subete

The Renegade Coder

Jul 05, 2023

CONTENTS

Subete is a library for interacting with the code found in the Sample Programs repository. Use the links below to navigate the docs.

CHAPTER

ONE

USAGE

Interested in interacting with the Sample Programs library in Python? Then subete is the official way to do it!

1.1 Installation

To get started, download and install subete using pip:

```
pip install subete
```

1.2 Basic Usage

From there, you can import the subete library as follows:

```
import subete
```

Then, all that's left to do is to load the Sample Programs repo:

```
repo = subete.load()
```

Keep in mind that the load() function relies on Git being available on the system to be able to clone the Sample Programs repo. Alternatively, you can download the Sample Programs repo yourself and supply the path as an argument:

```
repo = subete.load(source_dir="path/to/sample-programs/archive")
```

With that out of the way, the rest is up to you! Feel free to explore the repo as needed. For example, you can access the list of languages as follows:

```
languages = list(repo)
```

From there, you can browse the individual sample programs available for each language:

```
programs = list(languages)
```

Finally, you can access information about each individual program. For example, you can retrieve the raw code as follows:

```
code = programs[0].code()
```

There are many ways to interact with the repo. Feel free to use this Python API as needed.

1.3 Advanced Usage

Depending on your needs, Subete can be used to access information in more direct ways. For example, both the Repo and LanguageCollection objects are dictionaries under the hood. Rather than exposing that data, we made the objects directly subscriptable. For example, if you want to check out the Python collection, the following will get you there:

```
python_code = repo["Python"]
```

And to access an explicit program, you can use the any of the existing project names:

```
python_hello_world = repo["Python"]["Hello World"]
```

In addition to being subscriptable, both objects are also iterable. For example, to iterate over all of the languages in the repo, you can use the following:

```
for language in repo:
    print(language)
```

Unsurprisingly, the same can be done for each language:

```
for program in repo["Python"]:
    print(program)
```

Beyond that, the API is available for looking up any additional information you made need for each program or language.

CHAPTER

TWO

DOCUMENTATION

The documentation page lists out all of the relevant classes and functions for interacting with the Sample Programs repo.

2.1 subete

The subete module contains all the classes need to represent the Sample Programs repo. This module was designed with the intent of creating read-only objects that fully represent the underlying repo. Ideally, classes that make use of these objects should not need to know how they were generated. For example, we do not want users to poke around the source directory that was used to generate these files. As a result, users should make use of the public methods only.

subete.**load**(*sample_programs_repo_dir: str | None = None*, *sample_programs_website_repo_dir: str | None = None*) → *Repo*

Loads the Sample Programs repo as a Repo object. This is a convenience function which can be used to quickly generate an instance of the Sample Programs repo.

Assuming subete is imported, here's how you would use this function:

```
repo = subete.load()
```

Optionally, you can also provide a source directory which bypasses the need for git on your system:

```
repo = subete.load(sample_programs_repo_dir="path/to/sample-programs/archive")
```

Returns
the Sample Programs repo as a Repo object

2.2 subete.Repo

class subete.repo.**Repo**(*sample_programs_repo_dir: str | None = None*, *sample_programs_website_repo_dir: str | None = None*)

Bases: `object`

An object representing the Sample Programs repository.

Parameters
source_dir (*str*) – the location of the repo archive (e.g., C://…/sample-programs/archive)

__getitem__(*language: str*) → *LanguageCollection*

Makes a repo subscriptable. In this case, the subscript retrieves a language collection.

Assuming you have a Repo object called repo, here's how you would use this method:

```
language: LanguageCollection = repo["Python"]
```

Parameters
language (*str*) – the name of the language to lookup

Returns
the language collection by name

__iter__() → iter

A convenience method for iterating over all language collections in the repo.

Assuming you have a Repo object called repo, here's how you would use this method:

```
for language in repo:
    print(language)
```

Returns
an iterator over all language collections

approved_projects() → List[*Project*]

Retrieves the list of approved projects in the repo. Projects are returned as a list of strings where the strings represent the pathlike project names (e.g., hello-world).

Assuming you have a Repo object called repo, here's how you would use this method:

```
approved_projects: List[str] = repo.approved_projects()
```

Returns
the list of approved projects (e.g. [hello-world, mst])

languages_by_letter(*letter: str*) → List[*LanguageCollection*]

A convenience method for retrieving all language collections that start with a particular letter.

Assuming you have a Repo object called repo, here's how you would use this method:

```
langs: List[LanguageCollection] = repo.languages_by_letter("p")
```

Parameters
letter – a character to search by

Returns
a list of language collections where the language starts with the provided letter

random_program() → *SampleProgram*

A convenience method for retrieving a random program from the repository.

