

## クレジット:

UTokyo Online Education Education コンピュータシステム概論 2018 小林克志

## ライセンス:

利用者は、本講義資料を、教育的な目的に限ってページ単位で利用することができます。特に記載のない限り、本講義資料はページ単位でクリエイティブ・コモンズ 表示-非営利-改変禁止 ライセンスの下に提供されています。

<http://creativecommons.org/licenses/by-nc-nd/4.0/>

本講義資料内には、東京大学が第三者より許諾を得て利用している画像等や、各種ライセンスによって提供されている画像等が含まれています。個々の画像等を本講義資料から切り離して利用することはできません。個々の画像等の利用については、それぞれの権利者の定めるところに従ってください。



# Python の基本

この資料は [The Python Tutorial \(https://docs.python.org/3.6/tutorial/index.html#the-python-tutorial\)](https://docs.python.org/3.6/tutorial/index.html#the-python-tutorial) (日本語版 (<https://docs.python.jp/3/tutorial/>)) および [Python for Data Analysis:Wrangling with Pandas, Numpy and IPython \(http://shop.oreilly.com/product/0636920050896.do\)](http://shop.oreilly.com/product/0636920050896.do) を参考に作成した。

このチュートリアルでは他のプログラミング言語を習得している方を想定し、Python の特徴などを説明する。

## インデント(再掲)

Python では行頭の字下げ（インデント）でプログラムを構造化している。

同じインデントレベルの一連のプログラム文をコードブロックと呼び、インデントによる構造化は Python の特徴である。

他方、C, Java, Ruby といった他の言語では構造化にはおもに括弧 {}、（開始から終了までがコードブロック）が利用されている。

インデントはタブ、スペース(空白)いずれも使えるが、Python 標準ライブラリでは スペース 4文字を標準 (<http://pep8-ja.readthedocs.io/ja/latest/>)とするスタイル(書式)を採用している。

**Jupyter の標準もこれを探っており講義でもこの書式を利用する。** セルでタブを入力すれば勝手に変換されるのであまり気にする必要はない。

プログラムの例を使って説明する。

変数  $x$ ,  $y$  の両方が正符号をもつとき文字列を出力するプログラムは、if 文で以下のように書ける:

```
In [ ]: x = 1
        y = 1
        if x > 0 :
            print ("x has the positive sign, but y is unknown")
            if y > 0 :
                print ("Both 'x' and 'y' have the positive.")
```

3行目の if 文の行末の: はコードブロックの先頭を意味し、これに続く同じ字数だけインデントされた行の続く範囲までコードブロックは続く。そのコードブロックが直前の if 行の制御対象となる。5行目から二重の if 文となり、この制御範囲は、二重にインデントされた 6 行目の print()関数のみのコードブロックとなる。8 行目のインデントされていない print()行は、いずれの if 文の制御範囲ではなく、(x,y の符号にかかわらず) かならず実行される。

## 条件分岐 `if, elif, else`

`if` 文は最もよく知られる制御フロー文型であり、`if` と `:` の間の式を評価し、結果が `True` であれば、`:`以降のコードブロックを実行する。最も単純な `if` 文は以下のように書ける:

```
if x < 0:
    print("'x' is negative.")
```

`if` 文に続いて、一つあるいはそれ以上の `elif` 条件分岐ブロックや、全ての条件が `False` の場合実行される `else` ブロックを置くこともできる。

```
if x < 0:
    print("'x' is negative")
elif x == 0:
    print("'x' is zero")
elif 0 < x < 5:
    print("x is positive but smaller than 5")
else:
    print("x is positive and larger than or equal to 5")
```

また、`if` 文では、`if` あるいは `elif` いずれかの条件が `True` の場合以降の `elif` ~ `else` ブロックは実行されない。したがって、以下のプログラムでは、最初の `if True:` だけが評価される。

```
In [ ]: if True:
        print(x)
        elif x < 3:
            print ("x is less than 3")
        elif x < 2:
            print ("x is less than 2")
        elif x < 1:
            print ("x is less than 1")
        else:
            print(x)
```

### 練習問題

以下のプログラムは 3 行目、5 行目の `elif` は評価されない。すべての `elif` 文が評価対象となるように書き換えよ。

```
In [ ]: if x < 3:
        print ("x is less than 3")
        elif x < 2:
            print ("x is less than 2")
        elif x < 1:
            print ("x is less than 1")
        else:
            print(x)
```

## 分岐の評価

if 文に与える条件が or あるいは and で結合された複合条件の場合、条件は左から順に評価され、不要（以降の式を評価するまでもなく自明）な評価は省かれる。例えば or 演算子の場合、if a == 0 or b == 0: では、左辺, a == 0, が True の場合、右辺, b == 0, を評価することなく続くコードブロックが実行される。また、and の場合、if a == 0 and b == 0: では左辺が False の場合、右辺を評価することなくコードブロックはスキップされる。

以下のセルで示す2つの例のうち、最初のものは変数 x が未定義のためエラーとなるが、2つ目は正常に実行される。2つめの例では、複合条件のうち y > 5 を評価されることなく、変数 y が未定義にもかかわらずプログラムは正常に終了する。

```
In [ ]: x = 10          # del x のエラーを抑制するため
        y = 10

        del x          # x を未定義に

        if x > 5 or y > 5:
            print("'x' or 'y' is larger than 5")
```

```
In [ ]: x = 10
        y = 10          # del y のエラーを抑制するため

        del y          # y を未定義に

        if x > 5 or y > 5:
            print("'x' or 'y' is larger than 5")
```

## Ternary 式(参考)

Python では以下のように `if ~ else` 文を一行に書くこともできる。

```
sign = "positive" if x >= 0 else "negative"
```

