

## CASTEO de Tipos de Datos

El casteo de datos (o conversión de tipos) en Python es el proceso de transformar un tipo de dato a otro. Esto es útil cuando necesitas realizar operaciones con tipos de datos que no son directamente compatibles entre sí.

### **Principales funciones de casteo en Python:**

- 1. Casteo a entero (int):** Convierte un valor a un número entero. Si el valor es flotante, trunca la parte decimal.

#### **# De string a entero**

```
num_str = "10"
```

```
num = int(num_str) # Resultado: 10
```

#### **# De float a entero (trunca la parte decimal)**

```
num_float = 12.67
```

```
num = int(num_float) # Resultado: 12
```

- 2. Casteo a flotante (float):** Convierte un valor a un número con decimales.

#### **# De string a float**

```
num_str = "15.75"
```

```
num = float(num_str) # Resultado: 15.75
```

#### **# De entero a float**

```
num_int = 7
```

```
num = float(num_int) # Resultado: 7.0
```

- 3. Casteo a string (str):** Convierte un valor en una cadena de texto.

#### **# De entero a string**

```
num = 100
```

```
text = str(num) # Resultado: "100"
```

#### **# De float a string**

```
num = 45.32
```

```
text = str(num) # Resultado: "45.32"
```

- 4. Casteo a booleano (bool):** Convierte un valor en un valor booleano (True o False). En Python, valores como 0, None, False, '' (cadena vacía) se consideran falsos, mientras que cualquier otro valor se considera verdadero.

# De entero a booleano

```
num = 0
```

```
is_true = bool(num) # Resultado: False
```

# De string a booleano

```
text = "Hola"
```

```
is_true = bool(text) # Resultado: True (no está vacío)
```

- 5. Casteo a lista (list):** Convierte un objeto iterable (como una cadena, tupla o conjunto) a una lista.

# De string a lista (divide la cadena por caracteres)

```
text = "python"
```

```
lista = list(text) # Resultado: ['p', 'y', 't', 'h', 'o', 'n']
```

# De tupla a lista

```
tupla = (1, 2, 3)
```

```
lista = list(tupla) # Resultado: [1, 2, 3]
```

- 6. Casteo a tupla (tuple):** Convierte un objeto iterable a una tupla.

# De lista a tupla

```
lista = [1, 2, 3]
```

```
tupla = tuple(lista) # Resultado: (1, 2, 3)
```

- 7. Casteo a conjunto (set):** Convierte un objeto iterable en un conjunto, eliminando los elementos duplicados.

# De lista a set

```
lista = [1, 2, 2, 3]
```

```
conjunto = set(lista) # Resultado: {1, 2, 3}
```

8. **Casteo a diccionario (dict):** Convierte un iterable de pares clave-valor a un diccionario.

# De lista de tuplas a diccionario

```
lista_tuplas = [("a", 1), ("b", 2), ("c", 3)]
```

```
diccionario = dict(lista_tuplas) # Resultado: {'a': 1, 'b': 2, 'c': 3}
```

### ERROR DE CASTEO

Si el valor no es compatible para el tipo al que intentas convertirlo, Python generará un error, como en el caso de intentar convertir una cadena no numérica a entero.

```
text = "abc"
```

```
num = int(text) # Esto genera un ValueError
```

## Clases en Python

En Python, las **clases** permiten crear **objetos** que combinan datos y funcionalidades. Una clase es una plantilla para crear instancias, que son objetos individuales con atributos y métodos propios.

### Definición básica de una clase

La sintaxis para definir una clase es:

```
class NombreClase:

    def __init__(self, atributos):

        # Constructor

        self.atributo = valor
```

- **\_\_init\_\_():** Es el **constructor** de la clase. Se ejecuta cuando creas una instancia y se usa para inicializar los atributos.
- **self:** Es una referencia a la instancia actual del objeto. Se usa para acceder a los atributos y métodos de la instancia.

### Ejemplo 1: Clase básica

```
class Persona:

    def __init__(self, nombre, edad):

        self.nombre = nombre # Atributo de la instancia

        self.edad = edad     # Atributo de la instancia

    def saludar(self):

        print(f"Hola, me llamo {self.nombre} y tengo {self.edad} años.")
```

# Crear una instancia de la clase Persona

```
persona1 = Persona("Juan", 25)
```

```
persona1.saludar() # Salida: Hola, me llamo Juan y tengo 25 años.
```

#### Explicación:

1. **Definición de clase:** Persona tiene dos atributos (nombre y edad) que se inicializan en el constructor `__init__()`.
2. **Métodos:** La clase tiene un método `saludar()` que imprime un saludo con el nombre y la edad.
3. **Instancias:** `persona1` es una instancia de la clase `Persona`, y puede acceder a sus atributos y métodos.

