Pipeline & Workflow

"TBD."

Table of Contents


Overview

Moduler:

- ETL
- ELT
- FIletypes
- Workshop with Data Quality

Utbildningsmoment

- Dataplatfformar, bakgrund och syfte ✅
- Git och github i teamkontext
- Komponenter och teknologier i en data platform ✅
- ETL vs ELT ✅
- Utveckling av mjukvara mot databaser
- Använda Python mot relationsdatabaser och andra datakällor såsom csv, http xml/json
- Använda Python mot realtidsdataströmmar såsom message queues och/eller event streaming platforms
- Använda Python och för att rensa, validera och transformera data ✅
- Workflow processer ✅

02

ETL vs ELT


Project Setup
using uv

BONUS

Ignorera detta om du inte vill veta mer om hur *'activate'* fungerar!

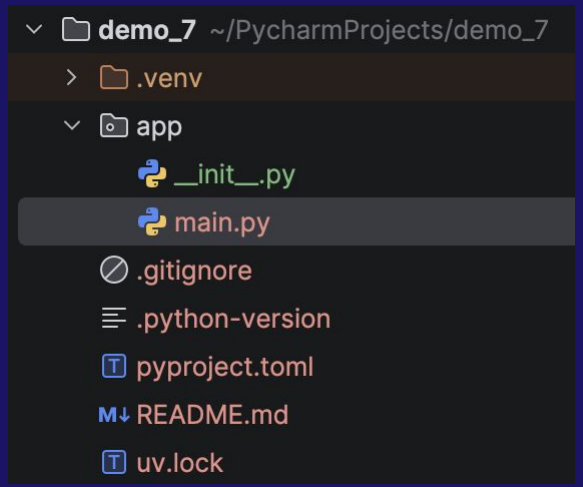
In a uv project, do NOT use this command:

```bash
source .venv/bin/activate
```

venv/bin/activate. If you use uv , **you actually should avoid to use source . venv/bin/activate** . uv makes it superfluous to activate your environment. 15 juli 2025

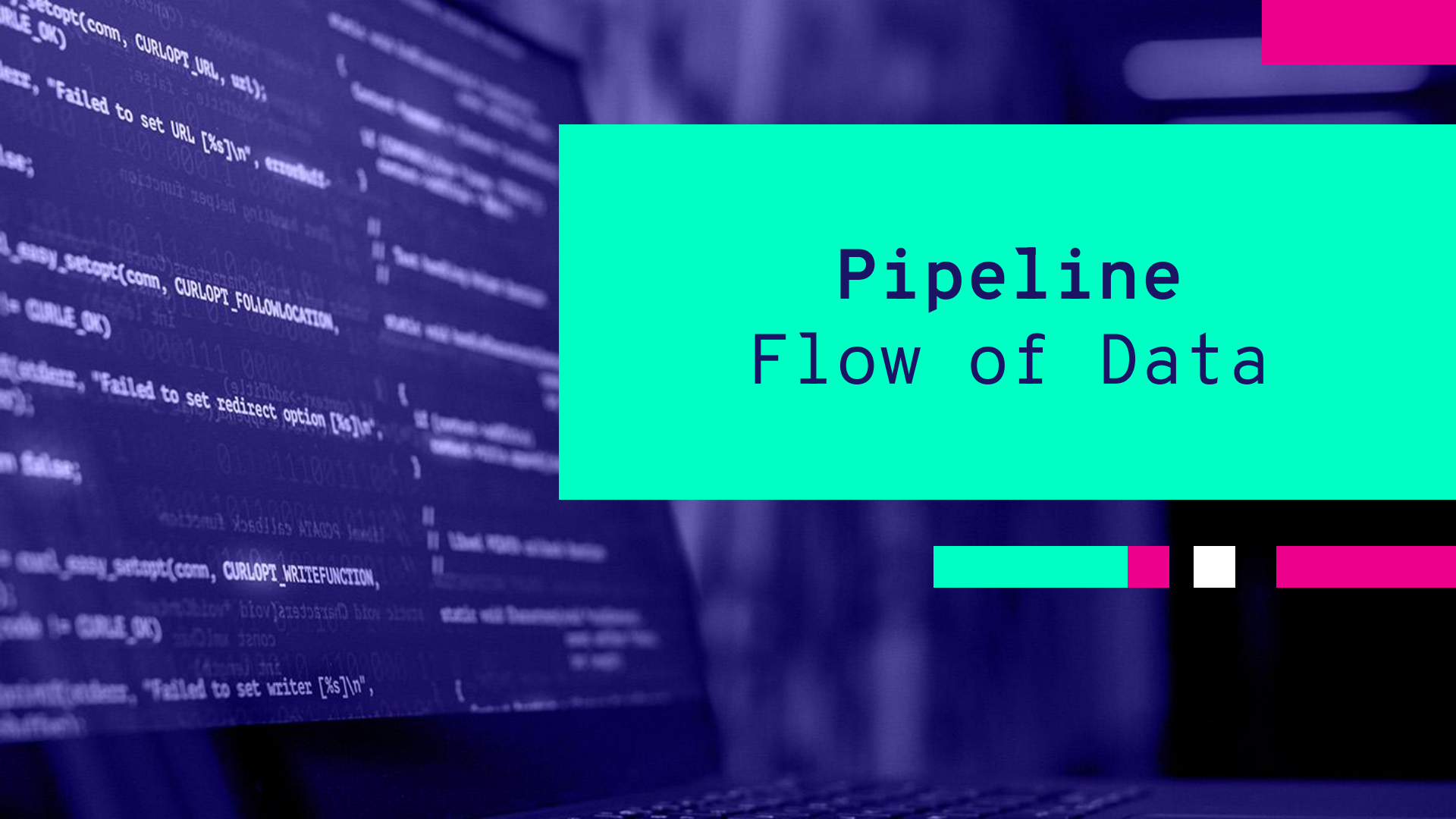
Source fungerar, men är onödigt i .venv projekt

https://stackoverflow.com/questions/79701540/pip-and-uv-point-to-local-bin-even-after-activating-virtual-environment




```
$ uv init
$ uv add “fastapi[standard]”
$ uv add “psycopg[binary]”
$ uv add “psycopg[pool]”
$ uv run uvicorn app.main:app --reload
```

```
app = FastAPI(title="demo_7")
NOTE: this ^ is required for ‘uv run’ to work!
```


Pipeline
Flow of Data

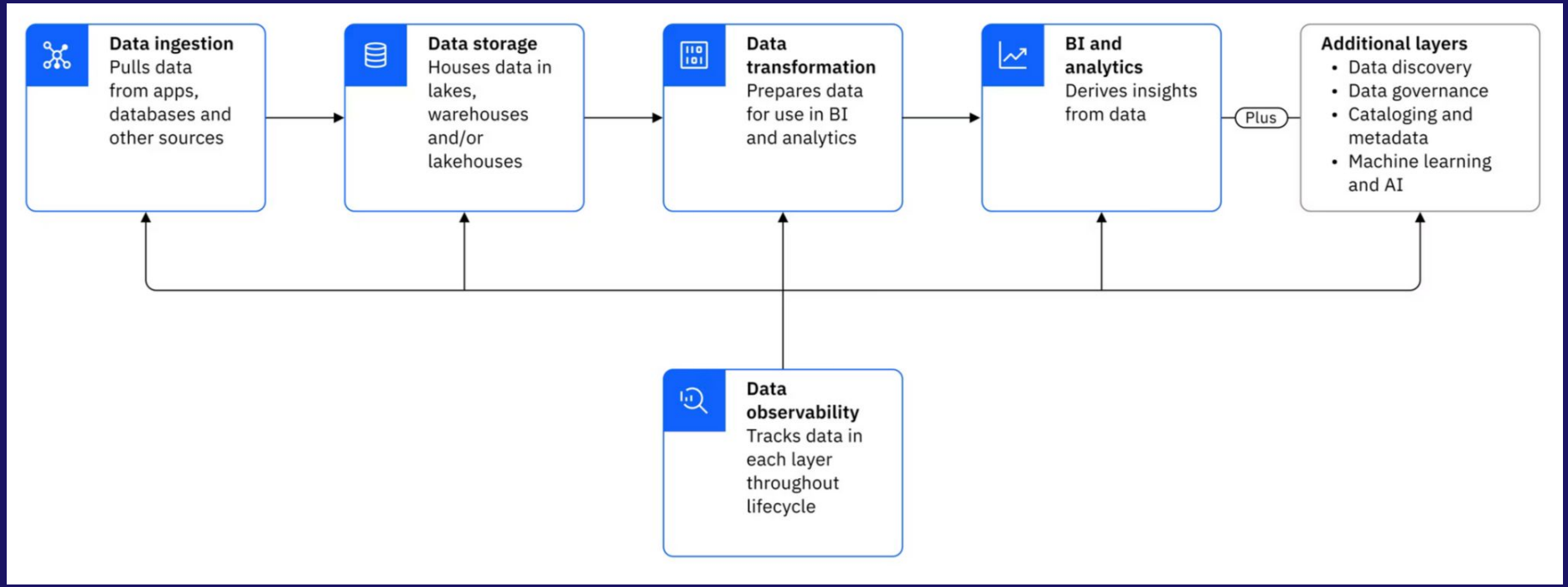
“A defined, repeatable process that moves data from a source to a destination, optionally transforming it along the way“

