COSC 264 – Assignment 1

Zachary Sanson
58520526

Created On: 30/7/18

COSC 264 Assignment

Andreas Willig
Dept. of Computer Science and Software Engineering
University of Canterbury

July 24, 2018

---

1 Administrivia

This assignment is part of the COSC 264 assessment process. It is worth 8% of the final marks. It centers around socket programming using the Python programming language. Your program should be a text mode program that can run from the command line.

Please use the "Question and Answer Forum" on the Learn platform for raising and discussing any unclear technical issues. Please **do not** send emails with technical questions directly to me or the tutors, instead use the learn forum. This way other people can benefit from the question (and the answer).

2 Plagiarism Warning

Your submissions are logged and originality detection software will be used to compare your solution with other solutions. Dishonest practice, which includes

- letting someone else create all or part of an item of work,
- copying all or part of an item of work from another person with or without modification, and
- allowing someone else to copy all or part of an item of work,

may lead to partial or total loss of marks, no grade being awarded and other serious consequences including notification of the University Proctor.

You are encouraged to discuss the general aspects of a problem with others. However, anything you submit for credit must be entirely your own work and not copied, with or without modification, from any other person. If you need help with specific details relating to your work, or are not sure what you are allowed to do, contact your tutors or lecturer for advice. If you copy someone else's work or share details of your work with anybody else, you are likely to

be in breach of university regulations and the Computer Science and Software Engineering department's policy. For further information please see

- Academic Integrity Guidance for Staff and Students
  `www.canterbury.ac.nz/ucpolicy/GetPolicy.aspx?file=Academic-Integrity-Guidance-For-Staff-And-Students.pdf`
- Academic Integrity and Breach of Instruction Regulations in the University Calendar
  `www.canterbury.ac.nz/regulations/general-regulations/academic-integrity-and-breach-of-instruction-regulations/`

You will have to sign a plagiarism declaration upon submission of your assignment report.

3 Problem Description

Your task is to write to write two programs in Python. The first one, called `server` will allow the other program, called `client`, to ask the server for the current date or time of day. The server offers to deliver these in three different languages.

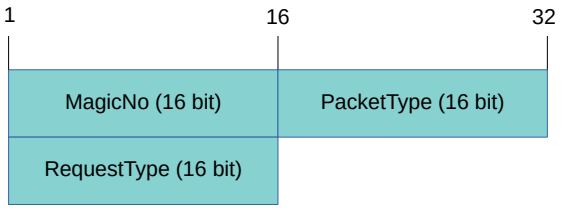
3.1 Packet Types and Packet Processing

The programs will require two different types of packets to be used: `DT-Request` and `DT-Response` packets (where 'DT' stands for "DateTime"). With a `DT-Request` a client requests either the date or the current time of day from the server. With a `DT-Response` packet the server returns both the date and current time of day in a binary representation, followed by either the date or the time of day (depending on the client's choice) in a textual representation.

These packets contain a number of 16- or 32-bit wide fields. All these fields shall use the big-endian format.

3.1.1 `DT-Request` packet

The format of the `DT-Request` packet is shown in the following figure:



It consists of the following fields:

- The first 16 bit field 'MagicNo' is taken up by a magic number, which needs to have the value `0x497E`. This is a simple safeguard to check whether a packet actually belongs to our Date-/Time protocol.

- The next 16 bit field 'PacketType' indicates the packet type within our DateTime protocol. For a `DT-Request` packet this field needs to have the value `0x0001`.
- The last 16 bit field 'RequestType' indicates the particular type of request the client makes. When it contains the value `0x0001` the client asks for a textual representation of the current date, when it contains the value `0x0002` the client asks for a textual representation of the current time of day. Other values are not allowed.

