CONTENIDO

1. PROGRAMACIÓN

1.1. Qué es la Programación

La programación es una disciplina que combina parte de otras disciplinas como las Matemáticas, Ingeniería y la Ciencia. Sin embargo, la habilidad más importante es resolver problemas. Es lo que se hará todos los días como programador o programadora.

La programación es una secuencia de instrucciones que le damos a la computadora para que haga lo que nosotros deseamos. Podemos construir una aplicación web, móvil, un programa que lleve cohetes a la luna o marte, resolver problemas de finanzas. Casi todos los programas tienen un input, output, operaciones matemáticas, ejecución condicional y repeticiones, entre otras.

1.2. Por qué programar

1.3. El camino del programador

1.4. La vida del programador

1.5. Retos actuales en la programación

2. ANTES DE INICIAR

A continuación, se presenta información relevante sobre los lenguajes de programación más importantes de la actualidad, seguido de algunos pasos para su instalación y uso:

2.1. PSeInt

2.1.1. Qué es PSeInt

2.1.2. Instalar PSeInt

2.1.3. Configurar PSeInt

2.1.4. Usar PSeInt

2.2. Python

2.2.1. El ZEN de Python

Hermoso es mejor que feo.
Explícito es mejor que implícito.
Simple es mejor que complejo.
Complejo es mejor que complicado.
Plano es mejor que anidado.
Escaso es mejor que denso.
La legibilidad cuenta.
Los casos especiales no son lo suficientemente especiales para romper las reglas.
Lo práctico supera a la pureza.
Los errores no deben pasar en silencio.
A menos que sean silenciados.
En cara a la ambigüedad, rechazar la tentación de adivinar.
Debe haber una - y preferiblemente sólo una - manera obvia de hacerlo.
Aunque esa manera puede no ser obvia en un primer momento a menos que seas holandés.
Ahora es mejor que nunca.
Aunque “nunca” es a menudo mejor que “ahora mismo”.
Si la aplicación es difícil de explicar, es una mala idea.
Si la aplicación es fácil de explicar, puede ser una buena idea.
Los espacios de nombres son una gran idea ¡hay que hacer más de eso!

2.2.2. Qué es Python

Python es uno de los mejores lenguajes para principiantes porque tiene una sintaxis clara, una gran comunidad y esto hace que el lenguaje sea muy amigable para los que están iniciando. Python está diseñado para ser fácil de usar, a diferencia de otros lenguajes donde la prioridad es ser rápido y eficiente. Python no es de los lenguajes más rápidos, pero casi nunca importa. Es el tercer lenguaje, según Github, entre los más populares. En StackOverflow se comenta que es uno de los lenguajes que mayor popularidad está obteniendo: ""Python cuando podamos, C++ cuando necesitemos"".

Python es un lenguaje de programación creado por Guido Van Rossum, con una sintaxis muy limpia, ideado para enseñar a la gente a programar bien. Se trata de un lenguaje interpretado o de script. Las características principales de Python son las siguientes:

- Legible: sintaxis intuitiva y estricta.
- Productivo: ahorra mucho código.
- Portable: para todo sistema operativo.
- Recargado: viene con muchas librerías por defecto.

2.2.3. Instalar Python

2.2.3.1. Windows

Para instalar Python en Windows, vaya al sitio https://www.Python.org/downloads/ y presione sobre el botón Download Python 3.12.1 (Versión en la que se elabora este documento). Se descargará un archivo de instalación con el nombre Python-3.12.1.exe, ejecútelo. Y siga los pasos de instalación. Al finalizar la instalación, haga lo siguiente para corroborar una instalación correcta:

- Presiona las teclas Windows + R para abrir la ventana de Ejecutar.
- Una vez abierta la ventana Ejecutar escribe el comando cmd y presiona ctrl+shift+enter para ejecutar una línea de comandos con permisos de administrador.
- Windows te preguntará si quieres abrir el Procesador de comandos de Windows con permisos de administrador, presiona sí.
- En la línea de comandos escribe:

```
py --version
```

2.2.3.2. MacOs

La forma sencilla es tener instalado homebrew y usar el comando: ** Para instalar la Versión 2.7**

```
brew install python
```

Para instalar la Versión 3.x

```
brew install python3
```

2.2.3.3. Linux

Generalmente Linux ya lo trae instalado, para comprobarlo puedes ejecutar en la terminal el comando Versión 2.7.

python -v	
python3 -v	Versión 3 de Python

Si el comando arroja un error quiere decir que no lo tienes instalado, en ese caso los pasos para instalarlo cambian un poco de acuerdo con la distribución de linux que estés usando. Generalmente el gestor de paquetes de la distribución de Linux tiene el paquete de Python. Si eres usuario de Ubuntu o Debian por ejemplo puedes usar este comando para instalar la versión 3.1:

```
$ sudo apt-get install python3.1
```

Si eres usuario de Red Hat o Centos por ejemplo puedes usar este comando para instalar Python

```
$ sudo yum install python
```

Si usas otra distribución o no has podido instalar Python, o si eres usuario habitual de linux también puedes descargar los archivos para instalarlo manualmente.

2.2.4. Usar Python: Terminal

Para usar Python debemos tener un editor de texto abierto y una terminal o cmd (línea de comandos en Windows) como administrador. **Nota:** Este documento utiliza como referencia el editor de código Visual Studio Code (1] Descargar Visual Studio Code y 2] extensiones vscode), para abrir la terminal, se debe ir a: Terminal / Run Terminal (Ctrl + Shift + ñ).

py --version	Muestra la versión de Python

clear		Limpia la terminal
py		Abre la consola de Python
>>>	a = 'Hola Mundo'	Almacena en la variable 'a' el valor '¡Hola Mundo!'
>>>	print(a)	Muestra el valor en consola '¡Hola Mundo!'
exit()		Cierra la consola de Python

Otros comandos útiles en la terminal o en el cmd:

cd ..			Devolverse una carpeta
cd carpeta/carpeta			Moverse a una carpeta en específico
touch archivo.py			Crea un nuevo archivo Python
archivo.py	1	a = 'Hola Mundo'	Almacena en la variable 'a' el valor '¡Hola Mundo!'
archivo.py	2	print(a)	Muestra en la terminal el valor '¡Hola Mundo!'
py archivo.py			Abre un archivo de Python
mkdir prueba			Crea la carpeta 'prueba'
mv archivo.py prueba			Mueve el archivo.py a la carpeta 'prueba'
cd prueba			Moverse a una carpeta prueba
ls			Información de la carpeta 'prueba'
rm archivo.py			Elimina el archivo.py (También carpetas)
cd ..			Devolverse una carpeta
rm prueba			Elimina la carpeta prueba

Cuando usamos Python debemos atender ciertas reglas de la comunidad, y esto lo encuentras en el libro PEP (Propuestas de mejora de Python).

2.3. JavaScript

2.3.1. Qué es JavaScript

2.3.2. Instalar JavaScript

2.3.3. Configurar JavaScript

2.3.4. Usar JavaScript

2.4. Php

2.4.1. Qué es Php

2.4.2. Instalar Php

2.4.3. Configurar Php

2.4.4. Usar Php

2.5. Java

2.5.1. Qué es Java

2.5.2. Instalar Java

2.5.3. Configurar Java

2.5.4. Usar Java

2.6. C#

2.6.1. Qué es C#

2.6.2. Instalar C#

2.6.3. Configurar C#

2.6.4. Usar C#

3. LÓGICA DE PROGRAMACIÓN

3.1. Algoritmo y comentarios

3.1.1. Explicación

En informática, se llaman **algoritmos** el conjunto de instrucciones sistemáticas y previamente definidas que se utilizan para realizar una determinada tarea. Estas instrucciones están ordenadas y acotadas a manera de pasos a seguir para alcanzar un objetivo.

Todo algoritmo tiene una entrada, conocida como input y una salida, conocida como output, y entre medias, están las instrucciones o secuencia de pasos a seguir. Estos pasos deben estar ordenados y, sobre todo, deben ser una serie finita de operaciones que permitan conseguir una determinada solución.

- ¿Qué son los algoritmos?

Un **comentario** en programación es la línea de texto en nuestro código fuente que el compilador ignora. Sabemos que nuestro código fuente está compuesto de instrucciones, y el compilador traduce esas instrucciones del lenguaje de programación que estamos usando a lenguaje máquina.

- ¿Qué es un comentario en Programación?

3.1.2. Ejemplos

Algoritmos y comentarios en:

- PSeInt
- Python
- JavaScript
- Php
- Java
- C#

3.1.3. Prácticas

1. Describa las partes de un algoritmo en cada uno de los lenguajes de programación vistos a través de comentarios.

3.2. Salida de datos y concatenación básica

3.2.1. Explicación

Cualquier programa informático debe comunicarse con los usuarios, tanto para recibir la información de entrada como para devolver los resultados de su ejecución. Esto es lo que se conoce con el nombre de procesos de entrada y salida de datos. La **salida de datos** son los resultados, es decir, el conjunto de instrucciones que toman los datos finales de la memoria interna y los envían a los dispositivos externos (pantallas o dispositivos de almacenamiento).

- Entrada y salida de datos

En el ámbito de la informática, la **concatenación** es una operación que consiste en la unión de dos o más caracteres para desarrollar una cadena de caracteres, conocida en inglés como *string*. Esta cadena es una secuencia finita y ordenada de elementos que forman parte de un determinado lenguaje formal. La concatenación puede llevarse a cabo incluso uniendo dos cadenas de caracteres o enlazando un carácter a otra cadena.

A lo largo del arduo y complejo proceso que supone el desarrollo de un programa informático, ya sea una aplicación, un videojuego o cualquier otra clase, se usa muy a menudo el concepto de **concatenación**, ya que es una de las operaciones más útiles de las cadenas de texto, otro de los elementos fundamentales de la programación

- Definición de concatenación

3.2.2. Ejemplos

Salida de datos en:

- PSeInt
- Python
- JavaScript
- Php
- Java
- C#

3.2.3. Prácticas

2. En una app web, para el módulo de usuarios en el rol de usuario, muestre en pantalla: Id, nombre, descripción.
3. En una app web, para el módulo de productos en la categoría de un producto, muestre en pantalla: Id, nombre, descripción.

