



中国科学院大学
University of Chinese Academy of Sciences

CS101

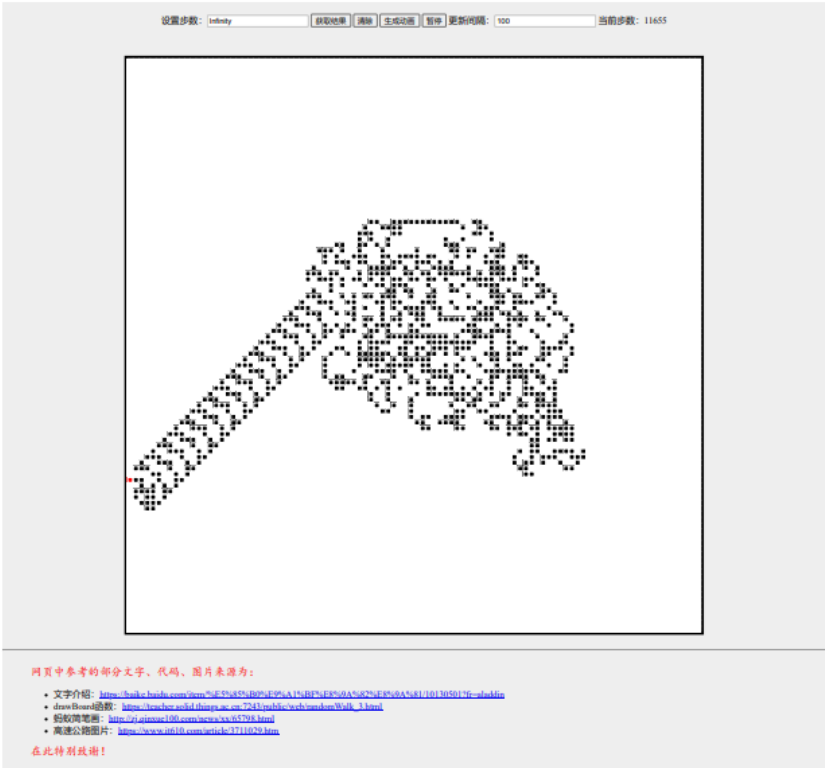
Lab Slides

Personal Artifact

Reference

- <https://www.w3schools.com/>
 - An online tutorial
- http://cs101ucas.edu.cn/file/Personal_Artifact/
 - The directory containing related code in this project

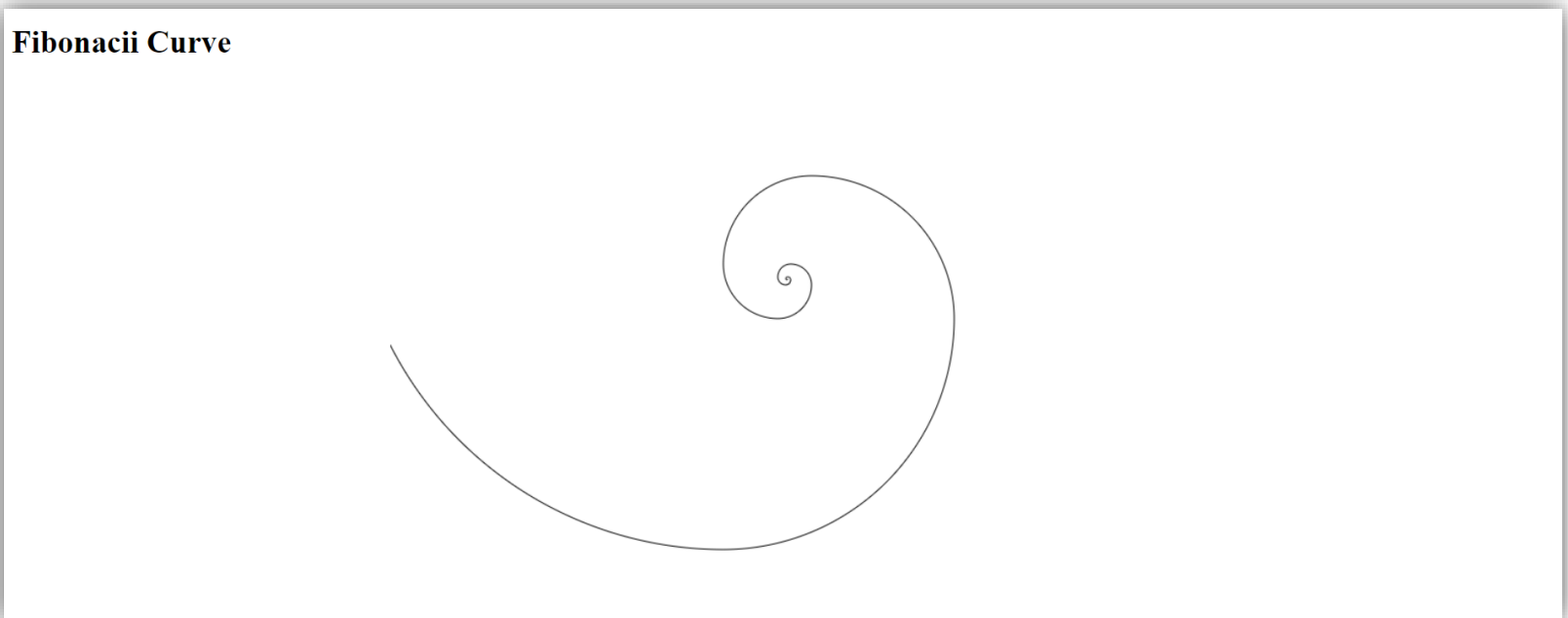
Langton's ant



- Langton's_Ant.html

Examples – Good Programming

- The webpage shows a fibonacci curve
- The programming practices is good



Fibonacci.html

Examples – Good Programming

```
// code executed directly:
var len = 14; ★ the number of Fibonacci numbers to form the Curve (Global variable)
var fibonacciArray = new Array(len); // Fibonacci numbers to form the Curve
calculate_fibonacci();
drawFibonacciArc();
// function definition:
function calculate_fibonacci(){
    fibonacciArray[0] = 1; fibonacciArray[1] = 1;
    for(var i=2;i<len;i++) fibonacciArray[i]=fibonacciArray[i-1]+fibonacciArray[i-2];
}
function drawFibonacciArc(){
    // show the whole curve by draw many 1/4 circles
    var canvas = document.getElementById("myCanvas"); // get Canvas
    var ctx = canvas.getContext("2d");
    var x0 = 400, y0 = 200;
    var x = x0, y = y0, startDegree = 0, endDegree = 90;

    for (var i = 0; i < len; i++) {
        // Draw 1/4 Circle whose radius is fibonacciNumber[i]
        var radius = fibonacciArray[i];
        ctx.beginPath();
        ctx.strokeStyle = "black";
        ctx.arc(x,y,radius,(startDegree/180)*Math.PI,(endDegree/180)*Math.PI);
        ctx.stroke();
        // update the position of the center of next 1/4 circle
        var direction = startDegree/90;
        if(i<len-1){
            var inc_x = (direction%2-2*Math.floor(direction/3))*(fibonacciArray[i+1]-fibonacciArray[i]);
            var inc_y = -(1-direction+2*Math.floor(direction/3))*(fibonacciArray[i+1]-fibonacciArray[i]);
            x += inc_x; y += inc_y;
        }
        // update the range of degrees of next 1/4 circle
        startDegree = (startDegree+90)%360;
        endDegree = (endDegree+90)%360;
    }
};
```

Put constant definitions up front.

