# Cosc264 Assignment 1

**Student: Bach Vu Viet**

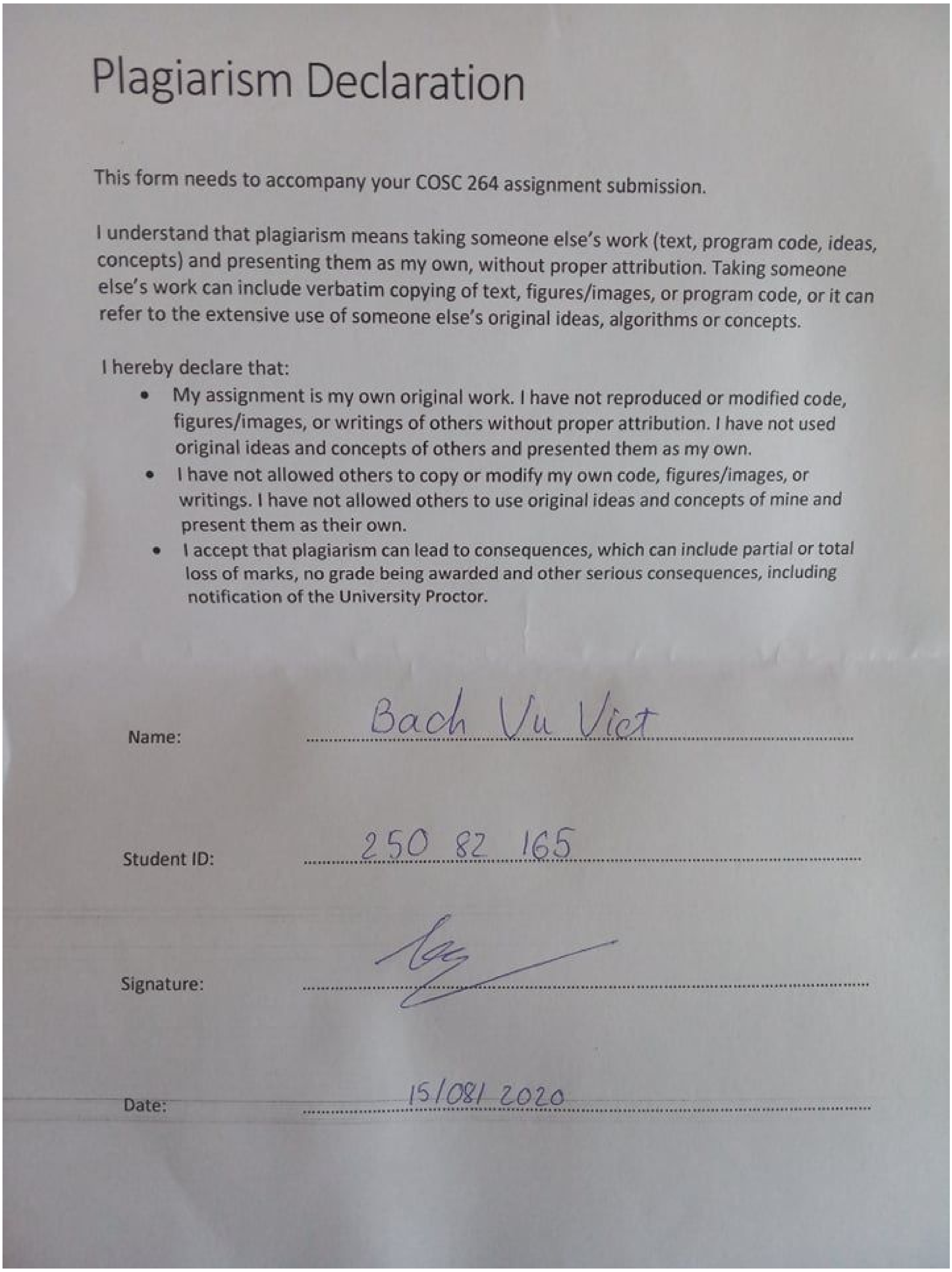
**StudentID: 25082165**

**Source included:**

- Plagiarism declaration (.pdf)
- packet.py: DT_Packet interface
- request.py: DT_Request class, to create requests -> bytearray
- response.py: DT_Response class, to decode bytearray received
- language.py: DT_Language class, support format server answer
- server.py: Program to be run on the host PC (always on)
- client.py: Program to be run on customer device (make 1 request at a time)

**Notes:**

- When create a DT_Packet, head_info are optional and assigned valude by class definition.
- When receive a DT_Packet, head_info must be not None to perform error checking, which come from static decoding method.



