Manual Programador InfoVerdeHuellas

Presentado por.
Richard Gómez
Cristian Ramírez
Jhon Rodríguez

Índice.

Introducción	3
1. Lenguaje de Programación.	3
1.1 Python	3
1.2 Versiones Python	4
2 estructura de la aplicación	6
2.1 Patrón de arquitectura de software	6
3 Buenas Prácticas	7
3.1 Definición de variables	7
3.2 Definición de Métodos	7
3.2.1 Funciones con parámetros de Entrada	7
3.3 Condicionales	8
3.4 Identacion del código	8
4 Aplicación	9
4.1 Variables y Estructuras de Datos	9
4.1.1 main.py	9
4.2 Contratos de Función	10
4.2.1 main.py	10
4.2.3 models.py	12
4.3 Descripción de las Clases	12
4.3.1 config.py	12
4.3.2 horarios.py	13
4.4 Ubicación de las clases	13
5 Framework	13
5.1 FLASK	13
5.2 Esquema de rutas en la aplicación	14
6. ORM	16
6.1 SQLalchemy	16
7 Protocolos	16
7.1 Protocolo HTTP	16
Bibliografía	17

Introducción

El propósito de este manual del programador es dar a conocer al desarrollador todos los artefactos del programa realizado. se va elaborar un repaso general a toda la estructura que compone la aplicación de forma que el usuario se familiarice con ella y sobre todo que sea capaz de orientarse a la hora de realizar modificaciones o ampliaciones en la herramienta. Para ello se tratará de forma amena y concisa un repaso de todas las Unidades, ficheros Include, ejecutables..., con el fin de que el usuario pueda modificar a su gusto algunos de los valores y parámetros de las funciones expuestas.

1. Lenguaje de programación

1.1 Python

Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

Python es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

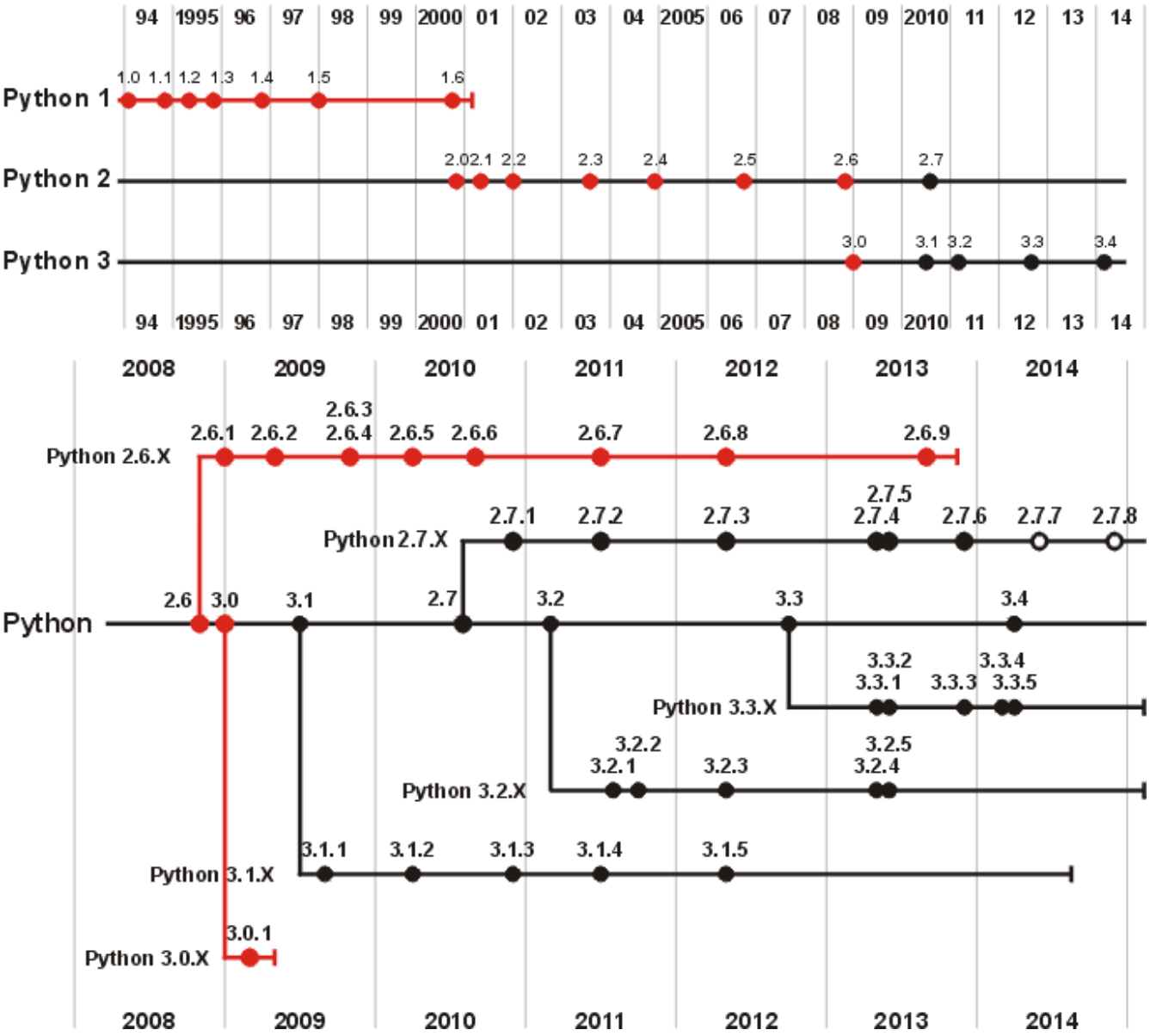
El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y de código fuente para las principales plataformas desde el sitio web de Python, https://www.python.org/, y puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, y documentación adicional.

