Instructions

Release

December 28, 2015

Contents

All developers are doing the same thing everyday - they're searching for some values inside iterable data structures, trying to filter them somehow or count elements inside which satisfy some requirements. We keep writing the same code over and over again from project to project. There's no more need to do this, Instructions to the rescue. Instructions is a library, written in Python to simplify lives of Python developers. This is how we could write some code using Instructions which searches for a string with a length of 3 inside a list of values:

```
>>> instructions.findstring__len(3).inside(['foo', 'bar', 'blah', 1, 2])
```

Instructions does all the hard work for you, just tell it what do you want to do in plain english and it will magically execute your instruction.

CHAPTER 1

Features

- Supports Python 2.6 - 3.4, PyPy and PyPy3
- Supports all built-in Python data types
- Easily extendable
- Extensively documented
- Comes with more than 100 filters

CHAPTER 2

Contacts and Support

I will be glad to get your feedback, pull requests, issues, whatever. Feel free to contact me for any questions.

CHAPTER 3

Donations and Sponsorship

If you like this project and want to support it you have 3 options:

1. Just give this project a star at the GitHub repository.
2. You can express your gratitude via Gratipay.
3. Become a sponsor. Contact me via `tepkeev at gmail dot com` if you are interested in becoming a sponsor and we will discuss the terms and conditions.

CHAPTER 4

Copyright and License

Instructions is licensed under Apache 2.0 license. Check the License for details.

CHAPTER 5

Table of contents

5.1 Installation

5.1.1 PyPI

The recommended way to install is from Python Package Index (PyPI) with pip:

```
$ pip install instructions
```

or with easy_install:

```
$ easy_install instructions
```

5.1.2 GitHub

Instructions is actively developed on GitHub. If you want to get latest development sources you have to clone the repository:

```
$ git clone git://github.com/maxtepkeev/instructions.git
```

Once you have the sources, you can install it into your site-packages:

```
$ python setup.py install
```

You can also install latest stable development version via pip:

```
$ pip install git+https://github.com/maxtepkeev/instructions.git@master
```

5.2 Concepts

Before starting to work with Instructions you have to understand a few concepts which are used across the whole project. There are 3 fundamental blocks: command, datatype and filter. Combined together they form an instruction which can be executed.

5.2.1 Command

Command is what you want your instruction to do, e.g. find, count, filter etc.

5.2.2 Datatype

Datatype is what you're interested in, e.g. string, tuple, dict etc.

5.2.3 Filter

Filter is how you're limiting your result set, e.g. startswith, contains, len etc.

5.2.4 Instruction

Instruction is the combination of 3 previous concepts. Consider the following example:

```
>>> instructions.findint__between(3, 6)
```

Given this example, `find` is the command, `int` is the datatype and `between` is the filter. __ is the divider between first part of the instruction and the filter. `(3, 6)` are the arguments that this filter takes. That means that every instruction can be written as the following:

```
>>> XY__Z(*args, **kwargs)
```

where `X` is the command, `Y` is the datatype, `Z` is the filter, `args` are the filter arguments and `kwargs` are the options that this instruction takes if any.

5.3 Modes

Instructions can operate in two modes: basic and advanced.

5.3.1 Basic

Basic mode should be used if you need just some simple instructions, e.g.:

```
>>> import instructions

>>> instructions.finddict__value_contains_str('foo').inside(container)
```

This will give you all dicts from container which have values that are strings and contain `foo` substring. All examples in the documentation are written using Instruction's basic mode.

5.3.2 Advanced

In advanced mode you can combine different filters together using operators, e.g.:

```
>>> from instructions import commands, datatypes

>>> commands.count(
...     datatypes.string.startswith('foo') & datatypes.string.endswith('bar') & ~datatypes.string.co
... ).inside(container)
```

This will give you the amount of strings inside a container which start with `foo`, end with `bar` but don't contain the `blah` substring, i.e. `foobar` will match but `fooblahbar` won't.

Here's the list of operators that can be used to combine filters together:

- `&` - logical AND operator
- `|` - logical OR operator
- `~` - logical NOT operator

While advanced mode requires one to write a little bit more code, it also gives maximum flexibility and allows to combine different filters of different datatypes together, constructing complex instructions.

5.4 Commands

Command is one of the building blocks of the instruction. Command specifies an action, i.e. what the instruction should do. Each command supports several options as keyword arguments which can influence on results of the instruction:

- `limit` - how many results to return, default is 0, which means to return all results.
- `level` - how deep inside nested iterable data structures to search, default is 0, which means to search inside everything. Let's have a look at the example to better understand this option. Imagine that we want to find all strings using the following code:

  ```
  >>> instructions.findstring(level=0).inside(['foo', ['bar', ['baz']]])
  ```

  Now if level will be set to 1, then only first level iterable will be searched and `foo` will be the only result, if level will be set to 2, then the result will contain `foo` and `bar`, finally if level will be set to 3, the result will contain `foo`, `bar` and `baz`. Level can be set to any positive integer value, e.g. if you set it to 54 then command will be searching 54 levels deep, of course if there are so many levels available, if not it will just stop at the deepest level available and return all the results it found.

- `ignore` - list or tuple of datatypes which should be ignored while searching, default is to search everything. Consider the following example:

  ```
  >>> instructions.findstring(ignore=[tuple]).inside(['foo', ['bar', ('baz',)]])
  ```

  Because ignore is set to a `tuple`, only `foo` and `bar` will be in the search results.

5.4.1 find

Find command is used when you want to find something e.g. a string, an integer or maybe several other datatypes all together inside iterable data structures i.e. list, tuple, dict, set etc. Find command returns it's results in the form of generator object to be memory efficient. That means that to see all the results immediately during debugging phase one needs to iterate over it, `list` is a good candidate to do that, e.g.:

```
>>> instructions.findnumeric__gte(7, level=1).inside([1, 3, 5, 7, 9.3, 11, [99]])
<generator object _command at 0x103ca41e0>

>>> list(instructions.findnumeric__gte(7, level=1).inside([1, 3, 5, 7, 9.3, 11, [99]]))
[7, 9.3, 11]
```

5.4.2 first

First command is the same as find, except that when it finds the first match it returns it and stops searching for other results. This is actually the shortcut to find command with limit set to 1, which means that setting a limit option doesn't make sense for this command.

```
>>> instructions.firstnumeric__gte(7).inside([1, 3, 5, 7, 9.3, 11, [99]])
7
```

5.4.3 last

Last command is the same as find, except that it will return last found result. Last command uses some optimizations to make searching for the last element as fast as possible, that means that if you need only last found result, use this command and not a find command with the last element taken from result. Setting a limit option doesn't make sense for this command.

```
>>> instructions.lastnumeric__gte(7).inside([1, 3, 5, 7, 9.3, 11, [99]])
99
```

5.4.4 exists

Exists command checks whether there is at least one result inside a searchable container. That means that it can only return `True` or `False`. Setting a limit option doesn't make sense for this command.

```
>>> instructions.existsnumeric__gte(7).inside([1, 3, 5, 7, 9.3, 11, [99]])
True
```

5.4.5 count

Count command counts how many results are there inside a searchable container. It returns a number of found results or 0 if nothing is found.

```
>>> instructions.countnumeric__gte(7).inside([1, 3, 5, 7, 9.3, 11, [99]])
4
```

5.5 Datatypes

Datatype is another important block of the instruction. Datatype specifies the target of the instruction, i.e. what the instruction should return. Datatype is also a way to group a set of filters together. While each datatype has it's own filters, there are some of them which are shared across different datatypes. It is also worth noting that each datatype is also a filter by itself, that is why it is possible to use it inside instructions.

Note: All examples are written using Python 2. They may have a slightly different syntax in Python 3.