3.3. Identificadores, tipos de datos, constantes y variables

3.3.1. Explicación

Los **identificadores** nos permiten usar fácilmente los datos que están almacenados en memoria, despreocupándonos de su posición en memoria, su dirección. Con objeto de aportar legibilidad a los programas, debemos usar identificadores auto explicativos, informando implícitamente del cometido de una variable, constante, arreglo, clase, objeto, atributo, método, entre otros.

- Identificadores01
- Identificadores02

En programación, los **tipos de datos** son una herramienta fundamental para el desarrollo de software. Estos definen el tipo de información que puede ser almacenado en una variable o estructura de datos; por ejemplo, carácter, cadena, entero, decimal, booleano, fecha, entre otros.

- ¿Qué son los tipos de datos?

Una **constante** es un valor que se establece en una sección del código y permanece constante durante la ejecución del programa. A diferencia de las variables, las constantes no cambian de valor mientras el programa está en funcionamiento. Las constantes se utilizan para asegurar que un valor crítico permanezca inalterable y no pueda ser modificado accidental o intencionalmente. En programación dato que no cambia su valor.

- ¿Qué es una constante en programación?

Una **variable** representa un contenedor o un espacio en la memoria física o virtual de una computadora, donde se almacenan distintos tipos de datos (valores) durante la ejecución de un programa. A cada variable se le asigna un nombre descriptivo o un identificador que se refiere al valor guardado. Los datos almacenados pueden cambiar de valor o ser constantes.

- ¿Qué es una variable en programación?

3.3.2. Ejemplos

Identificadores, tipos de datos, constantes y variables en:

- PSeInt
- Python
- JavaScript
- Php
- Java
- C#

3.3.3. Prácticas

4. En una app web, para el rol de un usuario, utilice variables para mostrar en pantalla: Id, nombre, descripción.
5. En una app web, para la categoría de un producto, utilice variables para mostrar en pantalla: Id, nombre, descripción.

3.4. Conversión de tipos de datos

3.4.1. Explicación

Los lenguajes de programación disponen de un conjunto de funciones que permiten pasar cualquier tipo de dato a cualquier tipo de dato. Por ejemplo, Processing (lenguaje de programación) dispone de la función int() que convierte el string "123" en el entero 123 y el string "UnoDosTres" en el entero 0. ¿Por qué hace esta conversión? Simplemente porque los programadores de Processing así lo decidieron.

Pero no es lo mismo hacer casting que usar una función de conversión. En un **casting** lo que hacemos es interpretar el contenido de la memoria como si fuera otro tipo de dato: se usa solo un valor. Por contra, en una función de **conversión** lo que hacemos es usar un valor como argumento de una función que nos retorna otro valor: se usan dos valores.

- Conversión de Tipos

3.4.2. Ejemplos

Conversión de tipos de datos en:

- PSeInt
- Python
- JavaScript
- Php
- Java
- C#

3.4.3. Prácticas

3.5. Expresiones y sentencias

3.5.1. Explicación

Una **expresión** es una unidad sintáctica del lenguaje que consiste en una combinación de uno o más valores, variables y operadores, que pueden ser evaluados a un valor. Las expresiones no realizan una acción, sino que devuelven su resultado.

En los lenguajes imperativos una **sentencia** (statement) es una unidad sintáctica del lenguaje que expresa una acción a realizar (por ejemplo, imprimir un valor en la pantalla). Las sentencias no devuelven un valor, sino que realizan una acción. Un tipo habitual de sentencia es la sentencia de asignación, que asigna el valor de una expresión a una variable, mediante el operador de asignación (=).

variable = expresión

Primero se evalúa la expresión, y al objeto resultante se le asigna el nombre de la variable.

3.5.2. Ejemplos

Expresiones en:

- PSeInt
- Python
- JavaScript
- Php
- Java
- C#

3.5.3. Prácticas

3.6. Operadores de concatenación, aritméticos, asignación, comparación y lógicos

3.6.1. Explicación

Los operadores son contextuales, dependen del tipo de valor. Un valor es la representación de una entidad que puede ser manipulada por un programa. Podemos conocer el tipo del valor con type() y nos devolverá algo similar a <class 'init'>, <class 'float'>, <class 'str'>. Dependiendo del tipo los operadores van a funcionar de manera diferente.

Los operadores son símbolos que le indican al intérprete que realice una operación específica, com

o aritmética, comparación, lógica, etc.

- Operadores básicos en Python con ejemplos

Los operadores aritméticos o arithmetic operators son los más comunes que nos podemos encontrar, y nos permiten realizar operaciones aritméticas sencillas, como pueden ser la suma, resta o exponente. En programación estos operadores son muy similares a nuestras clases básicas de matemáticas.

- //: Es división de entero, básicamente tiramos la parte decimal
- %: Es el residuo de la división, lo que te sobra.
- **: Exponente

- Ver ejemplo de operadores aritméticos

Los operadores de asignación o assignment operators nos permiten realizar una operación y almacenar su resultado en la variable inicial.

- Operadores de asignación
- Ver ejemplo de operadores de asignación

En Python las **constantes** no existen. Para guardar un valor constante se utiliza una variable pero, por convención, se utilizan las letras mayúsculas para darle nombre a dicha variable. Así, si al programar nos encontramos con un identificador en mayúsculas sabremos que no debe ser alterado.

- Ver ejemplo de operadores de comparación

Para comprender el flujo de nuestro programa debemos entender un poco sobre estructuras y expresiones booleanas

- == se refiere a igualdad
- != no hay igualdad.
- > mayor que
- < menor que
- >= mayor o igual
- <= menor o igual

- and unicamente es verdadero cuando ambos valores son verdaderos
- or es verdadero cuando uno de los dos valores es verdadero.
- not es lo contrario al valor. Falso es Verdadero. Verdadero es Falso.
- Ver ejemplo de operadores lógicos

3.6.2. Ejemplos

Operadores en:

- PSeInt
- Python
- JavaScript
- Php
- Java
- C#

3.6.3. Prácticas

3.7. Entrada de datos

3.7.1. Explicación

La **entrada de datos** es …

3.7.2. Ejemplos

Entrada de datos en:

- PSeInt
- Python
- JavaScript
- Php
- Java
- C#

3.7.3. Prácticas

3.8. Proyectos

En los lenguajes de programación vistos (PSeInt, Python, javascript, php, java, c#), desarrolle los siguientes requisitos funcionales:

3.8.1. Proyecto Calculadora

3.8.1.1. Módulo Operaciones

El Sistema de debe permitir al administrador registrar ...
El Sistema de debe permitir al administrador consultar ...
El Sistema de debe permitir al administrador actualizar ...
El Sistema de debe permitir al administrador eliminar ...

3.8.2. Proyecto Inventario

3.8.2.1. Módulo Usuarios

El Sistema de debe permitir al administrador registrar usuario.
El Sistema de debe permitir al administrador consultar usuarios.
El Sistema de debe permitir al administrador actualizar usuario.
El Sistema de debe permitir al administrador eliminar usuario.

3.8.2.2. Módulo Productos

El Sistema de debe permitir al administrador registrar producto.
El Sistema de debe permitir al administrador consultar productos.
El Sistema de debe permitir al administrador actualizar producto.
El Sistema de debe permitir al administrador eliminar producto.

4. ESTRUCTURAS DE DATOS

4.1. Arreglos

4.1.1. Explicación

Listas (list)

Son un grupo o array de datos, puede contener cualquiera de los datos anteriores. Sintaxis: [1,2,3, ”hola” , [1,2,3]], [1,“Hola”,True]

Python y todos los lenguajes nos ofrecen constructos mucho más poderosos, haciendo que el desarrollo de nuestro software sea

- Más sofisticado
- Más legible
- Más fácil de implementar

Estos constructos se llaman Estructuras de Datos que nos permiten agrupar de distintas maneras varios valores y elementos para poderlos manipular con mayor facilidad. Las listas las vas a utilizar durante toda tu carrera dentro de la programación e ingeniería de Software.

Las listas son una secuencia de valores. A diferencia de los strings, las listas pueden tener cualquier tipo de valor. También, a diferencia de los strings, son mutables, podemos agregar y eliminar elementos. En Python, las listas son referenciales. Una lista no guarda en memoria los objetos, sólo guarda la referencia hacia donde viven los objetos en memoria

- Se inician con [] o con la built-in function list.

Operaciones con listas

Ahora que ya entiendes cómo funcionan las listas, podemos ver qué tipo de operaciones y métodos podemos utilizar para modificarlas, manipularlas y realizar diferentes tipos de cómputos con esta Estructura de Datos.

- El operador +(suma) concatena dos o más listas.
- El operador *(multiplicación) repite los elementos de la misma lista tantas veces los queramos multiplicar
- Sólo podemos utilizar +(suma) y *(multiplicación).

Las listas tienen varios métodos que podemos utilizar.

- append nos permite añadir elementos a listas. Cambia el tamaño de la lista.
- pop nos permite sacar el último elemento de la lista. También recibe un índice y esto nos permite elegir qué elemento queremos eliminar.
- sort modifica la propia lista y ordenarla de mayor a menor. Existe otro método llamado sorted, que también ordena la lista, pero genera una nueva instancia de la lista
- delnos permite eliminar elementos vía indices, funciona con slices
- remove nos permite es pasarle un valor para que Python compare internamente los valores y determina cuál de ellos hace match o son iguales para eliminarlos.

Tuplas (tuples)

También son un grupo de datos igual que una lista con la diferencia que una tupla después de creada no se puede modificar. Sintaxis: (1,2,3, ”hola” , (1,2,3)), (1,“Hola”,True). Sin embargo, jamás podremos cambiar los elementos dentro de esa Tupla.

