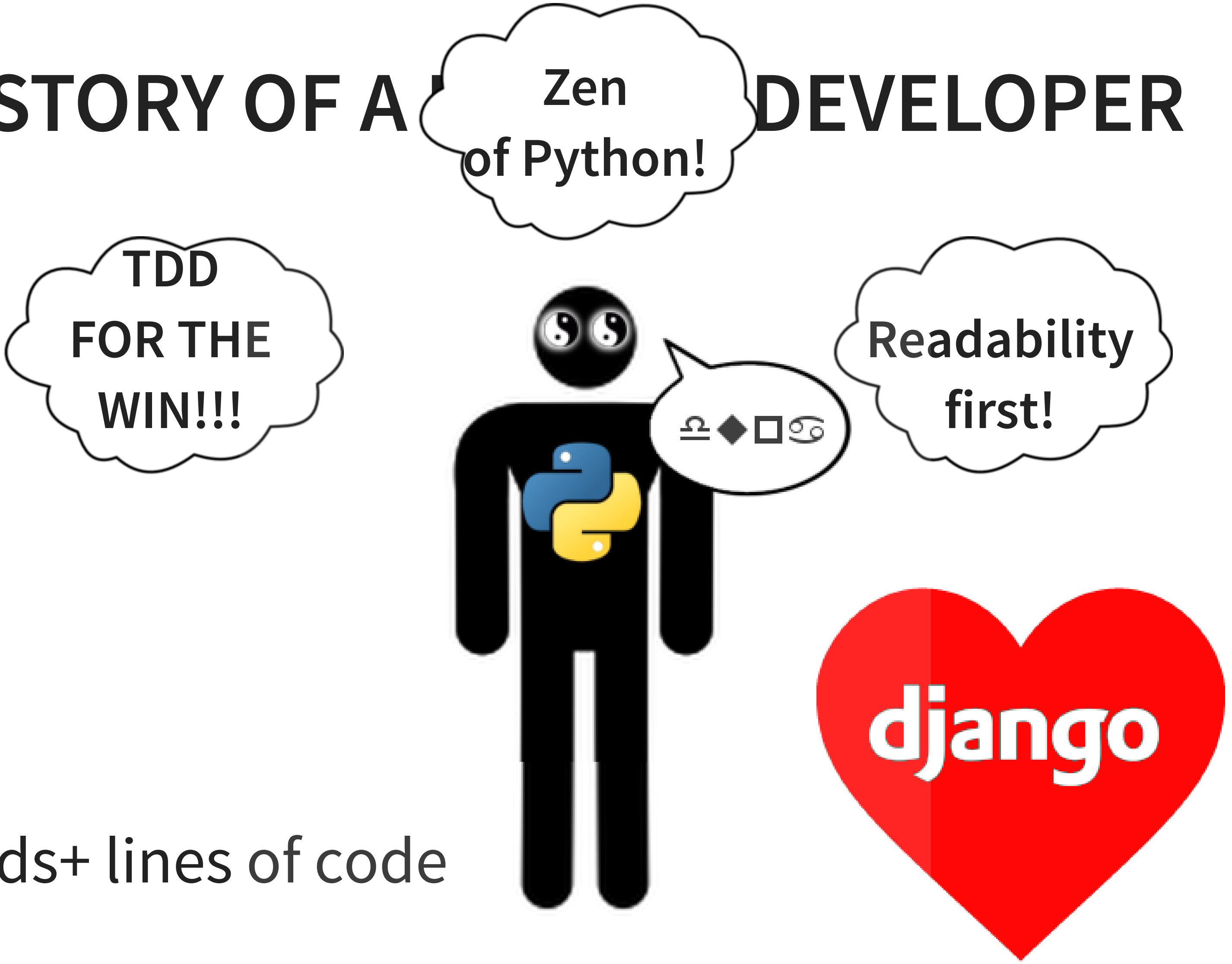
WHY YOU DON'T NEED DESIGN PATTERNS IN PYTHON?

