Programování 2

1. cvičení, 18-2-2025

Obsah

- co budeme dělat v tomto semestru
- co od sebe očekáváme a pravidla hry

Co budeme dělat

NMN112 Programování II

> Přednáška bude věnována zejména problematice algoritmického návrhu programů – tedy seznámení se základními algoritmy, datovými strukturami a programovacími technikami s ohledem na efektivitu vytvářených programů. Na cvičeních vedle toho doplníme a procvičíme znalosti programovacího jazyka Python.

Na těchto cvičeních budeme pokračovat tam, kde jste začli v minulém semestru - budeme se učit psát dobrý kód v Pythonu. Kromě psaní správného, čitelného a kompaktního kódu se budeme věnovat i

- efektivitě kódu - už nebude stačit syntakticky správný kód, který dělá to, co má: budu chtít, aby váš kód dělal, co má, s rozumným využitím výpočetních prostředků - paměti a času.
- Budeme se učit implementovat v Pythonu algoritmy z přednášek a teoretických cvičení. Budete psát hodně kódu, abyste se naučili psát kód rychle a rychle ho také odladit.
- Budete také mít spoustu možností vymýšlet si své vlastní algoritmy pro různé úlohy.

Přibližný program - zatím bez uspořádání podle týdnů:

1. Úvod, podmínky k zápočtu, ReCodEx. Instalace Pythonu a vývojová prostředí.
2. Jednoduché problémy, návrh a ladění algoritmů
3. Třídění - různé algoritmy, halda (heap) a její implementace
4. Permutace
5. Hry a optimální strategie
6. Rekurze, memoizace, dynamické programování
7. Grafové algoritmy, zásobníky a fronty

K tomuto "hlavnímu" programu pak přidáme "podprogramy" -- u každého cvičení se naučíme také něco nového z Pythonu, a budeme se učit číst kód.

GitHub

Všechny materiály k tomuto cvičení (zejména doprovodní texty) najdete v repozitáři na GitHubu, https://github.com/PKvasnick/Programovani-2. Použijte git v Linuxu anebo GitHub Desktop ve Windows, abyste si mohli stáhnout celý repozitář na svůj počítač. Nezapomeňte vždy před cvičením provést **pull**, abyste si stáhli nejnovější změny v repozitáři.

Pomoc

Budeme dělat jednoduché věci, ale programování je spojeno s častými pocity frustrace, když vám nebude fungovat něco, co by podle vás určitě fungovat mělo.

Základní postup v takovémto případě dlouho bylo **zeptat se lámanou angličtinou Googlu**. Zpravidla tak lze rychle najit kvalifikovanou odpověď a současně se naučit rozpoznat, co je kvalifkovaná odpověď.

Stejně dobře se můžete radit nebo si nechat napovědět od nějakého AI systému, od obecného jako\ GPT-3/4/5, nebo specializovaného jako GitHub Copilot nebo Claude. Ten vám také dokáže naprogramovat jednoduché úkoly. Může vám bez problémů pomoci s domácími úkoly. I když je volíme tak, aby nebyly pro GPT lehké, nemá úplně smysl bojovat proti větru.

Jak se učit programovat s AI

- Zkuste, pokud to jde, programovat sami. Vypněte AI asistenta a namísto sledování obsedantně generovaného kódu se soustřeďte na to, co chcete naprogramovat a jak to udělat efektivně, robustně a čistě.
- Dobře si prostudujte, co AI napsalo. Nechte si zobrazit vysvětlení a studujte idiomy. Popřemýšlejte, zda by nešlo kód vylepšit (velice často to jde)
- Pro většinu domácích úkolů dostanete dobré řešení tak, že zkopírujete zadání do komentáře. Všimněte si, že zadání jsou formulována tak, aby byla výstižná a vyčerpávající, takže jsou nejen pro vás, ale i pro AI dobře pochopitelné. Dělejte to taky tak: když napíšete dobré zadání, ono samo bude - kromě kódu - dalším užitečným produktem vaši práce. Naučíte se přitom, že AI někdy nerozumí věcem, které lidem připadají samozřejmé.
- Naučte se, pro jaká zadání je AI opravdu užitečné: věci, které jsou pracné, ale opravdu se vám kvůli nim nechce narychlo studovat další dva Pythonské moduly. Třeba jak vyextrahovat informace z hromady pdf souborů. Libovolný, třeba špinavý a neoptimalizovaný kód vám ušetří kopu otravné práce.

