Business Intelligence-relaterade programspråk

Data sources

Data sources

- Data can be provided to us via a number of sources.
- We will look at how we can handle some of them.
  - CSV files
  - Databases (MySQL)
  - APIs

CSV

Comma Separated Values

- A CSV file (Comma Separated Values file) is a type of plain text file that uses specific structuring to arrange tabular data. Because it's a plain text file, it can contain only actual text data—in other words, printable ASCII or Unicode characters.

- The structure of a CSV file is given away by its name. Normally, CSV files use a comma to separate each specific data value. Here's what that structure looks like:

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

- Notice how each piece of data is separated by a comma. Normally, the first line identifies each piece of data—in other words, the name of a data column. Every subsequent line after that is actual data and is limited only by file size constraints.

- In general, the separator character is called a delimiter, and the comma is not the only one used. Other popular delimiters include the tab (\t), colon (:) and semi-colon (;) characters. Properly parsing a CSV file requires us to know which delimiter is being used.

CSV

Where Do CSV Files Come From?

- CSV files are normally created by programs that handle large amounts of data. They are a convenient way to export data from spreadsheets and databases as well as import or use it in other programs. For example, you might export the results of a data mining program to a CSV file and then import that into a spreadsheet to analyze the data, generate graphs for a presentation, or prepare a report for publication.

- CSV files are very easy to work with programmatically. Any language that supports text file input and string manipulation (like Python) can work with CSV files directly.

CSV

Reading CSV Files With csv

- The `csv` library provides functionality to both read from and write to CSV files. Designed to work out of the box with Excel-generated CSV files, it is easily adapted to work with a variety of CSV formats. The `csv` library contains objects and other code to read, write, and process data from and to CSV files.

CSV

Reading CSV Files With csv

- Reading from a CSV file is done using the reader object. The CSV file is opened as a text file with Python's built-in `open()` function, which returns a file object. This is then passed to the reader, which does the heavy lifting.

- Here's the `employee_birthday.txt` file:

```
name,department,birthday month
John Smith,Accounting,November
Erica Meyers,IT,March
```

CSV

Reading CSV Files With csv

```
import csv

with open('employee_birthday.txt') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        else:
            print(f'\t{row[0]} works in the {row[1]} department, and was born in
{row[2]}.')
            line_count += 1
    print(f'Processed {line_count} lines.')
```

CSV

Path

```
with open('employee_birthday.txt') as csv_file:
```

- This path is relative to where the script is run from. You can find out by asking Python which the current directory is.
- Another way of working is by naming the absolute path to the file. In VS Code, you can get this by right clicking I file and selecting *Copy path*.

```
with open('/Users/micke/python/employee_birthday.txt') as employee_file:
```

- *Note:* In Windows, your path will likely have backslashes, which you need to escape.

```
with open('C:\\Users\\micke\\python\\employee_birthday.txt') as employee_file:
```

CSV

Reading CSV Files Into a Dictionary With csv

```
import csv

with open('employee_birthday.txt', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    line_count = 0
    for row in csv_reader:
        if line_count == 0:
            print(f'Column names are {", ".join(row)}')
            line_count += 1
        print(f'\t{row["name"]} works in the {row["department"]} department, and
was born in {row["birthday month"]}.')
        line_count += 1
    print(f'Processed {line_count} lines.')
```

CSV

Optional Python CSV reader Parameters

- The reader object can handle different styles of CSV files by specifying additional parameters, some of which are shown below:
- `delimiter` specifies the character used to separate each field. The default is the comma (,).
- `quotechar` specifies the character used to surround fields that contain the delimiter character. The default is a double quote (").
- `escapechar` specifies the character used to escape the delimiter character, in case quotes aren’t used. The default is no escape character.

CSV

Optional Python CSV reader Parameters

- These parameters deserve some more explanation. Suppose you're working with the following `employee_addresses.txt` file:

```
name,address,date joined
john smith,1132 Anywhere Lane Hoboken NJ, 07030,Jan 4
erica meyers,1234 Smith Lane Hoboken NJ, 07030,March 2
```

- This CSV file contains three fields: name, address, and date joined, which are delimited by commas. The problem is that the data for the address field also contains a comma to signify the zip code.