“Pipelines gör dataflöden pålitliga, förutsägbara och lätta att förstå”

Price = 999999

Price = none

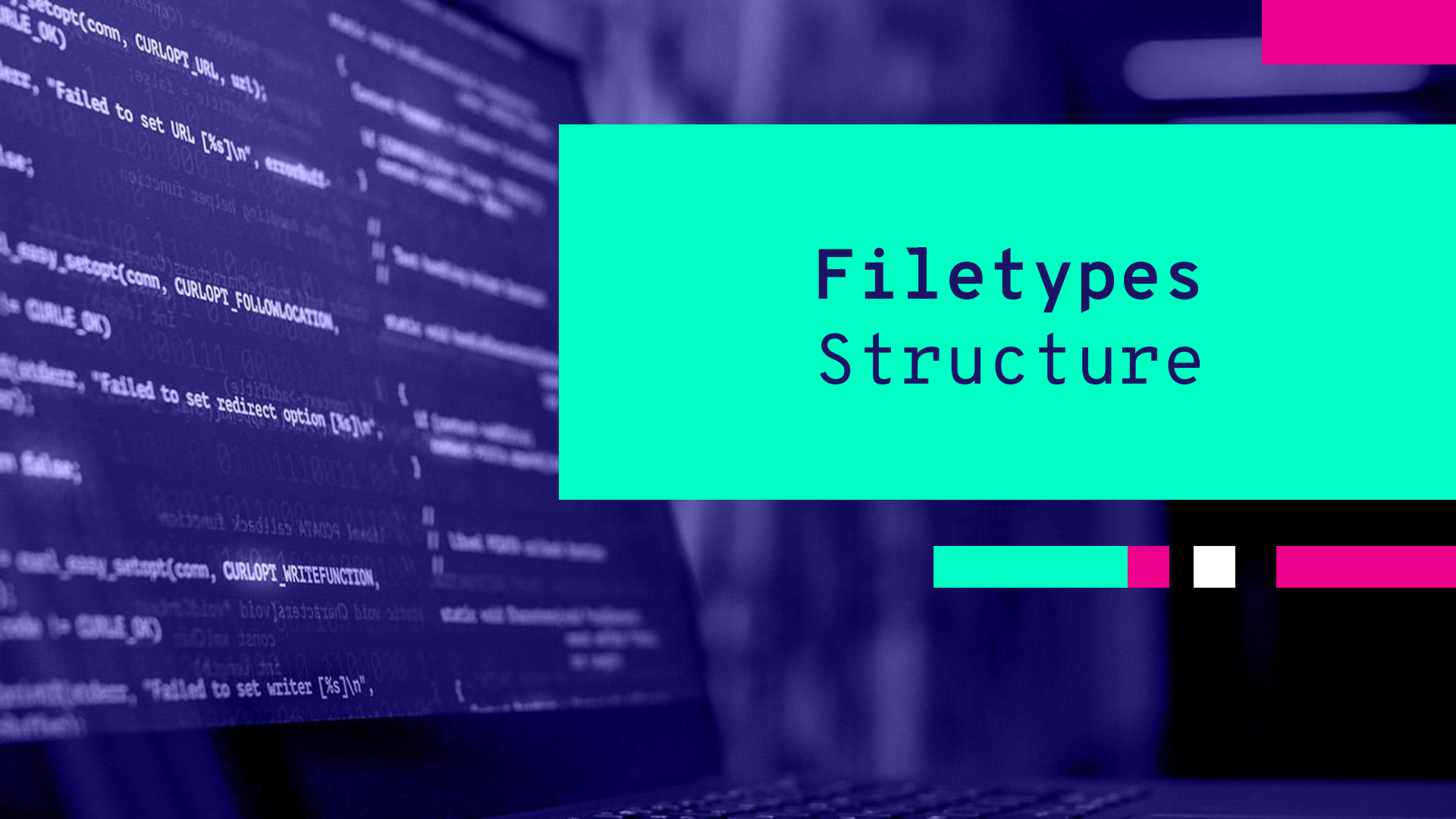
Price = “294.0”




Project
Setup

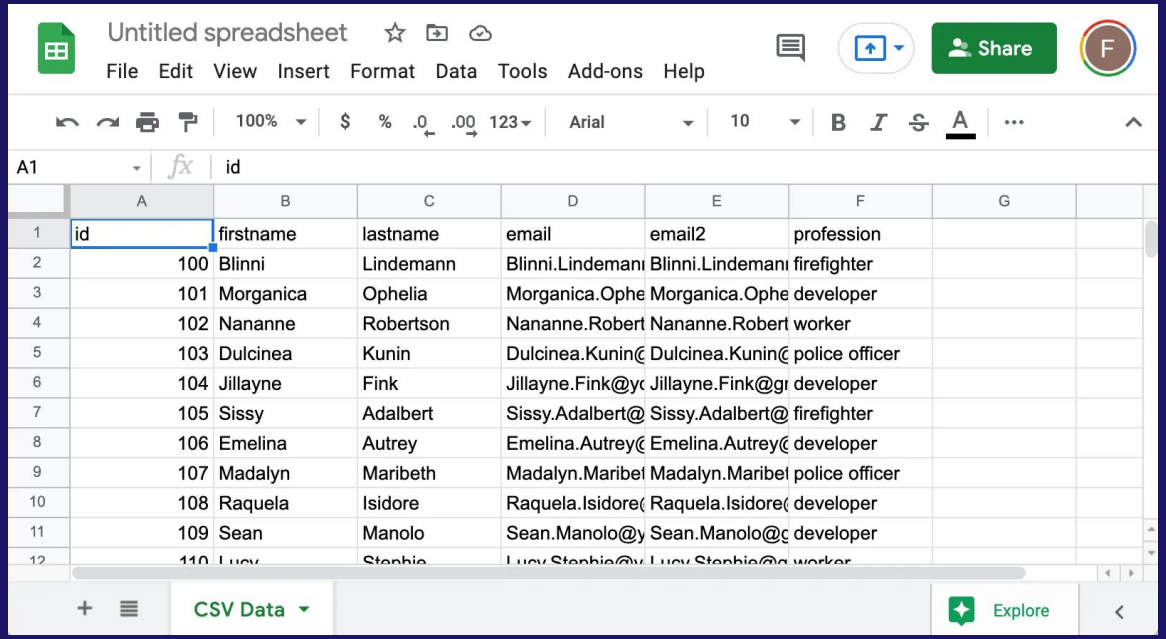
Git Clone
(Repository)

https://github.com/Krillinator/data-platform-lektion-6-preparation


Filetypes
Structure

Filetypes
(Comma Separated Values)

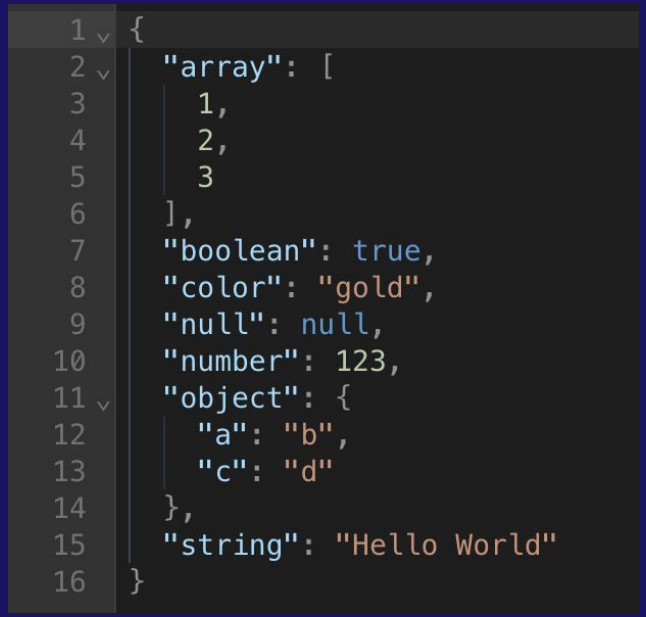
> CSV
JSON
JSONL / NDJSON
Parquet
Avro
ORC
XML
TXT / LOG
Excel (XLSX)
GZIP / ZIP



