1h 23m left

ALL

1

2

2. JavaScript: Notes Store

In this challenge, the task is to create a class NotesStore. The class will manage a collection of notes, with each note having a state and a name. Valid states for notes are: 'completed', 'active', and 'others'. All other states are invalid.

The class must have following methods:

1. addNote(state, name): adds a note with the given name and state to the collection. In addition to that:
   - If the passed name is empty, then it throws an Error with the message 'Name cannot be empty'.
   - If the passed name is non-empty but the given state is not a valid state for a note, then it throws an Error with the message 'Invalid state {state}'.
2. getNotes(state): returns an array of names of notes with the given state added so far. The names are returned in the order the corresponding notes were added. In addition to that:
   - If the given state is not a valid note state, then it throws an Error with the message 'Invalid state {state}'.
   - If no note is found in this state, it returns an empty array.

Note: The state names are case-sensitive.

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the functions call. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the provided code. The stubbed code joins the strings returned by the *getNotes* function with a comma and prints to the standard output. If *getNotes* returns an empty array, the stubbed code prints 'No Notes'. The stubbed code also prints messages of all the thrown errors.

▶ **Input Format For Custom Testing**

▼ **Sample Case 0**

Sample Input For Custom Testing

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Language: JavaScript (Node.js) Autocomplete Ready

```
1  'use strict';
23
24 class NotesStore {
25     //add your code here
26 }
27
28 function main() {
```

Line: 23 Col: 1

Test Results **Custom Input** Run Code Run Tests Submit

- If the passed name is empty, then it throws an Error with the message 'Name cannot be empty'.
- If the passed name is non-empty but the given state is not a valid state for a note, then it throws an Error with the message 'Invalid state {state}'.

2. getNotes(state): returns an array of names of notes with the given state added so far. The names are returned in the order the corresponding notes were added. In addition to that:
   - If the given state is not a valid note state, then it throws an Error with the message 'Invalid state {state}'.
   - If no note is found in this state, it returns an empty array.

Note: The state names are case-sensitive.

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the functions call. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the provided code. The stubbed code joins the strings returned by the *getNotes* function with a comma and prints to the standard output. If *getNotes* returns an empty array, the stubbed code prints 'No Notes'. The stubbed code also prints messages of all the thrown errors.

▸ **Input Format For Custom Testing**

▾ **Sample Case 0**

Sample Input For Custom Testing

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Sample Output

```
DrinkTea,DrinkCoffee
Study
Error: Invalid state foo
```

Explanation

For all 3 *addNote* operations, the *addNote* function is called with a state and a name. Then, the *getNotes* function is called for 'active' state and 'completed' state respectively, and the result is printed. Then, the *getNotes* function is called for 'foo' state, which throws an error since this state is invalid, and the error is printed.

▸ **Sample Case 1**

Language: JavaScript (Node.js) Autocomplete Ready

```
29              throw new Error('Name cannot be empty');
30          }
31          if(!this.states.includes(state)){
32              throw new Error('Invalid state ${state}');
33          }
34          this.collection.push({
35              state, name
36          })
37      }
38      getNotes(state){
39          if(!this.states.includes(state)){
40              throw new Error('Invalid state ${state}');
41          }
42          return this.collection.filter(i => i.state === state);
43      }
44  }
45
46 > function main() {···
```

Line: 42 Col: 63

Test Results | **Custom Input** | Run Code | Run Tests | Submit

Compiled successfully. Run all test cases

Input (stdin)

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Your Output (stdout)

```
[object Object],[object Object]
[object Object]
Error: Invalid state foo
```

1h 16m left

ALL

1

2

In this challenge, the task is to create a class NotesStore. The class will manage a collection of notes, with each note having a state and a name. Valid states for notes are: 'completed', 'active', and 'others'. All other states are invalid.

The class must have following methods:

1. addNote(state, name): adds a note with the given name and state to the collection. In addition to that:
   - If the passed name is empty, then it throws an Error with the message 'Name cannot be empty'.
   - If the passed name is non-empty but the given state is not a valid state for a note, then it throws an Error with the message 'Invalid state {state}'.
2. getNotes(state): returns an array of names of notes with the given state added so far. The names are returned in the order the corresponding notes were added. In addition to that:
   - If the given state is not a valid note state, then it throws an Error with the message 'Invalid state {state}'.
   - If no note is found in this state, it returns an empty array.

