Part1. Python の基本文法

プログラムを書いてコンピュータに計算をさせる方法を学ぶ。

1. Python はじめの一歩
2. 計算させる
3. 変数
4. 文字列と型
5. リスト
6. for ループ
7. 乱数
8. if 文と論理式

1-1. Python はじめの一歩

（１）起動しよう

スタートメニューから「プログラム」→「アクセサリ」→「コマンドプロンプト」を起動し、立ち上がったら

```
ipython notebook
```

と入力しよう。

ブラウザ（インターネットを見るためのソフト）が起動して次のような画面が表示されるはずだ。

そうしたら New Notebook ボタンを押して、python プログラミングを始めよう。

（２）Notebook の使い方

(1) 入力欄に「1+2」と入力する。Shift+Enter キーを押すと、入力したものが実行される。次のように結果の「3」が表示されたら成功だ。

(2) 次の入力欄に「1+」とだけ入力して、実行してみよう。

1-2. 計算させる

（１）四則演算とか

計算と記号

足し算：+
引き算：-
かけ算：*
わり算：/
あまり：%
べき乗：**
かっこ：()

まずはコンピュータに計算をさせる方法を学ぶ。コンピュータの強みはなんといってもその計算の速さと正確さなのだ。

python では右に書いたような計算をさせることができる。さっそく次の計算を実行してみよう。

・1+2*3+4　　結果は 6 か 11 か？算数と同じで ＊ は ＋ より先に計算される。
・(1+2)*(3+4)
・7373*1507
・2**10　（2 の 10 乗，2 を 10 回掛けたもの）
・123456789*987654321　　こんな無茶な計算も一瞬でできる。答えが 100000 ケタくらいに収まるものならどうにかやってくれる。それ以上のことをやらせようとすると計算が終わらず固まってしまう。

（２）整数と小数

コンピュータは小数の計算もできる。割り算をするときには注意が必要で、整数を整数で割ったときは結果は切り捨てられた整数に、整数と小数を混ぜたり小数同士で割ったときは結果は小数になる。次の計算をさせてみよう。

・10/3
・10/3.0
・10.0/3
・10.0/3.0
・10/0　　0 で割るとエラーになる

整数と小数

整数：0, -1, 100　　小数点なし
小数：-1000.1, 3.14, 1.0　　小数点あり

整数＋整数 → 整数
整数＋小数 → 小数　　(-, *, /も同様)
小数＋小数 → 小数

1-3. 変数

「変数」は、コンピュータに数を覚えてもらうときに使う。変数は「変わることができる数」という意味で、「変な数」ではない。次の例では、「a」「b」が変数であり、1, 2 行目で「=」記号を使って右辺の数をそれぞれの変数に覚えさせている（**代入**という）。3 行目の a*b で覚えておいた 100*200 が実行され、20000 が結果として出力されている。

```
In [31]: a = 100
         b = 200
         a*b

Out[31]: 20000
```

変数は、新しく=記号で代入することにより、**前に覚えていたものを忘れる**。逆に、一度何かを覚えたら、次に上書きされるまではずっと覚えている。

```
In [32]: print(a)      # さっき代入した100をまだ覚えている
         a = 12345     # 12345で上書きする
         print(a)

         100
         12345
```

突然出てきた print だが、これはカッコの中身を出力するというそれだけの命令である。**カッコの中身が変数なら、変数が記憶しているものが表示される**。どうしてこれを使うのかというと、標準では最後に書いた一行しか出力してくれないからである。ここでは二度 a を表示したかったので、print を使っている。

print文

print(…)　　…を出力する

また、シャープ#の後に書かれたものはコメントといって、コンピュータはこれを無視して実行する。

コメント

…　　…は実行されない（メモなどに使う）

まとめ

変数は…　ひとつの数を覚えられる。
数の代わりに使うことができる。
新しい数を覚えさせると、古い数を忘れる。
好きな名前を付けられる(aaa とか piyo とか。大文字小文字は区別される。)

ちなみに、代入と同時に四則演算をすることができる。

代入の記号	
=	左の変数に、右の式の内容を代入する
+=, -=, *=, /=	左の変数に、右の式の内容を足した（引、掛、割）ものを代入する

たとえば、a という変数について、
　a = 1　　a は 1 を覚える。
　a += 2　　a は 1 に 2 を足した 3 を覚える(a = a + 2 と同じこと）。
というように使う。

・課題
（１）変数 pi に、円周率 3.1415926535 を代入しよう
（２）変数 r に、半径 3 を代入しよう
（３）これらの変数を使って、半径 3 の円の面積と周の長さを計算せよ
（４）r に 15 を代入することで、今度は半径 15 の円の面積と周の長さを計算せよ

1-4. 文字列と型

（１）文字列

　変数は整数、小数だけでなく、文章も覚えることができる。文章は文字の列なので、プログラミングの世界では「文字列」といわれる。

　python では、シングルコーテーション（’’）あるいはダブルコーテーション（””）で囲ったものが文字列として扱われる。コーテーションの中ならば、プログラムっぽいもの(“print(a)”)とかが書かれていても、プログラムとしては扱われずにそういった文字列なんだなとして解釈される。逆に、文字列でない（かつコメントでもない）英文は、変数であったり命令であったりと必ず何らかの意味を持っている。

次の例では moji という変数に'hello'という文字列を覚えさせている。続く print 文でそれを表示している。

```
In [45]: moji = 'hello'
         print(moji)

hello
```

文字列同士は＋記号で連結させることができる。

```
In [46]: moji = 'hello '
         retsu = 'world'
         print(moji+retsu)

hello world
```

しかし、文字列と数とを足し合わせることは出来ない。これは、次に説明するように「型（タイプ）」が違うからである。

```
In [47]: moji+3.14

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-9bf2e6f49ccc> in <module>()
----> 1 moji+3.14

TypeError: cannot concatenate 'str' and 'float' objects
```

