## Contents

1 The fastnumbers module
   1.1 Quick Description ........................................... 3
   1.2 Installation .................................................... 4
   1.3 How to Run Tests ............................................... 5

2 Timing ................................................................. 7

3 fastnumbers API ......................................................... 9
   3.1 The “Built-In Replacement” Functions ............................ 9
   3.2 The “Error-Handling” Functions ................................ 11
   3.3 The “Checking” Functions ........................................ 18

4 Changelog ............................................................... 25
   4.1 07-01-2018 v. 2.0.5 ............................................ 25
   4.2 05-18-2018 v. 2.0.4 ............................................ 25
   4.3 05-14-2018 v. 2.0.3 ............................................ 25
   4.4 11-11-2017 v. 2.0.2 ............................................ 25
   4.5 04-30-2017 v. 2.0.1 ............................................ 25
   4.6 04-30-2017 v. 2.0.0 ............................................ 25
   4.7 04-23-2016 v. 1.0.0 ............................................ 25
   4.8 03-19-2016 v. 0.7.4 ............................................ 25
   4.9 03-08-2016 v. 0.7.3 ............................................ 25
   4.10 03-07-2016 v. 0.7.2 .......................................... 25
   4.11 02-29-2016 v. 0.7.1 .......................................... 25
   4.12 01-18-2016 v. 0.7.0 .......................................... 25
   4.13 11-01-2015 v. 0.6.2 .......................................... 25
   4.14 10-29-2015 v. 0.6.1 .......................................... 25
   4.15 10-27-2015 v. 0.6.0 .......................................... 25
   4.16 06-11-2015 v. 0.5.2 .......................................... 25
   4.17 06-04-2015 v. 0.5.1 .......................................... 25
   4.18 05-12-2015 v. 0.5.0 .......................................... 25
   4.19 05-03-2015 v. 0.4.0 .......................................... 25
   4.20 04-23-2015 v. 0.3.0 .......................................... 25
   4.21 09-03-2014 v. 0.2.0 .......................................... 25
   4.22 08-12-2014 v. 0.1.4 .......................................... 25
   4.23 08-12-2014 v. 0.1.3 .......................................... 25
   4.24 08-12-2014 v. 0.1.2 .......................................... 25
5 Indices and tables

Python Module Index
Contents:
The fastnumbers module

Super-fast and clean conversions to numbers.

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

Please see the Timing Documentation for timing details. Check out the API.

1.1 Quick Description

The below examples showcase the fast_float() function, which is a fast conversion functions with error-handling. Please see the API Documentation for other functions that are available from fastnumbers.

```python
>>> from fastnumbers import fast_float, float as fnfloat

>>> # Convert string to a float
>>> fast_float('56.07')
56.07

>>> # Unconvertable string returned as-is by default
>>> fast_float('bad input')
'bad input'

>>> # Unconvertable strings can trigger a default value
>>> fast_float('bad input', default=0)
0

>>> # 'default' is also the first optional positional arg
>>> fast_float('bad input', 0)
0

>>> # Integers are converted to floats
>>> fast_float(54)
54.0

>>> # One can ask inf or nan to be substituted with another value
>>> fast_float('nan')
```

(continues on next page)
nan
>>> fast_float('nan', nan=0.0)
0.0
>>> fast_float(float('nan'), nan=0.0)
0.0
>>> fast_float('56.07', nan=0.0)
56.07
>>> # The default built-in float behavior can be triggered with
>>> # "raise_on_invalid" set to True.
>>> fast_float('bad input', raise_on_invalid=True)
Traceback (most recent call last):
  ...  
ValueError: invalid literal for float(): bad input
>>> # A key function can be used to return an alternate value for invalid input
>>> fast_float('bad input', key=len)
9
>>> fast_float(54, key=len)
54.0
>>> # Single unicode characters can be converted.
>>> fast_float(u'Ⅴ')  # Roman numeral 5 (V)
5.0
>>> fast_float(u'⑦')  # 7 enclosed in a circle
7.0

NOTE: If you need locale-dependent conversions, supply the fastnumbers function of your choice to locale.atof().

```python
import locale
locale.setlocale(locale.LC_ALL, 'de_DE.UTF-8')
print(atof('468,5', func=fast_float))  # Prints 468.5
```

## 1.2 Installation

Installation of fastnumbers is ultra-easy. Simply execute from the command line:

```
$ pip install fastnumbers
```

You can also download the source from [https://pypi.org/project/fastnumbers/](https://pypi.org/project/fastnumbers/), or browse the git repository at [https://github.com/SethMMorton/fastnumbers](https://github.com/SethMMorton/fastnumbers).

