Turing-Complete Module Exports

…maxxing out Python's object orientation.

by Artur Roos.

Self

- Artur Roos.
- First Year, Bachelor’s of Computer Engineering.

Self

- Artur Roos.
- First Year, Bachelor's of Computer Engineering.
- Programming Language madness enthusiast.

Python is an Object Oriented language.

Agenda

- Dunder Methods
- Modules
- ???
- Enlightenment
- Q & A

Dunder/Magic Methods

```
class Vec2:
  def __init__(self, x, y):
      self.x = x
      self.y = y

  def __add__(lhs, rhs):
      return Vec2(lhs.x + rhs.x, lhs.y + rhs.y)

a = Vec2(2, 3)
b = Vec2(8, 9)
c = a + b  # Vec(2 + 8, 9 + 3)
```

Dunder/Magic Methods

- `__init__`
- `__del__`
- `__repr__`
- `__str__`
- `__bytes__`
- `__format__`
- `__hash__`
- `__bool__`
- `__getattr__`
- `__getattribute__`
- `__setattr__`
- `__setattribute__`
- `__dir__`
- `__class__`
- `__set_name__`
- …and others…

Dunder/Magic Methods

- `__init__` **OK ?**
- `__del__` **WTF ???**
- `__repr__`
- `__str__` **OK ?**
- `__bytes__` **WTF ???**
- `__format__` **WTF ???**
- `__hash__` **WTF ???**
- `__bool__` **OK ?**
- `__getattr__`
- `__getattribute__`
- `__setattr__`
- `__setattribute__`
- `__dir__`
- `__class__` **WTF ???**
- `__set_name__` **WTF ???**
- …and others…

Dunder/Magic Methods

- `__init__`
- `__del__`
- `__repr__`
- `__str__`
- `__bytes__`
- `__format__`
- `__hash__`
- `__bool__`
- **`__getattr__`**
- **`__getattribute__`**
- **`__setattr__`**
- **`__setattribute__`**
- `__dir__`
- `__class__`
- `__set_name__`
- …and others…

Dunder/Magic Methods

```python
class DictionaryObject:
    # supply the dictionary in the constructor
    def __init__(self, dictionary: dict):
        self.dictionary = dictionary

    # called when a member is accessed
    def __getattr__(self, value: str) -> str:
        return self.dictionary[value]
```

Dunder/Magic Methods

```python
dictionary = {'foo': 'bar'}
do = DictionaryObject(dictionary)
```

Dunder/Magic Methods

```
dictionary = {'foo': 'bar'}
do = DictionaryObject(dictionary)
```

```
>>> do.dictionary is dictionary
True
```

Dunder/Magic Methods

```
dictionary = {'foo': 'bar'}
do = DictionaryObject(dictionary)
```

```
>>> do.dictionary is dictionary
True
```

```
>>> do.__getattr__('foo')
'bar'
```

Dunder/Magic Methods

```
dictionary = {'foo': 'bar'}
do = DictionaryObject(dictionary)

>>> do.dictionary is dictionary
True

>>> do.__getattr__('foo')
'bar'

>>> do.foo
'bar'
```

Dunder/Magic Methods

```
dictionary = {'foo': 'bar'}
do = DictionaryObject(dictionary)

>>> do.dictionary is dictionary
True

>>> do.__getattr__('foo')
'bar'

>>> do.foo
'bar'

>>> do.qua
KeyError: 'qua'
```

Python is an Object Oriented language.

No, seriously. Yes, it’s really object oriented.

Agenda

- Dunder Methods
- **Modules**
- ???
- Enlightenment
- Q & A

Modules

```
>>> import math
```

```
>>> type(math)
<class 'module'>
```

```
>>> math.sin(0)
0.0
```

Modules

```
>>> import math
```

```
>>> type(math)
<class 'module'>
```

```
>>> math.sin(0)
0.0
```

```
# example.py
def sqr(x):
    return x * x
```

```
>>> import example
>>> maths.sqr(4)
16
```

Modules

```python
# crude, but valid
import math
import example
sin = math.sin
cos = math.cos
sqr = example.sqr
```

Modules

```
# crude, but valid
import math
import example
sin = math.sin
cos = math.cos
sqr = example.sqr

# idiomatic
from math import sin, cos
from example import sqr
```

Modules

```
import math        # type(math) == builtins.module
import example     # type(example) == builtins.module

sin = math.sin
cos = math.cos
sqr = example.sqr
```

Modules

```
import math         # type(math) == builtins.module
import example      # type(example) == builtins.module

sin = math.sin
cos = math.cos
sqr = example.sqr
```

Wait… MODULES ARE OBJECTS?..