Tuplas y conjuntos

Tuplas(tuples) son iguales a las listas, la única diferencia es que son inmutables, la diferencia con los strings es que pueden recibir muchos tipos valores. Son una serie de valores separados por comas, casi siempre se le agregan paréntesis para que sea mucho más legible.

Para poderla inicializar utilizamos la función tuple.

Uno de sus usos muy comunes es cuando queremos regresar más de un valor en nuestra función. Una de las características de las Estructuras de Datos es que cada una de ellas nos sirve para algo especifico. No existe en programación una navaja suiza que nos sirva para todos. Los mejores programas son aquellos que utilizan la herramienta correcta para el trabajo correcto.

Conjutos(sets) nacen de la teoría de conjuntos. Son una de las Estructuras más importantes y se parecen a las listas, podemos añadir varios elementos al conjunto, pero no pueden existir elementos duplicados. A diferencia de los tuples podemos agregar y eliminar, son mutables. Los sets se pueden inicializar con la función set. Una recomendación es inicializarlos con esta función para no causar confusión con los diccionarios.

- add nos sirve añadir elementos.
- remove nos permite eliminar elementos.

Diccionarios (dict)

Son un grupo de datos que se acceden a partir de una clave. En los diccionarios tienes un grupo de datos con un formato: la primera cadena o número será la clave para acceder al segundo dato, el segundo dato será el dato al cual accederás con la llave. Recuerda que los diccionarios son listas de llave:valor. Sintaxis: {“clave”:”valor”}, {“nombre”:”Fernando”}

- Ver ejemplo de estructuras de datos

Los diccionarios se conocen con diferentes nombres a lo largo de los lenguajes de programación como HashMaps, Mapas, Objetos, etc. En Python se conocen como Diccionarios. Un diccionario es similar a una lista sabiendo que podemos acceder a través de un indice, pero en el caso de las listas este índice debe ser un número entero. Con los diccionarios puede ser cualquier objeto, normalmente los verán con strings para ser más explicitos, pero funcionan con muchos tipos de llaves.

Un diccionario es una asociación entre llaves(keys) y valores(values) y la referencia en Python es muy precisa. Si abres un diccionario verás muchas palabras y cada palabra tiene su definición.

Para iniciar un diccionario se usa {} o con la función dict

Estos también tienen varios métodos. Siempre puedes usar la función dir para saber todos los métodos que puedes usar con un objeto. Si queremos ciclar a lo largo de un diccionario tenemos las opciones:

- keys: nos imprime una lista de las llaves
- values nos imprime una lista de los valores
- items. nos manda una lista de tuplas de los valores

4.1.2. Ejemplos

Arreglos en:

- PSeInt
- Python
- JavaScript
- Php
- Java
- C#

4.1.3. Prácticas

4.2. Conversión de estructuras de datos

1.1.1. Explicación

1.1.2. Ejemplos

1.1.2.1. En PSeInt

- Salida de Datos en PSeInt

1.1.2.2. En Python

- Salida de Datos en Python

1.1.2.3. En JavaScript

- Salida de Datos en JavaScript

1.1.2.4. En Php

- Salida de Datos en Php

1.1.2.5. En Java

- Salida de Datos en Java

1.1.2.6. En C#

- Salida de Datos en #

1.1.3. Prácticas

1.2. Proyecto: Módulo Usuarios

1. El Sistema de debe permitir al administrador, vendedor y cliente iniciar sesión.

- En PSeInt
- En Python
- En JavaScript
- En Php
- En Java
- En C#

1.3. Proyecto: Módulo Ventas

1.

5. PROGRAMACIÓN ESTRUCTURADA

a. Secuencial

b. Condicional

i. Condicional simple y doble

En esta clase seguiremos construyendo nuestro proyecto PlatziVentas haciéndolo un poco más interesante y conoceremos un poco sobre las Estructuras condicionales. En Python es importante importante la indentación, de esa manera identifica donde empieza y termina un bloque de código sin necesidad de llaves {} como en otros lenguajes.

Ten en cuenta que lo que contiene los paréntesis es la comparación que debe cumplir para que los elementos se cumplan. Los condicionales tienen la siguiente estructura.

```
if ( a > b ):
    elementos
elif ( a == b ):
    elementos
else:
    elementos
```

ii. Condicional compuesta

iii. Condicional múltiple

c. Repetición

Las iteraciones es uno de los conceptos más importantes en la programación. En Python existen muchas maneras de iterar pero las dos principales son los for loops y while loops. Los for loops nos permiten iterar a través de una secuencia y los while loops nos permiten iterara hasta cuando una condición se vuelva falsa.

- Tienen dos keywords break y continue que nos permiten salir anticipadamente de la iteración
- Se usan cuando se quiere ejecutar varias veces una o varias instrucciones.
- for [variable] in [secuencia]:

Es una convención usar la letra i como variable en nuestro for, pero podemos colocar la que queramos.

- range: Nos da un objeto rango, es un iterador sobre el cual podemos generar secuencias.

i. Repetición infinita

Al igual que las for loops, las while loops nos sirve para iterar, pero las for loops nos sirve para iterar a lo largo de una secuencia mientras que las while loops nos sirve para iterar mientras una condición sea verdadera. Si no tenemos un mecanismo para convertir el mecanismo en falsedad, entonces nuestro while loops se ira al infinito(infinite loop).

En este caso while tiene una condición que determina hasta cuándo se ejecutará. O sea que dejará de ejecutarse en el momento en que la condición deje de ser cierta. La estructura de un while es la siguiente:

```
while (condición):
    elementos
```

Ejemplo:

```
>>> x = 0
>>> while x < 10:
... print x
... x += 1
```

En este ejemplo preguntará si es menor que diez. Dado que es menor imprimirá x y luego sumará una unidad a x. Luego x es 1 y como sigue siendo menor a diez se seguirá ejecutando, y así sucesivamente hasta que x llegue a ser mayor o igual a 10.

Son un grupo o array de datos, puede contener cualquiera de los datos anteriores. Sintaxis: [1,2,3, ”hola” , [1,2,3]], [1,“Hola”,True]

ii. Repetición Finita

El bucle de for lo puedes usar de la siguiente forma: recorres una cadena o lista a la cual va a tomar el elemento en cuestión con la siguiente estructura:

```
for i in ____:
    elementos
```

Ejemplo:

```
for i in range(10):
    print (i)
```

En este caso recorrerá una lista de diez elementos, es decir el _print i _de ejecutar diez veces. Ahora i va a tomar cada valor de la lista, entonces este for imprimirá los números del 0 al 9 (recordar que en un range vas hasta el número puesto -1).

Son un grupo o array de datos, puede contener cualquiera de los datos anteriores. Sintaxis: [1,2,3, ”hola” , [1,2,3]], [1,“Hola”,True]

iii. Iterators and generators

Aunque no lo sepas, probablemente ya utilices iterators en tu vida diaria como programador de Python. Un iterator es simplemente un objeto que cumple con los requisitos del Iteration Protocol (protocolo de iteración) y por lo tanto puede ser utilizado en ciclos. Por ejemplo,

```
for i in range(10):
    print(i)
```

En este caso, la función range es un iterable que regresa un nuevo valor en cada ciclo. Para crear un objeto que sea un iterable, y por lo tanto, implemente el protocolo de iteración, debemos hacer tres cosas:

- Crear una clase que implemente los métodos iter y next
- iter debe regresar el objeto sobre el cual se iterará
- next debe regresar el siguiente valor y aventar la excepción StopIteration cuando ya no hayan elementos sobre los cual iterar.

Por su parte, los generators son simplemente una forma rápida de crear iterables sin la necesidad de declarar una clase que implemente el protocolo de iteración. Para crear un generator simplemente declaramos una función y utilizamos el keyword yield en vez de return para regresar el siguiente valor en una iteración. Por ejemplo,

```
def fibonacci(max):
    a, b = 0, 1
    while a < max:
```

```
        yield a
        a, b = b, a+b
```

Es importante recalcar que una vez que se ha agotado un generator ya no podemos utlizarlo y debemos crear una nueva instancia. Por ejemplo,

```
fib1 = fibonacci(20)
fib_nums = [num for num in fib1]
...
double_fib_nums = [num * 2 for num in fib1] # no va a funcionar
double_fib_nums = [num * 2 for num in fibonacci(30)] # sí funciona
```

6. PROGRAMACIÓN PROCEDIMENTAL

d. Funciones

En el contexto de la programación las funciones son simplemente una agrupación de enunciados(statments) que tienen un nombre. Una función tiene un nombre, debe ser descriptivo, puede tener parámetros y puede regresar un valor después que se generó el cómputo.

Python es un lenguaje que se conoce como batteries include(baterías incluidas) esto significa que tiene una librería estándar con muchas funciones y librerías. Para declarar funciones que no son las globales, las built-in functions, necesitamos importar un módulo.

Con el keyword def declaramos una función.

