

# **TypeScript**

## **In 8 Hours**



**For Beginners**  
**Learn Coding Fast!**

*Ray Yao*

**Type**

**In 8**

# TypeScript

## In 8 Hours

**For Beginners**  
**Learn Coding Fast**

**Ray Yao**

**Copyright © 2015 by Ray Yao**

**All Rights Reserved**

Neither part of this book nor whole of this book may be reproduced or transmitted in any form or by any means electronic, photographic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without prior written permission from the author. All rights reserved!

Ray Yao

**About the Author**

Certified PHP engineer by Zend, USA

Certified JAVA programmer by Sun, USA

Certified SCWCD developer by Oracle, USA

Certified A+ professional by CompTIA, USA

Certified ASP. NET expert by Microsoft, USA

Certified MCP professional by Microsoft, USA

Certified TECHNOLOGY specialist by Microsoft, USA

Certified NETWORK+ professional by CompTIA, USA

[www.amazon.com/author/ray-yao](http://www.amazon.com/author/ray-yao)

## **Recommended Books on Amazon**

[Advanced C++ in 8 Hours](#)

[Advanced Java in 8 Hours](#)

[AngularJs Programming](#)

[C# Programming](#)

[C# Interview Q&A](#)

[C++ Programming](#)

[C++ Interview Q&A](#)

[Django in 8 Hours](#)

[Go in 8 Hours](#)

[Html Css Programming](#)

[Html Css Interview Q&A](#)

[Java Programming](#)

[Java Interview Q&A](#)

[JavaScript Programming](#)

[JavaScript Interview Q&A](#)

[JQuery Programming](#)

[JQuery Interview Q&A](#)

[Kotlin in 8 Hours](#)

[Linux Command Line](#)

[Linux Interview Q&A](#)

[MySQL DataBase](#)

[Node . Js in 8 Hours](#)

[Perl in 8 Hours](#)

[Php MySQL Programming](#)

[Php Interview Q&A](#)

[PowerShell in 8 Hours](#)

[Python Programming](#)

[Python Interview Q&A](#)

[R Programming](#)

[Ruby Programming](#)

[Rust in 8 Hours](#)

[Scala in 8 Hours](#)

[Shell Scripting in 8 Hours](#)

[Swift in 8 Hours](#)

[Visual Basic Programming](#)

[Visual Basic Interview Q&A](#)

[Xml Json in 8 Hours](#)

## **Preface**

This book covers all essential TypeScript language knowledge. You can learn complete primary skills of TypeScript programming fast and easily. The book includes more than 60 practical examples for beginners and includes tests & answers for the college exam, the engineer certification exam, and the job interview exam.

## **Note:**

This book is only for TypeScript beginners, it is not suitable for experienced programmers.

## **Source Code for Download**

This book provides source code for download; you can download the source code for better study, or copy the source code to your favorite editor to test the programs.

## **Source code download link:**

<https://forms.aweber.com/form/83/138213183.htm>

## **Table of Contents**

### Hour 1

[What is TypeScript?](#)

[Why Using TypeScript?](#)

[Install TypeScript](#)

[Test Node . Js](#)

[Compile & Run](#)

[Hello World Program](#)

[TypeScript Comment](#)

### Hour 2

[TypeScript Reserved Words](#)

[TypeScript Datatype](#)

[Variable](#)

[Define a Variable](#)

[Any Type](#)

[Never Type](#)

[Type Assertion](#)

[Number Properties](#)

[Number Properties Example](#)

[Number Object](#)

### Hour 3

[Arithmetic Operators](#)

[Comparison Operators](#)

[Logical Operators](#)

[Assignment Operators](#)

[Ternary Operator](#)

[Typeof Operator](#)



[“+” and “-” Operators](#)

[If Statement](#)

[If-else Statement](#)

[Switch Statement](#)

[Let & Const](#)

## [Hour 4](#)

[For Loop](#)

[For in Loop](#)

[For of Loop](#)

[While Loop](#)

[Do-While Loop](#)

[Break Statement](#)

[Continue Statement](#)

[Function](#)

[Function with arguments](#)

[Return Values](#)

[Anonymous Function](#)

[Lambda Function](#)

## [Hour 5](#)

[String](#)

[String Length](#)

[Convert to String](#)

[Find a Character](#)

[Connect Two Strings](#)

[Locate a SubString.\(1\)](#)

[Locate a SubString.\(2\)](#)

[Replace a String](#)

[Get a Substring](#)

[Get Unicode](#)

[Case Conversion](#)

[String Functions](#)

## [Hour 6](#)

[Create an Array](#)

[Array Object](#)

[Iterating Over an Array](#)

[Array Assignment](#)

[Connect Two Arrays](#)

[Array Length](#)

[Unshift\(\) Function](#)

[Shift\(\) Function](#)

[Push\(\) Function](#)

[Pop\(\) Function](#)

[Sort Array](#)

[Array Functions](#)

## [Hour 7](#)

[Tuple](#)

[Access Tuple Element](#)

[Update a Tuple](#)

[Tuple Assign Values](#)

[Union Types \(1\)](#)

[Union Types \(2\)](#)

[Global & Local](#)

[Class Definition](#)

[Object Declaration](#)

[Class & Object](#)

[Extends](#)

## Hour 8

[Multiple Extending](#)

[Overriding](#)

[Super Keyword](#)

[Static Variable & Method](#)

[Private Modifier \(1\)](#)

[Private Modifier \(2\)](#)

[Protected Modifier \(1\)](#)

[Protected Modifier \(2\)](#)

[Interface \(1\)](#)

[Interface \(2\)](#)

[Object \(1\)](#)

[Object \(2\)](#)

## TypeScript — Questions & Answers

[Questions](#)

[Answers](#)

## Source Code Download

# Hour 1

# What is TypeScript?

TypeScript is an open source cross-platform programming language developed by Microsoft; it's an enhanced version of JavaScript. TypeScript is compiled to JavaScript code at runtime.

TypeScript extends the syntax of JavaScript, so any existing JavaScript program can run in the TypeScript environment. TypeScript is designed for the development of large applications and can be compiled to JavaScript easily.

TypeScript is written by Anders Hejlsberg, the chief architect of C#.

TypeScript is a superset of JavaScript; it mainly provides the type system and support for ES6, and open source on GitHub. TypeScript is an oriented object programming language with optional static type, which can be capable of developing large projects.

Therefore, we can say: TypeScript makes JavaScript become a Java.

# Why Using TypeScript?

01. TypeScript has more readability and maintainability for code.
02. TypeScript enhances the capabilities of the editor and IDE.
03. TypeScript has strong compatibility and it is very inclusive.
04. TypeScript can detect programming errors at compile time.
05. TypeScript can automatically make type inferences.
06. TypeScript types cover the ranges from simple to complex.
07. TypeScript can generate JS files even if it compiles errors.
08. TypeScript compatible third-party libraries perfectly
09. TypeScript extension name can be renamed to .ts from .js
10. TypeScript has an active community and external support.
11. TypeScript can create large applications like Angular, Vue...

# Install TypeScript

In order to install TypeScript, we need to install Node.js first.

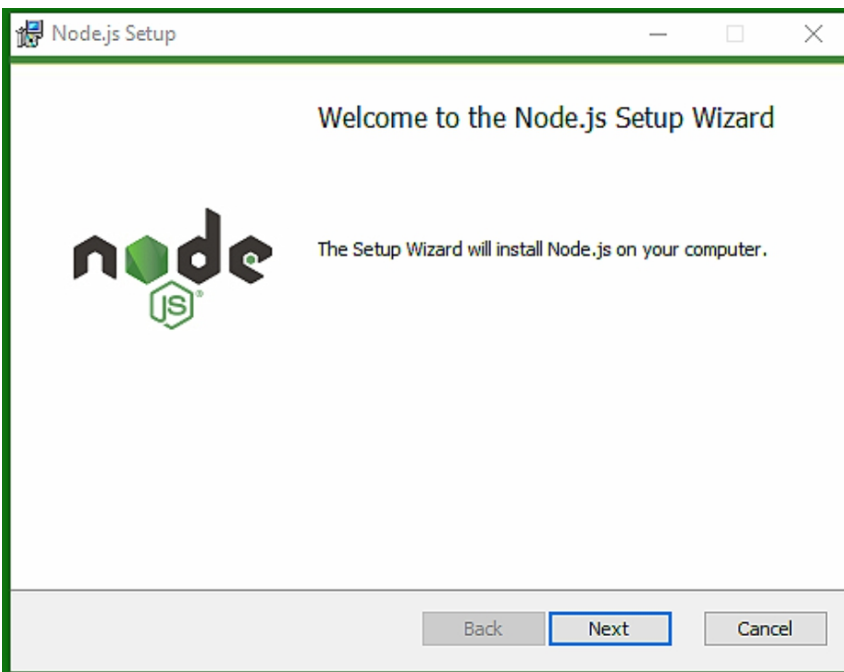
## Download Node.js

Installing node.js on Windows is convenient, and you only need to access node.js's official website