Use descriptive names

Avoid repetitive code

Use comments to document the code

Avoid magic numbers

Examples – Tips

```
// code executed directly:
var len = 14; // the number of Fibonacci numbers to form the Curve (Global variable)
var fibonacciArray = new Array(len); // Fibonacci numbers to form the Curve
calculate_fibonacci();
drawFibonacciArc();
// function definition:
function calculate_fibonacci(){
    fibonacciArray[0] = 1; fibonacciArray[1] = 1;
    for(var i=2;i<len;i++) fibonacciArray[i]=fibonacciArray[i-1]+fibonacciArray[i-2];
}
function drawFibonacciArc(){
    // show the whole curve by draw many 1/4 circles
    var canvas = document.getElementById("myCanvas"); // get Canvas
    var ctx = canvas.getContext("2d");
    var x0 = 400, y0 = 200;
    var x = x0, y = y0, startDegree = 0, endDegree = 90;

    for (var i = 0; i < len; i++) {
        // Draw 1/4 Circle whose radius is fibonacciNumber[i]
        var radius = fibonacciArray[i];
        ctx.beginPath();
        ctx.strokeStyle = "black";
        ctx.arc(x,y,radius,(startDegree/180)*Math.PI,(endDegree/180)*Math.PI);
        ctx.stroke();
        // update the position of the center of next 1/4 circle
        var direction = startDegree/90;
        if(i<len-1){
            var inc_x = (direction%2-2*Math.floor(direction/3))*(fibonacciArray[i+1]-fibonacciArray[i]);
            var inc_y = -(1-direction+2*Math.floor(direction/3))*(fibonacciArray[i+1]-fibonacciArray[i]);
            x += inc_x; y += inc_y;
        }
        // update the range of degrees of next 1/4 circle
        startDegree = (startDegree+90)%360;
        endDegree = (endDegree+90)%360;
    }
};
```

● Canvas

1. canvas =
2. document.getElementById("...")
3. ctx = canvas.getContext("2d")
4. ctx.XXX()
ctx.stroke()

Lab Objective

- Design a **dynamic** webpage of creative expression. Successful completion of this personal artifact needs students to demonstrate their **self-learning** capability.
- Each student needs to produce and show to the class a webpage:
 - A dynamic webpage including **HTML, CSS, and JavaScript** code
 - as well as other hyperlinked files,
 - which shows your **creative expression**

Outline

- Section 1: Make a static webpage
 - HTML Structure
 - HTML Grammar
- Section 2: Make a web calculator
 - Change the result of the web calculator
 - Read input operands
 - Perform the selected computation
- Section 3: Make a web clock
 - Add a static clock
 - Make the clock dynamic
 - Debug
 - Compare JS with Go

This course will take "Tom" making a web clock and calculator as an example to introduce related programming knowledge



中国科学院大学
University of Chinese Academy of Sciences

CS101

Section 1

In this section, we will make the following webpage

The growing diary of calculator and clock

Author: *Tom* * Time: 2020/02/26

Before

Before doing it, I am not sure I can make it.

List: Tom's Purpose

1. Learn CSS
2. Learn HTML
3. Learn JS
4. Make calculator and clock

During

During doing it, I find some material, tired but happy.

Table: Tom's hard journey

Event	Feeling
Learn CSS	One day awake
Learn HTML	One day awake again
Learn JavaScript	Missing Go

After

Finally, it's done. Although it's rough, it's also my own 'son'.

Calculator:

Operands1:

Operands2:

☐ Add ☐ Sub ☐ Mul ☐ Div ☐ Mod

Result: xxxx

blog1.html

Section 1 Outline

- HTML Structure
- HTML Grammar

HTML is the abbreviation of **HyperText Markup Language**, which describes what a webpage has

HTML Structure

```
<html>
```

```
  <head>
```

```
    some code
```

```
  </head>
```

```
  <body>
```

```
    some code
```

```
  </body>
```

```
</html>
```



Any html code begins with this basic structure

Note the existence of "/"!

HTML Structure—example

Create a new file named blog.html, type the following code to that file:

[the meaning of the code will be discussed later]

```
<html>
```

```
  <head>
```

```
    <meta charset="UTF-8" >
```

```
    <title>A Good Day</title>
```

```
  </head>
```

```
  <body>
```

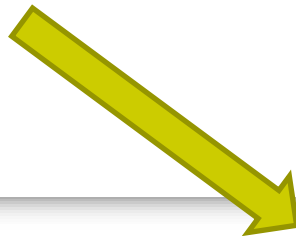
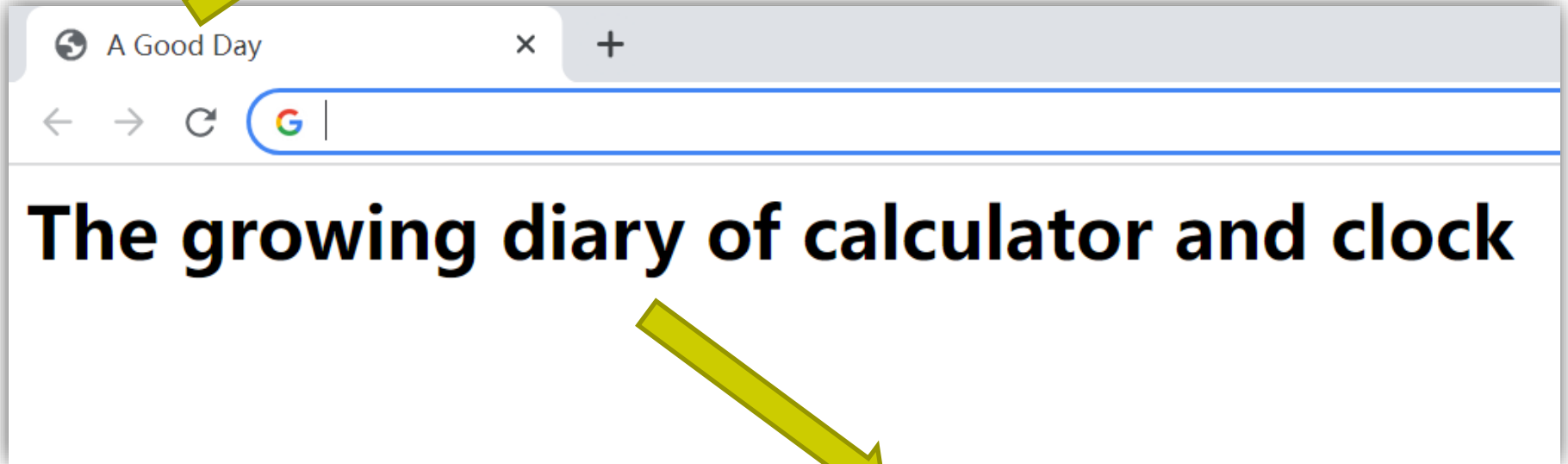
```
    <h1>The growing diary of calculator and clock</h1>
```

```
  </body>
```

```
</html>
```

open blog.html with a Web browser

<title>A Good Day</title>



<h1>The growing diary of calculator and clock</h1>

HTML Structure—example

- Any html source code file takes ".html" as filename extension

<html> html start tag

<head> head start tag

<meta charset="UTF-8"> meta tag: specify the encoding

<title>A Good Day</title> a pair of title tags: the title (name) of the webpage is inside the two tags

</head> head end tag

<body> body start tag

<h1>The growing diary of calculator and clock</h1> h1 tag: the words shown in the webpage

</body> body end tag

</html> html end tag

utf-8 is an
encoding
for text

Section 1 Outline

- HTML Structure
- HTML Grammar

HTML Grammar—Summary

- **Markup Language:** HTML is a markup language, which consists of many tags (for example, `<html>`, `</html>`, `<head>`, `</head>`, `<body>`, `</body>`)
- **Nesting** is allowed: A pair of tags can be defined inside another pair of tags. For example, inside `<html>...</html>`, `<head>...</head>` and `<body>...</body>` are defined, `<meta>` and `<title>` are defined inside `<head>...</head>`, `<h1>...</h1>` is defined inside `<body>...</body>`
- **Differ from Go:** Go is a programming language, and a Go program mainly consists of manipulation of digital symbols; however, HTML is used to specify the **content** of a webpage

HTML Grammar—concept

- **Tag:** HTML tag is a word that has the pattern "<keyword>", such as <html>
- Most HTML tags appear in pairs, following the pattern of <keyword>...</keyword>, e.g. <h1>...</h1>. The former <h1> is called **start tag**; the later </h1> is called **end tag** (**Note the /**)
- **Attribute:** HTML start tag can have attributes. An attribute's value is a string enclosed in quotation marks. The grammar of attribute is:

`<h1 attribute_name="...">...</h1>`

- **Element:** "<key_word>...</key_word>" is a HTML element:



HTML Grammar—elements in <head>

- Inside <head>...</head>, the <meta> tag and a pair of <title> tag usually appear, e.g.,

```
<head>  
  <meta charset="UTF-8">  
  <title>The Title</title>  
</head>
```

