wikitextparser Documentation

Release 0.55.13

Apr 19, 2024

CONTENTS:

CHAPTER

ONE

QUICK START GUIDE

A simple to use WikiText parsing library for MediaWiki.

The purpose is to allow users easily extract and/or manipulate templates, template parameters, parser functions, tables, external links, wikilinks, lists, etc. found in wikitexts.

Table of Contents

1.1 Installation

- Python 3.7+ is required
- `pip install wikitextparser`

1.2 Usage

```
>>> import wikitextparser as wtp
```

WikiTextParser can detect sections, parser functions, templates, wiki links, external links, arguments, tables, wiki lists, and comments in your wikitext. The following sections are a quick overview of some of these functionalities.

You may also want to have a look at the test modules for more examples and probable pitfalls (expected failures).

1.2.1 Templates

```
>>> parsed = wtp.parse("{{text|value1{{text|value2}}}}")
>>> parsed.templates
[Template('{{text|value1{{text|value2}}}}'), Template('{{text|value2}}')]
>>> parsed.templates[0].arguments
[Argument("|value1{{text|value2}}")]
>>> parsed.templates[0].arguments[0].value = 'value3'
>>> print(parsed)
{{text|value3}}
```

The `pformat` method returns a pretty-print formatted string for templates:

```
>>> parsed = wtp.parse('{{t1 |b=b|c=c| d={{t2|e=e|f=f}} }}')
>>> t1, t2 = parsed.templates
>>> print(t2.pformat())
{{t2
    | e = e
    | f = f
}}
>>> print(t1.pformat())
{{t1
    | b = b
    | c = c
    | d = {{t2
        | e = e
        | f = f
    }}
}}
```

`Template.rm_dup_args_safe` and `Template.rm_first_of_dup_args` methods can be used to clean-up pages using duplicate arguments in template calls:

```
>>> t = wtp.Template('{{t|a=a|a=b|a=a}}')
>>> t.rm_dup_args_safe()
>>> t
Template('{{t|a=b|a=a}}')
>>> t = wtp.Template('{{t|a=a|a=b|a=a}}')
>>> t.rm_first_of_dup_args()
>>> t
Template('{{t|a=a}}')
```

Template parameters:

```
>>> param = wtp.parse('{{{a|b}}}').parameters[0]
>>> param.name
'a'
>>> param.default
'b'
>>> param.default = 'c'
>>> param
Parameter('{{{a|c}}}')
>>> param.append_default('d')
>>> param
Parameter('{{{a|{{{d|c}}}}}}')
```

1.2.2 WikiLinks

```
>>> wl = wtp.parse('... [[title#fragmet|text]] ...').wikilinks[0]
>>> wl.title = 'new_title'
>>> wl.fragment = 'new_fragmet'
>>> wl.text = 'X'
>>> wl
WikiLink('[[new_title#new_fragmet|X]]')
>>> del wl.text
>>> wl
WikiLink('[[new_title#new_fragmet]]')
```

All WikiLink properties support get, set, and delete operations.

1.2.3 Sections

```
>>> parsed = wtp.parse("""
... == h2 ==
... t2
... === h3 ===
... t3
... === h3 ===
... t3
... == h22 ==
... t22
... {{text|value3}}
... [[Z|X]]
... """)
>>> parsed.sections
[Section('\n'),
 Section('== h2 ==\nt2\n=== h3 ===\nt3\n=== h3 ===\nt3\n'),
 Section('=== h3 ===\nt3\n'),
 Section('=== h3 ===\nt3\n'),
 Section('== h22 ==\nt22\n{{text|value3}}\n[[Z|X]]\n')]
>>> parsed.sections[1].title = 'newtitle'
>>> print(parsed)

==newtitle==
```

(continues on next page)

(continued from previous page)

```
t2
=== h3 ===
t3
=== h3 ===
t3
== h22 ==
t22
{{text|value3}}
[[Z|X]]
>>> del parsed.sections[1].title
>>>> print(parsed)

t2
=== h3 ===
t3
=== h3 ===
t3
== h22 ==
t22
{{text|value3}}
[[Z|X]]
```

1.2.4 Tables

Extracting cell values of a table:

```
>>> p = wtp.parse("""{|
... |  Orange    ||   Apple   ||   more
... |-
... |   Bread    ||   Pie     ||   more
... |-
... |   Butter   || Ice cream ||  and more
... |}""")
>>> p.tables[0].data()
[['Orange', 'Apple', 'more'],
 ['Bread', 'Pie', 'more'],
 ['Butter', 'Ice cream', 'and more']]
```