# 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: | Bach Vu Viet |
| Student ID: | 250 82 165 |
| Signature: | [illegible] |
| Date: | 15/08/ 2020 |

```python
# Packet Interface (packet.py)
# Bach Vu
# 01/08/2020

from abc import abstractmethod

class DT_Packet:
    def __init__(self, packetType):
        self.MagicNum = 0x497E
        self.packetType = packetType

    @staticmethod
    def intToBinStr(decimal, str_len):
        return bin(decimal)[2:].zfill(str_len)

    @staticmethod
    def byteArrToInt(byte):
        return int.from_bytes(byte, byteorder="big")

    @staticmethod
    def DT_hex(num):
        return '0x000' + str(num)

    def isValid(self):
        """ Check conditions of a Packet Type """
        return self.header_errorCode()

    @abstractmethod
    def __repr__(self):
        """ Output log """

    @abstractmethod
    def header_errorCode(self):
        """ Check conditions of a Packet Type """

    @abstractmethod
    def encodePacket(self):
        """ Get the actual bytearray store data of this packet """

    @abstractmethod
    def decodePacket(packet):
        """ Turn bytearray to object """





```

```python
# Packet Structure (request.py)
# Bach Vu
# 01/08/2020

from packet import *

class DT_Request(DT_Packet):
    ErrorMessage = [
        "Expecting received packet have MagicNum 0x497E.",
        "Expecting received packet have packetType 0x0001.",
        "Undefined outupt Type [date/time]?",
        "Packet header is shorter than expected"
    ]
    def __init__(self, mode, head_info=None):
        self.requestType = mode # 0x0001 or 0x0002
        if head_info is None:
            super().__init__(0x0001)
        else:
            self.MagicNum   = head_info[0]
            self.packetType = head_info[1]

    def __repr__(self):
        out = "<Magic: {}> <packetType: {}> <requestType: {}>\n"
        out = out.format(hex(self.MagicNum), DT_Packet.DT_hex(self.packetType), 
DT_Packet.DT_hex(self.requestType))
        out += type(self).__name__ + " with request type: " + ("DATE" if self.requestType==1 
else "TIME")
        return out

    def header_errorCode(self):
        error_code = 0
        if self.MagicNum != 0x497E:
            error_code = 1
        elif self.packetType != 0x0001:
            error_code = 2
        elif self.requestType < 0x0001 or self.requestType > 0x0002:
            error_code = 3

        return error_code

    def encodePacket(self):
        # Error check
        check = self.isValid()
        if check != 0:
            return check

        # Header
        header = ""
        header += DT_Packet.intToBinStr(self.MagicNum,16)
        header += DT_Packet.intToBinStr(self.packetType,16)
        header += DT_Packet.intToBinStr(self.requestType,16)
        header  = int(header, 2).to_bytes(6, byteorder='big')

        # Pack
        packet = bytearray()
        packet += header
        return packet

    @staticmethod
    def decodePacket(packet):
        if len(packet) < 6:
            return 4

        magic    = DT_Packet.byteArrToInt(packet[0:2])
        packType = DT_Packet.byteArrToInt(packet[2:4])
        mode = DT_Packet.byteArrToInt(packet[4:6])
        requestPack = DT_Request(mode)

        # Error check
        param = [magic, packType]
        requestPack = DT_Request(mode, tuple(param))
        check = requestPack.isValid()
        if check != 0:
            return check
        return requestPack

