Python - Medição de Desempenho

Um determinado problema pode ser resolvido por mais de um algoritmo alternativo. Portanto, precisamos otimizar o desempenho da solução. **O módulo timeit** do Python é uma ferramenta útil para medir o desempenho de um aplicativo Python.

A função timit() neste módulo mede o tempo de execução do seu código Python.

Sintaxe

```
timeit.timeit(stmt, setup, timer, number)
```

Parâmetros

- **stmt** - trecho de código para medição de desempenho.
- **setup** - a configuração detalha os argumentos ou variáveis a serem passados.
- **timer** - usa o timer padrão, portanto, pode ser ignorado.
- **number** - o código será executado esse número de vezes. O padrão é 1.000.000.

Exemplo

A instrução a seguir usa compreensão de lista para retornar uma lista de múltiplos de 2 para cada número no intervalo até 100.

```
>>> [n*2 for n in range(100)]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34,
36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68,
70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100,
102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126,
128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152,
154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178,
180, 182, 184, 186, 188, 190, 192, 194, 196, 198]
```

Para medir o tempo de execução da instrução acima, usamos a função timeit() da seguinte forma -

```
>>> from timeit import timeit
>>> timeit('[n*2 for n in range(100)]', number=10000)
0.0862189000035869
```

Compare o tempo de execução com o processo de anexar os números usando um loop for.

```
>>> string = '''
... numbers=[]
... for n in range(100):
... numbers.append(n*2)
... '''
>>> timeit(string, number=10000)
0.1010853999905521
```

O resultado mostra que a compreensão da lista é mais eficiente.

A string de instrução pode conter uma função Python para a qual um ou mais argumentos My podem ser passados como código de configuração.

Encontraremos e compararemos o tempo de execução de uma função fatorial usando um loop com o de sua versão recursiva.

A função normal usando o loop for é -

```
def fact(x):
   fact = 1
   for i in range(1, x+1):
      fact*=i
   return fact
```

Definição de fatorial recursivo.

```
def rfact(x):
   if x==1:
      return 1
   else:
      return x*fact(x-1)
```

Teste essas funções para calcular o fatorial de 10.

```
print ("Using loop:",fact(10))
print ("Using Recursion",rfact(10))
Result
Using loop: 3628800
Using Recursion 3628800
```

Agora encontraremos seus respectivos tempos de execução com a função timeit().

```
import timeit

setup1="""
from __main__ import fact
x = 10
```

```
"""

setup2="""
from __main__ import rfact
x = 10
"""

print ("Performance of factorial function with loop")
print(timeit.timeit(stmt = "fact(x)", setup=setup1, number=10000))

print ("Performance of factorial function with Recursion")
print(timeit.timeit(stmt = "rfact(x)", setup=setup2, number=10000))
```

Saída

```
Performance of factorial function with loop
0.00330029999895487
Performance of factorial function with Recursion
0.006506800003990065
```

A função recursiva é mais lenta que a função com loop.

Dessa forma, podemos realizar medições de desempenho do código Python.