	A	B	C	D	E	F	G
1	id	firstname	lastname	email	email2	profession	
2	100	Blinni	Lindemann	Blinni.Lindemani	Blinni.Lindemani	firefighter	
3	101	Morganica	Ophelia	Morganica.Ophe	Morganica.Ophe	developer	
4	102	Nananne	Robertson	Nananne.Robert	Nananne.Robert	worker	
5	103	Dulcinea	Kunin	Dulcinea.Kunin@	Dulcinea.Kunin@	police officer	
6	104	Jillayne	Fink	Jillayne.Fink@yc	Jillayne.Fink@gr	developer	
7	105	Sissy	Adalbert	Sissy.Adalbert@	Sissy.Adalbert@	firefighter	
8	106	Emelina	Autrey	Emelina.Autrey@	Emelina.Autrey@	developer	
9	107	Madalyn	Maribeth	Madalyn.Maribet	Madalyn.Maribet	police officer	
10	108	Raquela	Isidore	Raquela.Isidore@	Raquela.Isidore@	developer	
11	109	Sean	Manolo	Sean.Manolo@y	Sean.Manolo@g	developer	
12	110	Lucy	Stephie	Lucy.Stephie@y	Lucy.Stephie@g	worker	

Filetypes
(Javascript Object Notation)

CSV
> JSON
JSONL / NDJSON
Parquet
Avro
ORC
XML
TXT / LOG
Excel (XLSX)
GZIP / ZIP



```json
{
  "array": [
    1,
    2,
    3
  ],
  "boolean": true,
  "color": "gold",
  "null": null,
  "number": 123,
  "object": {
    "a": "b",
    "c": "d"
  },
  "string": "Hello World"
}
```

Filetypes
(JSONL / NDJSON – Explained)

CSV
JSON
> JSONL / NDJSON
Parquet
Avro
ORC
XML
TXT / LOG
Excel (XLSX)
GZIP / ZIP

Differences Between JSON and JSONL

Before diving deeper into how to work with **pandas jsonl**, let's consider the differences between JSON and JSONL.

- **JSON**: Typically stores data as one large object. It is suitable for smaller datasets or when you need to represent complex nested structures.
- **JSONL**: Each JSON object is line-separated. This format is more efficient when dealing with large datasets, as you can read or write one line at a time without loading the entire dataset into memory.

Filetypes
(JSONL / NDJSON - Showcase difference)

Json: Must be read as a whole object - the longer the object == longer time

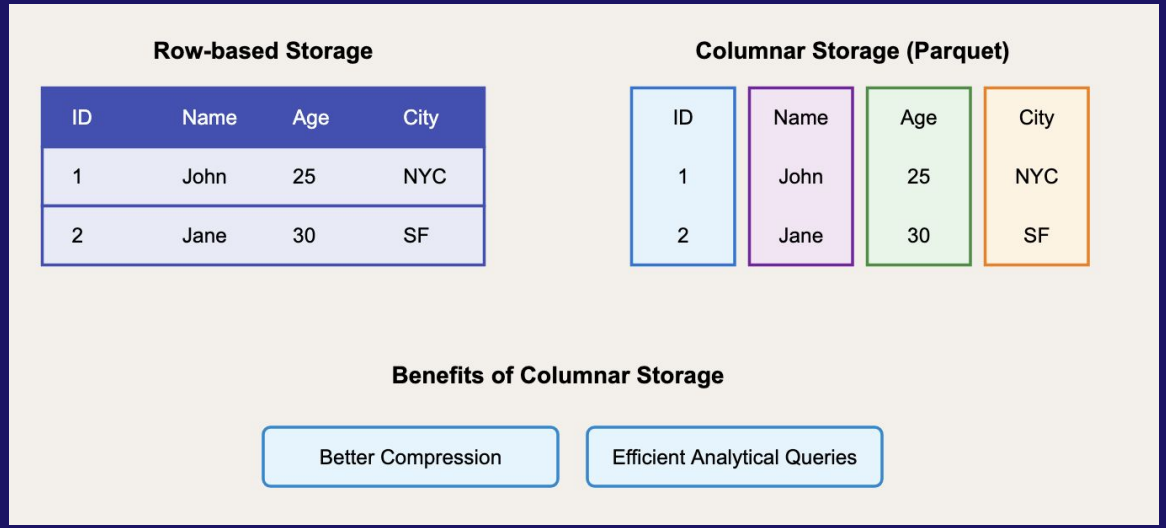
```
[
  { "name": "USB-C Cable", "price": 149.0, "quantity": 120 },
  { "name": "Wireless Mouse", "price": 299.0, "quantity": 75 },
  { "name": "Keyboard", "price": 1299.0, "quantity": 25 }
]
```

JsonL: Read separately in chunks

```
{ "name": "USB-C Cable", "price": 149.0, "quantity": 120 }
{ "name": "Wireless Mouse", "price": 299.0, "quantity": 75 }
{ "name": "Keyboard", "price": 1299.0, "quantity": 25 }
```

Filetypes (Parquet)

CSV
JSON
JSONL / NDJSON
> Parquet
Avro
ORC
XML
TXT / LOG
Excel (XLSX)
GZIP / ZIP



Source:
https://motherduck.com/learn-more/why-choose-parquet-table-file-format/

Filetypes
(Parquet - Showcase)

```
Parquet file
 ├─ Row Group 1
 │   ├─ Column: id
 │   ├─ Column: name
 │   ├─ Column: age
 │   └─ Column: city
 ├─ Row Group 2
 │   ├─ Column: id
 │   ├─ Column: name
 │   ├─ Column: age
 │   └─ Column: city
```

Filetypes
(Parquet – Showcase)

CSV
JSON
JSONL / NDJSON
Parquet
> Avro
ORC
XML
TXT / LOG
Excel (XLSX)
GZIP / ZIP

- Large data volumes
- Strict schemas
- Schema changes over time
- Streaming systems (Kafka)

Filetypes
(Avro – Showcase (JSON))

This is normal **JSON**, easy to read, stored as text *(ASCII with UTF-8 as possible extensions)*

```
{
  "name": "USB-C Cable",
  "price": 149.0,
  "quantity": 120
}
```

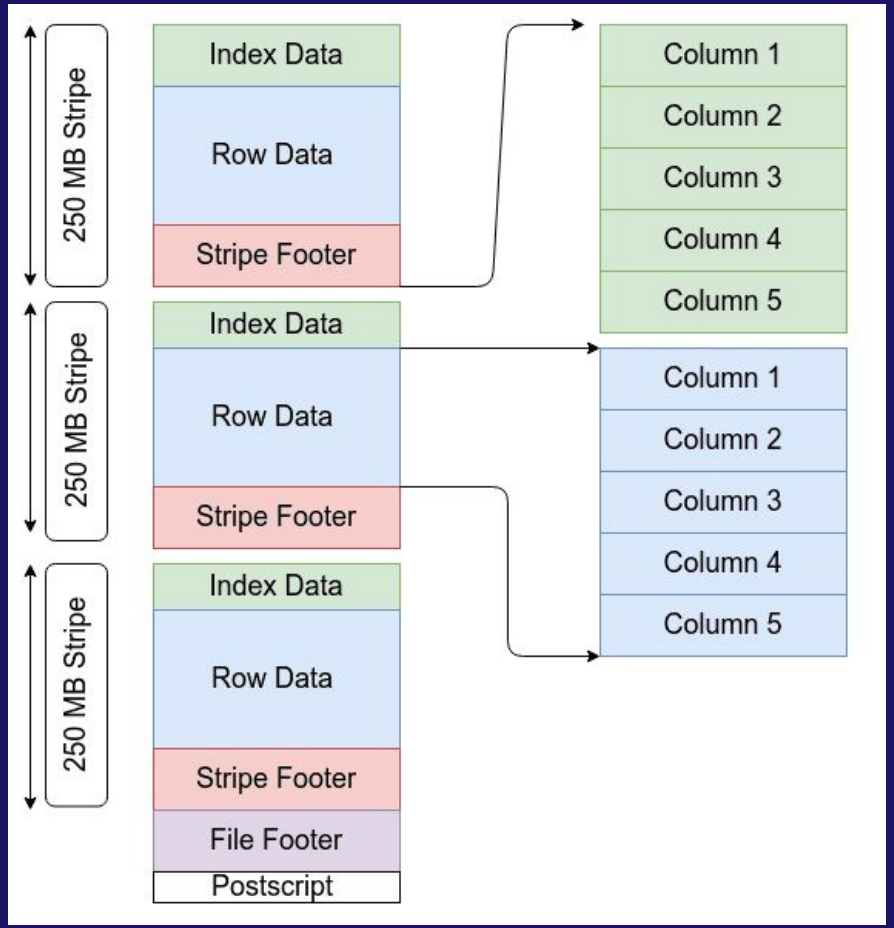
Filetypes
(Avro – Showcase (AVRO))