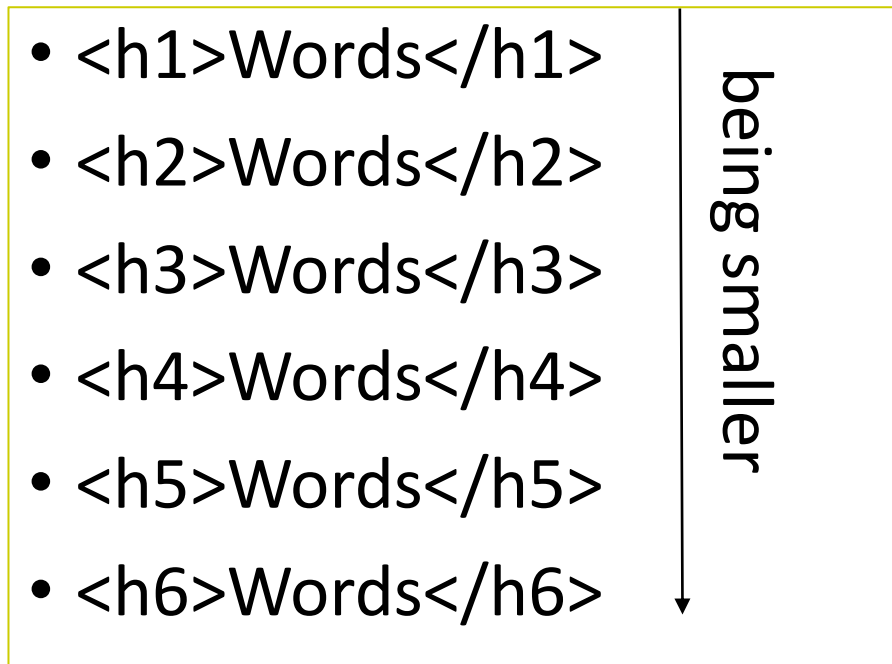
Namespace: legal strings allowed by browser (Chinese words, English words and space are legal, and multiple spaces will be merged to one space)

- Their meanings are:

Tag	Meaning
<meta charset="UTF-8">	Specify the usage of utf-8 encoding
<title>...</title>	Specify the title (name) of the webpage

Add headings and subheadings

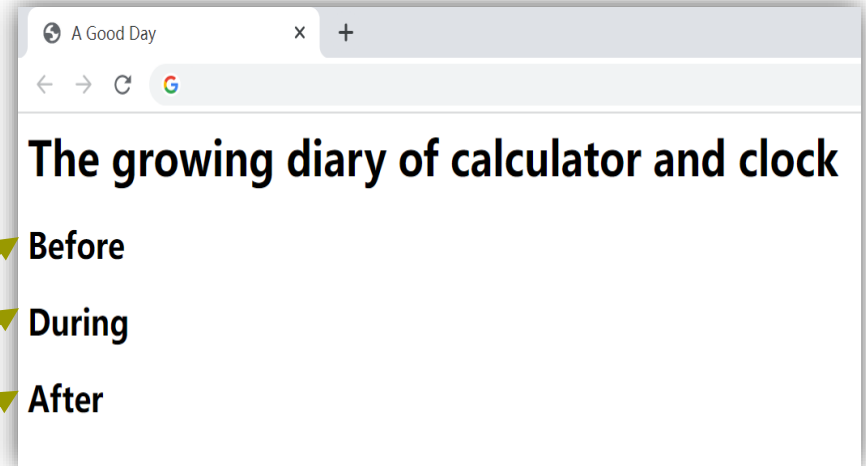
- Tom wants to talk about the creation process, so his blog consists of three parts and each part has a head. The three heads are "Before", "During" and "After". He uses the following knowledge:



Code:

```
<html>
<head>
  <meta charset="UTF-8">
  <title>A Good Day</title>
</head>
<body>
  <h1>The growing diary of calculator and clock</h1>
  <h2>Before</h2>
  <h2>During</h2>
  <h2>After</h2>
</body>
</html>
```

Result:



Add paragraph—paragraph, newline, horizontal line, comment

- After Tom adds subheads, he begins to write paragraphs below each subhead, hoping that there are empty lines between paragraphs. The knowledge he uses is as follows:

Paragraph: `<p>text</p>`

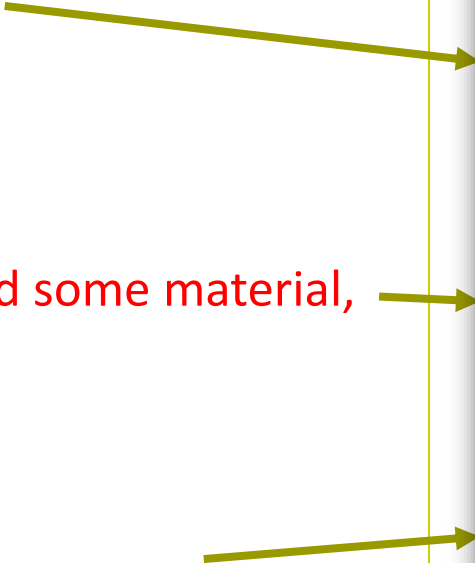
Newline: `
` (or `
`)

Horizontal line: `<hr/>` (or `<hr>`)

Comment: `<!--comment_content-->`

Code (body only):

```
<body>
  <h1>The growing diary of calculator and
clock</h1>
  <h2>Before</h2>
  <p>Before doing it, I am not sure I can
make it.</p>
  <br>
  <hr>
  <h2>During</h2>
  <p>During doing it, I find some material,
tired but happy.</p>
  <br>
  <hr>
  <h2>After</h2>
  <p>Finally, it's done. Although it's rough,
it's also my own 'son'.</p>
</body>
```



Result:

The growing diary of calculator

Before

Before doing it, I am not sure I can make it.

During

During doing it, I find some material, tired but happy.

After

Finally, it's done. Although it's rough, it's also my own 'son'.

Add Name and Time—text formatting

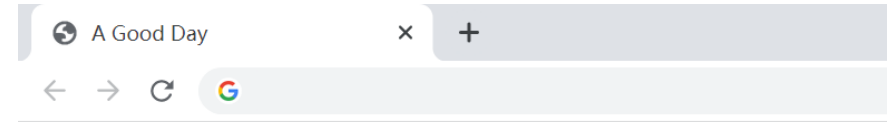
- After adding paragraphs, Tom wants to write his name and time to his blog. The name should be italicized with a small star, and the time should be italicized. **Text Formatting tags** are used.
 - Text formatting tags usually appear in pairs, and specify the format of the text between the two tags.

- Bold: `bold text`
- Italic: `<i>italic text</i>`
- Superscript: `^{superscript text}`
- Subscript: `_{subscript text}`

Code (body only):

```
<body>
  <h1>The growing diary of calculator
and clock</h1>
  Author: <b><i>Tom</i></b>
  <sup>*</sup> Time: <i>2020/02/26</i>
  <h2>Before</h2>
  <p>Before doing it, I am not sure I can
make it.</p>
  <br>
  <hr>
  <h2>During</h2>
  <p>During doing it, I find some
material, tired but happy.</p>
  <br>
  <hr>
  <h2>After</h2>
  <p>Finally, it's done. Although it's
rough, it's also my own 'son'.</p>
</body>
```

Result:



The growing diary of calculator

Author: ***Tom*** * Time: 2020/02/26

Before

Before doing it, I am not sure I can make it.

During

During doing it, I find some material, tired but happy.

After

Finally, it's done. Although it's rough, it's also my own 'son'.


Add progress chart—table

- After Tom adds name and time, he looks back on his process of creating his blog and wants to record the process in a table. The knowledge he uses is:
 - A **table** can be defined by `<table>...</table>`, inside which n rows can be defined by n `<tr>...</tr>`. Inside a `<tr>...</tr>`, m cells can be defined by m `<td>...</td>`. The text inside `<td>...</td>` will be displayed in the cell defined by the two tags.
 - An example:

```
<table>  
  <tr><td>(row0,col0)</td><td>(row0,col1)</td></tr>  
  <tr><td>(row1,col0)</td><td>(row1,col1)</td></tr>  
</table>
```

Code ("During" part):

The line thickness of
of table border



```
<table border="1">
```

```
  Table: Tom's hard journey
```

```
  <tr>
```

```
    <td>Event</td><td>Feeling</td>
```

```
  </tr>
```

```
  <tr>
```

```
    <td>Learn CSS</td><td>One day awake</td>
```

```
  </tr>
```

```
  <tr>
```

```
    <td>Learn HTML</td><td>One day awake again</td>
```

```
  </tr>
```

```
  <tr>
```

```
    <td>Learn JavaScript</td><td>Missing Go</td>
```

```
  </tr>
```

```
</table>
```

Result:

During

During doing it, I find some material, tired but happy.

Table: Tom's hard journey

Event	Feeling
Learn CSS	One day awake
Learn HTML	One day awake again
Learn JavaScript	Missing Go

Add purpose—ordered list

Replace
... with
... can delete
the order numbers.

