```
"""
COSC 264 - Assignment 1
Creation Date: 30/7/18
Name: Zachary Sanson
Student ID: 58520526
File: Server.py
"""
#   Note printing my program into a PDF makes a mess of my formatting


import socket
import select
import datetime


class DtRequest:
    """Class for a DT Request Packet"""
    def __init__(self):
        self.magicNo = 0x0000.to_bytes(2, byteorder='big')          # 16-bit
        self.packetType = 0x0000.to_bytes(2, byteorder='big')       # 16-bit
        self.requestType = 0x0000.to_bytes(2, byteorder='big')      # 16-bit 0x0001 or 0x0002

    def __str__(self):
        """Representation of our packet in string form"""
        return str(self.magicNo + self.packetType + self.requestType)

    def convert_bin(self, bin_string):
        """Converts a binary string to a DT Response type"""
        if len(bin_string) != 6:
            raise ValueError("DtRequest received an incorrect packet length.\n---Exiting---")
        self.magicNo = bin_string[:2]
        self.packetType = bin_string[2:4]
        self.requestType = bin_string[4:6]

    def check_request(self):
        """Checks if packet is a valid request packet"""
        # We do not need to check for packet length = 6 as it is covered in convert_bin
        magic_n, p_type = 0x497E.to_bytes(2, byteorder='big'), 0x0001.to_bytes(2, byteorder='big')
        requests = [0x0001.to_bytes(2, byteorder='big'), 0x0002.to_bytes(2, byteorder='big')]
        if self.magicNo == magic_n and self.packetType == p_type and self.requestType in requests:
            return True
        else:
            return False


class DtResponse:
    """Class for a DT Response Packet"""
    def __init__(self):
        self.magicNo = 0x497E.to_bytes(2, byteorder='big')          # 16-bit
        self.packetType = 0x0002.to_bytes(2, byteorder='big')       # 16-bit
        self.languageCode = 0x0000.to_bytes(2, byteorder='big')     # 16-bit, 0x0001 or 0x0002 or 0x0003
        self.year = 0x0000.to_bytes(2, byteorder='big')             # 16-bit, year < 2100
        self.month = 0x0000.to_bytes(1, byteorder='big')            # 8-bit, range(1, 12)
        self.day = 0x0000.to_bytes(1, byteorder='big')              # 8-bit, range(1, 31)
        self.hour = 0x0000.to_bytes(1, byteorder='big')             # 8-bit, range(0, 23)
        self.minute = 0x0000.to_bytes(1, byteorder='big')           # 8-bit, range(0, 59)
        self.length = 0x0000.to_bytes(1, byteorder='big')           # 8-bit
        self.text = 0x0000.to_bytes(1, byteorder='big')             # ?-bit, gets re-declared when class is created

    def __str__(self):
```

```
61          """Representation of our packet in string form"""
62          return str(self.magicNo + self.packetType + self.languageCode + self.year + self
    .month + self.day + \
63                     self.hour + self.minute + self.length + self.text)
64
65      def packet(self):
66          """Prepares DT Response packet for transfer"""
67          # Returns a bytearray that will be sent to clients
68          return self.magicNo + self.packetType + self.languageCode + self.year + self.
    month + self.day + self.hour + \
69              self.minute + self.length + self.text
70
71
72  def user_input():
73      """Prompts user for input for setup"""
74      server_input = [None, None, None]
75      # Defining either an IP address or a hostname of the server
76      usr_in = input("Enter an IP address or hostname for the server to run on: ")
77      try:
78          socket.getaddrinfo(usr_in, 00000)  # Port doesn't matter for checking address
79          machine_ip = usr_in
80      except OSError:
81          print("ValueError: encountered invalid input for a server address.\n---Exiting
    ---")
82      # Defining the three port numbers to run the server on
83      for count in range(0, 3):
84          interface = input("Input value for port {}: ".format(count + 1))
85          if int(interface) not in range(1024, 64000):
86              raise ValueError("entered port number is out of range 1,024 - 64,000.\n---
    Exiting---")
87          if int(interface) in server_input:
88              raise AssertionError("port number already used within the machine.\n---
    Exiting---")
89          server_input[count] = int(interface)
90      return server_input, machine_ip
91
92
93  def get_string(language, text_type):
94      """Creates the response string in accordance to the clients wishes"""
95      # Pretty straight forward, deciphering what language and string to use for our
    packet
96      date = datetime.datetime.now()
97      year, month, day, hour, minute = date.year, date.month, date.day, date.hour, date.
