1. JavaScript: User Transactions

1. JavaScript: User Transactions

Implement a function *getNumTransactions()*. It takes a *username* and it returns either the number of transactions associated with *username*, or a string "Username Not Found" if no such user exists. Make a GET request to the given REST APIs that contain information about a user.

Given a user name, the user details should be fetched by making a GET call to the API https://jsonmock.hackerrank.com/api/article_users?username=<username> where <username> is the parameter passed to the *getNumTransactions* function.

The response will be a JSON object with the following 5 fields:

- *page*: The current page of the results. (Number)
- *per_page*: The maximum number of results returned per page. (Number)
- *total*: The total number of results. (Number)
- *total_pages*: The total number of pages with results. (Number)
- *data*: Either an empty array or an array with a single object containing the user details record with the following schema:
  - id - user id (Number)
  - username - user name as passed above (String)
  - This object has other fields as well but they are not needed for this question.

If the username passed to the request does not exist in the system, the data array will be empty. In that case, the function should return "Username Not Found".

An example of a user record is as follows:

```
{
   "id": 1,
   "username": "epaga"
}
```

Language: JavaScript (Node.js) Environment — Autocomplete Ready

```
'use strict';

const axios = require('axios');
async function getNumTransactions(username) {
    // API endpoint: https://jsonmock.hackerrank.com/api/article_users?username=<username>
    // API endpoint: https://jsonmock.hackerrank.com/api/transactions?&userId=<userId>
    try{
        const{ data } = await axios.get(`https://jsonmock.hackerrank.com/api/article_users?username=${username}`);

    if(data.data && data.data.length !== 0){
        const userId = data.data[0].id

        const response = await axios.get(`https://jsonmock.hackerrank.com/api/transactions?&userId=${userId}`);
        return response.data.total;
    }else {
        return "Username Not Found"
    }
    } catch(err){
        console.log(err);
    }

}

async function main() {
```

Line: 42 Col: 6

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

An example of a user record is as follows:

```
{
   "id": 1,
   "username": "epaga"
}
```

If the user details is successfully fetched from the first API, use the ***id*** property of the details object to fetch the transactions information for the user. The API to fetch this is https://jsonmock.hackerrank.com/api/transactions?&userId=<userId> where <userId> is the ***id*** property fetched earlier.

The response will be a JSON object with the following 5 fields:

- *page*: The current page of the results. (Number)
- *per_page*: The maximum number of results returned per page. (Number)
- *total*: The total number of transactions for the given user id. (Number)
- *total_pages*: The total number of pages with results. (Number)
- *data*: Either an empty array or an array with transaction records of the user.

Here *total* is the total number of transactions of the user and is the final value to be returned by the function.

Here is an example of a response:

```
{
   page: 1,
   per_page: 10,
   total: 75,
   total_pages: 8,
   data: [
   // Data contains the list of user transactions matching the
user ID.
      {
         "id": 8,
         "userId": 1,
         "userName": "Francesco De Mello",
         "timestamp": 1548805761859,
         "txnType": "credit",
         "amount": "$1,214.44",
      ...
      }
   ]
```

Language: JavaScript (Node.js) Environment — Autocomplete Ready

```
'use strict';

const axios = require('axios');
async function getNumTransactions(username) {
    // API endpoint: https://jsonmock.hackerrank.com/api/article_users?username=<username>
    // API endpoint: https://jsonmock.hackerrank.com/api/transactions?&userId=<userId>
    try{
        const{ data } = await axios.get(`https://jsonmock.hackerrank.com/api/article_users?username=${username}`);

    if(data.data && data.data.length !== 0){
        const userId = data.data[0].id

        const response = await axios.get(`https://jsonmock.hackerrank.com/api/transactions?&userId=${userId}`);
        return response.data.total;
    }else {
        return "Username Not Found"
    }
    } catch(err){
        console.log(err);
    }

}

async function main() {
```

Line: 42 Col: 6

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

Here is an example of a response:

```
{
   page: 1,
   per_page: 10,
   total: 75,
   total_pages: 8,
   data: [
   // Data contains the list of user transactions matching the
user ID.
      {
         "id": 8,
         "userId": 1,
         "userName": "Francesco De Mello",
         "timestamp": 1548805761859,
         "txnType": "credit",
         "amount": "$1,214.44",
      ...
      }
   ]
}
```

As per this example final value to be returned is 75.

