

機械学習のための PythonマスターコースDay3



<https://to-kei.net>

全人類がわかる統計学 Presents



目次

- オブジェクト指向とは？
- クラスとは
 - ・クラスとは
 - ・クラス変数
- オブジェクトとは
 - ・インスタンス変数
 - ・クラスを使う関数
- コンストラクタとデストラクタ
- メソッド
- カプセル化とPythonにおけるgetter、setter
- 継承
- オーバーライド
- 関数オブジェクト



オブジェクト指向とは？~プログラム版

➤マウスをクリックしたらメモ帳を出すプログラムを作りたい。



色が違うのを作りたい。

全部形は四角い。

複数同時に生成できるようにしたい。

消しゴムとかページを増やすとかの機能を集約したい。



オブジェクト指向とは？ ~たい焼き版

➤たい焼きの屋台を出したい



こしあんとつぶあんと
カスタード味がいるな…
味は違うけど形は一緒。
大量に作りたい。

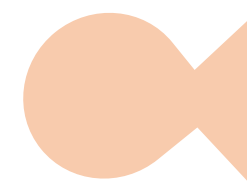
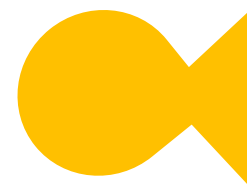
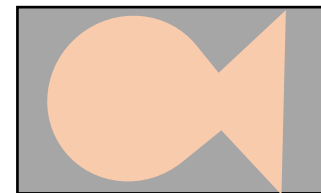
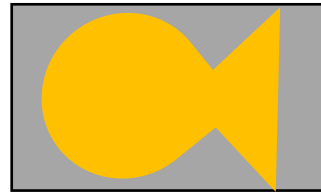
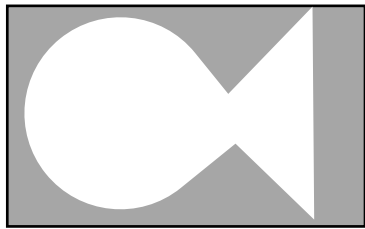


オブジェクト指向とは？

- クラス...オブジェクトの設計図のようなもの。メソッドやプロパティなどを一まとめにしてパッケージ化したもの。
- メソッド、プロパティ...オブジェクト中に含まれるそのオブジェクトの中で使われる処理（関数）や変数など。
- インスタンス...クラスから生成されたオブジェクトのこと。

材料を入れる（インスタンスの生成。具体的な変数の値などを設定する。）

色々なたい焼きができる（実際に使われるインスタンスができる。）



たい焼きの型(クラス。
たい焼きの形などを定め
ておく設計図。)

クラスとは



型と変数

```
a = 1
```

```
type(a) #int
```

変数 型

pythonでは、型が自動で決まる



イメージ

組み込み型

int	整数
float	浮動小数点数
list	配列
str	文字列
bool	真偽
dict	辞書型

クラス

Person

人間



名前
年齢
性別
...



クラスは型である



クラスは自由に定義できる



クラスの宣言

```
class クラス名:  
    pass
```

```
class Person:  
    pass
```

中身はないけど、Personというクラス(型)をつかった



クラス変数を追加

```
class クラス名:  
    クラス変数 = 値
```

```
print(クラス名.クラス変数名)
```

```
class Person:  
    species = "Homo sapiens"  
    number = 1  
print(Person.species)|  
print(Person.number)
```

```
Homo sapiens  
1
```

すべてのクラス(型)で共有する変数を作成できる

Personクラスはすべて、speciesとnumberというクラス変数を所有する



演習問題 1

- 問題1：Animalという名前のクラスを宣言しましょう
- 問題2：Animalクラスにnumberというクラス変数を追加し、値0を代入しましょう
- 問題3：Animalクラスのクラス変数、numberをprintしましょう

