
PyJSON5 Documentation

Release 0.3.6

René Kijewski

Jun 02, 2018

Contents

1	Installation	3
2	Table of Contents	5
2.1	Serializer / Encoder	5
2.1.1	Quick Encoder Summary	5
2.1.2	Full Encoder Description	6
2.1.3	Encoder Compatibility Functions	9
2.1.4	Encoder Exceptions	10
2.2	Parser / Decoder	10
2.2.1	Quick Decoder Summary	11
2.2.2	Full Decoder Description	11
2.2.3	Decoder Compatibility Functions	13
2.2.4	Decoder Exceptions	14
2.3	Exceptions	16
2.4	Performance	16
2.4.1	Decoder Performance	16
2.4.2	Encoder Performance	17
3	Quick Summary	19
4	Compatibility	21

A JSON5 serializer and parser library for Python 3.4 and later.

The serializer returns ASCII data that can safely be used in an HTML template. Apostrophes, ampersands, greater-than, and less-than signs are encoded as unicode escaped sequences. E.g. this snippet is safe for any and all input:

```
"<a onclick='alert(\" + encode(data) + \")'>show message</a>"
```

Unless the input contains infinite or NaN values, the result will be valid **JSON** data.

All valid **JSON5 1.0.0** and **JSON** data can be read, unless the nesting level is absurdly high.

CHAPTER 1

Installation

```
$ pip install pyjson5
```


2.1 Serializer / Encoder

The serializer returns ASCII data that can safely be used in an HTML template. Apostrophes, ampersands, greater-than, and less-than signs are encoded as unicode escaped sequences. E.g. this snippet is safe for any and all input:

```
"<a onclick='alert(" + encode(data) + ")'>show message</a>"
```

Unless the input contains infinite or NaN values, the result will be valid **JSON** data.

2.1.1 Quick Encoder Summary

<i>encode</i> (data, *[, options])	Serializes a Python object to a JSON5 compatible unicode string.
<i>encode_bytes</i> (data, *[, options])	Serializes a Python object to a JSON5 compatible bytes string.
<i>encode_callback</i> (data, cb[, supply_bytes, ...])	Serializes a Python object into a callback function.
<i>encode_io</i> (data, fp[, supply_bytes, options])	Serializes a Python object into a file-object.
<i>encode_noop</i> (data, *[, options])	Test if the input is serializable.
<i>dump</i> (obj, fp, **kw)	Serializes a Python object to a JSON5 compatible unicode string.
<i>dumps</i> (obj, **kw)	Serializes a Python object to a JSON5 compatible unicode string.
<i>Options</i>	Customizations for the <i>encoder_*</i> (...) function family.
<i>Json5EncoderException</i>	Base class of any exception thrown by the serializer.
<i>Json5UnstringifiableType</i> ([message, ...])	The encoder was not able to stringify the input, or it was told not to by the supplied <i>Options</i> .

2.1.2 Full Encoder Description

`pyjson5.encode(data, *, options=None, **options_kw)`
Serializes a Python object to a JSON5 compatible unicode string.

```
encode(['Hello', 'world!']) == '["Hello","world!"]'
```

Parameters

- **data** (*object*) – Python object to serialize.
- **options** (*Optional[Options]*) – Extra options for the encoder. If **options** and **options_kw** are specified, then `options.update(**options_kw)` is used.
- **options_kw** – See Option’s arguments.

Raises

- *Json5EncoderException* – An exception occurred while encoding.
- *TypeError* – An argument had a wrong type.

Returns

Unless `float('inf')` or `float('nan')` is encountered, the result will be valid JSON data (as of RFC8259).

The result is always ASCII. All characters outside of the ASCII range are encoded.

The result safe to use in an HTML template, e.g. `<a onclick='alert({{ encode(url) }})'>show message</a>`. Apostrophes `"'"` are encoded as `"\u0027"`, less-than, greater-than, and ampersand likewise.

