5　注意点

5.1　セミコロン;

次のようなプログラム

```
>>> a = 5
>>> b = 3
>>> c = a + b
>>> c
8
```

は，各文を改行ではなくセミコロンで区切って

```
>>> a = 5; b = 3; c = a+b; c
8
```

のように書くこともできます。セミコロンを多用するとプログラムの流れが見づらくなるので，あまり使わないほうがよいかもしれません。

5.2　代入演算子

Python では等号=は右辺を左辺の変数に代入するときに用いるため，これを代入演算子 (Assignment operator) と呼びます。したがって，3=4 といった式は数学では，偽の命題ですが，Python ではそもそも意味がない文なのでエラーとなります。また，変数 a の値に 5 を加えたい場合は

```
>>> a = 3
>>> a = a + 5    # aの値を5増やす
>>> a
8
```

のようにします。2 行目の a=a+5 はおかしな主張に見えますが，これは = が代入する機能を持つことを考えればおかしくはありません。次を試してみましょう。

```
>>> a = 3
>>> b = a      # bはaの値である。
>>> b
3              # もちろんbの値は3
>>> a = 4      # aの値を変える
>>> b
3              # bの値は変わらない
```

ここでも，b=a によって「a の値」が b に代入されるわけですが，変数 b が変数 a を参照しているのではないことに注意しましょう。

変数を箱だと考えると想像しやすいです，a=3 によって a という名前の箱に数値 3 が入ります。b=a によって名前 b の箱に a の中身 3 が入ります。箱 a の中身を 4 に変えても箱 b の中身は変わりません。

前にも少し説明しましたが，変数の数値を増やしたり減らしたりする複合代入演算子 +=，-=があります。

```
>>> a = 4; a += 1; a
5
>>> a -= 1  #aの値を1減らす
4
```

5.3　インタラクティブシェルと print

インタラクティブシェルでは変数の中身を知りたければ，次のようにその変数名を書けばよかったのですが，ファイルから実行する場合には，出力するものは必ず print しなければなりません。例えば，

```
>>> a = 3
>>> a
3
```

とすると変数 a の数値 3 が表示されましたが，次のプログラムでは何も表示されません。

- ファイル名：**print03.py**

```
a = 3
a
```

実行結果の例

a の値を知りたければ print をします。

- ファイル名：**print04.py**

```
a = 3
print(a)
```

実行結果の例

```
3
```

6　様々なデータの形式

6.1　リスト (list)

リストとはいくつかのデータ (要素) の集まりです。まずはリストを作る事から始めましょう。

```
>>> a = ['Alice', 'Bob', 2,3,8]    # リストを定義して変数aに入れる
>>> a                              # aの内容を確認
['Alice', 'Bob', 2,3,8]
```

当然ですが，リストの型は list です。

```
>>> type(a)                        # aの型を確認
<class 'list'>                     # aの型はリスト
```

続いて，リストの成分を呼び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
```

上のようにリストの成分の番号は 0 から始まります。存在しない成分を呼び出すとエラーになります：

```
>>> 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

具体的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']
>>> a[:3]
['a', 'b', 'c']
>>> a[-4:]
['e', 'f', 'g', 'h']
```

次にリストを足す (+) とどうなるか試してみましょう：

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

結合されたリストが返されました。

リスト自身への要素の追加は append() という『メソッド』を使って行います：

```
>>> 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除く
13
>>> aa
 ['a', 'b', 'c', 'd', 'e', 'b', 23, 8, 12]      # 最後の13が無くなった
>>> aa.pop(3)     #aaから3番目の要素を取り除く
'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
```

連続する数字列からなるリストは次のようにして作ることもできます。

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

6.2 タプル (tuple)

タプルとはプログラミングや数学でしか聞かない言葉ですが，これは組または順序を持つ組を意味します。タプルもリスト (list) のように要素を集めたものですが，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
>>> a
(3, 7, 'abc')
>>> type(a)
<class '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 なデータです。変更してはいけない大事なデータを誤って変えないためにこのような量が用意されています。

6.3 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}
```

これは織田信長のプロフィールです。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：

```
>>> 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：

```
>>> a.keys()        # aの『鍵』の集まりを返す
['death', 'type', 'name', 'birth']
>>> a.values()      # aの『値』の集まりを返す
[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)
{1: 'One', 2: 'Two', 3: 'Three'}
```

6.4 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。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}
>>> a
{1,2,3,4}     # 2の重複したが無くなり，順番が変化している。
>>> a.add(5)       # aに5を付け加える
>>> print(a)
{1, 2, 3, 4, 5}
```

要素はソートされ，重複が除かれているのがわかります。関数 `set()` を使い，リストを集合に変換することもできます。

```
>>> a = [1,2,3]
>>> b = set(a); b
{1,2,3}
```

集合の演算 $\cup, \cap, \setminus$ などの演算も用意されています。

- ファイル名：**set01.py**

```
A = {1,2,3,4}
B = {3,4,5,6}
print(A | B)    # A∪B
print(A & B)    # A∩B
print(A - B)    # A\B
```

実行結果の例

```
{1,2,3,4,5,6}
{3,4}
{1,2}
```

集合そのものに加えたり，共通部分を取ったりする複合代入演算子`|=`，`&=`，`-=`があります。例えば，

```
>>> a = {1,2,3}; b = {3,4}
>>> a |= b    # 集合aに集合bの要素を加える。
>>> a
{1,2,3,4}
```

