codegrapher Documentation

Release 0.2.1

Laura Rupprecht

January 08, 2017

Contents

Contents:

CHAPTER 1

Introduction to Codegrapher

1.1 codegrapher

1.1.1 Code that graphs code

Uses the python AST to parse Python source code and build a call graph.

1.1.2 Output

An example of the current output of the parser parsing itself.

1.1.3 Installation

```
pip install codegrapher
```

To generate graphs, graphviz must be installed.

1.1.4 Usage

At the command line

To parse a file and output results to the console:

```
codegrapher path/to/file.py --printed
```

To parse a file and output results to a file:

```
codegrapher path/to/file.py --output output_file_name --output-type png
```

To analyze a directory of files, along with all files it contains:

```
codegrapher -r path/to/directory --output multiple_file_analysis
```

And if you have a list of functions that aren't useful in your graph, add it to a *.cg_ignore* file:

```
# cg_ignore file
# all lines beginning with '#' are ignored

# every function calls this, so it's not helpful in my graph:
```

```
log_error

# I don't want to see this in my graph:
parse
lower
```

Then add the *–ignore* flag to your command. Using the flag *–remove-builtins* provides the same functionality for ignoring items found in *__builtins__*.

As a Python module

To easily parse code in Python :

```
from codegrapher.parser import FileObject

file_object = FileObject('path/to/file.py')
file_object.visit()
```

And then to add that code to a graph and render it (using graphviz):

```
from codegrapher.graph import FunctionGrapher

graph = FunctionGrapher()
graph.add_file_to_graph(file_object)
graph.name = 'name.gv'
graph.format = 'png'
graph.render()
```

Which will produce your code as a png file, *name.gv.png*, along with a dot file *name.gv*

More documentation for the Python module can be found at Read the Docs.

CHAPTER 2

codegrapher package

2.1 Submodules

2.2 codegrapher.graph module

exception codegrapher.graph.**FilenameNotSpecifiedException**
Bases: exceptions.Exception

An exception raised when a file name is not specified in a FunctionGrapher instance before calling FunctionGrapher.render() on it.

class codegrapher.graph.**FunctionGrapher**
Bases: object

FunctionGrapher is a class for producing graphviz graphs showing the call graph for sets of classes.

name
string

Name to be used when a graph is made.

nodes
set

Graphviz nodes to be graphed.

edges
set

Directional edges connecting one node to another.

format
string

File format for graph. Default is *pdf*.

add_classes_to_graph(*classes*, *relative_namespace*)
Adds classes with constructors to the set. This adds edges between a class constructor and the methods called on those items.

Parameters

- **classes** (*list*) – list of codegrapher.parser.ClassObject items.
- **relative_namespace** (*string*) – namespace of the current class.

add_dict_to_graph(*class_names*, *dictionary*, *relative_namespace*)
Creates a list of nodes and edges to be rendered. Deduplicates input.

Parameters

- **class_names** (*list*) – List of class names to be recognized by the graph as *class_name.__init__* nodes.
- **dictionary** (*dict*) – *ClassObject.call_tree* dict to be added to graph nodes and edges.
- **relative_namespace** (*string*) – Relative namespace for the current class, i.e. where the current class is located relative to the root, in dotted path notation.

add_file_to_graph(*file_object*)
When given a `codegrapher.parser.FileObject` object, this adds all classes to the current graph.

Parameters **file_object** (`codegrapher.parser.FileObject`) – Visitor objects to have all its classes added to the current graph.

render(*name=None*)

Renders the current graph. Graphviz must be installed for the graph to be rendered.

Parameters **name** (*string*) – filename to override *self.name*.

Raises `FilenameNotSpecifiedException` – If *FunctionGrapher.name* is not specified.

class `codegrapher.graph.`**Node**(*input_node*)
Bases: `object`

A class to more easily handle manipulations needed to properly display nodes in a graph. Optimized to handle nodes that represent functions in a program.

tuple
tuple

Contains the namespace, class, and function name for the current node. If namespace is an empty string, this contains just the class and function names. If a string is provided to the constructor this is a tuple containing just the function name.

represent
Provides a string representation of the current node

Returns (string): Dotted form of current node, as in *namespace.class.function_name*.

2.3 codegrapher.parser module

class `codegrapher.parser.`**CallInspector**
Bases: `ast.NodeVisitor`

Within a call, a Name or Attribute will provide the function name currently in use.

