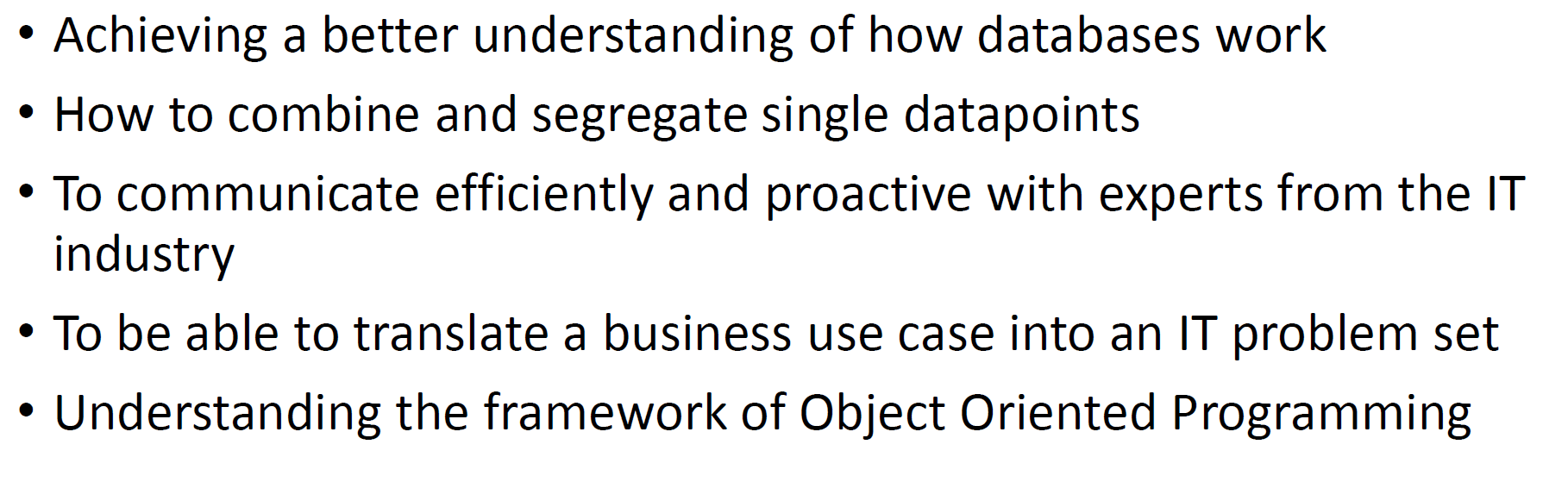
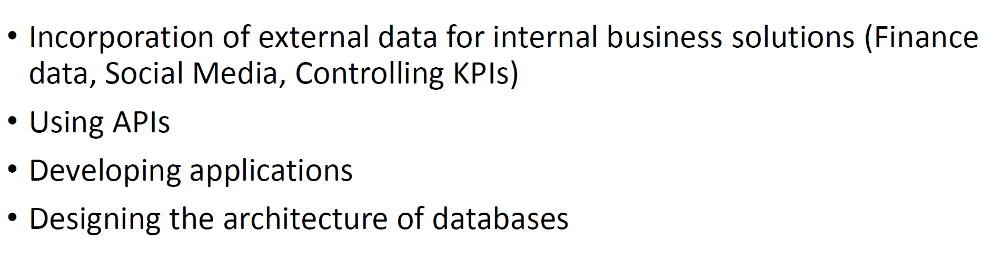
**OOP Notes – Exam 30.05.2018**



# Lecture 1

## Programming Paradigms

* Procedural: Follows a top town approach, bases on procedures, routines and subroutines
  + Fortran, ALGOL, COBOL and BASIC, Assembly
  + Two registers -> D = Data Register, A = Address register
* Functional: Avoids changing state and mutable data, similar approach to mathematical functions
  + Racket, R
  + Consists of Functions that take in data and output data
  + Pure Functional Programming -> functions don’t ever store state and don’t mutate the incoming data which means, that always a new array is returned.
  + The output ONLY depends on the input, it doesn’t matter what came before.
  + Data (Stata) and Procedures (Behaviour) are separate (unlike in oop)
  + Strengths:
    - We don’t have to worry about concurrency because we’re not modifying state (everything is just a read operation)
    - Ease of testing -> state doesn’t matter, data in data out, we never have to set the state of an underlying object first.
    - The code is really reusable.
    - Functional Programs tend to be really short.
* Objectural: Encapsulates data and methods in objects, objects can interact.
  + Object = Way of encapsulating state and behaviour
    - State = fields, attributes, variables
    - Behaviours = Methods, what we do with that state.
  + The Object is responsible for modifying its own internal state.
  + Strengths:
    - Objects ineract
    - Really good in modelling what we do in real life
    - Objects are reusable
    - Objects are easy to test -> we can test the behaviour of a single object

## Programming Principles

* KIS – Keep it simple -> no spaghetti code, no overengineering
* DRY – Do not repeat yourself -> if copy more than 3 times, make a function
* YAGNI – You ain’t gonna need it -> no hoarding of ideas

# Lecture 2

## Variables

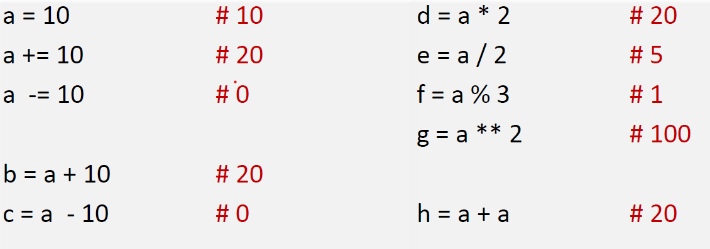
= Piece of memory that can store a value.

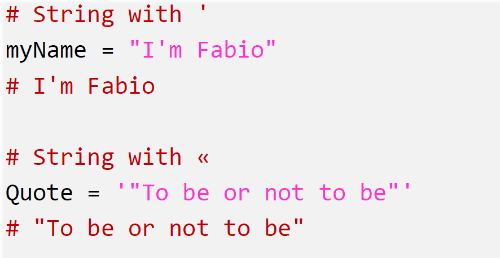
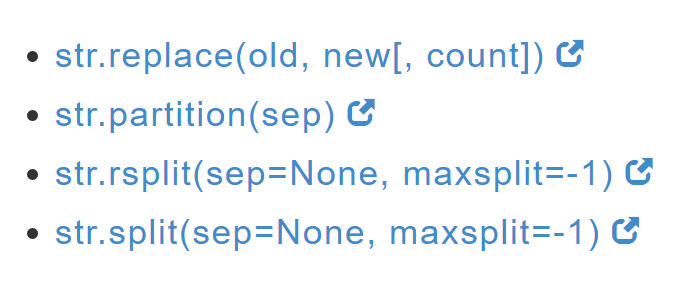
Naming conventions:

* Cannot start with a number
* No spaces in the name
* No special symbols
* Lowercase names

