inquirer Documentation

Release 2.1.11

Miguel Ángel García

Aug 12, 2017

Contents

CHAPTER 1

Overview

Born as a Inquirer.js clone, it shares part of the goals and philosophy.

So, **Inquirer** should ease the process of asking end user **questions**, **parsing**, **validating** answers, managing **hierarchical prompts** and providing **error feedback**.

You can *download the code from GitHub*.

CHAPTER 2

Contents

Installation

To install it, just execute:

```
pip install inquirer
```

Usage example:

```
import inquirer

if __name__ == '__main__':

    questions = [
        inquirer.Text('user', message='Please enter your github username',␣
↪validate=lambda _, x: x != '.'),
        inquirer.Password('password', message='Please enter your password'),
        inquirer.Text('repo', message='Please enter the repo name', default='default
↪'),
        inquirer.Checkbox('topics', message='Please define your type of project?',␣
↪choices=['common', 'backend', 'frontend'], ),
        inquirer.Text('organization', message='If this is a repo from a organization␣
↪please enter the organization name, if not just leave this blank'),
        inquirer.Confirm('correct',  message='This will delete all your current␣
↪labels and create a new ones. Continue?', default=False),
    ]

    answers = inquirer.prompt(questions)

    print answers
```

Usage

The idea is quite simple:

1. Create an array of `Questions`
2. Call the prompt render.

Each `Question` require some common arguments. So, you just need to know which kind of `Questions` and `Arguments` are available.

Question types

TEXT	Expects a text answer
PASSWORD	Do not prompt the answer.
CONFIRM	Requires a boolean answer
LIST	Show a list and allow to select just one answer.
CHECKBOX	Show a list and allow to select a bunch of them

There are pictures of some of them in the examples section.

Question Arguments

The main object is `Question`, but it should not be instantiated. You must use any of the subclasses, listed below. All of them have the next attributes that can be set in the initialization:

name

It will be the key in the hash of answers. So, it is **mandatory**.

You can use any `String` or `hashable` code as value.

message

Contains the prompt to be shown to the user, and is **mandatory** too.

You can use a new style formatted string, using the previous answers, and it will be replaced automatically:

```
questions = [
    Text(name='name', message="What's your name?"),
    Text(name='surname', message="What's your surname, {name}")
]
```

The value can be a `function`, with the next sign:

```
def get_message(answers): return str()
```

Example:

```
def get_message(answers):
    return "What's your name?"

Text(name='name', message= get_message)
```

Where `answers` is the dictionary with previous answers.

If the `message` is too long for the terminal, it will be cut to fit.

default

Stores the default value to be used as answer. This allow the user just to press *Enter* to use it. It is optional, using `None` if there is no input and no default value.

As in ‘‘message‘ , you can use a new format string or a function with the sign:

```
def get_default(answers): return str()
```

Where `answers` is a `dict` containing all previous answers.

Remember that it should be an array for *Checkbox* questions.

choices

Mandatory just for `Checkbox` and `List` questions; the rest of them do not use it.

It contains the list of selectable answers.

Its value can be a `list` of strings, new format style strings or pairs(tuples) or a *function* that returns that list, with the sign:

```
def get_choices(answers): return list(str())
```

If any of the list values is a pair, it should be a tuple like: `(label, value)`. Then the `label` will be shown but the `value` will be returned.

As before, the `answers` is a *dict* containing the previous answers.

validate

Optional attribute that allows the program to check if the answer is valid or not. It requires a *boolean* value or a *function* with the sign:

```
def validate(answers, current): return boolean()
```

Where `answers` is a *dict* with previous answers again and `current` is the current answer. Example:

```
Text('age', "how old are you?", validate=lambda _, c: 0 <= c < 120)
```

ignore

Questions are statically created and some of them may be optional depending on other answers. This attribute allows to control this by hiding the question.

It's value is *boolean* or a *function* with the sign:

```
def ignore(answers): return boolean()
```

where `answers` contains the *dict* of previous answers again.

Creating the Question object

With this information, it is easy to create a `Question` object:

```
Text('name', "What's your name?")
```

It's possible to load the `Question` objects from a `dict`, or even the whole list of them, with the method `load_from_dict` and `load_from_list`, respectively.