Identifies *Name* nodes, which are called as `name(args)`, and *Attribute* nodes, which are called as `object.attr(args)`

module
string

Current module name on which the current call is made.

identifier
string

Name of the function called.

visit_Attribute(*node*)

visit_Name(*node*)

class codegrapher.parser.**CallVisitor**(***kwargs*)
Bases: codegrapher.parser.ImportVisitor

Finds all calls present in the current scope and inspect them.

call_names
set

set of CallInspector.identifier items within current AST node.

calls
list

(module, identifier) items called within current AST node, with identifiers decoded form current alias, and modules expanded to their full import paths.

continue_parsing(*node*)

visit_Call(*node*)

class codegrapher.parser.**ClassObject**(*node=None*, *aliases=None*, *modules=None*)
Bases: object

Class for keeping track of classes in code.

modules
dict

dict of current modules with *alias: module_name*, *key:value pairs*.

aliases
dict

dict of current modules with *alias: original_name*, *key:value pairs*.

node
ast.AST

AST node for entire class.

name
string

Class name.

functions
list

FunctionObject items defined in the current class.

call_tree
dict

dict with *key:value* pairs *(module, FunctionObject.name): (module, identifier)*.

ignore_functions(*ignore_set*)
Ignores all functions matching those specified in a pre-defined ignore set.

Parameters **ignore_set** (*set*) – Functions whose calls should be removed (ignored) in the class call tree.

namespace(*relative_namespace*)
Take the relative namespace for the class and prepend it to each item defined in the current class.

Parameters **relative_namespace** (*string*) – Namespace to be prepended to each item in the call tree.

pprint()
Pretty print formatter for class object.

Returns string

remove_builtins()
For many classes, we may not want to include builtin functions in the graph. Remove builtins from the call tree and from called functions list.

visit()
Visits all the nodes within the current class AST node.

Updates *self.functions* and *self.call_tree* for the current instance.

class codegrapher.parser.**FileObject**(*file_name*, *modules=None*, *aliases=None*)
Bases: object

Class for keeping track of files.

modules
dict

dict of current modules with *alias: module_name*, *key:value pairs*.

aliases
dict

dict of current modules with *alias: original_name*, *key:value pairs*.

node
ast.AST

AST node for entire file.

name
string

File name.

classes
list

ClassObject items defined in the current file.

relative_namespace
string

The namespace for the current file, taken from the relative path of the current file

ignore
set

Functions to be ignored, as defined in a *.cg_ignore* text file.

add_ignore_file()
Use a file *.cg_ignore* to ignore a list of functions from the call graph

ignore_functions()
Ignore all functions in the current class which are present in the instance's *ignore* attribute.

namespace()
Programmatically change the name of items in the call tree so they have relative path information

remove_builtins()
Removes builtins from each class in a *FileObject* instance.

visit()
Visits all the nodes within the current file AST node.

Updates *self.classes* for the current instance.

class codegrapher.parser.**FileVisitor**(***kwargs*)
Bases: codegrapher.parser.ImportVisitor

First visitor that should be called on the file level.

classes
list

list of ClassObject instances defined in the current file.

continue_parsing(*node*)

remove_builtins()
Removes builtins from each class in a *FileVisitor* instance.

visit_ClassDef(*node*)

visit_Module(*node*)

class codegrapher.parser.**FunctionObject**(*node=None*, *aliases=None*, *modules=None*)
Bases: object

Object that stores information within a single function definition

modules
dict of current modules with *alias: module_name*, *key:value pairs*.

aliases
dict of current modules with *alias: original_name*, *key:value pairs*.

node
ast.AST

AST node for entire function.

name
string

function name.

calls
list

(module, identifier) tuples describing items called within current node, with identifiers decoded form current alias, and modules expanded to their full import paths.

decorator_list
list

list of decorators, by name as a string, applied to the current function definition.

is_classmethod
bool

True if the current function is designated as a classmethod by a decorator.

visit()
Visits all the nodes within the current function object's AST node.

Updates *self.calls*, *self.modules*, and *self.aliases* for the current instance.

class codegrapher.parser.**FunctionVisitor**(***kwargs*)
Bases: codegrapher.parser.ImportVisitor

Function definitions are where the function is defined, and the call is where the ast for that function exists.