（２）型

割り算の/記号は整数と小数でやることが違うし、+記号も小数と文字列とでは違う動作をした。このように扱うデータのタイプによって動作を決めるために、コンピュータはあるデータがどのタイプ（型）に属するのかを判断している。

type(...)と書くと、コンピュータが...をどの型だと判断しているかが分かる。

```
In [49]: type(1)

Out[49]: int   整数型

In [50]: type(1.0)

Out[50]: float 小数型

In [51]: type("1.0")

Out[51]: str   文字列型
```

「1」は int(整数)，「1.0」は float(小数)，「”1.0”」は str(文字列)として扱われる。

```
In [60]: a = 100
         print(type(a))
         a = "hello"
         print(type(a))

         <type 'int'>
         <type 'str'>
```

変数も型を持っている。もちろん違うタイプのデータを代入したら変数の型も変わる。

左の例では、最初に a に 100 が代入されて、そのときの a の型は int，次に'hello'で上書きされて、型は文字列型 str となる。

・課題

(1) 変数 s に文字列”abracadabra”を代入せよ

(2) len(s)を実行せよ。len はカッコの中の文字列の長さを答える命令である。

(3) 変数 i に整数 100 を代入し、len(i)を実行せよ。型があっていないというエラー「TypeError」が出ることが分かるだろう。

(4) s+s を実行せよ。文字列同士の足し算は連結である。

(5) s*10 を実行せよ。

1-5. リスト

「いくつかのデータを並べたもの」を扱うのがリストである。たとえば、
　　[1, 2, 3, 4, 5]
は 1 から 5 までの整数を並べたリストであり、
　　[“りんご”, “みかん”, “バナナ”]
は、りんご、みかん、バナナという 3 つの文字列を並べたリストである。

（1）リストの作り方

リストを作るには、次の例のように角カッコ[]の中に、リストに並べたいものをカンマ「,」で区切って書く。

```
In [73]: lst = [1, 2, 3, 4, 5]
         print(lst)

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

同じアイテムをたくさん並べたリストを作るには、次のように**かけ算**をすると良い。

```
In [77]: ["a"]  # 1個だけのリスト

Out[77]: ['a']

In [78]: ["a"]*10  # 掛け算すると、元のリストを掛けあわせた回数だけくっつけたリストを作る

Out[78]: ['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
         'a']
```

整数が順番に並んだリストを作るための **range 命令**がある。
range(1, 10)と書くと、1 から 9 まで（10 でない！）を並べたリストを作ることに注意。

```
In [82]: range(1, 10)    # range(a, b)命令は、a以上b未満の整数を並べたリストを答える

Out[82]: [1, 2, 3, 4, 5, 6, 7, 8,
         9]
```

（２）リストの操作

リストも**変数**に代入することができる。リストを覚えている変数の後に[n]をくっつけることで、n 番目(先頭から数えて n+1 個目)の要素を知ることができる。

```
In [90]: lis = range(1, 10)      # リストも変数に入れることができる。
         print(lis)

         [1, 2, 3, 4, 5, 6, 7, 8, 9]

In [84]: lis[5]   # リスト変数の後に[n]をくっつけると、そのリストのn番目の中身を得られる。
                  # ただし、lis[0]が最初の中身を、lis[1]が次の中身を…という順番になっているので
                  # (nは1からでなく0からはじまる)、この場合は5でなく6が答えられる。

Out[84]: 6

In [85]: lis[0]   # [0]が先頭の要素を指す
```

リストは変数を並べたものと考えることもできる。従って、次の例のように、**中身を上書き**することができる。

```
In [91]: lis[5] = "piyo"  # 5番目の要素を文字列"piyo"に上書きする
         print(lis)

         [1, 2, 3, 4, 5, 'piyo', 7, 8, 9]
```

また、リスト名に続いて「.pop(n)」と書くことで n 番目の要素を**削除**したり、「.append(x)」と書くことで、リストの最後に x を**追加**したりできる。

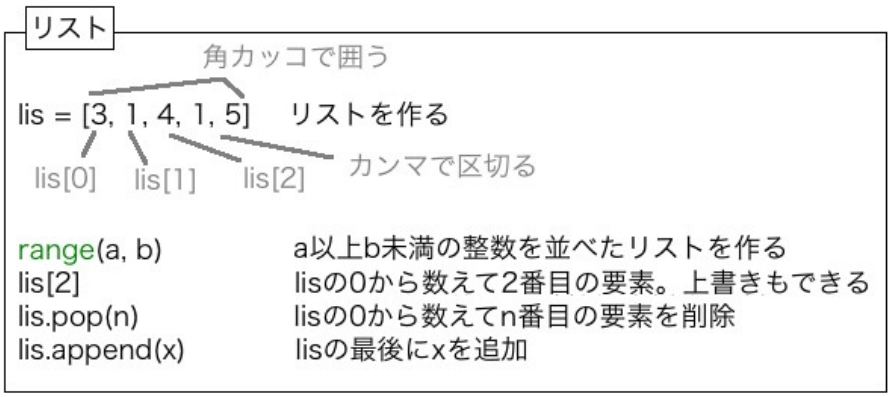
```
In [92]: lis.pop(4)       # 4番目の要素を削除する
         print(lis)

         [1, 2, 3, 4, 'piyo', 7, 8, 9]

In [93]: lis.append("hoge")   # リストの最後に文字列"hoge"を追加する
         print(lis)