El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades y tipos de datos implementados en C o C++ (u otros lenguajes accesibles desde C). Python también puede usarse como un lenguaje de extensiones para aplicaciones personalizables.

1.2 Versiones de Python.

Comienzo de la implementación - December, 1989
Publicación interna en CWI - 1990
Python 0.9 - 20 de febrero de 1991 (publicado en alt.sources)
Python 0.9.1 - Febrero de 1991
Python 0.9.2 - Otoño de 1991
Python 0.9.4 - 24 de diciembre de 1991
Python 0.9.5 - 2 de enero de 1992 (solo para Macintosh)
Python 0.9.6 - 6 de abril de 1992
Python 0.9.7 beta - 1992
Python 0.9.8 - 9 de enero de 1993
Python 0.9.9 - 29 de julio de 1993
Python 1.0 - Enero de 1994
Python 1.5 - 31 de diciembre de 1997
Python 1.6 - 5 de septiembre de 2000
Python 2.0 - 16 de octubre de 2000
Python 2.1 - 17 de abril de 2001
Python 2.2 - 21 de diciembre de 2001
Python 2.3 - 29 de julio de 2003
Python 2.4 - 30 de noviembre de 2004
Python 2.5 - 19 de septiembre de 2006
Python 2.6 - 1 de octubre de 2008
Python 2.7 - 3 de julio de 2010
Python 3.0 - 3 de diciembre de 2008
Python 3.1 - 27 de junio de 2009
Python 3.2 - 20 de febrero de 2011
Python 3.3 - 29 de septiembre de 2012
Python 3.4 - 16 de marzo de 2014

En la figura se muestra los tipos de versiones de Python y el respectivo año en que fue lanzada.