Return type `str`

`pyjson5.encode_bytes(data, *, options=None, **options_kw)`
Serializes a Python object to a JSON5 compatible bytes string.

```
encode_bytes(['Hello', 'world!']) == b'["Hello","world!"]'
```

Parameters

- **data** (*object*) – see *encode(...)*
- **options** (*Optional[Options]*) – see *encode(...)*
- **options_kw** – see *encode(...)*

Raises

- *Json5EncoderException* – An exception occurred while encoding.
- *TypeError* – An argument had a wrong type.

Returns see *encode(...)*

Return type `bytes`

`pyjson5.encode_callback(data, cb, supply_bytes=False, *, options=None, **options_kw)`
Serializes a Python object into a callback function.

The callback function `cb` gets called with single characters and strings until the input `data` is fully serialized.

```

encode_callback(['Hello', 'world!'], print)
#prints:
# [
#  "
#  Hello
#  "
#  ,
#  "
#  world!
#  "
#  ]

```

Parameters

- **data** (*object*) – see *encode(...)*
- **cb** (*Callable[[Union[bytes|str]], None]*) – A callback function. Depending on the truthiness of *supply_bytes* either *bytes* or *str* is supplied.
- **supply_bytes** (*bool*) – Call *cb(...)* with a *bytes* argument if true, otherwise *str*.
- **options** (*Optional[Options]*) – see *encode(...)*
- **options_kw** – see *encode(...)*

Raises

- *Json5EncoderException* – An exception occurred while encoding.
- *TypeError* – An argument had a wrong type.

Returns The supplied argument *cb*.

Return type *Callable[[Union[bytes|str]], None]*

`pyjson5.encode_io(data, fp, supply_bytes=True, *, options=None, **options_kw)`

Serializes a Python object into a file-object.

The return value of `fp.write(...)` is not checked. If *fp* is unbuffered, then the result will be garbage!

Parameters

- **data** (*object*) – see *encode(...)*
- **fp** (*IOBase*) – A file-like object to serialize into.
- **supply_bytes** (*bool*) – Call *fp.write(...)* with a *bytes* argument if true, otherwise *str*.
- **options** (*Optional[Options]*) – see *encode(...)*
- **options_kw** – see *encode(...)*

Raises

- *Json5EncoderException* – An exception occurred while encoding.
- *TypeError* – An argument had a wrong type.

Returns The supplied argument *fp*.

Return type *IOBase*

`pyjson5.encode_noop(data, *, options=None, **options_kw)`

Test if the input is serializable.

Most likely you want to serialize data directly, and catch exceptions instead of using this function!

```
encode_noop({47: 11}) == True
encode_noop({47: object()}) == False
```

Parameters

- **data** (*object*) – see *encode(...)*
- **options** (*Optional[Options]*) – see *encode(...)*
- **options_kw** – see *encode(...)*

Returns True iff data is serializable.

Return type bool

class `pyjson5.Options`

Customizations for the `encoder_*`(...) function family.

Immutable. Use `Options.update(**kw)` to create a new Options instance.

Parameters

- **tojson** (*str/False/None*) –
 - **str**: A special method to call on objects to return a custom JSON encoded string. Must return ASCII data!
 - **False**: No such member exists. (Default.)
 - **None**: Use default.
- **posinfinity** (*str/False/None*) –
 - **str**: String to represent positive infinity. Must be ASCII.
 - **False**: Throw an exception if `float('+inf')` is encountered.
 - **None**: Use default: "Infinity".
- **neginfinity** (*str/False/None*) –
 - **str**: String to represent negative infinity. Must be ASCII.
 - **False**: Throw an exception if `float('-inf')` is encountered.
 - **None**: Use default: "-Infinity".
- **nan** (*str/False/None*) –
 - **str**: String to represent not-a-number. Must be ASCII.
 - **False**: Throw an exception if `float('NaN')` is encountered.
 - **None**: Use default: "NaN".
- **intformat** (*str/False/None*) –
 - **str**: Format string to use with `int`.
 - **False**: Throw an exception if an `int` is encountered.
 - **None**: Use default: "%d".

