# プログラミング実習I クラス5 (井村担当)

知能・機械工学課程 井村 誠孝

m.imura@kwansei.ac.jp

# 今回の内容

- 前回


関数は既に活用している．
例えば print(), len(), ...
自分で関数を
作ってみよう．

- 今回


データ型は既に活用している．例えば turtle.Turtle, ...
自分でデータ型を
作ってみよう．

# データ型の復習

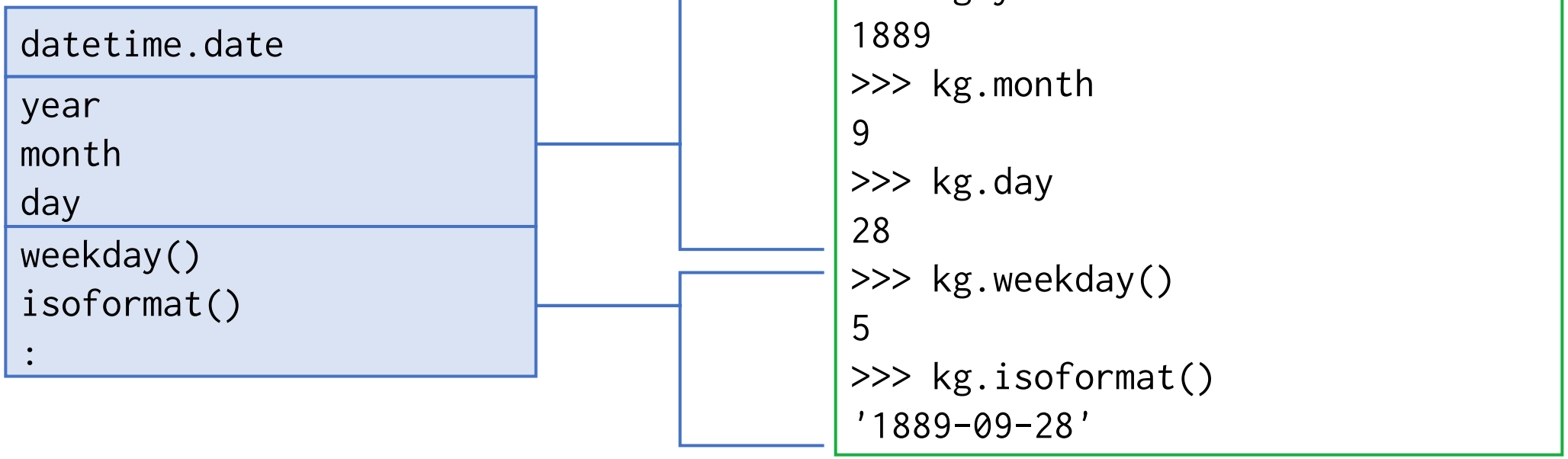
- データには型がある．
- データ型は2種類に分けられる．
  - 組み込み型
    - 例: 整数型(int)，浮動小数点型(float)，文字列型(str)，リスト型(list)，辞書型(dict) など
    - 組み込み型はいきなり使える．
  - 通常の型
    - 例: 日付を扱うdatetime.date型，タートルグラフィクスを扱うturtle.Turtle型 など
    - 通常の型は初期化メソッドを実行してから使う．

```
>>> i = 1
>>> type(i)
<class 'int'>
```

```
>>> import turtle
>>> kame = turtle.Turtle()
>>> type(kame)
<class 'turtle.Turtle'>
```

# データ型の構成

- データ型は，そのデータが表している内容と関連するデータ属性(attribute)とメソッド(method)から構成される．
  - データ属性: データ型と関連付けられた変数
  - メソッド: データ型と関連付けられた関数
- 例: datetime.date型

