From Client to DB

From client to DB and back again

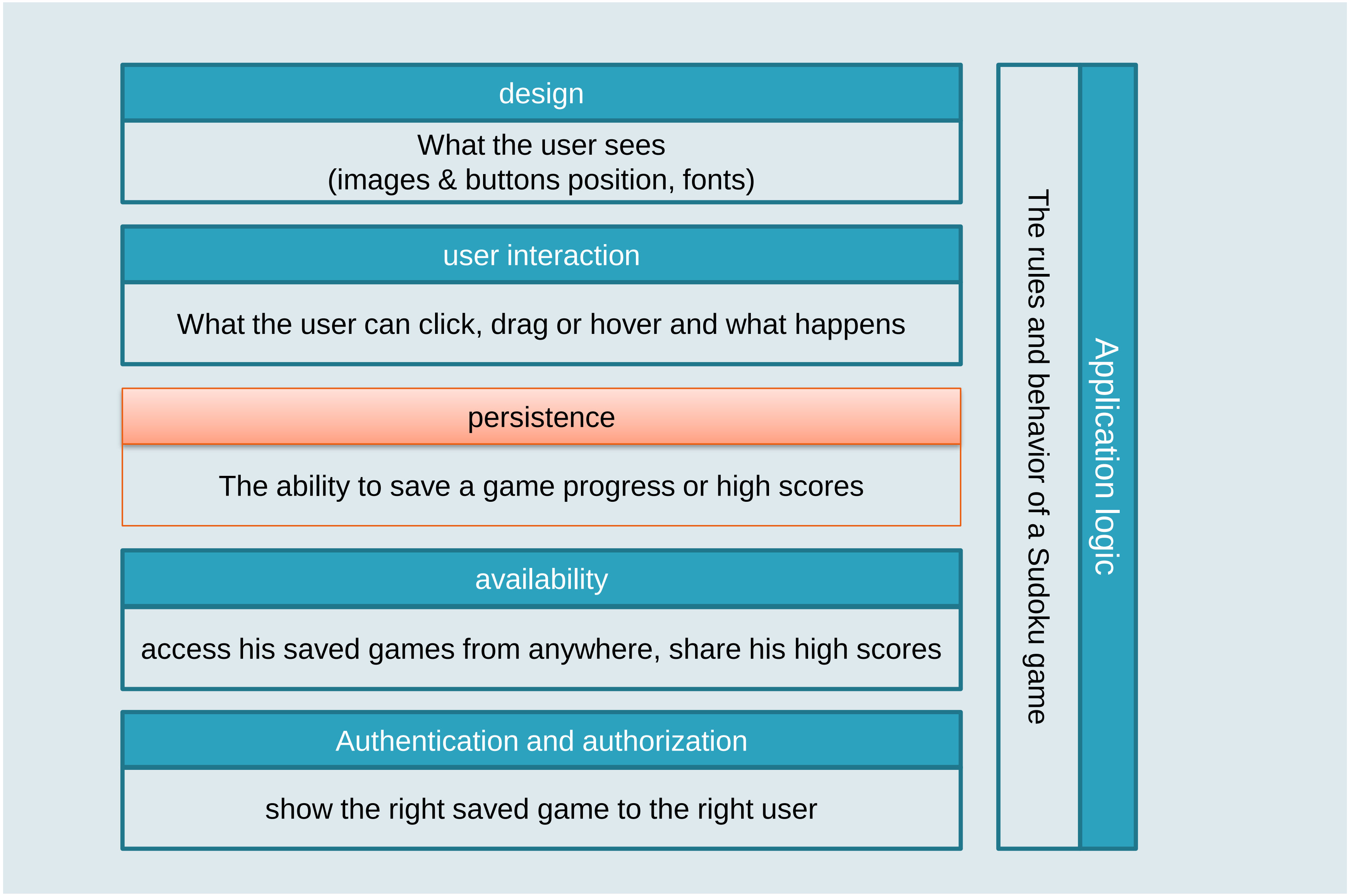
```
from os import getenv
import pymysql

server = getenv("PYMSSQL_TEST_SERVER")
user = getenv("PYMSSQL_TEST_USERNAME")
password = getenv("PYMSSQL_TEST_PASSWORD")

conn = pymysql.connect(server, user=user,
password=password,
 database="tempdb")
cursor = conn.cursor()
cursor.execute("""
IF OBJECT_ID('persons', 'U') IS NOT NULL
    DROP TABLE persons
CREATE TABLE persons (
    id INT NOT NULL,
    name VARCHAR(100),
    salesrep VARCHAR(100),
    PRIMARY KEY(id)
)
""")
```

```
cursor.executemany(
    "INSERT INTO persons VALUES (%d, %s, %s)",
    [(1, 'John Smith', 'John Doe'),
     (2, 'Jane Doe', 'Joe Dog'),
     (3, 'Mike T.', 'Sarah H.')])
# you must call commit() to persist your data if you don't
set autocommit to True
conn.commit()

cursor.execute('SELECT * FROM persons WHERE
salesrep=%s', 'John Doe')
row = cursor.fetchone()
while row:
    print("ID=%d, Name=%s" % (row[0], row[1]))
    row = cursor.fetchone()

conn.close()
```

Agenda

- ❖ Persistence
- ❖ Exception handling (general but also In DB scope)
- ❖ Try catch
- ❖ Python "with" statement
- ❖ Cursors and connections
- ❖ CRUD

Persistence!


design
What the user sees
(images & buttons position, fonts)
user interaction
What the user can click, drag or hover and what happens
persistence
The ability to save a game progress or high scores
availability
access his saved games from anywhere, share his high scores
Authentication and authorization
show the right saved game to the right user
The rules and behavior of a Sudoku game
Application logic

Persistence!

Once the user reloads the browser, closes the browser, even if we turn off the server etc.

In computer science, **persistence** refers to the characteristic of **state that outlives the process that created it**.

This is achieved in practice by **storing the state as data in computer data storage.**

In our case MySQL

Pymysql (the subject of this lecture)

Programs have to transfer data to and from storage devices and have to provide **mappings** from the native programming-language data structures to the storage device data structures

Converting a dict/JSON/text to DB query and the other way around

End to End

When starting a new (full stack) project,

The most basic test of the system is to see the data flow "from end to end".

Our Stack:

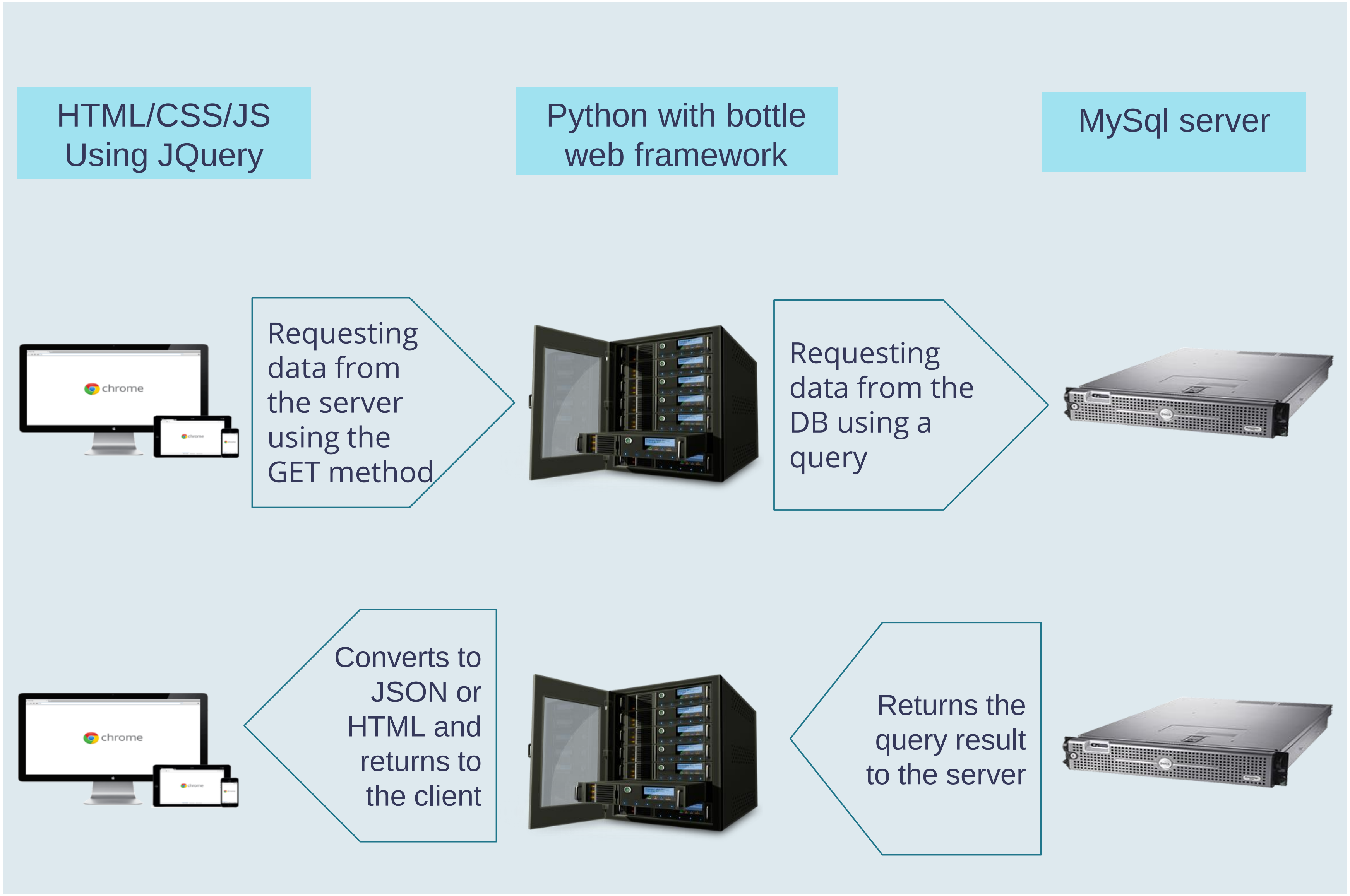
HTML/CSS/JS
Using JQuery

Python with bottle
web framework

MySql server

Clients, Servers and DB server appearance may vary

End to End Flow



HTML/CSS/JS Using JQuery

Python with bottle web framework

MySql server

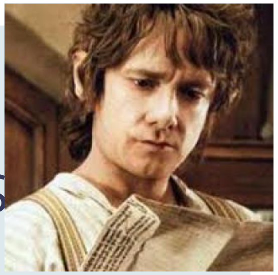
Requesting data from the server using the GET method

Requesting data from the DB using a query

Converts to JSON or HTML and returns to the client

Returns the query result to the server

So far...



