使いの方がいらっしゃれば、その方に聞いてみるのも良い方法です。 質問の際には、疑問点が具体的にわかるように事前に質問を整理しておくと良いでしょう。

2.2.1 hello.py の中身

このプログラムの中心部分は def Hello():で始まる三行のプログラムです。この三行のプログラムによって、関数 Hello() を定義しています。

```
def Hello():                # 関数の定義を宣言
    print("Hello, World!") # 関数の本体、インデントに注意
    return                  # 関数呼び出しの終了
```

これらの行を一行づつ説明していきます。

2.2.2 関数の宣言

まず、def Hello():で関数名が Hello という関数を定義することを宣言しています。

次の、() はこの関数には引数がないことを示しています。

:はこの後に関数の本体が続くことを示しています。

2.2.3 関数の本体

次の行がプログラムの本体です。

python3 では print は（文ではなく）関数です。引数に与えられたデータを文字列として、端末に表示します。

この print("Hello, World!") が行頭から始まっていない、つまりインデントされている、ことが Python の特徴です。C/C++では{}を使って一連のプログラム文が一つの単位(ブロック、スイート）となっていることを示しますが、Python ではこのブロック構造を行のインデントを使って表現します。(C/C++などでは、プログラムの読みやすさのためにブロック構造をインデントさせて表示することが普通ですが、Python ではこのような読みやすさのためのインデントをプログラム言語の仕様に含んでしまったというところです。）

2.2.4 関数呼び出しの終了

次の return 文はこの行が実行されると関数呼び出しが終了し、関数を呼び出したところからプログラムの実行が継続されます。この return 文もインデントされていますので、hello() 関数本体のブロックに含まれています。

2.2.5 関数の呼び出し

```
if __name__ == "__main__":
    Hello()
```

```
Hello, World!
```

続く二行のプログラムは、ファイルに python プログラムを保存する際に使われるイディオムです。（なぜこうするのか？は後で説明します。）

- if 文は、文中の条件 (ここでは__name__ == "__main__") が満足された時、

- if 文の本体 (インデントで区別された一連のプログラム文, ここでは Hello()) を実行します。

if 文の文末は:で、実行すべきブロックがインデントされていることに注意しましょう。

__name__は Python システムが利用する変数で、**name 属性はモジュールの完全修飾名に設定されなければなりません。**となっています。　python プログラムが python3 コマンドから起動されたとき (後述）は“**main**”に設定されるという約束になっています。

if __name__ == "__main__":は、%python3 Hello.py のように, コマンドラインからこのプログラムが実行された時、続くコードブロック (:があり、続く行がインデントされていることにご注意ください。）を実行することを意味していいます。ここでのコードブロックの中身は Hello() の一行だけです。Hello() が実行されると、その前に定義した Hello 関数の定義に従って、print("Hello, World!") が実行されて端末に “Hello, World!”が表示されると言うわけです。

この簡単なプログラムでは必要ないとも言えるのですが、今後 python の実用的なプログラムを書いていく上では、このように関数定義と、実際の実行部分を分離しておくことで開発の効率が良くなります。ぜひ習慣化してください。

なお、このプログラムの最初の二行は Unix 系の流儀で、このファイルの中身が“python3”のプログラムであること (#!python3)。また、このファイルの文字はユニコード（”utf-8") であること (# -*- coding: utf-8 -*-) を宣言しています。これも習慣として書いておくようにしましょう。　なお python のプログラムでは　行中の#から行末までは**コメント**として取り扱われ、プログラムの実行には影響しません。

###　ここまでのまとめ

- python プログラムの作成では関数を定義していく。
- 関数の定義は def 文を使う。
- 条件判断は　if 文を使う。
- def 文、if 文の中身のプログラムは、行頭を**インデント**する

