JQ parser

Command-line JSON processor

Introduction

- jq is a lightweight and flexible command-line JSON processor.
- jq is like `sed` for JSON data - you can use it to slice and filter and map and transform structured data with the same ease that `sed`, `awk`, `grep` and friends let you play with text.

Sed

Sed is a advance text editor for filtering and transforming text input.

Features of Sed

- Select text
- Subsitute text
- Add lines to text
- Delete lines from text
- Modify an original File

Sed Application

```
cat samples/zen_of_python.txt
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Sed Application

```
sed -n '3,8p' samples/zen_of_python.txt
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
```

Sed Application

```
sed -n '3,8p' samples/zen_of_python.txt | sed 's/better/fancy/'
```

```
Beautiful is fancy than ugly.
Explicit is fancy than implicit.
Simple is fancy than complex.
Complex is fancy than complicated..
Flat is fancy than nested.
Sparse is fancy than dense.
```

Getting Started

```
sudo apt install -y jq
```

```
jq '.' example.json
```

Selection

```
jq '.status' example.json
jq '.data' example.json
jq '.data[0]' example.json
jq '.data[0].name' example.json
```

Array Indexing

Indexing in `jq` is similar to like `python`

```
jq '.data[2:4]' example.json
```

```
num_list = [1,2,3,4,5]
num_list[2:4]
```

Array Selecting

Array Value Iterator

.data[]

```
{
  "name": "John",
  "age": 25
}
{
  "name": "James",
  "age": 29
}
{
  "name": "Mike",
  "age": 32
}
{
  "name": "Milan",
  "age": 22
}
{
  "name": "Jackson",
  "age": 26
}
```

Index

.data

```
[
  {
    "name": "John",
    "age": 25
  },
  {
    "name": "James",
    "age": 29
  },
  {
    "name": "Mike",
    "age": 32
  },
  {
    "name": "Milan",
    "age": 22
  },
  {
    "name": "Jackson",
    "age": 26
  }
]
```

Selecting Multiple Index

```
jq '.data[] | .name, .age' example.json
```

Construction

1. Array Construction: `[]`

2. Object Construction: `{}`

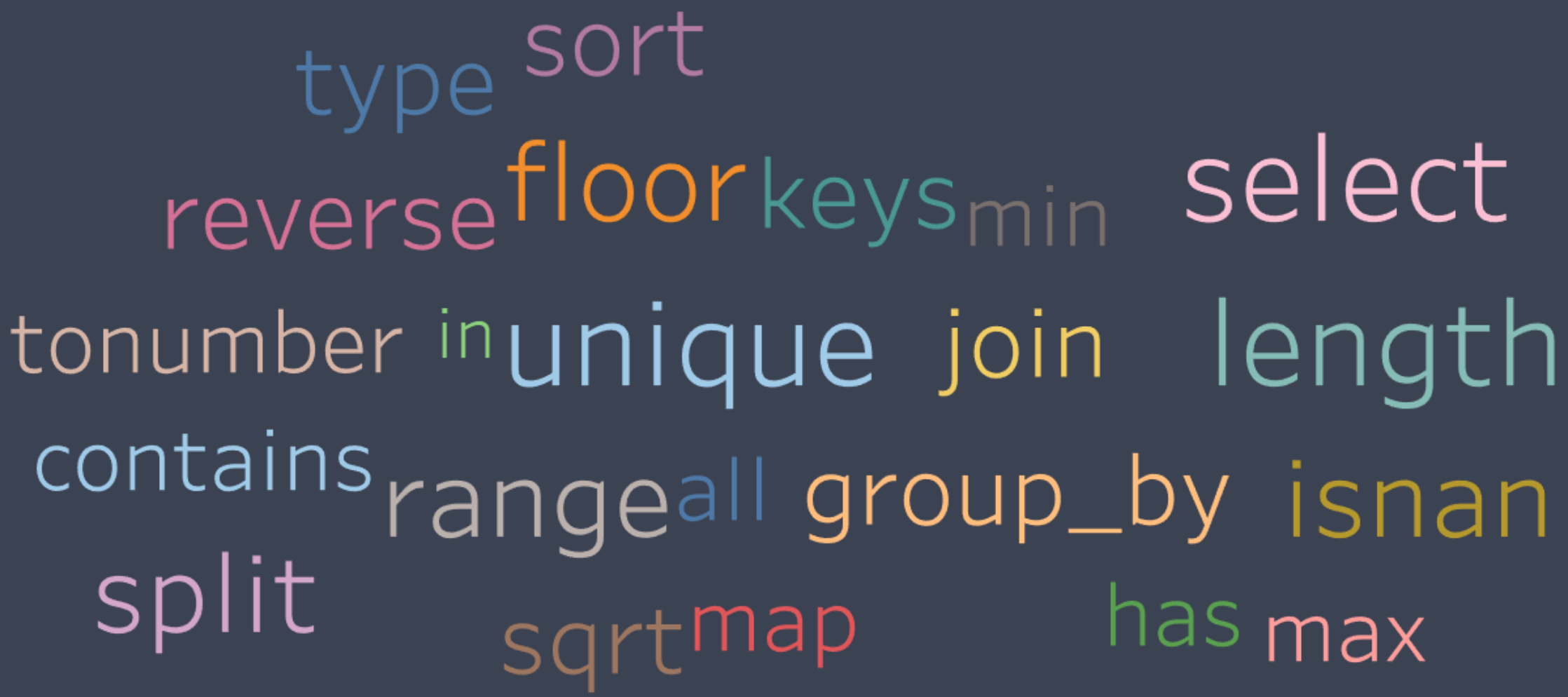
Array & Object constructor

```
jq '[.data[] | { name: .title, cost: .price}]' books.json
```


```
[
  {
    "name": "Lord Of The Flies",
    "cost": 900
  },
  {
    "name": "Paradise Lost",
    "cost": 1800
  },
  {
    "name": "The War Of The Worlds",
    "cost": 700
  },
  {
    "name": "Peter Pan",
    "cost": 800
  },
```

JQ built-in functions

Some useful functions



JQ built-in functions

```
jq '.data | length' books.json
```

```
jq '[.data[].price] | max, min' books.json
```

```
echo [9,3,2,6] | jq 'sort'
```

```
 jq '.data[0] | (.price | type), (.title | type)' books.json
```

Filtering

```
jq '.data[] | select(.price < 500)' books.json
jq '.data[] | select((.tags | length) > 2)' books.json
```

Advance I

Get authors and their book count

```
jq '.data | group_by(.author) | .[] | {author: .[0].author, books: . | length }' books.json
```

Advance II

Handle a property value with different datatype

```
[
    {
        "author": "Nnedi Okorafor",
        "books": "Lagoon"
    },
    {
        "author": "Natasha Farrant",
        "books": ["The Children Of Castle Rock","Voyage Of The Sparrowhawk"]
    }
]
```

```
jq '.[].books as $books | if $ books | type == "string" then [$books] else $books end' author.json
```

Advance III

Referencing value from another property

```
{
    "books": [
        {
            "title": "Lord Of The Flies",
            "authorId": "101"
        }
    ],
    "author": {
        "101": "William Golding"
    }
}
```

```
jq '.author as $author | .books[] | {title, author: $author[.authorId]}' store.json
```

References

- https://stedolan.github.io/jq/manual/
- https://lindevs.com/install-jq-on-ubuntu/
- http://www.compciv.org/recipes/cli/jq-for-parsing-json/
- https://earthly.dev/blog/jq-select/

Thank you

Any Questions???