▼ Input Format For Custom Testing

In the first and only line, there is a user name.

▼ Sample Case 0

Sample Input For Custom Testing

```
epaga
```

Sample Output

```
79
```

Explanation

First a call is made to API https://jsonmock.hackerrank.com/api/article_users?username=epaga to fetch the user id which is 1 per the response. Then a second call is made to API https://jsonmock.hackerrank.com/api/transactions?userId=1 to fetch the total number of transactions which is 79 per the response.

▸ Sample Case 1

```
user ID.
      {
         "id": 8,
         "userId": 1,
         "userName": "Francesco De Mello",
         "timestamp": 1548805761859,
         "txnType": "credit",
         "amount": "$1,214.44",
      ...
      }
   ]
}
```

As per this example final value to be returned is 75.

▼ Input Format For Custom Testing

In the first and only line, there is a user name.

▼ Sample Case 0

Sample Input For Custom Testing

```
epaga
```

Sample Output

```
79
```

Explanation

First a call is made to API https://jsonmock.hackerrank.com/api/article_users?username=epaga to fetch the user id which is 1 per the response. Then a second call is made to API https://jsonmock.hackerrank.com/api/transactions?userId=1 to fetch the total number of transactions which is 79 per the response.

▼ Sample Case 1

Sample Input For Custom Testing

```
jay
```

Sample Output

```
Username Not Found
```

Explanation

An API call is made to https://jsonmock.hackerrank.com/api/article_users?username=jay to get the user id but the *data* field in the response is an empty array. Hence 'Username Not Found' is returned by the function.

Language: JavaScript (Node.js) ⓘ Environment — Autocomplete Ready

```
        const userId = data.data[0].id

        const response = await axios.get(`https://jsonmock.hackerrank.com/api/transactions?&userId=${userId}`);
        return response.data.total;
    }else {
        return "Username Not Found"
    }
    } catch(err){
        console.log(err);
    }

}

async function main() {⋯
```

Line: 42 Col: 6

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
- Test case 5
- Test case 6

Input (stdin) — Run as Custom Input | Download

```
epaga
```

Your Output (stdout)

```
79
```

Expected Output — Download

```
79
```

Language: JavaScript (Node.js) ⓘ Environment — Autocomplete Ready

```
    try{
        const{ data } = await axios.get(`https://jsonmock.hackerrank.com/api/article_users?username=${username}`);

    if(data.data && data.data.length !== 0){
        const userId = data.data[0].id

        const response = await axios.get(`https://jsonmock.hackerrank.com/api/transactions?&userId=${userId}`);
        return response.data.total;
    }else {
        return "Username Not Found"
    }
    } catch(err){
        console.log(err);
    }
```

Line: 42 Col: 6

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
- Test case 5
- Test case 6

