Fundamentals of Programming Concepts

Learning programming concepts are much important than learning programming languages. Concepts are same for all hundreds of programming languages, independent of platform and machine. When you understand the programming concepts, you can develop program for any machine. This document guides you to understand the concepts through JavaScript.

Programming Languages

There are thousands of programming languages are around the world. Each one is designed and developed for particular machine and particular platform. Example, programming language is same like our language which helps an employer needs to communicate with the employee in their language to get things done. Every device in this world understands machine language which is just 0's and 1's. It's very difficult to understand those 0's and 1's, so we made human readable language like Assembly, Basic, C, Pascal, etc. Whatever we write it's then converted into machine language by a compiler. Compiler is a like translator which translates programming language into another language which machine understands. Every language has its own compiler. Before starting to write programs for machines, you need to take a look at the following things.

1. How many programming languages can do the same task?
2. Which language performs faster for that task?
3. Which language is worldwide supported?
4. Don't stick to the language you already know.
5. Study pro's and con's of different languages.
6. Look at competitor's product, which language they had used? What are the pro's and cons'?

When you had finalized everything, based on the task you need to decide programming concepts. Always try to write code for re-usable, flexible, and up-datable. Following sections covers most of the concepts in all the programming

languages, only the syntax change for different language but the concept remains same.

Variables

Variables are used to store values (alphabets and numbers), group of values (array) and function. A variable can be created, changed, and deleted at runtime (when program is running).

JavaScript syntax:

```
var a;
var b = “JavaScript”;
var c = 5;
var d = c;
```

Explanation: You can declare empty variables which can be updated during runtime. You can declare values for a variable while programming. You can also assign one variable to another variable, so that values form one variable will be stored in another variable.

Scope of variables

Scope of variables simply means deciding who are all have access to the variable. For example, when you declare variable outside of all functions, it will be treated as *global variable* which means from anywhere you can access the variable.

JavaScript syntax:

```
1. var a = “Hello”;
2. function sayHello(){
3.  alert(a);
4. }
5. function sayBye(){
6.  var b = “Bye”;
7.  alert(b);
8. }
9. sayHello(); sayBye();
10. alert(a);
11. alert(b);
```

Explanation: A program always executed line by line. In line 1, variable a is declared as global variable, functions will be executed only when they are called (invoked), so both functions will be skipped. In line 9, both functions are called one by one, so that the first function *sayHello* shows alert box. In second function, variable *b* declared as local variable, it will be accessed only inside the function *sayBye*, so it shows alert box. In the last two lines, only one alert box will be displayed, *alert(b)* will show error.

Note: When you access a variable name, first it looks for local variable with same name, if a local variable also created with same name *a* it will be accessed, otherwise it will look out for global variable with the same name.

Operators

Operators are used to perform mathematical tasks. The major types of operators available are arithmetic, assignment, string, comparison, and logical.

JavaScript syntax:

```
var a = 5;
var b = 4;
var c = a + b;      // +, -, *, /, %, ++, --
alert(c);           // 9
```

```
// Assignment operator
var c = b;
// String operator
var a = “Hello”;
var b = “World”;
var c = a + b;   // ”HelloWorld”
// Comparision operator
var a = 5;
var b = 4;
If ( a<b ){…}   // Returns true or false. True equals 1, false equals 0
// Logical operator
var a = 5;
var b = 4;
var c = 5;
If ( a = b || c ){…}   // OR ||   AND &&   NOT !
```

You can use *typeof* operator to check the data type during runtime.

```
typeof "John"                 // Returns string
typeof 3.14                   // Returns number
```

if and else..if

You can use if or else if statement to check whether the condition returns true or false. Based on the result, program will execute the respective part.

JavaScript syntax:

```
var a = “Hello”;

if (a == “Hello”){
    alert(“yes”);
}

if (a == “Hello”){
    alert(“Yes”);
}else{
    alert(“No”);
}

if (a == “Hello”){
    alert(“Yes”);
}else if(a == “Bye”){
    alert(“No”);
}else{
    Alert(“Don’t Know”);
}
```

Loops

If you want to repeat certain actions for n number of times, you need to use loops. There are three different type of loops in JavaScript, For…, While…, and Do…While.

JavaScript syntax:

```
for (i=0;i<100;i++){
    sayHello();
}
```

```
var i=0;
while (i<100){
    sayHello();
    i++;
}
```

```
var i=0;
do{
    sayHello();
    i++;
}while(i<100)
```

Explanation: All the above three loops have starting value, condition and increment operator. Each loop calls *sayHello* hundred times.

Note: Do…While loop always be executed one time even if the condition is false. Because *while* condition is mentioned in last, *do* part will be executed without checking the condition.

Functions

Function consists of many actions budled together with a name. Action inside the function block will be executed only if the function is called (invoked).

JavaScript syntax:

```
function sayHello(){
    alert("Hello!");
    alert("Welcome!");
}
sayHello();
```

```
var a = sum(5,4);
function sum(n1,n2){
    return n1+n2;
}
```

Explanation: Code inside *sayHello* function only executed at the time of calling. You can pass *parameters* (n1 and n2) to process actions and *return* will send back the result of a function to the line which invoked the function.