### Ejemplo 2: Atributos de clase vs. Atributos de instancia

Los **atributos de clase** son compartidos por todas las instancias, mientras que los **atributos de instancia** son únicos para cada objeto.

```
class Coche:

    ruedas = 4 # Atributo de clase

    def __init__(self, marca, modelo):

        self.marca = marca # Atributo de instancia

        self.modelo = modelo # Atributo de instancia
```

# Crear instancias de Coche

```
coche1 = Coche("Toyota", "Corolla")
```

```
coche2 = Coche("Honda", "Civic")
```

```
# Acceder a atributos
```

```
print(coche1.marca) # Salida: Toyota (Atributo de instancia)
```

```
print(coche2.ruedas) # Salida: 4 (Atributo de clase)
```

## Definición e Implementación de Métodos y Parámetros en Python

### Métodos

Un **método** en Python es una función que pertenece a una clase. Se definen usando la palabra clave `def` dentro de una clase y siempre deben tener el parámetro `self` como el primer parámetro, que hace referencia a la instancia de la clase.

#### Ejemplo básico de un método:

```
class Persona:
```

```
    def saludar(self): # Método sin parámetros
```

```
        print("Hola, ¿cómo estás?")
```

Aquí, `saludar` es un método que no recibe parámetros adicionales más allá de `self`. Para llamarlo, primero debes crear una instancia de la clase.

```
persona = Persona()
```

```
persona.saludar() # Salida: Hola, ¿cómo estás?
```

El uso de `self` es **obligatorio** cuando defines un método dentro de una clase en Python si ese método está destinado a operar sobre una **instancia** de la clase. Si no incluyes `self` como el primer parámetro, el método no sabrá a qué instancia pertenece y generará un error.

### Parámetros en Métodos

Los métodos pueden aceptar **parámetros** que permiten pasar datos adicionales para la lógica interna.

#### Ejemplo con parámetros:

```
class Persona:
```

```
    def saludar(self, nombre): # Método con un parámetro
```

```
        print(f"Hola, {nombre}, ¿cómo estás?")
```

En este caso, el método saludar recibe un parámetro nombre que se usa dentro del método.

```
persona = Persona()
```

```
persona.saludar("Carlos") # Salida: Hola, Carlos, ¿cómo estás?
```

### Tipos de Parámetros en Métodos

**Parámetros Posicionales:** Estos parámetros se pasan en el mismo orden en que se definen.

```
class Calculadora:
```

```
    def sumar(self, a, b): # Parámetros posicionales
```

```
        return a + b
```

```
calc = Calculadora()
```

```
print(calc.sumar(5, 3)) # Salida: 8
```

**Parámetros con Valor Predeterminado:** Puedes establecer un valor predeterminado para los parámetros. Si el valor no es proporcionado, se usa el valor por defecto.

```
class Calculadora:
```

```
    def sumar(self, a, b=10): # b tiene un valor predeterminado de 10
```

```
        return a + b
```

```
calc = Calculadora()
```

```
print(calc.sumar(5)) # Salida: 15
```

**Parámetros con Nombre (Keyword Arguments):** Estos permiten especificar qué valor se asigna a qué parámetro usando el nombre del parámetro.

```
class Calculadora:
```

```
    def restar(self, a, b):
```

```
        return a - b
```

```
calc = Calculadora()
```

```
print(calc.restar(b=5, a=10)) # Salida: 5
```

### Parámetros Arbitrarios (\*args y \*\*kwargs):

▢ \*args permite pasar una cantidad variable de argumentos posicionales como una tupla.

\*\*kwargs permite pasar una cantidad variable de argumentos nombrados como un diccionario.

class Operaciones:

```
def sumar_todos(self, *args): # *args para múltiples argumentos

    return sum(args)
```

```
def mostrar_info(self, **kwargs): # **kwargs para múltiples parámetros con
nombre
```

```
    for clave, valor in kwargs.items():

        print(f'{clave}: {valor}')
```

```
op = Operaciones()
```

```
print(op.sumar_todos(1, 2, 3, 4)) # Salida: 10
```

```
op.mostrar_info(nombre="Carlos", edad=30) # Salida: nombre: Carlos, edad: 30
```

### Métodos Especiales

Python tiene ciertos métodos especiales (también llamados "métodos mágicos") que comienzan y terminan con dos guiones bajos, como `__init__`, que es el **constructor** de la clase y se ejecuta automáticamente cuando se crea una instancia de la clase.