Input (stdin) — Run as Custom Input | Download

```
jay
```

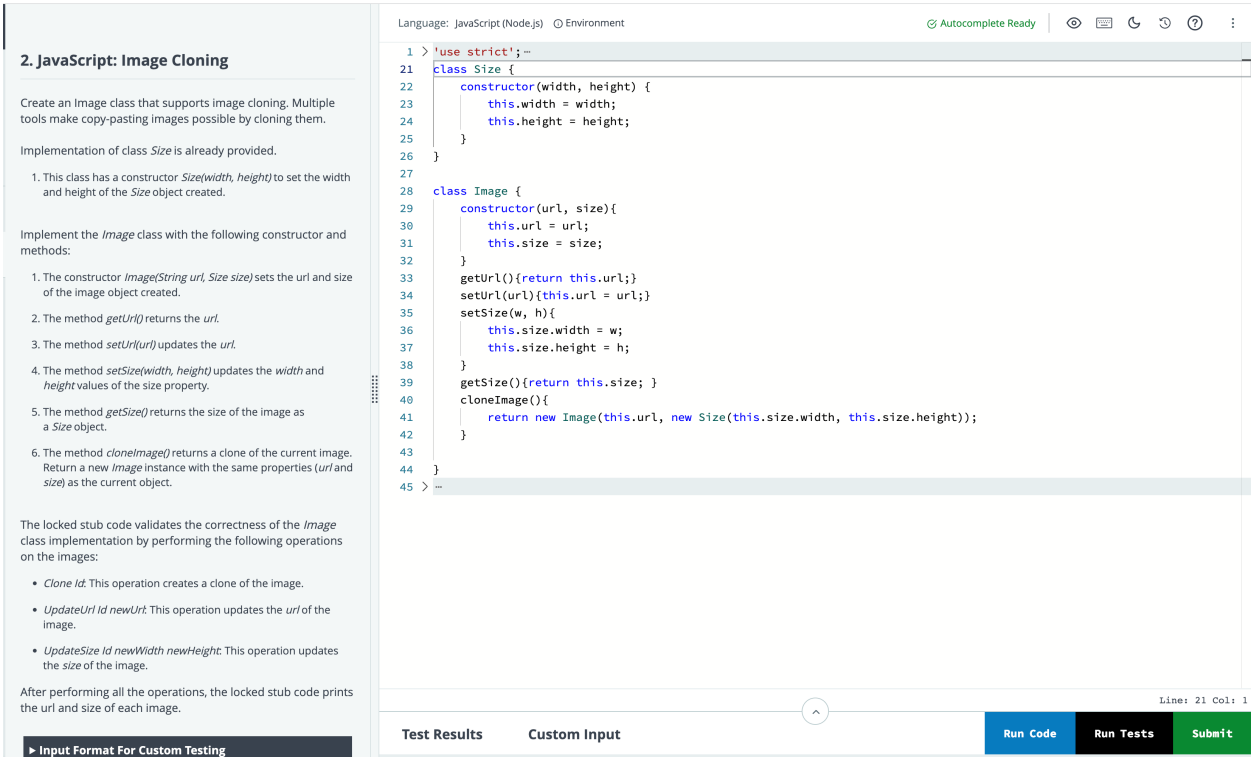
Your Output (stdout)

```
Username Not Found
```

Expected Output — Download

```
Username Not Found
```

2. JavaScript: Image Cloning



2. JavaScript: Image Cloning

Create an Image class that supports image cloning. Multiple tools make copy-pasting images possible by cloning them.

Implementation of class *Size* is already provided.

1. This class has a constructor *Size(width, height)* to set the width and height of the *Size* object created.

Implement the *Image* class with the following constructor and methods:

1. The constructor *Image(String url, Size size)* sets the url and size of the image object created.
2. The method *getUrl()* returns the *url*.
3. The method *setUrl(url)* updates the *url*.
4. The method *setSize(width, height)* updates the *width* and *height* values of the size property.
5. The method *getSize()* returns the size of the image as a *Size* object.
6. The method *cloneImage()* returns a clone of the current image. Return a new *Image* instance with the same properties (*url* and *size*) as the current object.

The locked stub code validates the correctness of the *Image* class implementation by performing the following operations on the images:

- *Clone Id*: This operation creates a clone of the image.
- *UpdateUrl Id newUrl*: This operation updates the *url* of the image.
- *UpdateSize Id newWidth newHeight*: This operation updates the *size* of the image.

After performing all the operations, the locked stub code prints the url and size of each image.