CSV

Optional Python CSV reader Parameters

- There are three different ways to handle this situation
  - **Use a different delimiter**
    That way, the comma can safely be used in the data itself. You use the `delimiter` optional parameter to specify the new delimiter.
  - **Wrap the data in quotes**
    The special nature of your chosen delimiter is ignored in quoted strings. Therefore, you can specify the character used for quoting with the `quotechar` optional parameter. As long as that character also doesn't appear in the data, you're fine.
  - **Escape the delimiter characters in the data**
    Escape characters work just as they do in format strings, nullifying the interpretation of the character being escaped (in this case, the delimiter). If an escape character is used, it must be specified using the `escapechar` optional parameter.

CSV

Writing CSV Files With csv

- You can also write to a CSV file using a `writer` object and the `.write_row()` method.

```
import csv

with open('employee_file.csv', mode='w') as employee_file:
    employee_writer = csv.writer(employee_file, delimiter=',', quotechar='"',
quoting=csv.QUOTE_MINIMAL)

    employee_writer.writerow(['John Smith', 'Accounting', 'November'])
    employee_writer.writerow(['Erica Meyers', 'IT', 'March'])
```

CSV

Writing CSV Files With csv

- The `quotechar` optional parameter tells the writer which character to use to quote fields when writing. Whether quoting is used or not, however, is determined by the quoting optional parameter:
  - If quoting is set to `csv.QUOTE_MINIMAL`, then `.writerow()` will quote fields only if they contain the delimiter or the quotechar. This is the default case.
  - If quoting is set to `csv.QUOTE_ALL`, then `.writerow()` will quote all fields.
  - If quoting is set to `csv.QUOTE_NONNUMERIC`, then `.writerow()` will quote all fields containing text data and convert all numeric fields to the float data type.
  - If quoting is set to `csv.QUOTE_NONE`, then `.writerow()` will escape delimiters instead of quoting them. In this case, you also must provide a value for the `escapechar` optional parameter.

CSV

Writing CSV File From a Dictionary With csv

```
import csv

with open('employee_file2.csv', mode='w') as
csv_file:
    fieldnames = ['emp_name', 'dept',
'birth_month']
    writer = csv.DictWriter(csv_file,
fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'emp_name': 'John Smith',
'dept': 'Accounting', 'birth_month':
'November'})
    writer.writerow({'emp_name': 'Erica
Meyers', 'dept': 'IT', 'birth_month': 'March'})
```

- Unlike DictReader, the fieldnames parameter is required when writing a dictionary. This makes sense, when you think about it: without a list of fieldnames,
the DictWriter can’t know which keys to use to retrieve values from your dictionaries. It also uses the keys in fieldnames to write out the first row as column names.

Exercise

csv

- Use the file `customers.csv` from the repo or studentportalen.
- Create a new csv file containing only the customers from USA and France.

MySQL

- Python can be used in database applications.
- One of the most popular databases is MySQL.
- You don’t need your own database to follow this lecture, but it will be easier for you to perform your own experiments if you download your own copy.
  - https://www.mysql.com/downloads/
    - Import the classic models database if you want the same dataset.
      - https://www.mysqltutorial.org/mysql-sample-database.aspx/

Install MySQL Driver

- Python needs a MySQL driver to access the MySQL database.
- In this tutorial we will use the driver "MySQL Connector".

```
C:\>pip install mysql-connector-python
```

- Now you have downloaded and installed a MySQL driver.

Test MySQL Connector

- To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content:

```
import mysql.connector
```

- If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.

Create connection

- Use the username and password from your MySQL database:

```
import mysql.connector

mydb = mysql.connector.connect(
    host="remotemysql.com", # your host, localhost for your local db
    user="AybEFDBRkG",      # username
    password="pbNAmdostq"   # password
)

print(mydb)
```

- Now you can start querying the database using SQL statements.

Check if Database Exists

- You can check if a database exist by listing all databases in your system by using the "`SHOW DATABASES`" statement.

```
# import & connect first

mycursor = mydb.cursor()

mycursor.execute("SHOW DATABASES")

for x in mycursor:
    print(x)
```

- Now you can start querying the database using SQL statements.

Access Database

- You can try to access the database when making the connection.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="remotemysql.com",
  user="AybEFDBRkG",
  password="pbNAmdostq",
  database="AybEFDBRkG"
)
```

- If the database does not exist, you will get an error.
- Ordinarily, you would have better username and db name, but with free services, you often get what you pay for.

Execute DDL statements

- For most statements you can use the `execute()` method.

