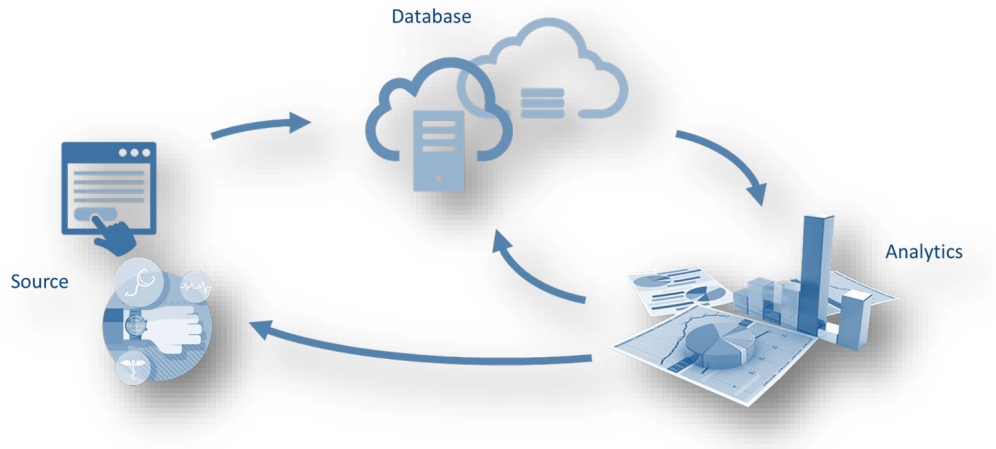
Data Exchange

Content

- Introduction
- Binary
- CSV
- XML
- JSON
- YAML

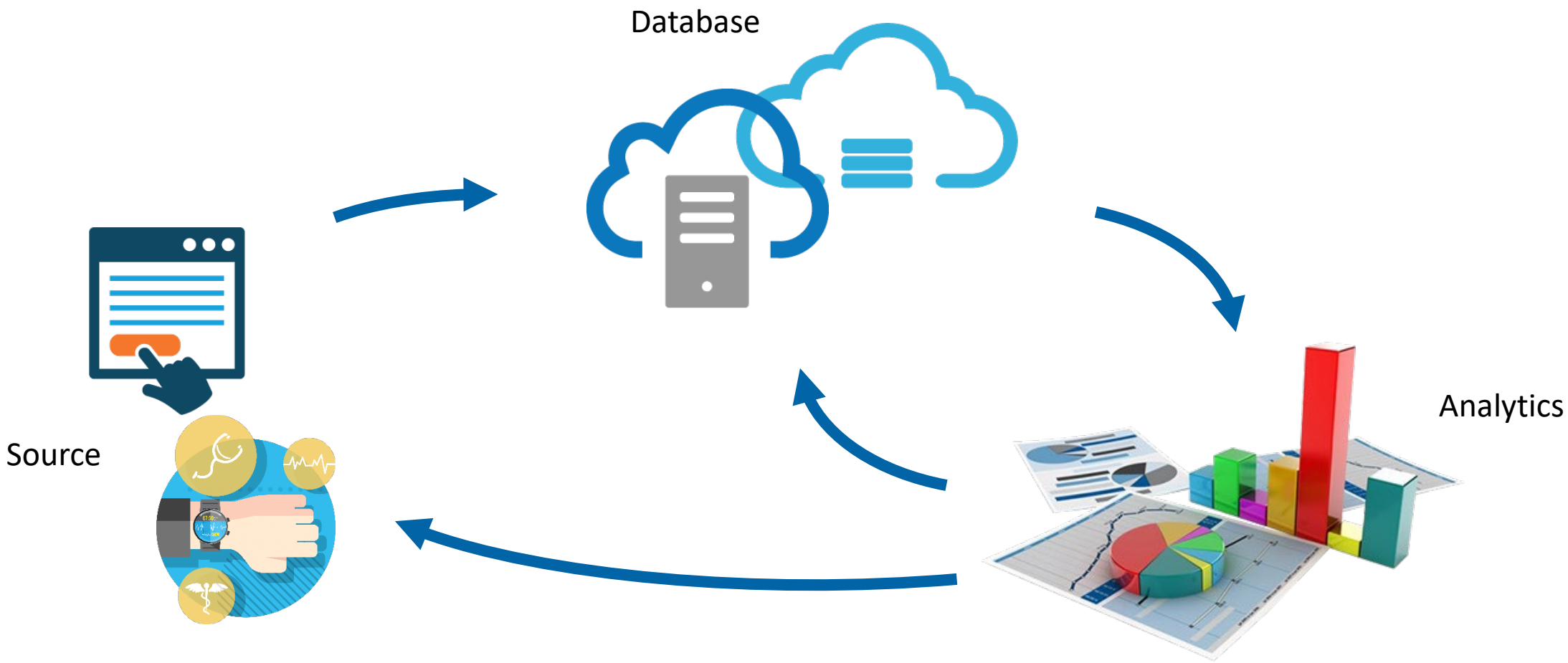
Content

- Introduction
- Binary
- CSV
- XML
- JSON
- YAML



