https://github.com/Instagram/Fixit

Fixit

a lint framework writes better Python code for you

Jimmy Lai

jimmylai@fb.com

09/05/2020

PyCon Taiwan 2020

About me

Jimmy Lai

- Software Engineer, Instagram
- I love Python and open source

https://github.com/Instagram/LibCST/

https://github.com/Instagram/Fixit

Agenda

- What’s Fixit?
- Why Fixit?
- How to build an Autofix rule?
- How to test a rule?
- Case study

What's Fixit

a lint framework writes better Python code for you

- Popular Python Lint Tools provide code suggestions.
  - Flake8
  - PyLint

```python
from typing import Dict


class C(object):
    attr = "ab" "cd" "ef" "gh"

    def method(self) -> Dict[int, str]:
        filtered_char = []
        for char in self.attr:
            if char is not "a":
                filtered_char.append(char)

        index_to_char = dict([(idx, char) for
        return index_to_char
```

How many code suggestions can you provide?

```
!flake8 example.py
```

```
example.py:10:16: F632 use ==/!= to compare constant literals (str, bytes, int, float, tuple)
```

What's Fixit

Fixit not only provide suggestion but also fix it for you

```
1  !python -m fixit.cli.apply_fix example.py --rules ComparePrimitivesByEqualRule
```

```
Scanning 1 files
example.py

All done! ✨ 🍰 ✨
1 file left unchanged.
example.py:10:16 [applied fix]
    ComparePrimitivesByEqualRule: Don't use `is` or `is not` to compare
    primitives, as they compare references. Use == or != instead.

Found 1 reports in 1 files in 0.32 seconds.
```

```
@@ -7,7 +7,7 @@ class C(object):
     def method(self) -> Dict[int, str]:
         filtered_char = []
         for char in self.attr:
-            if char is not "a":
+            if char != "a":
                 filtered_char.append(char)
```

Why Fixit?

- Manually fix lint suggestions looks easy. However,
  - In a large Python codebase like IG Server
    - Million lines of Python code
    - Hundreds of developers
  - A new lint suggestion may have thousands of existing violations
    - Lint suggestion can slow developers down
    - Fixit: help them write better code faster by fixing the issue for them

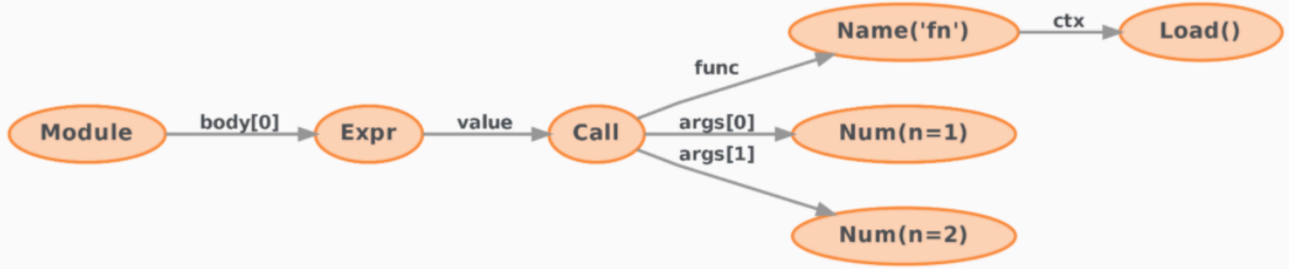
Why Fixit?

Our story of building Flake8 plugin

- Used AST visitor pattern
- Monolithic single-file AST traversal
  - Visit tree only once to be fast.
  - Highly coupled rules with shared states
  - Very hard to maintain
- Cannot disable a rule easily
- Not easy to build a rule and test it
- Not support advanced static analysis

```
fn(1, 2)  # calls fn
```

Show Code [+]




```
1  class TypingCollector(ast.NodeVisitor):
2
3      def visit_ClassDef(self, node: ast.ClassDef) -> Opti
4          self.stack.append(node.name.value)
5
6      def leave_ClassDef(self, node: ast.ClassDef) -> Non
7          self.stack.pop()
8
9      def visit_FunctionDef(self, node: ast.FunctionDef) -
10         self.stack.append(node.name.value)
11         self.annotations[tuple(self.stack)] = (node.para
12         return (
13             False
```

