Python勉強会@HACHINONE

第8章

クラスと
オブジェクト指向プログラミング

お知らせ

Python勉強会@HACHINOHEでは、ジョン・V・グッターグ『Python言語による
プログラミングイントロダクション』近代科学社、2014年をみんなで勉強しています。

この本は自分で読んで考えて調べると力が付くように書かれています。

自分で読んで考えて調べる前に、このスライドを見るのは、いわばネタバレを
聞かされるようなものでもったいないです。

是非、本を読んでからご覧ください。

オブジェクト指向プログラミング

Python勉強会@HACHINOHE

- 変化に対応できるプログラミングの指針
  - 分割: プログラミに構造を与える
  - 抽象化: 細部を隠す
    - 実装に依存しない
      - 整数、小数、リスト、文字列など、具体的な実装を知らなくても操作できプログラムを作れた
      - 実装を変えても、抽象が同等なら、そのまま動く
- クラスによるデータ抽象化
  - クラスは「データとそれを処理するメソッド」を定義したもの

Pythonのクラス定義とインスタンス化

Python勉強会@HACHINOHE

```
class クラス名(親クラス):
    """クラスの説明"""

    クラス変数名 = ...

    def __init__(self, ...): # コンストラクタ
        self.インスタンス変数名 = ...
        self.__プライベート変数名 = ... # _クラス名__プライベート変数名

    def メソッド名(self, ...):
        ...

    @classmethod
    def クラスメソッド名(...):
        ...

インスタンス = クラス名(...)
```

クラス定義の例

Python勉強会@HACHINOHE

```
# -*- coding: utf-8 -*-
class IntSet(object):
    """IntSetは整数の集合である"""
    # ここに実装に関する情報を書く(抽象化の情報ではない)。
    # 集合は、int型の要素からなるリストself.valsで表現される。
    # リストself.valsには同じ要素は複数含まれない。

    def __init__(self):
        """整数の空集合を生成する"""
        self.vals = []

    def insert(self, e):
        """eをint型とする。
           eがselfに含まれていなければ挿入する"""
        if not e in self.vals:
            self.vals.append(e)

    def member(self, e):
        """eをint型とする。
           eがselfに含まれればTrueを、なければFalseを返す"""
        return e in self.vals

    def remove(self, e):
        """eをint型とする。
           eをselfから削除する。
           eがselfに含まれなければValueError例外を発生させる。"""
        try:
            self.vals.remove(e)
        except:
            raise ValueError(str(e) + 'は含まれていません')

    def getMembers(self):
        """selfに含まれる要素のリストを返す。
           要素の順番は保証しない。"""
        return self.vals[:]

    def __str__(self):
        """selfの文字列表現を返す"""
        self.vals.sort()
        result = ''
        for e in self.vals:
            result = result + str(e) + ','
        return '{' + result[:-1] + '}' # -1としたのは最後のカンマを取り除くため
```

IntSet
vals:リスト
__init__() insert(e) member(e): 整数 remove(e) getMembers(): リスト __str__(): 文字列

インスタンス化、属性参照

Python勉強会@HACHINOHE

- インスタンス化
  - クラスのインスタンス(具現体)を生成すること

```
s = IntSet() # IntSetクラスのインスタンスを生成し、sと結びつける
```

  - インスタンス化のとき、インスタンスを生成後、続いて __init__メソッドが呼び出される
- 属性参照
  - クラスに関連付けられた属性に「.」演算子でアクセスできる

```
s.member # IntSetクラスのインスタンスsに関連付けられたメソッドmemberを参照
```

メソッドの第一引数、データ属性

Python勉強会@HACHINOHE

- インスタンスからのメソッド呼び出し
  - 第一引数を省略。第一引数には、暗黙にインスタンス自身が渡される
  - 通常、インスタンス自身の仮引数名はselfを使う
- データ属性
  - 以下の例の__init__では、空のリスト型オブジェクトを生成し、インスタンスの一部分となるvalsに関連付けている

```
def __init__(self):
    """整数の空集合を生成する"""
    self.vals = []
```

