

DEEP INSIDE OBJECT-ORIENTED PYTHON

Leonardo Giordani



PyCon Ireland 2014

lgiordani.com



ABOUT ME

I'm a software engineer, interested in operating systems, versioning, Python and software architecture.

Coder since around 1988, Linux user since 1998, Python lover since 1999.

Currently working with Python and C in the field of satellite remote sensing.



The Digital Cat
<http://lgiordani.com>



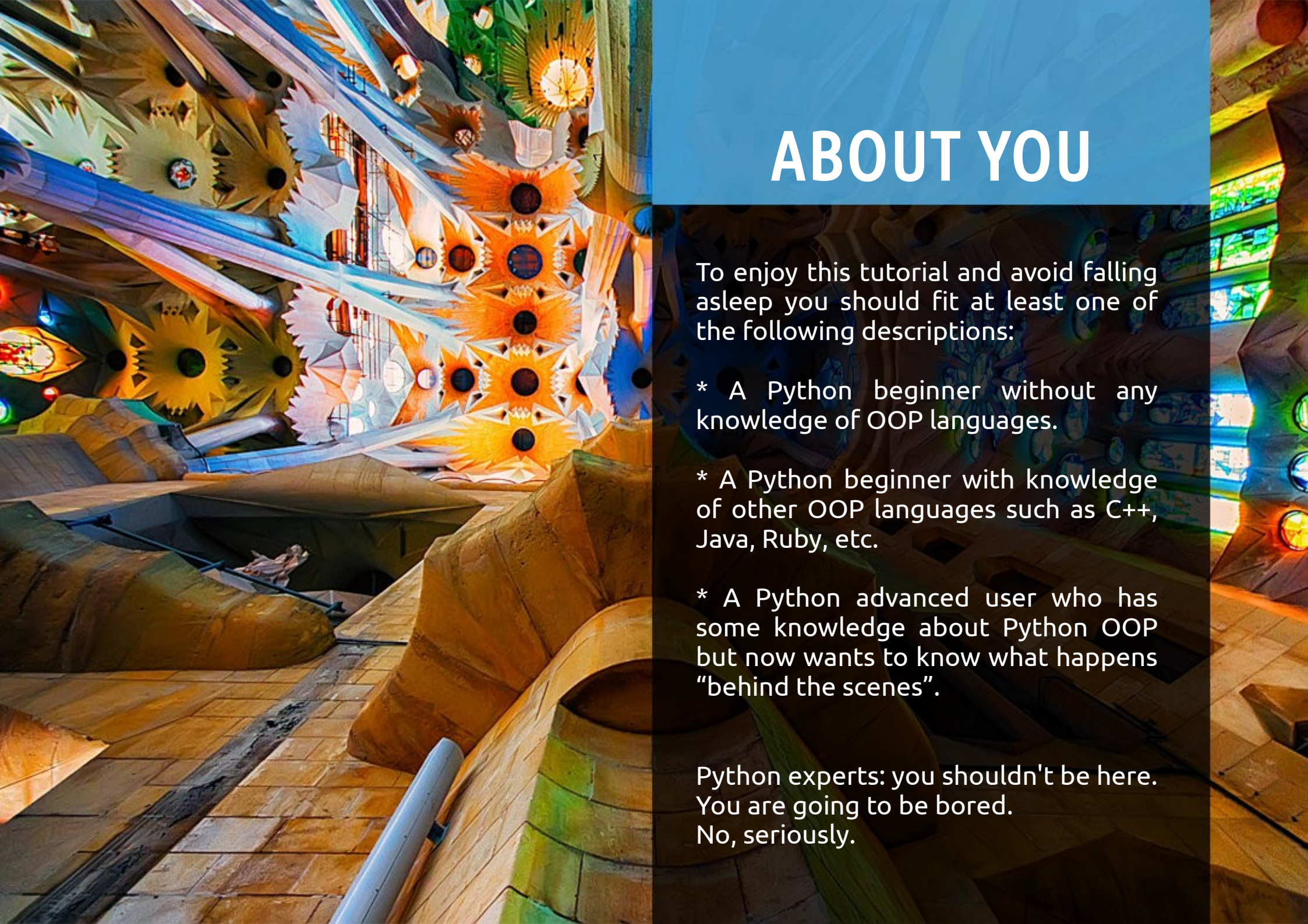
https://twitter.com/tw_lgiordani



<https://github.com/lgiordani>



<https://plus.google.com/u/LeonardoGiordani>

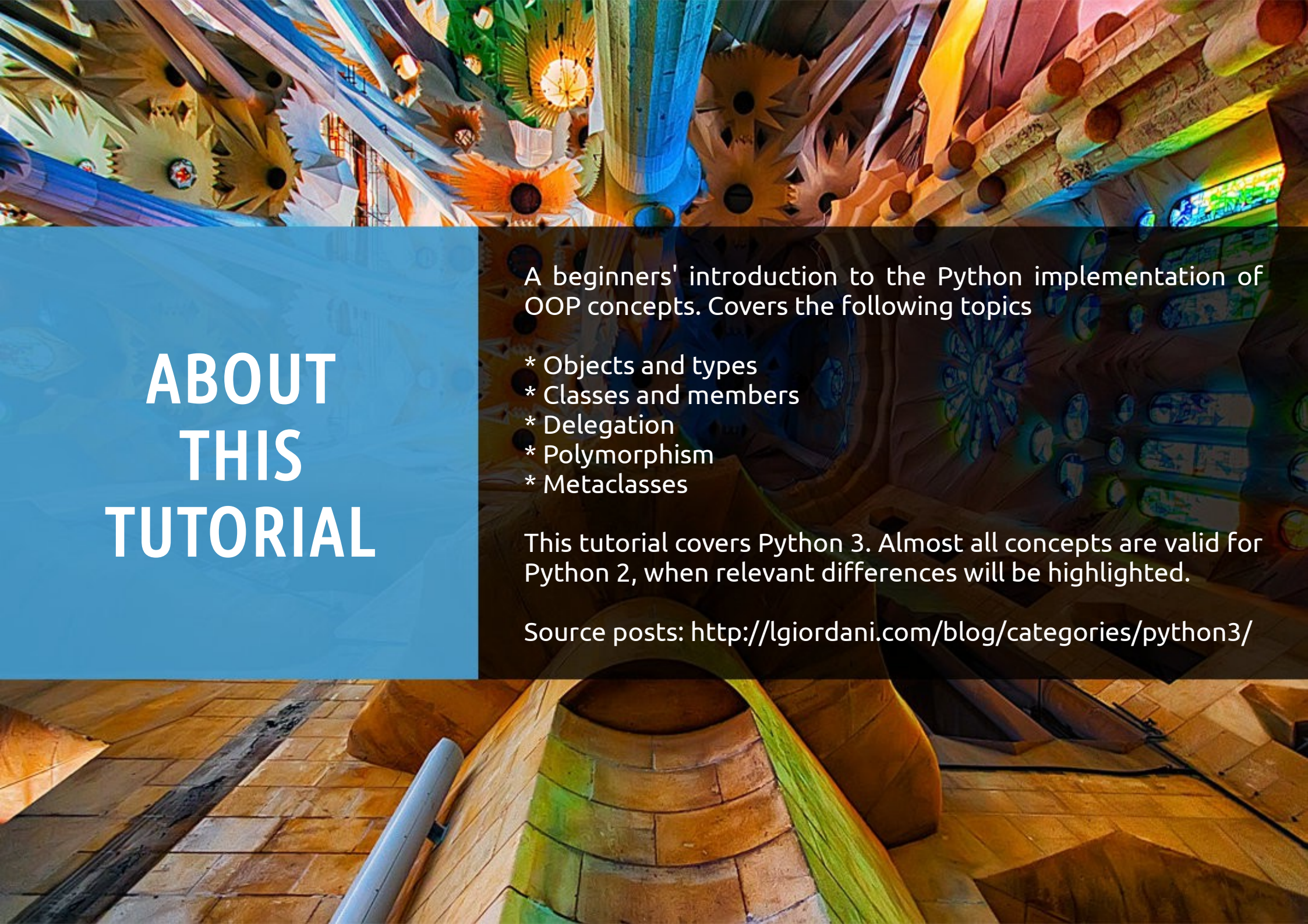


ABOUT YOU

To enjoy this tutorial and avoid falling asleep you should fit at least one of the following descriptions:

- * A Python beginner without any knowledge of OOP languages.
- * A Python beginner with knowledge of other OOP languages such as C++, Java, Ruby, etc.
- * A Python advanced user who has some knowledge about Python OOP but now wants to know what happens “behind the scenes”.

Python experts: you shouldn't be here. You are going to be bored. No, seriously.



ABOUT THIS TUTORIAL

A beginners' introduction to the Python implementation of OOP concepts. Covers the following topics

- * Objects and types
- * Classes and members
- * Delegation
- * Polymorphism
- * Metaclasses

This tutorial covers Python 3. Almost all concepts are valid for Python 2, when relevant differences will be highlighted.

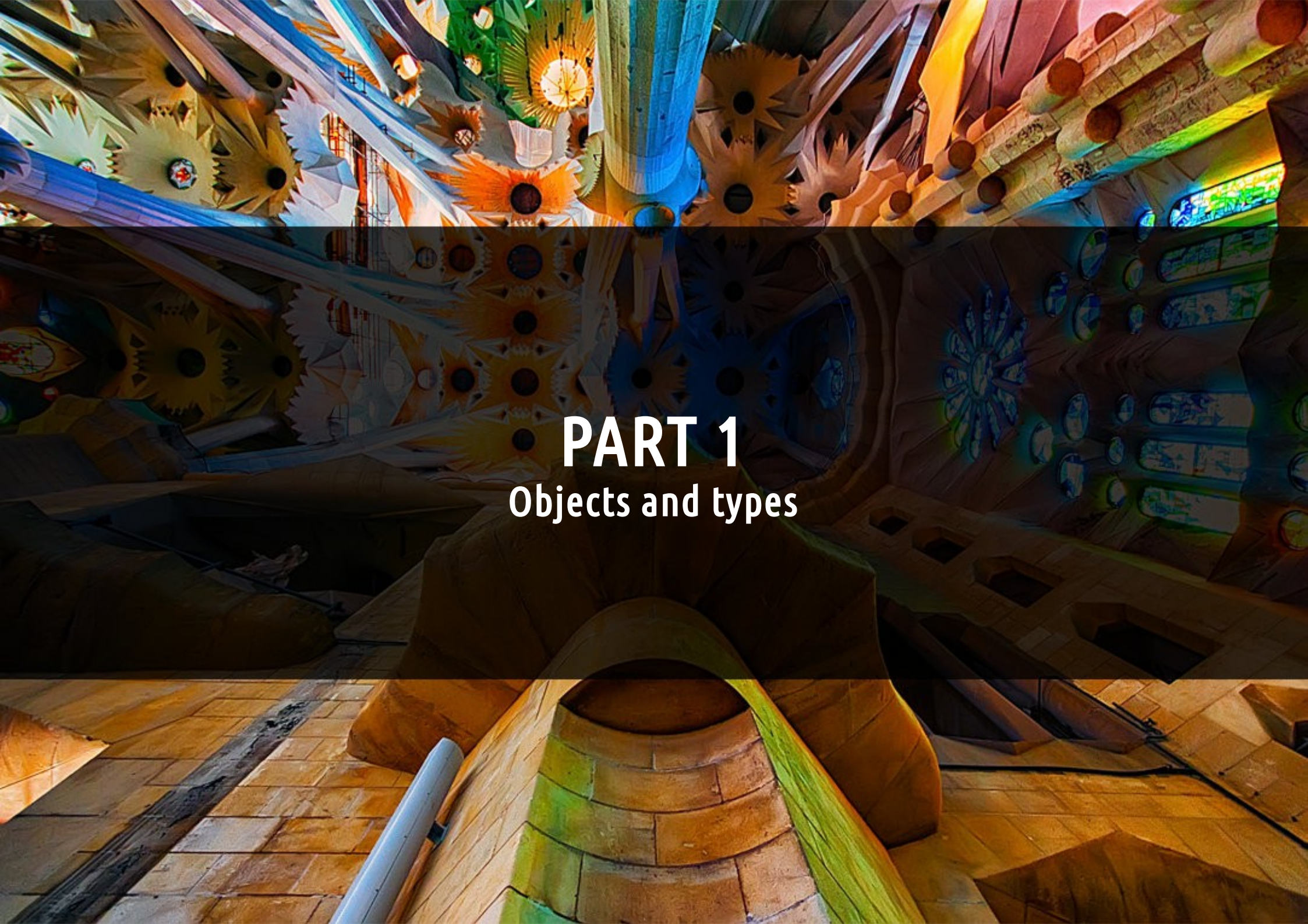
Source posts: <http://lgiordani.com/blog/categories/python3/>



SOURCE CODE

Code snippets used in this tutorial can be found at

[https://github.com/lgiordani/
deep_inside_oop_python](https://github.com/lgiordani/deep_inside_oop_python)