This is AVRO, stored in binary, not to be read by humans, but understood by machines

```
Obj��name
USB-C Cable
price��@
quantityx
```

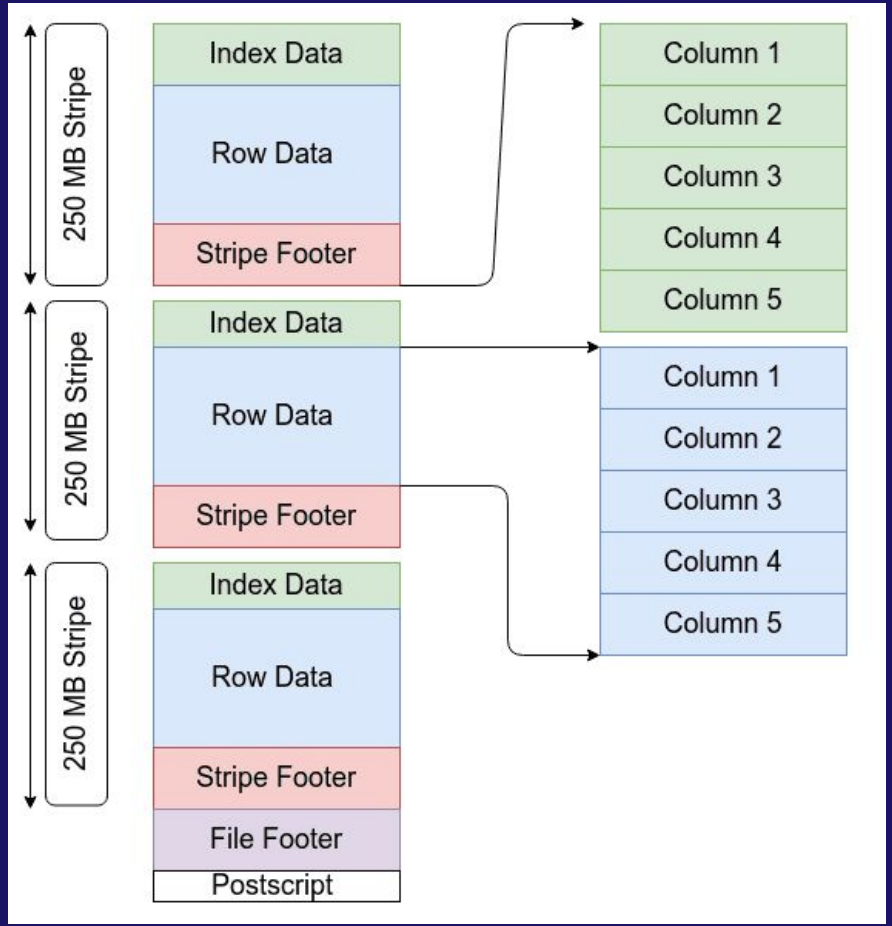
Filetypes (Optimized Row Columnar)

CSV
JSON
JSONL / NDJSON
Parquet
Avro
> ORC
XML
TXT / LOG
Excel (XLSX)
GZIP / ZIP



Filetypes (Optimized Row Columnar)

- Data is stored **by column**
- Grouped into **row groups** *(stripes)*
- Strong typing
- Heavy compression
- Less common today, Parquet is more preferred. Legacy systems still use this



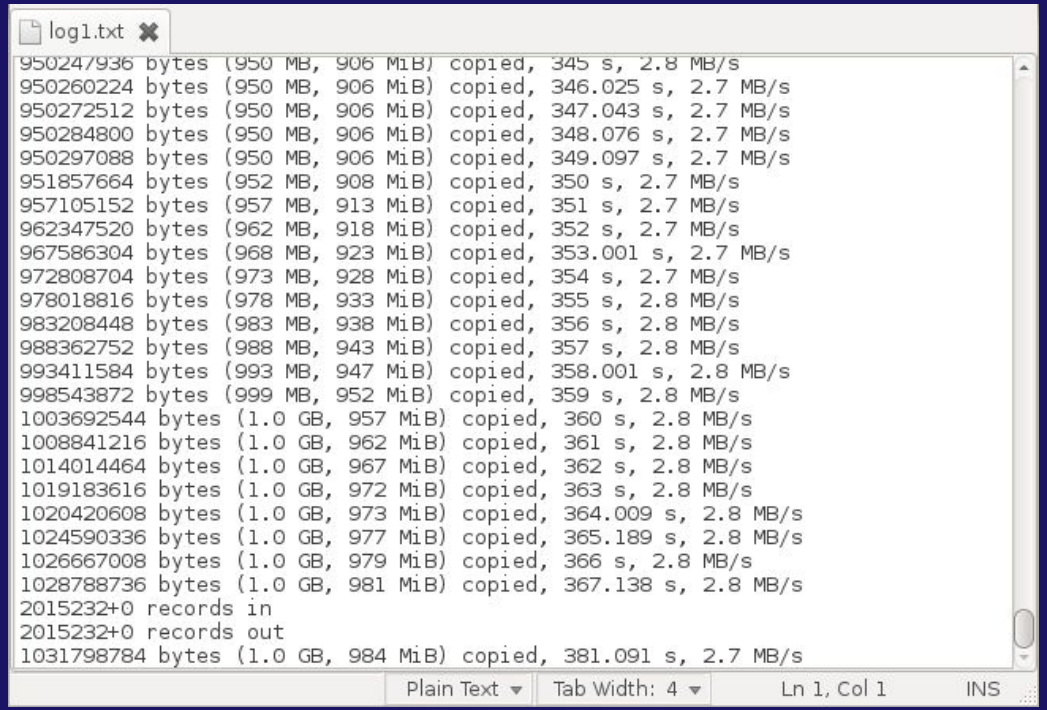
Filetypes
(eXtensible Markup Language)

CSV
JSON
JSONL / NDJSON
Parquet
Avro
ORC
> XML
TXT / LOG
Excel (XLSX)
GZIP / ZIP

https://www.w3schools.com/xml/cd_catalog.xml

Filetypes
(TXT / LOG)

CSV
JSON
JSONL / NDJSON
Parquet
Avro
ORC
XML
> TXT / LOG
Excel (XLSX)
GZIP / ZIP



Filetypes
(Excel Spreadsheet *(xml-based)*)

CSV
JSON
JSONL / NDJSON
Parquet
Avro
ORC
XML
TXT / LOG
> Excel (XLSX)
GZIP / ZIP

XLSX is a binary, structured file format for spreadsheets (xml based)

- Is a **ZIP archive**
- Contains **XML files**
  - With sheets,
  - styles,
  - formulas,
  - metadata