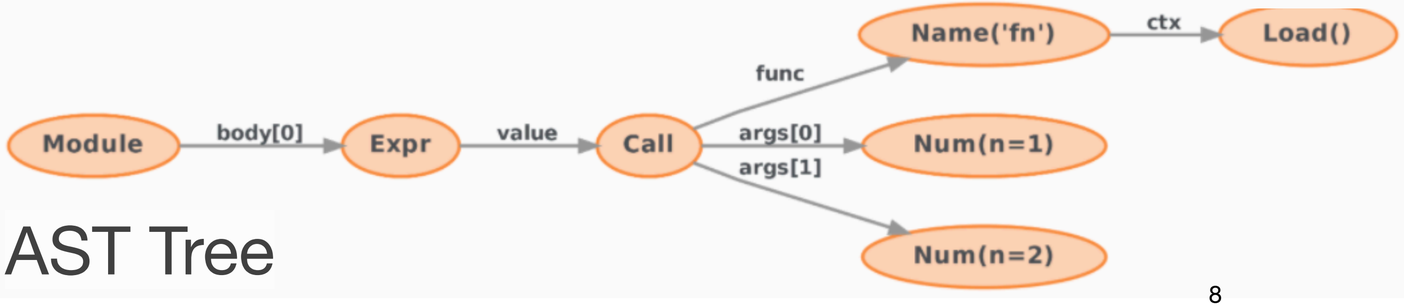
LibCST

Parse Concrete Syntax Tree

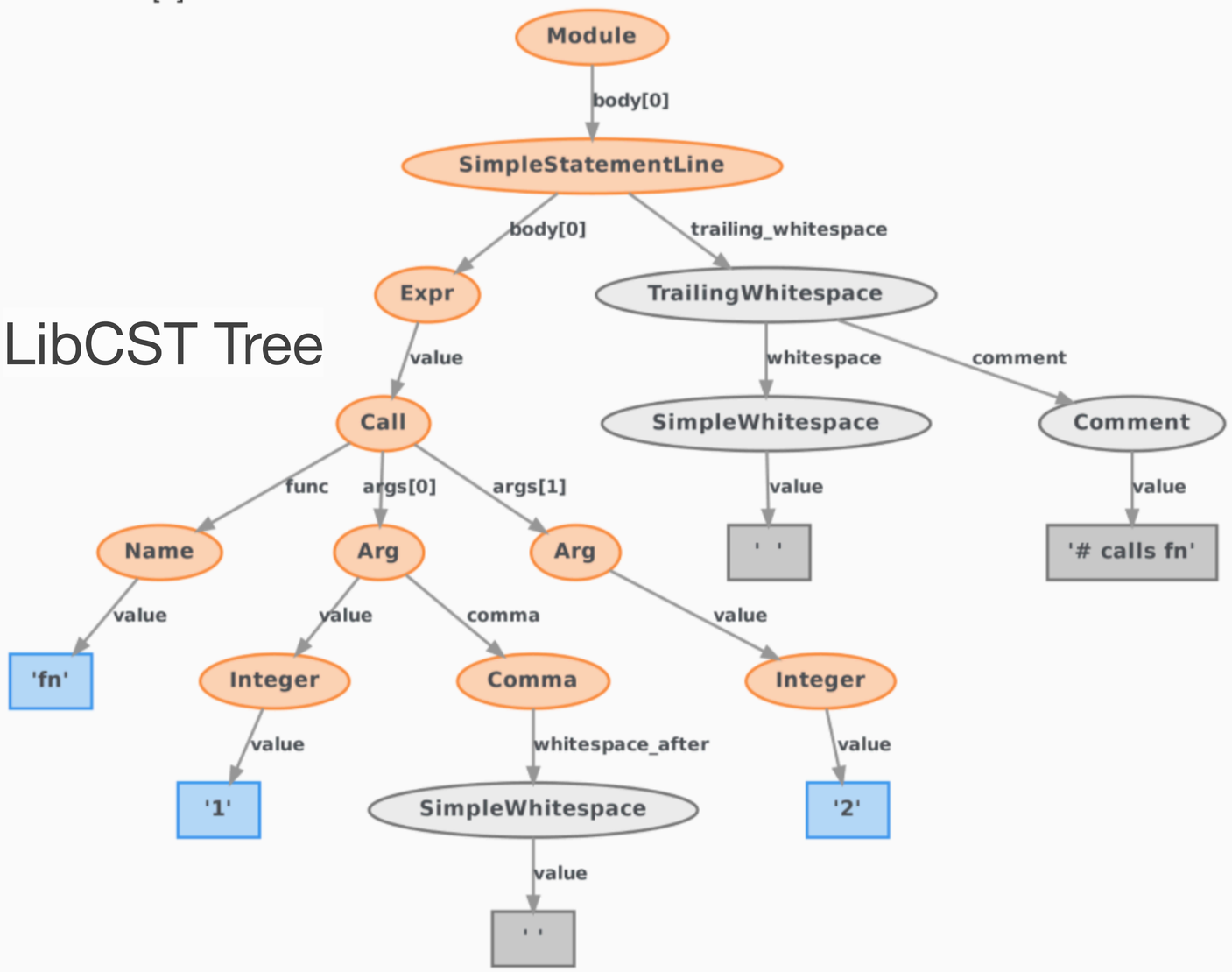
- LibCST parse code as a CST that looks like and feels like an AST
- Matcher for easy tree traversal
- Metadata for static analysis

```
fn(1, 2)  # calls fn
```

