intercepts

David Shriver

Jul 30, 2023

GETTING STARTED

Intercepts allows you to intercept function calls in Python and handle them in any manner you choose. For example, you can pre-process the inputs to a function, or apply post-processing on its output. Intercepts also allows you to completely replace a function with a custom implementation.

```
>>> increment(41)
42
>>> intercepts.register(increment, handler)
>>> increment(41)
40
>>> intercepts.unregister(increment)
>>> intercepts.register(increment, handler2)
>>> increment(41)
'The answer is: 42'
>>> intercepts.unregister_all()
```

Handler functions receive all paramters to the intercepted function call and can access the intercepted function through the variable _.

```
>>> def handler(num):
...   result = _(num)
...   return num - (result - num)
>>> def handler2(*args, **kwargs):
...   result = _(*args, **kwargs)
...   return f"The answer is: {result}"
```

The intercepts module also allows intercepting python built-in functions, such as `print` and `sorted`. For best results, the intercepts module should be the first module imported.

```
>>> def print_handler(message):
...     return _(''.join(reversed(message)))
>>> print("Hello world")
Hello world
>>> intercepts.register(print, print_handler)
>>> print("Hello world")
dlrow olleH
```

CHAPTER

ONE

FURTHER READING

1.1 Installation

For general usage, we recommend installing intercepts from PyPI using pip.

1.1.1 Install with pip

Intercepts requires Python 3.7+ on Linux or Windows. There are currently no additional dependencies. To install intercepts, run:

```
$ pip install requests
```

Or, to install the latest version from github, run:

```
$ pip install -U git+https://github.com/dlshriver/intercepts.git@main
```

1.1.2 Get the Source

The source code for intercepts is available on Github.

You can clone the public repository:

```
$ git clone git://github.com/dlshriver/intercepts.git
```

Or download the tarball:

```
$ curl -OL https://github.com/dlshriver/intercepts/tarball/main
```

Or, download the zip archive:

```
$ curl -OL https://github.com/dlshriver/intercepts/archive/main.zip
```

To install intercepts from source, after downloading and extracting the source code and navigating to the root of the source directory, run:

```
$ pip install .
```

1.2 Quickstart

Ready to get started? This page provides an introduction on getting started with intercepts.

First, make sure that intercepts is *installed*.

Then, let's start with some simple examples.

1.2.1 Register an Intercept

Intercepting calls with intercepts is very simple.

Begin by importing the intercepts module.

```
>>> import intercepts
```

Now lets define an intercept handler, like so:

```
def handler(*args, **kwargs):
     return _(*args, **kwargs) * 2
```

This intercept handler simply doubles the output of the original call. In general, an intercept handler is a function that will be called in place of the original call. The handler will receive the original all of the parameters passed to the original call. It also has access to a special variable `_`, which is a reference to the intercepted method.

Now that we have defined a handler, we can register it to intercept a call with `register()`.

```
>>> intercepts.register(sum, handler)
```

That's it. Now any call to sum, in any module will be intercepted by `handler`, and its result will be doubled.

```
>>> sum([1, 2, 3, 4, 5, 6])
42
```

1.2.2 Stacking Intercepts

Multiple intercept handlers can be registered for a Python call. For example, we can define pre and post processing handlers, such as:

```
def pre_handler(*args, **kwargs):
    print("Executing function:", _.__name__)
    return _(*args, **kwargs)

def post_handler(*args, **kwargs):
    result = _(*args, **kwargs)
    print("Executed function:", _.__name__)
    return result
```

Registering these handlers for a function will print a message before and after that methods execution.

```
>>> intercepts.register(sum, pre_handler)
>>> intercepts.register(sum, post_handler)
>>> sum([1, 2, 3, 4, 5, 6])
```

(continues on next page)

(continued from previous page)

```
Executing function: sum
Executed function: sum
42
```

The same handler can even be applied multiple times.

```
>>> intercepts.register(sum, handler)
>>> intercepts.register(sum, handler)
>>> sum([1, 2, 3, 4, 5, 6])
84
```

Intercept handlers are stored as a stack, meaning that the last handler registered will be the first one that is executed. For example:

```
def handler_0(*args, **kwargs):
    print("handler 0")
    return _(*args, **kwargs)

def handler_1(*args, **kwargs):
    print("handler 1")
    return _(*args, **kwargs)

intercepts.register(abs, handler_0)
intercepts.register(abs, handler_1)
```

In this example, a call to `abs` will print `handler 1` and then `handler_0`.

```
>>> abs(-42)
handler 1
handler 0
42
```

1.2.3 Unregister an Intercept

To unregister the intercept handlers for a function, use the `unregister()` function.

```
>>> intercepts.unregister(sum)
```

This will remove all handlers from the `sum` function.

An integer value, `depth` can also be passed to `unregister()` to remove the last `depth` handlers from the function.

```
>>> intercepts.unregister(sum, depth=1)
```

Finally, you can unregister all intercept handlers with `unregister_all()`.

```
>>> intercepts.unregister_all()
```

1.3 API

This part of the documentation covers the API of `intercepts`. All of `intercepts` functionality can be accessed by these 3 methods.