Lectura recomendada: https://static.platzi.com/media/public/uploads/lambdas_09e88ca0-df9a-4098-b475-9b9b6d0f4d7a.pdf

Las funciones las defines con “def” junto a un nombre y unos paréntesis que reciben los parámetros a usar y terminas con dos puntos (:). Sintaxis: def nombre_de_la_función(parametros):

- Ver ejemplo de funciones
- Alcance de variables: Global y Local

e. Funciones Comunes

i. Longitud

```
>>> len("key")
3
```

ii. Tipo de dato

```
>>> type(4)
< type int >
```

iii. Aplicar una conversión a un conjunto como una lista:

```
>>> map(str, [1, 2, 3, 4])
['1', '2', '3', '4']
```

iv. Redondear un flotante con x número de decimales:

```
>>> round(6.3243, 1)
                6.3
```

v. Generar un rango en una lista (esto es mágico):

```
>>> range(5)
                [0, 1, 2, 3, 4]
```

vi. Sumar un conjunto:

```
>>> sum([1, 2, 4])
                7
```

vii. Organizar un conjunto:

```
>>> sorted([5, 2, 1])
                [1, 2, 5]
```

viii. Conocer los comandos que le puedes aplicar a x tipo de datos:

```
>>>Li = [5, 2, 1]
>>>dir(Li)
>>>['append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort' son posibles comandos que puedes aplicar a una lista.

ix. Información sobre una función o librería:

```
>>> help(sorted)
        (Aparecerá la documentación de la función sorted)
```

f. Métodos especiales

- cmp(self,otro)
  Método llamado cuando utilizas los operadores de comparación para comprobar si tu objeto es menor, mayor o igual al objeto pasado como parámetro.

- len(self)
  Método llamado para comprobar la longitud del objeto. Lo usas, por ejemplo, cuando llamas la función len(obj) sobre nuestro código. Como es de suponer el método te debe devolver la longitud del objeto.

- init(self,otro)
  Es un constructor de nuestra clase, es decir, es un "método especial" que se llama automáticamente cuando creas un objeto.

g. Strings

Los strings o cadenas de textos tienen un comportamiento distinto a otros tipos como los booleanos, enteros, floats. Las cadenas son secuencias de caracteres, todas se pueden acceder a través de un índice. Podemos saber la longitud de un string, cuántos caracteres se encuentran en esa secuencia. Lo podemos saber con la built-in function global llamada len.

Algo importante a tener en cuenta cuando hablamos de strings es que estos son inmutables, esto significa que cada vez que modificamos uno estamos generando un nuevo objeto en memoria. El índice de la primera letra es 0, en la programación se empieza a contar desde 0

i. Operaciones con Strings en Python

Los strings tienen varios métodos que nosotros podemos utilizar.

- upper: convierte todo el string a mayúsculas
- lower: convierte todo el string a minúsculas
- find: encuentra el indice en donde existe un patrón que nosotros definimos
- startswith: significa que empieza con algún patrón.
- endswith: significa que termina con algún patrón
- capitalize: coloca la primera letra en mayúscula y el resto en minúscula
- in y not in nos permite saber con cualquier secuencia sin una subsecuencia o substrings se encuentra adentro de la secuencia mayor.

- dir: Nos dice todos los métodos que podemos utilizar dentro de un objeto.
- help: nos imprime en pantalla el docstrings o comentario de ayuda o instrucciones que posee la función. Casi todas las funciones en Python las tienen.

ii. Operaciones con Strings y el comando Update

En esta clase seguiremos construyendo nuestro proyecto PlatziVentas, agregaremos el comando update para poder actualizar nuestros clientes y pondremos en práctica lo aprendido en clases anteriores sobre Strings.

iii. Operaciones con Strings y el comando Delete

En esta clase seguiremos construyendo nuestro proyecto PlatziVentas, agregaremos el comando delete para poder borrar nuestros clientes y pondremos en práctica lo aprendido en clases anteriores sobre Strings.**Operaciones con Strings: Slices en Python**

Los slices en Python nos permiten manejar secuencias de una manera poderosa. Slices en español significa ““rebanada””, si tenemos una secuencia de elementos y queremos una rebanada tenemos una sintaxis para definir qué pedazos queremos de esa secuencia.

secuencia[comienzo:final:pasos]

h. Collections

El módulo collections nos brinda un conjunto de objetos primitivos que nos permiten extender el comportamiento de las built-in collections que poseé Python y nos otorga estructuras de datos adicionales. Por ejemplo, si queremos extender el comportamiento de un diccionario, podemos extender la clase UserDict; para el caso de una lista, extendemos UserList; y para el caso de strings, utilizamos UserString.

Por ejemplo, si queremos tener el comportamiento de un diccionario podemos escribir el siguiente código:

```
1  class SecretDict(collections.UserDict):
2
3    def _password_is_valid(self, password):
4        …
5
6    def _get_item(self, key):
7        …
8
9    def __getitem__(self, key):
10       password, key = key.split(':')
11
12       if self._password_is_valid(password):
13           return self._get_item(key)
14
15       return None
16
17 my_secret_dict = SecretDict(...)
18 my_secret_dict['some_password:some_key'] # si el password es válido, regresa el valor
```

Otra estructura de datos que vale la pena analizar, es namedtuple. Hasta ahora, has utilizado tuples que permiten acceder a sus valores a través de índices. Sin embargo, en ocasiones es importante poder nombrar elementos (en vez de utilizar posiciones) para acceder a valores y no queremos crear una clase ya que únicamente necesitamos un contenedor de valores y no comportamiento.

```
1  Coffee = collections.NamedTuple('Coffee', ('size', 'bean', 'price'))
```

```
2  def get_coffee(coffee_type):
3      If coffee_type == 'houseblend':
4          return Coffee('large', 'premium', 10)
```

El módulo collections también nos ofrece otros primitivos que tienen la labor de facilitarnos la creación y manipulación de colecciones en Python. Por ejemplo, Counter nos permite contar de manera eficiente ocurrencias en cualquier iterable; OrderedDict nos permite crear diccionarios que poseen un orden explícito; deque nos permite crear filas (para pilas podemos utilizar la lista).

En conclusión, el módulo collections es una gran fuente de utilerías que nos permiten escribir código más "Pythonico" y más eficiente.

i. Comprehensions

Las Comprehensions son constructos que nos permiten generar una secuencia a partir de otra secuencia.

Existen tres tipos de comprehensions:

- List comprehensions

  [element for element in element_list if element_meets_condition]

- Dictionary comprehensions

  {key: element for element in element_list if element_meets_condition}

- Sets comprehensions

  {element for element in element_list if elements_meets_condition}

j. Búsquedas binarias

Uno de los conceptos más importantes que debes entender en tu carrera dentro de la programación son los algoritmos. No son más que una secuencia de instrucciones para resolver un problema específico. Búsqueda binaria lo único que hace es tratar de encontrar un resultado en una lista ordenada de tal manera que podamos razonar. Si tenemos un elemento mayor que otro, podemos simplemente usar la mitad de la lista cada vez.

```
1  import random
2
3  def binary_search(data, target, low, high) :
4      if low > high :
5          return False
6
7      mid = (low + high) // 2
8
9      if target == data[mid] :
10         return True
11     elif target < data[mid] :
12         return binary_search(data, target, low, mid - 1)
13     else :
14         return binary_search(data, target, mid + 1, high)
15
16
17 if __name__ == '__main__' :
18
19     data = [random.randint(0,100) for i in range(10)]
20     data.sort()
21     print()
22     print(data)
23     print()
24     target = int(input('What number would you like to find?: '))
25     found = binary_search(data, target, 0, len(data) - 1)
26     print()
27     print(found)
28     print()
```

k. Manipulación de archivos

```
1  import csv
2  import os
3
4  CLIENT_SCHEMA = ['name', 'company', 'email', 'position']
5  CLIENT_TABLE = '.clients.csv'
6  clients = []
```

```
7
8  def create_client(client):
9      global clients
10
11     if client not in clients:
12         clients.append(client)
13     else:
14         print('Client already in the client\'s list')
15
16
17 def list_clients():
18     print('-' * 82)
19     print('id | name\t| company\t| email\t\t\t| position\t\t |')
20     print('-' * 82)
21     for idx, client in enumerate(clients):
22         print(' {uid} | {name} \t| {company} \t| {email} \t| {position}\t |'.format(
23             uid = idx,
24             name = client['name'],
25             company = client['company'],
26             email = client['email'],
27             position = client['position']
28         ))
29     print('-' * 82)
30
31
32 def update_client(client_id, update_client):
33     global clients
34
35     if len(clients) - 1 >= client_id:
36         clients[client_id] = update_client
37     else:
38         print()
39         print('Client not in client\'s list')
40
41
42 def delete_client(client_id):
43     global clients
44
45     for idx, client in enumerate(clients):
46         if idx == client_id:
47             del clients[idx]
```

```
48              break
49
50
51  def search_client(client_name):
52      for client in clients:
53          if client['name'] != client_name:
54              continue
55          else:
56              return True
57
58
59  def _get_client_field(field_name, message='What is the client {}?: '):
60      field = None
61
62      while not field:
63          field = input(message.format(field_name))
64
65      return field
66
67
68  def _get_client_from_user():
69      client = {
70          'name': _get_client_field('name'),
71          'company': _get_client_field('company'),
72          'email': _get_client_field('email'),
73          'position': _get_client_field('position'),
74      }
75
76      return client
77
78
79  def _initialize_clients_from_storage():
80      with open(CLIENT_TABLE, mode='r') as f:
81          reader = csv.DictReader(f, fieldnames=CLIENT_SCHEMA)
82
83          for row in reader:
84              clients.append(row)
85
86
87  def _save_clients_to_storage():
88      tmp_table_name = '{}.tmp'.format(CLIENT_TABLE)
```

```
    with open(tmp_table_name, mode='w') as f:
        writer = csv.DictWriter(f, fieldnames=CLIENT_SCHEMA)
        writer.writerows(clients)

        os.remove(CLIENT_TABLE)
    os.rename(tmp_table_name, CLIENT_TABLE)


def _print_welcome():
    print('WELCOME TO PLATZI VENTAS')
    print('*' * 50)
    print('What would you like to do today?')
    print('[C]reate client')
    print('[L]ist clients')
    print('[U]pdate client')
    print('[D]elete client')
    print('[S]earch client')
    print()


if __name__ == '__main__':
    _initialize_clients_from_storage()
    _print_welcome()

    command = input()
    command = command.upper()
    print()

    if command == 'C':
        client = {
            'name': _get_client_field('name'),
            'company': _get_client_field('company'),
            'email': _get_client_field('email'),
            'position': _get_client_field('position'),
        }
        create_client(client)
        print()
    elif command == 'L':
        list_clients()
    elif command == 'U':
        client_id = int(_get_client_field('id'))
```

