Ch.7: Introduction to classes

Hans Petter Langtangen[1,2] Joakim Sundnes[1,2]

Simula Research Laboratory[1]

University of Oslo, Dept. of Informatics[2]

Oct 27, 2017

Plan for Oct 25

- Exercises 6.10, 6.11
- Introduction to classes
- Exercise 7.1

Basics of classes (1)

- Classes are an essential part of object oriented programming
- We have used classes since day 1 in IN1900:

```
>>> S = "This is a string"
>>> type(S)
<class 'str'>
>>> L = S.split()
>>> type(L)
<class 'list'>
```

Basics of classes (2)

- Classes pack data and functions together
- Every time we make a string object in Python, we create an *instance* of the built-in class `str`
- Calls like `S.split()` calls the function `split()` associated with the instance `S`
- We will now learn how to make our own classes

Class = functions + data (variables) in one unit

- A class packs together data (a collection of variables) and functions as *one single unit*
- As a programmer you can create a new class and thereby a new object type (like `float`, `list`, `file`, ...)
- A class is much like a module: a collection of “global” variables and functions that belong together
- There is only one instance of a module while a class can have many instances (copies)
- Modern programming applies classes to a large extent
- It will take some time to master the class concept
- Let’s learn by doing!

Representing a function by a class; background

Consider a function of t with a parameter v_0:

$$y(t; v_0) = v_0 t - \frac{1}{2}gt^2$$

We need both v_0 and t to evaluate y (and $g = 9.81$), but how should we implement this?

Having t and v_0 as arguments:

```
def y(t, v0):
    g = 9.81
    return v0*t - 0.5*g*t**2
```

Having t as argument and v_0 as global variable:

```
def y(t):
    g = 9.81
    return v0*t - 0.5*g*t**2
```

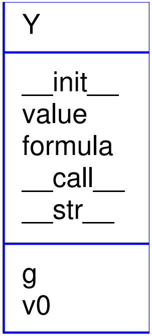
Motivation: $y(t)$ is a function of t only

Representing a function by a class; idea

- With a class, `y(t)` can be a function of `t` only, but still have `v0` and `g` as parameters with given values.
- The class packs together a function `y(t)` and data (`v0`, `g`)

Representing a function by a class; technical overview

- We make a class `Y` for $y(t; v_0)$ with variables `v0` and `g` and a function `value(t)` for computing $y(t; v_0)$
- Any class should also have a function `__init__` for initialization of the variables



Representing a function by a class; the code

```
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
```

Usage:

```
y = Y(v0=3)             # create instance (object)
v = y.value(0.1)        # compute function value
```

Representing a function by a class; the constructor

When we write

```
y = Y(v0=3)
```

we create a new variable (instance) `y` of type `Y`. `Y(3)` is a call to the *constructor*:

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

What is this `self` variable? Stay cool - it will be understood later as you get used to it

- Think of `self` as `y`, i.e., the new variable to be created. `self.v0 = ...` means that we attach a variable `v0` to `self` (`y`).
- `Y(3)` means `Y.__init__(y, 3)`, i.e., set `self=y`, `v0=3`
- Remember: `self` is always first parameter in a function, but never inserted in the call!
- After `y = Y(3)`, `y` has two variables `v0` and `g`

```
print y.v0
print y.g
```

In mathematics you don't understand things. You just get used to them. John von Neumann, mathematician, 1903-1957.

What is this `self` variable? Stay cool - it will be understood later as you get used to it

- Think of `self` as `y`, i.e., the new variable to be created. `self.v0 = ...` means that we attach a variable `v0` to `self` (`y`).
- `Y(3)` means `Y.__init__(y, 3)`, i.e., set `self=y`, `v0=3`
- Remember: `self` is always first parameter in a function, but never inserted in the call!
- After `y = Y(3)`, `y` has two variables `v0` and `g`

```
print y.v0
print y.g
```

In mathematics you don't understand things. You just get used to them. John von Neumann, mathematician, 1903-1957.

What is this `self` variable? Stay cool - it will be understood later as you get used to it

- Think of `self` as `y`, i.e., the new variable to be created. `self.v0 = ...` means that we attach a variable `v0` to `self` (`y`).
- `Y(3)` means `Y.__init__(y, 3)`, i.e., set `self=y`, `v0=3`
- Remember: `self` is always first parameter in a function, but never inserted in the call!
- After `y = Y(3)`, `y` has two variables `v0` and `g`

```
print y.v0
print y.g
```

In mathematics you don't understand things. You just get used to them. John von Neumann, mathematician, 1903-1957.

Representing a function by a class; the value method

- Functions in classes are called *methods*
- Variables in classes are called *attributes*

Here is the `value` method:

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

Example on a call:

```python
v = y.value(t=0.1)
```

`self` is left out in the call, but Python automatically inserts `y` as the `self` argument inside the `value` method. Think of the call as

```python
Y.value(y, t=0.1)
```

Inside `value` things "appear" as

```python
return y.v0*t - 0.5*y.g*t**2
```

`self` gives access to "global variables" in the class object.

Classes introduction - summary

- A class is simply a collection of functions and data that naturally belong together
- Functions in a class are usually called methods, data are called attributes
- We create instances (or objects) of a class, and each instance can have different values for the attributes
- All classes should have a method `__init__`, called a constructur, which is called every time a new instance is created
- The constructur will typically initialize all data in an instance
- All methods in a class should have `self` as first argument in the definition, but not in the call. This may be confusing at first, but one gets used to it.