```
mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(255), address VARCHAR(255))") # Example that would create a table
```

Insert Into

Single record

```
mycursor = mydb.cursor()

sql = "INSERT INTO `AybEFDBRkG`.`customers` (`customerNumber`, `customerName`, `contactLastName`, `contactFirstName`, `phone`, `addressLine1`, `city`, `country`) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)"

values = ('500', 'Kalles gatukök', 'Svensson', 'Karl', '12345', 'Stortorget 4', 'Norrköping', 'Sweden')

mycursor.execute(sql, values)
mydb.commit()

print(mycursor.rowcount, "record inserted.")
```

- Notice the statement: `mydb.commit()`. Depending on settings in your specific database this might be required to make the changes, otherwise no changes are made to the table.

Insert Into

Multiple records

- To insert multiple rows into a table, use the `executemany()` method.
- The second parameter of the `executemany()` method is a list of tuples, containing the data you want to insert.

```
mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
values = [
  ('Peter', 'Lowstreet 4'),
  ('Amy', 'Apple st 652'),
  ('Hannah', 'Mountain 21')
]

mycursor.executemany(sql, values)

mydb.commit()
```

Insert Into

Get Inserted ID

- You can get the id of the row you just inserted by asking the cursor object.
- *Note:* If you insert more than one row, the id of the last inserted row is returned.

```
mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
Values = ("Michelle", "Blue Village")
mycursor.execute(sql, values)

mydb.commit()

print("1 record inserted, ID:", mycursor.lastrowid)
```

Select From a Table

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Select From a Table

Getting a column value, alt 1

- Per default, you get a tuple, which don’t have any keys.
- You can get individual values by referencing to the index.

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x[1])
```

Select From a Table

Getting a column value, alt 2

- To get column names as keys, you can set the `dictionary` option to `True` when creating the cursor.

```
mycursor = mydb.cursor(dictionary=True)

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x["customerName"])
```

Select From a Table

fetchone()

- If you are only interested in one row, you can use the `fetchone()` method.
- The `fetchone()` method will return the first row of the result.

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchone()

print(myresult)
```

Delete

```
mycursor = mydb.cursor()

sql = "DELETE FROM customers WHERE address = 'Mountain 21'"

mycursor.execute(sql)

mydb.commit()
```

Update

```
mycursor = mydb.cursor()

sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"

mycursor.execute(sql)

mydb.commit()
```

Join

```
mycursor = mydb.cursor()

sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
  FROM users \
  INNER JOIN products ON users.fav = products.id"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Exercise

MySQL

- From *classic models*, list the ten most expensive products with name, price and a description of the product line.

API

API

Application Programming Interface

- Att göra ett anrop till en webbsida är lite som att anropa en funktion.
- När vi skriver in en adress i browsern och gör ett *request*, vad innehåller svaret? Vilken typ av data?
- Kan man få andra typer av resultat?

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <title>Random User Generator | Home</title>
      <meta charset='utf-8'>
      <meta name="description" content="Random user generator is a FR
      <meta name="apple-mobile-web-app-capable" content="yes">
      <meta name="viewport" content="width=device-width, initial-scal
      <link rel="stylesheet" type="text/css" href="dist/style.css">
      <script src="dist/all.js" type="fe497a15af4a3b658037ab8b-text/j
   </head>
   <body class="">
      <div id="navbar" class="">
         <div class="nav_toggle">
            <div class="icon">
               <div></div>
               <div></div>
               <div></div>
            </div>
         </div>
         <ul>
            <li><a href="index">Home</a></li>
            <li><a href="photos">User Photos</a></li>
            <li><a href="documentation">Documentation</a></li>
            <li><a href="changelog">Change Log</a></li>
            <li><a href="stats">Stats &amp;amp; Graphs</a></li>
            <li><a href="donate">Donate</a></li>
            <li><a href="copyright">Copyright Notice</a></li>
            <li class="blank"></li>
            <li><a href="photoshop">Photoshop Extension</a></li>
         </ul>
      </div>
      <header>
         <h1>Random User Generator</h1>
         <p>A free, <a id="openSource" href="https://github.com/Rando
         <a href="https://twitter.com/randomapi" class="twitter"><img
      </header>
      <div class="frame card_offset">
         <div class="card">
```

API

Application Programming Interface

- En server behöver inte returnera html, den kan lika gärna returnera andra typer av resultat, såsom bilder, css-filer, pdf-dokument eller json-filer.
- https://randomuser.me/api/
- Låter oss hämta det data vi behöver i ett format som vi kan bearbeta, vanligen JSON eller XML.
- HTML är bra för att märka upp innehåll, JSON och XML är bättre för att strukturera data.