Klidně se ptejte i mě, v průběhu cvičení, nebo e-mailem (*peter.kvasnicka@matfyz.cuni.cz*), nebo mi zavolejte a dohodneme se na konzultaci (605 386 052).

Podmínky zápočtu

Pro získání zápočtu budete v tomto semestru muset splnit **několik podmínek**.

1. **Domácí úkoly**

Budete dostávat domácí úkoly a odevzdávat je přes *ReCodEx*.

Budu zadávat dvě úlohy týdně, v den cvičení (úterý) s termínem pondělí 23:59.

Pro zápočet budu požadovat **70% správných odevzdaných domácích úkolů**.

2. **Zápočtový test**

Zápočtový test bude úkol s náročností běžného domácího úkolu, který budete muset vyřešit v učebně v určeném časovém limitu (90 minut).

3. **Zápočtový program**

Zápočtový program má být samostatný a kompletní softwarový produkt, tedy aplikace nebo knihovna, která má stanovenou funkcionalitu, má dobře napsaný kód, je otestovaná a validovaná, a je vybavená dokumentací technickou a uživatelskou dokumentací.

O tom, jaký bude váš zápočtový program, se začneme bavit v dubnu, a před koncem letního semestru bychom měli dospět ke specifikaci programu. První funkční verzi odevzdáte nejpozději v polovině září, abyste mohli do konce září získat zápočet.

Podrobnější informace naleznete zde*.

4. **Zápočet z teoretického cvičení**

   Nakonec budete také muset splnit podmínky pro udělení zápočtu z teoretického cvičení, tak jak vám je stanoví dr. Forstová.

Instalace Pythonu

Na počítačích v učebně byste měli najít vše potřebné. Pokud ne, *stěžujte si*.

Na vlastních počítačích máte vícero možností a nechám na váš výběr, kterou si zvolíte.

1. **Základní distribuce Pythonu**

   Stáhněte si instalátor pro svůj systém tady: https://www.python.org/downloads/.

   Zvolte si nejnovější verzi 3.14. Součástí je vlastní interpret a jednoduché IDE *Idle*.

2. **Anaconda**

   Toto je velká distribuce, která obsahuje rozsáhlou podporu pro využití Pythonu ke zpracování dat, strojové učení a pod. Stáhnete si ji tady: https://www.anaconda.com/products/individual a zabere vám docela hodně místa na disku. Součástí je i vyspělé IDE pro vývoj v Pythonu - *Spyder*.

3. **Google Colab notebooky**

   Nemusíte nic instalovat, stačí jít na colab.google.com a začít psát kód do notebooku.

4. **JupyterLab**

   JupyterLab existuje jako samostatná aplikace a lze ji bez externího serveru provozovat na vašem počítači. Pracuje s Jupyter notebooky a je velice jednoduchá na používání.

IDE pro Python

Existuje několik programovacích editorů a vývojových prostředí pro Python, například *PyCharm*, *VSCode*, Zed, atd. Nové generace IDE obsahují také AI asistenty.

V tomto semestru nám už Idle nebude stačit, budeme psát složitější kódy a budeme dělat těžší chyby, takže se naučíme i nějaké techniky ladění kódu. Na přednáškách budu používat -*PyCharm* a *VSCode*, abyste si na obě prostředí zvykli. Uvidíte ale, že pro množství kratších kódů je výhodnější "lehké" vybavení pozůstávající z nástroje pro správu virtuálního prostředí (např. *uv*) a lehkého editoru (*Zed* nebo *Neovim*).

ReCodEx

Zaregistrujte se prosím v ReCodExu, https://recodex.mff.cuni.cz/. Můžete použít svoje přístupové údaje do SISu. Pak se prosím **zaregistrujte do skupiny pro toto cvičení**, můžete tak učinit volbou "SIS integration".

V ReCodExu najdete své domácí úkoly a budete je tady i odevzdávat.

Platí to, co jste se o ReCodExu loni na vlastní kůži zjistili: ReCodEx někdy odmítá uznat řešení, které je podle všeho správné. V takovém případě požádejte o revizi anebo mi pošlete e-mail. Stejně postupujte, pokud si s nějakým úkolem nevíte rady.

