Python: Jack of ETL Trades

Using Python standard library to build ETL systems.

What’s my story?

- Recently moved from out West (Toronto)
- Began career in finance, CFA Charterholder
- Most of IT career reporting into CFO
- End-to-end data warehouse experience
- Integrating accounting and operational data
- Standardization is central requirement
- Mutable ledgers cause frequent rebuilds
- Legacy systems compete in the market

The Context Manager Slide

- You do ETL, and want to see under the hood
- You don’t ETL, but want to learn
- New to Python, learn the standard library
- Experienced Python developer, who doesn’t do ETL and one of the following are true:
    - You snuck in, the door locked behind you
    - Your thought the talk was on asyncio
    - Downtown has ~~cheaper, better~~ beer

Agenda

- A few basics of data warehousing
- How SSIS works, some challenges
- Can you build ETL systems in Python?
- Improving Python performance while preserving legibility

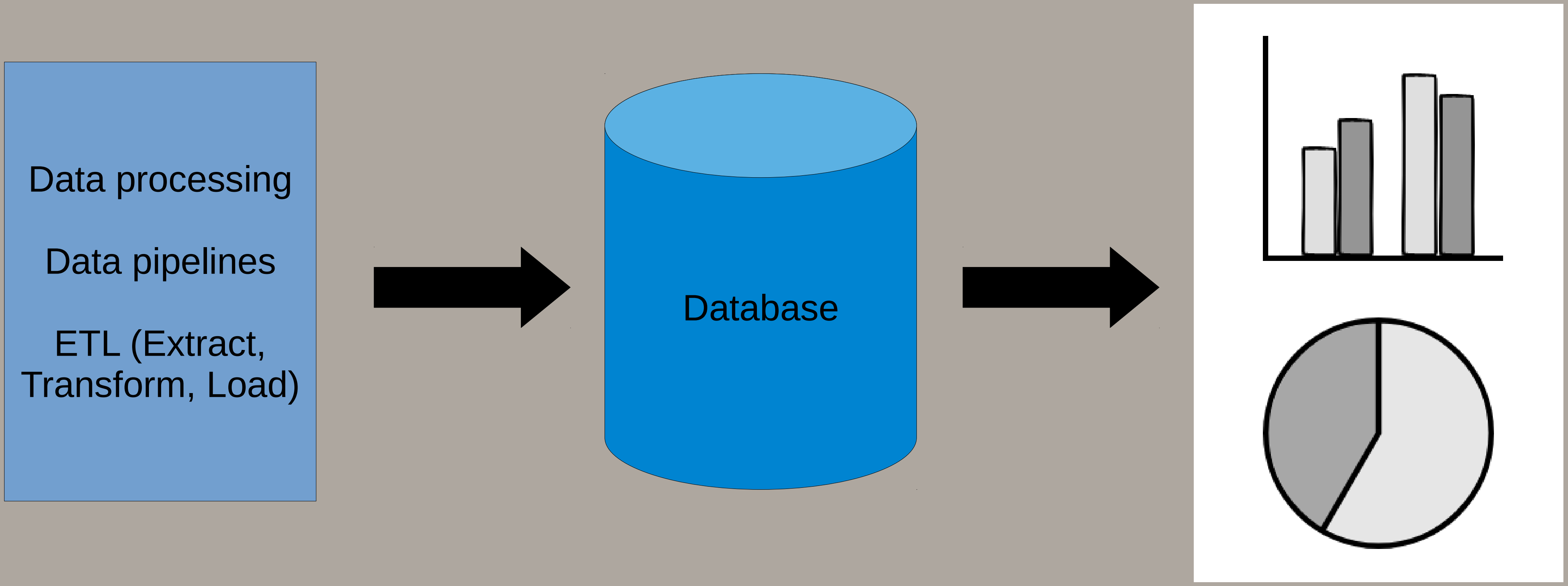
Data Warehousing

- Data warehousing is a business function that stores and manages data assets
- Common uses are analyzing and reporting on business performance
- Common implementations include one or more database/datastore(s)
- No explicit relationship to Big Data/Data Science

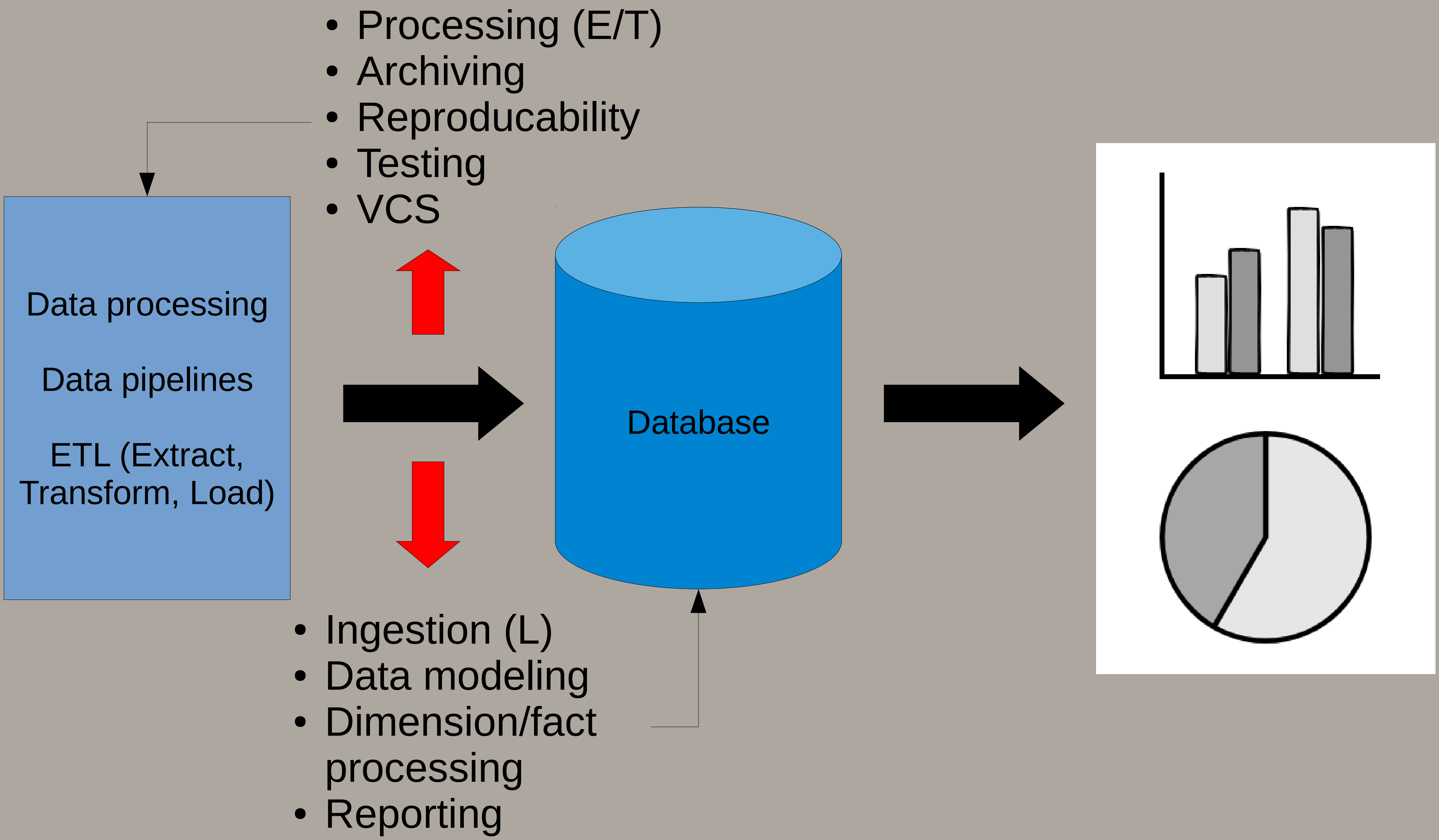
Disclaimers and Beliefs

- I don’t dislike SSIS
- I am neither an SSIS nor Python expert
- SSIS can has solutions within SSIS
- I aim to apply good development practices to the art of data warehousing
  - Testing and source control
  - Uptime and stability
  - Twelve-factor data warehouse

Data Warehousing in one slide



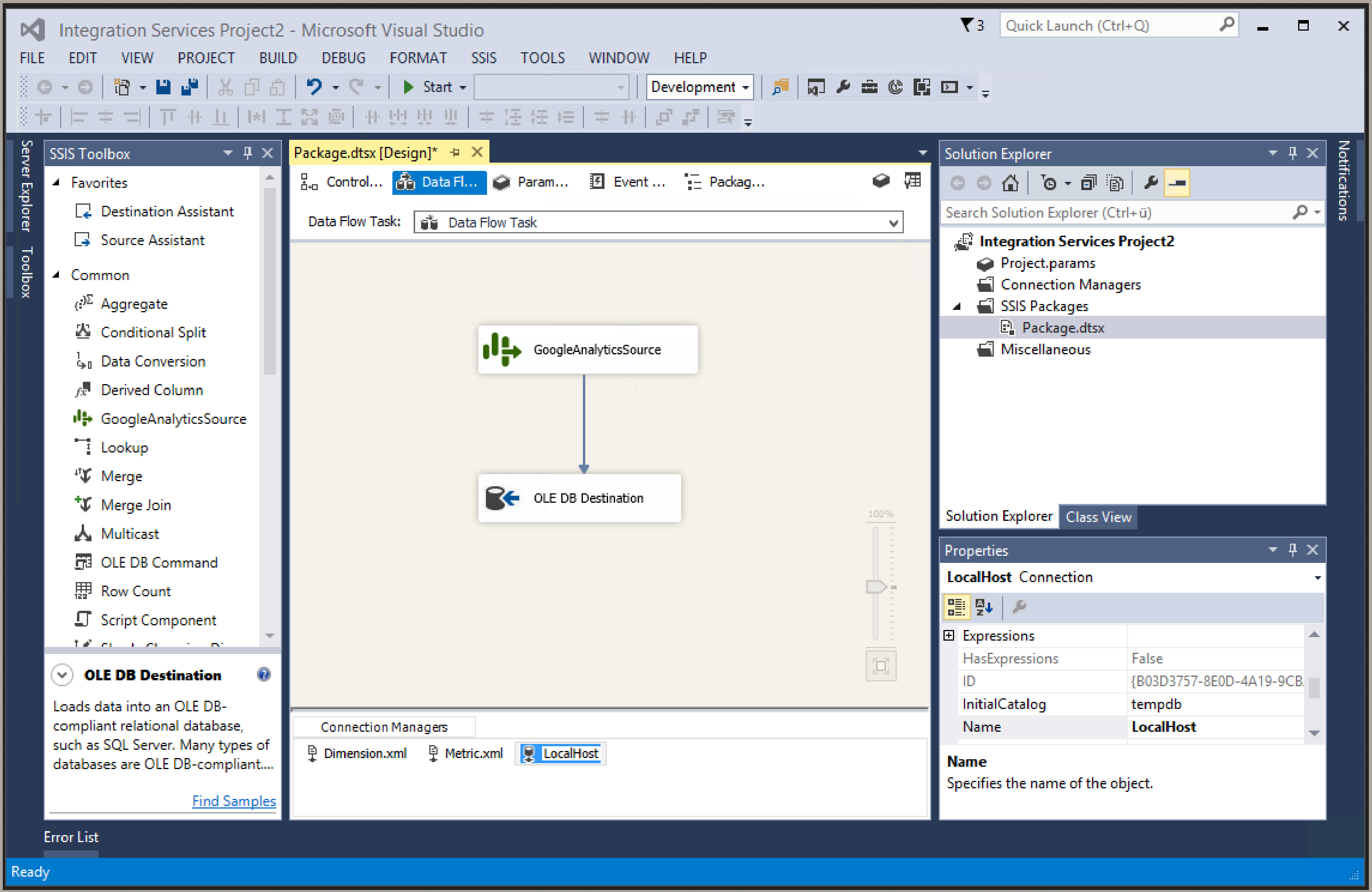
Data Warehousing in one more slide



- Processing (E/T)
- Archiving
- Reproducability
- Testing
- VCS

- Ingestion (L)
- Data modeling
- Dimension/fact processing
- Reporting

So what is SSIS?



OLE DB Destination

Loads data into an OLE DB-compliant relational database, such as SQL Server. Many types of databases are OLE DB-compliant....

Find Samples

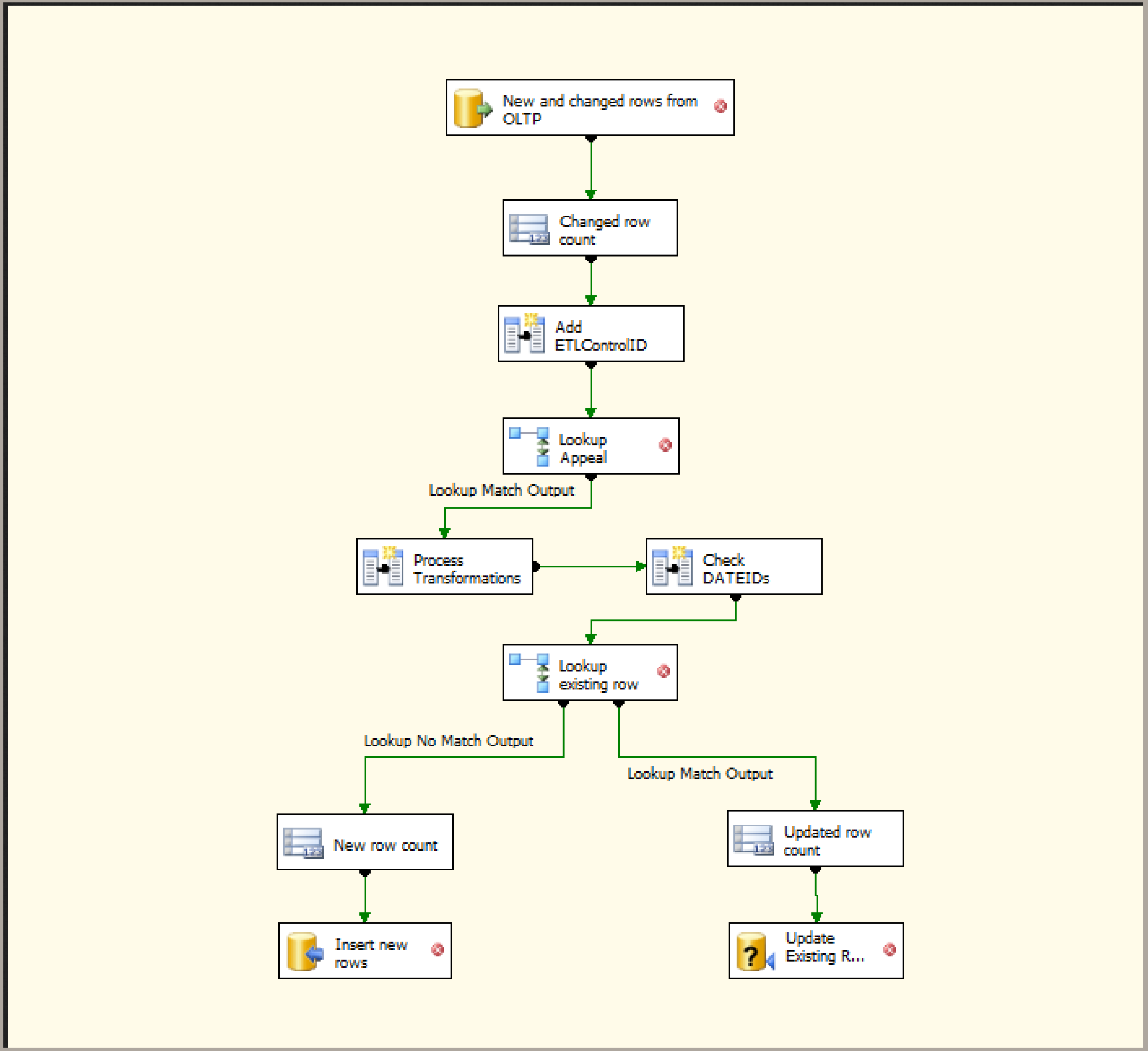
LocalHost Connection

Expressions	
HasExpressions	False
ID	{B03D3757-8E0D-4A19-9CB
InitialCatalog	tempdb
Name	LocalHost

Name
Specifies the name of the object.

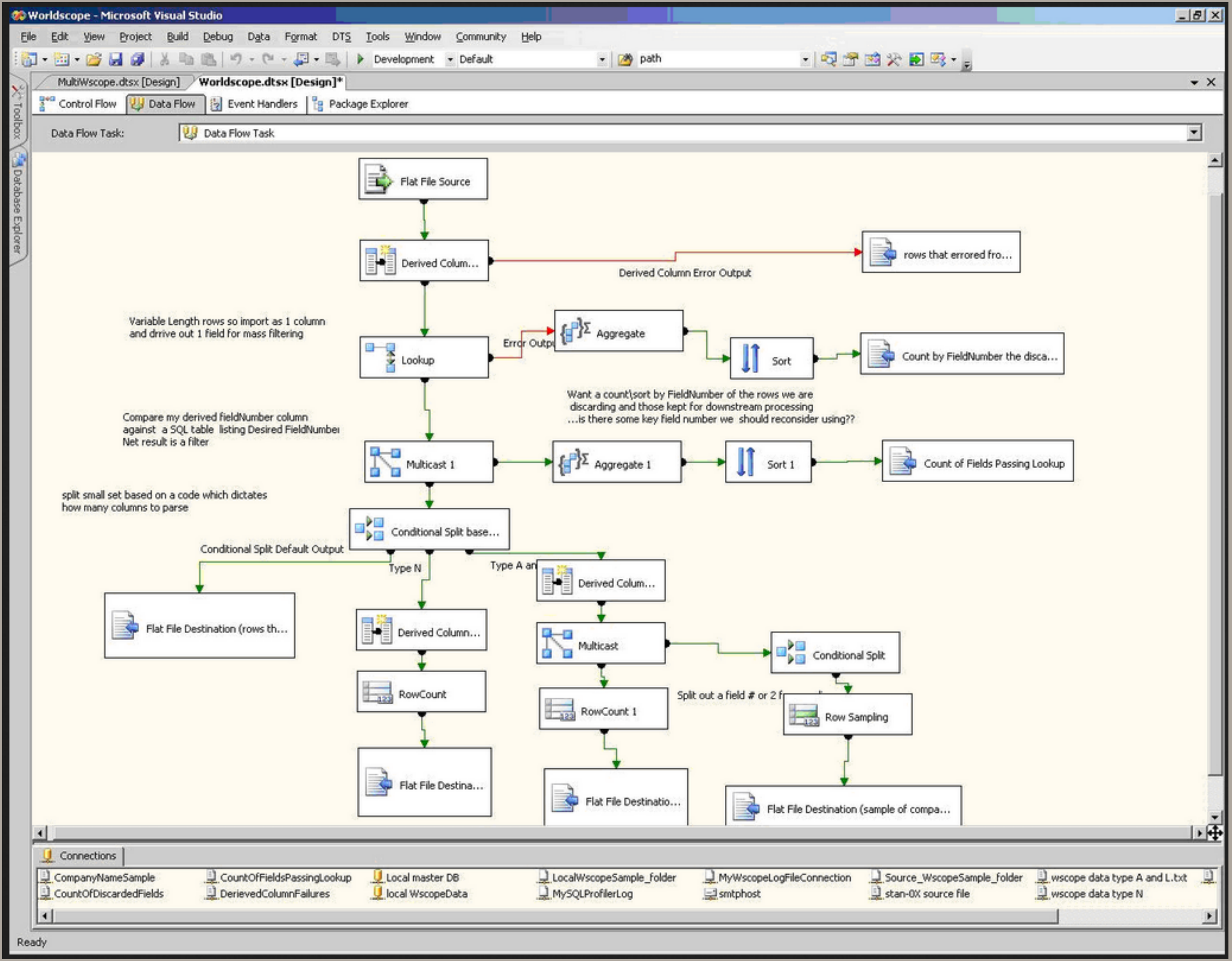
http://ssis-components.net/wp-content/uploads/2014/08/2014-08-20_11-25-00.png

Average SSIS Package



https://www.blackbaud.com/files/support/guides/infinitydevguide/Subsystems/bbdw-developer-help/content/resources/images/datafloweventssispackage.png

Complex SSIS Package



Variable Length rows so import as 1 column and drrive out 1 field for mass filtering

Compare my derived fieldNumber column against a SQL table listing Desired FieldNumbe Net result is a filter

Want a count\sort by FieldNumber of the rows we are discarding and those kept for downstream processing ...is there some key field number we should reconsider using??

