

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できます。

私自身、プログラミングを勉強し始めたときに、これを理解するのに時間がかかりました。

私は何冊も書籍を読んで、こういう順番であれば理解しやすいというプロセスで説明します。

最後まで見ていただければ理解できるかと思うので、最後まで見ていてください。

また、最後に確認問題もありますのでぜひ挑戦してみてください。

まずクラスについて説明します。

クラスとは、「データ」と「処理」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」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変数、メソッドはクラス内の関数ということになります。

クラスの定義

クラスを作ることをクラスを定義すると言います。

このレッスンでどんなクラスを定義するか説明します。

クラス名はStudentとします。

そのクラスに生徒の名前を代入する「name」というアトリビュートを定義します。

そして、数学と英語の点数の平均を計算するavgというメソッドを定義します。

コードを書いていきましょう

```
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関数でくくりましょう。

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と渡したい引数名の合計2つ。

メソッドに渡したい引数が2つの場合、メソッドの引数にselfを含めた合計3つの引数を記述します。

コードを書いていきましょう。

```
class Student:
```

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

今回はメソッドに渡す引数がないので、引数の記述は、selfのみです。

このselfの役割は、Pythonがプログラムの実行で使っているものです。

理屈が少し複雑なので、メソッドの引数には、どんな場合でもselfと書くと覚えてしまいましょう。

これでメソッドの定義は終わりです。

クラスを実際に使ってみたいと思いますが、クラスはこのままでは使うことができません。

クラスは、クラスから作られたインスタンスを変数に代入してから使います。

クラスは、インスタンスになって初めて使えるようになります。

コードを書いて、クラスの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()
```

クラスの使い方（インスタンス化）

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いました。

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よう。

実行結果：

75.0

75が表示されました。

```
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、アトリビュート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アトリビュートが存在するので、新しいインスタンスを作るごとに、アトリビュート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アトリビュートを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、記述しなければなりません。

その不便さを解消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したいので、nameのアトリビュートを定義しましょう。

self、ドット、nameでアトリビュートを定義することができます。

ちなみに、ここでもselfが出てきました。

selfと書くことにより、selfにインスタンスが代入されます。

引数のselfにa001が代入され、self.nameがa001.nameと

なるイメージです。

ここは難しい理屈なので、そういう仕組みになっているのだと思って覚えておきましょう。

ここでは、ダブルクォテーション2つで、空の値を代入させておきましょう。

では、インスタンス化をして、a001とa002のnameの中を見てみましょう。

avgメソッドの記述は消しておきます。

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を代入してみましよう。

a002にtanakaを代入してみましよう。

実行してみます。

実行結果：

```
sato  
tanaka
```

satoとtanakaが表示されました。

```
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、インスタンス化と同時に代入することもできます。

初期化メソッドの第2引数にnameという引数を記述します。

ダブルクォーテーションで空を代入していたところに、nameを記述します。

イメージとしては、第2引数のnameを初期化メソッド内のnameが受けて、それをself.nameに代入します。

では、a001にインスタンス化と同時に"sato"を渡してみましよう。

a002インスタンスにも"tanaka"を渡します。

表示させてみましょう。

実行結果：

sato

tanaka

satoさんとtanakaさんが表示されました。

クラスの便利なところ

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ーです。

コピー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のことは何と申うでしようか？
- ② Student()は何を呼び出しているでしようか？
- ③ def **init**(self):の部分は何と申うでしようか？
- ④ self.nameは何を定義しているでしようか？
- ⑤ def avg(self, math, english):は何と申うでしようか？


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

1 a001 = Student()
2 a001.name = "sato"
3 print(a001.name)
```

The diagram illustrates the components of a Python class and its instantiation. It features a code snippet with five numbered callouts in brown circles: 1 points to the class instantiation `a001 = Student()`; 2 points to the attribute assignment `a001.name = "sato"`; 3 points to the `__init__` method definition; 4 points to the `self.name` attribute assignment within the `__init__` method; and 5 points to the `avg` method definition.

1がインスタンス

2がクラス

3がコンストラクタ、初期化メソッドとも言います。

4はアトリビュート

5はメソッドです。