- We know how to create a client side UI using HTML and CSS
- We know how to write JS (with JQuery) code to make it interactive
- We know how to set-up a pythonic web server (called Bottle)
- We know how to serve the client with HTML and static files (images/css/js/fonts)
- We know how to call the server using ajax and get a JSON response
- We know how to set up a MySql database on our computer and run queries from the workbench

In order to get "full stack" we need to know how to access the DB from our python server

CRUD

The DB usually holds entities of our application

Users

Tweets,

Products

etc.

What services are required from a DB in an application?

CRUD

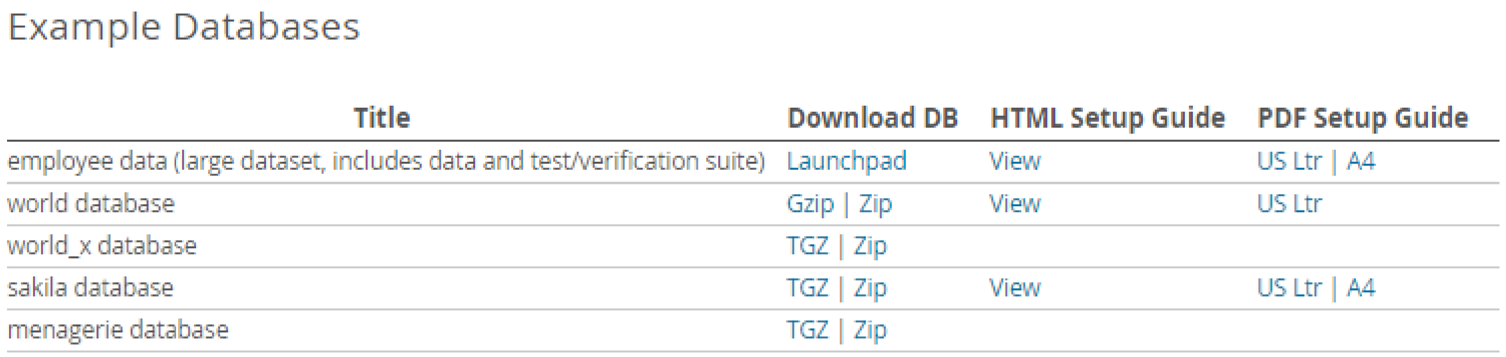
We need a way to:

- Create a new entity (like registering a user or writing a new tweet)
- Read the details of a specific entity
(like User email, tweet content, product price and description)
- Update a specific entity (changing a user’s password)
- Delete a specific entity (I never do that… ask me why and when)
- List all the Entities available (sometimes with a filter)

Example – World Database

For my next trick I will need a database…

Searching google I found the following DB under the MySql website…



Example Databases

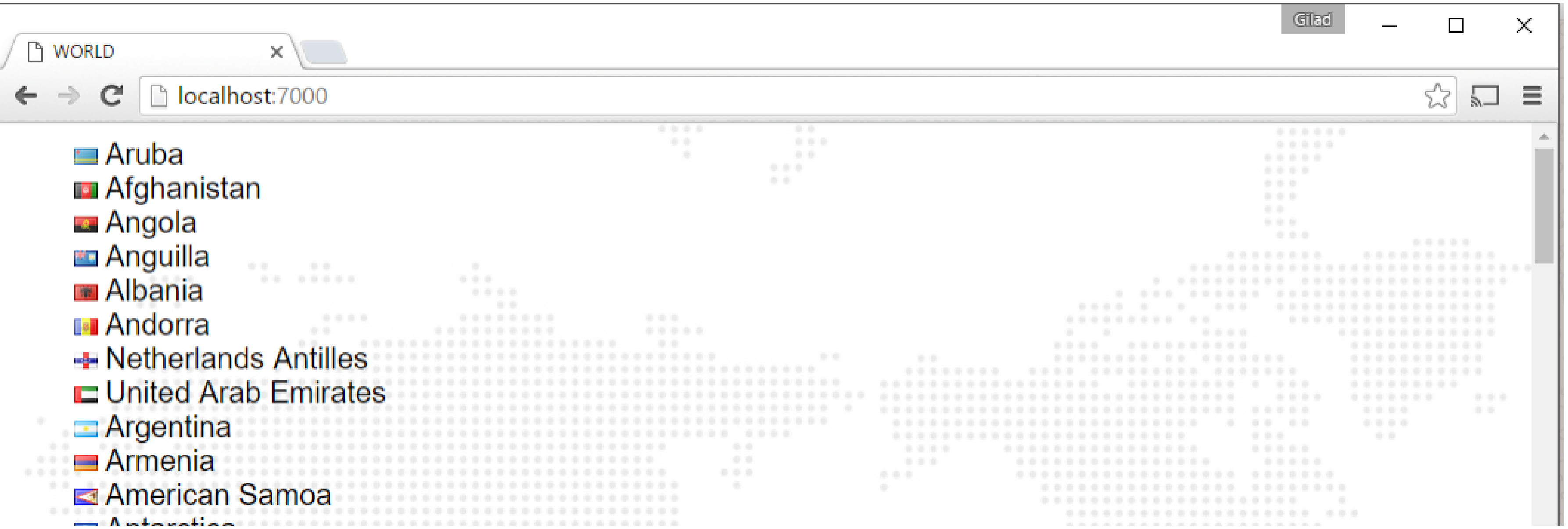
Title	Download DB	HTML Setup Guide	PDF Setup Guide
employee data (large dataset, includes data and test/verification suite)	Launchpad	View	US Ltr | A4
world database	Gzip | Zip	View	US Ltr
world_x database	TGZ | Zip		
sakila database	TGZ | Zip	View	US Ltr | A4
menagerie database	TGZ | Zip		

Example – World Database

Our application will include a list of all the countries in the world along with their flags

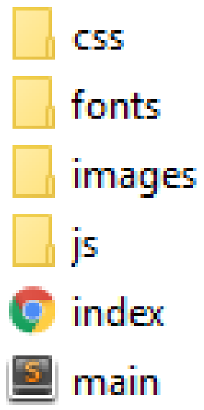
The list of the countries will be fetched from the DB

Clicking a country will display the country's population



- Aruba
- Afghanistan
- Angola
- Anguilla
- Albania
- Andorra
- Netherlands Antilles
- United Arab Emirates
- Argentina
- Armenia
- American Samoa

The Folder Structure



css
fonts
images
js
index
main

Python file (bottle server)

The HTML

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta charset="UTF-8">
        <title>WORLD</title>
        <link rel="stylesheet" href="./css/common.css">
        <script src="https://code.jquery.com/jquery-2.1.4.min.js" type="text/javascript"></script>
        <script src="./js/world.js" type="text/javascript"></script>
    </head>
    <body>
        <div id="content">loading...</div>
    </body>
</html>
```

Looks like we are going to generate a lot of things dynamically...

The CSS

```
#content{
    background: □rgba(255,255,255,0.7);
    font-size: 20px;
    font-family: Arial;
    padding:10px;
}

ul{
    list-style: none;
    margin:0px;
}

.country-name{
    margin-left:5px;
}
```

The JS

```
var World = {};

World.start = function(){
    $(document).ready(function() {
        var contentHolder = $("#content");
        World.loadCountries(contentHolder);
    });
};

World.loadCountries = function(contentHolder){
    $.get("/list_countries",function(countries){
            if ('error' in countries){
                alert(countries.error);
            }else{
                var countriesHolder = $("<ul>");
                for (i in countries){
                    var countryEntry = $("<li />").addClass("country-entry clickable");
                    var imgSrc = "/images/flags/" + countries[i].Code2 + ".png";
                    var countryFlag = $("<img />").attr("src",imgSrc);
                    var countryName = $("<span />").addClass("country-name").text(countries[i].Name);
                    countryEntry.append(countryFlag).append(countryName)
                    countriesHolder.append(countryEntry);
                }
                contentHolder.empty().append(countriesHolder);
            }
        },"json");
};

World.start();
```

What is the data structure we expect here?

A list/array → [

{"Name":"Aruba","Code2":"ar"

Of objects/dict → {"Name":"Israel","Code2":"il"},

.

.

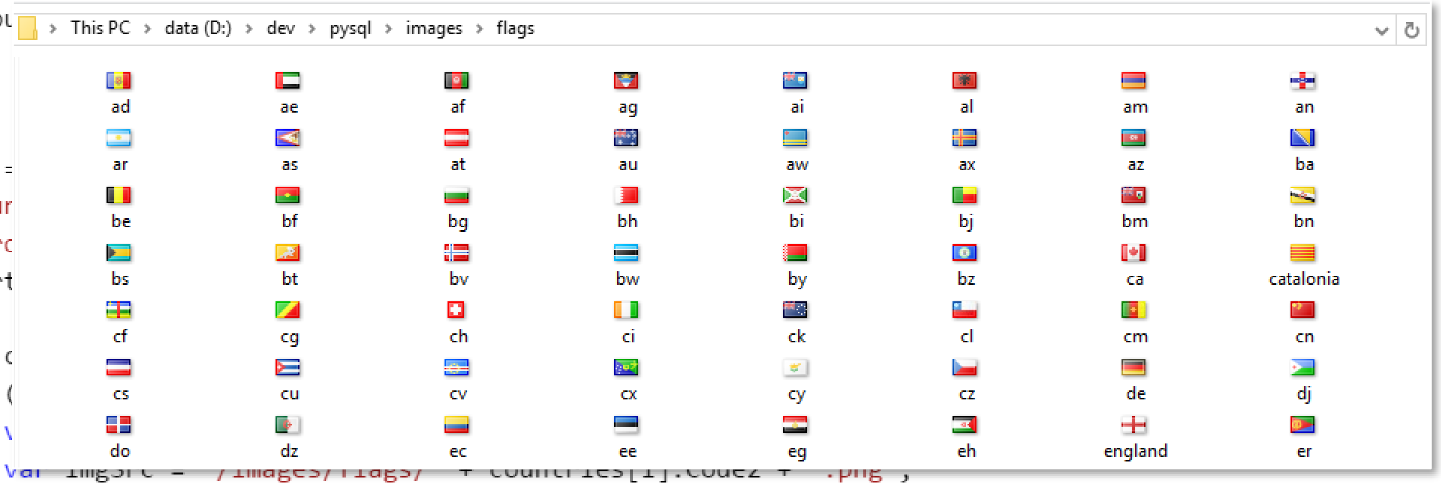
.

]

The JS