- Tom realizes that the "Before" part of his blog is still empty, so he writes his purpose of writing the webpage in an ordered list:
 - **Ordered list** is defined by ..., inside which an element in the list can be defined by ... which contains the text of the list element in the middle. The order number of elements in an ordered list is automatically added when the list is displayed in a browser.

```
<ol>
```

```
    <li>Text</li>
```

```
    <li>Text</li>
```

```
</ol>
```

Code ("Before" part):

List: Tom's Purpose

```
<ol>
  <li>Learn CSS</li>
  <li>Learn HTML</li>
  <li>Learn JS</li>
  <li>Make calculator and
clock</li>
</ol>
<!--br-->
```

Result:

List: Tom's Purpose

1. Learn CSS
2. Learn HTML
3. Learn JS
4. Make calculator and clock

Add calculator—form

- Tom, after adding his purpose in a list, wants to display a calculator in his blog, so he designs the appearance of the calculator with "form" in HTML.
- **Form** is composed of `<form>...</form>`, inside which some **single** `<input>` tag can be used to define some kinds of **Form Element**.
- Form is used to capture user's input and Form Elements are usually text box where user can enter text, or option where user can select it.

Add calculator—form element

- Different `<input>` elements can be distinguished by their **type** attributes

Grammar	Description
<code><input type="text"></code>	A text box where user enters text.
<code><input type="password"></code>	A text box where user enters password which will be displayed as many dots.
<code><input type="radio" name="xx"></code>	A radio. If many radios in a form have the same name attribute, only one can be selected.
<code><input type="checkbox"></code>	A checkbox.
<code><input type="button" value="xx"></code>	A Button(the value of value attribute is the text on the button when displayed)

Radios with the same **name** attribute inside a `<form>...</form>` is a group (a group means only one of the group can be selected)

Code ("After" part):

Calculator:

```
<form>
```

```
  Operands1: <input type="text"><br/>
```

```
  Operands2: <input type="text">
```

```
<br/>
```

```
<input type="radio" name="op">Add
```

```
<input type="radio" name="op">Sub
```

```
<input type="radio" name="op">Mul
```

```
<input type="radio" name="op">Div
```

```
<input type="radio" name="op">Mod
```

```
<br/>
```

```
<input type="button" value="Compute">
```

```
</form>
```

Result:

After

Finally, it's done. Although it's rough, it's also my own 'son'.

Calculator:

Operands1:

Operands2:

☐ Add ☐ Sub ☐ Mul ☐ Div ☐ Mod

- Note: Only Appearance of Calculator, no functions.

HTML Grammar—Block & Inline Elements

- Thinking what to write next in his blog, Tom finds the following knowledge:
- In a webpage, some HTML elements can be **embedded in a line**, such as `<i>...</i>`. Elements with this property are called **Inline Elements**. On the contrary, elements like `<h1>...</h1>` must **start with a new line**, and these elements are called **Block Elements**.
- Examples:
 - `<div>...</div>`: A block element which has no special meaning, and just divides a webpage into smaller blocks, making operations on all elements in such a block easier.
 - `...`: An inline element which has no special meaning. It can be used to combine some elements in the same line or embed some elements in a line to make operations on the embedded part more easier.

Code Blocking—division tag

- Tom thinks the code is a little bit messy, so he uses `<div>` to make the three parts in his blog into three `<div>` elements, which makes the arrangement of the code better.

```
<body>
.....
<div>
  <h2>Before</h2>
  .....
</div>
.....
<div>
  <h2>After</h2>
  .....
</div>
</body>
```

- Adding `<div>` just changes the structure of code, but the webpage displayed in a browser is not changed.

Leave space for calculator results—span tag

- Suddenly a strategy comes to Tom's mind: the result in the calculator can be added using `` in the `<form>`

Calculator:

`<form>`

Operands1: `<input type="text">
`

Operands2: `<input type="text">`

`
`

`<input type="radio" name="op">Add`

`<input type="radio" name="op">Sub`

`<input type="radio" name="op">Mul`

`<input type="radio" name="op">Div`

`<input type="radio" name="op">Mod`

`
`

Result: `xxxx
`

`<input type="button" value="Compute">`

`</form>`

Result:

After

Finally, it's done. Although it's rough, it's also my own 'son'.

Calculator:

Operands1:

Operands2:

☐ Add ☐ Sub ☐ Mul ☐ Div ☐ Mod

Result: xxxx

Add Styles

- Tom starts to think about how to make his webpage more beautiful, so he learns CSS
 - CSS(Cascading Style Sheets) is a language specifying the styles(e.g. font color, background color) of HTML elements. Its statements has the form of "name: value;".
 - An HTML file has two methods to insert CSS statements.
- Method 1: **Style attribute** in start tag. The value of the attribute is some CSS statements.
 - Example:
 - <p style="background-color: gold;">a paragraph with golden background</p>
 - <p style="color: red;">a paragraph with red font color</p>
 - You can specify more than one styles at the same time:
 - <h1 style="color: blue; text-align: center;">a blue and centered heading</h1>

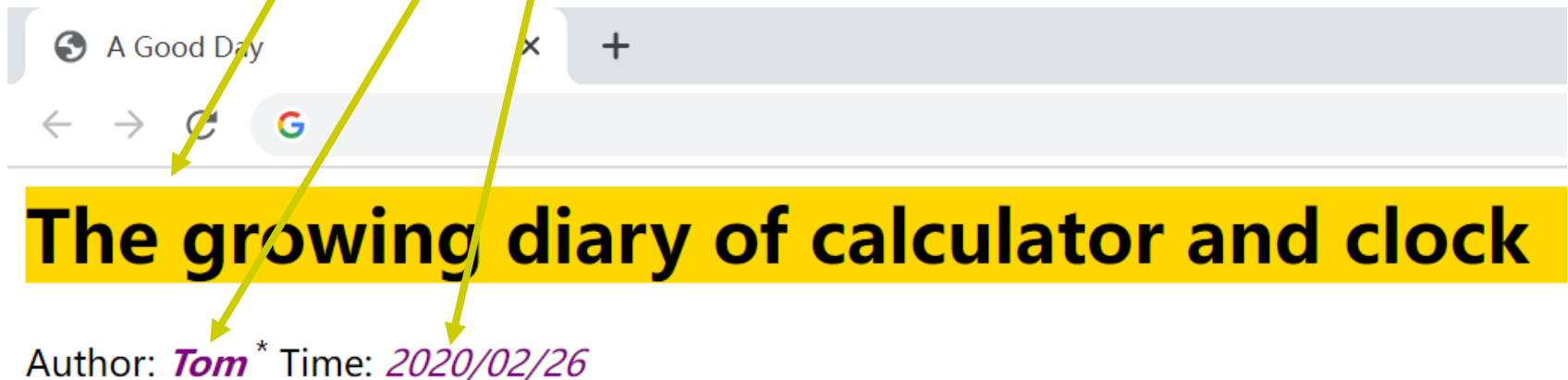
Note: the names and values in a CSS statements cannot be arbitrary strings, but the ones defined by CSS grammar.

Add color to elements

```
<h1 style="background-color: gold;">The growing diary of calculator  
and clock</h1>
```

```
  Author: <b><i style="color: purple;">Tom</i></b> <sup>*</sup>  
Time: <i style="color: purple;">2020/02/26</i>
```

Result:



Add color to elements

- Tom thinks his blog is not good enough, so he wants to make the three parts (Before, During, After) more beautiful. He uses the following knowledge:
- **Method 2:** Inside `<head>...</head>`, `<style>...</style>` can be defined, within which some CSS statements can appear.

- Example: Select some elements and change their styles by CSS statements in the curly braces

```
<style>  
  h1 {color: red;} /* Specify the style of a kind of tag: All <h1> elements have red fonts*/  
  #id_attribute {color: green;} /* Specify the style of an element: The element  
specified by the id attribute [learn later] has green fonts*/  
  .class_attribute {color: yellow;} /* Specify the style of a group of elements: The  
elements with the same class attributes [learn later] have yellow fonts*/  
</style>
```

Some Common Attributes

- Almost all elements can define the following attributes:

Attribute	Description
id	Identifier of an element. Unique. Can be any string.
name	Name of an element. Can be any string.
class	Class of an element. One element can have multiple class. Different elements can have the same class. Can be any string.
style	The style of an element.

- Grammar:
 - `<tag_name attribute1_name="..." attribute2_name="..." >...</tag_name>`

Add Attributes

```
<body>
.....
<div id="before">
.....
</div>
<div id="during">
.....
</div>
<div id="after">
.....
</div>
</body>
```

- id attribute cannot change the appearance of the webpage, but can be selected by CSS code inside `<style>...</style>`

Add color

No white space between #
and before

```
<style>
  #before {
    background-color: skyblue;
  }
  #during {
    background-color: springgreen;
  }
  #after {
    background-color: tomato;
  }
  h2 {
    background-color: violet;
  }
</style>
```

Result:

The growing diary of calculator and clock

Author: *Tom* * Time: 2020/02/26

Before

Before doing it, I am not sure I can make it.

