

---

# **Ask Me Anything Documentation**

*Release 0.2*

**Simon Kennedy**

**Feb 23, 2018**



---

## Contents

---

<b>1</b>	<b>License</b>	<b>3</b>
<b>2</b>	<b>Contact</b>	<b>5</b>
<b>3</b>	<b>Version History</b>	<b>7</b>
3.1	Questions in JSON . . . . .	7
3.2	Validation . . . . .	9
3.3	The ama Module . . . . .	10
3.4	ama.validator . . . . .	11
	<b>Python Module Index</b>	<b>15</b>



Ask Me Anything (AMA) is a package to prompt the user for answers to a series of questions and return the results as a dictionary.

Questions are specified using a *JSON dictionary*

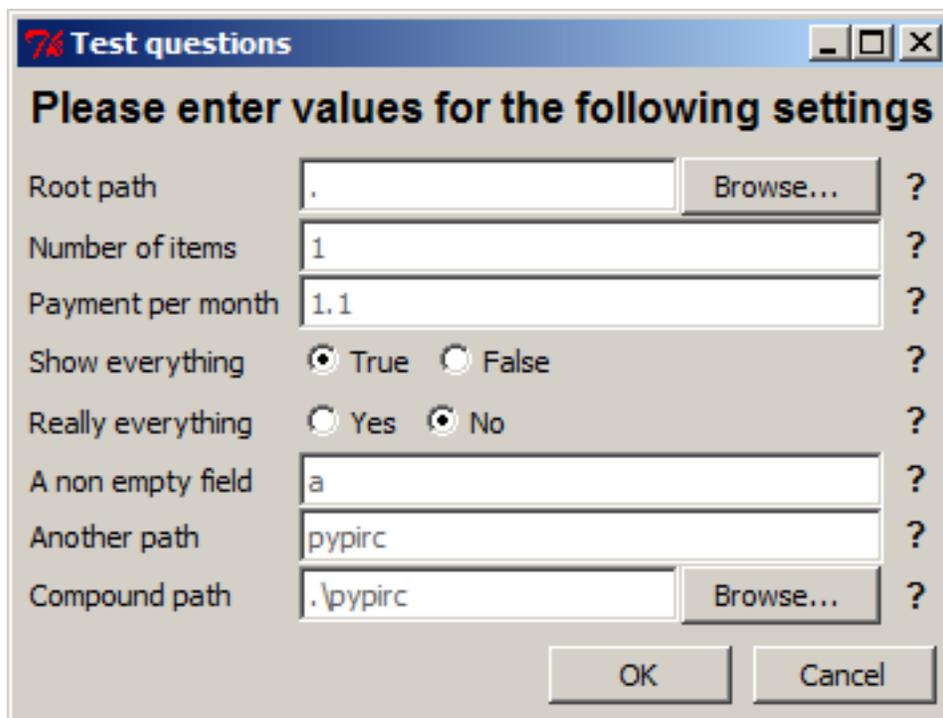
Currently you have 2 choices in how to ask the questions...

1. Using the terminal

```
Test questions
-----
Please enter values for the following settings

Root path [.] :
Number of items [1] :
Payment per month [1.1] :
Show everything [True] :
Really everything (y/n) [n] :
A non empty field [a] :
Another path [pypirc] :
Compound path [pypirc] :
```

2. Using a Tkinter interface.



The Tk interface has a few niceties; dialogs to select paths, dynamic updating of values based on others, a validation indication etc.

both generated from the same JSON file.



# CHAPTER 1

---

## License

---

This module is licensed under the terms of the [Apache V2.0](#) license.



## CHAPTER 2

---

### Contact

---

Simon Kennedy <[sffjunkie+code@gmail.com](mailto:sffjunkie+code@gmail.com)>



---

## Version History

---

Version	Description
0.2	<ul style="list-style-type: none"><li>• Added date, time, color handling</li><li>• Changed how questions are defined in JSON</li><li>• Simplification of validation</li></ul>
0.1	Initial version with terminal and Tkinter support

### 3.1 Questions in JSON

Each question is specified as an entry in a JSON dictionary.

After the dictionary key to use to lookup the question comes a list containing the following...

- a short description/label to be displayed
- help text to display
- the default value
- the type specified as a string “str”, “bool” etc.
- a custom specification to modify the validation (see *Validation*).

The default value can use Python [string formatting](#) operators to build a value from other answers as in the `join_path` question below.

---

**Note:** If you’re using the terminal you can only refer to previous answers as the questions are asked in order of definition and only once.

