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


import socket


class DtRequest:
    """Class for a DT Request Packet"""
    def __init__(self):
        self.magicNo = 0x497E.to_bytes(2, byteorder='big')          # 16-bit
        self.packetType = 0x0001.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 packet(self):
        """Prepares DT Request packet for transfer"""
        return self.magicNo + self.packetType + self.requestType


class DtResponse:
    """Class for a DT Response 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.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(2, byteorder='big')             # ?-bit, gets re-declared when class is created.

    def __str__(self):
        """Representation of our packet in string form"""
        return str(self.magicNo + self.packetType + self.languageCode + self.year + self.month + self.day +
                   self.hour + self.minute + self.length + self.text)

    def __len__(self):
        """Returns the bit length of our DT Response packet"""
        return len(self.magicNo + self.packetType + self.languageCode + self.year + self.month + self.day +
                   self.hour + self.minute + self.length + self.text)

    def convert_bin(self, bin_string):
        """Converts a binary string to a DT Response type"""
        if len(bin_string) < 13:
            raise ValueError("DtResponse received an incorrect packet length.")
        # We can index the binary string we receive to import it into our class
        self.magicNo = bin_string[:2]
        self.packetType = bin_string[2:4]
        self.languageCode = bin_string[4:6]
        self.year = bin_string[6:8]
        self.month = bin_string[8:9]
        self.day = bin_string[9:10]
        self.hour = bin_string[10:11]
        self.minute = bin_string[11:12]
        self.length = bin_string[12:13]
        self.text = bin_string[13:]
        if len(self.length) != (13 + len(self.text)):
            raise ValueError("DtResponse received an incorrect packet length.")

    def int_in_range(self, bit, x, y):
        """Checks if a self variable is within given range"""
        # Returns True if in range
        return int.from_bytes(bit, byteorder='big') in range(x, y)

    def check_response(self, type):
        """Checks if packet is a valid response packet"""
        # We don't need to check for length < 13 as it is covered in convert_bin
        # All ranges need to be increase by one due to pythons methods!!!
        if not (self.magicNo == 0x497E.to_bytes(2, byteorder='big') and
                self.packetType == 0x0002.to_bytes(2, byteorder='big') and
                self.int_in_range(self.languageCode, 1, 4) and
                self.__len__() == (int.from_bytes(self.length, byteorder='big') + 13)):
            raise ValueError("DT Response integrity check has failed.\n---Exiting---")
        # Check fields corresponding to whether we wanted time or date
        if type == 'date':
            if not(self.int_in_range(self.year, 0, 2101) and
                   self.int_in_range(self.month, 1, 13) and
                   self.int_in_range(self.day, 1, 32)):
                raise ValueError("DT Response integrity check has failed.\n---Exiting---")
        else:
            if not(self.int_in_range(self.hour, 0, 24) and self.int_in_range(self.minute, 0, 59)):
                raise ValueError("DT Response integrity check has failed.\n---Exiting---")


def user_input():
    """Prompts user for input for setup"""
    # Defining either date or time of package
    usr_in = input("Enter either 'date' or 'time' to proceed: ")
    if usr_in in ['date', 'time']:
        time = usr_in
    else:
        raise ValueError("input does not match.\n---Exiting---")
    # Defining either an IP address or a hostname of destination
    usr_in = input("Enter an IP address or hostname for your destination server: ")
    try:
        socket.getaddrinfo(usr_in, 00000)  # Port doesn't matter for checking address
        destination_address = usr_in
    except OSError:
        print("ValueError: encountered invalid input for a destination address.\n---Exiting---")
    # Defining server port number
    usr_in = input("Enter a port number that your destination server is on: ")
    server_port = int(usr_in)
    if server_port not in range(1024, 64000):
        raise ValueError("entered port number is out of range 1,024 - 64,000.\n---Exiting---")
    return time, destination_address, server_port


def print_packet(dt_response):
    """Prints out the DT Response packet"""
    # Client prints entire DT Response packet???
    # In form of an entire bytearray
    print("\nComposition of DT Response packet.")
    print(dt_response)
    contents = [(dt_response.magicNo, "Magic_No: "), (dt_response.packetType, "Packet_Type: "),
                (dt_response.languageCode, "Language_Code: "), (dt_response.year, "Year: "),
                (dt_response.month, "Month: "), (dt_response.day, "Day: "),
                (dt_response.hour, "Hour: "), (dt_response.minute, "Minute: "),
                (dt_response.length, "Length: "), (dt_response.text, "Text: ")]
    # In form of actual integers and strings
    for value in contents:
        if value[0] == dt_response.magicNo:
            print(value[1] + str(hex(int.from_bytes(value[0], byteorder='big'))))
        if value[0] == dt_response.text:
            print(value[1] + dt_response.text.decode('utf-8'))
        else:
            print(value[1] + str(int.from_bytes(value[0], byteorder='big')))
    # Print out our beautiful time/date in whatever language you want!
    print("\n>>>" + dt_response.text.decode('utf-8'))


def send_to(time, server_address):
    """Sends packet to server"""
    print("DT Request packet created, sending packet to server on {}:{}.\n...".format(server_address[0], server_address[1]))
    # Create a DT Request packet and set up sockets for transfer
    dt_request, dt_response = DtRequest(), None
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    except OSError:
        print("Could not establish an outgoing connection.\n---Exiting---")
    if time == "date":
        dt_request.requestType = 0x0001.to_bytes(2, byteorder='big')
    else:
        dt_request.requestType = 0x0002.to_bytes(2, byteorder='big')
    sock.sendto(dt_request.packet(), server_address)
    print("Packet sent, waiting on response from {}:{}.\n...".format(server_address[0], server_address[1]))
    try:
        # Set a timeout for the socket to one second
        sock.settimeout(1.0)
        received_packet, server_info = sock.recvfrom(300)
        # We no longer need this socket so to save resources we close the socket
        sock.close()
        dt_response = DtResponse()
        dt_response.convert_bin(received_packet)
    finally:
        # If we still come out of the try statement with no packet return an error
        if dt_response is None:
            raise TimeoutError("could not setup connection with {}.\n---Exiting---".format(server_address))
        print("Received packet from {}:{}, checking integrity.\n...".format(server_info[0], server_info[1]))
        # Packet checking
        dt_response.check_response(time)
        print("Checking complete: packet has been accepted.")
        print_packet(dt_response)
        return dt_response


def main():
    """Main call to our client program"""
    time, destination_address, server_port = user_input()
    # Create request packet
    print("Creating DT Request packet for transmission.\n...")
    # Send packet to server
    send_to(time, (destination_address, server_port))
    print("\n---End---")
```

```


main()

```