1. Knihovny a moduly pro matematické výpočty

Zadání:
V tomto kurzu jste se učili s některými vybranými knihovnami. Některé sloužily pro rychlé vektorové operace, jako numpy, některé mají naprogramovány symbolické manipulace, které lze převést na numerické reprezentace (sympy), některé mají v sobě funkce pro numerickou integraci (scipy). Některé slouží i pro rychlé základní operace s čísly (numba).

Vaším úkolem je změřit potřebný čas pro vyřešení nějakého problému (např.: provést skalární součin, vypočítat určitý integrál) pomocí standardního pythonu a pomocí specializované knihovny. Toto měření proveďte alespoň pro 5 různých úloh (ne pouze jiná čísla, ale úplně jiné téma) a minimálně porovnejte rychlost jednoho modulu se standardním pythonem. Ideálně proveďte porovnání ještě s dalším modulem a snažte se, ať je kód ve standardním pythonu napsán efektivně.

Řešení:
1. úloha: Skalární součin dvou vektorů
V této úloze jsem zkoušela skalární součet dvou vektorů s 1 000 000 náhodných čísel.
Časové porovnání této funkce v čistém Pythonu a s pomocí knihovny NumPy vyšlo takto:

```
Skalární součin (Python):  0.05787467956542969
Skalární součin (NumPy):   0.0008852481842041016
```

Z výsledku můžeme vidět, že knihovna NumPy nám umožňuje snadnější a rychlejší výpočet skalárního součinu.

2. úloha: Výpočet faktoriálu
Tato úloha se zabývá výpočtem faktoriálu. Nejdřív pomocí čistého Pythonu a poté pomocí knihovny SymPy. Vyšlo m toto:

```
Výpočet faktoriálu (Python):  0.016252756118774414
Výpočet faktoriálu (SymPy):   4.482269287109375e-05
```

Opět můžeme vidět, že výpočet pomocí knihovny je výrazně rychlejší než bez ní.

3. úloha: Násobení matic
Tato úloha má ukázat efektivitu násobení matic o velikosti 3x3 pomocí knihovny NumPy.

```
Násobení matic (Python):  6.4849853515625e-05
Násobení matic (NumPy):   3.409385681152344e-05
```

Výsledky ukazují, že i v případě násobení matic je použití knihovny NumPy výrazně rychlejší než implementace v čistém Pythonu. Čistý Python postup využívá tři vnořené smyčky k výpočtu, zatímco NumPy používá vektorové operace pro rychlejší výpočet.

4.úloha: Aritmetický průměr

V této úloze ukazuji aritmetický průměr listu čísel o délce 1 000 000 čísel. Výsledky:

```
Aritmetický průměr (Python):  0.003114461898803711
Aritmetický průměr (NumPy):   0.0006120204925537109
```

Ve výsledcích můžeme vidět, že pro výpočet aritmetického průměru je použití NumPy výrazně rychlejší než čistý Python. Čistý Python musí projít celý seznam a získat sumu prvků, což může trvat delší dobu. Zatímco NumPy vypočítá průměr pomocí své vestavěné funkce mean() mnohem efektivněji a rychleji.

5. úloha: Výpočet sin(x)

Tentokrát porovnávám 3 knihovny, konkrétně Math, NumPy a SymPy.

```
Výpočet sin(x) (Python):  5.173683166503906e-05
Výpočet sin(x) (NumPy):   0.0001895427703857422
Výpočet sin(x) (Sympy):   0.05016517639160156
```

Implementace pomocí Pythonu s použitím knihovny Math je nejrychlejší, jak ukazuje výsledek. Implementace pomocí NumPy je o něco pomalejší, a nakonec implementace pomocí knihovny SymPy je nejpomalejší.