

---

# **GuessIt Documentation**

***Release 0.8***

**Nicolas Wack**

January 19, 2015



<b>1</b>	<b>Features</b>	<b>3</b>
<b>2</b>	<b>User Guide</b>	<b>5</b>
2.1	Installation . . . . .	5
2.2	Command-line usage . . . . .	6
2.3	Python module usage . . . . .	7
<b>3</b>	<b>Web Service API</b>	<b>9</b>
<b>4</b>	<b>Developer Guide</b>	<b>11</b>
4.1	Understanding the MatchTree . . . . .	11
4.2	What does the IterativeMatcher do? . . . . .	11
4.3	Merging all the results in a MatchTree to give a final Guess . . . . .	13
4.4	Guess . . . . .	13
4.5	MatchTree . . . . .	15
4.6	Matchers . . . . .	17
<b>5</b>	<b>Support</b>	<b>19</b>
<b>6</b>	<b>Contribute</b>	<b>21</b>
<b>7</b>	<b>License</b>	<b>23</b>
	<b>Python Module Index</b>	<b>25</b>



## Release v0.8 (*Installation*)

GuessIt is a python library that tries to extract as much information as possible from a video file.

It has a powerful filename matcher that allows to guess a lot of metadata from a video using only its filename. This matcher works with both movies and tv shows episodes.

For example, GuessIt can do the following:

```
$ guessit "Treme.1x03.Right.Place,.Wrong.Time.HDTV.XviD-NoTV.avi"
For: Treme.1x03.Right.Place,.Wrong.Time.HDTV.XviD-NoTV.avi
GuessIt found: {
  [1.00] "mimetype": "video/x-msvideo",
  [0.80] "episodeNumber": 3,
  [0.80] "videoCodec": "XviD",
  [1.00] "container": "avi",
  [1.00] "format": "HDTV",
  [0.70] "series": "Treme",
  [0.50] "title": "Right Place, Wrong Time",
  [0.80] "releaseGroup": "NoTV",
  [0.80] "season": 1,
  [1.00] "type": "episode"
}
```



---

## Features

---

At the moment, the filename matcher is able to recognize the following property types:

```
[ title,                                # for movies and episodes
  series, season,                       # for episodes only
  episodeNumber, episodeDetails,       # for episodes only
  date, year,                           # 'date' instance of datetime.date
  language, subtitleLanguage,          # instances of babelfish.Language
  country,                             # instances of babelfish.Country
  fileSize, duration,                  # when detecting video file metadata
  container, format,
  videoCodec, audioCodec,
  videoProfile, audioProfile,
  audioChannels, screenSize,
  releaseGroup, website,
  cdNumber, cdNumberTotal,
  filmNumber, filmSeries,
  bonusNumber, edition,
  idNumber,                            # tries to identify a hash or a serial number
  other
]
```

Guessit also allows you to compute a whole lot of hashes from a file, namely all the ones you can find in the hashlib python module (md5, sha1, ...), but also the Media Player Classic hash that is used (amongst others) by OpenSubtitles and SMPlayer, as well as the ed2k hash.

If you have the ‘guess-language’ python package installed, GuessIt can also analyze a subtitle file’s contents and detect which language it is written in.

If you have the ‘enzyme’ python package installed, GuessIt can also detect the properties from the actual video file metadata.





This part of the documentation, which is mostly prose, shows how to use Guessit both from the command-line and as a python module which you can use in your own projects.

## 2.1 Installation

This part of the documentation covers the installation of GuessIt. The first step to using any software package is getting it properly installed.

### 2.1.1 Distribute & Pip

Installing GuessIt is simple with `pip`:

```
$ pip install guessit
```

or, with `easy_install`:

```
$ easy_install guessit
```

But, you really *shouldn't do that*.

### 2.1.2 Get the Code

GuessIt is actively developed on GitHub, where the code is *always available*.

