Python Mapping for Enumerations

Python does not have an enumerated type, so a Slice enumeration is emulated using a Python class: the name of the Slice enumeration becomes the name of the Python class; for each enumerator, the class contains an attribute with the same name as the enumerator. For example:

Slice

```
enum Fruit { Apple, Pear, Orange };
```

The generated Python class looks as follows:

Python

```
class Fruit(Ice.EnumBase):
    def valueOf(self, n):
        # ...
    valueOf = classmethod(valueOf)

    # ...

Fruit.Apple = ...
Fruit.Pear = ...
Fruit.Orange = ...
```

Each instance of the class has a `value` attribute providing the Slice value of the enumerator, and a `name` attribute that returns its name. The `valueOf` class method translates a Slice value into its corresponding enumerator, or returns `None` if no match is found.

Given the above definitions, we can use enumerated values as follows:

Python

```
f1 = Fruit.Apple
f2 = Fruit.Orange

if f1 == Fruit.Apple:               # Compare with constant
    # ...

if f1 == f2:                        # Compare two enums
    # ...

if f2.value == Fruit.Apple.value:   # Use Slice values
    # ...
elif f2.value == Fruit.Pear.value:
    # ...
elif f2.value == Fruit.Orange.value:
    # ...

Fruit.valueOf(1)  # Pear
```

As you can see, the generated class enables natural use of enumerated values. The `Fruit` class attributes are preinitialized enumerators that you can use for initialization and comparison.

Note that the generated class also defines a number of Python special methods, such as `__str__` and rich comparison operators, which we have not shown. The rich comparison operators compare the Slice value of the enumerator, which is not necessarily the same as its ordinal value.

Suppose we modify the Slice definition to include a custom enumerator value:

Slice

```
enum Fruit { Apple, Pear = 3, Orange };
```

We can use `valueOf` to examine the Slice values of the enumerators:

Python

```
Fruit.valueOf(0)  # Apple
Fruit.valueOf(1)  # None
Fruit.valueOf(3)  # Pear
Fruit.valueOf(4)  # Orange
```

See Also

- Enumerations
- Python Mapping for Identifiers
- Python Mapping for Modules
- Python Mapping for Built-In Types
- Python Mapping for Structures
- Python Mapping for Sequences
- Python Mapping for Dictionaries
- Python Mapping for Constants
- Python Mapping for Exceptions