Note: To better illustrate how datatypes and filters work, all examples in this section will be shown using the `find` command. For the sake of readability all examples won't be wrapped in a `list()` call, however keep in mind that in reality `find` command returns it's results in the form of generator object.

5.5.1 bool

Bool datatype is used to operate on Python's boolean type. If there is no need to apply any filter, but just to get all the booleans from a searchable container, one can use this code:

```
>>> instructions.findbool().inside(['foo', True, 1, False, 5, True])
[True, False, True]
```

exact

An exact match.

```
>>> instructions.findbool__exact(True).inside(['foo', True, 1, False, 5, True])
[True, True]
```

true

Limits results to all `True` values.

```
>>> instructions.findbool__true().inside(['foo', True, 1, False, 5, True])
[True, True]
```

false

Limits results to all `False` values.

```
>>> instructions.findbool__false().inside(['foo', True, 1, False, 5, True])
[False]
```

5.5.2 string

String datatype is an aggregated datatype which changes it's behaviour under different Python versions. If used with Python 2, it will operate on `str`, `unicode` and `bytearray` Python types, while on Python 3 the `str`, `bytes` and `bytearray` will be it's targets. If there is no need to apply any filter, but just to get all the strings from a searchable container, one can use this code:

```
>>> instructions.findstring().inside(['foo', True, 1, 'bar', 5, bytearray(b'baz')])
['foo', 'bar', bytearray(b'baz')]
```

exact

An exact match.

```
>>> instructions.findstring__exact('foo').inside(['foo', True, 1, 'bar', 5, bytearray(b'baz')])
['foo']
```

iexact

Case-insensitive version of the exact filter.

```
>>> instructions.findstring__iexact('foo').inside(['foo', True, 1, 'FOO', 5, 'bar'])
['foo', 'FOO']
```

contains

Checks that a string contains another string.

```
>>> instructions.findstring__contains('o').inside(['foo', True, 1, 'FOO', 5, 'bar'])
['foo']
```

icontains

Case-insensitive version of the contains filter.

```
>>> instructions.findstring__icontains('o').inside(['foo', True, 1, 'FOO', 5, 'bar'])
['foo', 'FOO']
```

startswith

Checks that a string starts with another string.

```
>>> instructions.findstring__startswith('f').inside(['foo', True, 1, 'FOO', 5, 'bar'])
['foo']
```

istartswith

Case-insensitive version of the startswith filter.

```
>>> instructions.findstring__istartswith('f').inside(['foo', True, 1, 'FOO', 5, 'bar'])
['foo', 'FOO']
```

endswith

Checks that a string ends with another string.

```
>>> instructions.findstring__endswith('r').inside(['foo', True, 1, 'BAR', 5, 'bar'])
['bar']
```

iendswith

Case-insensitive version of the endswith filter.

```
>>> instructions.findstring__iendswith('r').inside(['foo', True, 1, 'BAR', 5, 'bar'])
['BAR', 'bar']
```

len

Checks that a string has specified length.

```
>>> instructions.findstring__len(3).inside(['foo', True, 1, 'blah', 5, 'bar'])
['foo', 'bar']
```

lenlt

Checks that a string has length less than specified.

```
>>> instructions.findstring__lenlt(4).inside(['foo', True, 1, 'blah', 5, 'bar'])
['foo', 'bar']
```

lenlte

Checks that a string has length less than or equal to specified.

```
>>> instructions.findstring__lenlte(4).inside(['foo', True, 1, 'blah', 5, 'bar'])
['foo', 'blah', 'bar']
```

lengt

Checks that a string has length greater than specified.

```
>>> instructions.findstring__lengt(3).inside(['foo', True, 1, 'blah', 5, 'bar'])
['blah']
```

lengte

Checks that a string has length greater than or equal to specified.

```
>>> instructions.findstring__lengte(3).inside(['foo', True, 1, 'blah', 5, 'bar'])
['foo', 'blah', 'bar']
```

isalnum

Checks that all characters in the string are alphanumeric.

```
>>> instructions.findstring__isalnum().inside(['foo', True, 1, 'blah', 5, 'bar'])
['foo', 'blah', 'bar']
```

isalnums

Checks that all characters in the string are alphanumeric or space.

```
>>> instructions.findstring__isalnums().inside(['foo', True, 1, 'b lah', 5, 'b ar'])
['foo', 'b lah', 'b ar']
```

isalpha

Checks that all characters in the string are alphabetic.

```
>>> instructions.findstring__isalpha().inside(['foo', True, 1, 'blah', 5, 'bar'])
['foo', 'blah', 'bar']
```

isalphas

Checks that all characters in the string are alphabetic or space.

```
>>> instructions.findstring__isalphas().inside(['fo o', True, 1, 'blah', 5, 'b ar'])
['fo o', 'blah', 'b ar']
```

isdigit

Checks that all characters in the string are digits.

```
>>> instructions.findstring__isalpha().inside(['foo', True, 1, '1', 5, '2'])
['1', '2']
```

islower

Checks that all characters in the string are lowercase.

```
>>> instructions.findstring__islower().inside(['foo', True, 1, 'BLAH', 5, 'bar'])
['foo', 'bar']
```

isupper

Checks that all characters in the string are uppercase.

```
>>> instructions.findstring__isupper().inside(['foo', True, 1, 'BLAH', 5, 'bar'])
['BLAH']
```

isspace

Checks that there are only whitespace characters in the string.

```
>>> instructions.findstring__isspace().inside(['foo', True, 1, '   ', 5, 'bar'])
['   ']
```

istitle

Checks that the string is a titlecased string.

```
>>> instructions.findstring__istitle().inside(['Foo', True, 1, 'blah', 5, 'bar'])
['Foo']
```

5.5.3 unicode

Unicode datatype changes it's behaviour under different Python versions. If used with Python 2, it will operate on `unicode` Python type, while on Python 3 the `str` will be it's target. If there is no need to apply any filter, but just to get all the unicode strings from a searchable container, one can use this code:

```
>>> instructions.findunicode().inside([u'foo', True, 1, 'bar', 5, u'baz'])
[u'foo', u'baz']
```

exact

An exact match.

```
>>> instructions.findunicode__exact(u'foo').inside([u'foo', True, 1, 'bar', 5, u'baz'])
[u'foo']
```

iexact

Case-insensitive version of the exact filter.

```
>>> instructions.findunicode__iexact(u'foo').inside([u'foo', True, 1, u'FOO', 5, u'bar'])
[u'foo', u'FOO']
```

contains

Checks that a unicode string contains another unicode string.

```
>>> instructions.findunicode__contains(u'o').inside([u'foo', True, 1, u'FOO', 5, u'bar'])
[u'foo']
```

icontains

Case-insensitive version of the contains filter.

```
>>> instructions.findunicode__icontains(u'o').inside([u'foo', True, 1, u'FOO', 5, u'bar'])
[u'foo', u'FOO']
```

startswith

Checks that a unicode string starts with another unicode string.

```
>>> instructions.findunicode__startswith(u'f').inside([u'foo', True, 1, u'FOO', 5, u'bar'])
[u'foo']
```

istartswith

Case-insensitive version of the startswith filter.

```
>>> instructions.findunicode__istartswith(u'f').inside([u'foo', True, 1, u'FOO', 5, u'bar'])
[u'foo', u'FOO']
```

endswith

Checks that a unicode string ends with another unicode string.

```
>>> instructions.findunicode__endswith(u'r').inside([u'foo', True, 1, u'BAR', 5, u'bar'])
[u'bar']
```

iendswith

Case-insensitive version of the endswith filter.

```
>>> instructions.findunicode__iendswith(u'r').inside([u'foo', True, 1, u'BAR', 5, u'bar'])
[u'BAR', u'bar']
```

