オブジェクト指向プログラミング

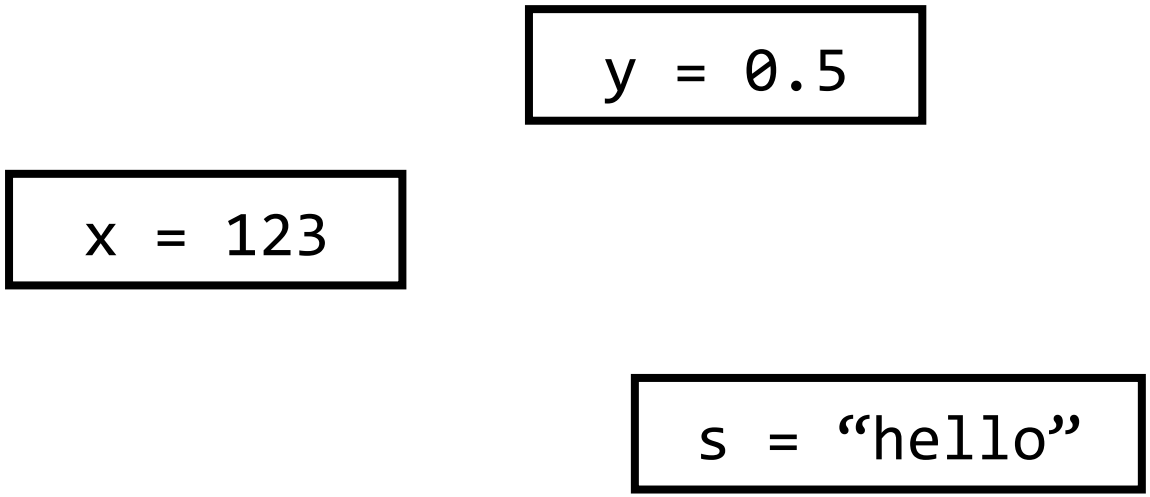
おおざっPython

B4

鈴木正敏

オブジェクト指向を使わない場合

- 数値や文字列などの変数(あるいは配列)を用いたプログラミング
- 1つの変数が、1つの単純なデータを表す



「モノ」を表現するには？

- 実世界のモノは複数の要素からなることが多い
  - 例①: 人間
    - 身長、体重、年齢、… → 数値で表現できる
    - 名前、職業、住所、… → 文字列で表現できる
  - 例②: 銀行口座
    - 口座番号、預金残高、名義人、…
  - 例③: 日付と時刻
    - 年、月、日、時、分、秒、といった数値からなる
- このような「モノ」をどのようなコードで表現すればよいのか？

C言語の場合

- 複数の変数をまとめた「構造体」で表現
  - 構造体の定義:

```
struct person {
    char  name[20];
    int   age;
    floor height;
};
```

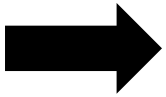
  - 変数宣言・初期化:

```
struct person p1 = {“Taro”, 23, 170.0}
```

  - 値の書き換え:

```
p1.height = 171.0
```

```
(person)
  .name   = (str)
  .age    = (int)
  .height = (floor)
```



```
p1
  .name   = “Taro”
  .age    = 23
  .height = 170.0
```

重要なイメージ

設計図
(1つ)

```
(person)
  .name   = (char)
  .age    = (int)
  .height = (floor)
```

初期化して変数に代入

実体
(いくらでも)

```
p1
  .name   = “Taro”
  .age    = 23
  .height = 170.0
```

```
p2
  .name   = “Jiro”
  .age    = 19
  .height = 165.0
```

オブジェクト指向の用語

クラス
(1つ)

```
(Person)
  .name   = (char)
  .age    = (int)
  .height = (floor)
```

初期化して変数に代入

インスタンス
(いくらでも)

```
p1
  .name   = “Taro”
  .age    = 23
  .height = 170.0
```

```
p2
  .name   = “Jiro”
  .age    = 19
  .height = 165.0
```

