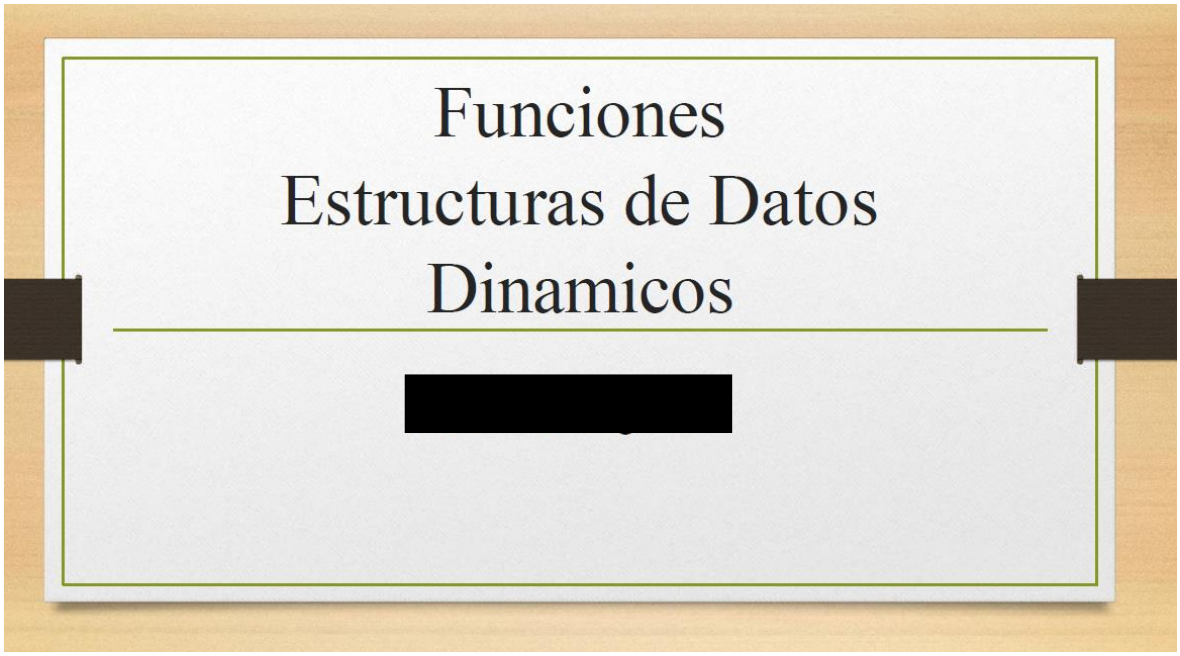
Clase 3_ Funciones y Listas



Funciones

- Una función es un bloque de código con un nombre asociado, que recibe cero o más argumentos como entrada, sigue una secuencia de sentencias, la cuales ejecuta una operación deseada y devuelve un valor y/o realiza una tarea, este bloque puede ser llamados cuando se necesite.
- El uso de funciones es un componente muy importante del paradigma de la programación llamada *estructurada*, y tiene varias ventajas:
- **modularización**: permite segmentar un programa complejo en una serie de partes o módulos más simples, facilitando así la programación y el depurado.
- **reutilización**: permite reutilizar una misma función en distintos programas.
- Python dispone de una serie de *funciones integradas* al lenguaje, y también permite crear funciones definidas por el usuario para ser usadas en su propios programas.

Sentencia def

- La sentencia def es una definición de función usada para crear objetos funciones definidas por el usuario.
- Una definición de función es una sentencia ejecutable. Su ejecución enlaza el nombre de la función en el namespace local actual a un objecto función (un envoltorio alrededor del código ejecutable para la función). Este objeto función contiene una referencia al namespace local global como el namespace global para ser usado cuando la función es llamada.
- La definición de función no ejecuta el cuerpo de la función; esto es ejecutado solamente cuando la función es llamada.