```
{
  "results": [
    {
      "gender": "male",
      "name": {
        "title": "Mr",
        "first": "Daniel",
        "last": "Wright"
      },
      "location": {
        "street": {
          "number": 3216,
          "name": "Port Hills Road"
        },
        "city": "Auckland",
        "state": "Manawatu-Wanganui",
        "country": "New Zealand",
        "postcode": 81215,
        "coordinates": {
          "latitude": "65.4371",
          "longitude": "-143.2856"
        },
        "timezone": {
          "offset": "+6:00",
          "description": "Almaty, Dhaka, Colombo"
```

RESTful API

REpresentational State Transfer

- Det finns många olika typer av API:er.
- Det vi oftast menar och använder kallas RESTful API och är ett sätt att skapa ”samarbetssystem” mellan datorer på internet.
- Från Wikipedia:
  - HTTP-based RESTful APIs are defined with the following aspects:
    - a base URI, such as http://api.example.com/collection/;
    - standard HTTP methods (e.g., GET, POST, PUT, PATCH and DELETE);
- Ett vanligt request-anrop, där vi hämtar information, använder GET.

API-verktyg

- Det finns många verktyg som låter oss testa och arbeta med API:er. Ett av dem är Postman.
- https://www.postman.com/product/api-client/
- Vi kommer inte att gå igenom Postman, men det är ett väldigt bra verktyg om man jobbar mycket med API:er.

API in Python

- You can fetch API data in different way in Python.
- A common way is to use the `requests` library.

```
import requests

response = requests.get("https://randomuser.me/api")

data = response.json()

