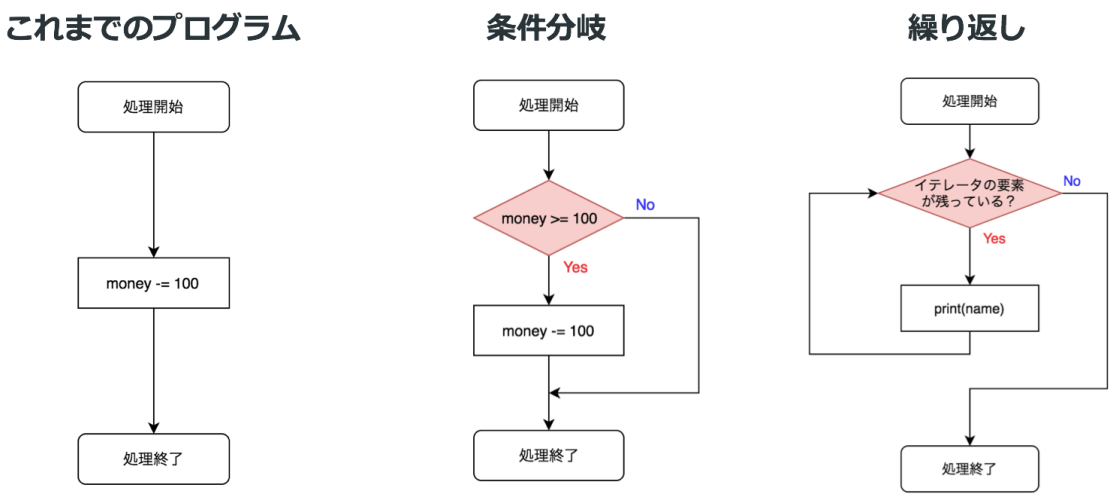
Tutorial Python vol.5

条件分岐と繰り返し

目次

本章では、プログラムの処理に**分岐**や**繰り返し**といった**制御構文**による動きを加えていきます。これを取り入れることで、**多彩な処理が可能になり、プログラミングの幅が広がります。**



5.1 条件指定の方法

条件分岐をするにあたって、その「条件」の書き方を覚える必要があります。まずは、その方法から学んでいきましょう。

5.1.1 bool型

bool型とは、**とある条件が満たされているかいないかを表すだけのデータ型**です。**True（真）**と**False（偽）**という2つの値だけを取ります。Pythonでは、条件式が満たされているときはbool型のTrueが、満たされていないときはFalseが返されます。

5.1.2 比較演算子

比較演算子とは、**条件式をつくるための記号**です。条件式とは、たとえば「2つの値は等しいか？」という式や「とある値は、もう一方の値より大きいか？」といったことを意味する記号です。適当な数値を定義して、それを題材に動作を確認します。また、比較演算子は以下のような種類があります。

- ==
- !=
- >
- <
- >=
- <=

In [3]:

```
# 適当な数値を定義
a = 1
b = 2
c = 3
```

"==" :「2つの値が等しいかどうか」

イコール記号を2つ並べます。このとき、**イコールをひとつだけにしないように気をつけてください。** Pythonではイコールを**変数の代入**に割り当てているため、比較演算子はこれと区別してイコールを2つ並べて表現しています。

In [4]:

```
# 2つの値が等しいかどうか
a == 1
```

Out[4]:

True

In [6]:

```
# 2つの値が等しいかどうか
b == 1
```

Out[6]:

False

"!=" :「2つの値が異なるかどうか」

逆に、**値が異なるかどうか**を確認したいときは、"!="を利用します。

In [5]:

```
# 2つの値が異なるかどうか
a != 1
```

Out[5]:

False

In [7]:

```
# 2つの値が異なるかどうか
b != 1
```

Out[7]:

True

">" or "<" :「左の値は右の値より大きいか or 小さいか」

大小関係を見るときは不等号を使用します。

In [9]:

```
# 左の値は右の値より大きいか
a > 2
```

Out[9]:

False

In [13]:

```
# 左の値は右の値より小さいか
a < 2
```

Out[13]:

True

">=" or "<=" :「左の値は右の値以上か or 以下か」

イコールも含める場合は、**不等号のあとにイコール**をつけて"≦"や"≧"を表現します。

In [10]:

```
# 左の値は右の値以上か
a >= 1
```

Out[10]:

True

In [12]:

```
# 左の値は右の値以下か
a <= 1
```

Out[12]:

True

5.1.3 論理演算子

論理演算子とは、**複数の条件を同時に指定したいときに使う記号**です。主に以下の3種類があります。

- and
- or
- not

and : 2つの条件がともに成り立っているかどうか

and演算子を利用すると、and演算子を挟んで並べた2つの条件式が**どちらも成り立っているときはTrue**、一方でも成り立っていないときはFalseが返されます。

In [18]:

```
# 2つの条件が同時に成り立っているかどうか
(a < b) and (b < c)
```

Out[18]:

True

or : 2つの条件のうち、少なくとも一方が成り立っているかどうか

2つの条件のうち、**どちらか一方だけでも成り立っている場合はTrue**が返されます。どちらの条件も成り立っていない場合にFalseが返されます。

In [20]:

```
# 2つの条件が少なくとも片方は成り立っているかどうか
(a < b) or (b > c)
```

Out[20]:

True

not : 条件が成り立っていないかどうか

not演算子を用いれば、とある条件がなりたって「いないかどうか」を返すことができます。通常の条件指定とは逆に、**条件が満たされていなければTrue**が、満たされていればFalseが返ってくるようになります。

In [21]:

```
# 条件式が成り立っていないかどうか
not (a > b)
```

Out[21]:

True

5.1.4 inメソッド

inメソッドとは、**コレクションに特定の要素が含まれているかどうか**を条件式に指定するメソッドです。**含まれていれば**Trueが、含まれていなければFalseが返されます。

In [1]:

```
# 適当なリストを定義
alphabet = ['a', 'b', 'c', 'd', 'e', 'f']
```

In [2]:

```
# cがalphabetに含まれるかどうか
'c' in alphabet
```

Out[2]:

True

In [3]:

```
# gがalphabetに含まれるかどうか
'g' in alphabet
```

Out[3]:

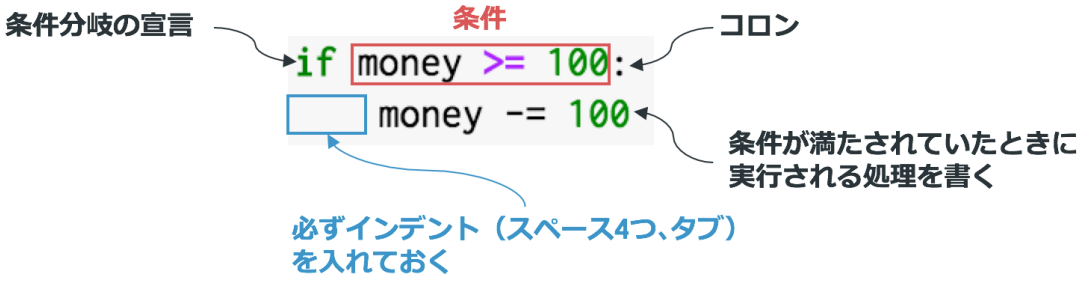
False

5.2 条件分岐

条件分岐とは、**条件によって処理を分けるプログラム**です。プログラムを組むときに、「とある条件を満たしているときはこの処理を、満たしていないときは別の処理をさせたい」という場面が出てきます。このように、状況に合わせて処理を分けたいというケースに対応します。