List: Tom's Purpose

1. Learn CSS
2. Learn HTML
3. Learn JS
4. Make calculator and clock

During

During doing it, I find some material, tired but happy.

Table: Tom's hard journey

Event	Feeling
Learn CSS	One day awake
Learn HTML	One day awake again
Learn JavaScript	Missing Go

After

Finally, it's done. Although it's rough, it's also my own 'son'.

Calculator:

Operands1:

Operands2:

☐ Add ☐ Sub ☐ Mul ☐ Div ☐ Mod

Result: xxxx



中国科学院大学
University of Chinese Academy of Sciences

CS101

Section 2

In this section, we will make the following webpage

The growing diary of calculator and clock

Author: *Tom* * Time: 2020/02/26

Before

Before doing it, I am not sure I can make it.

List: Tom's Purpose

1. Learn CSS
2. Learn HTML
3. Learn JS
4. Make calculator and clock

During

During doing it, I find some material, tired but happy.

Table: Tom's hard journey

Event	Feeling
Learn CSS	One day awake
Learn HTML	One day awake again
Learn JavaScript	Missing Go

After

Finally, it's done. Although it's rough, it's also my own 'son'.

Calculator:

Operands1: 365

Operands2: 24

☐ Add ☐ Sub ☒ Mul ☐ Div ☐ Mod

Result: 8760

Compute

Where to insert JavaScript code

```
<html>
  <head>
    <meta charset="UTF-8">
    <title>A Good Day</title>
  </head>
  <body>
    .....(HTML part)
    <script>
      ...
    </script>
  </body>
</html>
```



JavaScript part

Section 2 Outline

- Change the result of the web calculator
 - JavaScript object representing an HTML element
- Read input operands
 - JavaScript "onclick" event
- Perform the selected computation
 - JavaScript Array object
 - JavaScript for-loop

JavaScript Programming—Object

- **Object** can be seen as a compound data type, and it is a set of **properties** and **methods**, for example, a person can be a object, who has **properties** such as height, weight, age and gender, and **methods** such as walking, running and talking.
- The dot notation "." is used to access the properties and methods of an object.
- The properties of an object: like variables
 - `obj.property`
- The methods of an object: like functions (methods)
 - `obj.method();`

JavaScript Programming — Object representing an HTML element

- In order to implement the calculator, Tom firstly needs to know how to modify the result of the calculator, so he learns the knowledge of JS accessing HTML elements.
- JS can take defined HTML elements as object to access it (2 steps):
 1. Get the HTML element by its id attribute
 - `var x = document.getElementById("p1");`
 - the return value of `document.getElementById("p1")` is the object representing the HTML element whose id attribute has the value "p1"
 - therefore, x is the HTML element whose id attribute has the value "p1"
 2. Access the attributes of HTML elements with the object representing this HTML elements
 - `x.innerHTML = "newContent";`
 - set the innerHTML attribute of the HTML element represented by x to "newContent"
 - the content between the tags of the HTML element whose id attribute is "p1" becomes "newContent"
- The 2 steps can be realized by one statement:
 - `document.getElementById("p1").id = "newContent";`

Change Result

Code:

```
<script>  
    var z = 101;  
    var result = document.getElementById("result");  
    result.innerHTML = z;  
</script>
```

Result:

Calculator:

Operands1:

Operands2:

☒ Add ☐ Sub ☐ Mul ☐ Div ☐ Mod

Result: 101

Compute

Section 2 Outline

- Change the result of the web calculator
 - JavaScript object representing an HTML element
- Read input operands
 - JavaScript "onclick" event
- Perform the selected computation
 - JavaScript Array object
 - JavaScript for-loop

JavaScript Programming—event

- After Tom learns how to change the result of the calculator, he also needs to implement: Read the input numbers when the button is clicked. Therefore, He learns the knowledge of event:
- 3 Common events:
 - User click a button(<input type="button" **onclick="JS code">**)
 - User change his/her input(<input type="text" **oninput="JS code">**)
 - The browser completely loads a webpage (<script> window.onload = *a function definition* </script>)
- When an event happens, a JS code can be used to tackle the event.

Add event

- The "JS code" of onclick attribute is usually a function invocation statement. The code can be:

Code:

```
...  
<input type="button" value="Compute"  
  onclick="run_calculater()">  
...  
<script>  
  function run_calculater(){  
    var operand1 =  
document.getElementById('operand1').value - 0;  
    var operand2 =  
document.getElementById('operand2').value - 0;  
    var z = operand1 + operand2;  
    document.getElementById('result').innerHTML = z;  
  }  
</script>
```

Result:

Calculator:

Operands1:

Operands2:

☐ Add ☐ Sub ☐ Mul ☐ Div ☐ Mod

Result: xxxx

After click "Compute" button:

Calculator:

Operands1:

Operands2:

☐ Add ☐ Sub ☐ Mul ☐ Div ☐ Mod

Result: 303

Section 2 Outline

- Change the result of the web calculator
 - JavaScript object representing an HTML element
- Read input operands
 - JavaScript "onclick" event
- Perform the selected computation
 - JavaScript Array object
 - JavaScript for-loop

JavaScript Programming—Array object

- Tom reads the two operands of the calculator successfully, but he still needs to run the computing process, so he uses an array to store the five types of computation:
 - **Create** an Array object

```
var ar = new Array();           // variable ar stores an array (No length specified, just
                                // assign, the length will be determined automatically)
var ar = ["Java", "Script"];    // create an array using "[]"
var ar = [0, "ABC", [true,false]]; // one array can have different types of elements,
                                // and the index starts from 0
```
 - **Change** an element of an array

```
ar[0]=1;
```
 - **Property** of an array

```
var ar=[0,1,2];
var len = ar.length; // len=3, which is the length of the array
```

Use an array of functions

```
<script>
...
function add(x,y){ return x+y; } //Add
function sub(x,y){ return x-y; } //Sub
function mul(x,y){ return x*y; } //Mul
function div(x,y){ return x/y; } //Div
function mod(x,y){ return x%y; } //Mod
function run_calculator(){
    var operand1 =
document.getElementById('operand1').value - 0;
    var operand2 =
document.getElementById('operand2').value - 0;
    var function_array = [add, sub, mul, div, mod];
    var z = function_array[1](operand1, operand2);
    document.getElementById("result").innerHTML=z;
}
</script>
```

Result:

Calculator:
Operands1: 101
Operands2: 110
☐ Add ☐ Sub ☐ Mul ☐ Div ☐ Mod
Result: xxxx
Compute

After click "Compute" button:

Calculator:
Operands1: 101
Operands2: 110
☐ Add ☒ Sub ☐ Mul ☐ Div ☐ Mod
Result: -9
Compute

Section 2 Outline

- Change the result of the web calculator
 - JavaScript object representing an HTML element
- Read operands user entered
 - JavaScript "onclick" event
- Perform the selected computation
 - JavaScript Array object
 - JavaScript for-loop

JavaScript Programming—for loop

- In Tom's blog, the calculator can only perform one kind of computation, so it still needs to traverse the five options to know the option selected by user. Therefore, Tom learns the knowledge about **for** loop:
- for loop:

```
for(statement1;expression1;statement2){  
    //some code  
}
```

- Similar to for loop in Go, the difference is that there is a pair of parenthesis after for in JS.

Recall the HTML code of the calculator:

Calculator:

<form>

Operands1: <input type="text">

Operands2: <input type="text">

<input type="radio" name="op">Add

<input type="radio" name="op">Sub

<input type="radio" name="op">Mul

<input type="radio" name="op">Div

<input type="radio" name="op">Mod

Result: xxxx

<input type="button" value="Compute">

</form>

Result:

Calculator:

Operands1:

Operands2:

☐ Add ☐ Sub ☐ Mul ☐ Div ☐ Mod

Result: xxxx

Traverse array by for loop

```
function run_calculater(){
  ...
  var function_array = [add, sub, mul, div, mod];
  var options = document.getElementsByName("op");
  var z;
  var is_found = false;
  for(var i=0; i<function_array.length && !is_found; i++){
    if(options[i].checked == true){
      z = function_array[i](operand1, operand2);
      is_found = true;
    }
  }
  document.getElementById('result').innerHTML = z;
}
```

Result:

Calculator:

Operands1:

Operands2:

☐ Add ☐ Sub ☒ Mul ☐ Div ☐ Mod

Result: xxxx

After clicking "Compute" button:

Calculator:

Operands1:

Operands2:

☐ Add ☒ Sub ☐ Mul ☐ Div ☐ Mod

Result: 156



中国科学院大学
University of Chinese Academy of Sciences

CS101

Section 3

In this section, we will make the following webpage

The growing diary of calculator and clock

Author: *Tom* * Time: 2020/02/26

Before

Before doing it, I am not sure I can make it.

