Zadania rekrutacyjne Python | Praktyki 2019

Wymagania ogólne:

Rozwiązania powinny działać w Pythonie 3.5 w środowisku Ubuntu 16.04 lub 18.04 (dozwolona jest instalacja bibliotek pythonowych przez odpowiednie `requirements` w `setup.py`). Niedozwolona jest konieczność ręcznego instalowania bibliotek systemowych (przy pomocy apt, curl itp.).
Zadania będą testowane przede wszystkim automatycznie, dlatego bardzo ważne jest zadbanie o trzymanie się dokładnie treści poleceń.
Zabronione jest współpracowanie z innymi osobami przy rozwiązywaniu zadań.
Kod musi być odpowiednio udokumentowany oraz czytelny (komentarze, docstringi, `PEP8`).
Przy ocenie brane pod uwagę są:

- działanie zgodne ze specyfikacją
- czytelność, poprawność, złożoność obliczeniowa, reużywalność kodu (jeśli ma uzasadnienie)

Rozwiązania zadań od 1 do 3 należy spakować do archiwum zip o nazwie `nazwisko_imie_notifai.zip` (np. `nowak_jan_notifai.zip`) oraz wysłać je w odpowiedzi na maila w którym otrzymałeś(aś) ten dokument.
Uwaga, zadanie 4 wymaga wypełnienia formularza internetowego.
Ze względu na dużą liczbę kandydatów, tylko część osób będzie mogła być zaproszona na rozmowę. Wszystkie osoby, które prześlą zadania, a nie zostaną zaproszone na rozmowę, otrzymają raport z wykonanych automatycznych testów.
Zbiór zadań celowo jest w miarę przekrojowy. Zachęcamy do wysłania rozwiązań każdego, kto zrobi co najmniej połowę podpunktów.
Osoby zaproszone na rozmowę będą musiały uzasadnić swoje rozwiązania.

Zadanie 1

Kod rozwiązania umieść w **folderze** `zadanie1`.

Napisz serwis internetowy „todo lista” w dowolnym frameworku webowym (`Flask`, `Bottle`, `Pyramid`, `Django` itp.) z wykorzystaniem bazy sqlite, który:

a) Posiada widok dostępny pod adresem `` `/todolist` `` z metodą `GET`, który w postaci JSONA zwraca listę wcześniej zapisanych w bazie sqlite „zadań”:

```
[
  {
    "id": 1,
    "title": "Task name 1",
    "done": true,
    "author_ip": "123.45.67.89",
    "created_date": "2019-05-08 10:00:00",
    "done_date": "2019-05-09 15:44:55"
  },
  {
    "id": 2,
    "title": "Foobar",
    "done": false,
    "author_ip": "66.55.44.33",
    "created_date": "2019-05-08 18:12:33",
    "done_date": null
  },
  {
    "id": 6,
    "title": "Buy a PyCharm license",
    "done": true,
    "author_ip": "100.200.192.168",
    "created_date": "2019-05-09 12:11:22",
    "done_date": "2019-05-09 14:33:44"
  },
  {
    "id": 9,
    "title": "Learn Python",
    "done": false,
    "author_ip": "132.165.98.66",
    "created_date": "2019-05-10 06:59:59",
    "done_date": null
  }
]
```

Elementy reprezentujące pojedyncze zadanie powinny mieć strukturę:

```
{
    "id": 6,                                        identyfikator zadania w bazie
    "title": "Buy a PyCharm license",                                nazwa zadania
    "done": true,                              czy zadanie zostało zrobione (t/f)?
    "author_ip": "100.200.192.168",                        adres ip osoby tworzącej
    "created_date": "2019-05-09 12:11:22"           data utworzenia (strefa UTC)
    "done_date": "2019-05-09 14:33:44"     data oznaczenia jako done (strefa UTC)
  }
```

b) Posiada widok `/todolist` z metodą `POST`, który przyjmuje na wejściu JSON w formacie takim jak poniżej:

```
{
    "title": "Conquer the world",
    "done": false,
    "done_date": null
}
```

Widok ma:

dodać zadanie o tytule podanym w polu `title` (to jedyny wymagany parametr)

oznaczyć je jako zrobione lub niezrobione zależnie od wartości pola `done`

ustawić adres IP requestera jako `author_ip`

Parametry `done` i `done_date` są opcjonalne.

w przypadku niepodania pola `done`, należy ustawić zadanie jako niewykonane.
w przypadku podania `done: true` oraz `done_date`, należy ustawić obie wartości w bazie.
w przypadku podania `done: true` oraz niepodania `done_date`, należy użyć bieżącego czasu jako `done_date`.
w przypadku podania `done: false` oraz `done_date` innego niż `null`, należy zwrócić kod `HTTP 400`.

