```
Starting at 9:05 pm
```

Find the Line with the Most Digits

You are given the path to a text file. Each line in the file may contain letters, symbols, whitespace, and digits (0-9).

Your task is to find and return the line that contains the maximum number of digit characters.

A digit is any character from '0' to '9'. If multiple lines contain the same maximum number of digits, return the first such line. Ignore all non-digit characters while counting digits.

```
This is line 1. -> 1
Room 202 has 4 ACs. -> 4
There are 3 dogs and 4 cats. -> 2
ID: 007, Code: 123456 -> 9

Output: ID: 007, Code: 123456
```

```
Approach:
max_digits=-1 -> keeps track of max no of digits found so far
result_line="" -> stores the line with max no of digits
open a file using a given path in read mode
for each line
- Count the no of digits
- compare if this count exceeds the max_digits
After reading all lines, return the result_line


Example Walkthrough:
max_digits=-1 -> 2 -> 4 -> 5
result_line="Digits: 45678 are more."

Line one has 12 apples. -> digits ->1,2 -> 2
This line has 3 digits: 123 -> digits -> 3,1,2,3 -> 4
No digits here! -> digits -> 0
Digits: 45678 are more.  -> digits -> 4,5,6,7,8 -> 5

return Digits: 45678 are more.
```

```
line="hello World\n"
print(repr(line))
print(repr(line.rstrip('\n')))
```

```
'hello World\n'
'hello World'
```

```
max_digits=-1
result_line=""
```

```
    with open(file_path,'r') as file:
      for line in file:
        digit_count=0
        for char in line:
          if char.isdigit():
            digit_count+=1

        if digit_count>max_digits:
          max_digits=digit_count
          result_line=line.rstrip('\n')

    return result_line
```

Most requested URL from Log

You are given a web server access log. Each line contains an HTTP request. Your task is to parse the log and return the URL (request target) that was requested the most times.

Ignore HTTP method, protocol version, status code, and other fields — only the request target matters.

If multiple URLs have the same highest request count, return the one which attains the highest count first while scanning the file from top to bottom.

```
    192.168.1.11 - - [11/Aug/2025:10:00:02 +0530] "GET /search?q=logs HTTP/1.
    192.168.1.10 - - [11/Aug/2025:10:00:01 +0530] "GET /home HTTP/1.1" 200 10
    192.168.1.13 - - [11/Aug/2025:10:00:04 +0530] "GET /products?id=7 HTTP/1.
    192.168.1.12 - - [11/Aug/2025:10:00:03 +0530] "GET /home HTTP/1.1" 304 0
    192.168.1.14 - - [11/Aug/2025:10:00:05 +0530] "GET /search?q=logs HTTP/1.

    search - 2
    home - 2 -> reaches highest count first
    products - 1

    Output: /home
```

```
    METHOD <URL> HTTP/<version>
```

```
    Approach:
    use regex to capture the http request

    captures the HTTP METHOD(GET, POST, etc) -> [A-Z]+
    \s - space
    capture the url - \S+ (anythign apart from space, tabs, whitespcaes)
    HTTP - protocol part

    initialize count
    url - freq -> dict
```

```
import re
"GET /search?q=logs HTTP/1.1"
#METHOD <URL> HTTP/x.y
request_re=re.compile(r'"([A-Z]+)\s+(\S+)\s+HTTP/\d+(?:\.\d+)?"')

def most_requested_url(file_path):
  counts={}
  best_url=""
  best_count=0
  with open(file_path,"r",encoding="utf-8") as f:
    for line in f:
      m=request_re.search(line)
      if not m:
        continue
      url=m.group(2)

      new_count=counts.get(url,0)+1
      counts[url]=new_count

      #tie-breaker
      if new_count>best_count:
        best_count=new_count
        best_url=url

return best_url
```

BREAK: 10:04 - 10:15 pm

Extract Non-200 HTTP Status Code Logs

You are given the path to a log file. Each line in the file may contain an HTTP status code such as 200, 404, 500, etc.

Your task is to find and return all log lines that contain a non-200 HTTP status code.

