fastnumbers Documentation

Release 3.1.0

Seth M. Morton

Nov 21, 2020

Contents

- Source Code: https://github.com/SethMMorton/fastnumbers
- Downloads: https://pypi.org/project/fastnumbers/
- Documentation: https://fastnumbers.readthedocs.io/

Please see the GitHub main page for everything else, including

- Quick start
- High-level algortihm overview
- Installation instructions
- Testing instructions

CHAPTER 1

Timing

Just how much faster is `fastnumbers` than a pure python implementation? Please see the following Jupyter notebooks for timing information on various Python versions.

- https://nbviewer.jupyter.org/github/SethMMorton/fastnumbers/blob/master/TIMING_35.ipynb
- https://nbviewer.jupyter.org/github/SethMMorton/fastnumbers/blob/master/TIMING_36.ipynb
- https://nbviewer.jupyter.org/github/SethMMorton/fastnumbers/blob/master/TIMING_37.ipynb

CHAPTER 2

fastnumbers API

2.1 The "Built-In Replacement" Functions

Each of these functions acts as a (potentially) faster drop-in replacement for the equivalent Python built-in function. Please perform timing tests on your platform with your data to determine if these functions will actually provide you with a speed-up.

2.1.1 float()

fastnumbers.**float**(*x=0*)

Drop-in but faster replacement for the built-in *float*.

Behaves identically to the built-in *float* except for the following:

- Is implemented as a function, not a class, which means it cannot be sub-classed, and has no *fromhex* classmethod.
- A *ValueError* will be raised instead of a *UnicodeEncodeError* if a partial surrogate is given as input.
- You cannot use this function in `isinstance`.

If you need any of the above functionality you can still access the original *float* class through `builtins`.

```
>>> from fastnumbers import float
>>> isinstance(9.4, float)  # doctest: +ELLIPSIS
Traceback (most recent call last):
    ...
TypeError: ...
>>> import builtins
>>> isinstance(9.4, builtins.float)
True
```

2.1.2 int()

fastnumbers.**int**(*x=0*, *base=10*)

Drop-in but faster replacement for the built-in *int*.

Behaves identically to the built-in *int* except for the following:

- Cannot convert from the `__trunc__` special method of an object.
- Is implemented as a function, not a class, which means it cannot be sub-classed, and has no *from_bytes* classmethod.
- You cannot use this function in `isinstance`.

If you need any of the above functionality you can still access the original *int* class through `builtins`.

```
>>> from fastnumbers import int
>>> isinstance(9, int) # doctest: +ELLIPSIS
Traceback (most recent call last):
    ...
TypeError: ...
>>> import builtins
>>> isinstance(9, builtins.int)
True
```

2.1.3 real()

fastnumbers.**real**(*x=0.0*, *coerce=True*)

Convert to *float* or *int*, whichever is most appropriate.

If an *int* literal or string containing an *int* is provided, then an *int* will be returned. If a *float* literal or a string containing a non-*int* and non-*complex* number is provided, a *float* will be returned.

If *coerce* is *True* (the default), then if a *float* is given that has no decimal places after conversion or only zeros after the decimal point, it will be returned as an *int* instead of a *float*.

2.2 The "Error-Handling" Functions

Each of these functions will quickly convert strings to numbers (and also numbers to numbers) with fast and convenient error handling. They are guaranteed to return results identical to the built-in `float` or `int` functions.

2.2.1 `fast_real()`

`fastnumbers.`**`fast_real`**(*x*, *default=None*, *raise_on_invalid=False*, *on_fail=None*, *nan=None*, *inf=None*, *coerce=True*, *allow_underscores=True*)

Quickly convert input to an *int* or *float* depending on value.

Any input that is valid for the built-in *float* or *int* functions will be converted to either a *float* or *int*. An input of a single numeric unicode character is also valid. The return value is guaranteed to be of type *str*, *int*, or *float*.

If the given input is a string and cannot be converted to a *float* or *int*, it will be returned as-is unless *default* or *raise_on_invalid* is given.

Parameters

- **`input`** (*{str, float, int, long}*) – The input you wish to convert to a real number.
- **`default`** (*optional*) – This value will be returned instead of the input when the input cannot be converted. Has no effect if *raise_on_invalid* is *True*.
- **`raise_on_invalid`** (*bool, optional*) – If *True*, a *ValueError* will be raised if string input cannot be converted to a *float* or *int*. If *False*, the string will be returned as-is. The default is *False*.
- **`on_fail`** (*callable, optional*) – If given and the *input* cannot be converted, the input will be passed to the callable object and its return value will be returned. The function expect only one positional argument. For backwards-compatability, you may call this option *key* instead of *on_fail*, but this is deprecated behavior.
- **`nan`** (*optional*) – If the input value is NAN or can be parsed as NAN, return this value instead of NAN.
- **`inf`** (*optional*) – If the input value is INF or can be parsed as INF, return this value instead of INF.
- **`coerce`** (*bool, optional*) – If the input can be converted to an *int* without loss of precision (even if the input was a *float* or float-containing *str*) coerce to an *int* rather than returning a *float*.
- **`allow_underscores`** (*bool, optional*) – Starting with Python 3.6, underscores are allowed in numeric literals and in strings passed to *int* or *float* (see PEP 515 for details on what is and is not allowed). You can disable that behavior by setting this option to *False* - the default is *True*. Has no effect on Python versions before 3.6.