PART 1

Objects and types

Plain old procedures

```
1 # This is some data
2 data = (13, 63, 5, 378, 58, 40)
3
4 # This is a procedure that computes the average
5 def avg(d):
6     return sum(d)/len(d)
```

part1/01.py

```
>>> avg(data)
92.83333333333333
>>>
```

Very simple data: a sequence of numbers

Plain old procedures

Original data enters here

```
1 # This is some data
2 data = (13, 63, 5, 378, 58, 40)
3
4 # This is a procedure that computes the average
5 def avg(d):
6     return sum(d)/len(d)
```

part1/01.py

This returns new data

Procedures can modify data

```
1 # These are two numbered doors, initially closed
2 door1 = [1, 'closed']
3 door2 = [2, 'closed']
4
5 # This procedure opens a door
6 def open_door(door):
7     door[1] = 'open'
```

part1/02.py

```
>>> open_door(door1)
>>> door1
[1, 'open']
>>>
```

Things can get complicated

```
1 # These are two standard doors, initially closed
2 door1 = [1, 'closed']
3 door2 = [2, 'closed']
4
5 # This is a lockable door, initially closed and unlocked
6 ldoor1 = [1, 'closed', 'unlocked']
7
8 # This procedure opens a standard door
9 def open_door(door):
10     door[1] = 'open'
11
12 # This procedure opens a lockable door
13 def open_ldoor(door):
14     if door[2] == 'unlocked':
15         door[1] = 'open'
```

part1/03.py

```
>>> open_door(door1)
>>> door1
[1, 'open']
>>> open_ldoor(ldoor1)
>>> ldoor1
[1, 'open', 'unlocked']
```

Things can get complicated

```
1  # These are two standard doors, initially closed
2  door1 = [1, 'closed']
3  door2 = [2, 'closed']
4
5  # This is a lockable door, initially closed and unlocked
6  ldoor1 = [1, 'closed', 'unlocked']
7
8  # This procedure opens a standard door
9  def open_door(door):
10     door[1] = 'open'
11
12 # This procedure opens a lockable door
13 def open_ldoor(door):
14     if door[2] == 'unlocked':
15         door[1] = 'open'
```

part1/03.py

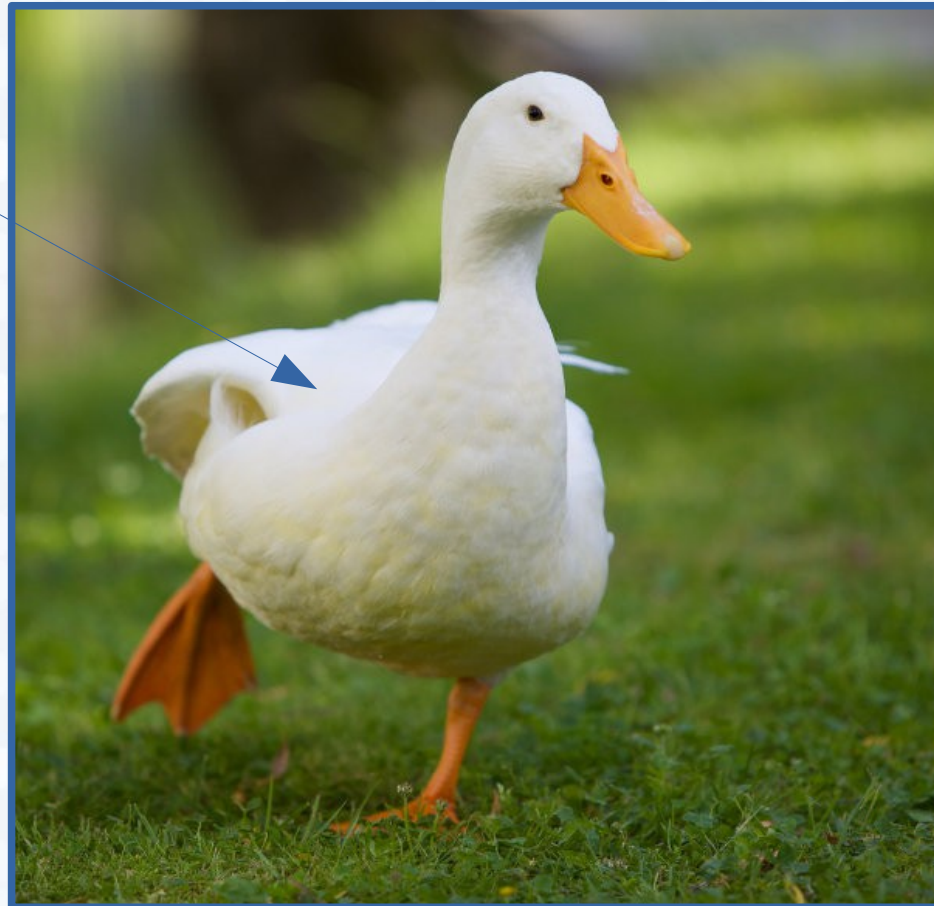
Same "action"

```
>>> open_door(door1)
>>> door1
[1, 'open']
>>> open_ldoor(ldoor1)
>>> ldoor1
[1, 'open', 'unlocked']
```

You must use the right function

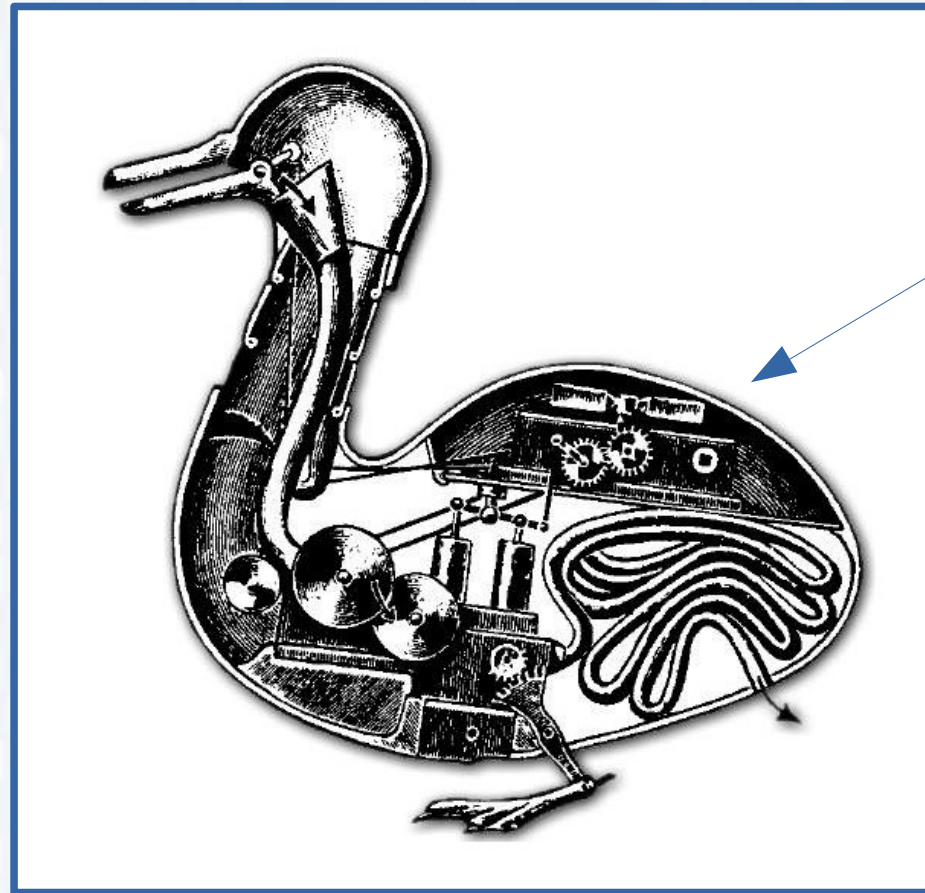
The meaning of the word 'type'

Behaves like a duck



Behavioural meaning

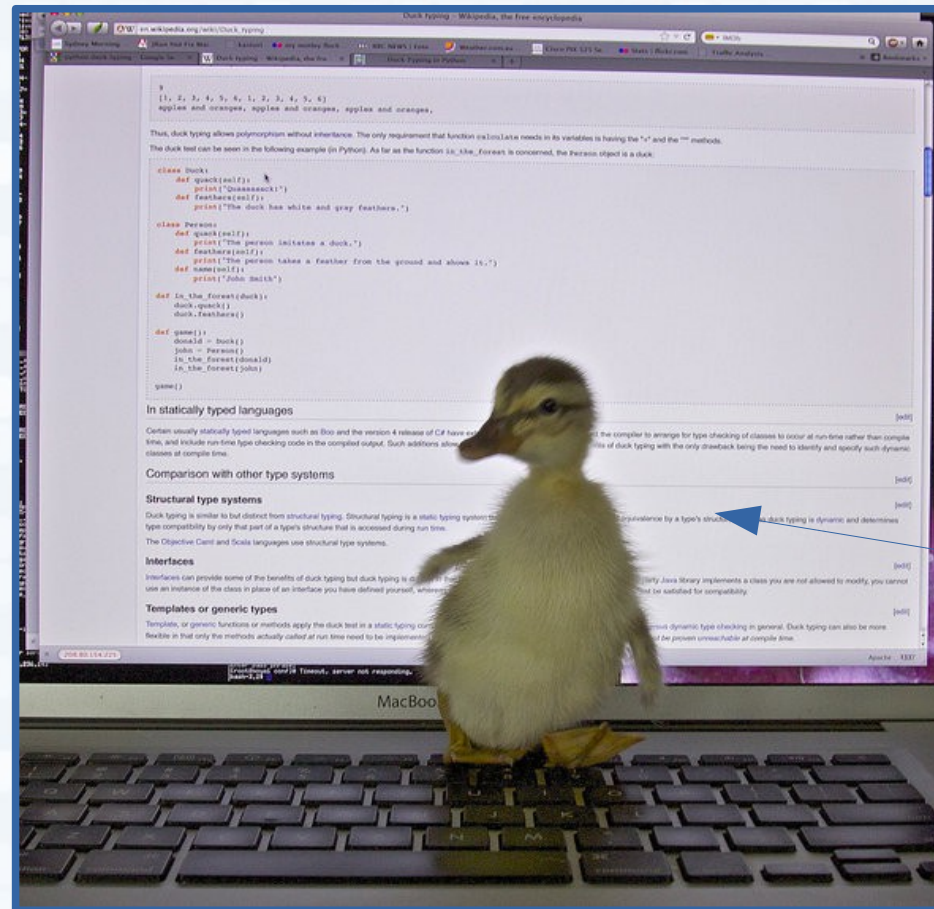
The meaning of the word 'type'



Dissection reveals
the truth

Structural meaning

The behavioural meaning is important

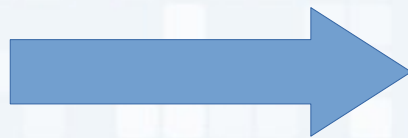
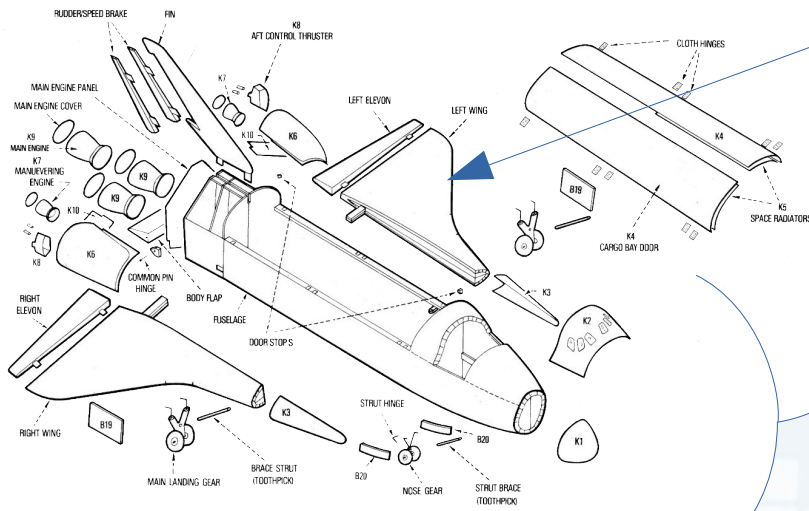


'Type' is a noun,
please.

Duck typing: making it behave like a duck

Class: the generic concept

Instance: a specific object of that type



Classes

```
>>> a = 6
>>> a
6
>>> type(a)
<class 'int'>
>>> a.__class__
<class 'int'>
>>>
```

Classes

```
>>> a = 6
>>> a
6
>>> type(a)
<type 'int'>
>>> a.__class__
<type 'int'>
>>>
```

Classes

```
>>> a = 6  
>>> a  
6  
>>> type(a)  
<class 'int'>  
>>> a.__class__  
<class 'int'>  
>>>
```

What is the type of 'a'?

The type of 'a' is the 'int' class

It is written INSIDE the variable!