print(data["results"][0]["location"]["country"])
```

Extra övning

Om tid finnes…

- Välj ut ett API. Om du vill kan du leta inspiration här:
  - http://apikatalogen.se/api
  - https://apilist.fun/
  - https://medium.com/@vicbergquist/18-fun-apis-for-your-next-project-8008841c7be9
  - https://dev.to/biplov/15-fun-apis-for-your-next-project-5053
  - Eller googla.
- Skapa en liten applikation som använder sig av ditt API.
- Extra uppgift: Skapa en liten applikation som använder resultaten från flera API:er. Kolla t ex när det finns en konsert (kanske inte finns så många just nu…) och hitta en buss som går till evenemanget.

Entiteter

- ”Substantiv”, ”saker”
  - Customer, Person, Building, Course mm
- Vilka entiteter behöver vi i vår applikation?
- Vilka egenskaper behöver varje entitet?
- Ser vi några likheter med klasser? JSON?
- Hur vill vi t ex att ett JSON-svar ska se ut när vi frågar efter kunder?

CRUD

- CRUD är fyra vanliga operationer vi ofta vill kunna göra med våra entiteter.
  - Create
  - Read
  - Update
  - Delete
- Det är vanligt att säga att man vill kunna CRUD:a exvis kunder. Det innebär att man vill kunna skapa, läsa, uppdatera och ta bort kunder.

Internet - http

112.39.10.114

2001:0db8:0000:0000:0000:0000:1428:07ab/64

Internet - http - vanliga statusar

- 200: OK
- 301: Moved Permanently
- 401: Unauthorized
- 403: Forbidden
- 404: Not Found
- 500: Internal Server Error

- HyperText Transfer Protocol
  - används för att överföra webbsidor
  - definierar åtta kommandon
    - GET
    - HEAD
    - POST
    - PUT
    - DELETE
    - TRACE
    - OPTIONS
    - CONNECT
  - Svaret från webbservern innehåller en HTTP-statuskod

API

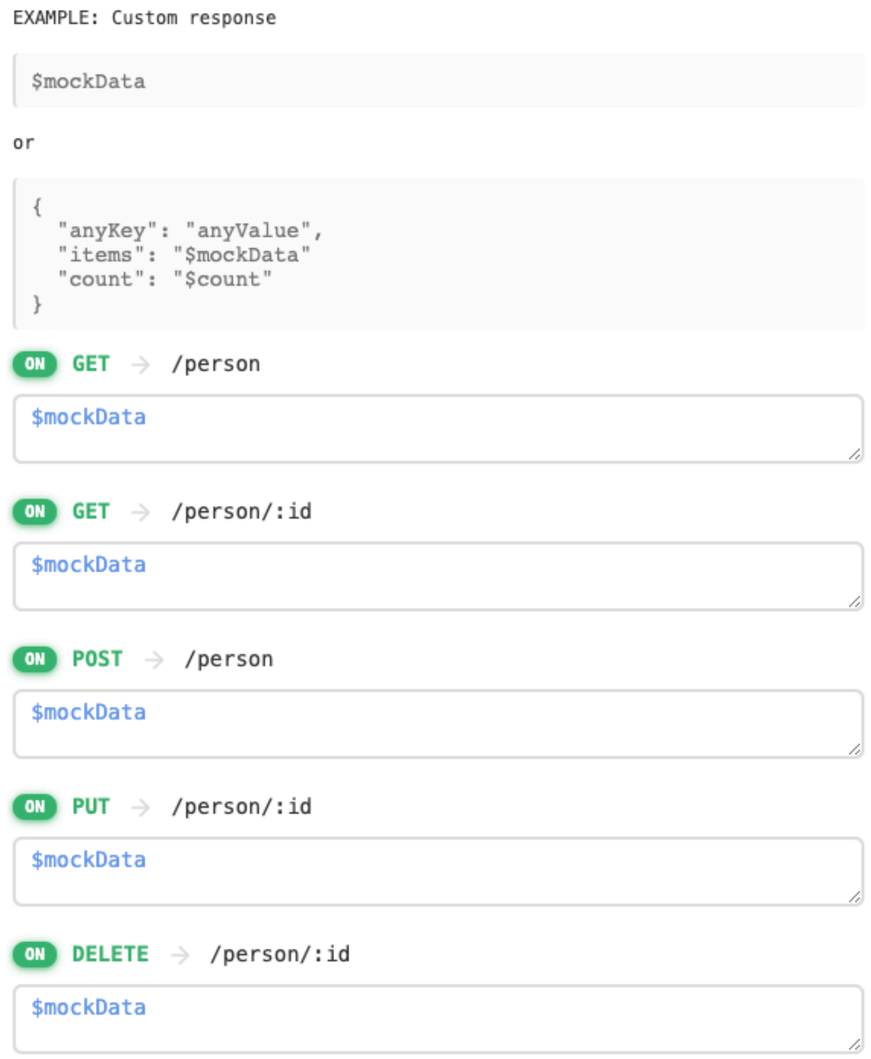
- Tidigare i föreläsningen gjorde vi GET-anrop till https://randomuser.me/api för att hämta data.
- Det finns fler typer av anrop vi kan göra, beroende på vad vi vill göra. Här är några av de vanligaste.

	HTTP verb	CRUD
	POST	Create
	GET	Read
	PATCH / PUT	Update/Replace
	DELETE	Delete

API

- Ett RESTful API skulle exempelvis kunna tillåta följande anrop. Anropen har en hel URI, t ex https://example.com/person.
- `GET /person` - Hämta alla personer
- `GET /person/:id` (ex /person/5) - Hämta en specifik person
- `POST /person` - Skapa person
- `PUT /person/:id` - Spara specifik person
- `DELETE /person/:id` - Ta bort specifik person

Endpoints



API - svar

- När vi gör ett anrop kommer vi att få tillbaka en status-kod och ev ett JSON-svar.

CATEGORY	DESCRIPTION
1xx: Informational	Communicates transfer protocol-level information.
2xx: Success	Indicates that the client's request was accepted successfully.
3xx: Redirection	Indicates that the client must take some additional action in order to complete their request.
4xx: Client Error	This category of error status codes points the finger at clients.
5xx: Server Error	The server takes responsibility for these error status codes.

API - svar

- 200 - OK
- 201 - Created
- 400 - Bad request
- 404 - Not found
- 405 - Method not allowed
- 500 - Internal server error
- 501 - Not implemented

Requests

- You can do different kinds of requests with Python.
- https://realpython.com/python-requests/