Returns **out** – If the input could be converted to an *int*, the return type will be *int*. If the input could be converted to a *float* but not an *int*, the return type will be *float*. Otherwise, the input *str* will be returned as-is (if *raise_on_invalid* is *False*) or whatever value is assigned to *default* if *default* is not *None*.

Return type {str, float, int}

Raises

- `TypeError` – If the input is not one of *str*, *float*, or *int*.
- `ValueError` – If *raise_on_invalid* is *True*, this will be raised if the input string cannot be converted to a *float* or *int*.

See also:

isreal(), *real()*

Examples

```
>>> from fastnumbers import fast_real
>>> fast_real('56')
56
>>> fast_real('56.0')
56
>>> fast_real('56.0', coerce=False)
56.0
>>> fast_real('56.07')
56.07
>>> fast_real('56.07 lb')
'56.07 lb'
>>> fast_real(56.07)
56.07
>>> fast_real(56.0)
56
>>> fast_real(56.0, coerce=False)
56.0
>>> fast_real(56)
56
>>> fast_real('invalid', default=50)
50
>>> fast_real('invalid', 50)  # 'default' is first optional positional arg
50
>>> fast_real('nan')
nan
>>> fast_real('nan', nan=0)
0
>>> fast_real('56.07', nan=0)
56.07
>>> fast_real('56.07 lb', raise_on_invalid=True) #doctest: +IGNORE_EXCEPTION_
↪DETAIL
Traceback (most recent call last):
  ...
ValueError: could not convert string to float: '56.07 lb'
>>> fast_real('invalid', on_fail=len)
7
```

Notes

It is roughly equivalent to (but much faster than)

```
>>> def py_fast_real(input, default=None, raise_on_invalid=False,
...                  on_fail=None, nan=None, inf=None):
```

(continues on next page)

(continued from previous page)

```
...     import math
...     try:
...         a = float(input)
...     except ValueError:
...         if raise_on_invalid:
...             raise
...         elif on_fail is not None:
...             return on_fail(input)
...         elif default is not None:
...             return default
...         else:
...             return input
...     else:
...         if nan is not None and math.isnan(a):
...             return nan
...         elif inf is not None and math.isinf(a):
...             return inf
...         else:
...             return int(a) if a.is_integer() else a
...
```

2.2.2 fast_float()

fastnumbers.**fast_float**(*x*, *default=None*, *raise_on_invalid=False*, *on_fail=None*, *nan=None*, *inf=None*, *allow_underscores=True*)

Quickly convert input to a *float*.

Any input that is valid for the built-in *float* function will be converted to a *float*. An input of a single numeric unicode character is also valid. The return value is guaranteed to be of type *str* or *float*.

If the given input is a string and cannot be converted to a *float* it will be returned as-is unless *default* or *raise_on_invalid* is given.

Parameters

- **input** (*{str, float, int, long}*) – The input you wish to convert to a *float*.
- **default** (*optional*) – This value will be returned instead of the input when the input cannot be converted. Has no effect if *raise_on_invalid* is *True*.
- **raise_on_invalid** (*bool, optional*) – If *True*, a *ValueError* will be raised if string input cannot be converted to a *float*. If *False*, the string will be returned as-is. The default is *False*.
- **on_fail** (*callable, optional*) – If given and the *input* cannot be converted, the input will be passed to the callable object and its return value will be returned. The function expect only one positional argument. For backwards-compatability, you may call this option *key* instead of *on_fail*, but this is deprecated behavior.
- **nan** (*optional*) – If the input value is NAN or can be parsed as NAN, return this value instead of NAN.
- **inf** (*optional*) – If the input value is INF or can be parsed as INF, return this value instead of INF.
- **allow_underscores** (*bool, optional*) – Starting with Python 3.6, underscores are allowed in numeric literals and in strings passed to *int* or *float* (see PEP 515 for details

on what is and is not allowed). You can disable that behavior by setting this option to *False* - the default is *True*. Has no effect on Python versions before 3.6.

Returns **out** – If the input could be converted to a *float* the return type will be *float*. Otherwise, the input *str* will be returned as-is (if *raise_on_invalid* is *False*) or whatever value is assigned to *default* if *default* is not *None*.