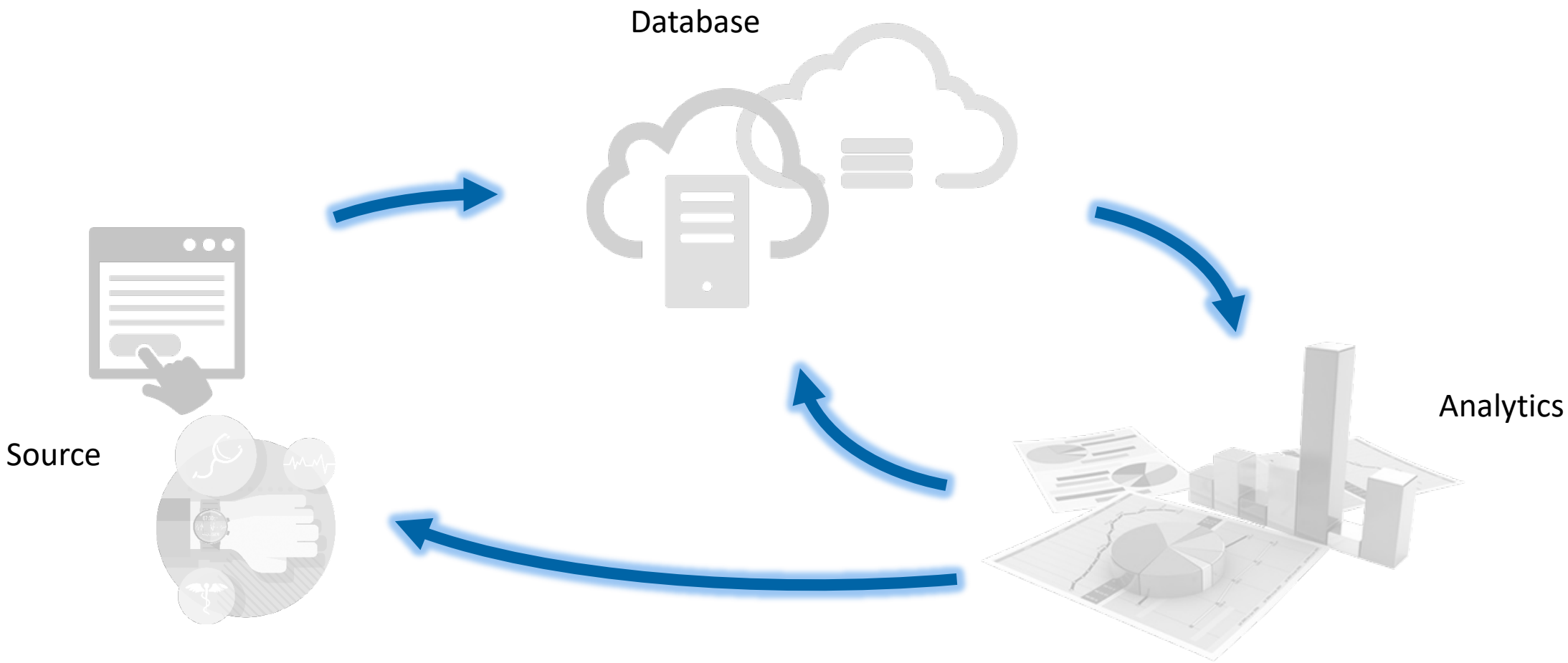
Introduction

Data Flow



Introduction

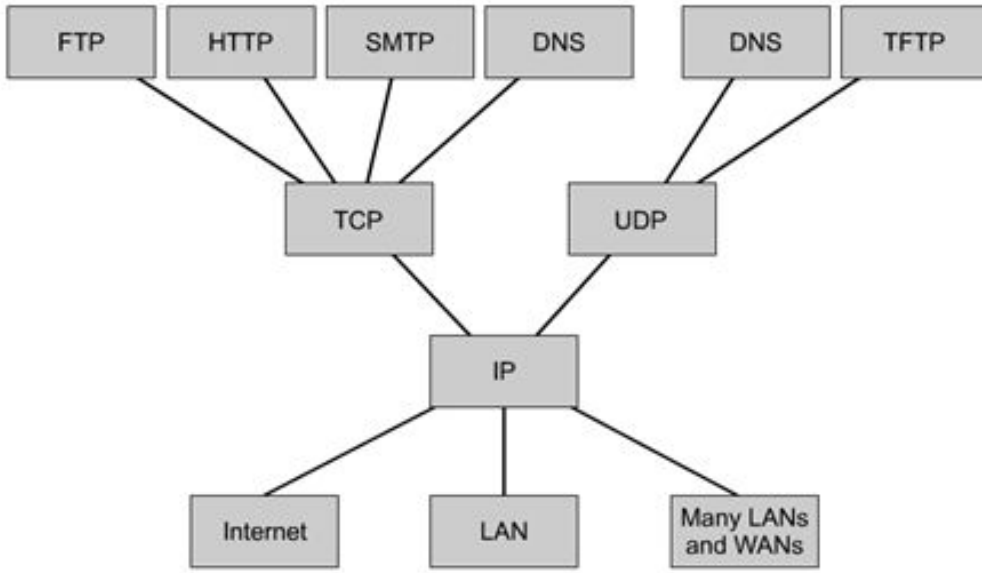
Data Flow



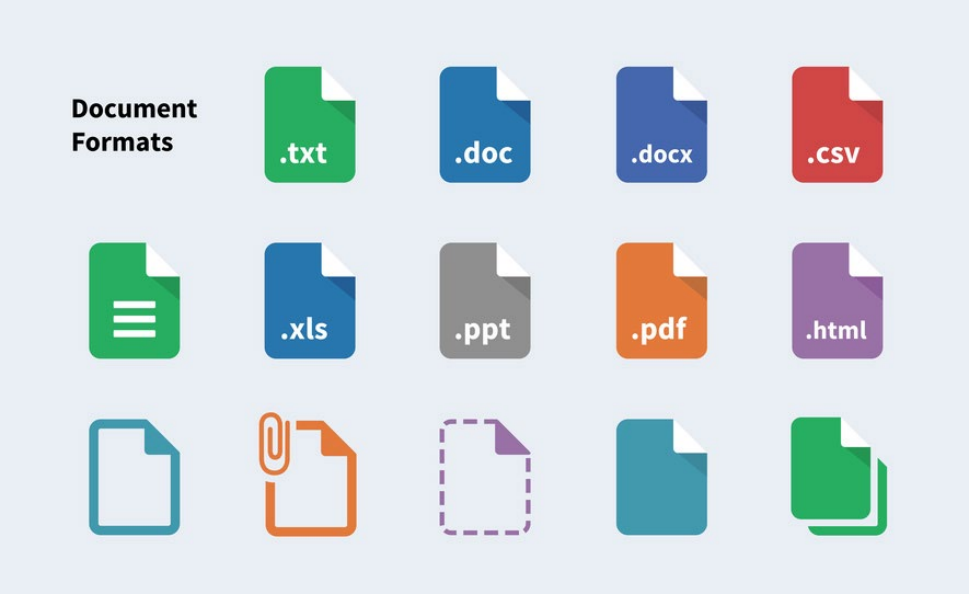
Introduction

Protocols vs Format