  - valsのようなオブジェクトをインスタンスの「データ属性」という
  - インスタンスのデータ属性は(メソッドも)、インスタンスごとに別々の中身

抽象データ型の実装の3つの要素

Python勉強会@HACHINOHE

- メソッドの実装
- さまざまな値をたばねるデータ構造
- 表現不変量
  - データ操作に関する重要なルール
  - IntSetの場合、valsが同じ値を複数含まないこと

さまざまな特別なメソッド

Python勉強会@HACHINOHE

- __init__ インスタンス化で呼び出される
- __str__ str(インスタンス)で呼び出される
- __hash__ id(インスタンス)で呼び出される
- __eq__、__lt__ オブジェクトの比較で呼び出される

多重定義 オーバーロード

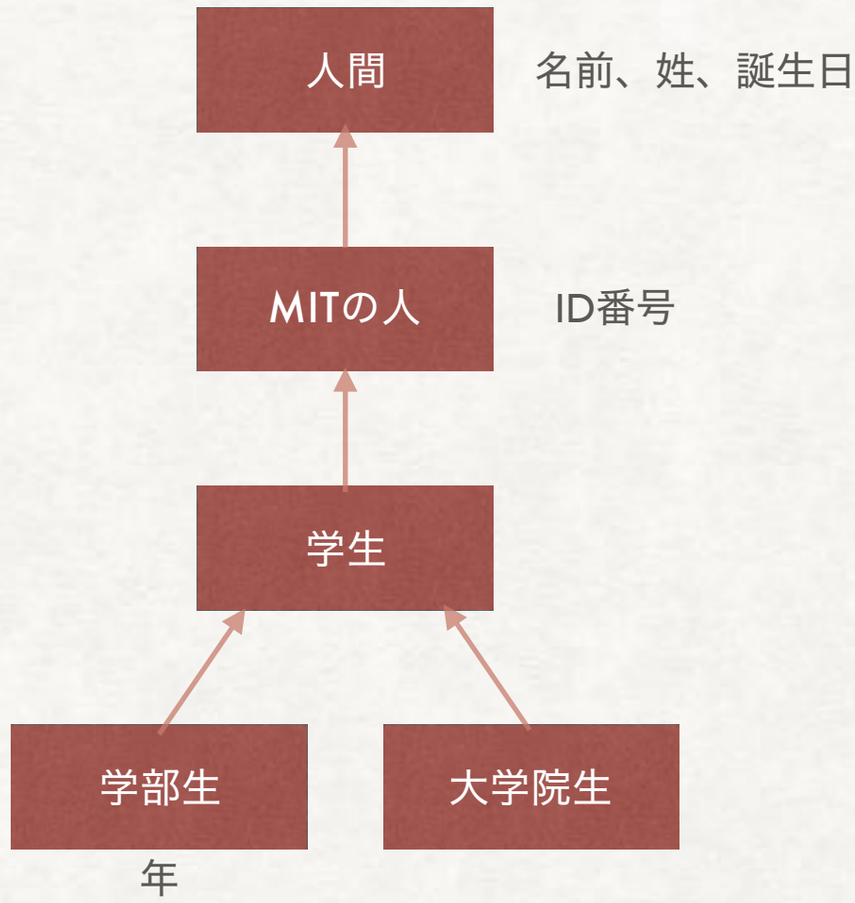
Python勉強会@HACHINOHE

- __lt__ は「<」をオーバーロードしている
- オーバーロードとは、型に応じて呼び出すメソッドを多重に定義すること

継承

Python勉強会@HACHINOHE

- あるクラス(スーパークラス)を元に、新たなクラス(サブクラス)を作る機能を継承という
- サブクラスでは、スーパークラスのデータ属性やメソッドを利用できる
- サブクラスで、スーパークラスのメソッドをオーバーライド(上書き)することもできる
- リスコフの置換原則
  - スーパークラスのインスタンスを用いたコードは、サブクラスでも正しく動作しなければならない

