

# Chapter 8: Classes part 2 - special methods

Joakim Sundnes<sup>1,2</sup>

<sup>1</sup>Simula Research Laboratory

<sup>2</sup>University of Oslo, Dept. of Informatics

Oct 15, 2023

## 0.1 Special methods

- The class constructor has a special name: `__init__`
- The name is recognized by Python, and ensures this method is called when a new class instance is created; e.g. `y = MyClass(4)`
- The constructor is an example of a *special method*
- Special methods have names with leading and trailing double underscores
- Special methods are recognized by Python, and automatically called when we perform various operations on the class instances

## 0.2 The call special method; motivation

Recall the class for representing a function:

```
class Y:
    def __init__(self, v0):
        self.v0 = v0
        self.g = 9.81

    def value(self, t):
        return self.v0*t - 0.5*self.g*t**2
```

```
y = Y(3)
v = y.value(0.1)
```

But it would be more natural to use the class like this:

```
y = Y(3)
v = y(0.1)
```

### 0.3 The call special method; implementation

Simply replace the `value` method by a *call* special method:

```
class Y:
    def __init__(self, v0):
        self.v0 = v0
        self.g = 9.81

    def __call__(self, t):
        return self.v0*t - 0.5*self.g*t**2
```

Now we can write

```
y = Y(3)
v = y(0.1) # same as v = y.__call__(0.1) or Y.__call__(y, 0.1)
```

Note:

- The instance `y` now behaves and looks as a function!
- The `value(t)` method does the same, but `__call__` allows nicer syntax for computing function values

### 0.4 Special method for printing

- In Python, we can usually print an object `a` by `print(a)`, works for built-in types (strings, lists, floats, ...)
- Python does not know how to print objects of a user-defined class, but if the class defines a method `__str__`, Python will use this method to convert an object to a string

Example:

```
class Y:
    ...
    def __call__(self, t):
        return self.v0*t - 0.5*self.g*t**2

    def __str__(self):
        return f'v0*t - 0.5*g*t**2; v0={self.v0}'
```

Demo:

```
>>> y = Y(1.5)
>>> y(0.2)
0.1038
>>> print(y)
v0*t - 0.5*g*t**2; v0=1.5
```

## 0.5 Class Y revisited with print method

```
class Y:
    """Class for function  $y(t; v_0, g) = v_0*t - 0.5*g*t**2$ ."""
    def __init__(self, v0):
        """Store parameters."""
        self.v0 = v0
        self.g = 9.81
    def __call__(self, t):
        """Evaluate function."""
        return self.v0*t - 0.5*self.g*t**2
    def __str__(self):
        """Pretty print."""
        return f'v0*t - 0.5*g*t**2; v0={self.v0}'
```

## 0.6 Special methods for arithmetic operations

```
c = a + b    # c = a.__add__(b)
c = a - b    # c = a.__sub__(b)
c = a*b     # c = a.__mul__(b)
c = a/b     # c = a.__div__(b)
c = a**e    # c = a.__pow__(e)
```

## 0.7 Special methods for comparisons

```
a == b      # a.__eq__(b)
a != b      # a.__ne__(b)
a < b       # a.__lt__(b)
a <= b      # a.__le__(b)
a > b       # a.__gt__(b)
a >= b      # a.__ge__(b)
```

## 0.8 The programmer is in charge of defining special methods!

How should, for instance, `__add__(self, other)` and `__mul__(self, other)` be defined?

This is completely up to the programmer, depending on what are meaningful results of `object1 + object2` and `object1 * object2`.

## 0.9 Class for vectors in the plane

Mathematical operations for vectors in the plane:

$$(a, b) + (c, d) = (a + c, b + d)$$

$$(a, b) - (c, d) = (a - c, b - d)$$

$$(a, b) \cdot (c, d) = ac + bd$$

$$(a, b) = (c, d) \text{ if } a = c \text{ and } b = d$$

Desired application code:

```
>>> u = Vec2D(0,1)
>>> v = Vec2D(1,0)
>>> print(u + v)
(1, 1)
>>> a = u + v
>>> w = Vec2D(1,1)
>>> a == w
True
>>> print(u - v)
(-1, 1)
>>> print(u*v)
0
```

