
sumtypes

Release 0.1a4

June 04, 2015

1	API	3
	Python Module Index	5

sumtypes provides Algebraic Data Types for Python. The main benefit is the implementation of Sum Types (aka [Tagged Unions](#)), which Python doesn't have any native representation for.

Decorate your classes to make them a sum type:

```
import attr
from sumtypes import sumtype, match

@sumtype
class MyType(object):
    # constructors specify names for their arguments
    MyConstructor = constructor('x')
    AnotherConstructor = constructor('x', 'y')

    # You can also make use of any feature of the attrs
    # package by using attr.ib in constructors
    ThirdConstructor = constructor(
        ('one', attr.ib(default=42)),
        ('two', attr.ib(validator=attr.validators.instance_of(int))))
```

(attrs package, and attr.ib documentation)

Then construct them by calling the constructors:

```
v = MyType.MyConstructor(1)
v2 = MyType.AnotherConstructor('foo', 2)
```

You can get the values from the tagged objects:

```
assert v.x == 1
assert v2.x == 'foo'
assert v2.y == 2
```

You check the constructor used:

```
assert type(v) is MyType.MyConstructor
```

And, like Scala case classes, the constructor type is a subclass of the main type:

```
assert isinstance(v, MyType)
```

And the tagged objects support equality:

```
assert v == MyType.MyConstructor(1)
assert v != MyType.MyConstructor(2)
```

Simple pattern matching is also supported. To write a function over all the cases of a sum type:

```
@match(MyType)
class get_number(object):
    def MyConstructor(x): return x
    def AnotherConstructor(x, y): return y
```

`match()` ensures that all cases are handled. If you really want to write a ‘partial function’ (i.e. one that doesn’t cover all cases), use `match_partial()`.

exception `sumtypes.PartialMatchError` (*unhandled_cases*)
Bases: `exceptions.Exception`

Raised when a match function doesn’t cover all cases.

`sumtypes.constructor` (**argspec*, ***kwargs*)
Register a constructor for the parent sum type.

Parameters *argspec* – each argument should be either a simple string indicating the name of an attribute, or two-tuples of (name, `attr.ib(...)`) where `attr.ib` is from the `attrs` package.

`sumtypes.match` (*adt*)

A class decorator that lets you write functions over all the constructors of a sum type. You provide the cases by naming the methods of the class the same as the constructors of the type, and the appropriate one will be called based on the way the value was constructed.

e.g.:

```
@sumtype
class MyType(object):
    NamedNum = constructor('name', 'num')
    AnonymousNum = constructor('num')

@match(MyType)
class get_num(object):
    def NamedNum(_, num): return num
    def AnonymousNum(num): return num

assert get_num(MyType.NamedNum('foo', 1)) == 1
assert get_num(MyType.AnonymousNum(2)) == 2
```

If not all constructors are handled, `PartialMatchError` will be raised. However, a default case can be implemented by defining a method named `_`, and it will be passed the value:

```
@match(MyType)
class get_name(object):
    def NamedNum(name, _): return name
    def _(_): return 'default'
```

`sumtypes.match_partial` (*adt*)

Like `match()`, but it allows not covering all the constructor cases.

In the case that

`sumtypes.sumtype` (*class*)

A class decorator that treats the class like a sum type.

Constructors should be wrapped/decorated with `constructor`.

Note that this will overwrite `__repr__`, `__eq__`, and `__ne__` on your objects. `__init__` is untouched, but it would be kind of weird to make something a sum type *and* have an `__init__`, so I recommend against that.

S

`sumtypes`, [3](#)

C

`constructor()` (in module `sumtypes`), 4

M

`match()` (in module `sumtypes`), 4

`match_partial()` (in module `sumtypes`), 4

P

`PartialMatchError`, 4

S

`sumtype()` (in module `sumtypes`), 4

`sumtypes` (module), 3