If you choose to install from source (will need a C compiler and the Python headers), you can unzip the source archive and enter the directory, and type:

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

If you want to build this documentation, enter:

```
$ python setup.py build_sphinx
```

fastnumbers requires python version 2.7 or greater (this includes python 3.x). Unit tests are only run on 2.7 and $\geq 3.4$. 

Chapter 1. The fastnumbers module
1.3 How to Run Tests

Please note that fastnumbers is NOT set-up to support python setup.py test.

The recommended way to run tests with with tox. Suppose you want to run tests for Python 3.6 - you can run tests by simply executing the following:

```bash
$ tox -e py36
```

Tox will create virtual a virtual environment for your tests and install all the needed testing requirements for you.

If you want to run testing on all of Python 2.7, 3.4, 3.5, 3.6, and 3.7 you can simply execute

```bash
$ tox
```

If you do not wish to use tox, you can install the testing dependencies and run the tests manually using pytest - fastnumbers contains a Pipfile for use with pipenv that makes it easy for you to install the testing dependencies:

```bash
$ pipenv install --skip-lock --dev
$ pipenv install --skip-lock -e .
$ pipenv run pytest
```

fastnumbers uses pytest to run its tests.
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_27.ipynb
- https://nbviewer.jupyter.org/github/SethMMorton/fastnumbers/blob/master/TIMING_35.ipynb
CHAPTER 3

fastnumbers API

• The “Built-In Replacement” Functions
  – float()
  – int()
  – real()

• The “Error-Handling” Functions
  – fast_real()
  – fast_float()
  – fast_int()
  – fast_forceint()

• The “Checking” Functions
  – isreal()
  – isfloat()
  – isnat()
  – isnatlike()

3.1 The “Built-In Replacement” Functions

Each of these functions acts as a faster drop-in replacement for the equivalent Python built-in function.
3.1.1 float()

\[ \text{float}(x=0) \]
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 \texttt{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 \texttt{isinstance}.

If you need any of the above functionality you can still access the original float class through \texttt{builtins}.

```python
>>> from fastnumbers import float
>>> isinstance(9.4, float)
Traceback (most recent call last):
  ...
TypeError: ...
>>> try:
...    import builtins
... except ImportError:
...    import __builtin__ as builtins # Python 2
...>>> isinstance(9.4, builtins.float)
True
```

3.1.2 int()

\[ \text{int}(x=0, \text{base}=10) \]
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 \texttt{__trunc__} special method of an object.
- Is implemented as a function, not a class, which means it cannot be sub-classed, and has no \texttt{from_bytes} classmethod.
- You cannot use this function in \texttt{isinstance}.

If you need any of the above functionality you can still access the original int class through \texttt{builtins}.

```python
>>> from fastnumbers import int
>>> isinstance(9, int)
Traceback (most recent call last):
  ...
TypeError: ...
>>> try:
...    import builtins
... except ImportError:
...    import __builtin__ as builtins # Python 2
...>>> isinstance(9, builtins.int)
True
```
3.1.3 real()

```python
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`.

3.2 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.

3.2.1 fast_real()

```python
fastnumbers.fast_real(x, default=None, raise_on_invalid=False, key=None, nan=None, inf=None, coerce=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` (or `long` on Python2).

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`.
- `key` (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 must take one and only one required argument.
- `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`.