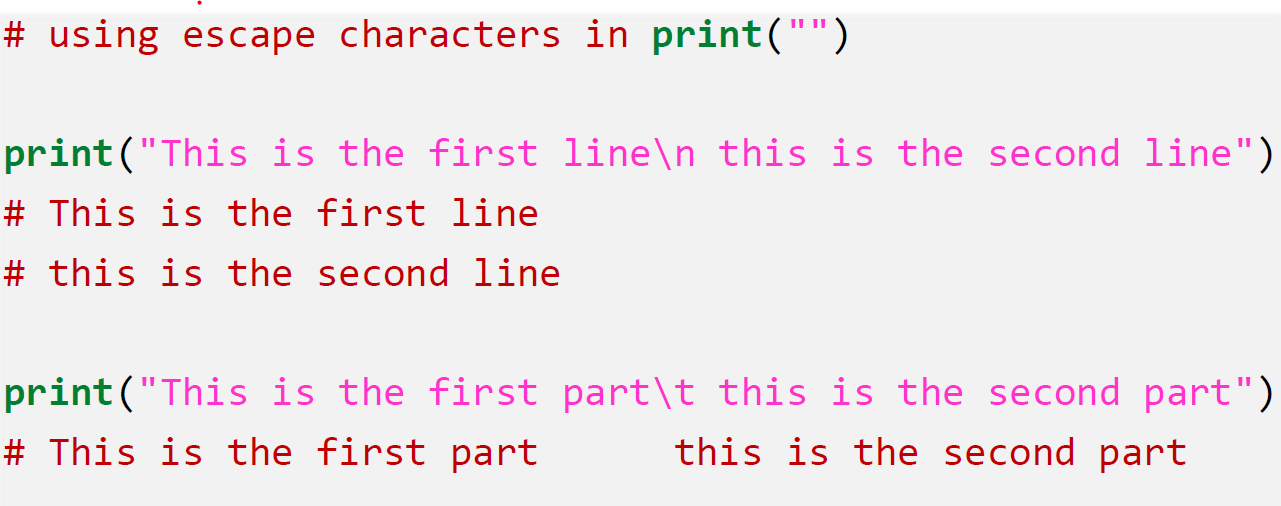
Types:

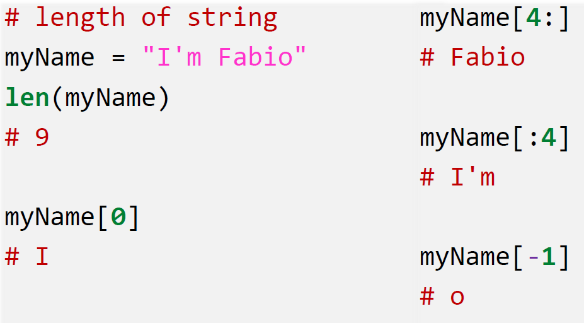
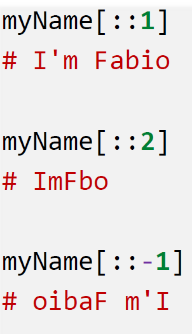
* Int: int(“”) or 24
* Float: float(“”) or 2.783234
* String: “”
* Multi-line String: “”” “””
* List: []
* Tuple: () -> The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
  + <https://www.tutorialspoint.com/python/python_tuples.htm>
* Boolean: True False –
  + Variables that return True:
    - True
    - Positive int
    - String
    - Dict {‘a’:’b’}
    - List [‘c’:’d’]
  + Variables that return False:
    - False
    - 0
    - Empty string/dict/list
    - None

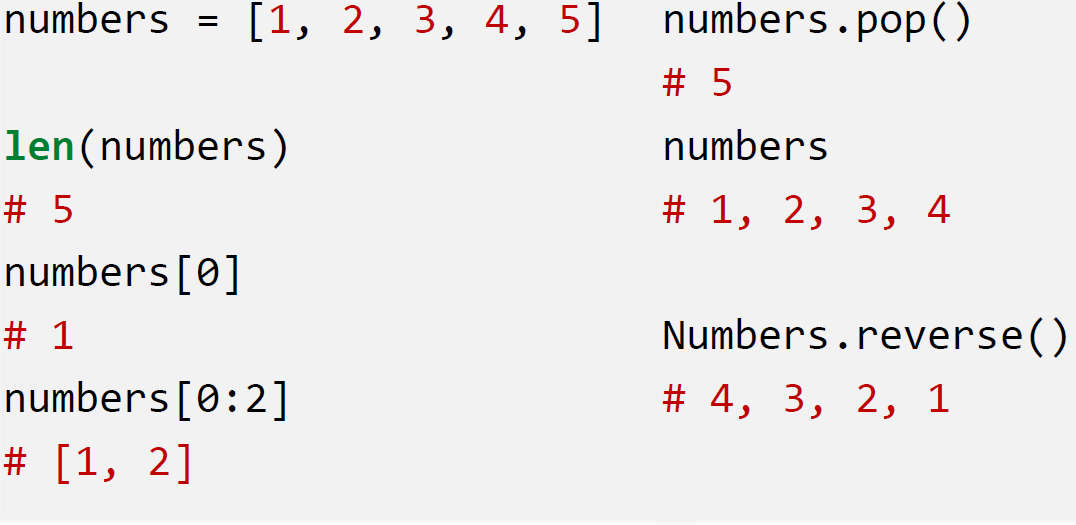
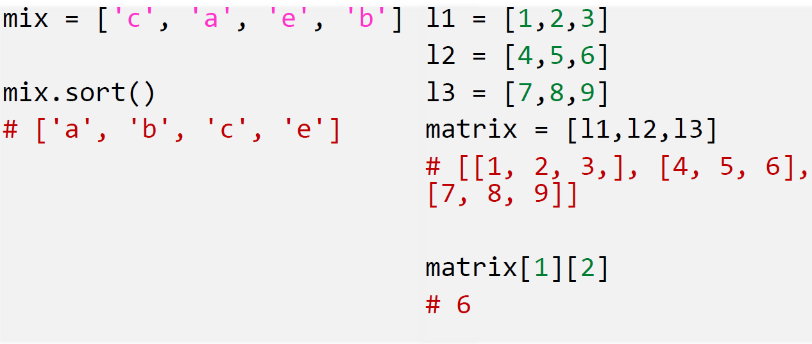
Arithmetic Operations:

  
String Operations:

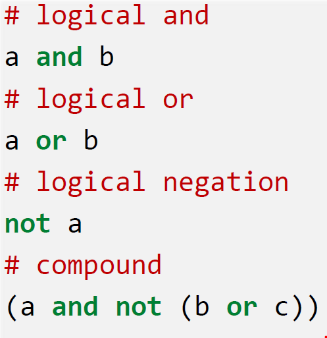
<https://www.pythoncentral.io/cutting-and-slicing-strings-in-python/>

Escape characters

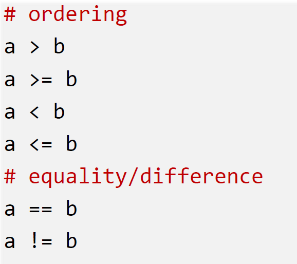
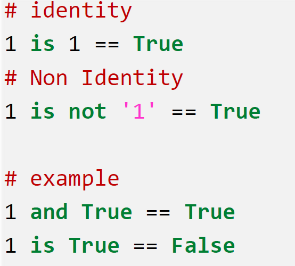
Length playing: -> Third parameter specifies the stride (length of Jumps) -> two semicolons

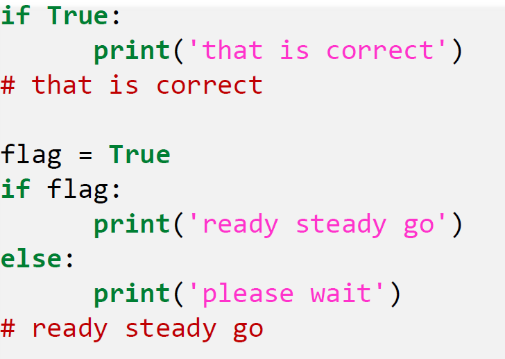
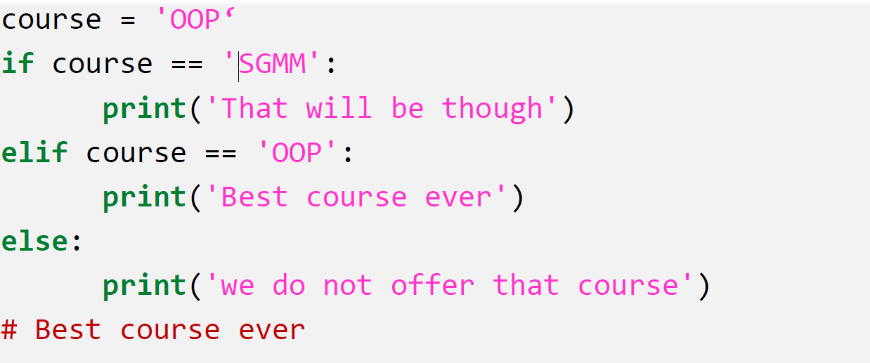
List Operations:

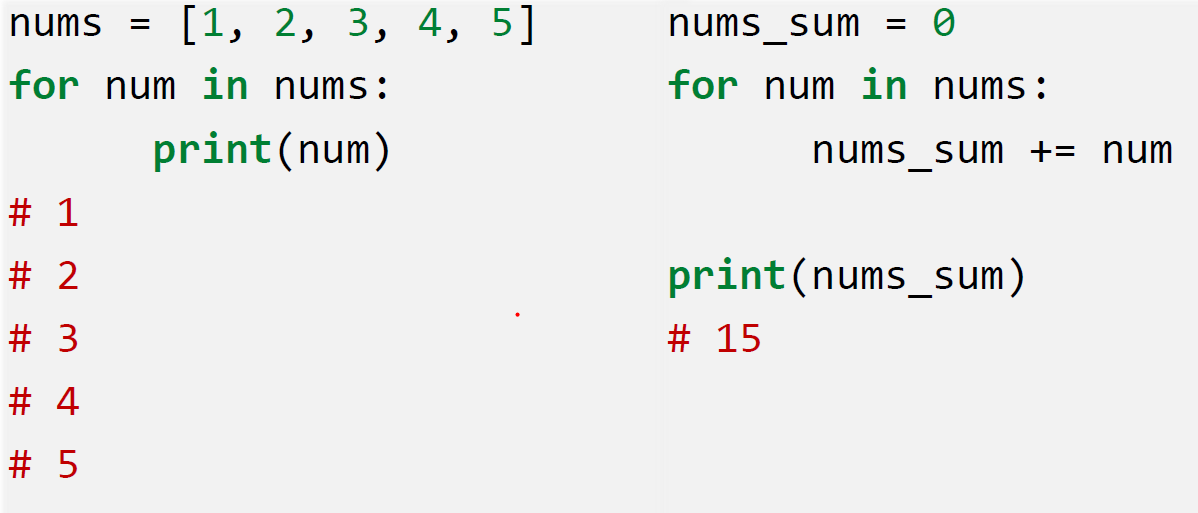
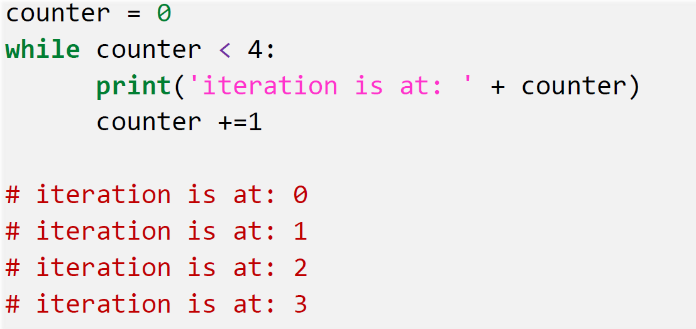
<https://docs.python.org/3/tutorial/datastructures.html> -> List Methods

Logical Comparison

If “not a” returns true, it means that a is empty or none or 0 or False

Arithmetic Comparison and Identity Comparison:

Conditionals:

For/While Loops:

The **pass** statement is a *null* operation; nothing happens when it executes.

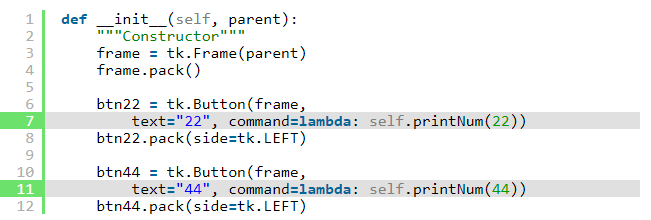
**Break** = It terminates the current loop and resumes execution at the next statement.

# Lecture 3

## Functions

## Lambda

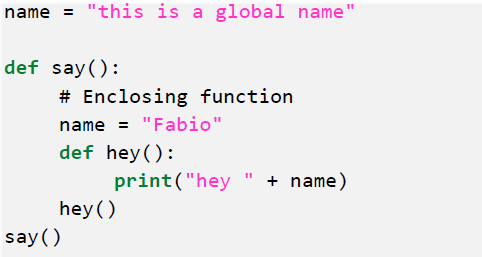
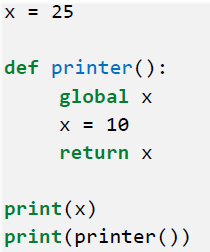
<https://pythonconquerstheuniverse.wordpress.com/2011/08/29/lambda_tutorial/>

* Anonymus function
* Has the same properties as a normal function
* Can be created ad-hoc without the need of def
* Only a single expression and not a block of code
* Can be used as a parameter in a function/method
* Can be used if the function to write is fairly simple and it is going to be used only once.
* Is often used in coding “callbacks” to GUI frameworks such as Tkinter and wxPython

What is an expression? – Difference to a statement: An expression returns a value, whereas a statement does not. What can be put into Lambda?

* If it doesn’t return a value, it isn’t an expression and can’t be put into a lambda.
* If you can imagine it in an assignment statement, on the right-hand side of the equal sign, it is an expression and can be put into a lambda.

## Scopes

* Local: variables created within a function
  + Exists for as long as the function is executing.
* Enclosing functions: from INNER to OUTER
* Global Variables: Defined in the main body of a file or as global within def
  + Visible throughout the file
  + Visible inside ANY file that imports the file.
  + Shouldn’t be used since they can have unintended consequences.
* Variables inside class:
  + Called attributes
  + Every Instance has the same class attributes, but different instance attributes.

# Lecture 4 & 5 - OOP

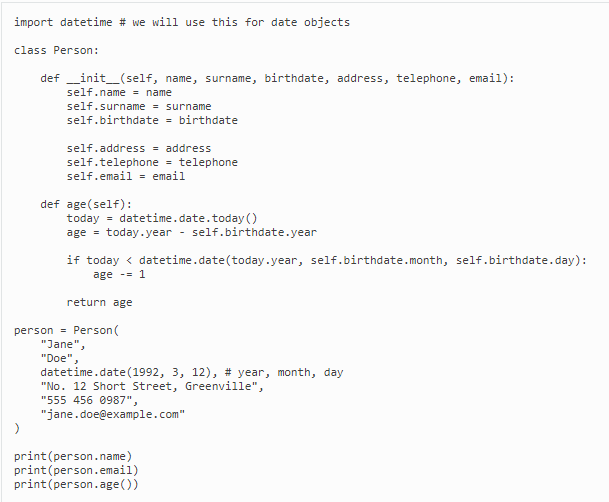
## Classes

In Python, everything is an object – everything is an instance of some class.

**Class** = way of grouping together related data and functions which act upon that data

= blueprint, definition and description of a case

A class is a kind of data type, just like a string, integer or list. When we create an object of that data type, we call it an instance of a class.



* Definition starts with a “class” keyword, followed by the class name and a colon
* We list any parent classes in between round brackets before the colon
* Inside the class body we define the methods:
  + \_\_init\_\_ = special method (Constructor/Initializer)
  + When the class is called, the \_\_init\_\_ method is directly executed with all parameters that we passed to the class object.
* “self” stands for the object itself which is then automatically used as the method’s first parameter. It does not need to be called self, but it’s a strongly followed consent.

