3 HTTP website results using Netcat:

```
Berks-MacBook-Pro-3:~ berkturkcapar$ nc -l 12345
GET http://meb.gov.tr/ HTTP/1.1
Host: meb.gov.tr
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1

Berks-MacBook-Pro-3:~ berkturkcapar$ nc -l 12345
GET http://yemeksepeti.com/ HTTP/1.1
Host: yemeksepeti.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1

Berks-MacBook-Pro-3:~ berkturkcapar$ nc -l 12345
GET http://boun.edu.tr/ HTTP/1.1
Host: boun.edu.tr
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Code Explanation:

I have decided to code in Python and I have used the `socket` module for listening to the port and sending GET requests and used `sys` module to read command line arguments (<port>). I also used the `datetime` module to compare the Last-Modified information of responses. My code consists of two functions: the `main` function and `download_file` function. For an additional feature, I have implemented a cache structure which checks if a file has already been downloaded. If it is in the cache it checks the Last-Modified information and downloads the file if the cached version is old.

Main Function:

```python
if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python ProxyDownloader.py <port>")
        sys.exit(1)
    proxy_port = sys.argv[1]
    # Initialize the cache as empty dictionary
    cache = {}
    # Create a socket and listen on the designated port
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind(("", int(proxy_port)))
    s.listen()
    print(f"Listening on port {proxy_port}...")
```

In the beginning of the main function, the length of the command line arguments is checked to make sure the program is run with a proper command. If the length is not 2 then an error message is printed and the program is closed. If there is no error the port number is obtained from the command line by using `sys.argv[1]`. Next, a socket called `s` is created to listen to the designated port and a message is printed notifying the user the program is listening to the port for incoming messages. s is bound to the specified port number by using `bind()` method. I did not specify any IP address as the first argument of the `bind()` method.

```
while True:
    print("Waiting for a connection...")
    conn, addr = s.accept()
    # Read the HTTP request
    request = b""
    while True:
        data = conn.recv(1024)
        request += data
        if b"\r\n\r\n" in request:
            break
    # Parse the URL from the HTTP request
    url = request.split(b"\r\n")[0].split(b" ")[1].decode()
    host_name = url.split("/")[2]
    print(f"Retrieved request from Firefox:")
    if host_name != 'www.cs.bilkent.edu.tr':
        print('WARNING: Host name is not www.cs.bilkent.edu.tr, moving on...\n')
        continue
    print(request.decode())
    # Download the file
    download_file(url, cache)
```

In this infinite loop, the program waits for incoming connections to the designated port. `accept()` method blocks and waits for any incoming connections to the socket. When a connection is received, it returns an object with two properties: `conn` which is used to communicate with the client and `addr` which returns the IP address and the port of the client. Next I created a byte string called `request` in order to read the incoming connection request. Inside a while loop, I obtained the data from the socket in chunks of 1024 bytes by using the `recv()` method. I checked if the end of request was reached and called `break` to end the loop. Next from `request` I parsed the `url` and `host_name`. For filtering any unnecessary request I checked if the `host_name` is `www.cs.bilkent.edu.tr`. If the request is from a different host this request is not processed and the user is notified about this. In order to send the HTTP GET request, download and save the file I created another function called download_file.

download_file Function:

```
def download_file(url, cache):
   # Parse the filename from the URL
   file_name = url.split("/")[-1]
   host_name = url.split("/")[2]
   s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   s.connect((host_name, 80))
   if file_name in cache:
       cache_last_modified = cache[file_name]
       print(f"Already downloaded '{file_name}' before...")
       print(f"Checking the Last-Modified information...")
            head_request = b"HEAD " + url.encode() + b" HTTP/1.1\r\nHost: " +
host_name.encode() + b"\r\nConnection: close\r\n\r\n"
       s.sendall(head_request)
       response = b""
       while True:
           data = s.recv(1024)
           if not data:
               break
           response += data
       # Check the status code and status message
       response_lines = response.decode().split("\r\n")
       status_line = response_lines[0]
                 status_code, status_message = int(status_line.split()[1]), "
".join(status_line.split()[2:])
       print(f"Retrieved: {status_code} {status_message}")
       # Check if response is OK and get the last modified time of the server file
       if status_code == 200:
           last_modified = response_lines[3].split(": ")[1].strip()
            last_modified = datetime.strptime(last_modified, "%a, %d %b %Y %H:%M:%S
%Z")
           # Compare the last modified times of the cached file and server file
           if last_modified <= cache_last_modified:
                   print("File is up-to-date in the cache, no need to send a new
request...")
               print("Moving on...\n")
               s.close()
               return
           else:
                    print("The file is not up-to-date in the cache, sending a new
request...")
       else:
           print("ERROR: Couldn't check the server for Last-Modified\n")
           s.close()
           return
```

The `download_file()` function accepts one input which is the byte string called `url`. First I parsed the `file_name` and the `host_name` from the url. Next, I created a socket object and connected it to port 80 which is the default port for HTTP requests. First, I checked if the file already exists in the dictionary. If it did I sent a HTTP HEAD request in order to obtain the Last-Modified information of the file. I used a HEAD request since it is faster than a GET request and I did not want to send an unnecessary GET request which may lower performance. I checked if the response was OK and if it was I parsed the Last-Modified information. I checked if the `last_modified` date from the HEAD request is newer than the `cache_last_modified`. If it is not newer I printed a message stating the file in the cache is up to date. If it is newer it goes on to send the GET request. Before all return statements I closed the socket binded to port 80.

```
    # Send an HTTP GET request to download the file
        request_body = b"GET " + url.encode() + b" HTTP/1.1\r\nHost: " +
host_name.encode() + b"\r\nConnection: close\r\n\r\n"
    s.sendall(request_body)
    response = b""
    while True:
        data = s.recv(1024)
        if not data:
            break
        response += data
    # Check the status code and status message
    response_lines = response.decode().split("\r\n")
    status_line = response_lines[0]
            status_code, status_message = int(status_line.split()[1]), "
".join(status_line.split()[2:])
    print(f"Retrieved: {status_code} {status_message}")
    # Check if response is OK
    if status_code != 200:
        print("ERROR: File not found, moving on...\n")
        return
    last_modified = response_lines[3].split(": ")[1].strip()
    cache[file_name] = datetime.strptime(last_modified, "%a, %d %b %Y %H:%M:%S %Z")
    # Save the file
    f = open(file_name, "wb")
    f.write(response.split(b"\r\n\r\n")[1])
    print(f"Downloading file '{file_name}'...")
    print("Saving file...\n")
    s.close()
```

Next I created the `request_body` as a byte string and used `sendall()` method to send the HTTP GET request. Next I created a byte string called `response` in order

to read the response of the request I have sent. Inside a while loop, I obtained the data from the socket in chunks of 1024 bytes by using the `recv()` method. I checked if the end of `response` was reached and called `break` to end the loop. After that, after sending the request and obtaining the response I parsed the response to check the `status_code` and `status_message`. Next, I checked if the `status_code` was 200 and if it was not 200 I printed an error message. If it was 200, I opened a file with the name `file_name` and wrote the body of the response to it. Finally, I printed a message indicating that the file has been downloaded and saved. Also, at the end, I closed the socket I binded to port 80.