CANVAS

What is our GOAL for this MODULE?

We learned to add components on the canvas.

What did we ACHIEVE in the class TODAY?

- Made a canvas web app to move a rover over a canvas.
- Changed the background of the canvas with the real mars images taken from NASA API.

Which CONCEPTS/ CODING did we cover today?

- Added images on canvas.
- Learned about ASCII values.
- Understood how the keyboard keys use ASCII values for identifying the key that is pressed.

How did we DO the activities?

HTML code:

You were given a pre-written HTML code in today's class.

```
<html>
<head>
<style>
body{
    background-image: url("mars.gif");
    background-position: center;
    background-size: cover;
}
#myCanvas
{
    border-width:10px;
    background-color: white;
    border-style:ridge;
}

h1,h4
{
    color: white !important;
}
</style>
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/css/bootstrap.min.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
    <title>Moving Rover On Mars</title>
</head>
<body>
<center>
<h1>Moving Rover On Mars</h1>
<h4>
    <b>NOTE : </b> IF ROVER IS NOT VISIABLE PRESS ANY ARROW KEY
</h4>

</center>
<script src="main.js">
</script>
</body>
</html>
```


The above pre-written code contains:

- The bootstrap links.
- The **style** for **body** tag adds the background to the screen.
- The **style** for **canvas** adds a border to the canvas.

- The **style** for the **h1** and **h4** tags changes the **h1** and **h2** header colors to white.
- **center** tag
- **h1** tag has the heading for the page.
- **h4** tag contains the note.
- **link** to the external JS file.

1. First, add an **onload** event with a function to the **body** tag.

```
<body onload="add();">
```

   - Once the page is loaded, the body is loaded and this **onload** event will occur.
   - So inside this **onload** event we have placed an **add** function. This means when the page is loaded the add function will be called automatically.
   - We are doing this because when the page is loaded we want the canvas, rover and the background image to be loaded.

2. Now add the canvas.

```
<canvas id="myCanvas" width="800" height="600">
</canvas>
```

   - Keep the id as **myCanvas.** Use it for setting the reference for the canvas in javascript code.
   - Keep the width and height the same as the above code so that the canvas looks big enough to play and does not go out of the screen.

3. Now add bootstrap class **text-danger** to the **b** tag which is inside the last **h4** tag.

```
<h4>
    <b class="text-danger">NOTE : </b> IF ROVER IS NOT VISIABLE PRESS ANY ARROW KEY
```

   - So the **text-danger** class will add red color to the note text as shown in the following screenshot:

      - This note is added because sometimes other images are overlapped by the background.

Javascript code:

1. First, get the reference of canvas and store it inside variables as follows:

```
canvas = document.getElementById('myCanvas');
ctx = canvas.getContext("2d");
```

2. Now, define the width of the rover image and store it inside a variable.

```
rover_width = 100;
```

3. Define the height of the rover image and store it inside a variable.

```
rover_height = 90;
```

   - Set the width as 100px and height as 90px because we don't want the rover to look too big on the canvas.
   - Store the width and height in a variable as it will be easy to use these variables later in the code, rather than remembering the actual value.

4. Define the **x coordinates** for the rover image and store it inside a variable.

```
rover_x = 10;
```

***Note: It is not compulsory to give 10 x-coordinates, it can be any number we want. Only remember that x-coordinates can't be more than 700 because we have set the width of the canvas to 800, and we have defined the width of the rover as 100. So we don't want the rover to be plotted outside the canvas, or half plotted on the canvas and half out of the canvas like shown in the image below:**

5. Define the **y** coordinates for the rover image and store it inside a variable.

```
rover_y = 10;
```

***Note: It is not compulsory to give 10 y-coordinates, it can be any number we want. Only remember that y-coordinates can't be more than 500, because we have set the height of the canvas to 600, and we have defined the height of the rover as 90. So we don't want the rover to be plotted outside the canvas, or halt plotted on the canvas and half out of the canvas like shown in the image below:**

6. Define two variables and store the image path of background and rover image inside these variables.

```
background_image = "mars.jpg"
rover_image = "rover.png";
```

7. Write the **add** function which is being called on loading of the body. This function will add a background image of Mars and add a rover image.