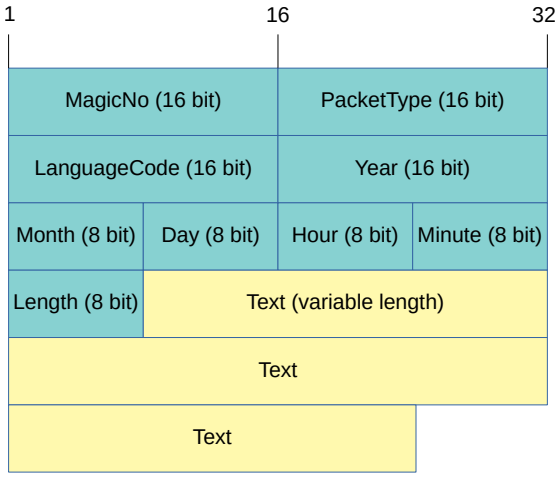
The `DT-Request` packet is being sent from the client to the server. Upon receiving such a packet the server will have to check:

- Whether the packet contains exactly six bytes of data.
- If it does, check whether the 'MagicNo' field contains the value `0x497E`.
- If it does, check whether the 'PacketType' field contains the value `0x0001`.
- If it does, check whether the 'RequestType' field contains either the value `0x0001` or `0x0002`.

If all these conditions are met the server will accept the packet for further processing (see below). If any of these conditions is not met, the server shall simply discard the packet without further action (but printing an appropriate error message on the terminal). Note that this in particular implies that the server will only process received packets with 'PacketType' being equal to `0x0001`, i.e. it processes only `DT-Request` packets.

3.1.2 `DT-Response` packet

The format of the `DT-Response` packet is shown in the following figure:



The first 13 bytes (shown in green) are the fixed header fields. The `DT-Response` packet consists of the following fields:

- The first 16 bit field 'MagicNo' is taken up by a magic number, which again needs to have the value `0x497E`.
- The 16 bit field 'PacketType' indicates the packet type within our Date-Time protocol. For a `DT-Response` packet this field needs to have the value `0x0002`.
- The 16 bit field 'LanguageCode' indicates the language used for the textual representation. The value `0x0001` indicates English, the value `0x0002` indicates Te reo Maori, and the value `0x0003` indicates German.
- The 16 bit field 'Year' contains the value for the year as a non-negative integer. As an example, for the current year 2018 this field would contain the value 2018.
- The eight bit field 'Month' contains the value for the month as a non-negative integer: 1 for January, 2 for February, and so on.
- The eight bit field 'Day' contains the value for the day of month as a non-negative integer. This number is allowed to range from 1 to 31.
- The eight bit field 'Hour' contains the hour of the day. This number is allowed to range from 0 to 23.
- The eight bit field 'Minute' contains the minute within the hour. This number is allowed to range from 0 to 59.
- The eight bit field 'Length' indicates the length of the following textual representation in bytes. For example if the text is "hello" (without the speech marks) then the 'Length' field wound contain the value 5.
- The 'Text' field is of variable length and contains the actual text being returned (a textual representation of either the date or the current time of day, see Section 3.4).

The `DT-Response` packet is being sent from the server to the client. Upon receiving such a packet the client will have to check:

- Whether the packet contains at least 13 bytes of data (i.e. all fixed header fields are present).
- If it does, check whether the 'MagicNo' field contains the value `0x497E`.
- If it does, check whether the 'PacketType' field contains the value `0x0002`.
- If it does, check whether the 'LanguageCode' field contains either of the values `0x0001`, `0x0002`, or `0x0003`.
- If it does, check whether the year is a number below 2,100.

- If it is, check whether the Month is a number from 1 to 12.
- If it is, check whether the Day is a number from 1 to 31.
- If it is, check whether the Hour is a number from 0 to 23.
- If it is, check whether the Minute is a number from 0 to 59.
- If it is, check whether the actual length of the entire packet you have received equals the sum of 13 (for the fixed header) plus the contents of the 'Length' field.

If all these conditions are met the client will accept the packet for further processing. If any of these conditions is not met, the client shall simply discard the packet, print an appropriate diagnostic message and exit the program. Note that this in particular implies that the client will only process received packets with 'PacketType' being equal to `0x0002`, i.e. it processes only `DT-Response` packets.