         [1, 2, 3, 4, 'piyo', 7, 8, 9, 'hoge']
```

・まとめ



・課題

(1) 変数 lis1 にリスト[“a”, “b”, “c”]を代入せよ
(2) range 命令を使って、変数 lis2 に[5, 6, 7, 8, 9]を代入せよ
(3) かけ算を使って、変数 lis3 に[1, 1, 1, 1, 1]を代入せよ
(4) lis2+lis3 を実行せよ。リスト同士の足し算は、文字列と同じで連結になる。
(5) range 命令を使って、変数 lis4 に
　　[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
　を代入せよ。
(6) lis4 の 7 と 10 を上書きして 0 にせよ。
　　[1, 2, 3, 4, 5, 6, 0, 8, 9, 0, 11, 12, 13, 14, 15]
(7) 次に 4 と 9 と 13 を pop で削除せよ。
　　[1, 2, 3, 5, 6, 0, 8, 0, 11, 12, 14, 15]
(8) 次に、append を使って最後に 16 と 17 をくっつけよ。
　　[1, 2, 3, 5, 6, 0, 8, 0, 11, 12, 14, 15, 16, 17]

1-6.　for ループ

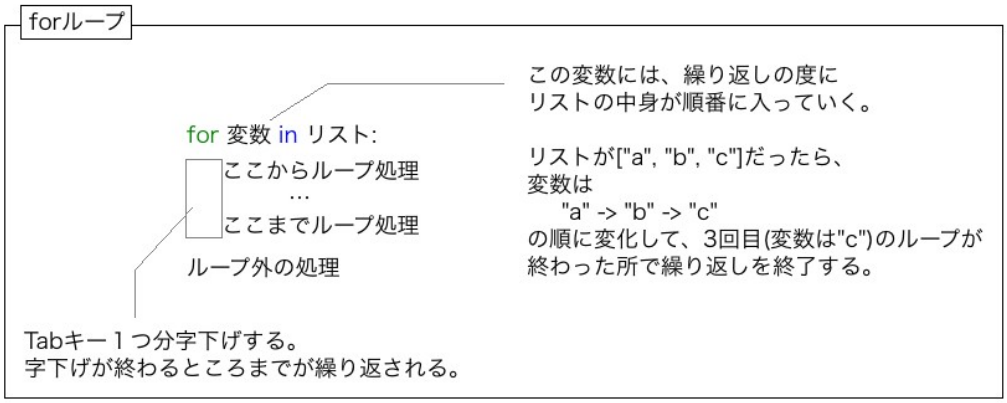
　ここからが本当のプログラミングだ！
for ループは、リストの中身を初めから順に 1 個ずつ眺めていくときに使う。
例えば次のプログラムは、range(0, 10)、つまりリスト[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]を初めから順に 1 個ずつ出力（print）するプログラムである。

```
In [94]: for i in range(0, 10):
             print(i)

0
1
2
3
4
5
6
7
8
9
```

for ループを使うことで「繰り返し」を作ることができる（上の例だと 10 回の繰り返し）。コンピュータは一秒間に何百万回もの繰り返しを行えるので、これは非常に強力な道具となる。

（1）for ループの書き方



（２）for ループの使用例

説明を聞くよりも実際の使用例を見たほうが速いだろう。次の３つの例を入力して実行してみよう。

```
In [95]: for i in range(0, 10):
             print(i, i*i)       # 繰り返す度にiとi*iを両方出力する
```

```
(0, 0)
(1, 1)
(2, 4)
(3, 9)
(4, 16)
(5, 25)
(6, 36)
(7, 49)
(8, 64)
(9, 81)
```

※print 命令のカッコの中に、カンマで区切って複数のデータや変数を書くと、この例のようにそれを全て出力してくれる。

```
In [96]: for st in ["apple", "orange", "banana"]:  # 変数stには、リスト内の各文字列が順番に格納される
             print(st, len(st))                    # 文字列stとその長さを両方出力する
```

```
('apple', 5)
('orange', 6)
('banana', 6)
```

```
In [97]: a = 0
         for i in range(0, 100):  # iは0, 1, ... 99と変化する。
             a += i               # 繰り返しの度にaにiが足されるので,
         print(a)                 # 繰り返し終了後にはaには0から99までの和が記憶される
```

```
4950
```

```
In [110]: lst = []                # []は何も入っていないリストを表す
          for i in range(0, 10):
              lst.append(i*2)     # 繰り返す度にi*2をリストの最後に追加していく
          print(lst)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

・課題

for ループを使うと出来ることが格段に増える。課題も難しくなってくるが、がんばろう。

(1) 前ページの最後の例を参考にして次のリストを作れ。
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

(2) 次のものを出力せよ。
（”a”*5 と書くと”aaaaa”となることを思い出そう）

```
a
aa
aaa
aaaa
aaaaa
aaaaaa
aaaaaaa
aaaaaaaa
aaaaaaaaa
aaaaaaaaaa
```

(3) 階乗
10 の階乗（1*2*3*. . . *10）を求めよ。
（3628800 が正解。0 になったり 362880 になる人は range の範囲を気をつけるべし）

(4) １？

$\frac{1}{2^1}+\frac{1}{2^2}+\frac{1}{2^3}+\frac{1}{2^4}+\cdots$ を計算せよ。繰り返すほど 1 に近づいていくことがわかる。1/2 などの計算の時、整数を整数で割ると整数になってしまうのに注意。少なくともどちらか片方は小数型にすべし。

(5) 円周率

$\left(6*\left(\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\frac{1}{5^2}+\ldots\right)\right)^{0.5}$ を計算せよ。

これは繰り返すほど円周率に近づいていく！

(6) フィボナッチ数列
append を使いながら次のようなリストを作れ。これは直前の二つを足した数が次の数になるような数列（フィボナッチ数列）である。
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]

1-7. 乱数

乱数とは、サイコロの目やおみくじの結果のような、ランダムな結果を返すもののことである。python では最初からは乱数が使えないようになっているので、まずは次の命令を実行して乱数を使えるようにしよう。
（動作を軽快にするために、最初には必要最小限の命令しか使えないようになっているのである。一度 import を実行すれば、それから後は使用可能となる。）

```
In [112]: import random
```

さて、これで乱数が使用可能となった。ここでは a 以上 b 未満のランダムな小数を答える random.uniform(a, b)という命令を学ぶ。

```
In [113]: random.uniform(0, 1)

Out[113]: 0.24743845391333164

In [114]: random.uniform(0, 1)

Out[114]: 0.46231589053557565

In [115]: random.uniform(0, 1)

Out[115]: 0.42900730329991044
```

random.uniform(0, 1)は 0 以上 1 未満のランダムな小数を答える。上の例のように、実行する度に結果が変わることが分かるだろう。コンピュータゲームでは乱数は敵の動きなどを決めるために無くてはならない存在である。

