Descriptors story

talented developers | flexible teams | agile experts

Adrian Dziubek - EuroPython - 2016-07-18
About me

Adrian Dziubek
Senior Python developer at STX Next in Wrocław,

Creating web applications using Python and Javascript.

Also
Cyclist, photographer, Ultimate Frisbee player.
A story about tree structures and accidental rediscovery of descriptors.

Knowledge vs usage

The plan
- Introduction to project,
- Fighting the lost war,
- Starting anew,
- Lessons learned.

Warning: legacy code ahead
The project

Positioning system

Input
Satellite provided position + radio visible

Storage
A map of radio stations with metadata

Output
Approximate position based on radio signal

Why?
Assisted GPS, positioning inside the buildings
The test project

Blackbox test tool in Python

Input testing
Binary data against Database requests

Output testing
Database setup + binary request against Binary response
Problems:

- star imports, race conditions,
- repetition, C++ styling,
- no use of introspection.
The data

**Problems:**
- attribute access chains,
- singleton pattern,
- can’t reuse locals.

```python
# top/protocol/Data.py DATA DEFINITION
class ABCData(object):
    @staticmethod
    def get( index = 0 ):
        abc_element = ABCElement()
        abc_element.setVersion(5, 1)
        if index == 0:
            cd_element = CDData.get(1)
            cd_element.sub_elem = SubElement()
            cd_element.sub_elem.infra_elem1(0x123)
            cd_element.sub_elem.infra_elem2(0x234)
            fg_element = FGElemeent()
            abc_element.children.append(cd_element)
            abc_element.children.append(fg_element)
        # ...        return abc_element

# top/scenario/ProtocolCase.py TEST CASE
class ProtocolCase(Runner):
    def abc( self ):
        abc_element = ABCData.get(0)
        cd_element = CDElement()
        fg_element = FGElemeent()
        cd_element.sub_elem.infra_elem1(1)
        cd_element.sub_elem.infra_elem1(0x123)
        cd_element.sub_elem.infra_elem2(0x234)
        abc_element.children.append([cd_element, fg_element])
    # test run below
```
What is the goal?

Easy to use trees for testers:
- data definition,
- difference reporting,
- server configuration.

Optimize data definition looks a bit like assignment.

```python
# top/protocol/Data.py DATA DEFINITION
children1 = [
    CDElement(
        sub_elem=SubElem(
            infra_elem1=0x123,
            infra_elem2=0x234
        ),
        FGEElement()
    ),
],
abc_element1 = ABCElement1(
    type=ABCType(ABCType.A),
    version=Version(5, 1),
    children=children1,
)

# top/scenario/ProtocolCase.py TEST CASE
from top.protocol import Data, Data2
abc = copy.deepcopy(Data.abc_element1)
abc.children = copy.deepcopy(Data2.children2)
```
Low hanging fruit:
- keyword arguments,
- use diff library on output.

Battles lost:
- star imports,
- repetition,
- serialization,
- other projects using it.
A new beginning

Assignment

- intercept, handle in child,
- `__setattr__()` and `_assign()`.

Repetition

- field ordering (Django),
- introspection
  - `dir()`
  - `isinstance()`,
- printing and serialization.

class NewBase(object):
    _creation_counter = 0

def __lt__(self, other):
    if isinstance(other, NewBase):
        return self._creation_counter < other.creation_counter
    return NotImplemented

def __init__(self, **init_values):
    self._creation_counter = NewBase._creation_counter
    NewBase._creation_counter += 1
    self.fields = self._sorted_fields()
    for name, value in init_values.items():
        setattr(self, name, value)

def __setattr__(self, name, value):
    sub = getattr(self, name)
    if isinstance(sub, NewBase):
        sub._assign(name, value)
    else:
        super(NewBase, self).__setattr__(name, value)

def _assign(self, name, other):
    if isinstance(other, type(self)):
        self._value = other._value
        for name in self.fields:
            setattr(self, name, getattr(value, name))
    else:
        self._value = other
class NewerBase(object):
    _creation_counter = 0
    def __lt__(self, other):
        if isinstance(other, NewerBase):
            return self._creation_counter < other._creation_counter
        return NotImplemented
    def __init__(self, **init_values):
        self._creation_counter = NewerBase._creation_counter
        NewerBase._creation_counter += 1
        self.fields = self._sorted_fields()
        for field in init_values.items():
            setattr(self, name, value)
    def __set__(self, instance, other):
        if self in instance.fields:
            if isinstance(other, NewerBase):
                self.value = other.value
            for name in self.fields:
                setattr(self, name, getattr(value, name))
        else:
            self.value = other
    def __get__(self, instance):
        return self
Scary call tree

Rules:
- Data descriptor (__set__ and __get__),
- Fallback to inheritance
  - instance,
  - class,
- Non-data descriptors,
- __getattr__ fallback,
A typical case

Overriding assignment:
- attribute read,
- attribute write,
- still use methods.

Used in:
- properties,
- class methods,
- static methods.

# top/protocol/library.py TYPE DEFINITION

class ABCType(atom.Enum):
    Unknown = 0
    A = 1
class Version(struct.Tree):
    major = atom.Uint8(max_value=99)
    minor = atom.Uint8(max_value=99)
class ConstrainedList(struct.List):
    allowed_types = (object,)
    #...

# top/protocol/data.py DATA DEFINITION
class ABCElement(struct.Tree):
    type = ABCType().A
    version = Version(major=1, minor=0)
    children = ConstrainedList(
        allowed_types=(CDElement, FGElement),
        default=[CDElement()]  
    )

# top/scenarios/protocol_case.py DATA USAGE
abc = ABCElement(version=Version(major=2, minor=1))
abc.version.minor = 3  # intercept assignment
abc.version.minor.is_valid()  # it’s still a field
abc.version.minor.serialize_to(stream)
abc.version.minor.pretty_print()
Copy-pastable tests
simple definition, easier to print, easier to read, copy-pasteable.

# pretty print output before
# ABCElement:
#  Type: A
#  Version: 5.1
#   Children:
#     CDElement: ...

# pretty print output AND definition after
ABCElem1(  
type=ABCType().A,  
version=Version(major=5, minor=1),  
children=[  
  CDElem(  
    sub_elem=SubElem(  
      infra_elem1=0x123,  
      infra_elem2=0x234,  
    ),  
  ),  
  FGElem(),  
])

# definition before
abcElement = ABCElem()  
abcElement.setVersion(5, 1)  
cdElement = CDElem()  

cdElement.SubElem = SubElem()  

cdElement.SubElem.InfraElem1(0x123)  

cdElement.SubElem.InfraElem2(0x234)  

gfElement = FGElem()  

gbElement = FGElement()  

abcElement.Children.append(cdElement)  
abcElement.Children.append(gfElement)
“The moral of the story is that you can prove anything with a contrived example…”
— Joel Spolsky
- choose most bang for the buck,
- Pareto principle 80-20,
- new solutions on new features,
- if it’s not broken, don’t fix it.
Thank you!

Questions?

Adrian Dziubek
adrian.dziubek@gmail.com
Ionel codelog: Understanding Python metaclasses
https://blog.ionelmc.ro/2015/02/09/understanding-python-metaclasses/

Python Documentation: Data model
https://docs.python.org/3/reference/datamodel.html

Introduction to Python descriptors at IBM