SOWID – EDYCJA TEKSTU I STRUKTURY DANYCH.

PAWEŁ JÓZIAK
WYDZIAŁ MATEMATYKI I NAUK INFORMACYJNYCH
POLITECHNIKI WARSZAWSKIEJ

- Nigdy nie wykonuj poleceń metodą *copy&paste*. Przepisz je ręcznie!
- Nie redukuj pracy nad zadaniem do przeczytania go. Wykonaj je i upewnij się, że rozumiesz jak i dlaczego wynik jest taki, jak na ekranie. Zajrzyj do prezentacji bądź zapytaj prowadzącego, jeśli coś nie będzie jasne.
- Linia zaczynająca się od $ reprezentuje prompt zwykłego użytkownika (nie przepisuj go!). Komendą jest to, co następuje po nim.
- Ostrożnie ze spacjami: nie dodawaj ich więcej ani nie usuwaj takowych, jeśli są!

1. Serializacja

(a) Sprawdź, jak w różnych metodach serializacji można zserializować obiekt `null` (aka `None`). Ile bajtów jest do tego potrzebne, w zależności od metody?[1]

(b) Spróbuj zserializować prosty słownik w pythonie, za pomocą
- `pickle`
- `marshal`
- `msgpack`
- `json`
- `yaml`
- `dicttoxml`

Która metoda jest najefektywniejsza? Która jest najmniej efektywna?

2. AWK/VIM

(a) Wprowadź do terminala poniższą instrukcję:
`$ awk '{print "Witaj w instruktażu komend awk"}'`
Jak on działa?[2] Jak go zakończyć?[3]

(b) Jaką opcją można nastawić separator pól?[4]? Jak pewnie pamiętasz, wiele plików konfiguracyjnych UNIXa korzysta z `:` jako separatora pól. Spróbuj:
`$ awk -F: '{print $1}' /etc/passwd`
Jak on działa?[5] Zmodyfikuj powyższy przykład, by uzyskać podobny efekt dla pozostałych istotnych informacji, np. grupy podstawowe.

---

[1] `https://en.wikipedia.org/wiki/Comparison_of_data-serialization_formats` albo ręcznie:

```
$ python -m venv lab8venv && source lab8venv/bin/activate && pip install msgpack pyyaml
$ cat test_serialization.py
import marshal; import pickle; import msgpack; import json; import yaml;
dump_info = {".pkl": ("wb", pickle.dump), ".mkl": ("wb", marshal.dump), ".mpk": ("wb",
msgpack.dump), ".json": ("w", json.dump), ".yml": ("w", yaml.dump)};
for extension, (mode, dump_fun) in dump_info.items():
  with open("null"+extension, mode) as fh:
    dump_fun(None, fh)
$ python3 test_serialization.py
```

[2] Dla każdej linii *stdin* sprawdzany jest domyślny warunek (prawda), a potem wykonywana instrukcja wypisująca *Witaj w instruktażu komend awk.*

[3] Albo <CTRL> <C> (`SIGINT`) albo <CTRL> <D> (`EOF`)

[4] `-F`

[5] Wypisuje pierwsze pole – tj. `username` – każdego wpisu do `/etc/password`

(c) Przygotuj poniższy jednolinijkowy skrypt, i zapisz go jako `testscript`:
`{print $1 " home at " $6}`
Dlaczego w powyższym pliku używamy nawiasów klamrowych { i }?[6] Odpal
`$ awk -F: -f testscript /etc/passwd`
Jaki jest wynik działania skryptu?[7]

(d) Napisz skrypt awk, który będzie parsował zawartość plików jak `/etc/passwd`, lecz będzie je wyświetlał jako `tsv` (*tab-separated values*), tj.: zmieni separator z dwukropka (`:`) na tabulację (`\t`).[8]

(e) Użyj sekcji `BEGIN` żeby napisać skrypt działający jak `cat`, lecz dopisujący nagłówek o treści: *Raport treści*[9]

(f) Zmodyfikuj powyższy skrypt, aby uzyskać efekt pisania również numeru linii przed linią.[10] A na końcu niech pojawi się zsumaryzowana liczba słów.[11]

(g) Rozszerz powyższy skrypt dalej: spraw, aby w nagłówku pojawił się napis informujący o nazwie pliku (np. *Raport treści plik.txt*).[12]

(h) Napisz skrypt awk (którego później użyjesz do plik `/etc/passwd`), który wyświetli informacje jak w pytaniu 2(c), lecz jedynie te linie będą wyświetlone, w których pole informujące o Powłoce zawiera faktyczną powłokę[13]

(i) Drugi raz to samo: wyświetl tylko informację o faktycznych użytkownikach (nie kontach systemowych)[14]

(j) Zapisz poniższe dane do pliku `teamlist.csv`:

```
Name,Team,First Test,Second Test,Third Test
Tom,Red,5,17,22
Joe,Green,3,14,22
Maria,Blue,6,18,21
Fred,Blue,2,15,23
Carlos,Red,-1,15,24
Phuong,Green,7,19,21
Enrique,Green,3,16,20
Nancy,Red,9,12,24
```

Napisz skrypt `awk`, który policzy średni wynik każdego ucznia, średni wynik każdego testu oraz średni wynik każdego zespołu (Team). Uwaga: ujemni wynik reprezentuje, że osoba była nieobecna na danym teście, i wynik nie powinien być brany pod uwagę przy liczeniu średnich.

