Pythonプログラミング入門

2018年3月17日（土）第６回～第１０回
講師：渡邉 貴之

本日の内容

- タイムテーブル

第6回	**13:00 ～ 13:45**
	クラスとオブジェクト
第7回	**13:45 ～ 14:30**
	モジュール
第8回	**14:30 ～ 15:15**
	データの集計とグラフ作成1
第9回	**15:15 ～ 16:00**
	データの集計とグラフ作成2
第10回	**16:00 ～ 16:45**
	データの集計とグラフ作成3

※途中、休憩時間を適宜取ります

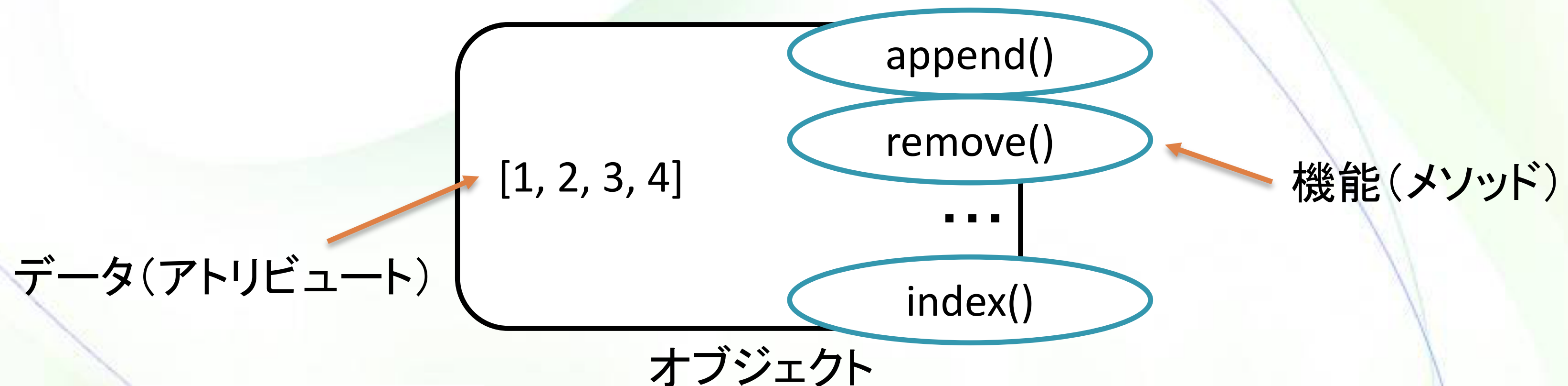
第6回　クラスとオブジェクト

目次

- オブジェクト
- クラス
- dateオブジェクトの使用例
- クラスを作る
- インスタンスの初期化とアトリビュート
- self以外の引数付きの初期化
- クラスアトリビュート
- アトリビュート補足
- アトリビュートの追加
- アトリビュートの追加制限
- アトリビュートの参照制限
- getterとsetter
- property関数
- 課題6

オブジェクト

- Pythonでは、すべての組み込みデータ型はオブジェクトの特徴を持っています。
  - オブジェクトは、データ（アトリビュート）と、データに対する機能（メソッド）を併せ持った存在です。
  - 例：リスト　a = [1, 2, 3, 4]




```
a = [1,2,3,4]
print(a)
del a[2]
print(a)
a.remove(1)
print(a)
```

del文を使ってインデックス2を削除

removeメソッドを使って値1を削除

```
[1, 2, 3, 4]
[1, 2, 4]
[2, 4]
```

クラス

- Pythonでは組み込み型以外の様々なオブジェクトが利用できます。
  - クラスはオブジェクトの設計図であり、組み込み型以外のオブジェクトは、クラスをもとに生成します。
  - クラスをもとにしたオブジェクト生成の構文例

```
インスタンス変数名 = クラス名(引数)
```

  - 例：日付を管理するためのdateクラス
    - dateクラスは、Pythonの標準モジュールであるdatetimeモジュールに備わっていますが、そのままでは使用できないので、事前にimportする（読み込む）必要があります。

```python
from datetime import date

a = date(2018,3,17)
print(a)
```

```
2018-03-17
```

  - インスタンスとは？
    - クラスをもとに生成した個々のオブジェクトを、クラスを具現化した存在という意味を明確化するためにインスタンス（実体）と呼ぶ場合があります。

dateオブジェクトの使用例

- dateクラスのオブジェクトには、便利なメソッドや演算が備わっています。
  - 曜日を数字で取得するweekdayメソッド
    - 月曜日を 0、日曜日を 6 として、曜日を整数で取得できます。
  - 日数計算が可能
    - インスタンスの差(ー)で日数が計算可能です。

```
from datetime import date

a = date(2018,3,17)
print(a)
b = date(2018,3,10)
print(b)
print(a.weekday())
print(a - b)
```

