Мета-классы

In [1]:
```
class Class:
    ...
```

In [2]:
```
obj = Class()
```

In [3]:
```
type(obj)
```

Out[3]: __main__.Class

In [4]:
```
type(Class)
```

Out[4]: type

In [5]:
```
type(type)
```

Out[5]: type

In [6]:
```
issubclass(Class, type)
```

Out[6]: False

In [7]:
```
issubclass(Class, object)
```

Out[7]: True

In [8]:
```
def dummy_factory():
    class Class:
        pass

    return Class


Dummy = dummy_factory()

print(Dummy() is Dummy())
```

```
False
```

In [9]:
```
NewClass = type('NewClass', (), {})

print(NewClass)
print(NewClass())
```

```
<class '__main__.NewClass'>
<__main__.NewClass object at 0x110cd7438>
```

In [10]:
```
class Meta(type):
    def __new__(cls, name, parents, attrs):
        print('Creating {}'.format(name))

        if 'class_id' not in attrs:
            attrs['class_id'] = name.lower()

        return super().__new__(cls, name, parents, attrs)


class A(metaclass=Meta):
    pass
```

```
Creating A
```

In [11]:
```
print('A.class_id: "{}"'.format(A.class_id))
```

```
A.class_id: "a"
```

In [12]:
```
class Meta(type):
    def __init__(cls, name, bases, attrs):
        print('Initializing — {}'.format(name))

        if not hasattr(cls, 'registry'):
            cls.registry = {}
        else:
            cls.registry[name.lower()] = cls

        super().__init__(name, bases, attrs)


class Base(metaclass=Meta): pass

class A(Base): pass

class B(Base): pass
```

```
Initializing — Base
Initializing — A
Initializing — B
```

In [13]:
```
print(Base.registry)
print(Base.__subclasses__())
```

```
{'a': <class '__main__.A'>, 'b': <class '__main__.B'>}
[<class '__main__.A'>, <class '__main__.B'>]
```

Абстрактные методы

In [14]:
```
from abc import ABCMeta, abstractmethod

class Sender(metaclass=ABCMeta):
    @abstractmethod
    def send(self):
        """Do something"""
```

In [15]:
```
class Child(Sender): pass

Child()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-5e10f1ccf1fd> in <module>()
      1 class Child(Sender): pass
      2
----> 3 Child()

TypeError: Can't instantiate abstract class Child with abstract methods send
```

In [16]:
```
class Child(Sender):
    def send(self):
        print('Sending')


Child()
```

Out[16]: <__main__.Child at 0x110cfa860>

In [17]:
```
class PythonWay:

    def send(self):
        raise NotImplementedError
```