現象数理1（2020年度 第6回授業）

黒岩大史 kuroiwa@riko.shimane-u.ac.jp

今回の授業日と課題〆切

対面：11月12日　オンデマンド推奨期間：10月29日〜11月18日　課題〆切：11月18日

今回の学習事項

今日は「リスト」を学習します[1]。リストは、現時点では数学でいう「数列のようなもの」と理解して構いません。まず、次のプログラムを見てください。このプログラムにおいて、データが5個ではなく100個だった場合、どうすればよいでしょうか?少なくとも変数に名前をつける事でさえ大変な作業となります。また、これらのデータの分散を求める際や、逆順に表示したり、データを数の大きな順に並べ換える（ソーティング）場合においても、大きな困難が生じます。この問題は、リストを使うことで解決できます。

```
a = 80
b = 56
c = 68
d = 75
e = 45
sum = a + b + c + d + e
print sum
```

1　リスト

まず、リストは x[0], x[1], x[2],... のように使います（添字が 0 から始まることに注意せよ）。

数列	x_0,	x_1,	x_2,	...,	x_n
Python	x[0],	x[1],	x[2],	...,	x[n]

リストは、使う前に準備が必要です。最も簡単な準備の方法は次の通りです。

```
x = [80, 56, 68, 75, 45]
```

これでサイズが5のリスト x ができました。x[0]=80, x[1]=56, x[2]=68, x[3]=75, x[4]=45 となっており、x[i], i=0,1,2,3,4 自体を変数として扱うことができます。このとき文字 x は普通の変数として使ってはいけないことに注意しましょう。また、x[i] はリスト x の第 i 要素と呼ばれます。「print(x)」でリスト x の全要素を出力することができます。

[1] ここでは NumPy を使わずに学習します。NumPy は高度な科学技術計算を Python で可能にするためのライブラリで、ベクトルの内積や行列の積や逆行列の計算など、数値計算のための様々な機能が含まれています。しかしこの授業ではこのような便利な機能を使わず、あえて地道な手法でプログラミングを行うことにします。

2020-6-1.py

```
x = [80, 56, 68, 75, 45]
print(x)
sum = 0
for i in range(0,5):
  sum = sum + x[i]
print(sum)
```

→ 実行結果

```
[80, 56, 68, 75, 45]
324
```

リストを用いて先のプログラムを書き直したものが左のプログラムです（print(x) は特に必要ではありませんが)。仮にデータ数が増えた場合でも容易に対応できるプログラムになりました。

Python では、リストの要素を逆順にする、平均、分散を求める、要素の大きさに応じて並び替える、要素を追加・削減する、などの操作を行うメソッド（方法）が揃っています。リストの理解のため、以下ではあえてそのようなメソッドを使わずに練習します。

練習 1. 上のプログラムの 2 行目まではそのまま使い、必要に応じてそれ以降を変更して以下のようにプログラミングして下さい。

(1) for 文を用いて、リスト x の各要素を逆順 (x[4], x[3], ..., x[0] の順) に表示するプログラムを作成して下さい。(2020-6-2.py)

(2) for 文と if 文を用いて、i が奇数のときの要素 x[i] を表示するプログラムを作成して下さい。(2020-6-3.py)

(3) for 文と if 文を用いて、要素 x[i] が奇数となる i を表示するプログラムを作成して下さい。(2020-6-4.py)

(4) リスト x の各要素の平均よりも大きなリスト x の要素を表示するプログラムを作成して下さい。(2020-6-5.py)

1.1 リストにおける「+」と「*」

ここでは、「+」によるリストの連結と、指定した回数のリストの結合「*」を学習します（他のメソッドも必要に応じて学習していきます)。

2020-6-6.py

```
x = [8]
y = [1,6]
print(x+y)
print(x*2)
print((x+y)*3)
print(x*3+y*3)
```

→ 実行結果

```
[8, 1, 6]
[8, 8]
[8, 1, 6, 8, 1, 6, 8, 1, 6]
[8, 8, 8, 1, 6, 1, 6, 1, 6]
```

「+」を使う場合、リスト同士は連結できますが、リストでは無いものを連結することはできません。上記のように [1,2]+[3] は [1,2,3] となりますが、例えば [1,2]+3 はエラーとなります。また「*」

は繰り返し「+」を実行する際に用いるので、演算子「*」の左右は、「リスト」と「回数（整数値）」である必要があります（整数値が 0 以下の場合には空のリスト [] となります）。

例題. 以下の２つのプログラムは、指定したサイズの整数要素のリストを作成し表示するプログラムです。一つ目は最初に空のリストを用意して繰り返し「+」で結合をしており、２つ目は最初に要素が全て 0 のサイズ n のリストを「*」で用意してから、要素に代入しています。

2020-6-7.py
```
x = []
n = int(input("リストのサイズ："))
for i in range(0,n):
    a = int(input("整数を入力せよ："))
    x = x + [a]
print(x)
```

2020-6-8.py
```
n = int(input("リストのサイズ："))
x = [0] * n
for i in range(0,n):
    x[i] = int(input("整数を入力せよ："))
print(x)
```

