Text adventure builder Documentation

Release 0.1

Erik Nyquist

May 08, 2022

API Documentation:

CHAPTER 1

chatbot_utils

1.1 chatbot_utils package

1.1.1 Submodules

class chatbot_utils.format_tokens.**FormattedResponse**(*response_text*, *match_groups*, *variables={}*)

Bases: object

Helper class for parsing variable assignments out of a response phrase, and applying format tokens to the response text

Variables

- **response_text** (*str*) – The unformatted response text containing format tokens and/or variable assignments
- **match_groups** (*list*) – The match groups from the input text that matched a regular expression
- **variables** (*dict*) – dict of variables that should be included when applying format tokens

__init__(*response_text*, *match_groups*, *variables={}*)

Initialize self. See help(type(self)) for accurate signature.

parse(*response_text*, *match_groups*, *variables={}*)

exception chatbot_utils.format_tokens.**InvalidFormatTokenError**

Bases: Exception

Raised when an unknown format token is used in a response phrase

class chatbot_utils.redict.**ReDict**(**args*, ***kwargs*)

Bases: dict

Special dictionary which expects values to be *set* with regular expressions (REs) as keys, and expects values to be retreived using input text for an RE as keys. The value corresponding to the regular expression which matches

the input text will be returned. In the case where the input text matches multiple REs, one of the matching values will be returned, but precisely which one is undefined.

Example usage:

```
>>> d = ReDict()
>>> d['hello( world(!)*)?'] = 1
>>> d['regex|dict key'] = 2
>>> d['hello']
1
>>> d['hello world!!!!']
1
>>> d['regex']
2
>>> d['dict key']
2
```

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

clear()
Clear all key/value pairs stored in this dict

compile()
Compile all regular expressions in the dictionary

copy()
Create a new ReDict instance and copy all items in this dict into the new instance

Returns new ReDict instance containing copied data

Return type *ReDict*

dump_to_dict()
Dump all pattern/value pairs to a regular dict, where the regular expressions are the keys

Returns dict of pattern/value pairs

Return type dict

groups()
Return tuple of all subgroups from the last regex match performed when fetching an item, as returned by re.MatchObject.groups()

Returns tuple of subgroups from last match

Return type tuple

items()
Return all values stored in this dict

Returns list of values

Return type list

iteritems()
Returns a generator to get all key/value pairs stored in this dict

Returns generator to get pattern/value pairs

keys()
Return all keys stored in this dict

Returns list of keys

Return type list

load_from_dict(*data*)
Load pattern/value pairs from a regular dict. This overwrites any existing pattern/value pairs

Parameters **data** (*dict*) – pattern/value pairs to load

pop(*text*)
Return and delete the first value associated with a pattern matching 'text'

Parameters **text** (*str*) – text to match against

Returns value associated with pattern matching 'text' (if any)

update(*other*)
Add items from 'other' into this dict

Parameters **other** (ReDict) – dict containing items to copy

values()
Return all values stored in this dict

Returns list of values

Return type list

class chatbot_utils.responder.**Context**(*lists=None*)
Bases: object

Class representing a "discussion" context, allowing for a Responder that responds with contextual awareness

__init__(*lists=None*)
Initialize self. See help(type(self)) for accurate signature.

add_chained_phrases(**pattern_response_pairs*)
Add multiple chained pattern/response pairs. A chain defines a sequence of pattern/response pairs that are expected in the order that they occur in the passed arguments to this method. Whenever a Responder is inside a context and input matching the first pattern/response pair in a chain is seen, the Responder will continually expect the next pattern in the current chain until another chain or another context is entered. When the last pattern in the chain is reached, Responders will continue expecting this pattern until another chain or context is entered.

Parameters **pattern_response_pairs** – one or more pattern/response pairs, where a pattern/response pair is a tuple of the form `(regexs, value)`, where `regexs` is a regular expression or list of regular expressions and `value` is an arbitrary object

add_context(*context*)
Add context that can only be entered when already in this context

Parameters **context** (chatbot_utils.responder.Context) – context instance to add

add_contexts(**contexts*)
Add one or more context instances to this context

Parameters **contexts** (chatbot_utils.responder.Context) – context instances to add

add_entry_phrase(*patterns*, *response*)
Add a pattern/response pair to be used as an entry point for this context. If input matching matching one of the patterns passed here is seen, Responders will return the corresponding response object and enter the context.