EuroPython 2017

EVERYTHING STARTS WITH A STORY...

STORY OF A DEVELOPER



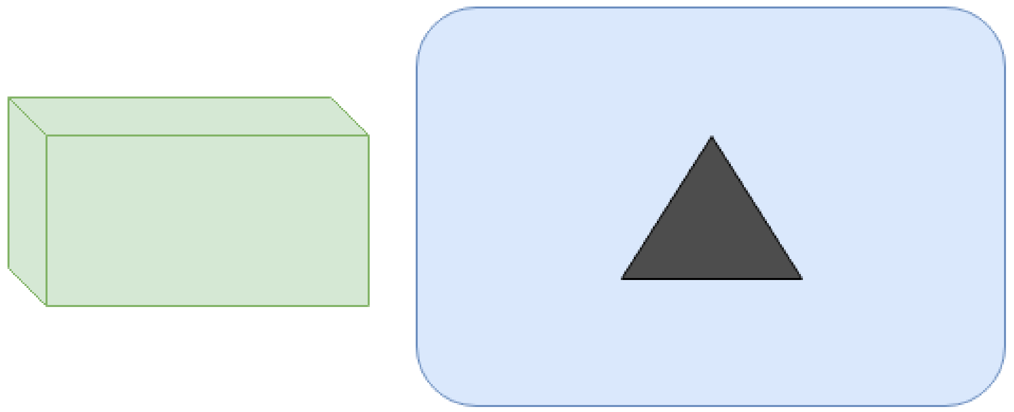
Thousands+ lines of code

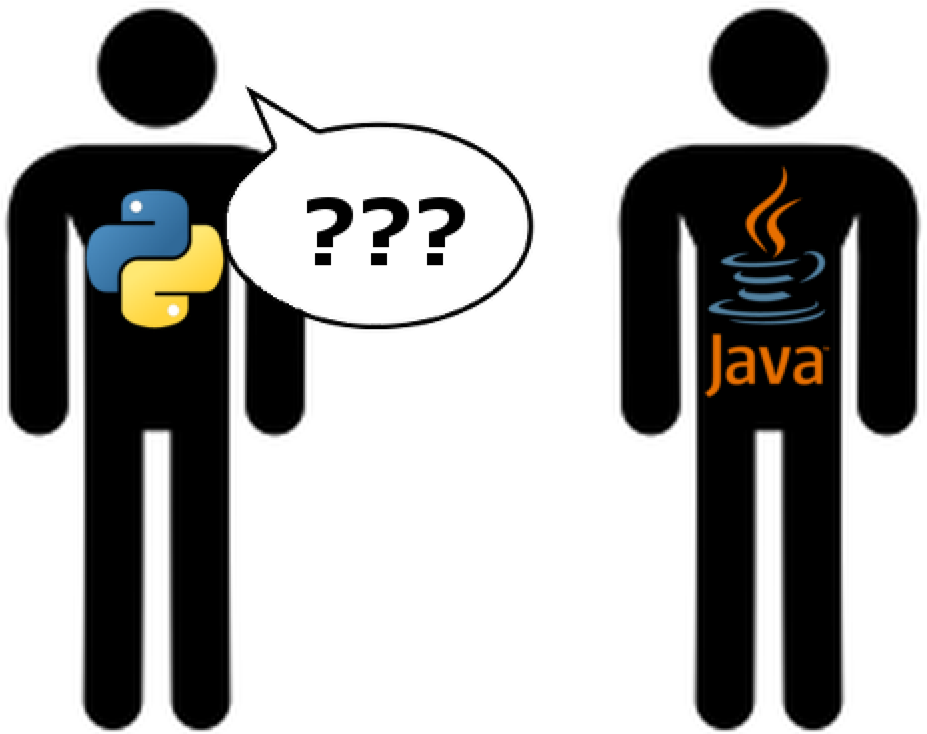
and then, one project changed everything