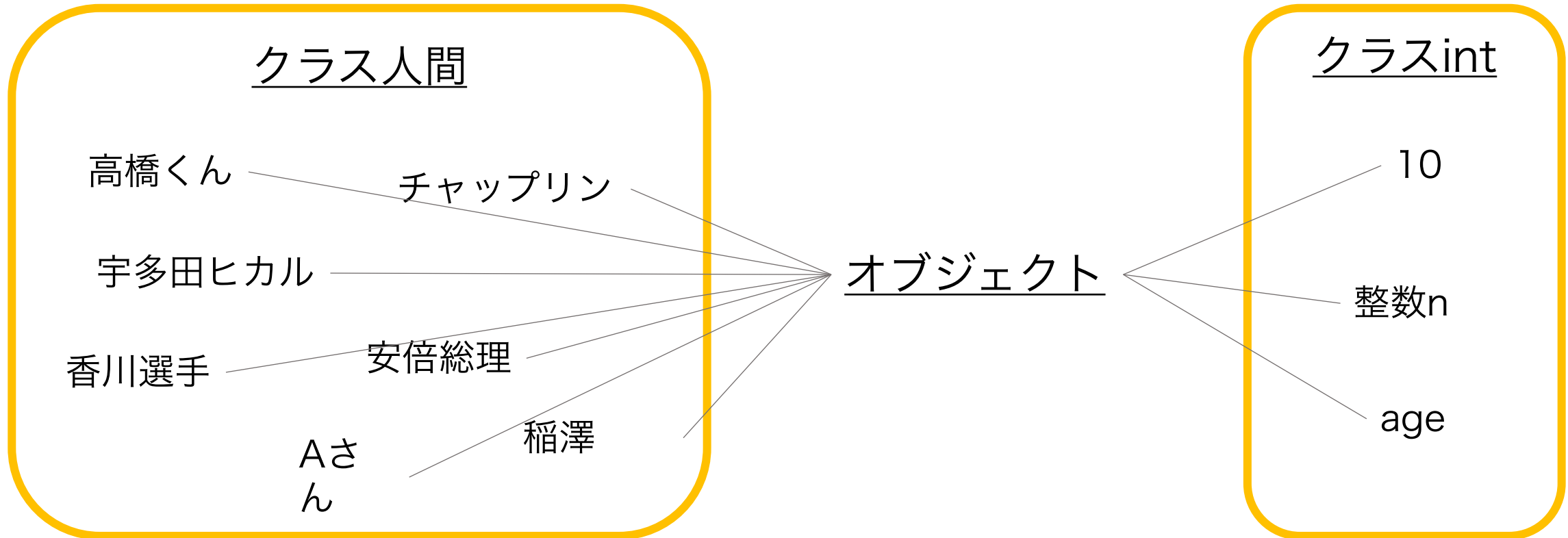
オブジェクトとは



オブジェクトのイメージ

オブジェクト=データ、またはそれを格納する変数

オブジェクトはインスタンスとも言う





オブジェクトの宣言

オブジェクトの宣言の仕方

オブジェクト名 = クラス名;

コード例	説明
<code>i = 1</code>	iはint型のオブジェクト
<code>num = 4.5</code>	numはfloat型のオブジェクト
<code>p = Person()</code>	pはPersonクラスのオブジェクト
<code>ina = Person()</code>	inaはPersonクラスのオブジェクト



インスタンス変数を追加

オブジェクト名.変数名 = 値

print(オブジェクト名.インスタンス変数名)

```
inazawa = Person()
inazawa.name = "inazawa"
inazawa.age = 24
print(inazawa.name)
print(inazawa.age)
```

```
inazawa
24
```

オブジェクトごとに固有の変数を作成できる

inazawaというオブジェクト(値)は、nameとageという変数を所有する



演習問題 2

- 問題1：Animalクラスのオブジェクトcatを宣言しましょう
- 問題2：オブジェクトcatにインスタンス変数nameを追加し、値を適当に代入しましょう

クラスを使う関数



クラスを引数にする

関数の引数にクラスを使う

```
def fanc1(person):  
    print(person.name)  
fanc1(inazawa)  
inazawa
```



クラスを返り値にする

関数の返り値にクラスを使う

```
def fanc2(name):  
    p = Person()  
    p.name = name  
  
    return p  
  
harayama = fanc2("harayama")  
print(harayama.name)  
harayama
```