By default, values are arranged according to `colspan` and `rowspan` attributes:

```
>>> t = wtp.Table("""{| class="wikitable sortable"
... |-
... ! a !! b !! c
... |-
... !colspan = "2" | d || e
... |-
... |}""")
>>> t.data()
[['a', 'b', 'c'], ['d', 'd', 'e']]
>>> t.data(span=False)
[['a', 'b', 'c'], ['d', 'e']]
```

Calling the `cells` method of a `Table` returns table cells as `Cell` objects. Cell objects provide methods for getting or setting each cell's attributes or values individually:

```
>>> cell = t.cells(row=1, column=1)
>>> cell.attrs
{'colspan': '2'}
>>> cell.set('colspan', '3')
>>> print(t)
{| class="wikitable sortable"
|-
! a !! b !! c
|-
!colspan = "3" | d || e
|-
|}
```

HTML attributes of Table, Cell, and Tag objects are accessible via `get_attr`, `set_attr`, `has_attr`, and `del_attr` methods.

1.2.5 Lists

The `get_lists` method provides access to lists within the wikitext.

```
>>> parsed = wtp.parse(
...     'text\n'
...     '* list item a\n'
...     '* list item b\n'
...     '** sub-list of b\n'
...     '* list item c\n'
...     '** sub-list of b\n'
...     'text'
... )
>>> wikilist = parsed.get_lists()[0]
>>> wikilist.items
[' list item a', ' list item b', ' list item c']
```

The `sublists` method can be used to get all sub-lists of the current list or just sub-lists of specific items:

```
>>> wikilist.sublists()
[WikiList('** sub-list of b\n'), WikiList('** sub-list of b\n')]
>>> wikilist.sublists(1)[0].items
[' sub-list of b']
```

It also has an optional `pattern` argument that works similar to `lists`, except that the current list pattern will be automatically added to it as a prefix:

```
>>> wikilist = wtp.WikiList('#a\n#b\n##ba\n#*bb\n#:bc\n#c', '\#')
>>> wikilist.sublists()
[WikiList('##ba\n'), WikiList('#*bb\n'), WikiList('#:bc\n')]
>>> wikilist.sublists(pattern='\*')
[WikiList('#*bb\n')]
```

Convert one type of list to another using the convert method. Specifying the starting pattern of the desired lists can facilitate finding them and improves the performance:

```
>>> wl = wtp.WikiList(
...     ':*A1\n:*#B1\n:*#B2\n:*:continuing A1\n:*A2',
...     pattern=':\*'
... )
>>> print(wl)
:*A1
:*#B1
:*#B2
:*:continuing A1
:*A2
>>> wl.convert('#')
>>> print(wl)
#A1
##B1
##B2
#:continuing A1
#A2
```

1.2.6 Tags

Accessing HTML tags:

```
>>> p = wtp.parse('text<ref name="c">citation</ref>\n<references/>')
>>> ref, references = p.get_tags()
>>> ref.name = 'X'
>>> ref
Tag('<X name="c">citation</X>')
>>> references
Tag('<references/>')
```

WikiTextParser is able to handle common usages of HTML and extension tags. However it is not a fully-fledged HTML parser and may fail on edge cases or malformed HTML input. Please open an issue on github if you encounter bugs.

1.2.7 Miscellaneous

`parent` and `ancestors` methods can be used to access a node's parent or ancestors respectively:

```
>>> template_d = parse("{{a|{{b|{{c|{{d}}}}}}}}").templates[3]
>>> template_d.ancestors()
[Template('{{c|{{d}}}}'),
 Template('{{b|{{c|{{d}}}}}}'),
 Template('{{a|{{b|{{c|{{d}}}}}}}}')]
>>> template_d.parent()
Template('{{c|{{d}}}}')
>>> _.parent()
Template('{{b|{{c|{{d}}}}}}')
>>> _.parent()
Template('{{a|{{b|{{c|{{d}}}}}}}}')
>>> _.parent()  # Returns None
```

Use the optional `type_` argument if looking for ancestors of a specific type:

```
>>> parsed = parse('{{a|{{#if:{{b{{c<!---->}}}}}}}}')
>>> comment = parsed.comments[0]
>>> comment.ancestors(type_='ParserFunction')
[ParserFunction('{{#if:{{b{{c<!---->}}}}}}')]
```

To delete/remove any object from its parents use `del object[:]` or `del object.string`.

The `remove_markup` function or `plain_text` method can be used to remove wiki markup:

```
>>> from wikitextparser import remove_markup, parse
>>> s = "''''a'''<!--comment--> [[b|c]] [[d]]"
>>> remove_markup(s)
'a c d'
>>> parse(s).plain_text()
'a c d'
```

1.3 Compared with mwparserfromhell

mwparserfromhell is a mature and widely used library with nearly the same purposes as `wikitextparser`. The main reason leading me to create `wikitextparser` was that `mwparserfromhell` could not parse wikitext in certain situations that I needed it for. See mwparserfromhell's issues 40, 42, 88, and other related issues. In many of those situation `wikitextparser` may be able to give you more acceptable results.