Return type {str, float}

Raises

- `TypeError` – If the input is not one of *str*, *float*, or *int*.
- `ValueError` – If *raise_on_invalid* is *True*, this will be raised if the input string cannot be converted to a *float*.

See also:

isfloat(), *float()*

Examples

```
>>> from fastnumbers import fast_float
>>> fast_float('56')
56.0
>>> fast_float('56.0')
56.0
>>> fast_float('56.07')
56.07
>>> fast_float('56.07 lb')
'56.07 lb'
>>> fast_float(56.07)
56.07
>>> fast_float(56)
56.0
>>> fast_float('invalid', default=50)
50
>>> fast_float('invalid', 50)  # 'default' is first optional positional arg
50
>>> fast_float('nan')
nan
>>> fast_float('nan', nan=0.0)
0.0
>>> fast_float('56.07', nan=0.0)
56.07
>>> fast_float('56.07 lb', raise_on_invalid=True) #doctest: +IGNORE_EXCEPTION_
↪DETAIL
Traceback (most recent call last):
  ...
ValueError: could not convert string to float: '56.07 lb'
>>> fast_float('invalid', on_fail=len)
7
```

Notes

It is roughly equivalent to (but much faster than)

```
>>> def py_fast_float(input, default=None, raise_on_invalid=False,
...                   on_fail=None, nan=None, inf=None):
...     try:
...         x = float(input)
...     except ValueError:
...         if raise_on_invalid:
...             raise
...         elif on_fail is not None:
...             return on_fail(input)
...         elif default is not None:
...             return default
...         else:
...             return input
...     else:
...         if nan is not None and math.isnan(x):
...             return nan
...         elif inf is not None and math.isinf(x):
...             return inf
...         else:
...             return x
...
```

2.2.3 fast_int()

fastnumbers.**fast_int**(*x*, *default=None*, *raise_on_invalid=False*, *on_fail=None*, *base=10*, *allow_underscores=True*)

Quickly convert input to an *int*.

Any input that is valid for the built-in *int* function will be converted to a *int*. An input of a single digit unicode character is also valid. The return value is guaranteed to be of type *str* or *int*.

If the given input is a string and cannot be converted to an *int* it will be returned as-is unless *default* or *raise_on_invalid* is given.

Parameters

- **input** (*{str, float, int, long}*) – The input you wish to convert to an *int*.
- **default** (*optional*) – This value will be returned instead of the input when the input cannot be converted. Has no effect if *raise_on_invalid* is *True*.
- **raise_on_invalid** (*bool, optional*) – If *True*, a *ValueError* will be raised if string input cannot be converted to an *int*. If *False*, the string will be returned as-is. The default is *False*.
- **on_fail** (*callable, optional*) – If given and the *input* cannot be converted, the input will be passed to the callable object and its return value will be returned. The function expect only one positional argument. For backwards-compatability, you may call this option *key* instead of *on_fail*, but this is deprecated behavior.
- **base** (*int, optional*) – Follows the rules of Python's built-in *int()*; see it's documentation for your Python version. If given, the input **must** be of type *str*.
- **allow_underscores** (*bool, optional*) – Starting with Python 3.6, underscores are allowed in numeric literals and in strings passed to *int* or *float* (see PEP 515 for details on what is and is not allowed). You can disable that behavior by setting this option to *False* - the default is *True*. Has no effect on Python versions before 3.6.

Returns **out** – If the input could be converted to an *int*, the return type will be *int*. Otherwise, the input *str* will be returned as-is (if *raise_on_invalid* is *False*) or whatever value is assigned to *default* if *default* is not *None*.

Return type {str, int}

Raises

- `TypeError` – If the input is not one of *str*, *float*, or *int*.
- `ValueError` – If *raise_on_invalid* is *True*, this will be raised if the input string cannot be converted to an *int*.

See also:

fast_forceint(), *isint()*, *int()*

Examples

```
>>> from fastnumbers import fast_int
>>> fast_int('56')
56
>>> fast_int('56.0')
'56.0'
>>> fast_int('56.07 lb')
'56.07 lb'
>>> fast_int(56.07)
56
>>> fast_int(56)
56
>>> fast_int('invalid', default=50)
50
>>> fast_int('invalid', 50)  # 'default' is first optional positional arg
50
>>> fast_int('56.07 lb', raise_on_invalid=True) #doctest: +IGNORE_EXCEPTION_DETAIL
Traceback (most recent call last):
  ...
ValueError: could not convert string to int: '56.07 lb'
>>> fast_int('invalid', on_fail=len)
7
```

Notes

It is roughly equivalent to (but much faster than)

```
>>> def py_fast_int(input, default=None, raise_on_invalid=False, on_fail=None):
...     try:
...         return int(input)
...     except ValueError:
...         if raise_on_invalid:
...             raise
...         elif on_fail is not None:
...             return on_fail(input)
...         elif default is not None:
...             return default
...         else:
```

(continues on next page)

(continued from previous page)

```
...             return input
...
```

2.2.4 fast_forceint()

fastnumbers.**fast_forceint**(*x*, *default=None*, *raise_on_invalid=False*, *on_fail=None*, *allow_underscores=True*)

Quickly convert input to an *int*, truncating if is a *float*.