第3章　プログラムの実行

作成したプログラムを hello.py という名前のファイルに保存します。このプログラムを実行するにはいくつかの方法があります。

3.1 コマンドラインからの実行

第一の方法は、シェルのコマンドプロプトから pytho3(or py -3) コマンドを使って実行する方法です。

ファイル hello.py を保存したディレクトリで、

```
% python3 hello.py
```

を実行しますと、

```
% python3 hello.py
Hello, World!
```

のように、端末に文字列が印刷されます。

このように動作するのは、hello.py に `if __name__ == "__main__":`以下のプログラムを書きこんだことによるものです。

jupyter/jupyterlab をお使いの方は、Code 入力セルにプログラムを入力したら、Shift+Enter を押してみましょう。

もしエラーがでても心配しないでください。まずは以下の解説をお読みください。

jupyterlab ではセルマジック%%bash を使うことで、シェルコマンドの入力結果を文書の中に取り入れることができます。

```
%%bash
python3 hello.py
```

```
Hello, World!
```

端末から同じプログラムを起動する別の方法もあります。

```
%%bash
python3 -m hello
```

```
Hello, World!
```

-m のパラメータには python モジュール名を指定します。 .py がついていないことにご注意ください。

3.2 シェルスクリプトとしての実行

Linux や macos では作成した python プログラムをシェルコマンドとして実行することもできます。

```
%%bash
chmod +x hello.py
./hello.py
```

```
Hello, World!
```

`chmod +x hello.py` は作成した Python プログラムファイルをシェルコマンドとして実行可能とするためのおまじないです。

hello.py の冒頭に

```
#!/usr/bin/env python3
```

があることも必要です。

python プログラムを実行可能ファイルとすることは、セキュリティホールになる可能性もありますので、慎重に行いましょう。

python プログラムをシェルスクリプトとして使う場合には、`sys` モジュールの `sys.exit()` 関数でプログラムが異常に終了した場合、呼び出した shell にエラーを返すことができます。

3.3 python モジュール内の関数としての実行

こうして作ったプログラムを別の Python プログラムとして組み合わせて使うことも可能です。

```
% python3
python3
Python 3.9.6 (v3.9.6:db3ff76da1, Jun 28 2021, 11:49:53)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import hello
import hello
>>> hello.hello()
hello.hello()
Hello, World!
```

別の Python プログラムで HelloWorld.py の中の hello() 関数を使うために、まず HelloWorld プログラムを Python モジュールとして import します。

```
import HelloWorld
```

この時、HelloWorld.py の if __name__ == "__main__":の中身は実行されないことに注意してください。

import した HelloWorld モジュール中の hello() 関数の実行は次のように記述します。
　つまりモジュール名 HelloWorld と関数名 hello を . で繋いだ名前で、HelloWorld モジュール中の hello() 関数を指定します。

```
HelloWorld.hello()
```

```
Hello, World!
```

```
'Welcome to the World.'
```

Python プログラム中の関数として実行した時には、関数が返した値も表示しているのに注意しましょう。

```
%%python3
import HelloWorld

print(HelloWorld.hello())
```

```
Hello, World!
Welcome to the World.
```

3.4 プログラムの実行 (jupyter あるいは jupyterlab)

jupyter あるいは jupyterlab でこの解説をご覧の方は、次のコードブロック (セル）を選択して、shift+Enter　キーを押してみてください。出力の Hello World! が更新されたのがお分かりでしょうか？

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# save this as HelloWorld.py
#
"""
Python 入門講座　第 2 回の例題プログラム
"""

_DEFALUT_RETURN_VALUE="Welcome to the World."
__DEFALUT_MSG="Hello, World!"

#デフォルトの値が"Hello, World!"である引数 msg をもつ関数　hello を定義する。

def hello(msg:str=__DEFALUT_MSG)->str:
    """
    引数　msg(省略時の値は"Hello, World") を端末に出力し、
    "Welcome to the world。"を値として返す。
    """
    print(msg)
    #関数の戻り値として""Welcome to the World."を返す。
    return _DEFALUT_RETURN_VALUE

def test():
    help(hello)
    reply=hello()
    print(reply)

if __name__ == "__main__":
    test()
```

```
Help on function hello in module __main__:

hello(msg: str = 'Hello, World!') -> str
    引数　msg(省略時の値は"Hello, World") を端末に出力し、
    "Welcome to the world。"を値として返す。
```

(次のページに続く)

(前のページからの続き)