```
130         updated_client = _get_client_from_user()
131
132         update_client(client_id, updated_client)
133         print()
134     elif command == 'D':
135         client_id = int(_get_client_field('id'))
136
137         delete_client(client_id)
138         print()
139     elif command == 'S':
140         client_name = _get_client_field('name')
141         found = search_client(client_name)
142         print()
143         if found:
144             print('The client is in our client\'s list')
145         else:
146             print('The client: {} is not in our client\'s list'.format(client_name))
147     else:
148         print('Invalid command')
149
150     print()
151     _save_clients_to_storage()
```

I. Decoradores

Python es un lenguaje que acepta diversos paradigmas como programación orientada a objetos y la programación funcional, siendo estos los temas de nuestro siguiente módulo.

Los decoradores son una función que envuelve a otra función para modificar o extender su comportamiento. En Python las funciones son ciudadanos de primera clase, first class citizen, esto significan que las funciones pueden recibir funciones como parámetros y pueden regresar funciones. Los decoradores utilizan este concepto de manera fundamental.

En esta clase pondremos en práctica lo aprendido en la clase anterior sobre decoradores.

Por convención la función interna se llama wrapper,

Para usar los decoradores es con el símbolo de @(arroba) y lo colocamos por encima de la función. Es un sugar syntax

*args **kwargs son los argumentos que tienen keywords, es decir que tienen nombre y los argumentos posicionales, los args. Los asteriscos son simplemente una expansión.

7. PROGRAMACIÓN ORIENTADA A OBJETOS

m. ¿Qué es la programación orientada a objetos?

La programación orientada a objetos es un paradigma de programación que otorga los medios para estructurar programas de tal manera que las propiedades y comportamientos estén envueltos en objetos individuales.

Para poder entender cómo modelar estos objetos debemos tener claros cuatro principios:

- Encapsulamiento.
- Abstracción
- Herencia
- Polimorfismo

Las clases simplemente nos sirven como un molde para poder generar diferentes instancias.

n. Clases

Clases es uno de los conceptos con más definiciones en la programación, pero en resumen sólo son la representación de un objeto. Para definir la clase usas_ class_ y el nombre. En caso de tener parámetros los pones entre paréntesis.

Para crear un constructor haces una función dentro de la clase con el nombre init y de parámetros self (significa su clase misma), nombre_r y edad_r:

```
>>>  class Estudiante(object):
...      def __init__(self,nombre_r,edad_r):
...          self.nombre = nombre_r
...          self.edad = edad_r
...
...      def hola(self):
...          return "Mi nombre es %s y tengo %i" % (self.nombre, self.edad)
...
>>> e = Estudiante(“Arturo”, 21)
>>> print (e.hola())
```

Mi nombre es Arturo y tengo 21

Lo que hicimos en las dos últimas líneas fue:

- En la variable e llamamos la clase Estudiante y le pasamos la cadena “Arturo” y el entero 21.
- Imprimimos la función hola() dentro de la variable e (a la que anteriormente habíamos pasado la clase).

Y por eso se imprime la cadena “Mi nombre es Arturo y tengo 21”

o. Programación orientada a objetos en Phyton

Para declarar una clase en Python utilizamos la keyword class, después de eso le damos el nombre. Una convención en Python es que todas las clases empiecen con mayúscula y se continua con CamelCase.

Un método fundamental es dunder init(__init__). Lo único que hace es inicializar la clase basado en los parámetros que le damos al momento de construir la clase. self es una referencia a la clase. Es una forma internamente para que podamos acceder a las propiedades y métodos.

p. Scopes and namespaces

En Python, un name, también conocido como identifier, es simplemente una forma de otorgarle un nombre a un objeto. Mediante el nombre, podemos acceder al objeto. Vamos a ver un ejemplo:

```
1  my_var = 5
2
3  id(my_var) # 4561204416
4  id(5) # 4561204416
```

En este caso, el identifier my_var es simplemente una forma de acceder a un objeto en memoria (en este caso el espacio identificado por el número 4561204416). Es importante recordar que un name puede referirse a cualquier tipo de objeto (aún las funciones).

```
1  def echo(value):
2      return value
3
```

```
4  a = echo
5
6  a('Billy') # 3
```

Ahora que ya entendimos qué es un name podemos avanzar a los namespaces (espacios de nombres). Para ponerlo en palabras llanas, un namespace es simplemente un conjunto de names. En Python, te puedes imaginar que existe una relación que liga a los nombres definidos con sus respectivos objetos (como un diccionario). Pueden coexistir varios namespaces en un momento dado, pero se encuentran completamente aislados. Por ejemplo, existe un namespace específico que agrupa todas las variables globales (por eso puedes utilizar varias funciones sin tener que importar los módulos correspondientes) y cada vez que declaramos una módulo o una función, dicho módulo o función tiene asignado otro namespace.

A pesar de existir una multiplicidad de namespaces, no siempre tenemos acceso a todos ellos desde un punto específico en nuestro programa. Es aquí donde el concepto de scope (campo de aplicación) entra en juego.

Scope es la parte del programa en el que podemos tener acceso a un namespace sin necesidad de prefijos. En cualquier momento determinado, el programa tiene acceso a tres scopes:

- El scope dentro de una función (que tiene nombres locales)
- El scope del módulo (que tiene nombres globales)
- El scope raíz (que tiene los built-in names)

Cuando se solicita un objeto, Python busca primero el nombre en el scope local, luego en el global, y por último, en la raíz. Cuando anidamos una función dentro de otra función, su scope también queda anidado dentro del scope de la función padre.

```
1  def outer_function(some_local_name):
2      def inner_function(other_local_name):
3          # Tiene acceso a la built-in function print y al nombre local some_local_name
4          print(some_local_name)
5
6          # También tiene acceso a su scope local
7          print(other_local_name)
```

Para poder manipular una variable que se encuentra fuera del scope local podemos utilizar los keywords global y nonlocal.

```
1  some_var_in_other_scope = 10
2
3  def some_function():
4      global some_var_in_other_scope
5
6      Some_var_in_other_scope += 1
```

q. Introducción a Click

Click es un pequeño framework que nos permite crear aplicaciones de Línea de comandos. Tiene cuatro decoradores básicos:

- @click_group: Agrupa una serie de comandos
- @click_command: Acá definiremos todos los comandos de nuestra apliacion
- @click_argument: Son parámetros necesarios
- @click_option: Son parámetros opcionales

Click también realiza las conversiones de tipo por nosotros. Esta basado muy fuerte en decoradores.

- pip install click : Ver vídeo de instalación: https://www.youtube.com/watch?v=j2Hg56guD4A

r. Definición a la API pública

En esta clase definiremos la estructura de nuestro proyecto PlatziVentas, los comandos, la configuración en nuestro setup.py y la instalaremos en nuestro entorno virtual con pip.

- Ver recursos: https://github.com/ProfeAlbeiro/adso_logica/tree/main/appwebinv2_logica_programacion/appwebinv4_Python/proy02_platzi_practical_Python_course_CRUD/docs/material/curso_Python3-14-what-is-oop

mkidr : Crea carpetas
touch : Crea archivos
tree . : Ver estructura de carpetas y archivos
pip install virtualenv : Instalar el entorno virtual
Python -m venv venv : Crear el entorno virtual
.\venv\Scripts\activate : Activar el entorno virtual

- Si no funciona este comando hacer lo siguiente:

tar como adiministrador

entUser / Enter / s

Más información: https://www.cdmon.com/es/blog/la-ejecucion-de-scripts-esta-deshabilitada-en-este-sistema-te-contamos-como-actuar

deactivate : Desactivar el entorno virtual
alias avenv=.\venv\Scripts\activate : Crear un Alias (No funciona aún)
pip install --editable . : Instalar nuestra aplicación
which pv : Línea de Comandos
pv --help : Ayuda General
pv clients --help : Ayuda de la aplicación

s. Clients

Modelaremos a nuestros clientes y servicios usando lo aprendido en clases anteriores sobre programación orientada a objetos y clases.

@staticmethod nos permite declarar métodos estáticos en nuestra clase. Es un método que se puede ejecutar sin necesidad de una instancia de una clase. No hace falta que reciba self como parámetro.

t. Servicios: Lógica de negocio de nuestra aplicación

cat .clients.csv : Visualizar un archivo csv

u. Interface de create: Comunicación entre servicios y el cliente

No hay descripción

v. Actualización de cliente

No hay descripción

w. Interface de actualización

- Ver recursos: https://github.com/ProfeAlbeiro/adso_logica/tree/main/appwebinv2_logica_programacion/appwebinv4_Python/proy02_platzi_practical_Python_course_CRUD/docs/material/curso_Python3-15-inheritance-polymorphism

x. Manejo de errores y jerarquía de errores en Python

Python tiene una amplia jerarquía de errores que nos da posibilidades para definir errores en casos como donde no se pueda leer un archivo, dividir entre cero o si existen problemas en general en nuestro código Python. El problema con esto es que nuestro programa termina, es diferente a los errores de sintaxis donde nuestro programa nunca inicia.

Para ""aventar"" un error en Python utilizamos la palabra raise. Aunque Python nos ofrece muchos errores es buena práctica definir errores específicos de nuestra aplicación y usar los de Python para extenderlos.

Podemos generar nuestros propios errores creando una clase que extienda de BaseException. Si queremos evitar que termine nuestro programa cuando ocurra un error, debemos tener una estrategia. Debemos utilizar try / except cuando tenemos la posibilidad de que un pedazo de nuestro código falle

- try : Significa que se ejecuta este código. Si es posible, solo ponemos una sola línea de código ahí como buena práctica
- except : Es nuestro manejo del error, es lo que haremos si ocurre el error. Debemos ser específicos con el tipo de error que vamos a atrapar.
- else : Es código que se ejecuta cuando no ocurre ningún error.
- finally : Nos permite obtener un bloque de código que se va a ejecutar sin importar lo que pase.

y. Context managers

Los context managers son objetos de Python que proveen información contextual adicional al bloque de código. Esta información consiste en correr una función (o cualquier callable) cuando se inicia el contexto con el keyword with; al igual que correr otra función cuando el código dentro del bloque with concluye. Por ejemplo:

```
1  with open('some_file.txt') as f:
2      lines = f.readlines()
```

Si estás familiarizado con este patrón, sabes que llamar la función open de esta manera, garantiza que el archivo se cierre con posterioridad. Esto disminuye la cantidad de información que el programador debe manejar directamente y facilita la lectura del código. Existen dos formas de

implementar un context manager: con una clase o con un generador. Vamos a implementar la funcionalidad anterior para ilustrar el punto:

```
1  class CustomOpen(object):
2      def __init__(self, filename):
3          self.file = open(filename)
4
5      def __enter__(self):
6          return self.file
7
8      def __exit__(self, ctx_type, ctx_value, ctx_traceback):
9          self.file.close()
10
11 with CustomOpen('file') as f:
12     contents = f.read()
```

Esta es simplemente una clase de Python con dos métodos adicionales: enter y exit. Estos métodos son utilizados por el keyword with para determinar las acciones de inicialización, entrada y salida del contexto. El mismo código puede implementarse utilizando el módulo contextlib que forma parte de la librería estándar de Python.