This only looks for items that are called within the scope of a function, and associates those items with the function.

defined_functions
set

names of functions found by function visitor instance.

functions
list

FunctionObject instances found by function visitor instance.

calls
dict

mapping from function names defined to calls within that function definition.

continue_parsing(*node*)

visit_FunctionDef(*node*)

class codegrapher.parser.**ImportVisitor**(*aliases=None*, *modules=None*)
Bases: ast.NodeVisitor

For import related calls, store the source modules and aliases used. Designed to be inherited by other classes that need to know about imports in their current scope.

modules
dict

dict of current modules with *alias: module_name*, *key:value pairs*.

aliases
dict

dict of current modules with *alias: original_name*, *key:value pairs*.

continue_parsing(*node*)

visit_Import(*node*)

visit_ImportFrom(*node*)

2.4 Module contents

CHAPTER 3

codegrapher

3.1 codegrapher package

3.1.1 Submodules

3.1.2 codegrapher.graph module

exception codegrapher.graph.**FilenameNotSpecifiedException**
Bases: exceptions.Exception

An exception raised when a file name is not specified in a FunctionGrapher instance before calling FunctionGrapher.render() on it.

class codegrapher.graph.**FunctionGrapher**
Bases: object

FunctionGrapher is a class for producing graphviz graphs showing the call graph for sets of classes.

name
string

Name to be used when a graph is made.

nodes
set

Graphviz nodes to be graphed.

edges
set

Directional edges connecting one node to another.

format
string

File format for graph. Default is *pdf*.

add_classes_to_graph(*classes*, *relative_namespace*)
Adds classes with constructors to the set. This adds edges between a class constructor and the methods called on those items.

Parameters

- **classes** (*list*) – list of codegrapher.parser.ClassObject items.

- **relative_namespace** (*string*) – namespace of the current class.

add_dict_to_graph(*class_names*, *dictionary*, *relative_namespace*)
Creates a list of nodes and edges to be rendered. Deduplicates input.

Parameters

- **class_names** (*list*) – List of class names to be recognized by the graph as *class_name.__init__* nodes.
- **dictionary** (*dict*) – *ClassObject.call_tree* dict to be added to graph nodes and edges.
- **relative_namespace** (*string*) – Relative namespace for the current class, i.e. where the current class is located relative to the root, in dotted path notation.

add_file_to_graph(*file_object*)
When given a `codegrapher.parser.FileObject` object, this adds all classes to the current graph.

Parameters **file_object** (`codegrapher.parser.FileObject`) – Visitor objects to have all its classes added to the current graph.

render(*name=None*)

Renders the current graph. Graphviz must be installed for the graph to be rendered.

Parameters **name** (*string*) – filename to override *self.name*.

Raises `FilenameNotSpecifiedException` – If *FunctionGrapher.name* is not specified.

class `codegrapher.graph.`**Node**(*input_node*)
Bases: `object`

A class to more easily handle manipulations needed to properly display nodes in a graph. Optimized to handle nodes that represent functions in a program.

tuple
tuple

Contains the namespace, class, and function name for the current node. If namespace is an empty string, this contains just the class and function names. If a string is provided to the constructor this is a tuple containing just the function name.

represent
Provides a string representation of the current node

Returns (string): Dotted form of current node, as in *namespace.class.function_name*.

3.1.3 codegrapher.parser module

class `codegrapher.parser.`**CallInspector**
Bases: `ast.NodeVisitor`

Within a call, a Name or Attribute will provide the function name currently in use.

Identifies *Name* nodes, which are called as `name(args)`, and *Attribute* nodes, which are called as `object.attr(args)`

module
string

Current module name on which the current call is made.

identifier
string

Name of the function called.

visit_Attribute(*node*)

visit_Name(*node*)

class codegrapher.parser.**CallVisitor**(***kwargs*)
Bases: codegrapher.parser.ImportVisitor

Finds all calls present in the current scope and inspect them.

call_names
set

set of CallInspector.identifier items within current AST node.

calls
list

(module, identifier) items called within current AST node, with identifiers decoded form current alias, and modules expanded to their full import paths.

continue_parsing(*node*)

visit_Call(*node*)

class codegrapher.parser.**ClassObject**(*node=None*, *aliases=None*, *modules=None*)
Bases: object

Class for keeping track of classes in code.

modules
dict

dict of current modules with *alias: module_name*, *key:value pairs*.

aliases
dict

dict of current modules with *alias: original_name*, *key:value pairs*.

node
ast.AST

AST node for entire class.

name
string

Class name.

functions
list

FunctionObject items defined in the current class.

call_tree
dict

dict with *key:value* pairs *(module, FunctionObject.name): (module, identifier)*.

ignore_functions(*ignore_set*)
Ignores all functions matching those specified in a pre-defined ignore set.

