Guía Completa para Principiantes: Sistema de Gestión de Contactos

Tabla de Contenidos

Conceptos Básicos que Necesitas Saber {#conceptos-básicos}

¿Qué es la Programación Orientada a Objetos (POO)?

Imagina que quieres hacer galletas:

- **Clase** = El molde de la galleta
- **Objeto** = Cada galleta individual que haces
- **Atributos** = Las características (sabor, tamaño, color)
- **Métodos** = Las acciones que puedes hacer (hornear, decorar, comer)

Ejemplo en nuestro proyecto:

python

```
# La CLASE es el molde
class Contacto:
    pass


# El OBJETO es cada contacto individual
contacto1 = Contacto("Juan", "123456", "juan@email.com", "Calle 1")
contacto2 = Contacto("María", "789012", "maria@email.com", "Calle 2")
```

¿Qué es una Lista?

Una lista es como una caja donde guardas cosas en orden:

python

```
# Lista de contactos
mis_contactos = [contacto1, contacto2, contacto3]

# Puedes acceder por posición
primer_contacto = mis_contactos[0]  # Juan

# Puedes agregar elementos
mis_contactos.append(nuevo_contacto)
```

¿Qué es un Diccionario?

Un diccionario es como una agenda donde cada cosa tiene una etiqueta:

python

```
# Diccionario de un contacto
contacto = {
    'nombre': 'Juan',      # etiqueta: valor
    'telefono': '123456',
    'email': 'juan@email.com'
}

# Accedes por la etiqueta
print(contacto['nombre'])  # Imprime: Juan
```

Configuración del Entorno {#configuración}

Paso 1: Instalar Python

1. Ve a python.org
2. Descarga la versión más reciente (3.7 o superior)
3. Durante la instalación, marca "Add Python to PATH"
4. Verifica la instalación abriendo terminal/cmd:

bash

```
python --version
```

Paso 2: Instalar Jupyter Notebook

Abre tu terminal/cmd y ejecuta:

bash

pip install notebook

Paso 3: Iniciar Jupyter

bash

jupyter notebook

Esto abrirá tu navegador web con Jupyter.

Paso 4: Crear un Nuevo Notebook

1. Click en "New" → "Python 3"
2. Nombra tu archivo: "Sistema_Gestion_Contactos"
3. ¡Listo para empezar!

Construcción del Sistema Paso a Paso {#construcción}

PASO 1: Entender la Clase Contacto

¿Qué hace? Representa un contacto individual (como una tarjeta de presentación).

Componentes:

python

```
class Contacto:
    # 1. Constructor: Se ejecuta al crear un contacto
    def __init__(self, nombre, telefono, email, direccion):
        self.nombre = nombre        # Guarda el nombre
        self.telefono = telefono    # Guarda el teléfono
        self.email = email          # Guarda el email
        self.direccion = direccion  # Guarda la dirección
```

Explicación:

- __init__ es el "nacimiento" del objeto
- self significa "este contacto específico"
- Cada self.nombre es como poner una etiqueta en el contacto

Pruébalo:

python

```
# Crear un contacto
mi_contacto = Contacto(
    nombre="Pedro Pérez",
    telefono="111-222-333",
    email="pedro@email.com",
    direccion="Calle Principal 123"
)

# Acceder a sus datos
print(mi_contacto.nombre)    # Imprime: Pedro Pérez
print(mi_contacto.telefono)  # Imprime: 111-222-333
```

PASO 2: Métodos de la Clase Contacto

Método `__str__` (Cómo se ve el contacto)

python

```
def __str__(self):
    return f"""
    Nombre: {self.nombre}
    Teléfono: {self.telefono}
    Email: {self.email}
    Dirección: {self.direccion}
    """
```

Uso:

python

```
print(mi_contacto)  # Imprime toda la información bonita
```

Método `to_dict` (Convertir a diccionario)

python

```
def to_dict(self):
    return {
        'nombre': self.nombre,
        'telefono': self.telefono,
        'email': self.email,
        'direccion': self.direccion
    }
```

Uso:

python

```
contacto_dict = mi_contacto.to_dict()
print(contacto_dict)  # {'nombre': 'Pedro Pérez', ...}
```

Método actualizar (Cambiar datos)

python

```
def actualizar(self, nombre=None, telefono=None, email=None, direccion=None):
    if nombre:
        self.nombre = nombre
    if telefono:
        self.telefono = telefono
    # ... y así con los demás
```

Uso:

```
python
mi_contacto.actualizar(telefono="999-888-777")
print(mi_contacto.telefono)  # Imprime: 999-888-777
```

PASO 3: Entender la Clase Agenda

¿Qué hace? Gestiona TODOS los contactos (como una agenda completa).

```
python
class Agenda:
    def __init__(self):
        self.contactos = []  # Lista vacía para guardar contactos
```

Explicación:

- La agenda tiene una lista de contactos
- Al inicio, la lista está vacía `[]`
- Iremos agregando contactos a esta lista

PASO 4: Métodos de la Agenda

1. Agregar Contacto

python

```
def agregar_contacto(self, nombre, telefono, email, direccion):
    # Validar que el nombre no esté vacío
    if not nombre.strip():
        print("Error: El nombre no puede estar vacío")
        return False

    # Crear el contacto
    nuevo_contacto = Contacto(nombre, telefono, email, direccion)

    # Agregarlo a la lista
    self.contactos.append(nuevo_contacto)

    print(f"Contacto '{nombre}' agregado exitosamente")
    return True
```

Uso:

python

```
agenda = Agenda()
agenda.agregar_contacto("Ana", "123", "ana@email.com", "Calle 1")
```

