1 もくじ

2 プログラミング言語におけるクラスについて

クラスとは

C言語で「車が動いた時のn分動いた時の走行距離とガソリン消費」のプログラムを書いてみましょう.
(プログラムは流し見程度で大丈夫です)

```
#include<stdio.h>


// 車の構造体
typedef struct car
{
    char *name;               // 車の名前
    double remain_fuel;       // 残り燃料 [mL]
    double velocity;          // 速度 [m/s]
    double fuel_consumption; // 燃費 [m/mL]
    double total_mileage;     // 総走行距離 [m]
} Car;



// 車が進む距離を返す
double get_mileage(Car *car, double drive_time){
```

```
    return (car->velocity) * drive_time;
    // car->total_mileage += mileage;
}

// 消費した燃料を返す
double get_fuel_consumption(Car *car, double mileage){
    return mileage / car->fuel_consumption;
}

// 総走行距離の加算
void add_total_mileage(Car *car, double mileage){
    car->total_mileage += mileage;
}

// 残り燃料の加減
void add_remain_fuel(Car *car, double add_fuel){
    car->remain_fuel += add_fuel;
}

// 車がn分走った時に伴う変化
// drive_time [minutes]
void drive_car(Car *car, double drive_time){
    double mileage = get_mileage(car, drive_time);
    double loss_fuel = get_fuel_consumption(car, mileage);
    add_total_mileage(car, mileage);
    add_remain_fuel(car, -loss_fuel);
}

// 車の現在の状態を表示する
void print_car_state(Car *car){
    printf("--- [%s] の 状態 ---\n", car->name);
    printf("  残り燃料 : %8.3lf [mL]\n", car->remain_fuel);
    printf("総走行距離 : %8.3lf  [m]\n", car->total_mileage);
}
```

```
void main(void) {
    Car boo1 = {"ふぃっと", 100.0, 30.0, 80.0, 0};

    print_car_state(&boo1);

    putchar('\n');

    drive_car(&boo1, 20.0);
    print_car_state(&boo1);

    putchar('\n');

    drive_car(&boo1, 10.8);
    print_car_state(&boo1);

    putchar('\n');

    drive_car(&boo1, 32.7);
    print_car_state(&boo1);

    putchar('\n');

    Car boo2 = {"すごいやつ", 50.0, 75.0, 120.0, 0};

    print_car_state(&boo2);

    putchar('\n');

    drive_car(&boo2, 20.0);
    print_car_state(&boo2);

    putchar('\n');
```

```
    drive_car(&boo2, 10.8);
    print_car_state(&boo2);

    putchar('\n');

    drive_car(&boo2, 32.7);
    print_car_state(&boo2);

}
```

```
--- [ふぃっと] の 状態 ---
  残り燃料 :  100.000 [mL]
総走行距離 :    0.000  [m]

--- [ふぃっと] の 状態 ---
  残り燃料 :   92.500 [mL]
総走行距離 :  600.000  [m]

--- [ふぃっと] の 状態 ---
  残り燃料 :   88.450 [mL]
総走行距離 :  924.000  [m]

--- [ふぃっと] の 状態 ---
  残り燃料 :   76.188 [mL]
総走行距離 : 1905.000  [m]

--- [すごいやつ] の 状態 ---
  残り燃料 :   50.000 [mL]
総走行距離 :    0.000  [m]

--- [すごいやつ] の 状態 ---
  残り燃料 :   37.500 [mL]
総走行距離 : 1500.000  [m]

--- [すごいやつ] の 状態 ---
  残り燃料 :   30.750 [mL]
総走行距離 : 2310.000  [m]
```

```
--- [すごいやつ] の 状態 ---
  残り燃料 :   10.312 [mL]
総走行距離 : 4762.500  [m]
```

自動車の持ついくつかの変数たちと, それらを操作する関数たちは密接な関係にあり, これらの関数自身も自動車の構造体以外に対しては使用しないと思います.

このように, C言語では関連のある変数をまとめる機能(構造体や共有体など)はありますが, それらと関係のある関数たちをまとめる術がありません.

プログラムではモジュール化の考え方から, 関係のあるもの同士をまとめて置いておきたがります. なのに, これは少し変だと思いませんか?

そこで,「ある要素を構成する変数たち」と「それらの変数を用いて動作し, 影響を与える関数」をまとめようとしたものが `class` になります.

`class` の根本的な考え方はどの言語でも大体同じなので, このことは覚えておいて損はないと思います.

クラスとインスタンス

クラス自体はいわば設計図の段階です.
クラスをもとに, 実体のある `インスタンス` を生成する必要があります.