3.2 Server

The server program is started on a host and takes three different port numbers as parameters on the command line. All three port numbers must be different, and they must be numbers between 1,024 and 64,000. If these conditions are not met then the server should exit with an appropriate error message. The first port number is used by clients wanting to get the date/time information in English, the second port number by clients wanting it in Te reo Maori, and the third port number by clients wanting it in German.

Next the server will attempt to open three UDP / datagram sockets and to bind these three sockets to the three given port numbers. If this fails, then the server will exit with an appropriate error message.

After binding the port numbers the server will enter an infinite loop. In this loop:

- The server uses the `select()` system call to wait for a request packet on any of the three sockets. Note that the server should not use any CPU resources while waiting.
- When a packet is received on any of these three sockets, the server:
  - Retrieves the packet from the socket using the Python equivalent of `recvfrom()`. The received packet is stored in a bytearray, and furthermore the server stores the IP address and port number of the packet sender and memoizes on which of the three sockets the packet has been received – this socket determines the language to be used for the textual representation and is also the socket through which the response packet needs to be sent.

- Performs all the checks necessary to see whether the packet is a valid `DT-Request` packet (see Section 3.1.1). If it is not, print a diagnostic message on the terminal, discard the packet and continue at the start of the loop.
- Determines from the request whether the client wants to know today's date or the current time of day.
- Uses a system call to obtain the current time of day $hh : mm$ and date $yyyy : mm : dd$.
- Prepares a `DT-Response` packet as follows:
  - Prepare the textual representation T of either the date or the time of day in the chosen language as a bytearray (see Section 3.4). Let $|T|$ be the length of the text in bytes (e.g. $|\text{hello}| = 5$). Check whether $|T| \leq 255$. If not, print a diagnostic message and return to the start of the loop without any further processing of this request.
  - Allocate a sufficiently large buffer for the `DT-Response` packet which can hold both the fixed fields and the text T.
  - Fill in all the fixed header fields (from 'MagicNo' to 'Minute') according to the specifications given in Section 3.1.2.
  - Fill in the 'Length' field with $|T|$.
  - Fill in the textual representation T.
- Finally sends this response packet back to the original sender through the socket on which the corresponding request packet has been received. You will have to use the Python equivalent of `sendto()`, using the client IP address and port number retrieved at the start.

3.3 Client

The client program is started on the same host or another host, and takes three parameters on the command line:

- The first is either the string "date" or the string "time", letting the user specify what he/she wishes to see. If the first parameter is not equal to either of these, the client program should print an error message and stop.
- The second parameter is either an IP address in dotted-decimal notation (e.g. "130.66.22.212") or the hostname of the computer running the server (e.g. "datetime.mydomain.nz"). The client will attempt to convert this parameter to an IP address using the Python equivalent of the `getaddrinfo()` function. If this conversion fails (e.g. because the hostname does not exist or an IP address given in dotted-decimal notation is not well-formed) then the client should print an error message and stop.
- The third parameter is the port number to use on the server. The port number should be between 1,024 and 64,000. If it is not, then the client should print an error message and stop.

After all command line parameters have been checked the client performs the following operations:

- It opens a UDP socket and prepares a `DT-Request` packet according to the format specified in Section 3.1.1.
- This packet is then sent to the server using the Python equivalent of the function `sendto()`, using the IP address and port number obtained from the parameters as the destination.
- After sending the packet the client should wait for a response packet, but only for a limited time of one second (you may want to look into the Python equivalent of `select()` to limit your waiting time on a socket to one second). If no response packet arrives within one second, the client should print an error message and exit.
- If a response packet arrives within one second, the client should retrieve it from the socket and check whether it is a proper `DT-Response` packet (see Section 3.1.1). If it is not, the client should print an error message and exit.
- The client prints the complete contents of the `DT-Response` packet and then exits.