#### Ejemplo del método `__init__`:

```
class Persona:
```

```
    def __init__(self, nombre, edad): # Método constructor

        self.nombre = nombre

        self.edad = edad
```

```
    def saludar(self):
```

```
print(f"Hola, me llamo {self.nombre} y tengo {self.edad} años.")

persona = Persona("Laura", 25)

persona.saludar() # Salida: Hola, me llamo Laura y tengo 25 años.
```

### Parámetros self y cls

- **self**: Es el primer parámetro de los métodos de instancia y se refiere a la instancia actual de la clase.
- **cls**: Es el primer parámetro de los métodos de clase y se refiere a la clase misma. `cls` se refiere a la **clase** en lugar de a una instancia específica.

### Ejemplo de un método de clase con cls:

```
class Persona:

    contador = 0

    def __init__(self, nombre):

        self.nombre = nombre

        Persona.contador += 1

    @classmethod
    def total_personas(cls):

        print(f"Total de personas creadas: {cls.contador}")

p1 = Persona("Ana")

p2 = Persona("Luis")

Persona.total_personas() # Salida: Total de personas creadas: 2
```

### Resumen

- **Métodos**: Son funciones dentro de una clase que operan en instancias de la clase.
- **Parámetros**: Los métodos pueden tener diferentes tipos de parámetros: posicionales, con valor predeterminado, por nombre y arbitrarios (\*args y \*\*kwargs).
- **self y cls**: Son especiales para referirse a la instancia y la clase, respectivamente.



## Metodos Estaticos

Los **métodos estáticos** en Python son aquellos que no dependen ni de la instancia de una clase ni de la propia clase. A diferencia de los métodos normales (que reciben el parámetro `self`) o de clase (que reciben el parámetro `cls`), los métodos estáticos no reciben automáticamente ningún parámetro adicional.

### ¿Cuándo usar un método estático?

Usas un **método estático** cuando la lógica del método no necesita acceder a ninguna propiedad o método de la instancia o de la clase. En lugar de operar sobre los datos de una instancia específica, un método estático se comporta como una función normal dentro de una clase.

Para definir un método estático, se utiliza el decorador `@staticmethod`.

### Sintaxis básica

```
class MiClase:

    @staticmethod

    def mi_metodo_estatico():

        print("Este es un método estático.")
```

### Ejemplo detallado de uso

Imagina que tienes una clase que representa a una persona, y deseas agregar un método que valide si una edad es válida para ser considerada adulta (sin necesidad de que el método dependa de ninguna instancia de `Persona`).

```
class Persona:

    edad_minima_para_adulto = 18

    @staticmethod

    def es_adulto(edad):

        """Determina si la edad indica que alguien es adulto."""

        return edad >= Persona.edad_minima_para_adulto
```

En este ejemplo, `es_adulto` es un método estático porque simplemente toma una edad y devuelve `True` si la persona es adulta o `False` si no lo es, sin necesidad de referirse a ninguna instancia de la clase.

## Llamada a un método estático

Puedes llamar a un método estático tanto desde la clase como desde una instancia de la clase:

# Llamada desde la clase

```
print(Persona.es_adulto(20)) # Salida: True
```

# Llamada desde una instancia (aunque no es necesario)

```
p = Persona()
```

```
print(p.es_adulto(16)) # Salida: False
```

## Resumen

- **Métodos estáticos** no dependen de la instancia ni de la clase.
- Se definen con el decorador `@staticmethod`.
- Son útiles cuando la lógica del método no necesita acceder a propiedades o métodos de la clase o la instancia.
- Pueden ser llamados tanto desde la clase como desde una instancia.

Los métodos estáticos son una buena opción cuando simplemente deseas agrupar funciones relacionadas dentro de una clase sin necesidad de interactuar con sus atributos o instancias.

## Metodos para operar con cadenas (strings)

Python ofrece una amplia variedad de **métodos de cadenas** (strings) que te permiten realizar operaciones como modificar, analizar, y formatear cadenas de texto. A continuación, te explico los métodos más comunes con ejemplos prácticos.