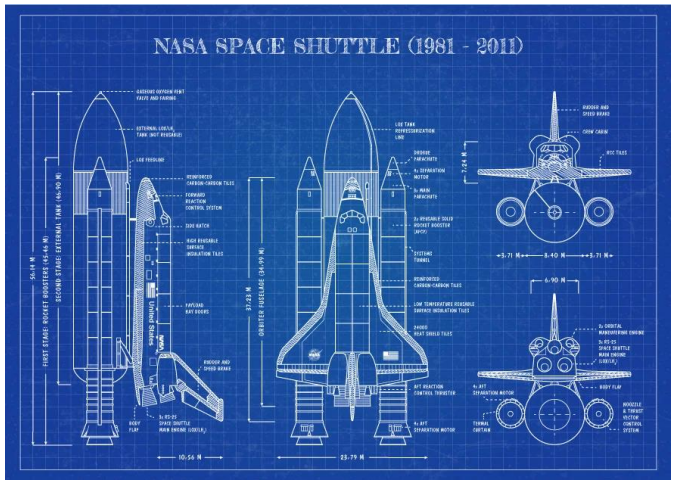


# オブジェクト指向プログラミング

- ある対象(オブジェクト)に関する情報(データ属性)と処理(動作，メソッド)をまとめてデータ型とする．
  - オブジェクトという言葉はあちこちで(節操なく)使われる傾向があるため，どういう意味で使われているか，注意が必要．
- データ型の設計図をクラスと呼ぶ．実際に使うためには実体を作る必要があり，実体のことをインスタンスと呼ぶ．
- 既にあるクラスを再利用し，一部を変更・拡張することで新たなクラスを作ることができる．この機能を継承と呼ぶ．
- クラスを構成する情報と処理のうち，外部に公開する必要があるものを選び，他は隠蔽する(カプセル化)．
  - カプセル化はプログラムの再利用性を高めるために重要であるが，Pythonの言語仕様には組み込まれていないため，プログラマが意識的に行う必要がある．

# クラスとインスタンスの関係

- クラスは仕様書
  - 仕様書の時点で，どのような部品を使うか(データ属性)，どのように動作するか(メソッド)が，細部まで決定されている．
- インスタンスは実際に作られたもの




クラス


インスタンス ×5

これまで: 既存のデータ型のインスタンスを生成して使用

今回やること: 新しいデータ型を設計し，インスタンスを生成して使用

# 新しいデータ型を作る

- データ型の設計図(クラス)を作る命令: class

クラスの定義は
classで開始

class データ型名:
　ブロック

- 動くコードの例

```
import random
class Dice:
    face_num = 6
    def shoot(self):
        r = random.randint(1, self.face_num)
        return r


dice = Dice()
print(dice.shoot())
```

クラス定義のブロックに現れている変数と関数は，どこでも使えるわけではない

クラス定義の開始: データ型の名前は Dice
データ属性 face_num の定義
メソッド shoot() の定義
1からface_numまでのランダムな整数を戻り値とする

Dice型のインスタンスの生成
インスタンスdiceのメソッドshoot()を実行
戻り値を表示する

# クラスの定義に現れる self

- 変数 self は，インスタンス自身
- メソッドの第1引数として必ずインスタンス自身が渡される規則にになっている．
  - 一般的に self という名前の仮引数で受け取る．
  - 呼び出す際には必要ない．自動的に付与される．
- 変数 self を使うことで，あるデータ型のインスタンスが，自分自身の持つデータ属性やメソッドにアクセスできる．

メソッドを定義する際の第1引数は必ず self
メソッド内でデータ属性やメソッドを使う場合は self. を前に付ける

# selfの使われ方を見てみる

```
import random
class Dice:
    face_num = 6
    def shoot(self):
        r = random.randint(1, self.face_num)
        return r

dice = Dice()
print(dice.shoot())
```

メソッドを定義する際の第1引数
呼び出し側には引数が無いことに注目

メソッドの第1引数のselfが無い場合のエラー

```
    print(dice.shoot())
TypeError: shoot() takes 0 positional
arguments but 1 was given
```

データ属性 face_num の値を得るために
使われている.
「自分自身の持つ」データ属性であることを
示すために必要.

データ属性の参照の際にselfが無い場合のエラー

```
        r = random.randint(1, face_num)
NameError: name 'face_num' is not defined
```

# インスタンス生成時の初期化

- 既存の型を使う場合を確認: datetime.date型のインスタンスを生成する際のコード

```
>>> import datetime
>>> kg = datetime.date(1889,9,28)
>>> kg.year
1889
>>> kg.month
9
>>> kg.day
28
>>> kg.weekday()
5
>>> kg.isoformat()
'1889-09-28'
```

生成時に引数を与えて
生成されるインスタンスのデータ属性に
値を設定している.