Show Code [+]



AST Tree



LibCST Tree

Build a rule - Pick a Good Name

https://fixit.readthedocs.io/en/latest/build_a_lint_rule.html

- In Flake8, a numeric code is used (e.g. F405) and it's not straightforward
- Pick a Good Name: sort and actionable
  - Describe the action, not the issue
    - AwaitInLoopLintRule < GatherSequentialAwaitRule
  - Ends with "Rule"

```
# an example with unnecessary object inheritance
class C(object):
    ...


# the above example can be simplified as this
class C:
    ...
```

NoInheritFromObjectRule

Build a rule - Identify a pattern

https://fixit.readthedocs.io/en/latest/build_a_lint_rule.html

- Implement CstLintRule
- LibCST: CSTNode, matchers

```python
# an example with unnecessary object inheritance
class C(object):
    ...


# the above example can be simplified as this
class C:
    ...
```

```python
class NoInheritFromObjectRule(CstLintRule):
    """
    In Python 3, a class is inherited from ``object`` by default.
    Explicitly inheriting from ``object`` is redundant, so removing
    it keeps the code simpler.
    """
    MESSAGE = ("Inheriting from object is a no-op. 'class Foo:' is just fine =)")

    def visit_ClassDef(self, node: cst.ClassDef) -> None:
        new_bases = tuple(
            base for base in node.bases if not m.matches(base.value, m.Name("object"))
        )
        if tuple(node.bases) != new_bases:
            self.report(node)
```

Build a rule - add an autofix

https://fixit.readthedocs.io/en/latest/build_a_lint_rule.html

- Provide a replacement CSTNode

```python
class NoInheritFromObjectRule(CstLintRule):
    MESSAGE = "Inheriting from object is a no-op. 'class Foo:' is just fine =)"

    def visit_ClassDef(self, node: cst.ClassDef) -> None:
        new_bases = tuple(
            base for base in node.bases if not m.matches(base.value, m.Name("object"))
        )

        if tuple(node.bases) != new_bases:
            # reconstruct classdef, removing parens if bases and keywords are empty
            new_classdef = node.with_changes(
                bases=new_bases,
                lpar=cst.MaybeSentinel.DEFAULT,
                rpar=cst.MaybeSentinel.DEFAULT,
            )

            # report warning and autofix
            self.report(node, replacement=new_classdef)
```