You can either clone the public repository:

```
git clone git://github.com/wackou/guessit.git
```

Download the *tarball*:

```
$ curl -L https://github.com/wackou/guessit/tarball/master -o guessit.tar.gz
```

Or, download the *zipball*:

```
$ curl -L https://github.com/wackou/guessit/zipball/master -o guessit.zip
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

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

## 2.2 Command-line usage

To have GuessIt try to guess some information from a filename, just run it as a command:

```
$ guessit "Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv"
For: Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
GuessIt found: {
    [1.00] "videoCodec": "h264",
    [1.00] "container": "mkv",
    [1.00] "format": "BluRay",
    [0.60] "title": "Dark City",
    [1.00] "releaseGroup": "CHD",
    [1.00] "screenSize": "720p",
    [1.00] "year": 1998,
    [1.00] "type": "movie",
    [1.00] "audioCodec": "DTS"
}
```

The numbers between square brackets indicate the confidence in the value, so for instance in the previous example, GuessIt is sure that the videoCodec is h264, but only 60% confident that the title is 'Dark City'.

You can use the `-v` or `--verbose` flag to have it display debug information.

You can use the `-p` or `-l` flags to display the properties names or the multiple values they can take.

You can also run a `--demo` mode which will run a few tests and display the results.

By default, GuessIt will try to autodetect the type of file you are asking it to guess, movie or episode. If you want to force one of those, use the `-t movie` or `-t episode` flags.

If input file is remote file or a release name with no folder and extension, you should use the `-n` or `--name-only` flag. It will disable folder and extension parsing, and any concrete file related analysis.

Guessit also allows you to specify the type of information you want using the `-i` or `--info` flag:

```
$ guessit -i hash_md5,hash_sha1,hash_ed2k tests/dummy.srt
For: tests/dummy.srt
GuessIt found: {
    [1.00] "hash_ed2k": "ed2k://|file|dummy.srt|44|1CA0B9DED3473B926AA93A0A546138BB|/",
    [1.00] "hash_md5": "e781de9b94ba2753a8e2945b2c0a123d",
    [1.00] "hash_sha1": "bfd18e2f4e5d59775c2bc14d80f56971891ed620"
}
```

You can see the list of options that guessit.py accepts like that:

```
$ guessit -h
Usage: guessit.py [options] file1 [file2...]
```

Options:

<code>-h, --help</code>	show this help message and exit
<code>-v, --verbose</code>	Display debug output
<code>-p, --properties</code>	Display properties that can be guessed.
<code>-l, --values</code>	Display property values that can be guessed.
<code>-s, --transformers</code>	Display transformers that can be used.
<code>-i INFO, --info=INFO</code>	The desired information type: filename, video, hash_mpc or a hash from python's hashlib module, such

	as hash_md5, hash_sha1, ...; or a list of any of them, comma-separated
-n, --name-only	Parse files as name only. Disable folder parsing, extension parsing, and file content analysis.
-t TYPE, --type=TYPE	The suggested file type: movie, episode. If undefined, type will be guessed.
-a, --advanced	Display advanced information for filename guesses, as json output
-y, --yaml	Display information for filename guesses as yaml output (like unit-test)
-d, --demo	Run a few builtin tests instead of analyzing a file
-b, --bug	Submit a wrong detection to the guessit.io service

## 2.3 Python module usage

The main entry points to the python module are the `guess_file_info`, `guess_movie_info` and `guess_episode_info`.

The `guess_file_info` function will try to autodetect the type of the file, either movie, moviesubtitle, movieinfo, episode, episodesubtitle or episodeinfo.

Pass them the filename and the desired information type:

```
>>> import guessit
>>> path = 'Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv'
>>> guess = guessit.guess_movie_info(path, info=['filename'])

>>> print type(guess)
<class 'guessit.guess.Guess'>

>>> print guess
{'videoCodec': 'h264', 'container': 'mkv', 'format': 'BluRay',
'title': 'Dark City', 'releaseGroup': 'CHD', 'screenSize': '720p',
'year': 1998, 'type': 'movie', 'audioCodec': 'DTS'}

