メソッド(method)

関数の一種ですが，「メソッド」と呼ぶ特殊関数があります．これは，**型毎に定義されている関数** で，「変数.関数名(引数 1，引数 2，...)」という呼び出し方をします．

たとえば，掛け算*は実は特別なメソッド__mul__で定義されていますが，

```
>> x = 10
>> x.__mul__(3) # `x * 3` と同じ
30
```

あれ？　このドットを使った関数呼び出し，モジュール内の関数の呼び出しと似てますね．でも明らかに違うことがあります．ドットの前に前置するものが違います．

- メソッドの場合は，変数.関数名（引数）
- モジュール（pyファイル）内の関数の場合は，モジュール.関数名（引数）

また，モジュール関数の場合は，「import モジュール」が必要でした．メソッド関数の場合はモジュールではないので当然importは要りません．

1. メソッドの意味

なぜメソッドなんてものがあるのか？なぜなら，同じ名前の関数でありながら，中の処理は変数毎に変えれると便利だからです．

例えば，さきほどの__mul__すなわち*は，変数が数値の場合と，文字列の場合とで全く異なる処理になっていることがわかります．

```
>> x = 10
>> x.__mul__(3) # `x * 3` と同じ．いわゆる普通の乗算
30

>> y = "a"
>> y.__mul__(3) # `y * 3`と同じ．文字列の場合は回数繰り返し
"aaa"
```

「同じ名前の関数でありながら，中の処理は変数毎に変えれると便利」というのは，関数を自作するときに実感できます．関数を自作するということは，その関数のために「何らかのわかりやすい名前をつける」ということにもなりますが，この名付け・命名で最も気にしないといけないのは「わかりやすい名前」でありながら「予約されていない（使用が禁じられていない）名前」であるということです．

メソッドは型毎に独立しているので，他とかぶる心配がありません．

2. メソッドの使われ方

ところで，メソッドの呼び出し方「変数.関数名(引数 1，引数 2，...)」において，カッコ内の引数も，ドット前置の変数も，どちらも変数ですね．

すなわち，

- モジュール関数だったら，file.関数(x, y)
- メソッド関数だったら，x.関数(y)

みたいなことになっているということです．そして，関数の意味を考えると，

- モジュール関数の場合は，xとyを用いて新しい何かを作る
- メソッド関数の場合は，xのある属性をyを用いて作る，xにyを作用させる

と捉えることができます．

また，関数というのは戻り値が無い場合もあるのでした．

しかしモジュール関数の場合，まったく新しい計算結果を作るので，何かに保存したくなります．すなわち戻り値は原則的にあることが多いです．

メソッド関数の場合は，変数の属性を作ります．属性というのはxに付いているものなので，戻り値をわざわざ返さなくてもいいかもしれません．

- オブジェクトへの関与：「引数は参照されるだけ」
  - メソッド関数：　object1　が書き換えられる（ミュータブル）ときもある
  - モジュール関数：　object1　が書き換えられない．
- 戻り値
  - メソッド関数：戻り値が無い場合もある．object1 自体が答えになる場合がある．
  - モジュール関数：原則的に戻り値がある．

3. メソッド関数の作り方，クラス

普通のモジュール関数の作り方は，

```
# myfunc.py

def func1(x,y):
    z = 4*x + 3*y
    return z
```

のように拡張子pyのファイルの中にdef文を書くのでした．

ではメソッド関数を自作するには？？？

メソッド関数は「型毎に定義されている」というわけですが， すでに概論のかなり前の回に　int, float, str, 最近だとコンテナ類の list, tuple, set, dict　などの既存の型を紹介していました．既存の型は既にメソッド関数がいくつか定義されています．変数にどんなメソッド関数が定義されているのかを調べるには，`dir(変数)`という関数を用います．ただし，`dir`関数は，メソッド関数だけでなく全ての「属性」を返します．モジュールに関数と同様に定数を定義できたように，型には関数（メソッド）と定数が定義されています．

```
>> x = 10
>> dir(x)
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__',
'__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__',
'__floordiv__',
'__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__',
'__hash__',
'__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__',
'__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__',
'__or__',
'__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__',
'__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__',
'__rmul__',
'__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__',
'__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'as_integer_ratio',
'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator',
'real',
'to_bytes']
```

