Clase 5 curso python para profesores 2026

Intermezzo

Antes de continuar con el videojuego vamos a hablar de enteros, cadenas de caracteres y ficheros.

Pero antes vamos a realizar un ejercicio:

Ejercicio: Utilizando la tabla siguiente

Librerías python	
Variables (definiendo el valor inicial y para qué se utilizará)	
Estructuras de control necesarias	
Escribe en castellano el algoritmo	

Resolvemos el siguiente ejercicio: El programa creará un número aleatorio entre 1 y 1000. Se preguntará, un máximo de 5 veces, al usuario por el número. Si el usuario acierta con el número se mostrará un mensaje con el texto “Acertaste el número: X con un total de X intentos”. Si el usuario no acierta con el número se le indicará si su número es mayor o menor que el número mágico. Al terminar el total de intentos se mostrará que el número no ha sido acertado y se mostrará el número mágico.

Enteros

Vamos a trabajar con los números enteros. Esos números que van de ...-2, -1, 0, 1, 2,… ¿Hasta dónde/cuándo?

Pues en lenguajes de programación como C (podemos ver el código en entero.c y entero_largo.c) está claro. Si utilizamos un tipo int se utilizan 32 bits: (2^32 / 2) -1 (Hagámoslo en python),

Si utilizamos long long ahora son 64 bits: (2^64 /2)-1 (Volvamos a hacerlo en python. ¿Y qué ocurre cuando llegamos a ese valor y sumamos uno. En el caso de C obtiene el número negativo más pequeño que se pueda representar (Observemos la ejecución de los dos códigos de C que se encuentran en la carpeta enteros).

En el caso de python, como casi siempre, la cosa cambia. Aquí no se utilizan 4 u 8 bytes. Python los guarda en una especie de cadena de caracteres lo que permite que la cantidad sea tan grande como la memoria RAM que tengamos en el ordenador y no esté siendo utilizada ya por el sistema operativo.

Reto 1: Contamos la leyenda de los granos de arroz sobre el tablero de ajedrez que muestran el rápido crecimiento exponencial y hacemos el programa que nos muestre el total de granos en todo el tablero.

Reto 2: Calcula el factorial de 100 y muestra el número resultante y la cantidad de dígitos que tiene ese número. Recuerda que podemos convertir números en cadenas de caracteres con la función str()

Si el número fuera mayor de **4300 dígitos** aparecería el error: *ValueError: Exceeds the limit (4300) for integer string conversion; use sys.set_int_max_str_digits() to increase the limit* Te has topado con la **"Guardia de Seguridad"** que Python implementó recientemente (a partir de las versiones 3.10.7, 3.11, etc.).

Este error ocurre porque Python, para evitar ataques de denegación de servicio (DoS), pone un límite a cuántos dígitos puede convertir de "número" a "texto" (o viceversa). Procesar números de millones de dígitos consume muchísima CPU, y Python ahora viene con este freno de mano echado por defecto en **4300 dígitos**.

Vamos a explicar la_bestia_exponencial.py. Recemos….

String(cadenas de caracteres)

En Python, un string no es texto plano. Es una secuencia inmutable. ¿Qué significa inmutable? Que una vez creada, no la puedes cambiar. Si quieres modificarla, Python tiene que crear una copia nueva.

Un String (cadena) y una Lista parecen gemelos: ambos son secuencias, ambos se pueden indexar y ambos se pueden recorrer con un for.

Pero existen diferencias:

Característica	Strings (str)	Listas (list)
Tipo de dato	**Inmutable** (No cambia)	**Mutable** (Se puede modificar)
Contenido	Solo caracteres (Unicode)	Cualquier objeto (int, str, otras listas)
Acceso (index)	cadena[0] (Lectura)	lista[0] (Lectura)
Modificación	cadena[0] = 'X' **(ERROR)**	lista[0] = 'X' **(OK)**
Memoria	Más eficiente para texto	Más pesada (reserva espacio extra)

Operaciones Avanzadas de Cadenas

Para compensar que no podemos cambiarlas directamente, Python nos da un "arsenal" de métodos que devuelven nuevas cadenas transformadas:

- **Slicing (Rebanado):** `cadena[inicio:fin:paso]`. Es la forma más potente de extraer trozos.
- **Limpieza:** `.strip()`, `.lstrip()`, `.rstrip()` para quitar espacios o caracteres molestos.
- **Búsqueda:** `.find()`, `.startswith()`, `.endswith()`.
- **Transformación:** `.replace("viejo", "nuevo")`, `.upper()`, `.lower()`

Ejercicio de "Cambio" en una cadena

Como hemos dicho que no se pueden cambiar, ¿cómo lo hacemos en la vida real? Hay tres estrategias clásicas. Imagina que tenemos la palabra **"Pithon"** y queremos corregirla a **"Python"**.

Estrategia A: El Reemplazo (La vía rápida)

```
original = "Pithon"
corregida = original.replace("i", "y")
print(corregida) # "Python"
```

Estrategia B: Slicing (Cirugía de precisión)

Cortamos la cadena antes y después del error y pegamos la pieza nueva.

```
original = "Pithon"
# [Hasta el índice 1] + "y" + [Desde el índice 2 en adelante]
corregida = original[:1] + "y" + original[2:]
print(corregida) # "Python"
```

Estrategia C: Conversión a Lista (El truco del "Paso Intermedio")

Esta es la más útil cuando tienes que hacer **muchos cambios** complejos.