## 0.10 Class for vectors; implementation

```
class Vec2D:
    def __init__(self, x, y):
        self.x = x; self.y = y

    def __add__(self, other):
        return Vec2D(self.x+other.x, self.y+other.y)

    def __sub__(self, other):
        return Vec2D(self.x-other.x, self.y-other.y)

    def __mul__(self, other):
        return self.x*other.x + self.y*other.y

    def __abs__(self):
        return math.sqrt(self.x**2 + self.y**2)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __str__(self):
        return f'({self.x}, {self.y})'
```

## 0.11 Class for polynomials; functionality

A polynomial can be specified by a list of its coefficients. For example,  $1 - x^2 + 2x^3$  is

$$1 + 0 \cdot x - 1 \cdot x^2 + 2 \cdot x^3$$

and the coefficients can be stored as `[1, 0, -1, 2]`

Desired application code:

```
>>> p1 = Polynomial([1, -1])
>>> print(p1)
1 - x
>>> print(p1(x=0.5))
0.5
>>> p2 = Polynomial([0, 1, 0, 0, -6, -1])
>>> p3 = (p1 + p2)
>>> print(p3.coeff)
[1, 0, 0, 0, -6, -1]
>>> print(p3)
1 - 6*x^4 - x^5
>>> p2.differentiate()
>>> print(p2)
1 - 24*x^3 - 5*x^4
```

How can we make class Polynomial?

## 0.12 Class Polynomial; basic code

```
class Polynomial:
    def __init__(self, coefficients):
        self.coeff = coefficients

    def __call__(self, x):
        s = 0
        for i in range(len(self.coeff)):
            s += self.coeff[i]*x**i
        return s
```

## 0.13 Class Polynomial; addition

```
class Polynomial:
    ...

    def __add__(self, other):
        # return self + other

        # start with the longest list and add in the other:
        if len(self.coeff) > len(other.coeff):
            coeffsum = self.coeff[:] # copy!
            for i in range(len(other.coeff)):
                coeffsum[i] += other.coeff[i]
        else:
            coeffsum = other.coeff[:] # copy!
            for i in range(len(self.coeff)):
                coeffsum[i] += self.coeff[i]
        return Polynomial(coeffsum)
```

## 0.14 Class Polynomial; multiplication

**Mathematics:** Multiplication of two general polynomials:

$$\left(\sum_{i=0}^M c_i x^i\right) \left(\sum_{j=0}^N d_j x^j\right) = \sum_{i=0}^M \sum_{j=0}^N c_i d_j x^{i+j}$$

The coeff. corresponding to power  $i + j$  is  $c_i \cdot d_j$ . The list  $r$  of coefficients of the result:  $r[i+j] = c[i]*d[j]$  ( $i$  and  $j$  running from 0 to  $M$  and  $N$ , resp.)

**Implementation:**

```
class Polynomial:
    ...
    def __mul__(self, other):
        M = len(self.coeff) - 1
        N = len(other.coeff) - 1
        coeff = [0]*(M+N+1) # or zeros(M+N+1)
        for i in range(0, M+1):
            for j in range(0, N+1):
                coeff[i+j] += self.coeff[i]*other.coeff[j]
        return Polynomial(coeff)
```

## 0.15 Class Polynomial; differentiation

**Mathematics:** Rule for differentiating a general polynomial:

$$\frac{d}{dx} \sum_{i=0}^n c_i x^i = \sum_{i=1}^n i c_i x^{i-1}$$

If  $c$  is the list of coefficients, the derivative has a list of coefficients,  $dc$ , where  $dc[i-1] = i*c[i]$  for  $i$  running from 1 to the largest index in  $c$ . Note that  $dc$  has one element less than  $c$ .