První domácí úkoly najdete v ReCodExu po skončení tohoto cvičení. V dalších týdnech budou domácí úkoly zveřejněné v době začátku cvičení.

Co jsme se naučili v předchozím semestru

Samozřejmě jste udělali první kroky v Pythonu - tedy alespoň ti z vás, kteří je neudělali už dříve. Získali jsme také ale několik magických schopností:

- práce se seznamy a znakovými řetězci
- třídění
- binární vyhledávání
- rekurzivní funkce
- generátory
- dekorátory
- (velice nekompletní) třídy

V tomto semestru tento seznam rozšíříme o nové skilly.

Pro rozcvičení

Součet dvou čísel

Na vstupu načteme dvě celá čísla jako znakové řetězce. Na výstupu má váš kód vypsat jejich součet, ale máte povolenou sečítat pouze číslice 0-9, ne celá čísla. Můžete si představit, že pro sčítání máte k dispozici tabulku typu

	0	**1**	**...**	**8**	**9**
0	(0, 0)				
1	(0, 1)	(0, 2)			
2	(0, 2)	(0, 3)			
...					

	0	1	...	8	9
8	(0, 8)	(0, 9)		(1, 6)	
9	(0, 9)	(1, 0)		(1, 7)	(1, 8)

kde první číslo je přenos a druhé jsou jednotky součtu. (Pořadí je diktované tím, co nám dá funkce `divmod`, abychom nemuseli přehazovat dvojice.)

> **Note**
>
> Tato úloha není úplně nesmyslná: Python používá celá čísla s prakticky neomezenou délkou a musí tedy operace mezi nimi vykonávat po kouskách.

O co nám půjde?:

- Jak to naprogramovat, aby kód byl hezký a rozumně efektivní? ("uzavřené" cykly: namísto `for` a `while` atd.)
- Funkce `zip` a funkce `itertools.zip_longest`
- Oplatí se pro takováto čísla (uspořádané seznamy číslic) zavést třídu?
- Samozřejmě budeme chtít také odečítat, násobit a dělit. Umíme vytvořit nějaký sjednocující algoritmus?

Zkusíme jednoduché věci:

1. Načtení čísla

```
digits = [int(c) for c in input()]
```

(Začneme jednodušeji a provedeme konverzi na int, i když s tabulkou to ani není třeba)

2. Seřazení čísel pod sebe se zarovnáním vpravo a průchod dvojicemi číslic ležícími nad sebou:

```
from itertools import zip_longest
```

Itertools.zip_longest()

This iterator falls under the category of Terminating Iterators. It prints the values of iterables alternatively in sequence. If one of the iterables is printed fully, the remaining values are filled by the values assigned to `fillvalue` parameter. **Syntax:**

```
zip_longest( iterable1, iterable2, fillval)
```

```
zip_longest(reversed(digits1), reversed(digits2),0)
```

Díky tomuto se taky nemusíme starat o to, které číslo je delší.

4. Sčítání s přechodem:

```
def add_and_carry(d1:int, d2:int) -> {int, int}:
    return divmod(d1 + d2, 10)
```

Tuto funkci nebudeme potřebovat, protože si můžeme předem uložit výsledky do tabulky:

```
add_table = {(i, j) : divmod(i + j, 10) for i in range(10) for j in range(10)}
```

Nakonec to všechno můžeme poskládat:

```
from itertools import zip_longest

# Sestrojíme slovník pro sčítání číslic
# Klíče: dvojice číslic (0-9). Hodnoty: dvojice (přenos, součet % 10
add_table = {(i, j) : divmod(i + j, 10) for i in range(10) for j in range(10)}

# Načteme dvě čísla jako seznamy číslic
a = [int(x) for x in input()]
b = [int(x) for x in input()]

result = []
carry = 0
for da, db in zip_longest(reversed(a), reversed(b), fillvalue=0):
    carry, digit_sum = add_table[(da, db)]
    result.append(digit_sum)
if carry > 0:
    result.append(carry)

a_print = "".join(map(str, a))
b_print = "".join(map(str, b))
result_print = "".join(map(str, reversed(result)))
print(f"{a_print} + {b_print} = {result_print}")
```

Toto je funkční kód a najdete ho v repozitáři jako `code/Ex01/soucet.py` .

