
FRESHBOOKS
painless billing

Python Metaclasses

Introduction

Why would you need them?

Metaclasses are deeper magic than 99% of users should ever worry about. If you
wonder whether you need them, you don't (the people who actually need them know
with certainty that they need them, and don't need an explanation about why).
-- Python Guru Tim Peters

But we still want to know what are they!

Sample use case

- Defining currency classes in evolve
  - See currency1.py
- How can we avoid manually “registering” the currencies?
- How can we make sure that all subclasses define all required attributes?

- Metaclasses!

Object Oriented Python

- Python has classes and objects
- Python classes are “templates” for objects (class instances).
- __dict__, __class__, vars(), type() and dir()
- attribute resolution order:
  - object -> class -> parents
- In Python EVERYTHING is an object
  - Integer, strings ...
  - Functions are callable objects, __call__

Class objects

- classes are callable objects too!
- C(*args) is equivalent to C.__call__(*args)
- C(*args) somehow calls C.__init__(self, *args)
- But wait, where is __call__ defined?
- It's not in C.__dict__
- attribute resolution order:
  - object -> bases -> class -> bases
  - class -> bases -> metaclass -> metabases
- Trivia: why myobj does not have __call__ ?

Object Construction

- Where is __call__ defined?
  - It's in C.__class__.__dict__
  - See typeobject.c
- __new__(cls, *args, **kw)
  - create and return an instance of cls
  - let you customize the allocation
- __init__(self, *args, **kw)
  - Initialize the instance
  - return None
  - let you configure the created object
  - not called when unpickling

Metaclasses

- A metaclass is the class of a class
- An instance of a metaclass is a class
- For any x, the class of x is type(x)
- For any x, the metaclass of x is type(type(x))
- If C is a class, we often say C's metaclass is type(C)
- Classes are callable object “factory”
  - myobj = MyClass(a, b, c)
- Metaclasses are callable class factory

Class construction

- Metaclasses are just normal classes. You can create class instances by calling the metaclass
  - type('C', (A, B), {'x': 3, 'f': <function>})
- When a class statement is executed
  - Call MetaClass(name, bases, attrs)
  - Returned value is assigned to the class name
  - This happens at the end of the class statement, after the body of the class has been executed.

Choosing the metaclass

- __metaclass__ attribute on the class
- __metaclass__ attribute on the base classes
- __metaclass__ global variable
- Overwise classic metaclass (types.ClassType) is used
  - Always inherit from object!

Class construction

- Metaclasses are just normal classes.
  - cls = M(name, bases, attrs) is effectively:
  - cls = M.__call__(M, name, bases, attrs)
  - Which is equivalent too:
  - cls = M.__new__(M, name, bases, attrs)
  - M.__init__(cls, name, bases, attrs)
- We can customize class creation by overriding __init__ of the metaclass (or __new__ or __call__ ...)
  - See currency2.py

Questions?

FRESHBOOKS
painless billing