Download Link: <http://www.nodejs.org/>

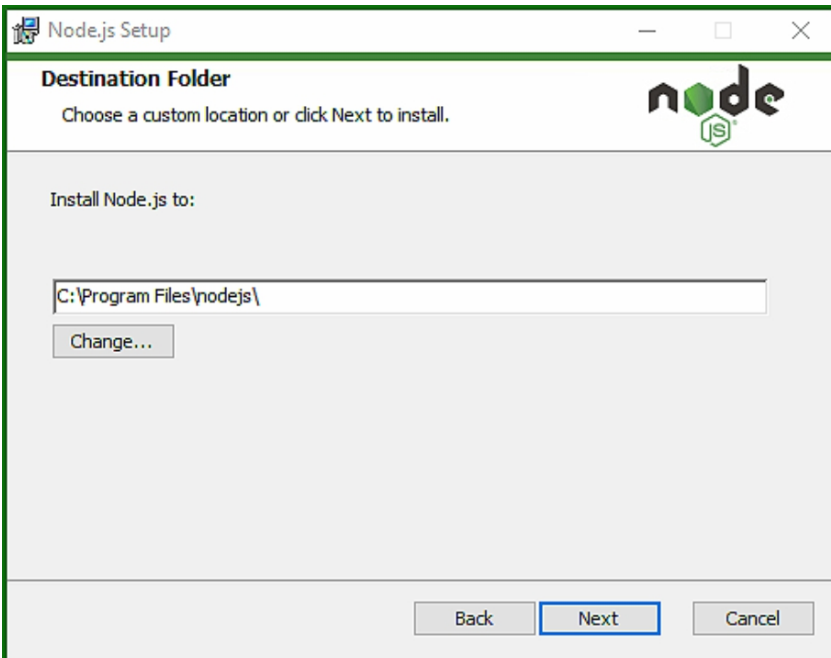
## Install Node.js

After the download is complete, double-click the installation package directly, just like any other software installation.....



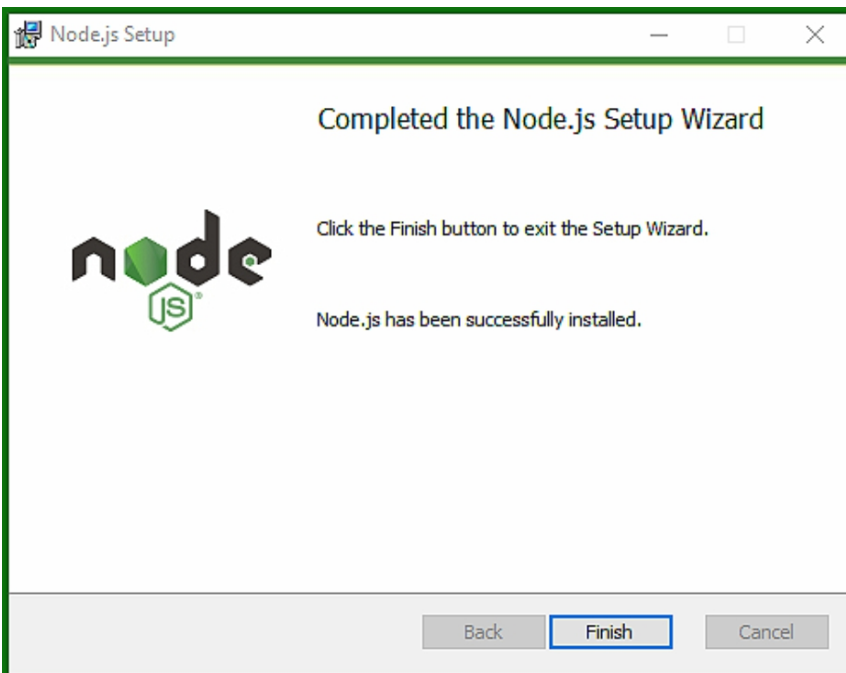
.....

Select a default installation folder.....



.....

Complete the installation.



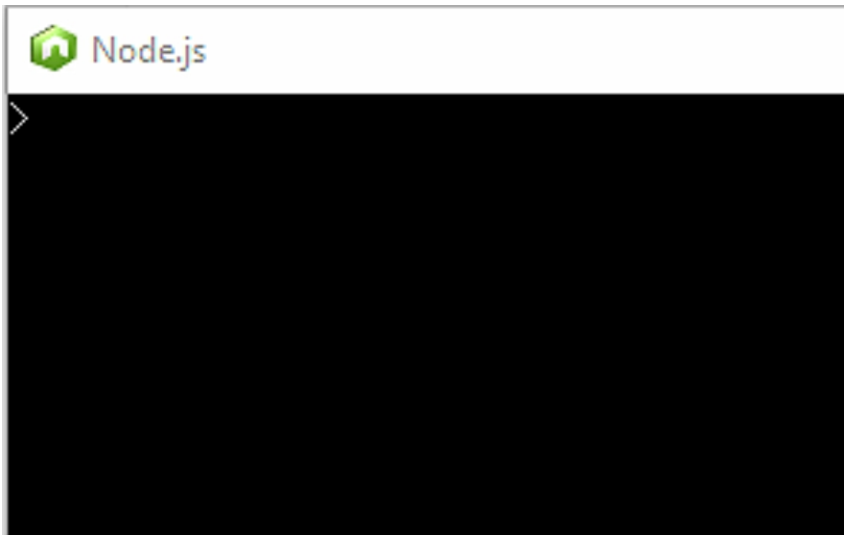
Click “Finish” button.



# Test Node . Js

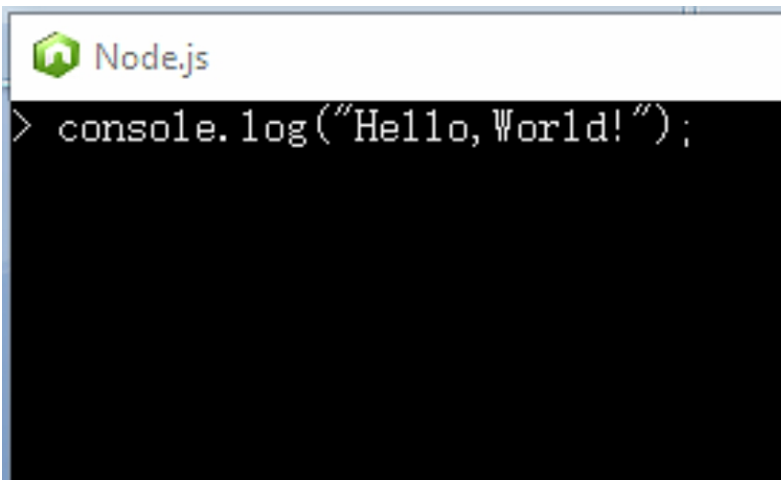
## Example

Click the Menu **Start > All Programs > Node.js Folder > Node.js** (green icon). You can see the Node. js interface.



Please input the following node. js code:

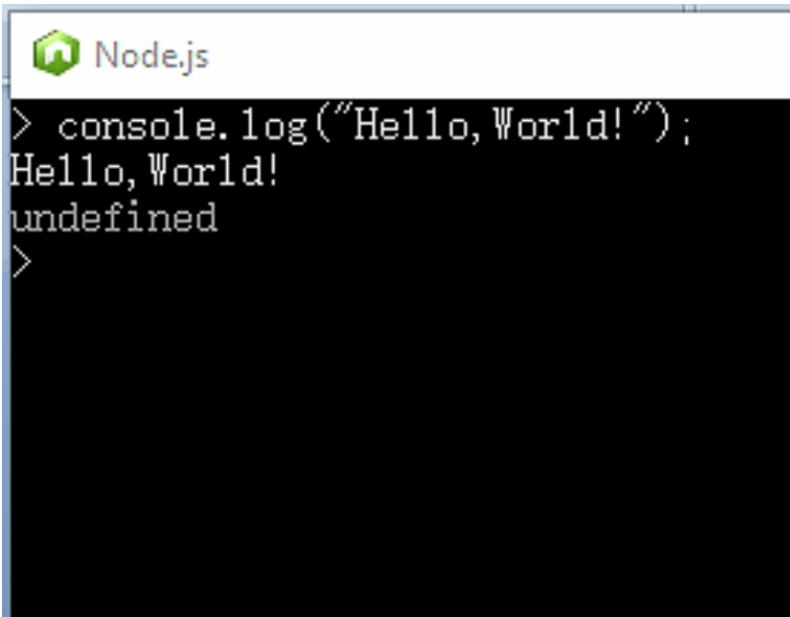
```
console.log("Hello, World! ");
```



Press "Enter" key.

### Output:

Hello World

A screenshot of a Node.js terminal window. The title bar at the top says "Node.js" with a green icon. The terminal has a black background with white text. It shows a prompt ">" followed by the command "console.log('Hello, World!');". Below the command, the output "Hello, World!" is displayed. The next line shows "undefined" and the prompt ">" again, indicating the end of the command execution.

```
> console.log("Hello, World!");  
Hello, World!  
undefined  
>
```

### Explanation:

Congratulation! Node. js has installed successfully!

“console. log(...)” is an output command, which is used to output some texts.

Each command of Node. js should be ended by a semicolon (; ).

For example:

The command “console. log( "C# in 8 Hours" ); ” will output the text “C# in 8 Hours”.

