Pyalect Documentation

Release 0.1.0

Ryan Morshead

May 25, 2020

CONTENTS

A dynamic dialect transpiler for Python that you can install with `pip`!

```
pip install pyalect
```

Note: Pyalect is still young! If you have ideas or find a problem let us know!

CHAPTER

ONE

BUT WHY?

Now why would you want to transpile Python one might ask? Well a transpiler probably isn't the first tool you should reach for when trying to solve a problem, but sometime's it's the only option. For example, sometimes the easiest way to express a given bit of logic turns out to be sub-optimal when it comes to performance - in this situation you could use a transpiler optimize your expressive, but inefficient code. On the otherhand you may have purely assthetic reasons for using a transpiler to transform pretty, but invalid syntax into an uglier, but valid form so that it can be executed. For example, one might want to transpile the following HTM style string template:

```
# dialect=html
dom = html"<div height=10px><p>hello!</p></div>"
```

Into valid Python:

```
# dialect=html
dom = html("div", {"height": "10px"}, [html("p", {}, ["hello!"])])
```

CHAPTER

TWO

BASIC USAGE

So what would it look like to implement the custom syntax above? All you need to do is register a "dialect" transpiler with Pyalect before importing the module containing the code in question. Consider the following directory structure:

```
my_project/
|-  entrypoint.py
|-  my_html_module.py
```

Inside `entrypoint.py` should be a *`Dialect`* that implements two methods:

```
from pyalect import Dialect

class HtmlDialect(Dialect, name="html"):
    def transform_src(source: str) -> str:
        """Called first to manipulate the module source as a string"""
        # your code goes here...

    def transform_ast(tree: ast.AST) -> ast.AST:
        """Called second to change the AST of the now transformed source"""
        # your code goes here...

import my_html_module
```

Where `my_html_module` is a normal Python file with a dialect header comment:

```
# dialect=html
dom = html"<div height=10px><p>hello!</p></div>"
```

With all this in place and the methods of `HtmlDialect` implemented you should be able to run `entrypoint.py` in your console to find `my_html_module` has been transpiler just before execution:

```
python entrypoint.py
```

Note: For a real world example implementation of an HTML transpiler check out IDOM!

CHAPTER

THREE

INTEGRATIONS

In most situations Pyalect should work out of the box, but some tools require special support.

3.1 IPython and Jupyter

Dialects are supported in IPython and Jupyter via magics:

```
%%dialect html
...
```

3.2 Pytest Asserts

Similarly to Pyalect, Pytest uses import hooks to transpile code at import-time. Since Pyalect's own import hook should take priority over Pytest's you'll have to import the builtin `pytest` dialect and include it in any test files where you're using your own dialects:

```
import pyalect.builtins.pytest
```

```
# dialect = my_dialect, pytest

def test_my_code():
    assert ...
```

CHAPTER

FOUR

API

`pyalect.dialect.`**`find_file_dialects`**(*filename*)

Find dialects in the source of the file at the given path.

See `find_source_dialects()` for more info.

Return type `List[str]`

`pyalect.dialect.`**`find_source_dialects`**(*source*)

Extract dialect from comment headers in module source code.

The comment should be of the form `# dialect=my_dialect` and must be before the first non-continuation newline.

Examples

```
# dialect=my_dialect
```

```
# coding=utf-8
# dialect=my_dialect
'''docstring'''
```

Return type `List[str]`

class `pyalect.dialect.`**`Dialect`**(*filename=None*)

Bases: `object`

A base class for defining a dialect transpiler.

The logic of transpiling can be roughly paraphrased as:

```
import ast

transpiler = MyDialect("my_module.py")
source = read_file_source()
new_source = transpiler.transform_src(source)
tree = ast.parse(new_source)
new_tree = transpiler.transform_ast(tree)

exec(compile(new_tree, "my_module.py", "exec"))
```

Note: A transpiler instance is only used **once** per module and **shouldn't** be reused. This means that a `Dialect` can keep state between calls to `Dialect.transform_src()` and `Dialect.transform_ast()`

Parameters **filename** (Optional[str]) – the name of the file being transpiled.

transform_src(*source*)
Implement this method to transform a raw Python source string.

Return type str

transform_ast(*node*)
Implement this method to transform an AST.

Return type AST

class pyalect.dialect.**DialectReducer**(*dialects*)
Bases: collections.abc.Sequence, typing.Generic

A reducer for applying many dialects at once.

It acts like a typing.Sequence, but with the same interface as a *Dialect* which makes it easy to work with.

transform_src(*source*)
Transform raw Python source code using the contained dialects.

Return type str

transform_ast(*node*)
Transform an AST tree using the contained dialects.

Return type AST

count(*value*) → integer – return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

pyalect.dialect.**apply_dialects**(*source*, *names*, *filename=None*)
Utility for applying dialect transpilers to source code.

Return type AST

pyalect.dialect.**dialect_reducer**(*names*, *filename=None*)
Get a *DialectReducer*

Examples

There's a couple different ways to create the reducer.

```
dialect_reducer("d1")
dialect_reducer("d1, d2, d3")
dialect_reducer(["d1", "d2", "d3"])
```

Return type *DialectReducer*

pyalect.dialect.**dialect**(*name*, *filename*)
Instantiate a dialect for use on the given file.

Parameters

- **name** (str) – The dialect name
- **filename** (Optional[str]) – The name of the file the *Dialect* will be used on.

Return type *Dialect*

`pyalect.dialect.registered()`

The set of dialect names already registered.

Return type Set[str]

`pyalect.dialect.register(dialect)`

Register a *Dialect* so it will be applied to imported modules.

Return type Type[*Dialect*]

`pyalect.dialect.deregister(*dialects)`

Deregister one or more *Dialect* classes.

Parameters **dialects** (Union[Type[*Dialect*], Iterable[str], str]) – the dialect name, or class

Return type None

PYTHON MODULE INDEX

p

INDEX