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: Oliver Garrett (oga20)

Student ID: 13019796

Signature: [illegible]
………

Date: 1/09/2021

COSC264 Assignment 1

In my codebase, there are 6 files:
server.py, **client.py**, **bithelper.py**, **request.py**, **response.py**, and **safesocket.py**.

bithelper, **request**, **response**, and **safesocket** must be accessible by both **server** and **client**.

client.py

```python
'''
client.py

Client python file

Author:
Oliver Garrett (oga20)

'''


from safesocket import safesocket
import socket
import request
import response
import os
import sys
import time


MAX_RECV = 4096

IP = socket.gethostbyname(socket.gethostname())


def init(ipv4, port):
    sock = safesocket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(1)
    try:
        sock.connect((ipv4, port))
        return sock
    except BaseException as e:
        sock.close()
        print("Client: Socket connection failed")
        print("Error: " + str(e))
        exit(-4)


def request_file(sock, fname):
    if os.path.exists(fname):
        return False
    packet = request.make(fname)
    sock.sendall(packet)
    return True


def getdata(sock):
    try:
        rpacket = sock.recv(MAX_RECV)
    except socket.timeout:
        print("Client: Socket timed out; server is likely busy or down.")
        exit(-5)
    return rpacket

def start(sock):
    rpacket = getdata(sock)
    status, dlen, data = response.unmake(rpacket)
    if not status:
        print("Client: Server file not available or malformed packet.")
        exit(-5)

    return dlen, data


def write(sock, file):
    response = getdata(sock)
    file.write(response)
    return len(response)


def run(sock, fname):
    print("Client: started...")
    success = request_file(sock, fname)

    if not success:
        print("Client: File already exists, will not overwrite")
        exit(-7)

    print("Client: Success in requesting file")
    data_left, filedata = start(sock)
    data_left -= len(filedata)

    with open(fname, "ab+") as file:
        file.write(filedata)

        while data_left > 0:
            datalen = write(sock, file)
            data_left -= datalen



def isip(ip):
    return ip.replace(".", "").isdigit()


def main():
    ipv4 = sys.argv[1]
    if not isip(ipv4):
        try:
            ipv4 = socket.getaddrinfo(ipv4)
        except:
            print(f"Client: Malformed or non existant domain: {ipv4}")
            exit(-1)

    portn = sys.argv[2]
    if not (portn.isdigit() and (1024 <= int(portn) <= 64000)):
        print("Client: Port number not an intger between 1024 and 64000")
        exit(-2)

    filename = sys.argv[3]
    if  os.path.isfile(filename):
        print(f"Client: Aborted! Overwriting file: {filename}")
        exit(-3)

    sock = init(ipv4, int(portn))
    run(sock, filename)

if __name__ == "__main__":
    main()
```

server.py

```
'''
Server file

Author:
Oliver Garrett
(oga20)

'''

import sys

sys.path.append("D:\\PROGRAMMING\\UNIVERSITY\\COSC264\\proj")
#  Temporary for my computer

import socket
from safesocket import safesocket
import time
import request
import response
import os
import math


MAX_RECV = 4096
```

```python
TIMEOUT = socket.timeout

socket.setdefaulttimeout(30)

def exit(*a,**ka): input("Halted")

def get_ip():
    return socket.gethostbyname(socket.gethostname())


def init(port):
    global sock
    sock = safesocket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        sock.bind((get_ip(), port))
        return sock
    except Exception as e:
        sock.close()
        print(f"Server: Error binding socket on port {port}: \n{e}")
        exit()



def sendbad(sock):
    packet = response.make(0, bytearray())
    sock.sendall(packet)


def gettime():
    return time.strftime('%X %x %Z')


def sendall(newsock, packet):
    iter = math.ceil(len(packet) / MAX_RECV)
    for i in range(iter):
        newsock.sendall(packet[i * MAX_RECV : (i + 1) * MAX_RECV])


def loop(sock):
    try:
        (newsock, addr) = sock.safeaccept()
        newsock.settimeout(1)
    except BaseException as e:
        print("Server: accept refused to work" + str(e))
        exit()

    t1 = gettime()
    print(f"Server: Accepted connection from {addr} at {t1}")

    try:
        success, fnamelen, fname = request.unmake(newsock.recv(MAX_RECV))
    except socket.timeout as e:
        print(f"Server: client stopped responding. Resetting\n")
        newsock.close()
        return

    if success != 1:
        print("Server: client packet is incorrect")
        return sendbad(sock)

    if fnamelen != len(fname):
        print("Server: filename does not agree with filename length")
        return sendbad(sock)

    if not os.path.exists(fname):
        print("Server: non existant file")
        return sendbad(sock)

    with open(fname, "rb") as f:
        t = time.time()
        byts = f.read()
        packet = response.make(1, byts)
        t1 = time.time()
        print("Server: time taken to read: " + str(t1-t))
        try:
            # previously: newsock.sendall(packet)
            sendall(newsock, packet)
            print(f"Server: File successfully sent to {addr} at {gettime()}\n")
        except socket.timeout as e:
            print(f"Server: client stopped responding\n")
            newsock.close()



def run(sock):
    try:
        sock.listen()
    except:
        print("Server: Listen failed.")
        sock.close()
        exit()

    print("Server: Listening on ")
    print("Server: waiting for clients:\n")

    while True:
        loop(sock)



def main():
    inp = sys.argv[1]

    if (not inp.isdigit()):
        print("Server: Wrong input format. Expected integer.")

    port = int(inp)
    if not (1024 <= port <= 64000):
        print("Server: input error: expected a port number between 1024 and 64000 inclusiv
e.")
        exit(-1)

    sock = init(port)
    print(f"Server: Running server on {get_ip()}, {port}")
    run(sock)


if __name__ == "__main__":
    main()
```

