



DCC
DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC3745 - TESTING

2023 - 1º SEMESTRE

UNIDAD 3 – CLASE PRÁCTICA:

- AST module
- Visitando y transformando el AST
- Warning Rules
- Transformation Rules

Alison Fernandez Blanco



UNIDAD 3 – CLASE PRÁCTICA:

- **AST module**
- Visitando el AST
- Warning Rules
- Transformation Rules

El módulo AST de Python ayuda a construir, recorrer y modificar árboles de sintaxis abstracta a partir de código Python.

Más información:

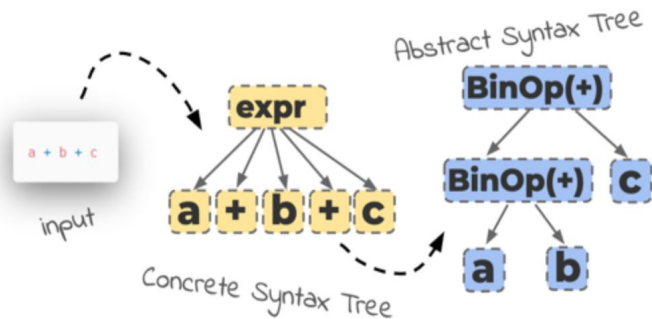
<https://docs.python.org/3.9/library/ast.html>

```
-- BoolOp() can use left & right?
expr = BoolOp(boolop op, expr* values)
| NamedExpr(expr target, expr value)
| BinOp(expr left, operator op, expr right)
| UnaryOp(unaryop op, expr operand)
| Lambda(arguments args, expr body)
| IfExp(expr test, expr body, expr orelse)
| Dict(expr* keys, expr* values)
| Set(expr* elts)
| ListComp(expr elt, comprehension* generators)
| SetComp(expr elt, comprehension* generators)
| DictComp(expr key, expr value, comprehension* generators)
| GeneratorExp(expr elt, comprehension* generators)
-- the grammar constrains where yield expressions can occur
| Await(expr value)
| Yield(expr? value)
| YieldFrom(expr value)
-- need sequences for compare to distinguish between
-- x < 4 < 3 and (x < 4) < 3
| Compare(expr left, cmpop* ops, expr* comparators)
| Call(expr func, expr* args, keyword* keywords)
| FormattedValue(expr value, int? conversion, expr? format_spec)
| JoinedStr(expr* values)
| Constant(constant value, string? kind)

-- the following expression can appear in assignment context
| Attribute(expr value, identifier attr, expr_context ctx)
| Subscript(expr value, expr slice, expr_context ctx)
| Starred(expr value, expr_context ctx)
| Name(identifier id, expr_context ctx)
| List(expr* elts, expr_context ctx)
| Tuple(expr* elts, expr_context ctx)
```

Funciones importantes:

- ❖ **ast.parse(source)**: Parsea el código fuente (source) en un AST.
- ❖ **ast.dump(tree)**: Devuelve una cadena (string) formateada en base al árbol AST (tree). Muy útil para depurar.
- ❖ **ast.unparse(tree)**: Genera una cadena que contiene el código respectivo que produce un AST.
- ❖ **ast.fix_missing_locations(node)**: Permite recursivamente actualizar la línea de código y la tabulación de un nodo del AST.



5

AST module

Code:

```
from ast import *  
  
print(dump(parse("1 + 1"), indent = 4))
```

Output:

```
Module(  
  body=[  
    Expr(  
      value=BinOp(  
        left=Constant(value=1),  
        op=Add(),  
        right=Constant(value=1)))],  
  type_ignores=[])
```



Visualizando AST con showast

showast **solo** funciona en Jupyter notebooks.