- **floatformat** (*str/False/None*) –
 - **str**: Format string to use with float.
 - **False**: Throw an exception if a float is encountered.
 - **None**: Use default: "%.6e".
- **decimalformat** (*str/False/None*) –
 - **str**: Format string to use with Decimal.
 - **False**: Throw an exception if a Decimal is encountered.
 - **None**: Use default: "%s".
- **mappingtypes** (*Iterable[type]/False/None*) –
 - **Iterable[type]**: Classes the should be encoded to objects. Must be iterable over their keys, and implement `__getitem__`.
 - **False**: There are no objects. Any object will be encoded as list of key-value tuples.
 - **None**: Use default: `[collections.abc.Mapping]`.

decimalformat

The creation argument decimalformat. None if False was specified.

floatformat

The creation argument floatformat. None if False was specified.

intformat

The creation argument intformat. None if False was specified.

mappingtypes

The creation argument mappingtypes. () if False was specified.

nan

The creation argument nan. None if False was specified.

neginfinity

The creation argument neginfinity. None if False was specified.

posinfinity

The creation argument posinfinity. None if False was specified.

tojson

The creation argument tojson. None if False was specified.

update (*self, **kw*)

Creates a new Options instance by modifying some members.

2.1.3 Encoder Compatibility Functions

`pyjson5.dump(obj, fp, **kw)`

Serializes a Python object to a JSON5 compatible unicode string.

Use `encode_io(...)` instead!

```
dump(obj, fp) == encode_io(obj, fp)
```

Parameters

- **obj** (*object*) – Python object to serialize.

- **fp** (*IOBase*) – A file-like object to serialize into.
- **kw** – Silently ignored.

`pyjson5.dumps(obj, **kw)`

Serializes a Python object to a JSON5 compatible unicode string.

Use *encode(...)* instead!

`dumps(obj) == encode(obj)`

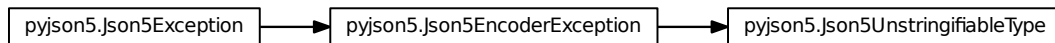
Parameters

- **obj** (*object*) – Python object to serialize.
- **kw** – Silently ignored.

Returns see `encode(data)`

Return type unicode

2.1.4 Encoder Exceptions



class `pyjson5.Json5EncoderException`

Base class of any exception thrown by the serializer.

message

Human readable error description

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class `pyjson5.Json5UnstringifiableType` (*message=None, unstringifiable=None*)

The encoder was not able to stringify the input, or it was told not to by the supplied `Options`.

message

Human readable error description

unstringifiable

The value that caused the problem.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

2.2 Parser / Decoder

All valid **JSON5 1.0.0** and **JSON** data can be read, unless the nesting level is absurdly high.

2.2.1 Quick Decoder Summary

<code>decode(data[, maxdepth, some])</code>	Decodes JSON5 serialized data from an <code>str</code> object.
<code>decode_buffer(obj[, maxdepth, some, wordlength])</code>	Decodes JSON5 serialized data from an object that supports the buffer protocol, e.g.
<code>decode_callback(cb[, maxdepth, some, args])</code>	Decodes JSON5 serialized data by invoking a callback.
<code>decode_io(fp[, maxdepth, some])</code>	Decodes JSON5 serialized data from a file-like object.
<code>load(fp, **kw)</code>	Decodes JSON5 serialized data from a file-like object.
<code>loads(s, *[, encoding])</code>	Decodes JSON5 serialized data from a string.
<code>Json5DecoderException([message, result])</code>	Base class of any exception thrown by the parser.
<code>Json5NestingTooDeep</code>	The maximum nesting level on the input data was exceeded.
<code>Json5EOF</code>	The input ended prematurely.
<code>Json5IllegalCharacter([message, result, ...])</code>	An unexpected character was encountered.
<code>Json5ExtraData([message, result, character])</code>	The input contained extraneous data.
<code>Json5IllegalType([message, result, value])</code>	The user supplied callback function returned illegal data.

