Magic Values ✨

“Surely I’ll never need to use a MetaClass, right?”

https://moll.dev/slides/magic

Intro.

- No, that’s not frostbite, it’s an incredibly bad sunburn ☹️

How can we use Generative AI to transform our business?

Sorry… This isn’t that talk…

Magic Numbers.

- Numeric Constants.
- Things that *probably* should be stored somewhere else.
- **Important strings.**

```
float Q_rsqrt( float number )
{
        long i;
        float x2, y;
        const float threehalfs = 1.5F;

        x2 = number * 0.5F;
        y  = number;
        i  = * ( long * ) &y;
        // evil floating point bit level hacking
        i  = 0x5f3759df - ( i >> 1 );
        y  = * ( float * ) &i;
        y  = y * ( threehalfs - ( x2 * y * y ) );

        return y;
}
```

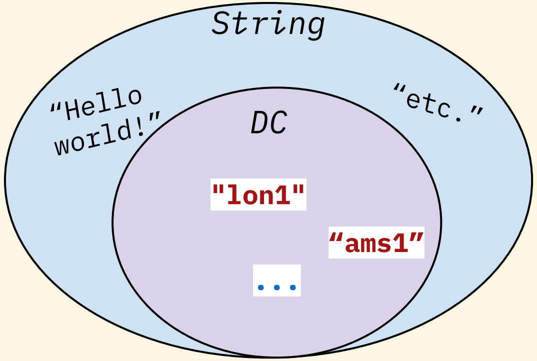
Magic Constants.

- Values that don’t change frequently.
    - Datacenter names
    - Availability zones
    - Environments
    - Etc. Business Logic
- Could be generated dynamically.
- Enums could work here

```
datacenter_london = “lon1”
datacenter_amsterdam = “ams1”

if src_datacenter == datacenter_london:
      return compute_price_non_eu()

if src_datacenter == datacenter_amsterdam:
      return compute_price_eu()
```

Magic Enums?

- Group magic values together!
- Enums provide:
  - Distinct subset of Values
  - Mapping between Symbol and Value
  - A distinct Type (namespace)
  - Some form of validation
  - MyPy technically works on them
- Limitations:
  - Can’t be composed (no Abstract types)
  - Confusing syntax [ ]vs( )



```
class DC(str, Enum):
    ams1 = "ams1"
    lon1 = "lon1"
```

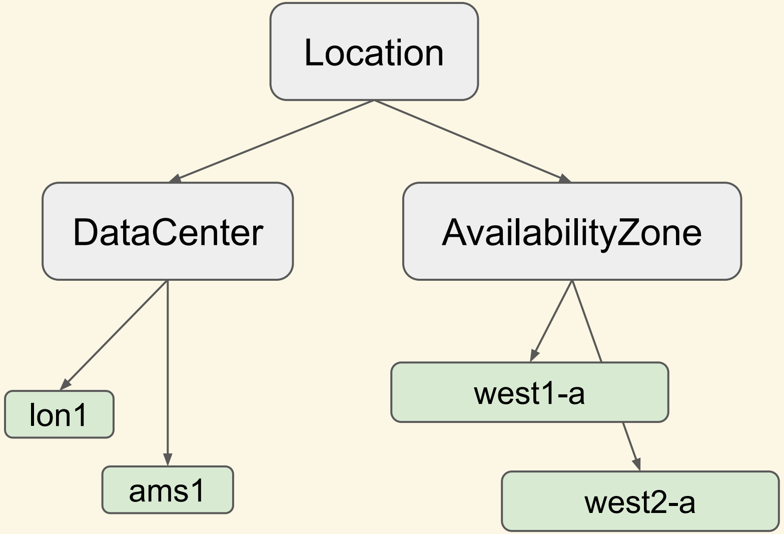
Eventually, any infrastructure code will need constant values, how can we build smarter constants that allow for more intuitive programming?

Ideal Magic Values?

- MyPy Type Friendly
- Constant-like
- Automatic input validation
- Automatic type coercion
- Composable
- Extensible

```
ams1 -> Datacenter.ams1
ams1() -> Value: ams1

Datacenter(“ams1”) -> ams1
Datacenter(“foo1”) -> ERROR

Location(“ams1”) == Datacenter(“ams1”) == ams1
Location.Datacenter.ams1 == ams1

def deploy(location: Location):
      ...
```

Value Composition.



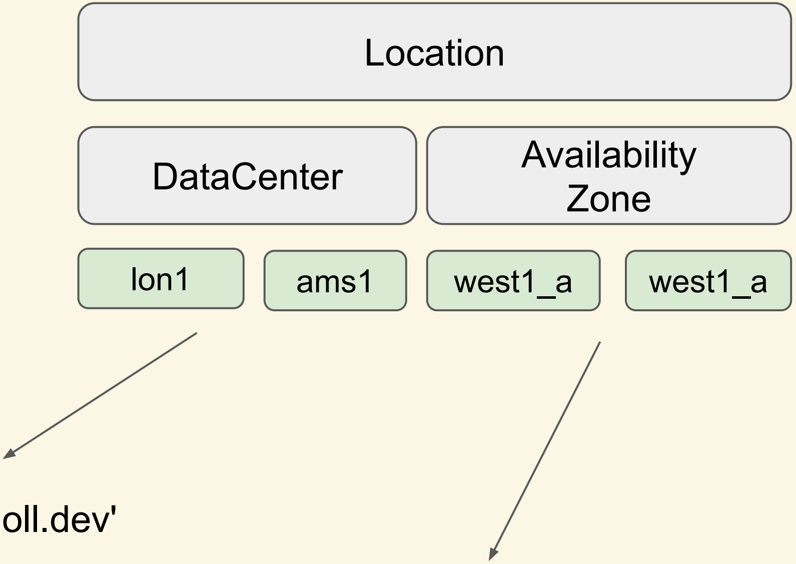
```
class Location(Constant):
    DataCenter: "DataCenter"
    AvailabilityZone: "AvailabilityZone"

class AvailabilityZone(Location):
    west1_a: "west1_a"
    west2_a: "west2_a"

class west1_a(AvailabilityZone):
    _value = "west1-a"

class west2_a(AvailabilityZone):
    _value = "west2-a"
```

```
class DataCenter(Location):
    lon1: "lon1"
    ams1: "ams1"

class lon1(DataCenter):
    _value = "lon1"

class ams1(DataCenter):
    _value = "ams1"
```

Live Coding Demo!

Multiple Dispatch?

- Using this type hierarchy how would we define a simple VM FQDN generator?



'test1-vm.lon1.moll.dev'

'test1-vm-west2-a.aws.moll.dev'

Live Coding Demo!

A slightly more complicated example.

↓env \ location →	lon1	ams1	west1-a	west2-a
prod	✅	✅	✅	✅
dev	❌	❌	✅	✅
pcc	❌	✅	❌	❌

Thanks!

Slides & Code
https://moll.dev/slides/magic