## Install TypeScript

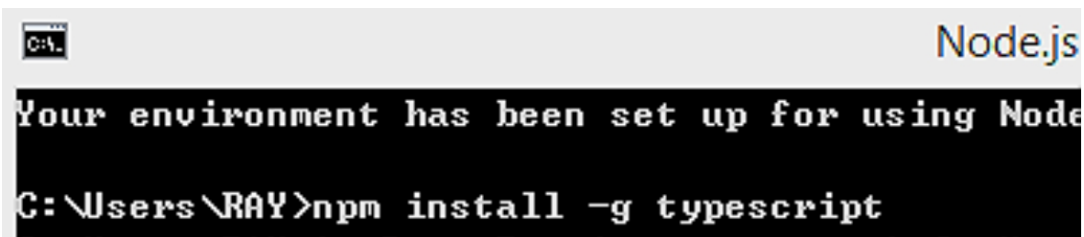
After installing Node.js, we begin to install TypeScript.

Click the Menu **Start > All Programs > Node.js Folder > Node.js Command Prompt**

Please input the following command:

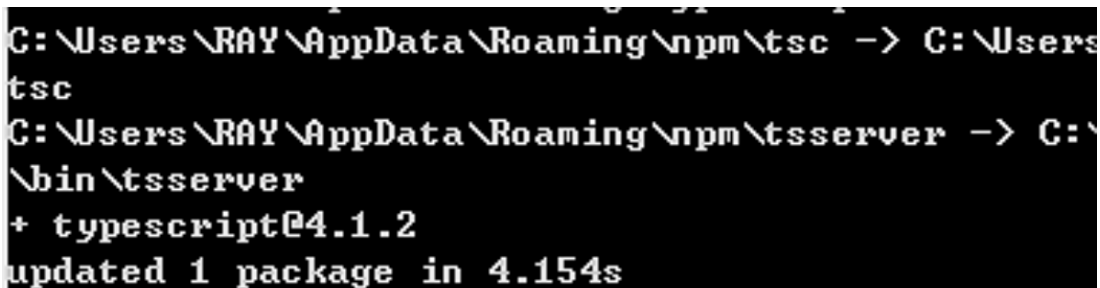
```
npm install -g typescript
```

As shown below:



```
C:\> npm install -g typescript
```

After installing TypeScript, you can see the information like this:



```
C:\Users\RAY\AppData\Roaming\npm\tsc -> C:\Users\RAY\AppData\Roaming\npm\tsc
C:\Users\RAY\AppData\Roaming\npm\tsserver -> C:\Users\RAY\AppData\Roaming\npm\tsserver
+ typescript@4.1.2
updated 1 package in 4.154s
```

The information indicates that TypeScript has been installed.

## Test TypeScript

To test TypeScript installation, please enter the following command:

```
tsc -v
```

This command is used to show the TypeScript version.

Then Typescript version is shown like this:

```
C:\Users\RAY>tsc -v  
Version 4.1.2
```

We can see the TypeScript version as 4. 1. 2.

Congratulation!

TypeScript has been installed successfully!

## Working Folder

To store and run Typescript file, we need to create a working folder in the path “C: \User\YourName\myTypeScript ”.

Now “**myTypeScript** ” is specified as the working folder.

# Compile & Run

1. A typescript program needs to be compiled to a javascript program first before running.

```
tsc myfile. ts
```

“tsc” is a compile command, which is used to compile a typescript file to a javascript file.

“ts” is an extension name of typescript file.

2. After the typescript file has compiled to a javascript file, we need use node. js command to run the file.

```
node myfile. js
```

“node” is a node. js command, which is used to run the javascript program.

“js” is an extension name of the javascript file.

## Note:

In this book, we use Notepad as a TypeScript editor.

You can select “Visual Studio Code” as a TypeScript editor, the download link is: [https : //code . visualstudio . com/](https://code.visualstudio.com/)

# Hello World Program

1.

Please open the Notepad, enter the following typescript code:

```
const hello : string = "Hello World! "  
console.log(hello)
```

Save this file as “hello. ts” in the working folder “myTypeScript”.

2.

Click the Menu **Start > All Programs > Node.js Folder > Node.js Command Prompt**

Enter the following command:

```
cd myTypeScript
```

This command changes the directory as “myTypeScript”.

```
C:\Users\RAY>cd myTypeScript
```

3.

Enter the following command:

```
tsc hello. ts
```

“tsc” is used to compile the typescript file to javascript file.

```
C:\Users\RAY>cd myTypeScript  
C:\Users\RAY\myTypeScript>tsc hello.ts
```

4.

After the typescript file “hello. ts” has been compiled to javascript file, we can use node command to run the javascript file.

Enter the following command:

```
node hello.js
```

```
C:\Users\RAY>cd myTypeScript  
C:\Users\RAY\myTypeScript>tsc hello.ts  
C:\Users\RAY\myTypeScript>node hello.js  
Hello World!
```

### **Output:**

Hello World!

### **Explanation:**

“tsc hello. ts” compile the file from typescript file to javascript file.

“node hello. js” run the javascript file.

“console. log()” is a typescript command to output contents.

# TypeScript Comment

```
// is used as a comment symbol for single line.  
/* */ are used as comment symbols for multi line.
```

The typescript compiler always ignores the comments.

## Example 1.1

```
const hello : string = "Hello World! "  
console.log(hello) // console.log() is an output command  
/* This is a hello-word program in typescript,  
in this program, we can know that the TypeScript  
Language is very excellent. */
```

## Output:

Hello World!

## Explanation:

“//” is used as a comment symbol for single line.

“/\* \*/” are used as comment symbols for multi line.



## Hour 2

# TypeScript Reserved Words

TypeScript Reserved Words are only used by itself, it cannot use as a variable name, string name, array name, function name...

break	as	catch	switch
case	if	throw	else
var	number	string	get
module	type	instanceof	typeof
public	private	enum	export
finally	for	while	void
null	super	this	new
in	return	true	false
any	extends	static	let
package	implements	interface	function
new	try	yield	const
continue	do		

# TypeScript Datatype

Datatype	Description
any	can be assigned any type value
number	double precision, 64-bit floating point values
string	a series characters enclosed by quotation marks
boolean	represents logical values : true and false
array	a variable with multiple values
tuple	an array with a known number & type of elements
enum	used to define a set of values
void	indicates that the method does not return a value
null	indicates that an object value is missing
undefined	indicates that an object value is missing
never	indicates a value that never appears

Note :

TypeScript and JavaScript have no integer types .

# Variable

A variable is a container to store changeable data value.

The rules of TypeScript variable naming are as follows:

1. Variable names can contain numbers and letters.

For example: var007, p2p

2. Variable names cannot begin with a number. The following variable names are invalid.

For example: 100scores, 007hero

3. Underline “\_” and Dollar symbol “\$” can be used in the variable name, other special symbol cannot be used in the variable name.

For example: \_var, \$var are valid. @var, &var are invalid

4. Space cannot be used in the variable name.

For example: “hello world” is an invalid variable name.

5. The variable name is case sensitive.

For example: jQuery and JQuery are two different variables.

# Define a Variable

We can use a keyword “var” to define a variable.

1.

The first syntax to define a variable is as follows:

```
var variableName : type = value;
```

## Example 2.1

```
var myVar001 : string = "Hello";
```

2.

The second syntax to define a variable is as follows:

```
var variableName = value;
```

## Example 2.2

```
var myVar02 = “World”;
```

### Example 2.3

```
var myBook: string = "Go in 8 Hours";  
var myScore: number = 100;  
var myStory: boolean = true;  
console.log("My book is: " + myBook)  
console.log("My exam score is: " + myScore)  
console.log("My story is: " + myStory)
```

### Output:

My book is: Go in 8 Hours

My exam score is: 100

My story is: true

### Explanation:

“myBook” is a variable, type is string, value is “Go in 8 Hours”.

“myScore” is a variable, type is number, value is 100.

“myStory” is a variable, type is boolean, value is true.

# Any Type

When the type of a variable may be different, we can use Any type to define a variable.

```
var variableName : any;
```

## Example 2.4

```
var myVariable: any ;    // myVariable is “any” type
myVariable = 100;        // myVariable is “number” type
console.log(myVariable);
myVariable = 'Scala in 8 Hours'; // myVariable is “string” type
console.log(myVariable);
myVariable = false;      // myVariable is “Boolean” type
console.log(myVariable);
```

## Output:

```
100
Scala in 8 Hours
false
```

## Explanation:

“var myVariable: **any** ;” declares the type of myVariable as “**any**”.

# Never Type

A variable with never type must not have any value.

```
var variableName : never;
```

## Example 2.5

```
var x: never ;    // never type  
var y: number = 100;  
x = y;    // error !  
console.log(x);
```

## Output:

Error: Type 'number' is not assignable to type 'never'.

## Explanation:

The x is Never type, so x cannot be assigned any value. But a Never type can assign value to another Never type:

```
var x: never ;  
var y: never ;  
x = y;    // correct !  
console.log(x);    // correct ! output: undefined
```



# Type Assertion

Type assertions can be used to specify the type of a value, that is, to change the variable type to another type.

`<any>variable`

## Example 2.6

```
var str = "100"    // string type
var num: number = <any> str    // change to number type
console.log(num)
```

## Output:

100

## Explanation:

“<any> str” can be changed its type to any other type. So in this example, the str can be changed to number type.