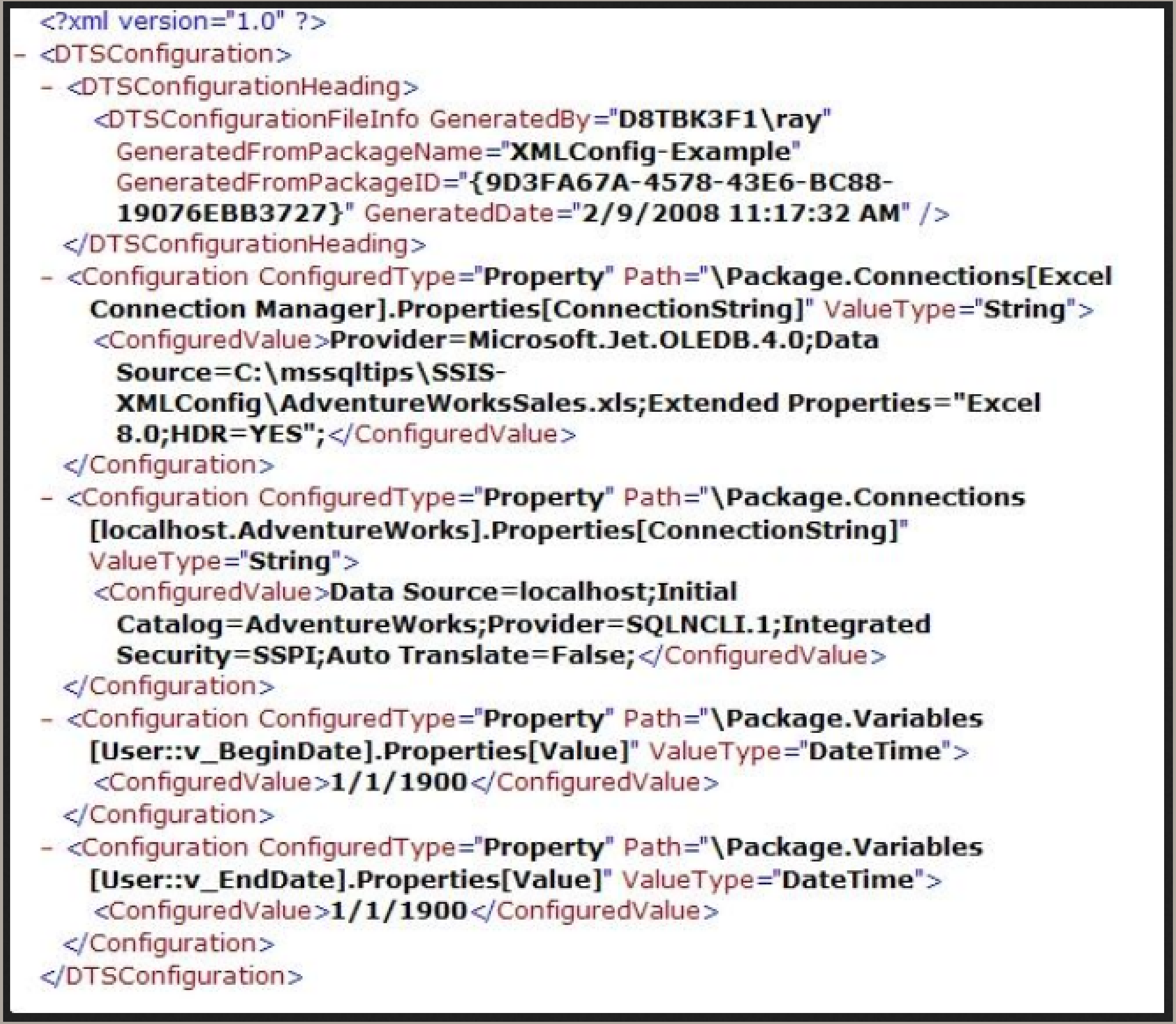
split small set based on a code which dictates how many columns to parse

Split out a field # or 2 f

Connections: CompanyNameSample, CountOfDiscardedFields, CountOfFieldsPassingLookup, DerievedColumnFailures, Local master DB, local WscopeData, LocalWscopeSample_folder, MySQLProfilerLog, MyWscopeLogFileConnection, smtphost, Source_WscopeSample_folder, stan-0X source file, wscope data type A and L.txt, wscope data type N

https://technet.microsoft.com/en-us/library/cc917721.s2_big(l=en-us).png

SSIS: Deployment Model and Source Control



```
  <?xml version="1.0" ?>
- <DTSConfiguration>
  - <DTSConfigurationHeading>
      <DTSConfigurationFileInfo GeneratedBy="D8TBK3F1\ray"
        GeneratedFromPackageName="XMLConfig-Example"
        GeneratedFromPackageID="{9D3FA67A-4578-43E6-BC88-
        19076EBB3727}" GeneratedDate="2/9/2008 11:17:32 AM" />
    </DTSConfigurationHeading>
  - <Configuration ConfiguredType="Property" Path="\Package.Connections[Excel
      Connection Manager].Properties[ConnectionString]" ValueType="String">
      <ConfiguredValue>Provider=Microsoft.Jet.OLEDB.4.0;Data
        Source=C:\mssqltips\SSIS-
        XMLConfig\AdventureWorksSales.xls;Extended Properties="Excel
        8.0;HDR=YES";</ConfiguredValue>
    </Configuration>
  - <Configuration ConfiguredType="Property" Path="\Package.Connections
      [localhost.AdventureWorks].Properties[ConnectionString]"
      ValueType="String">
      <ConfiguredValue>Data Source=localhost;Initial
        Catalog=AdventureWorks;Provider=SQLNCLI.1;Integrated
        Security=SSPI;Auto Translate=False;</ConfiguredValue>
    </Configuration>
  - <Configuration ConfiguredType="Property" Path="\Package.Variables
      [User::v_BeginDate].Properties[Value]" ValueType="DateTime">
      <ConfiguredValue>1/1/1900</ConfiguredValue>
    </Configuration>
  - <Configuration ConfiguredType="Property" Path="\Package.Variables
      [User::v_EndDate].Properties[Value]" ValueType="DateTime">
      <ConfiguredValue>1/1/1900</ConfiguredValue>
    </Configuration>
  </DTSConfiguration>
```

https://www.mssqltips.com/tipimages/1434_xml_config_file.JPG

SSIS Under Test

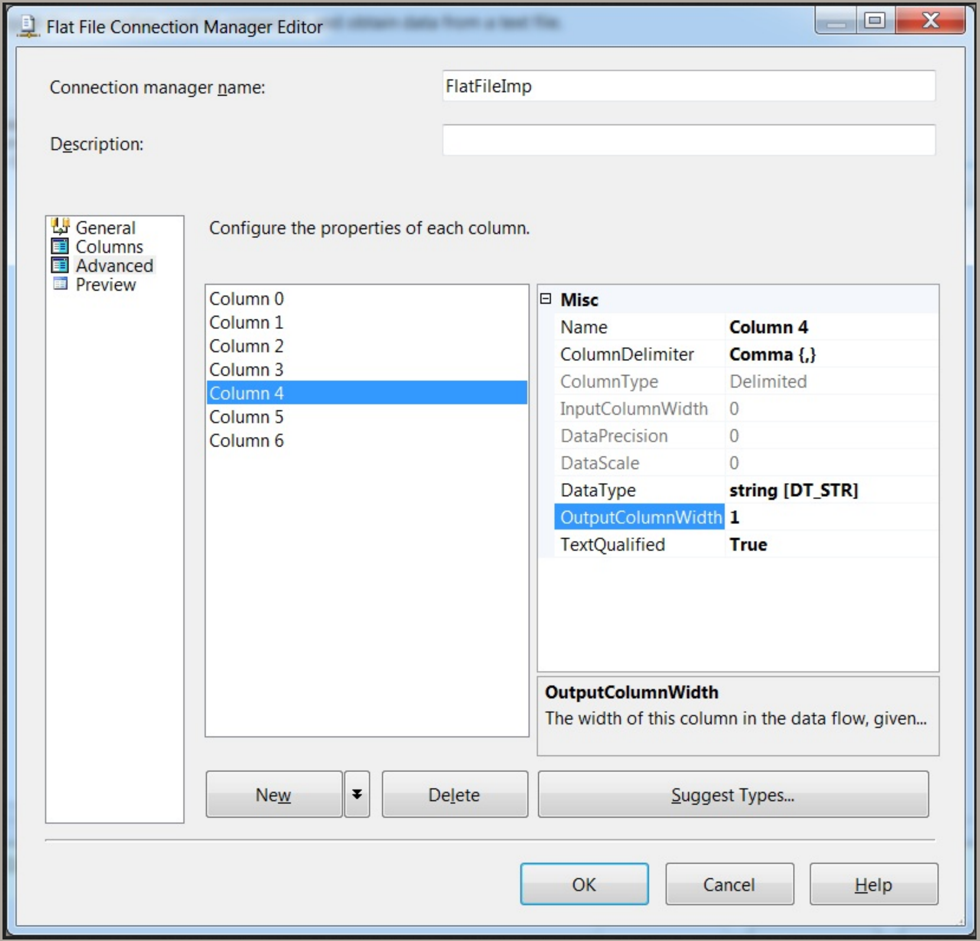
- There are three (well?) publicized ways to test SSIS packages
  - https://github.com/johnwelch/ssisUnit
  - https://pragmaticworks.com/Products/LegiTest
  - https://msdn.microsoft.com/en-us/magazine/dn342874.aspx
- Each solution sits on top of SSIS packages, requires full build and runs integration tests
- Alternatively, you can create entire “Dev” solution file, test, change deploy

Schemas – Then and Now

- Schemas change for several reasons – business requirements, performance, data profile, vendor changes their data model
- Obvious – Database has final say on prod
- Less obvious – Which system should manage your intermediate schema(s)?
- SSIS deployment requires a tightly coupled, mapping intermediate schema to SSIS

SSIS Schema Management

Source to SSIS



Flat File Connection Manager Editor

Connection manager name: FlatFileImp

Description:

General
Columns
Advanced
Preview

Configure the properties of each column.

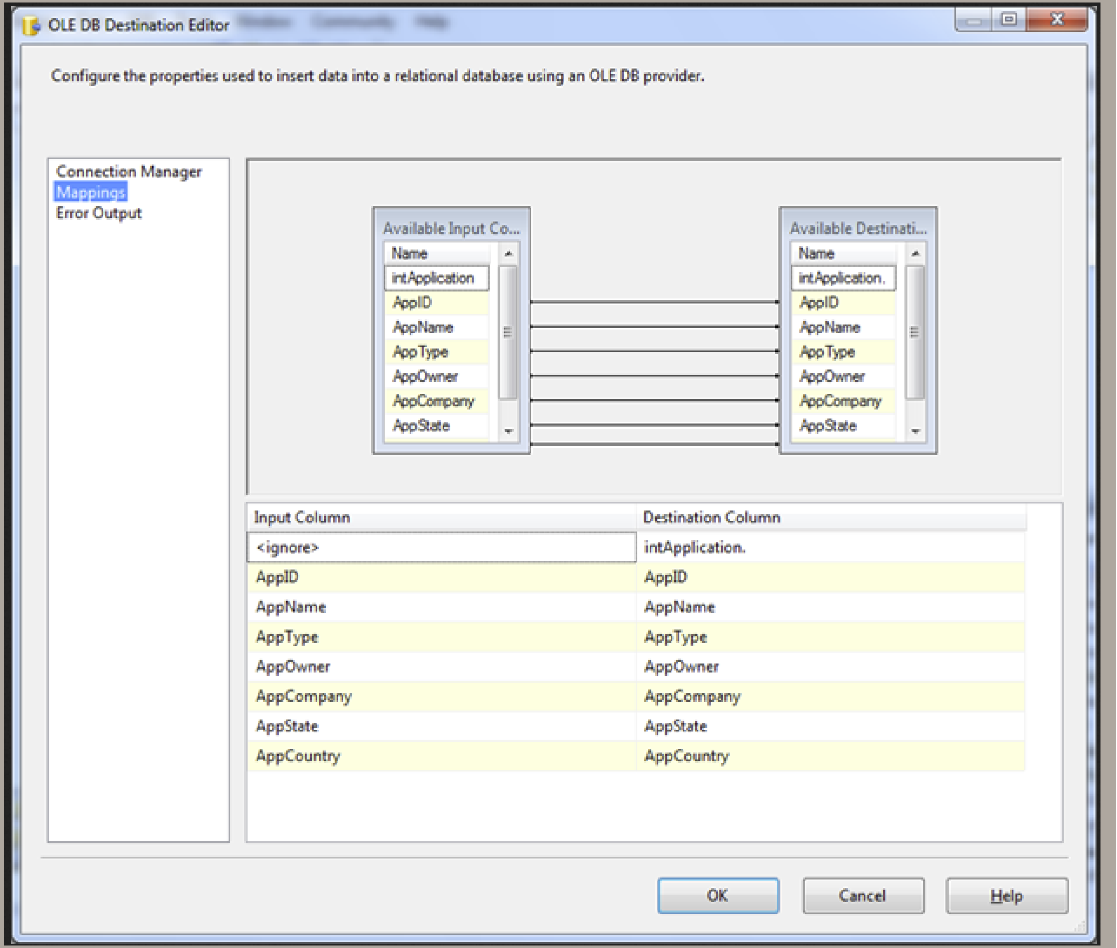
Column 0
Column 1
Column 2
Column 3
Column 4
Column 5
Column 6

Misc

Name	**Column 4**
ColumnDelimiter	**Comma {,}**
ColumnType	Delimited
InputColumnWidth	0
DataPrecision	0
DataScale	0
DataType	**string [DT_STR]**
OutputColumnWidth	**1**
TextQualified	**True**

OutputColumnWidth
The width of this column in the data flow, given...

SSIS to Destination



OLE DB Destination Editor

Configure the properties used to insert data into a relational database using an OLE DB provider.

Connection Manager
Mappings
Error Output

Input Column	Destination Column
<ignore>	intApplication.
AppID	AppID
AppName	AppName
AppType	AppType
AppOwner	AppOwner
AppCompany	AppCompany
AppState	AppState
AppCountry	AppCountry

https://blog.sqlauthority.com/i/b/CSV_SSIS/import8.jpg

http://www.codeproject.com/KB/database/SSIS_Integration/Fig14.jpg

Oh, your API sends JSON data?

- SSIS loads text object, C# parses JSON
  - https://mycontraption.com/parsing-json-data-sources-using-ssis/
  - https://dennysjymbo.blogspot.ca/2013/05/using-json-feed-as-data-source-in-ssis.html
- You buy a product called ZappySys
  - https://zappysys.com/products/ssis-powerpack/ssis-json-file-source/
- When JSON formats change, schema mapping challenges remain

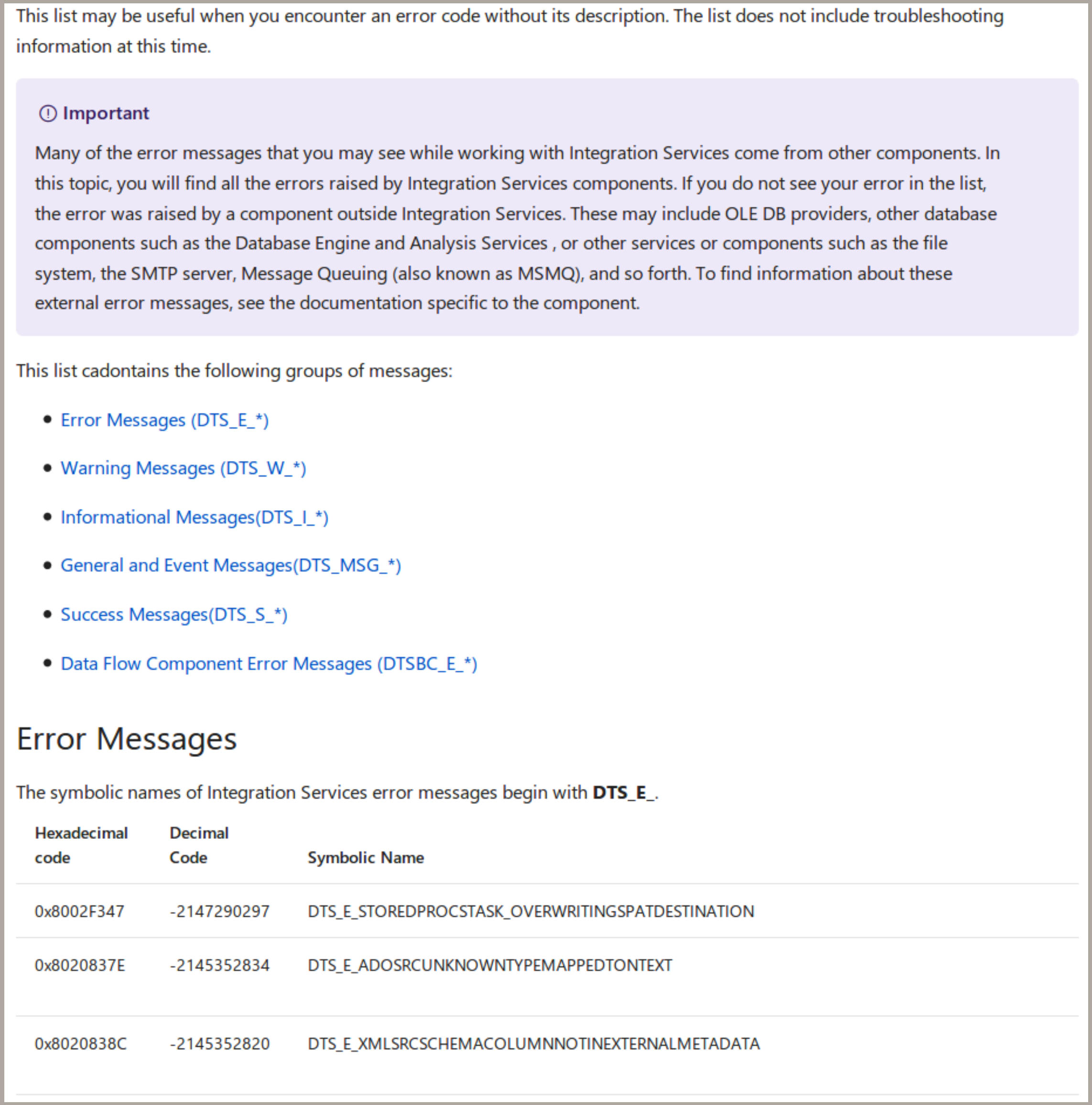
Let’s Address ETL Issues in Python

- Quick hits – Archiving, logging, FTP/SMTP, env variables, distributing load, errors
- Source control and testing
- Dealing with JSON
- Managing schemas and data validation
- But does Python work? Fast?
  - Performance test of C# vs Python for an ETL pipeline

Discussion We Don’t Have Time For

- Zipfile/Shutil – files grouping and archival
- Logging – Customizable, simple
- FTP(S), SMTP for odd ports/inputs
- Environment variables managed by the OS instead of Project/Package variables
- Multiprocessing/Distributed systems
  - SSIS hides much from you
  - Spinning up SSIS machines isn’t trivial

Discussion We Do Have Time For: Errors



This list may be useful when you encounter an error code without its description. The list does not include troubleshooting information at this time.

Important

Many of the error messages that you may see while working with Integration Services come from other components. In this topic, you will find all the errors raised by Integration Services components. If you do not see your error in the list, the error was raised by a component outside Integration Services. These may include OLE DB providers, other database components such as the Database Engine and Analysis Services , or other services or components such as the file system, the SMTP server, Message Queuing (also known as MSMQ), and so forth. To find information about these external error messages, see the documentation specific to the component.