1. Convertimos a lista (que sí es mutable).
2. Cambiamos lo que queramos.
3. Volvemos a unir con `.join()`. Ahora vamos a ver qué es esto de JOIN

```
original = "Pithon"

# 1. Convertir a lista: ['P', 'i', 't', 'h', 'o', 'n']
lista_caracteres = list(original)

# 2. Modificar el elemento con libertad
lista_caracteres[1] = "y"

# 3. Reconstruir el string usando el "Costurero" que vimos antes
corregida = "".join(lista_caracteres)

print(corregida) # "Python”
```

Dividir cadenas, unir listas en cadenas

`split()`: El Divisor

Este método toma una cadena de texto y la **rompe** en trozos basándose en un separador que tú elijas. El resultado es siempre una **lista**.

- **Sintaxis:** `cadena.split("separador")`
- **Si no pones separador:** Por defecto corta por cualquier espacio en blanco (espacios, tabs, saltos de línea).

```
data = "manzana,pera,limón"
frutas = data.split(",")
# Resultado: ["manzana", "pera", "limón"]
```

`join()`: La Unión

Este es el "hermano mayor" de `split`. Toma una **lista** de cadenas y las une en un solo **string**.

- **Sintaxis:** `"pegamento".join(lista)`
- **Nota importante:** El método pertenece al "pegamento", no a la lista (un error común cuando comenzamos con python).

```
lista = ["2026", "02", "07"]
fecha = "-".join(lista)
# Resultado: "2026-02-07"
```

Ejercicios de `split()`

1. **El Extractor de Usuarios:** Tienes una URL `https://github.com/usuario/repositorio`. Usa `split("/")` para obtener solo el nombre del usuario.
2. **Contador de Palabras:** Pide una frase al usuario y usa `split()` para contar cuántas palabras tiene (calculando la longitud de la lista resultante).
3. **Limpieza de CSV:** Tienes la línea `Juan;Pérez;25;Madrid`. Separa los datos para obtener una lista con el nombre, apellido, edad y ciudad.
4. **Extensión de Archivo:** Dado un nombre de archivo como `foto_vacaciones.png`, obtén solo la extensión (`png en este ejemplo`) usando el punto como separador.
5. **El Hack de las IPs:** Tienes una dirección IP `192.168.1.10`. Sepárala en sus 4 octetos y guarda el último número en una variable.

Ejercicios de `join()` (El Pegamento)

1. **Formateador de Nombres:** Tienes una lista `["perez", "garcia", "juan"]`. Únelas con un espacio para formar el nombre completo.
2. **Generador de Rutas:** Tienes una lista de carpetas `["C:", "Users", "Admin", "Documents"]`. Únelas usando la barra invertida `\` para crear una ruta de directorio.
3. **De Lista a Oración:** Convierte la lista `["Python", "es", "genial"]` en una frase que termine con un punto.
4. **Creador de Etiquetas (Hashtags):** Tienes una lista de palabras clave `["python", "code", "learn"]`. Únelas de modo que queden así: `#python #code #learn`
5. Toma una lista de caracteres de una contraseña (ej: `['a', '1', 'b', '2']`) y únelos sin ningún separador (pegamento vacío `""`) para mostrar la cadena final.

Un truco extra: "Limpiar y Reconstruir"

A veces usamos los dos juntos (join y split) para "normalizar" un texto. Por ejemplo, para quitar todos los espacios extra de una frase:

```
frase_fea = "  Python    es     muy     divertido  "
# Cortamos por cualquier espacio y volvemos a unir con uno solo
frase_limpia = " ".join(frase_fea.split())
print(frase_limpia) # "Python es muy divertido"
```

f-string

Las **f-strings** (Formatted String Literals) son la forma más moderna, rápida y legible de insertar variables o expresiones dentro de una cadena de texto en Python. Aparecieron en la versión 3.6 y desde entonces son el estándar.

Para usarlas, solo tienes que poner una `f` justo antes de las comillas de apertura. A partir de ahí, cualquier cosa que pongas dentro de llaves `{}` será ejecutada por Python.

Sintaxis Básica

```
nombre = "Neo"
pildora = "Roja"

# Así de fácil:
print(f"Bienvenido, {nombre}. Has elegido la píldora {pildora}.")
```

¿Qué puedes poner dentro de las llaves?

Casi cualquier cosa que sea una expresión válida en Python:

- **Matemáticas:** `f"El doble de 10 es {10 * 2}"`
- **Llamadas a funciones:** `f"Tu nombre en mayúsculas es {nombre.upper()}"`
- **Condicionales (Ternarios):** `f"Acceso {'Permitido' if nivel > 10 else 'Denegado'}"`

Formateo de números (Potencia pura)

Las f-strings permiten controlar cómo se ven los números sin necesidad de redondear manualmente:

- **Decimales:** `f"{3.14159:.2f}"` → `3.14` (limita a 2 decimales).
- **Separador de miles:** `f"{1000000:,}"` → `1,000,000`. ¿Podrías explicar ahora la siguiente línea que vimos en la leyenda_de_ajedrez.py: **`f"{total_granos:,}".replace(",", ".")`**?
- **Porcentajes:** `f"{0.85:.1%}"` → `85.0%`.