Python と SageMath

佐々木格　(信州大学理学部)

2019 年 4 月 25 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，それらには互換性がありません。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。現在のところ[*1]，Sage のプログラムは Python2 の文法に従っていますので，以下では Python2 について解説を行います。

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] 2019 年 4 月 22 日現在の最新版は SageMath ver.8.7

第 I 部

Python の基礎

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 3 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
(3) Sage ノートブックから実行する

(3) については後半の Sage の解説で紹介します。

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：

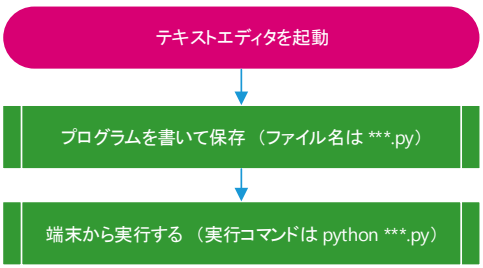


図 1　Python プログラムの実行方法 (1)

Python プログラムはテキストエディタで書きますが，以下では，テキストエディタとして Emacs を用いることを想定しています。次のようにファイル名 (hello.py) を付けてから Emacs を起動します。

```
1 user@debian:~$ cd ~/Desktop/dataproc1/              # ディレクトリを移動
2 user@debian:~/Desktop/dataproc1$ emacs hello.py &          # Emacsを起動
```

Emacs が起動したら C-x C-s を押してファイルを保存しましょう。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。Emacs ウィンドウの下部のバーの表示が Python モードになっていることを確認してください。

さて，hello.py へ次の内容を書きましょう：

- ファイル名：**hello.py**

```
1 print 'Hello World'
```

入力したら保存 (C-x C-s) 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，端末のウィンドウに戻りましょう[*2]。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。

```
1 user@debian:~/Desktop/dataproc1$ python hello.py
2 Hello World
3 user@debian:~/Desktop/dataproc1$
```

[*2] ウィンドウを切り替えは Alt+Tab で行い，極力マウスを使わないことをおすすめします。

上のように端末に Hello World と表示されれば成功です。Python では，プログラムを実行する際に，C 言語のように事前にコンパイルをする必要はありません。

1.2 日本語を含む Python プログラム

日本語を含む Python プログラムは，文字コードを utf-8 で書き，ファイルの先頭に次の一文を追加しておかなければなりません。

```
1 # -*- coding: utf-8 -*-
```

もしくは

```
1 # coding: utf-8
```

Emacs を使って編集する場合はひとつ目の方を推奨します。この一文を入れることで，Python 実行時に，日本語が含まれていることが認識され，日本語の文字の処理が正常に行われるようになります。

例えば，つぎのような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**

```
1 print 'こんにちは'
```

実行結果の例

```
sys:1: DeprecationWarning: Non-ASCII character 'xe3' in file test.py on line 1, but no
encoding declared; see http://www.python.org/peps/pep-0263.html for details
こんにちは
```

のようなエラーメッセージが出ます。日本語を扱うためには，ファイル先頭に次のように文字コードを明示する行を挿入しなければなりません。

- ファイル名：**hellojp.py**

```
1 # -*- coding: utf-8 -*-
2 print 'こんにちは'
```

実行結果の例

```
こんにちは
```

1.3 コメントアウト

Python ではプログラムのコメントは『#』の後ろ書きます[*3]。コメント部分は実行時には無視されます：

```
1 print 'Hello World'       # この部分は無視されます
```

1.4 Python インタラクティブシェル

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] を実行するだけです。

```
1 user@debian:~$ python
2 Python 2.7.9 (default, Jun 29 2016, 13:08:31)
3 [GCC 4.9.2] on linux2
4 Type "help", "copyright", "credits" or "license" for more information.
5 >>>
```

[*3] 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。LaTeX のコメントは % の後に書きます。