演習問題 3

- 問題1：Animalクラスのオブジェクトを返す関数を作成しましょう
- 問題2：dogという変数に、問題1の関数の戻り値を代入し、dogというオブジェクトを生成しましょう

コンストラクタとデストラクタ



コンストラクタ

```
class クラス名:  
    def __init__(self):  
        ～関数の中身～
```

```
class Person:  
    species = "Homo sapiens"  
    number = 1  
    def __init__(self):  
        print("コンストラクタだよ")  
inazawa = Person()
```

コンストラクタだよ

コンストラクタはオブジェクトの宣言時に必ず呼び出される関数

必ず引数にselfを持つ

selfは、そのオブジェクト自身を指す



コンストラクタに引数を追加

```
class クラス名:  
    def __init__(self, 引数1, 引数2):  
        self.インスタンス変数1 = 引数1  
        self.インスタンス変数2 = 引数2
```

```
class Person:  
    species = "Homo sapiens"  
    number = 1  
    def __init__(self, name_arg, age_arg):  
        print("コンストラクタだよ")  
        self.name = name_arg  
        self.age = age_arg  
inazawa = Person("inazawa", 24)  
print(inazawa.name)  
print(inazawa.age)
```

```
コンストラクタだよ  
inazawa  
24
```

コンストラクタはオブジェクトの宣言時に必ず呼び出される関数

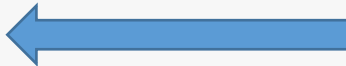
必ず引数にselfを持つ

selfは、そのオブジェクト自身を指す



コンストラクタでクラス変数を操作

```
class Person:
    species = "Homo sapiens"
    number = 1
    def __init__(self, name_arg, age_arg):
        self.name = name_arg
        self.age = age_arg
        Person.number += 1
        self.number += 1
```



この書き方は基本的にNG
numberがインスタンス変数に
変化してしまう

コンストラクタはオブジェクトの宣言時に必ず呼び出される関数

必ず引数にselfを持つ

selfは、そのオブジェクト自身を指す



デストラクタ

```
class クラス名:  
    def __del__(self):  
        ～関数の中身～
```

```
class Person:  
    def __init__(self):  
        print("コンストラクタだよ")  
    def __del__(self):  
        print("デストラクタだよ")  
inazawa = Person()  
print("-----")  
del inazawa
```

コンストラクタだよ

デストラクタだよ

デストラクタはオブジェクトの削除時に呼び出される関数
呼び出される保証がないので、処理の便りにするのは危険



演習問題 4

- 問題1：Animalクラスのコンストラクタにおいて、nameとspecies(種別),ageを引数にし、インスタンス変数に代入しましょう
- 問題2：Animalクラスのデストラクタにおいて、オブジェクトが削除されたことを示す適当な文言をprintしましょう

メソッド



メソッド

```
class クラス名:  
    def 関数名(self,引数1,...):  
        ～中身～
```

```
#メソッド  
class Person:  
    def birthday(self):  
        print(self.name+"の誕生日です")  
        self.age += 1  
  
inazawa = Person("inazawa",24)  
inazawa.birthday()  
print(inazawa.age)  
inazawaの誕生日です  
25
```

メソッドとは、クラス内で定義する関数のこと

クラスメソッドとは、クラスに属している関数のこと(cf クラス変数)

クラスメソッドは、引数にclsをもつ。



クラスメソッド

```
class クラス名:  
    @classmethod  
    def 関数名(cls, 引数1, ...):  
        ～中身～
```

```
#クラスメソッド  
class Person:  
    species = "Homo sapiens"  
    number = 1  
    @classmethod  
    def get_species(cls):  
        print("この種は" + cls.species + "です")  
  
Person.get_species()
```

この種はHomo sapiensです

メソッドとは、クラス内で定義する関数のこと

クラスメソッドとは、クラスに属している関数のこと(cf クラス変数)