Any input that is valid for the built-in *int* function will be converted to a *int*. An input of a single numeric unicode character is also valid. The return value is guaranteed to be of type *str* or *int*.

In addition to the above, any input valid for the built-in *float* will be parsed and the truncated to the nearest integer; for example, '56.07' will be converted to *56*.

If the given input is a string and cannot be converted to an *int* it will be returned as-is unless *default* or *raise_on_invalid* is given.

Parameters

- **input** (*{str, float, int, long}*) – The input you wish to convert to an *int*.
- **default** (*optional*) – This value will be returned instead of the input when the input cannot be converted. Has no effect if *raise_on_invalid* is *True*
- **raise_on_invalid** (*bool, optional*) – If *True*, a *ValueError* will be raised if string input cannot be converted to an *int*. If *False*, the string will be returned as-is. The default is *False*.
- **on_fail** (*callable, optional*) – If given and the *input* cannot be converted, the input will be passed to the callable object and its return value will be returned. The function expect only one positional argument. For backwards-compatability, you may call this option *key* instead of *on_fail*, but this is deprecated behavior.
- **allow_underscores** (*bool, optional*) – Starting with Python 3.6, underscores are allowed in numeric literals and in strings passed to *int* or *float* (see PEP 515 for details on what is and is not allowed). You can disable that behavior by setting this option to *False* - the default is *True*. Has no effect on Python versions before 3.6.

Returns **out** – If the input could be converted to an *int*, the return type will be *int*. Otherwise, the input *str* will be returned as-is (if *raise_on_invalid* is *False*) or whatever value is assigned to *default* if *default* is not *None*.

Return type {str, int}

Raises

- TypeError – If the input is not one of *str*, *float*, or *int*.
- ValueError – If *raise_on_invalid* is *True*, this will be raised if the input string cannot be converted to an *int*.

See also:

fast_int(), *isintlike()*

Examples

```
>>> from fastnumbers import fast_forceint
>>> fast_forceint('56')
56
>>> fast_forceint('56.0')
56
>>> fast_forceint('56.07')
56
>>> fast_forceint('56.07 lb')
'56.07 lb'
>>> fast_forceint(56.07)
56
>>> fast_forceint(56)
56
>>> fast_forceint('invalid', default=50)
50
>>> fast_forceint('invalid', 50)  # 'default' is first optional positional arg
50
>>> fast_forceint('56.07 lb', raise_on_invalid=True) #doctest: +IGNORE_EXCEPTION_
↪DETAIL
Traceback (most recent call last):
  ...
ValueError: could not convert string to float: '56.07 lb'
>>> fast_forceint('invalid', on_fail=len)
7
```

Notes

It is roughly equivalent to (but much faster than)

```
>>> def py_fast_forceint(input, default=None, raise_on_invalid=False, on_
↪fail=None):
...     try:
...         return int(input)
...     except ValueError:
...         try:
...             return int(float(input))
...         except ValueError:
...             if raise_on_invalid:
...                 raise
...             elif on_fail is not None:
...                 return on_fail(input)
...             elif default is not None:
...                 return default
...             else:
...                 return input
...
```

2.3 The "Checking" Functions

These functions return a Boolean value that indicates if the input can return a certain number type or not.

2.3.1 isreal()

fastnumbers.**isreal**(*x*, *str_only=False*, *num_only=False*, *allow_inf=False*, *allow_nan=False*, *allow_underscores=True*)

Quickly determine if a string is a real number.

Returns *True* if the input is valid input for the built-in *float* or *int* functions, or is a single valid numeric unicode character.

The input may be whitespace-padded.

Parameters

- **input** – The input you wish to test if it is a real number.
- **str_only** (*bool, optional*) – If *True*, then any non-*str* input will cause this function to return *False*. The default is *False*.
- **num_only** (*bool, optional*) – If *True*, then any *str* input will cause this function to return *False*. The default is *False*.
- **allow_inf** (*bool, optional*) – If *True*, then the strings 'inf' and 'infinity' will also return *True*. This check is case-insensitive, and the string may be signed (i.e. '+/-'). The default is *False*.
- **allow_nan** (*bool, optional*) – If *True*, then the string 'nan' will also return *True*. This check is case-insensitive, and the string may be signed (i.e. '+/-'). The default is *False*.
- **allow_underscores** (*bool, optional*) – Starting with Python 3.6, underscores are allowed in numeric literals and in strings passed to *int* or *float* (see PEP 515 for details on what is and is not allowed). You can disable that behavior by setting this option to *False* - the default is *True*. Has no effect on Python versions before 3.6.

