
char Documentation

stanislav

Aug 23, 2020

Contents

1	Contents	3
1.1	README	3
1.2	Module Reference	8
1.3	Other	10
2	Indices and tables	13
	Python Module Index	15
	Index	17

This is the documentation of python package **char**.

1.1 README

1.1.1 char

Table of Contents

- *char*
 - *Overview.*
 - * *Example*
 - *Differences from python type hinting*
 - *Installation via pip:*
 - *Usage with default settings*
 - * *Default prefixes*
 - * *Example*
 - *Usage with user defined settings*
 - * *Decorator arguments*

- * Decorator argument: *bool_is_to_skip_None_value*
- * Decorator argument: *dict_tuple_types_by_prefix_to_update_default*
- * Decorator argument: *dict_tuple_types_by_prefix*
- *Links*
- *Project local Links*
- *Contacts*
- *License*

Overview.

char stands for: check of arguments.

This library gives to user ability to check types of function arguments via one decorator.

If your team or you have some agreements how to name variables with defined types

Or if you are ready to use mine (which is derived from Hungarian notation (it will be described bellow)) then type checking will be *simple* and **pleasant**.

P.S. I like to use variables notation because it really improves readability of the code.

Example

Let's say that you've defined a function **func(int_x)** and you want to check if value that is given to variable **int_x** has type int

Usually you would have to check it by adding one more line at the start of the function somehow like this: **isinstance(int_x, int)**

With this decorator this can be done for you automatically.

```
from char import char
# OR from char import check_types_of_arguments # They are equivalent

@char
def func(int_x):
    pass
```

If you try to call function with wrong types of arguments like: **func("pewpew")** then you'll get a nice exception with description:

```
ArgumentTypeError: Incorrect type of variable was given to function: func
--> For variable: int_x
--> Were given value: pewpew
```

(continues on next page)

(continued from previous page)

```
---> With type: <class 'str'>  
---> Instead of: <class 'int'>
```

Differences from python type hinting

The main difference from the internal python type hinting is that type hinting do static analysis of the code and do not do any checks during the runtime.

So actually it can't protect a user from giving to some type hinted function arguments with the wrong types so even with type hinting you still have to write isinstance type checks.

Additional difference is that this package supports python ≥ 2.7 when type hinting is available only since python 3.5

In case if you don't want to use any notation of variables and want to use the type hinting then you can try the library **typeguard**.

Installation via pip:

```
pip install char
```

Usage with default settings

Default prefixes

Here will be a list of name prefixes and which type the variable is expected to be

If variable name doesn't start with any of the given prefixes then variable type won't be checked.

1. "any_" - object
2. "bool_" - bool
3. "b_" - bool
4. "is_" - bool
5. "has_" - bool
6. "str_" - str
7. "bytes_" - bytes
8. "int_" - int
9. "i_" - int
10. "float_" - float
11. "f_" - float
12. "list_" - list
13. "l_" - list
14. "dict_" - dict
15. "d_" - dict

16. “set_” - set
17. “tuple_” - tuple
18. “t_” - tuple

Example

```
from char import char

@char
def oh_my_god(
    int_arg,
    float_arg,
    list_arg,
    undef_arg,
    d_kwarg=None,
    i_kwarg=0,
    is_kwarg=False
):
    pass

oh_my_god(0, 0.0, [], 1) # Will PASS
oh_my_god(0, 0.0, None, "text") # Will PASS
oh_my_god(0, 0.0, {}, "text") # Will FAIL and raise an ArgumentTypeError
oh_my_god(0, 0.0, [], Exception, d_kwarg={0: 1}) # Will PASS
oh_my_god(0, 0.0, [], object, is_kwarg=0) # Will FAIL and raise an ArgumentTypeError
```

Usage with user defined settings

Decorator arguments

1. **bool_is_to_skip_None_value=True**: Flag what to do with None values, by default None values won't be checked.
2. **dict_tuple_types_by_prefix_to_update_default**: dictionary, which prefixes to add to the default ones
3. **dict_tuple_types_by_prefix**: dictionary, which prefixes to use instead of default ones

You can use any combination of given arguments for the decorator.

For simplicity will be shown usage of every argument separately.

Decorator argument: bool_is_to_skip_None_value

```
@char
def func_with_default_decorator(dict_x):
    pass

@char(bool_is_to_skip_None_value=False)
def func_with_custom_decorator(dict_x):
    pass
```

(continues on next page)

(continued from previous page)

```
func_with_default_decorator(None) # Will PASS
func_with_custom_decorator(None) # Will FAIL and raise an ArgumentTypeError
```

Decorator argument: dict_tuple_types_by_prefix_to_update_default

```
@char(dict_tuple_types_by_prefix_to_update_default={"num_": (int, float, bool)})
def very_complex_function(num_x, str_y=""):
    pass

very_complex_function(0, "hihi") # Will PASS
very_complex_function(0.5, "heyhey") # Will PASS
very_complex_function(True) # Will PASS
very_complex_function("True") # Will FAIL and raise an ArgumentTypeError
```

Decorator argument: dict_tuple_types_by_prefix

```
@char(dict_tuple_types_by_prefix={"exception": (BaseException)})
def function_with_only_one_check(int_x, exception_y=None):
    pass

function_with_only_one_check(0, Exception) # Will PASS
function_with_only_one_check(0.5, TypeError) # Will PASS because first variable won
↳ 't be checked
function_with_only_one_check(0.5, "ERROR") # Will FAIL and raise an ArgumentTypeError
```

Links

- [PYPI](#)
- [readthedocs](#)
- [GitHub](#)

Project local Links

- [CHANGELOG.](#)
- [CONTRIBUTING.](#)

Contacts

- Email: stas.prokopiev@gmail.com
- [vk.com](#)
- [Facebook](#)

License

This project is licensed under the MIT License.