```

```python
# Packet Structure (response.py)
# Bach Vu
# 01/08/2020

from packet import *
from datetime import datetime
from language import DT_Language

class DT_Response(DT_Packet):
    ErrorMessage = [
        "Expecting received packet have MagicNum 0x497E.",
        "Expecting received packet have packetType 0x0002.",
        "Undefined language outupt Type [Eng/Maori/Ger]?",
        "Year is over 2100. Data received must be invalid.",
        "Month is not between 1 and 12. Data received must be invalid.",
        "Day is not between 1 and 31. Data received must be invalid.",
        "Hour is not between 0 and 23. Data received must be invalid.",
        "Minute is not between 0 and 59. Data received must be invalid.",
        "Some data of displaying message is missing",
        "Packet header is shorter than expected"
    ]

    def __init__(self, language, mode, head_info=None):
        self.language = language

        if head_info is None:
            super().__init__(0x0002)
            now = datetime.now() # Time when obj created
            self.time = [now.year, now.month, now.day, now.hour, now.minute]
            dt = DT_Language(language, mode, self.time)
            self.message = dt.DTtoString().encode('utf8')
            self.m_len = len(self.message)
        else:
            self.MagicNum   = head_info[0]
            self.packetType = head_info[1]
            self.time       = head_info[2]
            self.message    = head_info[3]
            self.m_len      = head_info[4]

    def __repr__(self):
        out = "{}\n<Magic: {}> <packetType: {}> <lang: {}>\n<Time: {}> <MessLen: {}>"
        mess = type(self).__name__ + ": " + str(self.message, 'utf-8')
        return out.format(mess, hex(self.MagicNum),
                DT_Packet.DT_hex(self.packetType),
                DT_Packet.DT_hex(self.language),
                self.time, self.m_len)

    def header_errorCode(self):
        error_code = 0
        if self.MagicNum != 0x497E:
            error_code = 1
        elif self.packetType != 0x0002:
            error_code = 2
        elif self.language < 0x0001 or self.language > 0x0003:
            error_code = 3
        elif self.time[0] < 0 or self.time[0] > 2100:
            error_code = 4
        elif self.time[1] < 1 or self.time[1] > 12:
            error_code = 5
        elif self.time[2] < 1 or self.time[2] > 31:
            error_code = 6
        elif self.time[3] < 0 or self.time[3] > 23:
            error_code = 7
        elif self.time[4] < 0 or self.time[4] > 59:
            error_code = 8
        elif self.m_len != len(self.message):
            error_code = 9
        return error_code

    def encodePacket(self):
        """ Get the actual bytearray store data of this packet """
        # Error check
        check = self.isValid()
        if check != 0:
            return check

        # Header
        header = ""
        header += DT_Packet.intToBinStr(self.MagicNum,16)
        header += DT_Packet.intToBinStr(self.packetType,16)
        header += DT_Packet.intToBinStr(self.language,16)
        header += DT_Packet.intToBinStr(self.time[0],16)
        header += DT_Packet.intToBinStr(self.time[1],8)
        header += DT_Packet.intToBinStr(self.time[2],8)
        header += DT_Packet.intToBinStr(self.time[3],8)
        header += DT_Packet.intToBinStr(self.time[4],8)