request.py

```
'''
request.py

File for handling FileRequest packets

Author:
Oliver Garrett
(oga20)

'''


from bithelper import *


MAGIC_NUMBER = 0x497E
TYPE = 1 # type 1 is FileRequest


def make(filename):
    arr = bytearray()
    push16(arr, MAGIC_NUMBER)
    arr.append(TYPE)
    filenameLen = len(filename)
    push16(arr, filenameLen)
    if filenameLen > 1024:
        print("filename length too big")
        exit(1)

    for char in filename:
        arr.append(ord(char))

    return arr
```

```python
def unmake(bytes_) -> tuple([int, int, str]):
    arr = bytearray(bytes_)
    magic = get16(arr, 0)
    type_ = int(arr[2])
    filenameLen = get16(arr, 3)

    status = 1
    if magic != MAGIC_NUMBER or type_ != TYPE:
        status = 0

    fname = ''
    for char in arr[5:]:
        fname += chr(char)

    return (status, filenameLen, fname)


# Testing
if __name__ == "__main__":
    print("FileRequest: Running tests:")
    for x in ["djkfkdfkdfjkalndwoiejd.png", "q", "UUUUUUUUUU......ffff.cls"]:
        assert unmake(make(x))[-1] == x, str((unmake(make(x)), x))
    print("FileRequest: Tests passed.")
```

response.py

```python
'''
response.py

File for handling FileResponse packets

Author:
Oliver Garrett
(oga20)

'''


from bithelper import *
import random

MAGIC_NUMBER = 0x497E

TYPE = 2 # type 2 is FileResponse


def make(status, data : bytes):
    # if status is 0, failed, if status is 1, success!
    arr = bytearray()
    datalen = len(data)
    push16(arr, MAGIC_NUMBER)
    arr.append(TYPE)
    arr.append(status)

    push32(arr, datalen)
    return arr + data



def unmake(bytes_) -> tuple([int, int, bytearray]):
    arr = bytearray(bytes_)
    magic = get16(arr, 0)

    type_ = int(arr[2])
    if type_ != TYPE or magic != MAGIC_NUMBER:
        status = 0
    else:
        status = int(arr[3])
    datalen = get32(arr, 4)

    return (status, datalen, arr[8:])




# Testing
if __name__ == "__main__":
    print("FileResponse: Running tests:")
    for x in range(100):
        status = random.randint(0,20)
        data = bytearray(''.join(chr(random.randint(0,50)) for _ in range(10))
                         ,encoding='utf-8')
        assert unmake(make(status, data)) == (status, len(data), data), \
                (unmake(make(status, data)), (status, len(data), data))

    print("FileResponse: Tests passed")
```