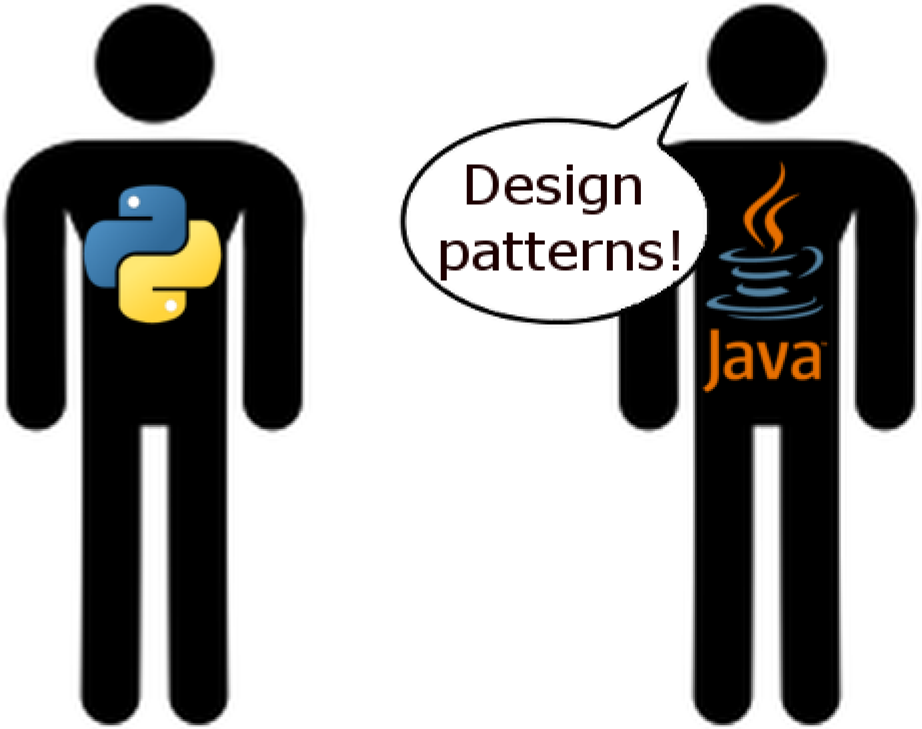
Weight of a project outside framework

FRAMEWORKS ARE SETS OF BUILDING BLOCKS

THEY WORK FINE FOR A SPECIFIC RANGE OF PROBLEMS




???
Java


Design patterns!
Java

Design pattern

...general reusable solution to a commonly occurring problem...

...formalized best practices...

SINGLETON

- Only one!
- clear way to get an instance

SINGLETON - __NEW__

```python
class Singleton:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super().__new__(cls, *args, **kwargs)

        return cls._instance
```

SINGLETON - __NEW__

```python
class Singleton:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super().__new__(cls, *args, **kwargs)

        return cls._instance
```

```python
one_instance = Singleton()
another_instance = Singleton()

one_instance is another_instance  # True
```

SINGLETON - @CLASSMETHOD

```
class Singleton:
    _instance = None

    @classmethod
    def get_instance(cls):
        if not cls._instance:
            cls._instance = cls()

        return cls._instance
```

```
one_instance = Singleton.get_instance()
another_instance = Singleton()

one_instance is another_instance  # False
```

THERE IS A SIMPLER WAY...

```
class Singleton:
    pass

singleton = Singleton()

# another modules
from my_code import singleton
```

SINGLETONS IN PYTHON?