```
var World = {};

World.start = function(){
    $(document).ready(function() {
        var contentHolder = $("#content");
        World.loadCou
    });
};

World.loadCountries =
    $.get("/list_cour
            if ('erro
                alert
            }else{
                var c
                for (
                    v
                    var imgSrc = "/images/flags/" + countries[i].code2 + ".png";
                    var countryFlag = $("<img />").attr("src",imgSrc);
                    var countryName = $("<span />").addClass("country-name").text(countries[i].Name);
                    countryEntry.append(countryFlag).append(countryName)
                    countriesHolder.append(countryEntry);
                }
                contentHolder.empty().append(countriesHolder);
            }
        },"json");
};

World.start();
```

What is the content of the /images folder?



This PC › data (D:) › dev › pysql › images › flags

ad	ae	af	ag	ai	al	am	an
ar	as	at	au	aw	ax	az	ba
be	bf	bg	bh	bi	bj	bm	bn
bs	bt	bv	bw	by	bz	ca	catalonia
cf	cg	ch	ci	ck	cl	cm	cn
cs	cu	cv	cx	cy	cz	de	dj
do	dz	ec	ee	eg	eh	england	er

The py

```python
@get("/")
def index():
    return template("index.html")


@get('/js/<filename:re:.*\.js>')
def javascripts(filename):
    return static_file(filename, root='js')


@get('/css/<filename:re:.*\.css>')
def stylesheets(filename):
    return static_file(filename, root='css')


@get('/images/<filename:re:.*\.(jpg|png|gif|ico)>')
def images(filename):
    return static_file(filename, root='images')


run(host='localhost', port=7000)
```

Just a regular bottle server (So far..)
But we are looking for this:

```javascript
World.loadCountries = function(contentHolder){
    $.get("/list_countries",function(countries){
```

The py

Let’s ignore the new syntax for a moment and see what we can infer on the DB structure

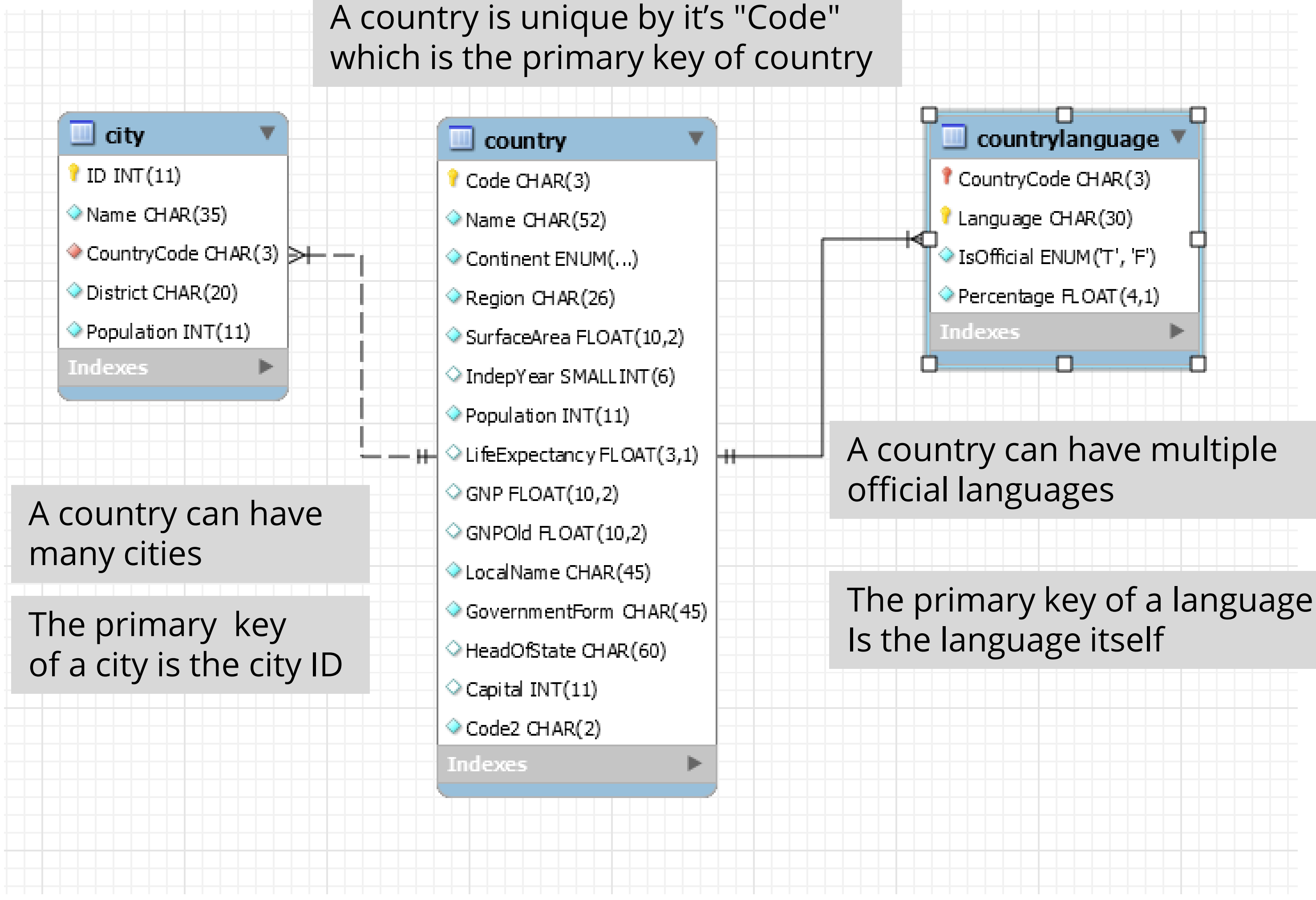
```
@get("/list_countries")
def list_countries():
    try:
        with connection.cursor() as cursor:
            sql = "SELECT Name , Code2 FROM country"
            cursor.execute(sql)
            result = cursor.fetchall()
            return json.dumps(result)
    except:
        return json.dumps({'error':'something is wrong with the DB'})
```

We have a table named country

With at least two columns , "Name" and "Code2"

The DB

Before we get back to the python… the DB (database) ERD (entity relation diagram):



A country is unique by it's "Code" which is the primary key of country

A country can have many cities

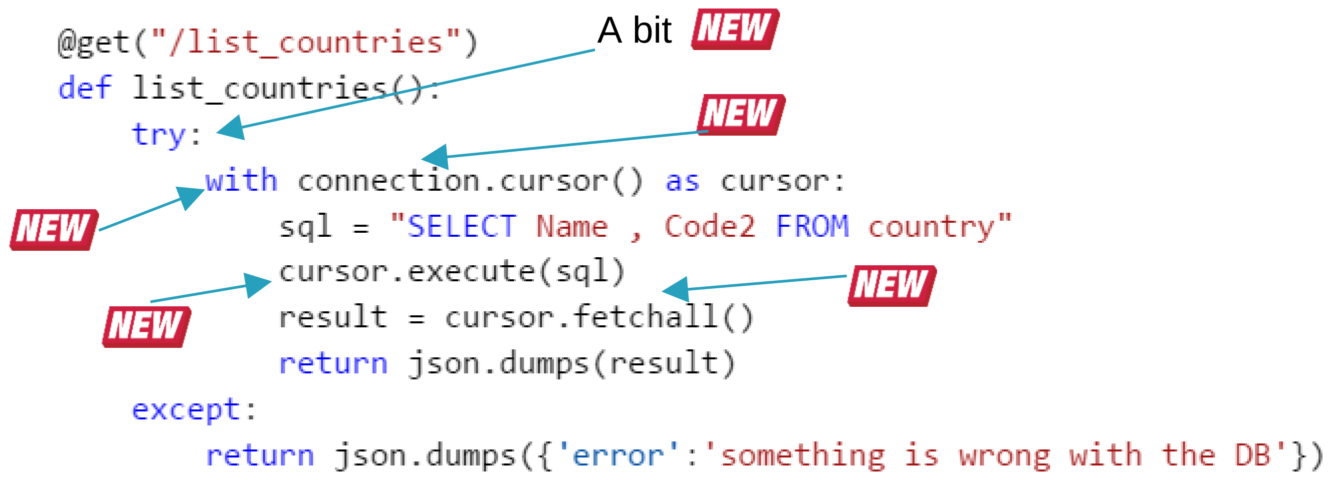
The primary key of a city is the city ID

A country can have multiple official languages

The primary key of a language Is the language itself

The py

Aaaaand back to python



```
@get("/list_countries")
def list_countries():
    try:
        with connection.cursor() as cursor:
            sql = "SELECT Name , Code2 FROM country"
            cursor.execute(sql)
            result = cursor.fetchall()
            return json.dumps(result)
    except:
        return json.dumps({'error':'something is wrong with the DB'})
```

A Few Words on Runtime Errors

A Few Words on Runtime Errors

- Runtime errors occur when the compiler (yes we compile code in python too) has checked the code the best it can… but still the programmer has left a few doorways for it to crash…
- What are these doorways?
- What is a crash?
- One widespread doorway is allowing the user is to input any data he or she wants (the little brother test)

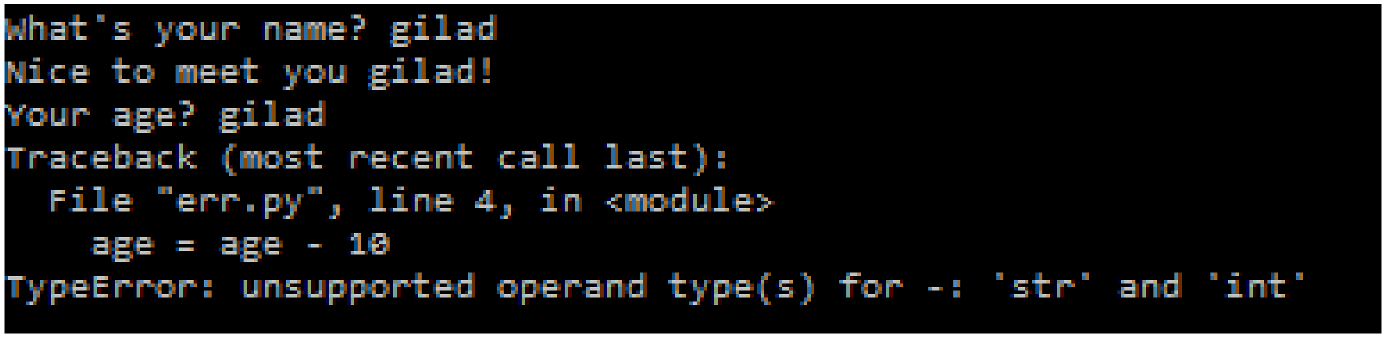
A Few Words on Runtime Errors

What is the doorway in the following code?