Code of function add:

```
function add() {
    background_imgTag = new Image(); //defining a variable with a new image
    background_imgTag.onload = uploadBackground; // setting a function, onloading this variable
    background_imgTag.src = background_image;    // load image

    rover_imgTag = new Image(); //defining a variable with a new image
    rover_imgTag.onload = uploadrover; // setting a function, onloading this variable
    rover_imgTag.src = rover_image;    // load image
```

Explaining the code:

- First, define the function.

```
function add() {
```

- Now add code for adding Mars background to the canvas.

```
background_imgTag = new Image(); //defining a variable with a new image
background_imgTag.onload = uploadBackground; // setting a function, onloading this variable
background_imgTag.src = background_image;    // load image
```

  - The preceding code is added so that the image is loaded in the browser before it is drawn on a canvas. It is compulsory to load the image first before using it on canvas.

Explaining each part of the preceding code:

```
background_imgTag = new Image();
```

  - This means we are setting a new image to the variable **background_imgTag**.
    - It is not compulsory to use this variable name, the variable name can be anything.

```
background_imgTag.onload = uploadBackground;
```

  - This means we are setting an **onload** function with the **background_imgTag** variable and we are assigning **uploadBackground** function to it.
  - So whenever the **background_imgTag** variable is called, the **uploadBackground** function will load automatically.
  - **Onload** is used because the variable **background_imgTag** should be loaded with the function **uploadBackground,** before the variable **background_imgTag** is called.

```
background_imgTag.src = background_image;
```

- We are setting an image src to **background_imgTag** variable and assigning the value of **background_image** variable, which has **mars.jpg**. This way, when the variable is called, it has the mars image.

- Now add the code for adding Rover image to the canvas.

```
rover_imgTag = new Image(); //defining a variable with a new image
rover_imgTag.onload = uploadrover; // setting a function, onloading this variable
rover_imgTag.src = rover_image;   // load image
```

- The preceding code is added so that the image is loaded in the browser before it is drawn on a canvas. It is compulsory to load the image first before using it on canvas.

Explaining each part of the preceding code:

```
rover_imgTag = new Image();
```

- This means we are setting a new image to the variable **rover_imgTag**.
  - It is not compulsory to use this variable name, the variable name can be anything.

```
rover_imgTag.onload = uploadrover;
```

- This means we are setting an **onload** function with this variable and then we are assigning **uploadrover** function to it.
  This function will come later in the class.
- So whenever the variable **rover_imgTag** is called, the **uploadrover** function will be loaded automatically.
- **onload** is used because the variable **rover_imgTag** should be loaded with the function **uploadrover,** before the variable **rover_imgTag** is called.

```
rover_imgTag.src = rover_image;
```

- We are setting an image src to this **rover_imgTag** variable and assigning the value of **rover_image** variable, which has ("rover.png"). This way, when the variable is called, it has the image of the rover.

8. Now add **uploadBackground** function. This is called to upload the background image.

```
function uploadBackground() {
    ctx.drawImage(background_imgTag, 0, 0, canvas.width, canvas.height);
}
```

- Define the function.

```
function uploadBackground() {
```

- Set the background on the canvas.
- The **drawImage**() method draws an image onto the canvas.
- **Syntax**: **ctx.drawImage(img,x,y,width,height)**;
  - **ctx** is the drawing reference of the canvas.
  - **img**: This should be the image variable, which should be defined before.
  - **x**: at what x coordinate the image should be plotted.
  - **y**: at what y coordinate the image should be plotted.
  - **width** of the image
  - **height** of the image

```
ctx.drawImage(background_imgTag, 0, 0, canvas.width, canvas.height);
```

Explaining the code:

- **ctx** is the reference of the canvas.
- **drawImage** is the function for drawing an image on canvas.
- **background_imgTag**: This will be the variable which we defined for setting of the background image.
- **0**: It is the x coordinate. We are setting it to 0 because we want the background to start from where the canvas begins.
- **0**: It is the y coordinate. We are setting it to 0 because we want the background to start from where the canvas begins.
- **canvas.width**: canvas.width means **800**. We are using this because we want

the background to cover the whole canvas.