La sintaxis para una definición de función en Python es:

```
def NOMBRE(LISTA_DE_PARAMETROS):
    """DOCSTRING_DE_FUNCION"""
    SENTENCIAS
    RETURN [EXPRESION]
```

A continuación se detallan el significado de pseudocódigo fuente anterior:
NOMBRE, es el nombre de la función.
LISTA_DE_PARAMETROS, es la lista de parámetros que puede recibir una función.
DOCSTRING_DE_FUNCION, es la cadena de caracteres usada para documentar la función.
SENTENCIAS, es el bloque de sentencias en código fuente Python que realizar cierta operación dada.
RETURN, es la sentencia return en código Python.
EXPRESION, es la expresión o variable que devuelve la sentencia return.

Un ejemplo simple de función esta seguidamente:

```
>>> def hola(arg):
...     """El docstring de la función"""
...     print "Hola", arg, "!"
...
>>> hola("Plone")
Hola Plone !
```

Advertencia:
Los bloques de function deben estar indentado como otros bloques estructuras de control.

La palabra reservada def se usa para definir funciones. Debe seguirle el nombre de la función en el ejemplo anterior hola() y la lista de parámetros formales entre paréntesis. Las sentencias que forman el cuerpo de la función empiezan en la línea siguiente, y deben estar indentado.

Funciones: Parámetros

Es posible ingresar datos al ser invocadas a estos datos se les denomina argumentos y son ligados a nombres, los cuales se conocen como parámetros. El número de argumentos ingresados debe corresponder al número de parámetros que se definen. En caso de que no se ingresen los argumentos necesarios, se generará un error de tipo *TypeError*.

```
def suma(numero1,numero2):
    '''función la cual suma dos números'''
    print numero1 + numero2
    print "\n"
```

Sentencia pass

Es una operación nula — cuando es ejecutada, nada sucede. Eso es útil como un contenedor cuando una sentencia es requerida sintácticamente, pero no necesita código que ser ejecutado, por ejemplo:

```
>>> # una función que no hace nada (aun)
... def consultar_nombre_genero(letra_genero): pass
...
>>> type(consultar_nombre_genero)
<type 'function'>
>>> consultar_nombre_genero("M")
>>>
>>> # una clase sin ningún método (aun)
... class Persona: pass
...
>>> macagua = Persona
>>> type(macagua)
<type 'classobj'>
```

Sentencia return

Las funciones pueden comunicarse con el exterior de las mismas, al proceso principal del programa usando la sentencia return. El proceso de comunicación con el exterior se hace devolviendo valores. A continuación, un ejemplo de función usando return:

```
def suma(numero1,numero2):
    '''función la cual suma dos números'''
    print numero1 + numero2
    print "\n"
```

```
>>> suma(23,74)
97
```

Retorno múltiple

Una característica interesante, es la posibilidad de devolver valores múltiples separados por comas:

```
>>> def prueba():
...     return "Plone CMS", 20, [1,2,3]
...
>>> prueba()
('Plone CMS', 20, [1, 2, 3])
```

En el código anterior los valores múltiples se tratan en conjunto como una *tupla* inmutable y se pueden reasignar a distintas variables:

```
>>> def prueba():
...     return "Plone CMS", 20, [1,2,3]
...
>>> prueba()
('Plone CMS', 20, [1, 2, 3])
>>> cadena, numero, lista = prueba()
>>> print cadena, type(cadena)
Plone CMS <type 'str'>
>>> print numero, type(numero)
20 <type 'int'>
>>> print lista, type(lista)
[1, 2, 3] <type 'list'>
```

Usos de la sentencia pass:

Imagina que estás construyendo una calculadora en Python. Quieres definir las funciones para las operaciones básicas (suma, resta, multiplicación, división), pero aún no has implementado la lógica de cada operación. Puedes usar pass como marcador de posición para evitar errores de sintaxis mientras te enfocas en la estructura general de tu programa.

```python
def sumar(a, b):
  """Suma dos números."""
  pass  # Implementación pendiente

def restar(a, b):
  """Resta dos números."""
  pass  # Implementación pendiente

def multiplicar(a, b):
  """Multiplica dos números."""
  pass  # Implementación pendiente

def dividir(a, b):
  """Divide dos números."""
  pass  # Implementación pendiente
```

```python
# Ejemplo de uso (aún no funciona correctamente)
resultado_suma = sumar(5, 3)
resultado_resta = restar(10, 4)

print("Resultado de la suma:", resultado_suma)  # Imprimirá None
print("Resultado de la resta:", resultado_resta) # Imprimirá None
```

Aclaraciones del uso:

Hemos definido cuatro funciones (sumar, restar, multiplicar, dividir) que representan las operaciones aritméticas básicas.

