未来领域

Deferred Tuition Program

Course Preparation

OVERVIEW & PURPOSE

Future Sphere's Deferred Tuition Program is a highly competitive and selective program. In order to be accepted to this program you will need to demonstrate commitment and learning capabilities to follow through our curriculum.

Please follow the instructions and take 1 week to complete the reading list & the learning objectives outlined below. A technical screening will be conducted at the end of your preparation period in the form of a video interview. We will examine your learning results by asking you questions based on your reading list and learning objectives.

Your performance during the technical screening will determine your admission to the Deferred Tuition Program.

READING LIST & TOOLS

1. JavaScript Essentials
    a. https://www.w3schools.com/js/default.asp
    b. https://javascript.info/devtools
    c. https://javascript.info/intro
2. Tools
    a. Visual Studio Code
    b. https://JsFiddle.net

LEARNING OBJECTIVES

JavaScript

Variables and syntax

a. Let vs. const vs. var

Data types

b. String
c. Number
d. Boolean
e. Array
f. Object

Output

g. console.log()
h. window.alert()

Arithmetic

i. Addition/Subtraction/Division/Multiplication

Number methods

Math methods

Functions

j. Closure and scope

Loop

k. For Loop
l. For Each
m. Filter
n. Map
o. Find
p. Includes
q. IndexOf

Conditional Statement

r. If
s. Else if
t. Else

Comparison and logical operators

u. &&
v. ||
w. Modula

GETTING STARTED

The first thing you need to do is to install the Visual Studio Code at https://code.visualstudio.com/, this is the most commonly used coding editor for software engineers. Your coding interview will be conducted on this coding editor and you will practice and learn using this coding editor.

Javascript is a programming language that runs in the browser. To run Javascript in the browser, you need the help of a web page. Upon installing VS Code, you can start by creating a new folder and a new file called index.html, and a new file called index.js.

You can use the index.html and index.js file to test and practice on the learning objectives. For the purpose of the upcoming interview, we will only be testing your javascript knowledge.

Part 1 : Variables and syntax

JavaScript Variables:

Example:

```
var x = 5;

let y = 6;

const z = x + y;
```

From the example above, you can expect:

- x stores the value 5
- y stores the value 6
- z stores the value 11

==========

JavaScript Syntax:

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

```
var x, y, z;       // How to declare variables

x = 5; y = 6;      // How to assign values

z = x + y;         // How to compute values
```

==========

Let vs. const vs. var:

var:

  function scoped

  undefined when accessing a variable before it's declared

let:

  block scoped

  ReferenceError when accessing a variable before it's declared

const:

  block scoped

  ReferenceError when accessing a variable before it's declared

  can't be reassigned

==========

Part2 : Data types

JavaScript variables can hold many data types: number, string, boolean, array, and objects:

```
var length = 16;

// Number: The number type represents both integer and floating-point numbers.

var lastName = "Johnson";

// String: A string in JavaScript must be surrounded by quotes.

var isAdmin = true;

// Boolean: The boolean type has only two values: true and false.
```

```
var score = [100,90,80,70];

// Array:a single variable that is used to store different elements.

var x = {firstName:"John", lastName:"Doe"};

// Object: for more complex data structures.
```

==========

Part3 : Output

JavaScript can "display" data in different ways:

- **Writing into an alert box, using `window.alert()`.**
- **Writing into the browser console, using `console.log()`.**

window.alert():

Example:

```
windows.alert(“hello world”);
```

==========

console.log():

The console.log() method writes a message to the console.

The console is useful for testing purposes.

Tip: When testing this method, be sure to have the console view visible (press F12 to view the console).

Example:

```
console.log("hello world");
```

==========

Part3 : JavaScript Arithmetic

Arithmetic Operators:

Arithmetic operators perform arithmetic on numbers (literals or variables).

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modula (Remaining from division)

==========

Part5 : Number Methods

toString() Method:

The `toString()` method returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

Example:

```
var x = 123;

x.toString();            // returns 123 from variable x

(123).toString();        // returns 123 from literal 123

(100 + 23).toString();   // returns 123 from expression 100 + 23
```

==========

toFixed() Method:

`toFixed()` returns a string, with the number written with a specified number of decimals:

Example:

```
var x = 9.656;

x.toFixed(0);           // returns 10

x.toFixed(2);           // returns 9.66

x.toFixed(4);           // returns 9.6560

x.toFixed(6);           // returns 9.656000
```

※`toFixed(2)` is perfect for working with money.

==========

valueOf() Method:

`valueOf()` returns a number as a number.

Example:

```
var x = 123;

x.valueOf();            // returns 123 from variable x

(123).valueOf();        // returns 123 from literal 123

(100 + 23).valueOf();   // returns 123 from expression 100 + 23
```

==========

Part6 : Math methods

The JavaScript Math object allows you to perform mathematical tasks on numbers.