・課題
(1) 0 以上 1 未満のランダムな小数が 100 個並んだリストを作れ
(2) 0 以上 100 未満のランダムな**整数**が 100 個並んだリストを作れ
　小数を整数に切り捨てるには、小数の手前に(int)と書く
　　例：　a = (int)(3.1415)　　# a は 3 になる

1-8.　if 文と論理式

「もしＡならばＢせよ」というような処理を実現するのが if 文である。

（１）論理式

「もしA」の部分ではＡが正しいか間違ってるかが判断出来る必要があり、このようなＡのことを論理式という。次の例のように、論理式の結果は「True（正しい）」か「False（間違ってる）」で返される。
次の例では、最初の論理式「1 > 2」のみ間違いで、残りは正しい。

```
In [150]: print(1 > 2)                    # 1が2より大きい
          print(1 <= 2 <= 3)              # 1≦2≦3
          print("aaa" != "bbb")           # 文字列"aaa"と"bbb"が等しくない
          print("a" == "b" or 1 != 2)     # "a"と"b"が等しい、または、1と2が等しくない
          print("c" != "d" and 1 < 2)     # "c"と"d"が等しくない、かつ、2が1より大きい

          False
          True
          True
          True
          True
```

論理式として使えるのは、主に「数の大小の比較」「文字列や数が等しいかどうかの比較」である。二つの論理式があって、どちらも等しい(and)か、少なくともどちらかが正しい(or)かを判断させることもできる。

論理式の記号

X == Y :等しい
X != Y :等しくない
X < Y :XがYより小さい
X <= Y :XがY以下
X > Y :XがYより大きい
X >= Y :XがY以上

論理式A and 論理式B :論理式Aが正しく、かつ、Bも正しい
論理式A or 論理式B :論理式Aが正しい、または、Bが正しい

（2）if 文

　if 文は、論理式の真偽にもとづいて、行う処理を変えるための構文である。
次の例では、変数 r に 0~100 の乱数を代入し、それが 30 未満ならば「小さい」と、30 以上 50 未満ならば「少し小さい」、50 以上 70 未満ならば「少し大きい」、それ以外（70 以上）ならば「大きい」と表示するプログラムである。

```
In [157]: r = random.uniform(0, 100)
          print(r)
          if r<30:
              print("小さい")
          elif r<50:
              print("少し小さい")
          elif r<70:
              print("少し大きい")
          else:
              print("大きい")

          35.1722561863
          少し小さい
```

「if」「elif」「else」というのが if 構文のキーワードである。elif は前の論理式が間違ってる時に、次の論理式を提示する。else ブロックにはそれまでの論理式全てが間違ってる時に実行される処理を書く。

elif ブロックはたくさんつなげても大丈夫だし、逆に elif や else が無くても良い。

（3）for ループと if 文と break

次のプログラムは、1 以上 15 未満の数で、3 で割ったあまりが 1 のものだけを print する。if や for が重なるときは、その数だけ字下げが増える。字下げを間違えるとちゃんと動かないので注意。

```
In [158]: for i in range(1, 15):
              if i%3==1:
                  print(i)          # forの中にifがあるので、字下げは Tab 2つ分になる
```

```
1
4
7
10
13
```

ところで、「特定の条件を満たしたら、途中でも for ループを終わらせたい」という時がある。このようなときには break という一行を書く。

```
In [164]: for i in range(1, 1000000):  # breakがないと百万回繰り返すことになる
              if i**3>300:             # iの3乗が300より大きかったら
                  break                # ループを打ち切る
              print(i**3)
```

```
1
8
27
64
125
216
```

・課題

(1) fizzbuzz

1 から順番に、3 で割り切れたら「fizz」、5 で割り切れたら「buzz」、3 でも 5 でも割り切れたら「fizz buzz」、そうでなければその数字をそのまま表示していくプログラムを作れ。出力は次のようになる。

```
1
2
fizz
4
buzz
fizz
7
8
fizz
buzz
11
fizz
13
14
fizz buzz
16
17
fizz
```

(2) 乱数

0 以上 1 未満の乱数を繰り返し生成して出力し、0.9 を超えたタイミングで繰り返しを打ち切るようなプログラムを書け。出力例は次のようになる。

```
0.0556852332567
0.714585195478
0.891537169237
0.157868147931
0.890927544645
0.368104709083
0.448074293021
0.618475945157
0.934780516201
```

```
0.790976137033
0.222702203139
0.915834371788
```

(3) 2 で割るやつ

変数 n に適当な整数を入力して、「n が 2 で割り切れたら 2 で割り、そうでなければ 1 を足す」という操作を n が 1 になるまで繰り返せ。例えば n=33 のとき左、n=50 から始めた時は右のようになる。

```
33
34
17
18
9
10
5
6
3
4
2
1
```

```
50
25
26
13
14
7
8
4
2
1
```

(4) 検索

まず、1 以上 5 以下の整数を 10 個並べたリストをつくる。このリストの中に 5 が含まれていたら、”Yes”を、含まれていなければ”No”を一行だけ出力せよ。

```
[1, 5, 5, 5, 2, 2, 1, 2, 2, 4]
No
Yes
Yes
Yes
No
No
No
No
No
No
```

失敗例

```
[3, 1, 4, 1, 3, 1, 1, 2, 4, 2]
No
```

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

成功例

(5) 最大値

まず乱数を 10 個並べたリストをつくる。次に、そのリストの中で最大の数を求めて出力するプログラムを作れ。

```
[30, 94, 40, 32, 85, 51, 41, 88, 51, 84]
94
```

```
[47, 77, 15, 2, 7, 98, 69, 94, 24, 68]
98
```