Dentro de cada función, hemos usado pass para indicar que la implementación real de la operación se realizará más adelante.

Al llamar a las funciones, obtendremos None como resultado, ya que aún no hemos agregado la lógica para realizar los cálculos.

Esto nos permite probar la estructura general de nuestra calculadora y asegurarnos de que las funciones se pueden llamar correctamente, incluso antes de implementar la lógica de las operaciones.

Cuáles son los beneficios de usar pass en este caso? :

Evitar errores de sintaxis: Python requiere que haya al menos una declaración dentro de una función. pass cumple con este requisito sin realizar ninguna acción.

Desarrollo iterativo: Puedes comenzar con la estructura general de tu programa y luego implementar la lógica de cada función de forma individual.

Claridad del código: Indica claramente que la implementación de una función está pendiente.

Recuerda que pass es solo un marcador de posición. Una vez que estés listo para implementar la lógica de las operaciones aritméticas, puedes reemplazar pass con el código correspondiente.

Uso de la sentencia return

La sentencia return se utiliza dentro de una función para devolver un valor al código que la llamó. Esto permite que las funciones comuniquen resultados al exterior y se integren con el flujo principal del programa.

Cuando se ejecuta return dentro de una función, la función finaliza inmediatamente y el valor especificado después de return se devuelve al punto donde se llamó la función.

Ejemplo:

```
def suma(numero1, numero2):
  """Función que suma dos números."""
  print(numero1 + numero2)
  print("\n")

suma(23, 74)
```

En este ejemplo:

Se define una función llamada suma que toma dos argumentos (numero1 y numero2).

Dentro de la función, se calcula la suma de los dos números y se imprime el resultado.

Se llama a la función suma con los argumentos 23 y 74.

La función imprime la suma (97) y un salto de línea.

Ejemplo 2:

```python
def suma(numero1, numero2):
  """Función que suma dos números y devuelve el resultado."""
  resultado = numero1 + numero2
  return resultado

resultado_suma = suma(23, 74)
print(resultado_suma)  # Imprime 97
```

En este ejemplo mejorado:

La función suma calcula la suma y la almacena en la variable resultado.

En lugar de imprimir el resultado directamente, la función usa return resultado para devolver el valor calculado.

La variable resultado_suma almacena el valor devuelto por la función suma.

Imprimimos resultado_suma para mostrar el resultado de la suma.

Beneficios de usar return:

Reutilización de resultados: Permite usar el resultado de una función en otras partes del programa.

Modularidad: Facilita la creación de funciones independientes que realizan tareas específicas.

Claridad del código: Mejora la legibilidad y organización del código al separar la lógica de cálculo de la presentación de resultados.

Estructura de datos dinámicos: Listas

Las listas son estructuras de datos que se encargan de almacenar y ordenar elementos, las listas son del tipo mutable ¿Que quiere decir esto? que se pueden añadir, eliminar y editar datos en tiempo de ejecución a diferencia de las tuplas que son inmutables y una vez creadas ya no se pueden modificar.

Las listas en Python pueden manejar varios tipos de datos a la vez, es decir puede tener elementos numéricos y cadenas de texto mezclados sin ningún problema, recordemos que las variables en Python son dinámicas, entonces no tenemos problemas para realizar esta declaración:

```
>> lista = [True, 1, "hola", 42, False]
```

Podemos ver que la sintaxis es muy simple, corchetes que contienen los elementos que van separados por comas, podemos observar que primero hay un booleano, luego un número, una cadena, otro numero y finalmente otro booleano.