2.2.2 Full Decoder Description

`pyjson5.decode(data, maxdepth=None, some=False)`

Decodes JSON5 serialized data from an `str` object.

```
decode('["Hello", "world!"]') == ['Hello', 'world!']
```

Parameters

- **data** (*unicode*) – JSON5 serialized data
- **maxdepth** (*Optional[int]*) – Maximum nesting level before the parsing is aborted.
 - If `None` is supplied, then the value of the global variable `DEFAULT_MAX_NESTING_LEVEL` is used instead.
 - If the value is 0, then only literals are accepted, e.g. `false`, `47.11`, or `"string"`.
 - If the value is negative, then the any nesting level is allowed until Python's recursion limit is hit.
- **some** (*bool*) – Allow trailing junk.

Raises

- `Json5DecoderException` – An exception occurred while decoding.
- `TypeError` – An argument had a wrong type.

Returns Deserialized data.

Return type `object`

`pyjson5.decode_buffer(obj, maxdepth=None, some=False, wordlength=None)`

Decodes JSON5 serialized data from an object that supports the buffer protocol, e.g. `bytearray`.

```
obj = memoryview(b'["Hello", "world!"]')
decode_buffer(obj) == ['Hello', 'world!']
```

Parameters

- **data** (*object*) – JSON5 serialized data. The argument must support Python’s buffer protocol, i.e. `memoryview(...)` must work. The buffer must be contiguous.
- **maxdepth** (*Optional[int]*) – see `decode(...)`
- **some** (*bool*) – see `decode(...)`
- **wordlength** (*Optional[int]*) – Must be 1, 2, 4 to denote UCS1, USC2 or USC4 data. Surrogates are not supported. Decode the data to an `str` if need be. If `None` is supplied, then the buffer’s `itemsize` is used.

Raises

- `Json5DecoderException` – An exception occurred while decoding.
- `TypeError` – An argument had a wrong type.
- `ValueError` – The value of `wordlength` was invalid.

Returns see `decode(...)`

Return type `object`

`pyjson5.decode_callback(cb, maxdepth=None, some=False, args=None)`
Decodes JSON5 serialized data by invoking a callback.

```
cb = iter('["Hello","world!"]').__next__  
  
decode_callback(cb) == ['Hello', 'world!']
```

Parameters

- **cb** (*Callable[Any, Union[str|bytes|bytearray|int|None]]*) – A function to get values from. The function is called like `cb(*args)`, and it returns:
 - **str, bytes, bytearray:** `len(...) == 0` denotes exhausted input. `len(...) == 1` is the next character.
 - **int:** `< 0` denotes exhausted input. `>= 0` is the ordinal value of the next character.
 - **None:** input exhausted
- **maxdepth** (*Optional[int]*) – see `decode(...)`
- **some** (*bool*) – see `decode(...)`
- **args** (*Optional[Iterable[Any]]*) – Arguments to call `cb` with.

Raises

- `Json5DecoderException` – An exception occurred while decoding.
- `TypeError` – An argument had a wrong type.

Returns see `decode(...)`

Return type `object`

`pyjson5.decode_io(fp, maxdepth=None, some=True)`
Decodes JSON5 serialized data from a file-like object.

```
fp = io.StringIO("""
    ['Hello', /* TODO look into specs whom to greet */
    'Wolrd' // FIXME: look for typos
    """)

decode_io(fp) == ['Hello']
decode_io(fp) == 'Wolrd'

fp.seek(0)

decode_io(fp, some=False)
# raises Json5ExtraData('Extra data U+0027 near 56', ['Hello'], "")
```

Parameters

- **fp** (*IOBase*) – A file-like object to parse from.
- **maxdepth** (*Optional[int] = None*) – see *decode(...)*
- **some** (*bool*) – see *decode(...)*

Raises

- *Json5DecoderException* – An exception occurred while decoding.
- *TypeError* – An argument had a wrong type.