Part2. Python のテクニック

1. 多重ループ
2. 組(タプル)
3. 辞書型
4. 命令の作成と呼び出し
5. ファイルの入出力
6. 文字列処理
7. リストの応用

2-1. 多重ループ

ループの中にループを入れることができる。どこからどこまでがループかは、**字下げで区別される**。だから、下の例の print(i, j, a)は内側のループに入っているが、print(“”)は内側のループに入っておらず、外側のループには入っている。

print(“”)は空白の行の出力であり、下の出力の(1, 4, 4)と(2, 1, 2)の間などがこの行からの出力である。

出力は(i, j, i*j)が順に出力されている。i より j の方が先に増えて、j が最後まで行ったら i が 1 増えるというふうになっていることがわかる。これは i のループの内側に j のループが入っているからである。

```
In [10]: print("-------")              # ループの外
         for i in range(1, 5):        #  -外側のループ
             for j in range(1, 5):    #  |  -内側のループ
                 a = i*j              #  |  |
                 print(i, j, a)       #  |  -
             print("")                #  -
         print("-------")             #
```

```
-------
(1, 1, 1)
(1, 2, 2)
(1, 3, 3)
(1, 4, 4)

(2, 1, 2)
(2, 2, 4)
(2, 3, 6)
(2, 4, 8)

(3, 1, 3)
(3, 2, 6)
(3, 3, 9)
(3, 4, 12)

(4, 1, 4)
(4, 2, 8)
(4, 3, 12)
(4, 4, 16)

-------
```

i \ j	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12
4	4	8	12	16

・課題

(1) 九九の表に現れる数を全部足したら何になるか求めよ。

(2) 文字列比較

二つの文字列 s1 と s2 を用意する。s1 の中に s2 が含まれているか調べよ。
ただし、「文字列 s1 の n 文字目」は s1[n]で得ることができる。
たとえば

s1 = “abracadabra”

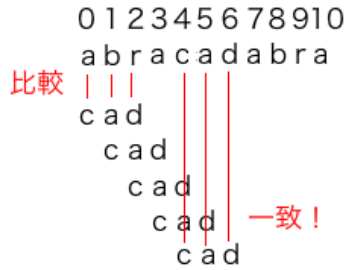
について s1[0]は”a”, s1[3]も”a”, s1[6]は”d”である。
このとき

s2 = “cad”

ならば s1 に含まれているので Yes と答え、

s2 = “dac”

ならば、含まれていないので No と答えよう。



ヒント：外側のループで s2 の 0 文字目を s1 の何文字目と合わせるかを決めて、内側のループで s2 の各文字について比較をし、全て一致してたら Yes。

(3) 素数判定

2 から 100 までの素数（1 とそれ自身以外で割り切れない数。2, 3, 5, 7, 11, ... ）を全て出力せよ。
まず、ある数 n が素数かどうかを判定する（2 以上 n 未満で n を割り切る数があるか調べる）部分を書いて（これは一重ループでできる）、その n を外側のループで 2 から 100 まで調べれば良い。

(4) 整列

まず乱数を 10 個ならべたリストを作り、それを大きい順に並び替えよ。

```
[63, 35, 78, 48, 93, 75, 34, 1, 63, 51]
[93, 78, 75, 63, 63, 51, 48, 35, 34, 1]
```

方針例：lst[0]から lst[9]までで一番大きい数を探して、それを lst[0]と交換する。次に lst[1]から lst[9]までで一番大きい数を探して、それは全体で 2 番目に大きい数となるので lst[1]と交換する…。これを最後まで繰り返せば大きい順に整列される。

2-2. 組（タプル）

タプルとは**要素の編集（上書きや追加など）ができない**リストのようなものである。リストの[ ]の代わりに( )を使う。

　　a = (“apple”, 120, 5)

と書くと、変数 a には”apple”, 120, 5 を組にしたタプルが代入される。

```
In [14]: a = ("apple", 120, 5)    # 「120円のりんごが5個」という情報をタプルにしてみる
         print(a[1])              # タプルではリストの角カッコ[]の代わりに丸カッコ()を使う
                                  # リストと同じように[n]をつけてn番目の要素を得る
120
```

```
In [15]: a[1] = 200     # 要素の変更はできない
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-920f2083c4ad> in <module>()
----> 1 a[1] = 200     # 要素の変更はできない

TypeError: 'tuple' object does not support item assignment
```

タプルをばらばらにして個々の変数にしたり、逆に変数たちをまとめて一つのタプルを作ったりできる（実はリストでもできる）

```
In [17]: a = (3, 5, 7)
         x, y, z = a          # 変数xに3, yに5, zに7が入る
         print(x, y, z)
         a = (x, z)           # aはx, zをくっつけたタプルになる
         print(a)
(3, 5, 7)
(3, 7)
```

タプルのリストを作ることもできる（実はリストのリストも作れるのだが）

```
In [18]: lst = []
         for i in range(0, 10):
             lst.append((i, i*i, i*i*i)) # タプル(i, i*i, i*i*i)をlstに追加
         print(lst)
[(0, 0, 0), (1, 1, 1), (2, 4, 8), (3, 9, 27), (4, 16, 64), (5, 25, 125),
(6, 36, 216), (7, 49, 343), (8, 64, 512), (9, 81, 729)]
```

じゃあリストがあるのにタプルを使う意味って何？ということになるが、

・リストよりも処理速度が速い

・辞書型のキーとして使うことができる（次章で述べる）

という利点はある。

従って、追加や削除や変更を頻繁に行わないようなデータの組を扱うときは、リストよりタプルを使うことが多い。このことから、ライブラリ（どこかのプログラマの作った便利な命令群）で含まれている命令では、タプルを受け取ったり答えたりするものがある。

2-3. 辞書型

辞書型というのは、今までのリストとかだと lst[3]のように数字で要素を指定していたけれど、数字の代わりに他のもので要素を指定できるようにしたものである。たとえば文字列 m[“apple”], タプル m[(1,2)], もちろん数字も m[3]。このように、要素の指定に使うものをキー（鍵）という。タプルはキーとして使えるが、リストを使うことは出来ない。