The first class

```
1  class Door:
2      def __init__(self, number, status):
3          self.number = number
4          self.status = status
5
6      def open(self):
7          self.status = 'open'
8
9      def close(self):
10         self.status = 'closed'
```

part1/04.py

The first class

```
1  class Door(object):
2      def __init__(self, number, status):
3          self.number = number
4          self.status = status
5
6      def open(self):
7          self.status = 'open'
8
9      def close(self):
10         self.status = 'closed'
```

part1/04.py

The first class

The 'class' keyword defines the class

```
1  class Door:
2      def __init__(self, number, status):
3          self.number = number
4          self.status = status
5
6      def open(self):
7          self.status = 'open'
8
9      def close(self):
10         self.status = 'closed'
```

part1/04.py

Everything under the 'class' keyword is part of the class

The first class

```
1  class Door:
2      def __init__(self, number, status):
3          self.number = number
4          self.status = status
5
6      def open(self):
7          self.status = 'open'
8
9      def close(self):
10         self.status = 'closed'
```

part1/04.py

Functions inside a class are called 'methods', and must accept 'self' as the first parameter

The first class

Attributes

```
1 class Door:
2     def __init__(self, number, status):
3         self.number = number
4         self.status = status
5
6     def open(self):
7         self.status = 'open'
8
9     def close(self):
10        self.status = 'closed'
```

part1/04.py

Both attributes and methods are 'members' of the class

The first class

The `__init__()` special method, part of the constructor mechanism

```
1  class Door:
2      def __init__(self, number, status):
3          self.number = number
4          self.status = status
5
6      def open(self):
7          self.status = 'open'
8
9      def close(self):
10         self.status = 'closed'
```

part1/04.py

The first class

```
1  class Door:
2      def __init__(self, number, status):
3          self.number = number
4          self.status = status
5
6      def open(self):
7          self.status = 'open'
8
9      def close(self):
10         self.status = 'closed'
```

part1/04.py

```
>>> door1 = Door(1, 'closed')
>>>
```

The first class

```
1 class Door:
2     def __init__(self, number, status):
3         self.number = number
4         self.status = status
5
6     def open(self):
7         self.status = 'open'
8
9     def close(self):
10        self.status = 'closed'
```

part1/04.py

```
>>> door1 = Door(1, 'closed')
>>> type(door1)
<class '__main__.Door'>
>>>
```

The first class

```
1 class Door:
2     def __init__(self, number, status):
3         self.number = number
4         self.status = status
5
6     def open(self):
7         self.status = 'open'
8
9     def close(self):
10        self.status = 'closed'
```

part1/04.py

```
>>> door1 = Door(1, 'closed')
>>> type(door1)
<class '__main__.Door'>
>>> door1.number
1
>>> door1.status
'closed'
>>>
```

The first class

```
1  class Door:
2      def __init__(self, number, status):
3          self.number = number
4          self.status = status
5
6      def open(self):
7          self.status = 'open'
8
9      def close(self):
10         self.status = 'closed'
```

part1/04.py

```
>>> door1 = Door(1, 'closed')
>>> type(door1)
<class '__main__.Door'>
>>> door1.number
1
>>> door1.status
'closed'
>>> door1.open()
>>>
```

The first class

```
1 class Door:
2     def __init__(self, number, status):
3         self.number = number
4         self.status = status
5
6     def open(self):
7         self.status = 'open'
8
9     def close(self):
10        self.status = 'closed'
```

part1/04.py

```
>>> door1 = Door(1, 'closed')
>>> type(door1)
<class '__main__.Door'>
>>> door1.number
1
>>> door1.status
'closed'
>>> door1.open()
>>> door1.number
1
>>> door1.status
'open'
>>>
```



PART 2

Classes and members

Everything is an object

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

Everything is an object

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



The type of an object is an object itself

Where is the class of an object?

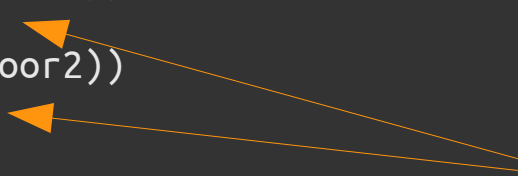
```
>>> door1 = Door(1, 'closed')  
>>> door2 = Door(1, 'closed')  
>>>
```

Where is the class of an object?

```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(1, 'closed')
>>> hex(id(door1))
'0xb67e148c'
>>> hex(id(door2))
'0xb67e144c'
>>>
```

Where is the class of an object?

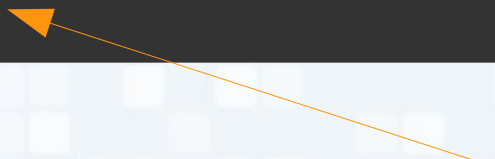
```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(1, 'closed')
>>> hex(id(door1))
'0xb67e148c'
>>> hex(id(door2))
'0xb67e144c'
>>>
```

A diagram consisting of two orange arrows. One arrow originates from the text box and points to the memory address '0xb67e148c' in the code output. The other arrow also originates from the text box and points to the memory address '0xb67e144c' in the code output.

Your numbers will be different from these

Where is the class of an object?

```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(1, 'closed')
>>> hex(id(door1))
'0xb67e148c'
>>> hex(id(door2))
'0xb67e144c'
>>> hex(id(door1.__class__))
'0xb685f56c'
>>> hex(id(door2.__class__))
'0xb685f56c'
```



The class is not a mere concept! It is an object in the system.

Class attributes

```
1  class Door:
2      colour = 'brown'
3
4      def __init__(self, number, status):
5          self.number = number
6          self.status = status
7
8      def open(self):
9          self.status = 'open'
10
11     def close(self):
12         self.status = 'closed'
```

part2/01.py

Class attributes

```
1  class Door:
2      colour = 'brown'
3
4      def __init__(self, number, status):
5          self.number = number
6          self.status = status
7
8      def open(self):
9          self.status = 'open'
10
11     def close(self):
12         self.status = 'closed'
```

No 'self' here

part2/01.py

Class attributes

```
1 class Door:
2     colour = 'brown'
   [...]
```

part2/01.py

```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(2, 'closed')
>>>
```

Class attributes

```
1 class Door:
2     colour = 'brown'
   [...]
```

part2/01.py

```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(2, 'closed')
>>> Door.colour
'brown'
>>> door1.colour
'brown'
>>> door2.colour
'brown'
>>>
```

Class attributes

```
1 class Door:
2     colour = 'brown'
   [...]
```

part2/01.py

```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(2, 'closed')
>>> Door.colour
'brown'
>>> door1.colour
'brown'
>>> door2.colour
'brown'
>>> Door.colour = 'white'
>>> Door.colour
'white'
>>> door1.colour
'white'
>>> door2.colour
'white'
>>>
```

Class attributes

```
1 class Door:
2     colour = 'brown'
   [...]
```

part2/01.py

```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(2, 'closed')
>>> Door.colour
'brown'
>>> door1.colour
'brown'
>>> door2.colour
'brown'
>>> Door.colour = 'white'
>>> Door.colour
'white'
>>> door1.colour
'white'
>>> door2.colour
'white'
>>> hex(id(Door.colour))
'0xb67e1500'
>>> hex(id(door1.colour))
'0xb67e1500'
>>> hex(id(door2.colour))
'0xb67e1500'
>>>
```

Let's dive into attribute resolution

```
1 class Door:
2     colour = 'brown'
   [...]
```

part2/01.py

```
>>> Door.__dict__
mappingproxy({'open': <function Door.open at 0xb68604ac>,
             'colour': 'white',
             '__dict__': <attribute '__dict__' of 'Door' objects>,
             '__weakref__': <attribute '__weakref__' of 'Door' objects>,
             '__init__': <function Door.__init__ at 0xb7062854>,
             '__module__': '__main__',
             '__doc__': None,
             'close': <function Door.close at 0xb686041c>})
>>>
```

Let's dive into attribute resolution

```
1 class Door:
2     colour = 'brown'
   [...]
```

part2/01.py

```
>>> Door.__dict__
mappingproxy({'open': <function Door.open at 0xb68604ac>,
             'colour': 'white',
             '__dict__': <attribute '__dict__' of 'Door' objects>,
             '__weakref__': <attribute '__weakref__' of 'Door' objects>,
             '__init__': <function Door.__init__ at 0xb7062854>,
             '__module__': '__main__',
             '__doc__': None,
             'close': <function Door.close at 0xb686041c>})
>>> door1.__dict__
{'number': 1, 'status': 'closed'}
>>>
```

Let's dive into attribute resolution

```
1 class Door:
2     colour = 'brown'
   [...]
```

part2/01.py

```
>>> Door.__dict__
mappingproxy({'open': <function Door.open at 0xb68604ac>,
             'colour': 'white',
             '__dict__': <attribute '__dict__' of 'Door' objects>,
             '__weakref__': <attribute '__weakref__' of 'Door' objects>,
             '__init__': <function Door.__init__ at 0xb7062854>,
             '__module__': '__main__',
             '__doc__': None,
             'close': <function Door.close at 0xb686041c>})
>>> door1.__dict__
{'number': 1, 'status': 'closed'}
>>> door1.__dict__['colour']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'colour'
>>>
```

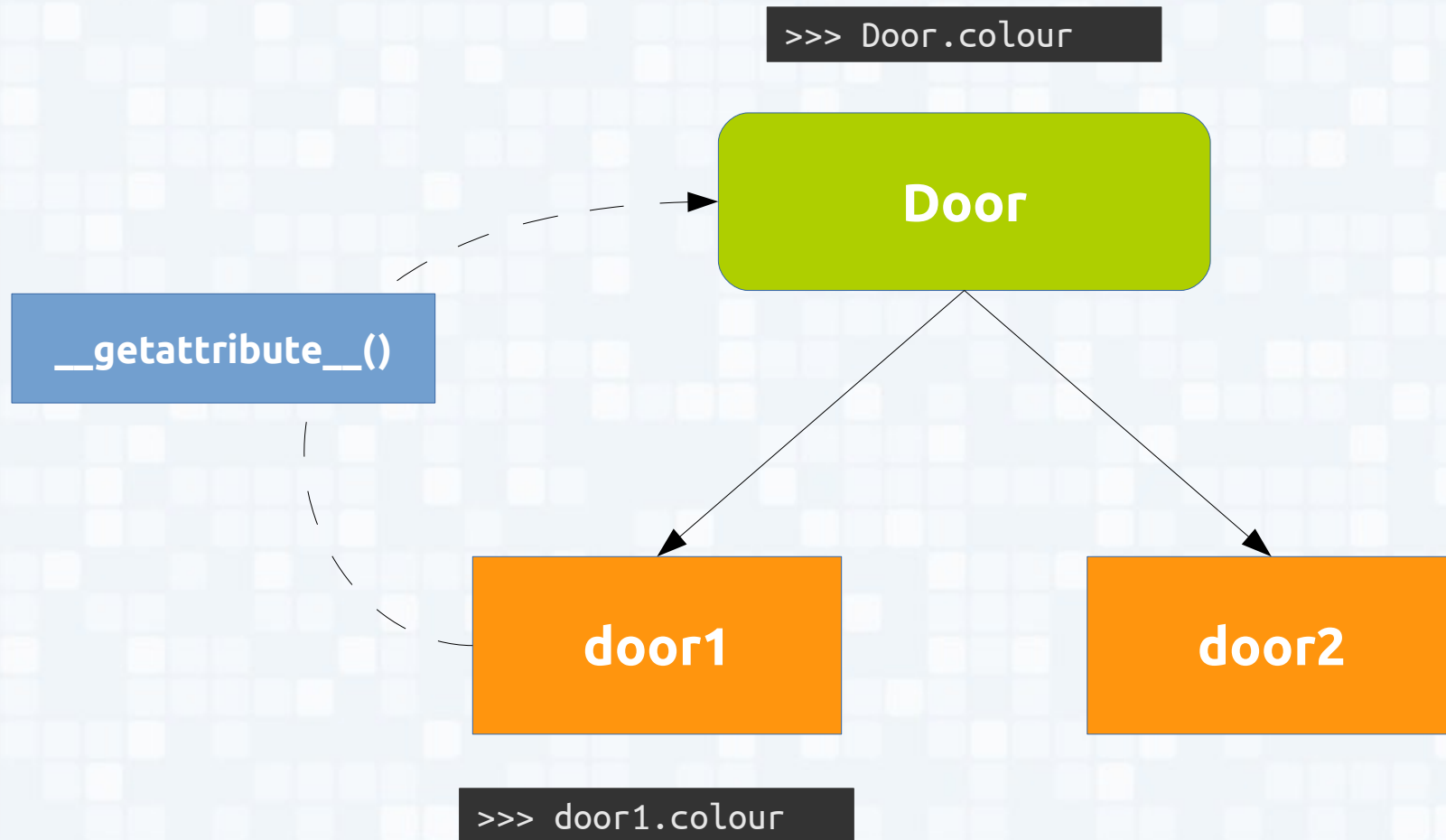
Let's dive into attribute resolution

```
1 class Door:
2     colour = 'brown'
   [...]
```

part2/01.py

```
>>> Door.__dict__
mappingproxy({'open': <function Door.open at 0xb68604ac>,
             'colour': 'white',
             '__dict__': <attribute '__dict__' of 'Door' objects>,
             '__weakref__': <attribute '__weakref__' of 'Door' objects>,
             '__init__': <function Door.__init__ at 0xb7062854>,
             '__module__': '__main__',
             '__doc__': None,
             'close': <function Door.close at 0xb686041c>})
>>> door1.__dict__
{'number': 1, 'status': 'closed'}
>>> door1.__dict__['colour']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'colour'
>>> door1.__class__.__dict__['colour']
'white'
>>> door1.colour is Door.colour
True
```

Let's dive into attribute resolution



Let's dive into attribute resolution

```
1 class Door:
2     colour = 'brown'
   [...]
```

part2/01.py

Remember to execute the class again (we manually changed the class 'colour' attribute)

```
>>> door1 = Door(1, 'closed')
>>> door1.colour = 'white'
>>>
```

Let's dive into attribute resolution

```
1 class Door:  
2     colour = 'brown'  
    [...]
```

part2/01.py

```
>>> door1 = Door(1, 'closed')  
>>> door1.colour = 'white'  
>>> door1.__dict__['colour']  
'white'  
>>> door1.__class__.__dict__['colour']  
'brown'  
>>>
```

Let's dive into attribute resolution

```
1 class Door:  
2     colour = 'brown'  
    [...]
```

part2/01.py

```
>>> door1 = Door(1, 'closed')  
>>> door1.colour = 'white'  
>>> door1.__dict__['colour']  
'white'  
>>> door1.__class__.__dict__['colour']  
'brown'  
>>> door1.colour  
'white'  
>>> Door.colour  
'brown'  
>>>
```

