32.3. `symtable` — Access to the compiler's symbol tables

Source code: Lib/symtable.py

Symbol tables are generated by the compiler from AST just before bytecode is generated. The symbol table is responsible for calculating the scope of every identifier in the code. `symtable` provides an interface to examine these tables.

32.3.1. Generating Symbol Tables

`symtable.`**`symtable`**(*code*, *filename*, *compile_type*)

Return the toplevel `SymbolTable` for the Python source *code*. *filename* is the name of the file containing the code. *compile_type* is like the *mode* argument to `compile()`.

32.3.2. Examining Symbol Tables

class `symtable.`**`SymbolTable`**

A namespace table for a block. The constructor is not public.

`get_type`()

Return the type of the symbol table. Possible values are `'class'`, `'module'`, and `'function'`.

`get_id`()

Return the table's identifier.

`get_name`()

Return the table's name. This is the name of the class if the table is for a class, the name of the function if the table is for a function, or `'top'` if the table is global (`get_type()` returns `'module'`).

`get_lineno`()

Return the number of the first line in the block this table represents.

`is_optimized`()

Return `True` if the locals in this table can be optimized.

`is_nested`()

Return `True` if the block is a nested class or function.

`has_children`()

Return `True` if the block has nested namespaces within it. These can be obtained with `get_children()`.

`has_exec`()

Return `True` if the block uses `exec`.

`get_identifiers`()

Return a list of names of symbols in this table.

`lookup`(*name*)

Lookup *name* in the table and return a `Symbol` instance.

`get_symbols`()

Return a list of `Symbol` instances for names in the table.

`get_children`()

Return a list of the nested symbol tables.

class `symtable.`**`Function`**

A namespace for a function or method. This class inherits `SymbolTable`.

`get_parameters`()

Return a tuple containing names of parameters to this function.

`get_locals`()

Return a tuple containing names of locals in this function.

`get_globals`()

Return a tuple containing names of globals in this function.

`get_frees`()

Return a tuple containing names of free variables in this function.

class `symtable.`**`Class`**

A namespace of a class. This class inherits `SymbolTable`.

`get_methods`()

Return a tuple containing the names of methods declared in the class.

class `symtable.`**`Symbol`**

An entry in a `SymbolTable` corresponding to an identifier in the source. The constructor is not public.

`get_name()`

Return the symbol’s name.

`is_referenced()`

Return `True` if the symbol is used in its block.

`is_imported()`

Return `True` if the symbol is created from an import statement.

`is_parameter()`

Return `True` if the symbol is a parameter.

`is_global()`

Return `True` if the symbol is global.

`is_declared_global()`

Return `True` if the symbol is declared global with a global statement.

`is_local()`

Return `True` if the symbol is local to its block.

`is_free()`

Return `True` if the symbol is referenced in its block, but not assigned to.

`is_assigned()`

Return `True` if the symbol is assigned to in its block.

`is_namespace()`

Return `True` if name binding introduces new namespace.

If the name is used as the target of a function or class statement, this will be true.

For example:

```
>>> table = symtable.symtable("def some_func(): pass", "string'
>>> table.lookup("some_func").is_namespace()
True
```

Note that a single name can be bound to multiple objects. If the result is `True`, the name may also be bound to other objects, like an int or list, that does not introduce a new namespace.

`get_namespaces()`

Return a list of namespaces bound to this name.

`get_namespace()`

Return the namespace bound to this name. If more than one namespace is bound, `ValueError` is raised.