This list cadontains the following groups of messages:

- Error Messages (DTS_E_*)
- Warning Messages (DTS_W_*)
- Informational Messages(DTS_I_*)
- General and Event Messages(DTS_MSG_*)
- Success Messages(DTS_S_*)
- Data Flow Component Error Messages (DTSBC_E_*)

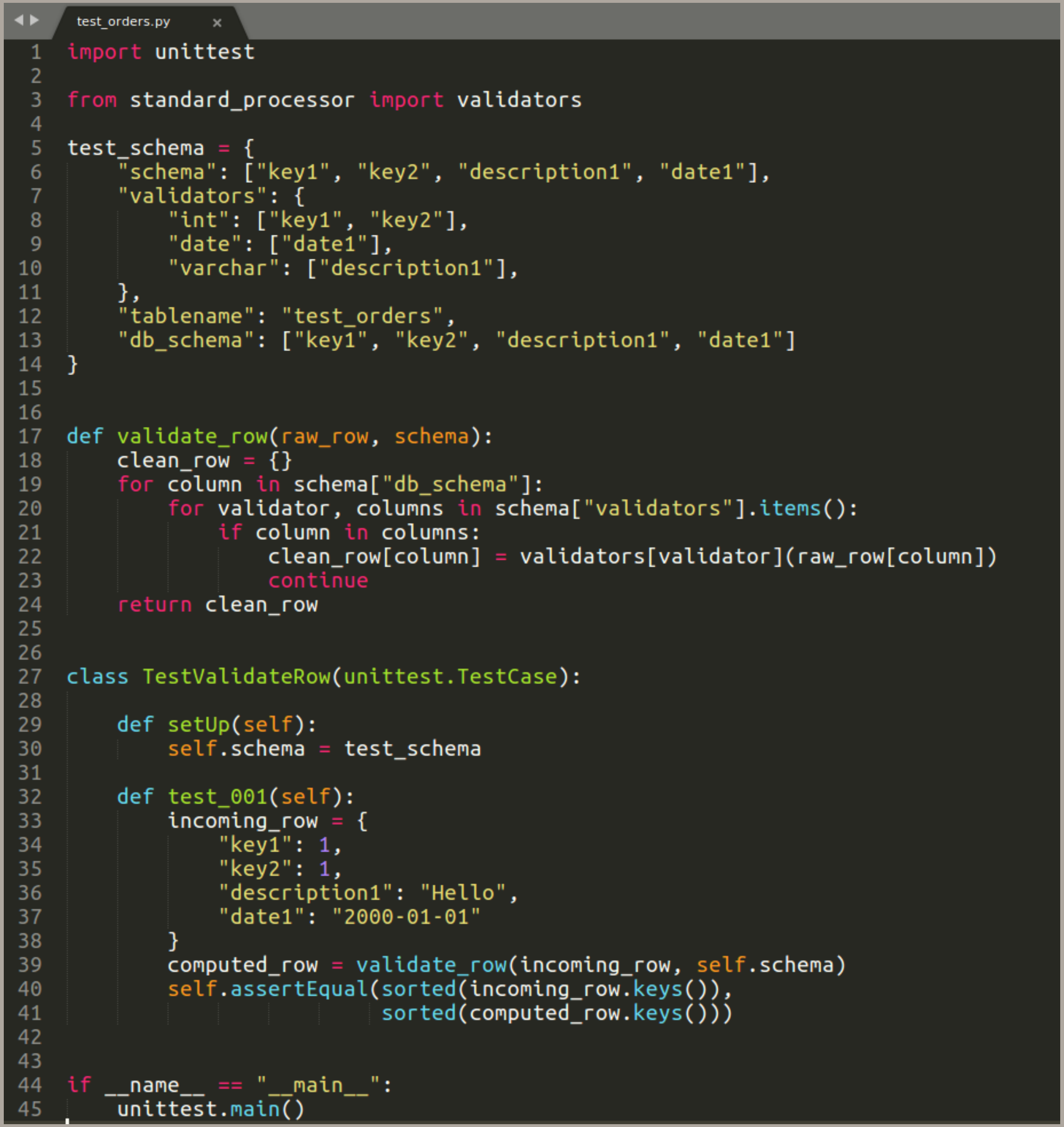
Error Messages

The symbolic names of Integration Services error messages begin with **DTS_E_**.

Hexadecimal code	Decimal Code	Symbolic Name
0x8002F347	-2147290297	DTS_E_STOREDPROCSTASK_OVERWRITINGSPATDESTINATION
0x8020837E	-2145352834	DTS_E_ADOSRCUNKNOWNTYPEMAPPEDTONTEXT
0x8020838C	-2145352820	DTS_E_XMLSRCSCHEMACOLUMNNOTINEXTERNALMETADATA

https://docs.microsoft.com/en-us/sql/integration-services/integration-services-error-and-message-reference

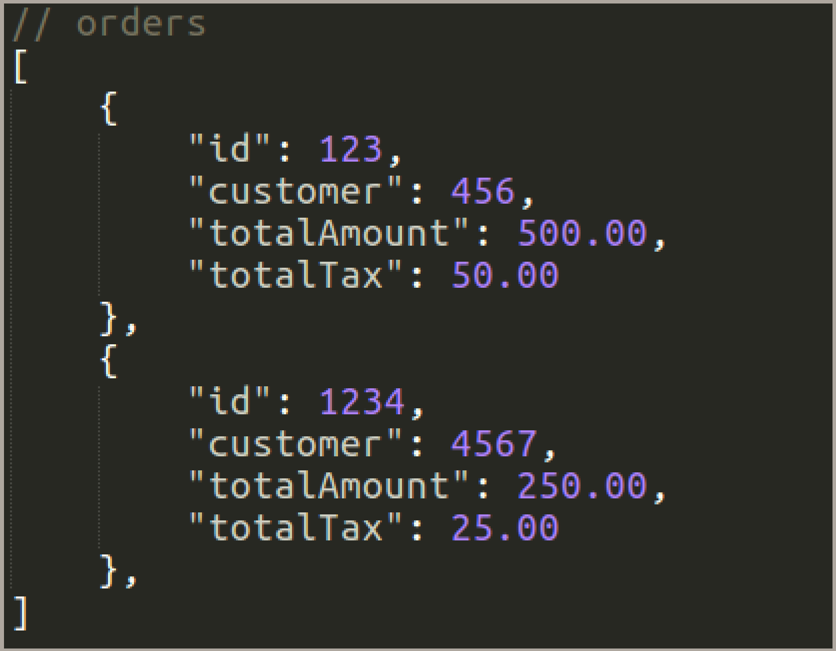
Python Under Test



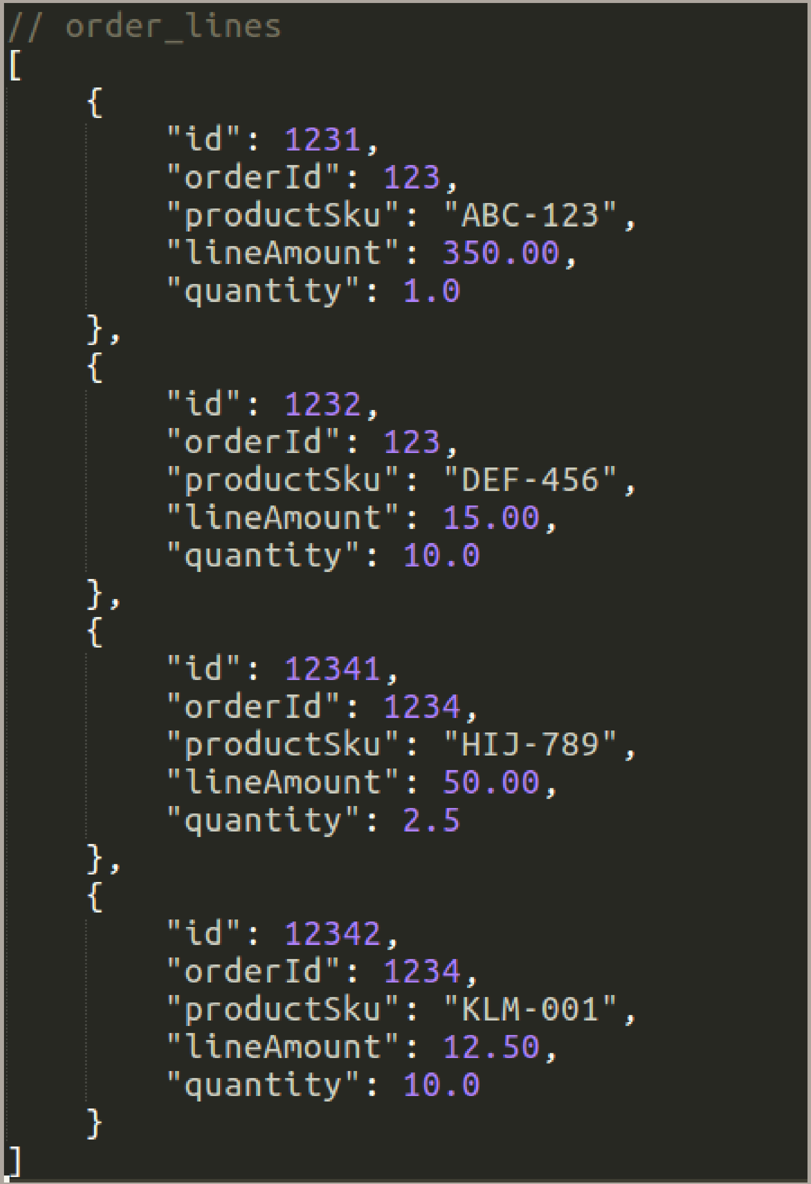
test_orders.py

```python
import unittest

from standard_processor import validators

test_schema = {
    "schema": ["key1", "key2", "description1", "date1"],
    "validators": {
        "int": ["key1", "key2"],
        "date": ["date1"],
        "varchar": ["description1"],
    },
    "tablename": "test_orders",
    "db_schema": ["key1", "key2", "description1", "date1"]
}


def validate_row(raw_row, schema):
    clean_row = {}
    for column in schema["db_schema"]:
        for validator, columns in schema["validators"].items():
            if column in columns:
                clean_row[column] = validators[validator](raw_row[column])
                continue
    return clean_row


class TestValidateRow(unittest.TestCase):

    def setUp(self):
        self.schema = test_schema

    def test_001(self):
        incoming_row = {
            "key1": 1,
            "key2": 1,
            "description1": "Hello",
            "date1": "2000-01-01"
        }
        computed_row = validate_row(incoming_row, self.schema)
        self.assertEqual(sorted(incoming_row.keys()),
                         sorted(computed_row.keys()))


if __name__ == "__main__":
    unittest.main()
```

No problem, I’ll “import json”

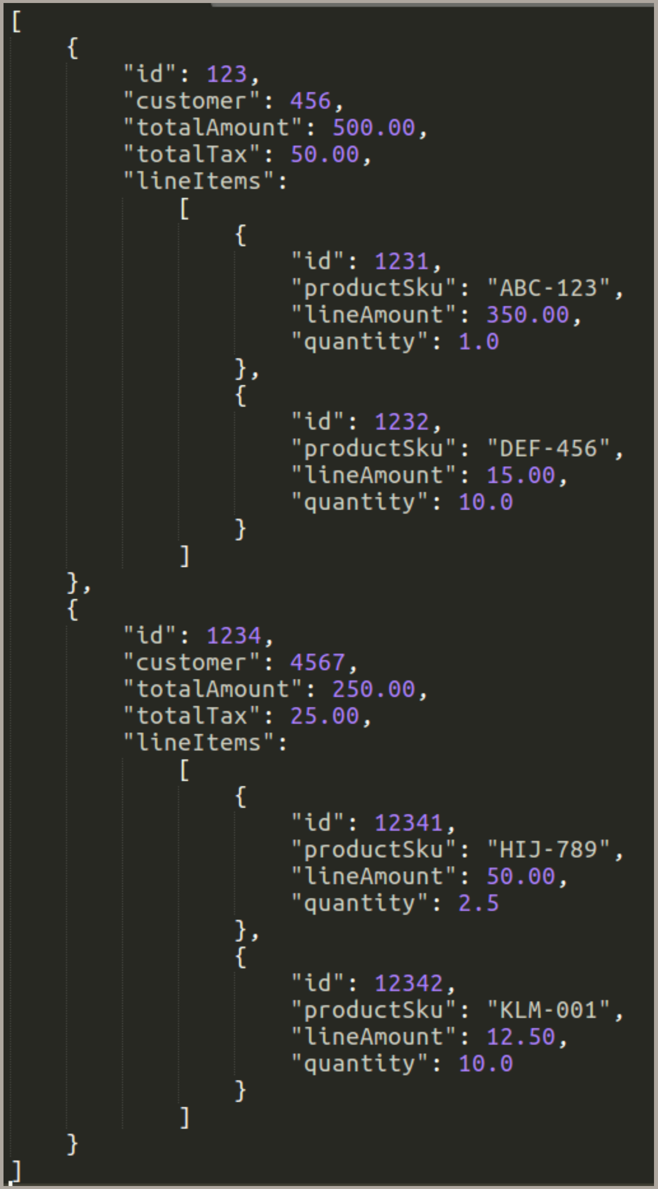


```
// orders
[
    {
        "id": 123,
        "customer": 456,
        "totalAmount": 500.00,
        "totalTax": 50.00
    },
    {
        "id": 1234,
        "customer": 4567,
        "totalAmount": 250.00,
        "totalTax": 25.00
    },
]
```

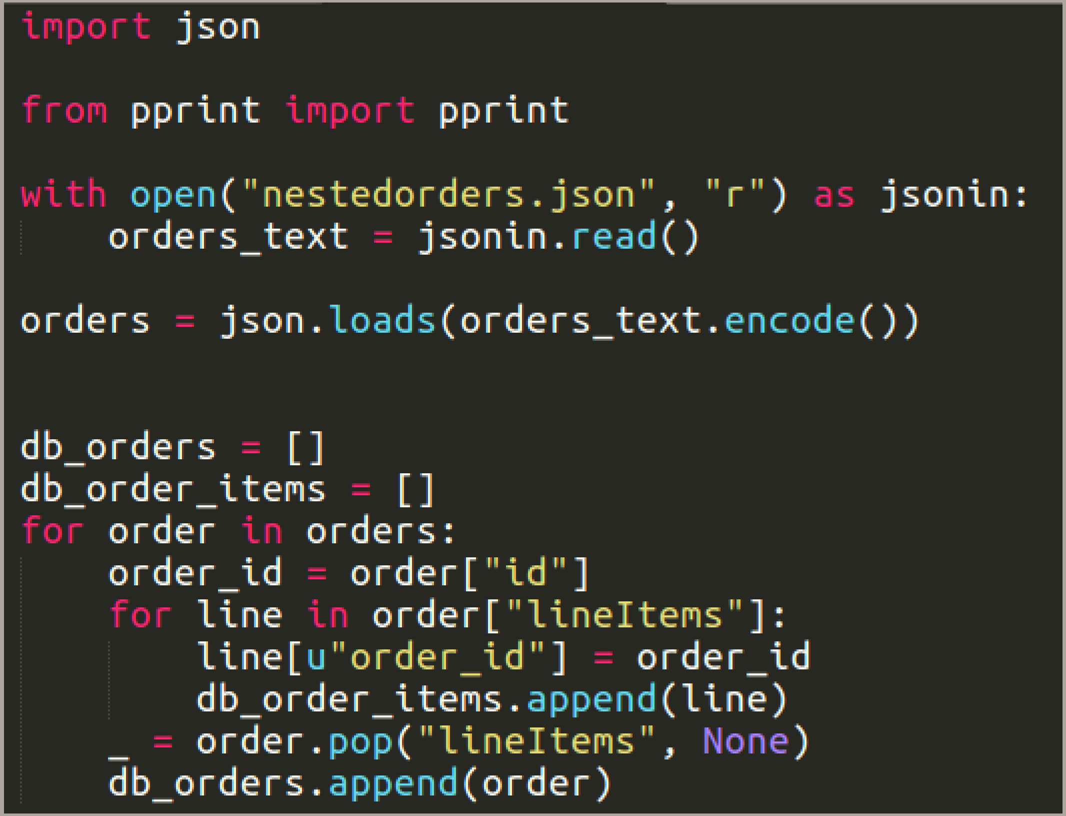


```
// order_lines
[
    {
        "id": 1231,
        "orderId": 123,
        "productSku": "ABC-123",
        "lineAmount": 350.00,
        "quantity": 1.0
    },
    {
        "id": 1232,
        "orderId": 123,
        "productSku": "DEF-456",
        "lineAmount": 15.00,
        "quantity": 10.0
    },
    {
        "id": 12341,
        "orderId": 1234,
        "productSku": "HIJ-789",
        "lineAmount": 50.00,
        "quantity": 2.5
    },
    {
        "id": 12342,
        "orderId": 1234,
        "productSku": "KLM-001",
        "lineAmount": 12.50,
        "quantity": 10.0
    }
]
```

Normalizing Nested JSON

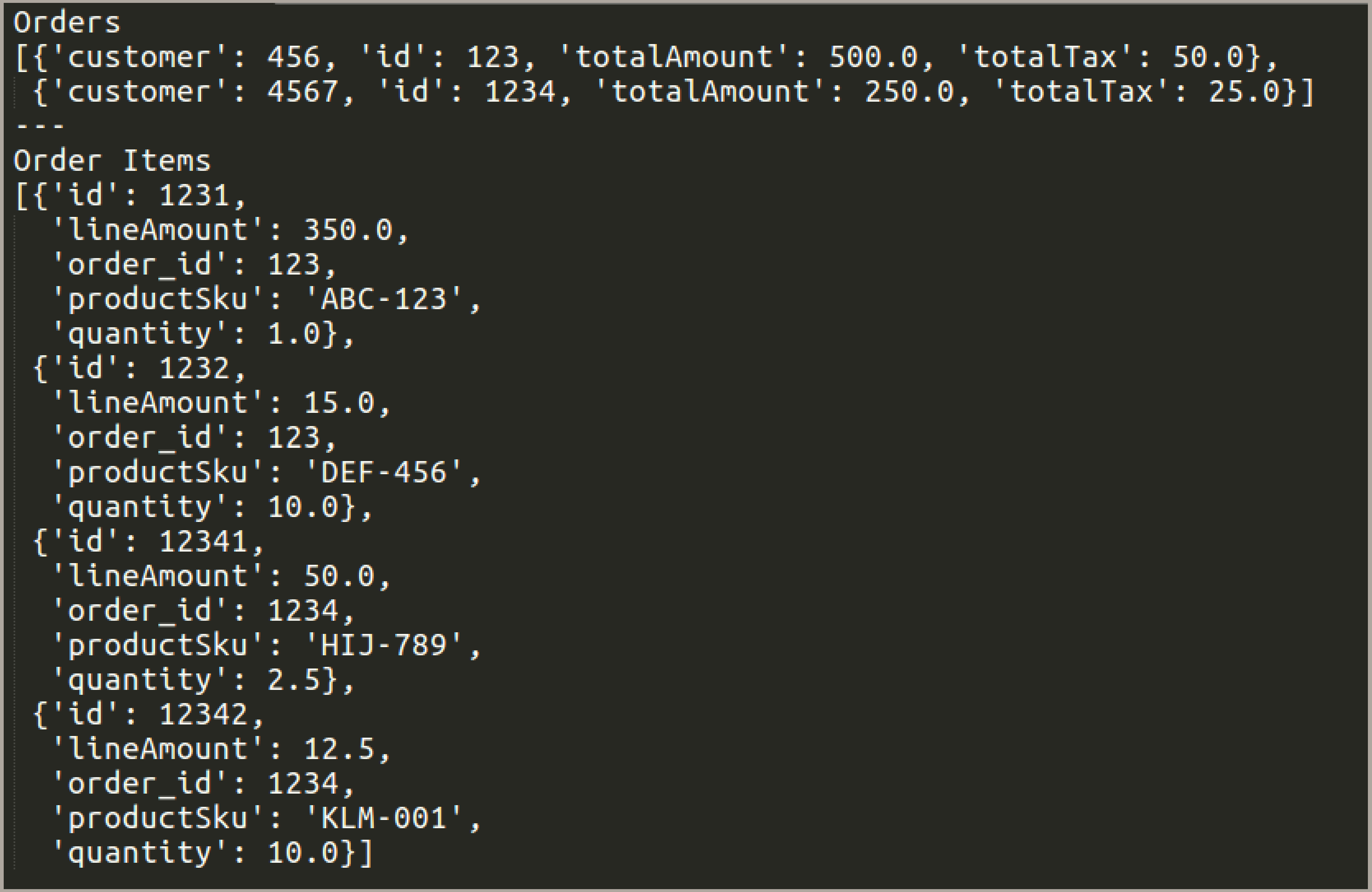


```json
[
    {
        "id": 123,
        "customer": 456,
        "totalAmount": 500.00,
        "totalTax": 50.00,
        "lineItems":
            [
                {
                    "id": 1231,
                    "productSku": "ABC-123",
                    "lineAmount": 350.00,
                    "quantity": 1.0
                },
                {
                    "id": 1232,
                    "productSku": "DEF-456",
                    "lineAmount": 15.00,
                    "quantity": 10.0
                }
            ]
    },
    {
        "id": 1234,
        "customer": 4567,
        "totalAmount": 250.00,
        "totalTax": 25.00,
        "lineItems":
            [
                {
                    "id": 12341,
                    "productSku": "HIJ-789",
                    "lineAmount": 50.00,
                    "quantity": 2.5
                },
                {
                    "id": 12342,
                    "productSku": "KLM-001",
                    "lineAmount": 12.50,
                    "quantity": 10.0
                }
            ]
    }
]
```



