Jump Statements

- continue - Skips current counter and continues to next
- break - Comes out of block but remains in script
- return - Comes out of script

Exception Handling Statements in JavaScript

PAGE No.
DATE / /

- In computer programming we have to handle 2 types of errors

a) Compile time errors
b) Runtime errors

a) Compile time errors

- These are the errors related to septure declaration
  - Application fails to ~~complie~~ compile
  - We can't view output

b) Runtime errors:

- These are errors identified during runtime
  - Application compiles successfully
  - Application starts working
  - Application faces catestrophic failure
  - This leads to abnormal termination of application.

FAQ: What is purpose of Exception handling ?

Ans: Used to avoid abnormal termination

- Javascript exception handling statements are

a) try
b) catch
c) throw
d) finally

a) try:
- It is monitoring block
- It contains statements to execute

b) catch:
- It is handler block
- ~~It contains statements~~
- It will catch exeception thrown by application

c) throw:
Explicitly throws exception & messaye

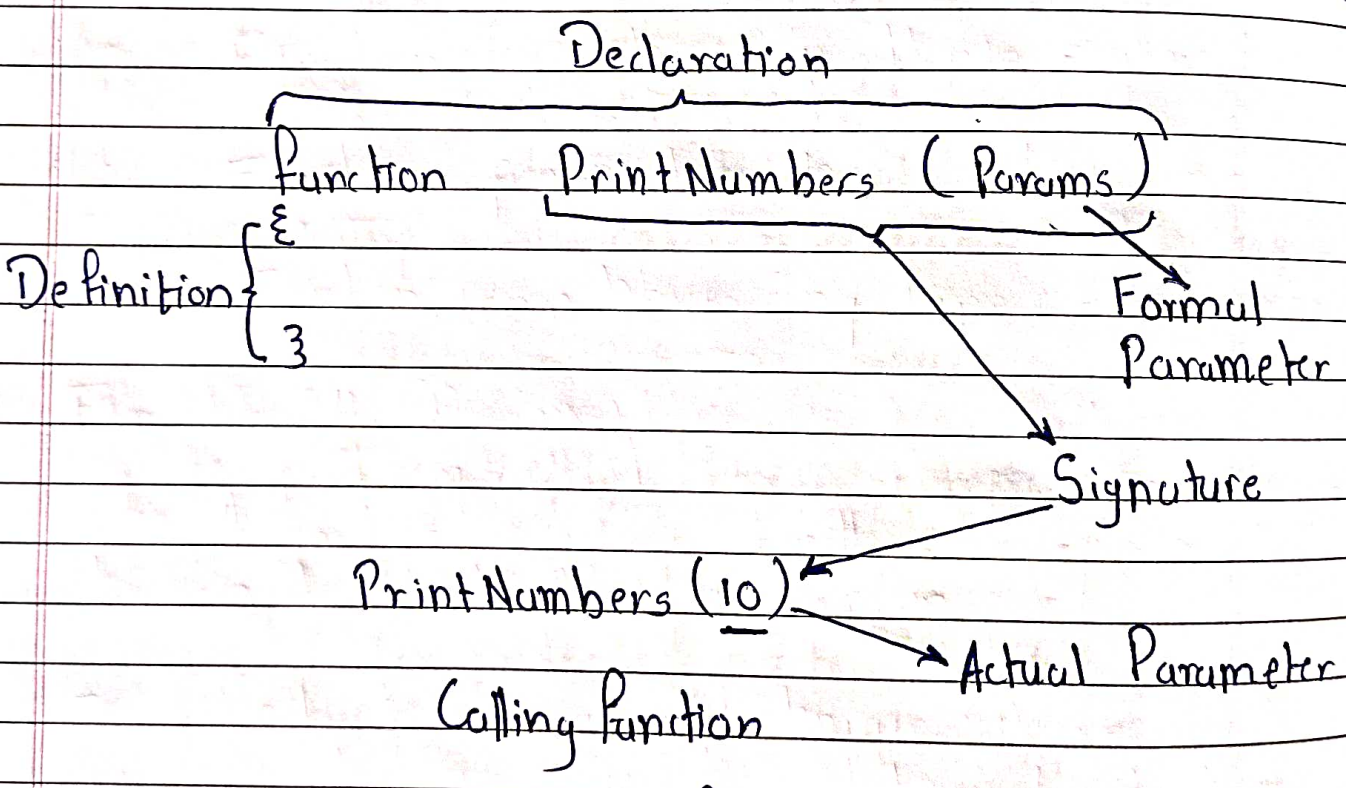
d) finally:
Comprises of statements to execute in all situations.