        header += DT_Packet.intToBinStr(self.m_len,8)
        header  = int(header, 2).to_bytes(13, byteorder='big')

        # Pack
        packet = bytearray()
        packet += header
        packet += self.message
        return packet

    @staticmethod
    def decodePacket(packet, mode):
        if len(packet) < 13:
            return 10

        """ Turn bytearray to object """
        magic    = DT_Packet.byteArrToInt(packet[0:2])
        packType = DT_Packet.byteArrToInt(packet[2:4])
        language = DT_Packet.byteArrToInt(packet[4:6])
        year     = DT_Packet.byteArrToInt(packet[6:8])
        month    = DT_Packet.byteArrToInt(packet[8:9])
        day      = DT_Packet.byteArrToInt(packet[9:10])
        hour     = DT_Packet.byteArrToInt(packet[10:11])
        minute   = DT_Packet.byteArrToInt(packet[11:12])
        length   = DT_Packet.byteArrToInt(packet[12:13])

        time = [year, month, day, hour, minute]
        mess = packet[13:]

        # Error check
        param = [magic, packType, time, mess, length]
        responsePack = DT_Response(language, mode, tuple(param))
        check = responsePack.isValid()
        if check != 0:
            return check
        return responsePack
```

```python
# coding= utf-8
# DT Language Structure (language.py)
# Bach Vu
# 01/08/2020

from datetime import datetime

class DT_Language:
    def __init__(self, langMode, outputType, time):
        # (0x0001:Eng, 0x0002:Maori, 0x0003:Ger)
        self.language = langMode - 1
        self.mode = outputType - 1
        self.time = time
        self.stringFormats = [
            ["Today's date is {} {}, {}", "The current time is {:02d}:{:02d}"],
            ["Ko te ra o tenei ra ko {} {}, {}", "Ko te wa o tenei wa {:02d}:{:02d}"],
            ["Heute ist der {}. {} {}", "Die Uhrzeit ist {:02d}:{:02d}"]
        ]
        self.Months = [
            ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"],
            ["Kohitātea", "Hui-tanguru", "Poutū-te-rangi", "Paenga-whāwhā", "Haratua", "Pipiri", "Hōngongoi", "Here-turi-kōkā", "Mahuru", "Whiringa-ā-nuku", " Whiringa-ā-rangi", "Hakihea"],
            ["Januar", "Februar", "März", "April", "Mai", "Juni", "Juli", "August", "September", "Oktober", "November", "Dezember"]
        ]

    def DTtoString(self):
        day, month, year = self.time[2], self.time[1], self.time[0]
        hour, minute = self.time[3], self.time[4]
        output = self.stringFormats[self.language][self.mode]
        if self.mode == 0:
            month_str = self.Months[self.language][month-1]
            if self.language != 2:
                output = output.format(month_str, day, year)
            else:
                output = output.format(day, month_str, year)
        elif self.mode == 1:
            output = output.format(hour, minute)
        return output

def test():
    dt = [2020,8,8,7,0]
    lang1 = DT_Language(0x0001, 0x0001, dt)
    print(lang1.DTtoString())
    lang1 = DT_Language(0x0001, 0x0002, dt)
    print(lang1.DTtoString())
    lang1 = DT_Language(0x0002, 0x0001, dt)
    print(lang1.DTtoString())
    lang1 = DT_Language(0x0002, 0x0002, dt)
    print(lang1.DTtoString())
    lang1 = DT_Language(0x0003, 0x0001, dt)
    print(lang1.DTtoString())
    lang1 = DT_Language(0x0003, 0x0002, dt)
    print(lang1.DTtoString())

if __name__ == "__main__":
    test()