3.4 Textual Representation

	Date	**Time**
English	Today's date is <name(*mm*)> <*dd*>, <*yyyy*> Today's date is January 7, 2018	The current time is <*hh*>:<*mm*> The current time is 23:22
Te reo Maori	Ko te ra o tenei ra ko <name(*mm*)> <*dd*>, <*yyyy*> Ko te ra o tenei ra ko Kohitatea 7, 2018	Ko te wa o tenei wa <*hh*>:<*mm*> Ko te wa o tenei wa 23:22
German	Heute ist der <*dd*>. <name(*mm*)> <*yyyy*> Heute ist der 7. Oktober 2013	Die Uhrzeit ist <*hh*>:<*mm*> Die Uhrzeit ist 23:45

Table 1: Proposed textual representations (including examples)

You should compute a textual representation in the chosen language for both the date and the current time of day. A suggestion for the three chosen languages is given in Table 1. In this representation days, years, minutes and seconds are given as numbers, but a proper name is used for the month (the months names are given in Table 2). Note that if you find it difficult to enter the diacritics (macrons, Umlaut) into the Python source code you can leave those away.

Important point: You should put together the textual representation in Python using normal Python strings. Strings in Python are represented using the UTF-8 character encoding system, and in this system a printed character may require a variable number of bytes to store in memory. Hence, a string which looks like having m characters may in fact need a number of $n \geq m$ of

English	**Te reo Maori**	**German**
January	Kohitātea	Januar
February	Hui-tanguru	Februar
March	Poutū-te-rangi	März
April	Paenga-whāwhā	April
May	Haratua	Mai
June	Pipiri	Juni
July	Hōngongoi	Juli
August	Here-turi-kōkā	August
September	Mahuru	September
October	Whiringa-ā-nuku	Oktober
November	Whiringa-ā-rangi	November
December	Hakihea	Dezember

Table 2: Month names

bytes to represent it, and our server program needs to transmit all n bytes. To achieve this, you can use the `encode` method in Python to convert a string to a byte array. For example:

```
...
mystring = "hello world"
mybytes = mystring.encode('utf-8')
...
```

and then you can use the `len` function to find out the length of the byte array.

4 Deliverables

Each student has to submit **a single pdf file** which includes the following items:

- A cover sheet with your name and student-id.
- A listing of your source code. We would appreciate if you use a pretty printer.
- You have to print the plagiarism declaration form from the learn page of COSC 264, sign it, scan it, convert the scanned form into a pdf and include this into your submission.

Warning:

- Submissions that are not in pdf format or which contain more than one file automatically receive 0 marks!!
- Submissions which do not include the **signed** plagiarism declaration form automatically receive 0 marks!!

The pdf file has to be submitted via the `learn` page of COSC 264 (see `https://learn.canterbury.ac.nz/course/view.php?id=542&home=1`). Please submit no later than **Sunday, August 19, 2018, 11:59 pm**. Late submissions are **not** accepted, except through the special consideration mechanism.

5 Marking

Marking will be based on the source code. In particular, we will mark it for its ability to produce the right results (e.g. is packet processing correctly implemented, are all the right socket calls there, is the one-second timeout in the client implemented correctly), and for the amount of error / consistency checking you do – if we find that you use system-/socket calls without any error checking or that you do not check incoming packets for correctness, we will apply deductions. We will check whether you have returned all resources (sockets, files) to the operating system. We will also have an eye on style, and may apply deductions if your code is particularly ugly or messy.

