クラスのつくり方（２）（継承の利活用）

おさらい

先週の授業では，最小限の湯沸しポットクラスに少しずつ機能を追加していくと，最終的に以下のようなソースコードになります．

```
class Kettle(object):

    def __init__(self, maxw=1000.0):
        self.water= 100.0
        self.temp= 20.0
        self.maxwater= maxw


    def putWater(self, amount=100, temperature=20):
        print("水を入れます。")
        maxamount= self.maxwater-self.water
        if(maxamount<amount):
            netamount= maxamount
            print("お湯がいっぱいです。")
        else:
            netamount= amount
        self.temp= (self.water*self.temp+netamount*temperature)/(self.water+netamount)
        self.water= self.water+netamount

    def printStatus(self):
        print(f"容量{self.maxwater:.1f}mlのポットには{self.temp:.1f}℃のお湯が\
        {self.water:.1f}mlはいっています。")


    def heat(self):
        print("水をあたためています。")
        self.temp= self.temp+1000.0/self.water*25.0
        if(self.temp>100):
            self.temp=100
            print("お湯が沸きました")

    def pour(self):
        print(f"{self.temp:.1f}℃のお湯を{self.water:.1f}ml注ぎました．")
        self.water= 0
        self.temp= 20


if __name__ == '__main__':
    k= Kettle(500)
    k.printStatus()
    k.putWater(200, 25)
    k.printStatus()
    k.putWater(300, 10)
    k.printStatus()
    k.heat()
    k.printStatus()
    k.heat()
    k.pour()
```

これを実行すると，以下の結果が得られます．

容量500.0mlのポットには20.0℃のお湯が100.0mlはいっています。
水を入れます。
容量500.0mlのポットには23.3℃のお湯が300.0mlはいっています。
水を入れます。
お湯がいっぱいです。
容量500.0mlのポットには18.0℃のお湯が500.0mlはいっています。
水をあたためています。
容量500.0mlのポットには68.0℃のお湯が500.0mlはいっています。
水をあたためています。
お湯が沸きました
100.0℃のお湯を500.0ml注ぎました．

水を入れるだけでなく，容量を決めたりお湯を作ったり注ぐことができるようになりました．

今日の話はこの続きです．

クラスの継承 -ポットにもっと機能を！-

できることが増えたとはいえ，これまでは，湯沸しポットがそれとして果たすべき機能の最小限を実現したという感じでした．

これからは，湯沸しポットに様々な付加価値をつけることを考えるとしましょう．たとえば，

- 使った水の量や，電気の使用量（カロリー）がわかる．
- 浄水機能をつける．ブリタ的なカートリッジ式とか．
- さらに，コーヒー豆を入れられるようにして，コーヒーメーカーにする．
- 使った時間が記録される．高齢者の見守り役的な．

など，いろいろ考えられます．しかし，これまでの機能追加の方法を用いてそれぞれ書こうとすると，毎回，Kettle.pyクラスのソースコードをコピーして書き 直してと，なんだか冗長で，賢い感じがしませんね．

それは，これらは，それぞれ当初の湯沸しポットの機能を踏襲した上で，さらに機能を特化した新たなポットであること，また，そのポットの部分はどれも共通で良いと 考えられるからでしょう．つまり，大元のポットから，それぞれの方向に機能をより特化したものということです．

このように，ある既存のクラスを，さらに機能を付け加えたりして，より特化したクラスを作りたい場合には，「クラスの継承」という仕組みがあります．

継承に基づくクラスの定義は，以下の通りに継承するクラスを冒頭で指定するだけです．それだけで，そのクラスの機能をあらかじめ持っているとみなされ，我々 は，そこからの追加分や変更分だけを記述すればよいことになります．

```
class サブクラス名(スーパークラス名):
    クラスの内容の追加や更新に関する記述
    ・・・
```

では例として，Kettleクラスを継承して，簡単なコーヒーメーカー機能がついたCoffeeMakerクラスを作ってみましょう．以下のように機能を考えま す．

- 入れたコーヒー豆の量（g）を保存するインスタンス変数beans
- 初期化するメソッド__init__(self, maxwater)．
- 豆を引数でしていた任意の量（g）だけ入れられるメソッドputBeans(self, b)
- いま入っているお湯と豆でコーヒーを入れるメソッドbrew(self)．豆が10g以上入っていて，温度が90度以上なら，いま入っているお湯で コーヒーを作ることができる．
- 現在のコーヒーメーカーの状態を表示するようにprintStatus(self)を改変．