Mostrar elementos de una Lista

Para mostrar en pantalla todos los elementos podemos usar la función print() así:

```
>> print("\nEl contenido de la lista es:\n",lista, "\n")
```

El contenido de la lista es:

[True, 1, 'hola', 42, False]

Para el mismo propósito y de manera similar, podemos hacer uso de un for donde imprimimos los valores de la variable "i" que cambia a lo largo del recorrido de la lista:

```
>>for i in lista:
>>    print(i)
True
1
hola
42
False
```

Mostrar elementos de una lista

Si queremos acceder a un elemento específico de la lista lo podemos hacer por medio de su índice, como en los Arrays, el índice parte de 0, entonces si queremos el primer término colocaríamos:

```
>>lista[0]
```

True

```
>>lista[3]
```

hola

Si queremos acceder a los elementos de una lista por medio de un rango, digamos por ejemplo del elemento 1 al 3, usamos [1:4] y muestra los elementos cuyo índice sea 1,2 ó 3 (no incluye el 4), por ejemplo:

```
>>lista[1:4]
```

[1, 'hola', 42]

```
>>lista[0:2]
```

[True, 1]

Métodos de las Listas

Las listas en Python tienen muchos métodos que podemos utilizar, entre todos ellos vamos a nombrar los más importantes. Para esto utilizaremos esta lista de ejemplo.

```
my_list = [2, 5, , 'DEFG', 1.2, 5]
```

Append()

Este método nos permite agregar nuevos elementos a una lista.

```
my_list.append(10) # [2, 5, 'DEFG', 1.2, 5, 10]
my_list.append([2,5]) # [2, 5, 'DEFG' , 1.2, 5, [2, 5]]
```

Podemos agregar cualquier tipo de elemento a una lista, pero tengan en cuenta lo que pasa cuando agregamos una lista dentro de otra, esta lista se agrega como uno y solo un elemento.

Métodos de las Listas

Extend()

Extend también nos permite agregar elementos dentro de una lista, pero a diferencia de append al momento de agregar una lista, cada elemento de esta lista se agrega como un elemento más dentro de la otra lista.

```
my_list.extend([2,5]) # [2, 5, 'DEFG', 1.2, 5, 2, 5]
```

Remove()

El método remove va a remover un elemento que se le pase como parámentro de la lista a donde se le esté aplicando.

```
my_list.remove(2) # [5, 'DEFG', 1.2, 5]
```

En este ejemplo estamos removiendo el elemento 2, de la lista que tiene por nombre "my_list".

Pop()

Quita el ítem en la posición dada de la lista, y lo devuelve. Si no se especifica un índice, a.pop() quita y devuelve el último ítem de la lista.

```
my_list.pop() # [5, 'DEFG', 1.2] #elimino el 5
```

Métodos de las Listas

Index()

Index devuelve el número de indice del elemento que le pasemos por parámetro.

```
my_list.index('DEFG') # 2
```

Aquí estamos preguntando por el indice de la cadena 'DEFG' dentro de la lista "my_list", esto devuelve 2.

Count()

Para saber cuántas veces un elemento de una lista se repite podemos utilizar el metodo count().

```
my_list.count(5) # 2
```

Contamos cuantas veces se repite el número 5 dentro de la lista, yesto devuelve 2.

Reverse()

También podemos invertir los elementos de una lista.

```
my_list.reverse() # [5, 1.2, 'DEFG', 5, 2]
```

Estas son algunos de los métodos más útiles ymás utilizados en las listas.

OPTATIVO II - PYTHON

EJERCITARIO DE CLASE:

PARTE 1

Ejercitario 4 – Funciones

Ejercicios Resueltos

1. Solicitar al usuario que ingrese un número entero e informar si es primo o no, utilizando una función booleana que lo decida.

Ejercicios Propuestos.

2. Escribir un programa que simule una calculadora y que contenga las siguientes funciones.

1º) Muestre un menú con 5 opciones:

- 1. Sumar dos números. (Función Suma)
- 2. Restar dos números. (Función Resta)
- 3. Multiplicar dos números. (Función Multiplicación)
- 4. Dividir dos números.(Función División)
- 5. Salir.

2º) Pida por teclado la opción deseada (dato carácter). Deberá ser introducida, mientras que, no sea mayor o igual que '1' y menor o igual que '5'.