```
import requests

url = 'https://www.w3schools.com/python/demopage.php'
myobj = {'somekey': 'somevalue'}

x = requests.post(url, data = myobj)

print(x.text)
```

Requests

Parameter		Description
url	Try it	Required. The url of the request
data	Try it	Optional. A dictionary, list of tuples, bytes or a file object to send to the specified url
json	Try it	Optional. A JSON object to send to the specified url
files	Try it	Optional. A dictionary of files to send to the specified url
allow_redirects	Try it	Optional. A Boolean to enable/disable redirection. Default `True` (allowing redirects)
auth	Try it	Optional. A tuple to enable a certain HTTP authentication. Default `None`
cert	Try it	Optional. A String or Tuple specifying a cert file or key. Default `None`
cookies	Try it	Optional. A dictionary of cookies to send to the specified url. Default `None`
headers	Try it	Optional. A dictionary of HTTP headers to send to the specified url. Default `None`
proxies	Try it	Optional. A dictionary of the protocol to the proxy url. Default `None`
stream	Try it	Optional. A Boolean indication if the response should be immediately downloaded (False) or streamed (True). Default `False`
timeout	Try it	Optional. A number, or a tuple, indicating how many seconds to wait for the client to make a connection and/or send a response. Default `None` which means the request will continue until the connection is closed
verify	Try it Try it	Optional. A Boolean or a String indication to verify the servers TLS certificate or not. Default `True`

Autentisera

- Anledningar:
  - Skydda data
  - Skydda servern från överbelastning
  - Statistik

Basic authentication

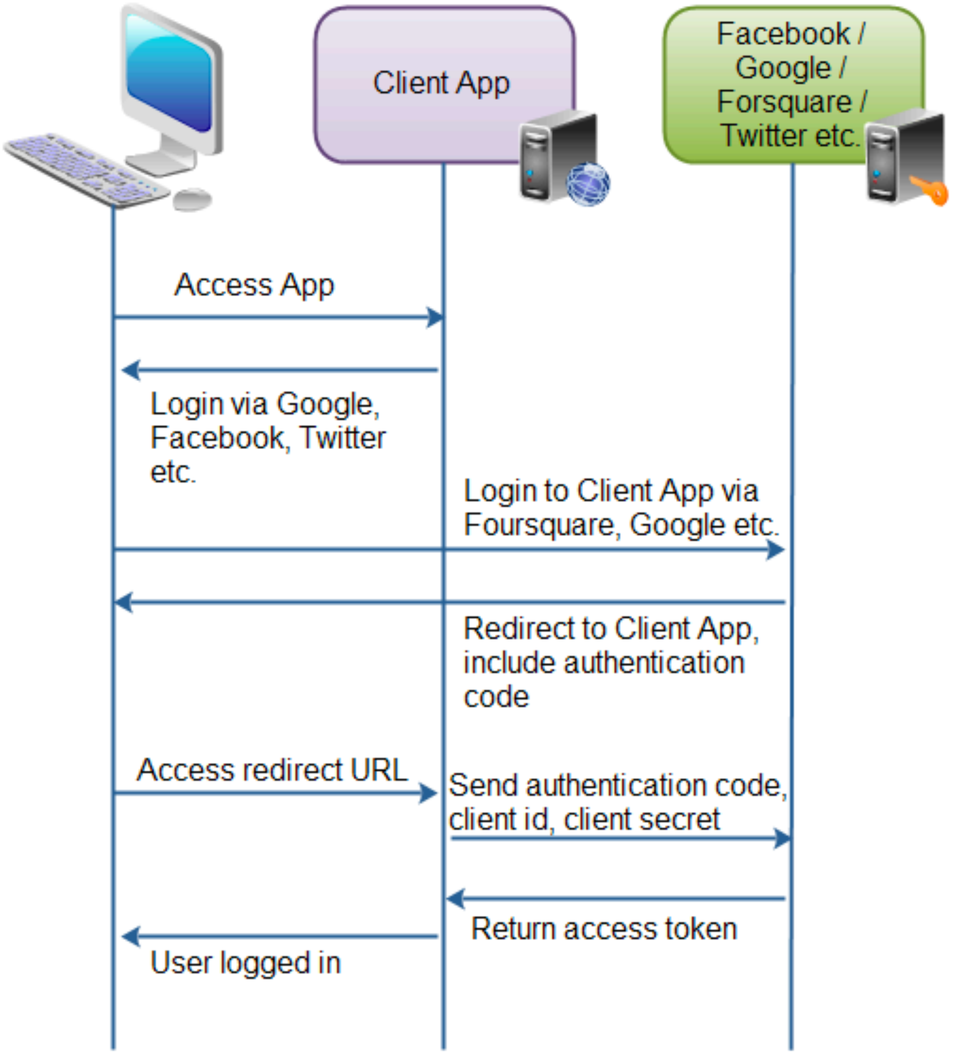
- Metod för att låta en HTTP-användare skicka namn och lösenord i ett request.
- Klienten autentiserar sig genom en header
  - `Authorization: Basic <credentials>`
  - Credentials är en base64-kodad sträng bestående av användare:lösenord.

API-key

- Ganska simpelt och vanligt sätt att hantera åtkomst.
- Varje användare / konto får en nyckel som används i varje request.
- Anges oftast som en parameter eller i en header.
- Låt oss kolla hur detta kan se ut i Postman.

Oauth2

- Ett protokoll för att låta applikationer komma åt varandras data.
- Ganska vanligt, men en nivå knepigare att förstå.
- Oauth1 och Oauth2 är inte kompatibla.
- Detta område går utanför kursen, men var åtminstone medvetna om att man kan behöva autentisera sig för att få komma åt vissa resurser.



Fejk-API

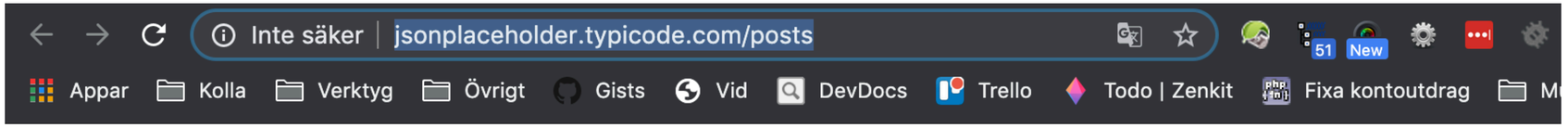
- Det finns många siter som agerar som API:er men som inte låter en exempelvis skapa poster på riktigt.
- Kan användas för tester.
- https://jsonplaceholder.typicode.com/
- Vad kan jag göra med listade endpoints?

Routes

All HTTP methods are supported.

GET	/posts
GET	/posts/1
GET	/posts/1/comments
GET	/comments?postId=1
GET	/posts?userId=1
POST	/posts
PUT	/posts/1
PATCH	/posts/1
DELETE	/posts/1

GET /posts