Plagiarism Declaration

This form needs to accompany your COSC 264 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name: Zachary Sanson

Student ID: 58520526

Signature: ZachSanson

Date: 30/7/18

```
"""
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:
```

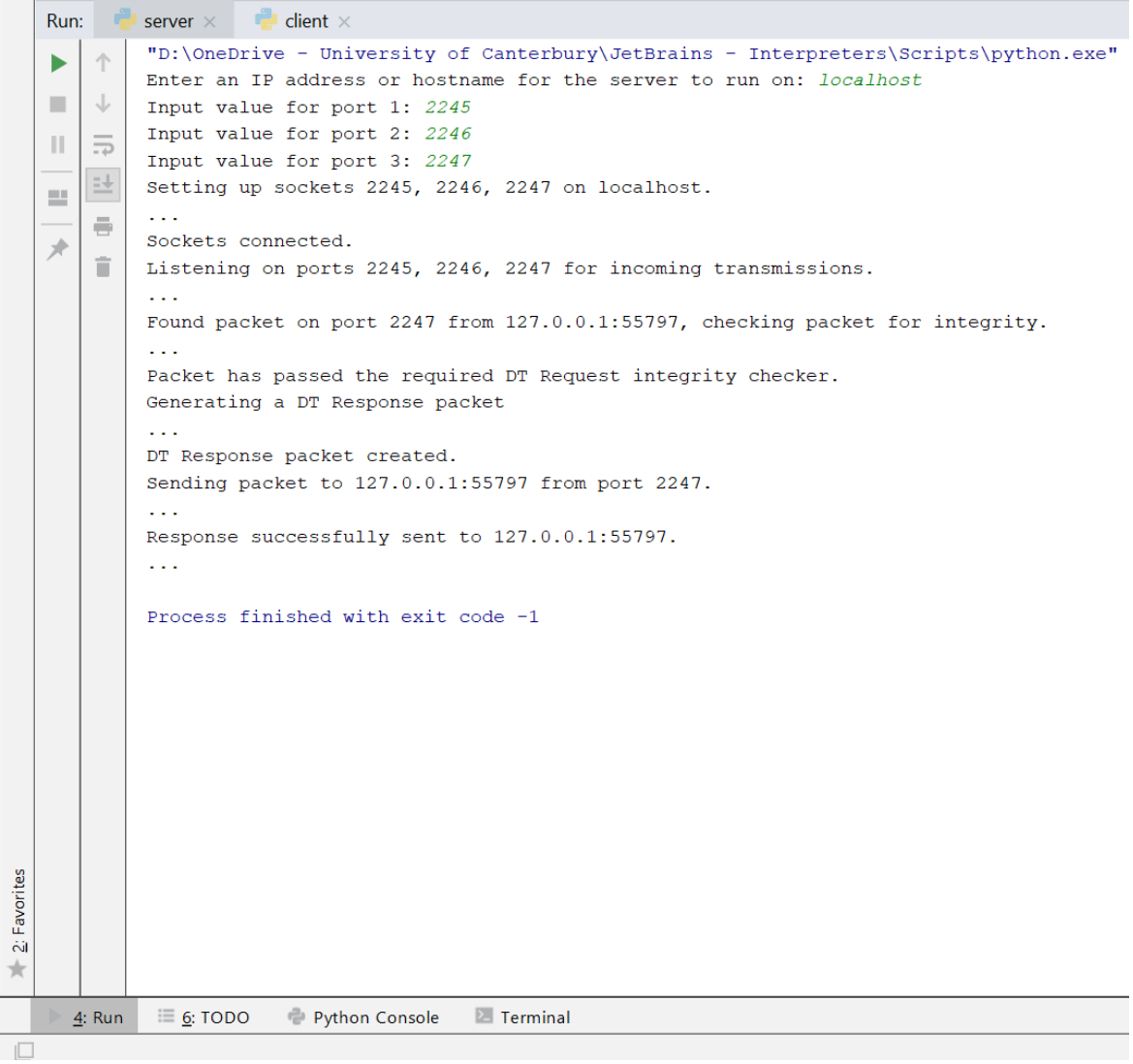
```
176         # Setting up sockets for server to run on
177         sock1, sock2, sock3 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM), \
178                               socket.socket(socket.AF_INET, socket.SOCK_DGRAM), \
179                               socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
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
```