**Returns**

- `out` – If the input could be converted to an `int`, the return type will be `int` (or `long` on Python2 if the integer is large enough). 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, long}`

**Raises**

- **TypeError** – If the input is not one of `str`, `float`, or `int` (or `long` on Python2).
- **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**

```python
>>> 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)
Traceback (most recent call last):
 ... 
ValueError: could not convert string to float: '56.07 lb'
>>> fast_real('invalid', key=len)
7
```

**Notes**

It is roughly equivalent to (but much faster than)
3.2.2 fast_float()

`fastnumbers.fast_float(x, default=None, raise_on_invalid=False, key=None, nan=None, inf=None)`

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.
- `key` *(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 must take one and only one required argument.
- `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.

**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 (or long on Python2).
- **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

```python
>>> 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)
Traceback (most recent call last):
...  ValueError: could not convert string to float: '56.07 lb'
>>> fast_float('invalid', key=len)
7
```

Notes

It is roughly equivalent to (but much faster than)

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

3.2.3 fast_int()

fastnumbers.fast_int(x, default=None, raise_on_invalid=False, key=None, base=10)

Quickly convert input to an `int`.

Any input that is valid for the built-in `int` (or `long` on Python2) function will be converted to a `int` (or `long` on Python2). An input of a single digit unicode character is also valid. The return value is guaranteed to be of type `str` or `int` (or `long` on Python2).

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`.
- `key (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 must take one and only one required argument.
- `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`.

Returns `out` – If the input could be converted to an `int`, the return type will be `int` (or `long` on Python2 if the integer is large enough). 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, long}`

Raises

- `TypeError` – If the input is not one of `str, float, or int` (or `long` on Python2).
- `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

```python
>>> 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)
Traceback (most recent call last):
  ...
ValueError: could not convert string to int: '56.07 lb'
>>> fast_int('invalid', key=len)
7
```

Notes

It is roughly equivalent to (but much faster than)

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

3.2.4 fast_forceint()

fastnumbers.fast_forceint(x, default=None, raise_on_invalid=False, key=None)

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

Any input that is valid for the built-in int (or long on Python2) function will be converted to a int (or long on Python2). An input of a single numeric unicode character is also valid. The return value is guaranteed to be of type str or int (or long on Python2).
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.
- **key** (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 must take one and only one required argument.

**Returns** out – If the input could be converted to an int, the return type will be int (or long on Python2 if the integer is large enough). 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, long}

**Raises**

- **TypeError** – If the input is not one of str, float, or int (or long on Python2).
- **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**

```python
>>> 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)
Traceback (most recent call last):
...
```

(continues on next page)
ValueError: could not convert string to float: '56.07 lb'
>>> fast_forceint('invalid', key=len)
7

Notes

It is roughly equivalent to (but much faster than)

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

### 3.3 The “Checking” Functions

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

#### 3.3.1 isreal()

`fastnumbers.isreal(x, str_only=False, num_only=False, allow_inf=False, allow_nan=False)`

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.

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

**Return type** bool

See also:

`fast_real()`

**Examples**

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

**Notes**

It is roughly equivalent to (but much faster than)

```python
>>> 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
```

(continues on next page)
3.3.2 isfloat()

`fastnumbers.isfloat(x, str_only=False, num_only=False, allow_inf=False, allow_nan=False)`

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.

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

Return type bool

See also:

`fast_float()`, `isreal()`

Examples

```python
>>> from fastnumbers import isfloat
>>> isfloat('56')
True
>>> isfloat('56.07')
True
>>> isreal('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')
```

(continues on next page)
False
>>> isfloat('nan', allow_nan=True)
True

Notes

It is roughly equivalent to (but much faster than)

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

3.3.3 isint()

`fastnumbers.isint(x, str_only=False, num_only=False)`

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`.