Returns see *decode(...)*

Return type *object*

2.2.3 Decoder Compatibility Functions

`pyjson5.load(fp, **kw)`

Decodes JSON5 serialized data from a file-like object.

Use *decode_io(...)* instead!

```
load(fp) == decode_io(fp, None, False)
```

Parameters

- **fp** (*IOBase*) – A file-like object to parse from.
- **kw** – Silently ignored.

Returns see *decode(...)*

Return type *object*

`pyjson5.loads(s, *, encoding='UTF-8', **kw)`

Decodes JSON5 serialized data from a string.

Use *decode(...)* instead!

```
loads(s) == decode(s)
```

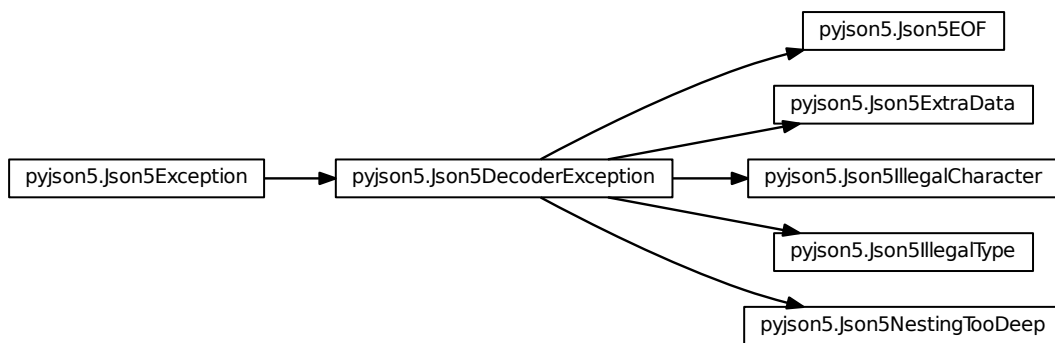
Parameters

- **s** (*object*) – Unless the argument is an `str`, it gets decoded according to the parameter encoding.
- **encoding** (*str*) – Codec to use if `s` is not an `str`.
- **kw** – Silently ignored.

Returns see `decode(...)`

Return type `object`

2.2.4 Decoder Exceptions



```
class pyjson5.Json5DecoderException(message=None, result=None, *args)
```

Base class of any exception thrown by the parser.

message

Human readable error description

result

Deserialized data up until now.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

```
class pyjson5.Json5NestingTooDeep
```

The maximum nesting level on the input data was exceeded.

message

Human readable error description

result

Deserialized data up until now.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

```
class pyjson5.Json5EOF
```

The input ended prematurely.

message

Human readable error description

result

Deserialized data up until now.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class pyjson5.Json5IllegalCharacter (*message=None, result=None, character=None, *args*)

An unexpected character was encountered.

character

Extraneous character.

message

Human readable error description

result

Deserialized data up until now.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class pyjson5.Json5ExtraData (*message=None, result=None, character=None, *args*)

The input contained extraneous data.

character

Extraneous character.

message

Human readable error description

result

Deserialized data up until now.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class pyjson5.Json5IllegalType (*message=None, result=None, value=None, *args*)

The user supplied callback function returned illegal data.

message

Human readable error description

result

Deserialized data up until now.

