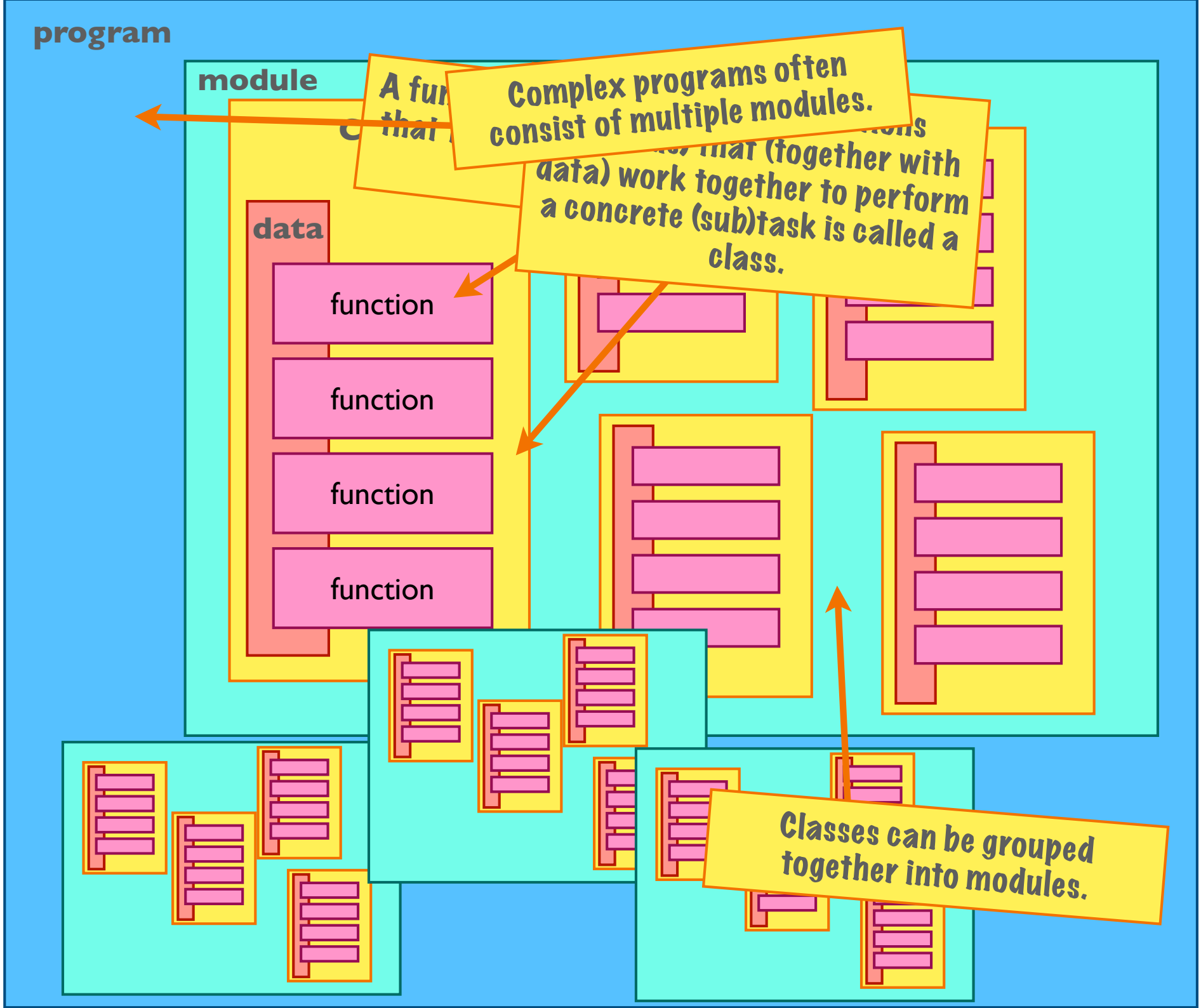
master DSLS

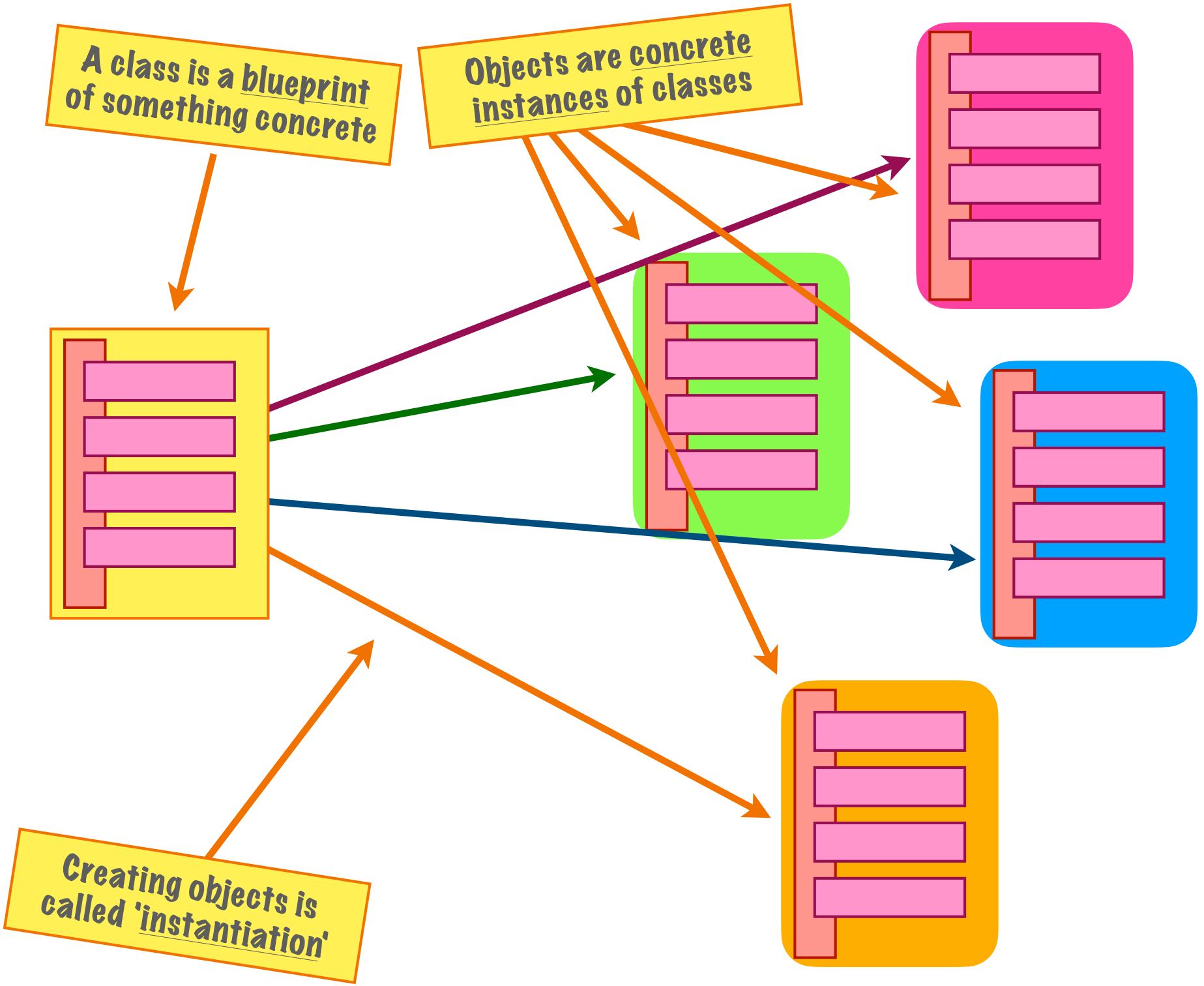
Programming 2

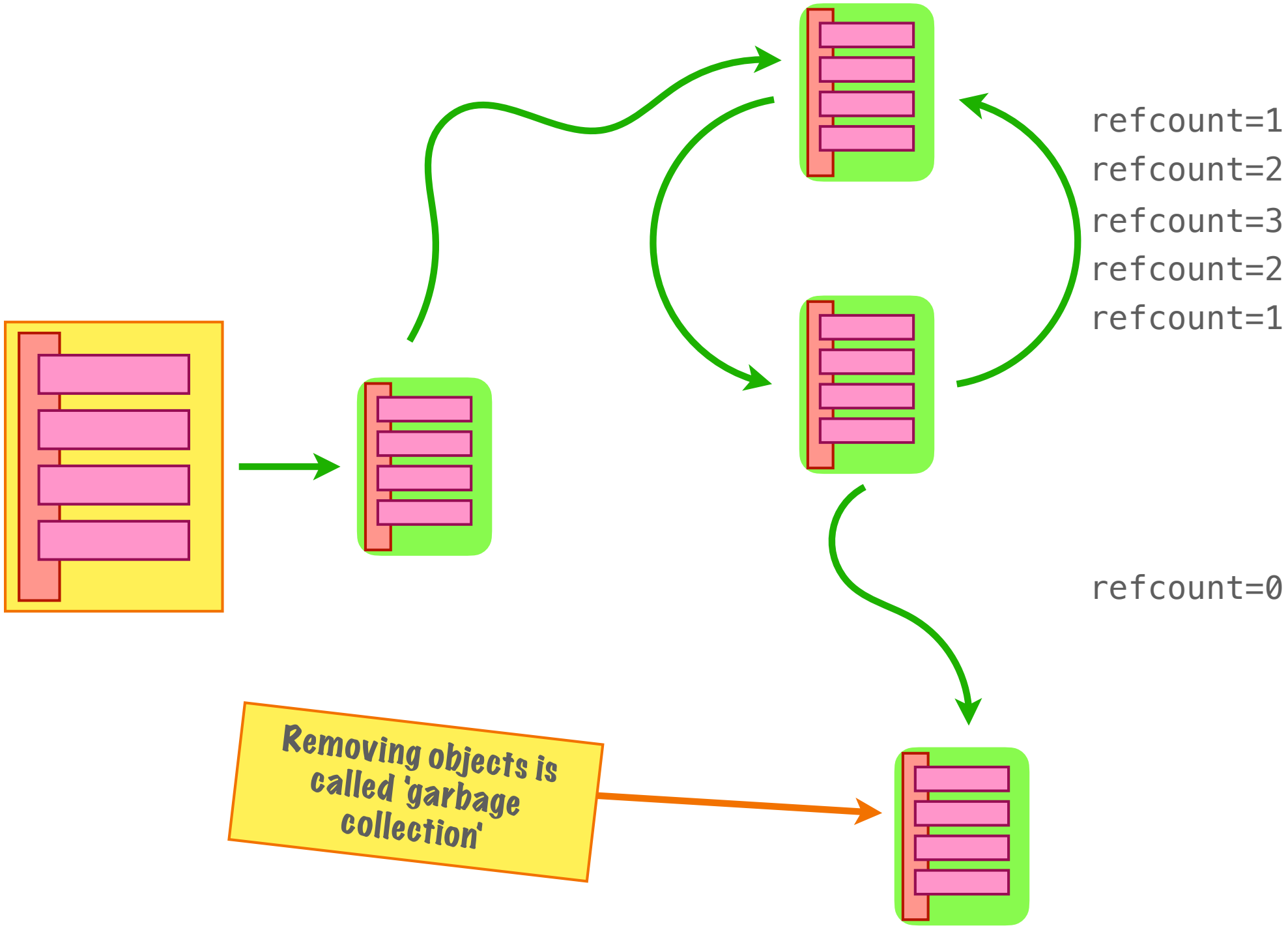
2. Classes and instances, methods