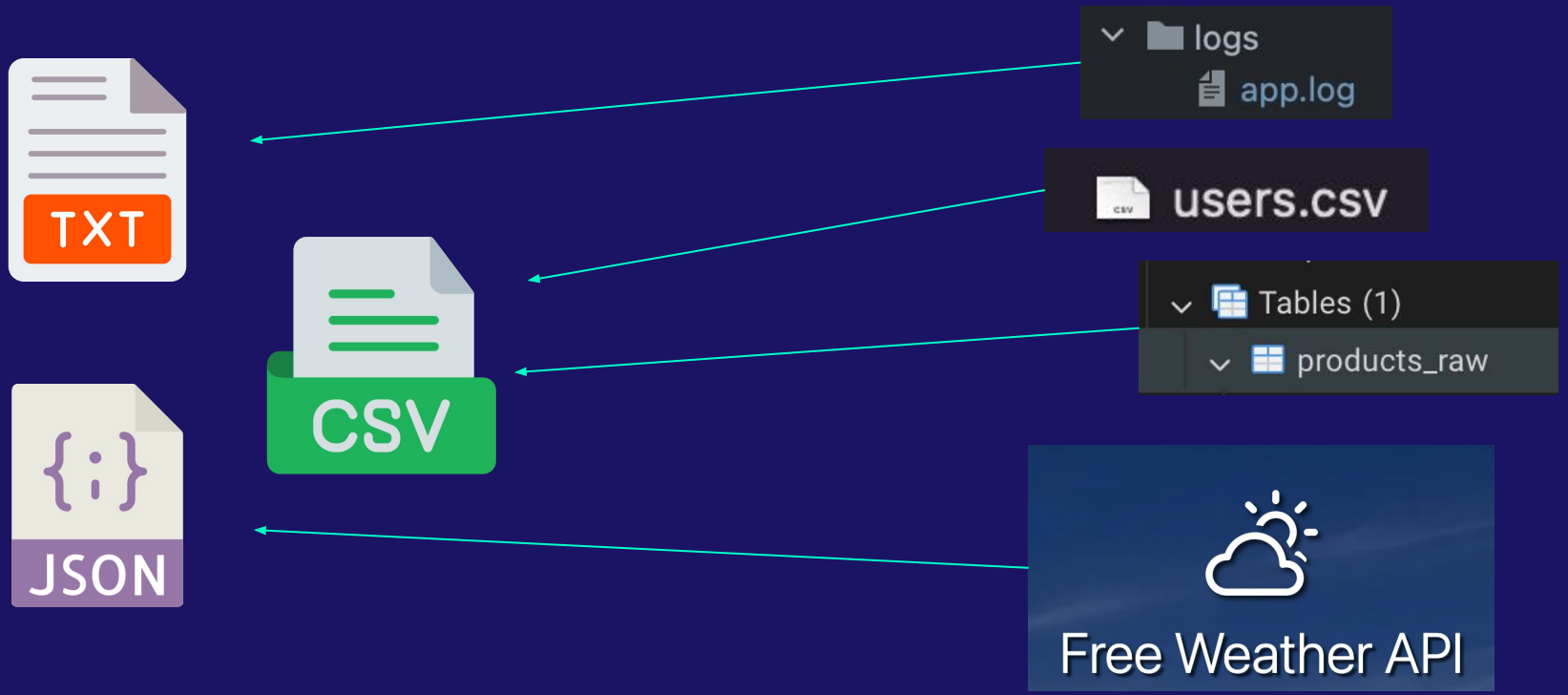
Filetypes
(GZIP / ZIP)

CSV
JSON
JSONL / NDJSON
Parquet
Avro
ORC
XML
TXT / LOG
Excel (XLSX)
> GZIP / ZIP

“How are files packaged and compressed”


ETL
Extract, Transform, Load

ETL
(Extract)


logs
app.log
TXT
users.csv
CSV
Tables (1)
products_raw
JSON
Free Weather API

ETL
(Is this ready to be stored?)

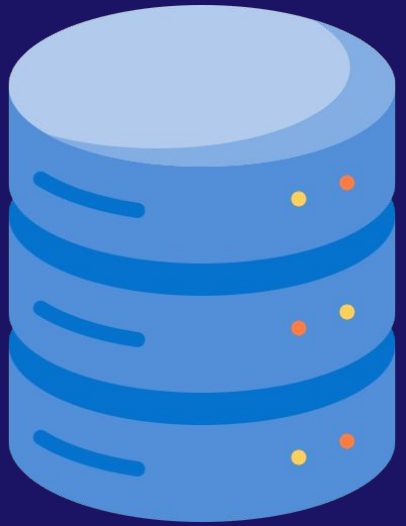
```
{
  "userId": "u-42",
  "name": "  Anna  Svensson ",
  "age": "29",
  "signupDate": "2024/01/15",
  "purchases": [
    { "item": "Book", "price": "199", "currency": "SEK" },
    { "item": "Pen", "price": "19", "currency": "sek" }
  ]
}

```

ETL
(Transform)

“Data transformation is everything you do ***to clean, reshape, and standardize data so it can be trusted and used”***

ETL
(Load)

Transformed data



The ETL Process Explained

Extract

Retrieves and verifies data from various sources

Transform

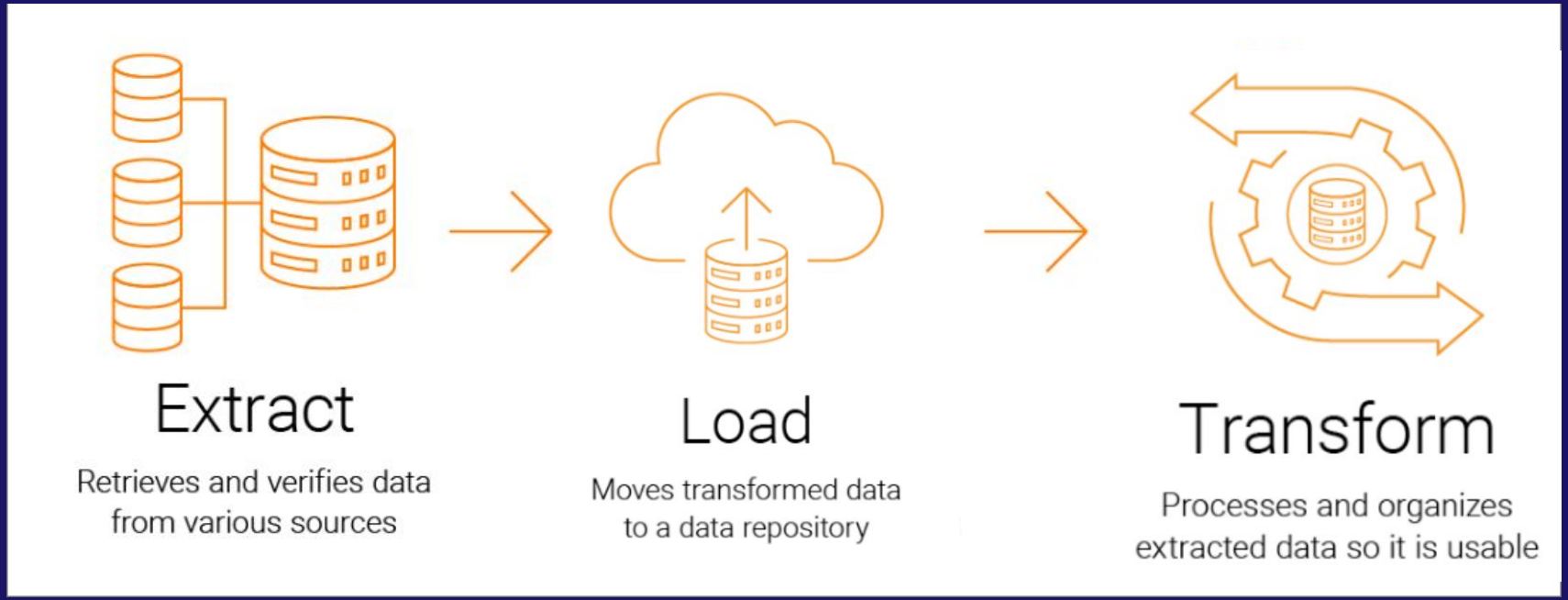
Processes and organizes extracted data so it is usable

Load

Moves transformed data to a data repository


ELT
Extract, Load,
Transform

ELT
(Extract, Load, Transform)



ETL vs ELT: What's the difference?