曜日の取得

日数計算

```
2018-03-17
2018-03-10
5
7 days, 0:00:00
```

クラスを作る

- Pythonに標準で備わっているクラスでは不足している場合には、自作のクラスを定義したり、既存のクラスを拡張したりすることができます。
- クラスを定義するにはclass文を使います。
  - 構文
    class クラス名:
    ｲﾝﾃﾞﾝﾄ クラスの内容

```
class MyClass:
    pass

a = MyClass()
print(a)
```

passは何もしないという予約語

オブジェクト（インスタンス）の生成

```
<__main__.MyClass object at 0x10724e940>
```

インスタンスの初期化とアトリビュート

- インスタンスを生成する際に、何らかの初期化を行いたい場合には、初期化メソッド__init__(self)を用意します。
  - 初期化メソッドはインスタンス生成時に自動的に呼び出されます。
  - 初期化メソッドでは、インスタンスが持つデータ（値）として、アトリビュートの設定を行います。
  - selfとは、インスタンスの自分自身を表す予約語で、必ずメソッドの第1引数として渡します。また、アトリビュートの前にself.として添えます。
- 構文

class クラス名:
[インデント] def __init__(self):
[インデント] [インデント] self.アトリビュート = 値

```
class Person:
    def __init__(self):
        self.weight = 60
        self.height = 160

a = Person()
print(a.weight)
```

```
60
```

アトリビュートはインスタンス変数名.アトリビュート名でアクセスができます。

self以外の引数付きの初期化

- 初期化メソッドのselfに続いて引数を置くことで、インスタンス生成時にデータを渡すことができます。

```
class Person:
    def __init__(self, w, h):
        self.weight = w
        self.height = h

a = Person(65, 155)
print(a.weight)
```

```
65
```

self以外にwとhを取る初期化メソッド

インスタンス生成時には、self以外を渡す

```
class Person:
    def __init__(self, w=60, h=160):
        self.weight = w
        self.height = h

a = Person()
print(a.weight)
b = Person(70)
print(b.weight)
```

```
60
70
```

デフォルト値の設定も可能

デフォルト値が設定してあれば、
引数の省略も可能

クラスアトリビュート

- クラス自体が持つデータとして、クラスアトリビュートを設定できます。
  - クラスアトリビュートはclass宣言の直下に記述します。
  - クラスアトリビュートにアクセスするには、クラス名.アトリビュート名としてアクセスします。
  - 例：クラスアトリビュートnumを使ってインスタンス数をカウントする

```
class Person:
    num = 0
    def __init__(self, w=60, h=160):
        self.weight = w
        self.height = h
        Person.num += 1

print(Person.num)
a = Person()
b = Person()
print(Person.num)
```

```
0
2
```