Protocols: rules to transfer data



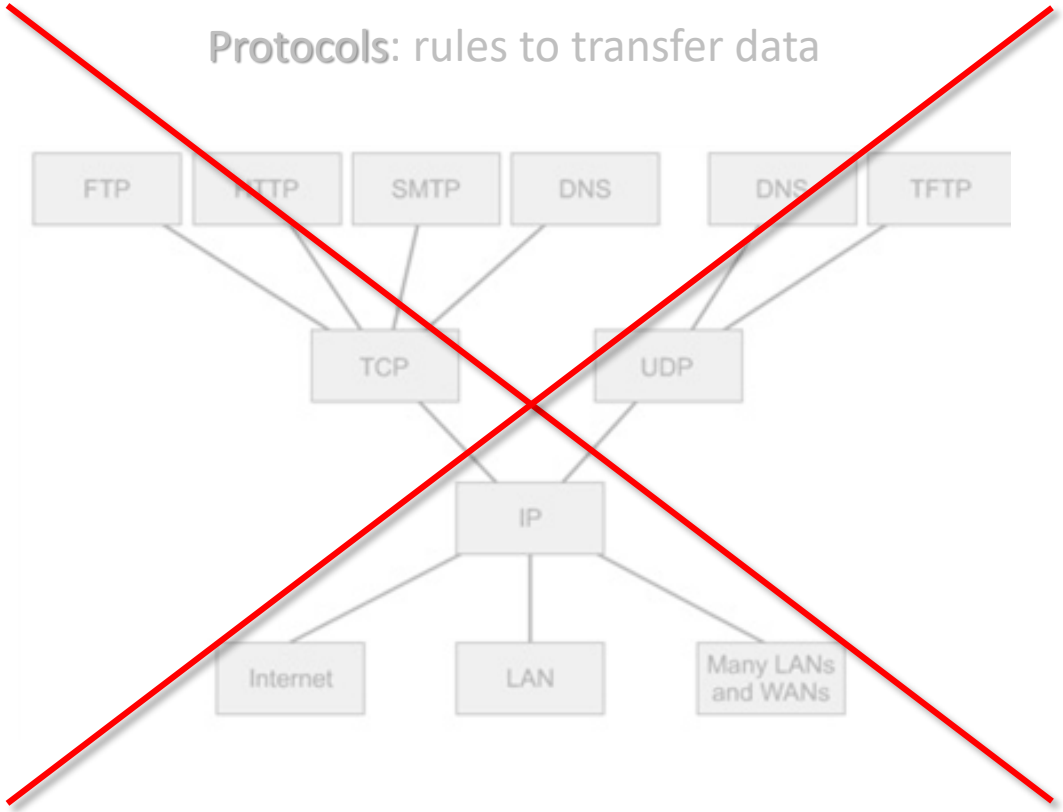
Format: rules to structure data



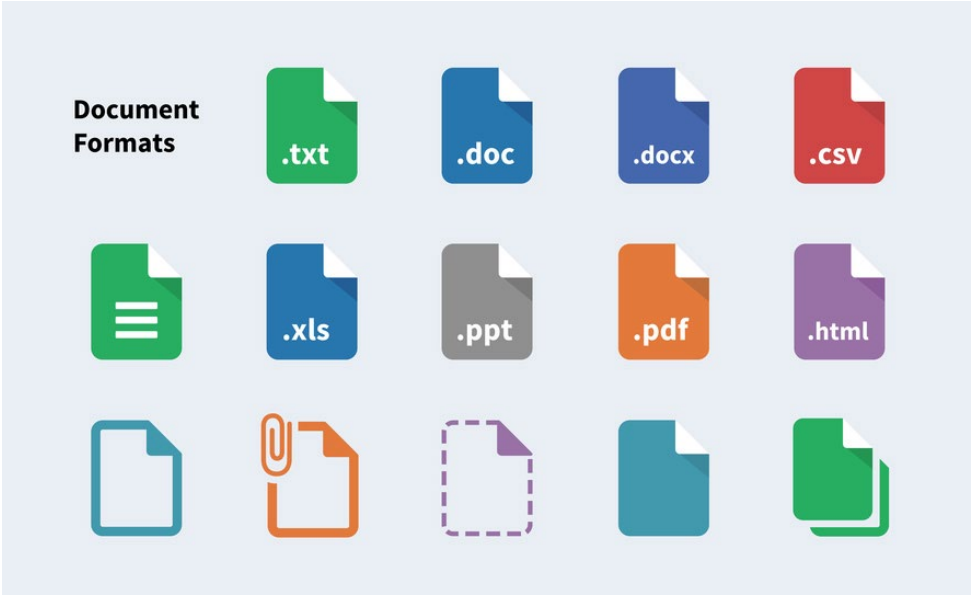
Introduction

Protocols vs Format

Protocols: rules to transfer data



Format: rules to structure data



Introduction

Data Format

```
series: 05
date: 01-01-2018
user: u101
```

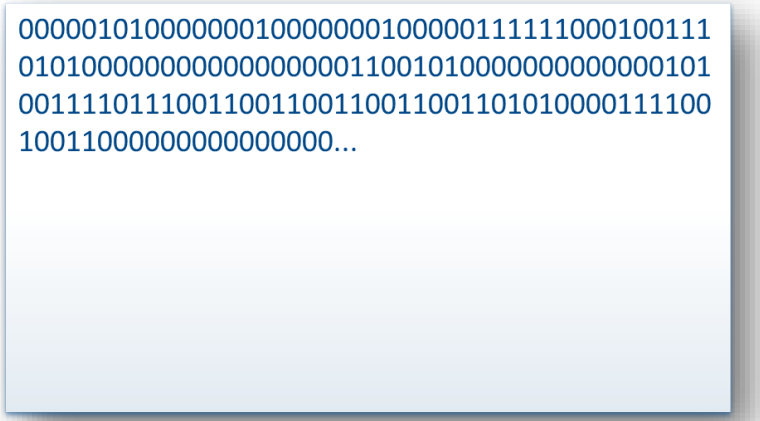
Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