Also note that `wikitextparser` is still using 0.x.y version meaning that the API is not stable and may change in the future versions.

The tokenizer in `mwparserfromhell` is written in C. Tokenization in `wikitextparser` is mostly done using the `regex` library which is also in C. I have not rigorously compared the two libraries in terms of performance, i.e. execution time and memory usage. In my limited experience, `wikitextparser` has a decent performance in realistic cases and should be able to compete and may even have little performance benefits in some situations.

If you have had a chance to compare these libraries in terms of performance or capabilities please share your experience by opening an issue on github.

Some of the unique features of `wikitextparser` are: Providing access to individual cells of each table, pretty-printing templates, a WikiList class with rudimentary methods to work with lists, and a few other functions.

1.4 Known issues and limitations

- The contents of templates/parameters are not known to offline parsers. For example an offline parser cannot know if the markup `[[{{z|a}}]]` should be treated as wikilink or not, it depends on the inner-workings of the `{{z}}` template. In these situations `wikitextparser` tries to use a best guess. `[[{{z|a}}]]` is treated as a wikilink (why else would anyone call a template inside wikilink markup, and even if it is not a wikilink, usually no harm is done).
- Localized namespace names are unknown, so for example `[[File:...]]` links are treated as normal wikilinks. `mwparserfromhell` has similar issue, see #87 and #136. As a workaround, Pywikibot can be used for determining the namespace.
- Linktrails are language dependant and are not supported. Also not supported by mwparserfromhell. However given the trail pattern and knowing that `wikilink.span[1]` is the ending position of a wikilink, it is possible to compute a WikiLink's linktrail.

- Templates adjacent to external links are never considered part of the link. In reality, this depends on the contents of the template. Example: `parse('http://example.com{{dead link}}').external_links[0].url == 'http://example.com'`
- List of valid extension tags depends on the extensions intalled on the wiki. The `tags` method currently only supports the ones on English Wikipedia. A configuration option might be added in the future to address this issue.
- `wikitextparser` currently does not provide an ast.walk-like method yielding all descendant nodes.
- Parser functions and magic words are not evaluated.

1.5 Credits

- python
- regex
- wcwidth

Changelog

CHAPTER

TWO

API REFERENCE

Contents

2.1 WikiText

class wikitextparser.**WikiText**(*string: MutableSequence[str] | str*, *_type_to_spans: Dict[str, List[List[int]]] = None*)

Bases: object

__call__(*start: int*, *stop: int | None = False*, *step: int = None*) → str

Return *self.string[start]* or *self.string[start:stop]*.

Return self.string[start] if stop is False. Otherwise return self.string[start:stop:step].

__contains__(*value: str | WikiText*) → bool

Return True if parsed_wikitext is inside self. False otherwise.

Also self and parsed_wikitext should belong to the same parsed wikitext object for this function to return True.

__delitem__(*key: slice | int*) → None

Remove the specified range or character from self.string.

Note: If an operation involves both insertion and deletion, it'll be safer to use the *insert* function first. Otherwise there is a possibility of insertion into the wrong spans.

__init__(*string: MutableSequence[str] | str*, *_type_to_spans: Dict[str, List[List[int]]] = None*) → None

Initialize the object.

Set the initial values for self._lststr, self._type_to_spans.

Parameters

- **string** – The string to be parsed or a list containing the string of the parent object.
- **_type_to_spans** – If the lststr is already parsed, pass its _type_to_spans property as _type_to_spans to avoid parsing it again.

__repr__() → str

Return repr(self).

__setitem__(*key: slice | int*, *value: str*) → None

Set a new string for the given slice or character index.

Use this method instead of calling *insert* and *del* consecutively. By doing so only one of the *_insert_update* and *_shrink_update* functions will be called and the performance will improve.

__str__() → str

Return str(self).

static ancestors(*type_: str | None = None*) → list

Return [] (the root node has no ancestors).

property comments: List[*Comment*]

Return a list of comment objects.

property external_links: List[*ExternalLink*]

Return a list of found external link objects.

Note:

Templates adjacent to external links are considered part of the link. In reality, this depends on the contents of the template:

```
>>> WikiText(
...     'http://example.com{{dead link}}'
...).external_links[0].url
'http://example.com{{dead link}}'
```

```
>>> WikiText(
...     '[http://example.com{{space template}} text]'
...).external_links[0].url
'http://example.com{{space template}}'
```

get_bolds(*recursive=True*) → List[*Bold*]

Return bold parts of self.