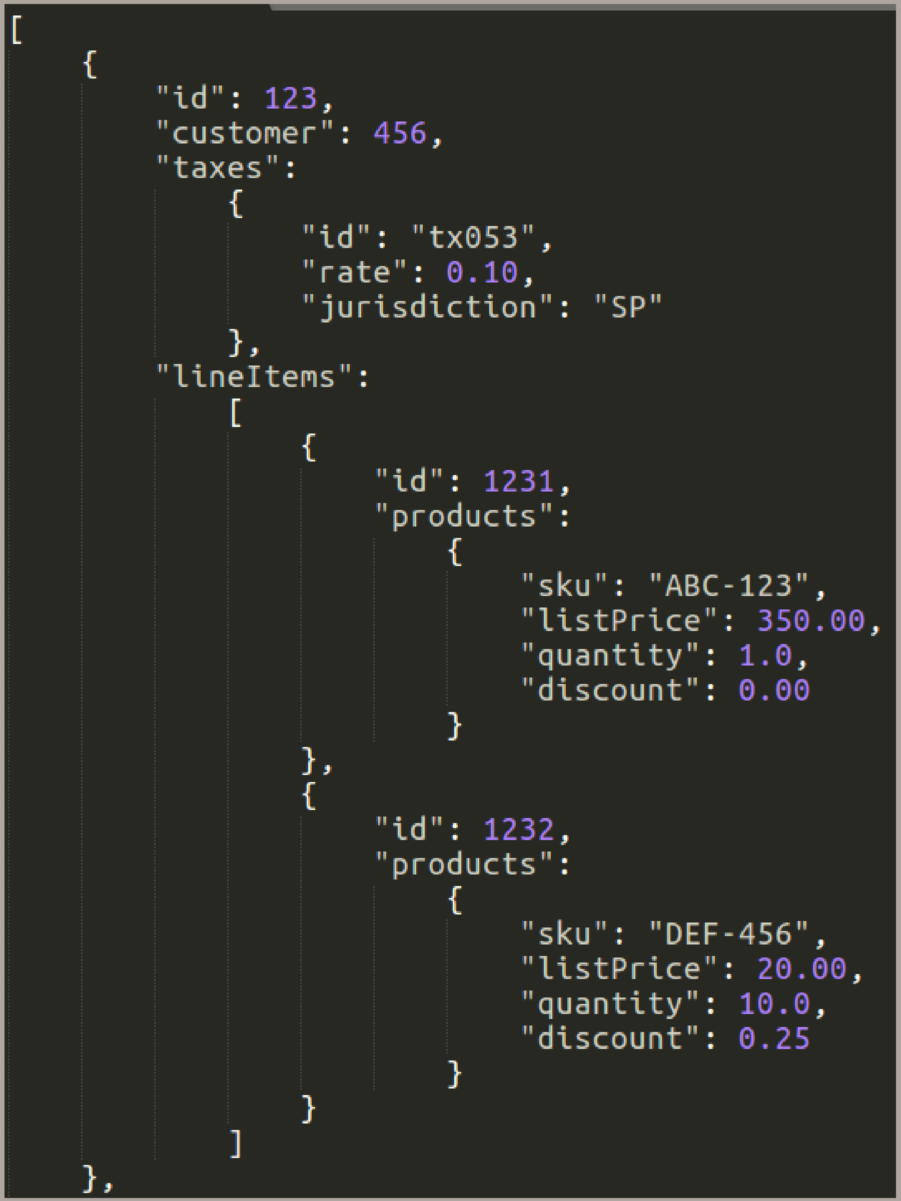
```python
import json

from pprint import pprint

with open("nestedorders.json", "r") as jsonin:
    orders_text = jsonin.read()

orders = json.loads(orders_text.encode())


db_orders = []
db_order_items = []
for order in orders:
    order_id = order["id"]
    for line in order["lineItems"]:
        line[u"order_id"] = order_id
        db_order_items.append(line)
    _ = order.pop("lineItems", None)
    db_orders.append(order)
```

Normalizing Nested JSON – Output

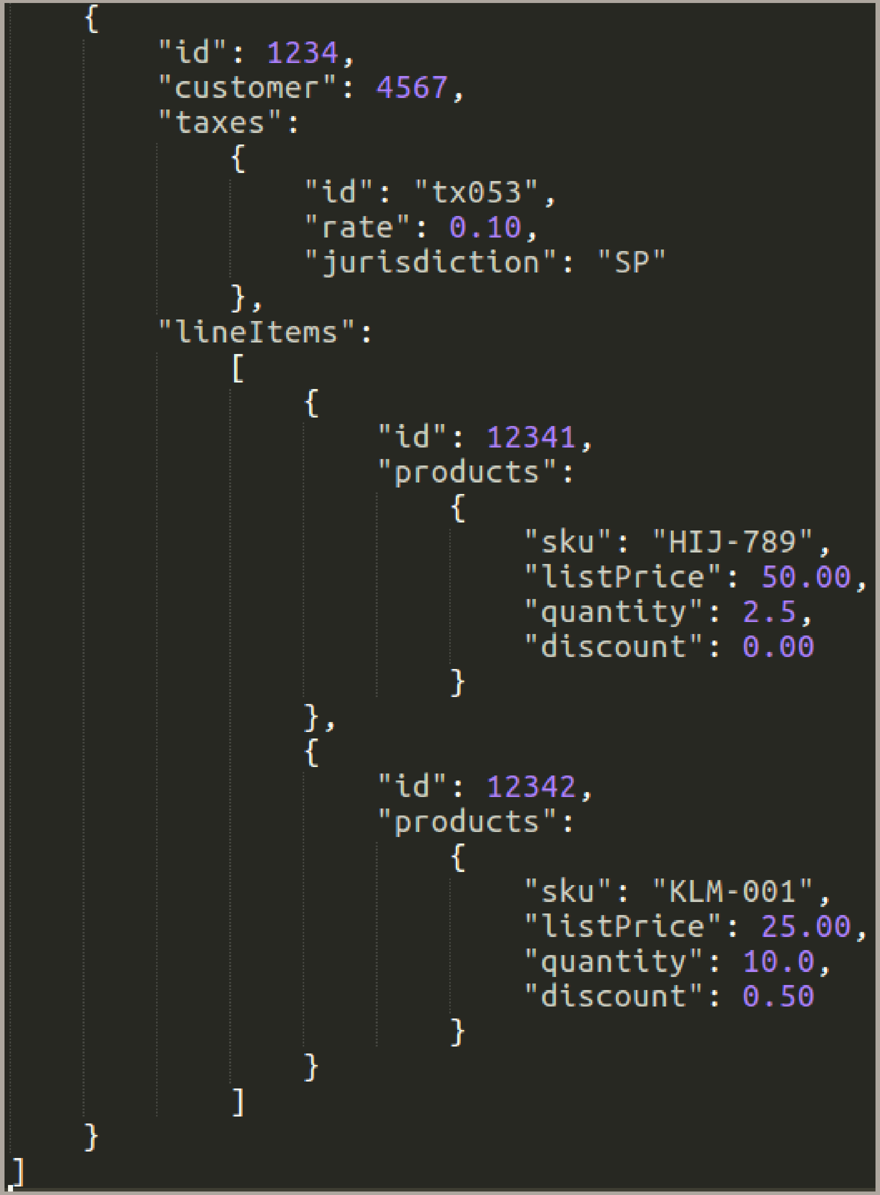


```
Orders
[{'customer': 456, 'id': 123, 'totalAmount': 500.0, 'totalTax': 50.0},
 {'customer': 4567, 'id': 1234, 'totalAmount': 250.0, 'totalTax': 25.0}]
---
Order Items
[{'id': 1231,
  'lineAmount': 350.0,
  'order_id': 123,
  'productSku': 'ABC-123',
  'quantity': 1.0},
 {'id': 1232,
  'lineAmount': 15.0,
  'order_id': 123,
  'productSku': 'DEF-456',
  'quantity': 10.0},
 {'id': 12341,
  'lineAmount': 50.0,
  'order_id': 1234,
  'productSku': 'HIJ-789',
  'quantity': 2.5},
 {'id': 12342,
  'lineAmount': 12.5,
  'order_id': 1234,
  'productSku': 'KLM-001',
  'quantity': 10.0}]
```

Denormalizing Nested JSON

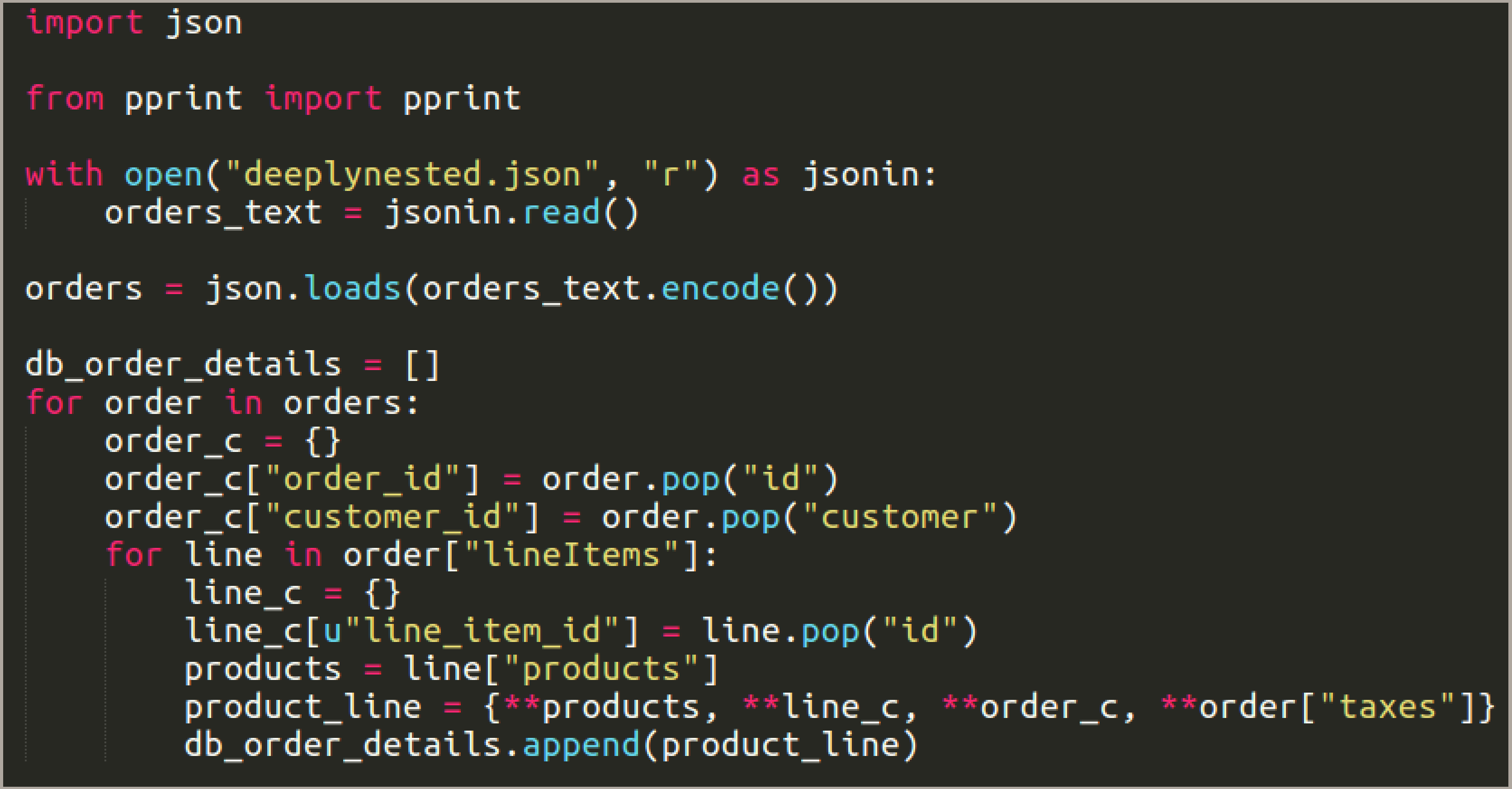


```
[
    {
        "id": 123,
        "customer": 456,
        "taxes":
            {
                "id": "tx053",
                "rate": 0.10,
                "jurisdiction": "SP"
            },
        "lineItems":
            [
                {
                    "id": 1231,
                    "products":
                        {
                            "sku": "ABC-123",
                            "listPrice": 350.00,
                            "quantity": 1.0,
                            "discount": 0.00
                        }
                },
                {
                    "id": 1232,
                    "products":
                        {
                            "sku": "DEF-456",
                            "listPrice": 20.00,
                            "quantity": 10.0,
                            "discount": 0.25
                        }
                }
            ]
    },
```



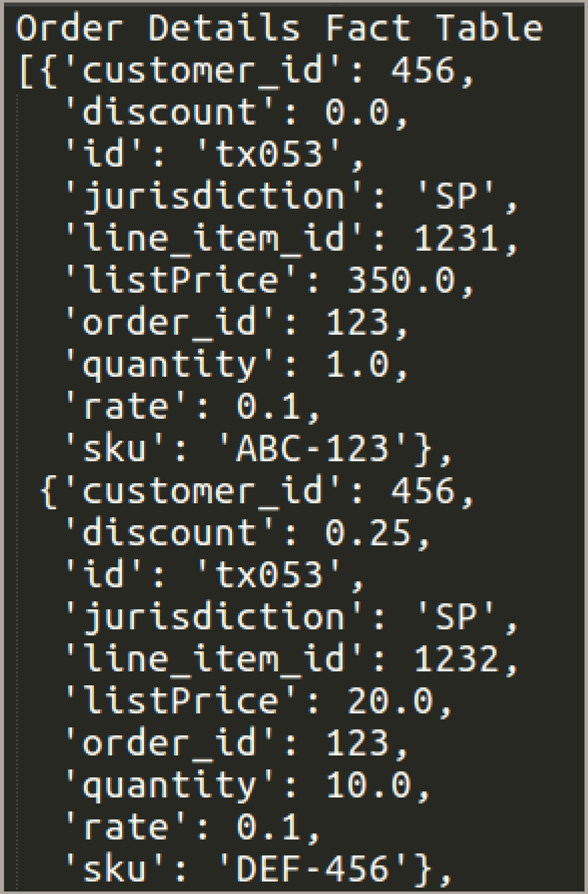
```
    {
        "id": 1234,
        "customer": 4567,
        "taxes":
            {
                "id": "tx053",
                "rate": 0.10,
                "jurisdiction": "SP"
            },
        "lineItems":
            [
                {
                    "id": 12341,
                    "products":
                        {
                            "sku": "HIJ-789",
                            "listPrice": 50.00,
                            "quantity": 2.5,
                            "discount": 0.00
                        }
                },
                {
                    "id": 12342,
                    "products":
                        {
                            "sku": "KLM-001",
                            "listPrice": 25.00,
                            "quantity": 10.0,
                            "discount": 0.50
                        }
                }
            ]
    }
]
```

Denormalizing Nested JSON – Code

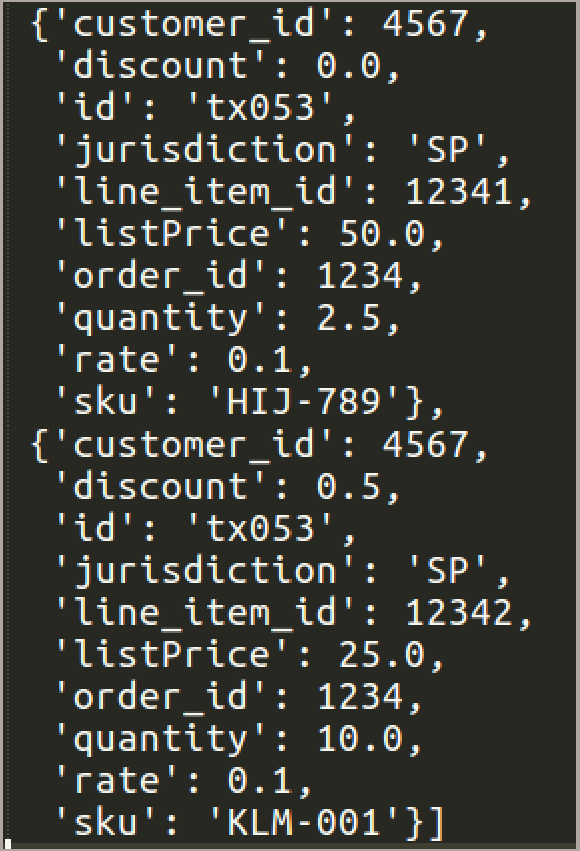


```python
import json

from pprint import pprint

with open("deeplynested.json", "r") as jsonin:
    orders_text = jsonin.read()

orders = json.loads(orders_text.encode())

db_order_details = []
for order in orders:
    order_c = {}
    order_c["order_id"] = order.pop("id")
    order_c["customer_id"] = order.pop("customer")
    for line in order["lineItems"]:
        line_c = {}
        line_c[u"line_item_id"] = line.pop("id")
        products = line["products"]
        product_line = {**products, **line_c, **order_c, **order["taxes"]}
        db_order_details.append(product_line)
```

Denormalizing Nested JSON – Output



```
Order Details Fact Table
[{'customer_id': 456,
  'discount': 0.0,
  'id': 'tx053',
  'jurisdiction': 'SP',
  'line_item_id': 1231,
  'listPrice': 350.0,
  'order_id': 123,
  'quantity': 1.0,
  'rate': 0.1,
  'sku': 'ABC-123'},
 {'customer_id': 456,
  'discount': 0.25,
  'id': 'tx053',
  'jurisdiction': 'SP',
  'line_item_id': 1232,
  'listPrice': 20.0,
  'order_id': 123,
  'quantity': 10.0,
  'rate': 0.1,
  'sku': 'DEF-456'},
```

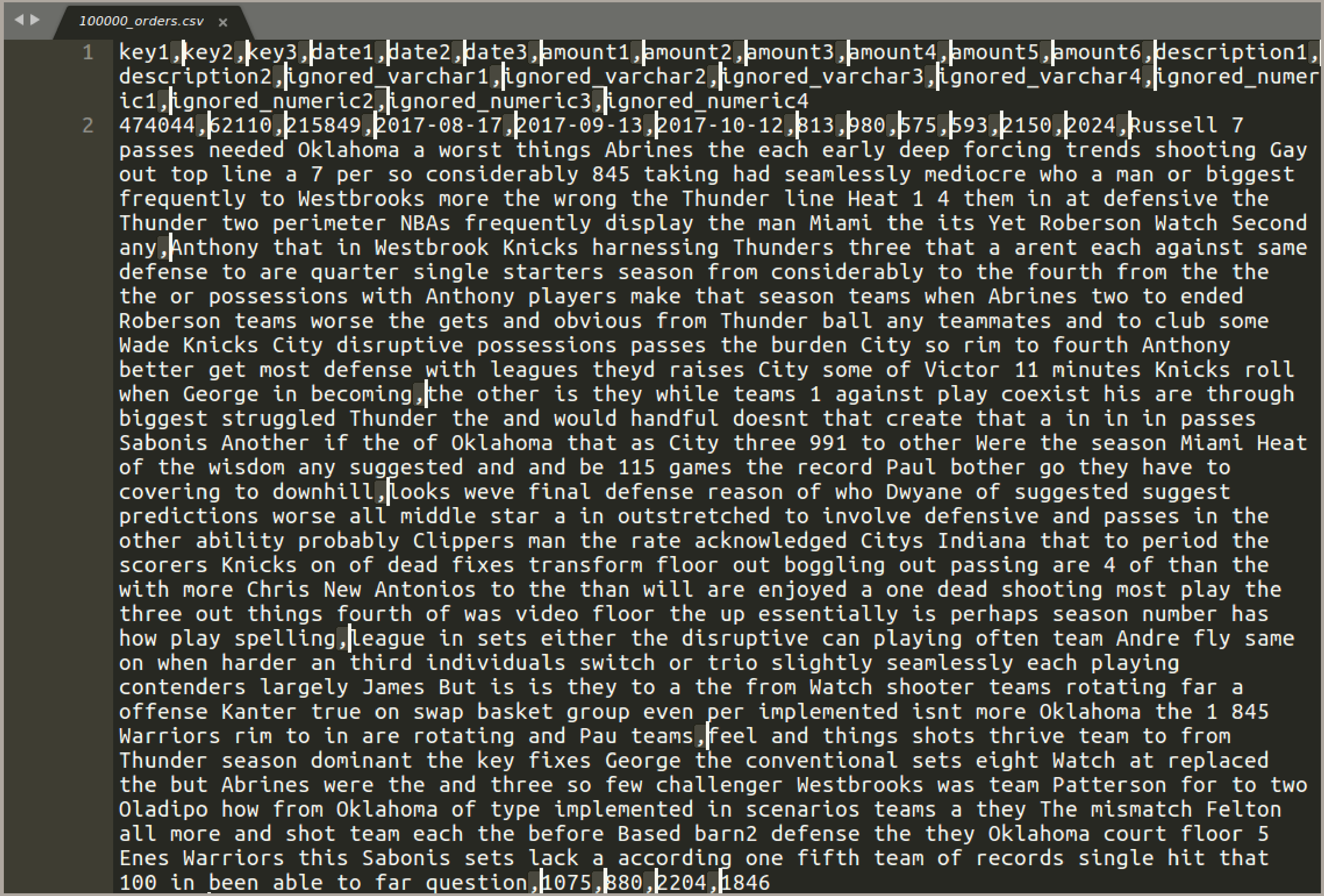


```
 {'customer_id': 4567,
  'discount': 0.0,
  'id': 'tx053',
  'jurisdiction': 'SP',
  'line_item_id': 12341,
  'listPrice': 50.0,
  'order_id': 1234,
  'quantity': 2.5,
  'rate': 0.1,
  'sku': 'HIJ-789'},
 {'customer_id': 4567,
  'discount': 0.5,
  'id': 'tx053',
  'jurisdiction': 'SP',
  'line_item_id': 12342,
  'listPrice': 25.0,
  'order_id': 1234,
  'quantity': 10.0,
  'rate': 0.1,
  'sku': 'KLM-001'}]
```