The output 100 is a number type.

# Number Properties

A number object includes following properties:

<b>NEGATIVE_INFINITY</b>
Returns this value when overflow, its value less than min_value .
<b>POSITIVE_INFINITY</b>
Returns this value when overflow, its value greater than min_value .
<b>MAX_VALUE</b>
The maximum number, its value is $1.79e+308$
<b>MIN_VALUE</b>
The minimum number, its value is $5e-324$ .
<b>NaN</b>
Not a Number . A invalid number .
<b>PROTOTYPE</b>
Used to add properties and methods to an object .
<b>CONSTRUCTOR</b>
Returns a reference of the Number function that created this object

# Number Properties Example

## Example 2.7

```
console . log("Minus infinity : " + Number . NEGATIVE_INFINITY );  
console . log("Positive infinity : " + Number . POSITIVE_INFINITY );  
console . log("Maximum : " + Number . MAX_VALUE );  
console . log("Minimum : " + Number . MIN_VALUE );
```

## Output:

Minus infinity: -Infinity

Positive infinity: Infinity

Maximum: 1.7976931348623157e+308

Minimum: 5e-324

## Explanation:

The example above shows the properties of some number objects.

# Number Object

The syntax to create a number object is as follows:

```
var obj = new Number(value);
```

A number object can reference various number methods.

## Example 2.8

```
var num = new Number(100.1234);  
console.log(num.toFixed(3));  
// Output a number that has three decimals  
console.log(num.toPrecision(4));  
// Output a number with length 4  
console.log(num.toString());  
// Output a decimal number  
console.log(num.toString(2));  
// Output a binary number  
console.log(num.toString(16));  
// Output a hexadecimal number
```

**Output:**

100.123

100.1

100.1234

1100100.00011111100101110010010001110100010100111001

64.1f972474539

**Explanation:**

“var num = new Number(100.1234);” creates a number object.

“toFixed(3)” outputs a number that has three decimals.

“toPrecision(4)” outputs a number with length 4.

“toString()” outputs a decimal number.

“toString(2)” outputs a binary number.

“toString(16)” outputs a hexadecimal number.

## Hour 3

# Arithmetic Operators

Operators	Running
+	add or connect strings
-	subtract
*	multiply
/	divide
%	get remainder
++ or --	increase 1 or decrease 1

% modulus operator divides the first operand by the second operand, returns the remainder. e. g.  $4\%2$ , the remainder is 0.

## Example 3.1

```
var x: number = 20
```

```
var y: number = 2
```

```
var result: number = 0
```

```
result = x + y; console.log("20 + 2 = "+result);    // 22
```

```
result = x / y; console.log("20 / 2 = "+result);    // 10
```

```
result = x % y; console.log("20 % 2 = "+result);    // 0
```

```
result = ++ x; console.log("++x = "+result);        // 21
```

# Comparison Operators

Operators	Running	Result
>	greater than	true or false
<	less than	true or false
>=	greater than or equal	true or false
<=	less than or equal	true or false
==	equal	true or false
!=	not equal	true or false

## Example 3.2

```
var a=100; var b=200;  
var result = (a > b); console.log ( result );  
var result = (a == b); console.log ( result );  
var result = (a != b); console.log ( result );
```

## Output:

false

false

true



# Logical Operators

Operators	Equivalent	Result
&&	and	true or false
	or	true or false
!	not	true or false

## Example 3.3

```
var x=true; var y=false;  
var a=x && y; console.log ( a ); // false  
var b=x || y; console.log ( b ); // true  
var c=! x; console.log ( c ); // false
```

## Explanation:

true && true; returns true;	true && false; returns false;	false && false; returns false;
true    true; returns true;	true    false; returns true;	false    false; return false;
! false; returns true;	! true; returns false;	

# Assignment Operators

Operators	Examples	Equivalent
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

## Example 3.4

var x: number = 20; var y: number = 2

x += y; console.log(x)    // x=x+y returns 22

var x: number = 20; var y: number = 2

x /= y; console.log(x)    // x=x/y returns 10

var x: number = 20; var y: number = 2

x %= y; console.log(x)    // x=x%y returns 0

# Ternary Operator

```
(test-expression) ? (if-true-do-this) : (if-false-do-this);
```

( test-expression) looks like  $a < b$ ,  $x \neq y$ ,  $m = n$ . etc.

## Example 3.5

```
var a = 100; var b = 200;  
var result = ( a < b ) ? "apple" : "banana";  
// (test-expression) ? (if-true-do-this) : (if-false-do-this);  
console.log ( result );
```

## Output:

apple

## Explanation:

The conditional operator use  $(a < b)$  to test the “a” and “b”, because “a” is less than “b”, it is true. Therefore, the output is “apple”.

# Typeof Operator

The “typeof” is a unary operator; it returns the data type the operand.

```
typeof variable
```

## Example 3.6

```
var myVariable1 = "Scala in 8 Hours";  
var myVariable2 = 100;  
console.log( typeof myVariable1 );  
console.log( typeof myVariable2 );
```

## Output:

string

number

## Explanation:

“typeof myVariable” returns the data type of “myVariable”.

## “+” and “-” Operators

“+” can connect two strings.

“-” can change the sign of a number.

### Example 3.7

```
var book: string = "Perl" + " in 8 Hours"  
console.log(book)  
var mark: number = -100  
var score = - mark;  
console.log("Old mark is: ",mark);  
console.log("New Score is: ",score);
```

### Output:

Perl in 8 Hours

Old mark is: -100

New Score is: 100

### Explanation:

"Perl" + " in 8 Hours" connects two strings by using “+”.

“- mark” changes the mark sign by using “-”.

# If Statement

```
if ( test-expression ) { // if true do this; }
```

“if statement” executes codes inside { ... } only if a specified condition is true, does not execute any codes inside {...} if the condition is false.

## Example 3.8

```
var a = 200;  
var b = 100;  
if ( a > b ) { // if true, do this  
  console.log ( "a is greater than b" );  
}
```

## Output:

a is greater than b

## Explanation:

( a>b ) is a test expression, namely (200>100), if returns true, it will execute the codes inside the { }, if returns false, it will not execute the codes inside the { }.

# If-else Statement

```
if ( test-expression) { // if true do this; }  
else { // if false do this; }
```

“if... else statement” runs some code if a condition is true and runs another code if the condition is false

## Example 3.9

```
var a = 100; var b = 200;  
if ( a > b ) { // if true do this  
  console.log ("a is greater than b. ")  
}  
else { // if false do this  
  console.log ( "a is less than b" );  
}
```

**Output:** a is less than b

## Explanation:

( a>b ) is a test expression, namely (100>200), if returns true, it will output “a is greater than b. ” if returns false, it will output “a is less than b”.

# Switch Statement

```
switch ( var variable ) {  
    case 1: if equals this case, do this; break;  
    case 2: if equals this case, do this; break;  
    case 3: if equals this case, do this; break;  
    default : if not equals any case, run default code; break;  
}
```

The value of the variable will compare each case first, if equals one of the “case” value; it will execute that “case” code. “break;” terminates the code running.

## Example 3.10

```
var number=20;  
switch ( number ) { // number value compares each case  
case 10 : console.log ( "Running case 10" ); break ;  
case 20 : console.log ( "Running case 20" ); break;  
case 30 : console.log ( "Running case 30" ); break;  
default : console.log ( "Running default code" ); break; }
```



**Output:**

Running case 20

**Explanation:**

The original number value is 20; it will match case 20, so it will run the code in case 20.

# Let & Const

“let” is used to declare a variable.

```
let variableName : type = value;
```

“const” is used to declare a constant.

```
const constantName : type = value;
```

## Example 3.11

```
let num: number = 100;    // define a variable
console.log ( num );

const PI: number = 3.14;  // define a constant
console.log ( PI );
```

## Output:

100

3.14

## Explanation:

“let” declares a variable, “const” declares a constant.

## Hour 4

# For Loop

```
for( init, test-expression, increment) { ... }
```

“for loop” runs a block of code repeatedly by the specified number of times.

## Example 4.1

```
for (var x = 0; x <= 4; x++) {    // repeat 4 times  
  console.log ( x );  
}
```

### Output:

0 1 2 3 4

### Explanation:

var x = 0 is an initializer, initializing variable “x” as 0.

x <= 4 is test-expression, the code will run at most 4 times.

x++ means that x will increase 1 each loop.

After 4 times loop, the code will output 01234.

# For in Loop

```
for (var val in list) { }
```

“for in loop” statement is used to iterate over a collection or list and outputs their values.

## Example 4.2

```
var x: any = "0 1 2 3 4"  
var y: any;  
for(y in x) {  
    console.log(x[y])  
}
```

## Output:

0 1 2 3 4

## Explanation:

“**for(y in x)**” iterates over each element of “x” collection, and stores each value to variable “y”, and output the “y” values.

# For of Loop

```
for (item val of object) { }
```

“for of loop” statement is used to iterate over the elements in a “collect” object and outputs their values.

## Example 4.3

```
var arr = [0, 1, 2, 3, 4];  
for (var item of arr) {  
    console.log(item);  
}
```

### Output:

0 1 2 3 4

### Explanation:

“**for (var item of arr)**” iterates over each element of “arr” collection, and stores each value to variable “item”, and output the “item” values.