Parameters
recursive – if True also look inside templates, parser functions, extension tags, etc.

get_bolds_and_italics(*, *recursive=True*, *filter_cls: type = None*) → List[*Bold* | *Italic*]

Return a list of bold and italic objects in self.

This is faster than calling `get_bolds` and `get_italics` individually. :keyword recursive: if True also look inside templates, parser

functions, extension tags, etc.

Parameters
filter_cls – only return this type. Should be *wikitextparser.Bold* or *wikitextparser.Italic*. The default is None and means both bolds and italics.

get_italics(*recursive=True*) → List[*Italic*]

Return italic parts of self.

Parameters
recursive – if True also look inside templates, parser functions, extension tags, etc.

get_lists(*pattern: str | Tuple[str] = ('\\#', '*', '[:;]')*) → List[*WikiList*]

Return a list of WikiList objects.

Parameters
pattern – The starting pattern for list items. If pattern is not None, it will be passed to the regex engine, so remember to escape the * character. Examples:

- *'#'* means top-level ordered lists
- *'#*'* means unordred lists inside an ordered one
- **Currently definition lists are not well supported, but you** can use *'[:;]'* as their pattern.

Tips and tricks:

Be careful when using the following patterns as they will probably cause malfunction in the *sublists* method of the resultant List. (However don't worry about them if you are not going to use the *sublists* or *List.get_lists* method.)

- **Use '*+' as a pattern and nested unordered lists will be** treated as flat.
- Use *'*s*'* as pattern to rtstrip *items* of the list.

get_sections(**args*, *include_subsections=True*, *level=None*, *top_levels_only=False*) → List[*Section*]

Return a list of sections in current wikitext.

The first section will always be the lead section, even if it is an empty string.

Parameters

- **include_subsections** – If true, include the text of subsections in each Section object.
- **level** – Only return sections where section.level == level. Return all levels if None (default).
- **top_levels_only** – Only return sections that are not subsections of other sections. In this mode, level cannot be specified and *include_subsections* must be True.

get_tables(*recursive=False*) → List[*Table*]

Return tables. Include nested tables if *recursive* is *True*.

get_tags(*name=None*) → List[*Tag*]

Return all tags with the given name.

insert(*index: int*, *string: str*) → None

Insert the given string before the specified index.

This method has the same effect as `self[index:index] = string`; it only avoids some condition checks as it rules out the possibility of the key being an slice, or the need to shrink any of the sub-spans.

property parameters: List[*Parameter*]

Return a list of parameter objects.

static parent(*type_: str | None = None*) → *WikiText* | None

Return None (The parent of the root node is None).

property parser_functions: List[*ParserFunction*]

Return a list of parser function objects.

pformat(*indent: str = ' '*, *remove_comments=False*) → str

Return a pretty-print formatted version of *self.string*.

Try to organize templates and parser functions by indenting, aligning at the equal signs, and adding space where appropriate.

Note that this function will not mutate self.

plain_text(**, replace_templates: bool | ~typing.Callable[[~wikitextparser._template.Template], str | None] = True, replace_parser_functions: bool | ~typing.Callable[[~wikitextparser._parser_function.ParserFunction], str | None] = True, replace_parameters=True, replace_tags=True, replace_external_links=True, replace_wikilinks=True, unescape_html_entities=True, replace_bolds_and_italics=True, replace_tables: ~typing.Callable[[~wikitextparser._table.Table], str | None] | bool = <function _table_to_text>, _is_root_node=False*) → str

Return a plain text string representation of self.

Comments are always removed. :keyword replace_templates:

A function mapping *Template* objects to strings. If True, replace *{{template|argument}}`s with `"*. If False, ignore templates.

Parameters

- **replace_parser_functions** – A function mapping *ParserFunction* objects to strings. If True, replace *{{#parser_function:argument}}'s with `''*. If False, ignore parser functions.
- **replace_parameters** – Replace *{{{a}}}* with `` and {{{a|b}}} with *b*.
- **replace_tags** – Replace *<s>text</s>* with *text*.
- **replace_external_links** – Replace *[https://wikimedia.org/ wm]* with *wm*, and *[https://wikimedia.org/]* with ``.
- **replace_wikilinks** – Replace wikilinks with their text representation, e.g. *[[a|b]]* with *b* and *[[a]]* with *a*.
- **unescape_html_entities** – Replace HTML entities like *Σ*, *Σ*, and *Σ* with .
- **replace_bolds** – replace *'''b'''* with *b*.
- **replace_italics** – replace *''i''* with *i*.

property sections: List[*Section*]

Return self.get_sections(include_subsections=True).

property span: tuple

Return the span of self relative to the start of the root node.

property string: str

Return str(self). Support get, set, and delete operations.

getter and deleter: Note that this will overwrite the current string,
emptying any object that points to the old string.

property tables: List[*Table*]

Return a list of all tables.

property templates: List[*Template*]

Return a list of templates as template objects.

property wikilinks: List[*WikiLink*]

Return a list of wikilink objects.