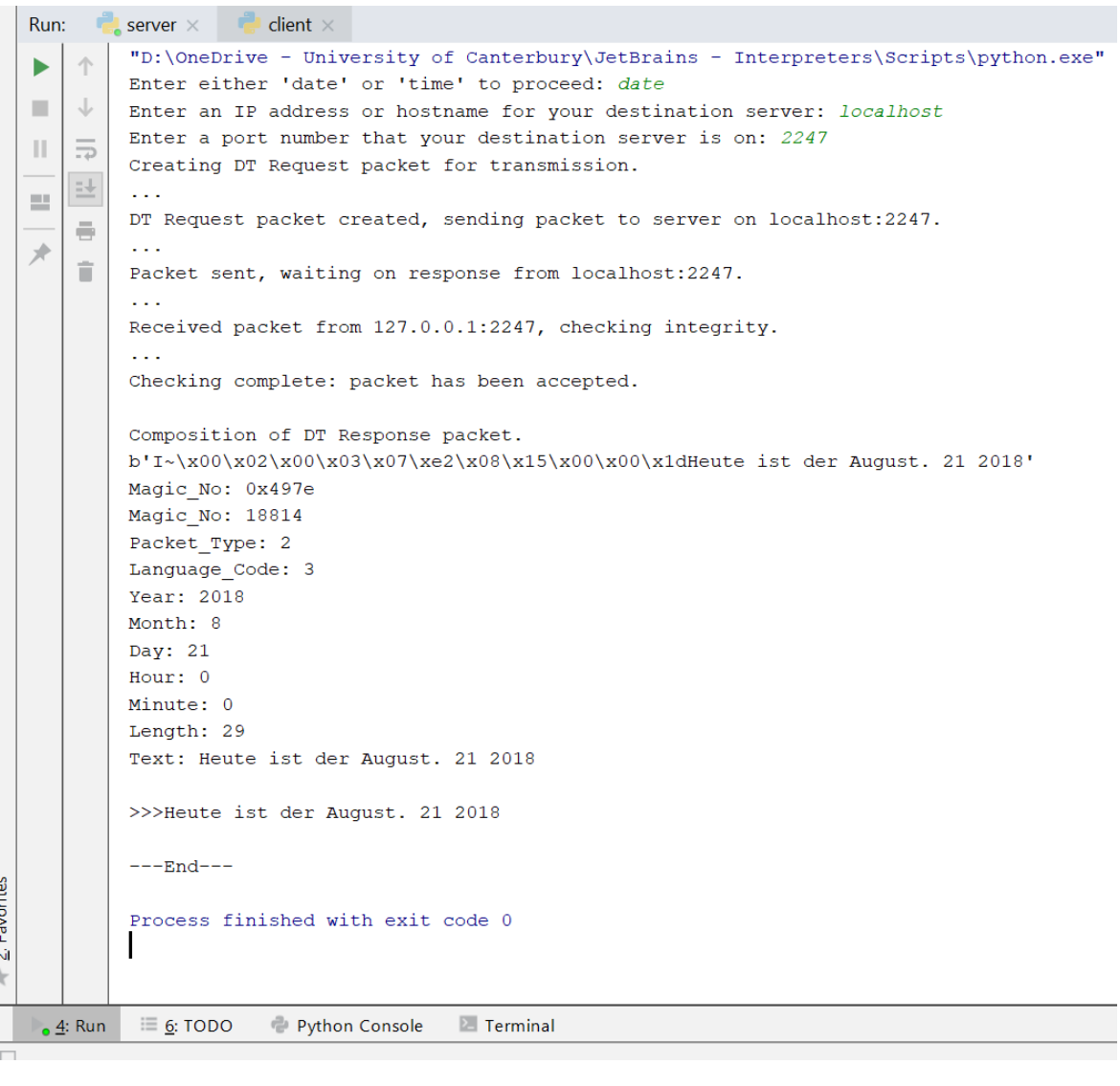
```
"""
COSC 264 - Assignment 1
Creation Date: 30/7/18
Name: Zachary Sanson
Student ID: 58520526
File: Client.py
"""
#   Note printing my program into a PDF makes a mess of my formatting


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]
```

```
63          self.month = bin_string[8:9]
64          self.day = bin_string[9:10]
65          self.hour = bin_string[10:11]
66          self.minute = bin_string[11:12]
67          self.length = bin_string[12:13]
68          self.text = bin_string[13:]
69          if len(self.length) != (13 + len(self.text)):
70              raise ValueError("DtResponse received an incorrect packet length.")
71
72      def int_in_range(self, bit, x, y):
73          """Checks if a self variable is within given range"""
74          # Returns True if in range
75          return int.from_bytes(bit, byteorder='big') in range(x, y)
76
77      def check_response(self, type):
78          """Checks if packet is a valid response packet"""
79          # We don't need to check for length < 13 as it is covered in convert_bin
80          # All ranges need to be increase by one due to pythons methods!!!
81          if not (self.magicNo == 0x497E.to_bytes(2, byteorder='big') and
82                  self.packetType == 0x0002.to_bytes(2, byteorder='big') and
83                  self.int_in_range(self.languageCode, 1, 4) and
84                  self.__len__() == (int.from_bytes(self.length, byteorder='big') + 13)):
85              raise ValueError("DT Response integrity check has failed.\n---Exiting---")
86          # Check fields corresponding to whether we wanted time or date
87          if type == 'date':
88              if not(self.int_in_range(self.year, 0, 2101) and
89                     self.int_in_range(self.month, 1, 13) and
90                     self.int_in_range(self.day, 1, 32)):
91                  raise ValueError("DT Response integrity check has failed.\n---Exiting
    ---")
92          else:
93              if not(self.int_in_range(self.hour, 0, 24) and self.int_in_range(self.minute
    , 0, 59)):
94                  raise ValueError("DT Response integrity check has failed.\n---Exiting
    ---")
95
96
97  def user_input():
98      """Prompts user for input for setup"""
99      # Defining either date or time of package
100     usr_in = input("Enter either 'date' or 'time' to proceed: ")
101     if usr_in in ['date', 'time']:
102         time = usr_in
103     else:
104         raise ValueError("input does not match.\n---Exiting---")
105     # Defining either an IP address or a hostname of destination
106     usr_in = input("Enter an IP address or hostname for your destination server: ")
107     try:
108         socket.getaddrinfo(usr_in, 00000)  # Port doesn't matter for checking address
109         destination_address = usr_in
110     except OSError:
111         print("ValueError: encountered invalid input for a destination address.\n---
    Exiting---")
112     # Defining server port number
113     usr_in = input("Enter a port number that your destination server is on: ")
114     server_port = int(usr_in)
115     if server_port not in range(1024, 64000):
116         raise ValueError("entered port number is out of range 1,024 - 64,000.\n---
    Exiting---")
117     return time, destination_address, server_port
118
119
120 def print_packet(dt_response):
121     """Prints out the DT Response packet"""
