

# Basic Usage

- Loading files/dictionaries
- Getting nodes
- Getting a value
- Setting values
- Saving
- Getting the names of the children
- Checking for a value
- Deleting a Node
- Getting the name of a Node
- Math and object manipulation
- Getting the children

# Loading files/dictionaries

## Syntax:

```
Node(file_or_dictionary, **file_parameters)
```

---

To load a file/dictionary, simply create a new Node object with the filename or dictionary as first argument, along with any optional file arguments you want to pass (See [The File object](#)). For default filetype mappings, see [File types](#).

```
db = Node('file.pyn')
```

```
db = Node('file.abc', filetype='txt', autsave=True)
```

# Getting nodes

## Syntax:

```
Node.get(*children)
```

There are two ways to retrieve a child Node:

```
root_node.get('child')  
root_node.child
```

You can also retrieve deeper nodes like so:

```
root_node.a.b.c  
root_node.get('a').b.get('c')
```

Notice that it doesn't particularly matter whether you use the `.get()` method or simply access the Node as an attribute.

Remember, getting a node means accessing the child node object - to get the value, see [Getting values](#).

# Getting a value

## Syntax:

```
Node()
```

---

pyntree takes a somewhat unique yet simple approach to getting the values stored in nodes thanks to python limitations. To get a node's value, simply call it:

```
root_node()
```

The same applies to child nodes.

To get the names of all the child nodes, see [Getting a list of all children](#).

# Setting values

## Syntax:

```
Node.set(*children, value)
```

To change a value or create a new node, you can use one of two general methods:

The set() method:

```
your_node.set('name', 'Jimmy')
```

Or by directly setting the attribute:

```
your_node.name = 'Jimmy'
```

Remember, if a node doesn't exist to begin with, you can't create children of it!  
The below code will NOT WORK:

```
your_node.doesnt_exist.value = 3
```

# Saving

## Syntax:

```
Node.save(filename=None)
```

Simply call the save method to save your data:

```
root_node.save()
```

You can also call the save method on child nodes, but it will save the data in the root node to the proper file.

You can also specify a different filename to (temporarily) save to:

```
root_node.save(filename="temp.file")
```

For changing the file to be saved to (in a more permanent sense), see [\[replaceme\]](#)

# Getting the names of the children

## Syntax:

```
Node._values
```

To get a list of all of a Node's children, simply use the `values` property:

```
Node._values
```

This will return a list of strings, not Nodes.

If you have a child Node named "`_values`", you will need to use the `get` function to retrieve it. See [here](#) for more details.

`.values` -> `._values` as of 1.0.0

# Checking for a value

## Syntax:

```
Node.has(*names)
```

---

To see if a Node has a child with a given name, use the `has` method:

```
Node.has(name)
```

# Deleting a Node

## Syntax:

```
Node.delete(*children)
```

The delete function can take either 0 or 1+ parameters:

```
Node.delete()
```

```
Node.delete(name)
```

In the first example, the node on which the function is called will be deleted. In the latter, the child with the specified name will be deleted.

When we speak of "deleting nodes" here, we are referring to deleting the data which those Nodes represent.

# Getting the name of a Node

## Syntax:

```
Node._name
```

Let's say you have a miscellaneous Node that got passed to a function somehow. What is its purpose? The name of the Node may shed some light on this:

```
from pyntree import Node
x = Node('test.pyn')
x._name # 'test.pyn'
x.a = 1
x.a._name # 'a'
```

A root node without a filename (pure dictionary) will return the string 'None' (not the object, for compatibility reasons).

If you have a child Node named "`_name`", you will need to use the `get` function to retrieve it. See [here](#) for more details.

`.name` -> `._name` as of 1.0.0

# Math and object manipulation

## Syntax:

```
Node += val
```

---

## Math operations

If you used pyndb, you probably had to do something like this:

```
x = PYNDatabase({})  
x.set("a", 1)  
x.set("a", x.a.val + 1)
```

Let's be real here, this **SUCKS**. pyntree does it better:

```
x = Node()  
x.a = 1  
x.a += 1
```

Wow, that's infinitely simpler and less painful, right?!

## Object operations

You can interact with a Node's data directly once you retrieve it:

```
x = Node()  
x.a = [1,2,3,4]  
x.a().append(5)  
print(x.a()) # -> [1,2,3,4,5]
```

# Getting the children

## Syntax:

```
Node._children
```

---

The `._children` property returns a list of Node objects (the children of the parent Node)