```
1  from contextlib import contextmanager
2
3  @contextmanager
4  def custom_open(filename):
5      f = open(filename)
6      try:
7          yield f
8      finally:
9          f.close()
10
11 with custom_open('file') as f:
12     contents = f.read()
```

El código anterior funciona exactamente igual que cuando lo escribimos con una clase. La diferencia es que el código se ejecuta al inicializarse el contexto y retorna el control cuando el keyword

yield regresa un valor. Una vez que termina el bloque with, el context manager toma de nueva cuenta el control y ejecuta el código de limpieza.

z. Aplicaciones de Python en el mundo real

Python tiene muchas aplicaciones. En las ciencias tiene muchas librerías que puedes utilizar como analisis de las estrellas y astrofisica; si te interesa la medicina puedes utilizar Tomopy para analizar tomografías. También están las librerías más fuertes para la ciencia de datos numpy, Pandas y Matplotlib.

En CLI por si te gusta trabajar en la nube y con datacenters, para sincronizar miles de computadoras:

- aws
- gcloud
- rebound
- geeknote

Aplicaciones Web:

- Django
- Flask
- Bottle
- Chalice
- Webapp2
- Gunicorn
- Tornado

8. PROGRAMACIÓN FUNCIONAL

9. FRAMEWORKS

9.1. Qué es un FrameWork

Marco de trabajo que proporciona una estructura, conjunto de herramientas, librerías, entre otras; que facilitan el proceso de trabajo en diseño y desarrollo de un software de forma ágil y reutilizable.

9.2. Proyecto Inventario con FrameWorks

10. FRAMEWORKS DE PYTHON

10.1. Django

10.1.1. Qué es Django

Django es un framework gratuito de código abierto escrito en Python. Es un software que puede utilizar para desarrollar aplicaciones web de forma rápida y eficiente. La mayoría de las aplicaciones web tienen varias funciones comunes, como la autenticación, la recuperación de información de una base de datos y la administración de cookies. Los desarrolladores tienen que codificar una funcionalidad similar en cada aplicación web que escriban. Django facilita su trabajo al agrupar las diferentes funciones en una gran colección de módulos reutilizables, llamada marco de aplicación web. Los desarrolladores utilizan el marco web de Django para organizar y escribir su código de manera más eficiente y reducir significativamente el tiempo de desarrollo web.

- ¿Qué es Django?
- Curso Django - OpenBootcamp

10.1.2. Instalación Django

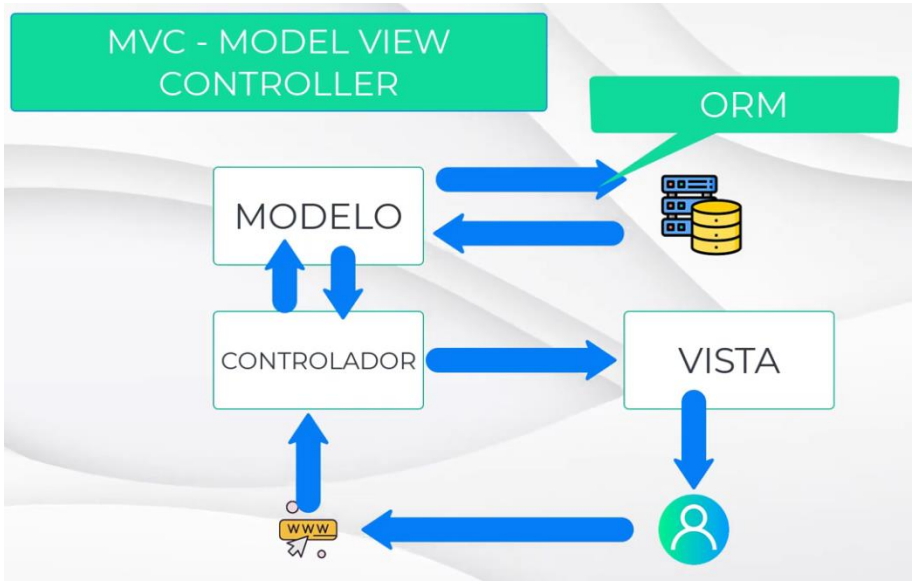
- Instalar Visual Studio Code: https://code.visualstudio.com/download
- Instalar extensiones de Visual Studio Code: https://kinsta.com/es/blog/extensiones-vscode/
- Instalar Python: https://www.python.org/downloads/. **Nota**: No olvidar marcar ☑ Add … / Instalar (Variables de Entorno)
- Verificar que las variables de entorno estén escritas correctamente. Ir a: Windows / Variables de entorno del sistema / Variables de Entorno / PATH. **Nota**: Si no existe: Python3XX\ y/o Python3XX\Scripts, hay que agregarlas manualmente.
- Uso del Terminal:

Terminal / New Terminal (ctrl + shift + ñ)	Abrir Terminal de Visual Studio Code
py --version ó python version	Muestra la versión de Python
clear ó cls	Limpia la terminal
pip install Django==5.0.1	Instala django: https://www.djangoproject.com/download/
pip freeze	Verifica los paquetes instalados
django-admin startproject dj02_mysite	Prepara entorno de trabajo
cd dj02_mysite	Ir al proyecto
ls	Revisar el interior de la carpeta
python manage.py runserver	Abrir el servidor de pruebas.

10.1.3. Patrón de Arquitectura MTV (Model Template View)

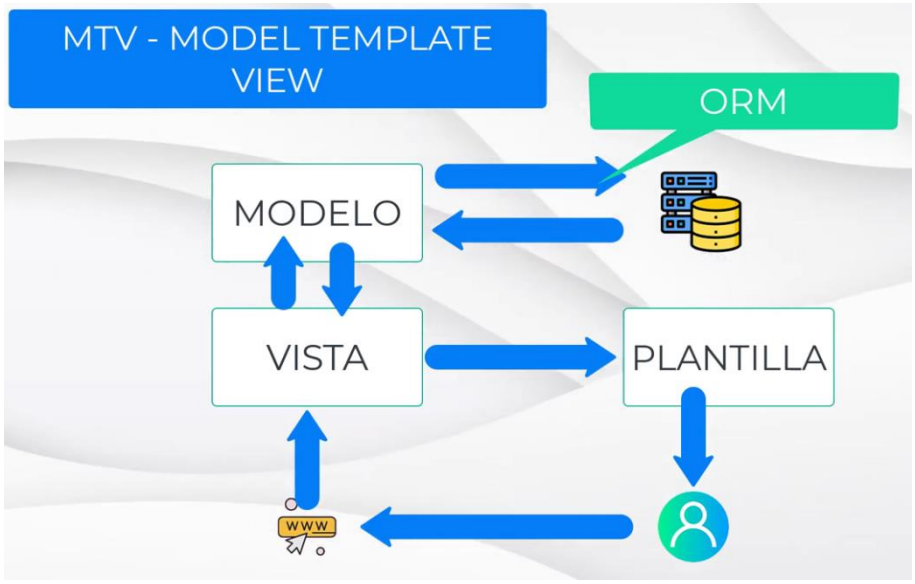
MTV: Model Template View (Modelo, Plantilla y Vista), es una variante del MVC (Modelo, Vista y Controlador).

Imagen 01. Modelo Vista Controlador



***NOTA**: Imagen tomada de*
https://www.youtube.com/watch?v=uhCGBzZR5g8&list=PLkVpKYNT_U9cl3hhVg_ROOlSY33uuBWZh&index=3

Imagen 02. Modelo Plantilla Vista



***NOTA**: Imagen tomada de*
https://www.youtube.com/watch?v=uhCGBzZR5g8&list=PLkVpKYNT_U9cl3hhVg_ROOlSY33uuBWZh&index=3

10.1.4. Estructura de archivos

Imagen 03. Estructura de archivos y carpetas

- El **mysite/**, directorio raíz externo: Es un contenedor para su proyecto. Su nombre no le importa a Django; puede cambiarle el nombre al que quiera.
- **manage.py**: Es una utilidad de línea de comandos que permite interactuar con este proyecto Django de varias maneras. Puede leer todos los detalles manage.py en django-admin y enable.py .
- El **mysite**/, directorio interno: Es el paquete Python real para su proyecto. Su nombre es el nombre del paquete Python que necesitará usar para importar cualquier cosa que contenga (p. ej. mysite.urls).
- **mysite/__init__.py**: Es un archivo vacío que le dice a Python que este directorio debe considerarse un paquete de Python. Si es un principiante de Python, lea más sobre los paquetes en los documentos oficiales de Python.
- **mysite/settings.py:** Ajustes/configuración para este proyecto Django. La configuración de Django le informará todo sobre cómo funcionan las configuraciones.
- **mysite/urls.py:** Las declaraciones de URL para este proyecto Django; una "tabla de contenidos" de su sitio con tecnología Django. Puede leer más sobre las URL en el despachador de URL.
- **mysite/asgi.py:** Un punto de entrada para que los servidores web compatibles con ASGI sirvan a su proyecto. Consulte cómo implementar con ASGI par obtener más detalles**.**
- **mysite/wsgi.py:** Un punto de entrada para que los servidores web compatibles con WSGI sirvan a su proyecto. Consulte Cómo implementar con WSGI para obtener más detalles.

➢ Escribir tu primera aplicación Django: https://docs.djangoproject.com/en/5.0/intro/tutorial01/

10.1.5. Hola Mundo

django-admin startproject dj05_hello_world	Prepara entorno de trabajo
cd dj05_hello_world	Ir al proyecto
python manage.py migrate	Crear la base de datos 'db.sqlite3'.
cd dj05_hello_world	Ir a la carpeta 'dj05_hello_world'.
touch views.py	Crear el archivo 'views.py' para gestionar las vistas.
	Modificar el archivo 'urls.py' para llamar la vista.
cd ..	Retornar a la raíz del proyecto
python manage.py runserver	Abrir el servidor de pruebas.
	Abrir la ruta del servidor en el navegador: http://127.0.0.1:8000/saludo/ y http://127.0.0.1:8000/despedida/
ctrl + c	Interrumpir el servidor

10.1.6. Rutas con parámetros

django-admin startproject dj06_routes_parameters	Prepara entorno de trabajo
cd dj06_routes_parameters	Ir al proyecto
python manage.py migrate	Crear la base de datos 'db.sqlite3'.
	Copiar y pegar los archivos del punto anterior. Modificar el archivo 'urls.py'. Modificar el archivo 'views.py'.
python manage.py runserver	Abrir el servidor de pruebas.
	Abrir la ruta del servidor en el navegador: http://127.0.0.1:8000/adulto/14/. El número (14) representa la edad
ctrl + c	Interrumpir el servidor

10.1.7. Uso de plantillas

django-admin startproject dj07_use_templates	Prepara entorno de trabajo
cd dj07_use_templates	Ir al proyecto
python manage.py migrate	Crear la base de datos 'db.sqlite3'.
mkdir templates	Crear la carpeta 'templates' en la raíz del proyecto.
cd templates	Ir a la carpeta 'templates'
touch simple.html	Crear el archivo 'simple.html'
	Copiar y pegar los archivos del punto anterior.

	Modificar el archivo 'settings.py'. Línea 57 Modificar el archivo 'urls.py'. Modificar el archivo 'views.py'.
cd ..	Retornar a la raíz del proyecto
python manage.py runserver	Abrir el servidor de pruebas.
	Abrir la ruta del servidor en el navegador: http://127.0.0.1:8000/simple/
ctrl + c	Interrumpir el servidor

10.1.8. Uso de contextos

django-admin startproject dj08_use_contexts	Prepara entorno de trabajo
cd dj08_use_contexts	Ir al proyecto
