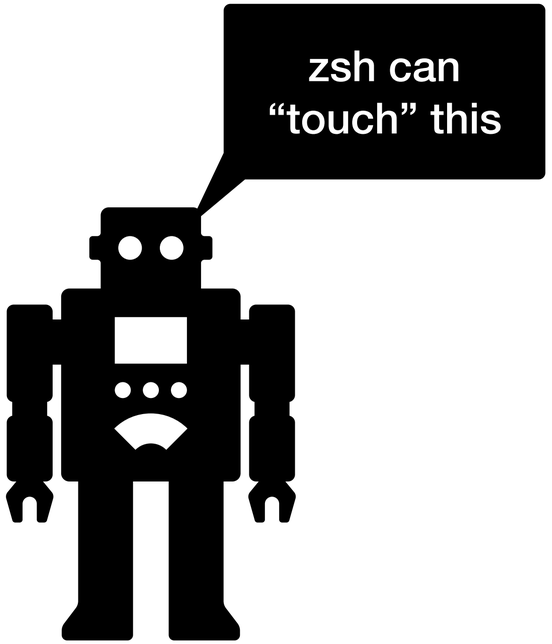
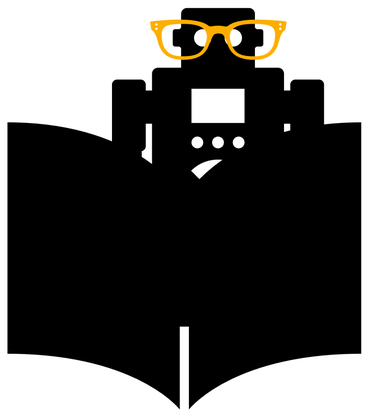
Stop, yaml time



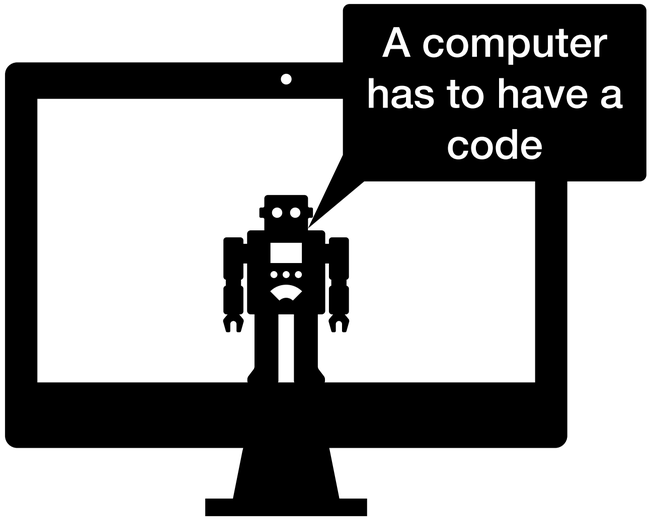
Michael Colaresi

Back to Basics: Text



- characters:
  - [a..zA-Z0-9]*
    - And more (punctuation, etc)
- strings:
  - Ordered list/seq of characters
    - “Hello World”… is [“H”, “e”, “l”, “l”, “o”, “ “, “W”, “o”, “r”, “l”, “d”, “\n”]

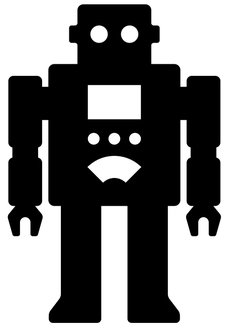
From text to code



- commands
  - eg — cd, mkdir, vim, etc
- arguments
  - cd ~/cd ; mkdir ../mkdir ; vim ~/vim; etc
- How do we tell the command apart from the argument?
  - Syntax
    - Rules for parsing text to extract meaning

From text to code

$ cd ~/cd

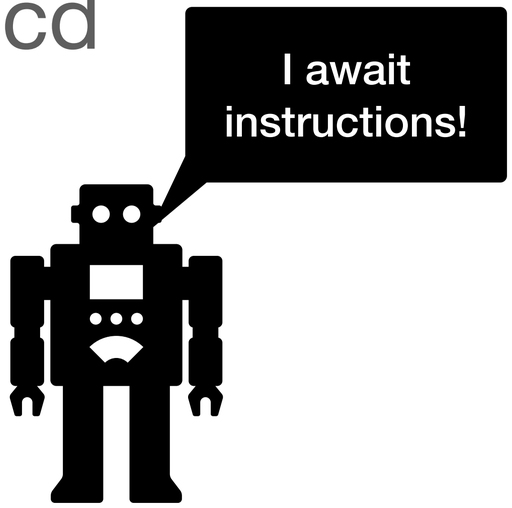


- commands
  - eg — cd, mkdir, vim, etc
- arguments
  - cd ~/cd ; mkdir ../mkdir ; vim ~/vim; etc
- How do we tell the command apart from the argument?
  - Syntax
    - Rules for parsing text to extract meaning