Only the first 3-digit number in each line is considered the status code. Ignore lines that do not contain any 3-digit number. HTTP status codes are always 3-digit integers.

```
[INFO] 2025-08-08 12:01:03 GET /index.html 200 OK
[WARNING] 2025-08-08 12:01:10 GET /not-found 404 Not Found
[ERROR] 2025-08-08 12:01:15 GET /server-error 500 Internal Server Error
[INFO] 2025-08-08 12:01:20 GET /health 200 OK
[DEBUG] Something unrelated to HTTP

Line 1 - Status code -> 200 ignore
Line 2 - Status code -> 404 included
Line 3 - Status code -> 500 included
Line 4 - Status code -> 200 ignore
Line 5 - Status code -> no code ignore

Output:
[WARNING] 2025-08-08 12:01:10 GET /not-found 404 Not Found
```

```
[WARNING] 2025-08-08 12:01:10 GET /not-found 404 Not Found
[ERROR] 2025-08-08 12:01:15 GET /server-error 500 Internal Server Error
```

```
Approach:
Read the file line by line
extract the first 3 digit number from the line
use regex  \b\d{3}\b
if it equals 200 -> skip
otherwise -> collect the line
if not found -> skip the line
output-> collected lines
```

```
import re

def non_200_status_lines(file_path):
  result_line=[]
  pattern=re.compile(r"\b(\d{3})\b")

  with open(file_path,"r",encoding="utf-8") as file:
    for line in file:
      match=pattern.search(line)
      if match:
        status_code=match.group(1)
        if status_code!="200":
          result_line.append(line.rstrip('\n'))

return result_line
```

Distributing Work Among Computers

You are given:

- requests → total number of tasks (like customer calls)
- computers → number of computers available

Distribute the tasks as evenly as possible. If it doesn't divide perfectly, give the extra tasks to the first few computers.

Return a list where each element represents the number of requests assigned to a corresponding computer.

```
from ast import mod
requests=14
computers=5
Distribution=[1,1,1,1,1]
remaining= 14-5=9
Distribution=[2,2,2,2,2]
remaining= 9-5=4
Distribution=[3,3,3,3,2] #output
remaining=0

Approach:
Calculate laod which can be distributed amongst all
```

```
14//5-> 2
Handle the remaining requests
use mod


requests=10
computers =3
- 10//3 -> 3 min load
- 10%3 -> 1 extra request
assign this one extra to the first computer
```

```
def distribute_load(requests,computers):
  base_load=request//computers #14//5=2
  remaining_load=request%computers #14%5=4
  distribution=[base_load]*computers #[2]*5->[2,2,2,2,2]

  #extra load
  for i in range(remaining_load): #4
    distribution[i]+=1  #[3,3,3,3,2]

  return distribution

#requests=14
#computers=5
```

```
DOUBTS:
```

```
sample_log=["[INFO] 2025-08-08 12:01:03 GET /index.html 200 OK",
"[WARNING] 2025-08-08 12:01:10 GET /not-found 404 Not Found",
"[ERROR] 2025-08-08 12:01:15 GET /server-error 500 Internal Server Error",
"[INFO] 2025-08-08 12:01:20 GET /health 200 OK",
"[DEBUG] Something unrelated to HTTP"]

with open("samplefile.log","w",encoding="utf-8") as f:
  for log in sample_log:
    f.write(log+"\n")

print("samplefile.log file created")
```

```
samplefile.log file created
```

```
import re
from typing import List

def non_200_status_lines(file_path):
  result_line=[]
  pattern=re.compile(r"\b(\d{3})\b")

  with open(file_path,"r",encoding="utf-8") as file:
    for line in file:
      match=pattern.search(line)
      if match:
        status_code=match.group(1)
```

```
            if status_code!="200":
              result_line.append(line.rstrip('\n'))
      return result_line

    file_path="samplefile.log"
    non_200_lines=non_200_status_lines(file_path)
    print(non_200_lines)
```

```
['[ERROR] 2025-08-08 12:01:15 GET /server-error 500 Internal Server Error']
```

```
with open("samplefile.log","r",encoding="utf-8") as f:
  for line in f:
    print(line.strip())
```

```
[INFO] 2025-08-08 12:01:03 GET /index.html 200 OK
[WARNING] 2025-08-08 12:01:10 GET /not-found 404 Not Found
[ERROR] 2025-08-08 12:01:15 GET /server-error 500 Internal Server Error
[INFO] 2025-08-08 12:01:20 GET /health 200 OK
[DEBUG] Something unrelated to HTTP
```