The method `load_from_json` has been added as commodity to use JSON inputs instead. Here you have an example:

```
import os
import sys
import re
import json
sys.path.append(os.path.realpath('.'))
from pprint import pprint

import inquirer

with open('examples/test_questions.json') as fd:
    questions = inquirer.load_from_json(fd.read())

answers = inquirer.prompt(questions)

pprint(answers)
```

The prompter

The last step is to call the *prompter* With the list of `Question`:

```
answers = inquirer.prompt(questions)
```

This line will ask the user for information and will store the answeres in a dict, using the question name as **key** and the user response as **value**.

Remember the `prompt` always require a list of `Question` as input.

Themes

You can change the colorscheme and some icons passing a theme object defined in inquirer.themes There are Default and GreenPassion themes, but you can define your own via class, dict or json!

```
import inquirer
from inquirer.themes import GreenPassion

q = [
    inquirer.Text('name',
                  message='Whats your name?',
                  default='No one'),
    inquirer.List('jon',
                  message='Does Jon Snow know?',
                  choices=['yes', 'no'],
                  default='no'),
    inquirer.Checkbox('kill_list',
                      message='Who you want to kill?',
                      choices=['Cersei', 'Littlefinger', 'The Mountain']
                      )
]

inquirer.prompt(q, theme=GreenPassion())
```

Result:

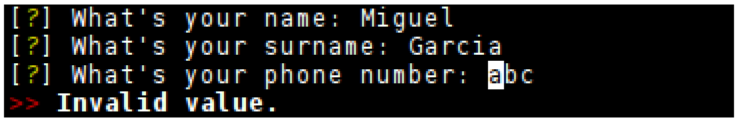
Examples

You can find all these examples at `examples` directory.

text.py

```
import os
import sys
import re
sys.path.append(os.path.realpath('.'))
from pprint import pprint

import inquirer

questions = [
    inquirer.Text('name',
                  message="What's your name?"),
    inquirer.Text('surname',
                  message="What's your surname, {name}?"),
    inquirer.Text('phone',
                  message="What's your phone number",
                  validate=lambda _, x: re.match('\+?\d[\d ]+\d', x),
                  )
]

answers = inquirer.prompt(questions)

pprint(answers)
```

Result on something like:



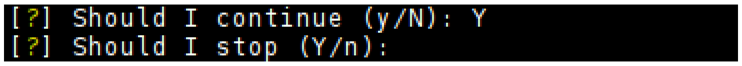
```
[?] What's your name: Miguel
[?] What's your surname: Garcia
[?] What's your phone number: abc
>> Invalid value.
```

confirm.py

```
import os
import sys
import re
sys.path.append(os.path.realpath('.'))
from pprint import pprint
```

```python
import inquirer

questions = [
    inquirer.Confirm('continue',
                  message="Should I continue"),
    inquirer.Confirm('stop',
                  message="Should I stop", default=True),
]

answers = inquirer.prompt(questions)

pprint(answers)
```

Result on something like:

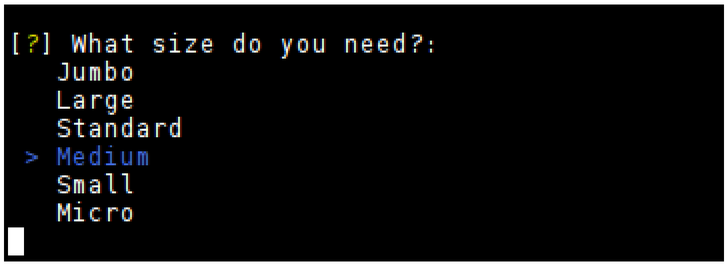


```
[?] Should I continue (y/N): Y
[?] Should I stop (Y/n):
```

list.py

```python
import os
import sys
import re
sys.path.append(os.path.realpath('.'))
from pprint import pprint

import inquirer

questions = [
    inquirer.List('size',
                  message="What size do you need?",
                  choices=['Jumbo', 'Large', 'Standard', 'Medium', 'Small', 'Micro'],
              ),
]

answers = inquirer.prompt(questions)

pprint(answers)
```

Result on something like:

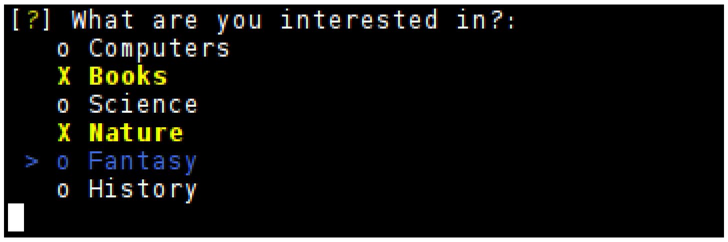


```
[?] What size do you need?:
   Jumbo
   Large
   Standard
 > Medium
   Small
   Micro
```

checkbox.py

```python
import os
import sys
import re
sys.path.append(os.path.realpath('.'))
from pprint import pprint

import inquirer

questions = [
    inquirer.Checkbox('interests',
                      message="What are you interested in?",
                      choices=['Computers', 'Books', 'Science', 'Nature', 'Fantasy',
↪'History'],
                      default=['Computers', 'Books']),
]

answers = inquirer.prompt(questions)

pprint(answers)
```