Content

- Introduction
- Binary
- CSV
- XML
- JSON
- YAML



Binary

Bits and Bytes

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

data.bin

```
00000101000000010000000100000111111000100111
01010000000000000001100101000000000000101
00111101110011001100110011001101010000111100
10011000000000000000...
```

Binary

Key must be known

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

Position	# Bytes	Information
0	1	Series
1 – 4	4	Date
5 – 8	4	User
9 – 10	2	# data points (=n)
11 – (11+n·8)	n·8	Data

Binary

Key must be known

Position	# Bytes	Information
0	1	Series
1 – 4	4	Date
5 – 8	4	User
9 – 10	2	# data points (=n)
11 – (11+n·8)	n·8	Data

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

```
0000010100000001000000010000011111100010011101010000000000000000011001010000000000000010100111011100110011001100110011010100001111001001100000000000000000...
```

Binary

Key must be known

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

Position	# Bytes	Information
0	1	Series
1 – 4	4	Date
5 – 8	4	User
9 – 10	2	# data points (=n)
11 – (11+n·8)	n·8	Data

0000010100000001000000010000011111100010011101010000000000000000110010100000000000000101001111011100110011001100110011010100001111001001100000000000000000...

Binary

Key must be known

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

Position	# Bytes	Information
0	1	Series
1 – 4	4	Date
5 – 8	4	User
9 – 10	2	# data points (=n)
11 – (11+n·8)	n·8	Data

```
00000101000000010000000100000111111000100111
01010000000000000000110010100000000000000101
00111101110011001100110011001101010000111100
10011000000000000000...
```

Binary

Key must be known

Position	# Bytes	Information
0	1	Series
1 – 4	4	Date
5 – 8	**4**	**User**
9 – 10	2	# data points (=n)
11 – (11+n·8)	n·8	Data

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

```
00000101000000010000000100000111111000100111
01010000000000000000001100101000000000000000101
00111101110011001100110011001101010000111100
10011000000000000000...
```

Binary

Access

```
00000101000000010000000100000111111000100111
01010000000000000000110010100000000000000101
00111101110011001100110011001101010000111100
10011000000000000000...
```

Python

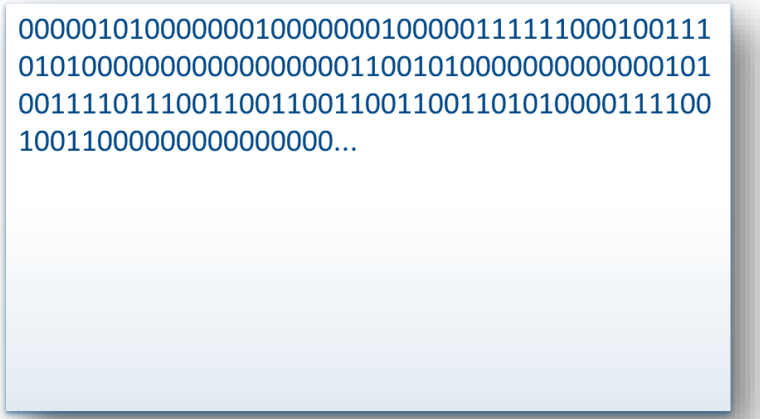
```
# open binary file
f = open("data.bin", "rb")

# read bytes
bytes = f.read(4)
do stuff with bytes

# close binary file
f.close()
```

Binary

Pros and Cons

Pros
- Fast read and write
- Small files

Cons
- Key must be known
- Read and write methods must be implemented



Content

- Introduction
- Binary
- CSV
- XML
- JSON
- YAML

```
0.1, 403
0.2, 417
0.3, 398
0.4, 378
0.5, 421
```

CSV

Comma Separated Values

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

05_20180101_u101.csv

```
0.1, 403
0.2, 417
0.3, 398
0.4, 378
0.5, 421
```

CSV

Table-like

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

05_20180101_u101.csv

```
0.1,403
0.2,417
0.3,398
0.4,378
0.5,421
```

New line defines «rows»

Delimiter defines «columns»

Delimiter must not necessarily be the comma

CSV

Access

```
0.1, 403
0.2, 417
0.3, 398
0.4, 378
0.5, 421
```

Python

```python
# open csv file
f = open("05_20180101_u101.csv")

# read rows
reader = f.reader(f,delimiter=',')
for row in reader:
    do stuff with the row

# close csv file
f.close()
```

CSV

Pros and Cons

Pros
- Very simple
- Libraries exist

Cons
- Data type of «columns» must be known
- Delimiter and quote character must be known
- Missing a default way to include meta data