```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut qua
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blandit
nisi nulla"
  },
  {
    "userId": 1,
    "id": 3,
    "title": "ea molestias quasi exercitationem repellat qui ipsa sit aut",
    "body": "et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusanti
  },
  {
    "userId": 1,
    "id": 4,
    "title": "eum et est occaecati",
    "body": "ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda provident rerum culpa\nquis h
  },
  {
    "userId": 1,
    "id": 5,
    "title": "nesciunt quas odio",
    "body": "repudiandae veniam quaerat sunt sed\nalias aut fugiat sit autem sed est\nvoluptatem omnis possimus es
  }
```

Exercise

- Get currency exchange rates.
  - https://exchangeratesapi.io/
    - You'll need to get a free API key to use this resource. Create an account at the site to obtain it.
    - Read the documentation.

  - It might be a good idea to try out your requests with Postman or similar to verify your results.
  - Make a small application that asks user for a currency, make a request to the API to find out the current exchange rate and print it out.

More exercises

If there is time

- See if you can find something fun and / or meaningful to do with these APIs.
  - Verify phone no:
    https://numverify.com/
  - Get Chuck Norris jokes:
    http://www.icndb.com/api/
  - List job opportunities from Github (PHP/San Fransisco if you want an example):
    https://jobs.github.com/api
- Other resources:
  https://github.com/toddmotto/public-apis

Felsökning

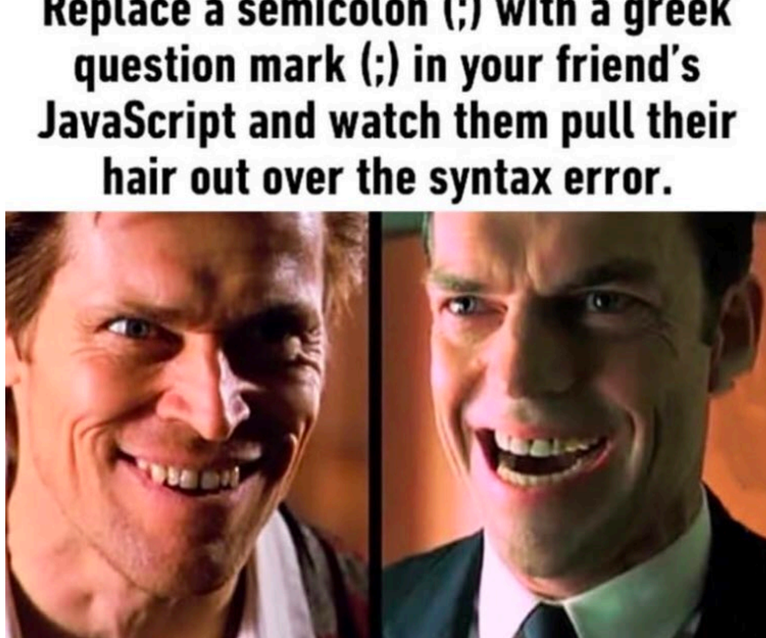
- Att hitta fel i en sida kan ofta vara komplicerat, speciellt i större projekt eller ramverk.
  - Många filer inkluderas, både interna och externa (t ex tredjepartsbibliotek).

Felsökning

- Vad är en bugg?
  - Syntax-fel
  - Logiska fel
  - Exekveringsfel
  - Feature

Felsökning

- Syntax-fel
  - När man använt språket på fel sätt, när Python-tolken inte förstår våra instruktioner.
  - Skrivfel
  - Utelämnade tecken, osynliga tecken
  - Felstavade identifierare
    (variabler, funktioner, konstanter klasser)
  - Odefinierade variabler



Felsökning

- Logiska fel
  - Vi använder språket korrekt men får inte det resultat vi har tänkt oss, t ex svar som blir fel bara för en viss sorts indata (specialfall).
  - Ett logikfel är ett fel i ett program som gör att det fungerar felaktigt, men inte att det avslutas onormalt (kraschar). Ett logikfel ger oönskad utdata eller annat beteende.

Felsökning

- Exekveringsfel (krasch)
  - Dividera med noll
  - Array utanför index
  - Oändliga loopar
  - Minnesläckor