### 1. `upper()` y `lower()`

Convierte todos los caracteres de una cadena a **mayúsculas** (`upper()`) o **minúsculas** (`lower()`).

```
texto = "Hola Mundo"
```

```
print(texto.upper()) # Salida: "HOLA MUNDO"
```

```
print(texto.lower()) # Salida: "hola mundo"
```

### 2. `capitalize()` y `title()`

- **`capitalize()`** convierte solo la primera letra de la cadena a mayúscula.
- **`title()`** convierte la primera letra de cada palabra en mayúscula.

```
texto = "hola mundo"

print(texto.capitalize()) # Salida: "Hola mundo"

print(texto.title())      # Salida: "Hola Mundo"
```

### 3. strip(), lstrip(), rstrip()

Elimina espacios en blanco (o caracteres específicos) de una cadena.

- **strip()** elimina los espacios en blanco al inicio y al final.
- **lstrip()** elimina los espacios al inicio.
- **rstrip()** elimina los espacios al final.

```
texto = "  Hola Mundo  "

print(texto.strip()) # Salida: "Hola Mundo"

print(texto.lstrip()) # Salida: "Hola Mundo  "

print(texto.rstrip()) # Salida: "  Hola Mundo"
```

### 4. replace()

Reemplaza todas las apariciones de una subcadena por otra.

```
texto = "Hola Mundo"

nuevo_texto = texto.replace("Mundo", "Amigo")

print(nuevo_texto) # Salida: "Hola Amigo"
```

### 5. split() y join()

- **split()** divide una cadena en una lista de subcadenas según un delimitador (por defecto es el espacio).
- **join()** une una lista de subcadenas en una sola cadena, usando un delimitador.

```
texto = "Hola, cómo estás"

palabras = texto.split() # Dividir por espacio

print(palabras)          # Salida: ['Hola,', 'cómo', 'estás']

texto_unido = " ".join(palabras) # Unir la lista con espacios

print(texto_unido)       # Salida: "Hola, cómo estás"
```

## 6. startswith() y endswith()

Verifica si una cadena comienza con una subcadena específica (startswith()) o si termina con una subcadena (endswith()).

```
texto = "Hola Mundo"
```

```
print(texto.startswith("Hola")) # Salida: True
```

```
print(texto.endswith("Mundo")) # Salida: True
```

## 7. find() y index()

- **find()** busca la primera aparición de una subcadena y devuelve su índice, o -1 si no la encuentra.
- **index()** hace lo mismo, pero genera un error (ValueError) si no se encuentra la subcadena.

```
texto = "Hola Mundo"
```

```
print(texto.find("Mundo")) # Salida: 5
```

```
print(texto.find("Amigo")) # Salida: -1
```

```
# print(texto.index("Amigo")) # Esto generaría un ValueError
```

## 8. count()

Cuenta cuántas veces aparece una subcadena en la cadena.

```
texto = "Hola Hola Hola"
```

```
print(texto.count("Hola")) # Salida: 3
```

## 9. isalpha(), isdigit(), isalnum()

- **isalpha()**: Verifica si la cadena contiene solo letras.
- **isdigit()**: Verifica si la cadena contiene solo dígitos.
- **isalnum()**: Verifica si la cadena contiene solo letras o dígitos.

```
texto = "Python"
```

```
print(texto.isalpha()) # Salida: True
```

```
numero = "12345"
```

```
print(numero.isdigit()) # Salida: True
```

```
alphanumeric = "Python123"
```

```
print(alphanumeric.isalnum()) # Salida: True
```

## 10. len()

Devuelve la longitud de la cadena (número de caracteres).

```
texto = "Hola Mundo"
```

```
print(len(texto)) # Salida: 10
```

## 11. format()

Se utiliza para formatear cadenas, incorporando valores en lugares específicos.

```
nombre = "Carlos"
```

```
edad = 30
```

```
texto = "Me llamo {} y tengo {} años".format(nombre, edad)
```

```
print(texto) # Salida: "Me llamo Carlos y tengo 30 años"
```

A partir de Python 3.6, puedes usar **f-strings** para un formato más conciso:

```
texto = f"Me llamo {nombre} y tengo {edad} años"
```

```
print(texto) # Salida: "Me llamo Carlos y tengo 30 años"
```