- **canvas.height** - canvas.height means **600**. We are using this because we want the background to cover the whole canvas.

9. Add **uploadrover** function. This is called to upload the rover image.

```
function uploadrover() {
    ctx.drawImage(rover_imgTag, rover_x, rover_y, rover_width, rover_height);
}
```

- Define the function.

```
function uploadrover() {
```

- Set the rover image on the canvas.

```
    ctx.drawImage(rover_imgTag, rover_x, rover_y, rover_width, rover_height);
```

Explaining the code:

- **ctx** is the reference of the canvas.
- **drawImage** is the function for drawing an image on canvas.
- **rover_imgTag**: This will be the variable which we defined for setting the rover image.
- **rover_x**: It is the x coordinate. **rover_x** will be 10, as we have done before**.**
- **rover_y**: It is the y coordinate. **rover_y** will be 10, as we have done before**.**
- **rover_width**: rover_width means **100**, as we have already defined rover width inside the variable **rover_width.**
- **rover_height**: rover_height means **90**, as we have already defined rover height inside the variable **rover_height.**

For the computer to understand what key we press from the keyboard there is something called ASCII values. Every key has a defined value. These values are known as ASCII (American Standard Code for Information Interchange) values. So every key in our keyboard has a defined value known as ASCII. When we press any key, the computer uses these ASCII values to see which key is pressed.

List of keys and their ASCII value:

Keys	ASCII Value
a	65
b	66
c	67
d	68
e	69
f	70
g	71
h	72
i	73
j	74
k	75
l	76
m	77
n	78
o	79
p	80
q	81
r	82
s	83
t	84
u	85
v	86
w	87

x	88
y	89
z	90
Up	38
Down	40
Left	37
Right	39
Alt	18
Ctl	91
Esc	27

10. Now add the code for controlling the rover.

```
window.addEventListener("keydown", my_keydown);

function my_keydown(e)
{
    keyPressed = e.keyCode;
    console.log(keyPressed);
        if(keyPressed == '38')
        {
            up();
            console.log("up");
        }
        if(keyPressed == '40')
        {
            down();
            console.log("down");
        }
        if(keyPressed == '37')
        {
            left();
            console.log("left");
        }
        if(keyPressed == '39')
        {
            right();
            console.log("right");
        }
}
```

- Use the **keydown** event listener for grabbing the value of the pressed key.

***Note: All the keys on the keyboard have ASCII value. For example: The up arrow key has value 38. The 'a' word on the keyboard has value 65.**

- First, assign the event listener to the window to grab the value of the key pressed.
    - Decide which key has been pressed using this key value and then call our function - same as it was done for other event listeners.

```
window.addEventListener("keydown", my_keydown);
```

    - **keydown** event means whenever a key is pressed this event will get executed and will call **my_keydown** function which we have defined. This is the same as a **mousedown** event. When the mouse is clicked,

the **mousedown** event gets executed.