Result on something like:



```
[?] What are you interested in?:
   o Computers
   X Books
   o Science
   X Nature
 > o Fantasy
   o History
```

theme.py

```
import inquirer
from inquirer.themes import GreenPassion

q = [
    inquirer.Text('name',
                  message='Whats your name?',
                  default='No one'),
    inquirer.List('jon',
                  message='Does Jon Snow know?',
                  choices=['yes', 'no'],
                  default='no'),
    inquirer.Checkbox('kill_list',
                      message='Who you want to kill?',
                      choices=['Cersei', 'Littlefinger', 'The Mountain']
                      )
]

inquirer.prompt(q, theme=GreenPassion())
```

Result on something like:

Changelog

2.1.11(2014/12/18)

Features

- #18 The `Prompt` shout raise `KeyboardInterrupt` if required.

2.1.3 (2014/12/27)

Bugs

- The `Question` start was not shown.

2.1.2 (2014/12/16)

Features

- #7 Adding default values for *Checkbox*, by ptbrowne

2.1.1 (2014/12/11)

Bugs

- Status bar was hiden by question
- Fixed a `force_new_line` problem with some environments.

2.1.0 (2014/12/10)

Features

- code refactors
- Adding ReadTheDocs documentation

Bugs

- #6 Removed new line after questions
- Confirmations will not raise an exception on unknown value

2.0.2 (2014/11/27)

Features

- Using pytest instead of nose
- Documentation updated

- Added `changes.rst` file with the changelog

Bugs

- #5: Fixed `List` and `Checkbox`, that were overriden if there was more `Questions`.

2.0.1 (2014/10/31)

Features

- **‘#4‘_**: Instantiate from JSON
  - Internal refactors
  - added load_from_dict and load_from_json factories, by mfwarren

2.0.0 (2014/10/19)

Features

- Complete refactor of `Question`, `ConsoleRender` and the way it was rendered with `blessings` library.

1.X.X

Special thanks to matiboy by his contributions to these releases.

Hall Of Fame

Contributors:

- matiboy
- mfwarren
- ptbrowne

The MIT License (MIT)

inquirer

inquirer package

Subpackages

inquirer.render package

Subpackages

inquirer.render.console package

Submodules

inquirer.render.console.base module

class `inquirer.render.console.base.`**`BaseConsoleRender`**(*question*, *theme=None*, *terminal=None*, *show_default=False*, **args*, ***kwargs*)

Bases: `object`

`get_current_value`()

`get_header`()

`get_options`()

`read_input`()

`title_inline` = **False**

Module contents

class `inquirer.render.console.`**`ConsoleRender`**(*event_generator=None*, *theme=None*, **args*, ***kwargs*)

Bases: `object`

`clear_bottombar`()

`clear_eos`()

`height`

`print_line`(*base*, *lf=True*, ***kwargs*)

`print_str`(*base*, *lf=False*, ***kwargs*)

`render`(*question*, *answers=None*)

`render_error`(*message*)

render_factory(*question_type*)

render_in_bottombar(*message*)

width

Module contents

class `inquirer.render.`**Render**(*impl=<class 'inquirer.render.console.ConsoleRender'>*)
Bases: `object`

render(*question, answers*)

Submodules

inquirer.errors module

exception `inquirer.errors.`**Aborted**
Bases: *inquirer.errors.InquirerError*

exception `inquirer.errors.`**EndOfInput**(*selection, *args, **kwargs*)
Bases: *inquirer.errors.InquirerError*

exception `inquirer.errors.`**InquirerError**
Bases: `exceptions.Exception`

exception `inquirer.errors.`**ThemeError**
Bases: `exceptions.AttributeError`

exception `inquirer.errors.`**UnknownQuestionTypeError**
Bases: *inquirer.errors.InquirerError*

exception `inquirer.errors.`**ValidationError**(*value, *args, **kwargs*)
Bases: *inquirer.errors.InquirerError*

inquirer.events module

class `inquirer.events.`**Event**
Bases: `object`

class `inquirer.events.`**KeyEventGenerator**(*key_generator=None*)
Bases: `object`