Let's dive into attribute resolution

```
1 class Door:  
2     colour = 'brown'  
    [...]
```

part2/01.py

```
>>> door1 = Door(1, 'closed')  
>>> door1.colour = 'white'  
>>> door1.__dict__['colour']  
'white'  
>>> door1.__class__.__dict__['colour']  
'brown'  
>>> door1.colour  
'white'  
>>> Door.colour  
'brown'  
>>> Door.colour = 'red'  
>>>
```

Let's dive into attribute resolution

```
1 class Door:
2     colour = 'brown'
   [...]
```

part2/01.py

```
>>> door1 = Door(1, 'closed')
>>> door1.colour = 'white'
>>> door1.__dict__['colour']
'white'
>>> door1.__class__.__dict__['colour']
'brown'
>>> door1.colour
'white'
>>> Door.colour
'brown'
>>> Door.colour = 'red'
>>> door1.__dict__['colour']
'white'
>>> door1.__class__.__dict__['colour']
'red'
>>>
```

What about methods?

```
>>> door1 = Door(1, 'closed')
>>> Door.__dict__
mappingproxy({'open': <function Door.open at 0xb68604ac>,
             'colour': 'red',
             '__dict__': <attribute '__dict__' of 'Door' objects>,
             '__weakref__': <attribute '__weakref__' of 'Door' objects>,
             '__init__': <function Door.__init__ at 0xb7062854>,
             '__module__': '__main__',
             '__doc__': None,
             'close': <function Door.close at 0xb686041c>})
>>> door1.__dict__
{'number': 1, 'status': 'closed'}
>>> door1.colour is Door.colour
True
>>>
```

What about methods?

```
>>> door1 = Door(1, 'closed')
>>> Door.__dict__
mappingproxy({'open': <function Door.open at 0xb68604ac>,
              'colour': 'brown',
              '__dict__': <attribute '__dict__' of 'Door' objects>,
              '__weakref__': <attribute '__weakref__' of 'Door' objects>,
              '__init__': <function Door.__init__ at 0xb7062854>,
              '__module__': '__main__',
              '__doc__': None,
              'close': <function Door.close at 0xb686041c>})
>>> door1.__dict__
{'number': 1, 'status': 'closed'}
>>> door1.colour is Door.colour
True
>>> door1.open is Door.open
False
>>>
```

What about methods?

```
>>> door1 = Door(1, 'closed')
>>> Door.__dict__
mappingproxy({'open': <function Door.open at 0xb68604ac>,
              'colour': 'brown',
              '__dict__': <attribute '__dict__' of 'Door' objects>,
              '__weakref__': <attribute '__weakref__' of 'Door' objects>,
              '__init__': <function Door.__init__ at 0xb7062854>,
              '__module__': '__main__',
              '__doc__': None,
              'close': <function Door.close at 0xb686041c>})
>>> door1.__dict__
{'number': 1, 'status': 'closed'}
>>> door1.colour is Door.colour
True
>>> door1.open is Door.open
False
>>> Door.__dict__['open']
<function Door.open at 0xb68604ac>
>>> Door.open
<function Door.open at 0xb68604ac>
>>>
```

What about methods?

```
>>> door1 = Door(1, 'closed')
>>> Door.__dict__
mappingproxy({'open': <function Door.open at 0xb68604ac>,
              'colour': 'brown',
              '__dict__': <attribute '__dict__' of 'Door' objects>,
              '__weakref__': <attribute '__weakref__' of 'Door' objects>,
              '__init__': <function Door.__init__ at 0xb7062854>,
              '__module__': '__main__',
              '__doc__': None,
              'close': <function Door.close at 0xb686041c>})
>>> door1.__dict__
{'number': 1, 'status': 'closed'}
>>> door1.colour is Door.colour
True
>>> door1.open is Door.open
False
>>> Door.__dict__['open']
<function Door.open at 0xb68604ac>
>>> Door.open
<function Door.open at 0xb68604ac>
>>> door1.open
<bound method Door.open of <__main__.Door object at 0xb67e162c>>
>>>
```

What about methods?

```
>>> door1 = Door(1, 'closed')
>>> Door.__dict__
dict_proxy({'__module__': '__main__',
            'colour': 'brown',
            '__weakref__': <attribute '__weakref__' of 'Door' objects>,
            '__dict__': <attribute '__dict__' of 'Door' objects>,
            'close': <function close at 0xb6a8a56c>,
            'open': <function open at 0xb6a8a534>,
            '__doc__': None,
            '__init__': <function __init__ at 0xb6a8a48c>})
>>> door1.__dict__
{'status': 'closed', 'number': 1}
>>> door1.colour is Door.colour
True
>>> door1.open is Door.open
False
>>> Door.__dict__['open']
<function open at 0xb68604ac>
>>> Door.open
<unbound method Door.open>
>>> door1.open
<bound method Door.open of <__main__.Door object at 0xb67e162c>>
>>>
```

From functions to bound methods

```
>>> Door.open()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: open() missing 1 required positional argument: 'self'  
>>>
```

From functions to bound methods

```
>>> Door.open()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unbound method open() must be called with Door instance  
as first argument (got nothing instead)  
>>>
```

From functions to bound methods

```
>>> Door.open()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: open() missing 1 required positional argument: 'self'
>>> Door.open(door1)
>>> door1.status
'open'
>>>
```

From functions to bound methods

```
>>> Door.open()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: open() missing 1 required positional argument: 'self'
>>> Door.open(door1)
>>> door1.status
'open'
>>> door1.__class__.__dict__['open']
<function Door.open at 0xb68604ac>
>>>
```

From functions to bound methods

```
>>> Door.open()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: open() missing 1 required positional argument: 'self'
>>> Door.open(door1)
>>> door1.status
'open'
>>> door1.__class__.__dict__['open']
<function Door.open at 0xb68604ac>
>>> dir(door1.__class__.__dict__['open'])
['__annotations__', '__call__', '__class__', '__closure__',
 '__code__', '__defaults__', '__delattr__', '__dict__', '__dir__',
 '__doc__', '__eq__', '__format__', '__ge__', '__get__',
 '__getattr__', '__globals__', '__gt__', '__hash__', '__init__',
 '__kwdefaults__', '__le__', '__lt__', '__module__', '__name__',
 '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__',
 '__repr__', '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__']
>>>
```

From functions to bound methods

```
>>> Door.open()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: open() missing 1 required positional argument: 'self'
>>> Door.open(door1)
>>> door1.status
'open'
>>> door1.__class__.__dict__['open']
<function Door.open at 0xb68604ac>
>>> dir(door1.__class__.__dict__['open'])
['__annotations__', '__call__', '__class__', '__closure__',
 '__code__', '__defaults__', '__delattr__', '__dict__', '__dir__',
 '__doc__', '__eq__', '__format__', '__ge__', '__get__',
 '__getattr__', '__globals__', '__gt__', '__hash__', '__init__',
 '__kwdefaults__', '__le__', '__lt__', '__module__', '__name__',
 '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__',
 '__repr__', '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__']
>>> door1.__class__.__dict__['open'].__get__
<method-wrapper '__get__' of function object at 0xb68604ac>
>>>
```

From functions to bound methods

```
>>> Door.open()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: open() missing 1 required positional argument: 'self'
>>> Door.open(door1)
>>> door1.status
'open'
>>> door1.__class__.__dict__['open']
<function Door.open at 0xb68604ac>
>>> dir(door1.__class__.__dict__['open'])
['__annotations__', '__call__', '__class__', '__closure__',
 '__code__', '__defaults__', '__delattr__', '__dict__', '__dir__',
 '__doc__', '__eq__', '__format__', '__ge__', '__get__',
 '__getattr__', '__globals__', '__gt__', '__hash__', '__init__',
 '__kwdefaults__', '__le__', '__lt__', '__module__', '__name__',
 '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__',
 '__repr__', '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__']
>>> door1.__class__.__dict__['open'].__get__
<method-wrapper '__get__' of function object at 0xb68604ac>
>>> door1.__class__.__dict__['open'].__get__(door1)
<bound method Door.open of <__main__.Door object at 0xb67e162c>>
>>>
```

From functions to bound methods

```
>>> door1.__class__.__dict__['open'].__get__(door1)
<bound method ?.open of <__main__.Door instance at 0xb6977aac>>
>>> door1.__class__.__dict__['open'].__get__(door1, Door)
<bound method Door.open of <__main__.Door object at 0xb73f956c>>
>>>
```

Class methods

part2/02.py

```
1  class Door:
2      colour = 'brown'
3
4      def __init__(self, number, status):
5          self.number = number
6          self.status = status
7
8      @classmethod
9      def knock(cls):
10         print("Knock!")
11
12     def open(self):
13         self.status = 'open'
14
15     def close(self):
16         self.status = 'closed'
```

```
>>> door1 = Door(1, 'closed')
>>> door1.knock()
Knock!
>>> Door.knock()
Knock!
>>>
```

Class methods

part2/03.py

```
1 class Door:
2     colour = 'brown'
3
4     def __init__(self, number, status):
5         self.number = number
6         self.status = status
7
8     @classmethod
9     def knock(cls):
10         print("Knock!")
11
12     @classmethod
13     def paint(cls, colour):
14         cls.colour = colour
15
16     def open(self):
17         self.status = 'open'
18
19     def close(self):
20         self.status = 'closed'
```

Class methods

```
>>> door1 = Door(1, 'closed')  
>>> door2 = Door(2, 'closed')  
>>>
```

Class methods

```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(2, 'closed')
>>> Door.colour
'brown'
>>> door1.colour
'brown'
>>> door2.colour
'brown'
>>>
```

Class methods

```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(2, 'closed')
>>> Door.colour
'brown'
>>> door1.colour
'brown'
>>> door2.colour
'brown'
>>> Door.paint('white')
>>>
```

Class methods

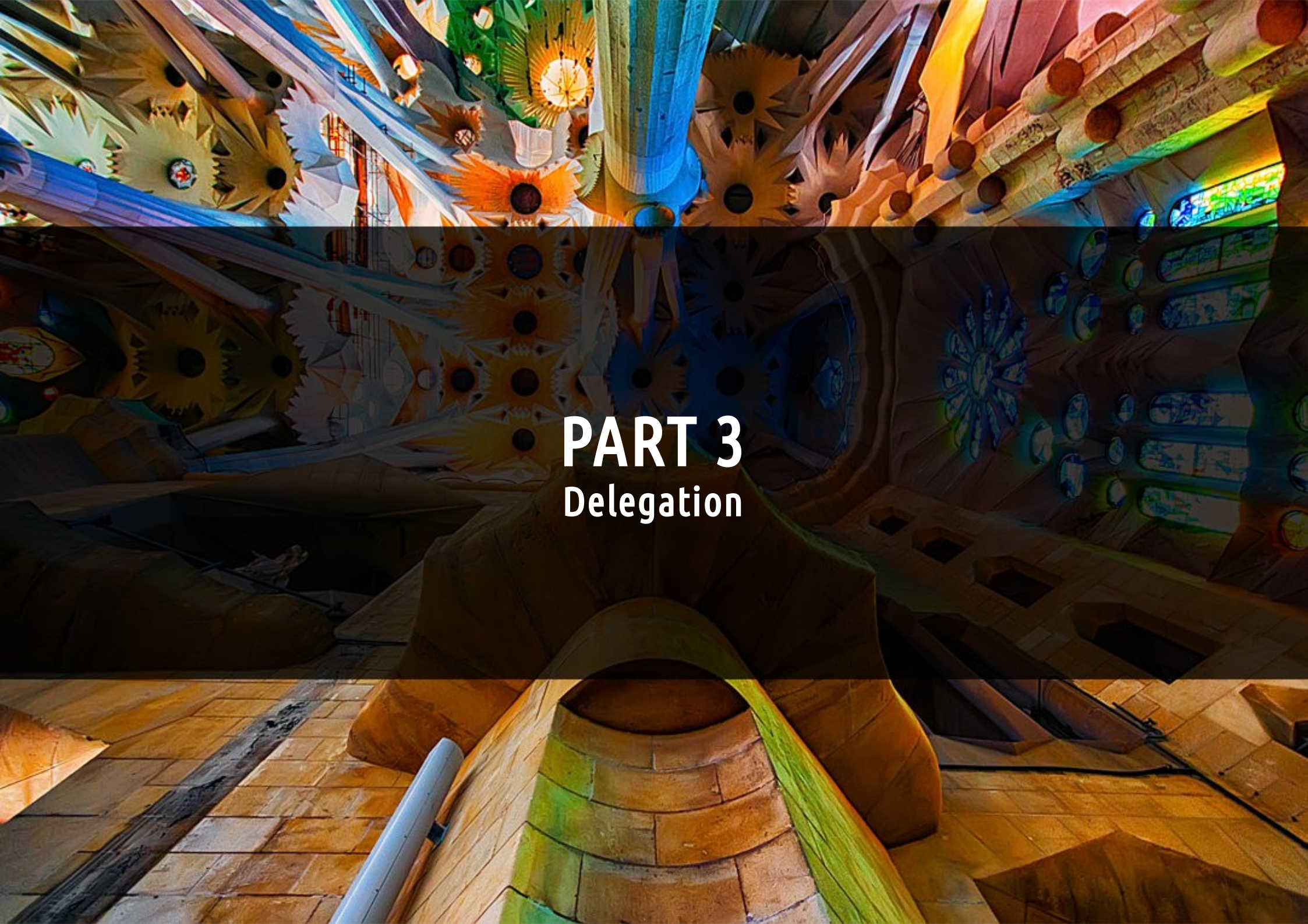
```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(2, 'closed')
>>> Door.colour
'brown'
>>> door1.colour
'brown'
>>> door2.colour
'brown'
>>> Door.paint('white')
>>> Door.colour
'white'
>>> door1.colour
'white'
>>> door2.colour
'white'
>>>
```

