Pythonでプログラミング

2019-07-08　　山内 長承　　yamanouc@is.sci.toho-u.ac.jp

前回の残り　Jupyter Notebook　〜〜　Pythonプログラミング環境

インストール：　コマンドラインに対して　`pip jupyter notebook`

起動：　自分が使いたいディレクトリ（例：work）へ`cd`で移動した後、コマンド`jupyter notebook`

終了の仕方：　①　まずプログラムを開いているすべての窓を　`File` → `Close and Halt` で閉じる

②　最初に出てきたファイルのリストのページの右上　`Logout`　でログアウト　これで終りだが

③　最初に起動したターミナル画面で、`Ctrl-C`　（Ctrlキーを押しながら`c`のキーを同時に押す）→ `y`

プログラム作成のウィンドウを開く　最初の画面の右上　`New` → `Python3` で新しいウィンドウが開く

すでにあるファイル（拡張子 ipynb）を開くには、最初の画面でファイル名をシングルクリック

プログラム作成のウィンドウで、名前を付ける　左上 `File` → `Rename`　さもないと`Untitled`になる

プログラムを書く　→　実行してみる　`Run` ボタン

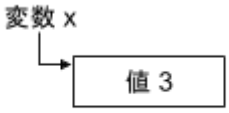
変数内容の確認：　基本的には、`print()` 関数を使う　〜　後述

基本的に自動セーブされている（されてしまう）　`File` → `Save and Checkpoint` で強制セーブ

プログラミングの出発点　〜〜　最低限のPython

変数（Variables）と定数（Constants）、　箱、中身（値）

箱があって、中に値が入る。　　箱のことを変数（variable）と呼ぶ　←　あとで少し見方を変える



変数の名前：　変数を区別する（識別する）ので「識別子」（identifier）という

識別子（変数の名前など）の付け方のルール：

英字 (大文字・小文字) + 数字 + `_`（漢字は不可）、長さは制限なし

予約語（キーワード）を除く

```
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
```

Pythonでは、変数は「宣言」しない　（他のコンパイラ言語では宣言するのが普通）　～　型も明示しない

代入：　変数（箱）の中に値を書込む　＝　代入（assign）する　⇒　`x = 3`

注意）　等号 `=` は「等しい」ではない。（右辺から左辺への）代入を表す。

ほとんどの言語では `=` を使うが、Rでは明確化のため `<-` を使う

代入の左辺は（普通は）変数で、右辺は何かの値

← たとえば `3`　　`3` は「定数」（constant）と呼ばれる。

文字列定数（あとで）：　`' ... '` または `" ... "` で囲んだ文字列

変数でもいい　`y = x`　右辺は `x` の値（中身）

式でもいい　`y = x + 3`　右辺は `x+3` の値（「式 `x+3` を評価する」という）

式　（Expressions）

定数、変数、（または関数呼出し）を、演算子で繋いだもの　`x + 3`　`2 + 3`　`x`

演算子：　`+` とか `-` とか

```
+       -       *       **      /       //      %       @
<<      >>      &       |       ^       ~
<       >       <=      >=      ==      !=
```

`**` べき乗　`/` 割り算　`//` 整数の割り算の商　`%` 剰余（整数の割り算の余り）

`<<` ビット左シフト　`>>` ビット右シフト

`&` AND　`|` OR　`^` XOR（排他的論理和）　`~` NOT

`<` <　`>` >　`<=` ≤　`>=` ≥　`==` 等しい　`!=` ≠

print()

関数 print() は、引数（パラメータ）に与えたものの内容を表示する。

```
x = 3
print(x)   --> 結果は 3 を表示
```

リスト・辞書

リスト：　要素が並んだもの。　`[1, 2, 3, 4, 5]`　`['東京タワー', '通天閣', 'スカイツリー']`

- 混在も可　`[1, 'John', 3.5]`
- リストのリストも可　`[[1, 2, 3], [1, 4, 9], [1, 8, 27]]`　要素は何でもいい
- 要素の取り出しは、何番目かを指定　`l = [0, 1, 2, 3, 4]` に対して、`l[2]` の値は `1`

