Python と SageMath

佐々木格　(信州大学理学部)

2020 年 4 月 26 日

概要

Python は非常に良くデザインされたプログラミング言語で，覚えやすく可読性の高いコードが書ける事が特徴です。本講義の後半では数式処理システム SageMath（セイジ，以下 Sage と略）を学習します。

Sage は 100 個ほどの数学ソフトウェアを統合した大規模なソフトウェアで，基礎代数，微分・積分，整数論，暗号理論，数値計算，可換代数，群論，組み合わせ論，グラフ理論等の計算を行うことができます。手軽にグラフを描画することもできるし，数学の研究で本格的に使うこともあります。

Python には系 2 と系 3 の二つの系統があり，それらには完全な互換性はありません。Sage のプログラムは Python の文法で記述しますので，本講義では，まずは Python の基本事項を学び，後半で Sage を使った数学的な計算を紹介します。最新の Sage[*1]の文法は Python3 の文法に従っていますので，以下では Python3 について解説を行います[*2]。

Python や Sage はフリーソフトウェアですから，インターネットから無料でダウンロードして自分のパソコンにインストールして使うことができます。これらは Windows，Mac，Linux 版がそれぞれ開発されており，大学の環境だけでなく，自分が普段使用しているマシンにインストールして自由に使うことができます。

[*1] 2020 年 4 月 17 日現在の最新版は SageMath ver.9.0

[*2] 昨年度までは Python2 を教えていたので，再履修の学生は注意してください。

第 I 部

Python の基礎

このプリントは Linux での実行を想定して書かれています。今期は Windows ユーザーを意識して書き直しましたが，Windows ユーザーは次のような読み替えを行ってください。

- 端末 → コマンドプロンプト，Windows PowerShell
- テキストエディタ → メモ帳[*3]
- ディレクトリ → フォルダ
- 記号「\(バックスラッシュ)」 → 記号「¥」

1　Python プログラムの実行手順

このプリントでは Python プログラムを実行する方法として次の 2 つを紹介します。

(1) Python のプログラムが書かれたファイルを作成して端末から実行する。
(2) インタラクティブシェルを使う。

本講義の後半で解説しますが，数式処理システム SageMath から実行することもできます。

1.1　最初のプログラム (Hello World)

(1) の手順を詳しく紹介します。プログラムの実行の流れは次の図の通りです：

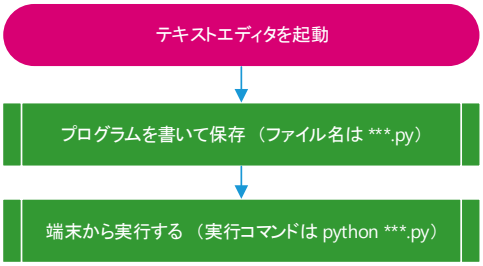


図 1　Python プログラムの実行方法 (1)

Python プログラムはテキストエディタで書きます。まず，デスクトップにある情報処理 I のフォルダ (`dataproc1`) をダブルクリックして開きます[*4]。すると先週作成した `test.py` という名前のファイルがあると思うので，これをメモ帳で開きます。まずは，左上の「ファイル」をクリックし，「名前を付けて保存」を選びましょう。ファイル名を `hello.py` にして保存をします。これで，デスクトップにあるディレクトリ `dataproc1` にファイル `hello.py` が作られました。さて，`hello.py` を次の内容に書き直してください：

- ファイル名：**hello.py**

```
1 print('Hello World!')
```

[*3] もしくは好みのテキストエディタ (秀丸エディタ，Emacs，Sublime Text など)。Word は不可。
[*4] 第 1 回目の講義でディレクトリを作成したのですが，これがない学生はデスクトップに「`dataproc1`」という名前のフォルダを作ってください

入力したら保存します。これで最初のプログラムは完成です。このプログラムを実行するために，dataproc1 のフォルダに戻りましょう。daraproc1 のフォルダ上の空きスペースで [Shift]+[右クリック] し，端末 (PowerShell) を起動します。端末から次をタイプすることで Python プログラムが実行されます。

```
PS C:\Users\*****\Desktop\dataproc1> python hello.py
Hello World!
```

上のように端末に `Hello World` と表示されれば成功です。上記の表示で*****の部分はユーザー名が入ります。

1.2　日本語を含む Python プログラム

Python2 では日本語を含む Python プログラムを書くには，文字コードを指定する必要がありましたが，Python3 からは不要になりました。次のプログラムを作成して，端末から `python hellojp.py` と実行してみましょう：