属性がメソッドなのか定数なのかを調べるのは`callable(変数.属性)`で判定できます．ですが，自作していない型の変数ならば，`help(変数)`のほうが情報が多いでしょう．

```
>> x = 10
>> callable(x.real) # Falseなら定数
False
>> x.real
10
>> callable(x.to_bytes) # True ならメソッド
True
>> # x.to_bytesはメソッドらしいが引数は？？
>> help(x)
Help on int object:

class int(object)
 |  int([x]) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is a number, return x.__int__().  For floating point
 |  numbers, this truncates towards zero.
 |
 |  If x is not a number or if base is given, then x must be a string,
 |  bytes, or bytearray instance representing an integer literal in the
 |  given base.  The literal can be preceded by '+' or '-' and be surrounded
 |  by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
 |  Base 0 means to interpret the base from the string as an integer literal.
 |  >>> int('0b100', base=0)
--- More ---
# SPACEキーでページ送り
```

```
  |  4
  |
  |  Built-in subclasses:
  |      bool
  |
  |  Methods defined here:
  |
  |  __abs__(self, /)
  |      abs(self)
  |
--- More ---
  |  to_bytes(self, /, length, byteorder, *, signed=False)
  |      Return an array of bytes representing an integer.
  |
  |      length
  |        Length of bytes object to use.  An OverflowError is raised if the
  |        integer is not representable with the given number of bytes.
  |      byteorder
  |        The byte order used to represent the integer.  If byteorder is 'big',
  |        the most significant byte is at the beginning of the byte array.  If
  |        byteorder is 'little', the most significant byte is at the end of the
  |        byte array.  To request the native byte order of the host system, use
  |        `sys.byteorder' as the byte order value.
  |      signed
  |        Determines whether two's complement is used to represent the integer.
  |        If signed is False and a negative integer is given, an OverflowError
  |        is raised.
--- More ---
# 'q'で中止
```

> メソッドto_bytesは integer（整数）のバイト表現を返す関数で，第1引数がバイト長lengthで，第2引数がバイト順byteorder（値は'big'か'little'），signed=Trueという引数を入れると2の補数表現となる

とのことです．

3.1. メソッドを作る（クラスを作る）

メソッドを自作するということは，型を自分で作るということになります． 全く新しい型を作ることもできますが，ほとんどの場合は既に定義されている型を原型にして作ります．原型の型の属性をすべて受け継いだ型となります．

```
class Pet(object):
    def __init__(self, name, species):
        self.name = name
        self.species = species

    def getName(self):
        return self.name

    def getSpecies(self):
```

```
        return self.species

    def __str__(self):
        return "%s is a %s" % (self.name, self.species)
```

`class Pet(object)`で，object型を原型にして，Petという型を作っています． object型とは云わば最低限の型であり，すべての型の始祖の型です．

型定義の中のdef文は関数の定義ですが．selfという引数が第1引数にある関数がメソッド関数です．selfの次からが実際に呼び出すときの引数になります．

変数selfには，属性の元である変数が入ります．

__ダブル・アンダーバーで囲まれたものは特別なオブジェクトで，

- メソッド`__init__`は，`x = Pet("ポチ","犬")`などと型名がメソッド名となるメソッド
- メソッド`__str__`は，関数`print`のことです．

ということで，上記の型（「クラス」とpythonでは言っている）では，

```
>> x = Pet("ポチ","犬")
>> print(x.getName())
ポチ
>> print(x.getSpecies())
犬
>> print(x)
ポチ is a 犬
```

という感じです．

クラスについては，また別の機会に勉強します．