クラスメソッドは、引数にclsをもつ。



演習問題 5


- 問題1：Animalクラスにおいて、自分の名前をprintするメソッドを追加しましょう
ヒント：自分の名前は、`self.name`に格納されている
- 問題2：Animalクラスにおいて、`age`(年齢)を一つ上げるメソッドを追加しましょう
- 問題3：問題1・2のメソッドを実行してみましょう
- 問題4：Animalクラスにおいて、クラス変数`number`を1つ上げるクラスメソッドを追加しましょう
ヒント：クラス変数のアクセスは、`cls.number`

カプセル化とPythonにおけるgetter、setter




カプセル化とは

- オブジェクト指向における概念の一つ。
関連する操作や関数などを一つのクラスやモジュールにまとめ、外部には必要な情報のみを提供する。外側にいる使用者に対し必要なインターフェースの身を提供することで、不必要に変数が書き換えられてしまったりすることを防ぐ効果がある。



あの処理をするにはまず
変数aに値を入れてから
bのメソッドを呼び出し
てcのメソッドに…



処理をまとめてある
メソッドに変数だけ渡せ
ば良いんだ！



Pythonにおけるカプセル化

- C++やJavaなどのオブジェクト指向言語では外部から直接アクセスできない変数やメソッドを作成することができる。
- Pythonは非公開の変数やメソッドを作成することはできない！
- 書き手がきちんと理解してプログラムを書くことが求められており、必ずしも他のオブジェクト指向言語と同様にgetter、setterを作ったり、擬似プライベート変数を作ったりする必要はない。



Pythonにおける擬似プライベート1

- Pythonが非公開のオブジェクトを作ることが出来ない点を踏まえた上で、擬似手に非公開オブジェクトを作成する方法は使用されている。
- 変数名の前にアンダーバーを付けて宣言する。
- この方法は動作には全く影響を与えず普通にオブジェクト名._変数名でアクセスすることができるが、読み手や書き手にこのオブジェクトは内部で使用されることを想定していると宣言するために慣習的に使用される。

_変数名

```
class ClassName():  
    ...  
    def __init__(self, arg):  
        ...  
        self._arg = arg
```



Pythonにおける擬似プライベート2

- 変数名の前に2つアンダーバーを付けて宣言する。
- この方法は1の方法と違い擬似的に非公開になっている。
- オブジェクト名.__変数名としてもアクセスできない。
- ただし、あくまで擬似であって実はオブジェクト名._class名__変数名でアクセスできる。
- この方法は名前マングリングと呼ばれある程度意図しないアクセスを防ぐことができるが、本来親と子クラスでの衝突を回避するために使われる。

__変数名

```
class ClassName():  
    ...  
    def __init__(self, arg):  
        ...  
        self.__arg = arg
```



他言語におけるgetterとsetter

- C++やJavaなどで作られるgetterとsetterは下図のようにメソッドを定義する方法である。
- しかし、Pythonではオブジェクトは非公開になっていないのでgetter、setterをこの方法で作っても使用される保証がなく、カプセル化の意味がない。

```
class ClassName():  
    ...  
    def __init__(self, arg):  
        ...  
        self._name = arg  
    ...  
    def set_name(arg):  
        ...  
        self._name = arg  
    ...  
    def get_name():  
        ...  
        return self._name
```



Pythonにおけるgetterとsetter

- ではどうやって呼び出すか？
- 素直に公開属性として宣言し、シンプルに呼び出せば良い。
(これがPython流である)

```
class ClassName():  
    ...  
    def __init__(self, arg):  
        ...  
        self.name = arg  
    ...  
  
sample = ClassName("test")  
sample.name = "newname"
```



ゲッターとセッター

- しかし、値の代入時に特別な振る舞いをしたい(代入される値が0以上であることをチェックするなど)必要な場合にgetterとsetterを定義するにはプロパティを使って以下のように書く。