Note: The state names are case-sensitive.

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the functions call. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the provided code. The stubbed code joins the strings returned by the *getNotes* function with a comma and prints to the standard output. If *getNotes* returns an empty array, the stubbed code prints 'No Notes'. The stubbed code also prints messages of all the thrown errors.

▸ **Input Format For Custom Testing**

▾ **Sample Case 0**

Sample Input For Custom Testing

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Sample Output

```
DrinkTea,DrinkCoffee
Study
```

Language: JavaScript (Node.js) Autocomplete Ready

```
31          if(!this.states.includes(state)){
32              throw new Error(`Invalid state ${state}`);
33          }
34          this.collection.push({
35              state, name
36          })
37      }
38      getNotes(state){
39          if(!this.states.includes(state)){
40              throw new Error(`Invalid state ${state}`);
41          }
42          return this.collection.filter(i => i.state === state).map(i => i.name);
43      }
44  }
45
46  function main() {
47      const ws = fs.createWriteStream(process.env.OUTPUT_PATH);
48
```

Line: 42 Col: 15

Test Results | **Custom Input** | Run Code | Run Tests | Submit

Compiled successfully. All available test case 0 passed

Test case 0

Test case 1

Test case 2

Test case 3

Test case 4

Test case 5

Test case 6

Test case 7

Input (stdin) — Run as Custom Input | Download

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Your Output (stdout)

```
DrinkTea,DrinkCoffee
Study
Error: Invalid state foo
```

1h 16m left

ALL

1

2

In this challenge, the task is to create a class NotesStore. The class will manage a collection of notes, with each note having a state and a name. Valid states for notes are: 'completed', 'active', and 'others'. All other states are invalid.

The class must have following methods:

1. addNote(state, name): adds a note with the given name and state to the collection. In addition to that:
   - If the passed name is empty, then it throws an Error with the message 'Name cannot be empty'.
   - If the passed name is non-empty but the given state is not a valid state for a note, then it throws an Error with the message 'Invalid state {state}'.
2. getNotes(state): returns an array of names of notes with the given state added so far. The names are returned in the order the corresponding notes were added. In addition to that:
   - If the given state is not a valid note state, then it throws an Error with the message 'Invalid state {state}'.
   - If no note is found in this state, it returns an empty array.

Note: The state names are case-sensitive.

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the functions call. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the provided code. The stubbed code joins the strings returned by the *getNotes* function with a comma and prints to the standard output. If *getNotes* returns an empty array, the stubbed code prints 'No Notes'. The stubbed code also prints messages of all the thrown errors.

▸ **Input Format For Custom Testing**

▾ **Sample Case 0**

Sample Input For Custom Testing

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Sample Output

```
DrinkTea,DrinkCoffee
Study
```

Language: JavaScript (Node.js) Autocomplete Ready

```
'use strict';
class NotesStore {
    collection = [];
    states = ['active', 'completed','others']
    addNote(state, name){
        if(!name){
            throw new Error('Name cannot be empty');
        }
        if(!this.states.includes(state)){
            throw new Error('Invalid state ${state}');
        }
        this.collection.push({
            state, name
        })
    }
    getNotes(state){
        if(!this.states.includes(state)){
            throw new Error('Invalid state ${state}');
        }
        return this.collection.filter(i => i.state === state).map(i => i.name);
    }
}

function main() {
    const ws = fs.createWriteStream(process.env.OUTPUT_PATH);
```

Line: 42 Col: 15

Test Results | **Custom Input** | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0

Test case 1

Test case 2

Test case 3

Test case 4

Input (stdin) — Run as Custom Input | Download

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

1h 16m left

ALL

1

2

In this challenge, the task is to create a class NotesStore. The class will manage a collection of notes, with each note having a state and a name. Valid states for notes are: 'completed', 'active', and 'others'. All other states are invalid.