▸ Input Format For Custom Testing

Language: JavaScript (Node.js) Environment — Autocomplete Ready

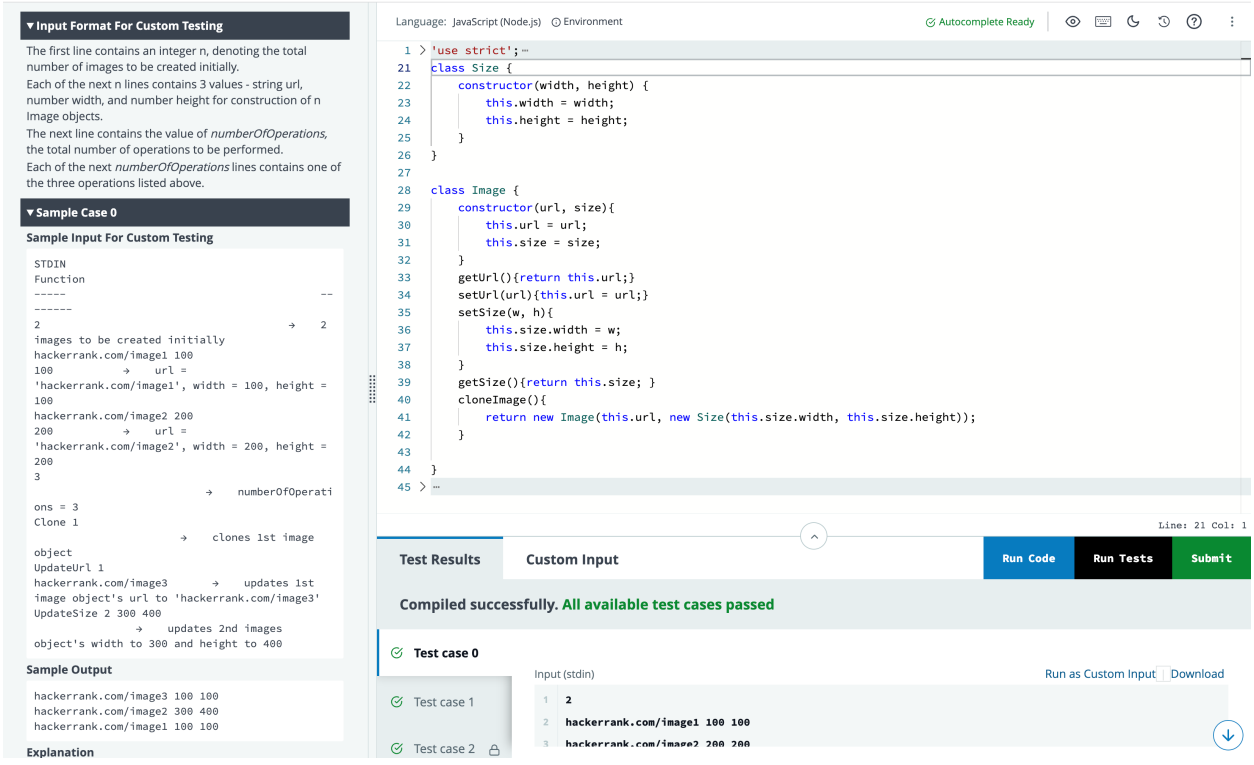
```
'use strict';
class Size {
    constructor(width, height) {
        this.width = width;
        this.height = height;
    }
}

class Image {
    constructor(url, size){
        this.url = url;
        this.size = size;
    }
    getUrl(){return this.url;}
    setUrl(url){this.url = url;}
    setSize(w, h){
        this.size.width = w;
        this.size.height = h;
    }
    getSize(){return this.size; }
    cloneImage(){
        return new Image(this.url, new Size(this.size.width, this.size.height));
    }

}
```

Line: 21 Col: 1

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



▾ Input Format For Custom Testing

The first line contains an integer n, denoting the total number of images to be created initially.
Each of the next n lines contains 3 values - string url, number width, and number height for construction of n Image objects.
The next line contains the value of *numberOfOperations*, the total number of operations to be performed.
Each of the next *numberOfOperations* lines contains one of the three operations listed above.

▾ Sample Case 0

Sample Input For Custom Testing

```
STDIN
Function
-----                                   --
------
2                                   →    2
images to be created initially
hackerrank.com/image1 100
100             →    url =
'hackerrank.com/image1', width = 100, height =
100
hackerrank.com/image2 200
200             →    url =
'hackerrank.com/image2', width = 200, height =
200
3
                            →    numberOfOperati
ons = 3
Clone 1
                        →    clones 1st image
object
UpdateUrl 1
hackerrank.com/image3       →    updates 1st
image object's url to 'hackerrank.com/image3'
UpdateSize 2 300 400
                    →    updates 2nd images
object's width to 300 and height to 400
```

Sample Output

```
hackerrank.com/image3 100 100
hackerrank.com/image2 300 400
hackerrank.com/image1 100 100
```

Explanation

Compiled successfully. All available test cases passed

Test case 0

Test case 1

Test case 2

Input (stdin) — Run as Custom Input | Download

```
2
hackerrank.com/image1 100 100
hackerrank.com/image2 200 200
```

the three operations listed above.