python manage.py migrate	Crear la base de datos 'db.sqlite3'.
	Copiar y pegar la carpeta 'templates' y los archivos 'urls.py' y 'views.py'. Modificar el archivo 'settings.py'. Línea 57 Modificar el archivo 'urls.py'. Modificar el archivo 'views.py'.
cd templates	Ir a la carpeta 'templates'.
touch dinamico.html	Crear el archivo 'dinamico.html'..
cd ..	Retornar a la raíz del proyecto
python manage.py runserver	Abrir el servidor de pruebas.
	Abrir la ruta del servidor en el navegador: http://127.0.0.1:8000/dinamico/Albeiro/
ctrl + c	Interrumpir el servidor

10.1.9. Bucles y condicionales en plantillas

django-admin startproject dj09_loops_conditionals	Prepara entorno de trabajo
cd dj09_loops_conditionals	Ir al proyecto
python manage.py migrate	Crear la base de datos 'db.sqlite3'.
	Copiar y pegar la carpeta 'templates' y los archivos 'urls.py' y 'views.py'. Modificar el archivo 'settings.py'. Línea 57 Modificar el archivo 'dinamico.html'.
python manage.py runserver	Abrir el servidor de pruebas.
	Abrir la ruta del servidor en el navegador:

	http://127.0.0.1:8000/dinamico/Albeiro/
ctrl + c	Interrumpir el servidor

10.1.10. Comentarios y filtros

django-admin startproject dj10_comments_filters	Prepara entorno de trabajo
cd dj10_comments_filters	Ir al proyecto
python manage.py migrate	Crear la base de datos 'db.sqlite3'.
	Copiar y pegar la carpeta 'templates' y los archivos 'urls.py' y 'views.py'. Modificar el archivo 'settings.py'. Línea 57 Modificar el archivo 'dinamico.html'.
python manage.py runserver	Abrir el servidor de pruebas.
	Abrir la ruta del servidor en el navegador: http://127.0.0.1:8000/dinamico/Albeiro/
ctrl + c	Interrumpir el servidor

10.1.11. Archivos Estáticos

django-admin startproject dj11_static_files	Prepara entorno de trabajo
cd dj11_static_files	Ir al proyecto
python manage.py migrate	Crear la base de datos 'db.sqlite3'.
	Modificar el archivo 'settings.py'. Línea 57, 118-123,
	Crear y/o modificar los siguientes archivos: 'urls.py' y 'views.py' 'templates' / 'estaticos.html' 'assets' / 'css' / 'styles.css' 'assets' / 'img' / 'logo-django.png' 'assets' / 'js' / 'scripts.js'
python manage.py runserver	Abrir el servidor de pruebas.
	Abrir la ruta del servidor en el navegador: http://127.0.0.1:8000/estaticos/
ctrl + c	Interrumpir el servidor

10.1.12. Herencia de Plantillas

django-admin startproject dj12_template_inheritance	Prepara entorno de trabajo
cd dj12_template_inheritance	Ir al proyecto

python manage.py migrate	Crear la base de datos 'db.sqlite3'.
	Modificar el archivo 'settings.py'. Línea 57, 118-123,
	Crear y/o modificar los siguientes archivos: 'urls.py' y 'views.py' 'templates' / 'layouts' / 'base.html' 'templates' / 'herencia.html' 'templates' / 'ejemplo.html' 'templates' / 'otra.html' 'assets' / 'css' / 'styles.css' 'assets' / 'css' / 'styles_pages.css' 'assets' / 'img' / 'logo-django.png' 'assets' / 'js' / 'scripts.js' 'assets' / 'js' / 'scripts_pages.js'
python manage.py runserver	Abrir el servidor de pruebas.
	Abrir la ruta del servidor en el navegador: http://127.0.0.1:8000/herencia/
ctrl + c	Interrumpir el servidor

10.1.13. Enlaces e inclusión

django-admin startproject dj13_links_inclusion	Prepara entorno de trabajo
cd dj13_links_inclusion	Ir al proyecto
python manage.py migrate	Crear la base de datos 'db.sqlite3'.
	Modificar el archivo 'settings.py'. Línea 57, 118-123,
	Crear y/o modificar los siguientes archivos: 'urls.py' y 'views.py' 'templates' / 'layouts' / 'base.html' 'templates' / 'layouts' / 'partials' / 'menu.html' 'templates' / 'layouts' / 'partials' / 'footer.html' 'templates' / 'index.html' 'templates' / 'herencia.html' 'templates' / 'ejemplo.html' 'templates' / 'otra.html' 'assets' / 'css' / 'styles.css' 'assets' / 'css' / 'styles_pages.css' 'assets' / 'img' / 'logo-django.png' 'assets' / 'js' / 'scripts.js' 'assets' / 'js' / 'scripts_pages.js'
python manage.py runserver	Abrir el servidor de pruebas.

	Abrir la ruta del servidor en el navegador: http://127.0.0.1:8000/herencia/
ctrl + c	Interrumpir el servidor

10.1.14. Documentación oficial

- Documentación Django 5.0

10.1.15. Práctica plantillas

Ps	Descripción	Acción
01	Preparar entorno de trabajo (**terminal**).	django-admin startproject dj15_practical_templates
02	Ir al proyecto (**terminal**).	cd dj15_practical_templates
03	Crear la base de datos 'db.sqlite3' (**terminal**).	python manage.py migrate
04	Modificar el archivo 'settings.py'.	settings.py. Línea 57, 118-123
05	Modificar el archivo 'url.py'.	urls.py
06	Crear el archivo 'views.py' en el módulo principal.	views.py
07	Crear la carpeta 'templates' en la raíz y sus subcarpetas.	templates / layouts / partials
08	Crear la carpeta 'assets' en la raíz y sus subcarpetas, así como sus archivos base. Fuentes: https://www.dafont.com/es/ Imágenes: https://www.flaticon.es/	assets / css / styles.css assets / fonts / cafe.ttf assets / fonts / fresh.otf assets / img / logo.png assets / js / scripts.js
09	Crear el archivo 'base.html'.	templates / layouts / base.html
10	Crear los archivos de componentes 'menú.html' y 'footer.html'.	templates / layouts / partials / menu.html templates / layouts / partials / footer.html
11	Crear los archivos principales 'index.html' y 'portfolio.html'.	templates / index.html templates / portfolio.html
12	Abrir el servidor de pruebas (**terminal**).	python manage.py runserver
13	Abrir la ruta del servidor en el navegador.	http://127.0.0.1:8000/
14	Interrumpir el servidor (**terminal**).	ctrl + c

10.1.16. Modularización

Ps	Descripción	Acción
01	Preparar entorno de trabajo (**terminal**).	django-admin startproject dj16_modularization
02	Ir al proyecto (**terminal**).	cd dj16_modularization
03	Crear el módulo 'comments' (**terminal**).	python manage.py startapp comments
04	Modificar el archivo 'settings.py' para reconocer el nuevo módulo.	settings.py. Línea 40
05	Verificar que el módulo 'comments' quedó instalado (**terminal**).	python manage.py check comments

10.1.17. Modelos de datos

Ps	Descripción	Acción
01	Preparar entorno de trabajo (**terminal**).	django-admin startproject dj17_data_models
02	Ir al proyecto (**terminal**).	cd dj17_data_models
03	Crear el módulo 'comments' (**terminal**).	python manage.py startapp comments
04	Modificar el archivo 'settings.py' para reconocer el nuevo módulo.	settings.py. Línea 40
05	Verificar que el módulo 'comments' quedó instalado (**terminal**).	python manage.py check comments
06	Modificar el archivo 'models.py' del módulo 'comments'	models.py
07	Solicitar a Django que se hagan las migraciones (**terminal**).	python manage.py makemigrations
08	Crear la base de datos 'db.sqlite3' (**terminal**).	python manage.py migrate
	Nota: Cada vez que se haga un cambio al archivo 'models.py' del módulo, se deben realizar las migraciones y la actualización de la base de datos (**terminal**).	python manage.py makemigrations python manage.py migrate

10.1.18. Delegación de rutas

Ps	Descripción	Acción
01	Preparar entorno de trabajo (**terminal**).	django-admin startproject dj18_delegation_routes
02	Ir al proyecto (**terminal**).	cd dj18_delegation_routes
03	Crear el módulo 'comments' (**terminal**).	python manage.py startapp comments
04	Modificar el archivo 'settings.py' para reconocer el nuevo módulo.	settings.py. Línea 40
05	Verificar que el módulo 'comments' quedó instalado (**terminal**).	python manage.py check comments
06	Modificar el archivo 'models.py' del módulo 'comments'	models.py
07	Solicitar a Django que se hagan las migraciones (**terminal**).	python manage.py makemigrations
08	Crear la base de datos 'db.sqlite3' (**terminal**).	python manage.py migrate