	ETL	ELT
Extraction	Raw data is extracted from disparate sources	Raw data is extracted from disparate sources
Transformation	Raw data is transformed on a secondary server or staging area	Data is transformed within the data warehouse or system
Loading	Data is loaded into the warehouse or system after transformation	Raw data is loaded directly into the warehouse or system

03

Workshop


Data Insertion
Creating Problems

Database Inserts

Work Lab

Dela upp er 2-2 eller 3-3

Krav innan ni kör igång:

- *GitHub Projekt (slide #15)*
- *Projekt kopplad till databas*
- *Postman/thunderclient öppen*

Database Inserts

Work Lab

Skapa produkter som går in mot databasen,
men som inte riktigt är helt rätt

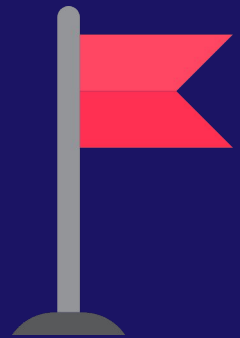
Exempelvis:

```
{
    "name": "Wireless Mouse",
    "price": 249.0,
    "quantity": "2",
    "currency": "SEK"
}
```


Data Quality
Checks

Vad gör vi när vi saknar fält värden?

order_id	customer_email	amount	quality_status	is_valid
1001	alice@example.com	250.00	OK	TRUE
1002		180.50	MISSING_EMAIL	FALSE
1003	bob@example.com		-MISSING_AMOUNT	FALSE



Tänk om datan är omöjlig?

raw_row	reason	rejected_at
{"order_id":null,"customer_email":"a@b.com","amount":120}	MISSING_ID	2026-02-09T00:42:00Z
{"order_id":"ABC","customer_email":"x@y.com","amount":-50}	INVALID_AMOUNT	2026-02-09T00:42:01Z
{"order_id":1003,"customer_email":"bad@@mail","amount":"NaN"}	BROKEN_DATATYPE	2026-02-09T00:42:02

Regler för datahantering i ETL

(Transformera - Flagga - Avvisa)
(Transform - flag - reject)

Grundprincip

- Ingen gissning. Inga antaganden. (never guess)
- Vi ändrar inte betydelse (the purpose of data remains unchanged)
- Vi gör data användbar eller synlig som problem (data is usable or visible as problem)
- Vi korrigerar format.. inte innehåll. *(correct format, not content)*

I kommande uppgifter, förlita dig på att vi arbetar mot postgreSQL som databas

```json
{
    "product_id": "USP239",
    "name": "Wireless Mouse",
    "price": 249.0,
    "currency": "SEK",
    "category": "Electronics",
    "brand": null
}
```

Ser datan korrekt ut, **acceptera**

```
{
  "product":  "  mouse.  ",
  "price": 199,
  "currency": "SEK"
}
```

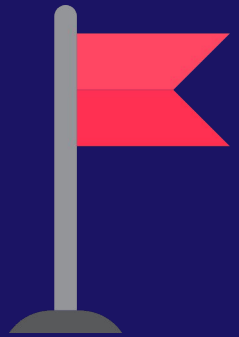
När datan **inte** är **korrekt** *(datatyp, mellanslag, fel ISO standard)*

Då behövs datan **städas upp** och transformeras till något mer användbart.

UTAN ATT ÄNDRA DATANS INNEBÖRD

```json
{
  "product": "HDMI cable",
  "price": 1,
  "currency": "SEK"
}
```

- tekniskt giltig
- misstänkt lågt värde
- kan vara korrekt *(kampanj, fel, bulk)*



Åtgärd

- behåll posten
- märk den som flaggad
- exkludera från beräkningar som standard

```json
{
  "product": "Laptop",
  "price": 12000
}
```

Problemtyp

- obligatoriskt fält saknas (`currency`)
- datan kan inte användas alls
- Omöjliga värden
- Ingen identitet

```json
{
  "original_data": {
    "product": "Laptop",
    "price": 12000
  },
  "reject_reason": "missing_required_field: currency"
}
```

Accept

OK or ✓

Clean/Transform

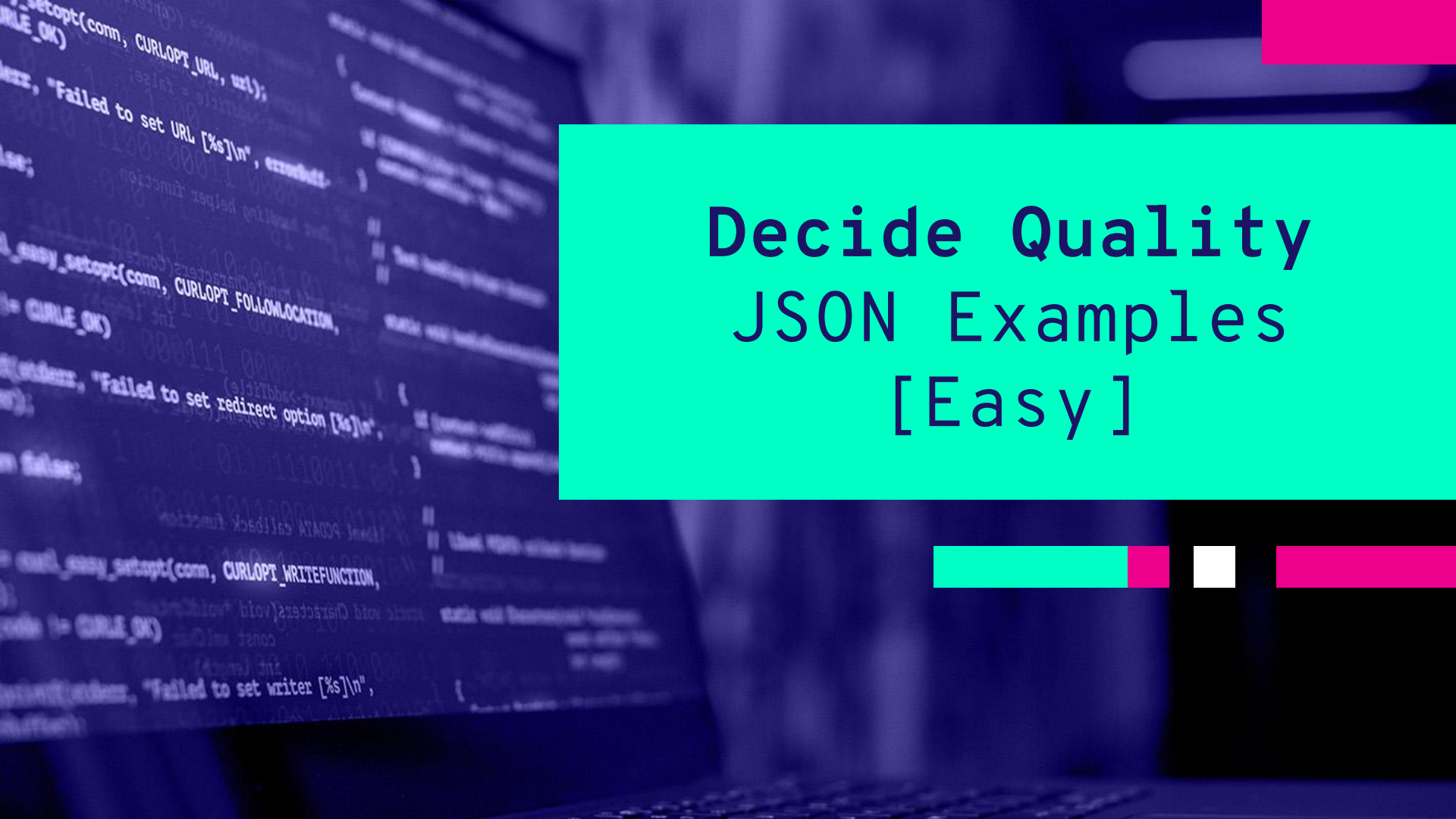
>> or ~>

Flag

! or ?

Reject

X or //


Decide Quality
JSON Examples
[Easy]

Example 1

```
{
  "id": "sku-29",
  "name": "  Milk  ",
  "price": 19,
  "currency": "sek"
}
```

Example 2

```
{
  "id": "sku-30",
  "name": "Bread",
  "price": "25",
  "currency": "SEK"
}
```

Example 3

```
{
  "id": "sku-31",
  "name": "Shoes",
  "price": -799,
  "currency": "SEK"
}
```

