Técnicas de programação

2025

Desempenho de código

- O mais importante: seu código funciona?
- Seu código precisa retornar um resultado dentro de um período de tempo específico?
- O usuário precisa agir com base nos resultados?
- Métricas são importantes!

Métodos para melhorar o desempenho

- Escolha do algoritmo
  - Evite iterações desnecessárias
- Escolha da estrutura de dados
  - É mais rápido procurar um valor em um dicionário do que em uma lista
- Use funções built-in
  - Muitas são implementadas em C
- Compilando Python
  - Cython (superconjunto do Pyton), Numba (subconjunto do python) e PyPy (reimplementação do Python)
  - Código Assíncrono
  - Realiza uma tarefa enquanto aguarda outra
- Computação paralela e computação distribuída
  - MapReduce
    - Modelo de programação para BigData

JIT – Just in time compilation

- “Compilação Just-In-Time”) a compilação acontece **durante a execução** do programa, em vez de antes da execução
- permite que o código seja traduzido de uma forma interpretada para código de máquina altamente otimizado em tempo real

Característica	JIT (Just-In-Time)	Compilação Tradicional
Momento da compilação	Durante a execução	Antes da execução
Flexibilidade	Pode adaptar e otimizar em tempo real	Código fixo após a compilação
Velocidade inicial	Mais lenta (devido à compilação em tempo real)	Mais rápida (pois já está compilado)
Velocidade após otimização	Muito rápida (depende do JIT)	Rápida, mas sem otimizações dinâmicas

Cronometrando a execução do código

```
def slow_way_to_calculate_mode(list_of_numbers):
        result_dict = {}
        for i in list_of_numbers:
                if i not in result_dict:
                        result_dict[i] = 1
                else:
                        result_dict[i] += 1
        mode_vals = []
        max_frequency = max(result_dict.values())
        for key, value in result_dict.items():
                if value == max_frequency:
                        mode_vals.append(key)
        return mode_vals
```

```
import numpy as np
import time
import timeit
```

```
ramdom_integers = np.random.randint(1,1_000_000,1_000_000)
start=time.time()
slow_way_to_calculate_mode(ramdom_integers)
end=time.time()
print(end-start)
```

```
mode_timer = timeit.Timer(stmt="slow_way_to_calculate_mode(ramdom_integers)",
setup="from __main__ import slow_way_to_calculate_mode, ramdom_integers")
time_taken = mode_timer.timeit(number=1)
print(f"Tempo de execucao:{time_taken} seconds")
```

Código mais rápido

```
from collections import Counter

ramdom_integers = np.random.randint(1, 1_000_000, 1_000_000)

# Função usando Counter
def mode_using_counter(list_of_numbers):
        c = Counter(list_of_numbers)
        return c.most_common(1)[0][0]


mode_timer = timeit.Timer(stmt="mode_using_counter(ramdom_integers)",
setup="from __main__ import mode_using_counter, ramdom_integers")

time_taken = mode_timer.timeit(number=1)
print(f"Tempo de execução: {time_taken:.4f} seconds")
```