- It’s text – beware the encoding (ASCII, UTF-8, ...)

```
0.1, 403
0.2, 417
0.3, 398
0.4, 378
0.5, 421
```

Content

- Introduction
- Binary
- CSV
- XML
- JSON
- YAML

```
<meta>
   <series>05</series>
   <date>01-01-2018</date>
   <user>u101</user>
</meta>
<data>
   <delta>0.1</delta><value>403</value>
   <delta>0.2</delta><value>417</value>
   <delta>0.3</delta><value>398</value>
   <delta>0.4</delta><value>378</value>
   <delta>0.5</delta><value>421</value>
</data>
```

XML

e**X**tended **M**arkup **L**anguage

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

data.xml

```
<meta>
  <series>05</series>
  <date>01-01-2018</date>
  <user>u101</user>
</meta>
<data>
  <delta>0.1</delta><value>403</value>
  <delta>0.2</delta><value>417</value>
  <delta>0.3</delta><value>398</value>
  <delta>0.4</delta><value>378</value>
  <delta>0.5</delta><value>421</value>
</data>
```

XML

Elements and Tags

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

data.xml

```
<meta>
  <series>05</series>
  <date>01-01-2018</date>
  <user>u101</user>
</meta>
<data>
  <delta>0.1</delta><value>403</value>
  <delta>0.2</delta><value>417</value>
  <delta>0.3</delta><value>398</value>
  <delta>0.4</delta><value>378</value>
  <delta>0.5</delta><value>421</value>
</data>
```

XML

Elements and Tags

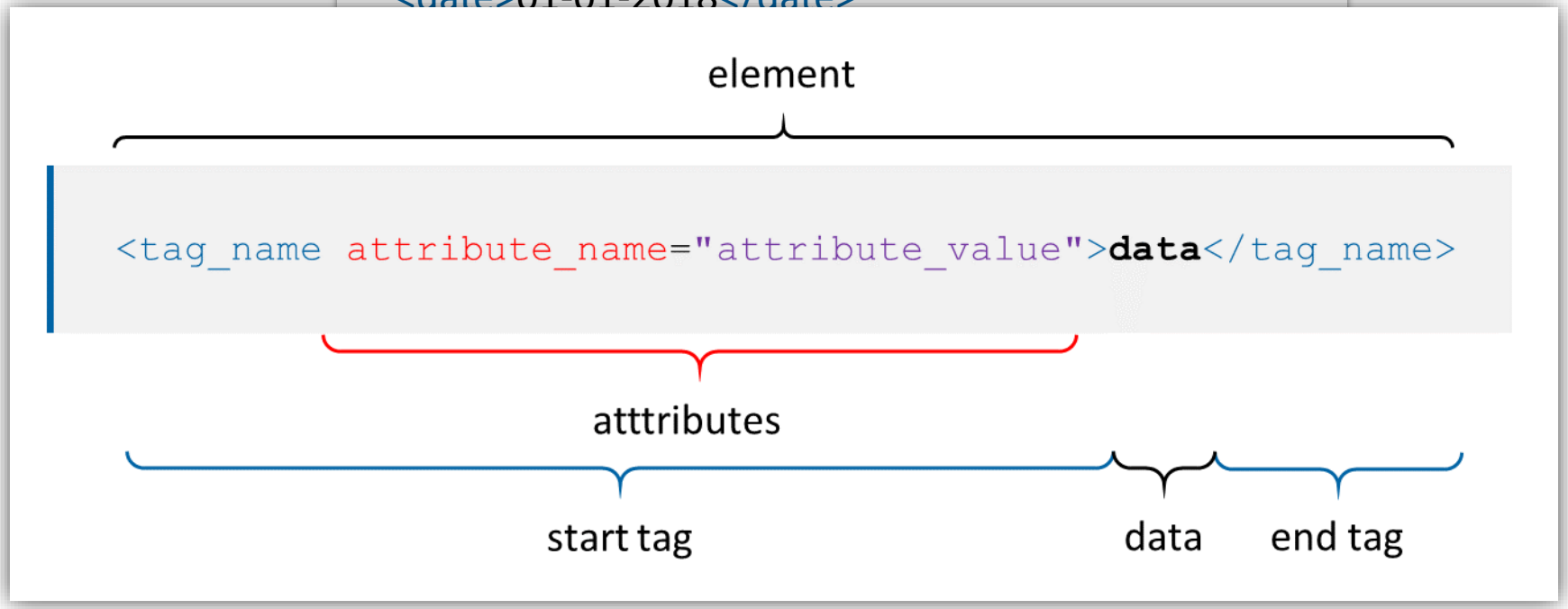
```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data.xml