Note:
We don't need exception handling while working with HTML

JavaScript Functions

PAGE No.
DATE / /

- A function is used for "refactor."
- Refactoring means encapsulating set of statements and keeping them under one name so that we can use them any where and for any no. of times.
- It allows encapsulation and reusability
- Function configuration comprises of following



Note: PrintNumbers function accepts one Argument

PAGE No.
DATE / /

FAQ: What is Formal Parameter?

Ans: Parameter defined in function declaration

FAQ: What is Actual Parameter?

Ans: Parameter defined into function while calling the function

FAQ: What are Arguments?

Ans: Arguments define how many parameters are configured in function

FAQ: How a function is called?

Ans: By using its signature
What a function can do is defined in definition

Syntax:

```
Function PrintNumbers()
{
    for(var i=1; i<=10; i++)
    {
        document.write(i + "<hr>");
    }
}
```

DATE / /

* Function parameters

- Function can be parameter less or parameterized.

- Function requires parameters in order to modify the functionality.

- Every parameter defined in function is mandatory

- Function can have multiple parameters

- Function parameters have order dependency

- Function can have any type of parameter

* Function As Parameters

- A function can use functions as parameters to handle call back mechanism.

- Call back is a technique where function is executed according to state and situation.

FAQ: How many params are not allowed in a function?

Ans: There is no limit for parameter but as per ECMA standards max 1024 parameters are recommended

* Rest Parameters

- JavaScript ES5 and higher versions support Rest parameter
- A single rest parameter can allow multiple arguments.
- A rest parameter is defined by using "...param".

Syntax:

```
function Name(...restParameter)
{
}
```

```
Name(arg1, arg2, arg3, arg4...);
```

- A function can have only one rest parameter
- Rest parameter must be last parameter in formal list

Syntax:

```
function Name(...rest1, ...rest2) // Invalid
function Name(...rest1, ID) // Invalid
function Name(ID, ...rest1) // Valid
function Name(...rest) // Valid.
```

FAQ: Why rest parameter is last parameter?

Ans: As it has to read values upto end.

FAQ: Why function can't have multiple resp parameters?

Ans: As one rest parameter will read upto end you don't need multiple rest parameters and function will not allow multiple.

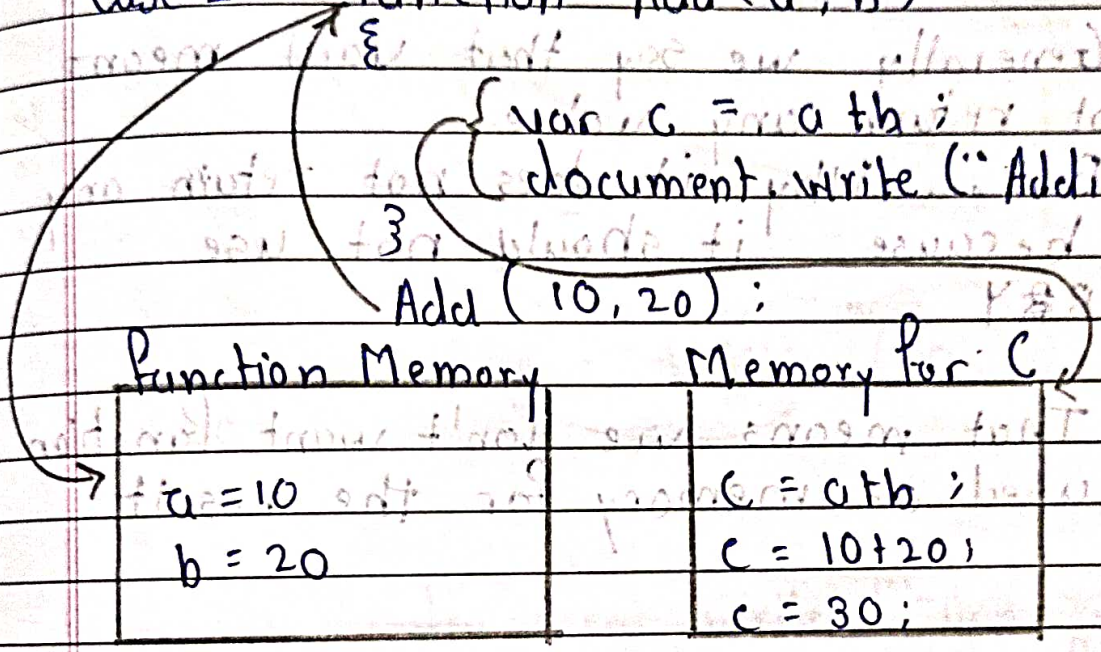
* Function with return type defined

Note :