先ほどの例で言えば, こういう動作をする「車」というものがあるよと示しているのがクラスで, そこへ具体的な値を入れる(先ほどであれば残り燃料や走行速度など)ことで, 実体のある `インスタンス` を作成します.

1つの同じクラスから, 違う変数値をもたせることで, いくつもの違う要素を持ったインスタンスを生成できます.

クラスにおける様々な名称

ここからは, クラスにおける様々な一般的な名称を紹介していきます.
言語によって名称が異なることがあるので, そこだけ注意してください.

フィールドとメソッド

クラスの持つ変数を `フィールド` と呼び,
クラスの持つ関数を `メソッド` と呼びます.

インスタンス

先ほどのべたように, クラスを実体化させたものです

コンストラクタとデストラクタ

クラスからインスタンスを生成するときに, 必ず実行される特殊なメソッドを `コンストラクタ` といいます.
主にクラスの持つフィールドの値をセットするのに使われます.

デストラクタはその逆で, インスタンスがデリートされるときに実行される特殊なメソッドです. 各変数のメモリ解放であったり, 終了時に行っておきたい処理を書いておきます.

継承/多重継承

かなりの汎用性のあるクラスから, すこしローカル化したクラスを生み出すような操作を継承といいます.
ベースを継承元クラスで作成し, 特殊な部分や独自の部分は継承先で追加する感じです.

先ほどの例でいくなら, 乗り物クラスを継承して, 自動車クラスや飛行機クラス, 自転車クラスを作るようなものですね.
「乗り物」という段階ではまだ確実にある要素(名前や乗り物の大きさ, スピードなど)は先に作っておき, 継承先で各々の特殊な要素(翼の形状, 燃料の消費, タイヤの摩耗など)を作成するという感じです.

継承はめちゃくちゃ便利な機能で, さいこーの機能なんですが, 個人で極小規模のプログラムを組む程度なら継承はいらないので, 説明はしません.

3 Pythonにおけるクラスの実装

ここではとりあえずどのように実装し, どのようにメソッドやフィールドを呼び出すか, インスタンスを生成するか, というものを見ていきます.

クラスの実装方法を覚えるというよりは, 後々たくさん使うことになる `インスタンス.メソッド()` や `インスタンス.フィールド` という記法が, こうなっているということを知ってほしいだけです.

クラスの実装

Pythonでは以下のようにクラスを作成します.

```
class Car:


    def __init__(self, name, remain_fuel, velocity, fuel_consumption):
        self.name = name                # 車の名前
        self.remain_fuel = remain_fuel  # 残りの燃料 [mL]
        self.velocity = velocity        # 速度 [m/s]
        self.fuel_consumption = fuel_consumption # 燃費 [m/mL]
        self.total_mileage = 0          # 総走行距離 [m]
```

```
    def get_mileage(self, drive_time):
        return self.velocity * drive_time

    def get_fuel_consumption(self, mileage):
        return mileage / self.fuel_consumption

    def add_total_mileage(self, mileage):
        self.total_mileage += mileage

    def add_fuel(self, add_fuel):
        self.remain_fuel += add_fuel

    def drive(self, drive_time):
        mileage = self.get_mileage(drive_time)
        loss_fuel = self.get_fuel_consumption(mileage)
        self.add_total_mileage(mileage)
        self.add_fuel(-loss_fuel)

    def print_state(self):
        print("--- [" + self.name + "] の 状態 ---")
        print("残りの燃料 : {:7.3f} [mL]".format(self.remain_fuel))
        print("総走行距離 : {:7} [m]".format(self.total_mileage))

if __name__ == '__main__':
    boo1 = Car("ふぃっと", 100, 30, 80)
    boo1.print_state()

    print()

    boo1.drive(20)
    boo1.print_state()

    print()
```

```
    boo1.drive(10.8)
    boo1.print_state()

    print()

    boo1.drive(32.7)
    boo1.print_state()
```

インスタンスの生成

定義したクラス名に `()` をつけ, `__init__` で定義した引数を与えてあげます.

```
instance = Class_name(argv1, argv2, …)
```

メソッドやフィールドの呼び出し

生成したインスタンスの後ろに `.` をつけ, フィールド/メソッドをその後ろに続けて書きます. また, メソッドであれば, `()` をつけ, 必要に応じて引数を渡します.

```
instance = Class_name(argv1, argv2, …)

# フィールドを参照する
instance.field_name

# メソッドを呼び出す
instance.method_name()
```

今後, `aaa.bbb()` や `aaa.ccc` みたいなのがたくさん出てきますが, それはそのクラスで定義されたメソッドやフィールドが呼び出されているということです.