Slide the First

Mythical Creatures

http://weknowyourdreams.com/images/unicorn/unicorn-04.jpg

https://shellback0608.files.wordpress.com/2016/07/bird-phoenix-flight-art-drawing.jpg

https://i.ytimg.com/vi/kKJQESF0qqk/maxresdefault.jpg

API Docs

Exploring JSON With jq

Aijaz Ansari

@_aijaz_

What is JSON?

JavaScript Object Notation

```
[
 { "name": "Alice", "age": 20},
 { "name": "Bob", "age": 30}
]
```

Reading JSON returned by remote servers

```
@interface AAAJSONResponseSerializer : AFJSONResponseSerializer
@end


@implementation AAAJSONResponseSerializer

- (id)responseObjectForResponse:(NSURLResponse *)response
                           data:(NSData *)data
                          error:(NSError *__autoreleasing *)error {

#ifdef DEBUG
  NSHTTPURLResponse * httpResponse = (NSHTTPURLResponse *) response;
  NSString *body = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
  DDLogInfo(@"RESPONSE BODY: %@", response.URL, body);
  DDLogInfo(@"RESPONSE: (%ld) %@ ", httpResponse.statusCode, response.URL);
#endif
  // ...
}

@end
```

JSON Looks Like This


```
[{"name":"Aang","sex":"M","born":-12,"died":153,"bending":["Air","Water","Earth","Fire","Energy"],"identity":{"nationality":"Southern Air Temple",
"ethnicity":"Air Nomad"},"spouse":"Katara","children":[{"sex":"M","name":"Bumi"},{"sex":"F","name":"Kya"},{"sex":"M","name":"Tenzin"}]},{"name":"Katara","sex":
"F","born":85,"died":null,"bending":["Water","Blood"],"identity":{"nationality":"Southern Water Tribe","ethnicity":"Water Tribe"},"spouse":"Aang","children":[{"
sex":"M","name":"Bumi"},{"sex":"F","name":"Kya"},{"sex":"M","name":"Tenzin"}]},{"name":"Sokka","sex":"M","born":84,"died":164,"bending":[],"identity":
{"nationality":"Southern Water Tribe","ethnicity":"Water Tribe"},"spouse":null,"children":[]},{"name":"Toph Beifong","sex":"F","born":88,"died":null,"bending":
["Earth","Metal"],"identity":{"nationality":"Gaoling, Earth Kingdom","ethnicity":"Earth Kingdom"},"spouse":null,"children":[{"sex":"F","name":
"Lin Beifong"},{"sex":"F","name":"Suyin Beifong"}]},{"name":"Iroh","sex":"M","born":null,"died":null,"bending":["Fire","Energy"],"identity":{"nationality":
"Fire Nation Capital, Fire Nation","ethnicity":"Fire Nation"},"spouse":null,"children":[{"sex":"M","name":"Lu Ten"}]},{"name":"Zuko","sex":"M","born":83,
"died":null,"bending":["Fire","Energy"],"identity":{"nationality":"Fire Nation Capital, Fire Nation","ethnicity":"Fire Nation"},"spouse":null,"children":
[{"sex":"F","name":"Izumi"}]},{"name":"Kya","sex":"F","born":null,"died":null,"bending":["Water"],"identity":{"nationality":"Southern Water Tribe","ethnicity":
"Water Tribe, Air Nomad"},"spouse":null,"children":[]},{"name":"Bumi","sex":"M","born":null,"died":null,"bending":["Air"],"identity":{"nationality":
"United Republic","ethnicity":"Water Tribe, Air Nomad"},"spouse":null,"children":[]},{"name":"Tenzin","sex":"M","born":null,"died":null,"bending":["Air"],
"identity":{"nationality":"Republic City, United Republic","ethnicity":"Water Tribe, Air Nomad"},"spouse":null,"children":[{"sex":"F","name":"Jinora"},{"sex":
"F","name":"Ikki"},{"sex":"M","name":"Meelo"},{"sex":"M","name":"Rohan"}]},{"name":"Lin Beifong","sex":"F","born":120,"died":null,"bending":["Earth","Metal"],
"identity":{"nationality":"Republic City, United Republic","ethnicity":"Earth Kingdom"},"spouse":null,"children":[]},{"name":"Suyin Beifong","sex":"F","born":
126,"died":null,"bending":["Earth","Metal"],"identity":{"nationality":"Republic City, United Republic","ethnicity":"Earth Kingdom"},"spouse":null,"children":
[{"sex":"M","name":"Bataar Jr."},{"sex":"F","name":"Opal"},{"sex":"M","name":"Wei"},{"sex":"M","name":"Wing"},{"sex":"M","name":"Huan"}]}]
```

jq

Lightweight & Flexible Command-Line JSON Processor.

https://stedolan.github.io/jq/

Pretty Printing

```
$ echo '[{"a": 1},{"b": 2}]' | jq .
[
  {
    "a": 1
  },
  {
    "b": 2
  }
]
$
```

If you remember only one thing about jq, let it be this, the '.' filter.