Case 1 : function Add(a, b)

{ var c = a+b;
document.write("Addition : " + c);
}

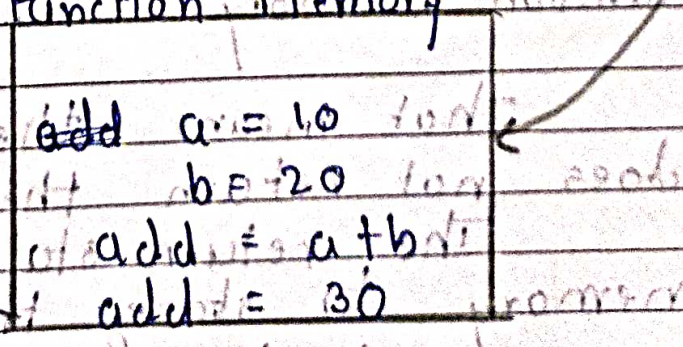
Add(10, 20);



Case 2 : function Add(a+b)

{
~~var~~ return a+b;
}

Add(10, 20);



Note:

What is meaning of void ?

-> void means discard the memory

Generally we say that void means do not return any value.

But... Why it does not return any value because it should not use MEMORY

That means we don't want Function to be used as memory for the result

Note:

What does return do ?

-> Return is a Jump Statement

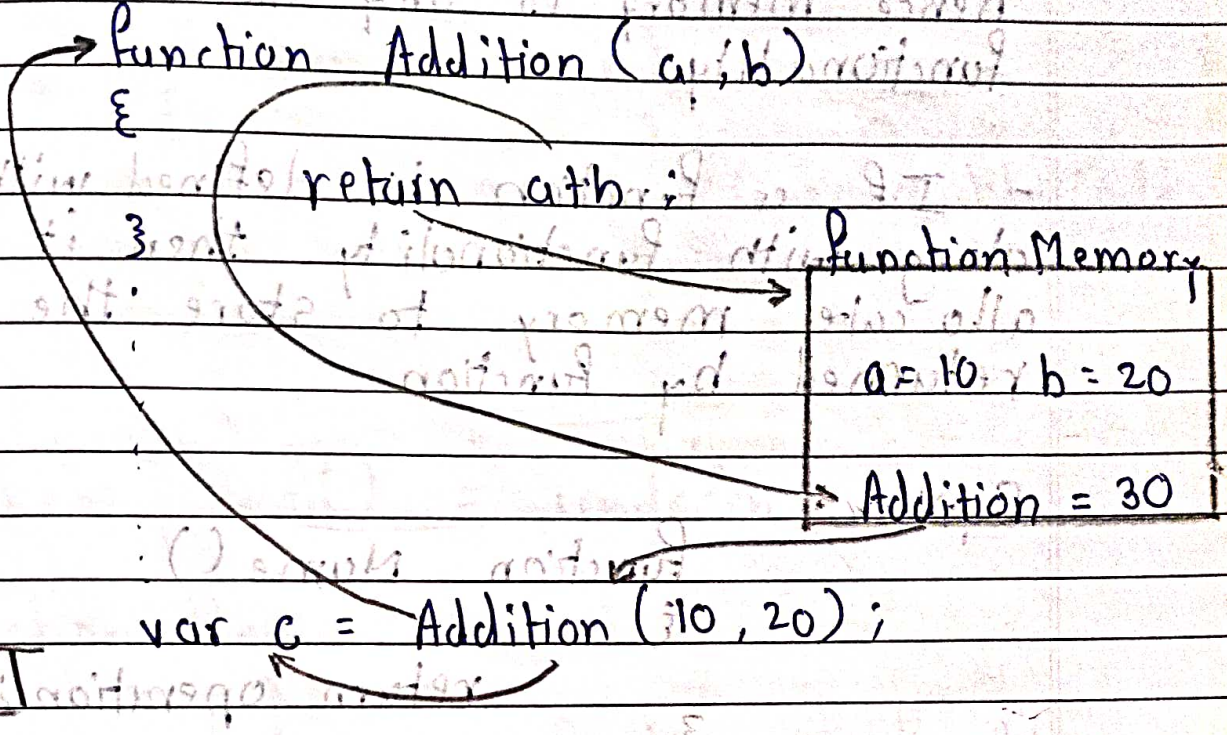
Return will store the result in the reference of Function

i.e. the result will be stored in function memory

That means the return statement does not return the value

The return statement is allocating memory to store the value returned by perticular function

Ex: (Conclusion):



document.write("Type of: " + typeof Addition);

Function

document.write("Type of: " + typeof Addition());

Number

- Function usually discards the return type hence memory is only allocated to perform functionality.