- ファイル名：**hellojp.py**

```
print('こんにちは')
```

このファイルの実行結果は次のようになるはずです。

```
PS C:\Users\*****\Desktop\dataproc1> python hellojp.py
こ ん に ち は
```

1.2.1　コメントアウト

Python ではプログラムのコメントは『#』の後に書きます[*5]。コメント部分は実行時には無視されます：

```
print('こんにちは')        # この部分は無視されます
```

このファイルの実行結果は上と同じです。

1.2.2　全角スペースに要注意！！

さて，ここで最も重要な注意事項を説明します。日本語を含む文章やプログラムを書くときには

全角スペース「　←これ」に
常に注意しなければなりません。

スペースには半角スペース「 」と全角スペース「　」があり，全角スペースは半角スペース 2 個分のサイズですが，これらは**全く異なる**ものです。例えば

```
>>> print('あ い')      # 「あ」と「い」の間には半角スペースが存在する
>>> print('う　え')     # 「う」と「え」の間には全角スペースが存在する
>>> print('お　か')     # 「お」と「か」の間には半角スペースが2個存在する
```

は見た目では半角スペース 2 個と全角スペースを区別することはできません。しかし，Python では文字列以外の全角スペースを用いるとたちまちエラーとなります。例えば，次では文字列 bb を定義した直後に全角スペースを入れてしまいました。

[*5] コメントの書き方は文章やプログラムによって異なります。例えば，LaTeX のコメントは % の後に書きます。

```
>>> aa = 3     # aaを3にする
>>> bb  = 4    # bbを4にする（bbの後ろに全角スペースが入ってしまった）
 File "<strin>'', line 1
 bb = 4
    ^
SyntaxError: invalid character in identifier
```

上のようにエラーが出てしまいますが，見た目では何が原因でエラーになっているのかわかりづらいです。ですので，初心者は SyntaxError: invalid character が表示されたら全角スペースの存在を疑ってください。そして，全角スペースは極力使わないようにしましょう。

1.3 Python インタラクティブシェル

次にインタラクティブシェルを用いて Python プログラムを実行する方法を紹介します。Python インタラクティブシェルは，入力したプログラムを順次実行していく簡易的なシステムです。インタラクティブシェルを起動するには端末から python [ENTER] と実行するだけです。

```
PS C:\Users\*****\Desktop\dataproc1> python
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64bit (AMD64
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

カーソルが最後の行で点滅して入力を待っています。四則演算と冪を試してみましょう：

```
>>> 1+3
4
>>> 5-3
2
>>> 4*3
12
>>> 9/4
2.25
>>> 10/3
3.3333333333333335    # 値は厳密ではない！
>>> 12%5    # 割り算のあまり
2
>>> 2**10   # 2の10乗
1024
```

バックスラッシュ (/) は割り算の記号ですが，上のように厳密ではない値になることがあるので注意しましょう。また Python2 では 9/4) は商 2 を返すので Python2 を使うときには注意してください。多くのプログラミング言語では冪は 2^10 のように表しますが，Python では冪は 2^{10} は 2**10 のように表します。後半で解説する Sage のユーザーでは^は冪を意味するように変更されています。また 5/3 は $\frac{5}{3}$ という有理数（厳密）を表すように変更されています。

インタラクティブシェルを終了するには exit() と入力します：

```
>>> exit()                          # もしくは ctrl+d
PS C:\Users\*****\Desktop\dataproc1>
```

2　変数，予約語，文字列，数値，データの型

様々なデータを扱う上で最も基本となる事項を紹介します。

2.1　変数

変数を使った計算を，インタラクティブシェルを用いて紹介します。

```
>>> a = 6          # 変数aに6を代入
>>> a              # aの内容を表示
6
>>> a = 8          # 変数aの値を8に変更
>>> a
8
>>> a = a + 5      #aに5を足す
>>> a
13
>>> a += 1         #aに1を足す
>>> a
14
```

上のプログラムで a=a+5 は a の値を a+5 に変える事を意味しています。このように多くのプログラミング言語では = は恒等式ではなく代入を意味します。

さて，存在しない文字を呼ぶとどうなるでしょうか？

```
>>> b
Traceback (most recent call last):                    # エラーメッセージ
  File "<stdin>", line 1, in <module>                 # エラーメッセージ
NameError: name 'b' is not defined                    # エラーメッセージ
>>> b = -4                                            # bに-4を入れる
>>> a+b
10
```

定義されていない変数を使おうとすると上のようにエラーメッセージが出ます。変数名は一文字である必要はありません。