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    # 注意！商を返す
2
>>> 9%4    # 割り算のあまり
1
>>> 9.0/4   # 小数点数での割り算
2.25        # 小数点以下も計算される
>>> 2**10   # 2の10乗
1024
```

Python2 では整数同士の割り算 (/) では余りを切り捨てるので注意が必要です。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 ではこの記法はビットごとの排他的論理和 (XOR) に割り当てられており，冪は 2^{10} は 2**10 のように表します。Python3 ではスラッシュ『/』で小数点以下も割り算がおこなわれるように変更されました。後半で解説する Sage のユーザーは数学者が多く，排他的論理和を多用する数学者は少ないので，^は冪を意味するように変更されています。また 5/3 は $\frac{5}{3}$ という有理数を表します。

インタラクティブシェルを終了するには CRTL+d もしくは exit() を入力します

```
>>> exit()                          # もしくは ctrl+d
user@debian:~$
```

2 変数

変数を使った計算を，インタラクティブシェルを使って紹介をします。

```
>>> 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
```

```
3   File "<stdin>", line 1, in ?                    # エラーメッセージ
4 NameError: name 'b' is not defined                # エラーメッセージ
5 >>> b = -4
6 >>> a+b
7 10
```

定義されていない変数を使おうとするとエラーメッセージが出ます。

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りません。変数名では大文字と小文字は区別されます。先頭以外では，さらに数字 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。

3　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
```

4　文字列，数値，データの型

ここまでに，Python で変数，文字列，数を扱いました。数は 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。

4.1　文字列

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。

```
1 >>> x = 'hello world'
2 >>> x
3 '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。

```
1 >>> aa = 'Alice'     # 変数aaにAliceを代入
2 >>> aa
3 'Alice'
4 >>> aa + 'Bob'
5 'AliceBob'
6 >>> aa + ' and Bob'
7 'Alice and Bob'
```

4.1.1　文字列の中でのコーテーションと改行

文字列を定義するときにはその文字を『" "』か『' '』で囲みます。文字列の中でコーテーションをし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'       # コーテーションマークを含む文字列
```

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
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『\』を使い，改行文字や，『\』，『"』，『'』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

4.2 データの方を調べる

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
```

4.3 数値の型

上で 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。そして Python では整数と浮動小数点数の演算は浮動小数点数として実行されます：

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

また整数型 int は-2147483647 から 2147483647 までしかサポートしていません (PC 環境にもよります)。それ以上になると長整数 long 型を使う必要があります。次のようにして，これを確かめてみましょう。

```
>>> dd = 2**62              # ddは2の63乗
>>> dd
4611686018427387904
>>> type(dd)
<type 'int'>                # これはint型だけど
>>> ee = 2**65              # eeは2の65乗
>>> ee
92233720368547758O8L        # 最後のLはLong型の印
>>> type(ee)
<type 'long'>               # これはlong型
```

なぜ，同じ整数を扱うのに int 型と long 型があるのでしょうか？自分が電卓を使って計算をすることを考えてみましょう。電卓の桁数の中で収まる計算なら，ボタンを数回押すだけで計算は直ちに終了しますが，電卓の桁数をはみ出るような計算の場合は，紙を用意して，数字を何桁かに分割して書いて，桁ごとにそれぞれ計算して，最後に繰り上がりに注意しながら和をとり，最後の答えを紙に書く，という作業をしなければなりません。コンピューターの内部でも同じことが起きていて，桁数の少ない数（といっても普通生活では出会わないような大きな数）の計算は用意された CPU の命令で高速で実行することができますが，ある桁数を超えた計算は，より遅くなる別の手順で行わなければなりません。

ちなみに Python3 では int 型とよばれる整数型しかありません。これは python2 の long 型と同じように振る舞います。Python3 は高速性を犠牲にして，整数についてより直感的にプログラムを書けることを重視したということでしょう。

4.4 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 ?                              # エラーメッセージ
TypeError: cannot concatenate 'str' and 'int' objects       # エラーメッセージ
```

上エラーを見ればわかるように str 型と int 型は足せません。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：

```
>>> 'Alice' + str(1999)      #1999を文字列'1999'にしてから'Alice'に付け加える
'Alice1999'
```