In Python, you can add new attributes, and even new methods, to an object on the fly. But initialising all our attributes in \_\_init\_\_, even if we just set them to empty values, makes our code less error-prone. It also makes it easier to read and understand – we can see at a glance what attributes our object has.

## OOP

The main additional advantage of object orientation, as we saw in the previous chapter, is that it combines data with the functions which act upon that data in a single structure. This makes it easy for us to find related parts of our code, since they are physically defined in close proximity to one another, and also makes it easier for us to write our code in such a way that the data inside each object is accessed as much as possible only through that object’s methods.

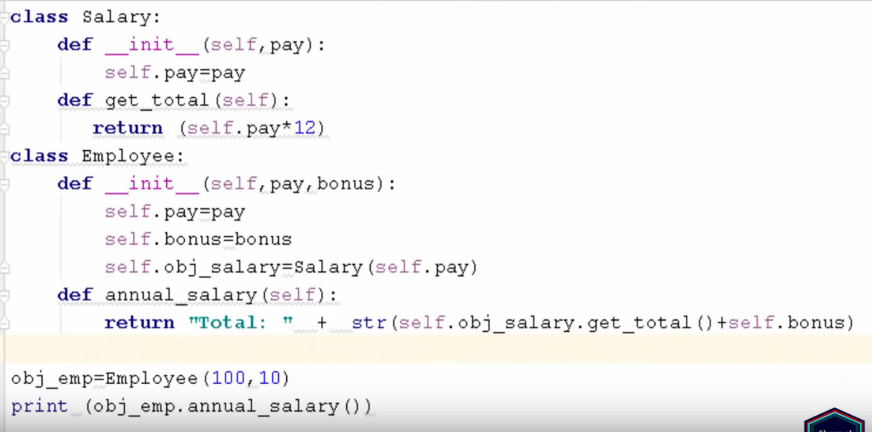
### Basic OOP principles

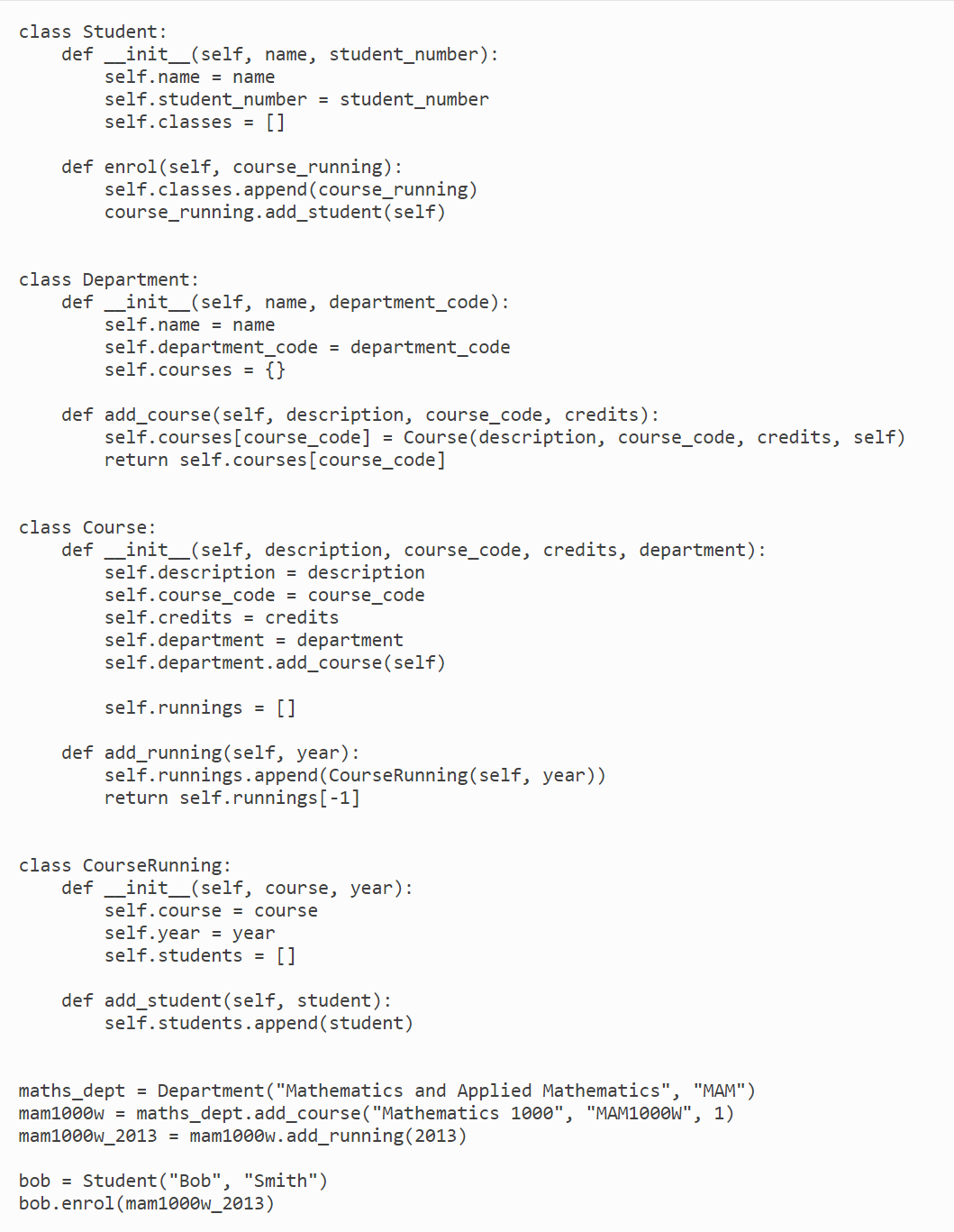
* Encapsulation: Data inside the object should only be accessed through a public interface -> the object’s methods.
  + Functionality defined in one place and not multiple places
  + Defined in logical place, where data is kept
  + Data inside object is not modified unexpectedly by external code in a completely different part of the program.
  + Methods are the same for every instance of a class.

The object KNOWS WHAT TO DO WITH HIS OWN DATA. We should not access its internals and do things with the data ourselves.

### Composition

Composition is a concept referred on how an object is defined. We define a new object that is “composed” of other, predefined objects, resp. instances of those objects. Strong relationship between owner object and owned. One of the classes playes the role of a container, the other the role of the content.

We can take humans and body parts as an example. A human can live without some bodyparts, whereas body parts cannot live on their own. The content objects are never called by themselves, only in combination with a container.



### Inheritance

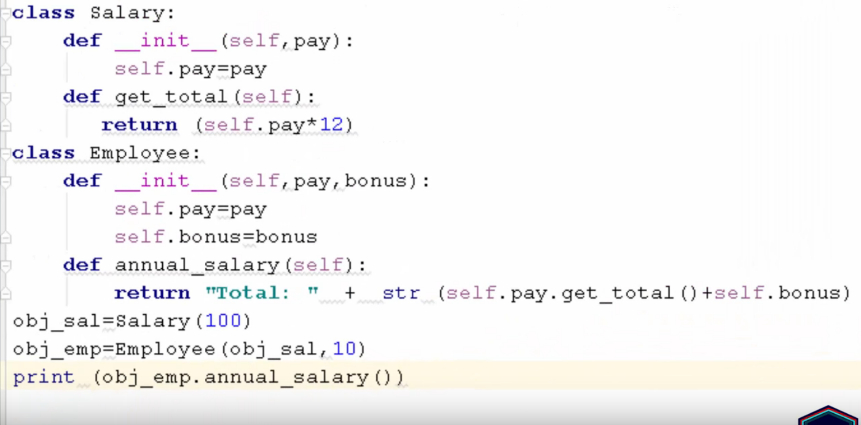
See Glossary - Defining a new object in terms of a predefined object. Inheriting all aspects and making additions and modifications.

