

クラスとは？

クラスにはインスタンスやコンストラクタなどの概念^{がいねん}がで
てきます。

私^{わたし}自身^{じしん}、プログラミング^{ぺんぎょう}を勉強^{べんきょう}し始め^{はじめ}たときに、これを
理解^{りかい}するのに時間^{じかん}がかかりました。

私^{わたし}が何冊^{なんさつ}も書籍^{しょせき}を^よんで、こういう順番^{じゅんばん}であれば理解^{りかい}しや
すいというプロセス^{せつめい}で説明^{せつめい}します。

最後^{さいご}まで見て^みいただければ理解^{りかい}できるかと思う^{おも}うので、最後^{さいご}
まで見て^みいてください。

また、最後^{さいご}に確認^{かくにん}問題^{もんだい}もありますのでぜひ挑戦^{ちょうせん}してみてく
ださい。

まずクラスについて説明^{せつめい}します。

クラスとは、「データ」と「処理^{しより}」をまとめたものになり
ます。

Pythonでは、「データ」のことをアトリビュートといい、
「処理^{しより}」のことをメソッドといいます。

アトリビュートとメソッド

アトリビュートは、クラス内で定義された変数のことです。

アトリビュートは、変数と同じように、数値や文字列を

代入したり、参照したりすることができます。

クラスにアトリビュートを作ることを「アトリビュートを

定義する」と言います。

アトリビュートと変数の違いは、クラスの中にあるかクラ

スの外にあるかの違いです。

次にメソッドについて説明します。

前のレッスンで関数は、いろいろな「処理」をまとめて1つ

にしたものと説明しました。

メソッドも関数と同じで、いろいろな「処理」をまとめて1

つにしたものです。

簡単にいうと、メソッドは、クラス内に定義された関数で

す。

メソッドも関数と同じようにdefで定義します。

まとめると、アトリビュートはクラス内の^{ない}変数^{へんすう}、メソッドはクラス内の^{ない}関数^{かんすう}ということになります。

クラスの^{ていぎ}定義

クラスを作る^{つくる}ことをクラスを^{ていぎ}定義^いすると言^いいます。

このレッスンでどんなクラスを^{ていぎ}定義^{せつめい}するか^い説明^いします。

クラス名^{めい}はStudentとします。

そのクラスに生徒^{せいと}の名前^{なまえ}を^{だいにゆう}代入^{ていぎ}する「name」というアトリビュートを^{ていぎ}定義^いします。

そして、数学^{すうがく}と英語^{えいご}の点数^{てんすう}の平均^{へいきん}を^{けいさん}計算^{せつめい}するavgというメソッドを^{ていぎ}定義^いします。

コードを書^かいてい^いきましょう

```
class Student:
```

```
    def avg():
```

```
        print((80 + 70) / 2)
```

まずclassと書いて、次にクラス名を書きます。

今回はStudentというクラス名なので、Student。

クラス名の最初の文字は小文字でも定義はできますが、

最初の文字を大文字にするのは、Pythonの慣習となっています。

クラス名の最初の文字は大文字にしましょう。

コロンを書いて改行です。

メソッドの定義

次にメソッドを定義していきます。

数学と英語の点数の平均を計算するメソッドです。

平均を算出するので、平均という意味のaverageを省略し

て、avgというメソッド名にします。

まずdefと書いて、メソッド名。

丸括弧を書いて、コロン。改行です。

数学が80点、英語が70点を取れたとして、それらを足して

2で割ります。

表示させるためにprint関数でくくります。

ここまで見た通り、メソッドは関数の定義のやり方と同じです。

ただし、引数について、メソッドと関数に違うところがあります。

スライドで説明します。

メソッドを定義する場合、必ず1つ引数を記述しなければならないです。

関数の場合は、渡したい引数がない場合、空欄でもよいです。

しかし、メソッドの場合は、渡したい引数がない場合でも必ず引数が1つ必要になります。

この引数は、どんな引数名でもよいのですが、selfと書くのが慣習です。

つまり、メソッドに渡したい引数がない場合、メソッドの引数にselfを記述します。

メソッドに渡したい引数が1つの場合、メソッドの引数に

selfと渡^{わた}したい引^ひ数^{きすう}名^{めい}の合^{ごう}計^{けい}2つ。

メソッドに渡^{わた}したい引^ひ数^{きすう}が2つの場^{ばあ}合^い、メソッドの引^ひ数^{きすう}にselfを含^{ふく}めた合^{ごう}計^{けい}3つの引^ひ数^{きすう}を記^き述^{じゆつ}します。