```
Hello, World!
Welcome to the World.
```

3.5 プログラムの実行：まとめ

- コマンドラインとして： ` python3 hello.py`
- シェルスクリプト　として：　` ./hello.py`
- (別の）python プログラム中で　：　`import hello; hello.hello`
- python　モジュールとしてコマンドラインから：　`python3 -m hello`
- 開発環境のエディタの中で実行する。　：開発環境に依存します。

第4章　変数名／関数名について

プログラムで使われる色々な名前 (変数名、関数名、クラス名、モジュール名、..) は識別子と呼ばれます。識別子は、これから説明するに従っていれば自由につけることができます。

とはいうものの、後でプログラムを再読したときに、それらの識別子がさすものの意味がわかるように名前をつけましょう。

4.1 識別子に使える文字

識別子はざっくり言って、

- 識別子の最初の文字は、大小のアルファベット (a-zA-Z), あるいはアンダースコア (’_’) が許される。- それ以降は、それに加えて数字 (0-9) も使用可能 - 長さには制限はない。 - 大文字小文字は区別される - python の予約語ではないこと

というルールに従います。

例えば、

x y address *my_name* a0 a1

などはいずれも有効な名前 (識別子）です。

4.2 Unicode 文字の利用

Python3 ではさらに　unicode 文字　も使えます。　識別子として使える unicode 文字の範囲については、Language Reference マニュアルをご覧ください。

ということで、こんなプログラムを書くこともできます。

```
#!python3
# -*- coding: utf-8 -*-

def 挨拶():
    print("Hello, World!")
    返事= "ようこそ, いらっしゃいませ。"
    return 返事

if __name__ == "__main__":
    挨拶()
```

```
Hello, World!
```

```
挨拶()
```

```
Hello, World!
```

```
'ようこそ, いらっしゃいませ。'
```

とはいえ、これをお薦めしているわけではありません。

4.2.1 予約語

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

これらの識別子は Python で予約されているため、　ユーザーが定義する、関数／変数／クラス／モジュール名などに使うことはできません。

これらの予約語は Python の固有の単語ということですから、この **35** 個の予約語の使い方をマスターすれば、どんな Python プログラムを読みこなすことが可能です（原理的には）。　簡単でしょう？

蛇足ですが、予約語の他にも記号　「~!@#%^&*()-+/={}[]|:;"’<>.,　」の使い方も学習する必要があります。

4.2.2 予約後語の分類

- 定数
  - False None True
- 論理演算
  - and not or in is
- 実行制御
  - if elif else while for
  - break continue pass return yield
- ライブラリ
  - import from as
- 非同期処理
  - async await
- 例外処理
  - try except finally raise
- 宣言／定義
  - class def del global nonlocal
- その他
  - lambda: ラムダ式
  - assert: デバッグ用
  - with :コンテキストマネージャ

4.2.3 ちょっと寄り道

python3 では print は関数名であって、予約語でもありません。だからこんなこともできてしまいます。

```
印刷=print
印刷 ("イロハ")
印刷=挨拶
印刷 ()
print('ABC')
印刷=print
```

```
イロハ
Hello, World!
ABC
```

4.3 文字列定数について

Python での文字列定数 (リテラル文字列）の記法には 2 種類あります。

- 一組の引用符で挟まれた文字列定数（ 「“…”」 あるは「‘…’」)
- 一組の 3 連引用符で挟まれた文字列定数（「"" “…” ""」 あるは「’’’ …’’’」)

後者は中身の文字列に改行を拭くんでいても良い、すなわち中身の文字列が複数行にまたがっていてもよいというところです。

なお、Python の文字列は C 言語のそれと異なりシングルクォート「’」を使った文字列とダブルクォート「”」を使った文字列に意味的な違いはありません。

また、Python の文字列は Unicode の文字の並びです。C 言語の文字列に相当するものは、`bytes` あるいは `bytearray` です。（いずれまた、しょうさい

文字列については、次回以降の講座でより詳しく説明します。

4.4 docstring を使おう。

ファイルの冒頭、関数定義の冒頭に置かれた文字列データは docstring と呼ばれます。コメントは後々プログラムを＊読む＊際に役立ちますが、docstring はそのプログラム／関数を利用する際に役立ちます。help() 関数を実行することで、docstring の中身が端末等に出力されます。

```
#!python3
# -*- coding: utf-8 -*-
# save this as HelloWorld.py
#
"""
Python 入門講座　第 2 回の例題プログラム
"""

_DEFALUT_RETURN_VALUE="Welcome to the World."
__DEFALUT_MSG="Hello, World!"