List: Tom's Purpose

1. Learn CSS
2. Learn HTML
3. Learn JS
4. Make calculator and clock

During

During doing it, I find some material, tired but happy.

Table: Tom's hard journey

Event	Feeling
Learn CSS	One day awake
Learn HTML	One day awake again
Learn JavaScript	Missing Go

After

Finally, it's done. Although it's rough, it's also my own 'son'.

Calculator:

Operands1:

Operands2:

☐ Add ☐ Sub ☐ Mul ☐ Div ☐ Mod

Result: 101

Clock:

2021/10/31 1:14:52

Section 3 Outline

- Add a static clock
 - JavaScript Date object
- Make the clock dynamic
 - JavaScript Animation
- Debug
 - console.log function
- Compare JS with Go

JavaScript Programming—Date object

- After completing the calculator, Tom starts to make a web-clock, and he wants to show year, month and day in the clock, so he learns Date object in JS:
- **Create** a Date object
 - `var date = new Date();` // Date object in variable date
 - `var date = new Date;` // The parentheses can be omitted
- **Use** a Date object
 - `date.getDate();` // return which day in the month (start with 1)
 - `date.getDay();` // return which day in the week (start with 1, Sunday is 0)
 - `date.getFullYear();` // return year
 - `date.getMonth();` // return month (start with 0)
 - `date.getHours();` // return which hour in the day
 - `date.getMinutes();` // return which minute in the hour
 - `date.getSeconds();` // return which second in the minute

Add a static clock

```
<div id="after">
```

```
.....
```

Clock:

```
<p id="clock"></p>
```

```
</div>
```

```
<script>
```

```
.....
```

```
var date = new Date;
```

```
var year = date.getFullYear();
```

```
var month = date.getMonth() + 1;
```

```
var day = date.getDate();
```

```
var hour = date.getHours();
```

```
var min = date.getMinutes();
```

```
var second = date.getSeconds();
```

```
var time = year + "/" + month + "/" + day + " " + hour + ": " + min + ": " + second;
```

```
var clock = document.getElementById("clock");
```

```
clock.innerHTML = time;
```

```
</script>
```

change the content between tags

whose id attribute is "clock"

get time: year, month,
day, hour, minute, second

when the type of one of the operands is string
, the + operator means string concatenation
(number will be firstly converted to string)

Result:

Clock:

2021/4/2 11:40:29

Section 3 Outline

- Add a static clock
 - JavaScript Date object
- Make the clock dynamic
 - JavaScript animation
- Debug
 - console.log function
- Compare JS with Go

JavaScript animation

- Tom makes a clock but the clock only shows a time, not a real dynamic clock. To make the clock run, Tom learns the following function:
- `setTimeout` function
 - Usage: `setTimeout(a function name, time(in ms), parameter1, parameter2,...);`
 - Meaning: Tell the browser that the function specified by the first parameter will be executed after the time specified by the second parameter (parameters start from the third one will be the parameters of the first parameter)
 - Only Tell, not execute. The following code can execute after telling.
- Example:
 - `var a = 0;`
 - `setTimeout(console.log,1000,a);` //After 1000ms, execute `console.log(a);`

- Put the code showing clock in a function that will be called every 1000ms

call the function only once

```
run_clock();  
function run_clock(){  
    ...(code showing clock)  
    setTimeout(run_clock, 1000);  
}
```

Every time the function comes to an end, it tells the browser that I will be called after 1000ms.

Result:

Clock:
2021/10/31 0:22:33

Clock:
2021/10/31 0:22:46

Clock:
2021/10/31 0:22:57

JavaScript animation

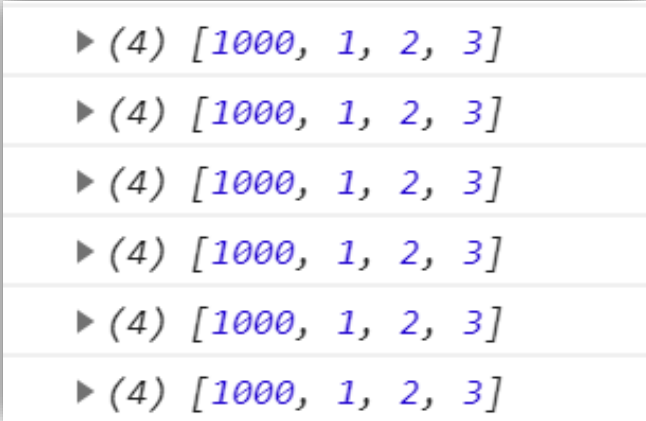
- Till now, Tom has completed a web clock, but in fact there is usually a fault when using setTimeout function
- Another example:

```
var ar=[0,1,2,3];  
for(var i=0;i<1000;i++){  
    setTimeout(console.log, 1000*i, ar);  
    ar[0]++;  
}
```

- What will be printed on console?
 - Does it sequentially output [0,1,2,3],[1,1,2,3],...,[999,1,2,3]?

JavaScript animation

- Console print:



```
▶ (4) [1000, 1, 2, 3]
▶ (4) [1000, 1, 2, 3]
▶ (4) [1000, 1, 2, 3]
▶ (4) [1000, 1, 2, 3]
▶ (4) [1000, 1, 2, 3]
▶ (4) [1000, 1, 2, 3]
```

- When an object is the parameter of a function, the address of the object is actually passed to the function, not a copy of the object.
- `setTimeout` tells the address of `ar` object to browser, and **there is only one `ar`**, so after telling browser 1000 times, `ar` has become `[1000,1,2,3]`.
- Although the time specified by `setTimeout` has passed, the function (first parameter of `setTimeout`) will be executed after the normal code comes to an end.
- So `console.log(ar)` prints `[1000,1,2,3]` only.

JavaScript animation

- Other examples using setTimeout

- pass number
 - pass the copy of number, not address

```
> x=2; setTimeout(console.log, 1000, x); x=3; x=4;  
◀ 4 // the return value of x=4;  
2 // what console.log prints
```

- After passing the object, we try to change the value storing the object

- Note that the address of the object is passed
 - Actually 4 objects is defined and assigned to x successively
 - the third parameter in each calling of setTimeout is different objects

```
> var x=[2,3,4];setTimeout(console.log,1000,x);x=[3,3,4];setTimeout(console.log,1000,x);x=[4,3,4];setTimeout(console.log,1000,x);x=[5,3,4];setTimeout(console.log,1000,x);  
◀ 7 // the return value of the last setTimeout  
▶ (3) [2, 3, 4] // what first console.log prints  
▶ (3) [3, 3, 4] // what second console.log prints  
▶ (3) [4, 3, 4] // what third console.log prints  
▶ (3) [5, 3, 4] // what fourth console.log prints
```

- After passing the object, we try to change the attribute of the object

- Note that the address of the object is passed
 - only one object is defined

```
> var x=[2,3,4]; setTimeout(console.log, 1000, x); x[0]++; x[0]++;  
◀ 3 // the return value of x[0]++;  
▶ (3) [4, 3, 4] // what console.log prints
```

JavaScript animation

● Other examples using setTimeout

● for loop

- Tell the browser that the increase function will be executed after 1000ms
- print ar
 - ar is not changed because increase has not executed