Bart Barnard

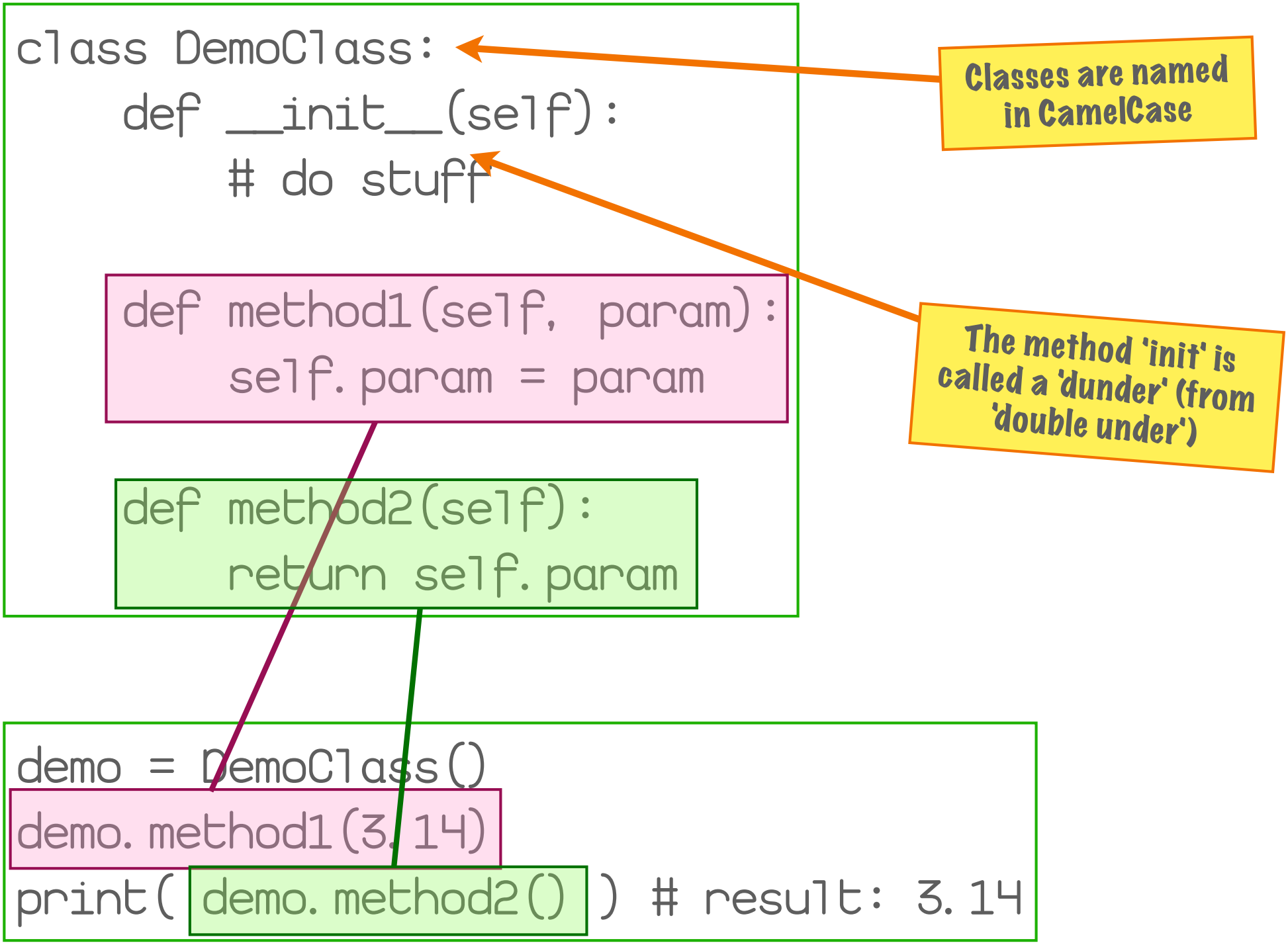
1. classes and instances

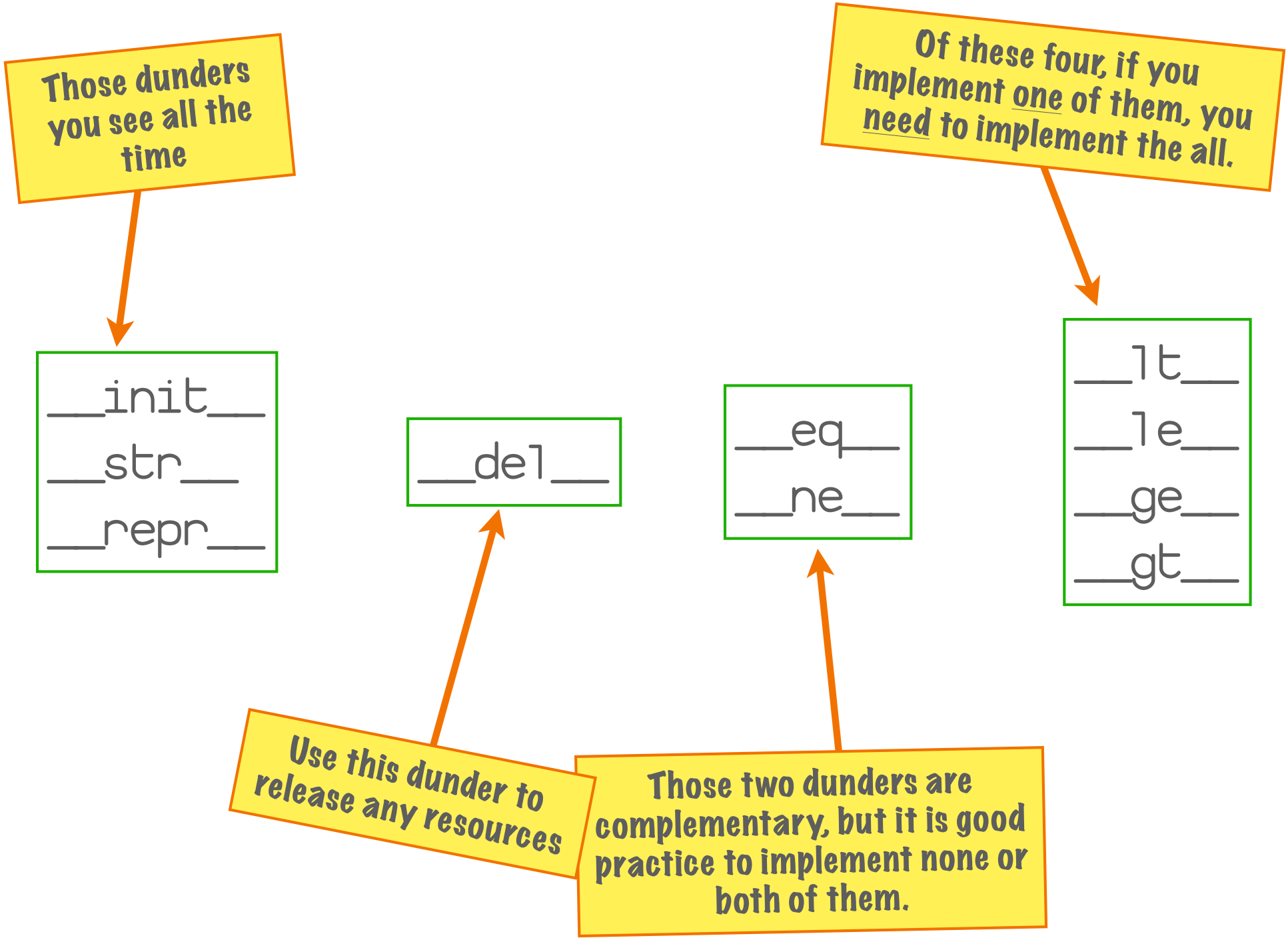

program
module
data
function
function
function
function
Complex programs often consist of multiple modules.
Classes can be grouped together into modules.


A class is a blueprint of something concrete
Objects are concrete instances of classes
Creating objects is called 'instantiation'


refcount=1
refcount=2
refcount=3
refcount=2
refcount=1
refcount=0
Removing objects is called 'garbage collection'