3º) Ejecute la opción seleccionada del menú.

4º) Repita los pasos 1º, 2º y 3º, mientras que, el usuario no seleccione la opción 5 (Salir) del menú.

OPTATIVO II - PYTHON

Solución del ejercicio:

Definimos las funciones

```python
def suma(num1, num2):
    """Función para sumar dos números."""
    return num1 + num2

def resta(num1, num2):
    """Función para restar dos números."""
    return num1 - num2

def multiplicacion(num1, num2):
    """Función para multiplicar dos números."""
    return num1 * num2

def division(num1, num2):
    """Función para dividir dos números."""
    if num2 == 0:
        return "Error: No se puede dividir por cero."
    return num1 / num2
```

Creamos el menú de opciones:

```python
# Menú principal
while True:
    print("\nCalculadora")
    print("1. Sumar")
    print("2. Restar")
    print("3. Multiplicar")
    print("4. Dividir")
    print("5. Salir")
```

OPTATIVO II - PYTHON

Trabajamos en las opciones del usuario y en el funcionamiento de las condicionales

```
    # Solicitar la opción al usuario
    opcion = input("Ingrese la opción deseada (1-5): ")

    # Validar la opción ingresada
    if opcion < '1' or opcion > '5':
        print("Opción inválida. Intente de nuevo.")
        continue

    # Si la opción es salir, terminar el programa
    if opcion == '5':
        print("¡Hasta luego!")
        break

    # Solicitar los números al usuario
    num1 = float(input("Ingrese el primer número: "))
    num2 = float(input("Ingrese el segundo número: "))

    # Realizar la operación seleccionada
    if opcion == '1':
        resultado = suma(num1, num2)
        print(f"Resultado: {resultado}")
    elif opcion == '2':
        resultado = resta(num1, num2)
        print(f"Resultado: {resultado}")
    elif opcion == '3':
        resultado = multiplicacion(num1, num2)
        print(f"Resultado: {resultado}")
    elif opcion == '4':
        resultado = division(num1, num2)
        print(f"Resultado: {resultado}")
```

OPTATIVO II - PYTHON

Aclaraciones del código:

Se definen funciones separadas para cada operación matemática (suma, resta, multiplicación, división).

El programa principal muestra un menú con las opciones disponibles.

Se solicita al usuario que ingrese la opción deseada y se valida la entrada.

Se solicitan los números al usuario y se realiza la operación correspondiente.

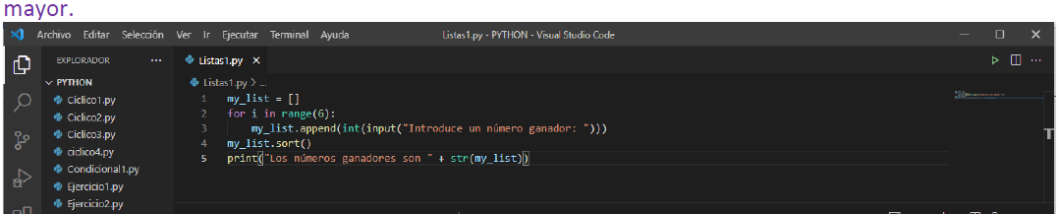
El programa se repite hasta que el usuario seleccione la opción de salir.

PARTE 2:

Ejercitario 5 – Estructuras de Datos Dinámicos (Listas)

Ejercicio Resuelto

1. Escribir un programa que pregunte al usuario los números ganadores de un sorteo, los almacene en una lista y los muestre por pantalla ordenados de menor a mayor.



Ejercicios propuestos – Listas

Escribir un programa que almacene las asignaturas de un curso (por ejemplo Matemáticas, Física, Química, Historia y Lengua, Castellano, Geología, Educación Física, etc.) en una lista.
- Las materias deben insertarse por el método append o insert mediante un input.
- Demostrar el método pop() o remove() eliminando un elemento de la lista.
- Recorrer la lista y mostrar en pantalla todos los elementos de la lista ("Asignatura:",listaAsignatura[i]) por cada asignatura de la lista.

Resultado ejemplo:
Asignatura: Matemáticas
Asignatura: Física
Asignatura: Química
Asignatura: Historia y Lengua….