逆に，文字列になっている数字列から整数や小数点数に変換してみましょう：

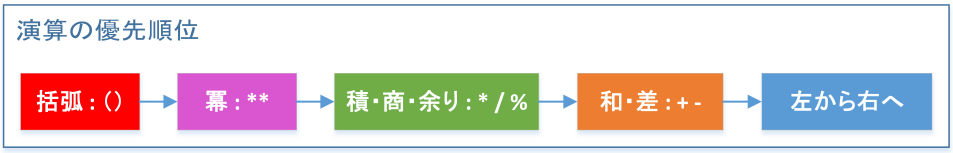
```
>>> aa = '123'       # '123'は文字列
>>> int(aa)
123                  # 123は整数
>>> float(aa)
123.0                # 123.0は浮動小数点数
```

5　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)	14/3 は商 4 を与えるが 14.0/3 は $4.66\cdots 67$ のように小数で計算される
%	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

括弧は例外なく，最優先で計算されます。演算の優先順位は次のようになっています：



例えば，5*4/3**2 は次のような順で計算されます：

```
5*4/3**2 = 5*4/(3**2)          # 冪が最初に計算される
         = 5*4/9
         = (5*4)/9             # 左から計算される
         = 20/9
         = 2                   # 整数同士の割り算は，商が返される
```

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 = 7625597484987L
(3**3)**3 = 19683
3**(3**3) = 7625597484987L
```

この例では，冪は右から順に計算されています。したがって，冪（**）を複数回使用する場合は、括弧で囲んで順番を明確にすることをおすすめします。

Python での割り算『/』は整数同士では商を返すが，少数を含む計算では少数を返すので注意が必要です。上の計算例では

```
   5.0*4/3**2 = 5.0*4/9
              = 20.0/9
              = 2.2222222222222223
```

となります。

6　プリント (print)

インタラクティブシェルでは入力したものが順次ディスプレイに表示されますが，ファイルから Python を実行したときには print を書かない限り，画面には表示されません。

- ファイル名：**print01.py**

```
a = 6
print 6
b = 4
a + b
print 10
```

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 行目についてだけ数字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コンマ『,』を付けます：

```
print 4,
print 2+3
```

実行結果の例

```
4 5
```

（注）上のように print 文の中で計算させることも出来ます。

7　リスト

リストとはデータのいくつかの集まりです。次を試してみましょう：

```
>>> a = ['Alice', 'Bob', 2,3,8]    # リストを定義してaに代入
>>> a                              # aの内容を確認
['Alice', 'Bob', 2,3,8]
>>> type(a)                        # aの型を確認
<type 'list'>                      # aの型はリスト
```

続いて，リストの成分を取り出すには次のようにします：

```
>>> a[0]                           # aの第1成分
'Alice'
>>> a[-1]                          # aの最後の成分
8
>>> a[3]                           # aの4番めの成分
```

```
3
>>> a[-2]                              # aの最後から2番めの成分
3
```

リストの成分の番号は 0 から始まることに注意しましょう。存在しない成分を呼び出すとエラーが出ます：

```
>>> a[8]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: list index out of range
```

スライスという機能を使って，リストの要素を簡単に取り出すことが出来ます：

リストからの要素の取り出し（スライス）

- a[:n] は最初の n 個の要素からなる新しいリスト，
- a[-n:] は最後の n 個の要素からなる新しいリスト，
- a[n:] は最初の n 個の要素を取り除いた新しいリスト，
- a[:-n] は最後の n 個の要素を取り除いた新しいリスト，
- a[n:m] は最初の n 個と最後の m 個を除いた新しいリスト

もっと大きなリストを作ってスライスしてみましょう：

```
>>> a = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p']
>>> print a[:5]
['a', 'b', 'c', 'd', 'e']
>>> print a[-5:]
['l', 'm', 'n', 'o', 'p']
```

次にリストを足し合わせるとどうなるか試してみましょう：

```
>>> a = [1,2,3]
>>> b = ['a', 'b', 'c']
>>> a + b
[1, 2, 3, 'a', 'b', 'c']
```

つまり，リストの和はリストを結合する事を意味します。リスト自身への要素の追加は append() という『メソッド』を使っても行うこともできます：

```
>>> a.append('Alice')
>>> a
[1,2,3,'Alice']
```

変数の値を代入で書き換えるように，リストの値も代入で書き換えることが出来ます：

```
>>> a = [1,2,3]
>>> a[0] = 'Alice'      #a[0]をAliceにする
>>> a
['Alice', 2, 3]
```

次に，リストから要素を削除するには pop(), remove() を使います：