Example 4

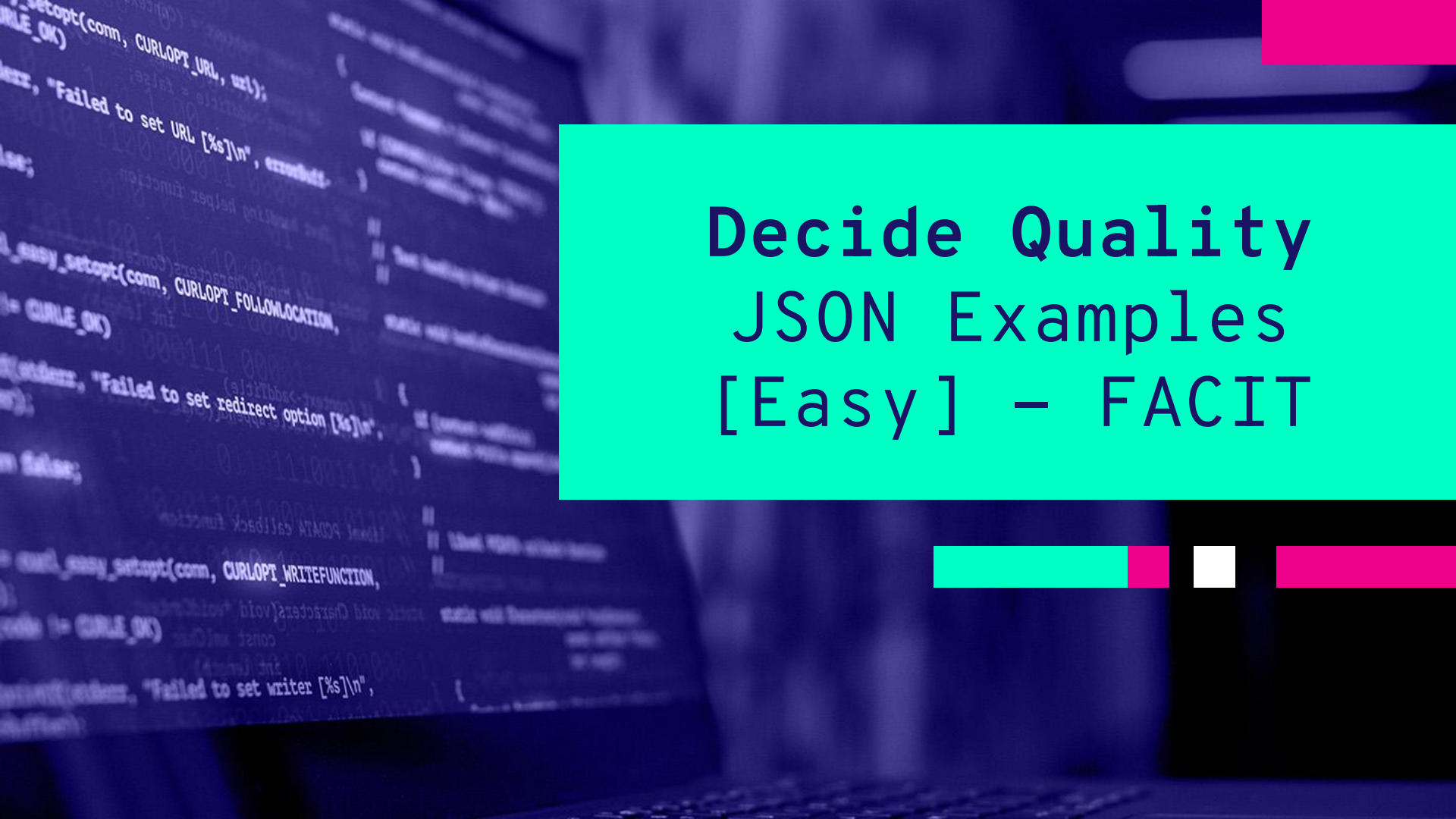
```
{
  "id": "sku-32",
  "name": "Socks",
  "price": 99,
  "quantity": “two”,
  "currency": "SEK"
}
```

Example 5

```
{
  "id": "sku-33",
  "name": "Headphones",
  "price": 199,
  "currency": "USD",
  "country": "SE"
}
```

Example 6

```
{
  "order_id": "A-1001",
  "customer": { "id": "c42", "name": "  anna  " },
  "items": [
    { "sku": "KB-1", "unit_price": "199", "quantity": "2", "currency": "sek" },
    { "sku": "MS-9", "unit_price": "99",  "quantity": "1", "currency": "SEK" }
  ]
}
```


Decide Quality
JSON Examples
[Easy] – FACIT

Example 1 (FACIT)

```
{
  "id": "sku-29",
  "name": "  Milk  ",
  "price": 19,
  "currency": "sek"
}
```

Transform >>

Trimming and uppercasing does not change meaning.

Example 2
(FACIT)

```
{
  "id": "sku-30",
  "name": "Bread",
  "price": "25",
  "currency": "SEK"
}
```

Transform >>

Clearly numeric, only representation is wrong.

Example 3
(FACIT)

```
{
  "id": "sku-31",
  "name": "Shoes",
  "price": -799,
  "currency": "SEK"
}
```

Flag !
Could be refund, error, or discount.. intent unclear.

Example 4
(FACIT)

```
{
  "id": "sku-32",
  "name": "Socks",
  "price": 99,
  "quantity": “two”,
  "currency": "SEK"
}
```

Reject X

Breaking type rules for ‘two’ aka 2

Example 5
(FACIT)

```
{
  "id": "sku-33",
  "name": "Headphones",
  "price": 199,
  "currency": "USD",
  "country": "SE",
  "pricing_currency": "SEK"
}
```

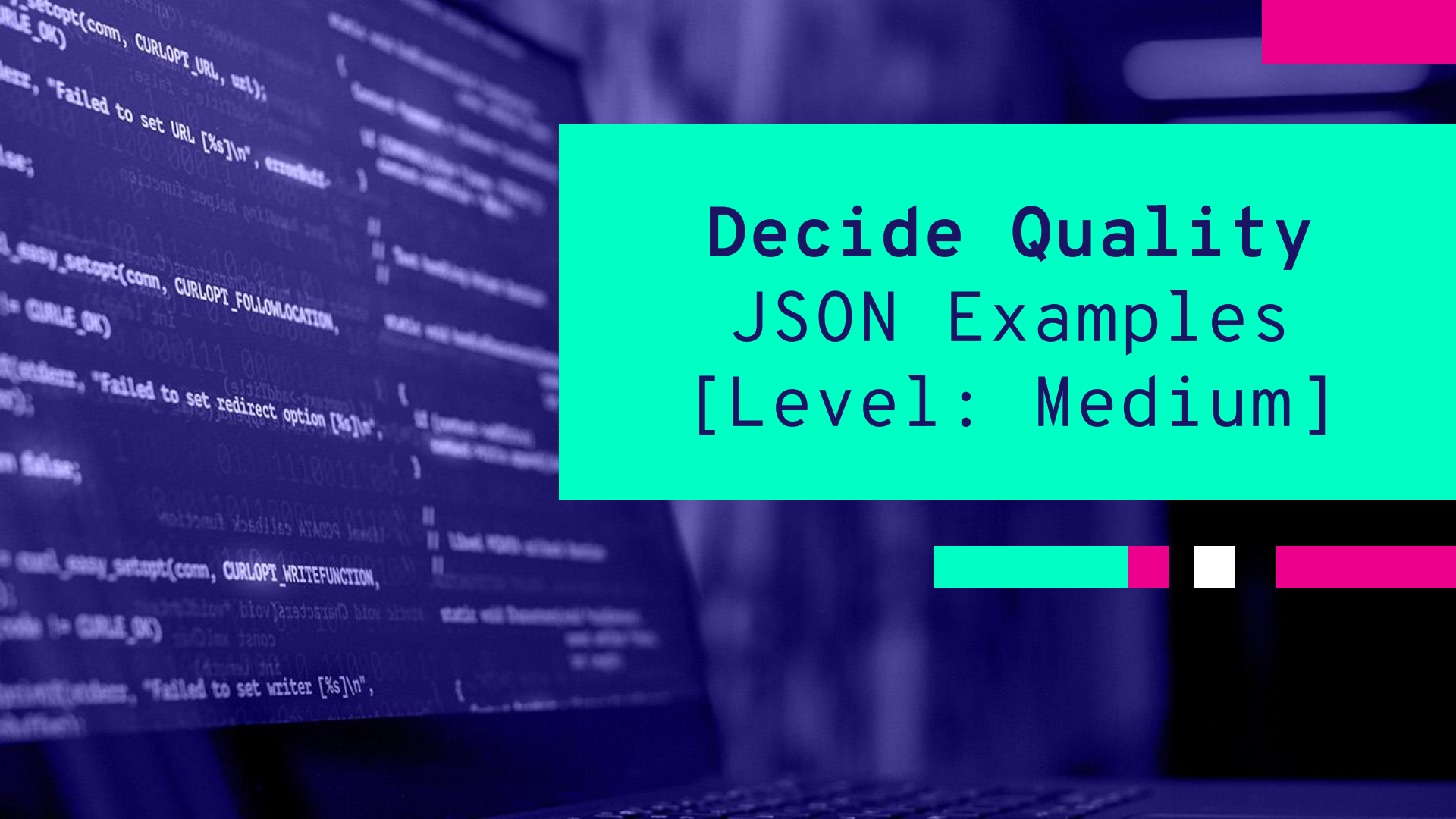
Reject X

Two currencies, no precedence rule

Example 6
(FACIT)

```
{
  "order_id": "A-1001",
  "customer": { "id": "c42", "name": "  anna  " },
  "items": [
    { "sku": "KB-1", "unit_price": "199", "quantity": "2", "currency": "sek" },
    { "sku": "MS-9", "unit_price": "99", "quantity": "1", "currency": "SEK" }
  ]
}
```

Transform >>
Trim name, convert price/quantity to numbers, uppercase currency. No meaning change.