Assuming you have a Repo object called repo, here's how you would use this method:

```
program: SampleProgram = repo.random_program()
```

Returns
a random sample program from the Sample Programs repository

`sample_programs_repo_dir()` → str

Retreives the directory containing the sample programs repository

Returns
the sample programs repository directory

`sorted_language_letters()` → List[str]

A convenience method which generates a list of sorted letters from the sample programs archive. This will return a list of letters that match the directory structure of the archive.

Assuming you have a Repo object called repo, here's how you would use this method:

```
letters: List[str] = repo.sorted_language_letters()
```

Returns
a sorted list of letters

`total_approved_projects()` → int

Retrieves the total number of approved projects in the repo. This value is derived from the number of projects listed in the projects directory of the website repo.

Assuming you have a Repo object called repo, here's how you would use this method:

```
count: int = repo.total_approved_projects()
```

Returns
the total number of approved projects as an int

`total_programs()` → int

Retrieves the total number of programs in the sample programs repo. This total does not include any additional files such as README or testinfo files.

Assuming you have a Repo object called repo, here's how you would use this method:

```
count: int = repo.total_programs()
```

Returns
the total number of programs as an int

`total_tests()` → int

Retrieves the total number of tested languages in the repo. This value is based on the number of testinfo files in the repo.

Assuming you have a Repo object called repo, here's how you would use this method:

```
count: int = repo.total_tests()
```

Returns
the total number of tested languages as an int

2.3 subete.LanguageCollection

class subete.repo.**LanguageCollection**(*name: str*, *path: str*, *file_list: List[str]*, *projects: List[Project]*)

Bases: object

An object representing a collection of sample programs files for a particular programming language.

Parameters

- **name** (*str*) – the name of the language (e.g., python)
- **path** (*str*) – the path of the language (e.g., .../archive/p/python/)
- **file_list** (*list[str]*) – the list of files in language collection
- **projects** (*list[Project]*) – the list of approved projects according to the Sample Programs docs

__getitem__(*program: str*) → str

Makes a language collection subscriptable. In this case, the subscript retrieves a sample program.

Assuming you have a LanguageCollection object called language, you can access a sample program as follows:

```
program: SampleProgram = language["Hello World"]
```

Parameters

program (*str*) – the name of the program to lookup

Returns

the sample program by name

__iter__()

Iterates over all sample programs in the language collection.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
for program in language:
    print(program)
```

Returns

an iterator over all sample programs in the language collection

__str__() → str

Generates as close to the proper language name as possible given a language name in plain text separated by hyphens.

- Example: google-apps-script -> Google Apps Script
- Example: c-sharp -> C#

Assuming you have a LanguageCollection object called language, you can use the following code to get the language name:

```
name: str = str(language)
```

Returns

a readable representation of the language name

doc_authors() → Set[str]

Retrieves the set of authors for this language article. Author names are generated from git blame.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
doc_authors: Set[str] = language.doc_authors()
```

Returns
the set of language article authors

doc_created() → datetime | None

Retrieves the date the language article was created. Created dates are generated from git blame, specifically the article author commits.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
doc_created: Optional[datetime.datetime] = language.doc_created()
```

Returns
the date the language article was created

doc_modified() → datetime | None

Retrieves the date the language article was last modified. Modified dates are generated from git blame, specifically the author commits.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
doc_modified: Optional[datetime.datetime] = language.doc_modified()
```

Returns
the date the language article was last modified

has_docs() → bool

Retrieves the documentation state of this language. Note that documentation may not be complete or up to date.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
state: bool = language.has_docs()
```

Returns
returns true if the language has a documentation folder created for it; false otherwise

has_testinfo() → bool

Retrieves the state of the testinfo file. Helpful when trying to figure out if this language has a testinfo file.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
state: bool = language.has_testinfo()
```

Returns
True if a test info file exists; False otherwise

`lang_docs_url()` → str

Retrieves the URL to the language documentation. The language URL is assumed to exist and therefore not validated. The language documentation URL is in the following form:

`https://sampleprograms.io/languages/{lang}/`

For example, here is a link to the Python documentation.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
link: str = language.lang_docs_url()
```

Returns
the language documentation URL as a string

`missing_programs()` → List[*Project*]

Retrieves the list of missing sample programs for this language.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
missing_programs: List[str] = language.missing_programs()
```

Returns
a list of missing sample programs

`missing_programs_count()` → int

Retrieves the number of missing sample programs for this language.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
missing_programs_count: int = language.missing_programs_count()
```

Returns
the number of missing sample programs

`name()` → str

Retrieves the name of the language in a human-readable format.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
name: str = language.name()
```

Returns
the name of the programming language (e.g., Python, Google Apps Script, C#)

`pathlike_name()`

Retrieves a pathlike name for this language. For example, instead of returning C# it would return c-sharp. Names are based on the folder names in the Sample Programs repo. These names are generated from the file names directly. Use *name()* to get the human-readable name or *str(self)*.