Filters

→ All jq programs are filters

→ A filter takes in input & produces output

→ **1** is the simplest filter: return `1`

→ **.** is the simplest useful filter: pretty print

Object Value Filter

```
$ echo '{"a": 1, "b": 2, "c":3}' | jq .c
3
$
```

Arrays

```
$ echo '["a", "b", "c", "d", "e"]' | jq '.[2]'
"c"
$ echo '["a", "b", "c", "d", "e"]' | jq '.[1:4]'
[
  "b",
  "c",
  "d"
]
$
```

Iterators

Read `[]` as `foreach`

```
$ echo '["a", "b", "c", "d", "e"]' | jq '.[]'
"a"
"b"
"c"
"d"
"e"
$
```

Read `.[]` as iterating with an identity filter

Addition

```
$ echo '["a", "b", "c", "d", "e"]' | jq '.[]+" hello"'
"a hello"
"b hello"
"c hello"
"d hello"
"e hello"
$
```

Addition

```
$ echo '["a", "b", "c", "d", "e"]' | jq '"hey "+ .[]+" hello"'
"hey a hello"
"hey b hello"
"hey c hello"
"hey d hello"
"hey e hello"
$
```

Addition

```
$ echo '[1,2,3,4,5]' | jq '.[]+10'
11
12
13
14
15
$
```

Collecting Results Into an Array

```
$ echo '[1,2]' | jq '.[]+10'
11
12
$ echo '[1,2]' | jq '[.[]+10]'
[
  11,
  12
]
```

Filters can be piped with |

Creating Objects

```
$ echo '[{"a": 1, "b": 2, "c": 3}, {"a": 10, "b": 20, "c": 30}]' | jq '.[]'
{
  "a": 1,
  "b": 2,
  "c": 3
}
{
  "a": 10,
  "b": 20,
  "c": 30
}
$
```

Creating Objects

```
$ echo '[{"a": 1, "b": 2, "c": 3}, {"a": 10, "b": 20, "c": 30}]' | jq '.[]|{a,c}'
{
  "a": 1,
  "c": 3
}
{
  "a": 10,
  "c": 30
}
$
```

If you remember only 2 things about jq, let this be the second: extracting just the keys you care about.

Creating Objects

```
$ echo '[{"fn": "Ann", "ln": "Smith"},{"fn": "Ted", "ln": "Jones"}]' \
\
  | jq '.[]|{"first": .fn, "last": .ln}'
{
  "first": "Ann",
  "last": "Smith"
}
{
  "first": "Ted",
  "last": "Jones"
}
$
```

Creating Objects

```
$ echo '[{"fn": "Ann", "ln": "Smith"},{"fn": "Ted", "ln": "Jones"}]' \
\
| jq '.[]|{(.fn): .ln}'
{
  "Ann": "Smith"
}
{
  "Ted": "Jones"
}
$
```

Functions

keys

```
$ echo '[{"a": 1, "b": 2, "c": 3}, {"a": 10, "b": 20, "c": 30}]' | jq '.[]|keys'
[
  "a",
  "b",
  "c"
]
[
  "a",
  "b",
  "c"
]
$
```

If you remember only 3 things about jq, let this be the third: examining keys.

Functions

length

```
$ echo '[{"a": 1, "b": 2, "c": 3}, {"a": 10, "b": 20, "c": 30}]' | jq '.[]|length'
3
3
$
```

Functions

map

```
$ echo '[1,2,3,4,5]' | jq 'map(.+10)'
[
  11,
  12,
  13,
  14,
  15
]
```

`map(x)` **is equivalent to** `[.[] | x]`

Addition

```
$ echo '[1,2,3,4,5]' | jq '.[]+10'
11
12
13
14
15
$
```

Addition

```
$ echo '[1,2,3,4,5]' | jq '[.[]+10]'
[
  11,
  12,
  13,
  14,
  15
]
$
```