len

Checks that a unicode string has specified length.

```
>>> instructions.findunicode__len(3).inside([u'foo', True, 1, u'blah', 5, u'bar'])
[u'foo', u'bar']
```

lenlt

Checks that a unicode string has length less than specified.

```
>>> instructions.findunicode__lenlt(4).inside([u'foo', True, 1, u'blah', 5, u'bar'])
[u'foo', u'bar']
```

lenlte

Checks that a unicode string has length less than or equal to specified.

```
>>> instructions.findunicode__lenlte(4).inside([u'foo', True, 1, u'blah', 5, u'bar'])
[u'foo', u'blah', u'bar']
```

lengt

Checks that a unicode string has length greater than specified.

```
>>> instructions.findunicode__lengt(3).inside([u'foo', True, 1, u'blah', 5, u'bar'])
[u'blah']
```

lengte

Checks that a unicode string has length greater than or equal to specified.

```
>>> instructions.findunicode__lengte(3).inside([u'foo', True, 1, u'blah', 5, u'bar'])
[u'foo', u'blah', u'bar']
```

isalnum

Checks that all characters in the unicode string are alphanumeric.

```
>>> instructions.findunicode__isalnum().inside([u'foo', True, 1, u'blah', 5, u'bar'])
[u'foo', u'blah', u'bar']
```

isalnums

Checks that all characters in the unicode string are alphanumeric or space.

```
>>> instructions.findunicode__isalnums().inside([u'foo', True, 1, u'b lah', 5, u'b ar'])
[u'foo', u'b lah', u'b ar']
```

isalpha

Checks that all characters in the unicode string are alphabetic.

```
>>> instructions.findunicode__isalpha().inside([u'foo', True, 1, u'blah', 5, u'bar'])
[u'foo', u'blah', u'bar']
```

isalphas

Checks that all characters in the unicode string are alphabetic or space.

```
>>> instructions.findunicode__isalphas().inside([u'fo o', True, 1, u'blah', 5, u'b ar'])
[u'fo o', u'blah', u'b ar']
```

isdigit

Checks that all characters in the unicode string are digits.

```
>>> instructions.findunicode__isalpha().inside([u'foo', True, 1, u'1', 5, u'2'])
[u'1', u'2']
```

islower

Checks that all characters in the unicode string are lowercase.

```
>>> instructions.findunicode__islower().inside([u'foo', True, 1, u'BLAH', 5, u'bar'])
[u'foo', u'bar']
```

isupper

Checks that all characters in the unicode string are uppercase.

```
>>> instructions.findunicode__isupper().inside([u'foo', True, 1, u'BLAH', 5, u'bar'])
[u'BLAH']
```

isspace

Checks that there are only whitespace characters in the unicode string.

```
>>> instructions.findunicode__isspace().inside([u'foo', True, 1, u'   ', 5, u'bar'])
['   ']
```

istitle

Checks that the unicode string is a titlecased string.

```
>>> instructions.findunicode__istitle().inside([u'Foo', True, 1, u'blah', 5, u'bar'])
[u'Foo']
```

isnumeric

Checks that all characters in the unicode string are numeric.

```
>>> instructions.findunicode__isnumeric().inside([u'Foo', True, 1, u'', 5, u'bar'])
[u'']
```

isdecimal

Checks that all characters in the unicode string are decimal.

```
>>> instructions.findunicode__isdecimal().inside([u'Foo', True, 1, u'', 5, u'bar'])
[u'']
```

5.5.4 bytes

Bytes datatype changes it's behaviour under different Python versions. If used with Python 2, it will operate on `str` Python type, while on Python 3 the `bytes` will be it's target. If there is no need to apply any filter, but just to get all the byte strings from a searchable container, one can use this code:

```
>>> instructions.findbytes().inside([b'foo', True, 1, b'bar', 5, u'baz'])
[b'foo', b'bar']
```

exact

An exact match.

```
>>> instructions.findbytes__exact(b'foo').inside([b'foo', True, 1, b'bar', 5, u'baz'])
[b'foo']
```

iexact

Case-insensitive version of the exact filter.

```
>>> instructions.findbytes__iexact(b'foo').inside([b'foo', True, 1, b'FOO', 5, b'bar'])
[b'foo', b'FOO']
```

contains

Checks that a byte string contains another byte string.

```
>>> instructions.findbytes__contains(b'o').inside([b'foo', True, 1, b'FOO', 5, b'bar'])
[b'foo']
```

icontains

Case-insensitive version of the contains filter.

```
>>> instructions.findbytes__icontains(b'o').inside([b'foo', True, 1, b'FOO', 5, b'bar'])
[b'foo', b'FOO']
```

startswith

Checks that a byte string starts with another byte string.

```
>>> instructions.findbytes__startswith(b'f').inside([b'foo', True, 1, b'FOO', 5, b'bar'])
[b'foo']
```

istartswith

Case-insensitive version of the startswith filter.

```
>>> instructions.findbytes__istartswith(b'f').inside([b'foo', True, 1, b'FOO', 5, b'bar'])
[b'foo', b'FOO']
```

endswith

Checks that a byte string ends with another byte string.

```
>>> instructions.findbytes__endswith(b'r').inside([b'foo', True, 1, b'BAR', 5, b'bar'])
[b'bar']
```

iendswith

Case-insensitive version of the endswith filter.

```
>>> instructions.findbytes__iendswith(b'r').inside([b'foo', True, 1, b'BAR', 5, b'bar'])
[b'BAR', b'bar']
```

len

Checks that a byte string has specified length.

```
>>> instructions.findbytes__len(3).inside([b'foo', True, 1, b'blah', 5, b'bar'])
[b'foo', b'bar']
```

lenlt

Checks that a byte string has length less than specified.

```
>>> instructions.findbytes__lenlt(4).inside([b'foo', True, 1, b'blah', 5, b'bar'])
[b'foo', b'bar']
```

lenlte

Checks that a byte string has length less than or equal to specified.

```
>>> instructions.findbytes__lenlte(4).inside([b'foo', True, 1, b'blah', 5, b'bar'])
[b'foo', b'blah', b'bar']
```

lengt

Checks that a byte string has length greater than specified.

```
>>> instructions.findbytes__lengt(3).inside([b'foo', True, 1, b'blah', 5, b'bar'])
[b'blah']
```

lengte

Checks that a byte string has length greater than or equal to specified.

```
>>> instructions.findbytes__lengte(3).inside([b'foo', True, 1, b'blah', 5, b'bar'])
[b'foo', b'blah', b'bar']
```

isalnum

Checks that all bytes in the byte string are alphanumeric.

```
>>> instructions.findbytes__isalnum().inside([b'foo', True, 1, b'blah', 5, b'bar'])
[b'foo', b'blah', b'bar']
```

isalnums

Checks that all bytes in the byte string are alphanumeric or space.

```
>>> instructions.findbytes__isalnums().inside([b'foo', True, 1, b'b lah', 5, b'b ar'])
[b'foo', b'b lah', b'b ar']
```

isalpha

Checks that all bytes in the byte string are alphabetic.

```
>>> instructions.findbytes__isalpha().inside([b'foo', True, 1, b'blah', 5, b'bar'])
[b'foo', b'blah', b'bar']
```

isalphas

Checks that all bytes in the byte string are alphabetic or space.

```
>>> instructions.findbytes__isalphas().inside([b'fo o', True, 1, b'blah', 5, b'b ar'])
[b'fo o', b'blah', b'b ar']
```

isdigit

Checks that all bytes in the byte string are digits.

```
>>> instructions.findbytes__isalpha().inside([b'foo', True, 1, b'1', 5, b'2'])
[b'1', b'2']
```

islower

Checks that all bytes in the byte string are lowercase.

```
>>> instructions.findbytes__islower().inside([b'foo', True, 1, b'BLAH', 5, b'bar'])
[b'foo', b'bar']
```

isupper

Checks that all bytes in the byte string are uppercase.

```
>>> instructions.findbytes__isupper().inside([b'foo', True, 1, b'BLAH', 5, b'bar'])
[b'BLAH']
```

isspace

Checks that there are only whitespace bytes in the byte string.

```
>>> instructions.findbytes__isspace().inside([b'foo', True, 1, b'   ', 5, b'bar'])
[b'   ']
```

istitle

Checks that the byte string is a titlecased string.

```
>>> instructions.findbytes__istitle().inside([b'Foo', True, 1, b'blah', 5, b'bar'])
[b'Foo']
```

