Bicycle Repair Man

Feb 02, 2022

Contents

BRM is a short way of rewriting python source with transformation of tokens. It offers a high level interface for token transformation with automating most of manual stuff.

CHAPTER 1

Features

- On the fly token transformation
- Auto location padding for single line transformations
- Ways to patch standard tokenizer with custom tokens and capturing them
- Grammar Rules like pattern syntax for easy capturing

CHAPTER 2

Installation

Install BRM by running:

```
pip install brm
```

or

```
pip install git+https://github.com/isidentical/BRM.git
```

CHAPTER 3

Tutorials

3.1 Let's Build Our First Token Transformer

BRM is about rewriting and transforming python sources. It offers you to roundtrip back to what it was earlier by the power of tokens and comfort of `Transformer` interfaces. This tutorial we are going to focus on rewriting a square root character () transformer. We'll handle all forms of square root operation like `9` or `16.0` and then transform them to `<number> ** 0.5`.

3.1.1 Creating First Transformer

A Transformer is a class that provides utilities and dispatching for tokens. For an example you can create a class and do nothing, just listen what it gets. Lets write one that listens numbers.

First of all you need to subclass `TokenTransformer` in order to add new methods. Let's do that

```
from brm import TokenTransformer

class NumberHandler(TokenTransformer):
```

For registering specific token types, you need to define a function with `visit_<token-type>`. It is like the `ast.NodeTransformer` but instead of nodes we use token types. You can get the name of all token types by checking `token` module and it's docs. Our token is `NUMBER`. If you want to see which tokens a python expression or statement consists from you can do it in interactive shell by instantiating `TokenTransformer` and calling `quick_tokenize` on it.

```
>>> import brm
>>> transformer = brm.TokenTransformer()
>>> transformer.quick_tokenize("1.0")
[TokenInfo(type=2 (NUMBER), string='1.0', start=(1, 0), end=(1, 3), line='1.0')]
>>> transformer.quick_tokenize("100")
[TokenInfo(type=2 (NUMBER), string='100', start=(1, 0), end=(1, 3), line='100')]
>>> pprint.pprint(transformer.quick_tokenize("100 + 100"))
```

(continues on next page)

(continued from previous page)

```
[TokenInfo(type=2 (NUMBER), string='100', start=(1, 0), end=(1, 3), line='100 + 100'),
 TokenInfo(type=54 (OP), string='+', start=(1, 4), end=(1, 5), line='100 + 100'),
 TokenInfo(type=2 (NUMBER), string='100', start=(1, 6), end=(1, 9), line='100 + 100')]
```

Let's add a `visit_number` method and see what happens to our `NumberHandler` when we call it with some numbers.

```
def visit_number(self, number):
    print("is a number? (always yes)", number.type == token.NUMBER)
    print("what it contains?", number.string)
    print("where it starts?", "y_start={}, x_start={}".format(*number.start))
    print("where it end?", "y_end={}, x_end={}".format(*number.end))
```

After this definition, we are going to instantiate our `NumberHandler` and call `transform` on it. `transform` method is responsible for everything related to source rewriting. It takes python source as a string (like what you read from the python file) and then it registers new tokens if they are available (like we are going to register our square root token in this step) it continues with invoking transformer methods like the one we just created (*visit_number*), if there are no defined transformer methods for that token, it calls `dummy`. You can probably implement something like that to your subclass for watching undefined nodes.

```
def dummy(self, unknown_token):
    print("Unhandled token:", unknown_token)
```

Let's test our `visit_number` and `dummy` out.

```
>>> number_handler = NumberHandler()
>>> number_handler.transform("2")
    is a number? (always yes) True
    what it contains? 2
    where it starts? y_start=1, x_start=0
    where it end? y_end=1, x_end=1
    Unhandled token: TokenInfo(type=4 (NEWLINE), string='', start=(1, 1), end=(1, 2),␣
↪line='')
    Unhandled token: TokenInfo(type=0 (ENDMARKER), string='', start=(2, 0), end=(2,␣
↪0), line='')
'2'
```

It took `"2"` and returned `"2"`, it also answered our questions. Now what? What are those unhandled tokens? One of them is just a newline token, which its name states. The other one is a marker token which indicates we reached end of input. We can actually check that with `token.ISEOF` in our `dummy` function.

```
def dummy(self, unknown_token):
    if token.ISEOF(unknown_token.type):
        print("Reached EOF without a problem, congratz")
    else:
        print("Unhandled token:", unknown_token)

>>> number_handler = NumberHandler()
>>> number_handler.transform("2")
is a number? (always yes) True
what it contains? 2
which line it was taken 2
where it starts? y_start=1, x_start=0
where it end? y_end=1, x_end=1
Unhandled token: TokenInfo(type=4 (NEWLINE), string='', start=(1, 1), end=(1, 2),␣
↪line='')
```

(continues on next page)

(continued from previous page)

```
Reached EOF without a problem, congratz
'2'
```

CHAPTER 4

Contribute

- Issue Tracker: github.com/isidentical/brm/issues
- Source Code: github.com/isidentical/brm

CHAPTER 5

Support

If you are having issues, please let us know.