Entrada y salida de datos: input(), print(), archivos (1 hora)

input()

La función `input()` permite obtener datos ingresados por el usuario en la consola. La función puede recibir un argumento que será el mensaje mostrado al usuario.

```
nombre = input("Ingrese su nombre: ")
print("Hola,", nombre)
```

Ejemplo:

```
numero = int(input("Ingrese un número entero: "))
print("El número ingresado es", numero)
```

print()

La función `print()` permite mostrar información en la consola. Puede recibir múltiples argumentos separados por comas, que serán concatenados automáticamente.

```
nombre = "Juan"
edad = 30
print("Mi nombre es", nombre, "y tengo", edad, "años")
```

Ejemplo:

```
x = 5
y = 3
suma = x + y
print(x, "+", y, "=", suma)
```

Archivos

Python facilita el manejo de archivos, tanto para leer como para escribir en ellos. Se pueden utilizar los modos "r" (lectura), "w" (escritura), "a" (añadir) y "x" (creación exclusiva).

Lectura de archivos

```
with open("archivo.txt", "r") as archivo:
    contenido = archivo.read()
    print(contenido)
```

Ejemplo:

```
with open("numeros.txt", "r") as archivo:
    numeros = [int(line.strip()) for line in archivo.readlines()]
    print("Números:", numeros)
```

Escritura de archivos

```
with open("nuevo_archivo.txt", "w") as archivo:
    archivo.write("Hola, mundo!")
```

Ejemplo:

```
nombres = ["Ana", "Carlos", "Beatriz"]
with open("nombres.txt", "w") as archivo:
    for nombre in nombres:
        archivo.write(nombre + "\n")
```

Añadir contenido a un archivo

```
with open("archivo_existente.txt", "a") as archivo:
    archivo.write("Esta línea se añadirá al final del archivo.\n")
```

Ejemplo:

```
nuevos_nombres = ["David", "Elena"]
with open("nombres.txt", "a") as archivo:
    for nombre in nuevos_nombres:
        archivo.write(nombre + "\n")
```

Lectura de archivos línea por línea

En algunos casos, es útil leer un archivo línea por línea en lugar de cargar todo el contenido en memoria. Esto se puede hacer con un bucle `for`:

```
with open("archivo.txt", "r") as archivo:
    for line in archivo:
        print(line.strip())
```

Ejemplo:

```
with open("palabras.txt", "r") as archivo:
    for linea in archivo:
        palabra = linea.strip()
        print("Palabra:", palabra)
```

Operaciones con archivos en modo binario

Para leer o escribir archivos en modo binario, se debe agregar la letra "b" al modo de apertura del archivo. Esto es útil cuando se trabaja con archivos que no son de texto, como imágenes o archivos comprimidos.

Lectura en modo binario:

```
with open("imagen.jpg", "rb") as archivo:
    contenido = archivo.read()
    print("Contenido en bytes:", contenido)
```

Escritura en modo binario:

```
datos_binarios = b'\x48\x6f\x6c\x61\x20\x6d\x75\x6e\x64\x6f\x21'
with open("archivo_binario.bin", "wb") as archivo:
    archivo.write(datos_binarios)
```

Formateo de cadenas de texto

Python proporciona diferentes maneras de formatear cadenas de texto para facilitar su lectura y presentación.

Método `format()`

El método `format()` permite reemplazar marcadores de posición en una cadena de texto con valores específicos:

```
nombre = "Juan"
edad = 30
frase = "Mi nombre es {} y tengo {} años".format(nombre, edad)
print(frase)
```

F-strings

Las F-strings (cadenas con formato) son una característica de Python 3.6 en adelante que permite incluir expresiones dentro de las cadenas de texto, utilizando llaves y un prefijo "f":

```
nombre = "Juan"
edad = 30
frase = f"Mi nombre es {nombre} y tengo {edad} años"
print(frase)
```

Ejemplo:

```
x = 5
y = 3
suma = x + y
print(f"{x} + {y} = {suma}")
```

Uso de archivos temporales

En algunos casos, es útil trabajar con archivos temporales que se eliminan automáticamente una vez que se cierra el archivo. Python proporciona el módulo `tempfile` para manejar este tipo de archivos.

```
import tempfile

with tempfile.TemporaryFile(mode="w+t") as archivo_temporal:
    archivo_temporal.write("Esta es una línea en un archivo temporal.")
    archivo_temporal.seek(0)
    contenido = archivo_temporal.read()
    print("Contenido del archivo temporal:", contenido)
```

Conversión de tipos de datos en entrada y salida

A menudo, es necesario convertir tipos de datos al leer o escribir información. Por ejemplo, al leer números de un archivo de texto, estos se encuentran en formato de cadena de caracteres. Es necesario convertirlos a números antes de realizar operaciones matemáticas.

Ejemplo:

```
with open("precios.txt", "r") as archivo:
    precios = [float(line.strip()) for line in archivo.readlines()]

total = sum(precios)
print(f"El total de los precios es: {total}")
```

Funciones de entrada y salida personalizadas

Para facilitar la reutilización de código y mejorar la legibilidad de los programas, es posible crear funciones personalizadas para operaciones de entrada y salida de datos.

Ejemplo:

```
def leer_numeros(archivo):
    with open(archivo, "r") as f:
        numeros = [int(line.strip()) for line in f.readlines()]
    return numeros

def guardar_numeros(archivo, numeros):
    with open(archivo, "w") as f:
        for numero in numeros:
            f.write(f"{numero}\n")

numeros = leer_numeros("numeros.txt")
numeros_ordenados = sorted(numeros)
guardar_numeros("numeros_ordenados.txt", numeros_ordenados)
```

Con estas notas y ejemplos, los estudiantes podrán comprender y practicar el manejo de entrada y salida de datos en Python, incluyendo el uso de funciones `input()` y `print()`, operaciones con archivos de texto y binarios, manejo de excepciones y formateo de cadenas de texto. Estas habilidades son fundamentales para desarrollar aplicaciones de Python más avanzadas y para trabajar con datos de manera efectiva.