```c
#include <stdio.h>

int main(void)
{
   int a = 0;
   while (a < 10) {
      printf("%d\n", a);
      if (a = 5)
         printf("a equals 5!\n");
      a++;
   }
   return 0;
}
```

```java
class Example {
     public static void main(String[] args) {
          char[] matrix = new char[] {'H', 'e', 'l', 'l', 'o'};
          System.out.println(matrix);

          /* Print each letter of the char array in a separate line. */
          for(int i = 0; i <= matrix.length; ++i) {
               System.out.println(matrix[i]);
          }
     }
}
```

Felsökning

- Feature
  - Ibland är en bugg inte en bugg utan ett missförstånd i vad en funktion verkligen ska göra.
  - Användaren förväntar sig att något ska hända som systemet inte är tänkt att hantera.
  - Viktigt att tänka på vid gränssnittsdesign.

Exempel

- Koda i fel fil/fel miljö
- Väder-mätstationen som slutade fungera varje kväll
- Backup-servern som gick ner

Felsökning

- Hur hittar man en bugg?
  - Isolering & uteslutning
  - Spårutskrifter
  - Debug-verktyg

Isolering & uteslutning

- Att isolera och utesluta buggar innebär att man testar så mycket man kan i både gränssnitt och kod för att ta bort faktorer som kan påverka problemet.
- Det kan t ex vara att testa vilka data som krävs för att återskapa buggen.
  - Är det bara om funktionen får en viss typ av data? (Sträng, int, negativt tal osv.)
  - Uppstår buggen bara under vissa omständigheter?
  - Kan man kommentera bort kodrader och se om problemet fortfarande kvarstår?

Spårutskrifter

- Man kan skriva ut variabler, uppbyggda strängar osv för att undersöka vad de innehåller.
- `print(variable);`

Felsöka i VS Code

- https://code.visualstudio.com/docs/editor/debugging
- Vi tittade på debug-verktyget på första föreläsningen.

Group assignment

- This assignment will combine some of the methods we have learned about today.
- You will do this exercise together in your groups and hand it in by Sunday.
  - Ask the user for a customerNumber.
  - Get all orders for that company from the MySQL database classic models.
  - For every order, we will get first name, last name, email and phone no from a random user from https://randomuser.me/api. (We'll pretend that this is actually an order system or something.)
  - Produce a new CSV file that contains suitable headers and where each row contains order number, order date, total order value and customer information (from the random user API).

Python
Övningar

- https://www.w3schools.com/python/exercise.asp
- https://www.w3resource.com/python-exercises/