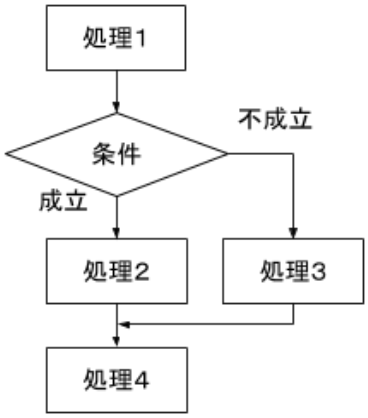
```
l = [0, 1, 2, 3, 4]
print(l[2])      #  結果は 1
```

- リストの長さを知る関数　`len(リスト)`　たとえば　`len([1, 2, 3, 4, 5])` は `5` を返す

辞書：　キーと値のペアが並んだもの。`d = {'東京タワー': 333, '通天閣': 108, 'スカイツリー': 634}`

- キーで引くと値が返ってくる　　`d['東京タワー']`　の値は333

if（条件分岐）



処理１の後、条件が成立すれば処理２を、成立しなければ処理３を行う。

```
if 条件:
    処理２
else:
    処理３
処理４
```

- `if 条件`　の後のコロン`:`　と　`else`　の後のコロン`:`
- 処理２、処理３　の段下げ（インデント）　　←　ブロックを示す

    段下げの終わりまでがブロックの範囲になる

        段下げの状況が（コンピュータから見てブロック範囲の識別に）重要になる

        段下げを揃えることが必要で、ずれると動かない（他言語ではいい加減でいい）

        段下げの文字数は決まりはないが、空白4文字か又はタブか、どちらかに統一する

            混在するとエラー、Jupyter Notebookではタブは自動的に空白4つに変換（変更可）

```
（xの値が決まっているとする）
if x < 0:
    x = -x
print(x)
```

```
（xの値が決まっているとする）
if x > 3.5:
    x = x ** 2
else:
    x = -1
print(x)
```

for　繰り返し（ループ）

Pythonの`for`による繰り返し（forループ）は、（リストとして）与えらえた要素のすべてに対して、ブロックを（繰返し）実行する

```
for u in [0, 1, 2, 3, 4]:   # <-- リストのすべての要素uについて、
    print(u**2);            # <--     2乗してprintする
```

`range(n)` 関数は `n` を与えると、`[0, 1, 2, ... , n-1]` を作ってくれる。 `n` が大きいときに便利。

```
for u in range(10):
    print(u**2)   # 0 1 4 9 ... 81
```

`range(b, e)` は始点 `b` から終点 `e-1` まで、 `range(b, e, s)` は始点 `b` から終点 `e-1` まで `s` おきに

```
for u in range(3, 10):
    print(u**2)   # 9 16 25 ... 81
for u in range(0, 10, 2):
    print(u**2)   # 0 4 16 36 64
```

`s` は負の値でもよい

```
for u in range(-1, -5, -1):
    print(u)      # -1 -2 -3 -4
```

`len` と `range` を組合せると

```
a = ['東京タワー', '通天閣', 'スカイツリー']
for u in range(len(a)):   # <-- [0, 1, 2, ... , len(a)] のすべての要素 u について
    print(u, a[u]);             # <--     uとa[u]をprintする
```

break 文を使って、ループを脱出することができる

```
for u in range(100):
    if u>= 3:
        break       # uが3になると脱出
    print(u)        # 結果は 0  1  2
```

［例題］ 1からNまでの和を求める。 Nを10や100にしてみる。

残っている話題：

- やや複雑なプログラム
- 関数が定義できるときれいになる
- 辞書型は便利、 enumerate、zip は便利
- ライブラリの利用 import文 プラス pipによるパッケージインストール
- pandasライブラリ Excelの表のイメージで計算できる
    - 表の作成、要素の参照、行の参照、列の参照、書込み
    - 条件による要素・行・列の抽出
    - 要素ごとの演算、行・列の集計演算
    - CSVファイルの読み書き、xlsxファイルの読み書き
- matplotlibライブラリ、seabornライブラリ
    - 折れ線・棒グラフ・パイチャート、ヒストグラム・箱ひげ図、散布図

この辺までくると、まあプログラムが書ける。その次のステップは

- クラス定義を読める・書ける 統計ライブラリ