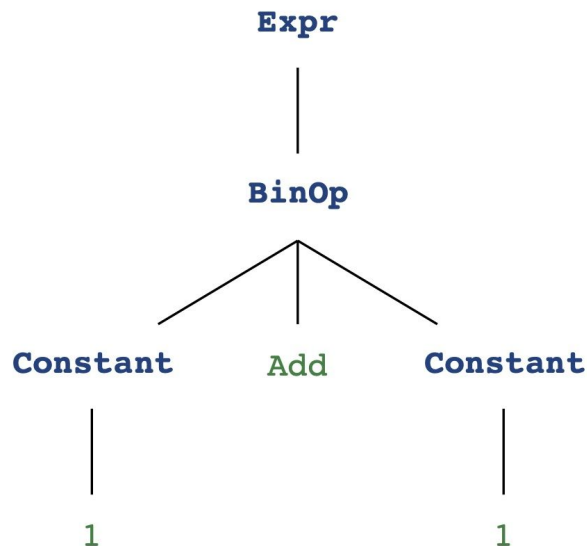
Más información:

https://github.com/hchasestevens/show_ast

```
import showast
import ast

tree = ast.parse("1+1")

showast.show_ast(tree)
```



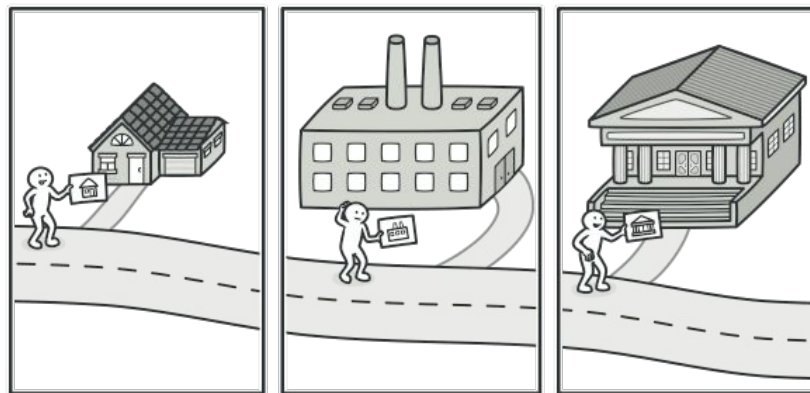


UNIDAD 3 – CLASE PRÁCTICA:

- AST module
- **Visitando y transformando el AST**
- Warning Rules
- Transformation Rules

ast.NodeVisitor: Una clase que permite recorrer el AST al llamar a una función visitante para cada nodo.

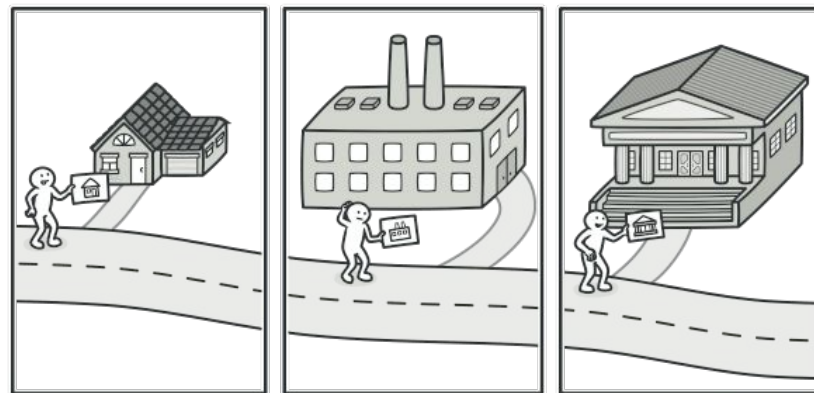
- ❖ **visit_<classname>(node):** Visita un nodo cuyo nombre de clase sea classname.
- ❖ **generic_visit(node):** Llama a visit() en todos los nodos hijo del nodo (node).



10 Transformando el AST

ast.NodeTransformer: Una subclase de **NodeVisitor** que permite recorrer el AST y modificar nodos. El retorno de la llamada a **visit()** es el nuevo valor del nodo.

- ❖ **visit_classname(node):** Visita un nodo cuyo nombre de clase sea *classname*.
- ❖ **generic_visit(node):** Llama a **visit()** en todos los nodos hijo del nodo (*node*).





UNIDAD 3 – CLASE PRÁCTICA:

- AST module
- Visitando y transformando el AST
- **Warning Rules**
- Transformation Rules

Warning Rules

Objetivo: Dado el código de un archivo Python, generar advertencias por problemas de estilo o problemas menores de programación. Las advertencias deben indicar el número de línea, el tipo de advertencia y una descripción.

Código: Los ejemplos que veremos en clases y la estructura base están en el folder rules del código base de la Tarea 2.

Nota: Tenga en cuenta que los archivos a ser analizados para testear los ejemplos de clase están en el folder input-code.