# While Loop

```
while ( test-expression ) { ... }
```

“while loop” loops through a block of code if the specified condition is true.

## Example 4.4

```
var counter = 0;  
while (counter < 4) { // run 4 times  
  console.log( "&" );  
  counter++;  
}
```

## Output:

& & & &

## Explanation:

“counter < 4” is a test expression, if the condition is true, the code will loop less than 4 times, until the counter is 4, then the condition is false, the code will stop running.

# Do-While Loop

```
do{ ... } while ( test-expression);
```

“do... while” loops through a block of code once, and then repeats the loop if the specified condition is true.

## Example 4.5

```
var counter=0;  
do {  
  console.log ( "@" );  
  counter++;  
} while (counter<4) ; // run 4 times
```

## Output:

@ @ @ @

## Explanation:

“counter < 4” is a test expression, if the condition is true, the code will loop less than 4 times, until the counter is 4, then the condition is false, the code will stop running.



# Break Statement

“break” keyword is used to stop the running of a loop according to the condition.

```
break;
```

## Example 4.6

```
var num=0;
while (num<10){
if (num==5) break ; // exit the while loop
num++;
}
console. log ( num );
```

## Output:

5

## Explanation:

“if (num==5) break;” is a break statement. If num is 5, the program will run the “break” command; the break statement will exit the loop, then run “console. log (num)”.

# Continue Statement

“continue” keyword is used to stop the current iteration, ignoring the following code, and then continue the next loop.

```
continue;
```

## Example 4.7

```
var num=0;
while (num<5){
  num++;
  if (num==3) continue ; // go the next while loop
  console.log ( num );
}
```

## Output:

1 2 4 5

## Explanation:

Note that the output has no 3.

“if (num==3) continue;” means: When the num is 3, the program will run “continue” command, skipping the next command “console.log ( num )”, and then continue the next while loop.

# Function

A function is a code block that can repeat to run many times. To define a function, use “function function-name ( ) { }”.

```
function function-name ( ) {.....}
```

To call a function, use “function-name ( );”

```
function-name ( );
```

## Example 4.8

```
function myFun() { // define a function
  console.log("Very Good! ")
}
myFun(); // call the function
```

**Output:** Very Good!

### Explanation:

“function myFun() {...}” defines a function “myFun”.

“myFun();” calls a function “myFun”.

# Function with arguments

A function can have one or more arguments inside the bracket. Arguments are used to pass data to function.

```
function function-name (args ) {.....}
```

To call a function, use “function-name ( );”

```
function-name (arguments );
```

## Example 4.9

```
function myFun(arg ) {    // define a function with arg
    console. log( arg + " in 8 Hours! " )
}
myFun( "C#" );    // call the function with arguments
```

**Output:** C# in 8 Hours!

## Explanation:

“function myFun(arg) {...}” defines a function “myFun” with arg.

“myFun( “C#” );” calls a function “myFun” with argument “C#”.

# Return Values

“return” can return a value to the caller.

```
function function-name ( var arg ) { return value }
```

To call a function, use “function-name (argument);”

```
function-name (argument );    // caller
```

## Example 4.10

```
function add(num1,num2) {  
  return num1+num2;    // pass the result value to the caller  
}  
console.log ("3 + 5 = " + add( 3, 5 ) );    // caller
```

**Output:** 3 + 5 = 8

## Explanation:

“**return** num1+num2” returns the result to caller “add(3,5)”, you can treat it as “add(3,5) = **return** num1+num2”, assigning the value to add(3,5). Namely add (3,5) =8.

# Anonymous Function

Anonymous function is a kind of function without function name.

```
var handle = function() {...}
```

To call an anonymous function, we can use “handle()”

```
handle()
```

If don't use handle(), you can any words you like, e. g. test()

## Example 4.11

```
var book = function() {    // define an anonymous function
    return "Perl in 8 Hours";
}
console.log(book() )    // call the anonymous function
```

## Output:

Perl in 8 Hours

## Explanation:

“var book = function()” defines an anonymous function.

book() is a handle which is used to call the anonymous function.

# Lambda Function

The syntax of the lambda function is as follows:

```
( [args] ) => statement;
```

“args” are parameters of the lambda.

## Example 4.12

```
var book = (str:string) => str + " in 8 Hours";  
console.log(book("Go"));
```

### Output:

Go in 8 Hours

### Explanation:

“(str: string) => str + " in 8 Hours"” is a lambda expression.

(str: string) receives the came-in parameter “Go”.

“book” is a handle.

“book(Go)” is a caller.

# Hour 5



# String

String is a series of characters. The string object is used to process text. String is always enclosed by a pair of quotes.

1. The first syntax to create a string object is:

```
var obj = new String("string");
```

2. The second syntax to create a string object is:

```
var obj = "string";
```

## Example 5.1

```
var obj1 = new String("Swift") ;  
var obj2 = " in 8 Hours";  
var myStr = obj1 + obj2;  
console.log (myStr);
```

**Output:** Swift in 8 Hours

## Explanation:

“var obj1 = new String("Swift")” creates an object1.

“var obj2 = " in 8 Hours";” creates an object2.

# String Length

The syntax to get the length of a string is:

```
str.  
length
```

## Example 5.2

```
var obj = new String("Kotlin in 8 Hours! ")  
var num = obj.length    // get the string length  
console.log("The length is: " + num)
```

## Output:

The length is: 18

## Explanation:

“**obj.length**” gets the length of the string “Kotlin in 8 Hours! ”.

# Convert to String

The syntax to convert a number to a string is:

```
number.toString();
```

## Example 5.3

```
var num1 = 100;  
var num2 = 200;  
var str1 = num1.toString() ;  
var str2 = num2.toString() ;  
var str3 = str1 + str2    // connect two strings  
console.log ( str3 );
```

## Output:

100200

## Explanation:

“number.toString()” converts the number to a string, so the output is 100200, instead of 300.

# Find a Character

`str.charAt(index)`

Return the character at the specified index.

## Example 5.4

```
var obj = new String("HTML");    // create a string object "obj"
console.log("obj.charAt(0) is: " + obj.charAt(0));
console.log("obj.charAt(1) is: " + obj.charAt(1));
console.log("obj.charAt(2) is: " + obj.charAt(2));
console.log("obj.charAt(3) is: " + obj.charAt(3));
```

## Output:

```
obj.charAt(0) is: H
obj.charAt(1) is: T
obj.charAt(2) is: M
obj.charAt(3) is: L
```

## Explanation:

“obj.charAt(0)” returns the character at the index 0.

# Connect Two Strings

The syntax to connect two strings is as follows:

```
str1.concat(str2.toString());
```

## Example 5.5

```
var str1 = new String( "Django" );  
var str2 = new String( " in 8 Hours" );  
var str3 = str1.concat(str2.toString());  
console.log(str3);
```

## Output:

Django in 8 Hours

## Explanation:

“str1.concat(str2.toString());” connects two strings.

## Locate a SubString (1)

Find the index of the first occurrence of a specified substring in a string:

```
str.indexOf ( " substring " )
```

### Example 5.6

```
var str = new String( "Kotlin in 8 Hours" );  
var index = str.indexOf( "in" ) ;  
console. log("The index is: " + index );
```

### Output:

The index is: 4

### Explanation:

“str. indexOf( "in" )” finds the index of the first occurrence of the “in” in the string “Kotlin in 8 Hours”. There are two “in”s in the string, the index of the first “in” is 4.

## Locate a SubString (2)

Find the index of the last occurrence of a specified substring in a string:

```
str.lastIndexOf ( " substring " )
```

### Example 5.7

```
var str = new String( "Kotlin in 8 Hours" );  
var index = str.lastIndexOf( "in" ) ;  
console.log("The index is: " + index );
```

### Output:

The index is: 7

### Explanation:

“str.lastIndexOf( "in" )” finds the index of the last occurrence of the “in” in the string “Kotlin in 8 Hours”. There are two “in”s in the string, the index of the last “in” is 7.

# Replace a String

The syntax to replace an old string with a new string

```
oldstring.replace( oldstring, newstring );
```

## Example 5.8

```
var oldstr = "JavaScript";  
var newstr = "TypeScript";  
var mystr = oldstr.replace( oldstr, newstr ) ;  
console.log( mystr );
```

## Output:

TypeScript

## Explanation:

“oldstr.replace( oldstr, newstr )” replaces the oldstr with newstr.



# Get a Substring

We can extract a substring of a string from the index1 to index 2.

```
str.substring(index1, index2);
```

## Example 5.9

```
var str = "Shell Scripting in 8 Hours";  
var newstr = str.substring(6, 15);  
console.log("The substring is: " + newstr);
```

## Output:

Scripting

## Explanation:

“str.substring(6, 15)” gets a substring from the index 6 to index 15 of the string “Shell Scripting in 8 Hours”.

# Get Unicode

```
str.charCodeAt(index);
```

Return the Unicode of the character that is at the specified index.