コードを書^かいてい^いきましょう。

```
class Student:
```

```
    def avg(self):  
        print((80 + 70) / 2)
```

今^{こん}回^{かい}はメソッドに渡^{わた}す引^ひ数^{きすう}がな^いいので、引^ひ数^{きすう}の記^き述^{じゆつ}は、selfのみです。

このselfの役^{やく}割^{わり}は、Pythonがプ^じロ^っグ^こラ^うムの実^じ行^{こう}で使^{つか}っ^てい^るものです。

理^り屈^{くつ}が少^{すこ}し複^ふ雑^{ざつ}な^{ので}、メソッドの引^ひ数^{きすう}には、ど^{ばあ}んな場^い合^いでもselfと書^かく^と覚^{おぼ}え^えてしま^いま^しょう。

これ^でメソッドの定^{てい}義^ぎは終^おわ^わり^りです。

クラスを^{じっさい}実際に^{つかっ}使ってみたいと思いますが、クラスはこのままでは^{つかう}使うことができません。

クラスは、クラスから^{つく}作られたインスタンスを^{へんすう}変数に^{だいにゅう}代入してから^{つかい}使います。

クラスは、インスタンスになって初めて^{はじめてつかえる}使えるようになります。

コードを^か書いて、クラスの^{つかいかた}使い方を見ていきましょう。

```
class Student:
```

```
    def avg(self):  
        print((80 + 70) / 2)
```

```
a001 = Student()  
a001.avg()
```

クラスの^{つかいかた}使い方（インスタンス化^か）

数学が80点、英語が70点という点数は、aという学級の

出席番号001番の人が取ったとします。

変数名をa001とします。

イコールを書いて、クラス名を書き、丸括弧を書きます。

これで、クラスを使えるようになりました。

クラスを使えるような状態にすることを「インスタンス化」「オブジェクト化」「オブジェクト生成」と言ったりします。

インスタンスとは、実体という意味です。

ですから、インスタンス化とは、実体化という意味です。

つまり、インスタンス化とは、クラスという型から、インスタンスという実際に使える「モノ」を作ることを行います。

変数にインスタンスを代入して、インスタンスとして

使えるようになったa001は、これからa001インスタンスと呼ぶことにします。

次にメソッドの実行方法についてです。

a001にドットをつけて、メソッド名です。

丸括弧も忘れないでください。

それでは実行してみましょう。

実行結果：

75.0

平均点の75点が表示されました。

ここまでは、80点と70点を直接、メソッド内に記述して
しました。

これだと生徒が変わるごとにメソッドの書き換えが必要で
す。

これを引数で渡して計算できるようにしましょう。

そうすることで、クラスの書き換えは不要になり、クラスを使い回すことができます。

```
class Student:
```

```
    def avg(self, math, english):  
        print((math + english)/2)
```

```
a001 = Student()  
a001.avg(80,70)
```

クラス内に記述しているメソッドの2番目の引数をmathとします。

3番目の引数をenglishとします。

そのmathとenglishの引数を、print関数のところに記述します。

avgメソッドに80点と70点を渡して実行してみましょう。

じっこうけっか
実行結果：

75.0

75が^{ひょうじ}表示されました。

```
class Student:
```

```
    def avg(self, math, english):  
        print((math + english)/2)
```

```
a001 = Student()  
a001.avg(30,70)
```