    minute
98      type_date = 0x0001.to_bytes(2, byteorder='big')
99      if language == 0x0001:
100         language_months = ['January', 'February', 'March', 'April', 'May', 'June', 'July
    ',
101                            'August', 'September', 'October', 'November', 'December']
102         if text_type == type_date:
103             language_string = "Today's date is {} {}, {}".format(language_months[month -
     1], day, year)
104         else:
105             language_string = "The current time is {}:{}".format(hour, minute)
106     elif language == 0x0002:
107         language_months = ['Kohitatea', 'Hui-tanguru', 'Poutu-te-rangi', 'Paenga-whawha'
    , 'Haratua', 'Pipiri',
108                            'Hongongoi', 'Here-turi-koka', 'Mahuru', 'Whiringa-a-nuku', '
    Whiringa-a-rangi',
109                            'Hakihea']
110         if text_type == type_date:
111             language_string = "Ko te ra o tenei ra ko {} {} {}".format(language_months[
    month - 1], day, year)
112         else:
113             language_string = "Ko te wa o tenei wa {}:{}".format(hour, minute)
114     else:
115         language_months = ['Januar', 'Februar', 'Marz', 'April', 'Mai', 'Juni', 'Juli',
```

```
115 'August', 'September',
116                             'Oktober', 'November', 'Dezember']
117         if text_type == type_date:
118             language_string = "Heute ist der {}. {} {}".format(language_months[month - 1
    ], day, year)
119         else:
120             language_string = "Die Uhrzeit ist {}:{}".format(hour, minute)
121     return language_string, date
122 
123 
124 def create_dt_response(language, date, string, text_type):
125     """Creates/updates the DT Response packet"""
126     dt_response = DtResponse()
127     dt_response.languageCode = language.to_bytes(2, byteorder='big')
128     # If text is date format
129     if text_type == 0x0001.to_bytes(2, byteorder='big'):
130         dt_response.year = date.year.to_bytes(2, byteorder='big')
131         dt_response.month = date.month.to_bytes(1, byteorder='big')
132         dt_response.day = date.day.to_bytes(1, byteorder='big')
133     # time otherwise
134     else: # Else is fine as the packet should have passed a check
135         dt_response.hour = date.hour.to_bytes(1, byteorder='big')
136         dt_response.minute = date.minute.to_bytes(1, byteorder='big')
137     # Check for length of text |T| >= 255
138     if len(string.encode('utf-8')) < 255:
139         dt_response.text = string.encode('utf-8')
140     else:
141         raise ValueError("Text field is not within 255 bytes.\n---Exiting---")
142     dt_response.length = len(dt_response.text).to_bytes(1, byteorder='big')
143     return dt_response
144 
145 
146 def packet_processing(packet, address, sock, port_no, port):
147     """Checks packet for integrity then sends out a response"""
148     # Creating a DtRequest packet and performing checks on the packet
149     dt_request = DtRequest()
150     dt_request.convert_bin(packet)
151     if dt_request.check_request():
152         print("Packet has passed the required DT Request integrity checker.\nGenerating
    a DT Response packet\n...")
153         text_type = dt_request.requestType
154         # Finds what language to use from the port we received the packet from
155         if port_no == 0:
156             language = 0x0001
157         elif port_no == 1:
158             language = 0x0002
159         elif port_no == 2:
160             language = 0x0003
161         # Creating text for the packet
162         string, date = get_string(language, text_type)
163         # Creating DT Response packet
164         dt_response = create_dt_response(language, date, string, text_type)
165         print("DT Response packet created.\nSending packet to {}:{} from port {}.\n...".
    format(address[0], address[1], port))
166         sock.sendto(dt_response.packet(), address)
167         print("Response successfully sent to {}:{}.\n...".format(address[0], address[1])
    )
168     else:
169         print("Received packet did not pass the DT Request integrity checker.\n---
    Ignoring Packet---")
170 
171 
172 def receive(port_list, server_ip):
173     """Sets up server sockets and checks for incoming data"""
174     print("Setting up sockets {}, {}, {} on {}.\n...".format(port_list[0], port_list[1],
     port_list[2], server_ip))
175     try:
```

```
176         # Setting up sockets for server to run on
177         sock1, sock2, sock3 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM), \
178                               socket.socket(socket.AF_INET, socket.SOCK_DGRAM), \
179                               socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
180         sockets, counter = [sock1, sock2, sock3], 0
181         # Binding each socket to their respective port
182         for sock in sockets:
183             sock.bind((server_ip, port_list[counter]))
184             sock.setblocking(0)
185             counter += 1
186     except OSError:
187         print("Could not open and/or bind specified server ports.\n---Exiting---")
188     print("Sockets connected.\nListening on ports {}, {}, {} for incoming transmissions.
    \n...".format(
189         port_list[0], port_list[1], port_list[2]))
190     while True:  # Indefinitely checks for incoming packets
191         # Waits until a packet arrives on any port
192         read_list, _, _ = select.select(sockets, [], [])
193         for readable_socket in read_list:
194             received_packet, address = readable_socket.recvfrom(60)
195             port_no = port_list.index(readable_socket.getsockname()[1])
196             print("Found packet on port {} from {}:{}, checking packet for integrity.\n
    ...".format(port_list[port_no], address[0], address[1]))
197             packet_processing(received_packet, address, readable_socket, port_no,
    port_list[port_no])
198
199
200 def main():
201     """Main call for the server program"""
202     port_list, server_ip = user_input()
203     receive(port_list, server_ip)
204
205
206 main()
207
```