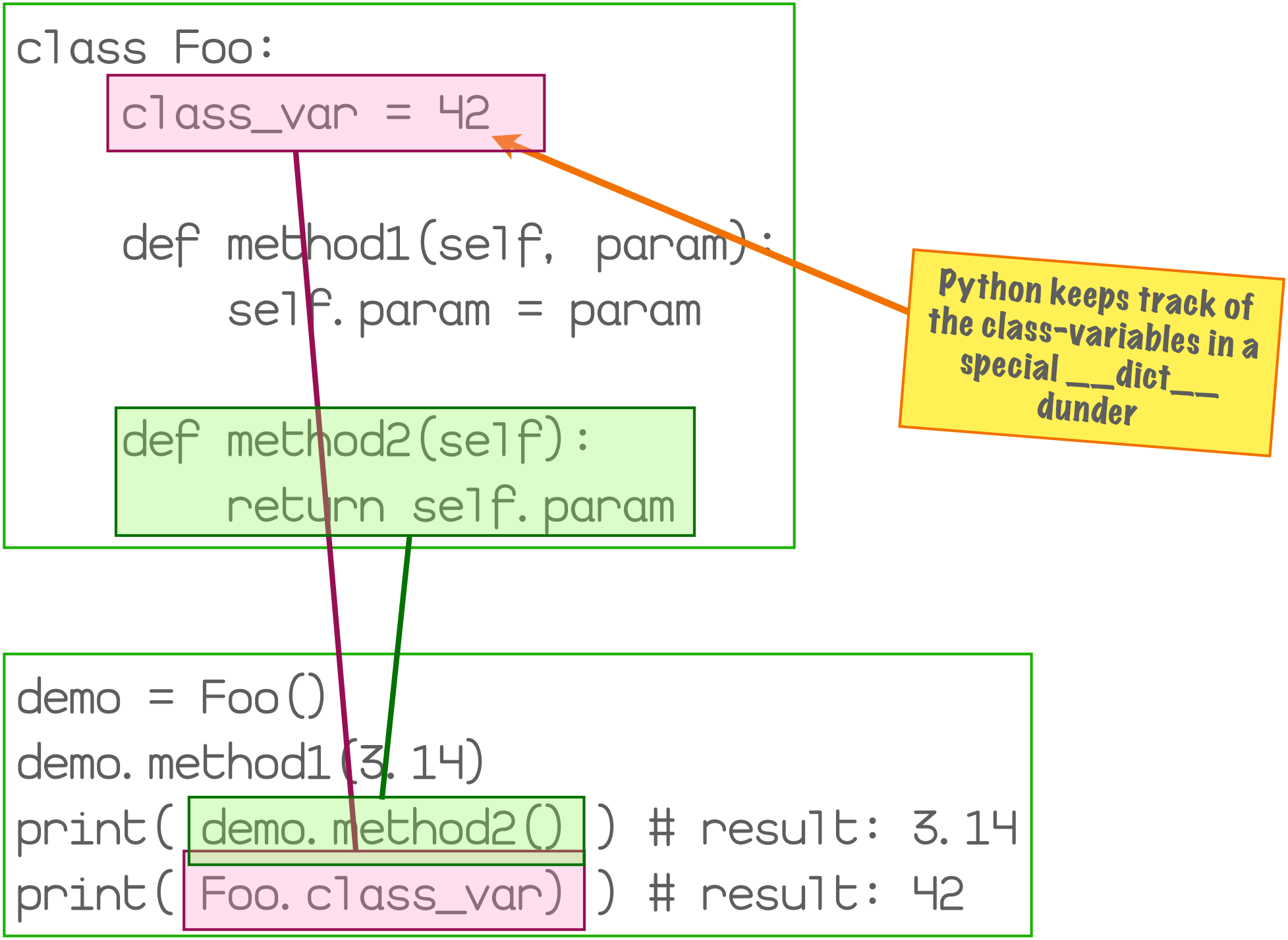
3. code and dunders

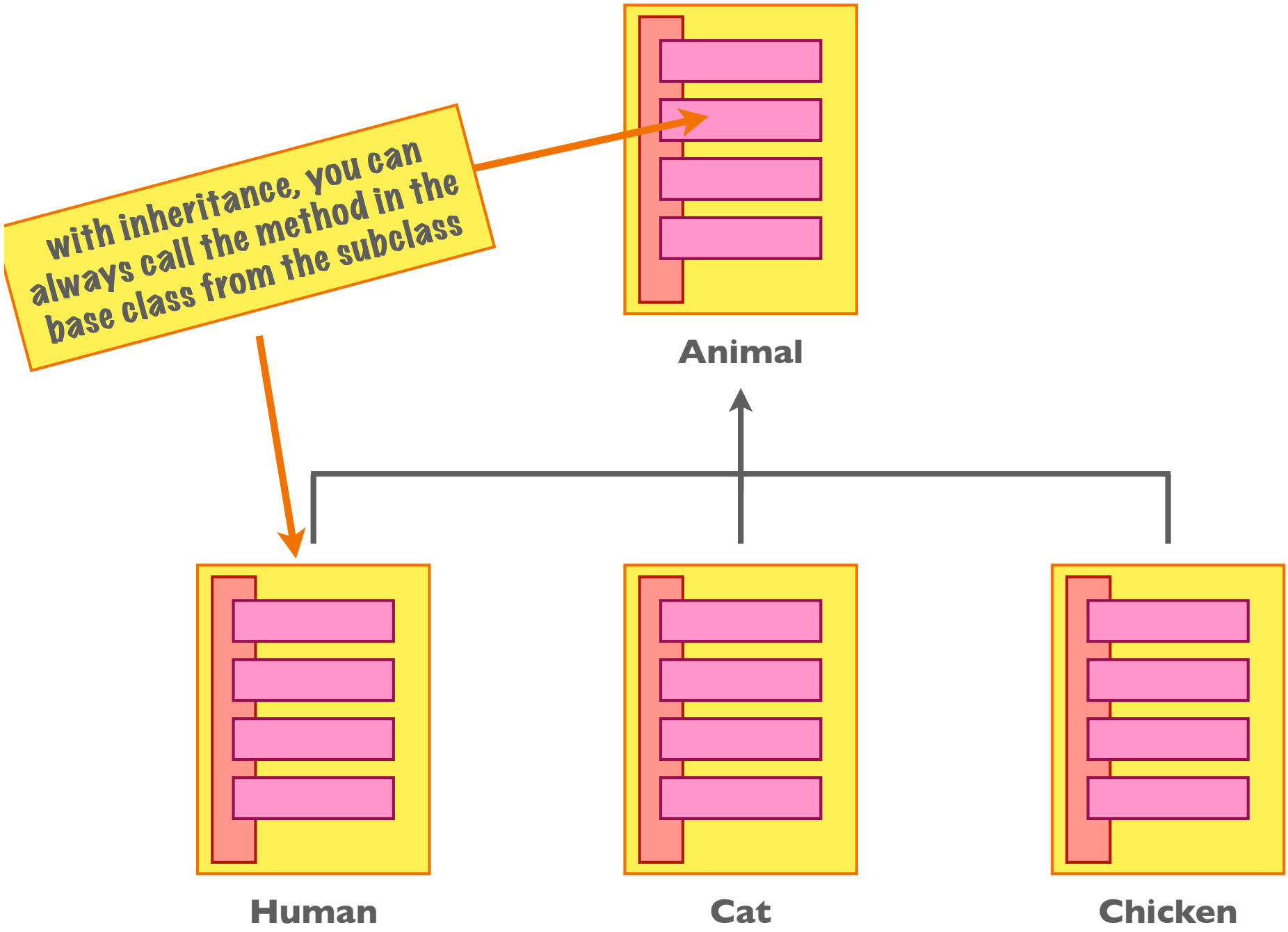

class DemoClass:
def __init__(self):
# do stuff
def method1(self, param):
self.param = param
def method2(self):
return self.param
demo = DemoClass()
demo.method1(3.14)
print(demo.method2()) # result: 3.14
Classes are named in CamelCase
The method 'init' is called a 'dunder' (from 'double under')


Those dunders you see all the time
__init__
__str__
__repr__
__del__
Use this dunder to release any resources
__eq__
__ne__
Those two dunders are complementary, but it is good practice to implement none or both of them.
Of these four, if you implement one of them, you need to implement the all.
__lt__
__le__
__ge__
__gt__

Sign	Dunder methods
+	__add__
-	__sub__
*	__mul__
@	__matmul__
/	__truediv__
//	__floordiv__
%	__mod__
divmod()	__divmod__
pow() / **	__pow__
<<	__lshift__
>>	__rshift__
&	__and__
^	__xor__
|	__or__