## Example 5.10

```
var str = new String("PERL");  
console.log("P Unicode is: " + str.charCodeAt(0) );  
console.log("E Unicode is: " + str.charCodeAt(1) );  
console.log("R Unicode is: " + str.charCodeAt(2) );  
console.log("L Unicode is: " + str.charCodeAt(3) );
```

## Output:

P Unicode is: 80

E Unicode is: 69

R Unicode is: 82

L Unicode is: 76

## Explanation:

“str.charCodeAt(0)” returns the Unicode of the character that is at index 0.

# Case Conversion

The syntax to convert a lowercase / uppercase is:

```
str.toLowerCase( )  
str.toUpperCase( )
```

“str.toLowerCase( )” converts the string into lowercase.

“str.toUpperCase( )” converts the string into uppercase.

## Example 5.11

```
var str = "Swift in 8 Hours";  
console.log(str.toLowerCase( ));  
console.log(str.toUpperCase( ));
```

### Output:

swift in 8 hours

SWIFT IN 8 HOURS

### Explanation:

“str.toLowerCase( )” converts the string into lowercase.

“str.toUpperCase( )” converts the string into uppercase.

# String Functions

## Other String Functions:

<b>match()</b>
Look for one or more string that matches the regular expression .
<b>search()</b>
Looks in a string for a value that matches the regular expression .
<b>slice()</b>
Extract and return a fragment of a string
<b>split()</b>
Split a string into an array of substrings
<b>substr()</b>
Get a substring according to the specified index .
<b>valueOf()</b>
Get the value of the string object .
<b>localeCompare()</b>
Compare two strings .

# Hour 6

# Create an Array

An array is a particular variable, which can contain one or more values at the same time.

The syntax to create an array is:

```
var array_name : type = [ val0, val1,  
val2...]
```

## Example 6.1

```
var arr:string[ ] = ['A', 'B', 'C', 'D']    // create an array  
console.log(arr[0]);    // access index 0 element  
console.log(arr[1]);  
console.log(arr[2]);  
console.log(arr[3]);
```

**Output:**    A   B   C   D

### Explanation:

Above code creates an array, array name is “arr”, and it has four elements: arr[0], arr[1], arr[2], arr[3]. Its indexes are 0, 1, 2 and 3. Its values are A, B, C, and D.

Note that index begins with zero.

# Array Object

The syntax to create an array object is:

```
var array_names: type[ ] = new Array(length)
```

## Example 6.2

```
var arr:string[ ] = new Array(4) // create an array object
arr[0] = "A";    // assign value A to the index 0 element
arr[1] = "B";
arr[2] = "C";
arr[3] = "D";
console.log (arr[0]);
console.log (arr[1]);
console.log (arr[2]);
console.log (arr[3]);
```

**Output:**    A   B   C   D

## Explanation:

“var arr: string[ ] = new Array(4)” creates an array object “arr”, which contains four elements, its type is a string type.

“arr[0] = "A";” assigns a value “A” to the zero element.

# Iterating Over an Array

Iterating Over an Array can access each element of the array.

```
for(index in array_name) {...}
```

Usually we use this way to get each element in an array.

## Example 6.3

```
var index: any;  
var arr: string[ ] = ["A", "B", "C", "D"]  
for(index in arr) {    // iterate over the array  
    console.log(arr[index])  
}
```

## Output:

A B C D

## Explanation:

“for(index in arr) {...}” iterates over each element of the array “arr” by using the index.



# Array Assignment

We can assign the each element value of an array to multiple variables.

```
var[ x, y, z ] = array;
```

Assign each value of the array to the variable x, y, z respectively.

## Example 6.4

```
var arr: number[] = [10, 11, 12]
var[ x, y, z ] = arr;    // assign the each element value
console.log(x)
console.log(y)
console.log(z)
```

## Output:

10 11 12

## Explanation:

“var[x,y,z] = arr;” means: x=10, y=20, z=30.

# Connect Two Arrays

The syntax to connect two arrays is:

```
var new_array = array1.concat(array2);
```

## Example 6.5

```
var arr1 = [1,2,3,4];  
var arr2 = [5,6,7,8];  
var myArr = arr1.concat(arr2);    // connect two arrays  
console.log("The connected array is: " + myArr );
```

## Output:

The connected array is: 1,2,3,4,5,6,7,8

## Explanation:

“var myArr = arr1.concat(arr2);” connects arr1 and arr2.

# Array Length

The syntax to get the length of an array is:

```
array.length
```

## Example 6.6

```
var arr1 = [1,2,3,4];  
var arr2 = [5,6,7,8];  
var myArr = arr1.length + arr2.length ;    // get length  
console.log("The length of the two arrays is: " + myArr );
```

## Output:

The length of the two arrays is: 8

## Explanation:

“arr1.length + arr2.length” gets the length of two arrays.

# Unshift() Function

“unshift()” can add an element to the beginning of an array.

```
array.unshift("element")
```

## Example 6.7

```
var arr = new Array(" A ", " B ", " C ", " D ");  
arr.unshift(" First "); // add an element to the beginning  
console.log("The return array elements are: " + arr );
```

## Output:

The return array elements are: First, A, B, C, D

## Explanation:

“arr. unshift(" First ")” adds an element “First” to the beginning of the array.

# Shift() Function

“shift()” can remove an element at the beginning of an array.

```
array.shift( );
```

## Example 6.8

```
var arr = new Array("First", " A ", " B ", " C ", " D ");  
arr.shift( );    // remove an element at the beginning  
console.log("The return array elements are: " + arr );
```

## Output:

The return array elements are: A , B , C , D

## Explanation:

“arr. shift( )” removes the element “First” at the beginning of the array.

# Push() Function

“push()” function can add an element to the end of an array.

```
array.push("element");
```

## Example 6.9

```
var arr = new Array(" A ", " B ", " C ", " D ");  
arr.push(" Last ") ;    // add an element to the end  
console.log("The return array elements are: " + arr );
```

## Output:

The return array elements are: A , B , C , D , Last

## Explanation:

“arr.push(" Last ")” adds an element “Last” to the end of the array.

# Pop() Function

“pop()” removes an element at the end of the array.

```
array.pop();
```

## Example 6.10

```
var arr = new Array(" A ", " B ", " C ", " D ", " Last ");  
arr.pop() ;    // removes an element at the end  
console.log("The return array is: " + arr );
```

## Output:

The return array is: A , B , C , D

## Explanation:

“arr. pop( )” removes the element “Last” at the end of the array.

# Sort Array

The syntax to sort all elements of an array is:

```
array.sort( );
```

## Example 6.11

```
var arr = new Array(" C ", " B ", " D ", " A ");  
var sorted = arr.sort() ;    // sort the array  
console.log("The sorted elements are: " + sorted );
```

## Output:

The sorted elements are: A, B, C, D

## Explanation:

“arr.sort()” sorts all elements of the array.



# Array Functions

<b>every()</b>
Checks whether each element meets the specified criteria
<b>filter()</b>
Checks all elements and returns a qualified array .
<b>forEach()</b>
Each element of the array executes a callback function once .
<b>indexOf()</b>
Returns the first occurrence index of a specified string element
<b>join()</b>
Join all the elements of the array as a string .
<b>lastIndexOf()</b>
Returns the last occurrence index of a specified string element
<b>reverse()</b>
Reverses the sequence of the array elements
<b>slice()</b>
Extracts a portion of the array and returns a new array
<b>some()</b>
Checks if any array element meets the specified criteria
<b>splice()</b>
Add or remove elements from an array .
<b>toString()</b>
Converts an array into a string.

## Hour 7

# Tuple

Tuple is a special array, the elements in the tuple can be various data types, namely a tuple can contain different type of elements.

1.

The first syntax to create a tuple is:

```
var tupleName = [value1,value2,value3,...]
```

The data type of every element can be different.

2.

The second syntax to create a tuple is:

```
var tupleName = [ ];  
tupleName[0] = value0;  
tupleName[1] = value1;  
tupleName[2] = value2;
```

Create an empty tuple first, and then initializes each element.

# Access Tuple Element

The syntax to access the element of a tuple is:

```
tupleName[index]
```

## Example 7.1

```
var tup = ["Score", 100, true];    // create a tuple
console.log( tup[0] )
console.log( tup[1] )
console.log( tup[2] )
```

## Output:

Score

100

true

## Explanation:

“tup[0]” accesses the first element of the tuple, and returns its value “Score”.

# Update a Tuple

The syntax to update a tuple element value is:

```
tupleName[ index ] = value;
```

## Example 7.2

```
var tup = ["Score", 100, true];    // create a tuple
tup[0] = 10
tup[1] = 11
tup[2] = 12
console.log( tup[0] + tup[1] + tup[2] )
```

## Output:

33

## Explanation:

“tup[0] = 10” updates the first element value as 10.

“tup[1] = 11” updates the second element value as 11.

“tup[2] = 12” updates the third element value as 12.

# Tuple Assign Values

We can assign each Tuple element value to multiple variables.

```
var [ variable1, variable2, variable3 ] = tupleName
```

## Example 7.3

```
var tup = ["Score", 100, true];    // create a tuple
var [ x, y, z ] = tup              // tuple assign values
console.log( x )
console.log( y )
console.log( z )
```

## Output:

Score

100

true

## Explanation:

“var [ x, y, z ] = tup” assigns each tuple element value to three variables x, y, and z.