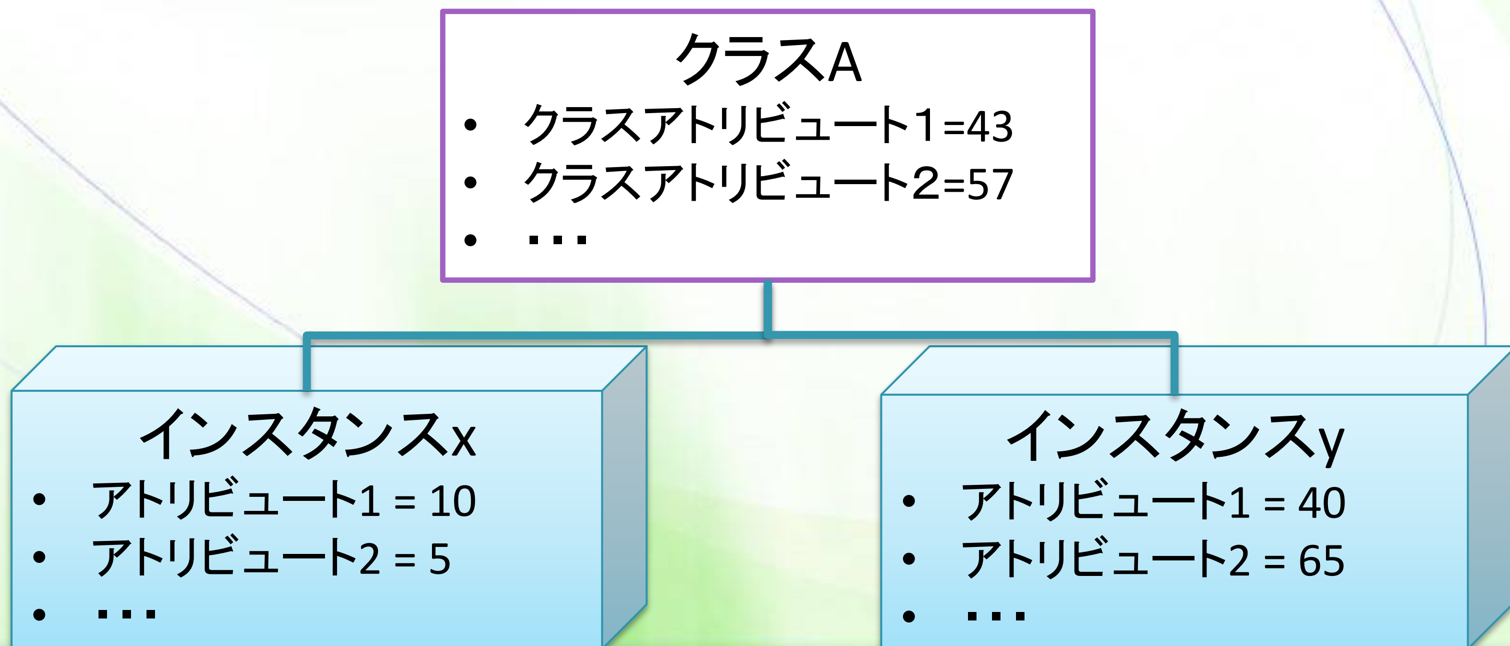
クラスアトリビュートnumを用意

インスタンスを生成するたびにnumを増加

クラスアトリビュートnumの表示

アトリビュート補足

- アトリビュートには、各インスタンスが持つものと、クラス側で共通に持つものの2種類があることに注意してください。
  - クラスアトリビュートは、通常、「クラス名.クラスアトリビュート名」としてアクセスしますが、「インスタンス変数名.クラスアトリビュート名」と記述した場合でも、インスタンス側に同名のアトリビュートがなければ、クラスアトリビュートの値を参照することができます。
  - 一方、「インスタンス変数名.クラスアトリビュート名 ＝ 値」と代入を記述すると、インスタンス側にクラスアトリビュートと同名のアトリビュートが別途作成されます。



アトリビュートの追加

- インスタンスのアトリビュートは、インスタンス生成後いつでも追加できます。
  - アトリビュートの追加の構文

インスタンス変数名. アトリビュート名 = 値

```
class Person:
    def __init__(self, w=60, h=160):
        self.weight = w
        self.height = h

a = Person()
a.name = "太郎"
print(a.name)
```

```
太郎
```

aにnameを追加し"太郎"を代入

アトリビュートの追加制限

- インスタンスが持つアトリビュートを、指定のものだけに制限したい場合は、__slots__というクラスアトリビュートを記述しておきます。
  - 構文

__slots__ = [許可するアトリビュート名のリスト]

許可するアトリビュート名のリスト

```
class Person:
    __slots__ = ["weight", "height"]
    def __init__(self, w=60, h=160):
        self.weight = w
        self.height = h

a = Person()
a.name = "太郎"
print(a.name)
```

nameは許可していないのでエラーとなる

```
---------------------------------------
AttributeError
```

アトリビュートの参照制限

- シナリオ
  - 例えば、エアコンを制御するインスタンスがあったとします。温度設定を管理するアトリビュートがあったとして、その値を勝手に100℃などに書き換えられてはエアコンが異常な状態になってしまいます。
- カプセル化
  - アトリビュートの値をインスタンスの外部から勝手に書き換えることを制限したり、参照を制限することをカプセル化などと呼びます。
- Pythonでは、アトリビュート名の前に__を付けることで、外部からの参照を制限することができます。

```
class Person:
    def __init__(self, w=60, h=160):
        self.__weight = w
        self.__height = h

a = Person()
print(a.__weight)
```

```
---------------------------------------
AttributeError
```

アトリビュート名の前に__をつける。

外部からは参照できないのでエラーとなる

getterとsetter

- アトリビュートをカプセル化した場合は、メソッドを使って値を参照したり設定をします。値を参照するメソッドをgetter、設定するメソッドをsetterと呼びます。
- メソッドを介して値を設定することで、異常な値の設定を防ぐことができます。

```
class Person:
    def __init__(self, w=60, h=160):
        self.__weight = w
        self.__height = h
    def getWeight(self):
        return self.__weight
    def setWeight(self, w):
        if w > 300:
            print("異常な体重です。")
        else:
            self.__weight = w

a = Person()
a.setWeight(500)
print(a.getWeight())
```

```
異常な体重です。
60
```

getterの例

setterの例（異常値のチェック）

setterを使って体重を設定

property関数

- getterやsetterを、あたかも直接アトリビュートを操作しているように呼び出すしくみがpropertyです。
  - 構文

（頭の__を取った）アトリビュート名 = property（getter, setter）

```
class Person:
    def __init__(self, w=60, h=160):
        self.__weight = w
        self.__height = h
    def getWeight(self):
        return self.__weight
    def setWeight(self, w):
        if w > 300:
            print("異常な体重です。")
        else:
            self.__weight = w
    weight = property(getWeight, setWeight)

a = Person()
a.weight = 500
print(a.weight)
```

```
異常な体重です。
60
```

property関数を使って__weightアトリビュートを操作するgetterとsetterを登録

あたかも直接操作しているかのようにgetterやsetterを呼び出せる

課題6

- 下記のPersonクラスに、bmiを計算するbmiメソッドを完成させてください。

```
class Person:
    def __init__(self, w=60, h=160):
        self.__weight = w
        self.__height = h

    def bmi(self):
        return [ここを考える]

a = Person(50, 150)
print(a.bmi())
```

```
22.2222222222222
```