・辞書の作成

```
In [24]: d = {"python":100, "java":80, "ruby":50, "C":20}
         # 辞書を作るときは、このように「キー：値」の形式で書く。要素同士はカンマで区切る。
         # このときd["python"]は100, d["java"]は80, ... のようになる。
         print(d)
         print(d["java"])

         {'python': 100, 'ruby': 50, 'C': 20, 'java': 80}
         80
```

・辞書への追加

```
In [30]: d[(3,1)] = "hoge"      # 存在しないキーで代入しようとすると、
                                # 新たにそのキー：値の対応が追加される。
                                # ここでは追加するものとしてタプルをキーに、文字列を値にしてみた。
         print(d[(3,1)])

         hoge
```

・要素の削除

```
In [33]: del(d[(3,1)])          # キー(3,1)を削除する
         print(d[(3,1)])        # 存在しないというエラーが出る

         ---------------------------------------------------------------------------
         KeyError                                  Traceback (most recent call last)
         <ipython-input-33-697b7c2855e5> in <module>()
         ----> 1 del(d[(3,1)])          # キー(3,1)を削除する
               2 d[(3,1)]

         KeyError: (3, 1)
```

タプルと辞書については、しばらくしたらまた出てくるので、そのときに「そんなのもあったなあ」と思い出してくれれば良い。

2-4. 命令

（１）命令とは

命令とはよく行う処理をまとめて、簡単な名前で呼び出せるようにしたものである。今まで使ってきた

```
print, range, random.uniform
```

なども命令である。python では、このような命令を自分で作ることができる。命令は、「**引数**を受け取って、何かをして、**戻り値**を答える」という構造をしている。例えば下の例では、「二つの数 a, b を受け取って、どちらが大きいか比較して、大きい方を答える」larger という名前の命令を作っている。

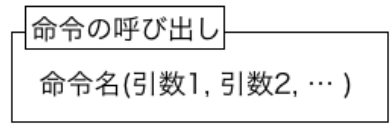
```
In [44]: def larger(a, b):        # largerという名前の命令をつくる。二つの引数を取り、それぞれa,bという変数になる
             if a>b:              # aがbより大きかったら
                 return a         # aを答える
             else:                # そうでなければ
                 return b         # bを答える

         print(larger(1, 3))
         print(larger(3.14, 2.71))
```

```
3
3.14
```

（２）命令を呼び出す

命令は、命令名の直後にカッコ()をつけて、その中に命令が受け取る引数たちを書くと呼び出すことができる。命令の戻り値は数のように使うことができる。



今まで使ってきた range 命令では、

```
range(1, 5)
```

と書いて命令を呼び出す時、1 と 5 が引数であり、戻り値は[1, 2, 3, 4]である。

（３）命令を作る

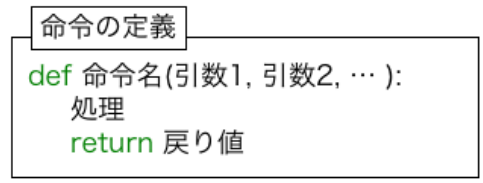
命令を作るには、def 構文を使う。

```
def hoge(a, b, c):
    ...
```

のように命令 hoge を作ったなら、これは 3 つの引数を受け取る命令となり、

```
hoge(1, 2, 3)
```

のように呼び出すことになる。こうすると a に 1, b に 2, c に 3 が代入された状態で処理が実行される。**引数や戻り値は無くても構わない**。また、if 文などと同じように、**字下げしているところが命令の範囲**となる。



（４）例

・次の命令 hello は、引数 n を受け取り、n 回だけ hello!と出力するだけのとてもシンプルな命令である。戻り値はない。

```
In [49]: def hello(n):
             print("hello!"*n)

         hello(1)
         hello(3)
         hello(10)
```

```
hello!
hello!hello!hello!
hello!hello!hello!hello!hello!hello!hello!hello!hello!hello!
```

課題：引数 n を受け取り、n が 2 で割り切れたら”偶数”，割り切れなかったら”奇数”と出力する命令を作れ。

・次の命令 mx は、引数としてリスト lst を受け取り、そのリストの中で最大の数を戻り値として答える命令である。このように、引数にはリストも与えることができる。

```
In [54]: def mx(lst):
             a = lst[0]
             for x in lst:
                 if a<x:
                     a = x
             return a

         print(mx([3,1,4,1,5]))
         print(mx([1,1,1,1,1]))
         print(mx(range(0, 100000)))
```

```
5
1
99999
```

課題：引数としてリスト lst と数 n を受け取って、lst に n が含まれていたら 1，含まれていなかったら 0 を戻り値として返す命令 search を作れ。

```
In [60]: lst = [3,1,4,1,5,9,2]
         print(search(lst, 5))
         print(search(lst, 6))
```

```
1
0
```

2-5.　ファイル入出力

　ここでは、他のファイルを読み込んだり、ファイルへ書き込んだりする方法を学ぶ。大きなデータベースを扱ったり、長大な出力結果を保存するためにはこの手法が必要となる。情報オリンピックの問題を解くときにも必要だったりする。

（１）ファイルを作る
　これから読み込むためのファイルを作る。まずデスクトップにある「notepad++」を開く。適当な文章を数行にわたって入力して、保存しよう。保存場所は指示するので聞いてほしい。

（２）ファイルを開く
　ファイルを扱うには、まず open 命令を使って python からファイルを読み書きできるようにする。次の例では「test.txt」という名前のファイルを「読み込みモード」で開いている。読み込みモードというのはファイルの読み出しだけを行なって、書き込みを行わないモードのことである。

```
In [67]: f = open("test.txt", "r")
```

この open 命令によって、f という変数が test.txt というファイルを表すようになった。このような f は「ファイルオブジェクト」といって、f に対して読み出しなどの処理を行うことができる。