それぞれの属性を「インスタンス変数」という

構造体と
クラスの違いは？

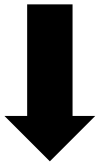
違い①: クラスは「メソッド」も持てる

↑「クラス特有の関数」みたいなもの

クラス
(1つ)

```
(Person)
  .name     = (char)
  .age      = (int)
  .height   = (floor)
  .greet()  = {print “Hello!”}
```

初期化して変数に代入



インスタンス
(いくらでも)

```
p1
  .name     = “Taro”
  .age      = 23
  .height   = 170.0
  .greet()  = {print “Hello!”}
```

```
p1.greet()
```

→ “Hello!” と出力される

インスタンスの属性を参照

クラス
(1つ)

```
(Person)
  .name     = (char)
  .age      = (int)
  .height   = (floor)
  .greet() = {print “My name is ” + self.name + “!”}
```

自分のnameを参照
↓

初期化して変数に代入

インスタンス
(いくらでも)

```
p1
  .name     = “Taro”
  .age      = 23
  .height   = 170.0
  .greet() = {...}
```

```
p2
  .name     = “Jiro”
  .age      = 19
  .height   = 165.0
  .greet() = {...}
```

`p1.greet()` → “My name is Taro!” と出力される

`p2.greet()` → “My name is Jiro!” と出力される

違い②: クラスは「再利用」できる

- Personクラスに属性schoolを追加した Studentクラスを作りたい

Personクラス

```
(Person)
  .name    = (str)
  .age     = (int)
  .height  = (floor)
  .greet() = {print “Hello!”}
```

Studentクラス

```
(Student)
  .name    = (str)
  .age     = (int)
  .height  = (floor)
  .greet() = {print “Hello!”}
  .school  = (str)
```

しかし同じ定義をまた書くのは冗長

クラスの「継承」

親クラス

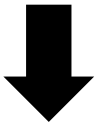
```
(Person)
  .name    = (str)
  .age     = (int)
  .height  = (floor)
  .greet() = {print “Hello!”}
```

子クラス

```
(Student) super: Person
  .name    = (str)
  .age     = (int)
  .height  = (floor)
  .greet() = {print “Hello!}
  .school  = (str)
```

親クラスを指定することで
親クラスの属性が
「継承」される
(親の属性は書かなくていい)

初期化して変数に代入



Studentの
インスタンス

```
s1
  .name    = “Show”
  .age     = 16
  .height  = 158.0
  .greet() = {print “Hello!”}
  .school  = “XY high school”
```

メソッドの「オーバーライド」

親クラス

```
(Person)
  .name     = (str)
  .age      = (int)
  .height   = (floor)
  .greet() = {print “Hello!”}
```

子クラス

```
(Student) super: Person
  .school   = (str)
  .greet() = {print “I am a ” + self.school + “ student!”}
```

親クラスのメソッドが
上書き(オーバーライド)される

初期化して変数に代入

Person、
Studentの
インスタンス

```
p1
  .name     = “Taro”
  .age      = 23
  .height   = 170.0
  .greet() = {...}
```

```
s1
  .name     = “Show”
  .age      = 16
  .height   = 158.0
  .greet() = {...}
  .school   = “XY high school”
```

```
p1.greet()
```
→ “Hello!” と出力される

```
s1.greet()
```
→ “I am a XY high school student!” と出力される

Pythonでは
どう書くのか？

クラスの宣言

クラス名
(大文字で始める)
↓

親クラス
(ないときはobjectを指定)
↓

```
class Person(object):

    def __init__(self, name, age, height):
        self.name = name
        self.age = age
        self.height = height

    def greet(self):
        s = “My name is “ + self.name + “!”
        print s
```