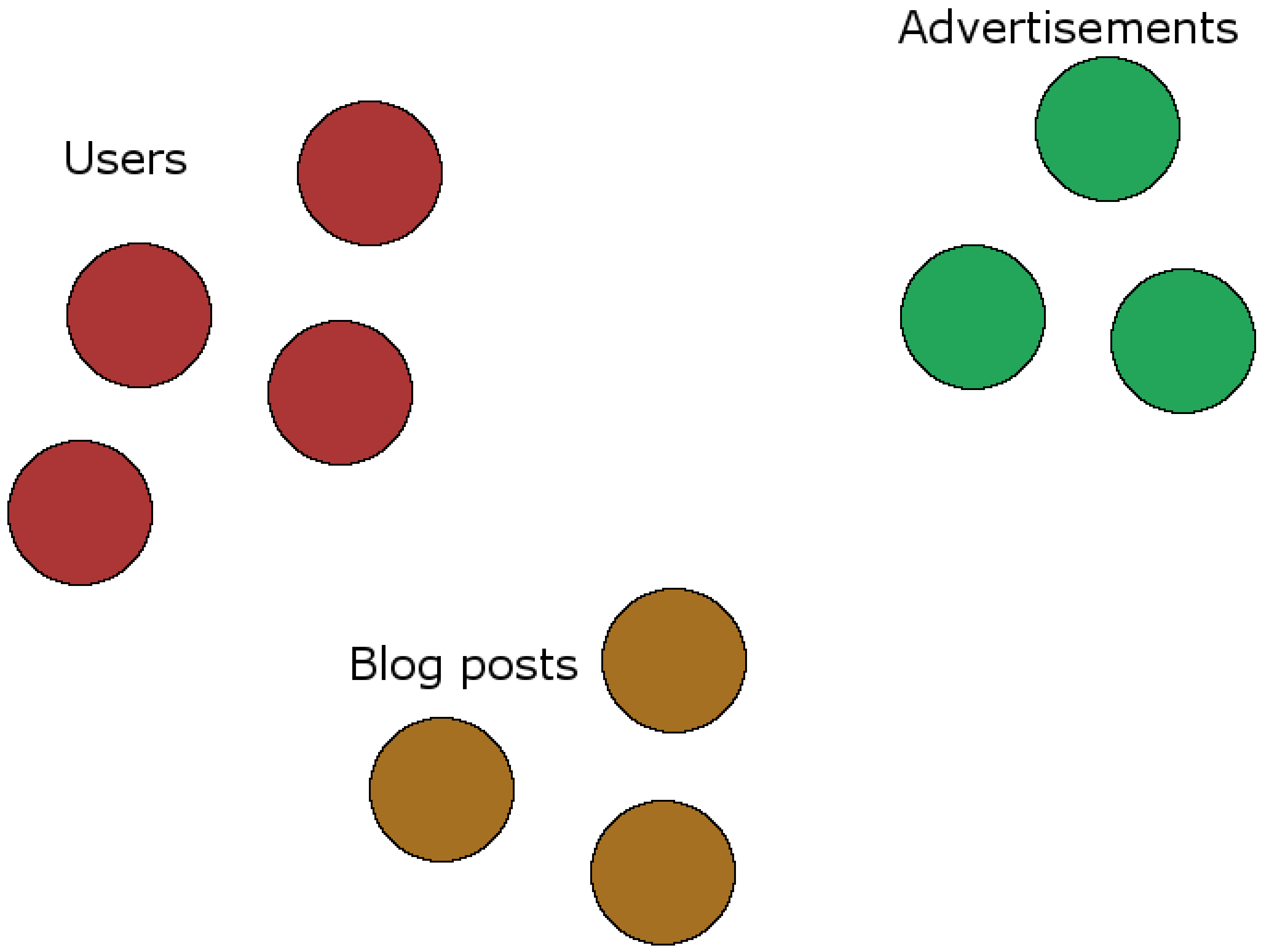
MODULES!

- Exactly one instance living in sys.modules
- Get an instance easily `import module`
- Recreate using `importlib.reload(module) #Py3`

