Komunikacja sieciowa i tworzenie interfejsu tekstowego w języku Python – zadania

Część 1 – komunikacja sieciowa (socket)

zad. 1 – Prosty serwer

1. Utwórz obiekt klasy `socket` (z modułu `socket`). W konstruktorze podaj następujące argumenty: `socket.AF_INET, socket.SOCK_STREAM`. Pierwszy z nich wskazuje, że będziemy korzystać z protokołu IP w wersji 4. Drugi określa typ gniazda jako gniazdo strumieniowe (wykorzystywane przez TCP).
2. Wywołaj na utworzonym sockecie metodę `bind((`*`ip, port`*`))`. Dowiąże ona nasz socket do konkretnego adresu IP i portu urządzenia. W miejsce *`ip`* wstaw adres loopback (127.0.0.1) jeśli chcesz testować działanie programu tylko w obrębie tego urządzenia lub adres interfejsu sieciowego, jeśli chcesz przetestować połączenie z innym urządzeniem (adres wpisz jako `string`). Jako *`port`* możesz podać dowolną liczbę z zakresu `1024-65535`. (`bind()` jako argument przyjmuje krotkę!)
3. Wywołaj na sockecie metodę `listen()`. Jako argument przyjmuje ona maksymalną dopuszczalną ilość niezaakceptowanych połączeń zanim system zacznie odrzucać kolejne. Będziemy chcieli obsługiwać tylko jednego klienta, więc możemy wpisać `1`.
4. Wywołaj na sokcecie metodę accept(). Spowoduje ona zatrzymanie programu w oczekiwaniu na nawiązanie połączenia przez klienta, a gdy to nastąpi zwróci dwuelementową krotkę. Pierwszy element krotki to socket klienta – będziemy z niego korzystać przy wysyłaniu i odbieraniu danych. Drugi to adres klienta. Zapisz elementy krotki do zmiennych.
5. Wywołaj na sokcecie klienta metodę `recv(4096)` (argument pozwala określić liczbę bajtów do odebrania). Ta instrukcja spowoduje zatrzymanie programu w oczekiwaniu na przybycie danych od klienta, a po ich przybyciu zwróci je. Zapisz odebrane dane do zmiennej.
6. Odebrane dane są obiektem typu `bytes`, należy je więc odkodować (metoda `decode()`). (Domyślnie metody `encode()` i `decode()` korzystają z kodowania utf-8). Po odkodowaniu dane to po prostu string.
7. Dane, które odeślemy do klienta niech będą danymi które nam wysłał, przerobionymi w jakiś sposób (np. odwróć kolejność znaków, dopisz coś do wiadomości...).
8. Wyślij "przetworzone" i zakodowane dane do klienta metodą `sendall()`.
9. Instrukcje z punktów 5-8 umieść w nieskończonej pętli. Całą pętlę umieść z kolei w bloku `try`.
10. Utwórz blok `except KeyboardInterrupt`. Zamknij w nim sokcet klienta, a następnie socket serwera metodą `close()`.
11. Utwórz blok `finally`. Zamknij sockety analogicznie do pkt. 10.

zad. 2 – Prosty klient

1. Utwórz obiekt klasy `socket` (analogicznie do zad. 1.1).
2. Wywołaj na utworzonym sockecie metodę `connect((`*`ip, port`*`))`. Posłuży ona do nawiązania połączenia z serwerem o podanym adresie IP na podanym porcie. Wpisz adres i port serwera z którym chcesz się połączyć (z zad. 1.).
3. Wyślij do serwera dane wprowadzone przez użytkownika.
4. Odbierz i wyświetl odpowiedź od serwera.
5. Instrukcje z punktów 3-4 umieść w nieskończonej pętli. Całą pętlę umieść z kolei w bloku `try`. Utwórz bloki `except` i `finally` analogicznie do zad. 1.

zad. 3 – Testowanie komunikacji

1. Przetestuj działanie serwera i klienta lokalnie na swoim komputerze lub połącz się z serwerem/klientem napisanym przez innego uczestnika zajęć.

Część 2 – tworzenie interfejsu tekstowego (npyscreen)

Do wykonania poniższych zadań konieczne będzie zainstalowanie modułu `npyscreen` (`pip install npyscreen`). W przypadku systemu Windows potrzebny będzie też moduł `windows-curses`.

zad. 4 – Tworzenie klasy formularza

Utwórz swoją klasę formularza w oparciu o poniższy kod:

```
class MyForm(npyscreen.Form):
    def create(self):
        self.myText = self.add(npyscreen.Textfield, editable=False, value="spam! "*8)

    def afterEditing(self):
        self.parentApp.setNextForm(None)
```

W metodzie `create()` dodajemy do formularza widgety. Jako pierwszy argument metody `add()` podajemy rodzaj widgetu do utworzenia, zaś kolejne argumenty (keyword arguments) pozwalają określić dodatkowe cechy widgetu. Metoda `afterEditing()` decyduje o tym jaki kolejny formularz zostanie wyświetlony po zamknięciu obecnego. `None` jako argument metody `setNextForm()` oznacza, że po zamknięciu formularza aplikacja zakończy działanie.
Umieść w swoim formularzu kilka różnych widgetów. Informacje o różnych widgetach i dostępnych argumentach znajdziesz w dokumentacji npyscreen: https://npyscreen.readthedocs.io/
Możesz poeksperymentować też z różnymi klasami formularzy.

zad. 5 – Tworzenie klasy aplikacji

Klasa aplikacji powinna zawierać w metodzie `onStart()` wszystkie formularze których chcemy użyć. Przy uruchomieniu aplikacji wyświetli się formularz z id `'MAIN'`. Skorzystaj z poniższej klasy aplikacji:

```
class MyApplication(npyscreen.NPSAppManaged):
    def onStart(self):
        self.addForm('MAIN', MyForm, name="my beautiful TUI")
```

zad. 6 – Uruchomienie aplikacji

Aby uruchomić TUI należy utworzyć obiekt klasy `MyApplication` i wywołać na nim metodę `run()`. Pamiętaj, że programy oparte o `npyscreen` należy uruchamiać z konsoli (TUI utworzy się w konsoli), uruchomienie z poziomu IDE raczej się nie uda.

Część 3 – połączenie informacji z poprzednich części – (klient sieciowy z TUI)

zad. 7 – Klient sieciowy z TUI

Dla aplikacji klienta z zad. 2 utwórz interfejs tekstowy, który umożliwi wprowadzanie danych i będzie wyświetlał odpowiedzi od serwera.