Widok ma zwracać JSONa o treści takiej jak poniższa:

```
{
    "task_id": 5
}
```

gdzie w polu `task_id` ma być podany identyfikator właśnie utworzonego zadania.

c) Posiada widok `/todolist/<id_zadania>` z metodą `PATCH`, który przyjmuje na wejściu JSONa i wyedytuje istniejące zadanie z podanym w URLu identyfikatorem. Zmienione mają być tylko te pola, które zostaną podane spośród: `title`, `done`, `done_date`.

W przypadku podania identyfikatora zadania które nie istnieje, powinien być zwrócony status `404`
w przypadku podania `done: true` oraz niepodania `done_date`, należy użyć bieżącego czasu jako `done_date`
w przypadku podania done: false oraz `done_date` innego niż `null`, należy zwrócić kod `HTTP 400`
w przypadku zmiany z done: true na `done: false`, należy wyczyścić w bazie pole `done_date`
w przypadku sukcesu powinien być zwrócony status `204 z pustą treścią odpowiedzi`

Przykładowe wejście 1:

```
PATCH `/todolist/2`

{
    "done": true
}
```

powinno ustawić zadanie o tytule `Foobar` jako wykonane z aktualną datą.

Przykładowe wejście 2:

```
PATCH `/todolist/2`

{
    "title": "Learn even more Python",
    "done": false
}
```

powinno zmienić tytuł zadania `Foobar` na `Learn even more Python` i zmienić stan na niewykonane oraz zmienić `done_date` na `null`.

d) Posiada widok `` `/todolist/<id_zadania>` `` z metodą `GET`, który w przypadku podania identyfikatora zadania które nie istnieje, zwraca status `404`. W przypadku istniejącego zadania, powinien być zwrócony JSON w postaci:

```
{
    "title": "Learn even more Python",
    "done": false,
    "author_ip": "123.45.67.89",
    "created_date": "2018-05-08 10:00:00",
    "done_date": "2019-05-09 14:33:44"
}
```

e) Posiada widok `` `/todolist/<id_zadania>` `` z metodą `DELETE`, który w przypadku podania identyfikatora zadania które nie istnieje, zwraca status `404`. W przypadku istniejącego zadania, zadanie z tym `id` powinno zostać usunięte z bazy, po czym powinien zostać zwrócony status `HTTP 204`.

Zadanie 2

Kod zadania zapisz w pliku `zadanie2.py`

a) Napisz klasę `Car`, która będzie reprezentowała samochód.
Podczas tworzenia instancji tej klasy mają być możliwe do podania parametry **`brand`** (marka samochodu), **`tank_capacity`** (maksymalna objętość baku w litrach), **`tanked_fuel`** (liczba litrów paliwa w baku). Podczas tworzenia instancji należy zalogować na poziomie `INFO` komunikat

`New car of brand <brand>, with tank full in XXX%.`

Gdzie `XXX` ma być procentowym napełnieniem baku z dokładnością do 1 miejsca po przecinku. Przykładowo dla samochodu z bakiem o pojemności 40 litrów i wypełnionym 10 litrami paliwa, `XXX` wynosi `25.0%`.