We often also say that a class is a subclass/childclass of a class from which it inherits, or that the other class is its superclass/parent class.

Inheritance allows the programmer to use parent class methods in the child class.

The method “super().\_\_init\_\_()” can be used to perform inheritance, resp. to get the attributes out of a parent class

### Aggregation

= weak form of composition. If the container object is deleted, then all of it’s content objects can still exists without the container object. For example a Team that is a collection of players. The players can still exist, once the team is deleted.

# Lecture 6

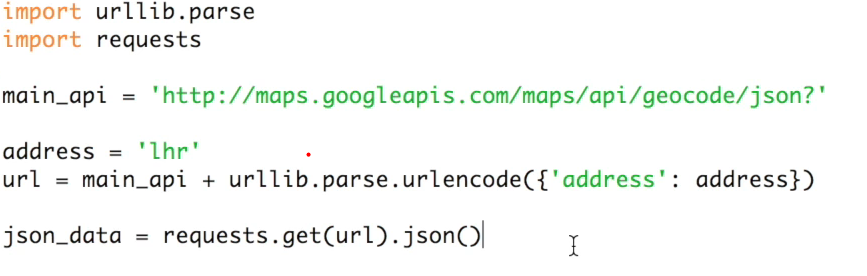
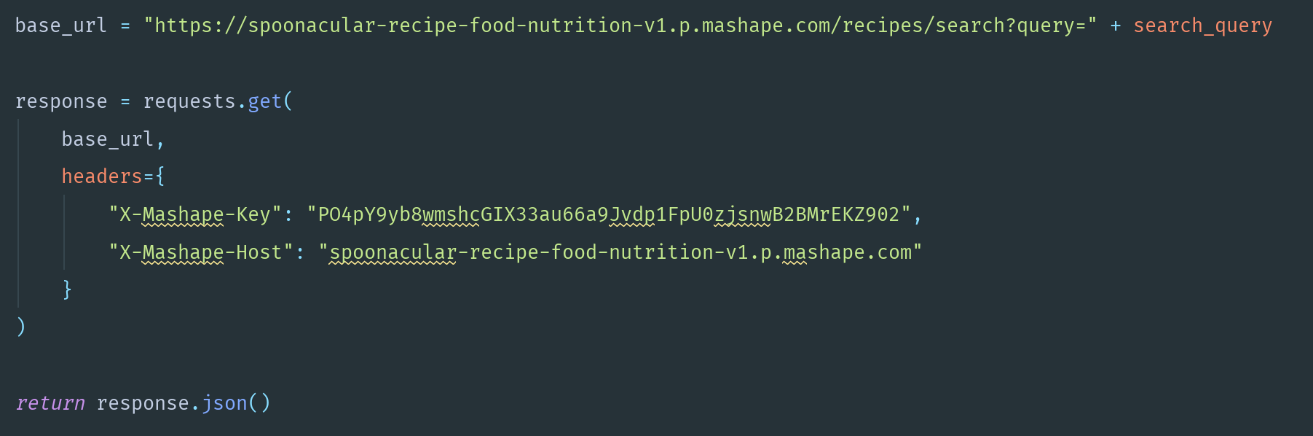
<https://www.dataquest.io/blog/python-api-tutorial/>

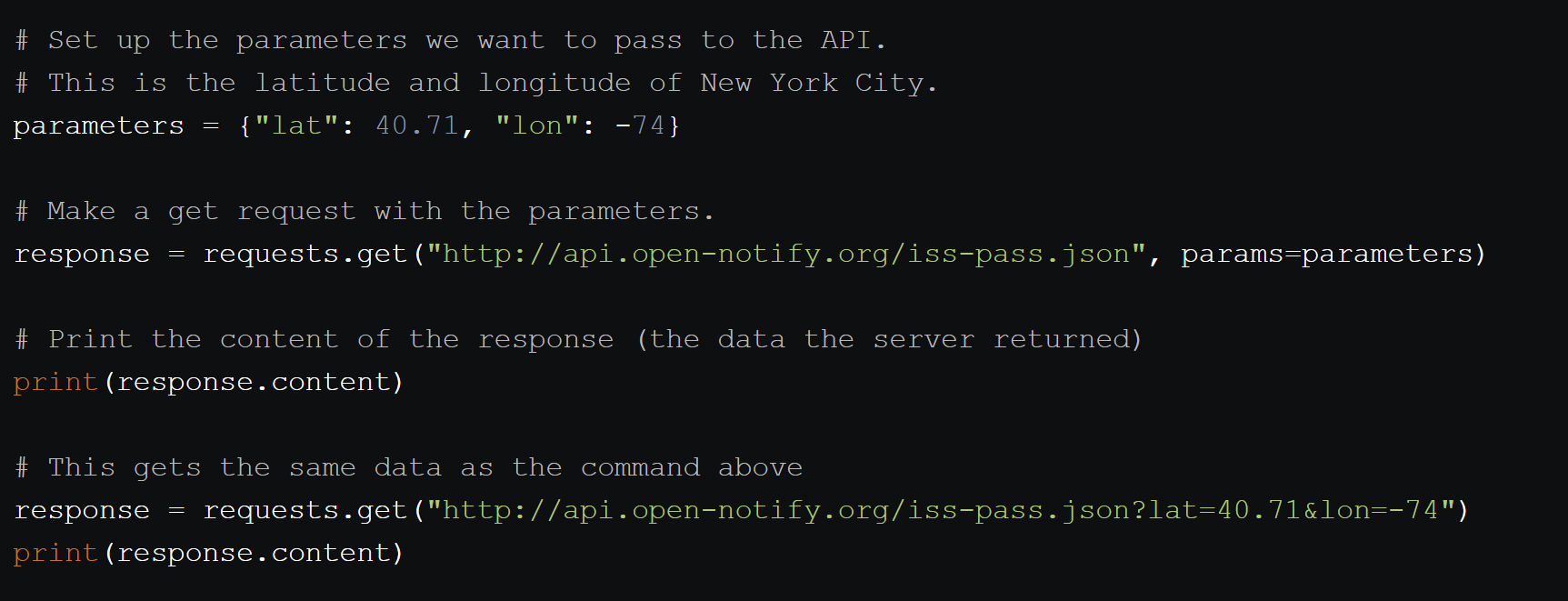
To use an API, you make a request to a remote web server, and retrieve the data you need. Why we should use APIS:

* Data is changing quickly -> Using API’s is faster and uses less bandwidth than generating a new dataset every minute.
* API’s can filter small pieces of data out of juge databases.
* Repeated computation is involved (Example Spotify music genre classifier)

## Request types:

* GET -> used to retrieve data. In order to get data the server endpoint is added to the base url of the API



If we need to input multiple parameters we can set them up as a dictionary. Requests takes care of some things that come up, like properly formatting the query params.

## Status codes:

Indicate information about what happened with a request.

* 200 – everything okay and results returned
* 301 – server is redirecting us to a different endpoint.
* 401 – server thinks we’re not authenticated.
* 400 – Server thinks we’ve made a bad request.
* 403 – the resource we’re trying to access is forbidden/we don’t have the right permissions to see it.
* 404 – the resource we tried to access wasn’t found on the server

## Dealing with JSON

JSON is a way to encode data structures like lists and dictionaries to strings that ensures that they are easily readable by machines. JSON is the primary format in which data is passed back and forth to APIs, and most API servers will send their responses in JSON format.

Python has great JSON support, with the json package. The json package is part of the standard library, so we don't have to install anything to use it. We can both convert lists and dictionaries to JSON, and convert strings to lists and dictionaries.