Returns **result** – Whether or not the input is a real number.

Return type bool

See also:

fast_real()

Examples

```
>>> from fastnumbers import isreal
>>> isreal('56')
True
>>> isreal('56.07')
True
>>> isreal('56.07', num_only=True)
False
>>> isreal('56.07 lb')
False
>>> isreal(56.07)
True
>>> isreal(56.07, str_only=True)
False
>>> isreal(56)
True
>>> isreal('nan')
False
```

(continues on next page)

(continued from previous page)

```
>>> isreal('nan', allow_nan=True)
True
```

Notes

It is roughly equivalent to (but much faster than)

```
>>> import re
>>> def py_isreal(input, str_only=False, num_only=False,
...               allow_nan=False, allow_inf=False):
...     if str_only and type(input) != str:
...         return False
...     if num_only and type(input) not in (float, int):
...         return False
...     try:
...         x = bool(re.match(r'[-+]?\d*\.?\d+(?:[eE][-+]?\d+)?$', input))
...     except TypeError:
...         return type(input) in (float, int)
...     else:
...         if x:
...             return True
...         elif allow_inf and input.lower().strip().lstrip('-+') in ('inf',
↪'infinity'):
...             return True
...         elif allow_nan and input.lower().strip().lstrip('-+') == 'nan':
...             return True
...         else:
...             return False
...
```

2.3.2 isfloat()

fastnumbers.**isfloat**(*x*, *str_only=False*, *num_only=False*, *allow_inf=False*, *allow_nan=False*, *allow_underscores=True*)

Quickly determine if a string is a *float*.

Returns *True* if the input is valid input for the built-in *float* function, is already a valid *float*, or is a single valid numeric unicode character. It differs from *isreal* in that an *int* input will return *False*.

The input may be whitespace-padded.

Parameters

- **input** – The input you wish to test if it is a *float*.
- **str_only** (*bool, optional*) – If *True*, then any non-*str* input will cause this function to return *False*. The default is *False*.
- **num_only** (*bool, optional*) – If *True*, then any *str* input will cause this function to return *False*. The default is *False*.
- **allow_inf** (*bool, optional*) – If *True*, then the strings 'inf' and 'infinity' will also return *True*. This check is case-insensitive, and the string may be signed (i.e. '+/-'). The default is *False*.
- **allow_nan** (*bool, optional*) – If *True*, then the string 'nan' will also return *True*. This check is case-insensitive, and the string may be signed (i.e. '+/-'). The default is *False*.

- **allow_underscores** (*bool, optional*) – Starting with Python 3.6, underscores are allowed in numeric literals and in strings passed to *int* or *float* (see PEP 515 for details on what is and is not allowed). You can disable that behavior by setting this option to *False* - the default is *True*. Has no effect on Python versions before 3.6.

Returns **result** – Whether or not the input is a *float*.

Return type bool

See also:

fast_float(), *isreal()*

Examples

```
>>> from fastnumbers import isfloat
>>> isfloat('56')
True
>>> isfloat('56.07')
True
>>> isfloat('56.07', num_only=True)
False
>>> isfloat('56.07 lb')
False
>>> isfloat(56.07)
True
>>> isfloat(56.07, str_only=True)
False
>>> isfloat(56)
False
>>> isfloat('nan')
False
>>> isfloat('nan', allow_nan=True)
True
```

Notes

It is roughly equivalent to (but much faster than)

```
>>> import re
>>> def py_isfloat(input, str_only=False, num_only=False,
...                allow_nan=False, allow_inf=False):
...     if str_only and type(input) != str:
...         return False
...     if num_only and type(input) != float:
...         return False
...     try:
...         x = bool(re.match(r'[-+]?\d*\.?\d+(?:[eE][-+]?\d+)?$', input))
...     except TypeError:
...         return type(input) == float
...     else:
...         if x:
...             return True
...         elif allow_inf and input.lower().strip().lstrip('-+') in ('inf',
↪'infinity'):
...             return True
```

(continues on next page)

(continued from previous page)

```
...         elif allow_nan and input.lower().strip().lstrip('-+') == 'nan':
...             return True
...         else:
...             return False
```

2.3.3 isint()

fastnumbers.**isint**(*x*, *str_only=False*, *num_only=False*, *allow_underscores=True*)

Quickly determine if a string is an *int*.

Returns *True* if the input is valid input for the built-in *int* function, is already a valid *int*, or is a single valid digit unicode character. It differs from *isintlike* in that a *float* input will return *False* and that *int*-like strings (i.e. '45.0') will return *False*.

The input may be whitespace-padded.