5.5.5 bytearray

Bytearray datatype is used to operate on Python's bytearray type. If there is no need to apply any filter, but just to get all the bytearrays from a searchable container, one can use this code:

```
>>> instructions.findbytearray().inside([bytearray(b'foo'), True, 1, bytearray(b'bar'), 5, u'baz'])
[bytearray(b'foo'), bytearray(b'bar')]
```

exact

An exact match.

```
>>> instructions.findbytearray__exact(bytearray(b'foo')).inside([bytearray(b'foo'), True, 1, bytearra
[bytearray(b'foo')]
```

iexact

Case-insensitive version of the exact filter.

```
>>> instructions.findbytearray__iexact(bytearray(b'foo')).inside([bytearray(b'foo'), True, 1, bytearr
[bytearray(b'foo'), bytearray(b'FOO')]
```

contains

Checks that a bytearray contains another byte string or bytearray.

```
>>> instructions.findbytearray__contains(b'o').inside([bytearray(b'foo'), True, 1, bytearray(b'FOO'),
[bytearray(b'foo')]
```

icontains

Case-insensitive version of the contains filter.

```
>>> instructions.findbytearray__icontains(b'o').inside([bytearray(b'foo'), True, 1, bytearray(b'FOO')
[bytearray(b'foo'), bytearray(b'FOO')]
```

startswith

Checks that a bytearray starts with another byte string or bytearray.

```
>>> instructions.findbytearray__startswith(b'f').inside([bytearray(b'foo'), True, 1, bytearray(b'FOO
[bytearray(b'foo')]
```

istartswith

Case-insensitive version of the startswith filter.

```
>>> instructions.findbytearray__istartswith(b'f').inside([bytearray(b'foo'), True, 1, bytearray(b'FO
[bytearray(b'foo'), bytearray(b'FOO')]
```

endswith

Checks that a bytearray ends with another byte string or bytearray.

```
>>> instructions.findbytearray__endswith(b'r').inside([bytearray(b'foo'), True, 1, bytearray(b'BAR'),
[bytearray(b'bar')]
```

iendswith

Case-insensitive version of the endswith filter.

```
>>> instructions.findbytearray__iendswith(b'r').inside([bytearray(b'foo'), True, 1, bytearray(b'BAR')
[bytearray(b'BAR'), bytearray(b'bar')]
```

len

Checks that a bytearray has specified length.

```
>>> instructions.findbytearray__len(3).inside([bytearray(b'foo'), True, 1, bytearray(b'blah'), 5, byt
[bytearray(b'foo'), bytearray(b'bar')]
```

lenlt

Checks that a bytearray has length less than specified.

```
>>> instructions.findbytearray__lenlt(4).inside([bytearray(b'foo'), True, 1, bytearray(b'blah'), 5, b
[bytearray(b'foo'), bytearray(b'bar')]
```

lenlte

Checks that a bytearray has length less than or equal to specified.

```
>>> instructions.findbytearray__lenlte(4).inside([bytearray(b'foo'), True, 1, bytearray(b'blah'), 5,
[bytearray(b'foo'), bytearray(b'blah'), bytearray(b'bar')]
```

lengt

Checks that a bytearray has length greater than specified.

```
>>> instructions.findbytearray__lengt(3).inside([bytearray(b'foo'), True, 1, bytearray(b'blah'), 5, b
[bytearray(b'blah')]
```

lengte

Checks that a bytearray has length greater than or equal to specified.

```
>>> instructions.findbytearray__lengte(3).inside([bytearray(b'foo'), True, 1, bytearray(b'blah'), 5,
[bytearray(b'foo'), bytearray(b'blah'), bytearray(b'bar')]
```

isalnum

Checks that all bytes in the bytearray are alphanumeric.

```
>>> instructions.findbytearray__isalnum().inside([bytearray(b'foo'), True, 1, bytearray(b'blah'), 5,
[bytearray(b'foo'), bytearray(b'blah'), bytearray(b'bar')]
```

isalnums

Checks that all bytes in the bytearray are alphanumeric or space.

```
>>> instructions.findbytearray__isalnums().inside([bytearray(b'foo'), True, 1, bytearray(b'b lah'), 5
[bytearray(b'foo'), bytearray(b'b lah'), bytearray(b'b ar')]
```

isalpha

Checks that all bytes in the bytearray are alphabetic.

```
>>> instructions.findbytearray__isalpha().inside([bytearray(b'foo'), True, 1, bytearray(b'blah'), 5,
[bytearray(b'foo'), bytearray(b'blah'), bytearray(b'bar')]
```

isalphas

Checks that all bytes in the bytearray are alphabetic or space.

```
>>> instructions.findbytearray__isalphas().inside([bytearray(b'fo o'), True, 1, bytearray(b'blah'), 5
[bytearray(b'fo o'), bytearray(b'blah'), bytearray(b'b ar')]
```

isdigit

Checks that all bytes in the bytearray are digits.

```
>>> instructions.findbytearray__isalpha().inside([bytearray(b'foo'), True, 1, bytearray(b'1'), 5, byt
[bytearray(b'1'), bytearray(b'2')]
```

islower

Checks that all bytes in the bytearray are lowercase.

```
>>> instructions.findbytearray__islower().inside([bytearray(b'foo'), True, 1, bytearray(b'BLAH'), 5,
[bytearray(b'foo'), bytearray(b'bar')]
```

isupper

Checks that all bytes in the bytearray are uppercase.

```
>>> instructions.findbytearray__isupper().inside([bytearray(b'foo'), True, 1, bytearray(b'BLAH'), 5,
[bytearray(b'BLAH')]
```

isspace

Checks that there are only whitespace bytes in the bytearray.

```
>>> instructions.findbytearray__isspace().inside([bytearray(b'foo'), True, 1, bytearray(b'    '), 5, b
[bytearray(b'    ')]
```

istitle

Checks that the bytearray is a titlecased string.

```
>>> instructions.findbytearray__istitle().inside([bytearray(b'Foo'), True, 1, bytearray(b'blah'), 5,
[bytearray(b'Foo')]
```