The json library has two main methods:

dumps -- Takes in a Python object, and converts it to a string.

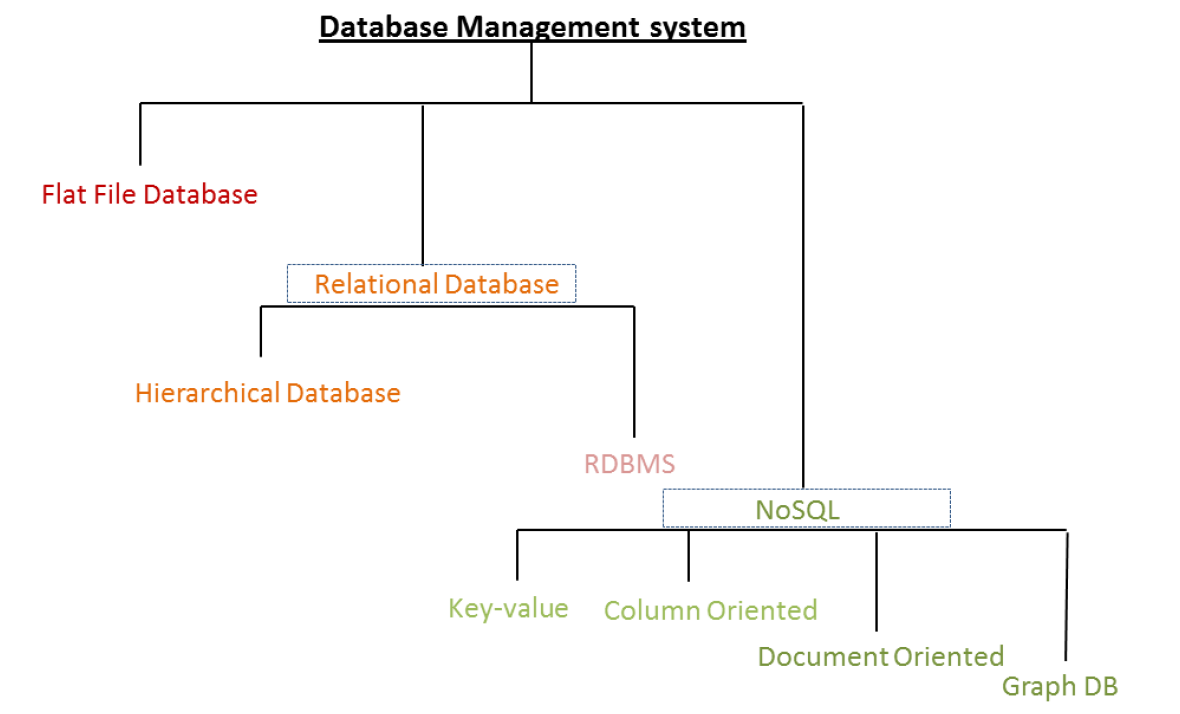
loads -- Takes a JSON string, and converts it to a Python object.

# Lecture 7/8

We use Databases because:

* Size
* Ease of Updating
* Accuracy
* Security
* Redundancy
* Importance

## Database types:



* NoSQL (Not only SQL): Built for huge amounts of data, are not relational.
  + Make distributing the database load on multiple hosts (scaling out) easier.
  + Designed with Web-Applications in mind.
  + Don’t use sql to query data
  + Don’t follow strict schemas
* Relational Database:
  + Used by the vast majority of people
  + Made up of a set of tables with data that fits into predefined category
  + Values within each table are related to each other. Table may also be related to other tables. Queries can be run across multiple tables at once.
  + Data is stored in structured format in table using rows and columns.
* Flat File Database:
  + All the data is in one table -> Excel
  + A lot of redundant data if two columns are linked to each other

## Key types

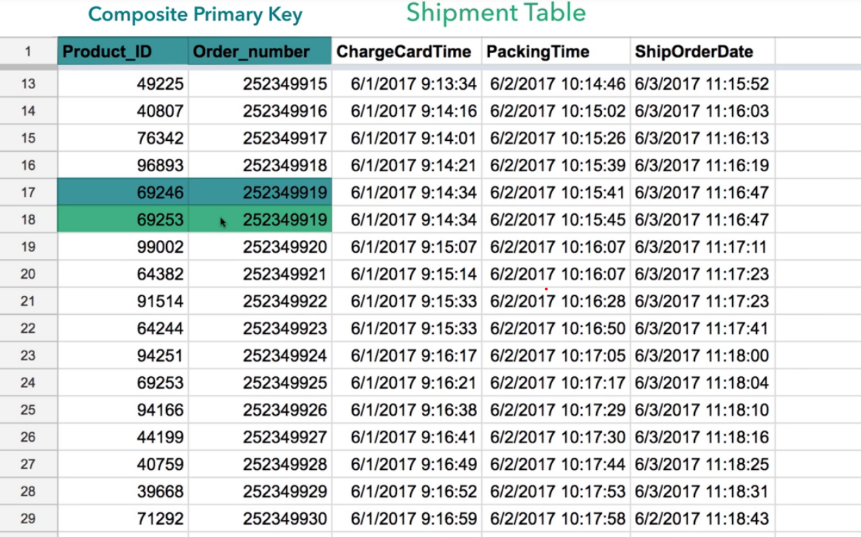
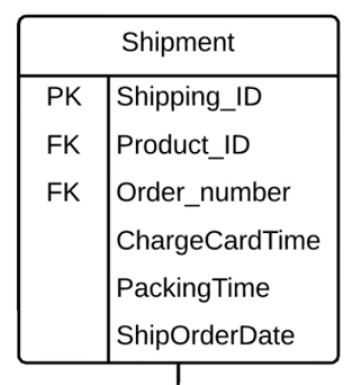
They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table.

**Primary Key:**

* Table must have a primary key, in order to qualify as relational table.
* Consists of one or more columns whose data contained within is used to **uniquely identify** each row in the table -> ADDRESS
* No value in the pk columns can be blank or null
* A table can only have one primary key
* It’s stored in an index

**Foreign Key:**

* Used to link two tables together
* Set of one or more columns in a table that refers to the primary key in another table.
* It is possible, that a foreign key is at the same time a primary key and it’s not even bad practice.
* Unlike primary keys, foreign keys can contain duplicate values. Also, it is OK for them contain NULL values.
* A table is allowed to contain more than one foreign key.

**Composite Primary keys:**

Rules:

* Use fewest number of attributes possible
* Don’t use attributes that are apt to change -> can get messy.

**ATTENTION:** Composite Primary keys can normally be replaced by a unique primary key.

## SQL

= Structured Query Language, standard computer language for relational database management and data manipulation. SQL is used to query, insert, update and modify data. Most relational databases support SQL.