Parameters

- **input** – The input you wish to test if it is an *int*.
- **str_only** (*bool, optional*) – If *True*, then any non-*str* input will cause this function to return *False*. The default is *False*.
- **num_only** (*bool, optional*) – If *True*, then any *str* input will cause this function to return *False*. The default is *False*.
- **allow_underscores** (*bool, optional*) – Starting with Python 3.6, underscores are allowed in numeric literals and in strings passed to *int* or *float* (see PEP 515 for details on what is and is not allowed). You can disable that behavior by setting this option to *False* - the default is *True*. Has no effect on Python versions before 3.6.

Returns **result** – Whether or not the input is an *int*.

Return type bool

See also:

fast_int(), *isintlike()*

Examples

```
>>> from fastnumbers import isint
>>> isint('56')
True
>>> isint('56', num_only=True)
False
>>> isint('56.07')
False
>>> isint('56.07 lb')
False
>>> isint(56.07)
False
>>> isint(56)
True
>>> isint(56, str_only=True)
False
```

Notes

It is roughly equivalent to (but much faster than)

```
>>> import re
>>> def py_isint(input, str_only=False, num_only=False):
...     if str_only and type(input) != str:
...         return False
...     if num_only and type(input) != int:
...         return False
...     try:
...         return bool(re.match(r'[-+]?\d+$', input))
...     except TypeError:
...         return False
...
```

2.3.4 isintlike()

fastnumbers.**isintlike**(*x*, *str_only=False*, *num_only=False*, *allow_underscores=True*)

Quickly determine if a string (or object) is an *int* or *int*-like.

Returns *True* if the input is valid input for the built-in *int* function, is already a valid *int* or *float*, or is a single valid numeric unicode character. It differs from *isintlike* in that *int*-like floats or strings (i.e. '45.0') will return *True*.

The input may be whitespace-padded.

Parameters **input** – The input you wish to test if it is a *int*-like.

Returns

- **result** (*bool*) – Whether or not the input is an *int*.
- **str_only** (*bool, optional*) – If *True*, then any non-*str* input will cause this function to return *False*. The default is *False*.
- **num_only** (*bool, optional*) – If *True*, then any *str* input will cause this function to return *False*. The default is *False*.
- **allow_underscores** (*bool, optional*) – Starting with Python 3.6, underscores are allowed in numeric literals and in strings passed to *int* or *float* (see PEP 515 for details on what is and is not allowed). You can disable that behavior by setting this option to *False* - the default is *True*. Has no effect on Python versions before 3.6.

See also:

fast_forceint()

Examples

```
>>> from fastnumbers import isintlike
>>> isintlike('56')
True
>>> isintlike('56', num_only=True)
False
>>> isintlike('56.07')
False
```

(continues on next page)

(continued from previous page)

```
>>> isintlike('56.0')
True
>>> isintlike('56.07 lb')
False
>>> isintlike(56.07)
False
>>> isintlike(56.0)
True
>>> isintlike(56.0, str_only=True)
False
>>> isintlike(56)
True
```

Notes

It is roughly equivalent to (but much faster than)

```
>>> import re
>>> def py_isintlike(input, str_only=False, num_only=False):
...     if str_only and type(input) != str:
...         return False
...     if num_only and type(input) not in (int, float):
...         return False
...     try:
...         if re.match(r'[-+]?\d+$', input):
...             return True
...         elif re.match(r'[-+]?\d*\.?\d+(?:[eE][-+]?\d+)?$', input):
...             return float(input).is_integer()
...         else:
...             return False
...     except TypeError:
...         if type(input) == float:
...             return input.is_integer()
...         elif type(input) == int:
...             return True
...         else:
...             return False
...
```

2.3.5 query_type()

fastnumbers.**query_type**(*x*, *allow_inf=False*, *allow_nan=False*, *coerce=False*, *allowed_types=**, *allow_underscores=True*)

Quickly determine the type that fastnumbers would return for a given input.

For string or bytes-like input, the contents of the string will be examined and the type *int* or *float* will be returned if the object contains a representation of an *int* or *float*, respectively. For all other cases, the type of the input object is returned, just like the built-in function *type*.

The input may be whitespace-padded.

Parameters

- **input** – The input of which you wish to query the type fastnumbers might return.

- **allow_inf** (*bool, optional*) – If *True*, then the strings 'inf' and 'infinity' will also return *float*. This check is case-insensitive, and the string may be signed (i.e. '+/-'). The default is *False*.
- **allow_nan** (*bool, optional*) – If *True*, then the string 'nan' will also return *float*. This check is case-insensitive, and the string may be signed (i.e. '+/-'). The default is *False*.
- **coerce** (*bool, optional*) – If *True*, then numbers that are given as *float* but could be converted to an *int* without loss of precision will return type *int* instead of *float*.
- **allowed_types** (*sequence of types, optional*) – If given, then only the given types may be returned, and anything else will return *None*.
- **allow_underscores** (*bool, optional*) – Starting with Python 3.6, underscores are allowed in numeric literals and in strings passed to *int* or *float* (see PEP 515 for details on what is and is not allowed). You can disable that behavior by setting this option to *False* - the default is *True*. Has no effect on Python versions before 3.6.