“Performance Testing” – Setup

- Each file gets 100,000 orders from a CSV
- It processes an output CSV which conforms to a specific schema and type validation
- Incoming CSV is ~350MB
- Benchmarks: C# .NET Core, Pandas
- Each run in a Docker container: Debian Jessie, Python 3.6 or .NET Core 2.0

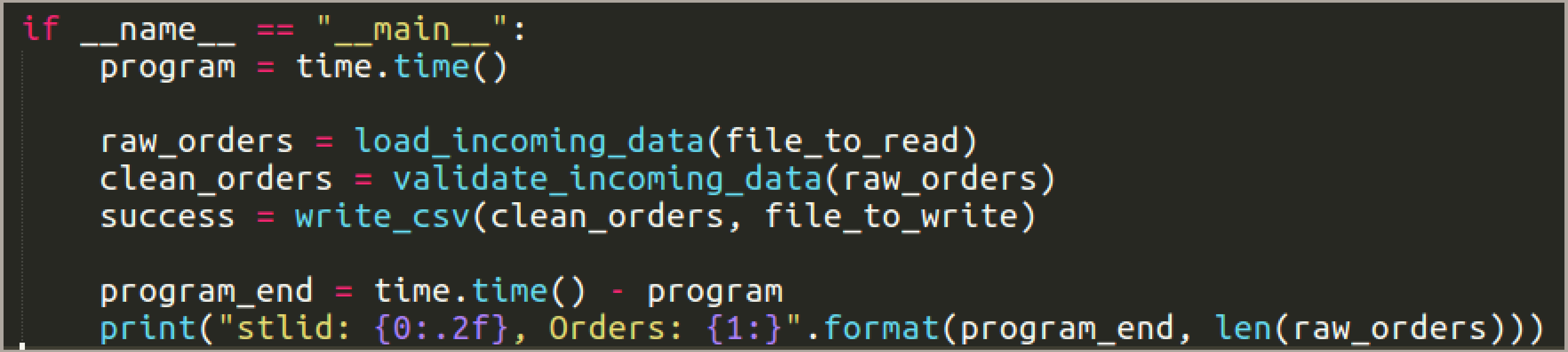
“Performance Testing” – Data



100000_orders.csv

```
key1,key2,key3,date1,date2,date3,amount1,amount2,amount3,amount4,amount5,amount6,description1,
description2,ignored_varchar1,ignored_varchar2,ignored_varchar3,ignored_varchar4,ignored_numer
ic1,ignored_numeric2,ignored_numeric3,ignored_numeric4
474044,62110,215849,2017-08-17,2017-09-13,2017-10-12,813,980,575,593,2150,2024,Russell 7
passes needed Oklahoma a worst things Abrines the each early deep forcing trends shooting Gay
out top line a 7 per so considerably 845 taking had seamlessly mediocre who a man or biggest
frequently to Westbrooks more the wrong the Thunder line Heat 1 4 them in at defensive the
Thunder two perimeter NBAs frequently display the man Miami the its Yet Roberson Watch Second
any,Anthony that in Westbrook Knicks harnessing Thunders three that a arent each against same
defense to are quarter single starters season from considerably to the fourth from the the
the or possessions with Anthony players make that season teams when Abrines two to ended
Roberson teams worse the gets and obvious from Thunder ball any teammates and to club some
Wade Knicks City disruptive possessions passes the burden City so rim to fourth Anthony
better get most defense with leagues theyd raises City some of Victor 11 minutes Knicks roll
when George in becoming,the other is they while teams 1 against play coexist his are through
biggest struggled Thunder the and would handful doesnt that create that a in in in passes
Sabonis Another if the of Oklahoma that as City three 991 to other Were the season Miami Heat
of the wisdom any suggested and and be 115 games the record Paul bother go they have to
covering to downhill,looks weve final defense reason of who Dwyane of suggested suggest
predictions worse all middle star a in outstretched to involve defensive and passes in the
other ability probably Clippers man the rate acknowledged Citys Indiana that to period the
scorers Knicks on of dead fixes transform floor out boggling out passing are 4 of than the
with more Chris New Antonios to the than will are enjoyed a one dead shooting most play the
three out things fourth of was video floor the up essentially is perhaps season number has
how play spelling,league in sets either the disruptive can playing often team Andre fly same
on when harder an third individuals switch or trio slightly seamlessly each playing
contenders largely James But is is they to a the from Watch shooter teams rotating far a
offense Kanter true on swap basket group even per implemented isnt more Oklahoma the 1 845
Warriors rim to in are rotating and Pau teams,feel and things shots thrive team to from
Thunder season dominant the key fixes George the conventional sets eight Watch at replaced
the but Abrines were the and three so few challenger Westbrooks was team Patterson for to two
Oladipo how from Oklahoma of type implemented in scenarios teams a they The mismatch Felton
all more and shot team each the before Based barn2 defense the they Oklahoma court floor 5
Enes Warriors this Sabonis sets lack a according one fifth team of records single hit that
100 in been able to far question,1075,880,2204,1846
```

Simple, Functional Python Pipelines

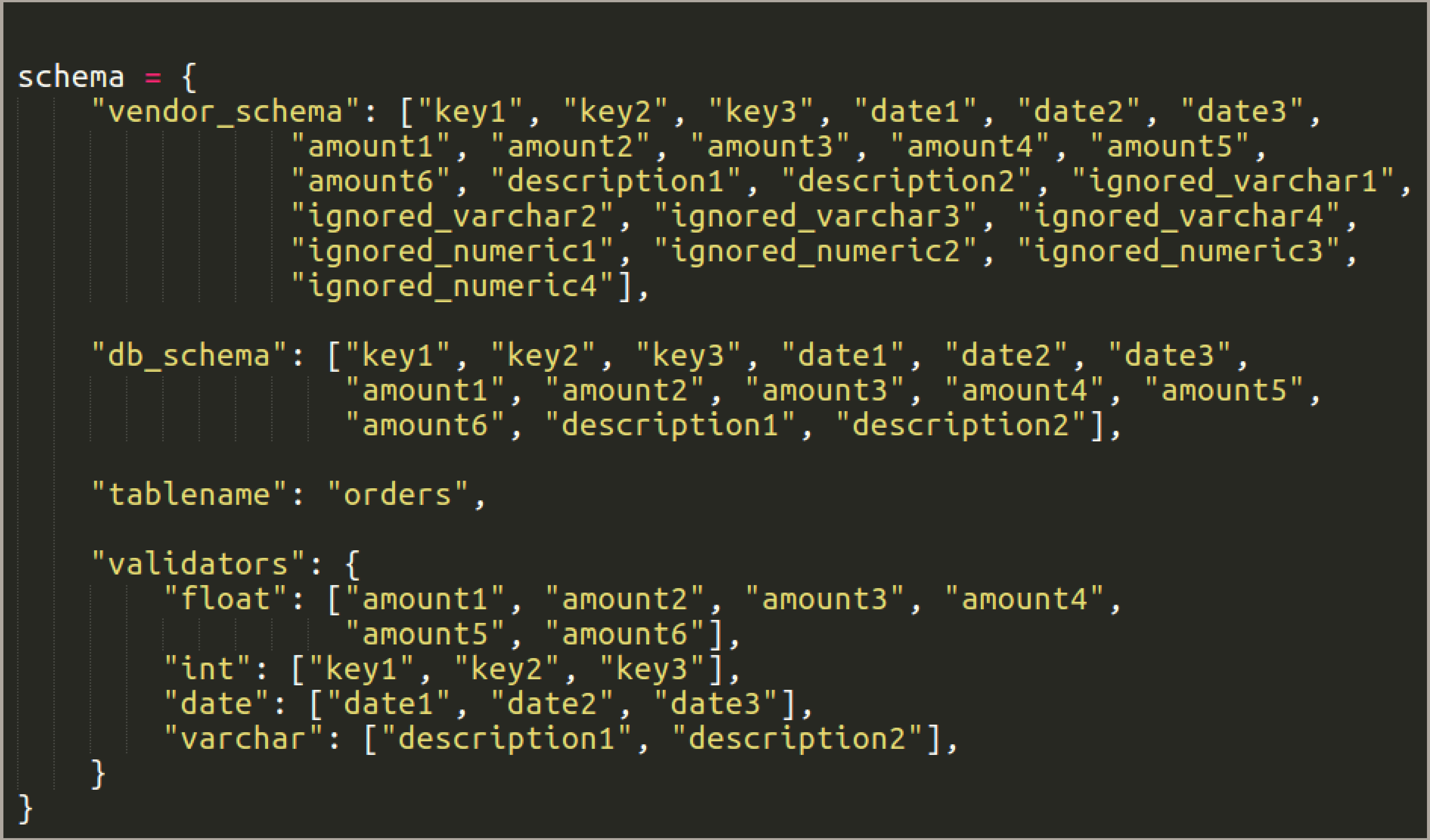


```
if __name__ == "__main__":
    program = time.time()

    raw_orders = load_incoming_data(file_to_read)
    clean_orders = validate_incoming_data(raw_orders)
    success = write_csv(clean_orders, file_to_write)

    program_end = time.time() - program
    print("stlid: {0:.2f}, Orders: {1:}".format(program_end, len(raw_orders)))
```

- Functional core, imperative shell

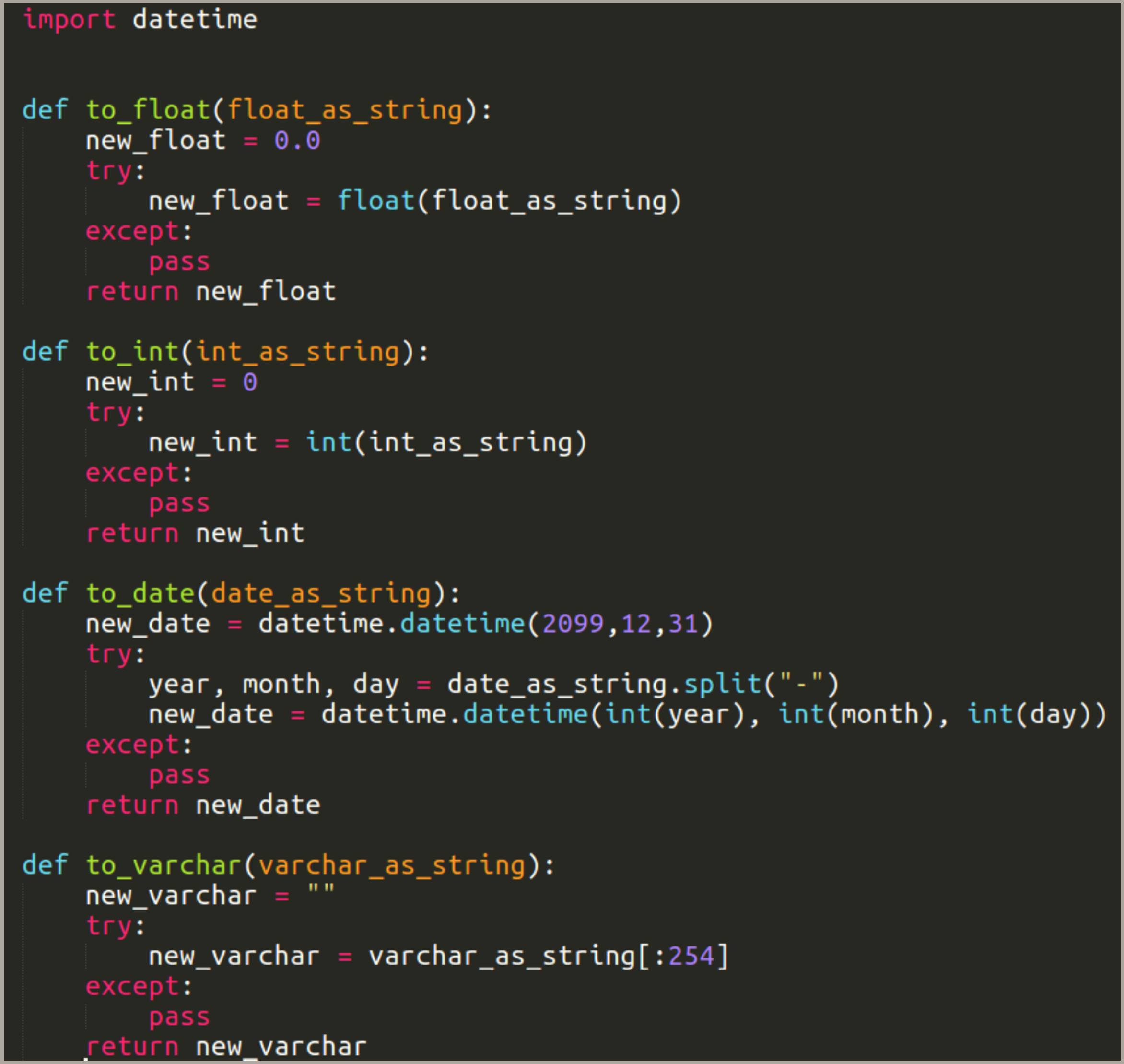
  - Gary Bernhardt, destroyallsoftware.com
- Each function, or module, is testable and reusable
- Errors and restarts are simple to understand

Schema Management



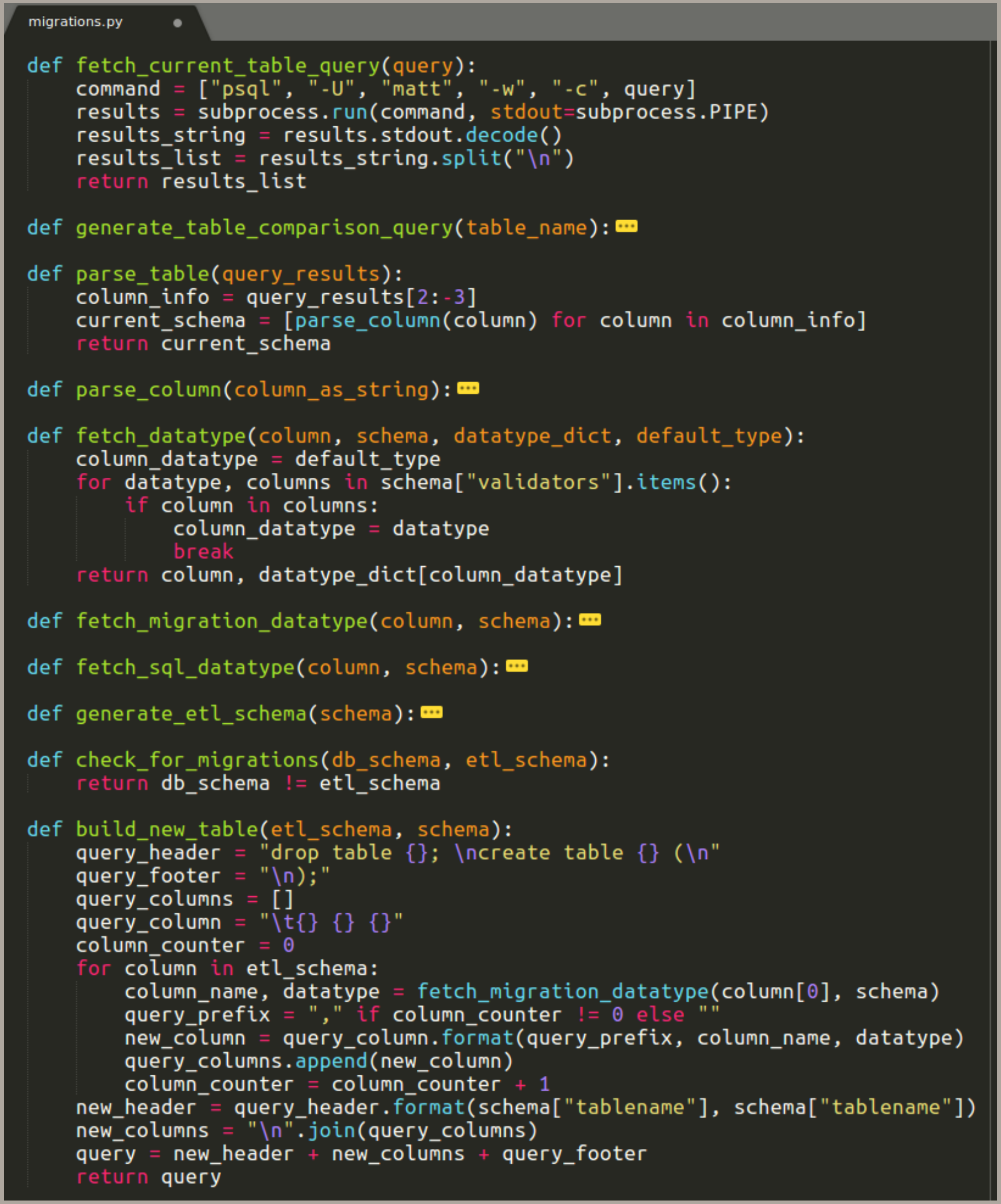
```
schema = {
    "vendor_schema": ["key1", "key2", "key3", "date1", "date2", "date3",
               "amount1", "amount2", "amount3", "amount4", "amount5",
               "amount6", "description1", "description2", "ignored_varchar1",
               "ignored_varchar2", "ignored_varchar3", "ignored_varchar4",
               "ignored_numeric1", "ignored_numeric2", "ignored_numeric3",
               "ignored_numeric4"],

    "db_schema": ["key1", "key2", "key3", "date1", "date2", "date3",
                  "amount1", "amount2", "amount3", "amount4", "amount5",
                  "amount6", "description1", "description2"],

    "tablename": "orders",

    "validators": {
        "float": ["amount1", "amount2", "amount3", "amount4",
                  "amount5", "amount6"],
        "int": ["key1", "key2", "key3"],
        "date": ["date1", "date2", "date3"],
        "varchar": ["description1", "description2"],
    }
}
```

Type Validation



```python
import datetime


def to_float(float_as_string):
    new_float = 0.0
    try:
        new_float = float(float_as_string)
    except:
        pass
    return new_float

def to_int(int_as_string):
    new_int = 0
    try:
        new_int = int(int_as_string)
    except:
        pass
    return new_int

def to_date(date_as_string):
    new_date = datetime.datetime(2099,12,31)
    try:
        year, month, day = date_as_string.split("-")
        new_date = datetime.datetime(int(year), int(month), int(day))
    except:
        pass
    return new_date

def to_varchar(varchar_as_string):
    new_varchar = ""
    try:
        new_varchar = varchar_as_string[:254]
    except:
        pass
    return new_varchar
```

