**PYTHON:**

Primitive Data Types:

Numbers: Integers/decimal(floats)/complex

Convert: int to float etc.… Random: importRandom print(randomrand.range(int)

Strings: literal text

Booleans: True/False F=0 T=1

Composite:

Tuples: immutable/groups of values/container for fixed sequence of data objects

Index at 0/ built in methods/ ()/ can store any data we want inside

Conditionals: if/elif/else

Loop: range

Lists: mutable/arrays in JS/index=0/built in methods/ square brackets/can store any data 6y66xcca

Left: location

Right: row

Append: add

Shing = num:num

Dictionaries: Group of Key Value pairs/{} /not indexed by numbers/extract values with keys/can contain nested sequences/ list dictionary tuple

Common Functions:

Print Type – find out variables’ data type

Print Len- length of data types

Concatenation:

“\_\_” + \_\_\_

“\_\_”, \_\_\_\_

Python can’t add string + number -> str(num) = str + str

F strings:

{f” {}{}}

Print(“~~~~{}{}).function(\_\_,\_\_,\_\_)

Return: return value/exit function vs print: display in console

Functions: a block of code to run at any point on the app

Can be defined called anywhere on the file

Takes in parameters

You give it arguments

Arguments become parameters when you call invoke function

Parameter: define output of functions

Argument: actual input

Define p actual pass a

OOP: Property: attribute/trunk leaves branches -> can be protected- no longer accessible in instance

Behaviors: methods

OOP: allows apps to be efficient/allows for scale up/maintain easily

Everything is an object

Planning: define attributes with blueprints(classes) => instance of class in function

-grouping properties with functionalities by object = implemented with classes -> constructor function:creates actual instance of obj

Self: referring to self/ rep of instance of obj

Constructor: Special method

-contains instructions for making a new instance of a class

-methods that the program will run whenever constants new instance

-whenever you want to do sth once a new instance is constructed- inside a constructor

Inheritance of a class:

Private not inherited

\

Methods: can do by protected though

Declaring a variable= creating an instance of a class

Setting attributes: - parameter of constructor function- applying to attributes of instances

-parameter of an instance method within a class =self

-self reference to instance of class

-pass on arguments to specify a shoe’s instance attributes

Methods: functions that belong to a class

Calling method from the instance=> implicit passage of self

Class V Static Methods:

Can also set and declare attributes on class itself= pass self to refer to instance of object-instance method = Shared among all instance of classes

Class method: class not instance cls()

Static: no lock to instance or class = values as arguments needed

Dictionaries V OOP classes:

D:

Have unique keys for stored values

Have access to many pieces of data through a single dict

Can add new kv pairs to an existing dictionary

Flexible but doesn’t suit data that should be uniform

Can share functions but does not inherently share these functions with other dicts

OOP Classes:

Have unique attribute names for shared classes

Have access to many attributes, values through a single object instance using an attribute name

Generally, cannot add new attributes to an existing object instance

Keeps data uniform but has little capacity for non-uniform data

A class can have instance methods not always shared across all instances

Relationships:

1-1

1 to many

Many to many

Not null = not blank

Foreign key= singular

Primary key = plural

Normalization:

1: Each column can have only one value

2: each column on table that is not a key must have unique values

3: you cannot have a non-key column that is dependent on another key column

Updated At: current timestamp on update current timestamp

Set=updating

Sql=structured query language

Select Insert Update Delete

Joins: join on same id

Left join: join whether or not same id

Include create schema

Drop schema= if only update schema already forward engineered/ will delete all data in database

Flask:

Micro framework for building web apps

Easy to setup

Easy to use libraries for all our server needs

MVC:

Model: Methods

May build database tables

Handles logic that relies on data

Interferes with database

View: templates

HTML page that gets sent to client

Many contain some logic to be handled by template engine

Controller: /mysqlconnection.py/routes

Receives incoming requests

Minimal logic

GET: shows all data into URL

POST: secure request made to the server

Hides the data being sent to a server on a body – forms – request.form

Post req- submit – flask – access

Form has attributes of name <- request.form[“\_\_\_”]

Keys assigned to this object- tuple

Redirecting: don’t render on post – duplicate sending of data

* Especially credit card data!! Charged twice

Session: all routes can share information/mainly to keep track who is logged on

Request.from cannot send in to other get requests – session

Session[‘name’] =request.form[‘name’]

Hidden inputs: form fields that are hidden from user

Transfer information between different pages

Retrieving and display data: validate

Data {Need to make data dict first!}

Circular imports: Many to Many -> depends on each other -> infinite loop

Import file instead classes