2. Buscar por Nombre

python

```
def buscar_por_nombre(self, nombre):
    resultados = []  # Lista para guardar resultados

    # Recorrer TODOS los contactos
    for contacto in self.contactos:
        # Si el nombre buscado está en el nombre del contacto
        if nombre.lower() in contacto.nombre.lower():
```

```
  resultados.append(contacto)  # Agregarlo a resultados
   return resultados
```

Explicación paso a paso:

1. Creamos una lista vacía `resultados`
2. Miramos cada contacto uno por uno (`for`)
3. Comparamos nombres (sin importar mayúsculas/minúsculas)
4. Si coincide, lo agregamos a resultados
5. Devolvemos todos los resultados encontrados

Uso:

```
python
encontrados = agenda.buscar_por_nombre("Ana")
if encontrados:
    for contacto in encontrados:
        print(contacto)
else:
    print("No se encontró ningún contacto")
```

3. Listar Todos

```
python
def listar_todos(self):
    if not self.contactos:  # Si la lista está vacía
        print("La agenda está vacía")
        return

    # Ordenar por nombre alfabéticamente
    contactos_ordenados = sorted(self.contactos,
                                 key=lambda x: x.nombre.lower())

  # Mostrar cada uno
    for i, contacto in enumerate(contactos_ordenados, 1):
        print(f"{i}. {contacto.nombre} - {contacto.telefono}")
```

Explicación de `sorted`:

- sorted() ordena la lista
- key=lambda x: x.nombre.lower() dice "ordena por nombre"
- lambda es una función pequeña (no te preocupes mucho por esto)

Cómo Ejecutar el Proyecto {#ejecución}

Método 1: En Jupyter (Recomendado para Principiantes)

Celda 1: Copiar la Clase Contacto

python

```
# Pega aquí el código de la Clase Contacto
class Contacto:
    # ... todo el código
```

Presiona: `Shift + Enter` para ejecutar

Celda 2: Copiar la Clase Agenda

python

```
# Pega aquí el código de la Clase Agenda
class Agenda:
    # ... todo el código
```

Presiona: `Shift + Enter`

Celda 3: Probar el Sistema

python

```
# Crear una agenda
mi_agenda = Agenda()


# Agregar contactos
mi_agenda.agregar_contacto("Juan", "111", "juan@email.com", "Calle 1")
mi_agenda.agregar_contacto("María", "222", "maria@email.com", "Calle 2")
```

```
# Listar todos
mi_agenda.listar_todos()


# Buscar
resultados = mi_agenda.buscar_por_nombre("Juan")
for contacto in resultados:
    print(contacto)
```

Presiona: `Shift + Enter`

Método 2: Ejecutar el Sistema Completo

python

```
# Después de ejecutar las celdas anteriores
ejecutar_sistema()
```

Esto iniciará el menú interactivo donde puedes:

- Agregar contactos
- Buscar contactos
- Editar contactos
- Eliminar contactos
- Ver estadísticas

Preguntas Frecuentes {#faq}

P: ¿Qué significa `self`?

R: `self` es como decir "este objeto específico".

Ejemplo:

python
contacto1.nombre = "Juan"   # El nombre de contacto1
contacto2.nombre = "María" # El nombre de contacto2

Cada objeto tiene su propio `self`.

P: ¿Por qué uso `__init__`?

R: `__init__` es el constructor, se ejecuta automáticamente cuando creas un objeto.

python
# Cuando haces esto:
contacto = Contacto("Juan", "123", "email", "dir")

Python automáticamente ejecuta __init__ con esos valores

P: ¿Qué es una lista vacía `[]`?

R: Es una lista sin elementos, como una caja vacía.

python
contactos = []   # Caja vacía
contactos.append(c1)   # Metes algo
contactos.append(c2)   # Metes otra cosa
print(len(contactos))   # Imprime: 2

P: ¿Qué significa `return`?

R: `return` devuelve un valor al lugar que llamó la función.

python
def sumar(a, b):

```
    return a + b

resultado = sumar(5, 3)  # resultado = 8
```

P: ¿Cómo funciona el `for`?

R: `for` repite algo para cada elemento de una lista.

```
python
frutas = ["manzana", "pera", "uva"]

for fruta in frutas:
    print(fruta)

# Imprime:
# manzana
# pera
# uva
```

P: ¿Qué es `.append()`?

R: Agrega un elemento al final de una lista.

```
python
numeros = [1, 2, 3]
numeros.append(4)
print(numeros)  # [1, 2, 3, 4]
```

Ejercicios Prácticos

Ejercicio 1: Crear tu primer contacto

python
```
# Tu turno: Crea un contacto con TU información
mi_contacto = Contacto(
    nombre="Tu Nombre",
    telefono="Tu Teléfono",
    email="Tu Email",
    direccion="Tu Dirección"
)

print(mi_contacto)
```

Ejercicio 2: Agregar varios contactos

python
```
agenda = Agenda()

# Agrega 3 contactos de tu familia o amigos
# ...

agenda.listar_todos()
```

Ejercicio 3: Buscar contactos

python
```
# Busca uno de los contactos que agregaste
resultados = agenda.buscar_por_nombre("...")

if resultados:
    for contacto in resultados:
        print(contacto)
```

Recursos Adicionales

- Python.org Tutorial Oficial
- Real Python - Guías para Principiantes
- W3Schools Python

Checklist de Aprendizaje

Entiendo qué es una clase
Entiendo qué es un objeto
Puedo crear un contacto
Puedo agregar contactos a la agenda
Puedo buscar contactos
Entiendo cómo funciona for
Entiendo cómo funciona una lista
Puedo ejecutar el sistema completo

Felicitaciones

Si llegaste hasta aquí, ya entiendes los conceptos básicos de:

- Programación Orientada a Objetos
- Listas y Diccionarios
- Métodos y funciones
- Estructuras de control