Returns **result** – The type that fastnumbers might return for the given input.

Return type type

See also:

isreal(), *isfloat()*, *isint()*, *isintlike()*

Examples

```
>>> from fastnumbers import query_type
>>> query_type('56')
<class 'int'>
>>> query_type('56.07')
<class 'float'>
>>> query_type('56.07 lb')
<class 'str'>
>>> query_type('56.07 lb', allowed_types=(float, int))  # returns None
>>> query_type('56.0')
<class 'float'>
>>> query_type('56.0', coerce=True)
<class 'int'>
>>> query_type(56.07)
<class 'float'>
>>> query_type(56)
<class 'int'>
>>> query_type('nan')
<class 'str'>
>>> query_type('nan', allow_nan=True)
<class 'float'>
```

CHAPTER 3

Changelog

3.1 Unreleased

3.2 3.1.0 - 2020-11-21

3.2.1 Added

- `query_type` function to determine what as type `fastnumbers` will interpret a given input

3.2.2 Fixed

- Support for Python 3.9 (eliminate use of private Python C function that is now hidden in 3.9) (issue #43)

3.3 3.0.0 - 2020-01-06

3.3.1 Added

- Support and tests for Python 3.8
- Text to highlight that `fastnumbers` is not always faster that native Python
- `on_fail` option that is identical to `key`, but has a more descriptive name
- Windows testing to Travis-CI
- Code quality checks to Travis-CI
- Deployment from Travis-CI
- RELEASING.md

3.3.2 Changed

- Cleaned up all test code so that it no longer includes unused code and also conformes to flake8/black
- Near-complete re-write of the README, hopefully to make the functionality of `fastnumbers` clearer, to support better navigation, and to better highlight caveats.
- Made all named options keyword-only except for `default`

3.3.3 Deprecated

- `key` function (it will forever remain allowed, but is "hidden" and cannot be given with `on_fail`)

3.3.4 Fixed

- Bug where the `coerce` option of `real()` was ignored
- Improved testing reproducibility by pinning all test dependencies

3.3.5 Removed

- Support for Python 2.7 and Python 3.4
- Appveyor service

3.4 2.2.1 - 2019-03-25

3.4.1 Fixed

- Formatting docstring
- Package metadata

3.5 2.2.0 - 2019-03-24

3.5.1 Changed

- Add `allow_underscores` option to toggle whether or not underscores are allowed inside numbers on Python >= 3.6
- Update CHANGELOG format to style from https://keepachangelog.com/
- Build system now uses pip in stead of pipenv (issue #22)
- Simplify tox.ini file

3.6 2.1.1 - 2018-08-19

3.6.1 Added

- A high-level description of the fastnumbers algorithm in the documentation.

3.6.2 Fixed

- Compile error on FreeBSD where fastnumbers' `"string.h"` shadowed the system `"string.h"`.

3.7 2.1.0 - 2018-08-03

3.7.1 Changed

- Speedup of conversions of ASCII-like unicode in Python 3.
- Speedup of conversions for large integers and large floats - fastnumbers should now be at least as fast as built-in functions.
- Restructure code base, hopefully improving simplicity.

3.7.2 Fixed

- Bug in converting unicode numbers on Python 3.7.

3.8 2.0.5 - 2018-07-01

3.8.1 Changed

- Source files are sorted before compilation.

3.8.2 Fixed

- Bug in `fast_real` that resulted in an `OverflowError` when given very large int-like floats.

3.9 2.0.4 - 2018-05-18

3.9.1 Fixed

- Install error on old versions of setuptools.

3.10 2.0.3 - 2018-05-14

3.10.1 Added

- ‘‘*bumpversion*‘ <https://github.com/c4urself/bump2version>‘_ infrastrucutre.

3.10.2 Changed

- Reorganized testing and development infrastructure.
- Development dependencies are now defined using Pipfile.

3.11 2.0.2 - 2017-11-11

3.11.1 Added

- Added testing for Python 3.7.

3.11.2 Changed

- Updated docstrings.
- Improved timing documentation.

3.12 2.0.1 - 2017-04-30

3.12.1 Fixed

- Bug in decimal digit limit on GCC.

3.13 2.0.0 - 2017-04-30

3.13.1 Added

- Support for Python 3.6 underscores.
- Drop-in replacements for the built-in `int()` and `float()` functions.
- Appveyor testing to ensure no surprises on Windows.