4. methods and inheritance

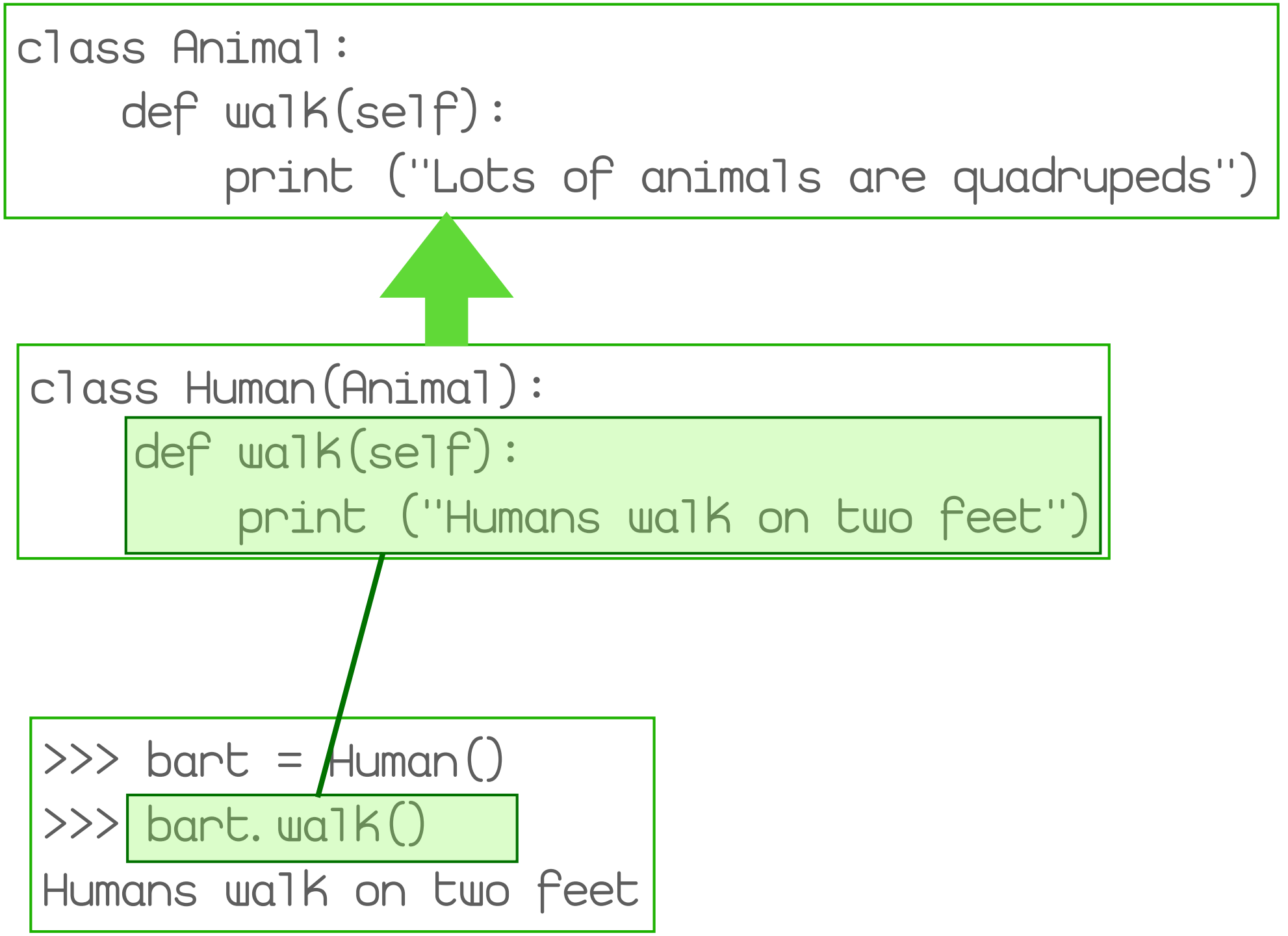

class Foo:
class_var = 42
def method1(self, param):
self.param = param
def method2(self):
return self.param
Python keeps track of the class-variables in a special __dict__ dunder
demo = Foo()
demo.method1(3.14)
print(demo.method2()) # result: 3.14
print(Foo.class_var)) # result: 42

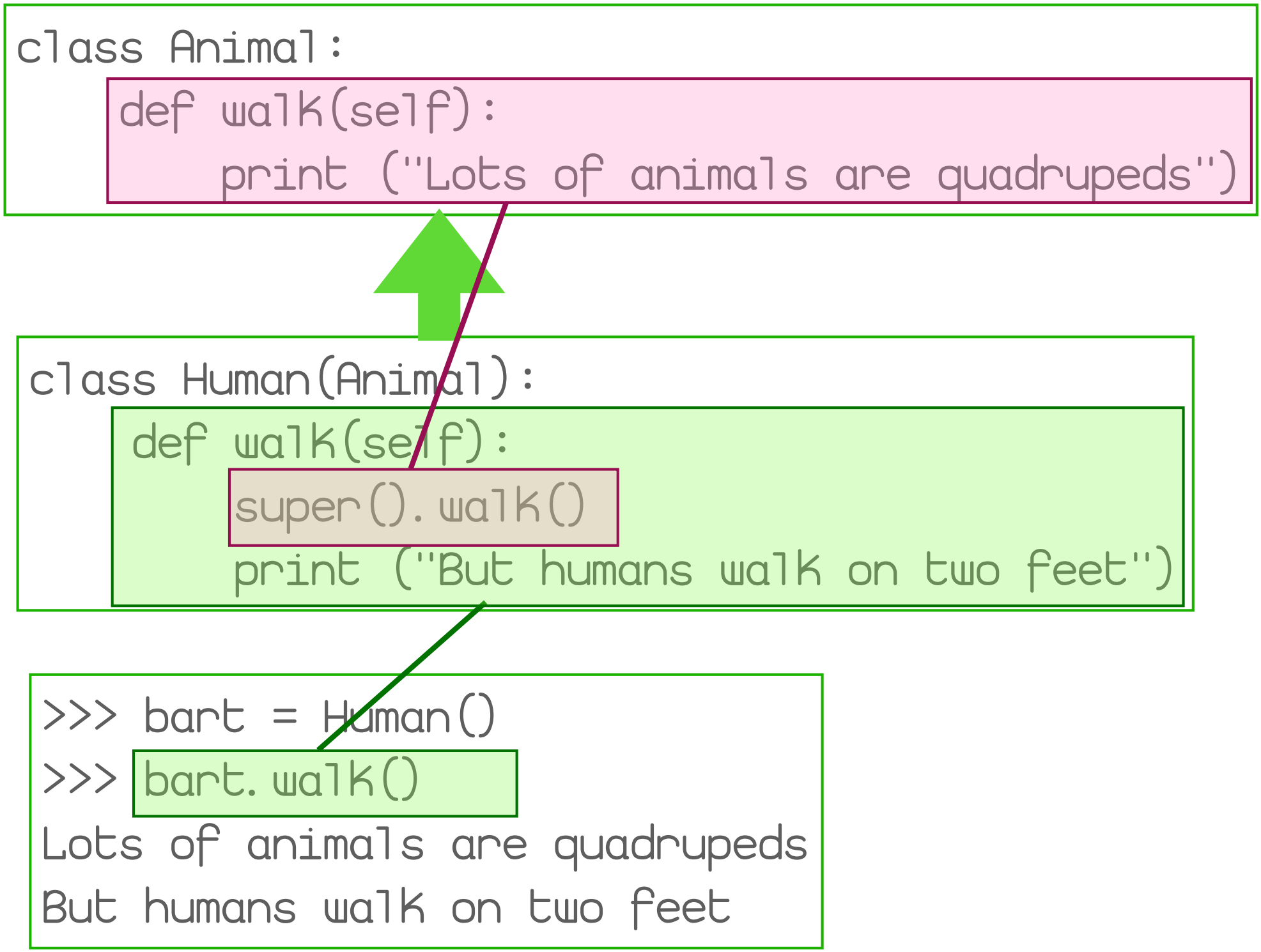

with inheritance, you can always call the method in the base class from the subclass
Animal
Human
Cat
Chicken