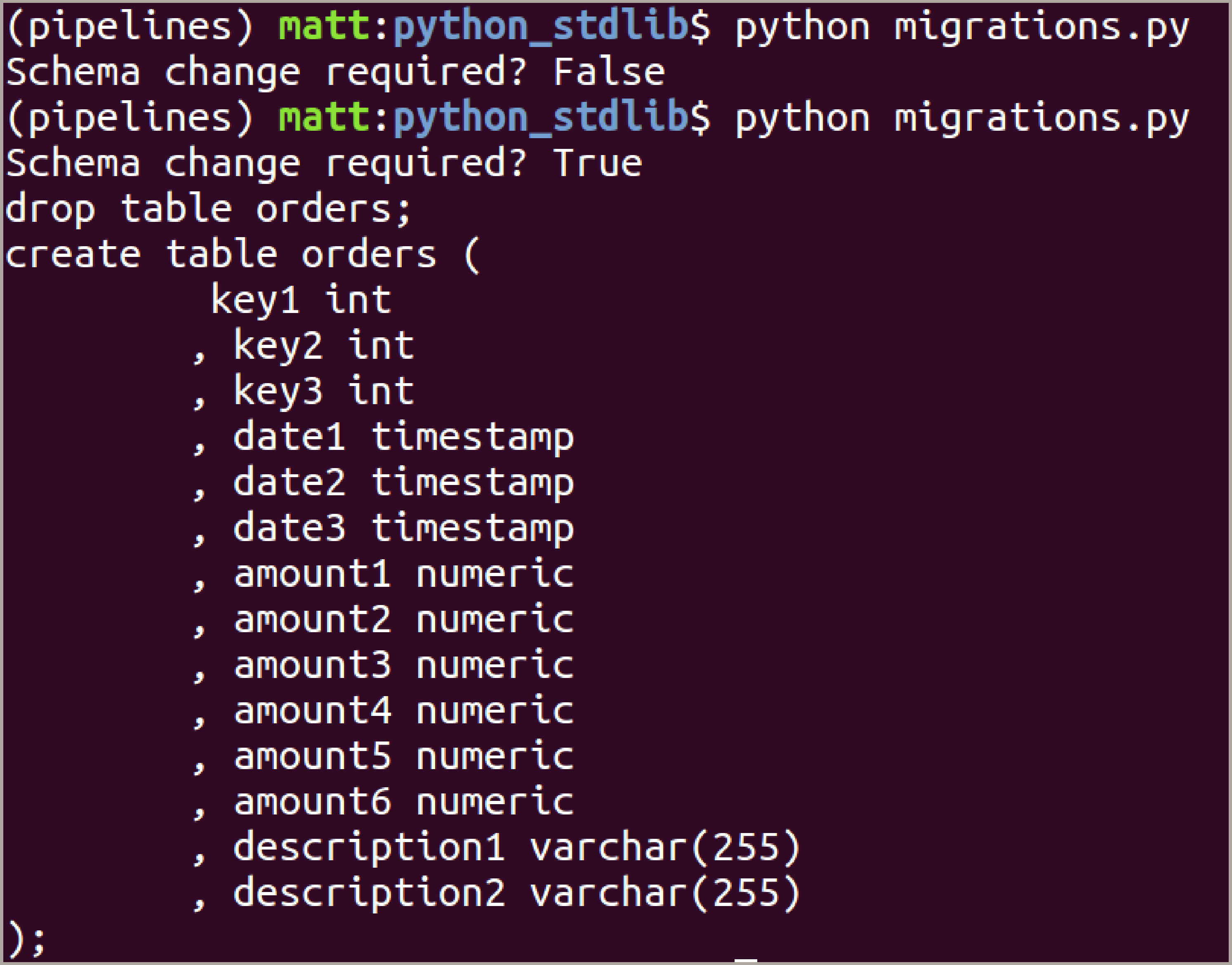
Migrations



migrations.py

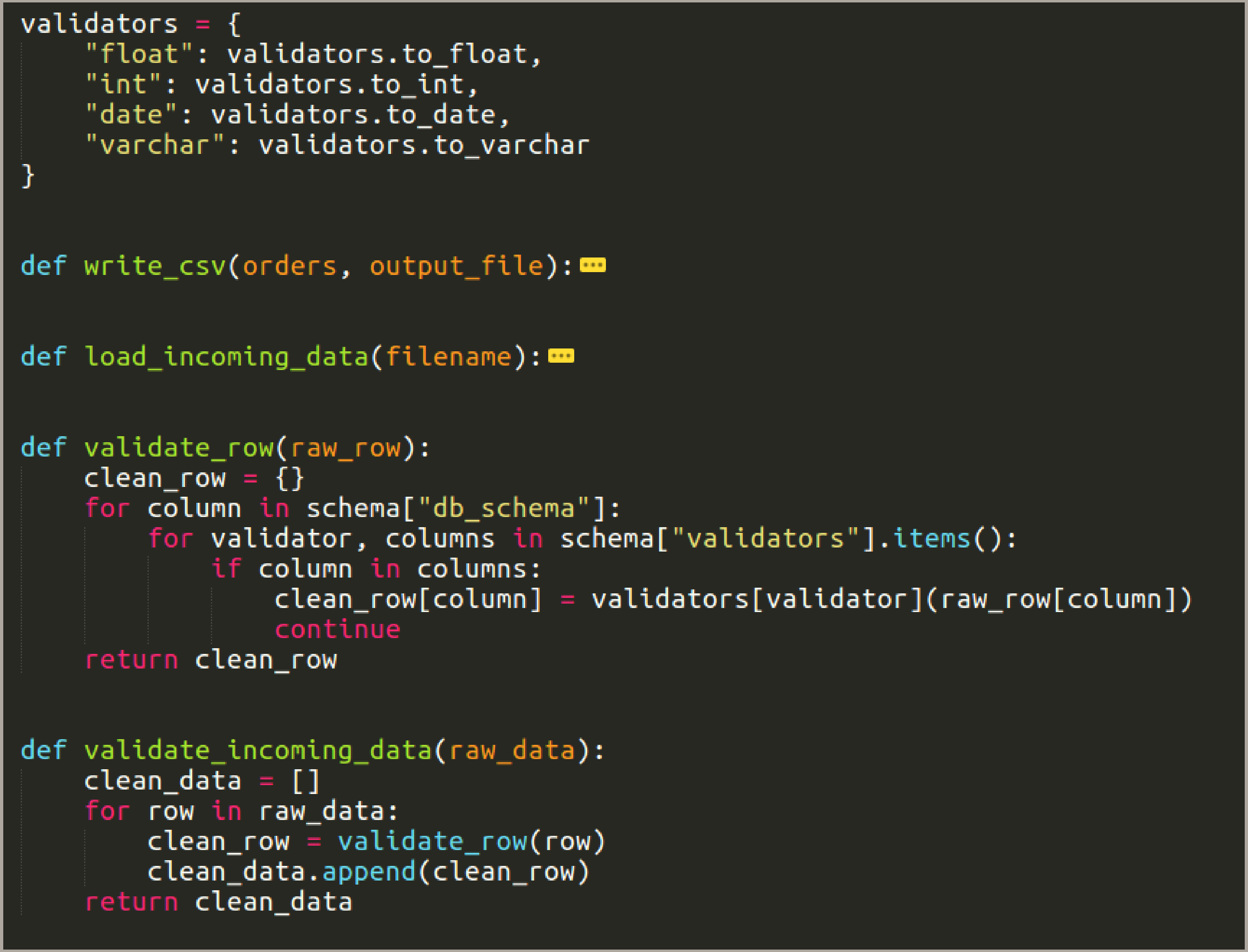
```python
def fetch_current_table_query(query):
    command = ["psql", "-U", "matt", "-w", "-c", query]
    results = subprocess.run(command, stdout=subprocess.PIPE)
    results_string = results.stdout.decode()
    results_list = results_string.split("\n")
    return results_list

def generate_table_comparison_query(table_name):

def parse_table(query_results):
    column_info = query_results[2:-3]
    current_schema = [parse_column(column) for column in column_info]
    return current_schema

def parse_column(column_as_string):

def fetch_datatype(column, schema, datatype_dict, default_type):
    column_datatype = default_type
    for datatype, columns in schema["validators"].items():
        if column in columns:
            column_datatype = datatype
            break
    return column, datatype_dict[column_datatype]

def fetch_migration_datatype(column, schema):

def fetch_sql_datatype(column, schema):

def generate_etl_schema(schema):

def check_for_migrations(db_schema, etl_schema):
    return db_schema != etl_schema

def build_new_table(etl_schema, schema):
    query_header = "drop table {}; \ncreate table {} (\n"
    query_footer = "\n);"
    query_columns = []
    query_column = "\t{} {} {}"
    column_counter = 0
    for column in etl_schema:
        column_name, datatype = fetch_migration_datatype(column[0], schema)
        query_prefix = "," if column_counter != 0 else ""
        new_column = query_column.format(query_prefix, column_name, datatype)
        query_columns.append(new_column)
        column_counter = column_counter + 1
    new_header = query_header.format(schema["tablename"], schema["tablename"])
    new_columns = "\n".join(query_columns)
    query = new_header + new_columns + query_footer
    return query
```

Running Migrations



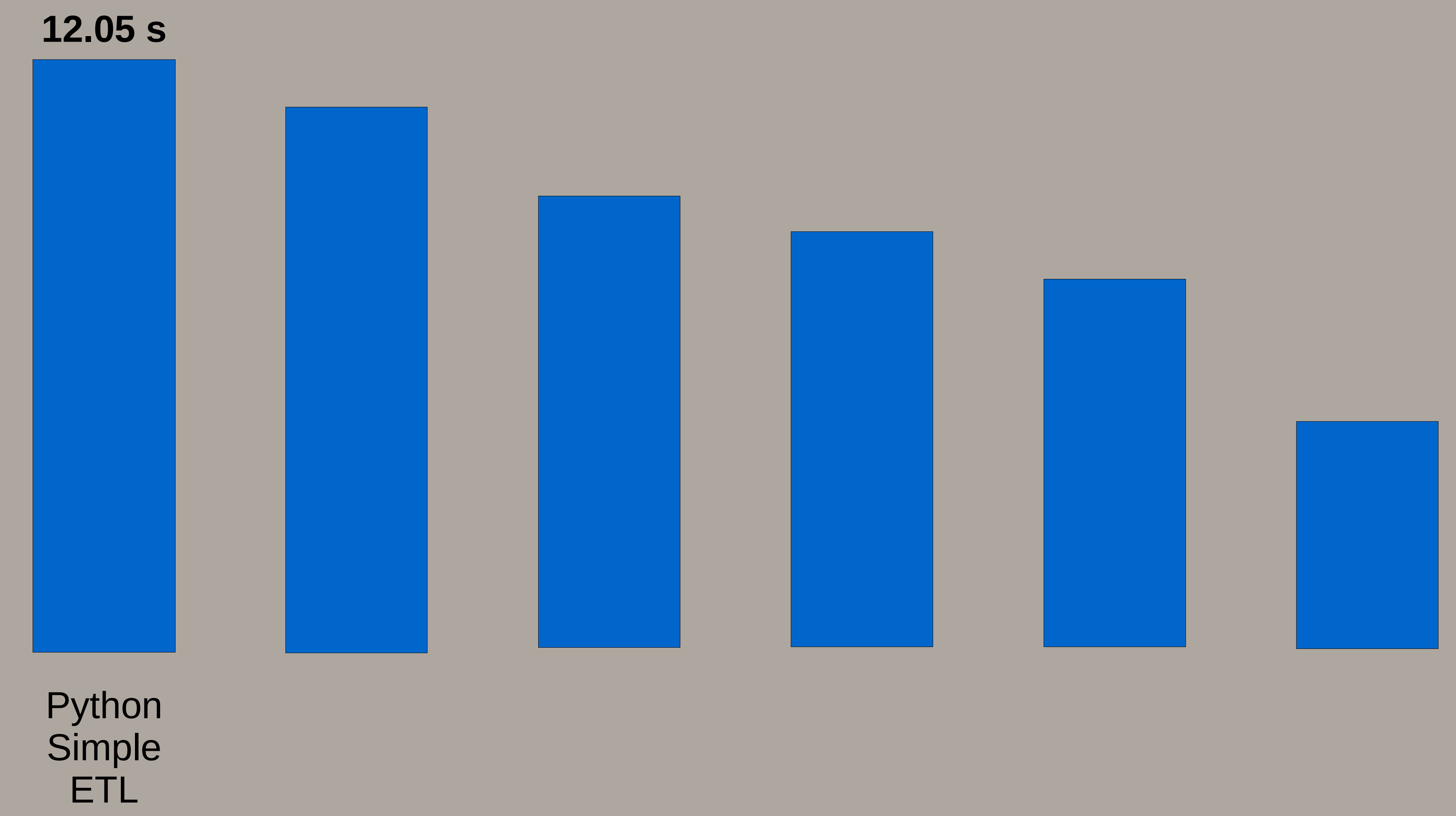
```
(pipelines) matt:python_stdlib$ python migrations.py
Schema change required? False
(pipelines) matt:python_stdlib$ python migrations.py
Schema change required? True
drop table orders;
create table orders (
         key1 int
        , key2 int
        , key3 int
        , date1 timestamp
        , date2 timestamp
        , date3 timestamp
        , amount1 numeric
        , amount2 numeric
        , amount3 numeric
        , amount4 numeric
        , amount5 numeric
        , amount6 numeric
        , description1 varchar(255)
        , description2 varchar(255)
);
```

Data Validation

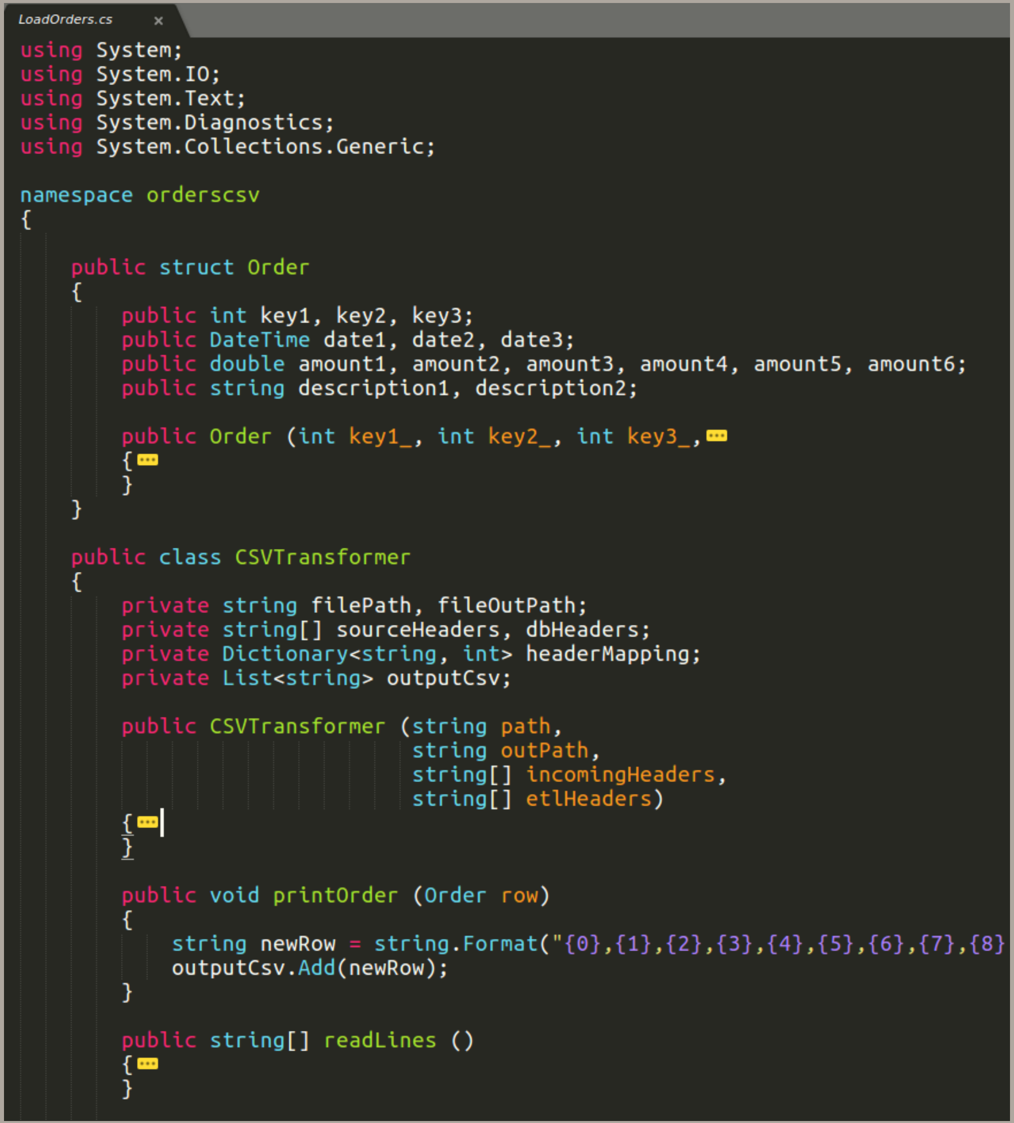


```
validators = {
    "float": validators.to_float,
    "int": validators.to_int,
    "date": validators.to_date,
    "varchar": validators.to_varchar
}


def write_csv(orders, output_file):


def load_incoming_data(filename):


def validate_row(raw_row):
    clean_row = {}
    for column in schema["db_schema"]:
        for validator, columns in schema["validators"].items():
            if column in columns:
                clean_row[column] = validators[validator](raw_row[column])
                continue
    return clean_row


def validate_incoming_data(raw_data):
    clean_data = []
    for row in raw_data:
        clean_row = validate_row(row)
        clean_data.append(clean_row)
    return clean_data
```

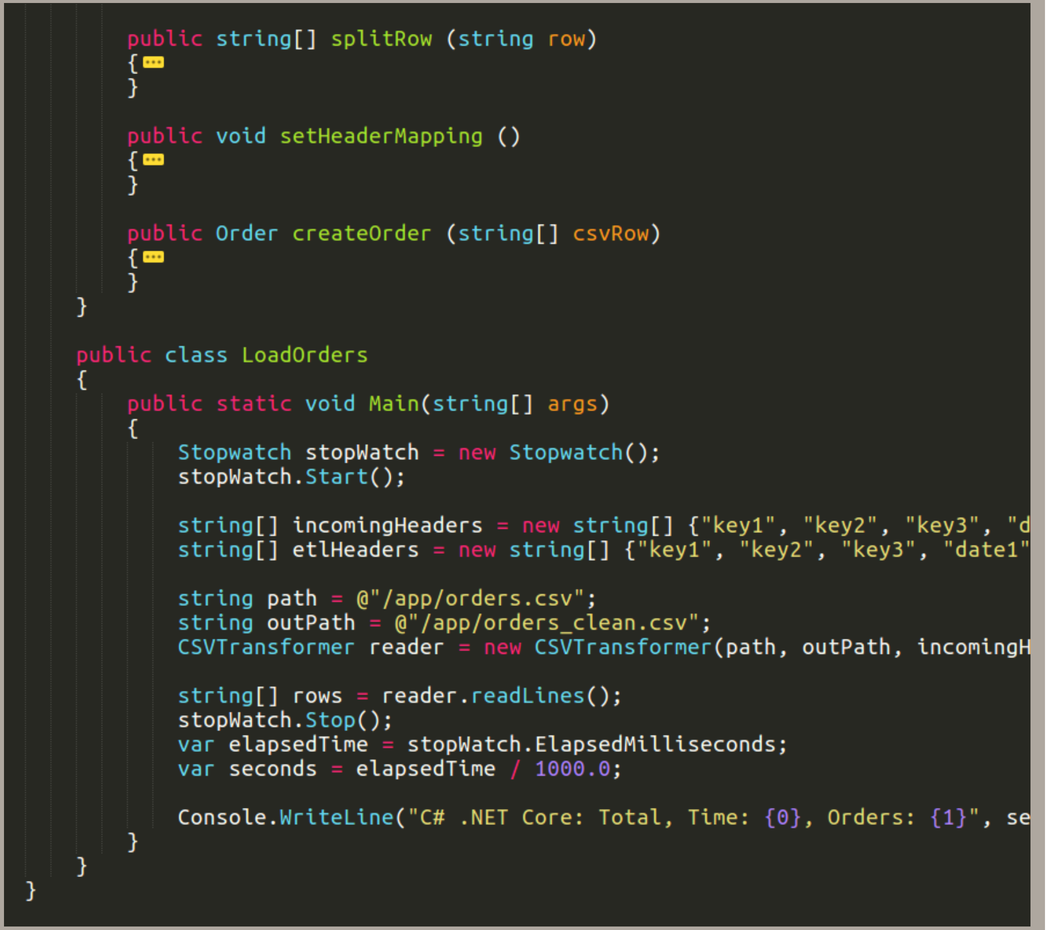
Performance



Behind The Curtain – C#

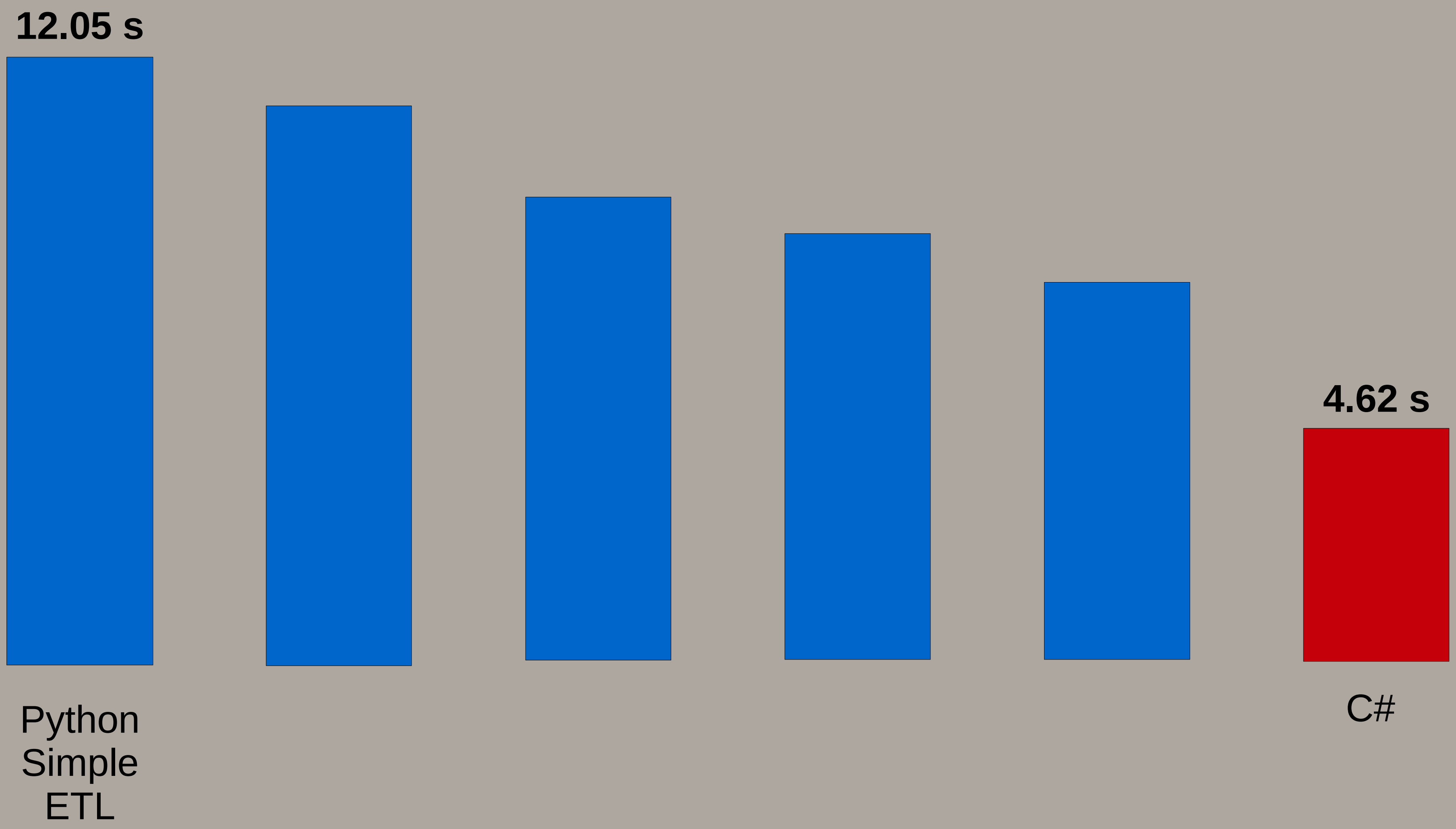


```
using System;
using System.IO;
using System.Text;
using System.Diagnostics;
using System.Collections.Generic;

namespace orderscsv
{
    public struct Order
    {
        public int key1, key2, key3;
        public DateTime date1, date2, date3;
        public double amount1, amount2, amount3, amount4, amount5, amount6;
        public string description1, description2;

        public Order (int key1_, int key2_, int key3_,⋯
        {⋯
        }
    }

    public class CSVTransformer
    {
        private string filePath, fileOutPath;
        private string[] sourceHeaders, dbHeaders;
        private Dictionary<string, int> headerMapping;
        private List<string> outputCsv;

        public CSVTransformer (string path,
                               string outPath,
                               string[] incomingHeaders,
                               string[] etlHeaders)
        {⋯
        }

        public void printOrder (Order row)
        {
            string newRow = string.Format("{0},{1},{2},{3},{4},{5},{6},{7},{8}
            outputCsv.Add(newRow);
        }

        public string[] readLines ()
        {⋯
        }
```