---

[6] Bo to instrukcja, nie warunek.

[7] Dla każdego wpisu do `/etc/passwd` wypisze linijkę typu:
```
root home at /root
joziakp home at /home2/samba/joziakp
```
itd

[8] `BEGIN {FS=":"; OFS="\t"} {$1=$1; print $0}`

[9] `$ awk 'BEGIN {print "Raport treści"} {print $0}' plik.txt`

[10] sama instrukcja: `{print NF, $0}`.

[11] całościowy skrypt:
```
BEGIN {words=0; print "Raport treści"}
  {words+=NF; print NR, $0}
END {print "Liczba słów", words}.
```

[12] całościowy skrypt:
```
BEGIN {words=0; print "Raport treści", ARGV[1]}
  {words+=NF; print NR, $0}
END {print "Liczba słów", words}.
```

[13] Można
- Napisać wyrażenie regularne na jedną z dostępnych powłok:
  `$7 ~ /\/bin\/((ba|c|z|tc|k|z)?sh|mini)/`
- Napisać wyrażenie regularne na jeden z typowych programów nie-powłok:
  `$7 ~ /\/(usr\/s)?bin\/(false|nologin|sync)/`

[14] `$3>=1000`, czyli UID

Wynik wypisz w następujący sposób: w pierwszej linii, imiona powinny być wyjustowane do lewej w polu szerokości 10[15], zaś średnie powinny być wpasowane w pole o siedmiu cyfrach wiodących, kropce i 2 miejscach dziesiętnych[16].

3. JQ

(a) Dane z przykładu 2j można wyreprezentować jako JSON na dwa sposoby: listę rekordów z kluczami jak kolumny, albo słownik kolumn z listą wartości. Przygotuj pliki pomocnicze na te 2 sposoby, zapisując do plików `teamlist_records.json` oraz `teamlist_values.json`.

```
$ cat teamlist_records.json
[{"Name": "Tom", "Team": "Red",
"First Test": 5, "Second Test": 17, "Third Test": 22},
{"Name": "Joe", "Team": "Green",
"First Test": 3, "Second Test": 14, "Third Test": 22},
{"Name": "Maria", "Team": "Blue",
"First Test": 6, "Second Test": 18, "Third Test": 21},
{"Name": "Fred","Team": "Blue",
"First Test": 2, "Second Test": 15, "Third Test": 23},
{"Name": "Carlos", "Team": "Red",
"First Test": -1, "Second Test": 15, "Third Test": 24},
{"Name": "Phuong", "Team": "Green",
"First Test": 7, "Second Test": 19, "Third Test": 21},
{"Name": "Enrique", "Team": "Green",
"First Test": 3, "Second Test": 16, "Third Test": 20},
{"Name": "Nancy", "Team": "Red",
"First Test": 9, "Second Test": 12, "Third Test": 24}]
$ cat teamlist_values.json
{"Name": ["Tom", "Joe", "Maria", "Fred", "Carlos", "Phuong", "Enrique", "Nancy"],
"Team": ["Red", "Green", "Blue", "Blue", "Red", "Green", "Green", "Red"],
"First Test": [5, 3, 6, 2, -1, 7, 3, 9],
"Second Test": [17, 14, 18, 15, 15, 19, 16, 12],
"Third Test": [22, 22, 21, 23, 24, 21, 20, 24]}
```

(b) Napisz komendę `jq`, dzięki której dokonasz agregacji wszystkich wyników pierwszego testu, raz używając jednego formatu danych, raz używając drugiego formatu danych.

(c) Napisz komendę `jq`, dzięki której dokonasz agregacji wyników drużyny *Red* – raz używając jednego formatu danych, raz używając drugiego formatu danych.

(d) Spróbuj napisać komendę `jq`, która przetransformuje jedną reprezentację danych JSONowych w drugą.

4. YQ

(a) Sprawdź, że wszystkie instrukcje `jq`, które rozwiązywały podpunkty zadania 3, działają też dla `yq`.

(b) Jak wyświetlić plik `.json` w standardowej reprezentacji `.yaml`?

5. SED

(a) Użyj wcześniej przygotowanego pliku `teamlist.csv` i spróbuj go spolszczyć (*Red* → *Czerwoni*, *Green* → *Zieloni* etc).

(b) Spróbuj odtworzyć funkcjonalności `cat`, `head`, `grep` za pomocą instrukcji `sed`, podobnie jak zrobiliśmy to dla `awk`.

[15] `printf "%-10s", var`
[16] `printf "%7.2f", var`