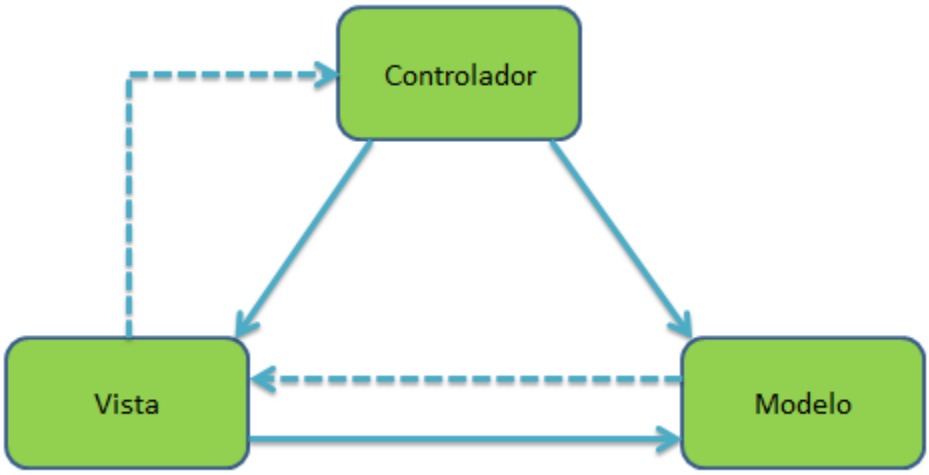


Actualmente Python se encuentra en su versión 3.6, sin embargo en la aplicación informática verde se utilizo la versión 3.5.

2. Estructura de la Aplicación

2.1 Patrón de arquitectura de software

Modelo Vista Controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.



El Modelo que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.

La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.

El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

El Modelo es responsable de.

Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
Define las reglas de negocio (la funcionalidad del sistema).
Lleva un registro de las vistas y controladores del sistema.

El Controlador es responsable de.
Recibe los eventos de entrada (un clic, un cambio en un campo de texto, etc.).
Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al método "Actualizar()". Una petición al modelo puede ser "Obtener el COe de una sala".

La Vista es Responsable.
Recibir datos del modelo y los muestra al usuario.
Tienen un registro de su controlador asociado (normalmente porque además lo instancia).
Pueden dar el servicio de "Actualización()", para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes).

3 Buenas prácticas.

3.1 Definición de Variables.

Al ser Python un lenguaje de programación de tipado dinámico, una misma variable puede tomar valores de distinto tipo en distintos momentos. Por lo cual no es necesario definir el tipo de dato de la variable a definir ejemplo.

A=1
a=”a”

Mala práctica ej. String a=”a” Int a=1

3.2 Definición de Métodos.

En Python, la definición de funciones se realiza mediante la instrucción def más un nombre de función descriptivo -para el cuál, aplican las mismas reglas que para el nombre de las variables- seguido de paréntesis de apertura y cierre. Como toda estructura de control en Python, la definición de la función finaliza con dos puntos (:) y el algoritmo que la compone, irá identado con 4 espacios:

```
def mi_funcion():
    # aquí el algoritmo
```

3.2.1 Funciones con parámetros de entrada.

Un parámetro es un valor que la función espera recibir cuando sea llamada (invocada), a fin de ejecutar acciones en base al mismo. Una función puede esperar uno o más parámetros (que irán separados por una coma) o ninguno.

```
def mi_funcion(nombre, apellido):
```

algoritmo

Los parámetros, se indican entre los paréntesis, a modo de variables, a fin de poder utilizarlos como tales, dentro de la misma función.

3.3 Condicionales.

- If : La estructura de control if ... permite que un programa ejecute unas instrucciones cuando se cumplan una condición.
  - if condición:
  - aquí van las órdenes que se ejecutan si la condición es cierta y que pueden ocupar varias líneas.
  - Las ordenes van con identacion de cuatro espacios.

- Ejemplo.
  - numero = int(input("Escriba un número positivo: "))
  - if numero < 0:
  - print("¡Le he dicho que escriba un número positivo!")
  - print(f"Ha escrito el número {numero}")

- If .. else :La estructura de control if ... else ... permite que un programa ejecute unas instrucciones cuando se cumple una condición y otras instrucciones cuando no se cumple esa condición.
  - if condición:
  - aquí van las órdenes que se ejecutan si la condición es cierta y que pueden ocupar varias líneas
  - else:
  - y aquí van las órdenes que se ejecutan si la condición es falsa y que también pueden ocupar varias líneas

- Ejemplo.
  - edad = int(input("¿Cuántos años tiene? "))
  - if edad < 18:
  - print("Es usted menor de edad")
  - else:
  - print("Es usted mayor de edad")
  - print("¡Hasta la próxima!")

