LABORATORIO 2

Para realizar pruebas de rendimiento en Python, puedes utilizar el módulo `timeit`. Este módulo proporciona una forma sencilla de medir el tiempo de ejecución de pequeñas piezas de código en Python.

El uso básico de `timeit` es crear un objeto `Timer`, que recibe como argumento la instrucción que deseas medir, y luego llamar al método `timeit()` para obtener el tiempo de ejecución en segundos. Aquí tienes un ejemplo de cómo utilizar `timeit` para medir el tiempo de ejecución de una función:

```python
import timeit

def my_function():
    # Código que deseas medir el tiempo de ejecución
    pass

# Crear un objeto Timer para la función
t = timeit.Timer(lambda: my_function())

# Ejecutar la función 1000 veces y medir el tiempo promedio
time_taken = t.timeit(number=1000) / 1000

print("Tiempo de ejecución promedio: {:.6f} segundos".format(time_taken))
```

En el ejemplo anterior, `my_function()` es la función que deseas medir, y el objeto `Timer` se crea utilizando una función anónima (`lambda`) que llama a `my_function()`. El método `timeit()` se llama con el argumento `number=1000`, lo que significa que se ejecutará la función 1000 veces y se medirá el tiempo total de ejecución.

Además de `timeit`, también existen otras bibliotecas en Python que se utilizan comúnmente para realizar pruebas de rendimiento, como `pytest-benchmark`, `perf` y `time`. Estas bibliotecas proporcionan funcionalidades adicionales para realizar pruebas de rendimiento más complejas y precisas.

Aqui tienes un ejemplo que mide el tiempo de ejecución de dos funciones diferentes que realizan la misma tarea (recorrer una lista y sumar sus elementos). La primera función utiliza un bucle `for`, mientras que la segunda utiliza la función `sum()` de Python, que es más eficiente:

```python
import timeit

# Función que recorre la lista con un bucle for
def sum_with_for_loop(lst):
    total = 0
    for num in lst:
        total += num
    return total

# Función que utiliza la función sum() de Python
def sum_with_sum_function(lst):
    return sum(lst)
```

```
# Crear una lista de números
numbers = list(range(1, 10001))

# Medir el tiempo de ejecución de sum_with_for_loop()
time_taken1 = timeit.timeit(lambda: sum_with_for_loop(numbers), number=1000)

# Medir el tiempo de ejecución de sum_with_sum_function()
time_taken2 = timeit.timeit(lambda: sum_with_sum_function(numbers), number=1000)

# Imprimir los tiempos de ejecución
print("Tiempo de ejecución con for loop: {:.6f} segundos".format(time_taken1))
print("Tiempo de ejecución con sum() function: {:.6f} segundos".format(time_taken2))
```

Este ejemplo crea dos funciones que realizan la misma tarea (sumar los elementos de una lista), pero una utiliza un bucle `for` y la otra utiliza la función `sum()`. Luego, se crea una lista de números y se mide el tiempo de ejecución de ambas funciones utilizando `timeit`. Al imprimir los tiempos de ejecución, se puede observar que la función que utiliza `sum()` es más eficiente y tiene un tiempo de ejecución menor que la función con el bucle `for`.