5.5.6 numeric

Numeric datatype is an aggregated datatype which changes it's behaviour under different Python versions. If used with Python 2, it will operate on `int`, `float` and `long` Python types, while on Python 3 the `int` and `float` will be it's targets. If there is no need to apply any filter, but just to get all the numerics from a searchable container, one can use this code:

```
>>> instructions.findnumeric().inside(['foo', True, 1, 'bar', 5, 9.32])
[1, 5, 9.32]
```

exact

An exact match.

```
>>> instructions.findnumeric__exact(1).inside(['foo', True, 1, 'bar', 5, 9.32])
[1]
```

gt

Checks that a numeric is greater than specified.

```
>>> instructions.findnumeric__gt(5).inside(['foo', True, 1, 'bar', 5, 9.32])
[9.32]
```

gte

Checks that a numeric is greater than or equal to specified.

```
>>> instructions.findnumeric__gte(5).inside(['foo', True, 1, 'bar', 5, 9.32])
[5, 9.32]
```

lt

Checks that a numeric is less than specified.

```
>>> instructions.findnumeric__lt(7).inside(['foo', True, 1, 'bar', 5, 9.32])
[1, 5]
```

lte

Checks that a numeric is less than or equal to specified.

```
>>> instructions.findnumeric__lte(9.5).inside(['foo', True, 1, 'bar', 5, 9.32])
[1, 5, 9.32]
```

between

Inclusively checks that a numeric is between two other numerics.

```
>>> instructions.findnumeric__between(5, 10).inside(['foo', True, 1, 'bar', 5, 9.32])
[5, 9.32]
```

ebetween

Exclusively checks that a numeric is between two other numerics.

```
>>> instructions.findnumeric__ebetween(5, 10).inside(['foo', True, 1, 'bar', 5, 9.32])
[9.32]
```

isodd

Checks that a numeric is odd. If the numeric is a float, it is casted to an int.

```
>>> instructions.findnumeric__isodd().inside(['foo', True, 1, 'bar', 5, 9.32])
[1, 5, 9.32]
```

iseven

Checks that a numeric is even. If the numeric is a float, it is casted to an int.

```
>>> instructions.findnumeric__iseven().inside(['foo', True, 1, 'bar', 2, 9.32])
[2]
```

divisibleby

Checks that a numeric is divisible by specified. If the numeric is a float, it is casted to an int.

```
>>> instructions.findnumeric__divisibleby(2).inside(['foo', True, 1, 'bar', 4, 9.32])
[4]
```

5.5.7 int

Int datatype is used to operate on Python's int type. If there is no need to apply any filter, but just to get all the ints from a searchable container, one can use this code:

```
>>> instructions.findint().inside(['foo', True, 1, 'bar', 5, 9.32])
[1, 5]
```

exact

An exact match.

```
>>> instructions.findint__exact(1).inside(['foo', True, 1, 'bar', 5, 9.32])
[1]
```

gt

Checks that an int is greater than specified.

```
>>> instructions.findint__gt(4).inside(['foo', True, 1, 'bar', 5, 9.32])
[5]
```

gte

Checks that an int is greater than or equal to specified.

```
>>> instructions.findint__gte(5).inside(['foo', True, 1, 'bar', 5, 9.32])
[5]
```

lt

Checks that an int is less than specified.

```
>>> instructions.findint__lt(7).inside(['foo', True, 1, 'bar', 5, 9.32])
[1, 5]
```

lte

Checks that an int is less than or equal to specified.

```
>>> instructions.findint__lte(5).inside(['foo', True, 1, 'bar', 5, 9.32])
[1, 5]
```

between

Inclusively checks that an int is between two other ints.

```
>>> instructions.findint__between(5, 10).inside(['foo', True, 1, 'bar', 5, 9.32])
[5]
```

ebetween

Exclusively checks that an int is between two other ints.

```
>>> instructions.findint__ebetween(4, 10).inside(['foo', True, 1, 'bar', 5, 9.32])
[5]
```

isodd

Checks that an int is odd.

```
>>> instructions.findint__isodd().inside(['foo', True, 1, 'bar', 5, 9.32])
[1, 5]
```

iseven

Checks that an int is even.

```
>>> instructions.findint__iseven().inside(['foo', True, 1, 'bar', 2, 9.32])
[2]
```

divisibleby

Checks that an int is divisible by specified.

```
>>> instructions.findint__divisibleby(2).inside(['foo', True, 1, 'bar', 4, 9.32])
[4]
```

5.5.8 float

Float datatype is used to operate on Python's float type. If there is no need to apply any filter, but just to get all the floats from a searchable container, one can use this code:

```
>>> instructions.findfloat().inside(['foo', True, 1.0, 'bar', 5, 9.32])
[1.0, 9.32]
```

exact

An exact match.

```
>>> instructions.findfloat__exact(9.32).inside(['foo', True, 1.0, 'bar', 5, 9.32])
[9.32]
```

gt

Checks that a float is greater than specified.

```
>>> instructions.findfloat__gt(5).inside(['foo', True, 1.0, 'bar', 5, 9.32])
[9.32]
```

gte

Checks that a float is greater than or equal to specified.

```
>>> instructions.findfloat__gte(5).inside(['foo', True, 1, 'bar', 5.0, 9.32])
[5.0, 9.32]
```

lt

Checks that a float is less than specified.

```
>>> instructions.findfloat__lt(7).inside(['foo', True, 1.0, 'bar', 5, 9.32])
[1.0]
```

lte

Checks that a float is less than or equal to specified.

```
>>> instructions.findfloat__lte(5).inside(['foo', True, 1, 'bar', 5.0, 9.32])
[5.0]
```

between

Inclusively checks that a float is between two other numerics.

```
>>> instructions.findfloat__between(5, 10).inside(['foo', True, 1, 'bar', 5.0, 9.32])
[5.0, 9.32]
```

ebetween

Exclusively checks that a float is between two other numerics.

```
>>> instructions.findfloat__ebetween(5, 10).inside(['foo', True, 1, 'bar', 5, 9.32])
[9.32]
```

isodd

Checks that a float is odd. Float is casted to an int before applying the filter.

```
>>> instructions.findfloat__isodd().inside(['foo', True, 1, 'bar', 5.0, 9.32])
[5.0, 9.32]
```

iseven

Checks that a float is even. Float is casted to an int before applying the filter.

```
>>> instructions.findfloat__iseven().inside(['foo', True, 1, 'bar', 2.02, 9.32])
[2.02]
```

divisibleby

Checks that a float is divisible by specified. Float is casted to an int before applying the filter.

```
>>> instructions.findfloat__divisibleby(2).inside(['foo', True, 1, 'bar', 4.34, 9.32])
[4.34]
```

isinteger

Checks that a float is finite with integral value.

```
>>> instructions.findfloat__isinteger().inside(['foo', True, 1.0, 'bar', 4.0, 9.32])
[1.0, 4.0]
```

5.5.9 long

Long datatype is used to operate on Python's long type, because there is no more long type in Python 3, Instructions will emulate it for you. If there is no need to apply any filter, but just to get all the longs from a searchable container, one can use this code:

```
>>> instructions.findlong().inside(['foo', True, 2 ** 64, 'bar', 5, 3 ** 64])
[18446744073709551616L, 3433683820292512484657849089281L]
```

exact

An exact match.

```
>>> instructions.findlong__exact(2 ** 64).inside(['foo', True, 2 ** 64, 'bar', 5, 3 ** 64])
[18446744073709551616L]
```

gt

Checks that a long is greater than specified.

```
>>> instructions.findlong__gt(2 ** 64).inside(['foo', True, 2 ** 64, 'bar', 5, 3 ** 64])
[3433683820292512484657849089281L]
```

gte

Checks that a long is greater than or equal to specified.

```
>>> instructions.findlong__gte(2 ** 64).inside(['foo', True, 2 ** 64, 'bar', 5, 3 ** 64])
[18446744073709551616L, 3433683820292512484657849089281L]
```

lt

Checks that a long is less than specified.

```
>>> instructions.findlong__lt(3 ** 64).inside(['foo', True, 2 ** 64, 'bar', 5, 3 ** 64])
[18446744073709551616L]
```

lte

Checks that a long is less than or equal to specified.

```
>>> instructions.findlong__lte(3 ** 64).inside(['foo', True, 2 ** 64, 'bar', 5, 3 ** 64])
[18446744073709551616L, 3433683820292512484657849089281L]
```

between

Inclusively checks that a long is between two other longs.

```
>>> instructions.findlong__between(2 ** 64, 3 ** 64).inside(['foo', True, 2 ** 64, 'bar', 5, 3 ** 64]
[18446744073709551616L, 3433683820292512484657849089281L]
```

ebetween

Exclusively checks that a long is between two other longs.

```
>>> instructions.findlong__ebetween(2 ** 64, 3 ** 64).inside(['foo', True, 2 ** 64, 'bar', 5, 3 ** 6
[]
```

isodd

Checks that a long is odd.

```
>>> instructions.findlong__isodd().inside(['foo', True, 2 ** 64, 'bar', 5, 3 ** 64])
   [3433683820292512484657849089281L]
```

iseven

Checks that a long is even.

```
>>> instructions.findlong__iseven().inside(['foo', True, 2 ** 64, 'bar', 2, 3 ** 64])
[18446744073709551616L]
```

divisibleby

Checks that a long is divisible by specified.

```
>>> instructions.findlong__divisibleby(2).inside(['foo', True, 2 ** 64, 'bar', 4, 3 ** 64])
[18446744073709551616L]
```