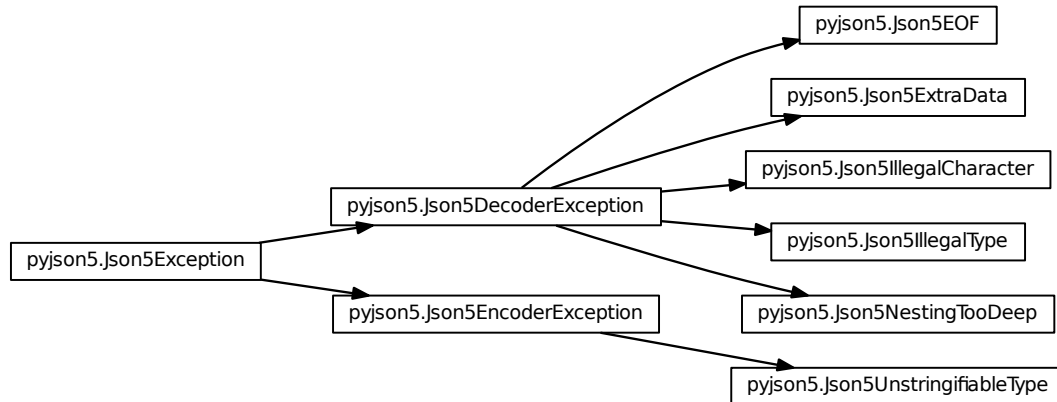
value

Value that caused the problem.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

2.3 Exceptions



```
class pyjson5.Json5Exception(message=None, *args)
```

Base class of any exception thrown by PyJSON5.

message

Human readable error description

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

2.4 Performance

This library is written in Cython for a better performance than a pure-Python implementation could give you.

2.4.1 Decoder Performance

The library is a bit slower than the shipped `json` module for *pure* JSON data. If you know that your input does not use JSON5 extension, then this library is probably not what you need.

- Dataset: <https://github.com/zemirco/sf-city-lots-json>
- CPU: Core i7-3770 @ 3.40GHz
- `pyjson5.decode()`: **4.58 s** ± 68.6 ms per loop (*lower is better*)
- `json.loads()`: **3.27 s** ± 27.7 ms per loop
- The decoder works correctly: `json.loads(content) == pyjson5.loads(content)`

2.4.2 Encoder Performance

The encoder generates pure JSON data if there are no infinite or NaN values in the input, which are invalid in JSON. The serialized data is XML-safe, i.e. there are no chevrons `<>`, ampersands `&`, apostrophes `'` or control characters in the output. The output is always ASCII regardless if you call `pyjson5.encode()` or `pyjson5.encode_bytes()`.

- Dataset: <https://github.com/zemirco/sf-city-lots-json>
- CPU: Core i7-3770 @ 3.40GHz
- `pyjson5.encode()`: **8.54** s $\pm$ 29.3 ms per loop (*lower is better*)
- `json.dumps()`: **4.68** s $\pm$ 20.4 ms per loop
- `json.dumps() + xml.sax.saxutils.escape()`: **5.02** s $\pm$ 141 ms per loop
- The encoder works correctly: `obj == json.loads(pyjson5.encode(obj, floatformat='%.16e'))`

Unless you need the advanced settings in `pyjson5.Options`, most most likely don't benefit from using this library as an encoder.

CHAPTER 3

Quick Summary

<code>decode(data[, maxdepth, some])</code>	Decodes JSON5 serialized data from an <code>str</code> object.
<code>decode_buffer(obj[, maxdepth, some, wordlength])</code>	Decodes JSON5 serialized data from an object that supports the buffer protocol, e.g.
<code>decode_callback(cb[, maxdepth, some, args])</code>	Decodes JSON5 serialized data by invoking a callback.
<code>decode_io(fp[, maxdepth, some])</code>	Decodes JSON5 serialized data from a file-like object.
<code>load(fp, **kw)</code>	Decodes JSON5 serialized data from a file-like object.
<code>loads(s, *[, encoding])</code>	Decodes JSON5 serialized data from a string.
<code>encode(data, *[, options])</code>	Serializes a Python object to a JSON5 compatible unicode string.
<code>encode_bytes(data, *[, options])</code>	Serializes a Python object to a JSON5 compatible bytes string.
<code>encode_callback(data, cb[, supply_bytes, ...])</code>	Serializes a Python object into a callback function.
<code>encode_io(data, fp[, supply_bytes, options])</code>	Serializes a Python object into a file-object.
<code>encode_noop(data, *[, options])</code>	Test if the input is serializable.
<code>dump(obj, fp, **kw)</code>	Serializes a Python object to a JSON5 compatible unicode string.
<code>dumps(obj, **kw)</code>	Serializes a Python object to a JSON5 compatible unicode string.
<code>Options</code>	Customizations for the <code>encoder_*(...)</code> function family.
<code>Json5EncoderException</code>	Base class of any exception thrown by the serializer.
<code>Json5DecoderException([message, result])</code>	Base class of any exception thrown by the parser.