Returns
the pathlike name of this programming language (e.g., c-plus-plus)

readme() → str | None

Retrieves the README contents. README contents are in the form of a markdown string.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
contents: str = language.readme()
```

Returns
the README contents as a string

testinfo() → dict | None

Retrieves the test data from the testinfo file. The YAML data is loaded into a Python dictionary.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
data: dict = language.testinfo()
```

Returns
the test info data as a dictionary

testinfo_url() → str

Retrieves the URL to the testinfo file for this language on GitHub. The testinfo URL is assumed to exist and therefore not validated. The testinfo URL is in the following form:

`https://github.com/TheRenegadeCoder/sample-programs/blob/main/archive/{letter}/{lang}/testinfo.yml`

For example, here is a link to the Python testinfo file.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
link: str = language.testinfo_url()
```

Returns
the testinfo URL as a string

total_line_count() → int

Retrieves the total line count of the language collection. Line count is computed from the sample programs only and does not include lines of code in testinfo or README files.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
lines: int = language.total_line_count()
```

Returns
the total line count of the language collection as an int

total_programs() → int

Retrieves the total number of sample programs in the language collection.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
programs_count: int = language.total_programs()
```

Returns
the number of sample programs as an int

`total_size`() → int

Retrieves the total byte size of the sample programs in the language collection. Size is computed from the size of all sample programs and is not computed from the testinfo or README files.

Assuming you have a LanguageCollection object called language, here's how you would use this method:

```
size: int = language.total_size()
```

Returns
the total byte size of the language collection as an int

2.4 subete.SampleProgram

class subete.repo.**SampleProgram**(*path: str*, *file_name: str*, *language:* LanguageCollection)

Bases: object

An object representing a sample program in the repo.

Parameters

- **path** (*str*) – the path to the sample program without the file name
- **file_name** (*str*) – the name of the file including the extension
- **language** (LanguageCollection) – a reference to the programming language collection of this sample program

__eq__(*o: object*) → bool

Compares an object to the sample program. Returns True if the object is an instance of SampleProgram and matches the following three fields:

- _file_name
- _path
- _language

Assuming you have a SampleProgram object called sample_program, here's how you would use this method:

```
is_sample_program: bool = sample_program == other_sample_program
```

Returns
True if the object matches the Sample Program; False otherwise.

__str__() → str

Renders the Sample Program in the following form: {name} in {language}.

Assuming you have a SampleProgram object called sample_program, here's how you would use this method:

```
name: str = str(sample_program)
```

Returns
the sample program as a string

`article_issue_query_url()` → str

Retrieves the URL to the article issue query for this sample program. The article issue query URL is guaranteed to be a valid search query on GitHub, but it is not guaranteed to have any results. The issue query url is in the following form:

`https://github.com//TheRenegadeCoder/sample-programs-website/issues?{query}"`

For example, here is a link to the Hello World in Python query.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
url: str = program.article_issue_query_url()
```

Returns
the issue query URL as a string

`authors()` → Set[str]

Retrieves the set of authors for this sample program. Author names are generated from git blame.

Assuming you have a SampleProgram object called sample_program, here's how you would use this method:

```
authors: Set[str] = sample_program.authors()
```

Returns
the set of authors

`code()` → str

Retrieves the code for this sample program. To save space in memory, code is loaded from the source file on each invocation of this method. As a result, there may be an IO performance penalty if this function is used many times. It's recommended to store the result of this function if it is used often.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
code: str = program.code()
```

Returns
the code for the sample program as a string

`created()` → datetime | None

Retrieves the date the sample program was created. Created dates are generated from git blame, specifically the author commits.

Assuming you have a SampleProgram object called sample_program, here's how you would use this method:

```
created: Optional[datetime.datetime] = sample_program.created()
```

Returns
the date the sample program was created

doc_authors() → Set[str]

Retrieves the set of authors for this sample program article. Author names are generated from git blame.

Assuming you have a SampleProgram object called sample_program, here's how you would use this method:

```
doc_authors: Set[str] = sample_program.doc_authors()
```

Returns
the set of article authors

doc_created() → datetime | None

Retrieves the date the sample program article was created. Created dates are generated from git blame, specifically the article author commits.

Assuming you have a SampleProgram object called sample_program, here's how you would use this method:

```
doc_created: Optional[datetime.datetime] = sample_program.doc_created()
```

Returns
the date the sample program article was created

doc_modified() → datetime | None

Retrieves the date the sample program article was last modified. Modified dates are generated from git blame, specifically the author commits.

Assuming you have a SampleProgram object called sample_program, here's how you would use this method:

```
doc_modified: Optional[datetime.datetime] = sample_program.doc_modified()
```

Returns
the date the sample program article was last modified

documentation_url() → str

Retrieves the URL to the documentation for this sample program. Documentation URL is assumed to exist and therefore not validated. The documentation URL is in the following form:

`https://sampleprograms.io/projects/{project}/{lang}/`

For example, here is a link to the Hello World in Python documentation.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
url: str = program.documentation_url()
```

Returns
the documentation URL as a string

has_docs() → bool

Retrieves the documentation state of this program. Note that documentation may not be complete or up to date.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
state: bool = program.has_docs()
```

Returns
returns true if the program has a documentation folder created for it; false otherwise

image_type() → str

Determine if sample program is actual an image, and if so, what type.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
image_type: str = program.image_type()
```

Returns
Image type if sample program is an image (e.g., "png"), empty string otherwise

language_collection() → *LanguageCollection*

Retrieves the language collection object that this sample program is a part of.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
name: str = program.language_collection()
```

Returns
the language collection that this program belongs to.

language_name() → str

Retrieves the language name for this sample program. Language name is generated from a call to str() for the source LanguageCollection.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
name: str = program.language_name()
```

Returns
the programming language as a titlecase string (e.g., Python)

language_pathlike_name() → str

Retrieves the language name in the form of a path for URL purposes. This is a convenience method that pulls directly from language collection's *pathlike_name()* method.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
name: str = program.language_pathlike_name()
```

Returns
the language name as a path name (e.g., google-apps-script, python)

line_count() → int

Retrieves the number of lines in the sample program.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
code: int = program.line_count()
```

Returns
the number of lines for the sample program as an integer

modified() → datetime | None

Retrieves the date the sample program was last modified. Modified dates are generated from git blame, specifically the author commits.

Assuming you have a SampleProgram object called sample_program, here's how you would use this method:

```
modified: Optional[datetime.datetime] = sample_program.modified()
```

Returns
the date the sample program was last modified

project() → *Project*

Retrieves the project object for this sample program.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
project: Project = program.project()
```

Returns
the project object for this sample program

project_name() → str

Retrieves the project name of this sample program. Project name is generated from the file name. Specifically, multiword project names are converted to titlecase (e.g., Convex Hull) while acronyms of three or less characters are uppercased (e.g., LPS). This method is an alias for *project.name()*.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
name: str = program.project_name()
```

Returns
the project name as a titlecase string (e.g., Hello World, MST)

project_path() → str

Retrieves the path to the project file.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
project_path: str = program.project_path()
```

Returns
the project path (e.g., . . . /archive/p/python/hello_world.py)

project_pathlike_name() → str

Retrieves the project name in the form of a path for URL purposes. This method is an alias for *project.pathlike_name()*.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
name: str = program.project_pathlike_name()
```

Returns

the project name as a path name (e.g., hello-world, convex-hull)

size() → int

Retrieves the size of the sample program in bytes.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
size: int = program.size()
```

Returns

the size of the sample program as an integer

2.5 subete.Project

class subete.repo.**Project**(*name: str*, *project_tests: Dict | None*)

Bases: object

An object representing a Project in the Sample Programs repo.

Parameters

- **name** (*str*) – the name of the project in its pathlike form (e.g., hello-world)
- **project_tests** – a dictionary containing the test rules for the project

doc_authors() → Set[str]

Retrieves the set of authors for this project article. Author names are generated from git blame.

Assuming you have a Project object called project, here's how you would use this method:

```
doc_authors: Set[str] = project.doc_authors()
```

Returns

the set of project article authors

doc_created() → datetime | None

Retrieves the date the project article was created. Created dates are generated from git blame, specifically the article author commits.

Assuming you have a Project object called project, here's how you would use this method:

```
doc_created: Optional[datetime.datetime] = project.doc_created()
```

Returns

the date the project article was created

doc_modified() → datetime | None

Retrieves the date the project article was last modified. Modified dates are generated from git blame, specifically the author commits.

Assuming you have a Project object called project, here's how you would use this method:

```
doc_modified: Optional[datetime.datetime] = project.doc_modified()
```

Returns
the date the project article was last modified

has_testing() → bool

Responds true if the project has tests.

Returns
True if the project is tested, False otherwise

name() → str

Retrieves the name of the project in its human-readable form.

Assuming you have a Project object called project, here's how you would use this method:

```
name: str = project.name()
```

Returns
the name of the project as a string

pathlike_name() → str

Retrieves the name of the project in its pathlike form (e.g., hello-world).

Assuming you have a Project object called project, here's how you would use this method:

```
pathlike_name: str = project.pathlike_name()
```

Returns
the name of the project in its pathlike form as a string

requirements_url() → str

Retrieves the URL to the requirements documentation for this sample program. Requirements URL is assumed to exist and therefore not validated. The requirements documentation URL is in the following form:

`https://sampleprograms.io/projects/{project}/`

For example, here is a link to the Hello World documentation.

Assuming you have a SampleProgram object called program, here's how you would use this method:

```
url: str = program.requirements_url()
```

Returns
the requirments URL as a string

CHAPTER

THREE

CHANGELOG

Below you'll find all the changes that have been made to the code with newest changes first.

3.1 0.17.x

- **v0.17.0**
  - Added ability to get authors, date/time created, date/time modified for language, sample program, and project articles.

3.2 0.16.x

- **v0.16.0**
  - Added ability to detect if sample program is an image and return image type
  - Added ability to get path to sample program

3.3 0.15.x

- **v0.15.0**
  - Added ability to get sample program repository directory

3.4 0.14.x

- **v0.14.0**
  - Added a feature to lets you check if programs have sources for documentation

3.5 0.13.x

- **v0.13.0**
  - Updated subete to pull from archive and docs separately, rather than relying on submodules which might be out of date

3.6 0.12.x

- **v0.12.1**
  - Fixed an issue where older versions of Git could not handle use of blame
- **v0.12.0**
  - Reworked the way project names are parsed to support new naming conventions
  - Cleaned up error logs for readability

3.7 0.11.x

- **v0.11.2**
  - Fixed a bug where code could not be loaded because the repo was deleted
- **v0.11.1**
  - Fixed an issue where local repo could cause stack overflow
  - Added sections to the changelog
- **v0.11.0**
  - Added support for git data: SampleProgram objects now include authors, created dates, and modified dates
  - Reorganized documentation, so objects have their own sections in the table of contents

3.8 0.10.x

- **v0.10.0**
  - Added support for the Glotter testing file: users can now check if a project is tested by Glotter

3.9 0.9.x

- **v0.9.3**
  - Changed docs dir to sources
- **v0.9.2**
  - Fixed an issue with the use of the SampleProgram constructor
  - Fixed an issue where the missing_programs() method did not work correctly
- **v0.9.1**
  - Updated official documentation
  - Fixed an issue where one of the type hints was wrong
- **v0.9.0**
  - Reworked several of the methods to use the new docs location for website

3.10 0.8.x

- **v0.8.0**
  - Updated URL from sample-programs.therenegadecoder.com to sampleprograms.io

3.11 0.7.x

- **v0.7.2**
  - Fixed a bug where the missing programs list shared the entire path
- **v0.7.1**
  - Fixed a bug where the missing programs feature failed for provided repos
- **v0.7.0**
  - Added Plausible support for analytics
  - Added feature which allows us to retrieve list of missing programs for a language

3.12 0.6.x

- **v0.6.0**
  - Added random program functionality
  - Fixed several documentation issues
  - Renamed some repo functions to match naming conventions
  - Expanded testing to include tests for random functions

3.13 0.5.x

- **v0.5.0**
  - Updated README to indicate alpha stage of project
  - Added logging support
  - Added method of retrieving pathlike name of LanguageCollection
  - Fixed type hinting of sample_programs() method
  - Removed extraneous print statement
  - Made Repo and LanguageCollection subscriptable

3.14 0.4.x

- **v0.4.1**
  - Fixed an issue where generated links were broken
- **v0.4.0**
  - Forced a convention for LanguageCollection and SampleProgram as strings
  - Added test URL functionality to LanguageCollection
  - Created usage docs

3.15 0.3.x

- **v0.3.1**
  - Fixed an issue where provided source directories would not run correctly
- **v0.3.0**
  - Refactored the majority of the underlying library
  - Added testing for Python 3.6 to 3.9

3.16 0.2.x

- **v0.2.1**
  - Fixed an issue where documentation wouldn't build due to sphinx_issues dependency
- **v0.2.0**
  - Added support for Sphinx documentation

3.17 0.1.x

- **v0.1.0**
  - Launches the library under the exact conditions it was in when it was removed from sample-programs-docs-generator

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

C

D

H

I

L

M

N

P

R

S

T