RECAP

- metaprogramming, as the meta- backformation implies, refers to code that is self-referential

- this advanced concept covers the programming paradigm where a program takes another program as input, and interprets, analyzes, or modifies it

- decorators, descriptors, and metaclasses are a few examples of commonly used language constructs that employ metaprogramming

RECAP

- in OOP classes act as blueprints for creating objects
- but in python everything, including classes, is an object, and a metaclass acts acts as the blueprint for building class objects
- type is the "class-creating class" which unlike any other object in python, has itself as a metaclass

RECAP

- the type() built-in is polymorphic in python
    - type(obj) **->** object.__class__
    - type(name, bases, namespace) **->** creates and returns a new type (or class)
- the class returned by this more functional approach is identical to the one defined using the class keyword

RECAP

- we define our own custom metaclasses by subclassing type
- using the metaclass keyword argument we could then use that custom metaclass in our class definition
- that class, its subclasses, as well as all instances of both will then follow the behaviour defined in the metaclass
- the advantage of this approach is not only code organization, but also performance

RECAP

- class creation could be broken down into 4 steps, each of which could be customized in the corresponding metaclass
  - **namespace** preparation (__prepare__): the key-value bindings defining the variables and methods are defined
  - **creation** (__new__): the class itself is created
  - **initialization** (__init__): the newly created class object is bound to other specific attributes and values
  - **instantiation** (__call__): the part when the class object is called to produce instances of the class

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).

Tim Peters