Warning Rules - Estructura Base

La estructura base contiene los siguientes archivos:

- ❖ **rules/__init__.py**: Archivo donde se realizan las configuraciones para importar módulos a partir de las reglas de warning.
- ❖ **rules/rule.py**: Archivo donde están presentes las clases **Rule**, **Warning** y **WarningNodeVisitor**.

14 Warning Rules - Configuración

rules/__init__.py: Archivo donde se realizan las configuraciones para importar módulos a partir de rules. En el caso de crear otras reglas, necesitan agregar una línea para importar las reglas creadas.

```
from .rule import *
from .eval_used import *
from .uncouple_method import *
from .dummy_if import *
from .uninitialized_attribute import *
from .many_arguments import *
```

15 Warning Rules - Clases básicas

`rules/rule.py`: Archivo donde están presentes las clases `Rule`, `Warning` y `WarningNodeVisitor`.

```
class Rule:

    def __init__(self):
        self.warningsList = []

    def analyze(self, ast):
        pass

    # Debe retornar una lista de objetos warnings
    def warnings(self):
        return self.warningsList
```

16 Warning Rules - Clases básicas

`rules/rule.py`: Archivo donde están presentes las clases `Rule`, `Warning` y `WarningNodeVisitor`.

```
class Warning:

    def __init__(self, name, line, description):
        self.name = name
        self.lineNumber = line
        self.description = description

    # Sobreescribe __str__ para dar una representación de cadena de Warning
    def __str__(self):
        return "[ Line " + str(self.lineNumber) + " ] " +
            self.name + " - " + self.description
```


17 Warning Rules - Clases básicas

`rules/rule.py`: Archivo donde están presentes las clases `Rule`, `Warning` y `WarningNodeVisitor`.

```
class WarningNodeVisitor(NodeVisitor):  
  
    def __init__(self, name, line, description):  
        self.warnings = []  
  
    def addWarning(self, name, lineo, description):  
        self.warnings.append(Warning(name, lineo, description))  
  
    def warningsList(self):  
        return self.warnings
```

18 Warning Rules - Ejemplos

Dado el código de un archivo Python, genera advertencias por problemas de estilo o problemas menores de programación. Por ejemplo:

- ❖ **Uso de if con constante "True"**
- ❖ **Uso de eval()**
- ❖ Atributos sin inicializar
- ❖ Uso de demasiados argumentos
- ❖ Métodos desacoplados.

19 Warning Rules - Dummy If

Uso de if con constante "True": A veces se usa 'if' para una constante que es verdadera y esta comparación es innecesaria porque siempre se ejecuta.

Código a analizar:

Generar advertencia ➡

```
def example1():  
    if True:  
        print("Example 1")
```

Generar advertencia ➡

```
def greaterThan100(x):  
    if (x > 100):  
        if True:  
            print(x + " is greater than 100")  
    else:  
        print(x + " is less than 100")
```

20 Warning Rules - Dummy If

Creamos la clase `DummyIfRule` para detectar los Dummy If y generar las advertencias.

Heredamos de `Rule`



Usamos nuestro propio visitor para generar advertencias.



```
class DummyIfRule(Rule):  
  
    def analyze(self, ast):  
        visitor = DummyIfVisitor()  
        visitor.visit(ast)  
        return visitor.warningsList()
```

21 Warning Rules - Dummy If

Creamos la clase `DummyIfVisitor` para agregar un warning cada vez que visitamos un nodo `If` cuyo `test.value` es igual a `True` (revisar definición de `ast module` en Python).

Heredamos de
`WarningNodeVisitor`



Verificamos que existe If True



Agregamos warning



Visitamos de manera general
los nodos hijos de `node`.



```
class DummyIfVisitor(WarningNodeVisitor):  
  
    def visit_If(self, node: If):  
        if isinstance(node.test, Constant):  
            if node.test.value == True:  
                self.addWarning('DummyIfWarning',  
                                node.lineno,  
                                'this if does not have any sense!')  
        NodeVisitor.generic_visit(self, node)
```

Abre consola o terminal y ejecuta el test correspondiente:

```
python test_clases.py TestWarnings.test_dummy_if
```



23 Warning Rules - Eval used

A veces se usa la función 'eval' con expresiones de fuentes poco fiables y esto es poco seguro.