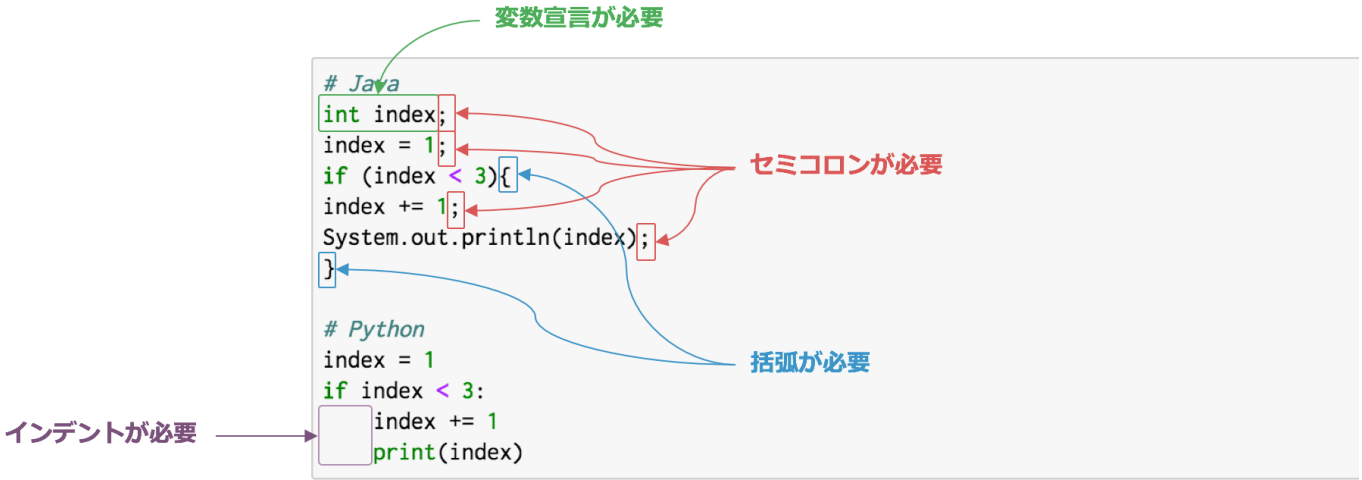
5.2.1 if文

条件分岐を書くときは、if文を利用します。if文の記法は、下図のようになっています。**必ず、構文の形を守るようにしてください。**



インデントの重要性

特に、処理を記すときにインデントを入れ忘れないように気をつけてください。他の通常のプログラミング言語と違って、**Pythonではインデントが絶対に必要**です。なぜなら、Pythonが**コードの構造をインデントによって表現**する言語だからです。**他の言語は"{}"によって構造を表現**するので、インデントはあってもなくても構いません。あったほうが少し見やすいかなと言った程度です。しかし、Pythonはそのような"{}"による入れ子構造がコードを見づらくする（**可読性**を落とす）として、インデントだけで構造を表現するようにしたのです。



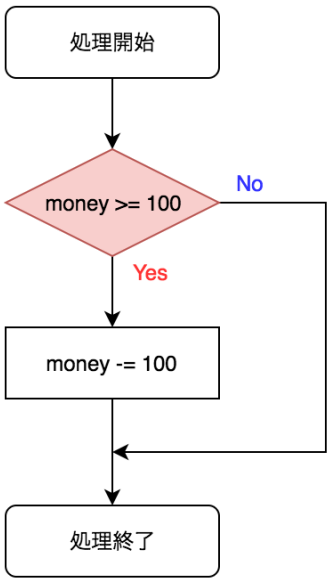
ここではサンプルとして、コンビニで買物をするプログラムを組んでみます。所持金が100円を超えていたら、100円の商品を購入して所持金が100円減少します。

In [1]:

```
# コンビニで買物をするプログラム
# 最初500円を持っている設定
money = 500

# 所持金が100円以上あれば、100円の商品を購入する
if money >= 100:
    money -= 100

# 処理後の所持金を確認
money
```