Class methods

```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(2, 'closed')
>>> Door.colour
'brown'
>>> door1.colour
'brown'
>>> door2.colour
'brown'
>>> Door.paint('white')
>>> Door.colour
'white'
>>> door1.colour
'white'
>>> door2.colour
'white'
>>> door1.paint('yellow')
>>>
```

Class methods

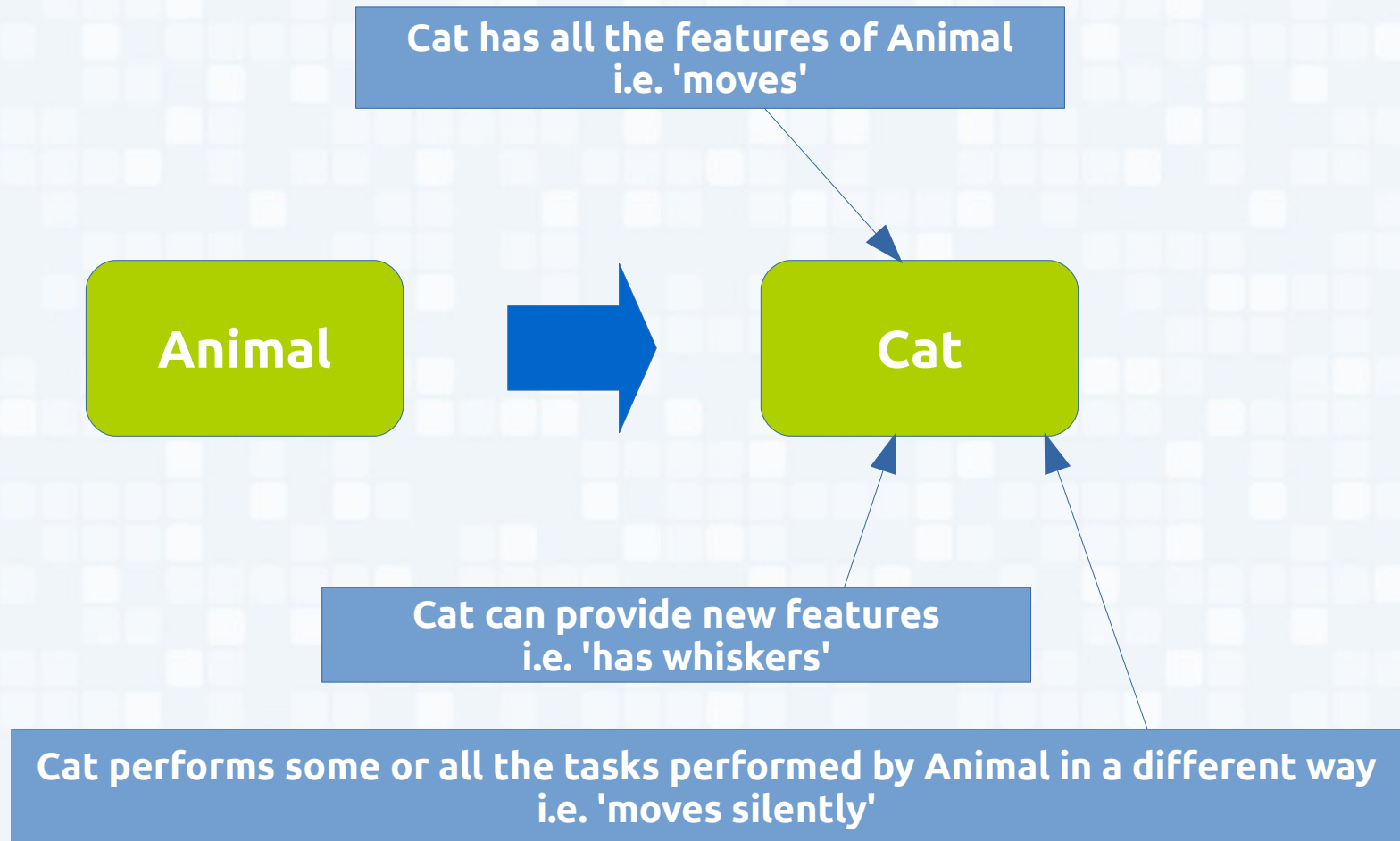
```
>>> door1 = Door(1, 'closed')
>>> door2 = Door(2, 'closed')
>>> Door.colour
'brown'
>>> door1.colour
'brown'
>>> door2.colour
'brown'
>>> Door.paint('white')
>>> Door.colour
'white'
>>> door1.colour
'white'
>>> door2.colour
'white'
>>> door1.paint('yellow')
>>> Door.colour
'yellow'
>>> door1.colour
'yellow'
>>> door2.colour
'yellow'
>>>
```



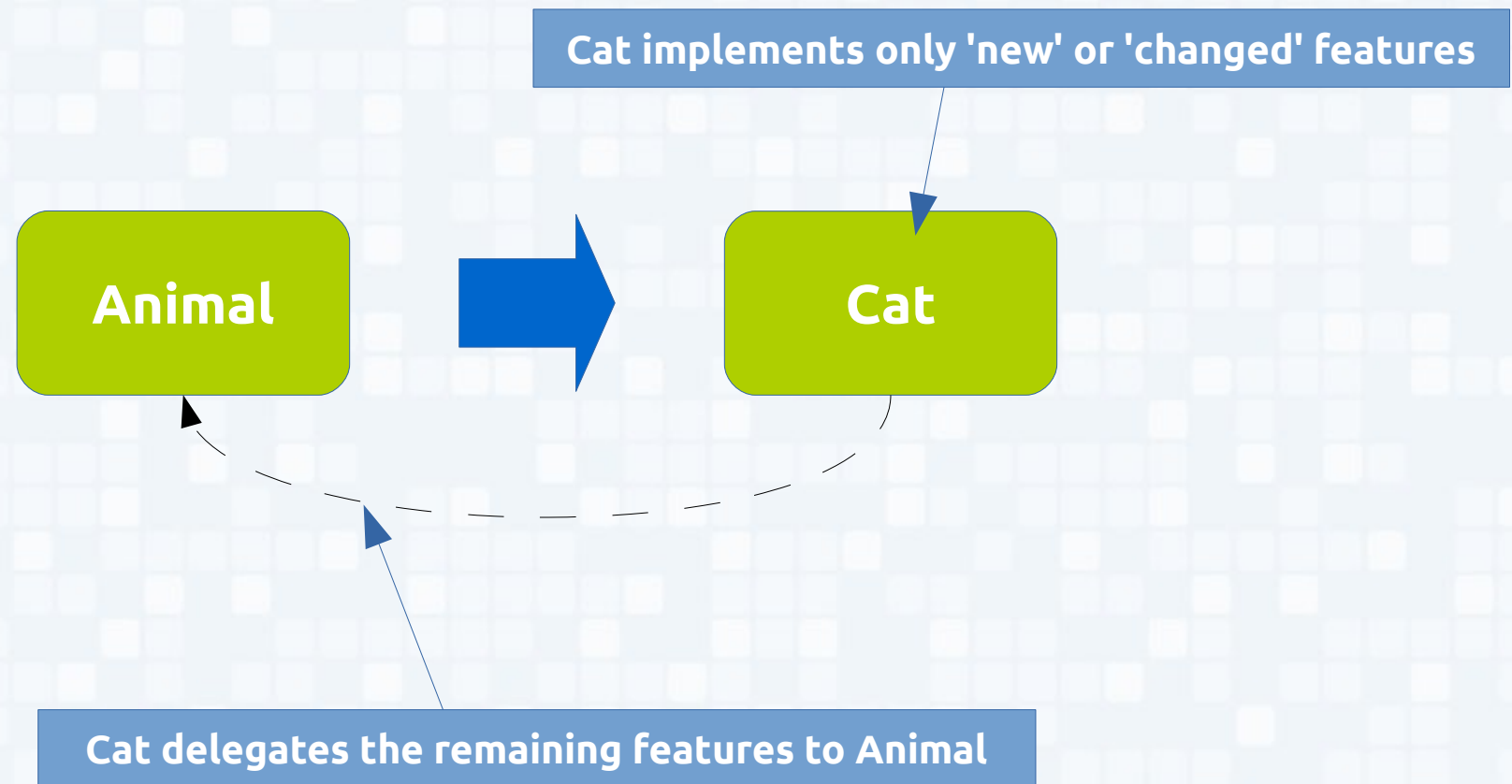
PART 3

Delegation

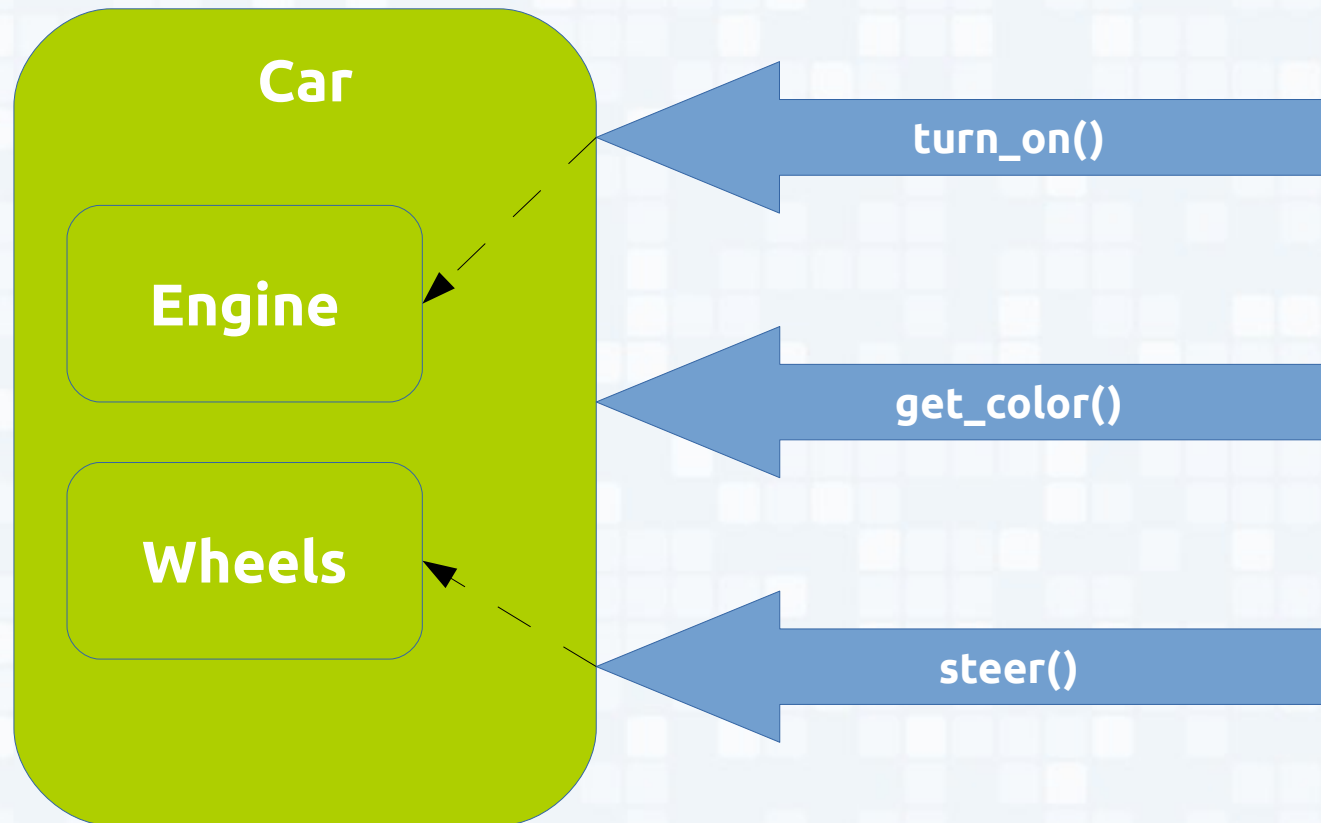
Specialization



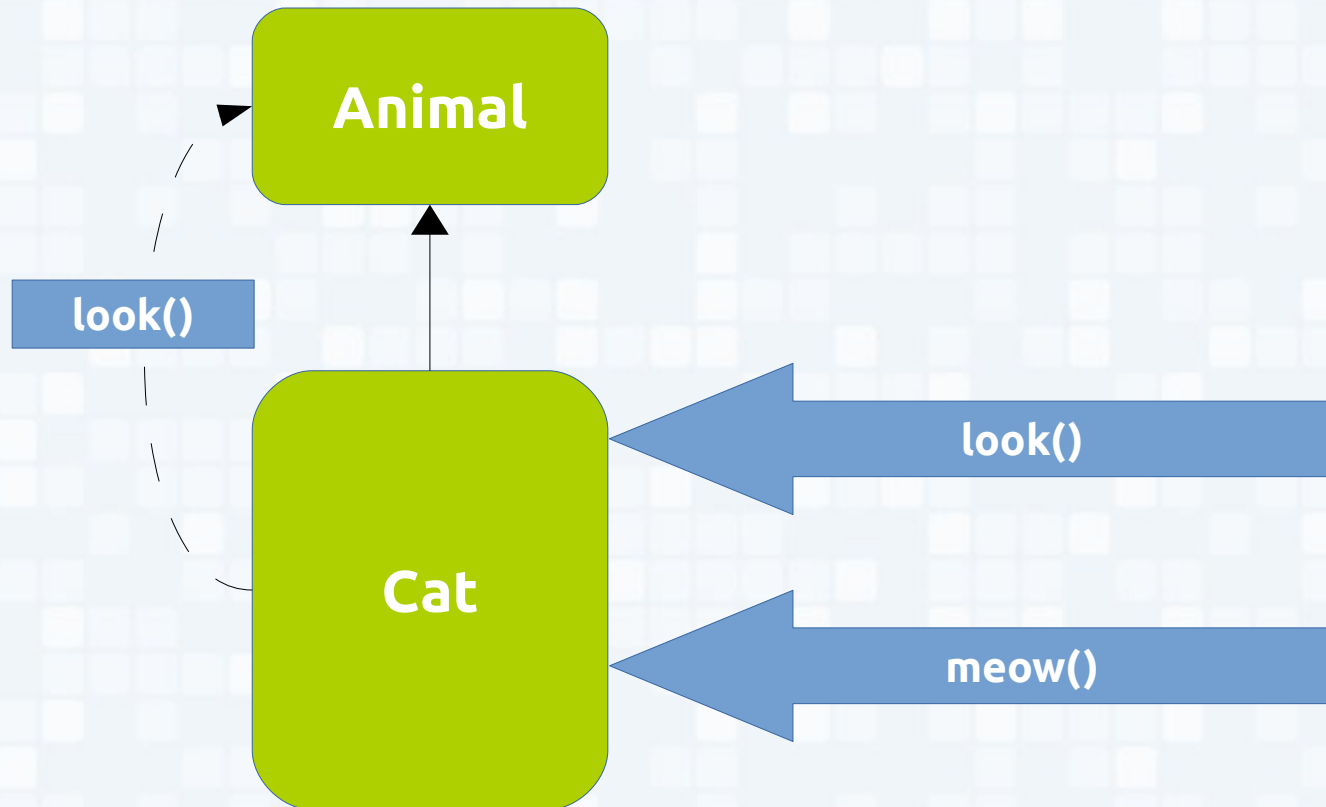
Delegation



Composition: 'has'