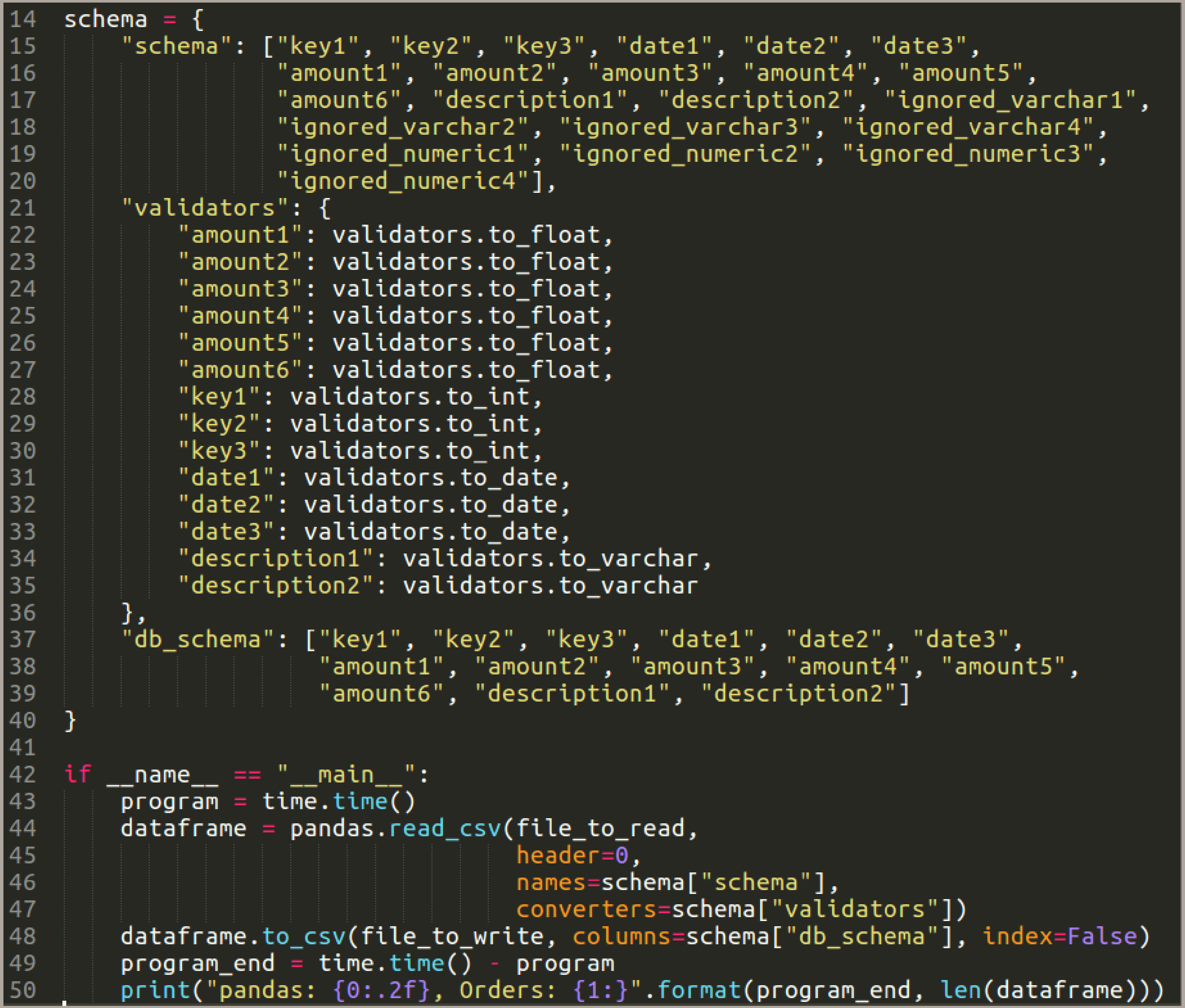


```
        public string[] splitRow (string row)
        {⋯
        }

        public void setHeaderMapping ()
        {⋯
        }

        public Order createOrder (string[] csvRow)
        {⋯
        }
    }

    public class LoadOrders
    {
        public static void Main(string[] args)
        {
            Stopwatch stopWatch = new Stopwatch();
            stopWatch.Start();

            string[] incomingHeaders = new string[] {"key1", "key2", "key3", "d
            string[] etlHeaders = new string[] {"key1", "key2", "key3", "date1"

            string path = @"/app/orders.csv";
            string outPath = @"/app/orders_clean.csv";
            CSVTransformer reader = new CSVTransformer(path, outPath, incomingH

            string[] rows = reader.readLines();
            stopWatch.Stop();
            var elapsedTime = stopWatch.ElapsedMilliseconds;
            var seconds = elapsedTime / 1000.0;

            Console.WriteLine("C# .NET Core: Total, Time: {0}, Orders: {1}", se
        }
    }
}
```

Performance

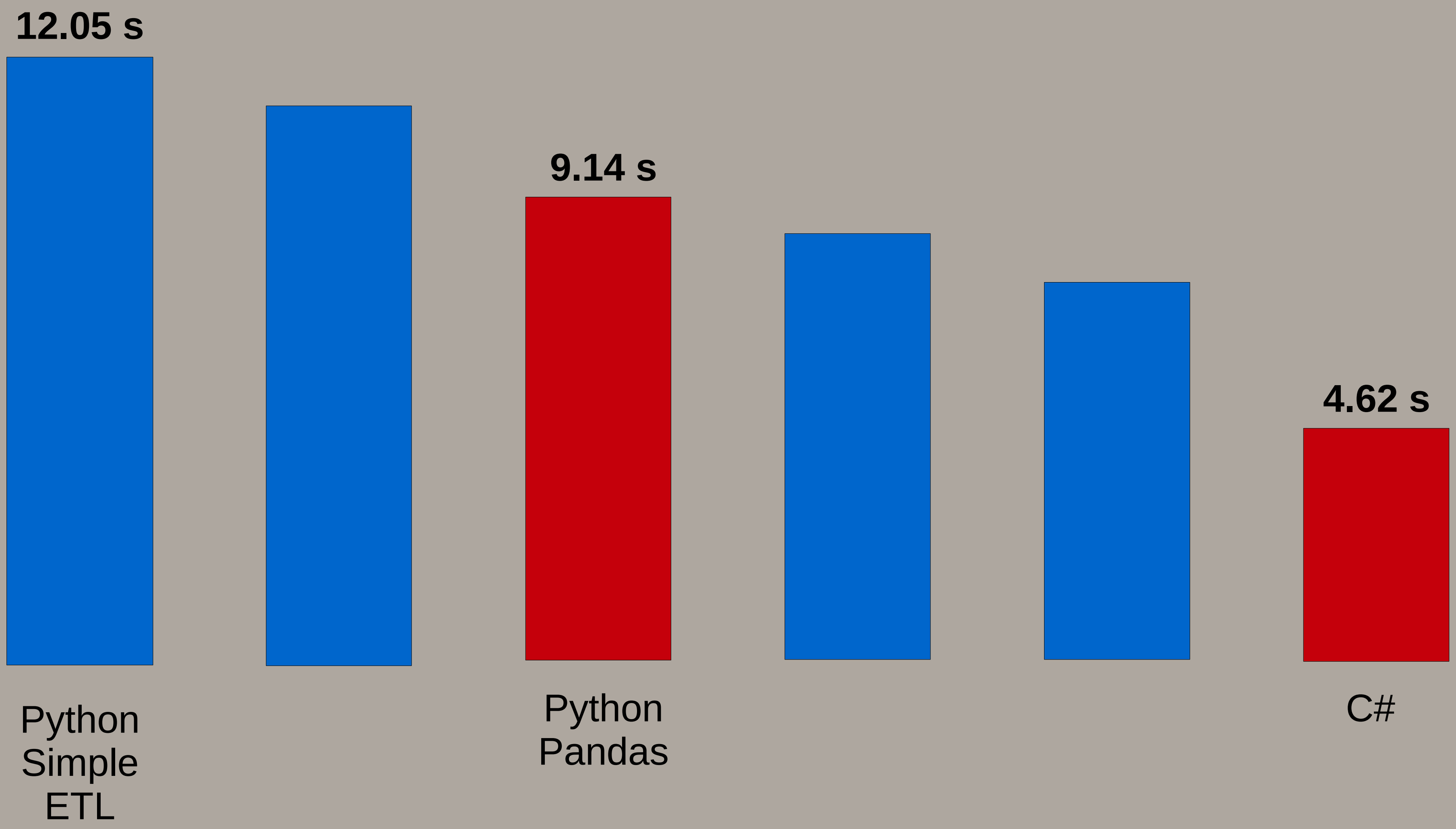


Behind The Curtain – Pandas

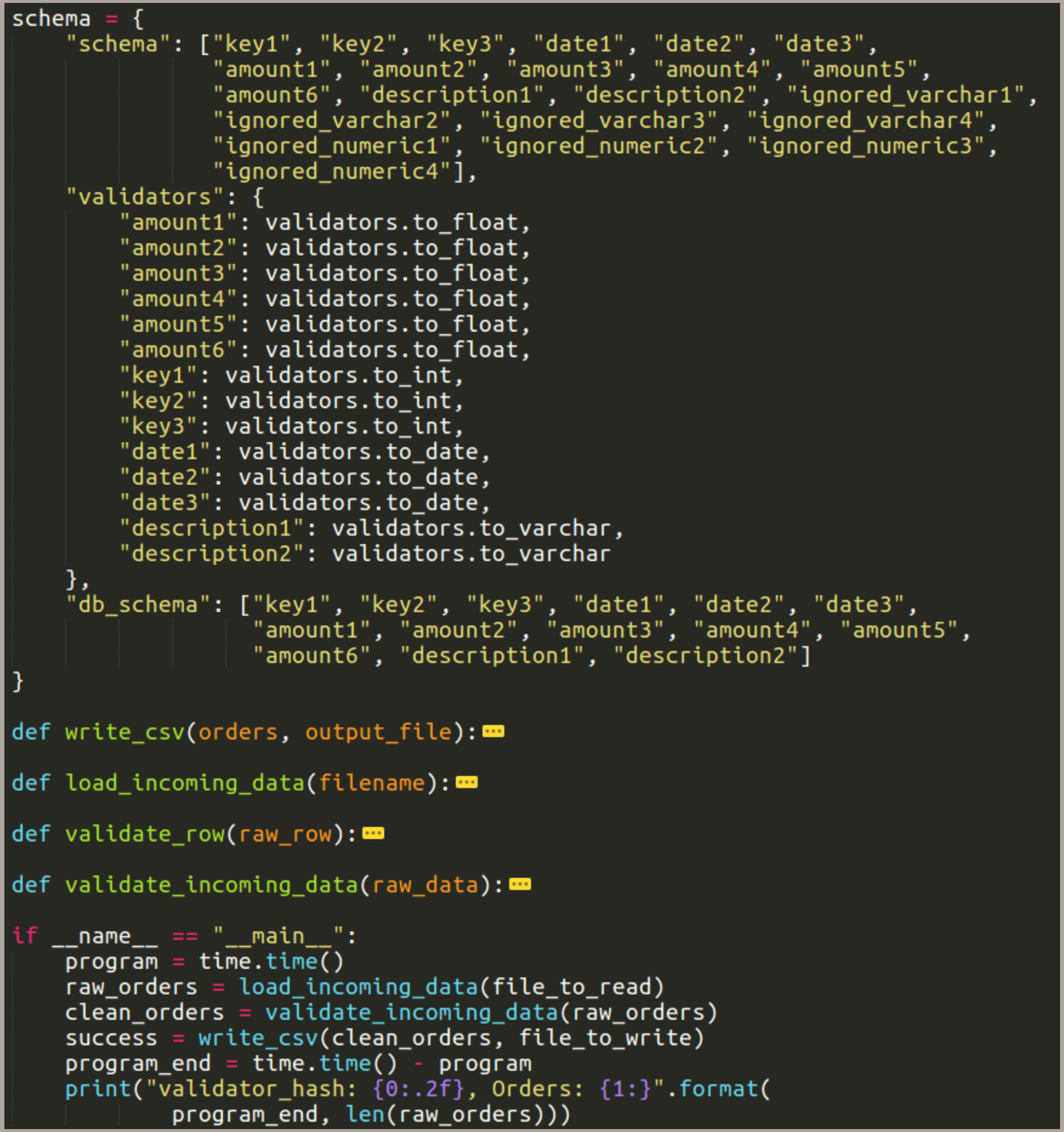


```python
schema = {
    "schema": ["key1", "key2", "key3", "date1", "date2", "date3",
               "amount1", "amount2", "amount3", "amount4", "amount5",
               "amount6", "description1", "description2", "ignored_varchar1",
               "ignored_varchar2", "ignored_varchar3", "ignored_varchar4",
               "ignored_numeric1", "ignored_numeric2", "ignored_numeric3",
               "ignored_numeric4"],
    "validators": {
        "amount1": validators.to_float,
        "amount2": validators.to_float,
        "amount3": validators.to_float,
        "amount4": validators.to_float,
        "amount5": validators.to_float,
        "amount6": validators.to_float,
        "key1": validators.to_int,
        "key2": validators.to_int,
        "key3": validators.to_int,
        "date1": validators.to_date,
        "date2": validators.to_date,
        "date3": validators.to_date,
        "description1": validators.to_varchar,
        "description2": validators.to_varchar
    },
    "db_schema": ["key1", "key2", "key3", "date1", "date2", "date3",
                  "amount1", "amount2", "amount3", "amount4", "amount5",
                  "amount6", "description1", "description2"]
}

if __name__ == "__main__":
    program = time.time()
    dataframe = pandas.read_csv(file_to_read,
                                header=0,
                                names=schema["schema"],
                                converters=schema["validators"])
    dataframe.to_csv(file_to_write, columns=schema["db_schema"], index=False)
    program_end = time.time() - program
    print("pandas: {0:.2f}, Orders: {1:}".format(program_end, len(dataframe)))
```

