Korpus studentských zdrojových souborů

kvantifikace syntaktické podobnosti

Jiří Fišer, Jiří Škvor, Pavel Beránek

Katedra informatiky PřF UJEP v Ústí nad Labem

QCorpus

- **korpus zdrojových kódů** vytvořených studenty středních škol a studentů prvního ročníku našeho bakalářského studia
- statistické zpracování resp. využití ML modelů pro klasifikace

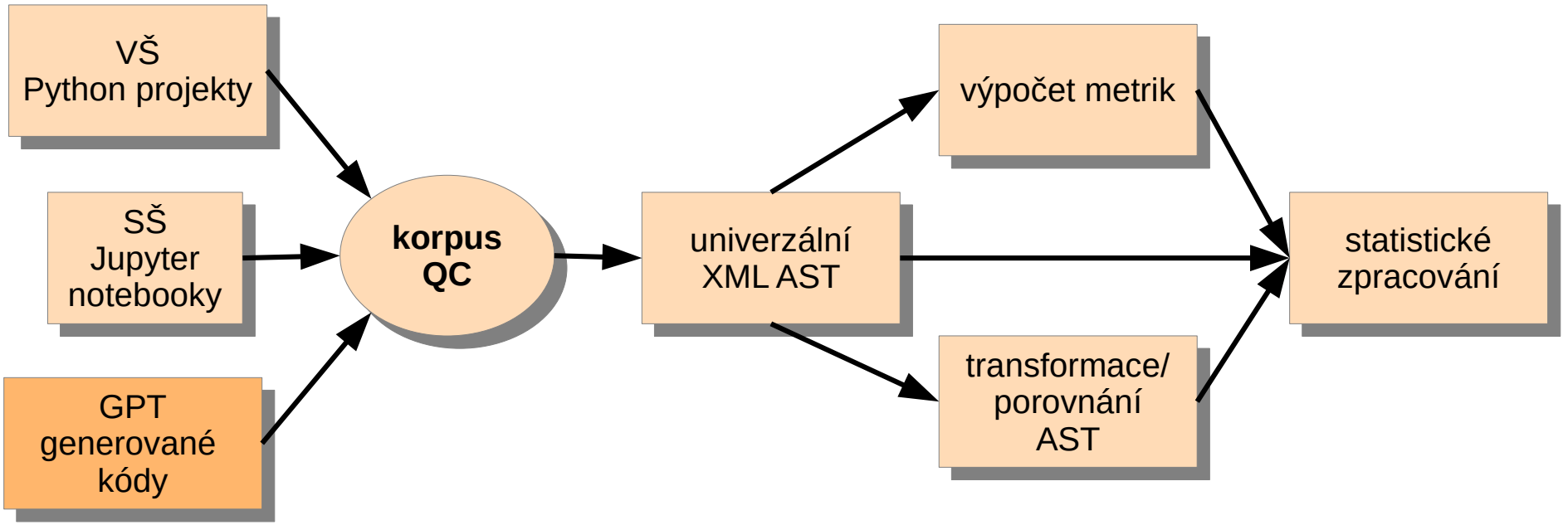
Cvičení 5.1.6: Přepište desetinná čísla na procenta

Následující kód je kód studenta, který se neúspěšně snažil o změnu desetinných čísel v kolekci na hodnotu v procentech. Přepište kód tak, aby fungoval. První hodnota v kolekci (0.01) se má přepsat na 1, druhá hodnota 0.15 se má přepsat na 15.

```
data = [0.01, 0.15, 0.95, 0.34, 0.65, 0.71]
for idx in range(len(data)):
  data[idx] *= 100
print(data)
```

```
[1.0, 15.0, 95.0, 34.0, 65.0, 71.0]
```

Procesy a data v QCorpusu



Umělá inteligence a generování kódů

- velké jazykové modely (LLM) jsou podstatou schopny generovat i zdrojové kódy
  programovací jazyky jsou lidské jazyky
  Generative pre-trained transformer foundational modely vytvářené Open AI jsou trénované i na zdrojových kódech
- první použití (2021): OpenAI Codex (modifikovaný GPT-3) pro Github Copilot
- ChatGPT (listopad 2022): zdarma, jednoduché rozhraní, dostatečný výkon, překonává schopnosti většiny studentů v cílových skupinách

ChatGPT — partner, který vám rozumí

Vytvoř v Pythonu funkci, která přijme seznam objektů a vrátí počet tříd ekvivalence podle operátoru ==.

```
def count_equivalence_classes(objects):
    equivalence_classes = {}  # Slovník pro uchování tříd ekvivalence

    for obj in objects:
        found = False
        for cls, instances in equivalence_classes.items():
            if obj == instances[0]:  # Porovnáme s prvním objektem v třídě
                instances.append(obj)
                found = True
                break
        if not found:
            equivalence_classes[obj] = [obj]

    return len(equivalence_classes)
```

Ale nerozumí matematice

Vytvoř v Pythonu funkci, která očekává seznam racionálních čísel a vrací *True* tehdy, pokud pro všechny z nich platí, že pokud na ně aplikujeme funkci *math.log*, tak nevyvolají výjimku *ValueError*.

```
import math

def check_log_values(numbers):
    for num in numbers:
        try:
            math.log(num)
        except ValueError:
            return False
    return True
```

Vytvoř v Pythonu funkci, která přijímá seznam řetězců a vrací *True* pokud platí, že i-tý řetězec je podřetězcem j-tého řetězce, pro všechna j > i.

```
def check_all_substrings(strings):
    n = len(strings)
    for i in range(n):
        for j in range(i + 1, n):
            if strings[i] not in strings[j]:
                return False
    return True
```