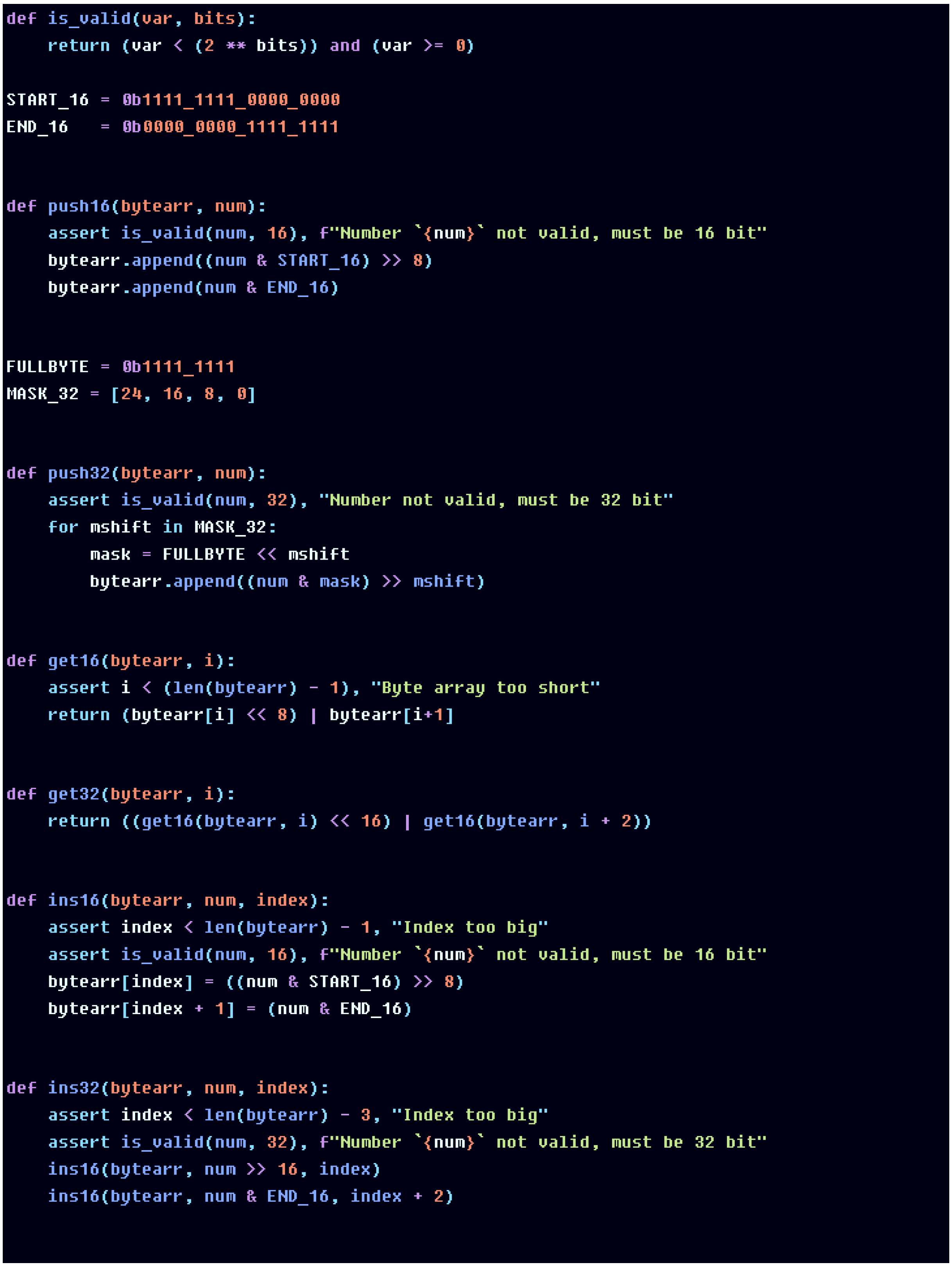
bithelper.py

```python
"""
bithelper.py

This module is used to aid in the process of creating, reading, and
modifying byte-like objects. (Primarily python bytearrays)

Author:
Oliver Garrett
oga20

"""
```



```python
def is_valid(var, bits):
    return (var < (2 ** bits)) and (var >= 0)

START_16 = 0b1111_1111_0000_0000
END_16   = 0b0000_0000_1111_1111


def push16(bytearr, num):
    assert is_valid(num, 16), f"Number `{num}` not valid, must be 16 bit"
    bytearr.append((num & START_16) >> 8)
    bytearr.append(num & END_16)


FULLBYTE = 0b1111_1111
MASK_32 = [24, 16, 8, 0]


def push32(bytearr, num):
    assert is_valid(num, 32), "Number not valid, must be 32 bit"
    for mshift in MASK_32:
        mask = FULLBYTE << mshift
        bytearr.append((num & mask) >> mshift)


def get16(bytearr, i):
    assert i < (len(bytearr) - 1), "Byte array too short"
    return (bytearr[i] << 8) | bytearr[i+1]


def get32(bytearr, i):
    return ((get16(bytearr, i) << 16) | get16(bytearr, i + 2))


def ins16(bytearr, num, index):
    assert index < len(bytearr) - 1, "Index too big"
    assert is_valid(num, 16), f"Number `{num}` not valid, must be 16 bit"
    bytearr[index] = ((num & START_16) >> 8)
    bytearr[index + 1] = (num & END_16)


def ins32(bytearr, num, index):
    assert index < len(bytearr) - 3, "Index too big"
    assert is_valid(num, 32), f"Number `{num}` not valid, must be 32 bit"
    ins16(bytearr, num >> 16, index)
    ins16(bytearr, num & END_16, index + 2)
```

safesocket.py

```python
'''
Safesocket file

Automatically closes sockets on python exit.

Author:
Oliver Garrett
(oga20)

'''


import socket
import atexit

class safesocket(socket.socket):
    '''
    safesocket class.
    Automatically closes sockets when python exits
    '''
    # A list of all sockets
    sbuffer = []

    def __init__(self, *a, **ka):
        super().__init__(*a, **ka)
        safesocket.sbuffer.append(self)
        self.setblocking(True)

    def safeaccept(self):
        '''
        Modifies accept
        '''
        (newsock, addr) = super().accept()
        safesocket.sbuffer.append(newsock)
        newsock.setblocking(True)
        return newsock, addr


def clearbuffer():
    for s in safesocket.sbuffer:
        s.close()

atexit.register(clearbuffer)
```