5.5.10 complex

Complex datatype is used to operate on Python's complex type. If there is no need to apply any filter, but just to get all the complex numbers from a searchable container, one can use this code:

```
>>> instructions.findcomplex().inside(['foo', True, 1j, 'bar', 5, 9.32])
[1j]
```

exact

An exact match.

```
>>> instructions.findcomplex__exact(1j).inside(['foo', True, 1j, 'bar', 5, 9.32])
[1j]
```

5.5.11 iterable

Iterable datatype is used to operate on all Python's iterable types which are instances of `collections.Iterable` except for `str`, `unicode` and `bytearray` in Python 2 and `str`, `bytes` and `bytearray` in Python 3. If there is no need to apply any filter but just to get all the iterables from a searchable container, one can use this code:

```
>>> instructions.finditerable().inside(['foo', True, {'a': 'b'}, ['bar', 5], (9.32,)])
[{'a': 'b'}, ['bar', 5], (9.32,)]
```

exact

An exact match.

```
>>> instructions.finditerable__exact({'a' : 'b'}).inside(['foo', True, {'a': 'b'}, ['bar', 5], (9.32,
[{'a': 'b'}]
```

len

Checks that an iterable has specified length.

```
>>> instructions.finditerable__len(2).inside(['foo', True, {'a': 'b'}, ['bar', 5], (9.32,)])
[['bar', 5]]
```

lenlt

Checks that an iterable has length less than specified.

```
>>> instructions.finditerable__lenlt(2).inside(['foo', True, {'a': 'b'}, ['bar', 5], (9.32,)])
[{'a': 'b'}, (9.32,)]
```

lenlte

Checks that an iterable has length less than or equal to specified.

```
>>> instructions.finditerable__lenlte(2).inside(['foo', True, {'a': 'b'}, ['bar', 5], (9.32,)])
[{'a': 'b'}, ['bar', 5], (9.32,)]
```

lengt

Checks that an iterable has length greater than specified.

```
>>> instructions.finditerable__lengt(1).inside(['foo', True, {'a': 'b'}, ['bar', 5], (9.32,)])
[['bar', 5]]
```

lengte

Checks that an iterable has length greater than or equal to specified.

```
>>> instructions.finditerable__lengte(1).inside(['foo', True, {'a': 'b'}, ['bar', 5], (9.32,)])
[{'a': 'b'}, ['bar', 5], (9.32,)]
```

contains

Checks that an iterable contains the specified value.

```
>>> instructions.finditerable__contains('bar').inside(['foo', True, ('foo', 'baz'), ['bar', 5], ('ba
[['bar', 5], ('bar', 9.32)]
```

contains_all

Checks that an iterable contains all specified values.

```
>>> instructions.finditerable__contains_all(['foo', 'baz']).inside(['foo', True, ('foo', 'baz'), ['ba
[('foo', 'baz')]
```

contains_any

Checks that an iterable contains any of specified values.

```
>>> instructions.finditerable__contains_any(['foo', 'bar']).inside(['foo', True, ('foo', 'baz'), ['ba
[('foo', 'baz'), ['bar', 5], ('bar', 9.32)]
```

str_contains_str

Checks that an iterable contains at least one string, which contains specified substring.

```
>>> instructions.finditerable__str_contains_str('ba').inside(['foo', True, ('foo', 'baz'), ['bar', 5
[('foo', 'baz'), ['bar', 5], ('bar', 9.32)]
```

5.5.12 list

List datatype is used to operate on Python's list type. If there is no need to apply any filter, but just to get all the lists from a searchable container, one can use this code:

```
>>> instructions.findlist().inside(['foo', True, {'a': 'b'}, ['bar', 5], (9.32,)])
[['bar', 5]]
```

exact

An exact match.

```
>>> instructions.findlist__exact([9.32]).inside(['foo', True, ['foo', 'bar'], ['bar', 5], [9.32]])
[[9.32]]
```

len

Checks that a list has specified length.

```
>>> instructions.findlist__len(2).inside(['foo', True, ['foo', 'bar'], ['bar', 5], [9.32]])
[['foo', 'bar'], ['bar', 5]]
```

lenlt

Checks that a list has length less than specified.

```
>>> instructions.findlist__lenlt(2).inside(['foo', True, ['foo', 'bar'], ['bar', 5], [9.32]])
[[9.32]]
```

lenlte

Checks that a list has length less than or equal to specified.

```
>>> instructions.findlist__lenlte(2).inside(['foo', True, ['foo', 'bar'], ['bar', 5], [9.32]])
[['foo', 'bar'], ['bar', 5], [9.32]]
```

lengt

Checks that a list has length greater than specified.

```
>>> instructions.findlist__lengt(1).inside(['foo', True, ['foo', 'bar'], ['bar', 5], [9.32]])
[['foo', 'bar'], ['bar', 5]]
```

lengte

Checks that a list has length greater than or equal to specified.

```
>>> instructions.findlist__lengte(1).inside(['foo', True, ['foo', 'bar'], ['bar', 5], [9.32]])
[['foo', 'bar'], ['bar', 5], [9.32]]
```

contains

Checks that a list contains the specified value.

```
>>> instructions.findlist__contains('bar').inside(['foo', True, ['foo', 'bar'], ['bar', 5], [9.32]])
[['foo', 'bar'], ['bar', 5]]
```

contains_all

Checks that a list contains all specified values.

```
>>> instructions.findlist__contains_all(['foo', 'baz']).inside(['foo', True, ['foo', 'baz'], ['bar',
[['foo', 'baz']]
```

contains_any

Checks that a list contains any of specified values.

```
>>> instructions.findlist__contains_any(['foo', 'bar']).inside(['foo', True, ['foo', 'bar'], ['bar',
[['foo', 'bar'], ['bar', 5]]
```

str_contains_str

Checks that a list contains at least one string, which contains specified substring.

```
>>> instructions.findlist__str_contains_str('ba').inside(['foo', True, ['foo', 'bar'], ['bar', 5], [
[['foo', 'bar'], ['bar', 5]]
```