122     # Client prints entire DT Response packet???
123     # In form of an entire bytearray
124     print("\nComposition of DT Response packet.")
```

```
125     print(dt_response)
126     contents = [(dt_response.magicNo, "Magic_No: "), (dt_response.packetType, "
    Packet_Type: "),
127                 (dt_response.languageCode, "Language_Code: "), (dt_response.year, "Year
    : "),
128                 (dt_response.month, "Month: "), (dt_response.day, "Day: "),
129                 (dt_response.hour, "Hour: "), (dt_response.minute, "Minute: "),
130                 (dt_response.length, "Length: "), (dt_response.text, "Text: ")]
131     # In form of actual integers and strings
132     for value in contents:
133         if value[0] == dt_response.magicNo:
134             print(value[1] + str(hex(int.from_bytes(value[0], byteorder='big'))))
135         if value[0] == dt_response.text:
136             print(value[1] + dt_response.text.decode('utf-8'))
137         else:
138             print(value[1] + str(int.from_bytes(value[0], byteorder='big')))
139     # Print out our beautiful time/date in whatever language you want!
140     print("\n>>>" + dt_response.text.decode('utf-8'))
141
142
143 def send_to(time, server_address):
144     """Sends packet to server"""
145     print("DT Request packet created, sending packet to server on {}:{}.\n...".format(
    server_address[0], server_address[1]))
146     # Create a DT Request packet and set up sockets for transfer
147     dt_request, dt_response = DtRequest(), None
148     try:
149         sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
150     except OSError:
151         print("Could not establish an outgoing connection.\n---Exiting---")
152     if time == "date":
153         dt_request.requestType = 0x0001.to_bytes(2, byteorder='big')
154     else:
155         dt_request.requestType = 0x0002.to_bytes(2, byteorder='big')
156     sock.sendto(dt_request.packet(), server_address)
157     print("Packet sent, waiting on response from {}:{}.\n...".format(server_address[0],
    server_address[1]))
158     try:
159         # Set a timeout for the socket to one second
160         sock.settimeout(1.0)
161         received_packet, server_info = sock.recvfrom(300)
162         # We no longer need this socket so to save resources we close the socket
163         sock.close()
164         dt_response = DtResponse()
165         dt_response.convert_bin(received_packet)
166     finally:
167         # If we still come out of the try statement with no packet return an error
168         if dt_response is None:
169             raise TimeoutError("could not setup connection with {}.\n---Exiting---".
    format(server_address))
170         print("Received packet from {}:{}, checking integrity.\n...".format(server_info[
    0], server_info[1]))
171         # Packet checking
172         dt_response.check_response(time)
173         print("Checking complete: packet has been accepted.")
174         print_packet(dt_response)
175         return dt_response
176
177
178 def main():
179     """Main call to our client program"""
180     time, destination_address, server_port = user_input()
181     # Create request packet
182     print("Creating DT Request packet for transmission.\n...")
183     # Send packet to server
184     send_to(time, (destination_address, server_port))
185     print("\n---End---")
```

```
186
187
188 main()
189
```