2.2 SubWikiText

class wikitextparser._wikitext.**SubWikiText**(*string: str | MutableSequence[str]*, *_type_to_spans: Dict[str, List[List[int]]] | None = None*, *_span: List[int] | None = None*, *_type: str | int | None = None*)

Bases: *WikiText*

Define a class to be inherited by some subclasses of WikiText.

Allow focusing on a particular part of WikiText.

__init__(*string: str | MutableSequence[str]*, *_type_to_spans: Dict[str, List[List[int]]] | None = None*, *_span: List[int] | None = None*, *_type: str | int | None = None*) → None

Initialize the object.

ancestors(*type_: str | None = None*) → List[*WikiText*]

Return the ancestors of the current node.

Parameters

type – the type of the desired ancestors as a string. Currently the following types are supported: {Template, ParserFunction, WikiLink, Comment, Parameter, ExtensionTag}. The default is None and means all the ancestors of any type above.

parent(*type_: str | None = None*) → *WikiText* | None

Return the parent node of the current object.

Parameters

type – the type of the desired parent object. Currently the following types are supported: {Template, ParserFunction, WikiLink, Comment, Parameter, ExtensionTag}. The default is None and means the first parent, of any type above.

Returns

parent WikiText object or None if no parent with the desired *type_* is found.

2.3 SubWikiTextWithAttrs

class wikitextparser._tag.**SubWikiTextWithAttrs**(*string: str | MutableSequence[str], _type_to_spans: Dict[str, List[List[int]]] | None = None, _span: List[int] | None = None, _type: str | int | None = None*)

Bases: *SubWikiText*

Define a class for SubWikiText objects that have attributes.

Any class that is going to inherit from SubWikiTextWithAttrs should provide _attrs_match property. Note that matching should be done on shadow. It's usually a good idea to cache the _attrs_match property.

property attrs: Dict[str, str]

Return self attributes as a dictionary.

del_attr(*attr_name: str*) → None

Delete all the attributes with the given name.

Pass if the attr_name is not found in self.

get_attr(*attr_name: str*) → str | None

Return the value of the last attribute with the given name.

Return None if the attr_name does not exist in self. If there are already multiple attributes with the given name, only return the value of the last one. Return an empty string if the mentioned name is an empty attribute.

has_attr(*attr_name: str*) → bool

Return True if self contains an attribute with the given name.

set_attr(*attr_name: str, attr_value: str*) → None

Set the value for the given attribute name.

If there are already multiple attributes with the given name, only set the value for the last one. If attr_value == '', use the implicit empty attribute syntax.

2.4 SubWikiTextWithArgs

class wikitextparser._parser_function.**SubWikiTextWithArgs**(*string: str | MutableSequence[str], _type_to_spans: Dict[str, List[List[int]]] | None = None, _span: List[int] | None = None, _type: str | int | None = None*)

Bases: *SubWikiText*

Define common attributes for *Template* and *ParserFunction*.

property arguments: List[*Argument*]

Parse template content. Create self.name and self.arguments.

get_lists(*pattern: str | Iterable[str] = ('\\#', '*', '[:;]')*) → List[*WikiList*]

Return the lists in all arguments.

For performance reasons it is usually preferred to get a specific Argument and use the *get_lists* method of that argument instead.

property name: str

Template's name (includes whitespace).

getter: Return the name. setter: Set a new name.

property nesting_level: int

Return the nesting level of self.

The minimum nesting_level is 0. Being part of any Template or ParserFunction increases the level by one.

2.5 Template

class wikitextparser.**Template**(*string: str | MutableSequence[str], _type_to_spans: Dict[str, List[List[int]]] | None = None, _span: List[int] | None = None, _type: str | int | None = None*)

Bases: *SubWikiTextWithArgs*

Convert strings to Template objects.

The string should start with {{ and end with }}.

del_arg(*name: str*) → None

Delete all arguments with the given then.

get_arg(*name: str*) → *Argument* | None

Return the last argument with the given name.

Return None if no argument with that name is found.

has_arg(*name: str, value: str = None*) → bool

Return true if the is an arg named *name*.

Also check equality of values if *value* is provided.