5.5.13 tuple

Tuple datatype is used to operate on Python's tuple type. If there is no need to apply any filter, but just to get all the tuples from a searchable container, one can use this code:

```
>>> instructions.findtuple().inside(['foo', True, {'a': 'b'}, ['bar', 5], (9.32,)])
[(9.32,)]
```

exact

An exact match.

```
>>> instructions.findtuple__exact((9.32,)).inside(['foo', True, ('foo', 'bar'), ('bar', 5), (9.32,)]
[(9.32,)]
```

len

Checks that a tuple has specified length.

```
>>> instructions.findtuple__len(2).inside(['foo', True, ('foo', 'bar'), ('bar', 5), (9.32,)])
[('foo', 'bar'), ('bar', 5)]
```

lenlt

Checks that a tuple has length less than specified.

```
>>> instructions.findtuple__lenlt(2).inside(['foo', True, ('foo', 'bar'), ('bar', 5), (9.32,)])
[(9.32,)]
```

lenlte

Checks that a tuple has length less than or equal to specified.

```
>>> instructions.findtuple__lenlte(2).inside(['foo', True, ('foo', 'bar'), ('bar', 5), (9.32,)])
[('foo', 'bar'), ('bar', 5), (9.32,)]
```

lengt

Checks that a tuple has length greater than specified.

```
>>> instructions.findtuple__lengt(1).inside(['foo', True, ('foo', 'bar'), ('bar', 5), (9.32,)])
[('foo', 'bar'), ('bar', 5)]
```

lengte

Checks that a tuple has length greater than or equal to specified.

```
>>> instructions.findtuple__lengte(1).inside(['foo', True, ('foo', 'bar'), ('bar', 5), (9.32,)])
[('foo', 'bar'), ('bar', 5), (9.32,)]
```

contains

Checks that a tuple contains the specified value.

```
>>> instructions.findtuple__contains('bar').inside(['foo', True, ('foo', 'bar'), ('bar', 5), (9.32,)
[('foo', 'bar'), ('bar', 5)]
```

contains_all

Checks that a tuple contains all specified values.

```
>>> instructions.findtuple__contains_all(['foo', 'baz']).inside(['foo', True, ('foo', 'baz'), ('bar',
[('foo', 'baz')]
```

contains_any

Checks that a tuple contains any of specified values.

```
>>> instructions.findtuple__contains_any(['foo', 'bar']).inside(['foo', True, ('foo', 'bar'), ('bar',
[('foo', 'bar'), ('bar', 5)]
```

str_contains_str

Checks that a tuple contains at least one string, which contains specified substring.

```
>>> instructions.findtuple__str_contains_str('ba').inside(['foo', True, ('foo', 'bar'), ('bar', 5),
[('foo', 'bar'), ('bar', 5)]
```

5.5.14 set

Set datatype is used to operate on Python's set type. If there is no need to apply any filter, but just to get all the sets from a searchable container, one can use this code:

```
>>> instructions.findset().inside(['foo', True, {'a': 'b'}, set(['bar', 5]), (9.32,)])
[{5, 'bar'}]
```

exact

An exact match.

```
>>> instructions.findset__exact(set((9.32,))).inside(['foo', True, set(('foo', 'bar')), set(('bar',
[{9.32}]
```

len

Checks that a set has specified length.

```
>>> instructions.findset__len(2).inside(['foo', True, set(('foo', 'bar')), set(('bar', 5)), set((9.3
[{'bar', 'foo'}, {5, 'bar'}]
```

lenlt

Checks that a set has length less than specified.

```
>>> instructions.findset__lenlt(2).inside(['foo', True, set(('foo', 'bar')), set(('bar', 5)), set((9
[{9.32}]
```

lenlte

Checks that a set has length less than or equal to specified.

```
>>> instructions.findset__lenlte(2).inside(['foo', True, set(('foo', 'bar')), set(('bar', 5)), set((9
[{'bar', 'foo'}, {5, 'bar'}, {9.32}]
```

lengt

Checks that a set has length greater than specified.

```
>>> instructions.findset__lengt(1).inside(['foo', True, set(('foo', 'bar')), set(('bar', 5)), set((9
[{'bar', 'foo'}, {5, 'bar'}]
```

lengte

Checks that a set has length greater than or equal to specified.

```
>>> instructions.findset__lengte(1).inside(['foo', True, set(('foo', 'bar')), set(('bar', 5)), set((9
[{'bar', 'foo'}, {5, 'bar'}, {9.32}]
```

contains

Checks that a set contains the specified value.

```
>>> instructions.findset__contains('bar').inside(['foo', True, set(('foo', 'bar')), set(('bar', 5)),
[{'bar', 'foo'}, {5, 'bar'}]
```

contains_all

Checks that a set contains all specified values.

```
>>> instructions.findset__contains_all(['foo', 'baz']).inside(['foo', True, set(('foo', 'baz')), set
[{'baz', 'foo'}]
```

contains_any

Checks that a set contains any of specified values.

```
>>> instructions.findset__contains_any(['foo', 'bar']).inside(['foo', True, set(('foo', 'bar')), set
[{'bar', 'foo'}, {5, 'bar'}]
```

str_contains_str

Checks that a set contains at least one string, which contains specified substring.

```
>>> instructions.findset__str_contains_str('ba').inside(['foo', True, set(('foo', 'bar')), set(('bar
[{'bar', 'foo'}, {5, 'bar'}]
```

isdisjoint

Checks that a set has no elements in common with specified set.

```
>>> instructions.findset__isdisjoint(set(['foo'])).inside(['foo', True, set(('foo', 'bar')), set(('ba
[{5, 'bar'}, {9.32}]
```

issubset

Checks that every element of a set is in the specified set.

```
>>> instructions.findset__issubset(set(['foo', 'bar'])).inside(['foo', True, set(('foo', 'bar')), set
[{'bar', 'foo'}]
```

eissubset

Checks that every element of a set is in the specified set and that they are not equal.

```
>>> instructions.findset__eissubset(set(['foo', 'bar', 'baz'])).inside(['foo', True, set(('foo', 'bar
[{'bar', 'foo'}]
```

issuperset

Checks that every element of a specified set is in the set.

```
>>> instructions.findset__issuperset(set(['bar'])).inside(['foo', True, set(('foo', 'bar')), set(('ba
[{'bar', 'foo'}, {5, 'bar'}]
```

eissuperset

Checks that every element of a specified set is in the set and that they are not equal.

```
>>> instructions.findset__eissuperset(set(['bar'])).inside(['foo', True, set(('foo', 'bar')), set(('b
[{'bar', 'foo'}, {5, 'bar'}]
```

5.5.15 frozenset

Frozenset datatype is used to operate on Python's frozenset type. If there is no need to apply any filter, but just to get all the frozensets from a searchable container, one can use this code:

```
>>> instructions.findfrozenset().inside(['foo', True, {'a': 'b'}, frozenset(['bar', 5]), (9.32,)])
[frozenset({5, 'bar'})]
```

exact

An exact match.

```
>>> instructions.findfrozenset__exact(frozenset((9.32,))).inside(['foo', True, frozenset(('foo', 'bar
[frozenset({9.32})]
```

len

Checks that a frozenset has specified length.

```
>>> instructions.findfrozenset__len(2).inside(['foo', True, frozenset(('foo', 'bar')), frozenset(('ba
[frozenset({'bar', 'foo'}), frozenset({5, 'bar'})]
```

lenlt

Checks that a frozenset has length less than specified.

```
>>> instructions.findfrozenset__lenlt(2).inside(['foo', True, frozenset(('foo', 'bar')), frozenset((
[frozenset({9.32})]
```

lenlte

Checks that a frozenset has length less than or equal to specified.

```
>>> instructions.findfrozenset__lenlte(2).inside(['foo', True, frozenset(('foo', 'bar')), frozenset(
[frozenset({'bar', 'foo'}), frozenset({5, 'bar'}), frozenset({9.32})]
```

lengt

Checks that a frozenset has length greater than specified.

```
>>> instructions.findfrozenset__lengt(1).inside(['foo', True, frozenset(('foo', 'bar')), frozenset((
[frozenset({'bar', 'foo'}), frozenset({5, 'bar'})]
```

lengte

Checks that a frozenset has length greater than or equal to specified.

```
>>> instructions.findfrozenset__lengte(1).inside(['foo', True, frozenset(('foo', 'bar')), frozenset(
[frozenset({'bar', 'foo'}), frozenset({5, 'bar'}), frozenset({9.32})]
```