これは、以下と等価である。

```
if x >= 0 :
    sign = "positive"
else:
    sign = "negative"
```

また、C, Java, Ruby の三項演算子 条件 ? 真:偽 と同等である。C の例では:

```
char *string = x > 0 ? "positive" : "negative";
```

## 繰り返し `for ~ in`

`for` 文の標準的な文法は以下のとおり。

```
for value in sequence:
    # 実行内容
```

`for` 文では `in` 以降に与えられたシーケンス型（リスト、文字列、あるいはタプル型など）にわたって反復をおこなう。実行順序はシーケンス中の要素の順番となる:

```
In [ ]: words = ["cat", "window", "deferenstrate"]
        for w in words:
            print(w, len(w))
```

## range() 関数

Python の for 文は、C、Java の for 文とは異なることに注意すること。C、Java では以下のように初期化、繰り返しステップと停止条件をユーザが定義できた：

```
for(int i = 0 ; i < 5 ; i++){
    printf("%d\n", i);
}
```

Python の for 文で繰り返し回数を指定する場合、range() 組み込み関数がいられる。range() 関数は引数の一つ、n、を与えられると、0, 1, ... n -1、という数列を生成する。上記 C プログラムは Python では以下のようになる：

```
In [ ]: for i in range(5):
        print(i);
```

range() 関数はリストオブジェクトを返さないことに注意する必要がある。リストオブジェクトを返すと、繰り返し回数の大きな for 文ではリストも大きくなり無駄が多い。リストオブジェクトが欲しい場合は以下のようにする：

```
In [ ]: list(range(5))
        # Return : [0,1,2,3,4]
```

## enumerate() 関数

for 文のシーケンス型にわたる繰り返し処理では、処理中の要素の順序を把握したいことはよくある。これまで説明した範囲では以下のように書ける：

```
In [ ]: i = 0
        for val in some_list:
            i += 1
            print(i, val)
        # くりかえさせたい処理
```

Python では enumerate() 関数が用意されており、上のプログラムは以下のように書き換えることができる：

```
In [ ]: for i, val in enumerate(some_list):
        # くりかえさせたい処理
```

## 帰属演算子 in

Python では for ループでリストを展開する in とは別に、リスト内の要素の有無を検査する in, not in 演算子が定義されている。以下のように、if 文の条件に in が出現した場合、for 文とは動作が異なるので注意すること:

```
In [ ]: colors = ["red", "green", "blue"]

print("for case:")
for color in colors:
    # do something
    print(color)

# color is "blue" as a result of the for loop


print("if case:")
if color in colors:
    # do something
    print(color)
```

## 繰り返し while

while 文ではコードブロックでは与えられた条件が False となるまで繰り返される。下記のプログラムでは、 $\sum_{x=0}^{100} x$  が total の値となる:

```
In [ ]: x = 0
total = 0
while x <= 100:
    total += x
```

## 練習

以下のプログラムでは 1 秒おきに print() 文が永遠に実行される。Jupyter-notebook で止まらない場合は、前に説明したとおり、 アイコン (Interrupt the Kernel) をクリックすれば停止させることができる。

以下のセルのプログラムを 10 秒後に while ループを終了するように書き換えよ。ただし、break 文を使わないこと:

```
In [1]: from time import sleep
```

```
while(True):  
    sleep(1)  
    print("Yeah!")
```

```
Yeah!  
Yeah!  
Yeah!
```

```
-----  
-----  
KeyboardInterrupt                                Traceback (most recent  
call last)  
<ipython-input-1-746afc312b14> in <module>()  
      2  
      3 while(True):  
----> 4     sleep(1)  
      5     print("Yeah!")
```

```
KeyboardInterrupt:
```

## break 文

break 文は for および while ループのコードブロックで利用可能である。break 文は実行中のプログラムで最も内側の繰り返し処理を中断し、ループを終了させる目的で利用される。下記のプログラムでは、colors リストの "black" 以降は処理されない:

```
In [ ]: colors = ["red", "green", "blue", "black", "white"]  
for c in colors:  
    if(c == "black"):  
        break  
    print(c)
```

## continue 文

continue 文は break 同様に、for および while ループのコードブロックで利用可能である。continue 文は実行中のプログラムで最も内側の繰り返し処理を中断し、次のループ繰り返しの処理を開始する。

下記のプログラムでは、colors リストの "black" は処理されないが "white" は処理される:

```
In [13]: colors = ["red", "green", "blue", "black", "white"]  
for c in colors:  
    if(c == "black"):  
        continue  
    print(c)
```

```
red  
green  
blue  
white
```



## 再び else(参考)

for および while 文ではelse ブロックを書くこともできる。このブロックは、ループの最後に一度だけ実行される。なお、for および while 文のelse ブロックの内容はcontinue で終了したときは実行される。一方、break でループを終了したときは実行されない。

```
In [ ]: colors = ["red", "green", "blue", "black", "white"]
        for c in colors:
            if(c == "black"):
                continue
            print(c)
        else:
            print("")
```

## pass 文

Python では空のコードブロックは許されていない。一方で、pass を利用した空白のコードブロックによって、プログラムが読みやすくなる場合がある。例えば以下の、if ~ elif ~ else プログラムはエラーとなる:

```
In [ ]: x = -1
        if x < 0:
            print("'x' is negative")
        elif x == 0:
            # blank block
        elif 0 < x < 5:
            print("x is poositive but smaller than 5")
        else:
            print("x is postive and larger than or equal to 5")
```

上は、pass 文を用いて、以下のように書き換えることで正常に実行される:

```
In [ ]: x = -1
        if x < 0:
            print("'x' is negative")
        elif x == 0:
            pass
        elif 0 < x < 5:
            print("x is poositive but smaller than 5")
        else:
            print("x is postive and larger than or equal to 5")
```