Note: If you just need to get an argument and you want to LBYL, it's
better to get_arg directly and then check if the returned value is None.

normal_name(*rm_namespaces=('Template',)*, ***, *code: str = None*, *capitalize=False*) → str

Return normal form of self.name.

- Remove comments.
- Remove language code.
- Remove namespace ("template:" or any of *localized_namespaces*.
- Use space instead of underscore.
- Remove consecutive spaces.
- Use uppercase for the first letter if *capitalize*.
- Remove #anchor.

Parameters

- **rm_namespaces** – is used to provide additional localized namespaces for the template namespace. They will be removed from the result. Default is ('Template',).
- **capitalize** – If True, convert the first letter of the template's name to a capital letter. See [[mw:Manual:$wgCapitalLinks]] for more info.
- **code** – is the language code.

Example:

```
>>> Template(
...     '{{ eN : tEmPlAtE : <!-- c --> t_1 # b | a }}'
... ).normal_name(code='en')
'T 1'
```

rm_dup_args_safe(*tag: str = None*) → None

Remove duplicate arguments in a safe manner.

Remove the duplicate arguments only in the following situations:

1. **Both arguments have the same name AND value. (Remove one of** them.)
2. **Arguments have the same name and one of them is empty. (Remove** the empty one.)

Warning: Although this is considered to be safe and no meaningful data is removed from wikitext, but the result of the rendered wikitext may actually change if the second arg is empty and removed but the first had had a value.

If *tag* is defined, it should be a string that will be appended to the value of the remaining duplicate arguments.

Also see *rm_first_of_dup_args* function.

rm_first_of_dup_args() → None

Eliminate duplicate arguments by removing the first occurrences.

Remove the first occurrences of duplicate arguments, regardless of their value. Result of the rendered wikitext should remain the same. Warning: Some meaningful data may be removed from wikitext.

Also see *rm_dup_args_safe* function.

set_arg(*name: str*, *value: str*, *positional: bool = None*, *before: str = None*, *after: str = None*, *preserve_spacing=False*) → None

Set the value for *name* argument. Add it if it doesn't exist.

- Use *positional*, *before* and *after* keyword arguments only when adding a new argument.
- If *before* is given, ignore *after*.
- If neither *before* nor *after* are given and it's needed to add a new argument, then append the new argument to the end.
- If *positional* is True, try to add the given value as a positional argument. Ignore *preserve_spacing* if positional is True. If it's None, do what seems more appropriate.

property templates: List[*Template*]

Return a list of templates as template objects.

2.6 ParserFunction

class wikitextparser.**ParserFunction**(*string: str | MutableSequence[str]*, *_type_to_spans: Dict[str, List[List[int]]] | None = None*, *_span: List[int] | None = None*, *_type: str | int | None = None*)

Bases: *SubWikiTextWithArgs*

property parser_functions: List[*ParserFunction*]

Return a list of parser function objects.

2.7 WikiLink

class wikitextparser.**WikiLink**(*string: str | MutableSequence[str]*, *_type_to_spans: Dict[str, List[List[int]]] | None = None*, *_span: List[int] | None = None*, *_type: str | int | None = None*)

Bases: *SubWikiText*

property fragment: str | None

Fragment identifier.

getter: target's fragment identifier (do not include the # character) setter: set a new fragment (do not include the # character) deleter: delete fragment, including the # character.

property target: str

WikiLink's target, including the fragment.

Do not include the pipe (|) in setter and getter. Deleter: delete the link target, including the pipe character.

Use *self.target = ''* if you don't want to remove the pipe.

property text: str | None

The [[inner text| of WikiLink]] (not including the [[link]]trail).

setter: set a new value for self.text. Do not include the pipe. deleter: delete self.text, including the pipe.

property title: str

Target's title

getter: get target's title (do not include the # character) setter: set a new title (do not include the # character) deleter: return new title, including the # character.

property wikilinks: List[*WikiLink*]

Return a list of wikilink objects.

2.8 ExternalLink

class wikitextparser.**ExternalLink**(*string: str | MutableSequence[str], _type_to_spans: Dict[str, List[List[int]]] | None = None, _span: List[int] | None = None, _type: str | int | None = None*)

Bases: *SubWikiText*

property external_links: List[*ExternalLink*]

Return a list of found external link objects.

Note:

Templates adjacent to external links are considered part of the link. In reality, this depends on the contents of the template:

```
>>> WikiText(
...     'http://example.com{{dead link}}'
...).external_links[0].url
'http://example.com{{dead link}}'
```

```
>>> WikiText(
...     '[http://example.com{{space template}} text]'
...).external_links[0].url
'http://example.com{{space template}}'
```

property in_brackets: bool

Return true if the ExternalLink is in brackets. False otherwise.

property text: str | None

The text part (the part after the url).

getter: Return None if this is a bare link or has no associated text. setter: Automatically put the ExternalLink in brackets if it's not

already.

deleter: Delete self.text, including the space before it.

property url: str

URL of the current ExternalLink object.

getter: Return the URL. setter: Set a new value for URL. Convert add brackets for bare

external links.