Parameters **ignore_set** (*set*) – Functions whose calls should be removed (ignored) in the class call tree.

namespace(*relative_namespace*)
Take the relative namespace for the class and prepend it to each item defined in the current class.

Parameters **relative_namespace** (*string*) – Namespace to be prepended to each item in the call tree.

pprint()
Pretty print formatter for class object.

Returns string

remove_builtins()
For many classes, we may not want to include builtin functions in the graph. Remove builtins from the call tree and from called functions list.

visit()
Visits all the nodes within the current class AST node.

Updates *self.functions* and *self.call_tree* for the current instance.

class codegrapher.parser.**FileObject**(*file_name*, *modules=None*, *aliases=None*)
Bases: object

Class for keeping track of files.

modules
dict

dict of current modules with *alias: module_name*, *key:value pairs*.

aliases
dict

dict of current modules with *alias: original_name*, *key:value pairs*.

node
ast.AST

AST node for entire file.

name
string

File name.

classes
list

ClassObject items defined in the current file.

relative_namespace
string

The namespace for the current file, taken from the relative path of the current file

ignore
set

Functions to be ignored, as defined in a *.cg_ignore* text file.

add_ignore_file()
Use a file *.cg_ignore* to ignore a list of functions from the call graph

ignore_functions()
Ignore all functions in the current class which are present in the instance's *ignore* attribute.

namespace()
Programmatically change the name of items in the call tree so they have relative path information

remove_builtins()
Removes builtins from each class in a *FileObject* instance.

visit()
Visits all the nodes within the current file AST node.

Updates *self.classes* for the current instance.

class codegrapher.parser.**FileVisitor**(***kwargs*)
Bases: codegrapher.parser.ImportVisitor

First visitor that should be called on the file level.

classes
list

list of ClassObject instances defined in the current file.

continue_parsing(*node*)

remove_builtins()
Removes builtins from each class in a *FileVisitor* instance.

visit_ClassDef(*node*)

visit_Module(*node*)

class codegrapher.parser.**FunctionObject**(*node=None*, *aliases=None*, *modules=None*)
Bases: object

Object that stores information within a single function definition

modules
dict of current modules with *alias: module_name*, *key:value pairs*.

aliases
dict of current modules with *alias: original_name*, *key:value pairs*.

node
ast.AST

AST node for entire function.

name
string

function name.

calls
list

(module, identifier) tuples describing items called within current node, with identifiers decoded form current alias, and modules expanded to their full import paths.

decorator_list
list

list of decorators, by name as a string, applied to the current function definition.

is_classmethod
bool

True if the current function is designated as a classmethod by a decorator.

visit()
Visits all the nodes within the current function object's AST node.

Updates *self.calls*, *self.modules*, and *self.aliases* for the current instance.

class codegrapher.parser.**FunctionVisitor**(***kwargs*)
Bases: codegrapher.parser.ImportVisitor

Function definitions are where the function is defined, and the call is where the ast for that function exists.

This only looks for items that are called within the scope of a function, and associates those items with the function.

defined_functions
set

names of functions found by function visitor instance.

functions
list

FunctionObject instances found by function visitor instance.

calls
dict

mapping from function names defined to calls within that function definition.

continue_parsing(*node*)

visit_FunctionDef(*node*)

class codegrapher.parser.**ImportVisitor**(*aliases=None*, *modules=None*)
Bases: ast.NodeVisitor

For import related calls, store the source modules and aliases used. Designed to be inherited by other classes that need to know about imports in their current scope.

modules
dict

dict of current modules with *alias: module_name*, *key:value pairs*.

aliases
dict

dict of current modules with *alias: original_name*, *key:value pairs*.

continue_parsing(*node*)

visit_Import(*node*)

visit_ImportFrom(*node*)

3.1.4 Module contents

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

C

A

C

D

E

F

I

M

N

P

R

T

V