Código a analizar:

Generar advertencia



```
eval("2")
```

Generar advertencia



```
def example2():  
    eval("2+2")
```

Generar advertencia



```
class Person:  
    def doit(self):  
        eval("[1,2,3]")
```

24 Warning Rules - Eval Used

Creamos la clase **EvalUsedRule** para detectar cuando se llama a la función 'eval' y generar las advertencias.

Heredamos de **Rule**



Usamos nuestro propio visitor para generar advertencias.



```
class EvalUsedRule(Rule):  
  
    def analyze(self, ast):  
        visitor = EvalVisitor()  
        visitor.visit(ast)  
        return visitor.warningsList()
```


25 Warning Rules - Eval Used

Creamos la clase **EvalVisitor** para agregar un warning cada vez que visitamos un nodo **Call** cuyo **func.id** es igual a **eval** (revisar definición de **ast module** en Python).

Heredamos de
WarningNodeVisitor



Verificamos si se llama a eval
Agregamos warning



Visitamos de manera general
los nodos hijos de **node**.



```
class EvalVisitor(WarningNodeVisitor):  
  
    def visit_Call(self, node: Call):  
        if node.func.id == 'eval':  
            self.addWarning('EvalWarning',  
                            node.lineno,  
                            'eval should not be used!!')  
        NodeVisitor.generic_visit(self, node)
```

Abre consola o terminal y ejecuta el test correspondiente:

```
python test_clases.py TestWarnings.test_eval_used
```



27 Warning Rules - Ejemplos

Dado el código de un archivo Python, genera advertencias por problemas de estilo o problemas menores de programación. Por ejemplo:

- ❖ Uso de if con constante "True"
- ❖ Uso de eval()
- ❖ **Atributos sin inicializar**
- ❖ **Uso de demasiados argumentos**
- ❖ **Métodos desacoplados**

Nota: Revisen el código respectivo, ya que les puede ser de utilidad para la Tarea 2.



UNIDAD 3 – CLASE PRÁCTICA:

- AST module
- Visitando el AST
- Warning Rules
- **Transformation Rules**

Transformation Rules

Objetivo: Dado el código de un archivo Python, transforma el código al encontrar infracciones de las “buenas prácticas” y entrega el AST transformado.

Código: Los ejemplos que veremos en clases y la estructura base están en el folder `rewriter` del código base de la Tarea 2.

Nota: Tenga en cuenta que los archivos a ser analizados para testear los ejemplos de clase están en el folder `input-code` y los archivos con la transformación esperada están en el folder `expected-code`.

Transformation Rules - Estructura Base

La estructura base contiene los siguientes archivos:

- ❖ **rewriter/__init__.py**: Archivo donde se realizan las configuraciones para importar módulos a partir de las reglas de transformación.
- ❖ **rewriter/rewriter.py**: Archivo donde está definida la clase **RewriterCommand**.

Transformation Rules - Configuración

rewriter/__init__.py: Archivo donde se realizan las configuraciones para importar módulos a partir de las reglas de transformación. En el caso de crear otras reglas de transformación, necesitan agregar una línea para importar las reglas creadas.

```
from .rewriter import *  
from .eval_rewriter import *  
from .if_true_rewriter import *
```

32 Transformation Rules - Clase básica

rewriter/rewriter.py: Archivo donde está definida la clase **RewriterCommand**.

```
class RewriterCommand:

    def apply(self, ast):
        # Por defecto retorna el mismo AST sin ninguna modificación
        return ast
```


Transformation Rules - Ejemplos

Dado el código de un archivo Python, transforma el código al encontrar infracciones de las “buenas prácticas” y entrega el AST transformado. Por ejemplo:

- ❖ Eliminación de if con constante “True”
- ❖ En vez de `eval()` use `literal_eval()`

Transformation Rules - If True Rewriter

Transformar el código que contiene `if True` al eliminar el `if` innecesario. Considere solo usar el body del `if` si la condición es la constante `True`.