● increase function

- execute after 1000ms
- add ar[0] by 1
 - the ar[0] printed in this function is increasing

what console.log in
for-loop prints

what console.log
in increase
function prints

```
> var ar = [0,1,2,3];
  for(var i = 0; i < 5; i ++){
    setTimeout(increase, 1000, ar);
    console.log(ar);
  }
  function increase(ar){
    ar[0]++;
    console.log(ar);
  }

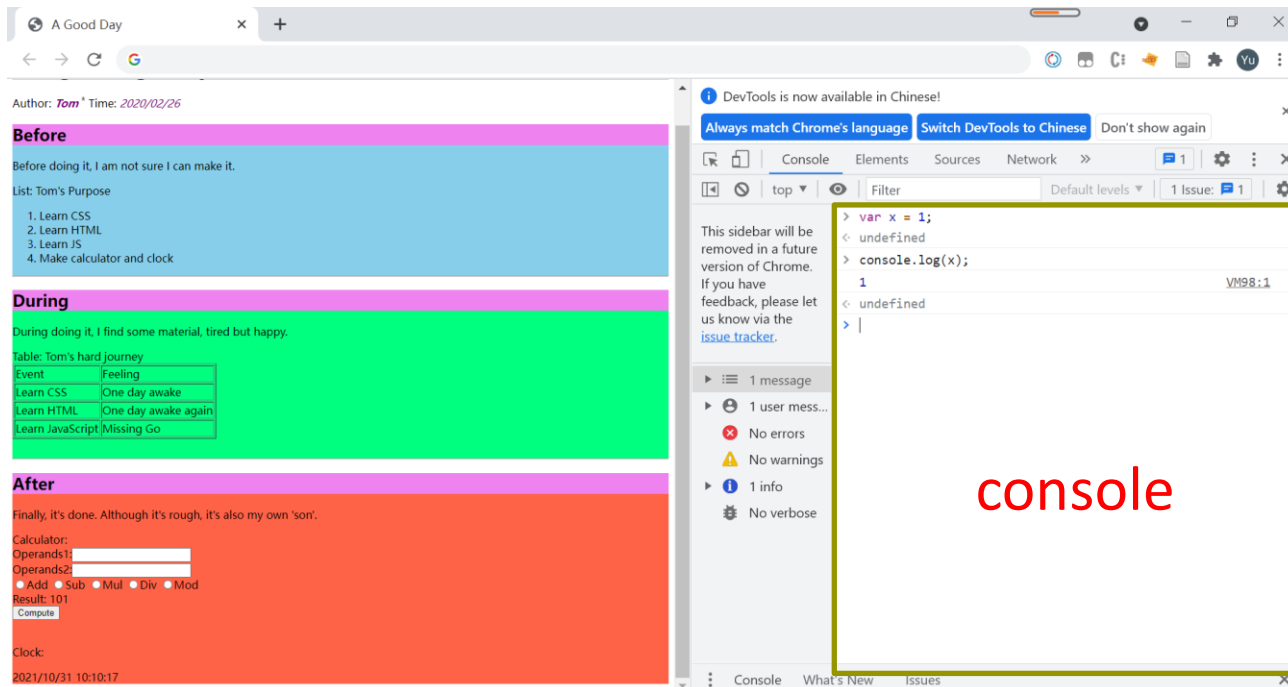
▶ (4) [0, 1, 2, 3]
▶ (4) [0, 1, 2, 3]
▶ (4) [0, 1, 2, 3]
▶ (4) [0, 1, 2, 3]
▶ (4) [0, 1, 2, 3]
< undefined
▶ (4) [1, 1, 2, 3]
▶ (4) [2, 1, 2, 3]
▶ (4) [3, 1, 2, 3]
▶ (4) [4, 1, 2, 3]
▶ (4) [5, 1, 2, 3]
> |
```

Section 3 Outline

- Add a static clock
 - JavaScript Date object
- Make the clock dynamic
 - JavaScript Animation
- Debug
 - `console.log` function
- Compare JS with Go

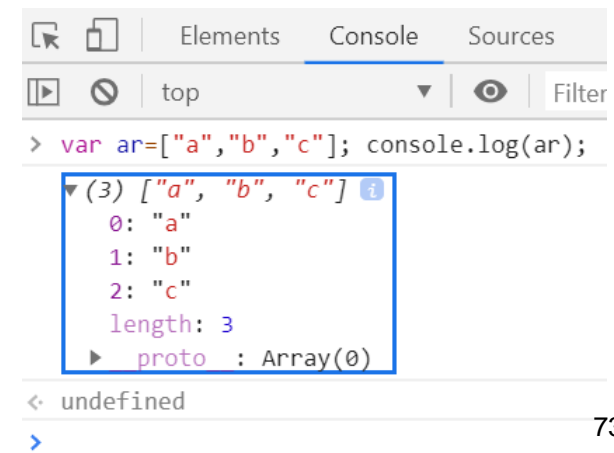
Console

- Each browser has a **console**, which is similar to terminal.
 - Console is the StdIn, StdOut and StdErr when you are debugging
 - You can execute JS statements directly in console without an html file
- Open a browser and press **F12**. The console can be opened.



Debug: console.log function

- Thanks to the function, Tom can tackle all the bugs in his webpage:
- console.log can print an object
 - can print the information of an object (methods, properties)
 - the returned value of many JS build-in functions is an object
 - when you are not sure the returned value of a function, print it with console.log.
- The console can execute the JS statements user enters.
- Type the following code on console, press Enter, and see what the console outputs:
 - `var ar = ["a", "b", "c"];`
 - `console.log(ar);`
 - The picture uses Chrome



Scene using console.log

- See the return value of a function

```
var options = document.getElementsByName("op");  
console.log(options);
```

```
▼ NodeList(5) [input, input, input, input, input] ⓘ  
  ▶ 0: input  
  ▶ 1: input  
  ▶ 2: input  
  ▶ 3: input  
  ▶ 4: input  
    length: 5  
  ▶ proto : NodeList
```

- Print intermediate variables

```
var operand1 = document.getElementById("operand1");  
var x = operand1.value - 0;  
var operand2 = document.getElementById("operand2");  
var y = operand2.value - 0;  
console.log("x=%d, y=%d", x, y); //can use %d (round  
                                down), %f, %s and %o(object)
```

Operands1:

Operands2:

x=1, y=2

Section 3 Outline

- Add a static clock
 - JavaScript Date object
- Make the clock dynamic
 - JavaScript Animation
- Debug
 - console.log function
- Compare JS with Go

Go vs JS

- Key Point

- Go is used to do “local” programming
 - Do I/O operation directly
 - manipulate file on local hard-disk
 - Read input from terminal directly
 - Print output to terminal directly
 - Access resources locally
- JS is used to do web programming
 - Do I/O operation through web browser
 - output to console that is in web browser
 - manipulate webpage that is displayed in web browser
 - Access resources of WWW

Go vs JS

	Go	JavaScript
Variable	Type is needed, and once declares, the type cannot change, e.g. <code>var a int = 1</code>	No type, and one variable can be assigned values of different types successively, e.g. <code>var a=1; a = "JS";</code>
Datatype	Has array, slice, char type	JS does not have slice; JS use object to implement array; JS does not have char type, only string type
Function	The types of parameters and return value is needed and a function can have multiple return values	No need of types of parameters nor return values. A function can have only one return value. Function is an object, can form array.
Arithmetic Operator	$5/2=2$ $5.0/2=2.5$ $3.2 \% 0.7$ is an compiling error	$5/2=2.5$ $3.2 \% 0.7 \approx 0.4$ (lose accuracy)
Comparison Operator	An Error occurs when compare values of different types.	When compare by <code>==(!=)</code> , operands of different types will be converted to the same type automatically before the comparison. When compare by <code>===(!==)</code> , the result of operands of different types is false(true)
Right shift <code>>></code>	<code>A>>B</code> When A is unsigned, the high bits will be made up by 0 When A is signed, the high bits will be made up by the sign-bit	<code>A>>B</code> : the high bits will be made up by 0 <code>A>>>B</code> : the high bits will be made up by the sign-bit

Go vs JS

	Go	JavaScript
Control flow	No parenthesis needed: if x<10 { }	Parenthesis is necessary: if (x<10){ }
object	N/A	Array, Date, Object representing HTML element
Print function	fmt.Println, fmt.Printf	console.log
Length of array	len(array)	array.length
Language type	Compiled language: The source code should be compiled to executable file which can be executed later.	Interpreted language: The interpreter inside browser will interpret and execute each JS statement
statement	No ; needed	; is optional

Quicksort Go vs JS

- Go

- When create a slice, the length of the underlying array is needed
- Go can pass slice that has length, so the index bound of the unsorted data in the array is not shown in the parameter list
- fmt.Printf() can output to terminal directly

```
func main() {  
    rand.Seed (time.Now().UnixNano())  
    n := 1*1024*1024  
    var da = make([]int,n)  
    for i:=0; i<n; i++ {  
        da[i] = rand.Int()  
    }  
    quicksort (da[0: n])  
    fmt.Printf(" %d\n %d\n  
%d\n",da[0],da[1],n-1,da[n-1])  
}
```

- JS

- When create an array, no length is needed
- The different way of generating random numbers
- JS has no slice, so the index bound should be passed.
- JS can only output to console which is embedded in the browser

```
<script>  
    var ar = new Array();  
    var n = 1*1024*1024;  
    for(var i=0; i<n; i++){  
        ar[i] = Math.floor(Math.random()*1000000);  
    }  
    quicksort(ar, 0, ar.length-1);  
    console.log(" %d\n %d\n %d\n", ar[0], ar[1],  
ar[ar.length-1]);  
    .....
```

