go-jmespath - A JMESPath implementation in Go

go-jmespath is a GO implementation of JMESPath, which is a query language for JSON. It will take a JSON document and transform it into another JSON document through a JMESPath expression.

Using go-jmespath is really easy. There's a single function you use, `jmespath.search`:

```
> import "github.com/jmespath/go-jmespath"
>
> var jsondata = []byte(`{"foo": {"bar": {"baz": [0, 1, 2, 3, 4]}}}`) // your data
> var data interface{}
> err := json.Unmarshal(jsondata, &data)
> result, err := jmespath.Search("foo.bar.baz[2]", data)
result = 2
```

In the example we gave the `search` function input data of `{"foo": {"bar": {"baz": [0, 1, 2, 3, 4]}}}` as well as the JMESPath expression `foo.bar.baz[2]`, and the `search` function evaluated the expression against the input data to produce the result `2`.

The JMESPath language can do a lot more than select an element from a list. Here are a few more examples:

```
> var jsondata = []byte(`{"foo": {"bar": {"baz": [0, 1, 2, 3, 4]}}}`) // your data
> var data interface{}
> err := json.Unmarshal(jsondata, &data)
> result, err := jmespath.search("foo.bar", data)
result = { "baz": [ 0, 1, 2, 3, 4 ] }


> var jsondata  = []byte(`{"foo": [{"first": "a", "last": "b"},
                           {"first": "c", "last": "d"}]}`) // your data
> var data interface{}
> err := json.Unmarshal(jsondata, &data)
> result, err := jmespath.search({"foo[*].first", data)
result [ 'a', 'c' ]


> var jsondata = []byte(`{"foo": [{"age": 20}, {"age": 25},
                           {"age": 30}, {"age": 35},
                           {"age": 40}]}`) // your data
> var data interface{}
> err := json.Unmarshal(jsondata, &data)
> result, err := jmespath.search("foo[?age > `30`]")
result = [ { age: 35 }, { age: 40 } ]
```

You can also pre-compile your query. This is usefull if you are going to run multiple searches with it:

```
> var jsondata = []byte(`{"foo": "bar"}`)
> var data interface{}
> err := json.Unmarshal(jsondata, &data)
> precompiled, err := Compile("foo")
> if err != nil{
>   // ... handle the error
> }
> result, err := precompiled.Search(data)
result = "bar"
```

More Resources

The example above only show a small amount of what a JMESPath expression can do. If you want to take a tour of the language, the *best* place to go is the JMESPath Tutorial.

One of the best things about JMESPath is that it is implemented in many different programming languages including python, ruby, php, lua, etc. To see a complete list of libraries, check out the JMESPath libraries page.

And finally, the full JMESPath specification can be found on the JMESPath site.