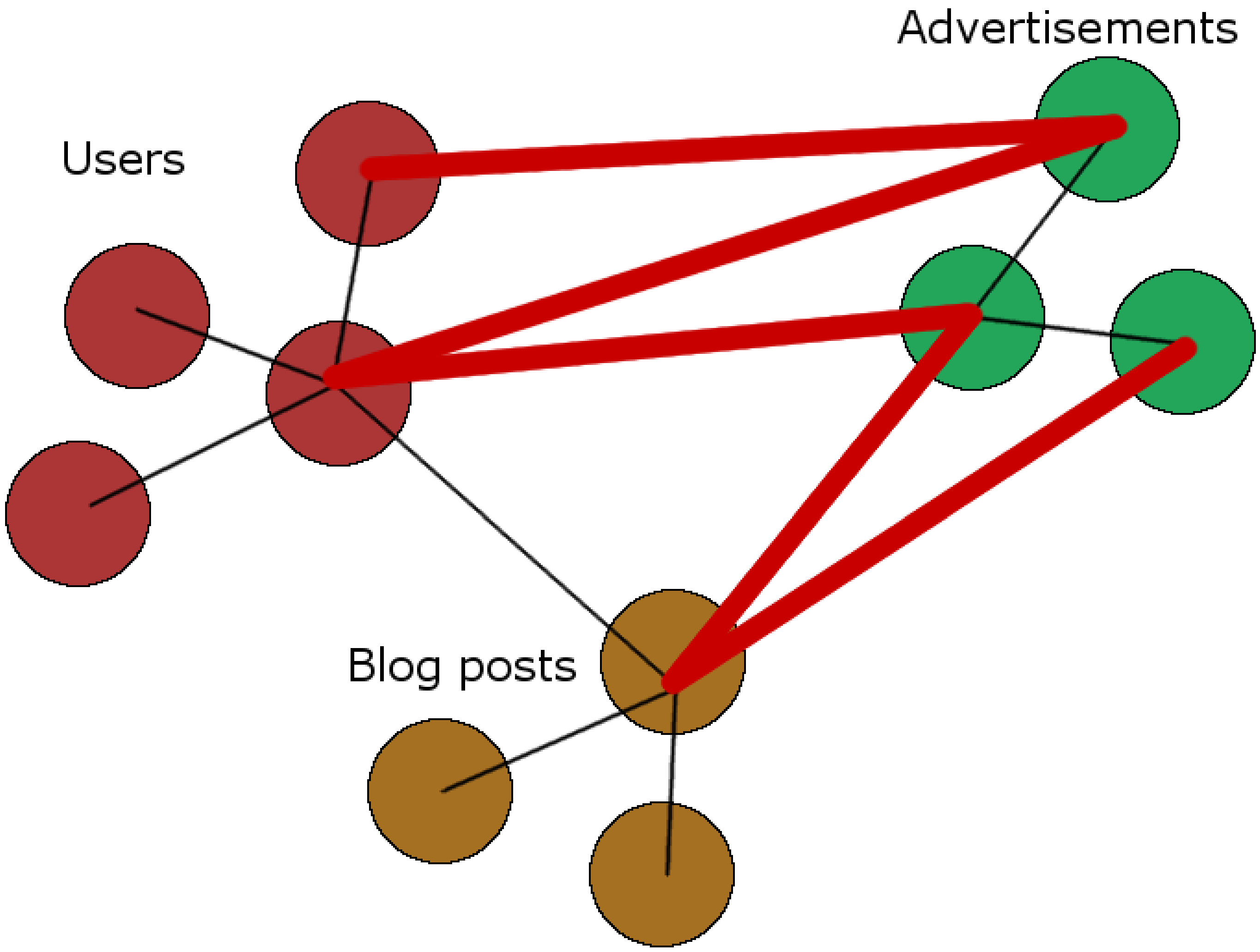
SINGLETON - CONCLUSION

Using a module may be better than creating class

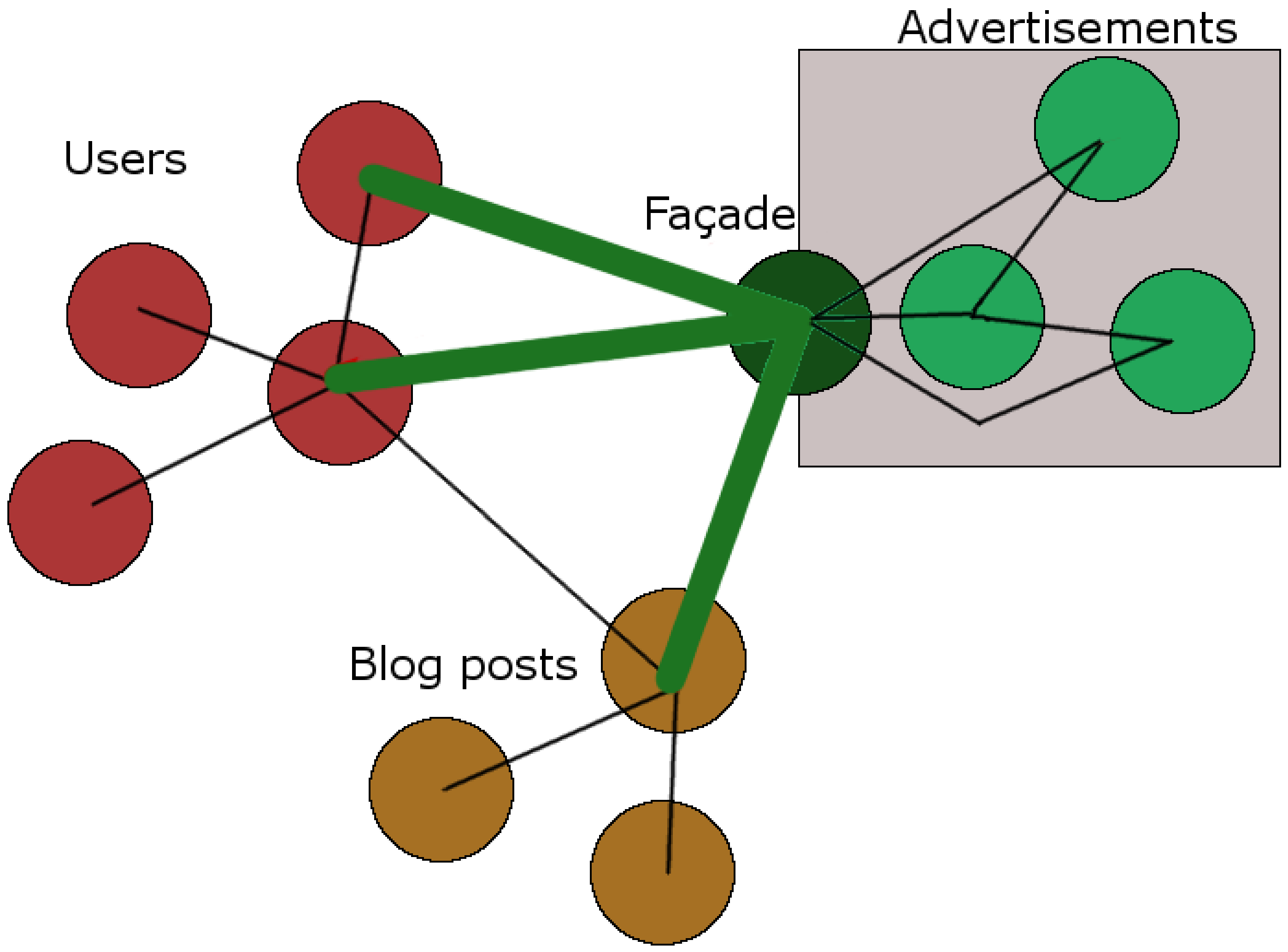
SPEAKING OF MODULES - FAÇADE



SPEAKING OF MODULES - FAÇADE



SPEAKING OF MODULES - FAÇADE



FAÇADE - CLASS

```python
class AdvertisementsFacade:

    @classmethod
    def get_advert_for_single_post(post):
        pass

    @classmethod
    def get_adverts_for_main_page(count):
        pass
```