（３）ファイルの読み出し
　ファイルを読む方法はいくつかあるが、ここでは for 文を使ったものを学ぶ。使い方は簡単で、for 構文のリストのところにファイルオブジェクトを持ってくると、for ループの繰り返しの度に、ループ変数にファイルの一行ずつが代入される。
　次の例は、左が元のファイル「test.txt」、右が出力結果である。出力が一行起きになっているのは、元のファイルに改行があったから出力にも改行が入り、print 命令がさらにもう一行改行を出力するからである。このような仕組みについては先の章でもう一度扱う。

```
hello world
python
3.141592
あいうえお
```

```
In [68]: for ln in f:
             print(ln)

hello world

python

3.141592

あいうえお
```

（４）ファイルへの書き出し

ファイルへの書き出しは、write 命令を使う。f.write と書くと、ファイルオブジェクト f に対して write 命令を行使するという意味になる。

次の例では test.txt を追記モードで(”a”)open している。追記モードでは、今までファイルに書いてあったものの後に続けて出力を行う。上書きモード(“w”)を使うと、今まで書いてあったものを削除して書き込みを行うこともできる。これを実行すると、左の内容だった test.txt が右のように変化する。つまり、

hello
hellohello
...

が追記されたのだ。

write 命令の引数となっている “hello”*i + “¥n” を説明しよう。

”hello”*i は今まで出てきたように、文字列 hello を i 回繰り返した文字列となる。続く+記号は文字列を連結する意味の+で、i 回繰り返した hello と文字列¥n をくっつけている。¥n とは実は「改行」を表す特殊な文字列であり、見て分かるように”¥n”という文字列そのものは出力に現れておらず、代わりにその位置で改行が行われる。

実は今まで使ってきた print 命令では、自動的に最後に改行が行われていた。ファイルへの書き出しで使う write 命令ではそれは行われないので、このように書いてやる必要がある。書かないと

hellohellohellohellohellohello...

となってしまう。

```
In [73]: f = open("test.txt", "a")
         for i in range(1, 5):
             f.write("hello"*i + "\n")
         f.close()
```

```
hello world
python
3.141592
あいうえお
```

→

```
hello world
python
3.141592
あいうえお
hello
hellohello
hellohellohello
hellohellohellohello
```

・まとめ

ファイル

```
ファイルを開く
f = open("ファイル名", "モード")

モード
  r : 読み込み
  w : 上書き(今までかかれてたものは消える)
  a : 追記(今までの後に続ける)

読む
for ln in f:    変数lnにはfの中身が一行ずつ代入される
    …

書く
f.write(文字列)      引数にした文字列が一行書き出される

処理終了時
f.close()
```

・課題

(1) random.txt というファイルに、0 以上 1000 未満のランダムな整数を 10000 行にわたって出力せよ。ここで、write 命令は文字列型しか引数に取らないことに注意。整数を出力したかったら、その整数を str 命令を使って文字列に変換する必要がある。例えば a が整数型の変数なら、
f.write(str(a) + “¥n”)のように書く必要があるだろう。

(2) 今作った random.txt を notepad++で開いてみよ。

(3) random.txt を読み込んで、すべての行に書かれている数の合計を求めよ。ここで、for ln in f: ... で ln に代入されるのは文字列型であることに注意。文字列 ln を整数に変換するには int(ln)と書く。

(4) 引数として整数 n をとり、random.txt のなかに整数 n が何回含まれているかを戻り値として答える命令を作れ。

2-6. 文字列処理

前のページで見たように、ファイル入出力命令は文字列型にしか対応していないので、数値データとか、リストとかをファイル入出力しようとしたら工夫が必要になる。ここではそのようなときに便利な命令を学ぶ。

（1）スライス

文字列の特定の部分だけを取り出したいとき、スライスと呼ばれる方法が使える。これは a という文字列変数に対し、a[x:y]と書くことで a の x 文字目から y-1 文字目までの部分文字列を取り出す方法である。次の例を見てもらえばわかると思う。文字列だけでなくリストやタプルに対しても使うことができる。

```
In [106]: a = "abcdefg"
          print(a[1:3])              # aの1以上3未満番目の文字列
          print(a[4:])               # aの4番目以降の文字列
          print(a[:3])               # aの3番目未満の文字列

bc
efg
abc
```

```
In [108]: b = [3,1,4,1,5,9,2]
          print(b[:4])               # リストでもできる

[3, 1, 4, 1]
```

（2）split 命令

split 命令は、

“1 2 3 4 5”

“aaa, bbb, ccccc, dd”

のように、空白やカンマといった特定の文字列での区切りがある文字列を、

[“1”, “2”, “3”, “4”, “5”]

[“aaa”, ”bbb”, ”ccccc”, ”dd”]

といった**文字列のリストに分解**する命令である。具体的には次の例のように使う。

```
In [111]: s = "1 2 3 4 5"
          print(s.split(" "))   # split命令のカッコの中には区切りの文字列を入れる

          s = "aaa,bbb,ccccc,dd"
          print(s.split(","))   # こっちはカンマで区切る

          s = "python...C...java...fortran"
          print(s.split("...")) # 区切りが長い文字列でも問題ない

['1', '2', '3', '4', '5']
['aaa', 'bbb', 'ccccc', 'dd']
['python', 'C', 'java', 'fortran']
```

（３）フォーマット

たとえば変数 x に身長、y に体重を表す数値が入っていたとして、

“あなたの身長は x cm で、体重は y kg です”

といった文字列を作りたいとする（x, y はそれらの変数が表す数にしたい）。文字列の連結を使えば

“あなたの身長は”+x+”cm で、体重は”+y+”kg です”

のように書けるのだが、もう少し見やすく書く方法がある。

```
In [115]: x, y = 170, 50
          s = "あなたの身長は%d cm, 体重は%d kgです" % (x, y)
          print(s)

          あなたの身長は170 cm, 体重は50 kgです
```

このような書き方をフォーマットを使った文字列という。ポイントは、2 行目の文字列中にある「%d」と、ダブルコーテーションの後ろにある「%」と、その右にある x と y のタプルである。

%d は整数が入る位置を表す特殊文字である。小数や文字列のときは次のものを使う。

%d 整数
%f 小数
%s 文字列

%d に入るべき整数を与えるのが、行末の(x, y)である。文字列中にある%d などの個数だけ、その位置に入れるデータを順番通りにタプルにして与える。ただし%d が 1 つだけならばタプルでなくて良い。このタプルと文字列との間は一文字の%で区切る。