# Union Types (1)

Union Types means that the multiple variables can be declared different type by a pipe symbol “|”. The syntax is:

```
type1 | type2 | type3
```

## Example 7.4

```
var value: string | number | boolean    // union types
value = "Score"; console.log( value + " is a string type" )
value = 100; console.log( value + " is a number type" )
value = true; console.log( value + " is a boolean type" )
```

## Output:

Score is a string type

100 is a number type

true is a boolean type

## Explanation:

“var value: string | number | boolean” declares three data types of three variables.

## Union Types (2)

We can declare different type for various arrays.

```
type1[ ] | type2[ ] | type3 [ ]
```

### Example 7.5

```
var arr: string[ ] | number[ ];    // union types
arr = ["A","B","C"]
for(n = 0;n<arr. length;n++) {
    console. log(arr[n])
}
var n: number;
arr = [10,11,12]
for(n = 0;n<arr. length;n++) {
    console. log(arr[n])
}
```

**Output:**    A    B    C    11 12 13

### Explanation:

“var arr: string[ ] | number[ ];” declares two array types; One array is a string type, another array is a number type.



# Global & Local

The global variable is defined outside a function, it can be used in anywhere.

The local variable is defined inside a function, it can be used only inside the current function.

## Example 7.6

```
var globalNum = 200;    // global variable
function myFunc(): void {
  var localNum = 100;    // local variable
  console.log("The local variable: " + localNum )
}
console.log("The global variable: " + globalNum )
myFunc();
```

**Output:** The global variable: 200, The local variable: 100.

## Explanation:

“var globalNum = 200” defines a global variable, which can be used in anywhere.

“var localNum = 100” defines a local variable, which can be used only inside the current function.

# Class Definition

Class is the general name for all things of the same nature.

Classes describe common properties and methods of objects.

The syntax to define a class is:

```
class ClassName {    // defines a class
variable: type;    // declare a variable member
constructor(arg: type) { }    // declare a constructor
function() : type { }    // declare a method
}    // constructor is used to initialize variable member
```

**For example:**

```
class Vehicle {    // define a class “Vehicle”
    car: string;    // declare a variable member “car”
    constructor(car: string) {    // declare a constructor
        this. car = car    // “this” represents the current object
    }    // constructor is used to initialize variable member “car”
    drive(): void {    // declare a method “drove()”
        console. log("I am driving a new " + this. car)
    }
}    // constructor is used to initialize variable member “car” .
```

# Object Declaration

An object is an instance of a class.

```
var obj = new ClassName( args );  
obj. variable;  
obj. function( );
```

“var obj = new ClassName(args );” creates an object named “obj” for the class, and passes the args to the constructor.

“obj. variable;” means that “obj” accesses a variable.

“obj. function( );” means that “obj” accesses a method.

## For example:

```
var obj = new Vehicle("limousine ") // create an object “obj”  
obj. car // “obj” accesses the variable “car” .  
this. car // “this” represents the current object “obj”  
obj. drive() // “obj” accesses the method “drive()”
```

When an object is created, the constructor will be called automatically, and the arguments will be passed to the constructor, and the variable members will be initialized.

# Class & Object

## Example 7.7

```
class Vehicle {    // define a class
    car: string;    // declare a variable member
    constructor(car: string) {    // declare a constructor
        this. car = car // “this” represents the current object
    }
    drive(): void {    // declare a method
        console. log("I am driving a new " + this. car)
    }
}
var obj = new Vehicle("limousine ")    // create an object
obj. drive()    // “obj” accesses the method
```

## Output:

I am driving a new limousine

## Explanation:

This is a complete example of the class and object

“class Vehicle” defines a class “Vehicle”.

“var obj = new Vehicle("limousine ")” creates an object, calls the constructor, passes the parameter “limousine” to the constructor.

# Extends

A child class can extend most features of the parent class.  
The syntax for a child class to extend its parent class is:

```
class ChildClass extends ParentClass
```

## Example 7.8

```
class Building {    // create a parent class
    High: number
    constructor(param: number) {    // constructor
        this.High = param
    }
}
class House extends Building {    // extends
    size(): void {
        console.log("The height of the house is "
            + this.High + " feet. ")
    }
}
var obj = new House(1000);    // create an object
obj.size()
```

## Output:

The height of the house is 1000 feet .

## Explanation:

“extends” is a keyword for a child class to extend a parent class.

“class House extends Building {...}” means that a child class House extends a parent class Building.

Note:

The child class cannot extend the constructor and private member of the parent class.

The child class cannot extend multiple parent classes.

But the child class can be extended by the grandchild class, namely: A can be extended by B, B can be extended by C.

# Hour 8

# Multiple Extending

In TypeScript, Parent class can be extended by Child class; Child class can be extended by Grandchild class.

## Example 8.1

```
class Parent {    // define a parent class
    str: string;
}
class Child extends Parent {} // child extends parent
class Grandchild extends Child {} // grandchild extends child
var obj = new Grandchild();
obj.str = "The greeting from a Grandchild. "
console.log(obj.str)
```

## Output:

The greeting from a Grandchild.

## Explanation:

“class Child extends Parent {}” and “class Grandchild extends Child {}” is a typical code of the multiple extending.



# Overriding

The method in the child class can override the method in the parent class, if two method names, two parameters are the same.

## Example 8.2

```
class ParentClass {  
    myFunc():void {  
        console.log("I am from parent class")  
    }  
}  
class ChildClass extends ParentClass {  
    myFunc():void {    // overriding  
        console.log("I am from child class")  
    }  
}  
var obj = new ChildClass();  
obj.myFunc();
```

**Output:** I am from child class

## Explanation:

The method “myFunc(){}” in the child class can override the method “myFunc(){}” in the parent class, if two method names, two parameters are the same.

# Super Keyword

“super” keyword words in the child class. “super” can access a variable or a method of the parent class.

## Example 8.3

```
class ParentClass {  
  parentFunc(): void {  
    console.log("I am a method of the parent class")  
  }  
}  
  
class ChildClass extends ParentClass {  
  childFunc(): void {  
    super.parentFunc()    // supper calls the parentFunc()  
  }  
}  
  
var obj = new ChildClass();  
obj.childFunc();    // obj calls the childFunc()
```

## Output:

I am a method of the parent class

## Explanation:

“super. parentFunc()” means that the “super” can access the a variable or a method of the parent class.

# Static Variable & Method

A static variable and static method can be directly accessed by a class. A non-static variable and non-static method only can be accessed by an object.

## Example 8.4

```
class MyClass {  
    static num: number;    // define a static variable  
    static myFunc(): void {    // define a static method  
        console.log("The num value is "+ MyClass.num)  
    }  
}  
  
MyClass.num = 100    // a class accesses a static variable  
MyClass.myFunc()    // a class calls a static method
```

**Output:** The num value is 100

### Explanation:

“MyClass.num = 100” means that the class “myClass” can directly access the static variable.

“MyClass.myFunc()” means that the class “myClass” can directly calls the static method.

# Private Modifier (1)

A private variable & private method can be accessible only within the current class.

## Example 8.5

```
class MyClass {  
  private str1: string = "Good! "  // define a private variable  
}  
var obj = new MyClass()  
console.log(obj.str1)  // obj tries to access str1
```

### Output:

Error Messages!

### Explanation:

Variable 'str1' is private and inaccessible outside the current class 'MyClass', “obj. str1” cannot access private variable. So an error occurs!

## Private Modifier (2)

A private variable & private method can be accessible only within the current class.

### Example 8.6

```
class MyClass {  
  private str2: string = "Good! "  // define a private variable  
  myFunc(): void {  
    console.log(obj.str2 )  // obj is in the current class  
  }  
}  
var obj = new MyClass()  
obj.myFunc()
```

### Output:

Good!

### Explanation:

Variable 'str2' is private and accessible inside the current class 'MyClass'. So “obj. str2” can access private variable.

# Protected Modifier (1)

A protected member is only accessible in the current class, and its parent class or its child class.

## Example 8.7

```
class Vehicle {  
  protected drive1(): void {    // declare a protected method  
    console.log( "Good! " )  
  } }  
var obj = new Vehicle()    // create an object  
obj.drive1()             // obj tries to access drive1()
```

## Output:

Error Messages!

## Explanation:

“drive1()” is a protected method, it is inaccessible outside the current class, and parent class, or child class. So an error occurs!

## Protected Modifier (2)

A protected member is only accessible in the current class, and its parent class or its child class.

### Example 8.8

```
class Vehicle {  
  protected drive2(): void {    // declare a protected method  
    console.log( "Good! " )  
  }  
}  
class Car extends Vehicle{  
  drive3(): void{ obj.drive2() }    // obj is in the child class  
}  
var obj = new Car()    // create an object  
obj. drive3()
```

### Output:

Good!

### Explanation:

“drive2()” is a protected method, it is accessible in the current class, and parent class, or child class. So “obj. drive2()” can access the protected function.

# Interface (1)

An interface is an abstract class, which can be implemented by a class. The interface syntax is:

```
interface InterfaceName {...}    // declare an interface
class ClassName implements InterfaceName    // implements
```

## Example 8.9

```
interface Book {    // declare an interface “Book”
    title: string
}
class Ebook implements Book { // implement the interface
    title: string = "Shell Scripting in 8 Hours"
}
var obj = new Ebook()
console.log( obj.title )
```

**Output:** Shell Scripting in 8 Hours

## Explanation:

“interface Book {...}” declares an interface “Book”.