例えばこんな風にかけるでしょう．以下をCofeeMaker.pyという名前でKettle.pyと同じディレクトリ（pythonfilesフォルダ） に保存してください．

```python
#Kettle.pyを読み込む。その中身は”Kettle.対象”でアクセス。
import Kettle


class CoffeeMaker(Kettle.Kettle):
    def __init__(self, maxwater):
        super().__init__(maxwater)
        self.beans= 0

    def putBeans(self, b):
       self.beans+= b
       print(f"コーヒー豆を{b:.1f}g入れました。")

    def brew(self):
        if(self.beans>=10 and self.temp>=90):
            super().pour()
            self.beans= 0
            print("おいしいコーヒーが出来上がりました． ")
        else:
            print("コーヒー豆が足りないか，温度が適切ではありません． ")

    def printStatus(self):
        print(f"容量{self.maxwater:.1f}mlのコーヒーメーカーには{self.temp:.1f}℃のお湯が\
```

```
        {self.water:.1f}ml, コーヒー豆が{self.beans:.1f}gはいっています。")


if __name__ == '__main__':
    c= CoffeeMaker(1000)
    c.putWater()
```

- まず注目すべきは冒頭の"import Kettle"．これまでランダムの関数などを使うためにimport文をおまじないのように使って来たけど，ここで意味がはっきりわかります．importは他の Pythonのスクリプト（ソースコード）をモジュールとして読み込みます．読み込んだモジュールはモジュール名（ファイル名）にドットをつけてその中身 の変数やクラス定義などを参照できます．
- その次のクラス定義を見ると，確かに上記のコーヒーメーカーとしての追加文だけが記述されています．
- ただし，見慣れない記述が一つあります．それは"super()"です．この関数は，継承したクラス（スーパークラス）を参照します．定義の中で，継 承したクラスのメソッドを使いたい場合にドットをつけて利用します．
- 例えば，この場合初期化する__init__メソッドは，基本的にはスーパークラスと同じくポットの容量を初期化し，その上で新しいインスタンス変数 beansを定義すればよいので，上記のようにsuper().__init__(maxwater)がつかえます．
- 同じように，brewの中でもsuper.pour()を使って，記述を端折っています．また，この中でKettleクラスのインスタンス変数 tempをこのクラスにも存在しているものとして使っています．
- 最後に，大元のKettleクラスにあったprintStatus()メソッドと同じ名前のメソッドを定義しています．これをメソッドのオーバーライ ドと呼んでいて，メソッドの定義が上書きされます．
- クラス定義に続いて，"if __name__ == '__main__':"という見慣れない条件文があります．これは，（やや端折った説明ですが)"このモジュールを直接実行した場合に条件が真となり，それ以降のインデ ントされたブロックが実行される．"ということを示しています．この例ではコーヒーメーカーを一つ作ったあと，水を入れる文が書かれていま す．putWaterメソッドはCofeeMakerクラスの定義の中にはありませんが，Kettleクラスと同様に使えることが，実行するとわかりま す．

（練習１）上記のように記述すると，Kettleクラスのインスタンス変数やメソッドも使えるCoffeeMakerクラスが出来上がります．以下の出力結 果を得られるように，"if __name__ == '__main__':"以下のブロックに文を追加して，おいしいコーヒーを入れてみましょう．

```
水を入れます。
容量1000.0mlのコーヒーメーカーには20.0℃のお湯が200.0ml，コーヒー豆が0.0gはいっています。
コーヒー豆が足りないか，温度が適切ではありません．
コーヒー豆を20.0g入れました。
容量1000.0mlのコーヒーメーカーには20.0℃のお湯が200.0ml，コーヒー豆が20.0gはいっています。
コーヒー豆が足りないか，温度が適切ではありません．
水をあたためています。
お湯が沸きました
容量1000.0mlのコーヒーメーカーには100.0℃のお湯が200.0ml，コーヒー豆が20.0gはいっています。
100.0℃のお湯を200.0ml注ぎました．
おいしいコーヒーが出来上がりました．
```