next()

class `inquirer.events.`**KeyPressed**(*value*)
Bases: *inquirer.events.Event*

class `inquirer.events.`**Repaint**
Bases: *inquirer.events.Event*

inquirer.prompt module

`inquirer.prompt.`**prompt**(*questions, render=None, answers=None, theme=<inquirer.themes.Default object>, raise_keyboard_interrupt=False*)

inquirer.questions module

Module that implements the questions types

class `inquirer.questions.`**`Checkbox`**(*name*, *message=''*, *choices=None*, *default=None*, *ignore=False*, *validate=True*, *show_default=False*)

Bases: *`inquirer.questions.Question`*

`kind` **= 'checkbox'**

class `inquirer.questions.`**`Confirm`**(*name*, *default=False*, ***kwargs*)

Bases: *`inquirer.questions.Question`*

`kind` **= 'confirm'**

class `inquirer.questions.`**`List`**(*name*, *message=''*, *choices=None*, *default=None*, *ignore=False*, *validate=True*, *show_default=False*)

Bases: *`inquirer.questions.Question`*

`kind` **= 'list'**

class `inquirer.questions.`**`Password`**(*name*, *message=''*, *choices=None*, *default=None*, *ignore=False*, *validate=True*, *show_default=False*)

Bases: *`inquirer.questions.Question`*

`kind` **= 'password'**

class `inquirer.questions.`**`Question`**(*name*, *message=''*, *choices=None*, *default=None*, *ignore=False*, *validate=True*, *show_default=False*)

Bases: `object`

`choices`

`choices_generator`

`default`

`ignore`

`kind` **= 'base question'**

`message`

`validate`(*current*)

class `inquirer.questions.`**`TaggedValue`**(*label*, *value*)

Bases: `object`

class `inquirer.questions.`**`Text`**(*name*, *message=''*, *choices=None*, *default=None*, *ignore=False*, *validate=True*, *show_default=False*)

Bases: *`inquirer.questions.Question`*

`kind` **= 'text'**

`inquirer.questions.`**`load_from_dict`**(*question_dict*)

Load one question from a dict. It requires the keys 'name' and 'kind'. :return: The Question object with associated data. :return type: Question

`inquirer.questions.`**`load_from_json`**(*question_json*)

Load Questions from a JSON string. :return: A list of Question objects with associated data if the JSON

contains a list or a Question if the JSON contains a dict.

Return type List or Dict

`inquirer.questions.`**`load_from_list`**(*question_list*)
Load a list of questions from a list of dicts. It requires the keys 'name' and 'kind' for each dict. :return: A list of Question objects with associated data. :return type: List

`inquirer.questions.`**`question_factory`**(*kind*, **args*, ***kwargs*)

Module contents

`inquirer.`**`prompt`**(*questions*, *render=None*, *answers=None*, *theme=<inquirer.themes.Default object>*, *raise_keyboard_interrupt=False*)

class `inquirer.`**`Text`**(*name*, *message=''*, *choices=None*, *default=None*, *ignore=False*, *validate=True*, *show_default=False*)
Bases: *`inquirer.questions.Question`*

`kind` **= 'text'**

class `inquirer.`**`Password`**(*name*, *message=''*, *choices=None*, *default=None*, *ignore=False*, *validate=True*, *show_default=False*)
Bases: *`inquirer.questions.Question`*

`kind` **= 'password'**

class `inquirer.`**`Confirm`**(*name*, *default=False*, ***kwargs*)
Bases: *`inquirer.questions.Question`*

`kind` **= 'confirm'**

class `inquirer.`**`List`**(*name*, *message=''*, *choices=None*, *default=None*, *ignore=False*, *validate=True*, *show_default=False*)
Bases: *`inquirer.questions.Question`*

`kind` **= 'list'**

class `inquirer.`**`Checkbox`**(*name*, *message=''*, *choices=None*, *default=None*, *ignore=False*, *validate=True*, *show_default=False*)
Bases: *`inquirer.questions.Question`*

`kind` **= 'checkbox'**

`inquirer.`**`load_from_dict`**(*question_dict*)
Load one question from a dict. It requires the keys 'name' and 'kind'. :return: The Question object with associated data. :return type: Question

`inquirer.`**`load_from_json`**(*question_json*)
Load Questions from a JSON string. :return: A list of Question objects with associated data if the JSON

contains a list or a Question if the JSON contains a dict.

Return type List or Dict

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index