3.4 Identacion del Código.

Indentación (también denominado espaciado, tabulación, sangría).Indentación es un anglicismo (de la palabra inglesa indentation) de uso común en informática, y que significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores, para así separarlo del margen izquierdo y mejor distinguirlo del texto adyacente; en el ámbito de la imprenta, este concepto siempre se ha denominado sangrado o sangría.

En los lenguajes de programación de computadoras, la indentación es un tipo de notación secundaria utilizado para mejorar la legibilidad del código fuente por parte de los programadores, teniendo en cuenta que los compiladores o intérpretes raramente consideran los espacios en blanco entre las sentencias de un programa. Sin embargo, en ciertos lenguajes de programación como Haskell, Occam y Python, la indentación se utiliza para delimitar la estructura del programa permitiendo establecer bloques de código.

Ejemplo.

```
def fib(n):
        print 'n =', n
        if n > 1:
                return n * fib(n - 1)
        else:
                print 'fin de la línea'
        return 1
print (“Aquí termina”)
```

Vemos que la función fib contiene todas las instrucciones debajo de ella gracias a que dichas instrucciones están sangradas o indentadas hacia dentro, por ejemplo con 4 espacios, por tanto todo lo que está indentado debajo de la instrucción del fib, es el cuerpo de la función. Dentro de esta función tenemos la instrucción if y else, dentro de cada una de ellas tenemos instrucciones indentadas , lo cual nos señala el cuerpo de instrucciones que pertenecen a if y a else. En cambio, la instrucción print (“Aquí termina”) no está indentada, es una instrucción sola y no está dentro de la función def fib(n).

4 Aplicación.

La aplicación infoVerdeHuellas contiene los siguientes características.

4.1 Variables e Instancias.

Se definen el nombre de las variables y el tipo de estructura de datos utilizada por cada una.

4.1.1 Main.py

listPotencia = list() #estrutura de datos tipo"lista" para almacenar la potencia calculada

listPotenciaFanta = list() #estrutura de datos tipo"lista" para almacenar la potencia fantasma calculada

4.1.2 Models.py

db = SQLAlchemy() #Se instancia el ORM

4.1.3 Horarios.py

class Horario:
salas = {} #Se crea un diccionario de sala con la URL de el calendario de GOOGLE.

4.2 Contratos de Función.

4.2.1 Main.py

Método Main.

```
if __name__ == '__main__':
    db.init_app(app)            #iniciamos la base de datos
    with app.app_context():            # Revisa si las tablas de la BD están creadas.
        db.create_all()            # crea todas las tablas que no estén creadas
    app.run(debug=True, port=5000)        #corremos ejecuta el servidor en el puerto 5000
```

def Responsable(Id=0):

- Función **Responsable** con parámetro de entrada Id=0 que apunta al primer registro que hay en la tabla responsable.

Propósito.

- El propósito de la función **Responsable** es la de realizar el CRUD, crear, leer, actualizar o eliminar los parámetros de la tabla responsable, para ello contiene 4 condicionales con su respectivo método ya sea GET, POST, PUT o DELETE.

Return.

- La función no retorna ningún parámetro.

def Machine(Id=0):

- Función **Machine** con parámetro de entrada Id=0 que apunta al primer registro que hay en la tabla maquina.

Propósito.

- El propósito de la función **Machine** es la de realizar el CRUD, crear, leer, actualizar o eliminar los parámetros de la tabla máquina, para ello contiene 4 condicionales con su respectivo método ya sea GET, POST, PUT o DELETE.

Return.

- La función no retorna ningún parámetro.

def Cargo(Id=0):

- Función **Cargo** con parámetro de entrada Id=0 que apunta al primer registro que hay en la tabla cargo.