（練習２）Kettle.pyの中の，"if __name__ == '__main__':"をとって実行してみましょう．どんなことがおきるか，確認しましょう．

（練習３）上の例の浄水機能つきのポットを表すクラスを，Britaという名前でKettleクラスを継承して作ってみましょう．CoffeeMaker モ ジュールのように使用例もつけてみましょう．

- cartridgeという名前のインスタンス変数を新たに定義し，現在までにputWaterメソッドで入れた水の総量（ml）を保存する．
- catridgeの量が1000（とかなんでもいい）mlを超えると，pourメソッドで注いだあとに"あれ，なんかにおうぞ？？"と表示する．
- カートリッジを交換するメソッドchage(self)を新たに定義し，実行するとcartridgeが0になり，そのことを表示する．
- 浄水機能の状態も含めて現在の状態を表示するようにprintStatus()を改変．

エラー（すでに紹介しているけれど復習）

上 のプログラムでは，putBeansメソッドの中では，豆の量を数値で引数として受け付けています．このとき，たとえば数値でない文字列が入力されていた 場合は，beansに割り当てられた数値と文字列を足し算しようとして，通常「TypeError」と呼ばれるエラーが出力されます．

```
>>> c=CoffeeMaker(1000)
>>> c.putBeans(20)
コーヒー豆を20.0g入れました。
>>> c.putBeans("まめ")
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    c.putBeans("まめ")
  File "C:/python2018/CoffeeMaker.py", line 11, in putBeans
    self.beans+= b
TypeError: unsupported operand type(s) for +=: 'int' and 'str'
```

多 くの場合，上のように一番下のメッセージから見ていくと，「？？？Error」という欄が見つかり，エラーの内容が示されます．その上に，どの場所でエ ラーが起きたかなどがさかのぼって書かれているので，問題を見つける手掛かりになります．

よく目にするエラーには例えば次のものがあります．

ImportError: No module named	そんな名前のモジュールはないよ
list index out of range	リストの添え字が範囲外だよ
takes exactly (x) arguments ((?) given)	x個の引数を取るはずなのに，?個いれられてるよ
NameError: name 'a' is not defined	a という名前（変数）は定義されてないよ
ValueError: invalid literal for int() with base 10	文字列を数値（この場合は整数）に変換しようとしたけど無理です
SyntaxError: invalid syntax	文法が間違ってるよ
AttributeError: ? instance has no attribute	?というインスタンス（オブジェクト）はその属性（インスタンス変数やメソッド）を持ってないよ
AttributeError: class ? has no attribute 'abc'	?というクラスにはabcという属性はないよ
unsupported operand type(s) for +	演算しようとしたけど+は定義されていないよ
IndentationError: expected an indented block	インデントしたブロックがあるはずなのにないのでおかしいよ．
TypeError: can only concatenate list (not "int") to list	リストと整数は結合できないよ
KeyError: 'abc'	absは辞書のキーにないよ
ZeroDivisionError	ゼロで割り算しちゃだめだよ

（http://nonbiri-tereka.hatenablog.com/entry/2014/06/02/083206　を参考）

例外

こ の場合，エラーがでてとまらずに，その入力を受け付けずにおかしな値が入力されていることを示すのが望ましい処理と言えます．

こ のような「通常想定内のことをしてくれている場合は問題ないのだけど，変なことをされる場合には対処しないといけない」ことはよくあります．Pythonに は，「例外処理」といって，ある特定のエラーが出る状況が生じた場合に，即座に処理を別の場所に移す構文「try ... except ...」があります．

try:
　例外処理の対象となる処理
　...

except エラー名:
　上のブロック実行中にエラーの状態に至った時に実行する処理
　...

上の例では，try:で始まるブロックを実行し，もしそのブロックの中でexceptで指定したエラーが起きた場合，except:以下のブロックに実行が うつされます．例えば，putBeansメソッドは以下のように処理することが考えられます．

```
def putBeans(self, b):
    try:
        self.beans+= b
        print("コーヒー豆を{0:.1f}g入れました。".format(b))
    except TypeError:
        print("すみません，よくわかりません．")
```

上記の改変を行って，意図した例外処理が行われるか，確認してみましょう．

（クラスの継承の利活用に関する課題）