The class must have following methods:

1. addNote(state, name): adds a note with the given name and state to the collection. In addition to that:
   - If the passed name is empty, then it throws an Error with the message 'Name cannot be empty'.
   - If the passed name is non-empty but the given state is not a valid state for a note, then it throws an Error with the message 'Invalid state {state}'.
2. getNotes(state): returns an array of names of notes with the given state added so far. The names are returned in the order the corresponding notes were added. In addition to that:
   - If the given state is not a valid note state, then it throws an Error with the message 'Invalid state {state}'.
   - If no note is found in this state, it returns an empty array.

Note: The state names are case-sensitive.

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the functions call. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the provided code. The stubbed code joins the strings returned by the *getNotes* function with a comma and prints to the standard output. If *getNotes* returns an empty array, the stubbed code prints 'No Notes'. The stubbed code also prints messages of all the thrown errors.

▸ **Input Format For Custom Testing**

▾ **Sample Case 0**

Sample Input For Custom Testing

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Sample Output

```
DrinkTea,DrinkCoffee
Study
```

Language: JavaScript (Node.js) Autocomplete Ready

```
'use strict';

class NotesStore {
    collection = [];
    states = ['active', 'completed','others']
    addNote(state, name){
        if(!name){
            throw new Error('Name cannot be empty');
        }
        if(!this.states.includes(state)){
            throw new Error('Invalid state ${state}');
        }
        this.collection.push({
            state, name
        })
    }
    getNotes(state){
        if(!this.states.includes(state)){
            throw new Error('Invalid state ${state}');
        }
        return this.collection.filter(i => i.state === state).map(i => i.name);
    }
}

function main() {
    const ws = fs.createWriteStream(process.env.OUTPUT_PATH);
```

Line: 42 Col: 80

Test Results | **Custom Input** | Run Code | Run Tests | Submit

Compiled successfully. All available test cases passed

Test case 0

Test case 1

Test case 2

Test case 3

Test case 4

Input (stdin) — Run as Custom Input | Download

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Activities Google Chrome Feb 14 23:36
Channel content - YouTub | Channel content - YouTub | HackerRank | Snappa
hackerrank.com/test/dff5pp65l92/questions/6rq6idpckeq
Apps | Dev19 | Gmail | YouTube | Snappa | Hackerrank | WhatsApp | QuestionBank | Firebase | Cisco Webex... | cPanel - Main | Hostinger | Nodejs | urbanfitwelln... | GoDaddy

1h 11m left

ALL

1

2

functions call. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the provided code. The stubbed code joins the strings returned by the *getNotes* function with a comma and prints to the standard output. If *getNotes* returns an empty array, the stubbed code prints 'No Notes'. The stubbed code also prints messages of all the thrown errors.

▼ Input Format For Custom Testing

In the first line, there is an integer, *n*, denoting the number of operations to be performed. Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains space-separated strings such that the first of them is a function name, and the remaining ones, if any, are parameters for that function.

▼ Sample Case 0

Sample Input For Custom Testing

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Sample Output

```
DrinkTea,DrinkCoffee
Study
Error: Invalid state foo
```

Language: JavaScript (Node.js) Autocomplete Disabled

```
'use strict';

class NotesStore {
    //add your code here
}

function main() {
    const ws = fs.createWriteStream(process.env.OUTPUT_PATH);

    const obj = new NotesStore();
    const operationCount = parseInt(readLine().trim());

    for(let i = 1; i <= operationCount; i++) {
        const operationInfo = readLine().trim().split(' ');
        try {
            if (operationInfo[0] === 'addNote') {
                obj.addNote(operationInfo[1], operationInfo[2] || '');
            } else if (operationInfo[0] === 'getNotes') {
                const res = obj.getNotes(operationInfo[1]);
                if (res.length === 0) {
                    ws.write('No Notes\n');
                } else {
                    ws.write(`${res.join(',')}\n`);
                }
            }
        } catch (e) {
            ws.write(`${e}\n`);
        }
    }
    ws.end();
}
```

code 2

Line: 27 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

Activities Google Chrome Feb 14 23:36
Channel content - YouTub | Channel content - YouTub | HackerRank | Snappa
hackerrank.com/test/dff5pp65l92/questions/6rq6idpckeq
Apps | Dev19 | Gmail | YouTube | Snappa | Hackerrank | WhatsApp | QuestionBank | Firebase | Cisco Webex... | cPanel - Main | Hostinger | Nodejs | urbanfitwelln... | GoDaddy

1h 10m left

ALL

1

2

Input Format For Custom Testing

In the first line, there is an integer, *n*, denoting the number of operations to be performed. Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains space-separated strings such that the first of them is a function name, and the remaining ones, if any, are parameters for that function.