Propósito.

- El propósito de la función **Cargo** es la de realizar el CRUD, crear, leer, actualizar o eliminar los parámetros de la tabla cargo, para ello contiene 4 condicionales con su respectivo método ya sea GET, POST, PUT o DELETE.

Return.

- La función no retorna ningún parámetro.

def Office(Id=0):

- Función **Office** con parámetro de entrada Id=0 que apunta al primer registro que hay en la tabla oficina.

Propósito.

- El propósito de la función **Office** es la de realizar el CRUD, crear, leer, actualizar o eliminar los parámetros de la tabla oficina, para ello contiene 4 condicionales con su respectivo método ya sea GET, POST, PUT o DELETE.

Return.

- La función no retorna ningún parámetro.

def Muestra(Id=0):

- Función **Muestra** con parámetro de entrada Id=0 que apunta al primer registro que hay en la tabla muestra.

Propósito.

- El propósito de la función **Muestra** es la de realizar el CRUD, crear, leer, actualizar o eliminar los parámetros de la tabla muestra, para ello contiene 4 condicionales con su respectivo método ya sea GET, POST, PUT o DELETE.

Return.

- La función no retorna ningún parámetro.

def FactorEmision(Id=0):

- Función **FactorEmision** con parámetro de entrada Id=0 que apunta al primer registro que hay en la tabla factorEmision.

Propósito.

- El propósito de la función **FactorEmision** es la de realizar el CRUD, crear, leer, actualizar o eliminar los parámetros de la tabla factorEmision, para ello contiene 4 condicionales con su respectivo método ya sea GET, POST, PUT o DELETE.

Return.

- La función no retorna ningún parámetro.

def detailSample(Id=0):

- Función **detailSample** con parámetro de entrada Id=0 que apunta al primer registro que hay en la tabla detalleSimple.

Propósito.

- El propósito de la función **detailSample** es la de realizar el CRUD, crear, leer, actualizar o eliminar los parámetros de la tabla detalleSimple, para ello contiene 4 condicionales con su respectivo método ya sea GET, POST, PUT o DELETE.

Return.

- La función no retorna ningún parámetro.

def totalPotencia():

- Función **totalPotencia** sin parámetro de entrada.

Propósito.

- El propósito de la función **totalPotencia** es la de realizar la sumatoria de de la potencia real consumida en dicha sala en el respectivo año.

Return.

- La función retorna el total de la potencia real de la sala en Wats.

def totalPotenciaFan():

- Función **totalPotenciaFan** sin parámetro de entrada.

Propósito.

- El propósito de la función **totalPotenciaFan** es la de realizar la sumatoria de de la potencia fantasma consumida en dicha sala en el respectivo año.

Return.

- La función retorna el total de la potencia fantasma de la sala en Wats.

4.2.3 Models.py

class BasTipoMaquina(db.Model):

```
#crea la clase de la tabla tipo de maquina con sus diferentes atributos
tablename__='BAS_TIPO_MAQUINA' #Nombre de la tabla
    tip_cod = db.Column(db.Integer, primary_key=True) #Atributo 1
    tip_dscrpcion = db.Column(db.String(40), nullable=False)#Atributo 2
    tip_referencia = db.Column(db.String(20), nullable=False)#Atributo 3
    tip_Ptncia_Nmnal = db.Column(db.Float)#Atributo 4
    detailSample = db.relationship('TrnDetalleMuestra',#Atributo 5
backref='BasTipoMaquina_TrnDetalleMuestra', cascade="all, delete-orphan")#Relacion
(tabla tipo maquina con la tabla detalle muestra)
```

De este mismo modo se crean tantas clase como tablas en la BD, se crean las siguientes clases.

class BasOficiona(db.Model):
class TrnDetalleMuestra(db.Model):
class BasFactorEmision(db.Model):
class BasMuestra(db.Model):
class BasResponsable(db.Model):
class BasCargo(db.Model):
class BasUser(db.Model):

class AlchemyEncoder(json.JSONEncoder):#esta clase serializa los objetos de la clase **Models.py**

4.3 Descripcion de las Clases

4.3.1 Config.py

permite configurara la información de la base de datos tales como contraseña nombre de la BD, indica al oro que tipo de BD se va a utilizar, ya sea MySQL, postres.

