

SHOBHIT RAM CHAURASIYA

Design a logger system that receives a stream of messages along with their timestamps. Each unique message should only be printed at most every 10 seconds (i.e. a message printed at timestamp t will prevent other identical messages from being printed until timestamp $t + 10$).

All messages will come in chronological order. Several messages may arrive at the same timestamp.

Implement the `Logger` class:

- `Logger()` Initializes the `logger` object.
- `bool shouldPrintMessage(int timestamp, string message)` Returns `true` if the message should be printed in the given timestamp, otherwise returns `false`

Example 1:

Input

```
["Logger", "shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage",  
"shouldPrintMessage", "shouldPrintMessage", "shouldPrintMessage"]  
[[], [1, "foo"], [2, "bar"], [3, "foo"], [8, "bar"], [10, "foo"], [11, "foo"]]
```

Output

```
[null, true, true, false, false, false, true]
```

Explanation

```
Logger logger = new Logger();  
  
logger.shouldPrintMessage(1, "foo"); // return true, next allowed timestamp  
for "foo" is 1 + 10 = 11  
  
logger.shouldPrintMessage(2, "bar"); // return true, next allowed timestamp  
for "bar" is 2 + 10 = 12  
  
logger.shouldPrintMessage(3, "foo"); // 3 < 11, return false
```

```

logger.shouldPrintMessage(8, "bar"); // 8 < 12, return false

logger.shouldPrintMessage(10, "foo"); // 10 < 11, return false

logger.shouldPrintMessage(11, "foo"); // 11 >= 11, return true, next allowed
timestamp for "foo" is

// 11 + 10 = 21

```

Constraints:

- $0 \leq \text{timestamp} \leq 10^9$
- Every timestamp will be passed in non-decreasing order (chronological order).
- $1 \leq \text{message.length} \leq 30$
- At most 10^4 calls will be made to `shouldPrintMessage`.

SOLUTION -

The screenshot shows a Python IDE with a file named `mid_assessment.py`. The code defines a `Logger` class with a `__init__` method that initializes a `timing` dictionary. The `shouldPrintMessage` method checks if a message has been printed within the last 10 units of time. If it has, it returns `False`; otherwise, it prints `True` and updates the `timing` dictionary with the current timestamp.

```

class Logger:
    def __init__(self):
        self.timing = {}

    def shouldPrintMessage(self, timestamp, message):
        if message in self.timing:
            previous_occurrence = self.timing[message]
            if timestamp >= previous_occurrence + 10:
                self.timing[message] = timestamp
                print("True")
            else:
                print("False")
        else:
            self.timing[message] = timestamp
            print("True")

```

The code also creates a `Logger` instance and calls `shouldPrintMessage` with the following inputs:

```

logger = Logger()
logger.shouldPrintMessage(1, "foo")
logger.shouldPrintMessage(2, "bar")
logger.shouldPrintMessage(3, "foo")
logger.shouldPrintMessage(8, "bar")
logger.shouldPrintMessage(10, "foo")
logger.shouldPrintMessage(11, "foo")

```

The output of the program is shown in the Run console:

```

True
True
False
False
False
True
True

```