Parameters

- **patterns** – regular expression or list of regular expressions. If the input passed to `get_response` matches one of these patterns, then the object passed here as `response` will be returned.
- **response** (*object*) – object to return from `get_response` if the passed input matches one of the regular expressions passed here as `response`. The response may be any object type, but if it is a single string, then the string may contain format tokens; see *details about format tokens*

add_entry_phrases(**pattern_response_pairs*)

Add one or more pattern/response pairs to be used as entry points for this context. If input matching matching one of the patterns passed here is seen, Responders will return the corresponding response object and enter the context.

Parameters **pattern_response_pairs** – one or more pattern/response pairs, where a pattern/response pair is a tuple of the form `(regexs, value)`, where `regexs` is a regular expression or list of regular expressions and `value` is an arbitrary object

add_exit_phrase(*patterns*, *response*)

Add a pattern/response pair to be used as an exit point for this context. If input matching matching one of the patterns passed here is seen, Responders will return the corresponding response object and exit the context.

Parameters

- **patterns** – regular expression or list of regular expressions. If the input passed to `get_response` matches one of these patterns, then the object passed here as `response` will be returned.
- **response** (*object*) – object to return from `get_response` if the passed input matches one of the regular expressions passed here as `response`. The response may be any object type, but if it is a single string, then the string may contain format tokens; see *details about format tokens*

add_exit_phrases(**pattern_response_pairs*)

Add one or more pattern/response pairs to be used as exit points for this context. If input matching matching one of the patterns passed here is seen, Responders will return the corresponding response object and exit the context.

Parameters **pattern_response_pairs** – one or more pattern/response pairs, where a pattern/response pair is a tuple of the form `(regexs, value)`, where `regexs` is a regular expression or list of regular expressions and `value` is an arbitrary object

add_response(*patterns*, *response*)

Add a pattern/response pair that will be only be recognized when a Responder is in this context

Parameters

- **patterns** – regular expression or list of regular expressions. If the input passed to `get_response` matches one of these patterns, then the object passed here as `response` will be returned.
- **response** (*object*) – object to return from `get_response` if the passed input matches one of the regular expressions passed here as `response`. The response may be any object type, but if it is a single string, then the string may contain format tokens; see *details about format tokens*

add_responses(**pattern_response_pairs*)

Add one more more pattern/response pairs that will be only be recognized when a Responder is in this context

Parameters **pattern_response_pairs** – one or more pattern/response pairs, where a pattern/response pair is a tuple of the form `(regexs, value)`, where `regexs` is a regular expression or list of regular expressions and `value` is an arbitrary object

compile()
Compile all regular expressions contained in this context so they are ready for immediate matching

get_response(*text*)
Find a response object associated with a pattern in this context that matches 'text', and return it (if any). If no matching patterns can be found, 'text' itself will be returned.

Parameters **text** (*str*) – input text to check for matching patterns against

Returns tuple of the form `(response, groups)`. `response` is the response object associated with the matching regular expression, if any, otherwise 'text'. `groups` is a tuple of subgroups from the regular expression match (as returned by re.MatchObject.groups), if any, otherwise None.

class chatbot_utils.responder.**NoResponse**
Bases: object

class chatbot_utils.responder.**Responder**
Bases: object

Represents a high-level responder object which can be used to register pattern/response pairs, and can accept input text to retrieve matching response objects

__init__()
Initialize self. See help(type(self)) for accurate signature.

add_context(*context*)
Add context instance to this responder

Parameters **context** (chatbot_utils.responder.Context) – context instance to add

add_contexts(**contexts*)
Add one or more context instances to this responder

Parameters **contexts** (chatbot_utils.responder.Context) – context instances to add

add_default_response(*response*)
Set response to return when no other matching responses can be found

Parameters **response** – object to return as default response

add_response(*patterns*, *response*)
Add a pattern/response pair that will always be recognized by a Responder, regardless of context

Parameters

- **patterns** (*list*) – list of regular expressions. If the input passed to `get_response` matches one of these patterns, then the object passed here as `response` will be returned.
- **response** (*object*) – object to return from `get_response` if the passed input matches one of the regular expressions passed here as `response`. The response may be any object type, but if it is a single string, then the string may contain format tokens; see *details about format tokens*

add_responses(**pattern_response_pairs*)
Add one or moe pattern/response pairs that will always be recognized by a Responder, regardless of context

Parameters **pattern_response_pairs** – one or more pattern/response pairs, where a pattern/response pair is a tuple of the form `(regexs, value)`, where `regexs` is a regular expression or list of regular expressions and `value` is an arbitrary object

compile()
Compile all regular expressions contained in this responder (including contexts), so they are ready for matching immediately

get_response(*text*)
Find a response object associated with a pattern that matches 'text', and return it (if any). If no matching patterns can be found, 'text' itself will be returned.