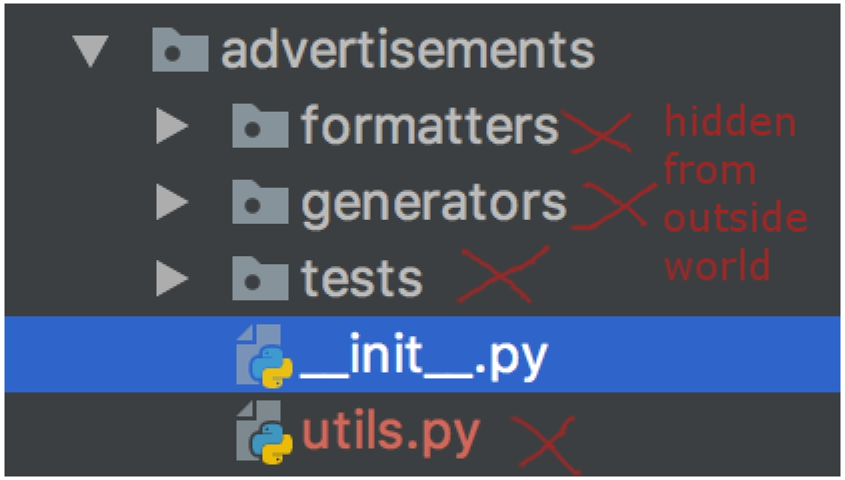
FAÇADE - MODULE

```python
def get_advert_for_single_post(post):
    pass

def get_adverts_for_main_page(count):
    pass

# in another module
import advertisements

adverts = advertisements.get_adverts_for_main_page(count=3)
```



FAÇADE - CONCLUSION

Helpful to organize code, no need for a class

COMMAND

Object oriented callback

```
class Client:
    def foo(self):
        some_obj = SomeClass()
        command = Command(some_obj)
        self.menu_item.set_command(command)

        # later in menu_item code
        self.command.execute()  # menu_item doesn't know anything
```

COMMAND - CLASS

```
class Command:
    ...

    def execute(discount_rate):
        self.object.notify_users_about_discount(discount_rate)
```

COMMAND - FUNCTION

```python
def command(discount_rate):
    some_obj.notify_users_about_discount()
```

or even simpler using standard library's goodies:

```python
import functools

command = functools.partial(
    some_obj.notify_users_about_discount, discount_rate=0.5
)

command()
# equals to
some_obj.notify_users_about_discount(discount_rate=0.5)
```

COMMAND - CONCLUSION

With classes makes a little sense in Python

VISITOR PATTERN

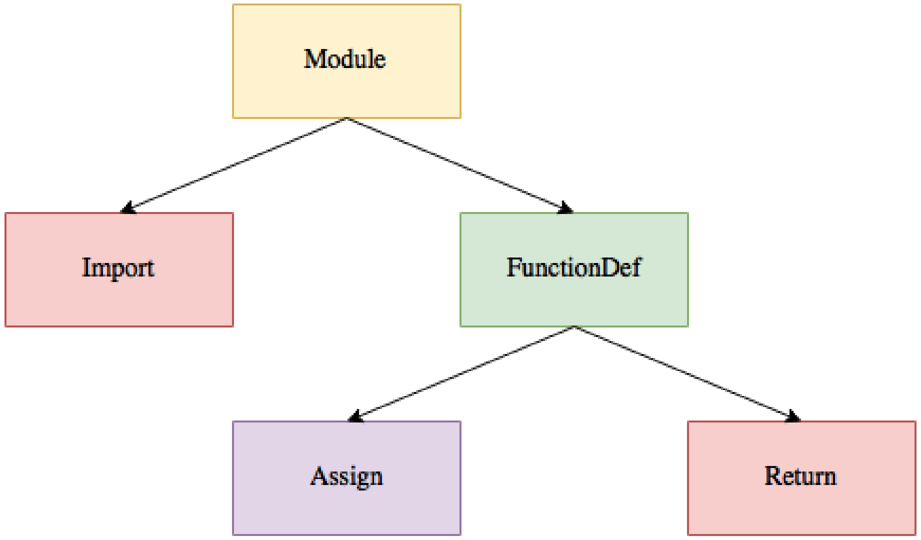
Let's say we have a complicated, nested data structure to parse.