Functions

map

```
$ echo '[1,2,3,4,5]' | jq 'map(.+10)'
[
  11,
  12,
  13,
  14,
  15
]
```

Functions

reduce

```
$ echo '[1,2,3,4,5]' | jq 'reduce .[] as $i (0 ; . + $i)'
15
$
```

Somewhat equivalent to:

```
$i = 0
for var in .[] {
    $i = $var + $i
}
```

More Functions

→ has("foo")

→ in({"a": "b"})

→ echo '[1,2,3]' | jq 'map(.+2)' # [3,4,5]

→ select(func) - like grep where func is true

→ ... and a whole lot more

Detailed Examples With Real Data

Sample JSON Body (sample.json)


```
[{"name":"Aang","sex":"M","born":-12,"died":153,"bending":["Air","Water","Earth","Fire","Energy"],"identity":{"nationality":"Southern Air Temple",
"ethnicity":"Air Nomad"},"spouse":"Katara","children":[{"sex":"M","name":"Bumi"},{"sex":"F","name":"Kya"},{"sex":"M","name":"Tenzin"}]},{"name":"Katara","sex":
"F","born":85,"died":null,"bending":["Water","Blood"],"identity":{"nationality":"Southern Water Tribe","ethnicity":"Water Tribe"},"spouse":"Aang","children":[{"
sex":"M","name":"Bumi"},{"sex":"F","name":"Kya"},{"sex":"M","name":"Tenzin"}]},{"name":"Sokka","sex":"M","born":84,"died":164,"bending":[],"identity":
{"nationality":"Southern Water Tribe","ethnicity":"Water Tribe"},"spouse":null,"children":[]},{"name":"Toph Beifong","sex":"F","born":88,"died":null,"bending":
["Earth","Metal"],"identity":{"nationality":"Gaoling, Earth Kingdom","ethnicity":"Earth Kingdom"},"spouse":null,"children":[{"sex":"F","name":
"Lin Beifong"},{"sex":"F","name":"Suyin Beifong"}]},{"name":"Iroh","sex":"M","born":null,"died":null,"bending":["Fire","Energy"],"identity":{"nationality":
"Fire Nation Capital, Fire Nation","ethnicity":"Fire Nation"},"spouse":null,"children":[{"sex":"M","name":"Lu Ten"}]},{"name":"Zuko","sex":"M","born":83,
"died":null,"bending":["Fire","Energy"],"identity":{"nationality":"Fire Nation Capital, Fire Nation","ethnicity":"Fire Nation"},"spouse":null,"children":
[{"sex":"F","name":"Izumi"}]},{"name":"Kya","sex":"F","born":null,"died":null,"bending":["Water"],"identity":{"nationality":"Southern Water Tribe","ethnicity":
"Water Tribe, Air Nomad"},"spouse":null,"children":[]},{"name":"Bumi","sex":"M","born":null,"died":null,"bending":["Air"],"identity":{"nationality":
"United Republic","ethnicity":"Water Tribe, Air Nomad"},"spouse":null,"children":[]},{"name":"Tenzin","sex":"M","born":null,"died":null,"bending":["Air"],
"identity":{"nationality":"Republic City, United Republic","ethnicity":"Water Tribe, Air Nomad"},"spouse":null,"children":[{"sex":"F","name":"Jinora"},{"sex":
"F","name":"Ikki"},{"sex":"M","name":"Meelo"},{"sex":"M","name":"Rohan"}]},{"name":"Lin Beifong","sex":"F","born":120,"died":null,"bending":["Earth","Metal"],
"identity":{"nationality":"Republic City, United Republic","ethnicity":"Earth Kingdom"},"spouse":null,"children":[]},{"name":"Suyin Beifong","sex":"F","born":
126,"died":null,"bending":["Earth","Metal"],"identity":{"nationality":"Republic City, United Republic","ethnicity":"Earth Kingdom"},"spouse":null,"children":
[{"sex":"M","name":"Bataar Jr."},{"sex":"F","name":"Opal"},{"sex":"M","name":"Wei"},{"sex":"M","name":"Wing"},{"sex":"M","name":"Huan"}]}]
```