Obiekty klasy `Car` mają posiadać następujące metody:

- `fill_tank` – bez podania żadnych argumentów bak samochodu wypełnia się paliwem do pełna. Metoda ma zwracać liczbę litrów **dolanego** paliwa.
- `fill_tank` – z podaniem argumentu `limit` wypełnia bak paliwem do podanego limitu, gdzie `limit` jest wartością z zakresu <0;1> i reprezentuje procentowe napełnienie baku. Przykładowo `fill_tank` z argumentem `limit` równym `0.25` powinno zatankować pusty bak o pojemności 40 litrów 10 litrami paliwa. W przypadku gdy bak zawierał już co najmniej tyle paliwa ile jest oczekiwane, tankowanie nie zmienia stanu napełnienia baku. Metoda ma zwracać liczbę litrów **dolanego** paliwa.
- `fill_tank` – z podaniem argumentu `liters` wypełnia bak podaną liczbą litrów paliwa. W przypadku gdyby maksymalna pojemność baku miałaby być przekroczona, oczekiwane jest rzucenie odpowiedniego wyjątku z odpowiednim komunikatem. Metoda ma zwracać liczbę litrów **dolanego** paliwa.
- magiczną metodę, która po napisaniu w konsoli pythona nazwy obiektu i naciśnięciu klawisza enter ma pokazać
  `<Car at <adres_w_pamięci> of brand <brand>, with tank full in XXX%>` (znaczenie `XXX` określone jak wcześniej)

b) Wszystkie metody muszą być rozsądnie zabezpieczone (rzucenie wyjątku) przed podaniem niepoprawnych danych, przykładowo niedozwolone powinno być wlanie "`stu`" litrów lub podanie limitu spoza zakresu `<0;1>` lub podanie równoczesne limitu oraz liczby wlewanych litrów (te argumenty wykluczają się wzajemnie).

c) Napisz funkcję `get_carpool`, która przyjmie jako jedyny argument liczbę samochodów które chcemy utworzyć i zwróci **zbiór** obiektów typu `Car` z losowymi pojemnościami baku, losowymi zapełnienieniami baków oraz losowymi, ale niepowtarzającymi się markami.

UWAGA: przy wylosowaniu nowego samochodu z taką samą marką (`brand`), taki „powtarzający się” samochód natychmiast odrzucamy, tzn. w naszym zbiorze samochodów nigdy nie istnieją dwa samochody z identyczną marką. **UWAGA: to była podpowiedź, która nabierze sensu po pierwszej próbie rozwiązania.**
UWAGA: podanie do funkcji `get_carpool` argumentu `3` oznacza, że chcemy otrzymać zbiór dokładnie 3 samochodów. Kod „tworzący” obiekty typu `Car` powinien to wziąć pod uwagę i w przypadku wygenerowania drugiego samochodu z taką samą marką jak

dowolnego samochodu już istniejącego w zbiorze, ten samochód pominąć i utworzyć nowy.

d) Napisz klasę `DieselCar`, która reprezentuje samochód, ale różni się tym, że jakiekolwiek wywołanie metody `fill_tank` rzuci wyjątek zadeklarowanego przez Ciebie typu `EnvironmentalError` z komunikatem `Diesel fuel not available due to environmental reasons.`

Zadanie 3

Odpowiedź zapisz w pliku `zadanie3.py`

a) Napisz funkcję `extract_elements(list1, list2)`, która po podaniu jako argumentów dwóch list zwróci elementy z pierwszej z nich o indeksach zapisanych w drugiej z tych list. Wykorzystaj list comprehension. Zakładamy, że indeksy podane w drugiej liście są podawane w standardzie indeksowania „od zera”, tzn. indeks `0` oznacza skrajnie lewy element. Przykładowo dla danych:

```
lista1 = [5, 10, 15, 20]
lista2 = [3, 1, 2]
```

wynik działania funkcji `extract_elements(lista1, lista2)` będzie równy `[20, 10, 15]` (czyli trzeci, pierwszy i drugi element z `lista1`)

Dla przypadków gdy funkcja zostanie uruchomiona z niepoprawnymi parametrami (przykładowo: `lista2` zawiera indeks `alamakota` lub indeks, który nie istnieje w `lista1`), nie jest oczekiwana specjalna obsługa błędów.
b) Napisz kilka testów jednostkowych, które sprawdzą poprawność działania funkcji z kilkoma różnymi przypadkami testowymi. Częścią zadania jest wymyślenie jakie przypadki warto przetestować.
c) Określ złożoność pamięciową i obliczeniową swojego rozwiązania, uzasadnij podaną wartość (napisz komentarz w kodzie).

Zadanie 4

Wypełnij formularz z pytaniami pod adresem https://forms.gle/rFeXFm9oX9YEJ7eL6