Test a Lint Rule

Code Examples as Unit Tests

- Provide VALID and INVALID test cases.
-

```
! python -m unittest fixit.tests.NoInheritFromObjectRule
```

```
....
----------------------------------------------------------------------
Ran 4 tests in 0.074s

OK
```

```python
from fixit import (
    CstLintRule,
    InvalidTestCase as Invalid,
    ValidTestCase as Valid,
)

class NoInheritFromObjectRule(CstLintRule):
    VALID = [
        Valid("class A(something):    pass"),
        Valid(
            """
            class A:
                pass"""
        ),
    ]
    INVALID = [
        Invalid(
            """
            class B(object):
                pass""",
            line=1,
            column=1,
            expected_replacement="""
            class B:
                pass""",
        ),
        Invalid(
            """
            class B(object, A):
                pass""",
            line=1,
            column=1,
            expected_replacement="""
            class B(A):
                pass""",
        ),
    ]
```

Test a Lint Rule

```python
from typing import Dict


class C(object):
    attr = "ab" "cd" "ef" "gh"

    def method(self) -> Dict[int, str]:
        filtered_char = []
        for char in self.attr:
            if char is not "a":
                filtered_char.append(char)
```

- Test on existing code

```
! python -m fixit.cli.run_rules example.py --rules NoInheritFromObjectRule
```

```
Scanning 1 files
Testing 1 rules

example.py:4:1
    NoInheritFromObjectRule: Inheriting from object is a no-op.  'class Foo:' is
    just fine =)

Found 1 reports in 1 files in 0.12 seconds.
```

Run Fixit in an existing codebase

Ignore lint suggestions temporarily or permanently

- Lint suggestion silence
  - lint-fixme: when you plan to fix this later
  - lint-ignore: when you want to ignore this lint suggestion
  - Don’t use noqa: explicitly over implicitly
  - Examples:
    - # lint-fixme: NoInheritFromObjectRule: will fix in version 2
    - # lint-ignore: ComparePrimitivesByEqualRule, NoInheritFromObjectRule:

Run Fixit in an existing codebase

Commands

- run_rules

```
1 ! python -m fixit.cli.run_rules example.py --rules NoInheritFromObjectRule
```

- apply_fix

```
1 !python -m fixit.cli.apply_fix example.py --rules NoInheritFromObjectRule
```

- insert_suppressions

```
1 ! python -m fixit.cli.insert_suppressions NoInheritFromObjectRule example.py
```

```
@@ -1,13 +1,15 @@
 from typing import Dict


+# lint-fixme: NoInheritFromObjectRule: Inheriting from object is a no-op.  'class Foo:'
+# lint: is just fine =)
 class C(object):
     attr = "ab" "cd" "ef" "gh"
```

Documentation

Generated from source code

NoInheritFromObjectRule ClassName

In Python 3, a class is inherited from `object` by default. Explicitly inherit
redundant, so removing it keeps the code simpler. Docstring

Message MESSAGE

Inheriting from object is a no-op. 'class Foo:' is just fine =)

Has Autofix: Yes

VALID

VALID Code Examples

1:

```
class A(something):    pass
```

2:

```
class A:
    pass
```

INVALID

INVALID Code Examples

1:

```
class B(object):
    pass
```

Autofix:

```
---
+++
@@ -1,3 +1,3 @@

-class B(object):
+class B:
     pass
```

2:

```
class B(object, A):
    pass
```

Autofix:

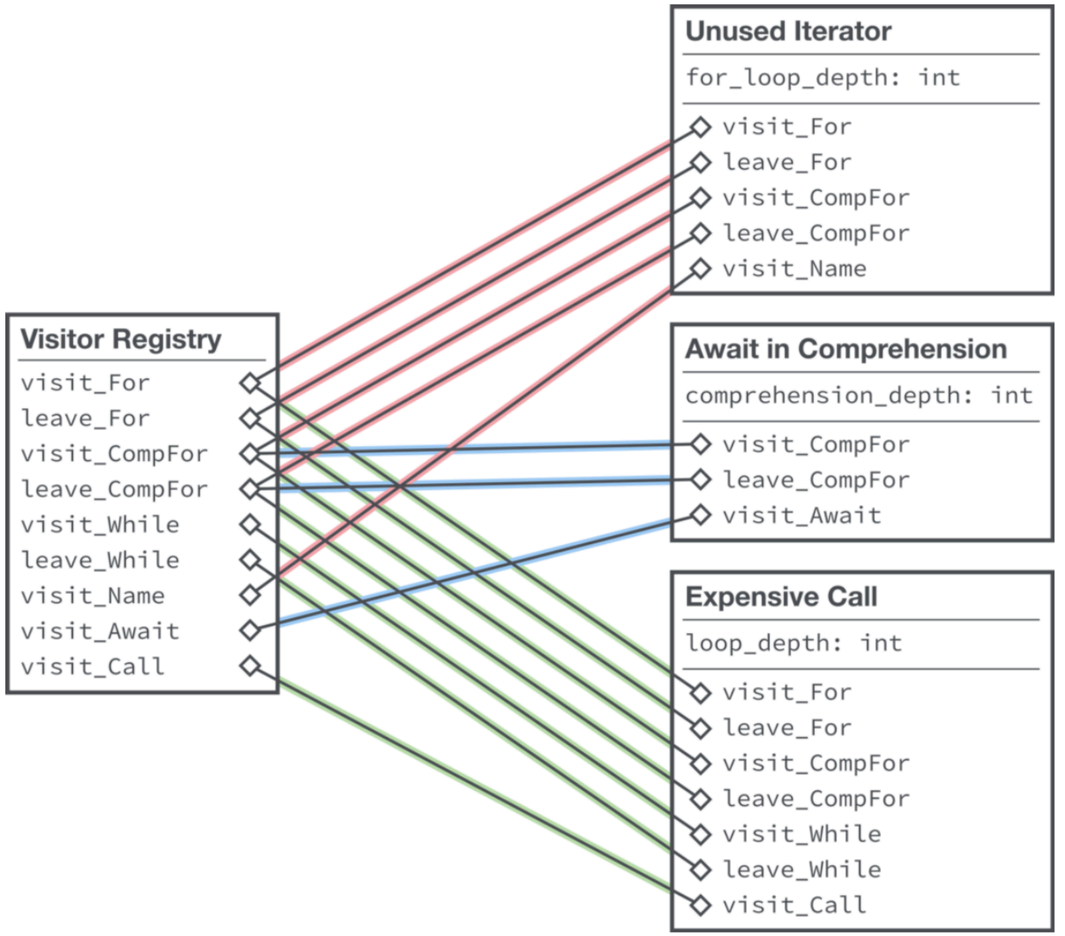
```
---
+++
@@ -1,3 +1,3 @@

-class B(object, A):
+class B(A):
     pass
```

https://fixit.readthedocs.io/en/latest/rules/NoInheritFromObjectRule.html

Efficiency

All lint rules run in a single syntax tree traversal

- Linter should run fast
- n rules, m source files
  - time complexity: O(n x m) => O(m)
- Batched Visitor with CstContext



UseFstringRule

https://fixit.readthedocs.io/en/latest/rules/UseFstringRule.html

- F-string is available in Python 3.6.
- It’s simpler and more efficient

```
@@ -1 +1 @@
-"a name: %s" % name
+f"a name: {name}"
```

```
-r"raw string value=%s" % val
+fr"raw string value={val}"
```

```
-"{%s" % val
+f"{{{val}"
```

RewriteToComprehensionRule

https://fixit.readthedocs.io/en/latest/rules/RewriteToComprehensionRule.html

- Literal comprehension is simpler and more efficient

```
-list(val for row in matrix for val in row)
+[val for row in matrix for val in row]
```

```
-dict((x, f(x)) for val in iterable)
+{x: f(x) for val in iterable}
```

```
-set([val for val in iterable])
+{val for val in iterable}
```

ImportConstraintsRule

https://fixit.readthedocs.io/en/latest/rules/ImportConstraintsRule.html

- Don’t allow import some package in some directory.
  - E.g. forbid importing testing code in production code