```
>>> ame = 4
>>> mikan = 5
>>> ame + mikan
9
```

変数名はある程度自由に決めることができますが，いくつかのルールがあります。変数名はアルファベット a–z, A–Z, から始めなければならず，大文字と小文字が区別されます。先頭以外では，数字 0–9, やアンダーバー『_』を使うことが出来ます。それと次に紹介する『予約語』を変数名としては使うことはできません。

2.2　予約語

次の単語は Python の文法上，特別な意味を持つので，変数名として使ってはいけません：

Python の予約語

```
print and for if elif else del is raise assert import from
lambda return break global not try class except or while
continue exec pass yield def finally in
```

2.3 文字列

ここまでに変数，文字列，数値を扱いました。数値は 65, −3, 9.23 等と表されたもの，文字列は'Hello' のようにコーテーションマークで囲まれたもの，変数は文字列や数値等のデータを名前を付けて管理するためのものです。それぞれについてもう少し詳しく解説します。

2.3.1 文字列の定義

文字列はコーテーションマーク' 'や" "で囲まれたものとして定義されます。

```
>>> x = 'hello world'
>>> x
'hello world'
```

ここで x は変数で，'hello world' が文字列です。2 つの文字列は + でつなげることができます。

```
>>> aa = 'Alice'     # 変数aaにAliceを代入
>>> bb = ' and Bob'
>>> cc = aa + bb
'Alice and Bob'        # 文字列aaとbbがつながっている
>>> print(cc)
Alice and Bob
```

最後のように文字列を print するとコーテーションマークがとれたものが表示されます。

2.3.2 文字列の中でのコーテーションと改行

文字列は『" "』か『' '』で囲んで定義しますが，文字列の中でコーテーションマークを使いたい場合にはこれらを使い分けます。

```
>>> a = "This is a 'pen'."
>>> print(a)
This is a 'pen'.
```

改行を含む文字列は次のように『\n』を挿入して作ります：

```
>>> a = 'aaa\nbbb\nccc'
>>> a
'aaa\nbbb\nccc'
>>> print(a)       #printすると\nの部分は改行される。
aaa
bbb
ccc
```

『\n』を使わずに改行をするには，""" """または''' ''' で囲みます。

```
>>> a = '''aaa
... bbb
... ccc'''
```

```
>>> a
'aaa\nbbb\nccc'
>>> print(a)
aaa
bbb
ccc
```

また『\』を使って，改行文字や記号『\』，『"』，『'』などを表すことが出来ます。代表的な例を紹介しておきます：

\改行	⇝	改行を無視する
\\	⇝	\
\"	⇝	"
\'	⇝	'
\n	⇝	行送り

例えば

```
>>> aa = 'コーテーションマークとは\"などのこと'
>>> print(aa)
コーテーションマークとは"などのこと
```

2.4　データの型

文字列は str 型 (string) と呼ばれます。

```
>>> type('hello')          # type('hello')の型を調べる
<type 'str'>               # str型である
```

数値の型でよく使うものに整数型 (int) と浮動小数点数 (float) があります。

```
>>> type(123)        # 123の型は？
<type 'int'>         # 整数型
>>> type(3.14)       # 3.14の型は？
<type 'float'>       # float型
>>> a = 3.14         # 変数aに3.14を代入する
>>> type(a)
<type 'float'>       # 変数aの表すデータ(3.14)はfloat型
```

上では int は整数型 (integer)，float は浮動小数点数 (floating point number) を意味しています。整数型の演算は厳密に行われますが，浮動小数点数の計算では誤差が生じます。どちらも数であることは同じのですが，Python の内部での処理の仕方が異なるために，異なるデータの型として認識されるわけです。では，int 型と float 型の和は何になるでしょうか？

```
>>> aa = 10             #int型
>>> bb = 3.14           #float型
>>> cc = aa + bb
>>> cc
13.14
>>> type(cc)
<type 'float'>       #float型
```

答えは float 型でした。扱う数値がすべて整数の場合に（除法を行わなければ）常に厳密な値で計算できますが，float が一つでも混ざる場合は，厳密性が失われてしまうので注意が必要です。このように，数値計算をするときには，自分が扱う数値がどの型なのかを常に気にする必要があります。

2.5　数値と文字列の変換

さて文字列と数値は足せるでしょうか？