メソッドに渡す引数を30点と70点にしてみましょう。

平均の50が表示されるはずです。

実行してみましょう。

実行結果：

50.0

50が表示されました。

アトリビュートの定義

```
class Student:
```

```
    def avg(self, math, english):  
        print((math + english)/2)
```

```
a001 = Student()  
a001.avg(80,70)  
  
a001.name = "sato"  
print(a001.name)
```

次にアトリビュートについてみていきましょう。

アトリビュートは、クラス内に定義された変数のことです。

a001にドット。アトリビュートを書いて、

値を代入します。

値はsatoさんとしましょう。

これでアトリビュートの定義は終わりです。

print関数で表示させてみましょう。

実行してみましょう。

実行結果：

75.0

sato

メソッドの結果の75とアトリビュートのsatoが表示されました。

```
class Student:
```

```
    def avg(self, math, english):  
        print((math + english)/2)
```

```
a001 = Student()
```

```
a001.avg(80,70)
```

```
a001.name = "sato"
```

```
print(a001.name)
```

```
print(a001.gender)
```

仮に、性別という意味のgenderというまだ定義していない

アトリビュートを表示させてみましょう。

もちろん、定義していないのでエラーになります。

実行してみましょう。

エラーです。

このように未定義のアトリビュートはエラーになります。

```
class Student:
```

```
    def avg(self, math, english):  
        print((math + english)/2)
```

```
a001 = Student()  
a001.avg(80,70)
```

```
a001.name = "sato"  
print(a001.name)
```

```
a002 = Student()  
print(a002.name)
```

また、a002というインスタンス名でインスタンス化をした
後に、
nameの属性を表示させてみましょう。
実行してみましょう。
エラーとなりました。
このように属性は、インスタンスごとに存在します。
逆の言い方をすれば、インスタンスごとに、属性
を定義しなければなりません。
つまり、インスタンスごとに属性が存在するので、
新しいインスタンスを作るときに、属性を
定義する必要があります。
そのため、10個インスタンスを作ったとすると、インスタ
ンスごとに属性を10個定義する記述をしなければ
なりません。
先ほどの例でいうと、「a001.name」のような記述をイン

スタンスごとに10個、^こ記述しなければなりません。

その不便^{ふべん}さを解消^{かいしょう}するものがコンストラクタです。

コンストラクタ

```
class Student:
```

```
    def __init__(self):  
        self.name = ""
```

```
    def avg(self, math, english):  
        print((math + english)/2)
```

```
a001 = Student()  
a001.name = "sato"  
print(a001.name)
```

```
a002 = Student()  
print(a002.name)
```

