




Python入門 第3回

総合科学研究会

2024.4.28



今回の内容

- 第1回～第2回の復習
- 条件文について

復習・変数とは

変数とは値を一時的に保存しておく場所のこと。

変数には名前を付けることが出来る。

変数に値を代入する際は=を使う。

変数xを宣言し
xに10を代入

xに $x+3=10+3$
を代入($x=13$)

xに $x*x=13*13$
を代入($x=169$)

`x = 10`

`x = x + 3`

`x = x * x`

`print(x)`

169

復習・関数とは

一連の手続きをまとめたものを関数という。

別の言い方をすればある特定の処理をする機能が関数であるとも言える出来る。

Pythonでは絶対値を返すabs(x)関数やprint関数等が標準で使うことが出来る。

(自ら関数を定義して使う事も出来る、今後触れる予定)

メソッドとは関数のうちある特定のデータ型(整数データ、浮動小数点データ、文字列データ、etc)にのみ付随する関数というイメージ

復習・モジュール

Pythonにおいてはモジュールというものを入れる事で色々な関数を使うことが出来る。

例えばmathモジュールを導入すると三角関数や対数関数等を使うことが出来たり、matplotlibモジュールを導入すると簡単にグラフを描画出来たりする。

モジュールを導入するときにはコードの先頭に**import** (モジュール名)と書いておく必要がある。また、モジュールの関数を呼び出す際はモジュール名.関数名という記法を使う。

✓
0 秒

```
[5] import math  
     math.sqrt(2.0)
```

```
1.4142135623730951
```

条件分岐

プログラムの制御構造について

プログラミング言語を使う目的

→コンピュータを制御(操作)する事！

コンピュータの操作に必要な制御構造は主に以下の3つからなる。

逐次処理：ソースコードの上の行から下の行へ向かって処理をする構造

分岐処理：ある特定の条件を満たす時に別の処理をする構造

反復処理：一定範囲内の処理を条件を満たしている間繰り返す処理構造

今回は、分岐処理を扱います。

比較演算子

数学での等号(=)・不等号(>, <)のように値を比較する演算子を比較演算子という。
Pythonでは比較演算子を用いてどちらの値が大きいかどうかを評価できる。

- 例: 数値変数をA, Bとすると、Pythonでは以下のように比較演算子が使われる。

$A > B$ (AがBより大きい?)

$A < B$ (AはBより小さい?)

$A == B$ (AとBは等しい?)

$A \geq B$ (AはB以上?)

$A \leq B$ (AはB以下?)

$A \neq B$ (AとBは異なる?)

プログラミング言語において等式は**==**
イコールが**2つ必要**なので注意

ブール型

比較演算子により値を評価した結果は、ブール型と呼ばれるもので返される。

ブール型とは、論理値の**真(True)**、**偽(False)**の2つの値からなる型で比較演算子が正しい場合(成り立つ場合)はTrue、そうでない場合はFalseを返す。

- 例：A=5,B=2,C=2とすると

$A > B \rightarrow 5 > 2 \rightarrow \text{True}$

$A < B \rightarrow 5 < 2 \rightarrow \text{False}$

$B == C \rightarrow 2 == 2 \rightarrow \text{True}$

$B != C \rightarrow 2 != 2 \rightarrow \text{False}$

$B >= C \rightarrow 2 >= 2 \rightarrow \text{True}$

文字列の比較

実は数値だけでなく、文字列についても比較することが出来る。
ただし、数値の比較とは異なる点がいくつかある。

文字列S,Tの比較評価において
S==TはSとTが全く同じ文字列であるかを評価する

不等号による比較評価では以下のような規則がある。

- 先頭の文字から比較する(abcd > bcd)
- 数字は0が最小、9が最大
- アルファベットはaが最小、zが最大かつ、大文字の方が小文字より小さい
- 数字はアルファベットよりも小さい($0 < 9 < A < Z < a < z$)

文字列の比較

文字列比較評価の例

"1234" < "5678" → **True**

"Programming" < "programming" → **True**

"python3" > "python" → **True**

"abc12" < "1abc2" → **False** (1 < a であるため)

"This" < "this" → **True** (T < t より)

"that" > "this" → **False** (a < i より)

条件文if

ある特定の条件を満たした場合にのみ処理したい時
→**if文**を利用する

Pythonのif文は右のような構造をしている。

条件文は比較演算のようなブール型(True,False)を返す式とする。

すると処理1、処理2、...は条件文を満たす時、すなわち条件文がTrueである時に実行されるようになる。

```
if 条件式:  
    処理1  
    処理2  
    ...
```

if 条件文とした後に:(コロン)を打つ
処理文の最初にスペース4つ分(もしくはTab)を空ける必要がある
この空白を**インデント**と呼ぶ

条件文の実行例

✓
0 秒

```
[3] x = 70  
    if x >= 60:  
        print("OK")
```

OK

x = 70の時
x >= 60はTrueなので
if文の中身が実行され、OKが表示される

✓
0 秒

```
[4] y = 30  
    if y >= 60:  
        print("OK")
```

y = 30の時
y >= 60はFalseなので
if文の中身は表示されず、OKが表示されない

else節

if文において条件文が実行されなかった場合の処理を設けたい場合、**else**を使う。

else節は右図のようにして、if文の処理行の後に、インデントを付けずにelse:と書く。

elseの中で行う処理についてもif文内と同じくインデントを付ける必要がある。

```
if 条件式:  
    処理1  
    処理2  
    ...  
else:  
    処理3  
    処理4  
    ...
```

if-else文の実行例

✓
0秒

```
[1] x = 40
    if x >= 60:
        print("OK")
    else:
        print("NG")
```

