1 オブジェクト指向プログラミング

今まで私たちが記述してきたプログラムは、手続き型プログラミングと呼ばれるプログラミングスタイルです。手続き型プログラミングは、処理を逐次的に記述していくプログラミングスタイルです。一方、オブジェクト指向プログラミングは、プログラムをオブジェクトと呼ばれる単位に分割し、それぞれのオブジェクトが持つデータと処理をまとめて扱うプログラミングスタイルです。

2 クラスとオブジェクト

オブジェクトとは、データ (属性) とそれを操作する関数 (メソッド) をカプセル化したものです。オブジェクト指向は現実世界を用いて説明するとわかりやすいです。例えば、猫というオブジェクトを考えてみましょう。猫は、名前や年齢、毛色などといった属性を持ち、鳴く、歩く、食べるといったメソッド (行動) を持っています。

オブジェクト指向プログラミングでは、オブジェクトを定義するための設計図としてクラスというものがあります。クラスはオブジェクトの設計図であり、いわば猫を作成するためのテンプレートです。この設計図を基にして、猫を現実世界に呼び出します (**インスタンス化**)。

3 クラスの定義

クラスの定義は以下のように行います。

ソースコード 1: クラスの定義

```
class ClassName:
    def __init__(self, arg1, arg2, ...):
        self.arg1 = arg1
        self.arg2 = arg2
        ...
    def method1(self, arg1, arg2, ...):
        ...
    def method2(self, arg1, arg2, ...):
        ...
```

ポイント

- クラス名は大文字で始める
- __init__メソッドはクラスの初期化を行うメソッドであり、インスタンスが生成される際に自動的に呼び出される
- インスタンス変数は self. 変数名で定義する
- メソッドの第一引数は self である

猫をクラスにしてみましょう。

ソースコード 2: Cat クラスの定義

```
class Cat:
    def __init__(self, name, age, color):
        self.name = name # 名前
        self.age = age # 年齢
        self.color = color # 毛色
    def cry(self):
        print(' にゃー')
    def walk(self):
        print(' 歩く')
```

4　オブジェクトの生成

クラスを定義したら、そのクラスを元にオブジェクトを生成します。オブジェクトの生成は以下のように行います。

ソースコード 3: オブジェクトの生成

```
オブジェクト名
 = クラス名引数 (1, 引数 2, ...)
```

Cat クラスを元に猫のオブジェクトを生成してみましょう。

ソースコード 4: Cat クラスのインスタンス化

```
tama = Catたま (’’, 3, 白’’)
```

これで、名前が「たま」、年齢が 3 歳、毛色が白の猫のオブジェクトが生成されました。続いて、たまが鳴くと歩くメソッドを呼び出してみましょう。

ソースコード 5: Cat クラスのメソッド呼び出し

```
tama.cry()
tama.walk()

# 実行結果
# にゃー
# 歩く
```

5　クラスの継承

クラスを定義する際に、既存のクラスを元に新しいクラスを定義することができます。これを**クラスの継承**といいます。継承元のクラスを**親クラス**、継承先のクラスを**子クラス**といいます。猫を基に考えると、猫は哺乳類であるため、猫クラスは哺乳類クラスを継承するといったイメージです。

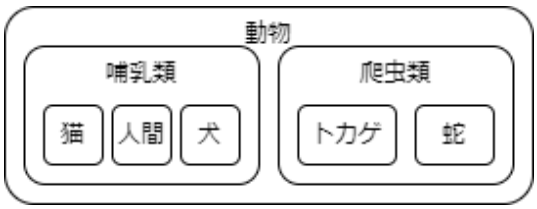

図 1: 継承のイメージ

クラスの継承は以下のように行います。

ソースコード 6: クラスの継承

```
class SubClassName(ClassName):
    def __init__(self, arg1, arg2, ...):
        super().__init__(arg1, arg2, ...)
        self.arg1 = arg1
        self.arg2 = arg2
        ...
```

ポイント

- 親クラスのメソッドを呼び出す際は super(). メソッド名 () を使用する

猫を基に、子猫クラスを定義してみましょう。

ソースコード 7: 子猫クラスの定義

```
class Kitten(Cat):
    def __init__(self, name, age, color, mother):
        super().__init__(name, age, color)
        self.mother = mother
    def cry(self):
        print(' にゃーにゃー')
```

子猫クラスは猫クラスを継承しており、cry メソッドをオーバーライドしています。また、子猫クラスは母猫のオブジェクトを受け取ります。

6 演習

メソッドの中身は print 文で何かしらの表示を行うだけでよいです。

1. 以下のクラスを定義し、オブジェクトを生成してください。
   - クラス名: Dog
   - 属性: 名前、年齢、毛色
   - メソッド: bark(吠える)、run(走る)
2. 以下のクラスを定義し、オブジェクトを生成してください。
   - クラス名: Puppy
   - 親クラス: Dog
   - 属性: 名前、年齢、毛色、母犬
   - メソッド: bark(吠える)
3. 以下のクラスを定義し、オブジェクトを生成してください。
   - クラス名: Character
   - 属性: 名前、HP、MP
   - メソッド: attack(攻撃)、heal(回復)
4. 以下のクラスを定義し、オブジェクトを生成してください。
   - クラス名: Hero
   - 親クラス: Character
   - 属性: 名前、HP、MP、武器
   - メソッド: attack(攻撃)、heal(回復)
5. 以下のクラスを定義し、オブジェクトを生成してください。
   - クラス名: Slime
   - 親クラス: Character
   - 属性: 名前、HP、MP
   - メソッド: attack(攻撃)、heal(回復)、split(分裂)