Days07

条件分岐：**if-elif-else**

基本形は，A,B,C が Bool 型（False(0) か．True(0 以外)）だとして，

```
pre_porcess

if A:    #  「もし」
    ProcA1
    ProcA2
elif B:  #「あるいはもし」
    ProcB
elif C:  #「あるいはもし」
    ProcC
else:    #「そうでなければ」
    ProcZ

post_process
```

となり，

> 「もし A が True なら ProcA，あるいはもし B が True なら ProcB，あるいはもし C が True なら ProcC，そうでなければ ProcZ」

elif は “else if”(「それ以外で，もし」) なので，「あるいは」が重要です．厳密に書くと，

> 「もし A が True なら ProcA，　もし $\neg A \wedge B$ が True なら ProcB，もし $\neg A \wedge \neg B \wedge C$ が True なら ProcC，そうでないなら ProcZ 」

となります．後ろの条件分岐はそれ以前の否定の上でおこなわれるということです．

A	B	C	
True	True/False	True/False	ProcA
False	True	True/False	ProcB
False	False	True	ProcC
False	False	False	ProcZ

セミコロンを打って，そのブロックはインデントします．
elif を使わないで if-else だけでも書くことができます．

```
pre_porcess

if A:
    ProcA
else:
    if B:
        ProcB
    else:
        if C:
```

```
            ProcC
        else:
            ProcZ

post_process
```

以下のような処理の場合はどう書けばいいでしょうか.

A	B	C	
True	True	True	ProcABC
True	True	False	ProcAB
True	False	True	ProcCA
True	False	False	ProcA
False	True	True	ProcBC
False	True	False	ProcB
False	False	True	ProcC
False	False	False	ProcZ

愚直に書くと

```
if A:
    if B:
        if C:
            ProcABC
        else:
            ProcAB
    else:
        if C:
            ProcCA
        else:
            ProcA
else:
    if B:
        if C:
            ProcBC
        else:
            ProcB
    else:
        if C:
            ProcC
        else:
            ProcZ
```

`else:if` は `elif` に書き換え可能です. 少し行数が減ります.

```
if A:
    if B:
```

```
        if C:
            ProcABC
        else:
            ProcAB
    elif C:
        ProcCA
    else:
        ProcA
elif B:
    if C:
        ProcBC
    else:
        ProcB
elif C:
    ProcC
else:
    ProcZ
```

行数は減りますが，どちらが可読性がよいかはビミョーですね．

Bool 型の A,B,C ですが，実際は，`if True` とか `elif False` とかはせず，True/False を答えとして返す演算がよく使われます．つまり，

- 比較演算子
  - `if A < B`
  - `if A >= B`
  - `if A == B`
  - などなど

がよく使われますが，数値だけではないですね．

- `if A is B`
- `if A is not C`

また，次回扱うような集合 A の包含関係なども使われます．

- `if a in A`
- `if b not in A`

また実は，0 か 0 以外の答えが出てくる演算でも構いません（つまりどんな演算でも構わないということ）．答えが 0 となるとき，False として扱われ，0 以外のときに True として扱われます．

- `if a - b`
- `elif a >> 2`
- `if 100`
- `else -3.14`

```
num = int(input())

if num > 100:
```

```
    print('100 < num')
elif 50 < num:                    # 左辺と右辺の左右はどうでもよい
    print('50 < num <= 100')
elif 0 < num <= 70:
    ```
    両辺で挟む形も OK.
    ちなみに（上の二つの条件の OR の否定）かつ（今回の条件）なので,
    ```
    print('0 < num <= 50')
elif num == 0:
    print('num == 0')
else:
    print('num < 0')
```

if-or-try: 例外処理，再び

例外処理の復習ですが,

```
c = a / b
print(c)
```

だと，b=0 が前もって実行されたら，「ZeroDivisionError」というエラーでプログラムが停止し，これを「例外」（SyntaxError 構文エラー以外）というのでした.
いつも停止するのは困るので，このときは例外処理で包む必要がありました.

```
try:
    c = a / b
except(ZeroDivisionError):
    print("b cannot be zero")
    c = a / 0.001

print(c)
```

この書き方，条件分岐みたいですね．以下のようにも書けそうです.

```
if b != 0:
    c = a / b
else:
    print("b cannot be zero")
    c = a / 0.001

print(c)
```

試しに実行して例外が発生したとき，if と try のどちらがいいでしょうか．良いか悪いかは個人の感覚の違いでしょうけれど，ベストな答えは「**if** でも書けるときは **if** で書く」です．言い換えると「**if** で書けないときは **try** で」.
つまり，例外が発生したときに，その例外を発生させない条件がわかるなら if 文で解決，条件がわからないなら try 文で逃げましょう.