Math.round():

`Math.round(x)` returns the value of x rounded to its nearest integer:

Example

```
Math.round(4.7);    // returns 5

Math.round(4.4);    // returns 4
```

==========

Math.pow():

`Math.pow(x, y)` returns the value of x to the power of y:

Example

```
Math.pow(8, 2);      // returns 64
```

==========

Math.sqrt():

`Math.sqrt(x)` returns the square root of x:

Example

```
Math.sqrt(64);      // returns 8
```

==========

Math.abs():

`Math.abs(x)` returns the absolute (positive) value of x:

Example

```
Math.abs(-4.7);     // returns 4.7
```

==========

Math.ceil():

`Math.ceil(x)` returns the value of x rounded up to its nearest integer:

Example

```
Math.ceil(4.4);     // returns 5
```

==========

Math.random():

`Math.random()` returns a random number between 0 (inclusive), and 1 (exclusive):

Example

```
Math.random();     // returns a random number
```

==========

Part7 : Functions

A JavaScript function is a block of code designed to perform a particular

task.

A JavaScript function is executed when "something" invokes it (calls it).

Example

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

==========

Closures:

JavaScript variables can belong to the local or global scope.

Global variables can be made local (private) with closures.

Example

```
var add = (function () {
  var counter = 0;
  return function () {counter += 1; return counter}
})();
add();
add();
add();
// the counter is now 3
```

Example Explained

The variable `add` is assigned the return value of a self-invoking

function.

The self-invoking function only runs once. It sets the counter to zero (0), and returns a function expression.

This way add becomes a function. The "wonderful" part is that it can access the counter in the parent scope.

This is called a JavaScript closure. It makes it possible for a function to have "private" variables.

The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

※ A closure is a function having access to the parent scope, even after the parent function has closed.

==========

Global Variables :

A `function` can access all variables defined inside the function, like this:

Example

```
function myFunction() {
  var a = 4;
  return a * a;
}
```

But a `function` can also access variables defined outside the function, like this:

```
var a = 4;
function myFunction() {
```

```
  return a * a;
}
```

In the last example, a is a global variable.

In a web page, global variables belong to the window object.

Global variables can be used (and changed) by all scripts in the page (and in the window).

In the first example, a is a local variable.

A local variable can only be used inside the function where it is defined. It is hidden from other functions and other scripting code.

Global and local variables with the same name are different variables. Modifying one, does not modify the other.

※ Variables created without a declaration keyword (`var`, `let`, or `const`) are always global, even if they are created inside a function.

==========

Part8 : Loop

For Loop:

The `for` loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

Example

```
for (i = 0; i < 5; i++) {

text += "The number is " + i}
```

From the example above, you can read:

Statement 1 sets a variable before the loop starts (var i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

==========

forEach() Method:

Definition and Usage:

The `forEach()` method calls a function once for each element in an array, in order.

Note: the function is not executed for array elements without values.

Syntax:

array.forEach(*function(currentValue, index, arr), thisValue*)

Parameter Values:

Parameter	Description

function(currentValue, index, arr)	**Required. A function to be run for each element in the array.** **Function arguments:** **Argument — Description** ***currentValue*** — **Required. The value of the current element** ***index*** — **Optional. The array index of the current element** ***arr*** — **Optional. The array object the current element belongs to**
thisValue	**Optional. A value to be passed to the function to be used as its "this" value.** **If this parameter is empty, the value "undefined" will be passed as its "this" value**

Example

For each element in the array: update the value with 10 times the original value:

```
var numbers = [65, 44, 12, 4];

numbers.forEach(myFunction)

function myFunction(item, index, arr) {

  arr[index] = item * 10;

}
```

For each element in the array: multiply the value with 10 and update the element value:

650,440,120,40

==========

filter() Method:

Definition and Usage

The filter() method creates an array filled with all array elements that pass a test (provided as a function).

Note: filter() does not execute the function for array elements without values.

Note: filter() does not change the original array.

Syntax

array.filter(*function(currentValue, index, arr), thisValue*)

Parameter Values

Parameter	Description
function(currentValue, index,arr)	**Required. A function to be run for each element in the array.** **Function arguments:**

Argument	Description
currentValue	**Required. The value of the current element**
index	**Optional. The array index of the current element**
arr	**Optional. The array object the current element belongs to**

thisValue	Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined" will be passed as its "this" value

==========

map() Method:

Definition and Usage

The `map()` method creates a new array with the results of calling a function for every array element.

The `map()` method calls the provided function once for each element in an array, in order.

Note: `map()` does not execute the function for array elements without values.

Note: this method does not change the original array.