```

```python
# Server Application (server.py)
# Bach Vu
# 01/08/2020

from request import *
from response import *
from socket import *
import sys, select

class DTServer():
    def __init__(self, hostname):
        self.sockets = [["English","Maori","German"], [None,None,None]]
        self.requests = [] # (byte_array, output_lang, sender_ip)
        self.hostName = hostname
        print("Server started with host name '{}'".format(hostname))

    def createSocket(self, ports):
        try:
            for i in range(3):
                sock = socket(AF_INET, SOCK_DGRAM)
                sock.bind((self.hostName, ports[i]))
                self.sockets[1][i] = sock
                print("Port {} is ready to receive {} requests".format(ports[i],
self.sockets[0][i]))
            return True
        except Exception as e:
            raise e

    def shutdown(self):
        for socket in self.sockets[1]:
            socket.close()

    def getRequest(self):
        # get socket has buffer increase (new request)
        readable, _, _ = select.select(self.sockets[1], [], [])
        for sock in readable:
            option = -1
            if sock is self.sockets[1][0]:
                option = 0  # 0x0001 for English
            elif sock is self.sockets[1][1]:
                option = 1  # 0x0002 for Maori
            elif sock is self.sockets[1][2]:
                option = 2  # 0x0003 for German
            data, ip_sender = self.sockets[1][option].recvfrom(1024) # in byte
            self.requests.append( (data, option+1, ip_sender) )

    def sendResponse(self, response, target, s_ID):
        packet = response.encodePacket()
        if isinstance(packet, bytearray):
            socket = self.sockets[1][s_ID]
            socket.sendto(packet, target)
            print("Responded to sender at: {}".format(target))
        else:
            print("Respond failed with code {}! Try again ... ".format(packet))

####################### Main Program #################
def mainloop(server):
    while True:
        print("\nWaiting DT_request")
        server.getRequest()
        while len(server.requests) > 0:
            # Receive request
            packet = server.requests.pop(0)
            request = DT_Request.decodePacket(packet[0])
            if isinstance(request, int):
                err = "A request discarded with error Code {}:\n{}"
                err_mess = DT_Request.ErrorMessage[request-1]
                print(err.format(int(request), err_mess))
                continue
            print(request)

            # Reply
            print("Preparing response in {}.".format(server.sockets[0][packet[1]-1]))
            response = DT_Response(packet[1], request.requestType)
            server.sendResponse(response, packet[2], packet[1]-1)

def checkInputArgv():
    if len(sys.argv) != 4:
        return 1

    ports = []
    try:
        ports = [int(sys.argv[1]), int(sys.argv[2]), int(sys.argv[3])]
    except BaseException:
        return 2

    for port in ports:
        if port < 1024 or port > 64000:
            return 3
    return 0

def startServer():
    print("\nWelcome to DT Finder (Server)")
    # Error Checking
    errMess = [
        "Argument input error.\n    python server.py {host} {port_eng} {port_maori} {port_ger}",
        "Ports input must be integer (whole number).",
        "Port must be between 1024 and 64000 inclusively!"
    ]
    errCode = checkInputArgv()
    if errCode != 0:
        print(errMess[errCode-1])
        sys.exit()

    # Create Server instance
    host = getfqdn()
    ports = [int(sys.argv[1]), int(sys.argv[2]), int(sys.argv[3])]
    server = DTServer(host)
    server.createSocket(ports)
    return server

if __name__ == "__main__":
    try:
        DT_server = startServer()
        mainloop(DT_server)
    except KeyboardInterrupt:
        DT_server.shutdown()
        print("Program exited!")
    except Exception as e:
        print(e)
        print()
```

```python
# Client Application (client.py)
# Bach Vu
# 01/08/2020

from request import *
from response import *
from socket import *
import sys, select

class DTClient():
    def __init__(self, hostname, port):
        self.socket = socket(AF_INET, SOCK_DGRAM)
        self.socket.setblocking(0)
        self.target = None
        try:
            hostname = getaddrinfo(hostname, port)[0][4][0]
            self.target = (hostname, port)
        except Exception as e:
            raise e

    def postRequest(self, request):
        packet = request.encodePacket()
        if isinstance(packet, bytearray):
            self.socket.sendto(packet, self.target)
            print("Request sent to {}:{}! Waiting for response ... ".format(self.target[0],
self.target[1]))
        else:
            print("Request sent failed with code {}! Try again ... ".format(packet))

    def getResponse(self):
        ready = select.select([self.socket], [], [], 1)
        if ready[0]:
            data, addr = self.socket.recvfrom(1024)
            return data
        return None

################## Main Program ##################
def main():
    print("\nWelcome to DT Finder (Client)")
    # Error Checking
    errMess = [
        "Argument input error.\n    python client.py {mode} {host_target}
{port_eng/maori/ger}",
        "Ports input must be integer (whole number).",
        "Port must be between 1024 and 64000 inclusively!",
        "mode must be 'time' or 'date'."
    ]
    errCode = checkInputArgv()
    if errCode != 0:
        print(errMess[errCode-1])
        sys.exit()

    # Request
    host, port = sys.argv[2], int(sys.argv[3])
    DT_client = DTClient(host, port)
    mo = 1 if sys.argv[1] == 'date' else 2
    request = DT_Request(mo)
    DT_client.postRequest(request)

    # Response
    response = DT_client.getResponse()
    if not response: # None type
        print("We received no data after 1sec (time-out).\n")
        return
    response = DT_Response.decodePacket(response, mo)

    if isinstance(response, int):
        err = "A response discarded with error Code {}:\n{}\n"
        err_mess = DT_Response.ErrorMessage[response-1]
        print(err.format(response, err_mess))
    elif response:
        print(response)
        print()

def checkInputArgv():
    if len(sys.argv) != 4:
        return 1

    mode = sys.argv[1]
    try:
        port = int(sys.argv[3])
    except BaseException:
        return 2

    if port < 1024 or port > 64000:
        return 3
    if mode != "date" and mode != "time":
        return 4
    return 0

if __name__ == "__main__":
    try:
        main()
    except Exception as e:
        print(e)
        print("Program exited unexpectedly.\n")

```