Código a analizar

```
def example1():  
    if True:  
        print("Example 1")  
  
def greaterThan100(x):  
    if (x > 100):  
        if True:  
            print(x + " is greater than 100")  
    else:  
        print(x + " is less than 100")
```



Código transformado

```
def example1():  
    print("Example 1")  
  
def greaterThan100(x):  
    if (x > 100):  
        print(x + " is greater than 100")  
    else:  
        print(x + " is less than 100")
```



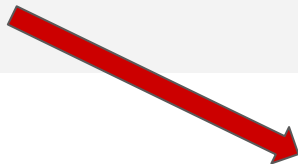
Transformation Rules - If True Rewriter

Creamos la clase `IfTrueRewriterCommand` para analizar el ast mediante nuestro transformador `IfTrueTransformer` y retornamos el nuevo AST transformado.

```
class IfTrueRewriterCommand(RewriterCommand):  
  
    def apply(self, ast):  
        new_tree = fix_missing_locations(IfTrueTransformer().visit(ast))  
        return new_tree
```



Heredamos de
`RewriterCommand`



Recorre los nodos del AST transformado y actualiza ciertos atributos (e.g., número de línea) considerando las modificaciones.

Transformation Rules - If True Rewriter

Creamos la clase `IfTrueTransformer` para modificar cada vez que visitamos un nodo `If` cuyo `test.value` es igual a `True` por solo el cuerpo del `If` (revisar definición de `ast` module en Python).

Heredamos de `NodeTransformer`

→

```
class IfTrueTransformer(NodeTransformer):
```

Transforma todos los hijos del nodo.

→

```
def visit_If(self, node: If):  
    NodeTransformer.generic_visit(self, node)
```

`statements` es el nodo con hijos transformados.

→

```
statements = node
```

Detectamos si hay `if True`.

→

```
if isinstance(node.test, Constant):  
    if node.test.value == True:
```

Actualizamos que el valor de `statements` con el cuerpo del nodo.

→

```
statements = node.body  
return statements
```

Abre consola o terminal y ejecuta el test correspondiente:

```
python test_clases.py TestTransformers.test_if_true_rewriter
```



Transformation Rules - Eval Rewriter

Transformar el código que contiene la llamada a 'eval' por 'literal_eval' para evaluar de manera más segura las cadenas.

Código a analizar

```
eval("2")

def example2():
    eval("2+2")

class Person:
    def doit(self):
        eval("[1,2,3]")
```



Código transformado

```
literal_eval("2")

def example2():
    literal_eval("2+2")

class Person:
    def doit(self):
        literal_eval("[1,2,3]")
```

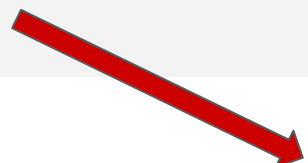


Transformation Rules - Eval Rewriter

Creamos la clase `LiteralEvalRewriterCommand` para analizar el ast mediante nuestro transformador `LiteralEvalTransformer` y retornamos el nuevo AST transformado.

```
class LiteralEvalRewriterCommand(RewriterCommand):  
  
    def apply(self, ast):  
        new_tree = fix_missing_locations(LiteralEvalTransformer().visit(ast))  
        return new_tree
```

Heredamos de
`RewriterCommand`



Recorre los nodos del AST transformado y actualiza ciertos atributos (e.g., número de línea) considerando las modificaciones.

Transformation Rules - Eval Rewriter

Creamos la clase `LiteralEvalTransformer` para modificar cada vez que visitamos un nodo `Call` cuyo `func.id` es igual a `'eval'` por un nodo `Call` cuyo `func.id` es igual a `'literal_eval'`.

Heredamos de `NodeTransformer`

Detectamos si se llama a `eval`

Retornamos un nodo `Call` a `literal_eval`

```
class LiteralEvalTransformer(NodeTransformer):  
  
    def visit_Call(self, node: Call):  
        if node.func.id == 'eval':  
            return Call(func=Name(id='literal_eval',  
                                   ctx=Load()),  
                        args=node.args,  
                        keywords=node.keywords)  
        else:  
            return node
```


Hora de probar

Abre consola o terminal y ejecuta el test correspondiente:

```
python test_clases.py TestTransformers.test_eval_rewriter
```



- ❖ Unit testing framework (<https://docs.python.org/3/library/unittest.html>)
- ❖ Design patterns (<https://refactoring.guru/design-patterns>)
- ❖ AST module (https://docs.python.org/3.9/library/ast.html#ast.fix_missing_locations)
- ❖ showast module (https://github.com/hchasestevens/show_ast)
- ❖ The missing Python AST docs (<https://greentreesnakes.readthedocs.io/en/latest/>)

¿Consultas?