Syntax

```
array.map(function(currentValue, index, arr), thisValue)
```

Parameter Values

Parameter	Description

function(currentValue, index, arr)	**Required. A function to be run for each element in the array.**

unction arguments:

Argument	Description
currentValue	**Required. The value of the current element**
index	**Optional. The array index of the current element**
arr	**Optional. The array object the current element belongs to**

thisValue	**Optional. A value to be passed to the function to be used as its "this" value.** **If this parameter is empty, the value "undefined" will be passed as its "this" value**

Example

Return an array with the square root of all the values in the original

array:

```
var numbers = [4, 9, 16, 25];

console.log(numbers.map(Math.sqrt));
```

Output:

2,3,4,5

==========

find() method:

Definition and Usage

The `find()` method returns the value of the first element in an array that pass a test (provided as a function).

The find() method executes the function once for each element present in the array:

- If it finds an array element where the function returns a *true* value, find() returns the value of that array element (and does not check the remaining values)
- Otherwise it returns undefined

Note: find() does not execute the function for empty arrays.

Note: find() does not change the original array.

Syntax

array.find(*function(currentValue, index, arr),thisValue*)

Parameter Values

Parameter	Description
function(currentValue, index, arr)	Required. A function to be run for each element in the array. Function arguments: Argument: *curr entV alue* — Required. The value of the current element Argument: *inde x* — Optional. The array index of the current element Argument: *arr* — Optional. The array object the current element belongs to
thisValue	Optional. A value to be passed to the function to be used as its "this" value. If this parameter is empty, the value "undefined"

will be passed as its "this" value

Example

Get the value of the *first* element in the array that has a value of 18 or more:

```
var ages = [3, 10, 18, 20];

function checkAdult(age) {

  return age >= 18;

}

function myFunction() {

  console.log(ages.find(checkAdult));

}
```

Output:

18

==========

includes() Method:

Definition and Usage

The includes() method determines whether an array contains a specified element.

This method returns *true* if the array contains the element, and *false* if not.

Note: The *includes()* method is case sensitive.

Syntax

array.includes(*element*, *start*)

Parameter Values

Parameter	Description
element	Required. The element to search for
start	Optional. Default 0. At which position in the array to start the search

Example

Check if an array includes "Mango":

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

var n = fruits.includes("Mango");

console.log(n);
```

Output:

True

==========

indexOf() Method

Definition and Usage

The indexOf() method searches the array for the specified item, and returns its position.

The search will start at the specified position, or at the beginning if no start position is specified, and end the search at the end of the array.

Returns -1 if the item is not found.

If the item is present more than once, the indexOf method returns the position of the first occurence.

Note: The first item has position 0, the second item has position 1, and so on.

Tip: If you want to search from end to start, use the lastIndexOf() method

Syntax

array.indexOf(*item*, *start*)

Parameter Values

Parameter	Description
item	Required. The item to search for
start	Optional. Where to start the search. Negative values will start at the given position counting from the end, and search to the end.

Example

Search an array for the item "Apple":

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
```

```
var a = fruits.indexOf("Apple");

console.log(a);
```

Output:2

==========

Part9 : Conditional Statement

Definition and Usage

The if/else statement executes a block of code if a specified condition is true. If the condition is false, another block of code can be executed.

The if/else statement is a part of JavaScript's "Conditional" Statements, which are used to perform different actions based on different conditions.

In JavaScript we have the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to select one of many blocks of code to be executed

Syntax

The if statement specifies a block of code to be executed if a condition is true:

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

The else statement specifies a block of code to be executed if the condition is false:

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

The else if statement specifies a new condition if the first condition is false:

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

Parameter Values

Parameter	Description
condition	Required. An expression that evaluates to true or false

Example

If the time is less than 20:00, create a "Good day" greeting, otherwise "Good evening":

```
var time = new Date().getHours();
if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

==========

Part10 : Comparison and logical operators

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x === 5 || y === 5) is false
%	Modulus (division remainder)	x = y % 2(result=1)

==========

INTERVIEW QUESTIONS BANK

- Write a function that
  - The return output is the sum of inputs “a” and “b”
- Write a function that
  - The return output is the difference of inputs “a” and “b”
- Write a function that

    - The return output is the product of inputs “a” and “b”
- Write a function that
    - The return output is the quotient of inputs “a” and “b”
- Write a function that
    - The return output is true if
        - The input is greater than 18
    - The returns output is false if
        - The input is less than 18

NEXT STEPS

1. Schedule a video interview with one of our instructors or admission consultant.
2. After you successfully pass your interview, submit your program deposit 15 days before the class begins to reserve your spot.
3. If you do not pass the interview, you may schedule another one after 2 weeks.
4. Each candidate has 3 chances to pass the interview.