Solución del ejercicio:

listas_ejer_asignturas.py > ...

```python
# Crear una lista para almacenar las asignaturas del curso
asignaturas = []

# Solicitar al usuario que ingrese las asignaturas del curso
while True:
    asignatura = input("Introduce una asignatura (o escribe 'fin' para terminar): ")
    if asignatura.lower() == 'fin':
        break
    asignaturas.append(asignatura)

# Listar asignaturas cargadas
if asignaturas:
    print("\nAsignaturas cargadas:")
    for asignatura in asignaturas:
        print(f"Asignatura: {asignatura}")
else:
    print("\nNo se han cargado asignaturas.")

# Solicitar asignatura a eliminar
if asignaturas:
    asignatura_a_eliminar = input("\nIntroduce la asignatura a eliminar: ")
    if asignatura_a_eliminar in asignaturas:
        asignaturas.remove(asignatura_a_eliminar)
        print(f"Se eliminó la asignatura: {asignatura_a_eliminar}")
        # Mostrar la nueva lista
        print("\nNueva lista de asignaturas:")
        for asignatura in asignaturas:
            print(f"Asignatura: {asignatura}")
    else:
        print(f"La asignatura '{asignatura_a_eliminar}' no existe en la lista.")
else:
    print("\nNo hay asignaturas para eliminar.")
```

OPTATIVO II - PYTHON

Aclaraciones del código:

Se crea una lista vacía llamada asignaturas para almacenar los nombres de las asignaturas.

Se utiliza un bucle while para solicitar al usuario que ingrese asignaturas hasta que escriba "fin".

Cada asignatura ingresada se agrega a la lista asignaturas usando el método append().

Se demuestra el método pop() eliminando el último elemento de la lista y mostrando el elemento eliminado.

Alternativamente, se puede usar el método remove() para eliminar un elemento específico de la lista.

Se recorre la lista asignaturas usando un bucle for y se muestra cada asignatura en la consola con el formato "Asignatura: [nombre de la asignatura]".

Se agrega un bloque if asignaturas: para verificar si la lista no está vacía antes de intentar listar las asignaturas.

Se utiliza un bucle for para iterar sobre la lista y mostrar cada asignatura en el formato "Asignatura: [nombre de la asignatura]".

Se agrega un mensaje si no se han cargado asignaturas.

Solicitar asignatura a eliminar:

Se agrega un bloque if asignaturas: para verificar si la lista no está vacía antes de solicitar la asignatura a eliminar.

Se solicita al usuario que ingrese la asignatura a eliminar.

Se verifica si la asignatura ingresada existe en la lista usando if asignatura_a_eliminar in asignaturas:.

Si la asignatura existe, se elimina usando asignaturas.remove(asignatura_a_eliminar) y se muestra un mensaje de confirmación.

Mostrar la nueva lista: Se agrega un bucle for para mostrar la lista de asignaturas actualizada después de la eliminación.

Si la asignatura no existe, se muestra un mensaje de error.

Se agrega un mensaje si no hay asignaturas para eliminar.

Ejercicio propuesto N° 3:

Gestión de Citas en una Veterinaria

Escribe un programa en Python que permita gestionar las citas de una veterinaria. El programa debe permitir:

*Agregar citas: El usuario debe poder ingresar el nombre del paciente, el tipo de mascota (perro, gato, etc.), la fecha y hora de la cita, y el motivo de la consulta.
*Listar citas: El programa debe mostrar todas las citas programadas, ordenadas por fecha y hora.
*Buscar citas: El usuario debe poder buscar citas por nombre del paciente o tipo de mascota.
*Eliminar citas: El usuario debe poder eliminar una cita específica.