Sample Case 0

Sample Input For Custom Testing

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Sample Output

```
DrinkTea,DrinkCoffee
Study
Error: Invalid state foo
```

Explanation

For all 3 *addNote* operations, the *addNote* function is called with a state and a name. Then, the *getNotes* function is called for 'active' state and 'completed' state respectively, and the result is printed. Then, the *getNotes* function is called for 'foo' state, which throws an error since this state is invalid, and the error is printed.

Sample Case 1

Language: JavaScript (Node.js) Autocomplete Disabled

```
'use strict';

class NotesStore {
    //add your code here
}

function main() {
    const ws = fs.createWriteStream(process.env.OUTPUT_PATH);

    const obj = new NotesStore();
    const operationCount = parseInt(readLine().trim());

    for(let i = 1; i <= operationCount; i++) {
        const operationInfo = readLine().trim().split(' ');
        try {
            if (operationInfo[0] === 'addNote') {
                obj.addNote(operationInfo[1], operationInfo[2] || '');
            } else if (operationInfo[0] === 'getNotes') {
                const res = obj.getNotes(operationInfo[1]);
                if (res.length === 0) {
                    ws.write('No Notes\n');
                } else {
                    ws.write(`${res.join(',')}\n`);
                }
            }
        } catch (e) {
            ws.write(`${e}\n`);
        }
    }
    ws.end();
}
```

Line: 38 Col: 28

Test Results | Custom Input | Run Code | Run Tests | Submit

Activities Google Chrome Feb 14 23:59
Channel content - YouTub Channel content - YouTub HackerRank Snappa
hackerrank.com/test/dff5pp65l92/questions/6rq6idpckeq
Apps i Dev19 Gmail YouTube Snappa Hackerrank WhatsApp QuestionBank Firebase Cisco Webex... cPanel - Main Hostinger Nodejs urbanfitwelln... GoDaddy

48m left

ALL

1

2

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the functions call. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the provided code. The stubbed code joins the strings returned by the *getNotes* function with a comma and prints to the standard output. If *getNotes* returns an empty array, the stubbed code prints 'No Notes'. The stubbed code also prints messages of all the thrown errors.

▸ **Input Format For Custom Testing**

▾ **Sample Case 0**

Sample Input For Custom Testing

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Sample Output

```
DrinkTea,DrinkCoffee
Study
Error: Invalid state foo
```