@property…getterの前につけるデコレータ

@変数名.setter…setterの前につけるデコレータ

```
class ClassName():  
    ... def __init__(self, arg):  
    ...     self._name = arg  
    ...  
    ... @property  
    ... def name(self):  
    ...     return self._name  
    ...  
    ... @name.setter  
    ... def name(self, name):  
    ...     if len(name) > 0:  
    ...         self._name = name  
    ...         return  
    ...  
    ... raise ValueError("1文字以上の名前を入れてください")
```



演習問題 6

- 問題1：Animalクラスにおいて、インスタンス変数ageを擬似プライベートにしましょう
- 問題2：秘匿にされたageに対してプロパティを使ってgetterとsetterを作成しましょう。setterに関しては、代入の際に値がint型で0以上であるかをチェックしましょう。
- 問題3：変数ageを通常の属性アクセスとgetterでのアクセスの両方を使ってアクセスしてみましょう。また問題2で作ったsetterを使用してみましょう。

繼承



継承

```
class 子クラス(親クラス):  
    pass
```

子クラス（派生クラス）は親クラス（基底クラス）のメンバ変数を持つ。

「子クラス」は「親クラス」の一種として考えよう。

例（親クラス：動物、子クラス：ネコ）

例（親クラス：人間、子クラス：博士）

```
#継承  
class NewPerson(Person):  
    pass  
  
new_inazawa = NewPerson("inazawa",24)  
new_inazawa.get_name()
```

コンストラクタだよ

'inazawa'



オーバーライド

- 親クラスから引き継いだメソッドを子クラス的设计に合わせて動作を変更したい際はオーバーライド(上書き)を使う。

```
class 親クラス():  
    def メソッド():  
        動作  
  
class 子クラス(親クラス):  
    def メソッド():  
        新しい動作
```

オーバーライドしている

```
class Parent():  
    ...def speak(self):  
    ...    print("This is Parent class.")  
    ...  
  
class Child(Parent):  
    ...def speak(self):  
    ...    print("This is Child class.")  
    ...  
  
a = Parent()  
c = Child()  
a.speak()  
c.speak()
```

```
This is Parent class.  
This is Child class.
```



子クラスで親クラスのメソッドを呼び出すsuper()

- 親クラスのコンストラクタに加え、子クラス特有のコンストラクタを加えたいときsuper()を使う
これを用いないと、親のコンストラクタが実行されない

```
class 子クラス(親クラス):  
    def __init__(self, 引数1...):  
        super().__init__(引数1...)  
        ～子クラスの特有の処理～
```

```
#super() クラス  
class NewPerson(Person):  
  
    def __init__(self, arg_name, arg_age, ability):  
        super().__init__(arg_name, arg_age)  
        self.ability = ability  
  
new_inazawa = NewPerson("inazawa", 24, "空を飛べる")  
print(new_inazawa.ability)  
print(new_inazawa.get_name())
```

コンストラクタだよ
空を飛べる
inazawa



演習問題 7

- 問題1 : Animalクラスを親クラスとして、それを継承したCatクラスを作成しましょう
- 問題2 : Catクラス独自のコンストラクタとして、ひげ(beard)の有無(True,False)を引数とし、インスタンス変数に追加しましょう
- 問題3 : Animalクラスに文を出力するメソッドを作り、それをCatクラスでオーバーライドしてみましょう。

関数オブジェクト



関数オブジェクトとは

➤ クラスを関数みたいに使うことができる方法のこと

```
class クラス名
    def __call__(self, 引数1・・・):
        ～処理～
```

```
def __call__(self):
    print("関数オブジェクトだよ")

new_inazawa()
```

関数オブジェクトだよ

オブジェクト.メソッドのような形式でなくてもメソッドを実行できる



演習問題 8

- 問題1：Catクラスにおいて、鳴き声(“にゃー”)をprintする関数オブジェクトを作成しましょう



アンケートのお願い

今後の講座のクオリティ向上にご協力お願いします。

<https://seminar.to-kei.net/qt/>

スマホでアンケート回答はこちらから



研修のご依頼・事業の相談お待ちしております。

メールでのお問い合わせはこちら
contact@avilen.co.jp

Webサイトからのお問い合わせはこちら
<https://avilen.co.jp/contact/>