4.3.2 Horarios.py

Permite extraer la información de la reserva de las salas de la uceva, esta información se encuentra en un calendario de Google la página principal de la uceva.

4.4 Ubicación de las Clases

En la carpeta principal de la aplicación se encuentran otras carpetas y la mayoría de las clases de Python como la siguiente descripción.

Static(Carpeta) → en esta carpeta se encuentra el controlador, imágenes y archivos CSS.
Template(Carpeta) → en esta carpeta se encuentra toda la vista de la aplicación como lo es los archivos HTML

Main.py
Models.py
Config.py
Horarios.py

5 Framework.

Para la aplicación infoVerdeHuellas se utilizó un framework llamado FLASK.

5.1 FLASK.

Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código.

Flask es un micro framework web basado en el kit de herramientas Werkzeug y el motor de plantillas Jinja2 . Tiene licencia BSD .

La última versión estable de Flask es 1.0 a partir de abril de 2018. Las aplicaciones que usan el marco de Flask incluyen Pinterest , LinkedIn, y la página web de la comunidad para Flask.

Flask se llama un micro framework porque no requiere herramientas o bibliotecas particulares. No tiene una capa de abstracción de base de datos, validación de formularios ni ningún otro componente donde las bibliotecas de terceros preexistentes proporcionan funciones comunes. Sin embargo, Flask admite extensiones que pueden agregar características de la aplicación como si estuvieran implementadas en el Flask mismo. Existen extensiones para mapeadores relacionales de objetos, validación de formularios, manejo de cargas, varias tecnologías de autenticación abierta y varias herramientas relacionadas con marcos comunes. Las extensiones se actualizan con mucha más frecuencia que el programa central Flask.

5.2 Esquema de rutas en la aplicación.

@app.route("/")
Ruta index de la aplicación o ruta raiz. Esta ruta es la primera vista que el usurio tiene de la aplicación contiene la función index() la cual retorna la vista indexHuellas.html

```
def index():
    return render_template('indexHuellas.html')#return
```

@app.route("/maquina")
Ruta para la vista de el admin maquinas en la aplicación contiene la función machine la cual retorna machine.html

```
def machine():
    return render_template('machine.html')#return
```

@app.route("/oficina")
Ruta para la vista de el admin oficinas en la aplicación contiene la función office la cual retorna office.html

```
def office():
    return render_template('office.html')#return
```

@app.route("/detalleMuestra")
Ruta para la vista de el admin detalle Muestra en la aplicación contiene la función detalleMuestra la cual retorna detalleMuestra.html

```
def detalleMuestra():
    return render_template('detalleMuestra.html')#return
```

@app.route("/cargo")
Ruta para la vista de el admin cargo en la aplicación contiene la función cargo la cual retorna cargo.html

```
def cargo():
    return render_template('cargo.html')#return
```

@app.route("/responsable")
Ruta para la vista de el admin responsable en la aplicación contiene la función responsable la cual retorna responsable.html

```
def responsable():
    return render_template('responsable.html')#return

@app.route("/muestra")
```
Ruta para la vista de el admin muestra en la aplicación contiene la función muestra la cual retorna muestra.html
```
def muestra():
    return render_template('muestra.html')#return


@app.route("/factorEmision")
```
Ruta para la vista de el admin factorEmision en la aplicación contiene la función factorEmision la cual retorna factorEmision.html
```
def factorEmision():
    return render_template('factorEmision.html')#return

@app.errorhandler(404)
```
Ruta para cuando no existe la ruta especificada redirecciona a la index
```
def pageNotFound(e):
    return render_template('index.html')#retono hacia index

@app.route("/api/responsable",methods=['GET','POST','PUT'])#Ruta del controlador de la vista responsable con los metodos GET,POST,PUT.
@app.route("/api/responsable/<int:Id>", methods=['DELETE'])#Ruta del controlador de la vista responsable con el parámetro id para el método DELETE
def Responsable(Id=0):#id 0 apunta al primer registro de la tabla responsable
```
contiene el método **def Responsable(Id=0):** que es el encagado de hacer el CRUD en la tabla responsable, este método ya fue especificado anteriormente. **Vease 4.2.1** contratos de función de la clase main.py