---

JSON Example

```
{
  "a_date":
    [
      "Select a date",
      "A date",
      null,
      "date",
      null
    ],
  "a_time":
    [
      "Select a time",
      "A time",
      null,
      "time",
      "%H:%M:%S"
    ],
  "a_color":
    [
      "Select a color",
      "A color",
      "#fdcbef",
      "color",
      "rgbhex"
    ],
  "path":
    [
      "Root path",
      "Root path for the documentation",
      ".",
      "path",
      "new"
    ],
  "count":
    [
      "Number of items",
      "How many items",
      1,
      "int",
      null
    ],
  "payment":
    [
      "Payment per month",
      "How many items",
      1.1,
      "float",
      null
    ],
  "show_all":
    [
      "Show everything",
      "Do you want to show every single item in the whole world",
      true,
      "bool",
      null
    ],
  "show_yesno":
    [
      "Really everything",
      "Do you really want to show every single item in the whole world",
      false,
      "yesno",
      null
    ],
  "something":

```

```

["A non empty field",
 "You really have to enter something",
 "a",
 "str",
 "nonempty"
],
"path2":
["Another path",
 "A second path used by the next question to join 2 together",
 "pypirc",
 "path",
 null
],
"join_path":
["Compound path",
 "Joining of path & path2",
 "{path}/{path2}",
 "path",
 null
]
}

```

## 3.2 Validation

### 3.2.1 Types

The types available are

- nonempty
- str
- bool
- yesno (like bool but displays ”(y/n)” at the prompt)
- int
- float
- path
- date
- time
- color (RGB and RGB Hex e.g. rgb(1.0, 0.0, 0.0) or #ff0000 for red)
- regular expressions
- An entry point definition

**nonempty** Enter any value to pass the validation.

**bool** Matches any of true, false, 1, 0.

**yesno** Matches any of yes, y, no, n with any case plus the bool values

**str** Can it be converted to a string.

**int** Can it be converted to an integer.

**float** Can it be converted to a float.

**path** Verifies that the value is a valid path. Various specs can be provided to modify the path validation

**empty** verifies that the path is empty

**nonempty** verifies that at least one file is found in the path

**new** verifies that the path does not exist and is a valid path name

**pathspec** Verifies that the path conforms to the *pathspec* given (*see below*)

**date** Verifies that a valid date is provided that matches the *datespec* where *datespec* follows the standard Python `strptime()` format string. If no specification is provided then `%Y-%m-%d` will be used.

**time** Verifies that the time is a valid time that matches the *timespec* where *timespec* follows the standard Python `strptime()` format string. If no specification is provided then `%H:%M` will be used.

**color** Verifies that the value is a valid color that matches the *colorspec* where *colorspec* is either `rgb` or `rgba`

**re** Verifies that the value specified matches the regular expression.

**Entry Point** If a `setuptools` entry point is specified then it will be loaded and used to validate the entry.

### 3.2.2 Path Specs

Path specs contain multiple `glob` patterns separated by commas each preceded by either a plus or minus sign.

A plus sign (+) indicates that a file matching the `glob` must be in the directory.

A minus sign (-) indicates that a file matching the `glob` must not be in the directory.

e.g. `+test.py, -*.txt` means the directory must have a `test.py` file included but no text files

## 3.3 The ama Module

Base Asker class

### 3.3.1 Asker

**class** `ama.Asker` (*ds=None, json\_string=None*)

An object which mediates the question asking.

#### Parameters

- **ds** (*Any object with a read method*) – A datastream to read the questions from
- **json\_string** (*str*) – A JSON formatted string to load the questions from

**ask** (*questions=None, initial\_answers=None, all\_questions=True*)

Ask the questions and return the answers

#### Parameters

- **questions** (*string or dict*) – The questions to prompt for answers. Can either be a json formatted string or a dict subclass
- **initial\_answers** (*dict*) – A dictionary containing the already answered questions
- **all\_questions** (*bool*) – If `True` only the already unanswered questions will be asked; if `False` all questions will be asked.

- **validators** (*dict*) – A dictionary of custom validator functions

**Returns** The answers

**Return type** OrderedDict

**add\_question** (*key, question*)

Overridden by subclasses to add a question to the list to ask. Called by the `ask()` method

**run** ()

Overridden by subclasses to ask the questions. Subclasses should return a dictionary of the answers

### 3.3.2 ama.terminal

TerminalAsker

### 3.3.3 ama.tk

TkAsker

TkQuestion

## 3.4 ama.validator

Provides access to a registry of validation functions.

Functions are returned via the `get_validator()` function and can be refined by passing a specification which alters what passes the validation.

All validators throw `TypeError` if the value's type cannot be validated and `ValueError` if the value fails validation.

Validator Name	Tests that the value...
<code>nonempty</code>	is not <code>None</code> or an empty string
<code>constant</code>	always returns the same value
<code>str</code>	can be converted to a string
<code>int</code>	can be converted to an integer value
<code>float</code>	can be converted to a floating point value
<code>bool</code>	can be converted to a boolean value
<code>yesno</code>	matches one of <code>yes</code> , <code>y</code> , <code>no</code> , <code>n</code> with any case plus <code>1</code> , <code>0</code> , <code>True</code> and <code>False</code>
<code>re</code>	matches the regular expression.
<code>path</code>	is a valid path
<code>date</code>	is a valid date
<code>time</code>	is a valid time
<code>color</code>	is a valid RGB or RGB hex color
<code>email</code>	is a valid email address

`ama.validator.NonEmpty` (*\*args, \*\*kwargs*)

Create a validator that checks that any value is provided

`ama.validator.Constant` (*\*args, \*\*kwargs*)

Create a validator that always return the same value.