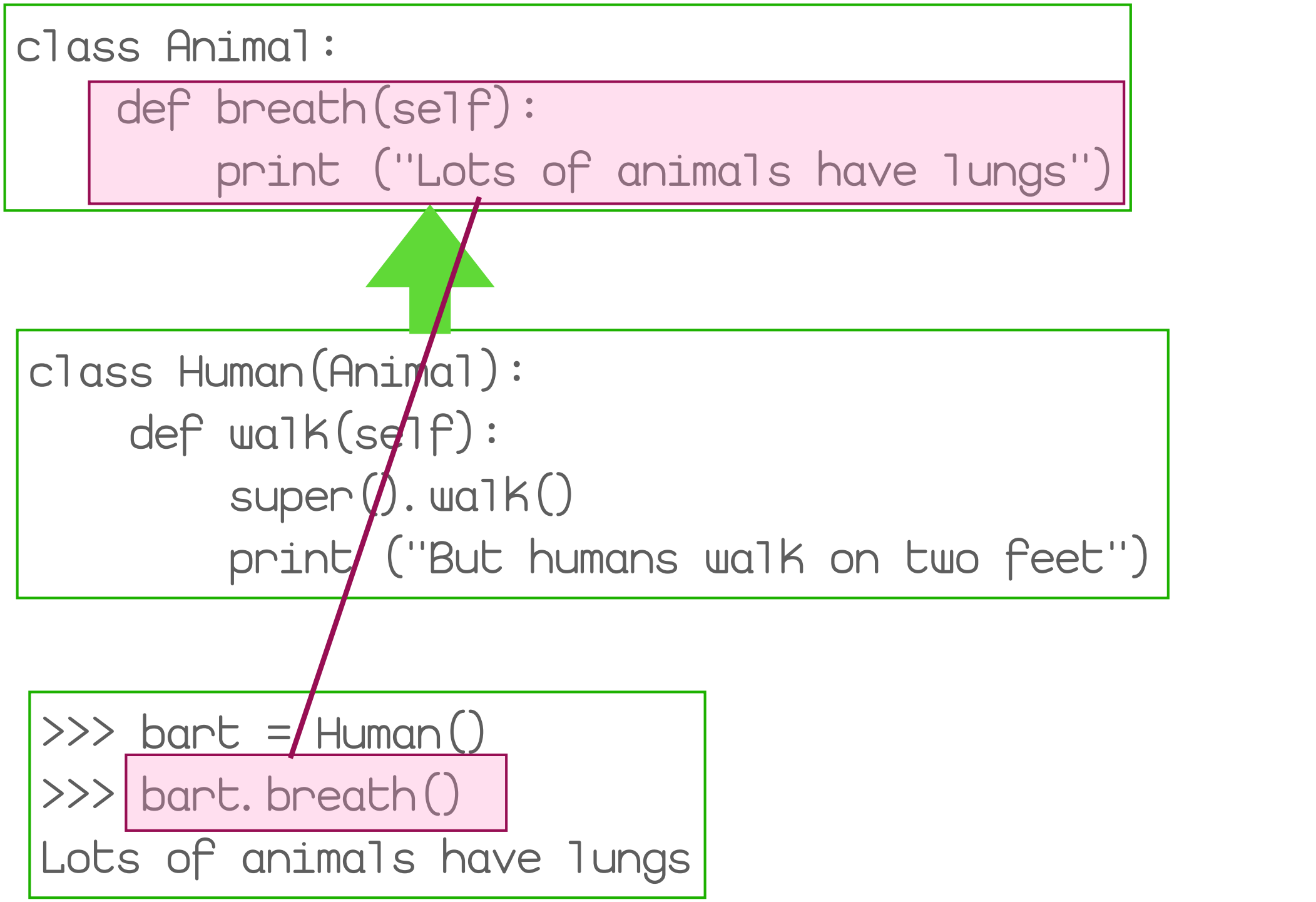
```
class Animal:
    def __init__(self):
        print ("Animal's initialiser")
```

```
class Human(Animal):
    def __init__(self):
        print ("Human's initialiser")
```

```
>>> bart = Human()
Human's initialiser
```


class Animal:
def walk(self):
print ("Lots of animals are quadrupeds")
class Human(Animal):
def walk(self):
print ("Humans walk on two feet")
>>> bart = Human()
>>> bart.walk()
Humans walk on two feet


class Animal:
def walk(self):
print ("Lots of animals are quadrupeds")
class Human(Animal):
def walk(self):
super().walk()
print ("But humans walk on two feet")
>>> bart = Human()
>>> bart.walk()
Lots of animals are quadrupeds
But humans walk on two feet


class Animal:
def breath(self):
print ("Lots of animals have lungs")
class Human(Animal):
def walk(self):
super().walk()
print ("But humans walk on two feet")
>>> bart = Human()
>>> bart.breath()
Lots of animals have lungs


Exercise