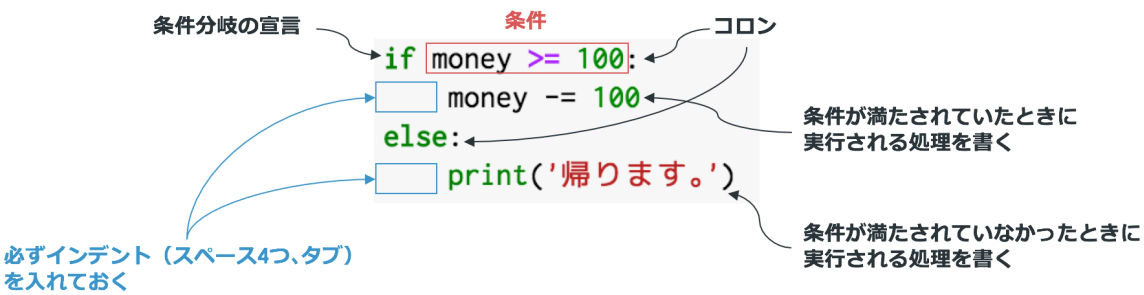
Out[1]:

400



5.2.2 else文

else文は、**条件が満たされていなかったときの処理を記述する文**です。この場合も、やはり**インデントには気をつけてください。**「所持金が100円に達していなかったら帰る」という機能を加えてみましょう。



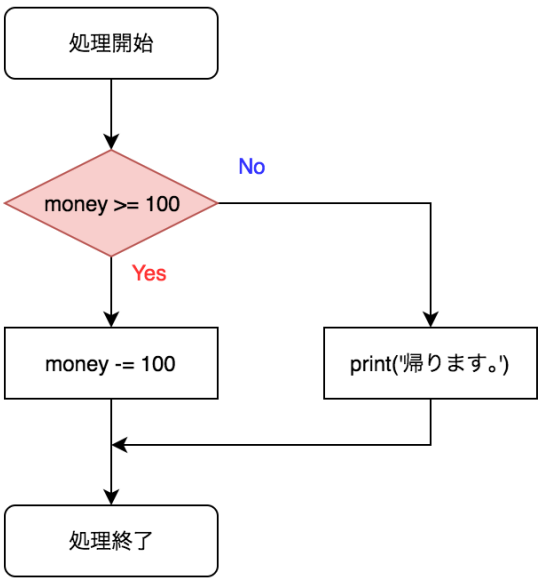
In [2]:

```
# コンビニで買物をするプログラム
# 最初75円を持っている設定
money = 75

# 所持金が100円以上あれば100円の商品を購入する
# なければ帰る
if money >= 100:
    money -= 100
else:
    print('帰ります。')

# 所持金の金額を確認
money
```

帰ります。

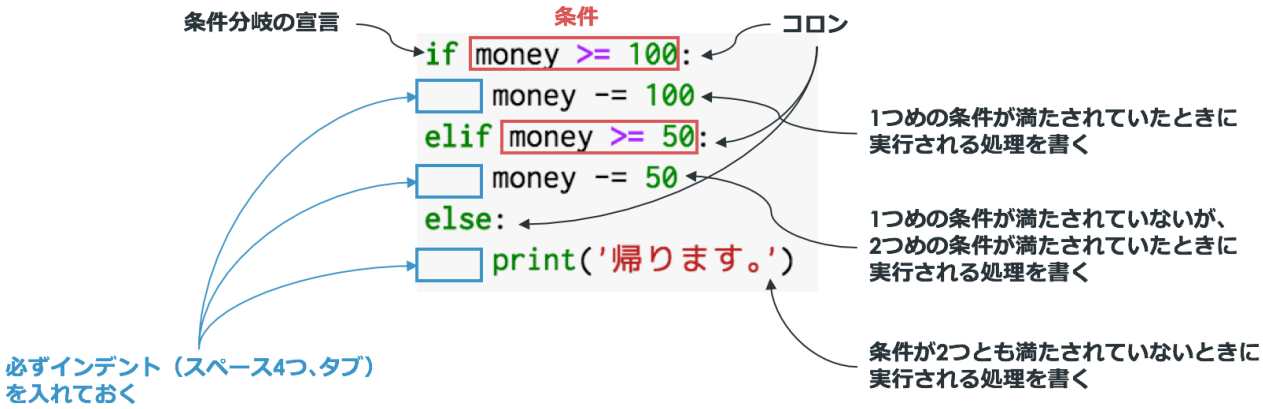
Out[2]:

75



5.2.3 elif文

elif文は、**とある条件式が満たされなかったときのために、新しい条件を設定する文**です。構文は以下のようになっています。

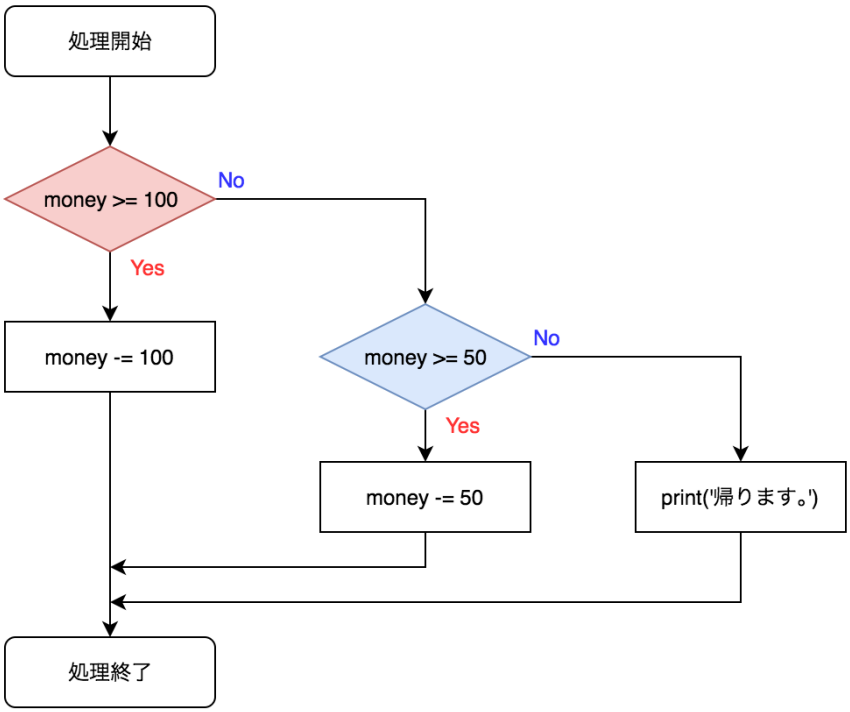


In [3]:

```
# コンビニで買物をするプログラム
# 最初75円を持っている設定
money = 75

# 所持金が100円以上あれば100円の商品を購入する
# なかった場合は、所持金が50円以上あれば50円の商品を購入する
# 50円もなかったら帰る
if money >= 100:
    money -= 100
elif money >= 50:
    money -= 50
else:
    print('帰ります。')

# 所持金の金額を確認
money
```

Out[3]:

25

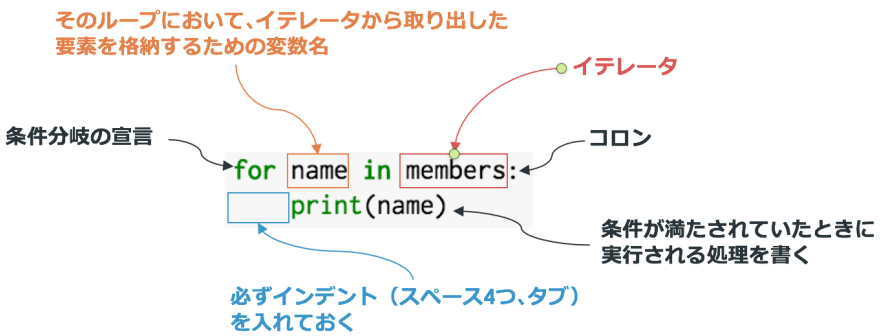


5.3 繰り返し

続いて、繰り返し処理です。チュートリアルでは、**特に使用頻度の高いfor文**を扱います。

5.3.1 for文の基本構造

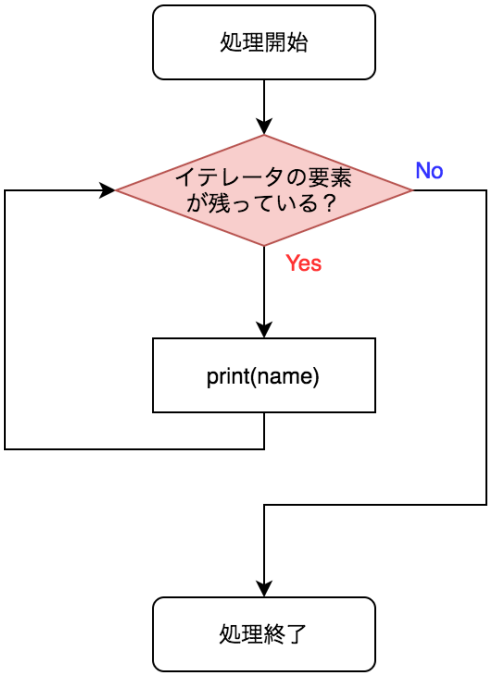
for文の基本構造は以下のようになっています。**イテレータ**という、**繰り返し処理の材料になるもの**を渡します。通常、**コレクション型**などが渡され、**ループのたびに中の要素を順番に取り出していきます。**また、if文のときと同様に、インデントを忘れないようにしてください。



In [4]:

```
# 適当なリストを定義
members = ['田中一郎', '佐藤二郎', '鈴木三郎']

# 名前を順番に呼んでいくプログラム
for name in members:
    print(name)
```

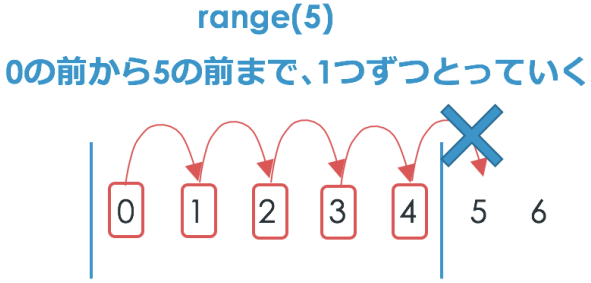
```
田中一郎
佐藤二郎
鈴木三郎
```



5.3.2 イテレータとして利用できるオブジェクト

ジェネレータ

ジェネレータとは、**コレクション型を生成するような手続きのこと**です。**range関数**などがその代表例です。0からの範囲を指定して、数値が並んだイテレータを生成できます。下の例だと、0から4までの数値が並んだイテレータが作られます。これらの数値を順番に出力するプログラムを組んでみましょう。

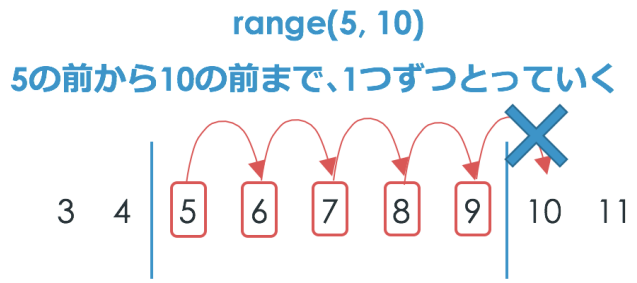


In [5]:

```
# 数字を0から4まで出力するプログラム
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

また、**スライシング**の要領で**数値の範囲を指定**することができます。下図の例だと、5から9までの数値を順に取り出せます。



In [34]:

```
#例(ステップ数指定：20から99までを5ずつカウントアップしている。)
for i in range(5, 10):
    print(i)
```

```
5
6
7
8
9
```

また、数値を何個おきに取り出すかということも指定できます。下図の例だと、0から9までの範囲の数値を2つおきに取り出しています。