## 12. zfill()

Rellena la cadena con ceros a la izquierda para alcanzar una longitud específica.

```
numero = "42"
```

```
print(numero.zfill(5)) # Salida: "00042"
```

## 13. swapcase()

Cambia las mayúsculas por minúsculas y viceversa.

```
texto = "Hola Mundo"
```

```
print(texto.swapcase()) # Salida: "hOLA mUNDO"
```

## 14. center(), ljust(), rjust()

- **center()** centra la cadena, rellenando con espacios u otro carácter.
- **ljust()** alinea la cadena a la izquierda.
- **rjust()** alinea la cadena a la derecha.

```
texto = "Hola"
```

```
print(texto.center(10)) # Salida: "  Hola  "
```

```
print(texto.ljust(10)) # Salida: "Hola   "
```

```
print(texto.rjust(10)) # Salida: "    Hola"
```

## Resumen

Estos son algunos de los métodos más comunes para trabajar con cadenas en Python. La mayoría de estos métodos generan **nuevas cadenas** sin modificar la cadena original, ya que las cadenas en Python son inmutables.

## Recursión

La **recursión** en Python es una técnica en la que una función se llama a sí misma para resolver un problema. Un problema se divide en subproblemas más pequeños y similares hasta llegar a un caso base, que es la condición que detiene la recursión.

### Componentes de la recursión:

1. **Caso base:** La condición que detiene la recursión.
2. **Llamada recursiva:** La función se llama a sí misma para resolver el problema más pequeño.

### Ventajas:

- Permite resolver problemas complejos de forma sencilla (e.g., algoritmos de búsqueda o cálculo de series).

### Desventajas:

- Puede consumir más memoria si la profundidad de recursión es grande.
- Puede ser menos eficiente que las soluciones iterativas en ciertos casos.

Ejemplo de cómo leer una cadena de texto carácter por carácter usando **recursión** en Python:

```
def leer_cadena_recursiva(cadena, index=0):
```

```
    if index == len(cadena): # Caso base: si el índice ha llegado al final de la cadena
```

```
        return
```

else:

```
print(cadena[index]) # Imprime el carácter actual
```

```
leer_cadena_recursiva(cadena, index + 1) # Llamada recursiva al siguiente carácter
```

# Ejemplo de uso

```
texto = "Hola"
```

```
leer_cadena_recursiva(texto)
```

### Explicación:

1. **Caso base:** Si el índice `index` es igual a la longitud de la cadena, se detiene la recursión.
2. **Llamada recursiva:** Se imprime el carácter en la posición `index` y luego se llama a la función de nuevo, incrementando el índice en 1.

Salida para `texto = "Hola"`:

H

o

l

a

## **Trabajo Practico Clases, Métodos y Parámetros**

**1-** CASTEO: Codifique un programa que solicite el ingreso de un numero decimal y asigne el mismo a una variable **valorDecimal**, aplique las operaciones de CASTING para convertir la variable a los siguientes tipos de datos, short, int, long, float investigue las diferentes formas de lograr la conversión. Muestre por pantalla el resultado de las conversiones.

**2-** Si se asigna un valor a una variable fuera de rango (mayor de lo establecido) ¿Qué ocurre? ¿Existe alguna forma de resolverlo? Ejemplifique.

**3-** Codifique un algoritmo que solicite el ingreso de un numero de 3 dígitos (100 - 999) y por medio del uso de las operaciones matemáticas módulo 10 y división por 10 efectué la suma de los 3 dígitos del número. Ejemplo ingreso 563, salida del algoritmo 14. Plantee el algoritmo planteando métodos para su resolución.

**4-** Desarrolle un programa que ayude a una cajera a determinar el número de billetes y monedas que se necesitan de cada una de las siguientes denominaciones 200, 100, 50, 20, 10, 5, 2 y 1, y monedas de 0.50, 0.25, 0.10 y 0.05 centavos para una cantidad de dinero dada. Ejemplo si la cantidad es 1390,55 se necesitan 6 billetes de 200, 1 billete de 100, 1 billete de 50, 2 billetes de 20, 1 moneda de 0.50 y una moneda de 0.05 centavos. Plantee el algoritmo planteando métodos para su resolución.

### **Ejercicios con cadenas:**

**5-** Solicite el ingreso de una cadena y elimine todos los espacios de la misma, muestre la cadena resultante.

**6-** De la siguiente cadena “La lluvia en Mendoza es escasa” indique cual es el tamaño de la cadena es decir su número de caracteres.