```
>>> aa = ['a', 'b', 'c', 'd', 'e', 'b', 23, 8, 13]
>>> aa.pop()    #aaの末尾の要素を取り出す
13
>>> aa
```

```
 ['a', 'b', 'c', 'd', 'e', 'b', 23, 8, 12]
>>> aa.pop(3)     #aaから3番目の要素を取り出す
'd'
>>> aa
['a', 'b', 'c', 'e', 'b', 23, 8]
>>> aa.remove('b')      #aaにある最初の'b'を取り除く
>>> aa
['a', 'c', 'e', 'b', 23, 8]      # 二つ目の'b'は削除されない
```

連続する数字列のような特別なリストは最初から用意されています

```
>>> range(5)
[0, 1, 2, 3, 4]
>>> range(3,6)
[3, 4, 5]
>>> range(-3,3)
[-3, -2, -1, 0, 1, 2]
```

8 Python で扱えるその他のデータ形式

8.1 タプル

リスト (list) のように要素を集めたものタプル (tuple) があります。タプルがリストと異なるところは，変更が出来ないことです。タプルは要素を丸括弧（）で囲ってコンマで区切ります：

```
>>> a = (3,7, 'abc')      # タプル(tuple)を定義
>>> print a
(3, 7, 'abc')
>>> type(a)
<type 'tuple'>      #aの型はタプル
```

タプルの要素を変更しようとするとエラーが起きます。

タプルのように変更不可能なものを Immutable(イミュータブル) といいます。文字列，数値，タプルは Immutable です。

8.2 辞書 (dictionary)

キー (key) と値 (value) の対応を集めたものを辞書といいます。辞書は次のようにして作ります：

```
>>> a = {'birth':1534, 'type':'A', 'name':'Nobunaga', 'death':1582}
```

辞書のキーを指定すると対応する値を呼び出すことが出来ます：

```
>>> print a['birth']
1534
```

keys() や values() を用いることでキーのリストと値のリストを得られます：

```
>>> b = a.keys()      #aの『鍵』の集まりをbとする
>>> print b
['death', 'type', 'name', 'birth']
>>> c = a.values()      #aの『値』の集まりをcとする
>>> print c
[1582, 'A', 'Nobunaga', 1534]
```

辞書から要素を削除するには，del を使って削除したいキーを指定します：

```
>>> del a['type']      # typeの鍵を削除
>>> print a
{'death': 1582, 'name': 'Nobunaga', 'birth': 1534}
```

辞書に要素を追加したい場合は，新しいキーと対応する値を次のように指示します：

```
>>> a['hobby'] = 'tea'
>>> print a
{'hobby': 'tea', 'death': 1582, 'name': 'Nobunaga', 'birth': 1534}
```

辞書のキーは immutable でなければなりません。つまりリストはキーにはなれませんが，タプルをキーにすることは出来ます：

```
>>> a = {(1,1,0):'police', (1,1,9):'fire', (1,7,7):'weather'}
print a
{(1, 1, 0): 'police', (1, 1, 9): 'fire', (1, 7, 7): 'weather'}
```

キーと値をペアにしたタプルからなるリストをつかって，辞書を定義することも出来ます：

```
>>> a = [(1,'One'), (2,'Two'), (3,'Three')]
>>> b = dict(a)
>>> print b
```

8.3 集合 (set)

要素を集めたものにリストやタプルがありましたが，順番を気にしない要素の集まりが set です。要素は immutable なものだけが set の要素になることができます。集合は次のように定義します：

```
>>> a = [1,4,3,2,2,2]
>>> b = set(a)
>>> print b
set([1, 2, 3, 4])
>>> b.add(5)      #bに5を付け加える
>>>print(b)
set([1, 2, 3, 4, 5])
```

要素はソートされていて重複が除かれているのがわかります。

9 練習問題

9.1 問題：$e^{\pi} - 20$ の計算

pp = 3.141592，ee = 2.718281 とする。$ee^{pp} - 20$ 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，端末から python problem01.py と実行したときに，答えを表示するようなプログラムでなければならない。

$e^{\pi} - 20$ 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。