コンストラクタは、インスタンス化するときに、自動的に
実行されるメソッドのことです。

コンストラクタは、初期化メソッドとも言います。

初期化メソッドは、インスタンス化をすれば、必ず実行されるメソッドです。

そのため、後から使うアトリビュートは、初期化メソッドで自動的に作っておけばよいのです。

初期化メソッドの記述方法を見ていきましょう。

初期化メソッドもメソッドです。

メソッドなので、まずdefと記述します。

アンダースコアを2つ。initと書いて、もう一度アンダースコアを2つ。

丸括弧を記述します。

メソッドを定義する場合、最初は必ずselfを書くのでselfを記述。コロンを書いて改行。

これで初期化メソッドの記述は終わりです。

フィールドには、佐藤さん、鈴木さんといったような名前

を代入^{だいにゅう}したいので、nameの属性^{ていぎ}を定義^{ていぎ}しましょう。

self、ドット、nameで属性^{ていぎ}を定義^{ていぎ}することができます。

ちなみに、ここでもselfが^で出てきました。

selfと書^かくことにより、selfにインスタンスが^{だいにゅう}代入^{だいにゅう}されます。

引数^{ひきすう}のselfにa001が^{だいにゅう}代入^{だいにゅう}され、self.nameがa001.nameとなるイメージです。

ここは難^むしい理屈^{かし}なので、そういう仕組み^{しきくみ}になっているのだと思^{おも}って覚^{おぼ}えておきましよう。

ここでは、ダブルクォテーション2つで、空^{そら}の値^ねを^{だいにゅう}代入^{だいにゅう}させておきましよう。

では、インスタンス化^かをして、a001とa002のnameの中^{なか}を見^みてみましょう。

avgメソッドの記述^{きじゆつ}は消^けしておきます。

a001にはsatoが、先ほどエラーになったa002には初期化
メソッドでアトリビュートを作ったので、

エラーにならず、空の値が入っているはずです。

実行してみましょう。

実行結果：

```
sato
```

エラーにならずに、satoと空の値が表示されました。

```
class Student:
```

```
    def __init__(self):  
        self.name = ""
```

```
def avg(self, math, english):  
    print((math + english)/2)
```

```
a001 = Student()  
a001.name = "sato"  
display(print(a001.name))
```

```
a002 = Student()  
a002.name = "tanaka"  
display(print(a002.name))
```

a001にsatoを代入^{だいにゅう}してみましょう。

a002にtanakaを代入^{だいにゅう}してみましょう。

実行^{じっこう}してみます。

実行結果^{じっこうけっか}：

sato

tanaka

satoとtanakaが^{ひょうじ}表示されました。

```
class Student:

    def __init__(self, name):
        self.name = name

    def avg(self, math, english):
        print((math + english)/2)

a001 = Student("sato")
print(a001.name)

a002 = Student("tanaka")
print(a002.name)
```

アトリビュートは、インスタンス化と^か同時に^{どうじ}代入^{だいにゅう}することもできます。

^{しよきか}初期化メソッドの^{だい}第2^{ひきすう}引数にnameという^{ひきすう}引数^{きじゆつ}を記述しま

す。

ダブルクォーテーションで空^{そら}を代入^{だいにゅう}していたところに、
name^{きじゆつ}を記述^{きじゆつ}します。

イメージとしては、第2引数^{だいひきすう}のname^{しよきか}を初期化メソッド^な内の
name^{うけ}が受けて、それをself.name^{だいにゅう}に代入^{だいにゅう}します。

では、a001にインスタンス化^かと同時に"sato"^{どうじ}を渡^{わた}してみま
しょう。

a002インスタンスにも"tanaka"^{わたし}を渡^{わた}します。

^{ひょうじ}表示^{ひょうじ}させてみましょう。

^{じっこうけっか}実行結果：

```
sato  
tanaka
```

satoさんとtanakaさんが^{ひょうじ}表示^{ひょうじ}されました。

クラスの便利^{べんり}なところ

以上がクラスの使い方です。

最後に、クラスの便利なところはどんなところでしょう？

クラスは1度、定義しておけば、後からいくらでもインスタンスを作ることができます。

車を作る「設計図」がクラス、たい焼きを作る「金型」がクラスと表現したりもします。

もう1つ表現するなら、クラスは、パソコンで使うコピペです。

コピペでどんどんインスタンスを作ることができます。

もしクラスがなければ、生徒ひとりひとりのためにStudent

クラスを書かなければいけないので面倒です。

クラスがあるから効率よくプログラミングすることができます。

確認問題

最後に確認問題をやっていきましょう。

このレッスンでは新しい用語がたくさん出てきたので、確認してみましょう。

① a001のことは何と言うのでしょうか？

② Student()は何を呼び出しているのでしょうか？

- ③ `def init(self):`の部分は^{ぶぶんに}何と言うのでしょうか？
- ④ `self.name`は^{なに}何を^{ていぎ}定義しているのでしょうか？
- ⑤ `def avg(self, math, english):`は^{なに}何を^いう^うのでしょうか？
-

```
class Student:
    def __init__(self):
        self.name = ""
    def avg(self, math, english):
        print((math + english)/2)

a001 = Student()
a001.name = "sato"
print(a001.name)
```

1: `class Student:`
2: `a001 = Student()`
3: `def __init__(self):`
4: `self.name = ""`
5: `def avg(self, math, english):`

1がインスタンス

2がクラス

3がコンストラクタ、^{しよぎ}初期化メソッドとも^い言^いいます。

4はアトリビュート

5はメソッドです。