```
name = input("What's your name? ")
print("Nice to meet you " + name + "!")
age = input("Your age? ")
age = age - 10
print("Wow but you look " + age + " years old, " + name + "!")
```

Smash head on keyboard to continue!

The little brother hits the keyboard and only inputs characters

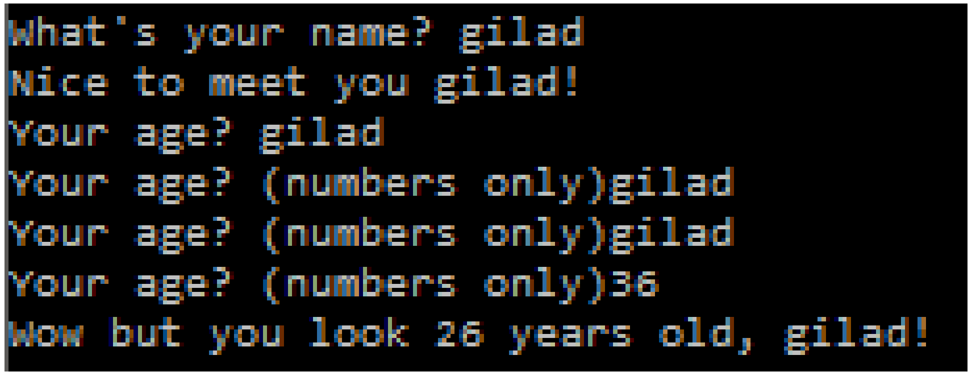


```
What's your name? gilad
Nice to meet you gilad!
Your age? gilad
Traceback (most recent call last):
  File "err.py", line 4, in <module>
    age = age - 10
TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

A Few Words on Runtime Errors

How can these be prevented?

```
name = input("What's your name? ")
print("Nice to meet you " + name + "!")
age = input("Your age? ")
while not age.isnumeric():
    age = input("Your age? (numbers only)")
age = int(age) - 10
print("Wow but you look " + str(age) + " years old, " + name + "!")
```

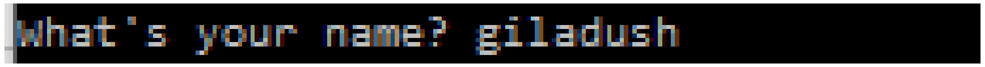


```
What's your name? gilad
Nice to meet you gilad!
Your age? gilad
Your age? (numbers only)gilad
Your age? (numbers only)gilad
Your age? (numbers only)36
Wow but you look 26 years old, gilad!
```

A Few Words on Runtime Errors

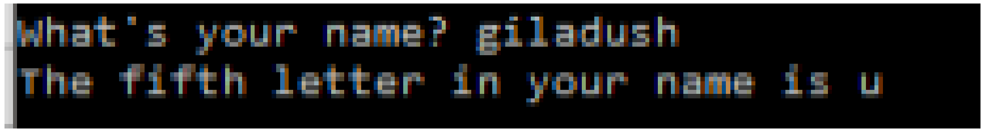
What will the runtime error be?

```
name = input("What's your name? ")
fifthLetter = name[5]
print("The fifth letter in your name is " + fifthLetter)
```



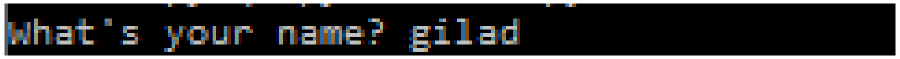
```
What's your name? giladush
```

No error…

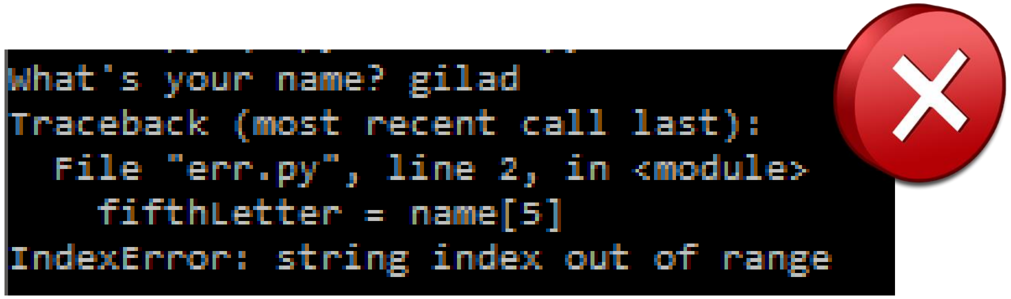


```
What's your name? giladush
The fifth letter in your name is u
```

And now?



```
What's your name? gilad
```



```
What's your name? gilad
Traceback (most recent call last):
  File "err.py", line 2, in <module>
    fifthLetter = name[5]
IndexError: string index out of range
```

A Few Words on Runtime Errors

This also can be prevented

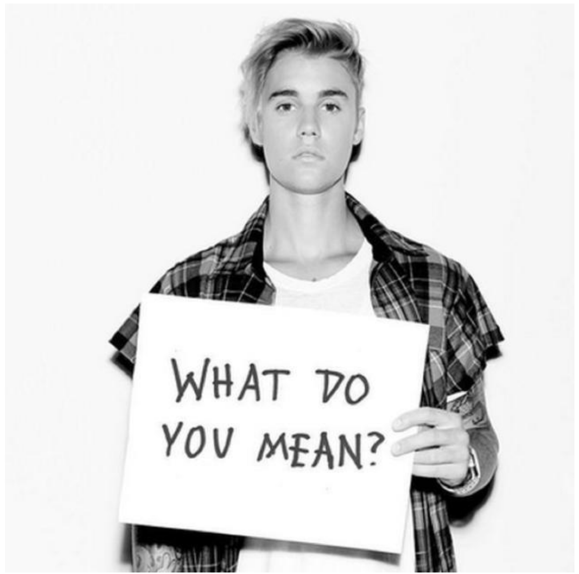
```
name = input("What's your name? ")
if len(name) > 5:
    fifthLetter = name[5]
    print("The fifth letter in your name is " + fifthLetter)
else:
    print("You don't have five letters in your name")
```

But according to the Zen of python,
We need to trust our input and write less defensive code

A Few Words on Runtime Errors

The Python exception handling mechanism allows us to decide how and on which level to handle the exception.

What do you mean by "which level"?



A Few Words on Runtime Errors

One of the flaps of a Boeing plane got stuck

The flap identifies the problem and report to flap array control program (because it doesn’t "know" the state of the other flaps)

The flaps array control program has no data regarding wind/altitude/speed so it doesn’t know how to set the rest of the flaps. this is why it will update the flight control module.

The flight control system in the cockpit receives the update and raises the air brakes on the opposite side

It does not alert the pilot

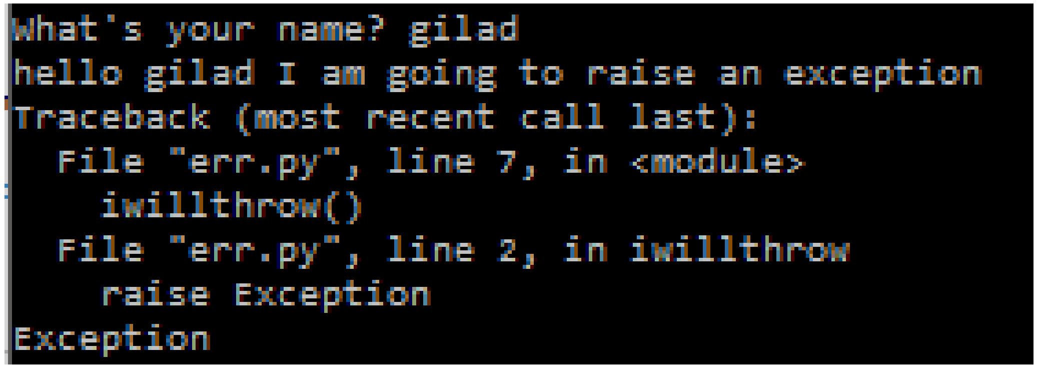
A Few Words on Runtime Errors

If a function encounters an exception it can do one of three things:

- Abort the execution of the function and raise (throw) the exception to the function that called it
- Handle the exception but raise the exception anyway, so that the caller is aware of the problem (for logging purposes, for example).
- Handle the exception and silence it.