**7-** Solicite el ingreso de una cadena y determine el tamaño de la misma y cuantas vocales tiene en total.

**8-** Reemplaza todas las vocales **a** de una cadena ingresada por teclado por una vocal **e**.

**9-** Recorre la cadena del ejercicio 6 y transforma cada carácter a su código ASCII. Muéstralos en línea recta, separados por un espacio entre cada carácter.

**10-** Convertir una cadena a mayúsculas o minúsculas, daremos opción a que el usuario pida que se desea hacer (convertir a mayúsculas o convertir a minúsculas) y mostrar el resultado por pantalla.

**11-** Pedir dos palabras por teclado, indicar si son iguales.

**12-** Dada la cadena “hipopotamo”, extraer la cuarta y quinta letra.

**13-** Pedir el ingreso de dos cadenas por teclado, indicar si la segunda cadena se encuentra dentro de la primera.

**14-** Indique si en Python existen o no variables de tipo valor y su contraparte tipo referencia como sucede en otros lenguajes como Java.



**15-** Indique que sucede si realizo la siguiente declaración de variable:

```
x = None print(x)
```

Explique y ejemplifique el uso de **None**

**16-** Codifique un método que reciba como parámetro una cadena y determine si la misma contiene o no números.

**17-** Cree una clase FuncionesPrograma y codifique una función estática **getFechaString** que reciba como parámetro una fecha y retorne la fecha como una cadena literal.

Ejemplo recibo 15/10/1900, la salida debe ser

Quince de Octubre de mil novecientos.

Cree una clase Principal que contenga un método main y haga uso de la función **getFechaString**.

**18-** En la clase FuncionesPrograma codifique una método **getFechaDate** estática que reciba como parámetro 3 valores enteros, día, mes, año y retorne la fecha de tipo date correspondiente.

En la clase Principal creada en el punto anterior haga uso de la función **getFechaDate**.

**19-** Cree una clase OperacionMatematica con dos atributos valor1 y valor2 y un atributo de nombre operación.

Agregue a la clase los siguientes 5 métodos e implemente la lógica correspondiente:

```
sumarNumeros()  
restarNumeros()  
multiplicarNumeros()  
dividirNumeros()
```

El quinto método será el siguiente:

```
aplicarOperacion(operacion){  
.....  
}
```

Cree una clase Calculo que contenga un método main, donde cree una instancia de la clase OperacionMatematica, asigne 2 valores para las variables de la instancia y ejecute la función aplicarOperacion, pasando como parámetro primero "+", después "-", a continuación "\*" y finalmente "/". Muestre por pantalla el resultado de las operaciones.

**20-** Cree una clase Fracción con dos atributos, numerador y denominador.

Agregue a la clase los siguientes 4 métodos e implemente los mismos:

```
sumarFracciones(Fraccion f1, Fraccion f2)
```

restarFracciones(Fraccion f1, Fraccion f2)  
multiplicarFracciones(Fraccion f1, Fraccion f2)  
dividirFracciones(Fraccion f1, Fraccion f2)

Todos los métodos deben retornar la fracción resultante de la operación.

Cree una clase **OperacionesFraccion** que contenga un método main donde se solicite al usuario el ingreso de 4 valores numéricos enteros con los cuales se crearan 2 objetos Fracción y finalizada la creación de los mismos se ejecutaran los 4 métodos implementados anteriormente asignando el resultado a una nueva variable de tipo Fracción y mostrando por pantalla el resultado de las operaciones realizadas.

### Ejercicios con Recursión

**21-** Codifique un programa que solicite un número entero mayor a cero y que mediante recursión sume todos los números naturales desde el número ingresado hasta 1.

Ejemplo: Ingreso 10

El programa debe sumar  $10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1$

**22-** Suma los dígitos de un número ingresado por el usuario de forma recursiva.

Ejemplo: el usuario ingresa 1596

El programa debe sumar  $1 + 5 + 9 + 6$

**23-** Crea un programa donde se pida el ingreso de una cadena y por medio de recursión mostrar la cadena de forma inversa.

Ejemplo: Ingreso "computadora de escritorio"

Invertir cadena "oirtotircse ed arodatupmoc"

**24-** Crea un programa que lea una cadena de texto carácter por carácter usando **recursión**.

Ejemplo: Ingreso "UTN FRM Mza"

Salida: U

T

N

F

R

M

M

z

a