Ještě bychom měli odstranit důležitou vadu: otevřený cyklus `for` . Měli bychom ho "zapouzdřit", aby běžel v C a ne v Pythonu.

Na výstupu budeme mít dvojici (přechod, součet). Jak ale budeme takovéto dvojice sečítat v cyklu?

```
from itertools import accumulate
```

itertools.**accumulate**(*iterable*[, *func*, *, *initial=None*])

Make an iterator that returns accumulated sums, or accumulated results of other binary functions (specified via the optional *func* argument).

If *func* is supplied, it should be a function of two arguments. Elements of the input *iterable* may be any type that can be accepted as arguments to *func*. (For example, with the default operation of addition, elements may be any addable type including `Decimal` or `Fraction`.)

Usually, the number of elements output matches the input iterable. However, if the keyword argument *initial* is provided, the accumulation leads off with the *initial* value so that the output has one more element than the input iterable.

Funkce `accumulate` bude postupně zpracovávat dvě dvojice číslic:

- (přenos, aktuální součet)
- dvojice dalších číslic

a z nich vytvoří novou dvojici (přenos, nový součet). Jak to uděláme?

```
def add_and_carry(accumulator: tuple[int, int], digits: tuple[int, int]) -> tuple[int, int]:
    """
    Přičte součet dvou dalších číslic a přenosu, vrací aktualizovaný součet a přenos.
    :param accumulator: tuple (přenos, dosavadní součet)
    :param digits: dvojice nových číslic
    :return: tuple (přenos, aktualizovaný součet)
    """
    carry, old_sum = accumulator
    a, b = digits
    new_sum = a + b + carry # tady bychom měli použít tabulku. Jak to zařídit?
    return divmod(new_sum, 10)  # returns (carry, digit_sum)
```

Teď už je lehké dát všechno dohromady:

```
from itertools import accumulate, zip_longest


def add_and_carry(accumulator: tuple[int, int], digits: tuple[int, int]) -> tuple[int, int]:
    """
    Přičte součet dvou dalších číslic a přenosu, vrací aktualizovaný součet a přenos.
    :param accumulator: tuple of the running sum and carry
    :param digits: tuple of two new digits to add
    :return: tuple of updated sum and carry
    """
    carry, old_sum = accumulator
    a, b = digits
    new_sum = a + b + carry
    return divmod(new_sum, 10)  # returns (carry, digit_sum)
```

```
def sum_by_digits(a: list[int], b: list[int]) -> list[int]:
    """
    Sums two integers given as lists of digits.
    :param a0: list of digits of the first number
    :param b0: list of digits of the second number
    :return: list of digits of the sum
    """
    tuples = list(accumulate(zip_longest(a, b, fillvalue=0), func=add_and_carry, initial=
(0,0)))
    tuples.reverse()
    tuples.pop()    # remove initial state
    carry = tuples[0][0]
    result = [carry] if carry != 0 else []
    result += [v for u, v in tuples]
    return result


def main() -> None:
    """
    Demonstrates the sum_by_digits function.
    """
    a = [int(c) for c in input()]
    b = [int(c) for c in input()]
    print("First number: ", *a, sep = "")
    print("Second number: ",*b, sep = "")
    digit_sum = sum_by_digits(a, b)
    print("Sum: ", *digit_sum, sep="")
    print("Check:")
    a_int = int("".join([str(d) for d in a]))
    b_int = int("".join([str(d) for d in b]))
    print("Int addition: ", a_int + b_int)


if __name__ == "__main__":
    main()
```

Náměty pro vlastní cvičení:

- Jak implementovat násobení a (dlouhé) dělení
- Jak to pak dát všechno dohromady?
- Umíte vysvětlit AI, co má naprogramovat, abyste dostali kód podle vaší představy?

Domácí úkoly

1. **Inverze slovníku**: Přeorganizovat slovník tak, že se z klíčů stanou hodnoty a z hodnot klíče.
2. **Pozice prvků ve sloučené posloupnosti**: Máme dvě setříděné posloupnosti a chceme je sloučit do jedné. Máte vypsat indexy ve výsledné posloupnosti, na kterých se octnou jednotlivé prvky.