From text to code

Beginning of prompt

$ cd ~/cd

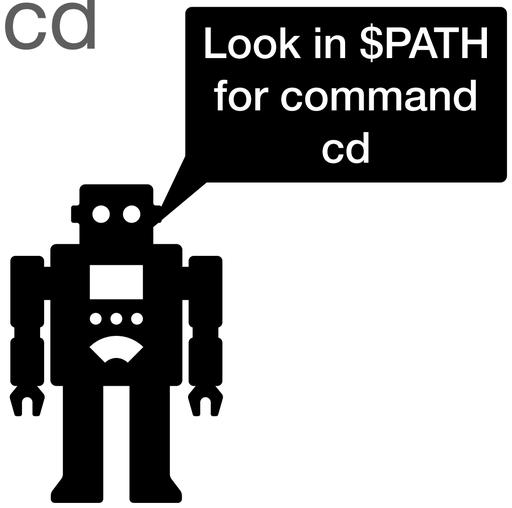


- commands
  - eg — cd, mkdir, vim, etc
- arguments
  - cd ~/cd ; mkdir ../mkdir ; vim ~/vim; etc
- How do we tell the command apart from the argument?
  - Syntax
    - Rules for parsing text to extract meaning

From text to code

Command next

$ cd ~/cd

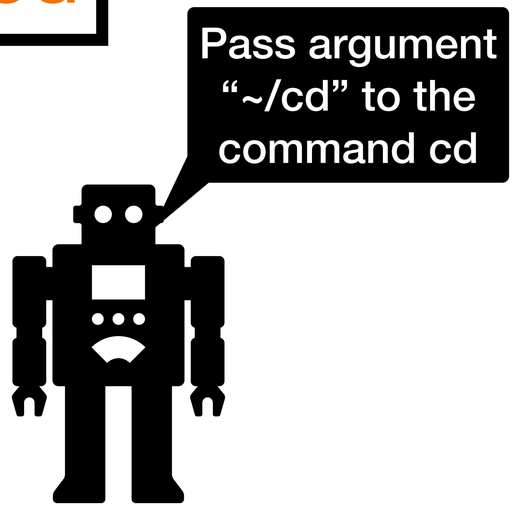


- commands
  - eg — cd, mkdir, vim, etc
- arguments
  - cd ~/cd ; mkdir ../mkdir ; vim ~/vim; etc
- How do we tell the command apart from the argument?
  - Syntax
    - Rules for parsing text to extract meaning

From text to code

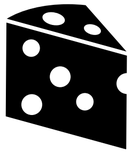
Then argument(s)

$ cd ~/cd



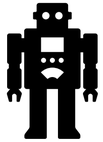
- commands
  - eg — cd, mkdir, vim, etc
- arguments
  - cd ~/cd ; mkdir ../mkdir ; vim ~/vim; etc
- How do we tell the command apart from the argument?
  - Syntax
    - Rules for parsing text to extract meaning

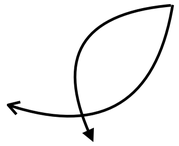
White space as “mark up”

- Zsh/Bash (and you!) use white space, the beginning of lines, ends of lines, blank lines, etc as “mark up”
  - The white space defines the context of the string
    - The gorgonzola 
    - The gorgon zola 
  - “” vs ‘’ in zsh/bash for variables
    - “$var” vs. ‘$var’

Yaml

Yaml Ain’t Markup Language (…although it started as one)

- Communicate “context”/“meaning” of terms with syntax
- A set of simple rules that is:
  - Readable by humans 
  - Parseable by computers 
  - Translates to other formats easily (json, python objects, etc) 
  - Relatively flexible (non-hierarchical structures can be represented)



Yaml

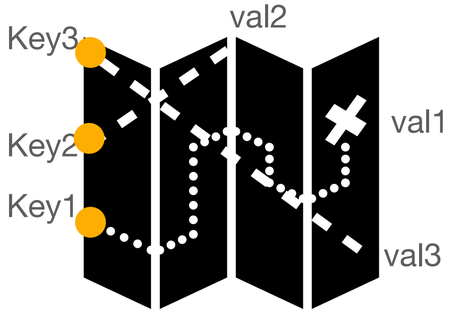
Syntax

- lists/seq

```
- item1
- item2
- item3
```

- maps/dictionaries

```
key1: val1
key2: val2
key3: val3
```



- scalars

```
1
My name
“trust”
```

Yaml

Combinations

```
key1:
    - item1
    - item2
    - item3
```

Yaml

Combinations

```
key1:
    - item1
    - item2
    - item3
```

Key is a scalar!