Run:
server
client
"D:\OneDrive - University of Canterbury\JetBrains - Interpreters\Scripts\python.exe"
Enter an IP address or hostname for the server to run on: localhost
Input value for port 1: 2245
Input value for port 2: 2246
Input value for port 3: 2247
Setting up sockets 2245, 2246, 2247 on localhost.
...
Sockets connected.
Listening on ports 2245, 2246, 2247 for incoming transmissions.
...
Found packet on port 2247 from 127.0.0.1:55797, checking packet for integrity.
...
Packet has passed the required DT Request integrity checker.
Generating a DT Response packet
...
DT Response packet created.
Sending packet to 127.0.0.1:55797 from port 2247.
...
Response successfully sent to 127.0.0.1:55797.
...
Process finished with exit code -1
2: Favorites
4: Run
6: TODO
Python Console
Terminal


Run:
server
client
"D:\OneDrive - University of Canterbury\JetBrains - Interpreters\Scripts\python.exe"
Enter either 'date' or 'time' to proceed: date
Enter an IP address or hostname for your destination server: localhost
Enter a port number that your destination server is on: 2247
Creating DT Request packet for transmission.
...
DT Request packet created, sending packet to server on localhost:2247.
...
Packet sent, waiting on response from localhost:2247.
...
Received packet from 127.0.0.1:2247, checking integrity.
...
Checking complete: packet has been accepted.
Composition of DT Response packet.
b'I~\x00\x02\x00\x03\x07\xe2\x08\x15\x00\x00\x1dHeute ist der August. 21 2018'
Magic_No: 0x497e
Magic_No: 18814
Packet_Type: 2
Language_Code: 3
Year: 2018
Month: 8
Day: 21
Hour: 0
Minute: 0
Length: 29
Text: Heute ist der August. 21 2018
>>>Heute ist der August. 21 2018
---End---
Process finished with exit code 0
2: Favorites
4: Run
6: TODO
Python Console
Terminal