Raise Exception

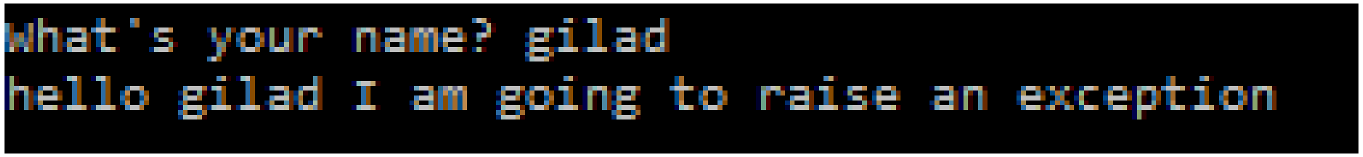
```
def iwillthrow():
    raise Exception


name = input("What's your name? ")
print("hello " + name + " I am going to raise an exception")
iwillthrow()
```



```
What's your name? gilad
hello gilad I am going to raise an exception
Traceback (most recent call last):
  File "err.py", line 7, in <module>
    iwillthrow()
  File "err.py", line 2, in iwillthrow
    raise Exception
Exception
```

Silence an Exception

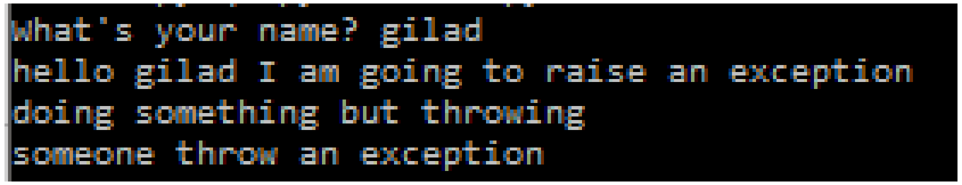
```python
def iwillthrow():
    raise Exception


name = input("What's your name? ")
print("hello " + name + " I am going to raise an exception")
try:
    iwillthrow()
except:
    pass
```



```
What's your name? gilad
hello gilad I am going to raise an exception
```

Handle Exception and Raise

```python
def iwillthrow():
    raise Exception

def iwillhandleandthrow():
    try:
        iwillthrow()
    except:
        print("doing something but throwing")
        raise Exception



name = input("What's your name? ")
print("hello " + name + " I am going to raise an exception")
try:
    iwillhandleandthrow()
except:
    print("someone throw an exception")
```

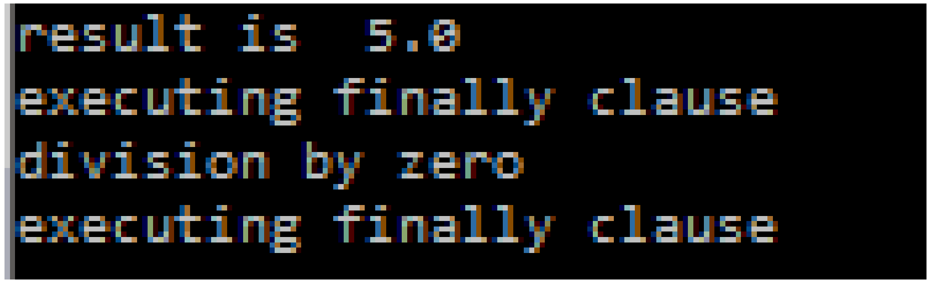


```
What's your name? gilad
hello gilad I am going to raise an exception
doing something but throwing
someone throw an exception
```

A Few Words on Runtime Errors

`except`, **`else`** and **`finally`** clauses

```python
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")
divide(10, 2)
divide(10, 0)
```

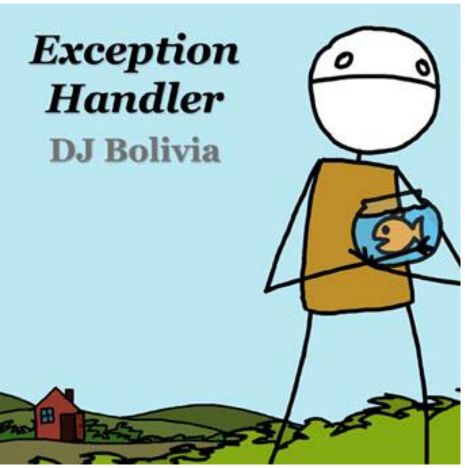


```
result is  5.0
executing finally clause
division by zero
executing finally clause
```

A Few Words on Runtime Errors

Back to our example:

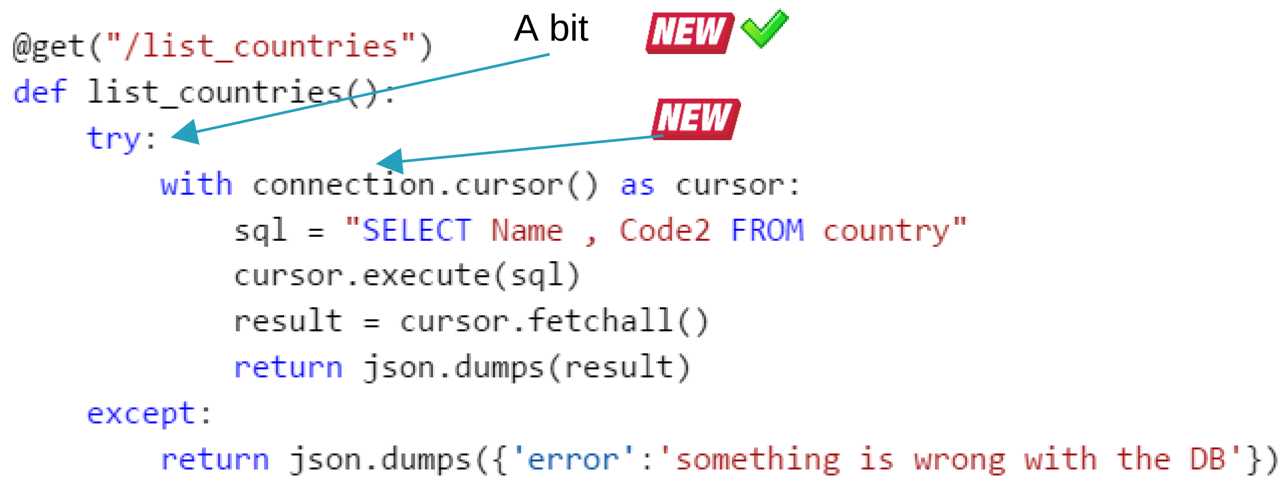
```
name = input("What's your name? ")
try:
    print("The fifth letter in your name is " + name[5])
except IndexError:
    print("You don't have five letters in your name")
```



Questions?

The py

Back to Python again...



```
@get("/list_countries")
def list_countries():
    try:
        with connection.cursor() as cursor:
            sql = "SELECT Name , Code2 FROM country"
            cursor.execute(sql)
            result = cursor.fetchall()
            return json.dumps(result)
    except:
        return json.dumps({'error':'something is wrong with the DB'})
```

A connection is our way to "talk" to the DB

Connecting to the DB

Creating a connection is rather easy:

```
# Connect to the database
connection = pymysql.connect(host='localhost',
                             user='bilbo',
                             password='myprecious',
                             db='world',
                             charset='utf8',
                             cursorclass=pymysql.cursors.DictCursor)
```

We just need to know the user, the password, the DB name and the charset in which strings are encoded as data

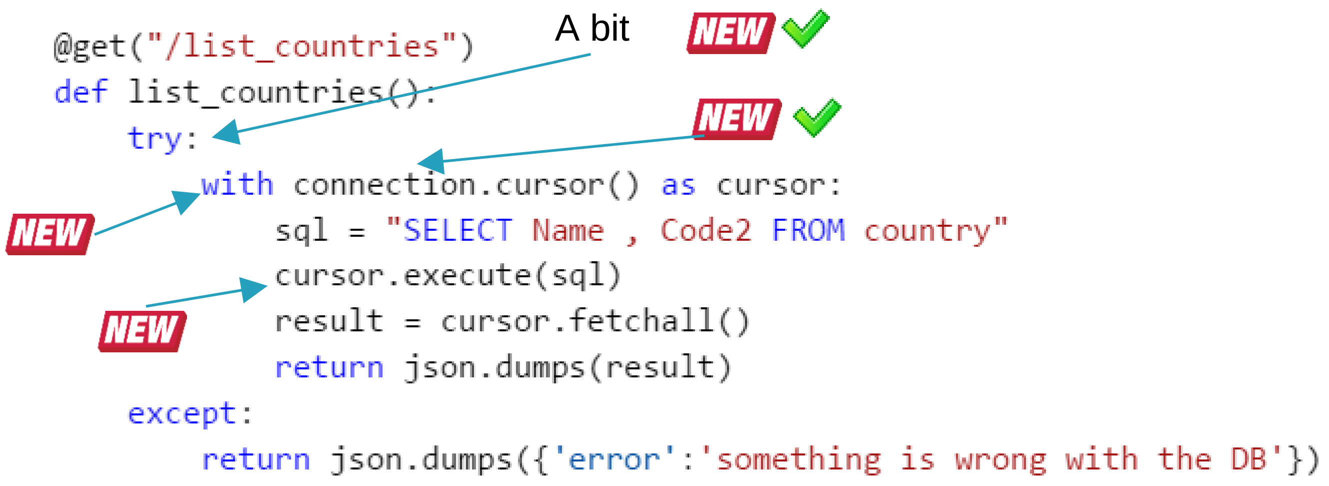
We must remember to import **pymsql**

```
import pymysql
```

That is not a built in library in python so **pip install** it first…

The py

Before understanding the "with" statement we need to know what a cursor is



```
@get("/list_countries")
def list_countries():
    try:
        with connection.cursor() as cursor:
            sql = "SELECT Name , Code2 FROM country"
            cursor.execute(sql)
            result = cursor.fetchall()
            return json.dumps(result)
    except:
        return json.dumps({'error':'something is wrong with the DB'})
```

The DB Cursor

A database cursor is an object used to traverse records in a database. Just like a typing cursor is used to signify where new text will appear on screen, a database cursor indicates the specific record in the database which is currently being worked upon.
Cursors are created in relation to a DB connection

A cursor can run any SQL query and iterate through the results

The DB Cursor

In our case we create a query to select the name and code2 from all of the countries in the DB

```
@get("/list_countries")
def list_countries():
    try:
        with connection.cursor() as cursor:
            sql = "SELECT Name , Code2 FROM country"
            cursor.execute(sql)
            result = cursor.fetchall()
            return json.dumps(result)
    except:
        return json.dumps({'error':'something is wrong with the DB'})
```

We call execute to, hmm, execute the query

We read all the entries from the result

The py

A bit NEW ✓ NEW ✓

NEW

NEW ✓

```
@get("/list_countries")
def list_countries():
    try:
        with connection.cursor() as cursor:
            sql = "SELECT Name , Code2 FROM country"
            cursor.execute(sql)
            result = cursor.fetchall()
            return json.dumps(result)
    except:
        return json.dumps({'error':'something is wrong with the DB'})
```

The "with" Statement

When a database query is performed, the cursor points to the first record in the query’s results set. Using various commands, the cursor can move to any location within the results set.