Math.random() returns a random number in [0,1)

Quicksort Go vs JS

- Go

- it does not need "()"
- array (the variable) is a slice representing a fragment of the underlying array (the data type) and the length of array (the variable) can be accessed by `len(array)`
- partition only needs the current array (the data type) as its parameters (not the whole array)

```
func quicksort(array []int) {  
    if len(array) < 2 {  
        return  
    }  
    left_array, right_array :=  
partition(array)  
    quicksort(left_array)  
    quicksort(right_array)  
}
```

- JS

- it needs "()"
- Because the index bound of ar is passed as parameters, the length of the unsorted data should be computed by the index bound
- partition also needs index bound as parameters

```
function quicksort(ar, left, right){  
    if (right - left + 1 < 2) {  
        return;  
    }  
    var pivotal_index = partition(ar, left, right);  
    quicksort(ar, left, pivotal_index-1);  
    quicksort(ar, pivotal_index+1, right);  
}
```


Quicksort Go vs JS

● Go

Random range is [0, len(array)]

```
func partition(array []int) ([]int, []int) {
    pivotal_index := rand.Intn(len(array))
    pivotal_value := array[pivotal_index]
    next := 0
    array[pivotal_index], array[len(array)-1] =
    array[len(array)-1], array[pivotal_index]

    for i := 0; i < len(array); i++ {
        if (array[i] < pivotal_value) {
            array[next], array[i] =
            array[i], array[next]
            next++
        }
    }
    array[next], array[len(array)-1] =
    array[len(array)-1], array[next]
    return array[0:next], array[next+1: len(array)]
}
```

assign to 2 variables
at the same time

return two slice

● JS

Random range is [left, right]

```
function partition(ar, left, right){
    var len = right - left + 1;
    var pivotal_index =
    Math.floor(Math.random()*len) + left;
    var pivotal_value = ar[pivotal_index];
    var next=left;
    // pivotal to last
    ar[pivotal_index] = ar[right];
    ar[right] = pivotal_value;
    // smaller than pivotal : left
    for(var i=left; i<=right; i++){
        if(ar[i] < pivotal_value) {
            // swap
            var temp = ar[next];
            ar[next] = ar[i];
            ar[i] = temp;
            next++;
        }
    }
    // swap pivotal back
    ar[right] = ar[next];
    ar[next] = pivotal_value;
    return next; return index
}
```

When swap two
values, one should
be stores in
another variable



中国科学院大学
University of Chinese Academy of Sciences

CS101

Additional Material

Additional Material Outline

- JavaScript variable & function
- JavaScript operator & expression
- JavaScript control flow
 - while loop
 - if-else statement
 - switch-case statement

JavaScript Basics—data types

Data type	Description	Example
Bool	Represent true or false; only 2 values: true, false	true, false
Number	64bit length, represent integers and floating point numbers, similar to float64 in Golang	-1, 0, 1, 0.1, -0.1
String	Represent a string	"abc", "" (empty string)
Null type	Null belongs to a special type specific to null Null represents that there is no value	null
Undefined type	Undefined belongs to a special type specific to null Undefined represents that a value is lacking: (1) A variable that is not assigned (2) A function needs arguments, but no parameters are provided (3) A function with no return value	undefined

JavaScript Programming—variable

- `<script>` part:

`var x,y,z;` variable declaration(no type)

`var s = "JavaScript";` declaration & assignment

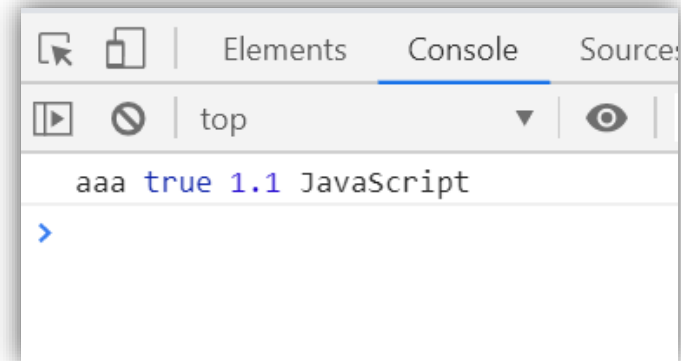
`x = 0;` variable assignment

`x = "aaa";` variable assignment
(assign different types of values to the same variable)

`y= true;` variable assignment

`z = 1.1;` variable assignment

`console.log(x,y,z,s);` → print multiple variables' values, and these variables are separated by comma



JavaScript Programming—function

- Function definition

JS: **function** add(x,y){
 return x + y;
 }

Go: **func** add(x,y **int**) **int**{
 return x + y
 }



only one return value while Go can return more than one.

- Function invocation

```
var a=1,b=1;
```

```
var z = add(a,b); // z=2;
```

Additional Material Outline

- JavaScript variable & function
- JavaScript operator & expression
- JavaScript control flow
 - while loop
 - if-else statement
 - switch-case statement

JavaScript operator & expression

- Operators that are the same with that in Golang
 - `+, -, *, ++, --`
- Operators that are different from that in Golang
 - `/` (division)
 - In JS: $3/2=1.5$, In Go: $3/2=1$ (round down)
 - If you want to round down in JS, you need to use `Math.floor(3/2)`
 - `%` (remainder)
 - In JS, operands of `%` can be real number while Go does not allow it.
 - In JS, $3.2 \% 0.7 \approx 0.4$ (lose precision) while Go does not allow $3.2\%0.7$

JavaScript operator & expression

- JS has not only arithmetic operators, but also almost all the operators of Go:
- Assignment Operators:
 - `=, +=, -=, *=, /=, %=`
 - `x op= y` equals to `x=x op y`, for example, `x+=y` equals to `x=x+y`
 - `op` can be `+, -, *, /, %`
- Comparison Operators
 - The result of comparison is true or false
 - Same as Go: `>, >=, <, <=`
 - Differ from Go: `==, !=, ===, !==`
 - when use `==` or `!=`, if the types of two operands is different, the two operands will be transformed to the same type before they are compared.
 - when use `===` (`!==`), if the types of two operands is different, the result is false(true)

```
> console.log(false==0)
```

```
true
```

```
< undefined
```

```
> console.log(false===0)
```

```
false
```

```
< undefined
```

JavaScript operator & expression

- Logical Operators:
 - Same as Go: `&&` (And), `||` (Or), `!` (Not)
 - Often used in conditional expression
 - Eg: `if(x<10 && x>0) { ... }`
- Bitwise Operators:
 - Same as Go: `&` (bitwise and), `|` (bitwise or), `^` (bitwise nor), `~` (bitwise not), `<<` (left shift)
 - Differ from Go: `>>` (Signed right shift), `>>>` (Zero fill right shift)
 - The operands of a bitwise operator will firstly be transformed to 32-bit integers.

JavaScript operator & expression

- Conditional Operator (has three operands)
 - condition ? the value when condition is true : the value when condition is false
 - For example: `var y = x > 0 ? x : -x; // y = the absolute value of x`
- String Concatenation Operator
 - + (the same as addition)
 - when the type of one or two of the two operands of + is string, the operand whose type is not string will firstly be transformed to string type before they are concatenated.
 - `var x = "Java" + "Script"; // x = "JavaScript"`
 - `var x = "0" + 1; // x = "01" (1 is transformed to string type)`

Additional Material Outline

- JavaScript variable & function
- JavaScript operator & expression
- JavaScript control flow
 - while loop
 - if-else statement
 - switch-case statement

JavaScript control flow—while loop

- while loop:

```
while(expression){  
    // some code  
}
```

- In each iteration: if the expression is true, the loop body is executed, otherwise the loop is terminated.

JavaScript control flow —if-else statement

- if-else statement:
 - Similar to Go, but the expression has **curly braces**

```
if(expression 1){  
    //some code  
}else if (expression 2){  
    //some code  
}else{  
    //some code  
}
```

The whole statement can have zero or more else-branches

JavaScript control flow—switch-case statement

- switch-case statement:
 - an alternative of if-else statement

```
switch(expression){  
  case value1: // some code, executed when expression equals value1  
    break; // Do not forget break, otherwise all statements following the matched case will execute  
           // unconditionally until another break is encountered or the end of the switch-case  
           // statement is reached  
  case value2: // some code, executed when expression equals value2 but not value1  
    break;  
  default: // some code, executed when expression does not equals value2 and value1  
           //default can be omitted  
}
```