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

**Return type** `bool`

See also:
fast_int(), isnintlike()

Examples

```python
>>> 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)

```python
>>> 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
...
```

3.3.4 isnintlike()

fastnumbers.isintlike(x, str_only=False, num_only=False)

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 isnint 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.

See also:

`fast_forceint()`

Examples

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

```python
>>> 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
```
4.1 07-01-2018 v. 2.0.5

• Fix bug in `fast_real` that resulted in an `OverflowError` when given very large int-like floats.
• Source files are sorted before compilation.

4.2 05-18-2018 v. 2.0.4

• Fix install error on old versions of setuptools.

4.3 05-14-2018 v. 2.0.3

• Reorganized testing and development infrastructure.
• Add bumpversion infrastructure.
• Development dependencies are now defined using Pipfile.

4.4 11-11-2017 v. 2.0.2

• Updated docstrings.
• Improved timing documentation.
• Added testing for Python 3.7.
4.5 04-30-2017 v. 2.0.1

- Fixed bug in decimal digit limit on GCC.

4.6 04-30-2017 v. 2.0.0

- Dropped support for Python 2.6.
- Added support for Python 3.6 underscores.
- Added drop-in replacements for the built-in `int()` and `float()` functions.
- 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`.
- Added Appveyor testing to ensure no surprises on Windows.
- Revamped documentation.
- Refactored internal mechanism for assessing overflow to be faster in the most common cases.

4.7 04-23-2016 v. 1.0.0

- “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
- Fixed bug where very small numbers are not converted properly
- Testing now includes Python 2.6.
- Removed `safe_*` functions (which were deprecated since version 0.3.0)
- Fixed unicode handling on Windows.
- Fixed Python2.6 on Windows.

4.8 03-19-2016 v. 0.7.4

- Added the “coerce” option to `fast_real`.

4.9 03-08-2016 v. 0.7.3

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

4.10 03-07-2016 v. 0.7.2

- Fixed overflow bug in exponential parts of floats.
4.11  02-29-2016 v. 0.7.1

• Fixed compilation bug with MSVC.
• Added “key” function to transform invalid input arguments.

4.12  01-18-2016 v. 0.7.0

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

4.13  11-01-2015 v. 0.6.2

• 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.

4.14  10-29-2015 v. 0.6.1

• Fixed segfault on Python 3.5 that seemed to be related to a change in the PyObject_CallMethod C function.
• Sped up unit testing.
• Added tox.ini.

4.15  10-27-2015 v. 0.6.0

• Fixed issue where giving a default of None would be ignored.
• 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.
• Improved documentation.
• Improved testing.

4.16  06-11-2015 v. 0.5.2

• Fixed compile error Visual Studio compilers.

4.17  06-04-2015 v. 0.5.1

• Solved rare segfault when parsing Unicode input.
• Made handling of Infinity and NaN for fast_int and fast_forceint consistent with the built-in int function.
4.18 05-12-2015 v. 0.5.0

• Made ‘default’ first optional argument instead of ‘raise_on_invalid’ for conversion functions.
• Added ‘num_only’ option for checker functions.

4.19 05-03-2015 v. 0.4.0

• Added support for conversion of single Unicode characters that represent numbers and digits.

4.20 04-23-2015 v. 0.3.0

• 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.
• In response to the above change, the safe_* functions are now deprecated, and internally now use the same code as the fast_* functions.
• Updated all unit testing to use the hypothesis module, which results in better test coverage.

4.21 09-03-2014 v. 0.2.0

• Added a ‘default’ option to the conversion functions.

4.22 08-12-2014 v. 0.1.4

• 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.
• 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.

4.23 08-12-2014 v. 0.1.3

• 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.

4.24 08-12-2014 v. 0.1.2

• 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.
• Fixed bug where ‘safe_forceint’ did not handle ‘nan’ correctly.
4.25 08-11-2014 v. 0.1.1

- ‘fastnumbers’ now understands ‘inf’ and ‘nan’.

4.26 08-10-2014 v. 0.1.0

- Initial release of ‘fastnumbers’.
CHAPTER 5

Indices and tables

- genindex
- modindex
- search
f

fastnumbers, 1
<table>
<thead>
<tr>
<th>Module</th>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fastnumbers</td>
<td>fast_float()</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>fast_forcint()</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>fast_int()</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>fast_real()</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>fastnumbers (module)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>float()</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>int()</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>isfloat()</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>isint()</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>isintlike()</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>isreal()</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>real()</td>
<td>11</td>
</tr>
</tbody>
</table>