criteo

R.Bauvin / H.Lerebours

13/7/2017

Inspect (or gadget?)

A quick introduction to introspection

The hiring test problem

How to check compliance with question requirements like:

```
Fix this code by changing a single line so that…

Implement a generator that…
```

when all you have is:

```
# Add test code for the candidate.
# To use its code, you have to import the module Answer
import Answer
```


criteo

The hiring test solution

When you don’t know, ask your best friend…

Solution:

```
inspect.getsource(object)
inspect.isgenerator(object)
```


criteo

Introspection according to Wiktionary

introspection (*plural* **introspections**)

1. (*object-oriented programming*) *Clipping of* **type introspection**.
2. (*psychology*) A looking inward; specifically, the act or process of self-examination, or inspection of one's own thoughts and feelings; the cognition which the mind has of its own acts and states; self-consciousness; reflection.

type introspection (*plural* **type introspections**)

1. (*object-oriented programming*) Ability of a program to examine at runtime the type or properties of an object.

Cool! In Python, ‘everything is an object’*

criteo

You already used introspection

```
def main():
    pass

if __name__ == '__main__':
    main()
```

Name of the current module / namespace

criteo

Builtins functions

id(x)	dir(x)	type(x)
hasattr(x, s)	getattr(x, s)	issubclass(x, s)
isinstance(x, s)	callable(x, s)	vars(x)
globals(x)	locals(x)	

+ plenty of object attributes to help – see documentation of `inspect` library

Type	Attribute	Description
module	__doc__	documentation string
	__file__	filename (missing for built-in modules)
class	__doc__	documentation string
	__name__	name with which this class was defined
	__qualname__	qualified name
	__module__	name of module in which this class was defined
method	__doc__	documentation string
	__name__	name with which this method was defined
	__qualname__	qualified name
	__func__	function object containing implementation of method
	__self__	instance to which this method is bound, or None
function	__doc__	documentation string
	__name__	name with which this function was defined

criteo

What can we get with builtins?

```python
def my_function_max(foo, bar):
    """Return the biggest among foo and bar and None if both are equal"""
    if foo > bar:
        return foo
    if foo < bar:
        return bar
    return None
```

introspection.py

>>> **type(**my_function_max**)**
<type 'function'>

>>> my_function_max**.__doc__**
'Return the biggest (…) are equal'

>>> my_function_max**.__module__**
‘introspection'

>>> my_function_max**.__name__**
‘my_function_max'

>>> **dir(**my_function_max**)**
['__annotations__', '__call__', '__class__',
'__closure__', '__code__', (…) , '__repr__', '__se
tattr__', '__sizeof__', '__str__', '__subclasshook__']

criteo

`inspect` – Inspect live objects

- Inspect library = ~60 functions providing :
  - type checking

    `ismodule, isclass, ismethod, isfunction, …`
  - getting source code

    `getmodule, getsourcefile, …`
  - inspecting classes and functions

    `signature, …`
  - examining the interpreter stack
- More info: https://docs.python.org/3.6/library/inspect.html

criteo

What can we get from `inspect`?

```python
def my_function_max(foo, bar):
    """Return the biggest among foo and bar and None if both are equal"""
    if foo > bar:
        return foo
    if foo < bar:
        return bar
    return None
```

introspection.py

```
>>> inspect.signature(my_function_max).parameters
mappingproxy(OrderedDict([
('foo', <Parameter "foo">),
('bar', <Parameter "bar">)]))
```

```
>>> inspect.getsourcelines(my_function_max)
(['def my_function_max(foo, bar):\n',
  '    """Return the biggest among foo and bar and
None if both are equal"""\n',
  '    if foo > bar:\n',
  '        return foo\n',
  '    if foo < bar:\n',
  '        return bar\n',
  '    return None\n'],
 12)
```


criteo

What about realistic use cases?

DEMO time!

When to use introspection?

General principle:

Should be only when you don’t know the information at compile time

Good candidates

- Exploration/Learning/Debugging
- I/O
- Polymorphism
- Interface
- ‘Meta-programming’

Warnings

- Performance
  - Pay attention to some introspections’ cost
  - Only check (once) what comes from outside
- Portability
  - Depending on Python implementation, some callables are not inspectable
- Maintainability
  - Depending on IDE, less help when doing refactoring

criteo

criteo

Thanks!

BTW,
Criteo is hiring! ☺