Python is an Object Oriented language.

Modules

```
# counter.py
i = 0
def __iadd__(self, rhs):
    global i
    i += rhs

>>> import counter
>>> counter.__iadd__(2)
>>> counter += 4
>>> counter.i
6
```

Agenda

- Dunder Methods
- Modules
- ??? (you are here)
- Enlightenment
- Q & A

???

```
# attrprinter.py

def __getattr__(self, value):
    print(f"Access to {value}!")

>>> from attrprinter import LionKing, alphabet, _
Access to LionKing!
Access to alphabet!
Access to _!
```

Agenda

- Dunder Methods
- Modules
- ~~??? (you are here)~~ **Usecases!**
- Enlightenment
- Q & A

Usecases: HTBuilder

```
>>> from htbuilder import b, i
>>> hello = b(i("Hello, world!"))
<htbuilder.HtmlElement object at 0x7ffb63da9650>
```

```
>>> str(hello)
'<b><i>Hello, world!</i></b>'
```

Usecases: HTBuilder

```
>>> from htbuilder import b, i
>>> hello = b(i("Hello, world!"))
<htbuilder.HtmlElement object at 0x7ffb63da9650>

>>> str(hello)
'<b><i>Hello, world!</i></b>'

>>> from htbuilder import amogus
>>> amogus("ඞ")
'<amogus>ඞ</amogus>'
```

Usecases: SymPy

```
>>> from sympy import symbols, sin, cos, pi
>>> x, y, z = symbols("x, y, z")
>>> expr = cos(x) + sin(y)
>>> expr.subs(cos(x), y)
z + sin(y)
>>> expr.subs(x, 0).subs(y, pi / 2)
2
```

Usecases: SymPy

This is lovely…

```
from sympy import symbols
x, y, z = symbols("x, y, z")
```

Usecases: SymPy

This is lovely…

```python
from sympy import symbols
x, y, z = symbols("x, y, z")
```

But this is better!

```python
from sympy.abc import x, y, z
```

Usecases: SymPy

```python
# hooke.py
from sympy.abc import x, k
F = -k * x

# parabola.py
from sympy.abc import x
y = x**2 - x - 1
```

Usecases: SymPy

```
# hooke.py
from sympy.abc import x, k
F = -k * x

# parabola.py
from sympy.abc import x
y = x**2 - x - 1
```

Usecases: SymPy

```
# hooke.py
from sympy.abc import x, k
F = -k * x

# parabola.py
from sympy.abc import x
y = x**2 - x - 1
```

```
from sympy.abc import J
>>> W = F * y
>>> W
(-k*x) * (x**2 - x - 1)
```

Usecases: SymPy

```python
# hooke.py
from sympy.abc import x, k
F = -k * x

# parabola.py
from sympy.abc import x
y = x**2 - x - 1
```

```python
from sympy.abc import J

>>> W = F * y
>>> W
(-k*x) * (x**2 - x - 1)

>>> W.subs(hookes_x, J)
(-J*k) * (J**2 - J - 1)
```

💥💥💥💥💥💥💥

¯_(ツ)_/¯

Usecases: Units with Pint

```
# pint.py
GLOBAL_REGISTRY = UnitRegistry()
def __getattr__(self, unit):
    if unit in GLOBAL_REGISTRY:
        return getattr(GLOBAL_REGISTRY, unit)
    else:
        raise KeyError(...)
>>> from pint import m, cm
>>> 3 * m + 4 * cm
<Quantity(3.04, 'meter')>
```

Usecases: Dynamic Function Loading

Current usage example:

```
# library.dll
dll = ctypes.CDLL("library.dll")
function = getattr(dll, "function")
```