▾ Sample Case 0

Sample Input For Custom Testing

```
STDIN
Function
-----                                   --
------
2                                   →    2
images to be created initially
hackerrank.com/image1 100
100           →    url =
'hackerrank.com/image1', width = 100, height =
100
hackerrank.com/image2 200
200           →    url =
'hackerrank.com/image2', width = 200, height =
200
3
                              →    numberOfOperati
ons = 3
Clone 1
                         →    clones 1st image
object
UpdateUrl 1
hackerrank.com/image3       →    updates 1st
image object's url to 'hackerrank.com/image3'
UpdateSize 2 300 400
                   →    updates 2nd images
object's width to 300 and height to 400
```

Sample Output

```
hackerrank.com/image3 100 100
hackerrank.com/image2 300 400
hackerrank.com/image1 100 100
```

Explanation

The first line of input contains integer 2, which denotes 2 images that need to be created initially. 2 images are created with the inputs from the 2nd and 3rd lines. The next contains number 3, the number of operations to be performed. The next operation clones the images with id 1 (i.e. the first image object). The next line operation updates the url of the image with id 1 to 'hackerrank.com/image3'. The next operation updates the width and height of the image with id 2 to 300 and 400 respectively.

Language: JavaScript (Node.js) Environment — Autocomplete Ready

```
'use strict';…
class Size {
    constructor(width, height) {
        this.width = width;
        this.height = height;
    }
}

class Image {
    constructor(url, size){
        this.url = url;
        this.size = size;
    }
    getUrl(){return this.url;}
    setUrl(url){this.url = url;}
    setSize(w, h){
        this.size.width = w;
        this.size.height = h;
    }
    getSize(){return this.size; }
    cloneImage(){
        return new Image(this.url, new Size(this.size.width, this.size.height));
    }

}
…
```

Line: 21 Col: 1

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

Input (stdin) — Run as Custom Input | Download

```
3
hackerrank.com/image1 100 100
hackerrank.com/image2 200 200
```

the three operations listed above.

▾ Sample Case 0

Sample Input For Custom Testing

```
STDIN
Function
-----                                   --
------
2                                   →    2
images to be created initially
hackerrank.com/image1 100
100           →    url =
'hackerrank.com/image1', width = 100, height =
100
hackerrank.com/image2 200
200           →    url =
'hackerrank.com/image2', width = 200, height =
200
3
                              →    numberOfOperati
ons = 3
Clone 1
                         →    clones 1st image
object
UpdateUrl 1
hackerrank.com/image3       →    updates 1st
image object's url to 'hackerrank.com/image3'
UpdateSize 2 300 400
                   →    updates 2nd images
object's width to 300 and height to 400
```

Sample Output

```
hackerrank.com/image3 100 100
hackerrank.com/image2 300 400
hackerrank.com/image1 100 100
```

Explanation

The first line of input contains integer 2, which denotes 2 images that need to be created initially. 2 images are created with the inputs from the 2nd and 3rd lines. The next contains number 3, the number of operations to be performed. The next operation clones the images with id 1 (i.e. the first image object). The next line operation updates the url of the image with id 1 to 'hackerrank.com/image3'. The next operation updates the width and height of the image with id 2 to 300 and 400 respectively.

Language: JavaScript (Node.js) Environment — Autocomplete Ready

```
'use strict';…
class Size {
    constructor(width, height) {
        this.width = width;
        this.height = height;
    }
}

class Image {
    constructor(url, size){
        this.url = url;
        this.size = size;
    }
    getUrl(){return this.url;}
    setUrl(url){this.url = url;}
```

Line: 21 Col: 1

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
Test case 5
Test case 6

```
5
Clone 2
UpdateUrl 2 hackerrank.com/image10
UpdateUrl 3 hackerrank.com/image20
UpdateSize 1 300 400
Clone 3
```

Your Output (stdout)

```
hackerrank.com/image1 300 400
hackerrank.com/image10 200 200
hackerrank.com/image20 200 200
hackerrank.com/image2 200 200
hackerrank.com/image20 200 200
```