How to Make Sense of an API?

How do you learn to use a library or an API that you've never worked with before? It might take you a bit of time to familiarize yourself with how the library operates, but that's okay. It's worth spending some time understanding the way the functions are organized, the inputs and outputs, and the general expectations of the library.

In general, a good API should be descriptive. You should be able to look at a function's name and have a pretty good idea of what it will do. A well-designed API will follow patterns and ***naming conventions***. That means that the functions, classes and methods should have names that help you understand what to expect from them. And when the name isn't enough, you should have access to the documentation for each of the functions that will help you figure out what they do.

For example, when we visit the reference page for the Image object in Pillow, we see this piece of example code:

```
from PIL import Image
im = Image.open("bride.jpg")
im.rotate(45).show()

```

This piece of code is pretty straightforward. Even without having seen this library before, you can probably guess that it opens an image called bride.jpg, rotates it 45 degrees, and then shows it on the screen.

But how can we know for sure? We can look up each of the functions in the documentation and check what they're supposed to do. When dealing with open-source libraries, we can even check out how the function is implemented to see if it matches our expectations. For a web service API or a closed-source library, you might not have access to the underlying code, but you should have access to the documentation that's generated by the code.

For a Python library like PIL, the code is documented using ***docstrings***. If you remember from waaaay back in our first course, docstrings are documentation that lives alongside the code. You've been using them ever since! When you use "help()" to describe a function, or read a description of what a Python function does in your IDE, what you're reading comes from docstrings in the code.

For example, let's take a look at the documentation for PIL:

```
>>> help(PIL)

Help on package PIL:

NAME
    PIL - Pillow (Fork of the Python Imaging Library)

DESCRIPTION
    Pillow is the friendly PIL fork by Alex Clark and Contributors.
        https://github.com/python-pillow/Pillow/

    Pillow is forked from PIL 1.1.7.

    PIL is the Python Imaging Library by Fredrik Lundh and Contributors.
    Copyright (c) 1999 by Secret Labs AB.

    Use PIL.__version__ for this Pillow version.
    PIL.VERSION is the old PIL version and will be removed in the future.

    ;-)

PACKAGE CONTENTS
    BdfFontFile
    BlpImagePlugin
    BmpImagePlugin
    BufrStubImagePlugin
    ContainerIO
    CurImagePlugin
    DcxImagePlugin
    DdsImagePlugin
    EpsImagePlugin
...

```

Lots of Python modules also publish their documentation online. Pillow's full documentation is published here. There, the docstrings have been compiled into a browsable reference, and they've also written a handbook with tutorials for you to get familiar with the library's API. Woohoo!

Mark as completed

Like Dislike Report an issue