VISITOR - EXAMPLE

ASTs

```
import time

def ten_seconds_ago():
    now = time.time()
    return now - 10
```



VISITOR IMPLEMENTATION - JAVA

```
public class ASTVisitor {
    public void visit(Import import) {}

    public void visit(FunctionDef functionDef) {}

    public void visit(Assign assign) {}
}
```

PYTHON *NAIVE* IMPLEMENTATION

```python
class ASTVisitor:
    def visit(node):
        if type(node) == Import:
            self.visit_import()
        elif type(node) == FunctionDef:
            self.visit_functiondef()
        elif type(node) == Assign:
            self.visit_assign()
        else:
            raise AttributeError
```

PYTHON *BETTER* IMPLEMENTATION

```python
class ASTVisitor:
    def visit(node):
        normalized_type_name = type(node).__name__.lower()
        # 'assign'
        method_name = '_visit_' + normalized_type_name
        # '_visit_assign'

        method = getattr(self, method_name)
        method()
```

This example comes from *Python Cookbook 3rd edition*

PYTHON WITH @SINGLEDISPATCH

```python
from functools import singledispatch

@singledispatch
def visit(node):
    type_name = type(node).__name__
    raise AttributeError(f'No handler found for {type_name}')
```

```python
from ast_nodes import Assign, FunctionDef

@visit.register(Assign)
def visit(node):
    pass

@visit.register(FunctionDef)
def visit(node):
    pass
```

Can't be used in classes :(

DECORATOR

Decorator pattern != @decorator functions in Python

- Extend behaviour of a given object
- Possible during runtime
- Multiple times, with different decorators and order

DECORATOR - EXAMPLE

```
assert hasattr(original_object, 'anyattr')

decorated_object = Decorator(original_object)
assert hasattr(decorated_object, 'anyattr')

assert type(original_object) != type(decorated_object)
```

Different types, but hey - duck typing

DECORATOR - EXAMPLE 2

```
class OriginalClass:
    def get_text(self):
        pass

    def get_number(self):
        pass


class Decorator:
    def __init__(self, decorated_obj):
        self.decorated_obj = decorated_obj

    def get_text(self):
        return f'<b>{self.decorated_obj.get_text()}</b>'

    def get_number(self):
        return self.decorated_obj.get_number()
```

We have to reimplement all methods

WAIT A SEC... WHAT HAPPENS IF I REQUEST AN ATTRIBUTE?

```
some_object.some_method
#<bound method SomeClass.some_method of <SomeClass object at 0x0>>
```

Methods are just attributes

Firstly, Python calls a special `__getattribute__`

If no attribute was found, `__getattr__` is called. By default it just throws an exception

WHAT HAPPENS IF I REQUEST AN ATTRIBUTE? - __DICT__

```python
class ClsVsObject:
    some_attr = 1

    def __init__(self):
        self.some_attr = 2

example = ClsVsObject()
example.__dict__['some_attr']  # 2, == example.some_attr
example.__class__.__dict__['some_attr'] # 1 == ClsVsObject.some_attr

example.some_attr  # 2
ClsVsObject.some_attr  # 1
```

DECORATOR - IMPLEMENTATION

```
class Decorator:
    def __init__(self, decorated_obj):
        self.decorated_obj = decorated_obj

    def get_text(self):
        return f'{self.decorated_obj.get_text()}'


    def __getattr__(self, attr_name):
        return getattr(self.decorated_obj, attr_name)
```

To get a full compatiblity, add other methods: __setattr__, __delattr__ and so on.

SUMMARY

Python is a very flexible tool

IS MAGIC WORTH THE EFFORT?

SUMMARY (THIS TIME FOR REAL)

- know well your tools (Python!)
- get inspiration from other languages and communities
- know a business domain of your project

SEBASTIAN BUCZYŃSKI

Working for STX Next
Blogging under breadcrumbscollector.tech
Twitter: EnforcerPL

STX NEXT