Inheritance: 'is'



LOVE YOUR SYSTEM

www.compositionoverinheritance.com



Inheritance

```
1 class SecurityDoor(Door):  
2     pass
```

part3/01.py

```
>>> sdoor = SecurityDoor(1, 'closed')  
>>>
```

Inheritance

```
1 class SecurityDoor(Door):  
2     pass
```

part3/01.py

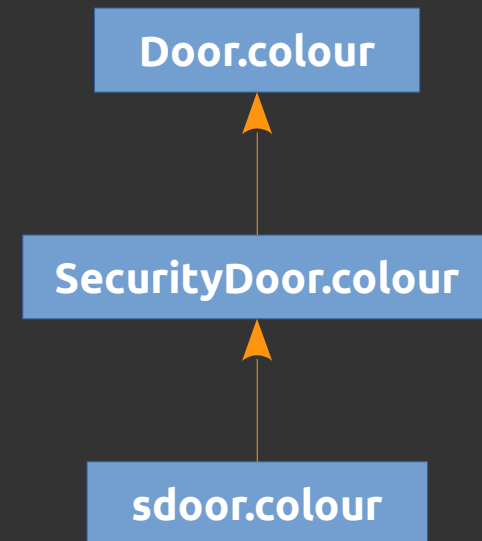
```
>>> sdoor = SecurityDoor(1, 'closed')  
>>> SecurityDoor.colour is Door.colour  
True  
>>> sdoor.colour is Door.colour  
True
```

Inheritance

```
1 class SecurityDoor(Door):  
2     pass
```

part3/01.py

```
>>> sdoor = SecurityDoor(1, 'closed')  
>>> SecurityDoor.colour is Door.colour  
True  
>>> sdoor.colour is Door.colour  
True
```



Inheritance

```
1 class SecurityDoor(Door):  
2     pass
```

part3/01.py

```
>>> sdoor.__dict__  
{'number': 1, 'status': 'closed'}  
>>>
```

Inheritance

```
1 class SecurityDoor(Door):  
2     pass
```

part3/01.py

```
>>> sdoor.__dict__  
{'number': 1, 'status': 'closed'}  
>>> sdoor.__class__.__dict__  
mappingproxy({'__doc__': None, '__module__': '__main__'})  
>>>
```

Inheritance

```
1 class SecurityDoor(Door):  
2     pass
```

part3/01.py

```
>>> sdoor.__dict__  
{'number': 1, 'status': 'closed'}  
>>> sdoor.__class__.__dict__  
mappingproxy({'__doc__': None, '__module__': '__main__'})  
>>> Door.__dict__  
mappingproxy({'__dict__': <attribute '__dict__' of 'Door' objects>,  
              'colour': 'yellow',  
              'open': <function Door.open at 0xb687e224>,  
              '__init__': <function Door.__init__ at 0xb687e14c>,  
              '__doc__': None,  
              'close': <function Door.close at 0xb687e1dc>,  
              'knock': <classmethod object at 0xb67ff6ac>,  
              '__weakref__': <attribute '__weakref__' of 'Door' objects>,  
              '__module__': '__main__',  
              'paint': <classmethod object at 0xb67ff6ec>})  
>>>
```

Inheritance

```
1 class SecurityDoor(Door):  
2     pass
```

part3/01.py

```
>>> SecurityDoor.__bases__  
(<class '__main__.Door'>,)  
>>>
```

Inheritance

```
1 class SecurityDoor(Door):  
2     pass
```

part3/01.py

```
>>> SecurityDoor.__bases__  
(<class '__main__.Door'>,)  
>>> sdoor.knock  
<bound method type.knock of <class '__main__.SecurityDoor'>>  
>>>
```

Inheritance

```
1 class SecurityDoor(Door):  
2     pass
```

part3/01.py

```
>>> SecurityDoor.__bases__  
(<class '__main__.Door'>,)  
>>> sdoor.knock  
<bound method type.knock of <class '__main__.SecurityDoor'>>  
>>> sdoor.__class__.__bases__[0].__dict__['knock'].__get__(sdoor)  
<bound method type.knock of <class '__main__.SecurityDoor'>>  
>>>
```

Overriding

```
1 class SecurityDoor(Door):  
2     colour = 'grey'  
3     locked = True  
4  
5     def open(self):  
6         if not self.locked:  
7             self.status = 'open'
```

part3/02.py

Overriding

```
1 class SecurityDoor(Door):
2     colour = 'grey'
3     locked = True
4
5     def open(self):
6         if not self.locked:
7             self.status = 'open'
```

part3/02.py

Overriding blocks implicit delegation

```
>>> SecurityDoor.__dict__
mappingproxy({'__doc__': None,
              '__module__': '__main__',
              'open': <function SecurityDoor.open at 0xb6fcf89c>,
              'colour': 'grey',
              'locked': True})
>>>
```

Calling the parent implementation

```
1 class SecurityDoor(Door):
2     colour = 'grey'
3     locked = True
4
5     def open(self):
6         if self.locked:
7             return
8         Door.open(self)
```

part3/03.py

```
>>> sdoor = SecurityDoor(1, 'closed')
>>>
```

Calling the parent implementation

```
1 class SecurityDoor(Door):
2     colour = 'grey'
3     locked = True
4
5     def open(self):
6         if self.locked:
7             return
8         Door.open(self)
```

part3/03.py

```
>>> sdoor = SecurityDoor(1, 'closed')
>>> sdoor.status
'closed'
>>> sdoor.open()
>>> sdoor.status
'closed'
>>>
```

Calling the parent implementation

```
1 class SecurityDoor(Door):
2     colour = 'grey'
3     locked = True
4
5     def open(self):
6         if self.locked:
7             return
8         Door.open(self)
```

part3/03.py

```
>>> sdoor = SecurityDoor(1, 'closed')
>>> sdoor.status
'closed'
>>> sdoor.open()
>>> sdoor.status
'closed'
>>> sdoor.locked = False
>>> sdoor.open()
>>> sdoor.status
'open'
>>>
```

Avoid strong coupling

```
1 class SecurityDoor(Door):
2     colour = 'grey'
3     locked = True
4
5     def open(self):
6         if self.locked:
7             return
8         super().open()
```

part3/04.py

super() to the rescue!

```
>>> sdoor = SecurityDoor(1, 'closed')
>>> sdoor.status
'closed'
>>> sdoor.open()
>>> sdoor.status
'closed'
>>> sdoor.locked = False
>>> sdoor.open()
>>> sdoor.status
'open'
>>>
```

Avoid strong coupling

```
1 class SecurityDoor(Door):
2     colour = 'grey'
3     locked = True
4
5     def open(self):
6         if self.locked:
7             return
8         super(SecurityDoor, self).open(self)
```

part3/04.py

Composition

```
1  class SecurityDoor:
2      colour = 'grey'
3      locked = True
4
5      def __init__(self, number, status):
6          self.door = Door(number, status)
7
8      def open(self):
9          if self.locked:
10             return
11             self.door.open()
12
13      def close(self):
14          self.door.close()
```

part3/05.py

Composition

```
1  class SecurityDoor:
2      colour = 'grey'
3      locked = True
4
5      def __init__(self, number, status):
6          self.door = Door(number, status)
7
8      def open(self):
9          if self.locked:
10             return
11             self.door.open()
12
13      def close(self):
14          self.door.close()
```

part3/05.py

No way to get the status

Python magic to the rescue

```
1  class SecurityDoor:
2      colour = 'grey'
3      locked = True
4
5      def __init__(self, number, status):
6          self.door = Door(number, status)
7
8      def open(self):
9          if self.locked:
10             return
11             self.door.open()
12
13      def close(self):
14          self.door.close()
15
16      def __getattr__(self, attr):
17          return getattr(self.door, attr)
```

part3/06.py

Python magic to the rescue

```
1  class SecurityDoor:
2      #colour = 'grey'
3      locked = True
4
5      def __init__(self, number, status):
6          self.door = Door(number, status)
7
8      def open(self):
9          if self.locked:
10             return
11             self.door.open()
12
13      #def close(self):
14      #    self.door.close()
15
16      def __getattr__(self, attr):
17          return getattr(self.door, attr)
```

part3/07.py

Composed inheritance?

```
1 class ComposedDoor:
2     def __init__(self, number, status):
3         self.door = Door(number, status)
4
5     def __getattr__(self, attr):
6         return getattr(self.door, attr)
```

part3/08.py

Composed inheritance?

```
1 class ComposedDoor:
2     def __init__(self, number, status):
3         self.door = Door(number, status)
4
5     def __getattr__(self, attr):
6         return getattr(self.door, attr)
```

part3/08.py

getattr() != __getattr__()



getattr()

```
>>> l = [1,2,3]
>>> dir(l)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__',
 '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append',
 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop',
 'remove', 'reverse', 'sort']
>>>
```

getattr()

```
>>> l = [1,2,3]
>>> dir(l)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__',
 '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append',
 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop',
 'remove', 'reverse', 'sort']
>>> l.append
<built-in method append of list object at 0xb70a2c2c>
>>> a = l.append
>>> a
<built-in method append of list object at 0xb70a2c2c>
>>>
```

getattr()

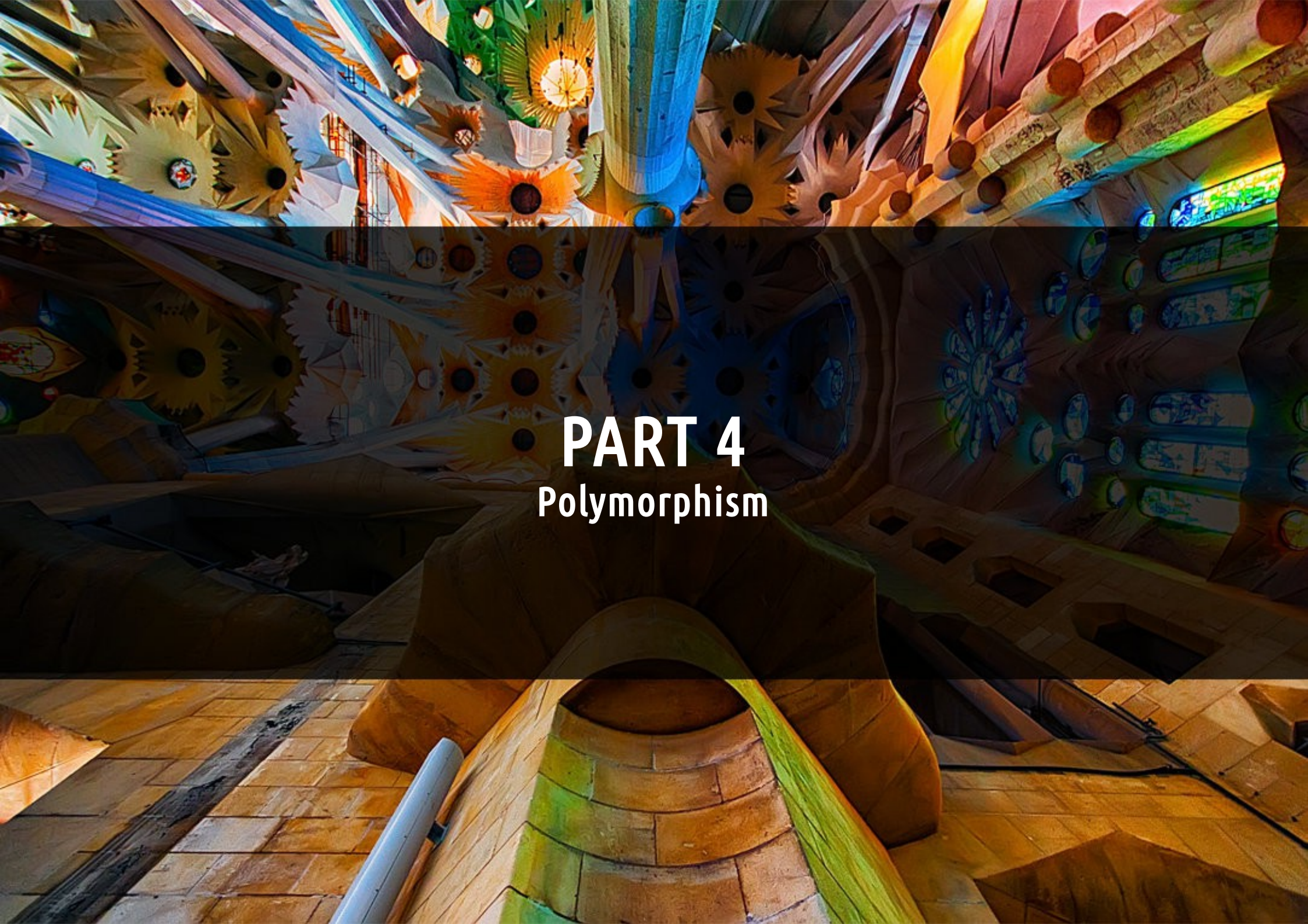
```
>>> l = [1,2,3]
>>> dir(l)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__',
 '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append',
 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop',
 'remove', 'reverse', 'sort']
>>> l.append
<built-in method append of list object at 0xb70a2c2c>
>>> a = l.append
>>> a
<built-in method append of list object at 0xb70a2c2c>
>>> b = getattr(l, 'append')
>>> b
<built-in method append of list object at 0xb70a2c2c>
>>>
```