	Nota: Cada vez que se haga un cambio al archivo 'models.py' del módulo, se deben realizar las migraciones y la actualización de la base de datos (**terminal**).	python manage.py makemigrations [y/N] y python manage.py migrate
09	Crear el archivo 'urls.py' del módulo 'comments'	urls.py
10	Modificar el archivo 'views.py' del módulo 'comments'	views.py
11	Modificar el archivo 'url.py' del módulo 'dj18_delegation_routes'	urls.py
12	Abrir el servidor de pruebas (**terminal**).	python manage.py runserver
13	Abrir la ruta del servidor en el navegador.	http://127.0.0.1:8000/comments/
14	Interrumpir el servidor (**terminal**).	ctrl + c

10.1.19. Creación y borrado de datos

Ps	Descripción	Acción
01	Preparar entorno de trabajo (**terminal**).	django-admin startproject dj19_create_delete_data
02	Ir al proyecto (**terminal**).	cd dj19_create_delete_data
03	Crear el módulo 'comments' (**terminal**).	python manage.py startapp comments
04	Modificar el archivo 'settings.py' para reconocer el nuevo módulo.	settings.py. Línea 40

05	Verificar que el módulo 'comments' quedó instalado (**terminal**).	python manage.py check comments
06	Modificar el archivo 'models.py' del módulo 'comments'	models.py
07	Solicitar a Django que se hagan las migraciones (**terminal**).	python manage.py makemigrations
08	Crear la base de datos 'db.sqlite3' (**terminal**).	python manage.py migrate
	Nota: Cada vez que se haga un cambio al archivo 'models.py' del módulo, se deben realizar las migraciones y la actualización de la base de datos (**terminal**).	python manage.py makemigrations [y/N] y python manage.py migrate
09	Crear el archivo 'urls.py' del módulo 'comments'	urls.py
10	Modificar el archivo 'views.py' del módulo 'comments'	views.py
11	Modificar el archivo 'url.py' del módulo 'dj19_create_delete_data'	urls.py
12	Abrir el servidor de pruebas (**terminal**).	python manage.py runserver
13	Abrir la ruta del servidor en el navegador.	http://127.0.0.1:8000/comments/
14	Interrumpir el servidor (**terminal**).	ctrl + c

10.1.20. Estructura y claves foráneas

Ps	Descripción	Acción
01	Preparar entorno de trabajo (**terminal**).	django-admin startproject dj20_structure_foreign_keys
02	Ir al proyecto (**terminal**).	cd dj20_structure_foreign_keys
03	Crear el módulo 'post' (**terminal**).	python manage.py startapp post
04	Modificar el archivo 'settings.py' para reconocer el nuevo módulo.	settings.py. Línea 40
05	Verificar que el módulo 'post' quedó instalado (**terminal**).	python manage.py check post
06	Modificar el archivo 'models.py' del módulo 'post'	models.py
07	Solicitar a Django que se hagan las migraciones (**terminal**).	python manage.py makemigrations
08	Crear la base de datos 'db.sqlite3' (**terminal**).	python manage.py migrate
	Nota: Cada vez que se haga un cambio al archivo 'models.py' del módulo, se deben realizar las migraciones y la actualización de la base de datos (**terminal**).	python manage.py makemigrations [y/N] y python manage.py migrate
09	Modificar el archivo 'url.py' del módulo 'dj20_structure_foreign_keys'	urls.py
10	Crear el archivo 'urls.py' del módulo 'post'	urls.py

10.1.21. Seeding y paquetes

Ps	Descripción	Acción
01	Preparar entorno de trabajo (**terminal**).	django-admin startproject dj21_package_embedding
02	Ir al proyecto (**terminal**).	cd dj21_package_embedding
03	Crear el módulo 'post' (**terminal**).	python manage.py startapp post
04	Modificar el archivo 'settings.py' para reconocer el nuevo módulo.	settings.py. Líneas 40, 41
05	Verificar que el módulo 'post' quedó instalado (**terminal**).	python manage.py check post
06	Modificar el archivo 'models.py' del módulo 'post'	models.py
07	Solicitar a Django que se hagan las migraciones (**terminal**).	python manage.py makemigrations
08	Crear la base de datos 'db.sqlite3' (**terminal**).	python manage.py migrate

	Nota: Cada vez que se haga un cambio al archivo 'models.py' del módulo, se deben realizar las migraciones y la actualización de la base de datos (**terminal**).	python manage.py makemigrations [y/N] y python manage.py migrate
09	Modificar el archivo 'url.py' del módulo 'dj21_package_embedding'	urls.py
10	Crear el archivo 'urls.py' del módulo 'post'	urls.py
11	Consultar en internet 'django-seed'	https://github.com/Brobin/django-seed
12	Instalar paquete 'pip install django-seed' ó 'pip install -m django-seed', si hay una versión anterior en Django (**terminal**).	pip install django-seed
13	["Revisar paso **04**"]. Verificar que el módulo 'django_seed' quedó instalado (**terminal**).	python manage.py check django_seed
14	Corregir error en la instalación de 'django_seed'	pip install psycopg2-binary
15	Insertar datos masivos a la base de datos	python manage.py seed post --number=50

10.1.22. Consulta de datos I

Ps	Descripción	Acción
01	Preparar entorno de trabajo (**terminal**).	django-admin startproject dj22_data_query_partone
02	Ir al proyecto (**terminal**).	cd dj22_data_query_partone
03	Crear el módulo 'post' (**terminal**).	python manage.py startapp post
04	Modificar el archivo 'settings.py' para reconocer el nuevo módulo.	settings.py. Líneas 40, 41, 59
05	Verificar que los módulos 'post' y 'django_seed' quedaron instalados (**terminal**).	python manage.py check post python manage.py check django_seed
06	Modificar el archivo 'models.py' del módulo 'post'	models.py
07	Solicitar a Django que se hagan las migraciones (**terminal**).	python manage.py makemigrations
08	Crear la base de datos 'db.sqlite3' (**terminal**).	python manage.py migrate
	Nota: Cada vez que se haga un cambio al archivo 'models.py' del módulo, se deben realizar las migraciones y la actualización de la base de datos (**terminal**).	python manage.py makemigrations [y/N] y python manage.py migrate
09	Modificar el archivo 'url.py' del módulo 'dj22_data_query_partone'	urls.py
10	Crear el archivo 'urls.py' del módulo 'post'	urls.py
11	Consultar en internet 'django-seed'	https://github.com/Brobin/django-seed
12	Instalar paquete 'pip install django-seed' ó 'pip install -m django-seed', si hay una versión anterior en Django (**terminal**).	pip install django-seed
13	Corregir error en la instalación de 'django_seed'	pip install psycopg2-binary
14	Insertar datos masivos a la base de datos	python manage.py seed post --number=50
15	Modificar el archivo 'views.py' del módulo 'post'	views.py
16	Crear la carpeta 'templates' en la raíz y sus subcarpetas.	templates / post
	Crear el archivo 'queries.html'.	templates / post / queries.html
21	Abrir el servidor de pruebas (**terminal**).	python manage.py runserver
22	Abrir la ruta del servidor en el navegador.	http://127.0.0.1:8000/post/queries/
23	Interrumpir el servidor (**terminal**).	ctrl + c

1.1. Flask

1.1.1. Instalación

1.1.1.1. xxx

1.1. Entorno virtual en Python y su importancia

Paquetes de terceros:

- PyPi (Python package index) es un repositorio de paquetes de terceros que se pueden utilizar en proyectos de Python.
- Para instalar un paquete, es necesario utilizar la herramienta pip.
- La forma de instalar un paquete es ejecutando el comando pip install paquete.
- También se puede agrupar la instalación de varios paquetes a la vez con el archivo requirements.txt
- Es una buena práctica crear un ambiente virtual por cada proyecto de Python en el que se trabaje.

Ambientes virtuales:

- Buscar en Google: pip intallation / Clic a la versión más reciente / clic a get-pip.py para ver el script / descargar el script al proyecto / ejecutarlo
- Esto evita conflictos de paquetes en el intérprete principal.
- pip install virtualenv
- virtualenv venv
- .\venv\Scripts\activate
- pip freeze
- pip install flask o pip insall django
- touch requirements.txt
- ls
- requirements.txt / Flask==3.0.0
- pip insatall -r requirements.txt
- deactivate

2. FRAMEWORKS DE JAVASCRIPT