- In the preceding code:
  - **window** means the whole screen
  - a **keydown** event is listening if any key is pressed
  - then **my_keydown** function will get executed.
- Define the **my_keydown** function

```
function my_keydown(e)
{
```

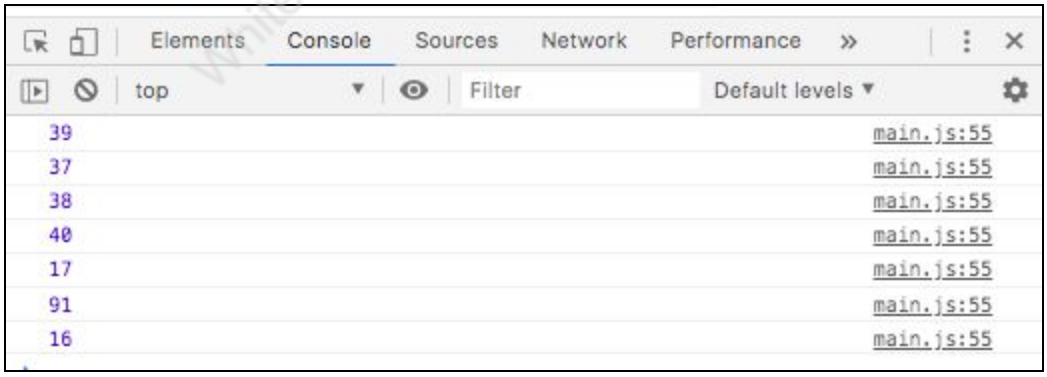
- Now fetch the value of the key pressed.
  - **e.keyCode** will get the value of the **keypressed**. Store it inside a variable **keyPressed**.

```
keyPressed = e.keyCode;
```

- Now console this value.

```
console.log(keyPressed);
```

- Now open and run the file. Then open the console and press any key. You will see as you press the key the value of that key is printed on the console screen.
- Output:

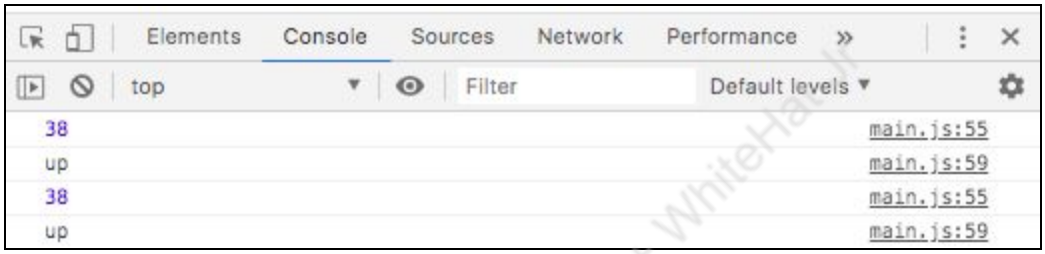


- Check **if** the key value is equal to the **up** arrow key value. The value of the **up** arrow key is **38**.
- It checks **if** the value of the **keyPressed** variable is **38** then it executes:
  - First, it **calls a function up()** (This will be covered in the next class.)

- Then console **up**. This is to confirm that everything is working fine.

```
if(keyPressed == '38')
{
    up();
    console.log("up");
}
```

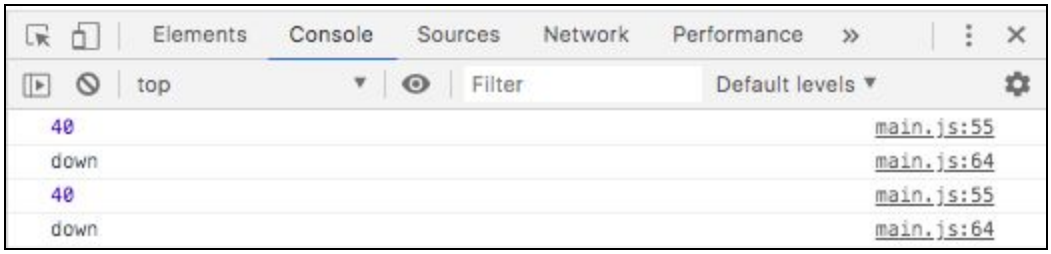
- Output:



- Now check **if** the key value is equal to the **down** arrow key value. The value of the **down** arrow key is **40.**
- It checks **if** the value of the **keyPressed** variable is **40** then it executes:
  - First, it calls an **up()** function. (This will be covered in the next class.)
  - Then console **down**. This is to confirm that everything is working fine.

```
if(keyPressed == '40')
{
    down();
    console.log("down");
}
```