getattr()

```
>>> l = [1,2,3]
>>> dir(l)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__',
 '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append',
 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop',
 'remove', 'reverse', 'sort']
>>> l.append
<built-in method append of list object at 0xb70a2c2c>
>>> a = l.append
>>> a
<built-in method append of list object at 0xb70a2c2c>
>>> b = getattr(l, 'append')
>>> b
<built-in method append of list object at 0xb70a2c2c>
>>> a == b
True
>>>
```

getattr()

```
>>> l = [1,2,3]
>>> dir(l)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__',
 '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append',
 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop',
 'remove', 'reverse', 'sort']
>>> l.append
<built-in method append of list object at 0xb70a2c2c>
>>> a = l.append
>>> a
<built-in method append of list object at 0xb70a2c2c>
>>> b = getattr(l, 'append')
>>> b
<built-in method append of list object at 0xb70a2c2c>
>>> a == b
True
>>> a is b
False
>>>
```



PART 4

Polymorphism

References

```
>>> a = 5  
>>> a  
5  
>>>
```

References

```
>>> a = 5
>>> a
5
>>> type(a)
<class 'int'>
>>>
```



This was not declared!

References

```
>>> a = 5
>>> a
5
>>> type(a)
<class 'int'>
>>> hex(id(a))
'0x83fe540'
>>>
```

References

```
>>> a = 5
>>> a
5
>>> type(a)
<class 'int'>
>>> hex(id(a))
'0x83fe540'
>>> a = 'five'
>>> a
'five'
>>>
```

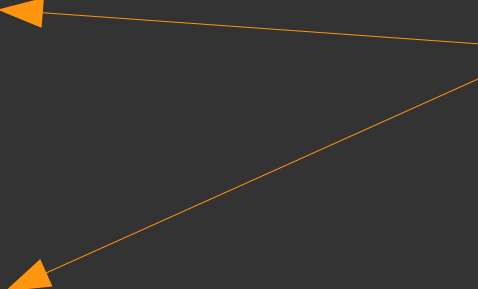
References

```
>>> a = 5
>>> a
5
>>> type(a)
<class 'int'>
>>> hex(id(a))
'0x83fe540'
>>> a = 'five'
>>> a
'five'
>>> type(a)
<class 'str'>
>>> hex(id(a))
'0xb70d6560'
>>>
```

References

```
>>> a = 5
>>> a
5
>>> type(a)
<class 'int'>
>>> hex(id(a))
'0x83fe540'
>>> a = 'five'
>>> a
'five'
>>> type(a)
<class 'str'>
>>> hex(id(a))
'0xb70d6560'
>>>
```

A strong type system: every variable has a type



References

```
>>> a = 5
>>> a
5
>>> type(a)
<class 'int'>
>>> hex(id(a))
'0x83fe540'
>>> a = 'five'
>>> a
'five'
>>> type(a)
<class 'str'>
>>> hex(id(a))
'0xb70d6560'
>>>
```

A strong type system: every variable has a type

A dynamic type system: the type changes with the content

Every variable is a reference

```
1 | def echo(a):  
2 |     return a
```

part4/01.py

Every variable is a reference

```
1 def echo(a):  
2     return a
```

part4/01.py

```
>>> echo(5)  
5  
>>> echo('five')  
'five'  
>>>
```

Every variable is a reference

Both are references

```
1 def echo(a):  
2     return a
```

part4/01.py

```
>>> echo(5)  
5  
>>> echo('five')  
'five'  
>>>
```

What is polymorphism?

```
>>> 5 + 6  
11  
>>> 5.5 + 6.6  
12.1  
>>>
```

What is polymorphism?

```
>>> 5 + 6
11
>>> 5.5 + 6.6
12.1
>>> "just a" + " string"
'just a string'
>>> [1,2,3] + [4,5,6]
[1, 2, 3, 4, 5, 6]
>>> (1,2,3) + (4,5,6)
(1, 2, 3, 4, 5, 6)
>>>
```

What is polymorphism?

```
>>> 5 + 6
11
>>> 5.5 + 6.6
12.1
>>> "just a" + " string"
'just a string'
>>> [1,2,3] + [4,5,6]
[1, 2, 3, 4, 5, 6]
>>> (1,2,3) + (4,5,6)
(1, 2, 3, 4, 5, 6)
>>> {'a':4, 'b':5} + {'c':7}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
>>>
```

What is polymorphism?

```
>>> l = [1, 2, 3]
>>> len(l)
3
>>>
```

What is polymorphism?

```
>>> l = [1, 2, 3]
>>> len(l)
3
>>> s = "Just a sentence"
>>> len(s)
15
>>> d = {'a': 1, 'b': 2}
>>> len(d)
2
>>>
```

What is polymorphism?

```
>>> l = [1, 2, 3]
>>> len(l)
3
>>> s = "Just a sentence"
>>> len(s)
15
>>> d = {'a': 1, 'b': 2}
>>> len(d)
2
>>> i = 5
>>> len(i)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no len()
>>>
```

What is polymorphism?

```
>>> l = [1, 2, 3]
>>> len(l)
3
>>> s = "Just a sentence"
>>> len(s)
15
>>> d = {'a': 1, 'b': 2}
>>> len(d)
2
>>> i = 5
>>> len(i)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no len()
>>> l.__len__()
3
>>> s.__len__()
15
>>> d.__len__()
2
>>> i.__len__()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'int' object has no attribute '__len__'
>>>
```

Polymorphism is based on delegation

```
>>> [1,2,3].__add__([4,5,6])  
[1, 2, 3, 4, 5, 6]  
>>>
```

Polymorphism is based on delegation

```
>>> [1,2,3].__add__([4,5,6])
[1, 2, 3, 4, 5, 6]
>>> dir([1,2,3])
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
 '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear',
 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
 'reverse', 'sort']
>>>
```

Polymorphism is based on delegation

```
>>> [1,2,3].__add__([4,5,6])
[1, 2, 3, 4, 5, 6]
>>> dir([1,2,3])
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
 '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__',
 '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__',
 '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear',
 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
 'reverse', 'sort']
>>> 1 in [1,2,3]
True
>>> [1,2,3].__contains__(1)
True
>>> 6 in [1,2,3]
False
>>> [1,2,3].__contains__(6)
False
>>>
```

Polymorphism is based on delegation

```
1 | def sum(a, b):  
2 |     return a + b
```

part4/02.py

```
>>> sum(5,6)  
11  
>>> sum("Being ", "polymorphic")  
'Being polymorphic'  
>>> sum([1,2,3], [4,5,6])  
[1, 2, 3, 4, 5, 6]  
>>>
```

Polymorphism is based on delegation

```
1 | def sum(a, b):  
2 |     return a + b
```

part4/02.py

```
>>> sum(5,6)  
11  
>>> sum("Being ", "polymorphic")  
'Being polymorphic'  
>>> sum([1,2,3], [4,5,6])  
[1, 2, 3, 4, 5, 6]  
>>> sum([1,2,3], 8)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
  File "<stdin>", line 2, in sum  
TypeError: can only concatenate list (not "int") to list  
>>>
```

Polymorphism in action

```
1  class Room:
2      def __init__(self, door):
3          self.door = door
4
5      def open(self):
6          self.door.open()
7
8      def close(self):
9          self.door.close()
10
11     def is_open(self):
12         return self.door.is_open()
```

part4/03.py

Polymorphism in action

part4/03.py

```
1  class Door:
2      def __init__(self):
3          self.status = "closed"
4
5      def open(self):
6          self.status = "open"
7
8      def close(self):
9          self.status = "closed"
10
11     def is_open(self):
12         return self.status == "open"
13
14
15     class BooleanDoor:
16         def __init__(self):
17             self.status = True
18
19         def open(self):
20             self.status = True
21
22         def close(self):
23             self.status = False
24
25         def is_open(self):
26             return self.status
```

Polymorphism in action

```
>>> door = Door()  
>>> bool_door = BooleanDoor()  
>>> room = Room(door)  
>>> bool_room = Room(bool_door)  
>>>
```

Polymorphism in action

```
>>> door = Door()
>>> bool_door = BooleanDoor()
>>> room = Room(door)
>>> bool_room = Room(bool_door)
>>>
>>> room.open()
>>> room.is_open()
True
>>> room.close()
>>> room.is_open()
False
>>>
```

Polymorphism in action

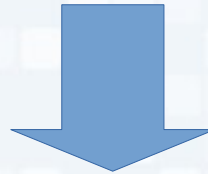
```
>>> door = Door()
>>> bool_door = BooleanDoor()
>>> room = Room(door)
>>> bool_room = Room(bool_door)
>>>
>>> room.open()
>>> room.is_open()
True
>>> room.close()
>>> room.is_open()
False
>>>
>>> bool_room.open()
>>> bool_room.is_open()
True
>>> bool_room.close()
>>> bool_room.is_open()
False
>>>
```

“Ask for permission” style

```
1  if hasattr(someobj, 'open'):  
2      [...]  
3  else:  
4      [...]
```

“Ask for forgiveness” style

```
1  if hasattr(someobj, 'open'):
2      [...]
3  else:
4      [...]
```

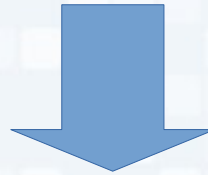


```
1  try:
2      someobj.open()
3      [...]
4  except AttributeError:
5      [...]
```

“Ask for forgiveness” style

```
1  if hasattr(someobj, 'open'):  
2      [...]  
3  else:  
4      [...]
```

It 'has' the attribute



```
1  try:  
2      someobj.open()  
3      [...]  
4  except AttributeError:  
5      [...]
```

It behaves like it has the attribute



PART 5

Metaclasses

Everything is an object (again)

```
>>> a = 5  
>>>
```

Everything is an object (again)

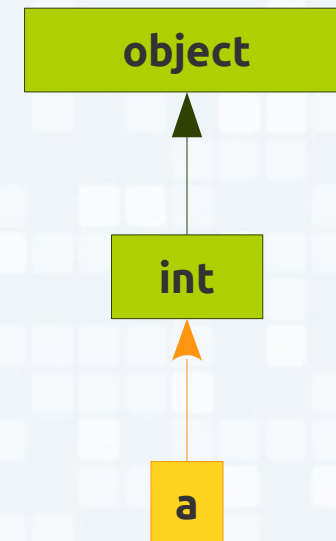
```
>>> a = 5
>>> type(a)
<class 'int'>
>>> a.__class__
<class 'int'>
>>>
```

Everything is an object (again)

```
>>> a = 5
>>> type(a)
<class 'int'>
>>> a.__class__
<class 'int'>
>>> a.__class__.__bases__
(<class 'object'>,)
>>> object.__bases__
()
>>>
```

Everything is an object (again)

```
>>> a = 5
>>> type(a)
<class 'int'>
>>> a.__class__
<class 'int'>
>>> a.__class__.__bases__
(<class 'object'>,)
>>> object.__bases__
()
>>>
```

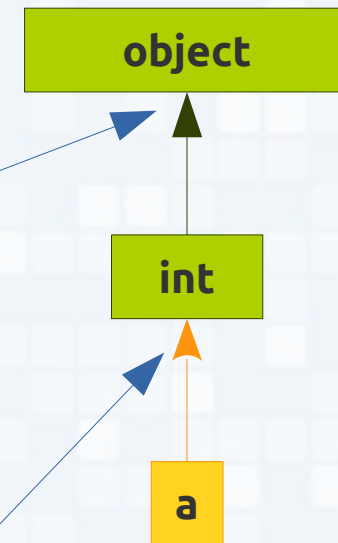


Everything is an object (again)

```
>>> a = 5
>>> type(a)
<class 'int'>
>>> a.__class__
<class 'int'>
>>> a.__class__.__bases__
(<class 'object'>,)
>>> object.__bases__
()
>>>
```

'int' inherits from (its base is) 'object'

'a' is an instance of (its type is) 'int'