3.13.2 Changes

- Incorporated unit tests from Python’s testing library to ensure that any input that Python can handle will also be handled the same way by `fastnumbers`.
- Revamped documentation.
- Refactored internal mechanism for assessing overflow to be faster in the most common cases.

3.13.3 Removed

- Dropped support for Python 2.6.

3.14 1.0.0 - 2016-04-23

3.14.1 Changed

- "coerce" in `fast_real` now applies to any input, not just numeric; the default is now *True* instead of *False*.
- Now all ASCII whitespace characters are stripped by fastnumbers
- Typechecking is now more forgiving
- fastnumbers now checks for errors when converting between numeric types
- Testing now includes Python 2.6.

3.14.2 Fixed

- Bug where very small numbers are not converted properly
- Unicode handling on Windows.
- Python2.6 on Windows.

3.14.3 Removed

- Removed `safe_*` functions (which were deprecated since version 0.3.0)

3.15 0.7.4 - 2016-03-19

3.15.1 Added

- The `coerce` option to `fast_real`.

3.16 0.7.3 - 2016-03-08

3.16.1 Changed

- Newline is now considered to be whitespace (for consistency with the builtin `float` and `int`).

3.17 0.7.2 - 2016-03-07

3.17.1 Fixed

- Overflow bug in exponential parts of floats.

3.18 0.7.1 - 2016-02-29

3.18.1 Added

- `key` function option to transform invalid input arguments.

3.18.2 Fixed

- Compilation bug with MSVC.

3.19 0.7.0 - 2016-01-18

3.19.1 Changed

- Sped up functions by eliminating an unnecessary string copy.
- Broke all functions into smaller components, eliminating a lot of duplication.
- Improved documentation.

3.20 0.6.2 - 2015-11-01

3.20.1 Fixed

- Bug that caused a SystemError exception to be raised on Python 3.5 if a very large int was passed to the "fast" functions.

3.21 0.6.1 - 2015-10-29

3.21.1 Added

- `tox.ini`

3.21.2 Changed

- Sped up unit testing.

3.21.3 Fixed

- Segfault on Python 3.5 that seemed to be related to a change in the PyObject_CallMethod C function.

3.22 0.6.0 - 2015-10-27

3.22.1 Added

- The `nan` and `inf` options to `fast_real` and `fast_float`. These options allow alternate return values in the case of *nan* or *inf*, respectively.

3.22.2 Changed

- Improved documentation.
- Improved testing.

3.22.3 Fixed

- Fixed issue where giving a default of *None* would be ignored.

3.23 0.5.2 - 2015-06-11

3.23.1 Fixed

- Compile error with Visual Studio compilers.

3.24 0.5.1 - 2015-06-04

3.24.1 Changed

- Made handling of Infinity and NaN for `fast_int` and `fast_forceint` consistent with the `built-in` int function.

3.24.2 Fixed

- Solved rare segfault when parsing Unicode input.

3.25 0.5.0 - 2015-05-12

3.25.1 Added

- Added `num_only` option for checker functions.

3.25.2 Changed

- Made `default` the first optional argument instead of `raise_on_invalid\` for conversion functions.

3.26 0.4.0 - 2015-05-03

3.26.1 Added

- Support for conversion of single Unicode characters that represent numbers and digits.

3.27 0.3.0 - 2015-04-23

3.27.1 Changed

- Updated all unit testing to use the `hypothesis` module, which results in better test coverage.
- Updated the `fast_*` functions to check if an overflow loss of precision has occurred, and if so fall back on the more accurate number conversion method.

3.27.2 Deprecated

- In response to the above change, the `safe_*` functions are now deprecated, and internally now use the same code as the `fast_*` functions.

3.28 0.2.0 - 2014-09-03

3.28.1 Added

- A `default` option to the conversion functions.

3.29 0.1.4 - 2014-08-12

3.29.1 Changed

- The method to catch corner-cases like '.', '+', 'e', etc. has been reworked to be more general... case-by-case patches should no longer be needed.

3.29.2 Fixed

- Bug where '.' was incorrectly identified as a valid float/int and converted to 0. This bug only applied to the `fast_*` and `is*` functions.

3.30 0.1.3 - 2014-08-12

3.30.1 Fixed

- Bug where 'e' and 'E' were incorrectly identified as a valid float/int and converted to 0. This bug only applied to the `fast_*` and `is*` functions.

3.31 0.1.2 - 2014-08-12

3.31.1 Fixed

- Bug where '+' and '-' were incorrectly identified as a valid float/int and converted to 0. This bug only applied to the `fast_*` and `is*` functions.
- Bug where `safe_forceint` did not handle `nan` correctly.

3.32 0.1.1 - 2014-08-11

3.32.1 Added

- Support for `inf` and `nan`

3.33 0.1.0 - 2014-08-10

- Initial release of `fastnumbers`

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

F

I

Q

R