```
>>> 'Alice' + 1999
Traceback (most recent call last):                    # エラーメッセージ
  File "<stdin>", line 1, in <module>                 # エラーメッセージ
TypeError: can only concatenate 'str'(not "int") to str  # エラーメッセージ
```

このように str 型と int 型を足すことはできません。文字列と数値をくっつけるには，数値を文字列に変換する必要があります。

文字列，整数，小数の変換

str()：数値を文字列に変換する（例：str(123)='123'）
int()：文字列の数字を，整数に変換する（例：int('123')=123）
float()：文字列の数字を，浮動小数点数に変換する（例：float('123.45')=123.45）

次のプログラムはうまくいくはずです：

```
>>> aaa = str(1999)      # 数値1999を文字列にしたものをaaaに入れる
>>> aaa
'1999'
>>> type(aaa)
<class 'str'>            # aaaの型は確かにstringである
>>> 'Alice' + aaa        # 文字列'Alice'とaaaをつなげる
'Alice1999'
```

逆に，「文字列になっている数字の列」から整数や小数点数に変換してみましょう：

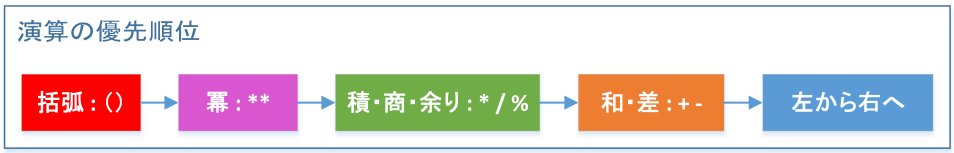
```
>>> aa = '123'       # '123'は文字列
>>> int(aa)          # aaを整数にして返す
123
>>> float(aa)        # aaを浮動小数点数にして返す
123.0
```

2.6　演算子と計算の順序

Python では，数値計算は次の演算子 (operator) で行います：

記号	意味	例
+	和 (Addition)	10+20 は 30 を与える
-	差 (Subtraction)	20-10 は 10 を与える
*	積 (Multiplication)	4*5 は 20 を与える
/	除法 (Division)	10/5 は float 型の数 2.0 を返す。
%	余り (Modulus)	8%5 は余り 3 を与える
**	冪 (Power)	2**3 は $2^3 = 8$ を与える

括弧で囲んだ部分は，例外なく最優先で計算されます。演算の優先順序は次のようになっています：



例えば，5*6/2**2 は，内部では次のような順で計算されているわけです：

```
 5*6/2**2 = 5*6/(2**2)          # 冪が最初に計算される
          = 5*6/4
          = (5*6)/4             # 左から計算される
          = 30/4
          = 7.5
```

ただし，上の順序が適用されない例外がただ一つだけあります。それは冪の計算です。たとえば：

```
>>> 3**3**3
7625597484987
>>> (3**3)**3
19683
>>> 3**(3**3)      # 3**3**3はこちらに等しい
7625597484987
```

この例では，冪は右から順に計算されています。

プログラムには自分で気づかないうちにエラーが混入してしまうものです。単純なミスを防ぐためにも，積や除法を含む計算では，括弧で囲んで順番を明確にするのがよいでしょう。

3　プリント (print)

インタラクティブシェルでは「返されたもの」が順次画面に表示されますが，ファイルから Python を実行する場合は print しない限り，画面には何も表示されません。次のファイルを作り端末から実行してみましょう。

- ファイル名：**print01.py**

```
a = 6
print(a)
b = 4
a + b
print(a+b)
```

実行結果の例

```
6
10
```

結果を見ればわかるように，3 行目，4 行目については何も出力がなく，2 行目, 5 行目で print された数字だけが出力されています。また print 文が複数回あるときは，改行されて表示されます。print 文の後に改行をさせたくない場合は『, end=" "』を書きます：

- ファイル名：**print02.py**

```
a=6
b=4
print(a, end=" ")  # 改行されないで一つスペースが入る
print(b)
```

実行結果の例

```
6 4
```

4　練習問題

4.1　問題：$e^{\pi} - 20$ の計算

今日の課題も簡単です。

pp = 3.141592，ee = 2.718281 とする。$\texttt{ee}^{\texttt{pp}} - 20$ を計算して表示 (print) する Python のプログラムを作成せよ。ファイル名は problem01.py とし，端末 (PowerShell) から

```
PS C:\Users\*****\Desktop\dataproc1> python problem01.py
```

のように実行したときに，その数値を表示するようなプログラムでなければならない。

ちなみに，$e^{\pi} - 20$ は円周率に非常に近い値をとるが，これには何か理由があるのか，それとも偶然なのかわかっていない。