Analyzing Your Code

Once pylint is installed, you can run it on the command line without any arguments to see what options it accepts. If that doesn't work, you can type out the full path like this:

```
c:\Python34\Scripts\pylint
```

Now we need some code to analyze. Here's a piece of code that has four errors in it. Save this to a file named *crummy_code.py*:

```
import sys

class CarClass:
    """"""

    def __init__(self, color, make, model, year):
        """Constructor"""
        self.color = color
        self.make = make
        self.model = model
        self.year = year

        if "Windows" in platform.platform():
            print("You're using Windows!")

        self.weight = self.getWeight(1, 2, 3)

    def getWeight(this):
        """"""
        return "2000 lbs"
```

Can you spot the errors without running the code? Let's see if pylint can find the issues!

```
pylint crummy_code.py
```

When you run this command, you will see a lot of output sent to your screen. Here's a partial example:

Here's a partial example:

```
c:\py101>c:\Python34\Scripts\pylint crummy_code.py
No config file found, using default configuration
************* Module crummy_code
C:  2, 0: Trailing whitespace (trailing-whitespace)
C:  5, 0: Trailing whitespace (trailing-whitespace)
C: 12, 0: Trailing whitespace (trailing-whitespace)
C: 15, 0: Trailing whitespace (trailing-whitespace)
C: 17, 0: Trailing whitespace (trailing-whitespace)
C:  1, 0: Missing module docstring (missing-docstring)
C:  3, 0: Empty class docstring (empty-docstring)
C:  3, 0: Old-style class defined. (old-style-class)
E: 13,24: Undefined variable 'platform' (undefined-variable)
E: 16,36: Too many positional arguments for function call (too-many-function-args)
C: 18, 4: Invalid method name "getWeight" (invalid-name)
C: 18, 4: Empty method docstring (empty-docstring)
E: 18, 4: Method should have "self" as first argument (no-self-argument)
R: 18, 4: Method could be a function (no-self-use)
R:  3, 0: Too few public methods (1/2) (too-few-public-methods)
W:  1, 0: Unused import sys (unused-import)
```

Let's take a moment to break this down a bit. First we need to figure out what the letters designate: C is for convention, R is refactor, W is warning and E is error. pylint found 3 errors, 4 convention issues, 2 lines that might be worth refactoring and 1 warning. The 3 errors plus the warning were what I was looking for. We should try to make this crummy code better and reduce the number of issues. We'll fix the imports and change the getWeight function to get_weight since camelCase isn't allowed for method names. We also need to fix the call to get_weight so it passes the right number of arguments and fix it so it has "self" as the first argument. Here's the new code:

```
# crummy_code_fixed.py
import platform

class CarClass:
    """"""

    def __init__(self, color, make, model, year):
        """Constructor"""
        self.color = color
        self.make = make
        self.model = model
        self.year = year

        if "Windows" in platform.platform():
            print("You're using Windows!")

        self.weight = self.get_weight(3)

    def get_weight(self, this):
        """"""
```

```
        return "2000 lbs"
```

Let's run this new code against pylint and see how much we've improved the results. For brevity, we'll just show the first section again:

```
c:\py101>c:\Python34\Scripts\pylint crummy_code_fixed.py
No config file found, using default configuration
************* Module crummy_code_fixed
C: 1,0: Missing docstring
C: 4,0:CarClass: Empty docstring
C: 21,4:CarClass.get_weight: Empty docstring
W: 21,25:CarClass.get_weight: Unused argument 'this'
R: 21,4:CarClass.get_weight: Method could be a function
R: 4,0:CarClass: Too few public methods (1/2)
```

That helped a lot! If we added docstrings, we could halve the number of issues. Now we're ready to take a look at pyflakes!