2.9 Argument

class wikitextparser.**Argument**(*string: str | MutableSequence[str]*, *_type_to_spans: Dict[str, List[List[int]]] | None = None*, *_span: List[int] | None = None*, *_type: str | int | None = None*, *_parent:* SubWikiTextWithArgs *= None*)

Bases: *SubWikiText*

Create a new Argument Object.

Note that in MediaWiki documentation *arguments* are (also) called parameters. In this module the convention is: {{{parameter}}}, {{template|argument}}. See https://www.mediawiki.org/wiki/Help:Templates for more information.

__init__(*string: str | MutableSequence[str]*, *_type_to_spans: Dict[str, List[List[int]]] | None = None*, *_span: List[int] | None = None*, *_type: str | int | None = None*, *_parent:* SubWikiTextWithArgs *= None*)

Initialize the object.

property name: str

Argument's name.

getter: return the position as a string, for positional arguments. setter: convert it to keyword argument if positional.

property positional: bool

True if self is positional, False if keyword.

setter:
If set to False, convert self to keyword argumentn. Raise ValueError on trying to convert positional to keyword argument.

property value: str

Value of self.

Support both keyword or positional arguments. getter:

Return value of self.

setter:
Assign a new value to self.

2.10 Parameter

class wikitextparser.**Parameter**(*string: str | MutableSequence[str]*, *_type_to_spans: Dict[str, List[List[int]]] | None = None*, *_span: List[int] | None = None*, *_type: str | int | None = None*)

Bases: *SubWikiText*

append_default(*new_default_name: str*) → None

Append a new default parameter in the appropriate place.

Add the new default to the innter-most parameter. If the parameter already exists among defaults, don't change anything.

Example:

```
>>> p = Parameter('{{{p1|{{{p2|}}}}}}')
>>> p.append_default('p3')
>>> p
Parameter("'{{{p1|{{{p2|{{{p3|}}}}}}}}}'")
```

property default: str | None

The default value of current parameter.

getter: Return None if there is no default. setter: Set a new default value. deleter: Delete the default value, including the pipe character.

property name: str

Current parameter's name.

getter: Return current parameter's name. setter: set a new name for the current parameter.

property parameters: List[*Parameter*]

Return a list of parameter objects.

property pipe: str

Return | if there is a pipe (default value) in the Parameter.

Return '' otherwise.

2.11 Section

class wikitextparser.**Section**(**args*, ***kwargs*)

Bases: *SubWikiText*

__init__(**args*, ***kwargs*)

Initialize the object.

property contents: str

Contents of this section.

getter: return the contents setter: Set contents to a new string value.

property level: int

The level of this section.

getter: Return level which as an int in range(1,7) or 0 for the lead section.

setter: Change the level.

property title: str | None

The title of this section.

getter: Return the title or None for lead sections or sections that don't have any title.

setter: Set a new title. deleter: Remove the title, including the equal sign and the newline after it.

2.12 Comment

class wikitextparser.**Comment**(*string: str | MutableSequence[str]*, *_type_to_spans: Dict[str, List[List[int]]] | None = None*, *_span: List[int] | None = None*, *_type: str | int | None = None*)

Bases: *SubWikiText*

property comments: List[*Comment*]

Return a list of comment objects.

property contents: str

Return contents of this comment.

2.13 Table

class wikitextparser.**Table**(**args*, ***kwargs*)

Bases: *SubWikiTextWithAttrs*

__init__(**args*, ***kwargs*)

Initialize the object.

property caption: str | None

Caption of the table. Support get and set.

property caption_attrs: str | None

Caption attributes. Support get and set operations.

cells(*row: int = None*, *column: int = None*, *span: bool = True*) → List[List[Cell]] | List[Cell] | Cell

Return a list of lists containing Cell objects.

Parameters

- **span** – If is True, rearrange the result according to colspan and rospan attributes.
- **row** – Return the specified row only. Zero-based index.
- **column** – Return the specified column only. Zero-based index.

If both row and column are provided, return the relevant cell object.

If only need the values inside cells, then use the `data` method instead.

data(*span: bool = True*, *strip: bool = True*, *row: int = None*, *column: int = None*) → List[List[str]] | List[str] | str

Return a list containing lists of row values.