1.2 Module Reference

1.2.1 char

char package

Submodules

char.main module

This is one and only file with working code in the whole package

exception `ArgumentTypeError`

Bases: `TypeError`

Error that type of argument is incorrect

Parameters `TypeError` (*Exception*) – Wrong type of argument was given to function

char (*function=None, dict_tuple_types_by_prefix=None, dict_tuple_types_by_prefix_to_update_default=None, bool_is_to_skip_none_value=True*)

Decorator for checking types of arguments in function

Check is done according to prefixes that was given (or default ones) E.G.

if name of variable starts with `int_` and there is prefix “`int_`” in dict which describe how to check types then if for the argument will be given value with any another type then `ArgumentTypeError` Exception will be raised

Parameters

- **function** (*function, optional*) – To call dec without arguments. Defaults to `None`.
- **dict_tuple_types_by_prefix** (*dict, optional*) – Rules how to check types. Defaults to `None`.
- **dict_tuple_types_by_prefix_to_update_default** (*dict, optional*) – Additional to default Rules how to check types. Defaults to `None`.
- **bool_is_to_skip_none_value** (*bool, optional*) – Flag what to do with `None` values. Defaults to `True`.

Returns Decorator without arguments

Return type function

check_type_of_1_argument (*str_function_name, str_argument_name, argument_value, tuple_types_var_can_be*)

Check type of one argument for function

Parameters

- **str_function_name** (*str*) – Name of function from which method is called
- **str_argument_name** (*str*) – Name of argument to check
- **argument_value** (*Any*) – Value that was given to this argument
- **tuple_types_var_can_be** (*tuple of types*) – Types this arg can be

Raises `ArgumentTypeError` – Wrong type of argument, child from `TypeError`

Module contents

char (*function=None, dict_tuple_types_by_prefix=None, dict_tuple_types_by_prefix_to_update_default=None, bool_is_to_skip_none_value=True*)

Decorator for checking types of arguments in function

Check is done according to prefixes that was given (or default ones) E.G.

if name of variable starts with `int_` and there is prefix “`int_`” in dict which describe how to check types then if for the argument will be given value with any another type then `ArgumentTypeError` Exception will be raised

Parameters

- **function** (*function, optional*) – To call dec without arguments. Defaults to `None`.
- **dict_tuple_types_by_prefix** (*dict, optional*) – Rules how to check types. Defaults to `None`.
- **dict_tuple_types_by_prefix_to_update_default** (*dict, optional*) – Additional to default Rules how to check types. Defaults to `None`.
- **bool_is_to_skip_none_value** (*bool, optional*) – Flag what to do with `None` values. Defaults to `True`.

Returns Decorator without arguments

Return type function

check_arguments (*function=None, dict_tuple_types_by_prefix=None, dict_tuple_types_by_prefix_to_update_default=None, bool_is_to_skip_none_value=True*)

Decorator for checking types of arguments in function

Check is done according to prefixes that was given (or default ones) E.G.

if name of variable starts with `int_` and there is prefix “`int_`” in dict which describe how to check types then if for the argument will be given value with any another type then `ArgumentTypeError` Exception will be raised

Parameters

- **function** (*function, optional*) – To call dec without arguments. Defaults to `None`.
- **dict_tuple_types_by_prefix** (*dict, optional*) – Rules how to check types. Defaults to `None`.
- **dict_tuple_types_by_prefix_to_update_default** (*dict, optional*) – Additional to default Rules how to check types. Defaults to `None`.
- **bool_is_to_skip_none_value** (*bool, optional*) – Flag what to do with `None` values. Defaults to `True`.

Returns Decorator without arguments

Return type function

check_types_of_arguments (*function=None*, *dict_tuple_types_by_prefix=None*,
dict_tuple_types_by_prefix_to_update_default=None,
bool_is_to_skip_none_value=True)

Decorator for checking types of arguments in function

Check is done according to prefixes that was given (or default ones) E.G.

if name of variable starts with **int_** and there is prefix “**int_**” in dict which describe how to check types then if for the argument will be given value with any another type then `ArgumentTypeError` Exception will be raised

Parameters

- **function** (*function*, *optional*) – To call dec without arguments. Defaults to None.
- **dict_tuple_types_by_prefix** (*dict*, *optional*) – Rules how to check types. Defaults to None.
- **dict_tuple_types_by_prefix_to_update_default** (*dict*, *optional*) – Additional to default Rules how to check types. Defaults to None.
- **bool_is_to_skip_none_value** (*bool*, *optional*) – Flag what to do with None values. Defaults to True.

Returns Decorator without arguments

Return type function

1.3 Other

1.3.1 License

The MIT License (MIT)

Copyright (c) 2020 stanislav

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.3.2 Authors

- Stanislav Prokopyev <stas.prokopyev@gmail.com>

Contacts

- email: stas.prokopiev@gmail.com
- vk.com
- [Facebook](https://www.facebook.com)

1.3.3 Changelog

Version 0.1.1

- Updated docs
- Improved the code style of the python files

Version 0.1

- Initial release of this python package

Template 0.x.y

- Feature A added
- FIX: nasty bug #1729 fixed
- add your changes here!

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

C

`char`, 9

`char.main`, 8

A

`ArgumentTypeError`, 8

C

`char` (*module*), 9

`char()` (*in module char*), 9

`char()` (*in module char.main*), 8

`char.main` (*module*), 8

`check_arguments()` (*in module char*), 9

`check_type_of_1_argument()` (*in module char.main*), 8

`check_types_of_arguments()` (*in module char*), 9