CHAPTER 4

Compatibility

At least CPython 3.4, and a C++14 compatible compiler (such as GCC 5.2+) is needed.

Other interpreters such as Pypy and older CPython versions are not supported.

[Glossary / Index](#)

C

character (pyjson5.Json5ExtraData attribute), 15
character (pyjson5.Json5IllegalCharacter attribute), 15

D

decimalformat (pyjson5.Options attribute), 9
decode() (in module pyjson5), 11
decode_buffer() (in module pyjson5), 11
decode_callback() (in module pyjson5), 12
decode_io() (in module pyjson5), 12
dump() (in module pyjson5), 9
dumps() (in module pyjson5), 10

E

encode() (in module pyjson5), 6
encode_bytes() (in module pyjson5), 6
encode_callback() (in module pyjson5), 6
encode_io() (in module pyjson5), 7
encode_noop() (in module pyjson5), 7

F

floatformat (pyjson5.Options attribute), 9

I

intformat (pyjson5.Options attribute), 9

J

Json5DecoderException (class in pyjson5), 14
Json5EncoderException (class in pyjson5), 10
Json5EOF (class in pyjson5), 14
Json5Exception (class in pyjson5), 16
Json5ExtraData (class in pyjson5), 15
Json5IllegalCharacter (class in pyjson5), 15
Json5IllegalType (class in pyjson5), 15
Json5NestingTooDeep (class in pyjson5), 14
Json5UnstringifiableType (class in pyjson5), 10

L

load() (in module pyjson5), 13

loads() (in module pyjson5), 13

M

mappingtypes (pyjson5.Options attribute), 9
message (pyjson5.Json5DecoderException attribute), 14
message (pyjson5.Json5EncoderException attribute), 10
message (pyjson5.Json5EOF attribute), 14
message (pyjson5.Json5Exception attribute), 16
message (pyjson5.Json5ExtraData attribute), 15
message (pyjson5.Json5IllegalCharacter attribute), 15
message (pyjson5.Json5IllegalType attribute), 15
message (pyjson5.Json5NestingTooDeep attribute), 14
message (pyjson5.Json5UnstringifiableType attribute), 10

N

nan (pyjson5.Options attribute), 9
neginfinity (pyjson5.Options attribute), 9

O

Options (class in pyjson5), 8

P

posinfinity (pyjson5.Options attribute), 9

R

result (pyjson5.Json5DecoderException attribute), 14
result (pyjson5.Json5EOF attribute), 15
result (pyjson5.Json5ExtraData attribute), 15
result (pyjson5.Json5IllegalCharacter attribute), 15
result (pyjson5.Json5IllegalType attribute), 15
result (pyjson5.Json5NestingTooDeep attribute), 14

T

tojson (pyjson5.Options attribute), 9

U

unstringifiable (pyjson5.Json5UnstringifiableType attribute), 10
update() (pyjson5.Options method), 9

V

value (pyjson5.Json5IllegalType attribute), [15](#)

W

with_traceback() (pyjson5.Json5DecoderException method), [14](#)

with_traceback() (pyjson5.Json5EncoderException method), [10](#)

with_traceback() (pyjson5.Json5EOF method), [15](#)

with_traceback() (pyjson5.Json5Exception method), [16](#)

with_traceback() (pyjson5.Json5ExtraData method), [15](#)

with_traceback() (pyjson5.Json5IllegalCharacter method), [15](#)

with_traceback() (pyjson5.Json5IllegalType method), [15](#)

with_traceback() (pyjson5.Json5NestingTooDeep method), [14](#)

with_traceback() (pyjson5.Json5UnstringifiableType method), [10](#)