Parameters

- **span** – If true, calculate rows according to rowspans and colspans attributes. Otherwise ignore them.
- **row** – Return the specified row only. Zero-based index.
- **column** – Return the specified column only. Zero-based index.
- **strip** – strip data values

Note: Due to the lots of complications that it may cause, this function won't look inside templates, parser functions, etc. See https://www.mediawiki.org/wiki/Extension:Pipe_Escape for how wiki-tables can be inserted within templates.

property nesting_level: int

Return the nesting level of self.

The minimum nesting_level is 0. Being part of any Table increases the level by one.

property row_attrs: List[dict]

Row attributes.

Use the setter of this property to set attributes for all rows. Note that it will overwrite all the existing attr values.

2.14 Tag

class wikitextparser.**Tag**(**args*, ***kwargs*)

Bases: *SubWikiTextWithAttrs*

__init__(**args*, ***kwargs*)

Initialize the object.

property contents: str | None

Tag contents. Support both get and set operations.

setter:
Set contents to a new value. Note that if the tag is self-closing, then it will be expanded to have a start tag and an end tag. For example: >>> t = Tag('<t/>') >>> t.contents = 'n' >>> t.string '<t>n</t>'

get_tags(*name=None*) → List[*Tag*]

Return all tags with the given name.

property name: str

Tag's name. Support both get and set operations.

property parsed_contents: *SubWikiText*

Return the contents as a SubWikiText object.

2.15 WikiList

class wikitextparser.**WikiList**(*string: str | MutableSequence[str]*, *pattern: str*, *_match: Match = None*, *_type_to_spans: Dict[str, List[List[int]]] = None*, *_span: List[int] = None*, *_type: str = None*)

Bases: *SubWikiText*

Class to represent ordered, unordered, and definition lists.

__init__(*string: str | MutableSequence[str]*, *pattern: str*, *_match: Match = None*, *_type_to_spans: Dict[str, List[List[int]]] = None*, *_span: List[int] = None*, *_type: str = None*) → None

Initialize the object.

convert(*newstart: str*) → None

Convert to another list type by replacing starting pattern.

property fullitems: List[str]

Return list of item strings. Includes their start and sub-items.

get_lists(*pattern: str | Iterable[str] = ('\\#', '*', '[:;]')*) → List[*WikiList*]

Return a list of WikiList objects.

Parameters

pattern – The starting pattern for list items. If pattern is not None, it will be passed to the regex engine, so remember to escape the * character. Examples:

- *'#'* means top-level ordered lists
- *'#*'* means unordred lists inside an ordered one
- **Currently definition lists are not well supported, but you** can use *'[:;]'* as their pattern.

Tips and tricks:

Be careful when using the following patterns as they will probably cause malfunction in the *sublists* method of the resultant List. (However don't worry about them if you are not going to use the *sublists* or *List.get_lists* method.)

- **Use '*+' as a pattern and nested unordered lists will be** treated as flat.
- Use *'*s*'* as pattern to rtstrip *items* of the list.

property items: List[str]

Return items as a list of strings.

Do not include sub-items and the start pattern.

property level: int

Return level of nesting for the current list.

Level is a one-based index, for example the level for * *a* will be 1.

sublists(*i: int = None*, *pattern: str | Iterable[str] = ('\\#', '*', '[:;]')*) → List[*WikiList*]

Return the Lists inside the item with the given index.

Parameters

- **i** – The index of the item which its sub-lists are desired.
- **pattern** – The starting symbol for the desired sub-lists. The *pattern* of the current list will be automatically added as prefix.

2.16 SubWikiText

class wikitextparser._comment_bold_italic.**BoldItalic**(*string: str | MutableSequence[str]*, *_type_to_spans: Dict[str, List[List[int]]] | None = None*, *_span: List[int] | None = None*, *_type: str | int | None = None*)

Bases: *SubWikiText*

property text: str

Return text value of self (without triple quotes).

2.17 Bold

class wikitextparser.Bold(*string: str | MutableSequence[str], _type_to_spans: Dict[str, List[List[int]]] | None = None, _span: List[int] | None = None, _type: str | int | None = None*)

Bases: BoldItalic

2.18 Italic

class wikitextparser.Italic(*string: str | MutableSequence[str], _type_to_spans: Dict[str, List[List[int]]] | None = None, _span: List[int] | None = None, _type: str | int | None = None, end_token: bool = True*)

Bases: BoldItalic

__init__(*string: str | MutableSequence[str], _type_to_spans: Dict[str, List[List[int]]] | None = None, _span: List[int] | None = None, _type: str | int | None = None, end_token: bool = True*)

Initialize the Italic object.

Parameters

end_token – set to True if the italic object ends with a '' token False otherwise.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

B

C

D

E

F

G

H