>>> print guess.nice_string()
{
  [1.00] "videoCodec": "h264",
  [1.00] "container": "mkv",
  [1.00] "format": "BluRay",
  [0.60] "title": "Dark City",
  [1.00] "releaseGroup": "CHD",
  [1.00] "screenSize": "720p",
  [1.00] "year": 1998,
  [1.00] "type": "movie",
  [1.00] "audioCodec": "DTS"
}
```

A `Guess` instance is a dictionary which has an associated confidence for each of the properties it has.

A `Guess` instance is also a python dict instance, so you can use it wherever you would use a normal python dict.

If you have the `Enzyme` python package installed, then the `guess_video_metadata` function is also available, which will return a guess with the properties from the video file metadata.



---

## Web Service API

---

The guessit.io server also provides a free webservice that allows you to perform filename detection, even you don't have python installed (eg: you need to use it from an Android app, or NodeJS, etc.).

You can look at the documentation for the web API here: <http://api.guessit.io>



# Developer Guide

If you want to contribute to the project, this part of the documentation is for you.

## 4.1 Understanding the MatchTree

The basic structure that the filename detection component uses is the `MatchTree`. A `MatchTree` is a tree covering the filename, where each node represent a substring in the filename and can have a `Guess` associated with it that contains the information that has been guessed in this node. Nodes can be further split into subnodes until a proper split has been found.

This makes it so that all the leaves concatenated will give you back the original filename. But enough theory, let's look at an example:

[illegible]

The last line contains the filename, which you can use a reference. The previous line contains the type of property that has been found. The line before that contains the filename, where all the found groups have been blanked. Basically, what is left on this line are the leftover groups which could not be identified.

The lines before that indicate the indices of the groups in the tree.

For instance, the part of the filename ‘BDRip’ is the leaf with index (2, 2, 0, 0, 0, 1) (read from top to bottom), and its meaning is ‘format’ (as shown by the  $\mathfrak{f}$ ’s on the last-but-one line).

## 4.2 What does the IterativeMatcher do?

The goal of the *api/matcher* is to take a `MatchTree` which contains no information (yet!) at the beginning, and apply a succession of rules to try to guess parts of the filename. These rules are called transformations and work in-place on the tree, splitting into new leaves and updating the nodes's guesses when it finds some information.

Let's look at what happens when matching the previous filename.

### 4.2.1 Splitting into path components

First, we split the filename into folders + basename + extension This gives us the following tree, which has 4 leaves (from 0 to 3):

```
000000 1111111111111111 222222222222222222222222222222222222222222222222222 333
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
```

### 4.2.2 Splitting into explicit groups

Then, we want to split each of those groups into “explicit” groups, i.e.: groups which are enclosed in parentheses, square brackets, curly braces, etc.:

```

000000 1111111111111111 2222222222222222222222222222222222222222222222222 333
000000 0000000000111111 00000000001111112222222222222222222222222222 000
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.____.
                                                                CCC
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv

```

As you can see, the containing folder has been split into 2 sub-groups, and the basename into 3 groups (separated by the year information).

Note that we also got the information from the extension, as you can see above.

### 4.2.3 Finding interesting patterns

Now that this first split has been made, we can start finding some known patterns which we can identify in the filename. That is the main objective of the `IterativeMatcher`, which will run a series of transformations which can identify groups in the filename and will annotate the corresponding nodes.

For instance, the year:

```

000000 1111111111111111 2222222222222222222222222222222222222222222222222 333
000000 0000000000011111 000000000011111222222222222222222222222222222 000
                                011112          011112
Movies/Dark City (____)/Dark.City. (____).DC.BDRip.720p.DTS.X264-CHD.____
                                YYYY          YYYY                                CCC
Movies/Dark City (1998)/Dark.City. (1998).DC.BDRip.720p.DTS.X264-CHD.mkv

```

Then, known properties usually found in video filenames:

```

000000 1111111111111111 222222222222222222222222222222222222 333
000000 0000000000111111 00000000001111112222222222222222222222 000
          011112              011112000000000000000000000000111
                                0000000000000000000011112
                                0000000000111122222
                                0000111112      01112

Movies/Dark City (____)/Dark.City.(____).DC._____._____.____-____.____
                yyyy               yyyy         fffff ssss aaa vvvv rrr ccc
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
```

As you can see, this starts to branch pretty quickly, as each found group splits a leaf into further leaves. In this case, that gives us the year (1998), the format (BDRip), the screen size (720p), the video codec (x264) and the release group (CHD).



#### 4.2.4 Using positional rules to find the ‘title’ property

Now that we found all the known patterns that we could, it is time to try to guess what is the title of the movie. This is done by looking at which groups in the filename are still unidentified, and trying to guess which one corresponds to the title by looking at their position:

```
000000 1111111111111111 222222222222222222222222222222222222 333
000000 0000000000111111 000000000011111122222222222222222222 000
          011112          0111120000000000000000000000000000111
                                0000000000000000000011112
                                00000000000111122222
                                0000111112      01112
Movies/_____(____)/Dark.City.(____).DC.____.____.____.____-____.____
      tttttttttt yyyy          yyyy      ffffff ssss aaa vvvv rrr ccc
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
```

In this case, as the containing folder is composed of 2 groups, the second of which is the year, we can (usually) safely assume that the first one corresponds to the movie title.

### 4.3 Merging all the results in a MatchTree to give a final Guess

Once that we have matched as many groups as we could, the job is not done yet. Indeed, every leaf of the tree that we could identify contains the found property in its guess, but what we want at the end is to have a single `Guess` containing all the information.

There are some simple strategies implemented to try to deal with conflicts and/or duplicate properties. In our example, ‘year’ appears twice, but as it has the same value, so it will be merged into a single ‘year’ property, but with a confidence that represents the combined confidence of both guesses. If the properties were conflicting, we would take the one with the highest confidence and lower it accordingly.

Here:

```
>>> path = 'Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv'
>>> print guessit.guess_movie_info(path)
{'videoCodec': 'h264', 'container': 'mkv', 'format': 'BluRay',
'title': 'Dark City', 'releaseGroup': 'CHD', 'screenSize': '720p',
'year': 1998, 'type': 'movie', 'audioCodec': 'DTS'}
```

And that gives you your final guess!

You may also want to familiarize yourself with the following classes:

## 4.4 Guess

**class** `guessit.guess.Guess` (*\*args*, *\*\*kwargs*)

A `Guess` is a dictionary which has an associated confidence for each of its values.

As it is a subclass of `dict`, you can use it everywhere you expect a simple `dict`.

**metadata** (*prop=None*)

Return the metadata associated with the given property name

If no property name is given, get the `global_metadata`

**nice\_string** (*advanced=False*)

Return a string with the property names and their values, that also displays the associated confidence to each property.

FIXME: doc with param

**to\_dict** (*advanced=False*)

Return the guess as a dict containing only base types, ie: where dates, languages, countries, etc. are converted to strings.

if advanced is True, return the data as a json string containing also the raw information of the properties.

**update\_highest\_confidence** (*other*)

Update this guess with the values from the given one. In case there is property present in both, only the one with the highest one is kept.

`guessit.guess.choose_int(g1, g2)`

Function used by `merge_similar_guesses` to choose between 2 possible properties when they are integers.

`guessit.guess.choose_string(g1, g2)`

Function used by `merge_similar_guesses` to choose between 2 possible properties when they are strings.

If the 2 strings are similar, or one is contained in the other, the latter is returned with an increased confidence.

If the 2 strings are dissimilar, the one with the higher confidence is returned, with a weaker confidence.

Note that here, ‘similar’ means that 2 strings are either equal, or that they differ very little, such as one string being the other one with the ‘the’ word prepended to it.

```
>>> s(choose_string(('Hello', 0.75), ('World', 0.5)))
('Hello', 0.25)
```

```
>>> s(choose_string(('Hello', 0.5), ('hello', 0.5)))
('Hello', 0.75)
```

```
>>> s(choose_string(('Hello', 0.4), ('Hello World', 0.4)))
('Hello', 0.64)
```

```
>>> s(choose_string(('simpsons', 0.5), ('The Simpsons', 0.5)))
('The Simpsons', 0.75)
```

`guessit.guess.merge_similar_guesses(guesses, prop, choose)`

Take a list of guesses and merge those which have the same properties, increasing or decreasing the confidence depending on whether their values are similar.

`guessit.guess.merge_all(guesses, append=None)`

Merge all the guesses in a single result, remove very unlikely values, and return it. You can specify a list of properties that should be appended into a list instead of being merged.