“class Ebook implements Book {...}” means that a class Ebook implements the interface Book.



## Interface (2)

An interface is an abstract class, which can be implemented by a variable.  
The interface syntax is :

```
interface InterfaceName {...}    // declare an interface
var VariableName : InterfaceName // implements
```

### Example 8.10

```
interface Books {    // define an interface “Book”
    title: string
}
var eBook:Books = {    // implement the interface
    title: "Perl in 8 Hours",
}
console.log(eBook. title)
```

**Output:** Perl in 8 Hours

### Explanation:

“interface Book {...}” declares an interface “Book”.

“var Ebook : Book {...}” means that a variable Ebook implements the interface Book.

# Object (1)

A object can contain multiple key / value pairs.

```
var objectName = {key1: "val1", key2: "val2"... }
```

## Example 8.11

```
var books = { // define an object "books"
  title1: "Swift in 8 Hours", // key/value pair
  title2: "Scala in 8 Hours" // key/value pair
};
console.log(books.title1)
console.log(books.title2)
```

## Output:

Swift in 8 Hours

Scala in 8 Hours

## Explanation:

“var books = {...}” is an object containing two key/value pairs.

“title1: "Swift in 8 Hours",” is a key/value pair

## Object (2)

The object value can be an anonymous function.

```
key: function () { }
```

### Example 8.12

```
var books = {    // define an object “books”  
  title: function () { }    // key/value pair  
};  
books.title = function () {    // define an anonymous function  
  console.log("Kotlin in 8 Hours");  
};  
books.title();    // reference the anonymous function
```

### Output:

Kotlin in 8 Hours

### Explanation:

“var books = {...}” is an object containing one key/value pair.

“title: function () { }” is a key/value pair.

“books.title = function () {...}” define an anonymous function.

# **TypeScript**

## **Questions & Answers**

# Questions

Please fill in the correct answers.

01.

```
var myScore: number = 100;  
var myStory: fill in = true; // data type  
console.log("My exam score is: " + myScore)  
console.log("My story is: " + myStory)
```

- A. string
- B. char
- C. bool
- D. boolean

02.

```
var myVariable1 = "Scala in 8 Hours";  
var myVariable2 = 100;  
console.log( fill in myVariable1 ); // output : string  
console.log( fill in myVariable2 ); // output : number
```

- A. string
- B. number
- C. typeof
- D. datatype

03.

```
var x: any = "1 2 3 4"  
var y: any;  
for( y fill in x ) {
```

```
        console.log( x[y] )
    }
```

A. ==

B. in

C. <

D. >

04.

```
var str = new String( "Kotlin in 8 Hours" );
```

```
var index = str. fill in ( "in" );
```

*/\* Find the index of the last occurrence of the “in” in the string “Kotlin in 8 Hours” . \*/*

```
console.log("The index is: " + index );
```

A. lastIndexOf

B. indexOf

C. index

D. Of

05.

```
var arr = new Array(" A ", " B ", " C ", " D ");
```

```
arr. fill in (" First " );
```

*// add an element to the beginning of an array .*

```
console.log("The return array elements are: " + arr );
```

A. pop

B. push

C. shift

D. unshif

06.

```
var tup = fill in "Score", 100, true fill in ;    // create a tuple
```

```
console.log( tup[0] )
```

```
console.log( tup[1] )
```

```
console.log( tup[2] )
```

A. {..... }

B. [..... ]

C. (..... )

D. <.... >

07.

```
class Parent {    // define a parent class
```

```
    str: string;
```

```
}
```

```
class Child fill in Parent {}    // child extends parent
```

```
class Grandchild fill in Child {}    // grandchild extends child
```

```
var obj = new Grandchild();
```

A. inherit

B. :

C. extends

D. implement

08 .

What is the output based on the following code?

```
var x: never;
```

```
var y: number = 100;
```

```
x = y;
```

```
console.log(x);
```

A. 100

B. error messages

- C. 0
- D. undefined

09.

**fill in** num: number = 100;    // define a variable

console.log ( num );

- A. def
- B. define
- C. \$
- D. let

10.

var **fill in** = function() {    // define an anonymous function

    return "Perl in 8 Hours";

}

console.log(**book()** )    // call the anonymous function

- A. anonymous
- B. function
- C. book
- D. def

11.

var str = new String("HTML");    // create a string object “str”

console.log("str. **fill in** (0) is: " + str.charAt(0));

// return the character at the specified index 0 .

- A. charAt
- B. indexAt
- C. charOf
- D. indexOf



12.

```
var arr = new Array(" A ", " B ", " C ", " D ", " Last ");
```

```
arr. fill in () ;
```

```
// removes an element at the end of the array .
```

```
console. log("The return array is : " + arr );
```

A. pop

B. push

C. shift

D. unshif

13.

```
var value: fill in      // union types
```

```
value = "Score"; console. log( value + " is a string type" )
```

```
value = 100; console. log( value + " is a number type" )
```

```
value = true; console. log( value + " is a boolean type" )
```

A. number | string | boolean

B. boolean | number | string

C. string | Boolean | number

D. string | number | boolean

14.

```
class ParentClass {
```

```
    myFunc(): void {
```

```
        console. log("I am from parent class")
```

```
    }
```

```
}
```

```
class ChildClass extends ParentClass {
```

```
    fill in {      // overriding
```

```
        console. log("I am from child class")
```

```
}  
}
```

- A. overriding
- B. function
- C. myFunc(): void
- D. function myFunc(): number

15.

What is the output based on the following code?

```
var num = new Number(100.1234);  
console.log(num.toFixed(3));
```

- A. 100
- B. 100.123
- C. 123
- D. 3

16.

**fill in** PI: number = 3.14;    // define a constant

```
console.log ( PI );
```

- A. const
- B. constant
- C. def
- D. let

17.

```
var book = (str: string) fill in str + " in 8 Hours";
```

// complete a lambda expression

```
console.log(book("Go"));
```

- A. >=

- B. <=
- C. =>
- D. =<

18.

// get the Unicode of the character that is at the specified index .

```
var str = new String("PERL");  
console.log("P Unicode is: " + str. fill in (0));  
console.log("E Unicode is: " + str. fill in (1));  
console.log("R Unicode is: " + str. fill in (2));  
console.log("L Unicode is: " + str. fill in (3));
```

- A. unicodeAt
- B. charAt
- C. codeAt
- D. charCodeAt

19.

```
var arr = new Array(" A ", " B ", " C ", " D ");  
arr. fill in (" Last ");  
// add an element to the end of an array  
console.log("The return array elements are : " + arr );
```

- A. pop
- B. push
- C. shift
- D. unshift

20.

```
fill in {    // defines a class  
variable: type;    // declare a variable member
```

```
constructor(arg: type) { }    // declare a constructor  
function() : type { }    // declare a method  
}
```

- A. ClassName: class
- B. class: ClassName
- C. ClassName class
- D. class ClassName

21.

```
class ParentClass {  
  parentFunc(): void {  
    console.log("I am a method from the parent class")  
  }  
}  
  
class ChildClass extends ParentClass {  
  childFunc(): void {  
    fill in . parentFunc()    // calls the parentFunc()  
  }  
}
```

- A. super
- B. obj
- C. object
- D. ParentClass

22.

```
var arr = new Array("First", " A ", " B ", " C ", " D ");  
arr. fill in ();  
// remove an element at the beginning of an array .  
console.log("The return array elements are: " + arr );  
A. pop  
B. push
```

- C. shift
- D. unshif

23.

```
var obj = fill in ClassName( args );  
// create an object
```

- A. create
- B. object
- C. new
- D. implement

24.

```
interface InterfaceName { ... } // declare an interface  
var VariableName fill in InterfaceName // implements
```

- A. implement
- B. implements
- C. extends
- D. :

# Answers

01. D	09. D	17. C
02. C	10. C	18. D
03. B	11. A	19. B
04. A	12. A	20. D
05. D	13. D	21. A
06. B	14. C	22. B
07. C	15. B	23. C
08. B	16. A	24. D

**Source code download link:**

<https://forms.aweber.com/form/83/138213183.htm>

# Source Code Download

[Angularjs Programming](#)

[C# Programming](#)

[C++ Programming](#)

[Django in 8 Hours](#)

[Go in 8 Hours](#)

[Html Css Programming](#)

[Java Programming](#)

[JavaScript Programming](#)

[JQuery Programming](#)

[Kotlin in 8 Hours](#)

[MySql Database](#)

[Node . Js in 8 Hours](#)

[Perl in 8 Hours](#)

[Php MySql Programming](#)

[PowerShell in 8 Hours](#)

[Python Programming](#)

[R Programming](#)

[Ruby Programming](#)

[Rust in 8 Hours](#)

[Scala in 8 Hours](#)

[Shell Scripting in 8 Hours](#)

[Swift in 8 Hours](#)

[Visual Basic Programming](#)

[Xml Json in 8 Hours](#)

**Source code download link:**

<https://forms.aweber.com/form/83/138213183.htm>