いくつか例を見てみよう。

```
In [116]: a, b = 10.0, 3.0
          print("%f/%f=%f" % (a, b, a/b))     # 最後の%fには式a/bを与えている

          10.000000/3.000000=3.333333

In [117]: s = "hello"
          print("%s%s%s%s%s" % (s,s,s,s,s))  # %dなどは連続してててもいい

          hellohellohellohellohello

In [118]: lst = [1,2,3,2,1]
          print("lstの中身は%s" % lst)        # リストを%sに与えると自動的に文字列に変換される

          lstの中身は[1, 2, 3, 2, 1]
```

（４）その他

有用な文字列操作の命令として、次のようなものもある。

・strip 命令：先頭と末尾の空白、改行の削除

ファイルからデータを一行読んだ時に、末尾に空白が入ったままだったりすると厄介なので strip 命令で処理することが多い。

```
In [123]: s = " qwerty \n"    # この文字列は先頭と末尾に空白や改行が入っている
          print(s+s+s)        # 連結すると、改行や空白付きで出力される

          s2 = s.strip()      # strip命令は、文字列の先頭と末尾の空白や改行を抹消する
          print(s2+s2+s2)     # qwertyqwertyqwertyと連続で出力される


          qwerty
          qwerty
          qwerty

          qwertyqwertyqwerty
```

・replae 命令：文字列の置き換え

文字列中に登場する特定の文字列を、別の文字列に置き換える。特定の文字を消したりするのにも使える。

```
In [130]: s = "C言語は最高のプログラミング言語だからみんなもっとC言語を使うべき。C言語万歳！C言語万歳！"
          print(s)
          print(s.replace("C言語", "python"))    # "C言語"を"python"に置き換える

          C言語は最高のプログラミング言語だからみんなもっとC言語を使うべき。C言語万歳！C言語万歳！
          pythonは最高のプログラミング言語だからみんなもっとpythonを使うべき。python万歳！python万歳！
```

```
In [129]: s = "'a','b','c','d'"
          print(s)
          print(s.replace("'", ""))    # シングルコーテーション「'」を消す(無に置き換える)

          'a','b','c','d'
          a,b,c,d
```

・課題

(1) str 命令を使うとリストを文字列に変換できる。たとえば
str([1,2,3,4,5])は”[1, 2, 3, 4, 5]”とう文字列に変換される。このような元々整数のリストだった文字列が引数として与えられた時、元のリストを復元して戻り値として返す命令を作れ。

方針：引数 s = “[1, 2, 3, 4, 5]”とする。
まずスライスで両側の[]を除いた部分を取り出す。
“1, 2, 3, 4, 5”
split でカンマで分割する。文字列のリストが出来る。
[“1”, “ 2”, “ 3”, “ 4”, “ 5”]
strip で、2,3,4,5 の手前に入ってる空白を消す。
[“1”, “2”, “3”, “4”, “5”]
int で整数に変換する(int(文字列))。
[1, 2, 3, 4, 5]

この命令ができると、リストのデータを文字列にしてファイルに保存して、後でそれをまた読みだす、といったことができるようになる。

(2) 日本人の苗字が多い順に書かれたファイルがあるので、苗字が与えられた時にそれが何位か答える命令を作れ。使うファイルなど詳しくは当日指示します。

(3) 2008 年情報オリンピック予選問題 1 を解け。
http://www.ioi-jp.org/joi/2008/2009-yo-prob_and_sol/2009-yo-t1/2009-yo-t1.html

2-7. リストの応用

（１）簡単なリスト生成

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]というリストを作りたかったら

```
In [140]: lst = []
          for i in range(0, 10):
              lst.append(i*2)
```

と書けばいいのだが、じつはもっと簡単に

```
In [141]: lst = [i*2 for i in range(0, 10)]
```

で生成することができる。

[... for ... in ... if ...]

の形を使うことが出来る。次の例を見れば使い方は大体わかるだろう。

```
In [143]: print([10-i for i in range(0, 10)])                          # [10, 9, ..., 1]
          print([i for i in range(0, 10) if i%2==0])                   # 0以上10未満の偶数からなるリスト [0, 2, 4, 6, 8]
          print([(i, i*i) for i in range(1, 1000) if i*i<300])         # [(1, 1), (2, 4), (3, 9), ... (17, 289)]
          print([[i*j for i in range(1, 10)] for j in range(1, 10)])   # 九九の表

[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
[0, 2, 4, 6, 8]
[(1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81), (10, 100), (11, 121), (12, 144), (13,
169), (14, 196), (15, 225), (16, 256), (17, 289)]
[[1, 2, 3, 4, 5, 6, 7, 8, 9], [2, 4, 6, 8, 10, 12, 14, 16, 18], [3, 6, 9, 12, 15, 18, 21, 24, 27], [4, 8, 12, 16, 20,
24, 28, 32, 36], [5, 10, 15, 20, 25, 30, 35, 40, 45], [6, 12, 18, 24, 30, 36, 42, 48, 54], [7, 14, 21, 28, 35, 42, 49,
56, 63], [8, 16, 24, 32, 40, 48, 56, 64, 72], [9, 18, 27, 36, 45, 54, 63, 72, 81]]
```

（２）多重リスト

上の九九の表のように、リストの中にリストを入れたものを作ることができる。

A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

という多重リストは、

1 2 3
4 5 6
7 8 9

のような表だと考えると分かりやすい。このときA[1]は[4, 5, 6]というリストを差しており、A[1][2]のようにさらに[n]をくっつけることで一つの成分にアクセスできる。ちなみにA[1][2]は[4, 5, 6]の２番目なので６である。

A[0][0] A[0][1] A[0][2]
A[1][0] A[1][1] A[1][2]
A[2][0] A[2][1] A[2][2]

３重、４重のリストも作ろうと思えば作れる。