Yaml

Combinations

Setting up a map/dict

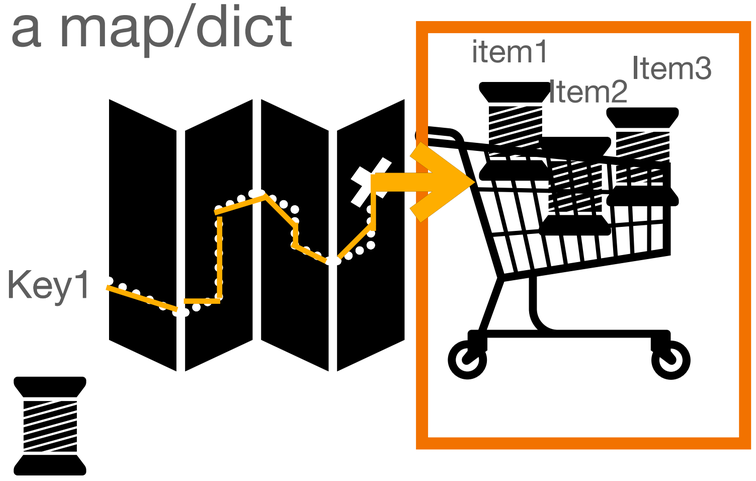
```
key1:
    - item1
    - item2
    - item3
```

Yaml

Combinations

Setting up a map/dict

```
key1:
  - item1
  - item2
  - item3
```



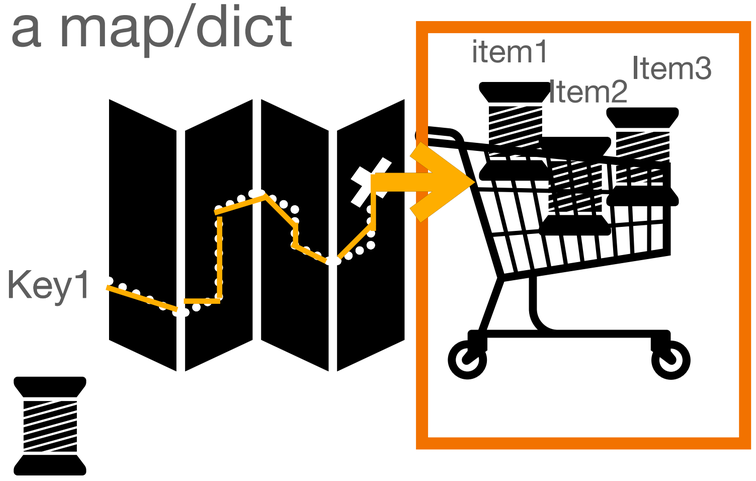
Value is a list/seq holding scalars, “item1”, “item2”, “item3”

Yaml

Combinations

Setting up a map/dict

```
key1:
  - item1
  - item2
  - item3
```



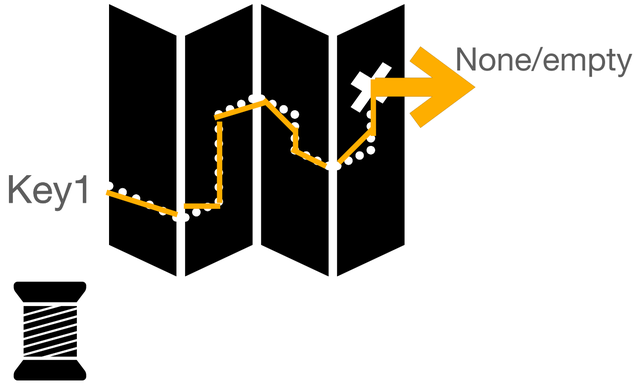
Indentation is important,
it says the list/seq is inside the
map/dictionary

Yaml

If we have this…

Setting up a map/dict

```
key1:
- item1
- item2
- item3
```



Indentation is important,
it says the list/seq is inside the map/dictionary

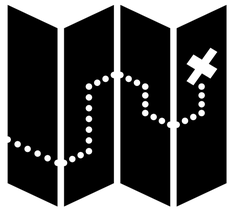
Another example

```
recordID: fl21_2002_5
flights:
  - flightID:  fl21_2002_5_1
    plane: FL456
    date: 9/24/2002
    origin: “Johnstown, NC, USA”
    destination: London, UK“
  - flightID:  fl21_2002_5_2
    plane: FL456
    date: 9/26/2002
    origin: “London, UK”
    destination: Warsaw, Poland“
```

Another example

recordID: fl21_2002_5
flights:
- flightID: fl21_2002_5_1
  plane: FL456
  date: 9/24/2002
  origin: “Johnstown, NC, USA”
  destination: London, UK“
- flightID: fl21_2002_5_2
  plane: FL456
  date: 9/26/2002
  origin: “London, UK”
  destination: Warsaw, Poland“