Parameters **text** (*str*) – input text to check for matching patterns against

Returns tuple of the form `(response, groups)`. `response` is the response object associated with the matching regular expression, if any, otherwise 'text'. `groups` is a tuple of subgroups from the regular expression match (as returned by re.MatchObject.groups), if any, otherwise None.

class chatbot_utils.utils.**ContextCreator**(*context_parent*, *entry_phrases=None*)
Bases: `object`

Context manager for populating a chatbot_utils.responder.Context object and adding it to either a Responder object or another Context object.

Example usage:

```
from chatbot_utils.responder import Responder
from chatbot_utils.utils import ContextCreator

responder = Responder()

with ContextCreator(responder) as ctx:
    # Add entry phrase for context #1
    ctx.add_entry_phrase(...)

    # Add nested subcontext
    with ContextCreator(ctx) as subctx:
        # Add entry phrase for subcontext #1
        subctx.add_entry_phrase(...)
```

Variables

- **context_parent** – The object that the new context should be added to; should be a chatbot_utils.responder.Responder instance or chatbot_utils.responder.Context instance
- **entry_phrases** – Optional list of entry phrase tuples for this context. If non-None, then this will be passed to the 'add_entry_phrases' method of the chatbot_utils.responder.Context object after instantiation.

__init__(*context_parent*, *entry_phrases=None*)
Initialize self. See help(type(self)) for accurate signature.

chatbot_utils.utils.**get_input**(*prompt=None*)
Helper function, maps to "raw_input" in py2 and "input" in py3

CHAPTER 2

Format tokens

2.1 Using format tokens to "remember" some earlier data that was provided

Response text may include format tokens that reference matching text within parenthesis groups in the pattern. These tokens should be of the form `{{pN}}`, where `N` is an integer representing the position of the parenthesis group within the pattern, from left-to-right.

For example, given the pattern `I like ([a-z]*) and ([a-z]*)`, and the response text `I like {{p0}} too, but not {{p1}}`, an input of `I like cats and dogs` would yield a response of `I like cats too, but not dogs`.

2.2 Creating new format tokens with special response syntax

The provided response text may also contain commands to create custom format tokens on the fly. Custom format tokens may be mapped to arbitrary literal strings, or to other format tokens. This is achieved by appending `;;` to the end of the response text, to mark the beginning of the custom format token assignments, followed by one or more comma-separated assignment statements of the form `name=value` (both name and value may be any string of characters, except for `,` and =).

For example, given the pattern `I like (.*) and (.*)`, and the response text `OK, {{p0}} and {{p0}};;like1={{p0}},like2={{p1}}`, an input of `I like green and red` would yield a response of `OK, green and red`, and would create two new format tokens named "like1" and "like2" that can be used in future response phrases. For example, the response text `you like {{like1}} and {{like2}}` would yield "you like green and red" when triggered.

2.3 Example bot that remembers your favourite movie

Here is an example implementation of a simple bot that can remember your favourite movie and report it to you when asked:

```
from chatbot_utils.responder import Responder
from chatbot_utils.utils import ContextCreator, get_input

responder = Responder()

responder.add_default_response("Please tell me what your favourite movie is")

responder.add_responses(
    # When the bot is told what my favourite film is, it will save whatever film I␣
↪said (4th
    # parenthesis group, or p3) in a new variable named "faveMovie"
    (["(.* )?(favou?rite|fave) (movie|film) is (.*)$"],
     "Cool, I will remember that your favourite film is {p3}!;;faveMovie={p3}"),

    # When the bot is asked to recall what my favourite film is, it will report the␣
↪value of 'faveMovie'
    (["(.*)?(what is|what'?s|(can you )?tell me )?(what('?s)? )?my (fave|favou?rite)␣
↪(movie|film).*"],
     "Your favourite film is {faveMovie}!")
)

# Simple prompt to get input from command line and pass to responder
while True:
    text = get_input(" > ")
    resp, groups = responder.get_response(text)
    print("\n\"%s\"\n" % resp)
```

And here is some sample output from running the above example script:

```
$> python example_format_tokens_bot.py

> howdy!

"Please tell me what your favourite movie is"

> hmm, OK, I guess my favourite film is Gone With The Wind

"Cool, I will remember that your favourite film is Gone With The Wind!"

> hey, can you tell me what my fave movie is?

"Your favourite film is Gone With The Wind!"

> alright, now my favorite movie is spiderman 2

"Cool, I will remember that your favourite film is spiderman 2!"

> what's my favourite film?

"Your favourite film is spiderman 2!"

>
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

A

C

D

F

G

I