```
>>> s(merge_all([ Guess({'season': 2}, confidence=0.6),
...                  Guess({'episodeNumber': 13}, confidence=0.8) ]))
... ) == {'season': 2, 'episodeNumber': 13}
True
```

```
>>> s(merge_all([ Guess({'episodeNumber': 27}, confidence=0.02),
...                  Guess({'season': 1}, confidence=0.2) ]))
... ) == {'season': 1}
True
```

```
>>> s(merge_all([ Guess({'other': 'PROPER'}, confidence=0.8),
...                  Guess({'releaseGroup': '2HD'}, confidence=0.8) ],
...          append=['other']))
```

```
... ) == {'releaseGroup': '2HD', 'other': ['PROPER']}
True
```

## 4.5 MatchTree

```
class guessit.matchtree.BaseMatchTree (string=u'', span=None, parent=None)
```

A `BaseMatchTree` is a tree covering the filename, where each node represents a substring in the filename and can have a `Guess` associated with it that contains the information that has been guessed in this node. Nodes can be further split into subnodes until a proper split has been found.

**Each node has the following attributes:**

- string = the original string of which this node represents a region
- span = a pair of (begin, end) indices delimiting the substring
- parent = parent node
- children = list of children nodes
- guess = Guess()

BaseMatchTrees are displayed in the following way:

```
>>> path = 'Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv'
>>> print(guessit.IterativeMatcher(path).match_tree)
000000 1111111111111111 22222222222222222222222222222222222222222222222222222 333
000000 0000000000011111 000000000011111122222222222222222222222222222222 000
          011112              0111120000111111222222222222222222222222 000
                                   0111122222222222222222222222222222222222
                                           00000111122222
                                               01112      0111
Movies/_____(____)/Dark.City.(____).DC._____.____._____-._____.____
        tttttttt yy y           yy y       ffffff ssss aaa vvvv rrr ccc
Movies/Dark City (1998)/Dark.City.(1998).DC.BDRip.720p.DTS.X264-CHD.mkv
```

The last line contains the filename, which you can use a reference. The previous line contains the type of property that has been found. The line before that contains the filename, where all the found groups have been blanked. Basically, what is left on this line are the leftover groups which could not be identified.

The lines before that indicate the indices of the groups in the tree.

For instance, the part of the filename ‘BDRip’ is the leaf with index  $(2, 2, 1)$  (read from top to bottom), and its meaning is ‘format’ (as shown by the  $\mathfrak{f}$ ’s on the last-but-one line).

**add\_child**(*span*)

Add a new child node to this node with the given span.

clean value

Return a cleaned value of the matched substring, with better presentation formatting (punctuation marks removed, duplicate spaces, ...)

depth

Return the depth of this node.

```
get_partition_spans(indices)
```

Return the list of absolute spans for the regions of the original string defined by splitting this node at the given indices (relative to this node)

**info**  
Return a dict containing all the info guessed by this node, subnodes included.

**is\_leaf()**  
Return whether this node is a leaf or not.

**leaves()**  
Return a generator over all the nodes that are leaves.

**next\_leaf(*leaf*)**  
Return next leaf for this node

**next\_leaves(*leaf*)**  
Return next leaves for this node

**node\_at(*idx*)**  
Return the node at the given index in the subtree rooted at this node.

**node\_idx**  
Return this node's index in the tree, as a tuple. If this node is the root of the tree, then return ().

**nodes()**  
Return all the nodes and subnodes in this tree.

**nodes\_at\_depth(*depth*)**  
Return all the nodes at a given depth in the tree

**partition(*indices*)**  
Partition this node by splitting it at the given indices, relative to this node.

**previous\_leaf(*leaf*)**  
Return previous leaf for this node

**previous\_leaves(*leaf*)**  
Return previous leaves for this node

**root**  
Return the root node of the tree.

**to\_string()**  
Return a readable string representation of this tree.

**The result is a multi-line string, where the lines are:**

- line 1 -> N-2: each line contains the nodes at the given depth in the tree
- line N-2: original string where all the found groups have been blanked
- line N-1: type of property that has been found
- line N: the original string, which you can use a reference.

**value**  
Return the substring that this node matches.

**class** guessit.matchtree.**MatchTree**(*string=u'', span=None, parent=None*)

The MatchTree contains a few “utility” methods which are not necessary for the BaseMatchTree, but add a lot of convenience for writing higher-level rules.

**first\_leaf\_containing(*property\_name*)**  
Return the first leaf containing the given property.

**is\_explicit()**  
Return whether the group was explicitly enclosed by parentheses/square brackets/etc.





---

### Support

---

The project website for GuessIt is hosted at [ReadTheDocs](#). There you will also find the User guide and Developer documentation.

This project is hosted on GitHub: <https://github.com/wackou/guessit>

Please report issues and/or feature requests via the [bug tracker](#).

You can also report issues using the command-line tool:

```
$ guessit --bug "filename.that.fails.avi"
```





---

# Contribute

---

GuessIt is under active development, and contributions are more than welcome!

1. Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug. There is a Contributor Friendly tag for issues that should be ideal for people who are not very familiar with the codebase yet.
2. Fork [the repository](#) on Github to start making your changes to the **master** branch (or branch off of it).
3. Write a test which shows that the bug was fixed or that the feature works as expected.
4. Send a pull request and bug the maintainer until it gets merged and published. :)



---

### License

---

GuessIt is licensed under the [LGPLV3](#) license.



## g

`guessit.guess`, [13](#)

`guessit.matcher`, [17](#)

`guessit.matchtree`, [15](#)



**A**

`add_child()` (`guessit.matchtree.BaseMatchTree` method), 15

**B**

`BaseMatchTree` (class in `guessit.matchtree`), 15

**C**

`choose_int()` (in module `guessit.guess`), 14  
`choose_string()` (in module `guessit.guess`), 14  
`clean_value` (`guessit.matchtree.BaseMatchTree` attribute), 15

**D**

`depth` (`guessit.matchtree.BaseMatchTree` attribute), 15

**F**

`first_leaf_containing()` (`guessit.matchtree.MatchTree` method), 16

**G**

`get_partition_spans()` (`guessit.matchtree.BaseMatchTree` method), 15  
`Guess` (class in `guessit.guess`), 13  
`guessit.guess` (module), 13  
`guessit.matcher` (module), 17  
`guessit.matchtree` (module), 15

**I**

`info` (`guessit.matchtree.BaseMatchTree` attribute), 15  
`is_explicit()` (`guessit.matchtree.MatchTree` method), 16  
`is_leaf()` (`guessit.matchtree.BaseMatchTree` method), 16  
`IterativeMatcher` (class in `guessit.matcher`), 17

**L**

`leaves()` (`guessit.matchtree.BaseMatchTree` method), 16  
`leaves_containing()` (`guessit.matchtree.MatchTree` method), 16

**M**

`matched()` (`guessit.matchtree.MatchTree` method), 17  
`MatchTree` (class in `guessit.matchtree`), 16  
`merge_all()` (in module `guessit.guess`), 14  
`merge_similar_guesses()` (in module `guessit.guess`), 14  
`metadata()` (`guessit.guess.Guess` method), 13

**N**

`next_leaf()` (`guessit.matchtree.BaseMatchTree` method), 16  
`next_leaves()` (`guessit.matchtree.BaseMatchTree` method), 16  
`nice_string()` (`guessit.guess.Guess` method), 13  
`node_at()` (`guessit.matchtree.BaseMatchTree` method), 16  
`node_idx` (`guessit.matchtree.BaseMatchTree` attribute), 16  
`nodes()` (`guessit.matchtree.BaseMatchTree` method), 16  
`nodes_at_depth()` (`guessit.matchtree.BaseMatchTree` method), 16

**P**

`partition()` (`guessit.matchtree.BaseMatchTree` method), 16  
`previous_leaf()` (`guessit.matchtree.BaseMatchTree` method), 16  
`previous_leaves()` (`guessit.matchtree.BaseMatchTree` method), 16  
`previous_leaves_containing()` (`guessit.matchtree.MatchTree` method), 17  
`previous_unidentified_leaves()` (`guessit.matchtree.MatchTree` method), 17

**R**

`root` (`guessit.matchtree.BaseMatchTree` attribute), 16

**T**

`to_dict()` (`guessit.guess.Guess` method), 14  
`to_string()` (`guessit.matchtree.BaseMatchTree` method), 16

## U

`unidentified_leaves()` (`guessit.matchtree.MatchTree`  
method), [17](#)

`update_highest_confidence()` (`guessit.guess.Guess`  
method), [14](#)

## V

`value` (`guessit.matchtree.BaseMatchTree` attribute), [16](#)