2 range() について

これまで説明を保留してきましたが、range() で作られるものは「ほぼ」リストです。list(range(0,5)) が [0,1,2,3,4] と同じ意味です（過去の Python のバージョンでは list() 無しでも同じ意味でした）。また range(0,5) は「0,」を省略して range(5) と書くことができます。さらに range に３つ目の引数を入れることで、次のような意味となります。（ただし k は、c>0 のときは a+kc<b、c<0 のときは a+kc>b をみたす最大の 0 以上の整数）

range	リスト
range(a,b)	[a,a+1,...,b-1]
range(a,b,c)	[a,a+c,a+2c,...,a+kc]

これらについて次で確認してみましょう。

2020-6-9.py
```
print(list(range(10)))
print(list(range(0,10)))
print(list(range(0,10,2)))
print(list(range(10,0,-2)))
```

→

実行結果
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
[10, 8, 6, 4, 2]
```

for 文では、「range()」も「リスト」もいずれも使うことができます。つまりこれまで「for i in range(0,5):」と書いていましたが、直接対応するリストを書いて「for i in [0,1,2,3,4]:」とすることもできます。すなわち、今日の最初のプログラムは、右のように書くこともできます。この場合、n がリストの全ての要素を動きます。

2020-6-10.py
```
x = [80, 56, 68, 75, 45]
print(x)
sum = 0
for n in x:
  sum = sum + n
print(sum)
```

3 応用：エラストテネスのふるいによる素数の検索

エラストテネスのふるいとは、2 から指定した自然数までの素数をすべて見つけ出すための方法で、リストを使う効率の良いアルゴリズムです。以下は 100 以下の素数を列挙するプログラムです。

2020-6-11.py

```
x = [1]*101
for i in range(2,101):
   for j in range(2*i, 101, i):
      x[j]=0
print("2 から 100 までの素数は次の通り")
for i in range(2,101):
   if(x[i]==1):
      print(i)
```

まず x[2]=1, x[3]=1, ..., x[100]=1 となるリスト x を用意します（x[0], x[1] は使いません）。これはスタートの段階では 2, 3, ..., 100 は全て素数候補という意味です。次に、i=2 のとき

```
   for j in range(2*i, 101, i):
      x[j]=0
```

により x[4]=0, x[6]=0, x[8]=0, ..., x[100]=0 とし、4, 6, 8, ..., 100 を素数候補から落とします。これらは約数 2 を持つ合成数だからです。i=3 のときも同様に、x[6]=0, x[9]=0, x[12]=0, ..., x[99]=0 とします。これを i=100 まで繰り返していくと、x[j]=1 のままの j が素数となります。

このプログラムの 6 行目以降の部分は、以下のように簡潔に書くことができます。

```
print([i for i in range(2,101) if x[i]==1])
```

この表記はリスト内包表記と言われ、次のような書き方をします。

```
[式 for i in 「range() やリスト等」 if 条件式]
```

この書き方は練習 1 (1)〜(4) でも利用でき、例えば練習 1 (1) なら次のようになりますが、ここでは詳しくは述べませんし、必ずしも理解しなければならない訳ではありません。

```
print([x[4-i] for i in range(5)])
```

4 発展：ソーティング（メッソド sorted() とバブルソート）

データを何らかの順番（数字なら小さな順あるいは大きな順）に並べ換えることをソーティング (sorting) と言います。整列とも言います。まずは、リストの要素をソーティングするための Python のメソッド sorted() を使ってみましょう。

2020-6-12.py

```
x = [3,5,1,2,6,4]
print(x)
print(sorted(x))
print(sorted(x,reverse=True))
```

→ 実行結果

```
[3, 5, 1, 2, 6, 4]
[1, 2, 3, 4, 5, 6]
[6, 5, 4, 3, 2, 1]
```

また、ソーティングの初歩について学びます。データの並べ替えには様々な方法があります。Python の sorted() は非常に早く、マージソートと呼ばれる効率の良い方法が使われているようです。以下では、バブルソートと呼ばれる最も簡単で効率の悪いソーティングの手法を述べます。隣り合っている要素を交換する動作が、「泡（バブル）の上昇」に似ていることから名付けられています。なおこのプログラムでは、for 文の中に for 文が入っており、このような文を二重ループと言います（この場合、if 文は 25 回実行されています）。覚えなくても良いですが、どのように動作しているのかだけは確認しておいて下さい。

2020-6-13.py

```
x = [3, 5, 1, 2, 6, 4]
print(x)
for i in range(5):
   for j in range(5):
      if(x[j] > x[j+1]):
          tmp = x[j]
          x[j] = x[j+1]
          x[j+1] = tmp
print(x)
```

なお x[j] と x[j+1] の値を入れ替える際、下記の左のようにしてはいけません（なぜなら、x[j] の値が無くなるからです）。一度別の変数（ここでは tmp）に x[j] の値を代入しておく必要があります。また、Python では下記の一番右のように、複数の値を同時に代入することができます。

不適切

```
x[j] = x[j+1]
x[j+1] = x[j]
```

どの言語でも OK

```
tmp = x[j]
x[j] = x[j+1]
x[j+1] = tmp
```

Python なら OK

```
x[j], x[j+1] = x[j+1], x[j]
```

今回の学習事項

今回はリストと関連事項、リストの応用について学習した。

- リストは数学でいう数列のようなもの
- x[i] でリスト x の第 i 要素を表し、i は 0 から始まる
- リストを使うと、複数の要素やデータが用意に扱うことができる
- range() の詳細について学習した
- 応用としてエラストテネスのふるいを、発展としてソーティングを学習した