となります。

7　キーボードからのデータの取得

キーボードから入力した文字や数値に応じて，プログラムの結果を変化させる事を考えます。キー入力を取得するには input() という関数を使います。次のファイルを作って実行してみましょう：

- ファイル名：**input1.py**

```
1 namae = input('あなたの名前を入力してください。：')
2 print('こんにちは', namae, 'さん')
```

上で作ったファイルを実行すると，次のような表示が出ます：

```
1 > python input1.py
2 あなたの名前を入力してください。：
```

そこで，信州花子と入力して Enter キーを押せば

```
1 こんにちは 信州花子 さん
```

と出力されます。上のプログラムは次の手順で実行されました：

1.「あなたの名前を入力してください。：」と画面に表示
2. 変数 namae を作成。キーボード入力を待つ。
3. 入力された文字列を変数 namae に代入
4.「こんにちは・・・さん」と画面に表示

input() で入力された文字列は変数 namae に格納されるわけです。

実は，入力するものが数値である場合には上の手順では不十分です。Python のプログラムは，入力されたものが数値であること判断できず，常に文字列として扱うからです。数値にするには，int 関数か float 関数を使って，文字列としての数を数値データに変換する必要があります。次はキー入力を受け付けて，入力した数値の 2 乗を出力するプログラムです。

- ファイル名：**input2.py**

```
1 aa = input('数値（整数）を入力してください：')     # aaは入力された文字列になる。
2 num = int(aa)          # ここで文字列を整数(int型)に変換
3 print('入力された数値の2乗は', num**2, 'です。')
```

これを実行すると次のようになります：

```
1 > python input2.py
2 数値(整数)を入力してください：9
3 入力された数値の2乗は 9 です。
```

上のプログラムで，入力を 1.5 などの整数以外にするとエラーとなります。少数点数を扱いたい場合は，入力された文字列を float 関数を使って浮動小数点数に変換します。

上の二つを合わせて次のような BMI を計算するプログラムを作ってみましょう：

- ファイル名：**bmi1.py**

```
1 namae = input('あなたの名前を入力してください：')
2 shintyo = input('あなたの身長は何センチですか？：')
3 shintyo = float(shintyo)  # shintyoを浮動小数点数に変換
4 weight  = input('あなたの体重は何キログラムですか？：')
```

```
5 weight = float(weight)     # weightを浮動小数点数に変換
6 bmi = 10000*weight/(shintyo**2)
7 print(namae, 'さんのBMIは', bmi, 'です。')
```

プログラムを実行して自分の BMI を計算してみましょう。

8　論理型と比較演算子

数学では，正しい命題は真，間違った命題は偽であるといいます。Python でも真偽値というものがあり，条件が正しいかどうかで真・偽の値が決まります。これらは条件に応じて処理を変化させるときに使います。

Python では真を True，偽を False で表します。これらは予約語であり特別な意味を持ちます。文法に注意しながら，インタラクティブシェルでの例を見てください：

```
1 >>> a = 3       #aに3を代入
2 >>> a == 3     #aは3だろうか？
3 True           #a==3は正しい！
4 >>> a == 2     #aは2だろうか？
5 False          #a==2は正しくない
6 >>> a > 2
7 True
8 >>> 3 != 2     #3は2に等しくないだろうか？
9 True           #3と2は異なるので，上の条件は真
```

この例のように，二つの数値を比較して真か偽かの条件を調べる事ができます。上の例で『==，>，!=』という記号を用いました。これらはデータを比較するときに用いるので**比較演算子**と呼ばれています。数値に対して使える使える比較演算子には次があります：

数値に対して使える比較演算子

==　!=　<　>　<=　>=

これらは，それぞれ伝統的な数学記号 $=$，$\neq$，$<$，$>$，$\leq$，$\geq$ に対応しています。

不等号と等号の順番は英語の”less than or equal” の順番と同じだと憶えましょう。『!=』は”not equal” と憶えます。

True と False には論理演算 and，or，not を行うことが出来ます。

```
1 >>> a = True       # aをTrueとする
2 >>> b = False      # bをFalseとする
3 >>> a and b       # aかつbは？
4 False
5 >>> a or b        # aまたはbは？
6 True
7 >>> not a         # aの否定は？
8 False
```

不等号は次のように連続して使うことができます。

```
1 >>> a = 4
2 >>> 3 < a < 5
3 True
```

上の 2 行目は 3<a and a<5 と同じ意味の文です。

リストや集合などコレクションに対して使える比較演算子もあります：

リストや集合などに対して使える比較演算子

```
==　　!=　　in
```

==と!= の説明は不要でしょう。in は次のように用います：

```
>>> a = [3,6,5,1]　　#aをリスト[3,6,5,1]とする
>>> 5 in a　　#5はaの中にいるだろうか？
True
>>> 7 in a　　#7はaの中にいるだろうか？
False
```

次のようにして，型を調べる type と組み合わせて使うことも出来ます：

```
>>> a = [3,5,6,1]
>>> type(a)
<class 'list'>
>>> type(a) == list
True
>>> type(a) == tuple
False
```