A continuacion se listan otras rutas similares a la vista anteriormente.

```
@app.route("/api/machine",methods=['GET','POST','PUT'])
@app.route("/api/machine/<int:Id>", methods=['DELETE'])

@app.route("/api/position",methods=['GET','POST','PUT'])
@app.route("/api/position/<int:Id>", methods=['DELETE'])

@app.route("/api/office",methods=['GET','POST','PUT','PATCH'])
@app.route("/api/office/<int:Id>", methods=['DELETE'])

@app.route("/api/muestra",methods=['GET','POST','PUT'])
@app.route("/api/muestra/<int:Id>", methods=['DELETE'])

@app.route("/api/factorEmision",methods=['GET','POST','PUT'])
@app.route("/api/factorEmision/<int:Id>", methods=['DELETE'])

@app.route("/api/detailSample",methods=['GET','POST','PUT','PATCH'])
@app.route("/api/detailSample/<Id>", methods=['DELETE'])
```

6 ORM.

El mapeo objeto-relacional (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo)

6.1 sqlalchemy

SQLAlchemy es un conjunto de herramientas de código abierto SQL y mapeador relacional de objetos (ORM) para el lenguaje de programación Python.

SQLAlchemy proporciona "un conjunto completo de conocidos patrones de persistencia a nivel empresarial, diseñados para el acceso a bases de datos eficientes y de alto rendimiento, adaptados a un lenguaje de dominio simple y Pythonic". La filosofía de SQLAlchemy es que las bases de datos SQL se comportan cada vez menos como colecciones de objetos cuanto más tamaño y rendimiento comienzan a importar, mientras que las colecciones de objetos se comportan cada vez menos como tablas y filas a medida que la abstracción comienza a importar. Por esta razón, ha adoptado el patrón de mapeador de datos (como Hibernate para Java) en lugar del patrón de registro activo utilizado por un número de otros mapeadores relacionales de objetos. Sin embargo, los complementos opcionales permiten a los usuarios desarrollar usando la sintaxis declarativa.

7 Protocolos de Comunicación.

7.1 Protocolo HTTP.

se usa POST para crear un recurso en el servidor
se usa GET para obtener un recurso
se usa PUT para cambiar el estado de un recurso o actualizarlo
se usa DELETE para eliminar un recurso

Bibliografía.

http://www.upv.es/~csahuqui/julio/pfc/programa.pdf --→ Guia.

https://es.wikipedia.org/wiki/Python --→ Python

http://docs.python.org.ar/tutorial/3/reference.html#library-index → biblioteca de referencia

https://es.wikipedia.org/wiki/Historia_de_Python —> versiones de pyhon

https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html -→modelo vista controlador

http://librosweb.es/libro/python/capitulo_4/definiendo_funciones.html → definicion funciones en Python

http://www.mclibre.org/consultar/python/lecciones/python-if-else.html → condicionales en Python

https://es.scribd.com/document/250343889/Indentacion-en-python → identacion

http://flask.pocoo.org/docs/0.12/ → Documentacion FLASK.

https://en.wikipedia.org/wiki/Flask_(web_framework) → Que es FLASK?

https://es.wikipedia.org/wiki/Mapeo_objeto-relacional → ORM que es?

http://docs.sqlalchemy.org/en/latest/orm/ → Documentacion SQLalchemy

https://en.wikipedia.org/wiki/SQLAlchemy → SQLalchemy que es?

https://dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful → servicios web