Explanation

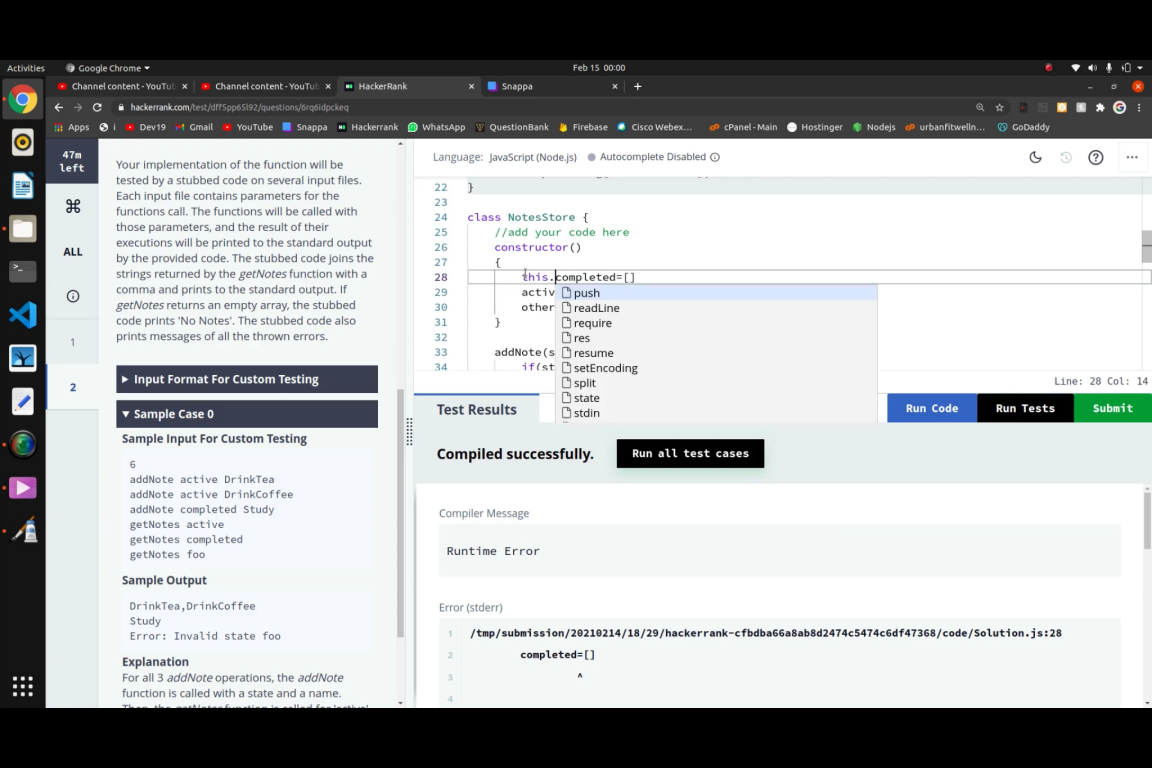
For all 3 *addNote* operations, the *addNote* function is called with a state and a name.

Language: JavaScript (Node.js) Autocomplete Disabled

```
26      constructor()
27      {
28          completed=[]
29          active=[]
30          others=[]
31      }
32
33      addNote(state,name){
34          if(state=="active")
35              this.active.push(name)
36          else if (state=="completed")
37              this.completed.push(name)
38          else if (state=="others")
39              this.others.push(name)
40      }
41
42      getNotes(state){
43          if(state=="active")
44              return this.active
45          else if (state=="completed")
46              this.completed.push(hame)
47          else if (state=="others")
48              this.others.push(name)
49      }
50  }
51
52  function main() {
53      const ws = fs.createWriteStream(process.env.OUTPUT_PATH);
54
55      const obj = new NotesStore();
56      const operationCount = parseInt(readLine().trim());
57
```

Line: 46 Col: 37

Test Results Custom Input Run Code Run Tests Submit


Activities
Google Chrome
Feb 15 00:00
Channel content - YouTub
Channel content - YouTub
HackerRank
Snappa
hackerrank.com/test/dff5pp65l92/questions/6rq6idpckeq
Apps
Dev19
Gmail
YouTube
Snappa
Hackerrank
WhatsApp
QuestionBank
Firebase
Cisco Webex...
cPanel - Main
Hostinger
Nodejs
urbanfitwelln...
GoDaddy
47m left
ALL
1
2
Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the functions call. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the provided code. The stubbed code joins the strings returned by the getNotes function with a comma and prints to the standard output. If getNotes returns an empty array, the stubbed code prints 'No Notes'. The stubbed code also prints messages of all the thrown errors.
Input Format For Custom Testing
Sample Case 0
Sample Input For Custom Testing
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
Sample Output
DrinkTea,DrinkCoffee
Study
Error: Invalid state foo
Explanation
For all 3 addNote operations, the addNote function is called with a state and a name.
Language: JavaScript (Node.js)
Autocomplete Disabled
22 }
23
24 class NotesStore {
25 //add your code here
26 constructor()
27 {
28 this.completed=[]
29 activ
30 other
31 }
32
33 addNote(s
34 if(st
push
readLine
require
res
resume
setEncoding
split
state
stdin
Line: 28 Col: 14
Test Results
Run Code
Run Tests
Submit
Compiled successfully.
Run all test cases
Compiler Message
Runtime Error
Error (stderr)
/tmp/submission/20210214/18/29/hackerrank-cfbdba66a8ab8d2474c5474c6df47368/code/Solution.js:28
completed=[]
^