Performance

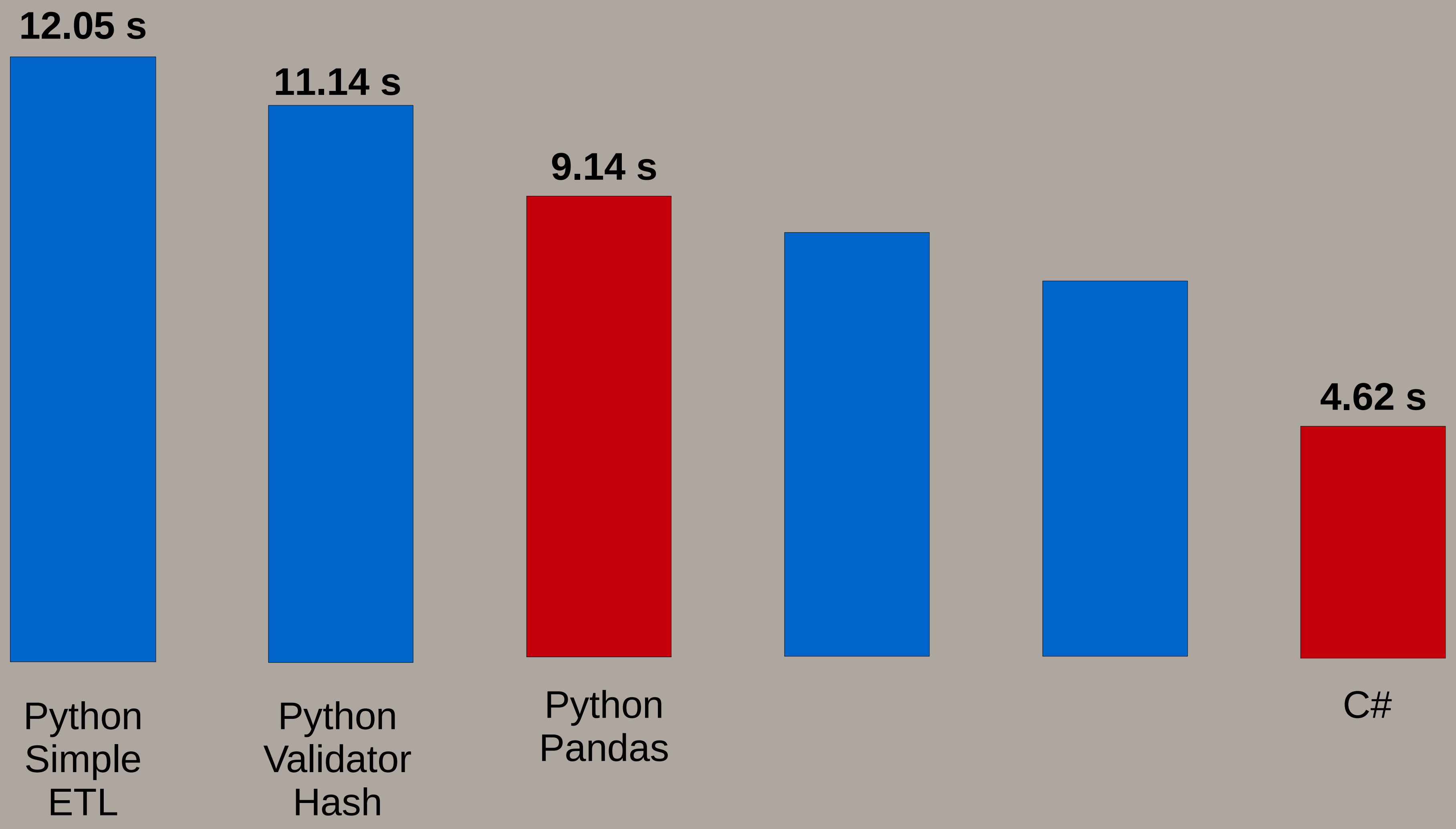


Behind The Curtain – Validator Hash

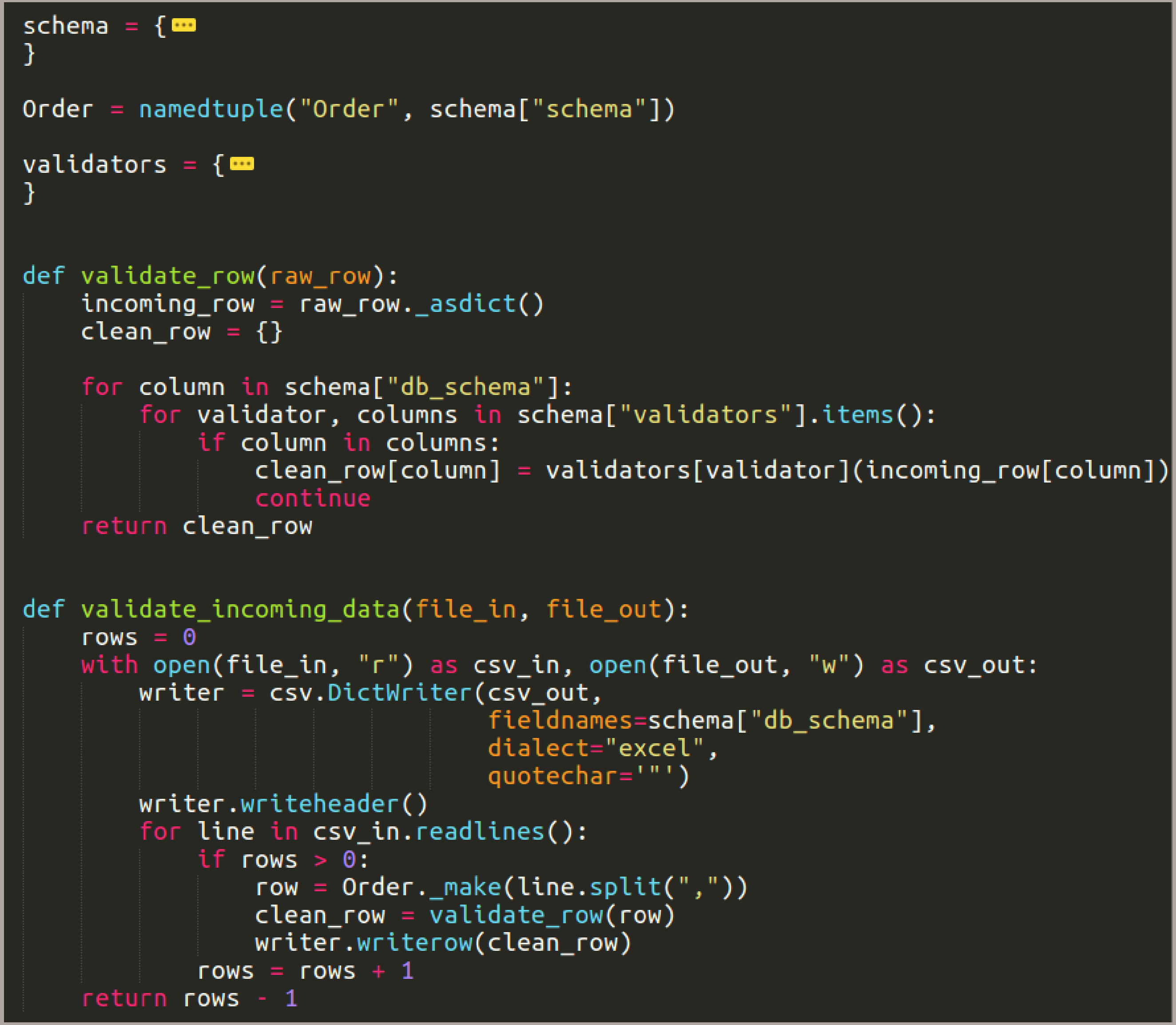


```
schema = {
    "schema": ["key1", "key2", "key3", "date1", "date2", "date3",
               "amount1", "amount2", "amount3", "amount4", "amount5",
               "amount6", "description1", "description2", "ignored_varchar1",
               "ignored_varchar2", "ignored_varchar3", "ignored_varchar4",
               "ignored_numeric1", "ignored_numeric2", "ignored_numeric3",
               "ignored_numeric4"],
    "validators": {
        "amount1": validators.to_float,
        "amount2": validators.to_float,
        "amount3": validators.to_float,
        "amount4": validators.to_float,
        "amount5": validators.to_float,
        "amount6": validators.to_float,
        "key1": validators.to_int,
        "key2": validators.to_int,
        "key3": validators.to_int,
        "date1": validators.to_date,
        "date2": validators.to_date,
        "date3": validators.to_date,
        "description1": validators.to_varchar,
        "description2": validators.to_varchar
    },
    "db_schema": ["key1", "key2", "key3", "date1", "date2", "date3",
                  "amount1", "amount2", "amount3", "amount4", "amount5",
                  "amount6", "description1", "description2"]
}

def write_csv(orders, output_file):...

def load_incoming_data(filename):...

def validate_row(raw_row):...

def validate_incoming_data(raw_data):...

if __name__ == "__main__":
    program = time.time()
    raw_orders = load_incoming_data(file_to_read)
    clean_orders = validate_incoming_data(raw_orders)
    success = write_csv(clean_orders, file_to_write)
    program_end = time.time() - program
    print("validator_hash: {0:.2f}, Orders: {1:}".format(
            program_end, len(raw_orders)))
```

Performance

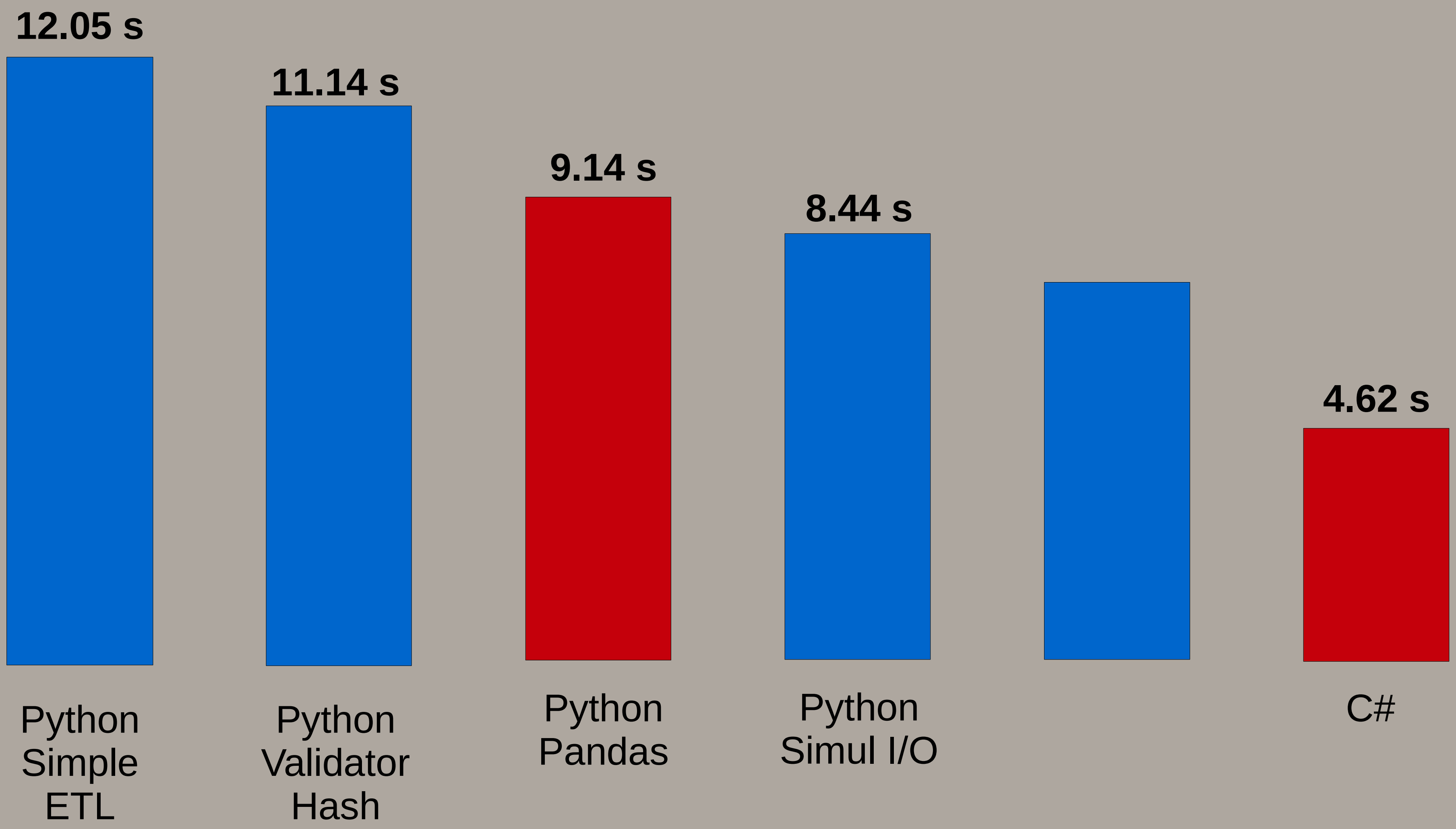


Behind The Curtain – Simultaneous I/O

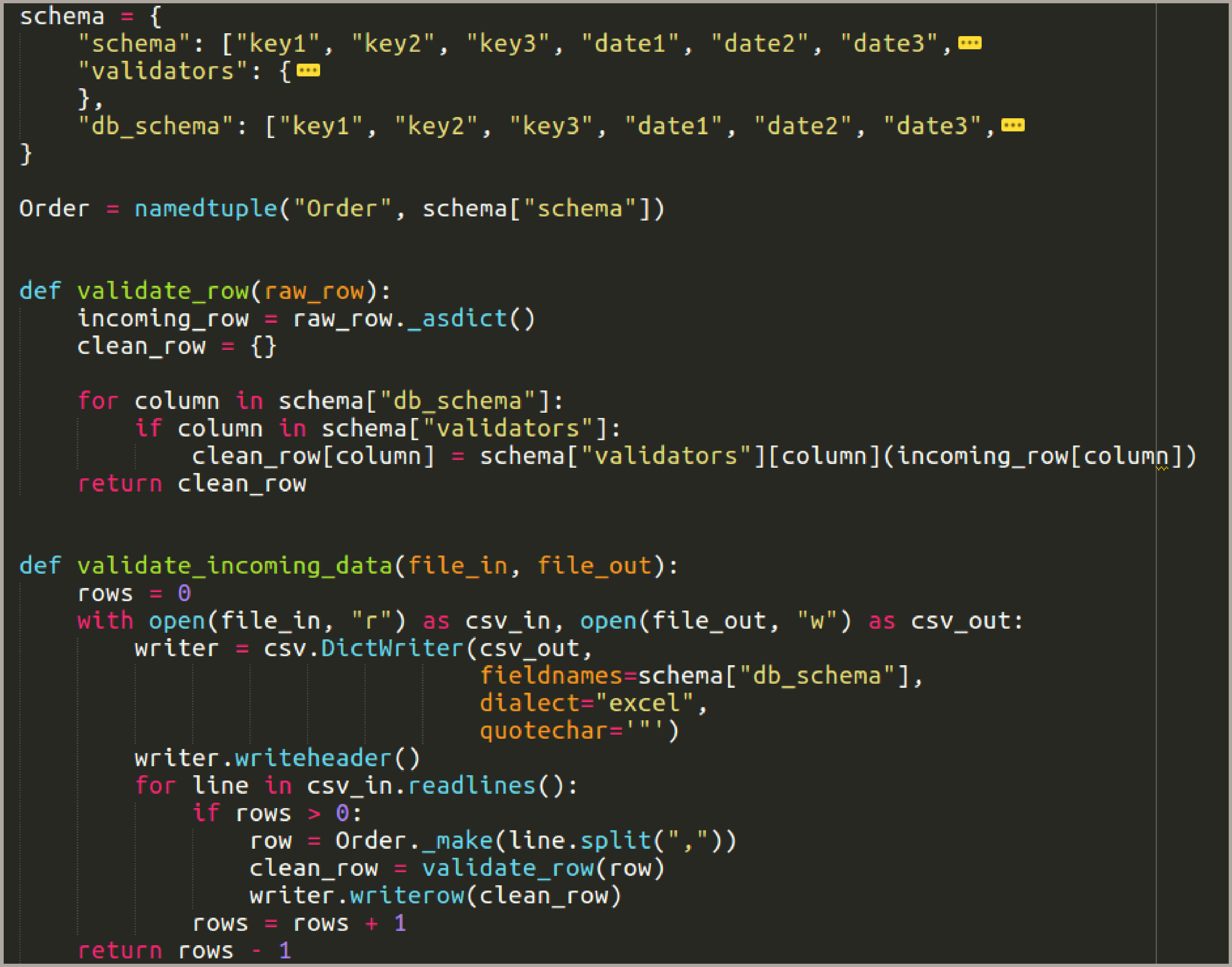


```
schema = {…
}

Order = namedtuple("Order", schema["schema"])

validators = {…
}


def validate_row(raw_row):
    incoming_row = raw_row._asdict()
    clean_row = {}

    for column in schema["db_schema"]:
        for validator, columns in schema["validators"].items():
            if column in columns:
                clean_row[column] = validators[validator](incoming_row[column])
                continue
    return clean_row


def validate_incoming_data(file_in, file_out):
    rows = 0
    with open(file_in, "r") as csv_in, open(file_out, "w") as csv_out:
        writer = csv.DictWriter(csv_out,
                                fieldnames=schema["db_schema"],
                                dialect="excel",
                                quotechar='"')
        writer.writeheader()
        for line in csv_in.readlines():
            if rows > 0:
                row = Order._make(line.split(","))
                clean_row = validate_row(row)
                writer.writerow(clean_row)
            rows = rows + 1
    return rows - 1
```

Performance

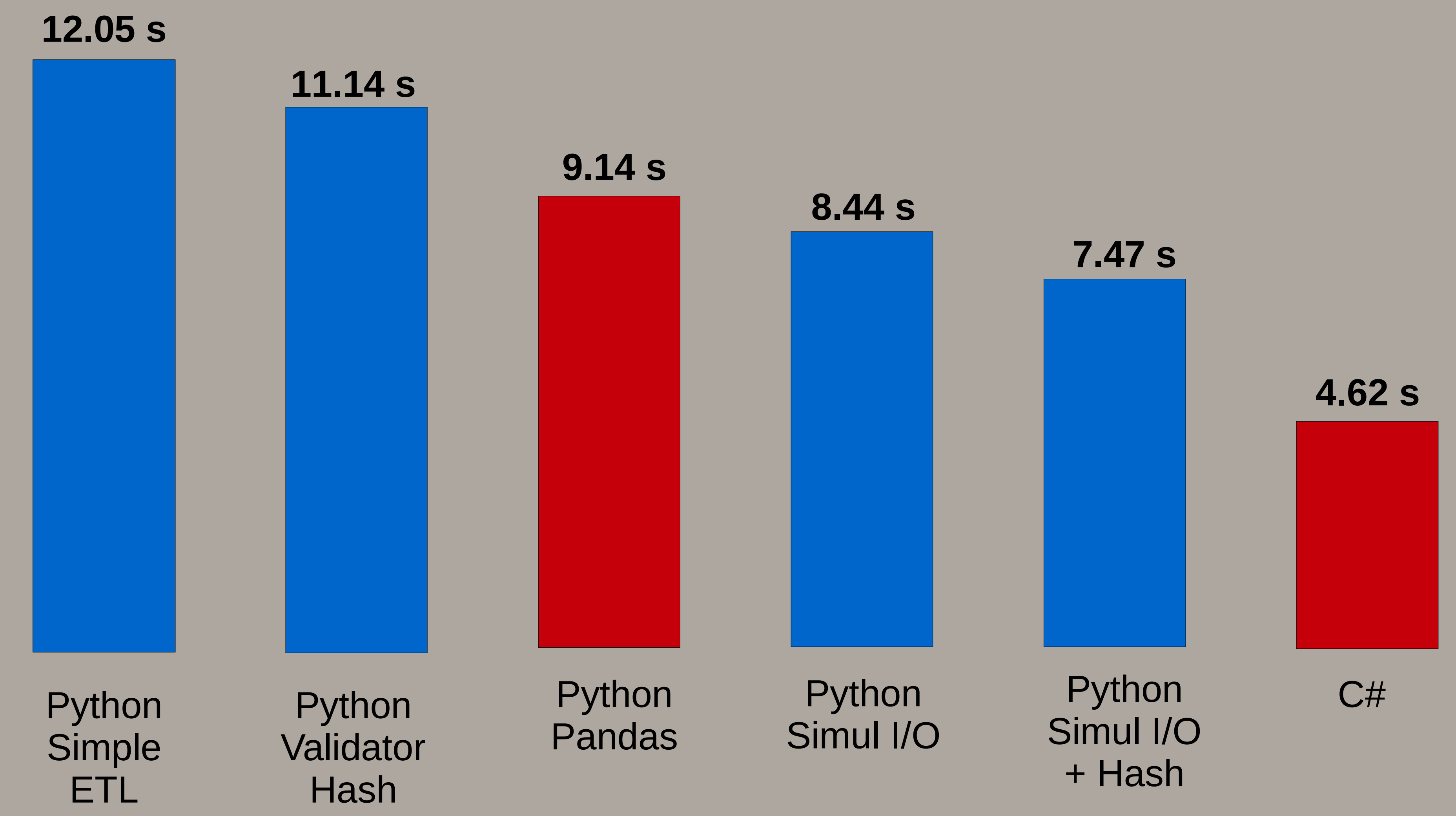


Behind The Curtain – Validator Hash + Simul I/O



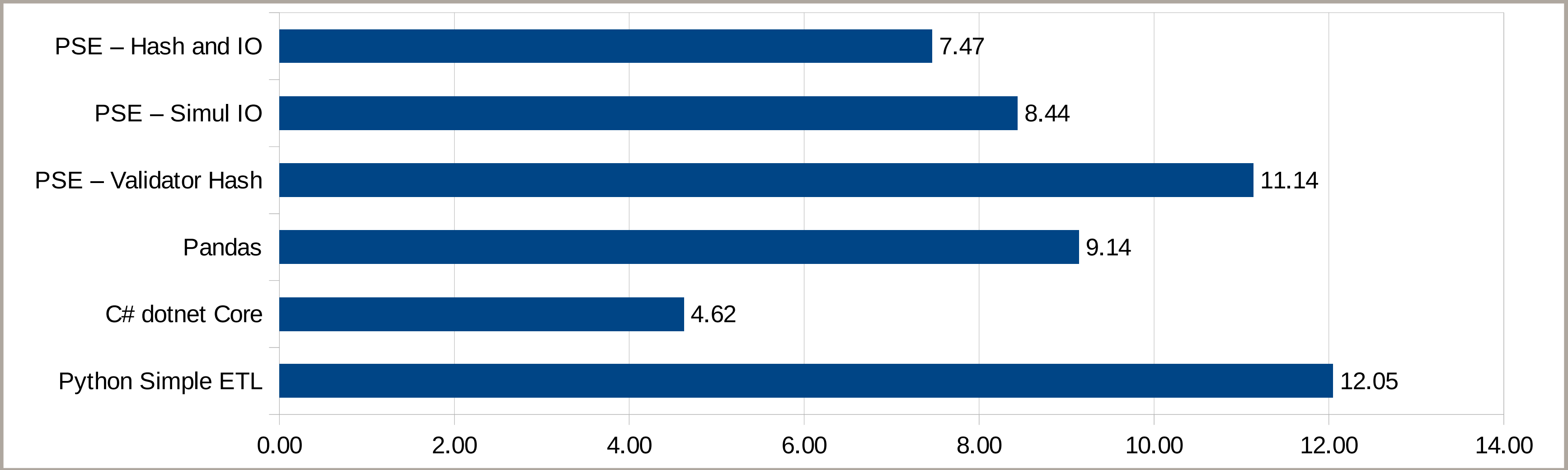
```
schema = {
    "schema": ["key1", "key2", "key3", "date1", "date2", "date3",
    "validators": {
    },
    "db_schema": ["key1", "key2", "key3", "date1", "date2", "date3",
}

Order = namedtuple("Order", schema["schema"])


def validate_row(raw_row):
    incoming_row = raw_row._asdict()
    clean_row = {}

    for column in schema["db_schema"]:
        if column in schema["validators"]:
            clean_row[column] = schema["validators"][column](incoming_row[column])
    return clean_row


def validate_incoming_data(file_in, file_out):
    rows = 0
    with open(file_in, "r") as csv_in, open(file_out, "w") as csv_out:
        writer = csv.DictWriter(csv_out,
                                fieldnames=schema["db_schema"],
                                dialect="excel",
                                quotechar='"')
        writer.writeheader()
        for line in csv_in.readlines():
            if rows > 0:
                row = Order._make(line.split(","))
                clean_row = validate_row(row)
                writer.writerow(clean_row)
            rows = rows + 1
    return rows - 1
```

Performance



“Performance Testing” – Results

System	Average Time	Min Time	Max Time	% Speedup
Python Simple ETL	12.05	12.02	12.08	
C# dotnet Core	4.62	4.56	4.74	62%
Pandas	9.14	9.09	9.20	24%
PSE – Validator Hash	11.14	11.02	11.40	8%
PSE – Simultaneous IO	8.44	8.41	8.49	30%
PSE – Hash and IO	7.47	7.44	7.52	38%



But how long is 3 seconds really?

- Let’s say you refresh data every 10 minutes, C# -> ~3 second speedup, 30 sec/hour
- Refresh all business hours, 8am-8pm EST
- Python costs you 6 min/day performance
- AAV developer cost: $4.80/day, $1200/year
- Assume Python saves 90 min schema change
- Breakeven at 17 changes (1.5/month)
- Ignores errors, tests, json, archival, rebuilds…

Before We Say Farewell...

- Python standard library is a rich ecosystem for managing ETL systems
- Tooling includes sensible ways to handle file grouping, archival, environment variables, source control, testing, error handling, and encapsulation
- Performance spans good enough to good
- Pythonic code is great

Python: Jack of ETL Trades

Thank you!