```
<meta>
  <series>05</series>
  <date>01-01-2018</date>
```



XML

Access

```
<meta>
  <series>05</series>
  <date>01-01-2018</date>
  <user>u101</user>
</meta>
<data>
  <delta>0.1</delta><value>403</value>
  <delta>0.2</delta><value>417</value>
  <delta>0.3</delta><value>398</value>
  <delta>0.4</delta><value>378</value>
  <delta>0.5</delta><value>421</value>
</data>
```

Python

```python
# import
from xml.dom import minidom

# parse document
dom = minidom.parse('data.xml')

# get values
values = dom.getElementsByTagName('value')
```

XML

Access

Python

```
<meta>
  <series>05</series>
  <date>01-01-2018</date>
  <
</m
<da
  <
  <
  <
  <
  <
</d
```

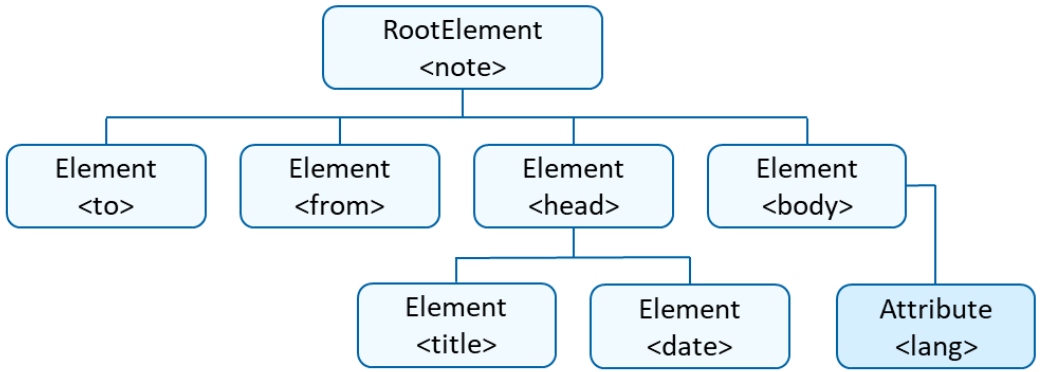
```
# import
from xml.dom import minidom
```

Document Object Model (DOM)

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <head>
    <title>Reminder</title>
    <date>14/02/2017</date>
  </head>
  <body lang='EN-US'>
    Theater this Sunday!
  </body>
</note>
```

Document

parsing



DOM

XML

Pros and Cons

Pros
- Human readable
- Libraries exist
- Default way of adding meta data

Cons
- Files can become very large
- Elements must be known
- Nesting can be very complex

- It’s text – beware the encoding (ASCII, UTF-8, ...)

```
<meta>
  <series>05</series>
  <date>01-01-2018</date>
  <user>u101</user>
</meta>
<data>
  <delta>0.1</delta><value>403</value>
  <delta>0.2</delta><value>417</value>
  <delta>0.3</delta><value>398</value>
  <delta>0.4</delta><value>378</value>
  <delta>0.5</delta><value>421</value>
</data>
```

Content

- Introduction
- Binary
- CSV
- XML
- JSON
- YAML

```
{
 "meta":{"series":5,
          "date":"01-01-2018",
          "user":"u101"},
 "data":[ { "delta":0.1, "value":403 },
          { "delta":0.2, "value":417 },
          { "delta":0.3, "value":398 },
          { "delta":0.4, "value":378 },
          { "delta":0.5, "value":421 } ]
}
```

JSON

JavaScript Object Notation

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

data.json

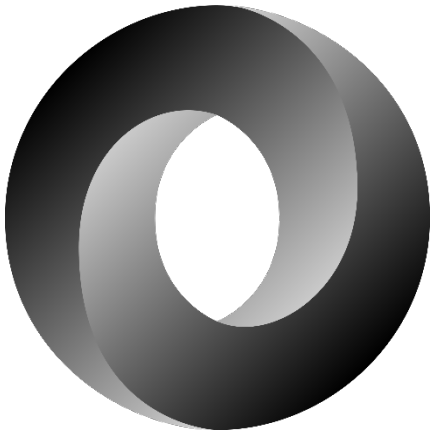
```
{
 "meta":{"series":5,
          "date":"01-01-2018",
          "user":"u101"},
 "data":[ { "delta":0.1, "value":403 },
          { "delta":0.2, "value":417 },
          { "delta":0.3, "value":398 },
          { "delta":0.4, "value":378 },
          { "delta":0.5, "value":421 } ]
}
```

JSON

Java**S**cript **O**bject **N**otation

JSON is a syntax for storing and exchanging data.
JSON is text, written with JavaScript object notation.