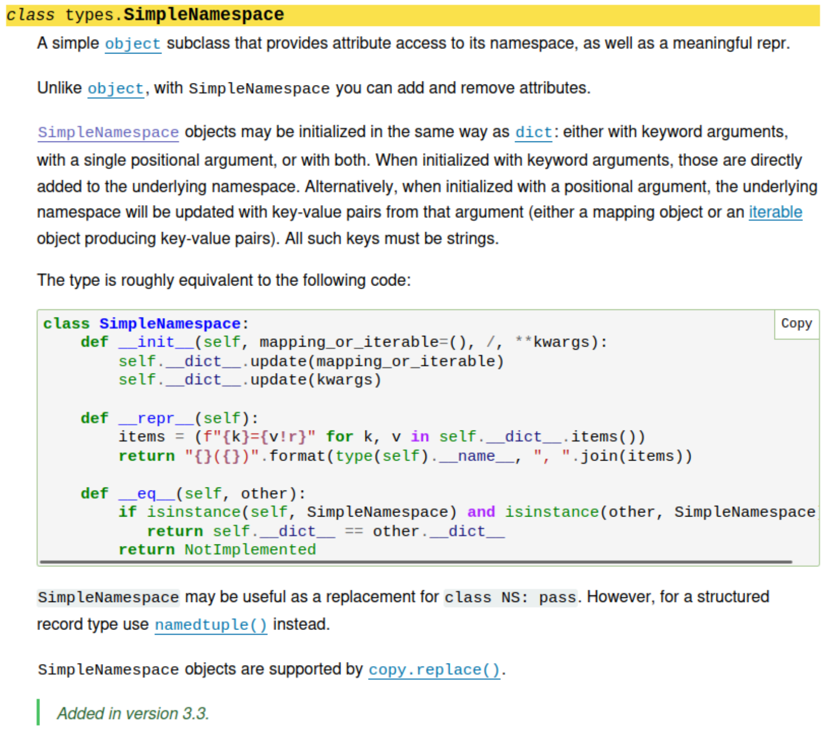
Usecases: Dynamic Function Loading

Current usage example:

```
# library.dll
dll = ctypes.CDLL("library.dll")
function = dll.__getattr__("function")  # getattr
dll.function
```

This is how it is already done in the standard library!

Usecases: `DictionaryObject`



class types.**SimpleNamespace**

A simple `object` subclass that provides attribute access to its namespace, as well as a meaningful repr.

Unlike `object`, with `SimpleNamespace` you can add and remove attributes.

`SimpleNamespace` objects may be initialized in the same way as `dict`: either with keyword arguments, with a single positional argument, or with both. When initialized with keyword arguments, those are directly added to the underlying namespace. Alternatively, when initialized with a positional argument, the underlying namespace will be updated with key-value pairs from that argument (either a mapping object or an iterable object producing key-value pairs). All such keys must be strings.

The type is roughly equivalent to the following code:

```
class SimpleNamespace:
    def __init__(self, mapping_or_iterable=(), /, **kwargs):
        self.__dict__.update(mapping_or_iterable)
        self.__dict__.update(kwargs)

    def __repr__(self):
        items = (f"{k}={v!r}" for k, v in self.__dict__.items())
        return "{}({})".format(type(self).__name__, ", ".join(items))

    def __eq__(self, other):
        if isinstance(self, SimpleNamespace) and isinstance(other, SimpleNamespace
           return self.__dict__ == other.__dict__
        return NotImplemented
```

`SimpleNamespace` may be useful as a replacement for `class NS: pass`. However, for a structured record type use `namedtuple()` instead.

`SimpleNamespace` objects are supported by `copy.replace()`.

Added in version 3.3.

```
from types import
SimpleNamespace as SN

person = SN(name="Alice",
age=30)

person.name  # Alice
person.age   # 30
```

Agenda

- Dunder Methods
- Modules
- ???
- **Enlightenment**
- Q & A

Python is an Object Oriented language.

Enlightenment

```python
# module.py
public = 0
private = 1
__all__ = ["public"]
```

Enlightenment

```
# module.py
public = 0
private = 1
__all__ = ["public"]
```

```
>>> from module import *
>>> public
0
>>> private
NameError: name 'private' is not defined
```

Enlightenment

```
# htbuilder.py

def __getattr__(self, tag):
    ...

__all__ = ['span', 'div']

>>> from htbuilder import *
>>> div(span("Hello, world!"))
'<div><span>Hello, world!</span></div>'
```

Enlightenment

- `__init__`
- `__del__`
- `__repr__`
- `__str__`
- `__bytes__`
- `__format__`
- `__hash__`
- `__bool__`
- `__getattr__`
- `__getattribute__`
- `__setattr__`
- `__setattribute__`
- `__dir__`
- `__class__`
- `__set_name__`
- …and others…

Python is an Object Oriented language.

Questions?