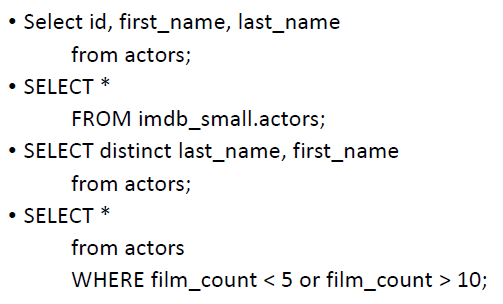
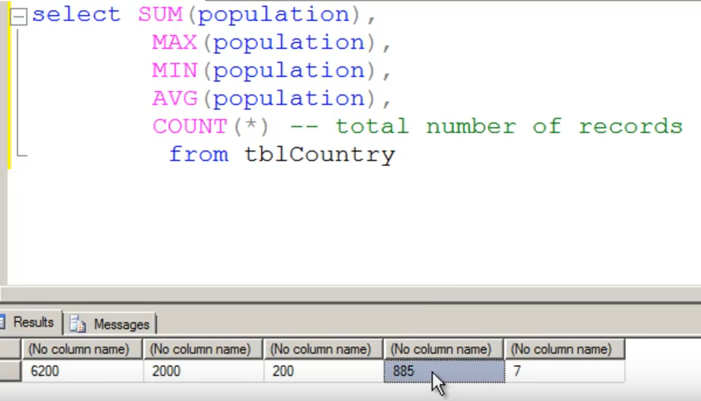
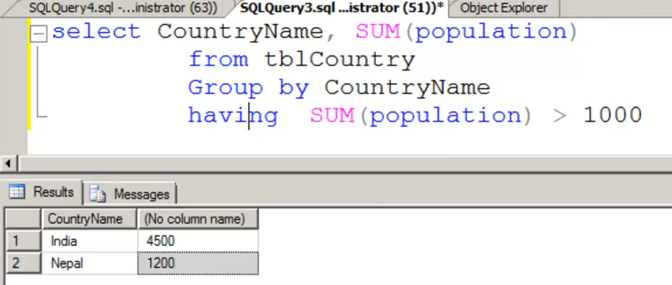
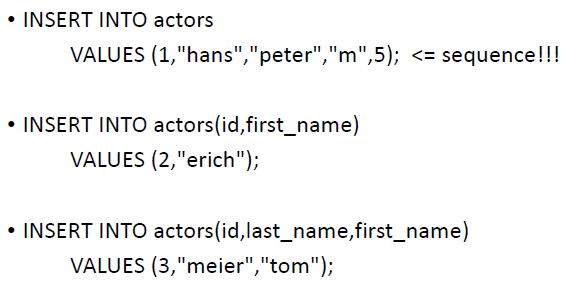
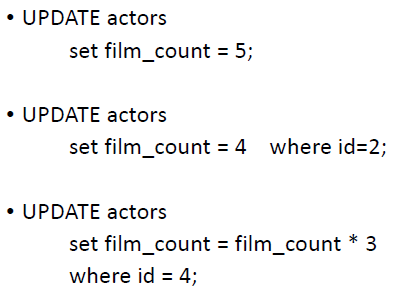
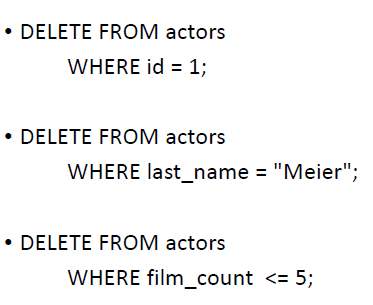
### Data Definition Language (DDL)

DDL statement define the database structure or schema:

* CREATE
* ALTER
* DROP – delete objects from the database
* TRUNCATE (remove all records from a table, including all spaces allocated for the records

### Data Manipulation Language (DML)

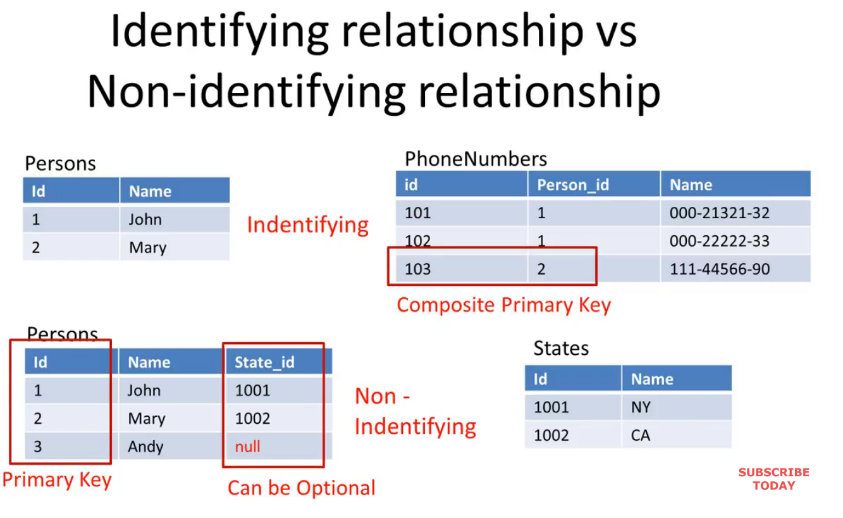
DML statements are used for managing data within schema objects

* SELECT
* INSERT
* UPDATE – updates existing data within a table
* DELETE – deletes all records from a table, the space for the records remain (in contrast to TRUNCATE)

## Relationships

Identifying Relationship: Child table cannot be uniquely identified without its parent (Parent table is the one where we took the primary key from). The existance of a row in the child table, depends on the existance of a row in the parent table. Normally the foreign key is included in the child table’s primary key.

Non identifying Relationships: Foreign key Value is not included in the primary key, can be optional.



## Enigneering

### Forward Engineering

Transformation of a visual data model into a physical database on a target MySQL Server

### Reverse Engineering

Extraction of database tables, attributes, relationships, indexes, views, stored procedures and other objects from the database into a diagram.

## Relationships (Cardinality)

To find out what relation the different tables have one can ask (for both directions) the following. Example Orders and Customers:

* How many orders can a customer have at least? -> 0 How many orders can a customer have at max -> many
* How many customers can an order have? -> 1 (and only one)

**Many-to-many relationships ->** We use bridge tables to store all necessary information.

## Types of Anomalies

* Insertion Anomaly = Insert redundant data for every new row
* Deletion Anomaly = Loss of related dataset when some other dataset is deleted.
* Updation/Modification Anomaly = redundant work when updating redundant data
  + Can lead to data inconsistency.

## Normalization

= Process of reorganizing data in a database so that it meets two basic requirements:

* NO redundant data
* Data dependencies are logical -> Data integrity.

### First normal form (1NF)

* Each table only holds atomic values (no entries like (X,Y) or (W,X))
* All values in a column are of the same type
* Each column should have a unique name
* Order in which the data is stored, doesn’t matter

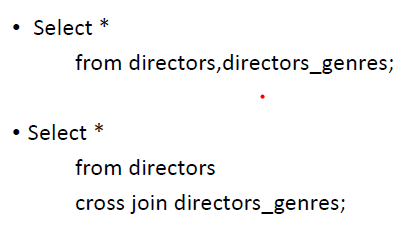
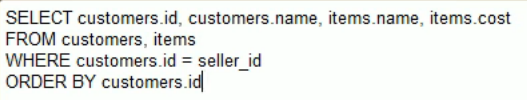
### Second normal form (2NF)

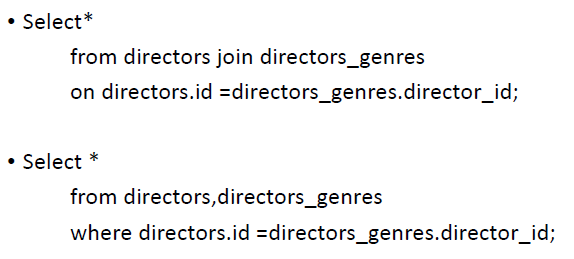
* Table is in first normal form
* No partial dependencies -> How to go about this: Just remove partially dependent column and move it to another table/ create an own table for the partially dependent column
* All attributes (Non-Key-Columns) dependant on the key

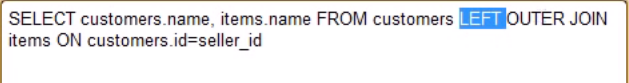
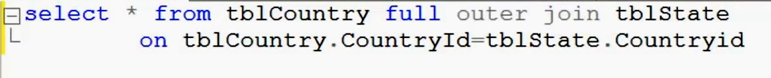
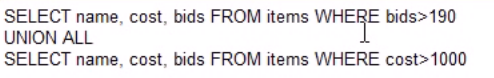
### Third normal form (3NF)