Map! With key “recordID” and a scalar value

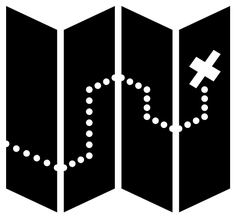


Another example

recordID: fl21_2002_5
flights:
- flightID: fl21_2002_5_1
  plane: FL456
  date: 9/24/2002
  origin: “Johnstown, NC, USA”
  destination: London, UK“
- flightID: fl21_2002_5_2
  plane: FL456
  date: 9/26/2002
  origin: “London, UK”
  destination: Warsaw, Poland“

Map! With key “flights”

Value is …



Another example

recordID: fl21_2002_5
flights:
- flightID: fl21_2002_5_1
  plane: FL456
  date: 9/24/2002
  origin: “Johnstown, NC, USA”
  destination: London, UK“
- flightID: fl21_2002_5_2
  plane: FL456
  date: 9/26/2002
  origin: “London, UK”
  destination: Warsaw, Poland“

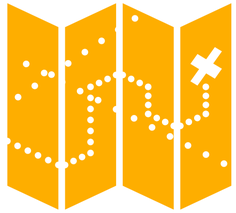
Map! With key “flights”

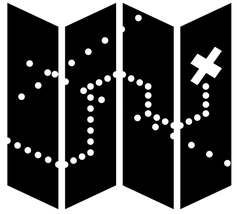
Value is a seq/list

Another example

recordID: fl21_2002_5
flights:
- flightID: fl21_2002_5_1
  plane: FL456
  date: 9/24/2002
  origin: “Johnstown, NC, USA”
  destination: London, UK“
- flightID: fl21_2002_5_2
  plane: FL456
  date: 9/26/2002
  origin: “London, UK”
  destination: Warsaw, Poland“

The items in that seq/list are maps/dicts

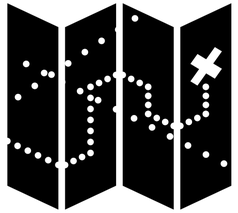


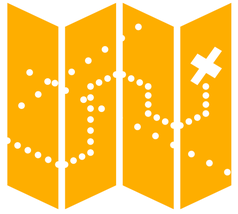


Another example

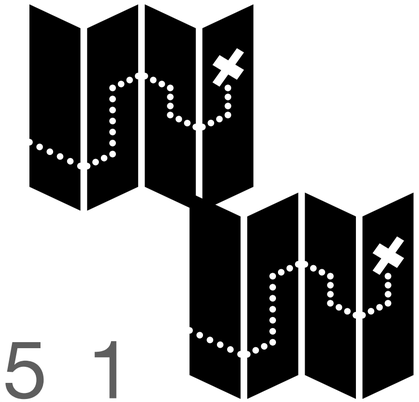
recordID: fl21_2002_5
flights:
- flightID: fl21_2002_5_1
  plane: FL456
  date: 9/24/2002
  origin: “Johnstown, NC, USA”
  destination: London, UK“
- flightID: fl21_2002_5_2
  plane: FL456
  date: 9/26/2002
  origin: “London, UK”
  destination: Warsaw, Poland“

The items in that seq/list are maps/dicts





So we have:



recordID: fl21_2002_5
flights:
- flightID: fl21_2002_5_1
  plane: FL456
  date: 9/24/2002
  origin: “Johnstown, NC, USA”
  destination: London, UK“
- flightID: fl21_2002_5_2
  plane: FL456
  date: 9/26/2002
  origin: “London, UK”
  destination: Warsaw, Poland“

So we have:

recordID: fl21_2002_5
flights:
- flightID: fl21_2002_5_1
  plane: FL456
  date: 9/24/2002
  origin: “Johnstown, NC, USA”
  destination: London, UK“
- flightID: fl21_2002_5_2
  plane: FL456
  date: 9/26/2002
  origin: “London, UK”
  destination: Warsaw, Poland“

Yaml for Mammals

Meta data for code and data coding

- Meta data for code
  - You often have values as part of your project that you need to keep track of
    - Number of simulation runs
    - Prior means, std deviations
    - List of variances
    - Locations of input data and urls
    - WRITE THESE IN yaml files!

Yaml for Mammals

Meta data for code and data coding

- Data coding
  - When you code data
    - Often from unstructured formats
    - You can store it in yaml format
    - flexible, readable, transferable, subset-able (chunk it into smaller pieces)
    - Open source!

Linting for yaml

- Proper yaml is not too hard
  - But linting can help
  - Linter written in python is yamllint
  - On Mac: brew install yamllint
    - If installed ALE will use it whenever a .yaml or .yml file is loaded in a buffer
  - Linux based install directions here: https://yamllint.readthedocs.io/en/stable/quickstart.html#installing-yamllint