Programování 2

1. cvičení, 20-2-2024

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 jsme 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. 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 je **zeptat se lámanou angličtinou Googlu**. Zpravidla rychle najdete kvalifikovanou odpověď.

Stejně dobře se můžete radit s GPT-3/4. Ten vám také dokáže naprogramovat jednoduché úkoly. Může vám také pomoci s domácími úkoly, ty ale volíme tak, aby nebyly pro GPT lehké.

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). Pohotově se zeptat je schopnost, která vám může ušetřit kopu času a zbytečné frustrace, takže je dobré ji kultivovat. Žádná otázka není hloupá.

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*.

Tak jako v minulém semestru se budu snažit zadávat tři úlohy týdně.

Pro zápočet budu požadovat **60% 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 Doc. Töpfer.

Instalace Pythonu

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

Tady máme 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 3.9. 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*, *Atom*, *Sublime Text*, atd.

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.

ReCodEx: systém pro kontrolu domácích úloh

Zaregistrujte se v ReCodExu, https://recodex.mff.cuni.cz/. Můžete použít svovje 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 jsme si o ReCodExu loni řekli a co jste mnozí na vlastní kůži zjistili: pokud se vám zdá, že váš kód je správný, ale ReCodEx je k vám nespravedlivý, nahrajte řešení a napište buď komentář k úloze anebo mi pošlete e-mail.

První úlohy najdete v ReCodExu dnes večer.

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
- (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	(1, 0)	(2, 0)			
2	(2, 0)	(3, 0)			
...					
8	(8, 0)	(9, 0)		(6, 1)	
9	(9, 0)	(0, 1)		(7, 1)	(8, 1)

kde první číslo jsou jednotky součtu a druhé číslo je "přenos".

Poznámky

- 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()]
```

2. Iterátor, posílající číslice v opačném pořadí:

```
def get_digits_reversed(digits):
    yield from reversed(digits)
```

3. Dvojice číslic pozpátku:

```
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(get_digits_reversed(digits1), get_digits_reversed(digits2),0)
```

4. Sečítání s přechodem:

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

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 a z nich vytvoří novou dvojici. Jak to uděláme?

5. Teď už je lehké dát všechno dohromady:

```
from itertools import zip_longest

def get_digits_reversed(digits):
    yield from reversed(digits)

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

digits1 = [int(c) for c in input()]
digits1 = [int(c) for c in input()]
...
```

A co kdybychom si namísto počítání vytvořili tabulku a jenom z ní odčítali součty a přenosy?