- Output:



- Now check **if** the key value is equal to the **left** arrow key value. The value of the **left** arrow key is **37.**
- It checks **if** the value of the **keyPressed** variable is **37** then it executes:
    - First, it calls a **left()** function. (This will be covered in the next class.)
    - Then console **left**. This is to confirm that everything is working fine.

```
if(keyPressed == '37')
{
    left();
    console.log("left");
}
```

- Output:

```
37                                                    main.js:55
left                                                  main.js:69
37                                                    main.js:55
left                                                  main.js:69
```

- Now check **if** the key value is equal to the **right** arrow key value. The value of the **left** arrow key is **39.**
- It checks **if** the value of the **keyPressed** variable is **39** then it executes:
    - First, it calls a **right()** function. (This will be covered in the next class.)
    - Then console **right**. This is to confirm that everything is working fine.

```
if(keyPressed == '39')
{
    right();
    console.log("right");
}
```

- Output:

```
39                                                    main.js:55
right                                                 main.js:74
39                                                    main.js:55
right                                                 main.js:74
```

What's NEXT?

We will learn how to move components on canvas.

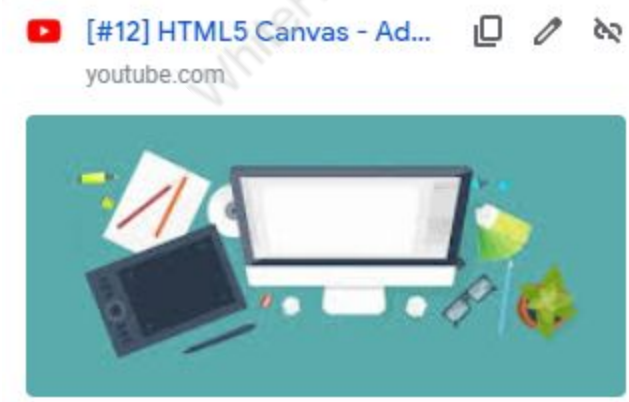
EXTEND YOUR KNOWLEDGE

Here are some Best References we've compiled together to enhance your knowledge and understanding of the concepts we learned today in the class. This will help you become a pro at coding and creating industry-grade tech products!

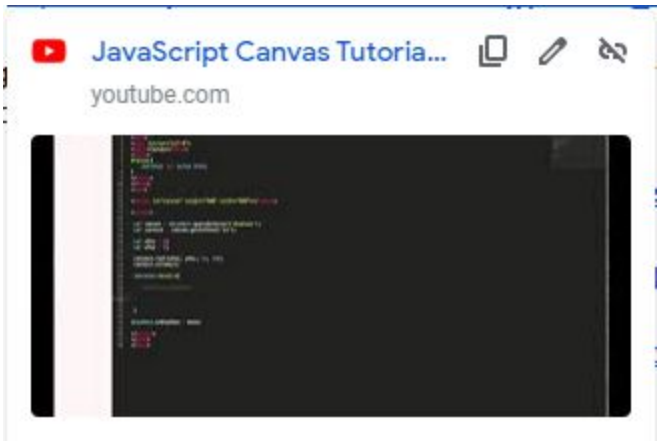
Short Videos: Watch these Short Videos to understand the application of the concepts learned in class in real-world applications.

1. https://www.youtube.com/watch?v=yFI9Kb_7X_E

2. https://www.youtube.com/watch?v=8xOKGC-g6NE



3. https://www.youtube.com/watch?v=OAjyunGPW_s



Coding Playground: Try out these code examples to get more practice in making Websites and Playstore ready apps.

1. https://www.w3schools.com/jsref/event_onkeypress.asp

2. https://www.w3schools.com/jsref/event_onload.asp

3. https://www.w3schools.com/jsref/event_onloadeddata.asp