The developer must take care not to have too many open cursors. Each cursor uses an (admittedly small) amount of memory. However, if cursors are never closed (i.e., discarded after serving their purpose), they can pile-up in memory and, with time, cause performance problems, up to a crash.

Bottom line we need to close the cursor when we finish working with it

The with statement

Because pretty is better than ugly,
Python allows developers to define special methods inside any object (like the cursor object)

One method is invoked when we start working with the object and other other when we finished working with it.
The methods are called **`__enter__`** and **`__exit__`** respectively
When we use the "with" statement, Python automatically invokes the **`__enter__`** function upon entering the "with" block, and **`__exit__`** when we exit the "with" block.

The "with" Statement

So instead of writing something like

```
cursor = connection.cursor()
cursor.init()
#dosomethingwithcursor...
cursor.close()
```

We can do this:

```
with connection.cursor() as cursor:
    #dosomethingwithcursor...
```

The py

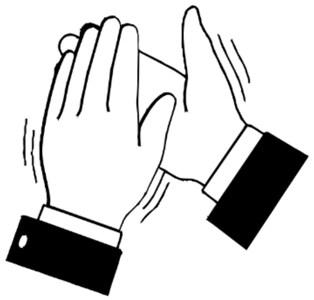
A bit NEW ✓

NEW ✓

NEW ✓

NEW ✓

```
@get("/list_countries")
def list_countries():
    try:
        with connection.cursor() as cursor:
            sql = "SELECT Name , Code2 FROM country"
            cursor.execute(sql)
            result = cursor.fetchall()
            return json.dumps(result)
    except:
        return json.dumps({'error':'something is wrong with the DB'})
```

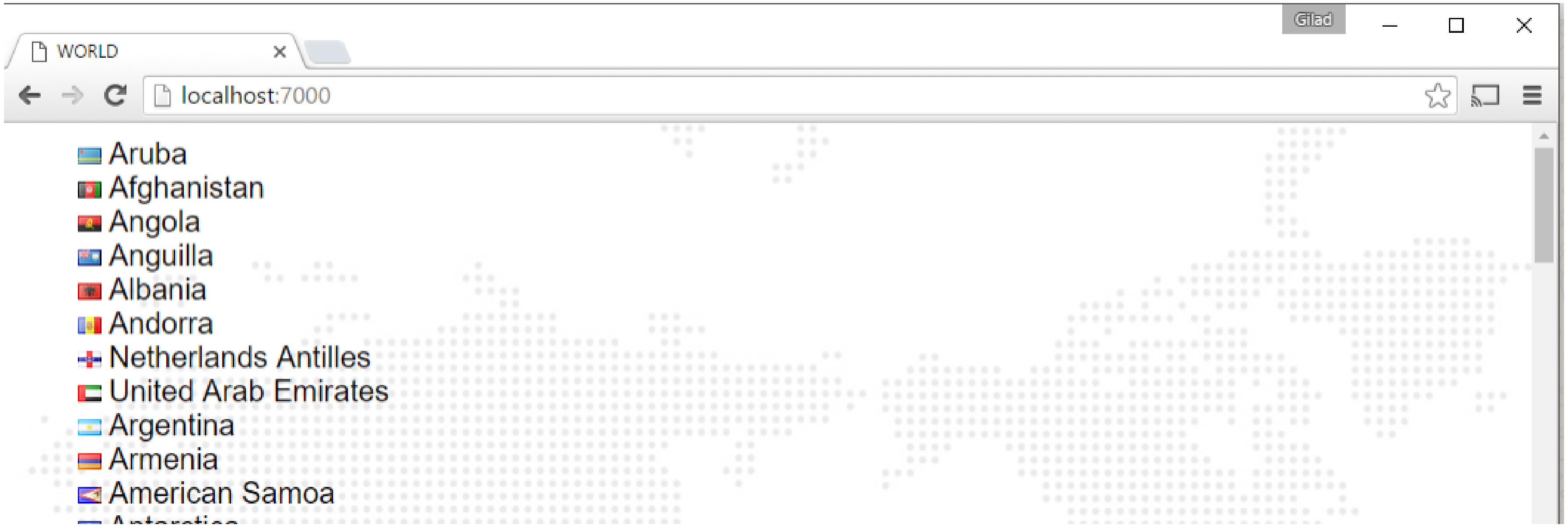


Questions?

Example – World Database

- ✓ Our application will include a list of all the countries in the world alongside their flags
- ✓ The list of the countries will be fetched from the DB

Clicking a country will display the country's population

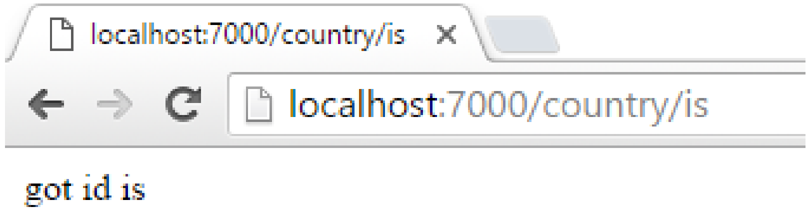


Aruba
Afghanistan
Angola
Anguilla
Albania
Andorra
Netherlands Antilles
United Arab Emirates
Argentina
Armenia
American Samoa

Example – World Database?

First let's create the `/country_details` handler

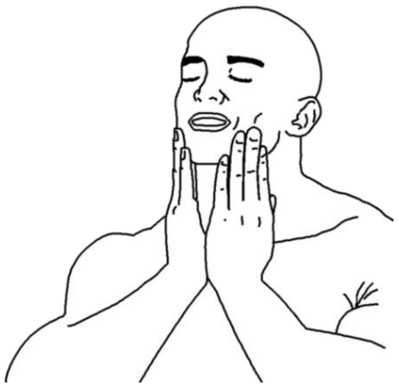
```
@get("/country/<id>")
def country_details(id):
    return "got id " + id
```



localhost:7000/country/is

localhost:7000/country/is

got id is



Example – World Database

Formatting a string is better that concatenating multiple strings

```
@get("/country/<id>")
def country_details(id):
    try:
        with connection.cursor() as cursor:
            sql = "SELECT * FROM country WHERE Code = '{}'".format(id)
            cursor.execute(sql)
            result = cursor.fetchone()
            return json.dumps(result)
    except Exception as e:
        return json.dumps({'error':'something is wrong with the DB ' + repr(e)})
```

NEW

NEW Fetches only one entry

NEW

Capture the error message, otherwise we don’t know what happened

From Client to DB

And the client side:

```
World.loadCountries = function(contentHolder){
    $.get("/list_countries",function(countries){
            if ('error' in countries){
                alert(countries.error);
            }else{
                var countriesHolder = $("<ul>");
                for (i in countries){
                    var countryEntry = $("<li />").addClass("country-entry clickable");
                    countryEntry.attr("id",countries[i].Code);
                    var imgSrc = "/images/flags/" + countries[i].Code2 + ".png";
                    var countryFlag = $("<img />").attr("src",imgSrc);
                    var countryName = $("<span />").addClass("country-name").text(countries[i].Name);
                    countryEntry.append(countryFlag).append(countryName);
                    countriesHolder.append(countryEntry);
                    countryEntry.click(function(e){
                        var currentCountry = $(this);
                        var currentCountryCode = currentCountry.attr("id");
                        $.get("/country/" + currentCountryCode,function(countryDetails){
                            var msg = currentCountry.find(".country-name").text();
                            msg += " population is " + countryDetails.Population;
                            alert(msg);
                        },"json");
                    });
                }
                contentHolder.empty().append(countriesHolder);
            }
        },"json");
};
```

Adding the primary key

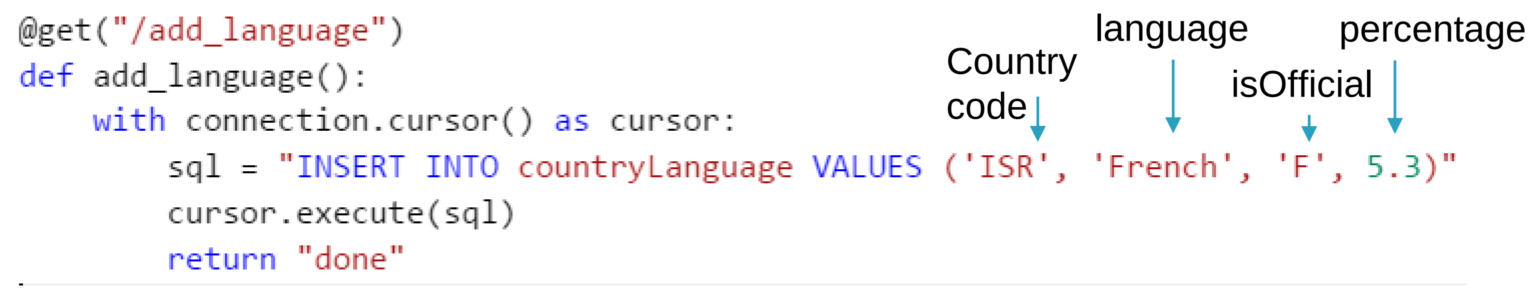
Getting the Population

< itc >

Questions?