Utiliza una lista para almacenar las citas. Cada cita debe ser un diccionario con la siguiente estructura:

```
{
    "paciente": "Nombre del paciente",
    "mascota": "Tipo de mascota",
    "fecha": "Fecha de la cita",
    "hora": "Hora de la cita",
    "motivo": "Motivo de la consulta"
}
```

OPTATIVO II - PYTHON

Solución del ejercicio:

Función agregar:

```python
def agregar_cita(citas):
    """Agrega una nueva cita a la lista de citas."""
    paciente = input("Nombre del paciente: ")
    mascota = input("Tipo de mascota: ")
    fecha = input("Fecha de la cita (AAAA-MM-DD): ")
    hora = input("Hora de la cita (HH:MM): ")
    motivo = input("Motivo de la consulta: ")
    cita = {
        "paciente": paciente,
        "mascota": mascota,
        "fecha": fecha,
        "hora": hora,
        "motivo": motivo
    }
    citas.append(cita)
    print("Cita agregada con éxito.")
```

Función listar

```python
def listar_citas(citas):
    """Lista todas las citas programadas, ordenadas por fecha y hora."""
    if not citas:
        print("No hay citas programadas.")
        return
    citas_ordenadas = sorted(citas, key=lambda cita: (cita["fecha"], cita["hora"]))
    print("Citas programadas:")
    for cita in citas_ordenadas:
        print(f"- Paciente: {cita['paciente']}, Mascota: {cita['mascota']}, Fecha: {cita['fecha']}, Hora: {cita['hora']}, Motivo: {cita['
```

Función buscar

```python
def buscar_citas(citas):
    """Busca citas por nombre del paciente o tipo de mascota."""
    if not citas:
        print("No hay citas programadas.")
        return
    busqueda = input("Ingrese el nombre del paciente o tipo de mascota a buscar: ")
    resultados = [cita for cita in citas if busqueda.lower() in cita["paciente"].lower() or busqueda.lower() in cita["mascota"].lower()]
    if resultados:
        print("Citas encontradas:")
        for cita in resultados:
            print(f"- Paciente: {cita['paciente']}, Mascota: {cita['mascota']}, Fecha: {cita['fecha']}, Hora: {cita['hora']}, Motivo: {ci
    else:
        print("No se encontraron citas.")
```

Función eliminar

```python
def eliminar_cita(citas):
    """Elimina una cita específica."""
    if not citas:
        print("No hay citas programadas.")
        return
    listar_citas(citas)
    indice = int(input("Ingrese el número de la cita a eliminar: ")) - 1
    if 0 <= indice < len(citas):
        cita_eliminada = citas.pop(indice)
        print(f"Cita eliminada: {cita_eliminada}")
    else:
        print("Número de cita inválido.")
```

Menú de opciones:

```python
# Programa principal
citas = []
while True:
    print("\nGestión de Citas Veterinaria")
    print("1. Agregar cita")
    print("2. Listar citas")
    print("3. Buscar citas")
    print("4. Eliminar cita")
    print("5. Salir")
    opcion = input("Ingrese una opción: ")
    if opcion == "1":
        agregar_cita(citas)
    elif opcion == "2":
        listar_citas(citas)
    elif opcion == "3":
        buscar_citas(citas)
    elif opcion == "4":
        eliminar_cita(citas)
    elif opcion == "5":
        break
    else:
        print("Opción inválida.")
```

Ejercicio de aplicación

Gestión de Inventario de una Tienda de Mascotas

Escribe un programa en Python que permita gestionar el inventario de una tienda de mascotas. El programa debe permitir:

*Agregar productos: El usuario debe poder ingresar el nombre del producto, la categoría (alimento, juguetes, accesorios, etc.), la cantidad en stock y el precio.

*Listar productos: El programa debe mostrar todos los productos en el inventario, ordenados por categoría.

*Buscar productos: El usuario debe poder buscar productos por nombre o categoría.

*Actualizar stock: El usuario debe poder actualizar la cantidad en stock de un producto específico.

*Calcular valor total del inventario: El programa debe calcular y mostrar el valor total del inventario (cantidad * precio).

Utiliza una lista para almacenar los productos. Cada producto debe ser un diccionario con la siguiente estructura:

```
{
    "nombre": "Nombre del producto",
    "categoria": "Categoría del producto",
    "stock": "Cantidad en stock",
    "precio": "Precio del producto"
}
```

Implementa funciones para cada una de las funcionalidades del programa (agregar, listar, buscar, actualizar, calcular valor total).