インスタンスを
初期化する時に
呼ばれるメソッド

メソッドの定義

すべてのメソッドには第1引数selfが必要

インスタンスの生成

```
class Person(object):

    def __init__(self, name, age, height):
        self.name = name
        self.age = age
        self.height = height

    def greet(self):
        s = “My name is “ + self.name + “!”
        print s
```

```
p1 = Person(“Taro”, 23, 170.0)
```

← Personクラスのインスタンス p1を生成

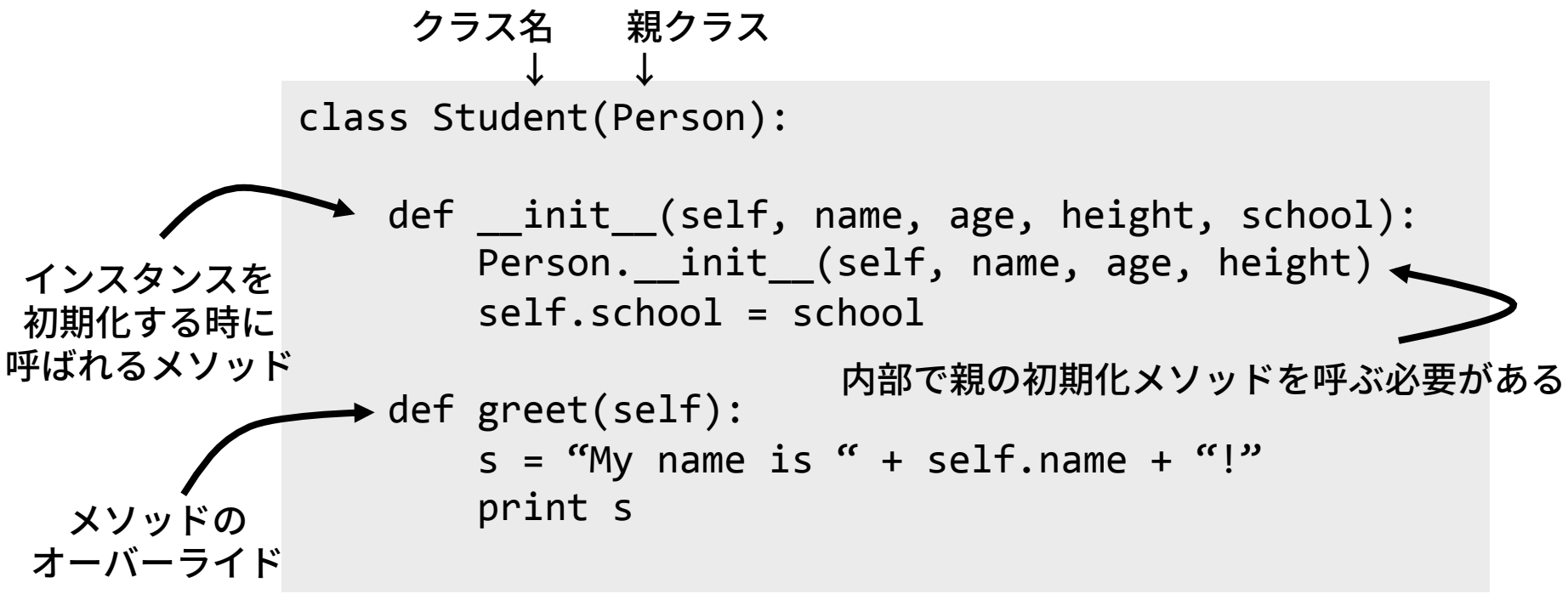
引数にselfはいらない

```
p1.greet()
```

← p1のメソッドgreet()を実行

出力: `My name is Taro!`

クラスの宣言 (継承あり)



すべてのメソッドには第1引数selfが必要

まとめ

- オブジェクト指向によって、
  データを実世界のモノのように
  扱うことができる
- 重要な概念:
  - クラス
  - インスタンス
  - インスタンス変数、メソッド
  - 継承、オーバーライド