Insert Using a Cursor

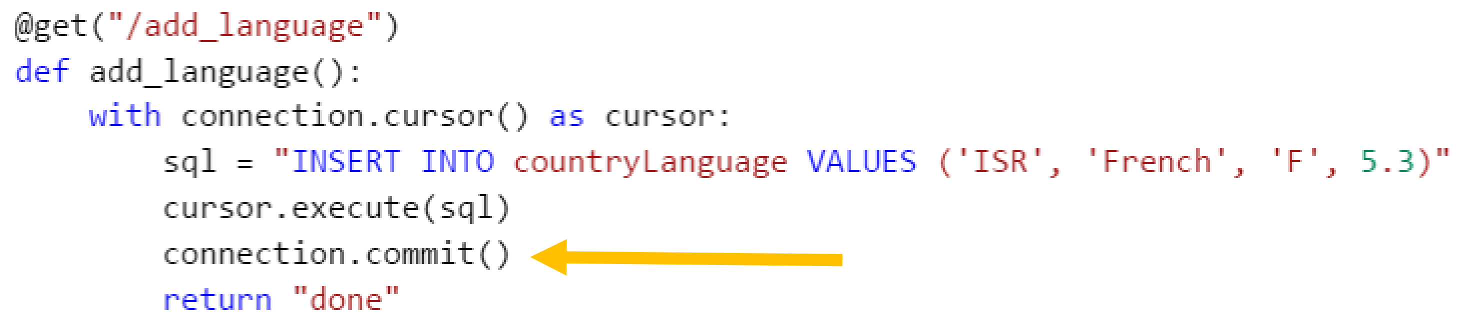
Altering the DB (committing changes (the ‘C’ in CRUD))



```
@get("/add_language")
def add_language():
    with connection.cursor() as cursor:
        sql = "INSERT INTO countryLanguage VALUES ('ISR', 'French', 'F', 5.3)"
        cursor.execute(sql)
        return "done"
```

This will not work…
The code will pass without an error but the entry will not be saved

We must commit changes when we write to the DB (using the **connection**)



```
@get("/add_language")
def add_language():
    with connection.cursor() as cursor:
        sql = "INSERT INTO countryLanguage VALUES ('ISR', 'French', 'F', 5.3)"
        cursor.execute(sql)
        connection.commit()
        return "done"
```

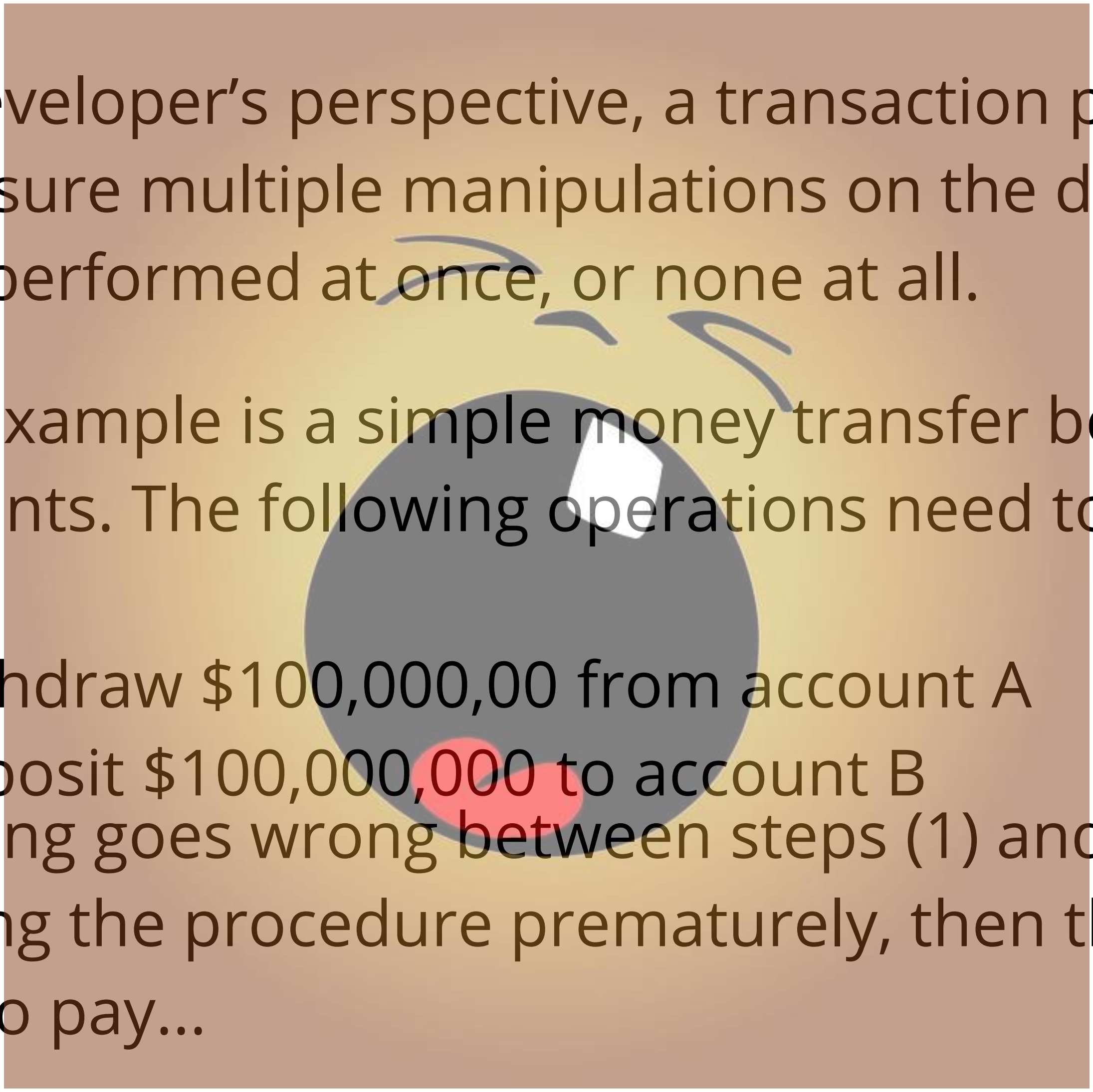
Database Transaction

A **transaction** symbolizes a unit of work performed against a database, and treated in a coherent and reliable way independent of other transactions. A transaction generally represents any change in a database. Transactions in a database environment have two main purposes:

1. To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.
2. To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the programs' outcomes are possibly erroneous.

https://en.wikipedia.org/wiki/Database_transaction

Commit & Rollback



From a developer’s perspective, a transaction provides a way to ensure multiple manipulations on the database either all performed at once, or none at all.

A classic example is a simple money transfer between two accounts. The following operations need to take place:

1. Withdraw $100,000,00 from account A
2. Deposit $100,000,000 to account B

If something goes wrong between steps (1) and (2), terminating the procedure prematurely, then there will be "hell" to pay...

Commit & Rollback

The set of operations which needs to take place all or none, is called a **transaction**.

If all goes well, the operations are **commit**-ed to the database.

If the program decides to explicitly abort the operation, or if something unexpected happens which interrupts the code, then all the changes made until that point are **rolled-back**.

Pagination?

Showing all the countries in one page is not comfortable on the human eye

Because

Subitizing

Subitizing is the ability to 'see' a small amount of objects and know how many there are without counting

Pagination?

The core of our app is completed

Everything we are adding now is a feature

If we were a team we would be able to implement that feature in parallel

Meaning both client side and server side in the same time

Pagination?

Define the server behavior:

We need **"/list_countries"** to return 30 countries at a time

We need to be able to tell **"/list_countries**" which page we want

"/list_countries/<page>"

We need to be able to know if there are more countries to show
(in order to hide the "next" button if no more countries available)

A new json for the result:

```
{
    "countries":[list of countries],
    "has_more":true/false
}
```

Pagination?

Define the client behavior:

- We need to show "next" and "prev" buttons
- Show the pagination buttons only if we have more results (both ways)
- Bind the buttons to use **"/list_countries/<page>**"
- Adjust our code to the new JSON format

Pagination?

The handler decorators and signature

```
@get("/list_countries")
@get("/list_countries/<page>")
def list_countries(page='0'):
```

Selecting from the DB

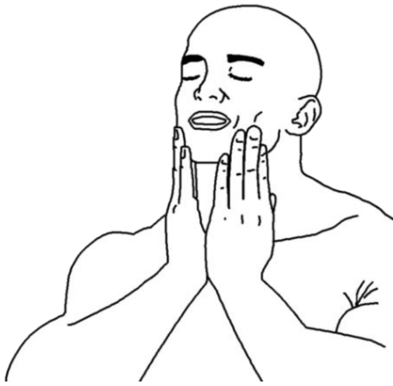
How many

Where to start

```
sql = "SELECT Code, Name , Code2 FROM country LIMIT %s OFFSET %s"
cursor.execute(sql,((COUNTRIES_PER_PAGE + 1), (int(page)*COUNTRIES_PER_PAGE)))
```

A constant

```
COUNTRIES_PER_PAGE = 30
```



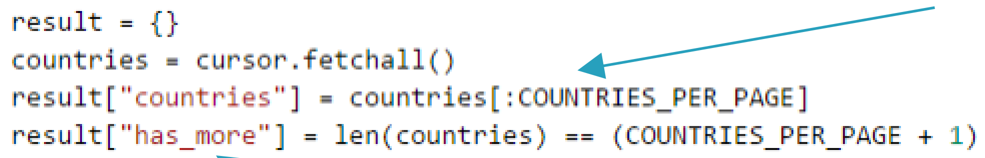
Pagination?

Why COUNTRIES_PER_PAGE + 1?

This is a little trick to know if there are more entries,
If we get less than 31 entries in our case it means we don't have more to show...

Pagination?

Why COUNTRIES_PER_PAGE + 1?



```
result = {}
countries = cursor.fetchall()
result["countries"] = countries[:COUNTRIES_PER_PAGE]
result["has_more"] = len(countries) == (COUNTRIES_PER_PAGE + 1)
```

Take only 30
(we have 31)

For the client side to
know if we have more

Pagination

The complete server side:

```
COUNTRIES_PER_PAGE = 30

@get("/list_countries")
@get("/list_countries/<page>")
def list_countries(page='0'):
    try:
        with connection.cursor() as cursor:
            sql = "SELECT Code, Name , Code2 FROM country LIMIT %s OFFSET %s"
            cursor.execute(sql,((COUNTRIES_PER_PAGE + 1), (int(page)*COUNTRIES_PER_PAGE)))
            result = {}
            countries = cursor.fetchall()
            result["countries"] = countries[:COUNTRIES_PER_PAGE]
            result["has_more"] = len(countries) == (COUNTRIES_PER_PAGE + 1)
            return json.dumps(result)
    except:
        return json.dumps({'error':'something is wrong with the DB'})
```

Questions?

Pagination?