* Must be in second normal form
* No transitive dependencies -> take columns that depend on each other and create a new table.
* All fields can be determined only by the key in the table and no other columns

## Joining Tables

* Cross Join
* Inner Join



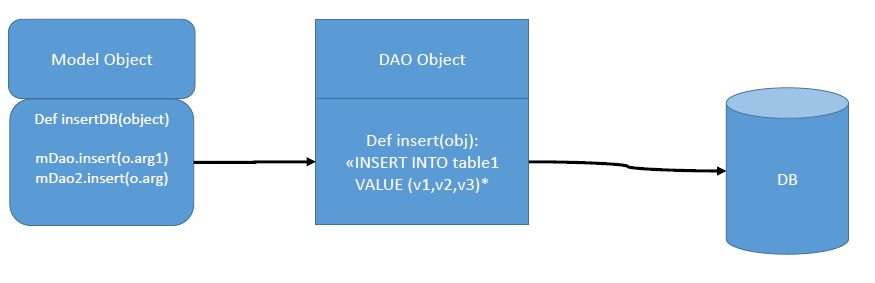
* Left/right outer join
  + It is regarded good coding style to never use right join and always reformulate the statement
* Full Outer Join

# Lecture 9

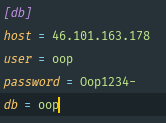
## Model View Controller (See Glossary)

* **Model**
  + Interacts with the database, processes data from or to the database and communicates only with the Controller
* **View**
  + User Interface and the only thing the end user sees. It communicates only with the Controller and can except dynamic values from the controller.
* **Controller**
  + Brain and Middel-Man that receives and passes input from and to the user and gets and sends data from or to the model

## Data Access objects (Persistence Layer)

Is between the Model and the Database and separates a data resource’s client form its data access mechanism. It implements the commands to the Database and is reusable (i.e. if the data source changes only the DAO has to be newly written and not the whole code.).

## Create Database Connection



## Full standard Query

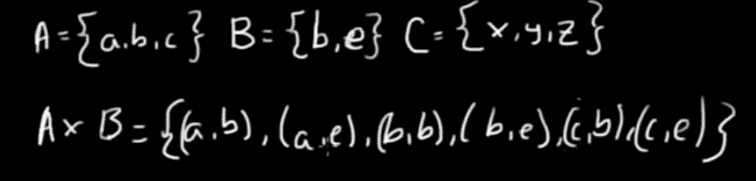


# 

## Commit statement

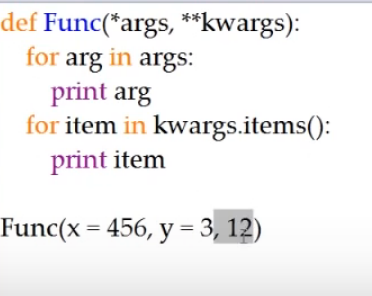
* Confirms the executed statements to the database. Without it, the data does not actually get transferred into the database.
* There is no clear time when to commit transactions.
  + Commit when all the data which belongs to one object has been executed into the database.

# Glossary

* **Code**: The sequence of instructions in a program
* **Syntax**: The set of legal structures and commands that can be used in a particular programming language
* **Output**: The messages printed to the user by a program
* **Console**: The text box onto which output Is printed
* **Compiling**: Process of translating a program into a form that the machine understands and can execute.
* **Interpreting**: Process of directly passing a program into machine instructions.
* **Variables**: Piece of memory that can store a value, resp. a label for a location in memory. In Python we may use reuse the same variable to store values of any type.
* **Function**: Groups multiple lines of code into one reusable block and allows to specify input parameters which are used in the function Common tasks are best written into a function.
* **Lambda**: Tool for building function objects.
* **Scope**: Part of a program where a variable is accessible
* **Parse**: To split a file or other input into pieces of data that can be easily stored or manipulated.
* **Errors**:
  + **Syntax Errors**:
    - leaving out a keyword
    - putting a keyword in the wrong place
    - leaving out a symbol, such as a colon, comma or brackets
    - misspelling a keyword
    - incorrect indentation
    - empty block
  + **Runtime Errors:**
    - division by zero
    - performing an operation on incompatible types
    - using an identifier which has not been defined
    - accessing a list element, dictionary value or object attribute which doesn’t exist
    - trying to access a file which doesn’t exist
  + **Logical Errors:**
    - using the wrong variable name
    - indenting a block to the wrong level
    - using integer division instead of floating-point division
    - getting operator precedence wrong
    - making a mistake in a boolean expression
    - off-by-one, and other numerical errors
* **Class:** way of grouping together related data and functions which act upon that data
* **Composition:** Way of aggregating objects together by making some objects attributes of other objects. If we can express a relationship between two classes using the phrase has-a, it is a composition relationship. (A car “HAS A” Body, and “HAS AN” engine.
* **Inheritance:** Way of arranging objects in a hierarchy from most general to the most specific. If we can describe the relationship between two objects using the phrase is-a, that relationship is inheritance. (One objects “IS A” form of another object)
* **API:** Application programming interface is code that allows two software programs to communicate with each other. It provides the guidance and assistance that a developer should need to interface two systems
* **Database:** Regular data structure that stores organized information that can be easily managed, accessed and updated
* **Relationship Cardinality:** Degree of relationship, number of occurrences in one entitiy which are associated to the number of occurrences in another. There are three degrees of relationship (1:1, 1:M, M:N)
* **ERD:** Entity relationship diagrams
* **Normalization:** Technique of organizing the data into multiple related tables, to minimize data redundancy.
* **Data redundancy:** repetition of similar data at multiple places.
* **Data integrity:** ensure data is recorded exactly as intended (such as a database correctly rejecting mututally exclusive possibilities) and upon later retrieval, ensure the data is the same as it was when it was originally recorded. -> Aims to prevent unintentional changes to information.
* **Data inconsistency:** Anything that affects the data integrity results in data inconsistency, especially if you’re using the database as a System of Record.
* **Partial dependency:** In many to many relationships, we might need more than 1 primary key to uniquely identify each row of the table. If we have a primary key composed out of two columns, it might be the case that we have other columns that only depend on one of the two primary key columns. This means we have partial dependency.
* **Transitive dependency:** If we have columns in our table that do not depend on another column of the table
* **Cartesian Product:**
* **Model View Controller [MVC]:** One of the most frequently used software architectural design patterns that separates application functionality and promotes organized and structured programming.

# Random Stuff

## Args and Kwargs



### Args

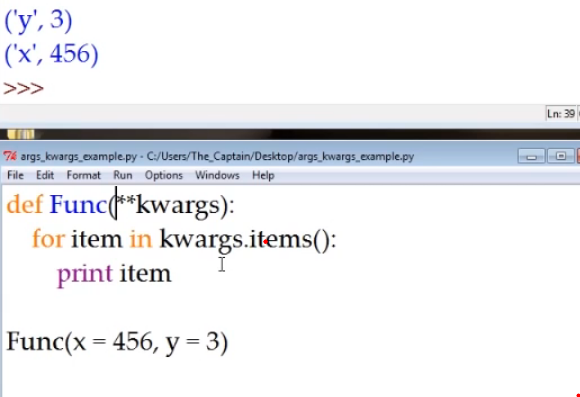
\*args

* Takes UNLIMITED regular arguments
* Place in function argument to use
* Prevents program from crashing if we don’t know how many arguments we will pass in!
* If we pass in a list and want the content to be treated as individual arguments, we can put a \* before the name of the argument function(\*list)

### Kwargs (Keyword Arguments)

\*kwargs

* Declared variables within the arguments
* Use to set default values that may be overridden
* \*\*kwargs is taken as a dictionary



## Getters and Setters

<https://www.youtube.com/watch?v=jCzT9XFZ5bw> -> Getters and setters through property decorators.