contains

Checks that a frozenset contains the specified value.

```
>>> instructions.findfrozenset__contains('bar').inside(['foo', True, frozenset(('foo', 'bar')), froze
[frozenset({'bar', 'foo'}), frozenset({5, 'bar'})]
```

contains_all

Checks that a frozenset contains all specified values.

```
>>> instructions.findfrozenset__contains_all(['foo', 'baz']).inside(['foo', True, frozenset(('foo',
[frozenset({'baz', 'foo'})]
```

contains_any

Checks that a frozenset contains any of specified values.

```
>>> instructions.findfrozenset__contains_any(['foo', 'bar']).inside(['foo', True, frozenset(('foo',
[frozenset({'bar', 'foo'}), frozenset({5, 'bar'})]
```

str_contains_str

Checks that a frozenset contains at least one string, which contains specified substring.

```
>>> instructions.findfrozenset__str_contains_str('ba').inside(['foo', True, frozenset(('foo', 'bar'))
[frozenset({'bar', 'foo'}), frozenset({5, 'bar'})]
```

isdisjoint

Checks that a frozenset has no elements in common with specified set or frozenset.

```
>>> instructions.findfrozenset__isdisjoint(set(['foo'])).inside(['foo', True, frozenset(('foo', 'bar
[frozenset({5, 'bar'}), frozenset({9.32})]
```

issubset

Checks that every element of a frozenset is in the specified set or frozenset.

```
>>> instructions.findfrozenset__issubset(set(['foo', 'bar'])).inside(['foo', True, frozenset(('foo',
[frozenset({'bar', 'foo'})]
```

eissubset

Checks that every element of a frozenset is in the specified set or frozenset and that they are not equal.

```
>>> instructions.findfrozenset__eissubset(set(['foo', 'bar', 'baz'])).inside(['foo', True, frozenset
[frozenset({'bar', 'foo'})]
```

issuperset

Checks that every element of a specified set or frozenset is in the frozenset.

```
>>> instructions.findfrozenset__issuperset(set(['bar'])).inside(['foo', True, frozenset(('foo', 'bar
[frozenset({'bar', 'foo'}), frozenset({5, 'bar'})]
```

eissuperset

Checks that every element of a specified set or frozenset is in the frozenset and that they are not equal.

```
>>> instructions.findfrozenset__eissuperset(set(['bar'])).inside(['foo', True, frozenset(('foo', 'ba
[frozenset({'bar', 'foo'}), frozenset({5, 'bar'})]
```

5.5.16 dict

Dict datatype is used to operate on Python's dict type. If there is no need to apply any filter, but just to get all the dicts from a searchable container, one can use this code:

```
>>> instructions.finddict().inside(['foo', True, {'a': 'b'}, ['bar', 5], (9.32,)])
[{'a': 'b'}]
```

exact

An exact match.

```
>>> instructions.finddict__exact({'ab': 'ba'}).inside(['foo', True, {'a': 'b', 'b': 'a'}, {'ab': 'ba
[{'ab': 'ba'}]
```

len

Checks that a dict has specified length.

```
>>> instructions.finddict__len(2).inside(['foo', True, {'a': 'b', 'b': 'a'}, {'ab': 'ba'}, {'abc': '
[{'a': 'b', 'b': 'a'}]
```

lenlt

Checks that a dict has length less than specified.

```
>>> instructions.finddict__lenlt(2).inside(['foo', True, {'a': 'b', 'b': 'a'}, {'ab': 'ba'}, {'abc':
[{'ab': 'ba'}, {'abc': 'bca'}]
```

lenlte

Checks that a dict has length less than or equal to specified.

```
>>> instructions.finddict__lenlte(2).inside(['foo', True, {'a': 'b', 'b': 'a'}, {'ab': 'ba'}, {'abc'
[{'a': 'b', 'b': 'a'}, {'ab': 'ba'}, {'abc': 'bca'}]
```

lengt

Checks that a dict has length greater than specified.

```
>>> instructions.finddict__lengt(1).inside(['foo', True, {'a': 'b', 'b': 'a'}, {'ab': 'ba'}, {'abc':
[{'a': 'b', 'b': 'a'}]
```

lengte

Checks that a dict has length greater than or equal to specified.

```
>>> instructions.finddict__lengte(1).inside(['foo', True, {'a': 'b', 'b': 'a'}, {'ab': 'ba'}, {'abc'
[{'a': 'b', 'b': 'a'}, {'ab': 'ba'}, {'abc': 'bca'}]
```

contains_key

Checks that a dict contains key equals to the specified value.

```
>>> instructions.finddict__contains_key('ab').inside(['foo', True, {'a': 'b', 'b': 'a'}, {'ab': 'ba'
[{'ab': 'ba'}]
```

contains_all_keys

Checks that a dict contains all keys equal to the specified values.

```
>>> instructions.finddict__contains_all_keys(['a', 'b']).inside(['foo', True, {'a': 'b', 'b': 'a'},
[{'a': 'b', 'b': 'a'}]
```

contains_any_keys

Checks that a dict contains any keys equal to the specified values.

```
>>> instructions.finddict__contains_any_keys(['a', 'ab']).inside(['foo', True, {'a': 'b', 'b': 'a'},
[{'a': 'b', 'b': 'a'}, {'ab': 'ba'}]
```

key_contains_str

Checks that a dict contains at least one key which is a string, which contains specified substring.

```
>>> instructions.finddict__key_contains_str('ab').inside(['foo', True, {'a': 'b', 'b': 'a'}, {'ab':
[{'ab': 'ba'}, {'abc': 'bca'}]
```

contains_value

Checks that a dict contains value equals to the specified.

```
>>> instructions.finddict__contains_value('ba').inside(['foo', True, {'a': 'b', 'b': 'a'}, {'ab': 'ba
[{'ab': 'ba'}]
```

contains_all_values

Checks that a dict contains all values equal to the specified.

```
>>> instructions.finddict__contains_all_values(['a', 'b']).inside(['foo', True, {'a': 'b', 'b': 'a'},
[{'a': 'b', 'b': 'a'}]
```

contains_any_values

Checks that a dict contains any values equal to the specified.

```
>>> instructions.finddict__contains_any_values(['b', 'ba']).inside(['foo', True, {'a': 'b', 'b': 'a'
[{'a': 'b', 'b': 'a'}, {'ab': 'ba'}]
```

value_contains_str

Checks that a dict contains at least one value which is a string, which contains specified substring.

```
>>> instructions.finddict__value_contains_str('b').inside(['foo', True, {'a': 'b', 'b': 'a'}, {'ab':
[{'a': 'b', 'b': 'a'}, {'ab': 'ba'}, {'abc': 'bca'}]
```

5.6 Exceptions

Instructions tries it's best to provide human readable errors in all situations. This is a list of all exceptions that Instructions can throw.

exception `instructions.exceptions.`**`InstructionsError`**`(`**args*, ***kwargs*`)`
    Base exception class for Instructions exceptions.

exception `instructions.exceptions.`**`FilterImplementationError`**`(`*reason*`)`
    Filter not implemented correctly.

exception `instructions.exceptions.`**`FilterUsageError`**`(`*reason*`)`
    Incorrect usage of Filter.

exception `instructions.exceptions.`**`FilterTypeError`**
    Provided filter should be of Filter type.

exception `instructions.exceptions.`**`DataTypeInitializationError`**`(`*cls*, *reason*`)`
    DataType class can't be initialized.

exception `instructions.exceptions.`**`CommandOptionError`**`(`*option*, *message*`)`
    Command option error.

exception `instructions.exceptions.`**`CommandOptionTypeError`**`(`*option*, *type_*`)`
    Provided option doesn't have the needed type.

5.7 License

Copyright 2015 Max Tepkeev

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

5.8 Changelog

5.8.1 0.1.0 (2015-02-24)

- Initial release

Index