Introspecting Objects

Austin Bingham
COFOUNDER - SIXTY NORTH

@austin_bingham

Robert Smallshire
COFOUNDER - SIXTY NORTH

@robsmallshire

Overview

Introspecting the attributes of objects

Accessing attributes by string names

Details of how Python stores metadata for objects

Use these tools to build an interesting function

Introspecting Objects

```
', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '_
_rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__',
 '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__'
, 'as_integer_ratio', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'i
mag', 'numerator', 'real', 'to_bytes']
>>> getattr(a, 'denominator')
1
>>> a.denominator
1
>>> getattr(a, 'conjugate')
<built-in method conjugate of int object at 0x10a7d2ed0>
>>> callable(getattr(a, 'conjugate'))
True
>>> a.conjugate.__class__.__name__
'builtin_function_or_method'
>>> getattr(a, 'index')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'int' object has no attribute 'index'
>>> hasattr(a, 'bit_length')
True
>>> hasattr(a, 'index')
False
>>>
```

Easier to Ask Forgiveness

You should generally prefer "EAFP" style programming

Programs using `hasattr()` can quickly become messy

The optimistic approach can actually be faster

```python
from fractions import Fraction


def mixed_numeral(vulgar):
    if not (hasattr(vulgar, 'numerator') and hasattr(vulgar, 'denominator')):
        raise TypeError("{} is not a rational number".format(vulgar))

    integer = vulgar.numerator // vulgar.denominator
    fraction = Fraction(vulgar.numerator - integer * vulgar.denominator,
                        vulgar.denominator)
    return integer, fraction
```

```
>>> from numerals import mixed_numeral
>>> from fractions import Fraction
>>> mixed_numeral(Fraction('11/10'))
(1, Fraction(1, 10))
>>> mixed_numeral(1.7)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/private/var/folders/0k/58g36_tx22xcxqd9mwqzg_h00000gp/T/tmp2hsuse0z/sli
de_spec/mixed-1/numerals.py", line 6, in mixed_numeral
    raise TypeError("{} is not a rational number".format(vulgar))
TypeError: 1.7 is not a rational number
>>>
```

```python
from fractions import Fraction



def mixed_numeral(vulgar):
    integer = vulgar.numerator // vulgar.denominator
    fraction = Fraction(vulgar.numerator - integer * vulgar.denominator,
                        vulgar.denominator)
    return integer, fraction
```

```
>>> from fractions import Fraction
>>> from numerals import mixed_numeral
>>> mixed_numeral(Fraction('11/10'))
(1, Fraction(1, 10))
>>> mixed_numeral(1.7)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/private/var/folders/0k/58g36_tx22xcxqd9mwqzg_h00000gp/T/tmpgw_b4cxp/sli
de_spec/mixed-2/numerals.py", line 5, in mixed_numeral
    integer = vulgar.numerator // vulgar.denominator
AttributeError: 'float' object has no attribute 'numerator'
>>>
```

```python
from fractions import Fraction


def mixed_numeral(vulgar):
    try:
        integer = vulgar.numerator // vulgar.denominator
        fraction = Fraction(vulgar.numerator - integer * vulgar.denominator,
                            vulgar.denominator)

        return integer, fraction
    except AttributeError as e:
        raise TypeError("{} is not a rational number".format(vulgar)) from e
```

```
    integer = vulgar.numerator // vulgar.denominator
AttributeError: 'float' object has no attribute 'numerator'

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/private/var/folders/0k/58g36_tx22xcxqd9mwqzg_h00000gp/T/tmpyu0l9ksu/sli
de_spec/mixed-3/numerals.py", line 12, in mixed_numeral
    raise TypeError("{} is not a rational number".format(vulgar)) from e
TypeError: 1.7 is not a rational number
>>>
```

Summary

`dir()` lists the attributes of an object

Methods are just attributes of objects

`int` includes attributes allowing it to be used as a rational or complex number

`getattr()` allows you to access attributes by string name

`getattr()` raises `AttributeError` if the attribute does not exist

`callable()` determines if an object can be called like a function

Summary

Objects store their type information on their `__class__` attribute

Class objects store their name on their `__name__` attribute

`hasattr()` determines if an object has an attribute with a given name

It's generally better to use "Easier to Ask Forgiveness than Permission" rather than "Look Before You Leap" style programming in Python

The EAFP style is often cleaner and faster