Where classes come from

```
>>> type(a)
<class 'int'>
>>>
```

Where classes come from

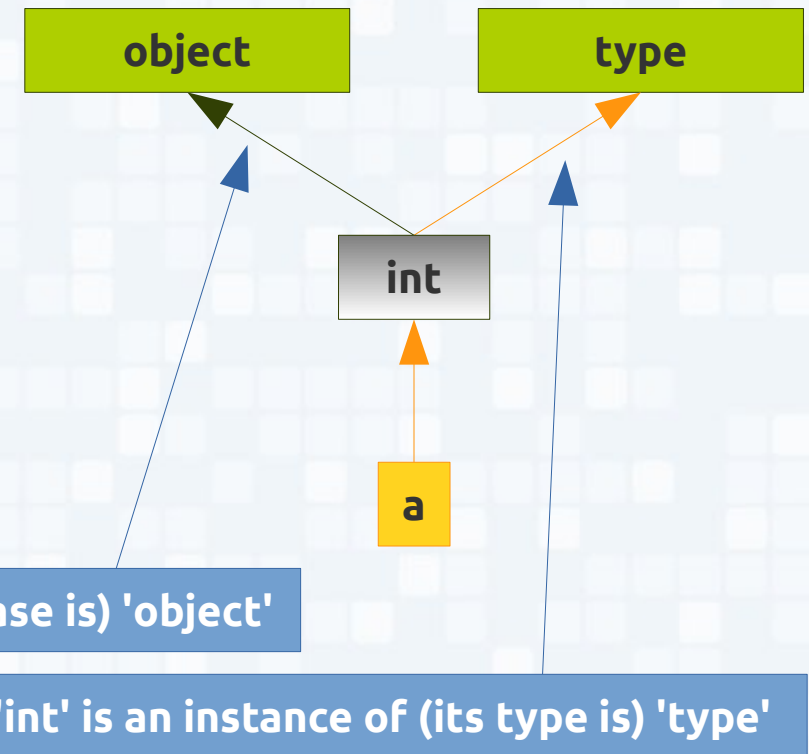
```
>>> type(a)
<class 'int'>
>>> type(int)
<class 'type'>
>>>
```

Where classes come from

```
>>> type(a)
<class 'int'>
>>> type(int)
<class 'type'>
>>> type(float)
<class 'type'>
>>> type(dict)
<class 'type'>
>>>
```

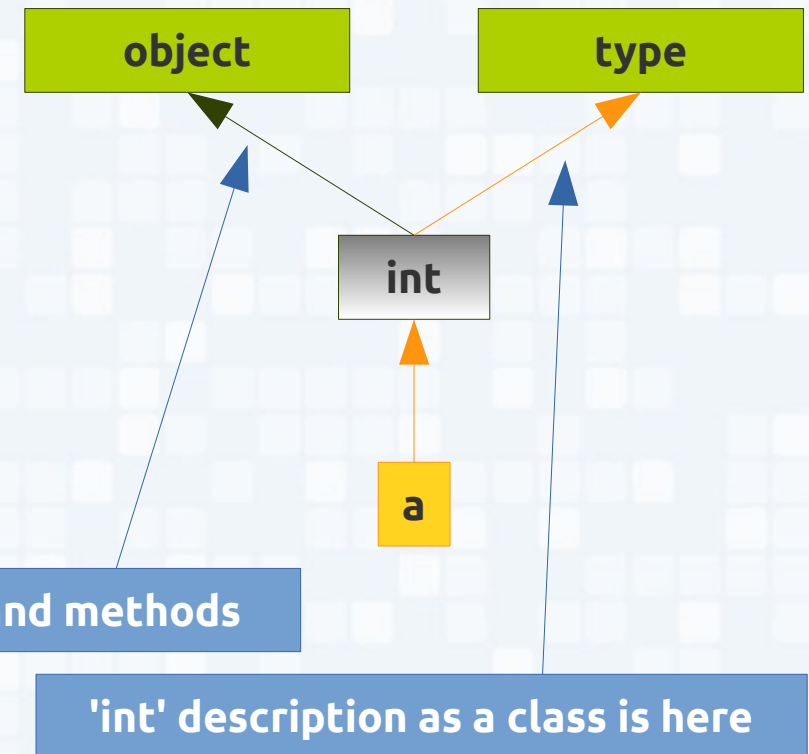
Everything is an object (again)

```
>>> type(a)
<class 'int'>
>>> type(int)
<class 'type'>
>>> type(float)
<class 'type'>
>>> type(dict)
<class 'type'>
>>>
```



Everything is an object (again)

```
>>> type(a)
<class 'int'>
>>> type(int)
<class 'type'>
>>> type(float)
<class 'type'>
>>> type(dict)
<class 'type'>
>>>
```

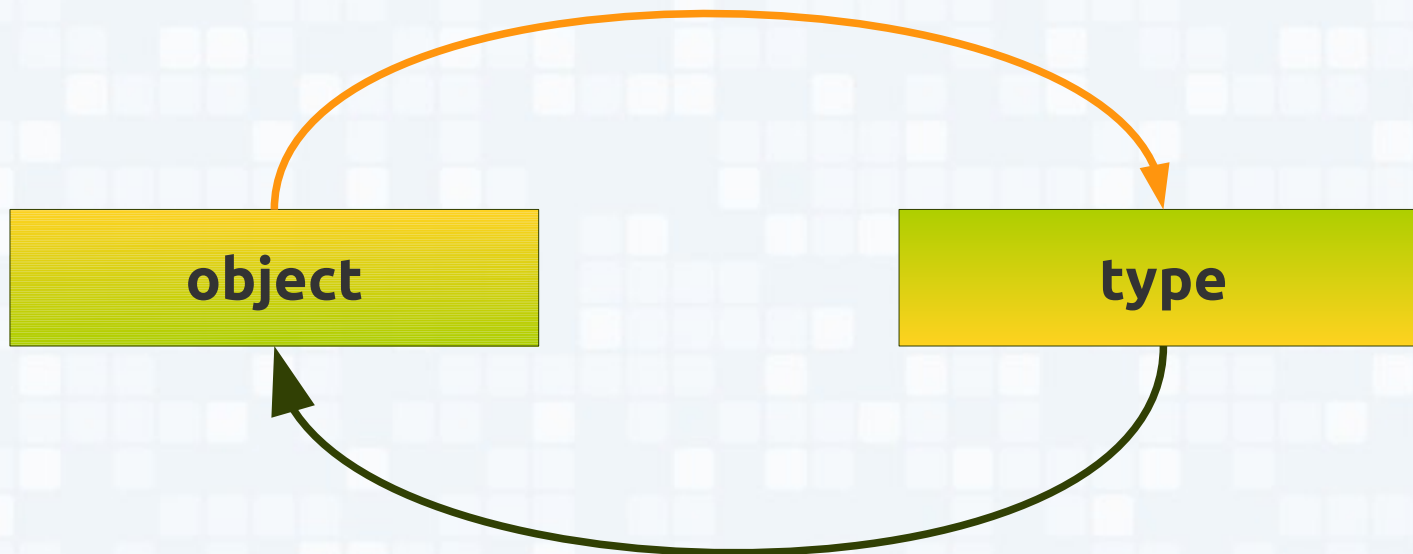


'int' finds here attributes and methods

'int' description as a class is here

Food for thought

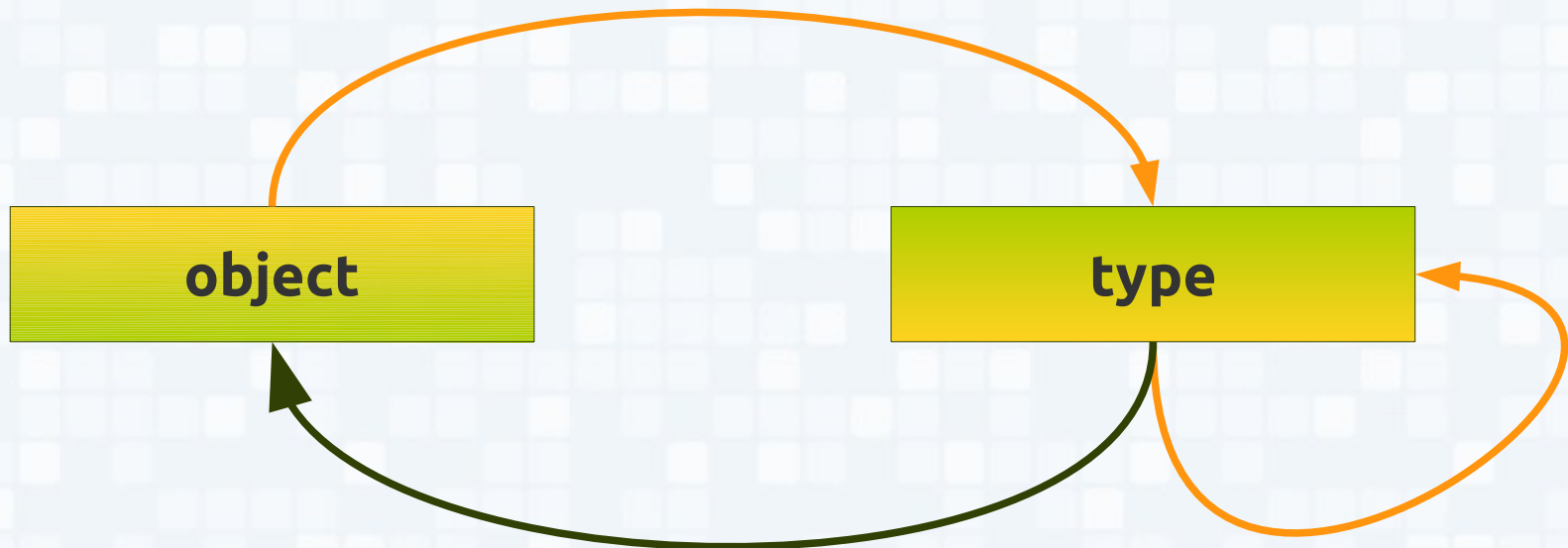
```
>>> type(object)
<class 'type'>
>>> type.__bases__
(<class 'object'>,)
>>>
```



Fatality



```
>>> type(type)  
<class 'type'>
```



Fatality

```
>>> type(type)  
<type 'type'>
```

Metaclasses

```
1 class MyType(type):  
2     pass  
3  
4 class MySpecialClass(metaclass=MyType):  
5     pass
```

part5/01.py

```
1 class MyType(type):  
2     pass  
3  
4 class MySpecialClass(object):  
5     __metaclass__ = MyType
```

part5/01.py

Metaclasses

```
1 class MyType(type):  
2     pass  
3  
4 class MySpecialClass(metaclass=MyType):  
5     pass
```

part5/01.py

```
>>> msp = MySpecialClass()  
>>>
```

Metaclasses

```
1 class MyType(type):  
2     pass  
3  
4 class MySpecialClass(metaclass=MyType):  
5     pass
```

part5/01.py

```
>>> msp = MySpecialClass()  
>>> type(msp)  
<class '__main__.MySpecialClass'>  
>>>
```

Metaclasses

```
1 class MyType(type):  
2     pass  
3  
4 class MySpecialClass(metaclass=MyType):  
5     pass
```

part5/01.py

```
>>> msp = MySpecialClass()  
>>> type(msp)  
<class '__main__.MySpecialClass'>  
>>> type(MySpecialClass)  
<class '__main__.MyType'>  
>>>
```

Metaclasses

```
1 class MyType(type):  
2     pass  
3  
4 class MySpecialClass(metaclass=MyType):  
5     pass
```

part5/01.py

```
>>> msp = MySpecialClass()  
>>> type(msp)  
<class '__main__.MySpecialClass'>  
>>> type(MySpecialClass)  
<class '__main__.MyType'>  
>>> MySpecialClass.__bases__  
(<class 'object'>,)  
>>>
```

Metaclasses in action: singleton

```
1 class Singleton(type):
2     instance = None
3     def __call__(cls, *args, **kw):
4         if not cls.instance:
5             cls.instance = super(Singleton, cls).
6                 __call__(*args, **kw)
7         return cls.instance
8
9 class ASingleton(metaclass=Singleton):
10     pass
```

part5/02.py

Metaclasses in action: singleton

```
1 class Singleton(type):
2     instance = None
3     def __call__(cls, *args, **kw):
4         if not cls.instance:
5             cls.instance = super(Singleton, cls).
6                 __call__(*args, **kw)
7         return cls.instance
8
9 class ASingleton(metaclass=Singleton):
10     pass
```

part5/02.py

```
>>> a = ASingleton()
>>> b = ASingleton()
>>>
```

Metaclasses in action: singleton

```
1 class Singleton(type):
2     instance = None
3     def __call__(cls, *args, **kw):
4         if not cls.instance:
5             cls.instance = super(Singleton, cls).
6                 __call__(*args, **kw)
7         return cls.instance
8
9 class ASingleton(metaclass=Singleton):
10     pass
```

part5/02.py

```
>>> a = ASingleton()
>>> b = ASingleton()
>>> a is b
True
>>>
```

Metaclasses in action: singleton

```
1 class Singleton(type):
2     instance = None
3     def __call__(cls, *args, **kw):
4         if not cls.instance:
5             cls.instance = super(Singleton, cls).
6                 __call__(*args, **kw)
7         return cls.instance
8
9 class ASingleton(metaclass=Singleton):
10     pass
```

part5/02.py

```
>>> a = ASingleton()
>>> b = ASingleton()
>>> a is b
True
>>> hex(id(a))
'0xb68030ec'
>>> hex(id(b))
'0xb68030ec'
>>>
```



SOME LINKS

The original posts

<http://lgiordani.com/blog/categories/python3/>

Some links about Python OOP

<http://goo.gl/UBdJDT>

THANK YOU!

Any comment, correction or contribution is warmly welcome.

Feel free to contact me.



The Digital Cat
<http://lgiordani.com>



https://twitter.com/tw_lgiordani



<https://github.com/lgiordani>



<https://plus.google.com/u/LeonardoGiordani>