Activities Google Chrome Feb 15 00:03
Channel content - YouTub | Channel content - YouTub | HackerRank | Snappa
hackerrank.com/test/dff5pp65l92/questions/6rq6idpckeq
Apps | Dev19 | Gmail | YouTube | Snappa | Hackerrank | WhatsApp | QuestionBank | Firebase | Cisco Webex... | cPanel - Main | Hostinger | Nodejs | urbanfitwelln... | GoDaddy

44m left

ALL

1

2

Your implementation of the function will be tested by a stubbed code on several input files. Each input file contains parameters for the functions call. The functions will be called with those parameters, and the result of their executions will be printed to the standard output by the provided code. The stubbed code joins the strings returned by the *getNotes* function with a comma and prints to the standard output. If *getNotes* returns an empty array, the stubbed code prints 'No Notes'. The stubbed code also prints messages of all the thrown errors.

▸ **Input Format For Custom Testing**

▾ **Sample Case 0**

Sample Input For Custom Testing

```
6
addNote active DrinkTea
addNote active DrinkCoffee
addNote completed Study
getNotes active
getNotes completed
getNotes foo
```

Sample Output

```
DrinkTea,DrinkCoffee
Study
Error: Invalid state foo
```

Explanation

For all 3 *addNote* operations, the *addNote* function is called with a state and a name.

Language: JavaScript (Node.js) Autocomplete Disabled

```
32
33      addNote(state,name){
34          if(state=="active")
35              this.active.push(name)
36          else if (state=="completed")
37              this.completed.push(name)
38          else if (state=="others")
39              this.others.push(name)
40              else
41              throw new Error(`Invalid state ${state}`)
42      }
43
44      getNotes(state){
45          if(state=="active")
46              return this.active
47          else if (state=="completed")
48              return this.completed
49          else if (state=="others")
50              return this.others
51          else
52              throw new Error(`Invalid state ${state}`)
53      }
54  }
55
56  function main() {
57      const ws = fs.createWriteStream(process.env.OUTPUT_PATH);
58
59      const obj = new NotesStore();
60      const operationCount = parseInt(readLine().trim());
61
62      for(let i = 1; i <= operationCount; i++) {
63          const operationInfo = readLine().trim().split(' ');
```

Line: 41 Col: 54

Test Results | Custom Input | Run Code | Run Tests | Submit

Activities Google Chrome Feb 15 00:06
Channel content - YouTub | Channel content - YouTub | HackerRank | Snappa
hackerrank.com/test/dff5pp65l92/questions/6rq6idpckeq
Apps | Dev19 | Gmail | YouTube | Snappa | Hackerrank | WhatsApp | QuestionBank | Firebase | Cisco Webex... | cPanel - Main | Hostinger | Nodejs | urbanfitwelln... | GoDaddy

41m left

ALL

1

2

```
Study
Error: Invalid state foo
```

Explanation

For all 3 *addNote* operations, the *addNote* function is called with a state and a name. Then, the *getNotes* function is called for 'active' state and 'completed' state respectively, and the result is printed. Then, the *getNotes* function is called for 'foo' state, which throws an error since this state is invalid, and the error is printed.

▼ Sample Case 1

Sample Input For Custom Testing

```
3
addNote active
addNote foo Study
getNotes completed
```

Sample Output

```
Error: Name cannot be empty
Error: Invalid state foo
No Notes
```

Explanation

The *addNote* function is called with 'active' state and an empty name, which throws an error since the name is empty. Then, *addNote* is called with 'foo' state, which throws an error since the state is invalid and an error is thrown. Finally, the *getNotes* function is called for 'completed' state, an empty array is returned since no note exists in this state, and 'No Notes' is printed by the stubbed code.

Language: JavaScript (Node.js) Autocomplete Disabled

```
32
33  addNote(state,name){
34
35      if(name=='')
36          throw new Error(`Name cannot be empty`)
37
38      if(state=="active")
39          this.active.push(name)
40      else if (state=="completed")
41          this.completed.push(name)
42      else if (state=="others")
43          this.others.push(name)
44      else
```

Line: 36 Col: 30

Test Results | Custom Input | Run Code | Run Tests | Submit

Uploading