http://aijaz.net/2016/10/25/jq/

http://feelgrafix.com/data_images/out/28/973590-avatar-the-last-airbender.jpg

Example data of characters from the 'Avatar' series.

```
# Pretty print the jq
$ cat sample.json | jq '.'
...
      {
        "sex": "M",
        "name": "Huan"
      }
    ]
  }
]
$

# The ']' on the last line tells me this is an array
# The '}' on the 2nd-last line tells me this is an array of objects

# From now on, for space reasons,
# 'cat sample.json ' will be abbreviated to 'cj'
```

```
# Just show me the first object
$ cj | jq '.[0]'
```

```
# Show me an object with the name and
# sex of the first object
$ cj | jq '.[0]|{name, sex}'
{
  "name": "Aang",
  "sex": "M"
}
$
```

```
# Show me all the keys of each object
$ cj | jq '.[]|keys'
...
[
  "bending",
  "born",
  "children",
  "died",
  "identity",
  "name",
  "sex",
  "spouse"
]
$
```

```
# Show me all the number of keys of each object
$ cj | jq '.[]|length' # same as .[]|keys|length
8
8
8
8
8
8
8
8
8
8
8
8
$
```

```
# The name of each character
$ cj | jq '.[].name'
"Aang"
"Katara"
"Sokka"
"Toph Beifong"
"Iroh"
"Zuko"
"Kya"
"Bumi"
"Tenzin"
"Lin Beifong"
```

```
# The number of records
$ cj | jq '[.[]]|length'
11
$
```

```
# All female characters
$ cj | jq '.[]|select(.sex == "F")'
```

```
# The names of all female characters
# Take the resulting objects, put thmm in an array
# Then iterate over that array, printing the name
$ cj | jq '[.[]|select(.sex == "F")]|.[].name'
"Katara"
"Toph Beifong"
"Kya"
"Lin Beifong"
"Suyin Beifong"
$
```

```
# The name of all members of the Fire Nation
# You can drill down objects with multiple .s
# e.g.: .identity.ethnicity
$ cj | jq '[.[]|select(.identity.ethnicity == "Fire Nation")]|.[].name'
"Iroh"
"Zuko"
```

```
# The name and bending  list of all characters who can bend Fire
# Similar to previous example: iterate, select, construct a list, iterate on that list,
# and extract the name and bending fields
$ cj | jq -c  '[.[]|select(.bending[]|select(contains("Fire")))]|.[]|{name, bending}'
{"name":"Aang","bending":["Air","Water","Earth","Fire","Energy"]}
{"name":"Iroh","bending":["Fire","Energy"]}
{"name":"Zuko","bending":["Fire","Energy"]}
$
```

```
# Similar to previous, but make the name be the key of the object
$ cj | jq -c  '[.[]|select(.bending[]|select(contains("Fire")))]|.[]|{(.name): .bending}'
{"Aang":["Air","Water","Earth","Fire","Energy"]}
{"Iroh":["Fire","Energy"]}
{"Zuko":["Fire","Energy"]}
```

```
# Names and firstborns of each character, as hashes in an array
$ cj | jq '[.[]|{name, firstBorn: .children[0].name}]'
```

```
# How many children does each person have?
$ cj | jq '[.[]|.name, (.children|length)]'
[
  {
    "Aang": 3
  },
  {
    "Katara": 3
  },
 ...
  {
    "Suyin Beifong": 5
  }
]
$
```

```
# List of each person, along with the number of their children
# Essentially, merging the array of objects from the previous example into 1 big object
$ cj | jq '[.[]|[.name, (.children|length)]] | reduce .[] as $i ({}; . + { ($i[0]): $i[1]} )'
{
  "Aang": 3,
  "Katara": 3,
  "Sokka": 0,
  "Toph Beifong": 2,
  "Iroh": 1,
  "Zuko": 1,
  "Kya": 0,
  "Bumi": 0,
  "Tenzin": 4,
  "Lin Beifong": 0,
  "Suyin Beifong": 5
}
$
```

```
# Take the object matrix from two slides ago, and transpose it
$ cj | jq '[.[]|[.name, (.children|length)]] | transpose'
[
  [
    "Aang",
    "Katara",
    "Sokka",
    "Toph Beifong",
...
    "Suyin Beifong"
  ],
  [
    3,
    3,
    0,
    2,
...
    5
  ]
]
$
```

Three Most Useful Filters

```
'.'           # Pretty print
'.[]|{a, b}' # Extract just the a and b keys from each object
'.[]|keys'    # Display the keys of each object in the array
```

Honorable Mention

```
'[.[]|select(C)]'  # Return an array of only those objects where condition C is true
```

Resources

→ jq: https://stedolan.github.io/jq/

→ jq manual: https://stedolan.github.io/jq/manual/

→ Sample JSON file: http://aijaz.net/2016/10/25/jq/

Thanks!