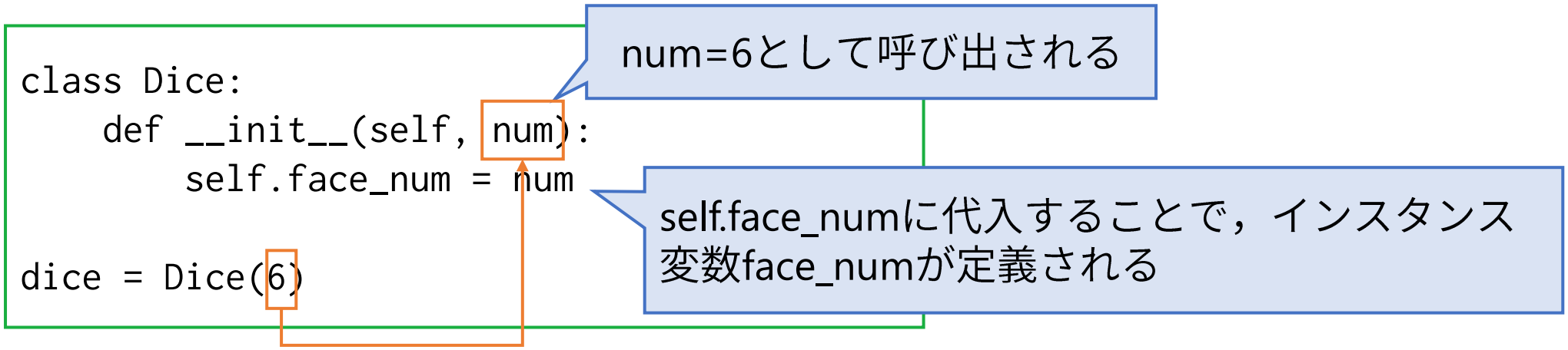
自作のデータ型でもこの機能を使いたい

# 初期化メソッド

- 初期化の際に実行される特別な名前のメソッド

  `__init__()`

- 初期化メソッドを定義することで，インスタンスの初期状態を設定できる．



# データ属性(変数)には2種類ある

- クラス変数(属性)
  - データ型のインスタンスで共有される変数
- インスタンス変数(属性)
  - 各インスタンスに固有の変数

```
class Dice:
    def __init__(self, num):
        self.face_num = num
    def shoot(self):
        r = random.randint(1, self.face_num)
        return r
```

```
import random
class Dice:
    face_num = 6
    def shoot(self):
        r = random.randint(1, self.face_num)
        return r
```

これは実はクラス変数

基本的にデータ属性はインスタンス変数として使用する
クラス変数は定数としての利用に留める

# インスタンス変数はいつ定義されるの?

- メソッドは定義が必要
- クラスの定義内で新しい変数に代入するとクラス変数になる
- では，インスタンス変数はどうやって定義する?

- 答: メソッド内で self.XXXX に代入すると，その時点で(存在していなければ)自動的に生成される．
  - 初期化メソッド中で適切な初期値を設定してしまうのがよい．

補足: どのようなメソッドがあるかはクラスの定義を見れば一目瞭然だが，インスタンス変数はわかりにくく，またタイプミスをするとエラーにならず別のインスタンス変数が生成されてしまう．
→ インスタンス変数の追加を制限する機能(スロット)が用意されている(説明は割愛)．

# 初期化メソッドの引数

- 引数にはデフォルト値を与えることができる．
  - デフォルト値が与えられた引数は省略可能である．

```
class Dice:
    def __init__(self, num):
        self.face_num = num

dice = Dice()
```

引数の省略はできない

```
class Dice:
    def __init__(self, num=6):
        self.face_num = num

dice = Dice()
```

- 仮引数をインスタンス変数と同じ名前にしてもよい．
  - 仮引数とインスタンス変数はself.の有無で区別できるため．
  - 仮引数名を考える手間が省けるので個人的にはおすすめ．

```
class Dice:
    def __init__(self, face_num=6):
        self.face_num = face_num
```