And in the client side

```
World.start = function(){
    $(document).ready(function() {
        var contentHolder = $("#content");
        World.loadCountries(contentHolder,0);
    });
};

World.loadCountries = function(contentHolder,page){
    $.get("/list_countries/" + page,function(result){
```

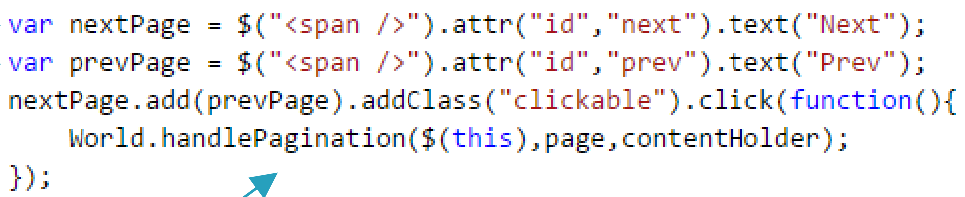
Load Countries now accepts a page number

We concatenate the page number

Pagination?

And in the client side

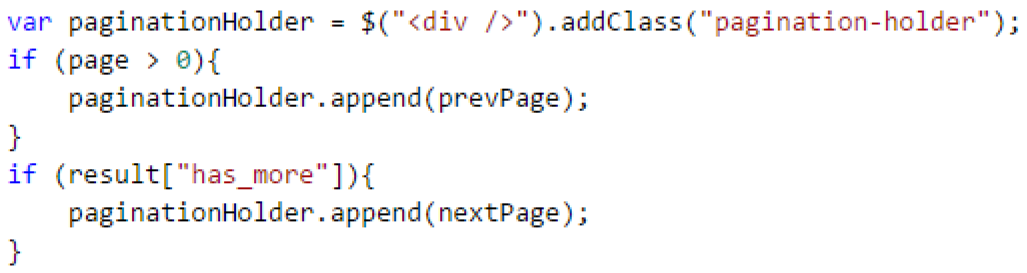
Creating the buttons



```
var nextPage = $("<span />").attr("id","next").text("Next");
var prevPage = $("<span />").attr("id","prev").text("Prev");
nextPage.add(prevPage).addClass("clickable").click(function(){
    World.handlePagination($(this),page,contentHolder);
});
```

Binding the click to a new function

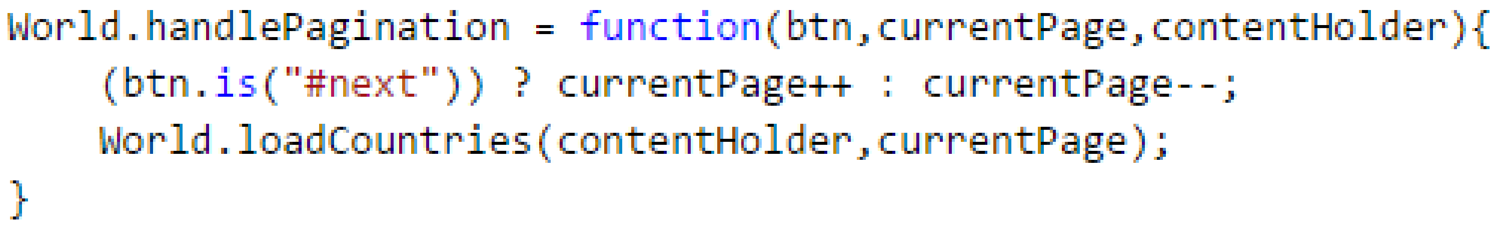
Appending only if needed



```
var paginationHolder = $("<div />").addClass("pagination-holder");
if (page > 0){
    paginationHolder.append(prevPage);
}
if (result["has_more"]){
    paginationHolder.append(nextPage);
}
```

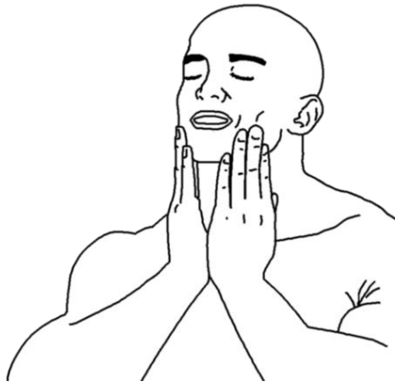
Pagination?

And in the client side



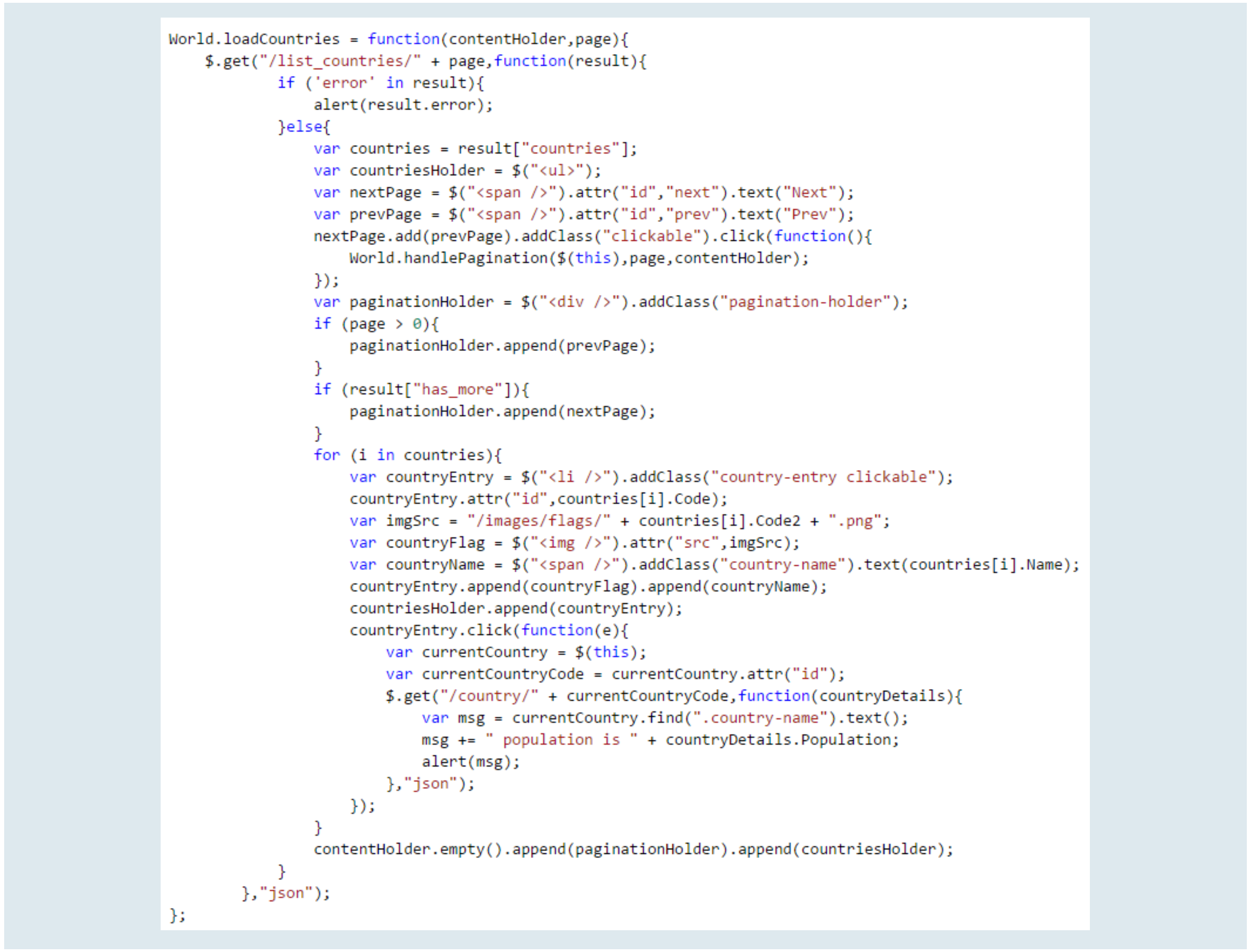
```
World.handlePagination = function(btn,currentPage,contentHolder){
    (btn.is("#next")) ? currentPage++ : currentPage--;
    World.loadCountries(contentHolder,currentPage);
}
```

Call load countries with the correct page



Pagination

The complete client side:



```
World.loadCountries = function(contentHolder,page){
    $.get("/list_countries/" + page,function(result){
            if ('error' in result){
                alert(result.error);
            }else{
                var countries = result["countries"];
                var countriesHolder = $("<ul>");
                var nextPage = $("<span />").attr("id","next").text("Next");
                var prevPage = $("<span />").attr("id","prev").text("Prev");
                nextPage.add(prevPage).addClass("clickable").click(function(){
                    World.handlePagination($(this),page,contentHolder);
                });
                var paginationHolder = $("<div />").addClass("pagination-holder");
                if (page > 0){
                    paginationHolder.append(prevPage);
                }
                if (result["has_more"]){
                    paginationHolder.append(nextPage);
                }
                for (i in countries){
                    var countryEntry = $("<li />").addClass("country-entry clickable");
                    countryEntry.attr("id",countries[i].Code);
                    var imgSrc = "/images/flags/" + countries[i].Code2 + ".png";
                    var countryFlag = $("<img />").attr("src",imgSrc);
                    var countryName = $("<span />").addClass("country-name").text(countries[i].Name);
                    countryEntry.append(countryFlag).append(countryName);
                    countriesHolder.append(countryEntry);
                    countryEntry.click(function(e){
                        var currentCountry = $(this);
                        var currentCountryCode = currentCountry.attr("id");
                        $.get("/country/" + currentCountryCode,function(countryDetails){
                            var msg = currentCountry.find(".country-name").text();
                            msg += " population is " + countryDetails.Population;
                            alert(msg);
                        },"json");
                    });
                }
                contentHolder.empty().append(paginationHolder).append(countriesHolder);
            }
        },"json");
};
```

Questions?

Summary

- You needed to understand:
  - The uses of a DB in a web app
  - The concept of Exceptions handling
- You need to remember:
  - The CRUD concept, pagination concept
- You need to be able to do:
  - Use cursors and connections and use a DB in the scope of a web application
  - Create applications that remembers a state (persistent)

Questions?