Ale přesto je užitečný

```python
import requests

def check_freezing_in_prague(api_key):
    base_url = "http://api.openweathermap.org/data/2.5/weather"
    city_name = "Prague"
    units = "metric"  # Pro teplotu ve stupních Celsia

    params = {"q": city_name, "units": units, "appid": api_key}

    response = requests.get(base_url, params=params)
    data = response.json()

    if response.status_code == 200:
        temperature_celsius = data["main"]["temp"]
        if temperature_celsius <= 0:
            return True
        else:
            return False
    else:
        print("Chyba při získávání dat:", data.get("message", "Neznámá chyba"))
        return False
```

Při chybě získání dat by bylo lepší vyhodit výjimku nikoliv vracet False.

```
message = data.get("message", "Neznámá chyba")
raise WeatherDataError(f"Chyba při získávání dat: {message}")
```

Důsledky AI jazykových modelů na výuku programování

výuka programování se musí změnit

- **porozumění zdrojovému kódu** (a tím i koncepci konkrétních programovacích jazyků)
- **schopnost přesně definovat cíl** (což mnohdy znamená matematizovat zadání)
- **schopnost identifikovat nedostatky** v generovaném kódu
  - nevhodný algoritmus
  - neoptimální kód
  - nedodržení vstupních požadavků (i okrajové případy)

Univerzální XML representace

- XML representace využívající univerzální stavební prvky **procedurálních jazyků**
- **XML prvky** = obecné konstrukce (cykly, přiřazovací příkazy, definice procedur, výrazy)
- sémantika prvků je zpřesněna pomocí XML atributů
- jednoduchý model s důrazem na výuku (ukrytí impl. detailů)
- hlavní cíl: výpočet metrik pomocí jediné implementace pro větší počet procedurálních jazyků hlavního proudu (Python, C#)

Ukázka univerzální representace AST

```
<module xmlns="http:/ki.ujep.cz/ns/qc_ast" ...>
 <command type="assignment">
  <left>
   <qc:expression><variable name="data"/></qc:expression>
  </left>
  <right>
   <qc:expression>
    <collection type="list">
     <literal type="float" value="0.01"/><literal type="float" value="0.015"/>
    </collection>
   </qc:expression>
  </right>
 </command>
 <loop type="foreach" target="idx">
  <iterable>
   <qc:expression>
    <call name="range" complex="false">
        <arguments><argument type="positional" unpacking="false">
```

```
data = [0.01, 0.015]
for idx in range(len(data)):
    data[idx] *= 100
print(data)
```

Problémy univerzální representace

- (neprocedurální) ideosynkrazie v jádře jazyků hlavního proudu

  `x.reverse().index(0)`

  základní interpretace: `index(reverse(x), 0)`

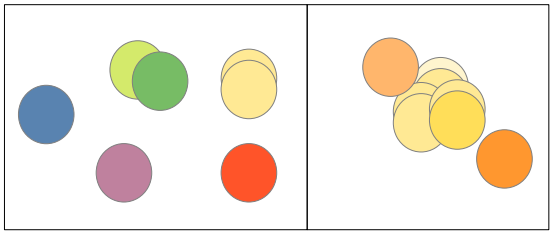
  ≈ proud. zpracování: `x |> reverse() |> index(0)`

- skutečná representace: [[x.reverse]().index](0) = využití komplexních uzávěrů obdoba zápisu: f(x)(y)

  `a.m(x)` – versus – `math.sin(x)`

Testování podobnosti v QCorpusu

- **cílem není odhalení plagiátů (klonů)**
- výsledky porovnání jsou vstupem do procesu zpracování dat (shluková analýza) či předzpracování dat
- podobnost je standardní stav, studenti sdílejí řešení či ke stejnému řešení dospějí nezávisle => **odhalení netradičních řešení**
- využití univerzální XML representace (optimálně jedno řešení pro celou třídu programovacích jazyků, křížová porovnání)



Typy podobnosti zdrojových kódů

Typ 1: (identické klony)

zdrojový kódy, které jsou navzájem identické po normalizaci mezer, odsazení, eliminaci poznámek, apod.

Typ 2: (parametrizované klony)

zdrojové kódy, které jsou identické až na sémanticky irelevantní záměnu jmen identifikátorů, obsahu řetězců

Typ 3: (strukturální klony)

permutace příkazů, přidání či eliminace příkazů (v omezeném rozsahu bez změny struktury)

Typ 4: (sémantické klony) stejná sémantika (měkký vstupní předpoklad)

Sémantická transparence

- **identifikátory proměnných, metod, tříd apod.**

  vnitřní identifikátory: lokálně (studentem) definované

  vnější identifikátory: importované z knihoven

  Lze je určit? Lze omezit počet testovaných permutací?
- **literály** (primárně řetězcové)

  vnitřní: řídící sémantiku programu

  vnější: ovlivňující formát vstupu/výstupu

  Lze je odlišit? (kam patří formátovací řetězce?) Lze je určit?

Kvantifikace syntaktické podobnosti

- podobnost dvou zdrojových kódů není nemusí být ekvivalencí (nemusí splňovat tranzitivnost)
- podobnost může být **fuzzy relací**
- podobnost může být **metrikou**

  Jak ji definovat? Existuje objektivní definice?

Malý dotazník

Slido.com

s kódem #2597968