def hello(msg:str=__DEFALUT_MSG)->str:
    """
    引数　msg(省略時の値は"Hello, World") を端末に出力し、
    "Welcome to the world。"を値として返す。
    """
    print(msg)
    return _DEFALUT_RETURN_VALUE

def test():
    help(hello)
    reply=hello()
    print(reply)

if __name__ == "__main__":
    test()
```

```
Help on function hello in module __main__:

hello(msg: str = 'Hello, World!') -> str
    引数　msg(省略時の値は"Hello, World") を端末に出力し、
    "Welcome to the world。"を値として返す。

Hello, World!
Welcome to the World.
```

4.5 まとめ

- 関数定義には `def <関数名> (<引数のリスト>):` の構文を使う。
- 行末の“:”は　インデントされたコードブロックが続く事を示している。
- 関数の本体は　インデントされた python の文の集合となる。
- #から行末まではコメントとなる
- if 文は `if <条件> :`の構文を使う。
- <条件>の論理式としての値が`False`でないとき、続くコードブロックが実行される。

プログラム言語学習の定番である“Hello World!”プログラムの python 版を例に、python でのプログラムの最初の一歩をご紹介しました。　Python では「プログラムのブロック構造をインデントで表現する」と言う手法をとっています。これはその他のプログラム言語との大きな違いとなっており、他のプログラム言語を習得済みの方は特に、戸惑うところかもしれません。　Python でのプログラム開発をサポートしてくれるエディタなどを使うことで、この点は軽減することができるでしょう。

さて、hello.py を作成して実行してみたが、エラーメッセージが出てしまった方は、上記の解説と端末にあらわれたエラ～メッセージを読み解いて、プログラムを修正してみてください。また、エラーなく実行できた方は、　表示する文字列を変えてみて、実行してみましょう。

さあ、無事に最初の一歩は踏み出せたでしょうか？　次はどこに向かいましょうか？

蛇足：

`import this` の Zen と合わせて Yoda に帰せられるとされる次の格言も覚えておきましょう。

Yoda:.. If once you start down the dark path, forever will it dominate your destiny, consume you it will.

…

Yoda:　You will know. When your code you try to read six months from now.

4.6 おまけ：名前も印刷してみよう

出力するメッセージに、メッセージの受け手の名前も加えてみましょう。

引数としてメッセージの受けての名前を受け付ける、新しい関数 HelloTo を定義してみましょう。

```
"""
引数をもつ関数の定義の例
単独のスクリプトして実行された時には、標準のモジュール getpass を使ってユーザの名前を取り出して使う。
"""
def HelloTo(name:str)->bool:
    """
    引数 name に与えられた名前の文字列を使って、グリーティング メッセージを端末に表示する。
    """
    print("Hello {}!".format(name))
    print("Welcome to the Python world!!")
    return True

import getpass # getpass モジュールを import = モジュールの提供する関数などを使えるようにする。

def main():
    username=getpass.getuser() # getpass モジュールの getuser() 関数を使って、username を取得する。
    HelloTo(username)

if __name__ == "__main__":
    main()
```

```
Hello noboru!
Welcome to the Python world!!
```

関数の引数に値を与えて実行して見ます。

```
HelloTo("Albert")
HelloTo(name="Einstein")
main()
```

```
Hello Albert! Welcome to the Python world!!
Hello Einstein! Welcome to the Python world!!
Hello noboru! Welcome to the Python world!!
```

4.6.1 文字列データの Format(整形）

print() 関数は複数の引数を受け取った時、渡されたデータ型に応じて整形した文字列を端末に出力します。

出力される文字列をより細かく制御するためには文字列データの.format メソッドを使います。

(Python2 との互換性のために、別の方法 (""%) もサポートされていますが、.format メソッドが python3 での標準的な方法です。)

```
print("Hello {}!".format(name))
```

4.6.2 エスケープ文字列

文字列定数中に改行コードなどを挿入するには、C 言語と同じように、エスケープ文字\をつかったエスケープ文字定数を使います。

- 改行->\n, タブ　-> \t, ベル->\a,
- 八進数-> \012,　十六進数-> \x0a ,
- 一重引用符 (’)-> \', 二重引用符 (")-> \", エスケープ文字-> \\