`ama.validator.Str` (*\*args, \*\*kwargs*)

Create a validator that checks that the value is a valid string according to the *spec*

**Parameters** `spec (str)` – The specification to check the string against. Can be either

**None** Anything that can be converted to a string passes

**The string nonempty** a string of length greater than 1 passes

**A string of argument=value pairs separated by commas.** Checks the string matches based on the arguments specified

The following arguments can be specified.

`min` - The minimum number of characters

`max` - The maximum number of characters

e.g. “min=3,max=6” means the string must be between 3 and 6 characters long.

`ama.validator.Int (*args, **kwargs)`

Create a validator that checks that the value is a valid integer according to the `spec`

**Parameters** `spec (str)` – The specification to check the integer against. Can be either

**None** Anything that is an integer passes. e.g. 1 and “1” are valid integers but 1.2, “1.2” or “chas” are not.

**A string of argument=value pairs separated by commas.** Alters how the integer is validated.

The following arguments can be specified.

`min` - The minimum value

`max` - The maximum value

e.g. “min=3,max=6” means the value must be between 3 and 6.

`ama.validator.Float (*args, **kwargs)`

Create a validator that checks that the value is a valid float according to the `spec`

**Parameters** `spec (str)` – The specification to check the float against. Can be either

**None** Anything that is a float passes. e.g. 1.2 and “1.2” are valid floats but 1, “1” or “dave” are not.

**A string of argument=value pairs separated by commas.** Alters how the float is validated.

The following arguments can be specified.

`min` - The minimum value

`max` - The maximum value

`decimal` - The character to consider as the decimal separator

`nocoerce` - Disable coercing int to float

e.g. “min=3.1,max=6.0” means the value must be between 3.1 and 6.0; “decimal=,” means that “33,234” is a valid float.

`ama.validator.Number (*args, **kwargs)`

Create a validator that checks that the value is a valid number according to the `spec`

**Parameters** `spec (str)` – The specification to check the integer against. Can be either

**None** Anything that is a number passes.

**A string of argument=value pairs separated by commas.** Check s the integer matches based on the arguments specified

The following arguments can be specified.

`min` - The minimum value

`max` - The maximum value

`decimal` - The character to consider as the decimal separator

e.g. “min=3,max=6” means the value must be between 3 and 6.

`ama.validator.Bool(*args, **kwargs)`

Create a validator that checks that the value is a valid bool.

`ama.validator.Regex(*args, **kwargs)`

Create a validator that checks that the value matches a regular expression.

`ama.validator.Path(*args, **kwargs)`

Create a validator that checks that the value is a valid path.

The meaning of valid is determined by the *spec* argument

**Parameters** *spec* (*str*) – Determines what is a valid path.

**existing** is a path that exists (the default)

**empty** is a path that is empty

**nonempty** is a path that is not empty

**new** is a path that does not exist and is a valid name for a path

**pathspec** is a valid path name that contains files that conform to *pathspec*

*pathspec* is of the form `[+-] glob` where the leading `+` indicates that the path must include a file that matches the *glob* and `-` indicates that it must not include files that match the *glob*.

Multiple *pathspec*s can be specified separated by commas.

`ama.validator.Date(*args, **kwargs)`

Create a validator that checks that the value is a valid date.

**Parameters** *spec* (*str*) – The date format to accept if a string value is used.

*spec* follows the standard Python `strptime` format string.

`ama.validator.Time(*args, **kwargs)`

Create a validator that checks that the value is a valid time.

**Parameters** *spec* (*str*) – The time format to accept if a string value is used.

*spec* follows the standard Python `strptime` format string.

`ama.validator.Color(*args, **kwargs)`

Create a validator that checks that the value is a valid color

The color format, which is determined by the *spec* argument, can be one of the following

- An RGB hex representation i.e. `#` followed by either 3 or 6 hex digits.
- A string of the form `'rgb(R, G, B)'` where R, G and B are floating point values between 0.0 and 1.0

**Parameters** *spec* (*str*) – The color type to accept either `'rgbhex'` or `'rgb'`

`ama.validator.Email(*args, **kwargs)`

Create a validator that checks that the value is a valid email address.

If the `pyisemail` module is available then that is used to validate the email address otherwise a regular expression is used (which may produce false positives.)



**a**

`ama`, 10

`ama.validator`, 11



## A

add\_question() (ama.Asker method), 11  
ama (module), 10  
ama.validator (module), 11  
ask() (ama.Asker method), 10  
Asker (class in ama), 10

## B

Bool() (in module ama.validator), 13

## C

Color() (in module ama.validator), 13  
Constant() (in module ama.validator), 11

## D

Date() (in module ama.validator), 13

## E

Email() (in module ama.validator), 13

## F

Float() (in module ama.validator), 12

## I

Int() (in module ama.validator), 12

## N

NonEmpty() (in module ama.validator), 11  
Number() (in module ama.validator), 12

## P

Path() (in module ama.validator), 13

## R

Regex() (in module ama.validator), 13  
run() (ama.Asker method), 11

## S

Str() (in module ama.validator), 11

## T

Time() (in module ama.validator), 13