**Implementation:**

```
class Polynomial:
    ...
    def differentiate(self): # change self
        for i in range(1, len(self.coeff)):
            self.coeff[i-1] = i*self.coeff[i]
        del self.coeff[-1]

    def derivative(self): # return new polynomial
        dpdx = Polynomial(self.coeff[:]) # copy
        dpdx.differentiate()
        return dpdx
```

## 0.16 Class Polynomial; pretty print

```
class Polynomial:
    ...
    def __str__(self):
        s = ''
        for i in range(0, len(self.coeff)):
            if self.coeff[i] != 0:
                s += f' + {self.coeff[i]}*x^{i}'
```

```

# fix layout (lots of special cases):
s = s.replace('+ -', '- ')
s = s.replace(' 1*', ' ')
s = s.replace('x^0', '1')
s = s.replace('x^1 ', 'x ')
s = s.replace('x^1', 'x')
if s[0:3] == '+ ': # remove initial +
    s = s[3:]
if s[0:3] == '- ': # fix spaces for initial -
    s = '- ' + s[3:]
return s

```

## 0.17 Class for polynomials; usage

Consider

$$p_1(x) = 1 - x, \quad p_2(x) = x - 6x^4 - x^5$$

and their sum

$$p_3(x) = p_1(x) + p_2(x) = 1 - 6x^4 - x^5$$

```

>>> p1 = Polynomial([1, -1])
>>> print(p1)
1 - x
>>> p2 = Polynomial([0, 1, 0, 0, -6, -1])
>>> p3 = (p1 + p2)
>>> print p3.coeff
[1, 0, 0, 0, -6, -1]
>>> p2.differentiate()
>>> print(p2)
1 - 24*x^3 - 5*x^4

```

## 0.18 Example; "automatic" differentiation

Given some mathematical function in Python, say

```

def f(x):
    return x**3

```

can we make a class `Derivative` and write

```

dfdxd = Derivative(f)

```

so that `dfdxd` behaves as a function that computes the derivative of `f(x)`?

```

print(dfdxd(2)) # computes 3*x**2 for x=2

```

## 0.19 Automagic differentiation; solution

**Method.** We use numerical differentiation “behind the curtain”:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

for a small (yet moderate)  $h$ , say  $h = 10^{-5}$

**Implementation.**

```
class Derivative:
    def __init__(self, f, h=1E-5):
        self.f = f
        self.h = float(h)

    def __call__(self, x):
        f, h = self.f, self.h      # make short forms
        return (f(x+h) - f(x))/h
```

## 0.20 Automagic differentiation; demo

```
>>> from math import *
>>> df = Derivative(sin)
>>> x = pi
>>> df(x)
-1.000000082740371
>>> cos(x) # exact
-1.0
>>> def g(t):
...     return t**3
...
>>> dg = Derivative(g)
>>> t = 1
>>> dg(t) # compare with 3 (exact)
3.000000248221113
```

## 0.21 Automagic differentiation; useful in Newton’s method

Newton’s method solves nonlinear equations  $f(x) = 0$ , but the method requires  $f'(x)$

```
def Newton(f, xstart, dfdx, epsilon=1E-6):
    ...
    return x, no_of_iterations, f(x)
```

Suppose  $f'(x)$  requires boring/lengthy derivation, then class `Derivative` is handy:

```
>>> def f(x):
...     return 100000*(x - 0.9)**2 * (x - 1.1)**3
...
>>> df = Derivative(f)
>>> xstart = 1.01
>>> Newton(f, xstart, df, epsilon=1E-5)
(1.0987610068093443, 8, -7.5139644257961411e-06)
```



## 0.22 Class introduction - summary

- Classes pack together data and functions that naturally belong together
- We define a class, and then create *instances* (or objects) of that class
  - Different instances will have different data, but they all have the same functions operating on that data
- In IN1900 codes, classes are never really necessary, but sometimes convenient
- In "real-world" programs, with tens of 1000s of lines, the extra organization offered by classes may be the difference between a code that works and one that doesn't

## 0.23 Summary of special methods

- $c = a + b$  implies  $c = a.\_add\_(b)$
- There are special methods for  $a+b$ ,  $a-b$ ,  $a*b$ ,  $a/b$ ,  $a**b$ ,  $-a$ ,  $if\ a:$ ,  $len(a)$ ,  $str(a)$  (pretty print), etc.
- With special methods we can create new mathematical objects like vectors, polynomials and complex numbers and write "mathematical code" (arithmetics)
- The call special method is particularly handy:  $v = c(5)$  means  $v = c.\_call\_(5)$
- Functions with parameters should be represented by a class with the parameters as attributes and with a call special method for evaluating the function