

引き継ぎ資料 Vol.4

オブジェクトの話

2016/08/??

コンセプト

“オブジェクトって何?” を解決

目次

1. 導入
2. クラス
3. カプセル化
4. 繙承
5. ポリモーフィズム
6. まとめ

目次

1. 導入
2. クラス
3. カプセル化
4. 繙承
5. ポリモーフィズム
6. まとめ

導入

Q.

オブジェクトって何?

導入

Q.

オブジェクトって何?

A.

データや構造としての“何か”

例

```
>>> int_variable = 10 # int object
>>> float_variable = 3.7 # float object
>>> str_object = "Hello" # str object
>>> none_object = None # NoneType object
>>> list_object = [] # list object
>>> dict_object = {} # dict object

>>> # numpy.ndarray object
>>> array_object = numpy.array([1, 2, 3])
```

例

```
>>> int_variable = 10 # int object
>>> float_variable = 3.7 # float object
>>> str_object = "Hello" # str object
>>> none_object = None # NoneType object
>>> list_object = [] # list object
>>> dict_object = {} # dict object

>>> # numpy.ndarray object
>>> array_object = numpy.array([1, 2, 3])
```

なんでもオブジェクト

オブジェクトの中身

- ・ 関数 (正確にはメソッド)
- ・ 他のオブジェクト

例

```
>>> list_object = [1, 2, 3, 4] # list object
>>> # list_objectの中にあるappendメソッド
>>> list_object.append(5)
>>> # list_objectがappendという操作を受け付けて
>>> # リストの中に5を追加
>>> print(list_object)
[1, 2, 3, 4, 5]
>>> # numpy.ndarray object
>>> array_object = numpy.array([
...     [1, 2, 3],[4, 5, 6]
... ])
>>> print(a.shape) # aの中のタプルオブジェクト
(2, 3)
```

型

オブジェクトは様々な中身を持つことが可能
中身のテンプレート → 型

オブジェクトの作成 = 型から物を作成



型って自分で作れないの？

型って自分で作れないの？

自作できる型 → クラス

目次

1. 導入
2. クラス
3. カプセル化
4. 繙承
5. ポリモーフィズム
6. まとめ

クラスの話をする前に…

C の “構造体” 覚えていますか？

クラス

クラス…C の構造体 α

クラス

クラス…C の構造体 $+\alpha$

$+\alpha$ の部分がとても巨大

復習 - 構造体 -

“車”を表す構造体

```
typedef struct Car{  
    double x;  
    double y;  
} Car;
```

- ・メンバは x と y
- ・それぞれには. でアクセス

復習 - 構造体 -

```
int main(int argc, char** argv){  
    Car car1, car2;  
    car1.x = 1.0;  
    car1.y = 1.0;  
    car2.x = 5.0;  
    car2.y = 5.0;  
    return 0;  
}
```

car1 と car2 は別物

復習 - 構造体 -

```
int main(int argc, char** argv){  
    Car car1, car2;  
    car1.x = 1.0;  
    car1.y = 1.0;  
    car2.x = 5.0;  
    car2.y = 5.0;  
    return 0;  
}
```

car1 と car2 は別物

car1 と car2 は Car 型のオブジェクト

Python で構造体的なこと

```
class Car:  
    def __init__(self):  
        self.x = 0.0  
        self.y = 0.0  
  
def main():  
    car1 = Car()  
    car2 = Car()  
    car1.x, car1.y = 1.0, 1.0  
    car2.x, car2.y = 5.0, 5.0
```

Car はクラス

__init__

```
class Car:  
    def __init__(self):  
        self.x = 0.0  
        self.y = 0.0
```

- ・ __init__ は関数
- ・ クラスの中に宣言されている → メンバ関数
- ・ オブジェクトの作成時に呼び出し
- ・ self は自分自身を指す

2つの run 関数は同じ処理

```
class Car:  
    def __init__(self):  
        self.x, self.y = 0.0, 0.0  
    def run(self):  
        self.x += 1.0 # selfのxを増加  
  
def run(car):  
    car.x += 1.0 # 与えられたcarのxを増加  
  
def main():  
    car1 = Car()  
    car1.run()  
    run(car1) # car1.x == 2.0
```

メンバ関数

どちらの関数でも同じ処理が可能

メンバ関数

どちらの関数でも同じ処理が可能

メンバ関数が不要?

メンバ関数

どちらの関数でも同じ処理が可能

メンバ関数が不要?

そんなことはない

2つの run 関数は同じ？

```
class Car:  
    # 中略  
    def run(self): # ここにはCarしかこない  
        self.x += 1.0  
  
def run(car): # ここにはCarに関係ないものが来れる  
    car.x += 1.0
```

- ・ 関係あるものは関係する場所に配置
- ・ 事故防止は大事

2つの run 関数は同じ？

```
class Car:  
    # 中略  
    def run(self): # ここにはCarしかこない  
        self.x += 1.0  
  
def run(car): # ここにはCarに関係ないものが来れる  
    car.x += 1.0
```

- ・ 関係あるものは関係する場所に配置
- ・ 事故防止は大事

メンバ関数超大事

オブジェクト指向

プログラム全体をオブジェクトの集合で構成する手法

概念

- ・ カプセル化
- ・ 繙承
- ・ ポリモーフィズム

目次

1. 導入
2. クラス
3. カプセル化
4. 繙承
5. ポリモーフィズム
6. まとめ

目次

1. 導入
2. クラス
3. カプセル化
4. 繙承
5. ポリモーフィズム
6. まとめ

目次

1. 導入
2. クラス
3. カプセル化
4. 繙承
5. ポリモーフィズム
6. まとめ

目次

1. 導入
2. クラス
3. カプセル化
4. 繙承
5. ポリモーフィズム
6. まとめ

“オブジェクト”の概念は使いこなせば便利
データ解析の分野では作る必要は無い可能性が高い
使えないのは危険

Questions?