INVALID

config:

```
rule_config:
  ImportConstraintsRule:
    some_dir:
      rules:
      - - '*'
        - deny
```

path: `some_dir/file.py`

```
import common
```

VALID

config:

```
rule_config:
  ImportConstraintsRule:
    some_dir:
      rules:
      - - common
        - allow
      - - '*'
        - deny
```

path: `some_dir/file.py`

```
import common
```

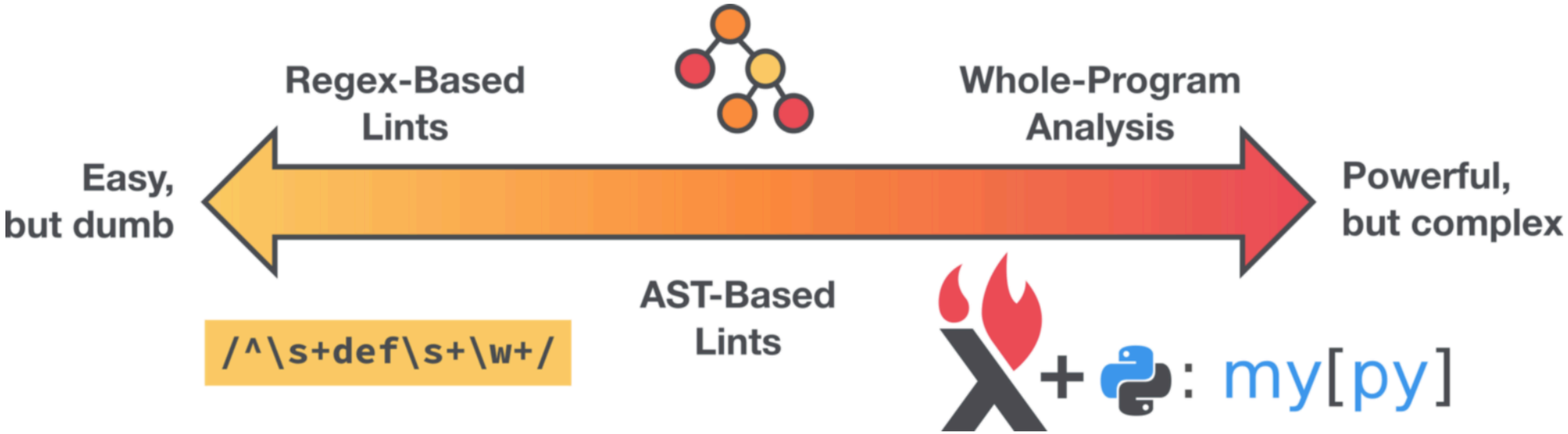
Apply all fixes

```
1 !python -m fixit.cli.apply_fix example.py
```

```
@@ -1,14 +1,14 @@
 from typing import Dict


-class C(object):
-    attr = "ab" "cd" "ef" "gh"
+class C:
+    attr = "ab" + "cd" + "ef" + "gh"

     def method(self) -> Dict[int, str]:
         filtered_char = []
         for char in self.attr:
-            if char is not "a":
+            if char != "a":
                 filtered_char.append(char)

-        index_to_char = dict([(idx, char) for idx, char in enumerate(filtered_char)])
+        index_to_char = {idx: char for idx, char in enumerate(filtered_char)}
         return index_to_char
```

Advanced Lint Rule - Static Analysis

- Source syntax tree only provide limited information



Fixit integrates with Pyre, a type checker

AwaitAsyncCallRule

- Get inferred type of each Name from Pyre and check if an await is missing

```
async def foo() -> bool: pass
if foo():
    do_stuff()
```

Autofix:

```
---
+++
@@ -1,4 +1,4 @@

 async def foo() -> bool: pass
-if foo():
+if await foo():
     do_stuff()
```

```
async def foo(): pass
async def bar():
    foo()
```

Autofix:

```
---
+++
@@ -1,4 +1,4 @@

 async def foo(): pass
 async def bar():
-    foo()
+    await foo()
```

Contributing to Fixit

https://fixit.readthedocs.io/en/latest/contributing.html

- You’re very welcome to contribute to Fixit
- New Autofixer Rules that help developers write simpler, safer and more efficient code.
- IDE (e.g. VSCode, PyCharm) and workflow (e.g. Github action) integration to build Fixit into development process for better usability.
- Bug and document fixes.
- New feature requests.
- Simply submit a Pull Request or an Issue.

Q&A

https://fixit.readthedocs.io/en/latest/index.html

https://github.com/Instagram/Fixit

https://www.facebook.com/careers/