Decide Quality
JSON Examples
[Level: Medium]

Example 1

```json
{
  "order_id": "A-1002",
  "customer": { "id": "c43", "name": "Erik" },
  "items": [
    { "sku": "KB-1", "unit_price": 199, "quantity": 2, "currency": "SEK" },
    { "unit_price": 99, "quantity": 1, "currency": "SEK" }
  ]
}
```

Example 2

```
{
  "order_id": "A-1003",
  "customer": { "id": "c44", "name": "Sara" },
  "items": [
    { "sku": "HDMI-2", "unit_price": 1, "quantity": 3, "currency": "SEK" },
    { "sku": "SSD-1T", "unit_price": 9999, "quantity": 1, "currency": "SEK" }
  ],
  "totals": { "declared_total": 4, "currency": "SEK" }
}
```

Example 3

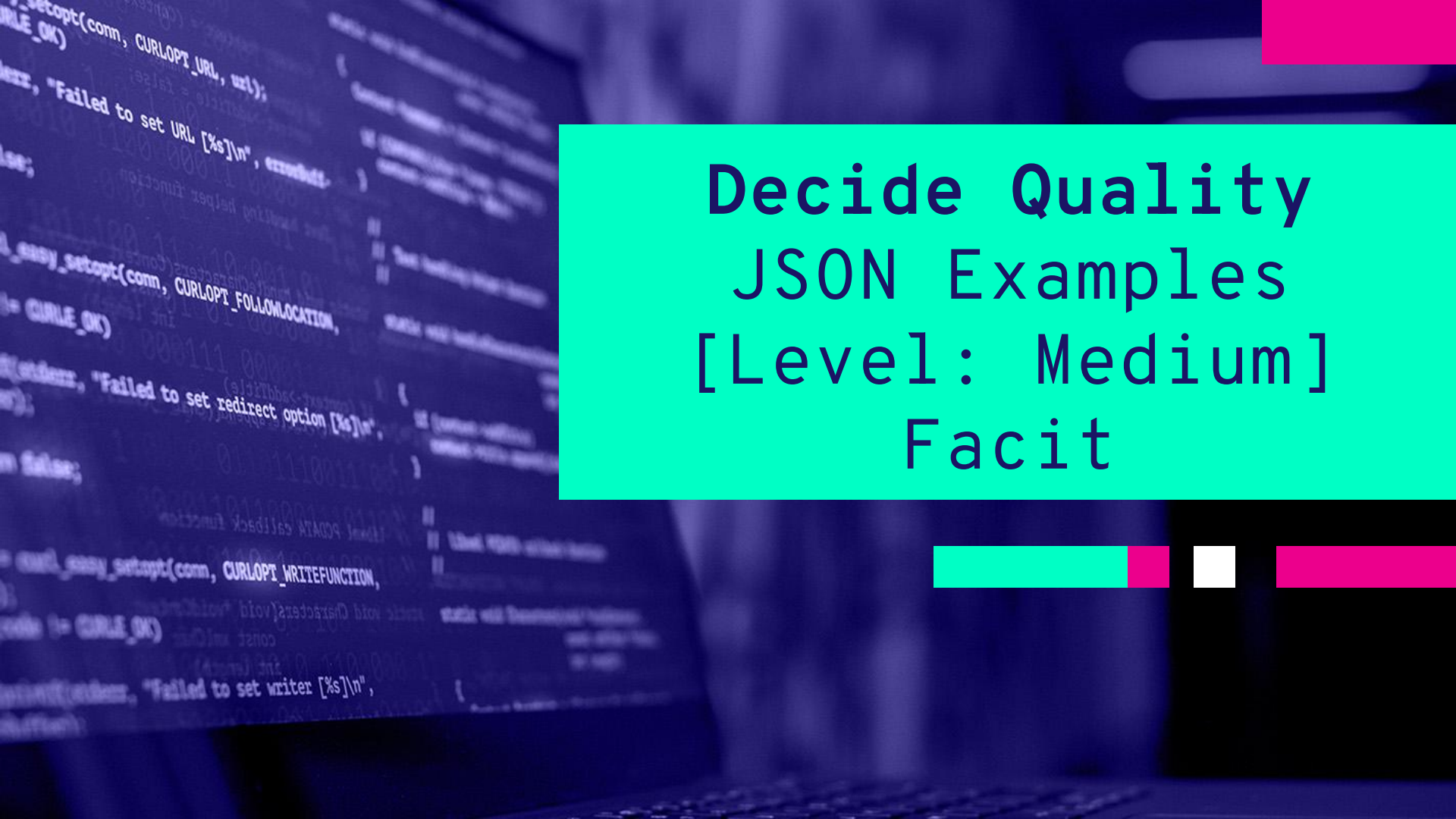
```json
{
  "order_id": "A-2001",
  "customer": {
    "id": "c51",
    "name": "Maria"
  },
  "items": [
    {
      "sku": "KB-1",
      "unit_price": 199,
      "quantity": "two",
      "currency": "SEK"
    }
  ]
}
```

Example 4

```
{
  "subscription_id": "S-1001",
  "start_date": "2024-06-01",
  "end_date": "2024-05-01"
}
```

THE FINALE

```
{
  "order_id": "1005",
  "customer": {
    "id": "c55",
    "is_vip": "true"
  },
  "shipment": {
    "shipped_at": "2024-13-13",
    "delivered_at": "2024-13-14",
    "tracking": { "code": 123456, "carrier": "DHL" }
  },
  "items": [
    { "sku": "KB-1", "unit_price": "199.00", "quantity": 2, "currency": "SEK" }
  ],
  "paid": "yes"
}
```



Decide Quality JSON Examples [Level: Medium] Facit

Example 1
(FACIT)

```
{
  "order_id": "A-1002",
  "customer": { "id": "c43", "name": "Erik" },
  "items": [
    { "sku": "KB-1", "unit_price": 199, "quantity": 2, "currency": "SEK" },
    { "unit_price": 99, "quantity": 1, "currency": "SEK" }
  ]
}
```

Reject X

Second item missing required sku, invalid record.
If you decide to keep it, only reject that nested item

Example 2
(FACIT)

```json
{
  "order_id": "A-1003",
  "customer": { "id": "c44", "name": "Sara" },
  "items": [
    { "sku": "HDMI-2", "unit_price": 1, "quantity": 3, "currency": "SEK" },
    { "sku": "SSD-1T", "unit_price": 9999, "quantity": 1, "currency": "SEK" }
  ],
  "totals": { "declared_total": 4, "currency": "SEK" }
}
```

Accept OK
All looks great!

Example 3
(FACIT)

```
{
  "order_id": "A-2001",
  "customer": {
    "id": "c51",
    "name": "Maria"
  },
  "items": [
    {
      "sku": "KB-1",
      "unit_price": 199,
      "quantity": "two",
      "currency": "SEK"
    }
  ]
}
```

Reject X

Cannot safely transform without guessing.
Hard datatype rule violated

Example 4
(FACIT)

```
{
  "subscription_id": "S-1001",
  "start_date": "2024-06-01",
  "end_date": "2024-05-01"
}
```

Reject X

Dates are valid ISO strings but the logic is impossible. End cannot be before start date.

FACIT – REJECT

```
{
  "order_id": "1005",
  "customer": {
    "id": "c55",
    "is_vip": "true"
  },
  "shipment": {
    "shipped_at": "2024-13-13",
    "delivered_at": "2024-13-14",
    "tracking": { "code": 123456, "carrier": "DHL" }
  },
  "items": [
    { "sku": "KB-1", "unit_price": "199.00", "quantity": 2, "currency": "SEK" }
  ],
  "paid": "yes"
}
```

THANKS !

Do you have any questions?
kristoffer.johansson@sti.se