JSON uses JavaScript syntax, but the JSON format is text only.
Text can be read and used as a data format by any programming language.



Like XML Because

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages

Unlike XML Because

- JSON doesn't use end tag
- JSON is shorter
- JSON can use arrays

JSON

Name-Value Pairs

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

data.json

```
{
 "meta":{"series":5,
          "date":"01-01-2018",
          "user":"u101«},
 "data":[ { "delta":0.1, "value":403 },
          { "delta":0.2, "value":417 },
          { "delta":0.3, "value":398 },
          { "delta":0.4, "value":378 },
          { "delta":0.5, "value":421 } ]
}
```

JSON

Access

```
{
 "meta":{"series":5,
          "date":"01-01-2018",
          "user":"u101"},
 "data":[ { "delta":0.1, "value":403 },
          { "delta":0.2, "value":417 },
          { "delta":0.3, "value":398 },
          { "delta":0.4, "value":378 },
          { "delta":0.5, "value":421 } ]
}
```

Python

```python
# import
import json

# open file
f = open('data.json')

# parse document
dataseries = json.load(f)

# get value of first measurement
val = dataseries['data'][0]['value']

# close file
f.close()
```

JSON

Pros and Cons

Pros
- Human readable
- Libraries exist
- Default way of adding meta data

Cons
- Names must be known

- It’s text – beware the encoding (ASCII, UTF-8, ...)

```
{
 "meta":{"series":5,
          "date":"01-01-2018",
          "user":"u101"},
 "data":[ { "delta":0.1, "value":403 },
          { "delta":0.2, "value":417 },
          { "delta":0.3, "value":398 },
          { "delta":0.4, "value":378 },
          { "delta":0.5, "value":421 } ]
}
```

Content

- Introduction
- Binary
- CSV
- XML
- JSON
- YAML

```
---
meta:
  series: 5
  date: 01-01-2018
  user: u101
data:
- delta: 0.1
  value: 403
- delta: 0.2
  value: 417
- delta: 0.3
  value: 398
- delta: 0.4
  value: 378
- delta: 0.5
  value: 421
```

YAML

YAML Ain't Markup Language

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

data.yaml

```
---
meta:
  series: 5
  date: 01-01-2018
  user: u101
data:
- delta: 0.1
  value: 403
- delta: 0.2
  value: 417
- delta: 0.3
  value: 398
- delta: 0.4
  value: 378
- delta: 0.5
  value: 421
```

YAML

YAML Ain't Markup Language

YAML is a human-readable, data-serialization language.
It is commonly used for configuration files and in applications where data is being stored or transmitted.

YAML uses both Python-style indentation to indicate nesting, and a more compact format that uses [] for lists and {}

```
url: yaml.org
```

YAML knows a document seperator "---" which allows multiple documents in one file

YAML is a superset of JSON

- Every JSON file is a valid YAML file
- The same comparison between JSON and XML also holds for YAML and XML

YAML

Key-Value Pairs

```
series: 05
date: 01-01-2018
user: u101
```

Δt	value
0.1	403
0.2	417
0.3	398
0.4	378
0.5	421

meta

data

data.yaml

```yaml
---
meta:
  series: 5
  date: 01-01-2018
  user: u101
data:
- delta: 0.1
  value: 403
- delta: 0.2
  value: 417
- delta: 0.3
  value: 398
- delta: 0.4
  value: 378
- delta: 0.5
  value: 421
```

YAML

Access

```
---
meta:
  series: 5
  date: 01-01-2018
  user: u101
data:
- delta: 0.1
  value: 403
- delta: 0.2
  value: 417
- delta: 0.3
  value: 398
- delta: 0.4
  value: 378
- delta: 0.5
  value: 421
```

Python

```python
# import
import yaml

# open file
with open('data.yaml') as f:
  try:
    # parse document
    dataseries = yaml.load(f)
  except yaml.YAMLError as err:
    print(err)

# get value of first measurement
val = dataseries['data'][0]['value']
```

YAML

Pros and Cons

Pros
- Human readable
- Libraries exist
- Default way of adding meta data

Cons
- Keys must be known

- It’s text – beware the encoding (ASCII, UTF-8, ...)

```
---
meta:
  series: 5
  date: 01-01-2018
  user: u101
data:
- delta: 0.1
  value: 403
- delta: 0.2
  value: 417
- delta: 0.3
  value: 398
- delta: 0.4
  value: 378
- delta: 0.5
  value: 421
```