NG

x = 40の時
x >= 60はFalseなので
else文の中身が実行され、NGが表示される

✓
0秒

```
[2] y = 60
    if y >= 60:
        print("OK")
    else:
        print("NG")
```

OK

y = 60の時
y >= 60はTrueなので
OKが表示される(elseの中身は実行されない)

条件式について

if文の条件分岐判定を行う条件式は比較演算子以外にもある。

例えば、文字列メソッドの1つであるisdecimal()は、文字列が全て数字からなる場合にTrueを、そうでない場合にFalseを返す。これを条件式として組み込むことが出来る。

Trueの場合

```
[1] string1 = "1234"
    string2 = "123a321"
    if string1.isdecimal():
        print(string1,"is a number.")
    else:
        print(string1,"is not a number.")
```

1234 is a number.

Falseの場合

```
▶ string1 = "1234"
    string2 = "123a321"
    if string2.isdecimal():
        print(string2,"is a number.")
    else:
        print(string2,"is not a number.")
```

📄 123a321 is not a number.

elif節

$a > b$ ならAを実行し、 $a \leq b$ ならBを実行するという条件文はif-elseで表現可能。

```
if a > b:
```

```
    do A
```

```
else:
```

```
    do B
```

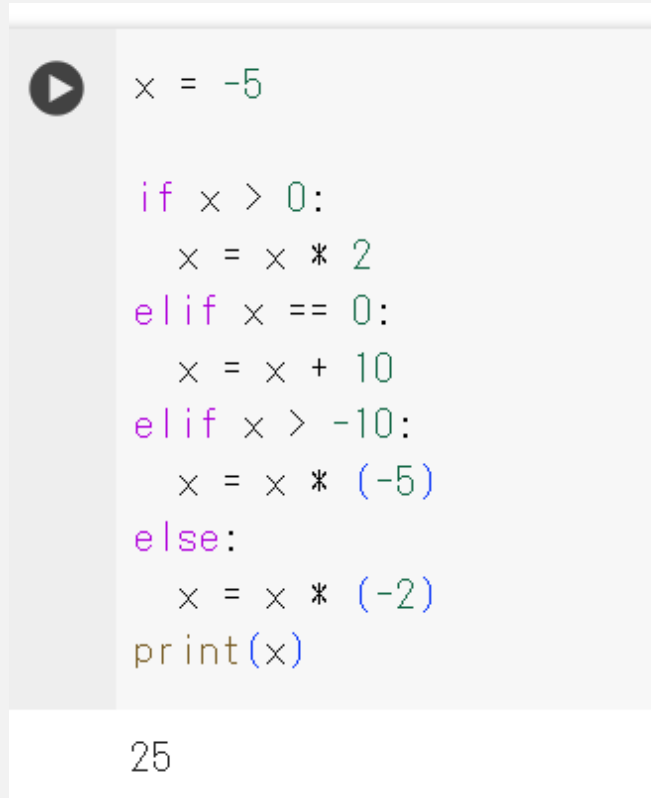
では $a > b$ ならAを実行し、 $a < b$ ならBを実行、 $a == b$ ならCを実行するという条件文を表現したい時は？

ここで登場するのが**elif**節である。

elif節

先ほどの $a > b$ ならAを実行し、 $a < b$ ならBを実行、 $a == b$ ならCを実行するという条件文をelifで記述すると以下のようなになる。

```
if a > b:  
    do A  
elif a < b:  
    do B  
else:  
    do C
```



```
x = -5  
  
if x > 0:  
    x = x * 2  
elif x == 0:  
    x = x + 10  
elif x > -10:  
    x = x * (-5)  
else:  
    x = x * (-2)  
print(x)
```

25

if文のネスト(入れ子構造)

if文の中にif文を入れる事も出来る。

こうした文を**ネスト(入れ子構造)**と呼んだりする。

入れ子構造にする事で、1つのif文では条件判定が難しい判定式を表現できる。

右のコードでは3つの変数が全て60以上である時にOKを返す。また60未満の値の変数が存在した場合、どの変数であったかを示すことが出来る。

```
[ ] x = 65
    y = 55
    z = 80

    if x >= 60:
        if y >= 60:
            if z >= 60:
                print("OK")
            else:
                print("NG(z)")
        else:
            print("NG(y)")
    else:
        print("NG(x)")
```

NG(y)