応用プログラミング ex2 演習課題

後半の演習の進め方については、後半第1回の0.はじめにをご覧ください。

1. sys.argv でコマンドライン引数を使う

1-1. argv.py

1. /roes/sample/sano/apro/argv.py をホームディレクトリ以下の適当な場所にコピーしなさい。
2. argv.py を適当なコマンドライン引数と共に実行し、その動作を確認しなさい。
3. argv.py を修正し、コマンドライン引数として与えられた整数を全て加算して出力する add.py を作成しなさい。ただし、引数に整数値に変換できない文字列が与えられた場合の処理は考えなくてもよい。

実行例：

```
(aprog) t190900@s01cd0542-161:~/apro$ python add.py
0
(aprog) t190900@s01cd0542-161:~/apro$ python add.py 1 3 9
13
(aprog) t190900@s01cd0542-161:~/apro$ python add.py -1 3 -9
-7
(aprog) t190900@s01cd0542-161:~/apro$ python add.py 1 2 -34 56 -78 90
37
```

1-2. echoServer.py

1. echoServer.py を修正し、コマンドライン引数でサーバのポート番号を指定できるように修正しなさい。
2. ただし、引数が与えられない場合はポート番号を 50007 とし、また、2つ以上の引数が与えられた場合はコマンドの利用方法を表示して終了すること。

実行例(^C は Control+C での強制終了)

```
(aprog) t180900@s01cd0542-161:~/apro$ python echoServer.py
port = 50007
^C
(aprog) t180900@s01cd0542-161:~/apro$ python echoServer.py 1234
port = 1234
^C
(aprog) t180900@s01cd0542-161:~/apro$ python echoServer.py 1234 5678
Usage: echoServer.py [port]
```

サーバプログラムを続けて実行すると、

```
OSError: [Errno 98] Address already in use
```

のようなエラーが表示され、サーバがうまく立ち上がらないことがあります。 これは、以前に実行したサーバプログラムで利用したアドレスやポートが、プログラム終了後もすぐにはOSから開放されずに再利用できないことが原因です。 他のポート番号を利用して新しいサーバプログラムを実行すると、以前のものと衝突せずに起動することができます。 また、他の方法としては、ソケット作成後にそのソケットに以下のようなオプションを設定することでも、この問題を回避することができます。

```
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, True)
```

1-3. echoClient-ex.py

1. echoClient.py を修正し、コマンドライン引数でサーバのホスト名(IPアドレス)、送信文字列、ポート番号を指定できるプログラム **echoClient-ex.py**を作成しなさい。
2. ただし、ポート番号の指定は省略できるものとし、引数で与えられない場合はポート番号を 50007 とする。
3. また、引数が2つ未満でホスト名と送信文字列が与えられない場合、または、4つ以上の引数が与えられた場合はコマンドの利用方法を表示して終了すること。
4. **echoClient-ex.py を作成するために行ったプログラムの修正箇所を説明しながら、echoClient-ex.py を実行する様子をTAに示しなさい。**

実行例：(サーバは送信文字列を2倍して返信)

```
(aprog) t180900@s01cd0542-161:~/apro$ python echoClient-ex.py
localhost あいうえお 5007
localhost あいうえお 5007
Send あいうえお
Received あいうえおあいうえお
(aprog) t180900@s01cd0542-161:~/apro$ python echoClient-ex.py
127.0.0.1 hello 5007
127.0.0.1 hello 5007
Send hello
Received hellohello
(aprog) t180900@s01cd0542-161:~/apro$ python echoClient-ex.py
127.0.0.1 hello
127.0.0.1 hello 50007
Send hello
Received hellohello
(aprog) t180900@s01cd0542-161:~/apro$ python echoClient-ex.py
localhost
Usage: echoClient-ex.py <server> <message> [port]
(aprog) t180900@s01cd0542-161:~/apro$ python echoClient-ex.py
localhost hello 5007 123
Usage: echoClient-ex.py <server> <message> [port]
```

2. 繰り返しの送受信

1. echoServer.py は、クライアントとの通信を1回行うごとにプログラムの実行を終了するため、続けて通信するためには再び echoServer.py を起動し直す必要がある。
2. echoServer.py を修正し、1つのクライアントとの送受信を完了しても終了せず、再び受信待機(Listen)状態に復帰し、引き続き次のクライアントと通信を行うことができるプログラム echoServer-ex.py を作成しなさい。
3. **echoServer-ex.py を作成するために行ったプログラムの修正箇所を説明しながら、実際に繰り返し通信を実行する様子をTAに示しなさい。**

実行例(^C は Control+C での強制終了)

```
(aprog) t180900@s01cd0542-161:~/apro$ python echoServer-ex.py 5001
port = 5001
Connected by ('127.0.0.1', 57680)
Received Hello
Send Hello
Connected by ('127.0.0.1', 57683)
Received HelloHello
Send HelloHello
Connected by ('127.0.0.1', 57686)
Received Hello World
Send Hello World
^C
```

3. ファイルの内容を送信するサーバ htServer.py

1. htServer.py は、クライアントから 'GET' というコマンドを受け取るとhtServer.pyと同じディレクトリにあるテキストファイル（例えばserver.txtの内容を返信し、それ以外のコマンドを受け取ると 'not a command' のようなメッセージを返信する。このようなサーバプログラム **htServer.py を作成し、TAにプログラムの内容を説明しながら実行の様子を示しなさい。**

テキストファイルの例: server.txt

```
これは server.txt ファイルの中身です
```

実行例: htServer.py

```
(aprog) t180900@s01cd0542-161:~/apro$ python htServer.py 50001
port = 50001
Connected by ('127.0.0.1', 55810)
Received GET
これは server.txt ファイルの中身です
Connected by ('127.0.0.1', 55824)
Received get
get is not a command
```

実行例: echoClient-ex.py

```
(aprog) t180900@s01cd0542-161:~/apro$ python echoClient.py
localhost GET 50001
localhost GET 50001
Send GET
Received これは server.txt ファイルの中身です
(aprog) t180900@s01cd0542-161:~/apro$ python echoClient.py
localhost get 50001
localhost get 50001
Send get
Received get is not a command
```