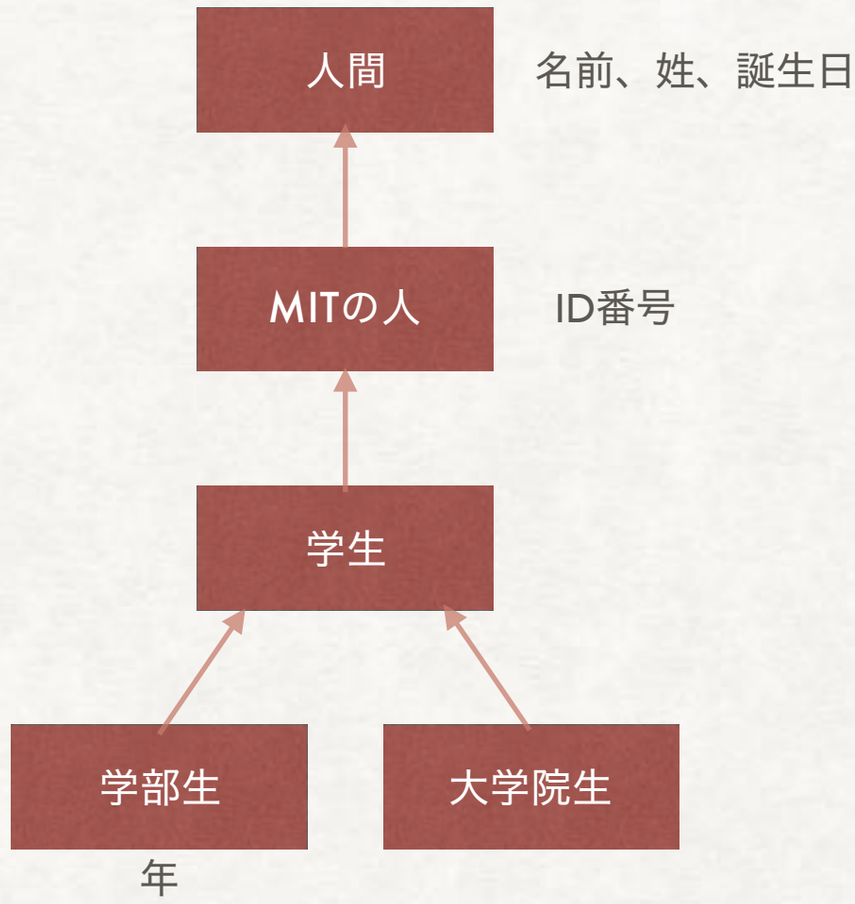


is-a関係

Python勉強会@HACHINOHE

- is-a関係
  - MITの人は、人間の一種
- isinstance()で、インスタンスかどうかを調べられる
- is-a関係がある場合もTrue

```
isinstance(MITの人のインスタンス，人間) # True
```



カプセル化と情報隠蔽

Python勉強会@HACHINOHE

- カプセル化
  - データ群とその操作をひとまとめにすること
  - オブジェクトを使って実現されている
- 情報隠蔽
  - オブジェクトやクラスの内部を隠して、本当に外部からアクセスするべきものだけを見せる
  - 通常、データ属性は直接見せないで、セッターやゲッターというメソッドを用意する
  - Javaでは、private、protected、publicなどの指定ができる
  - Pythonでは…。しかし、それを意識しよう

ジェネレーター

Python勉強会@HACHINOHE

- 集合をくりかえし処理するイテレータを作成するツール
  - さまざまな集合があり、個々の要素にアクセスする方法は実装によりさまざま
  - それらをくりかえし処理する統一的な方法があると便利

- Pythonでは、yieldを使うと、集合をくりかえし処理でき、しかも局所変数は処理をはじめた段階の結びつきのまま

```
    def getStudents(self):
        """成績ブックに収められた学生のリストを返す"""
        if not self.isSorted:
            self.students.sort()
            self.isSorted = True
        # return self.students[:] # 学生のリストのコピーを返す
        for s in self.students:
            yield s
```