- If a function is defined with return along with functionality then it can allocate memory to store the result returned by function

Syntax:

```
Function Name()
{
    return operation;
}
```

- If return is not ~~used~~ using any operation then it is used to terminate the execution.

- Any statement after return statement is not reachable to compiler

FAQ: Can we define return keyword in a function that doesn't return value?

Ans: Yes it is used as Jump Statement

Syntax:

```
Function f1 (a,b)
{
    return a+b;          // Unreachable Code
}
```

Then f1() = undefined

* Function Recursion

- It is technique of calling a function within the context of current function.
- It is used for batch operations.

Ex.

```
<script>
    function Factorial (n)
    {
        if (n==0)
        {
            return 1;
        }
        return n* Factorial (n-1);
    }
    document.write (`Factorial : ${
                    Factorial (5)}`);
</script>
```

PAGE No.
DATE / /

* Function Closure

- It is a technique of configuring a function inside another function
- The members of outer function are accessible to inner function.

. The member of inner function are not directly accessible to outer function

- Closure technique allows the outer function to access the members of inner function.

- In this technique we design the inner function such that it returns a value and stores in the reference.

Ex.

```
<script>
    function Outer ()
    {
        var msg = " Outer Function "
        function Inner () {

            var OuterMsg = msg;
            var innerMsg = "Inner
                            Function"
```

```
        return outerMsg + "<br>" + innerMsg;
    }
    return Inner();
}
document.write(Outer());
</script>
```

Syntax:

```
function Outer()
{
    function Inner()
    {
        return value;
    }
    return Inner();
}
Outer();
```

* Anonymous Function:

- It is a function without name.
- Anonymous functions are used in call backs.

Ex:

```
var array = ['Hi', true, function(){}]
array[2]();
function Login(pass, success, fail){}
Login('1234', function(){}, function(){});
```

- Call back is technique of accessing function and loading into memory according to state and situation.

Ex.

```
<Script>
    var hello = function () {
        document.write ("Hello");
    }
    hello ();
</Script>
```

↳ Used to call anonymous function

* Arrow Function

- Arrow functions are used to define a function in short hand technique
- You can minify the function declaration and definition.

() : Configure function with parameters

=> : Configure return value

{} : Configure set of statements

The Arrow functions are called LAMDA Syntax

Syntax:

```
function hello (msg)
{
    return `Hello ! ${msg}`;
}
```

Can be Converted To..

```
var hello := msg => `Hello ! ${msg}`;
```

Syntax:

```
function welcome()
{
    document.write("Welcome");
}
```

```
var welcome = () => document.write
                    ("Welcome");
```

Syntax:

```
function print()
{
    document.write("Statement 1");
    document.write("Statement 2");
}
```

```
var print = () => {
    document.write("Statement 1");
    document.write("Statement 2");
}
```

Syntax:

```
function add (a, b)
{
    return a+b;
}
```

```
var add = (a,b) => a+b;
```

JavaScript OOP

PAGE No.
DATE / /

- In real time development various programming systems are used, like

a) POPS : Process Oriented Programming
b) OBPS : Object Based Programming
c) OOPS

a) POPS : Process Oriented Programming System

- Supports low level features
- Can directly interact with hard ware services.
- Usess less memory
- Faster in communication

Ex: C, Pascal, COBOL etc.

- Code reusability issues
- Code seperation issues
- No dynamic memory allocations
- Code security issues.

b) OBPS : Object Based Programming System

- Supports code seperation
- Supports code reusability
- Dynamic memory allocations.
- Suppors extensions

Ex. JavaScript Visual Basic etc.

- Code level security issues
- No contracts
- No templets
- No dynamic polymorphism

c) OOPS : Object Oriented Programming System

- Supports: contracts templets dynamic polymorphism
- Support code security

Ex. TypeScript, C++, Java, .Net langs etc

- They are tidious
- Use more memory
- Complexity in Configuration
- Slow.

FAQ : Is JS OOP language?

Ans : No. It supports certain features of OOP

PAGE No.
DATE / /

Note:

Features of OOPs	Characteristics of OOPs
Code reusability	Inheritance
Code Seperation	Encapsulation
Code Extensibility	Polymorphism
Code Security	Abstraction

* Object oriented Features of JavaScript

* Modules in JavaScript

- Module is set of functions values and classes
- Modules are used to build library for application
- You can import and implement library in your project

Modules provide

Reusability
Maintainability
Testability
Extensibility
Seperation

- To use modules ~~system~~ in a project you need a module system working in PC.

- There are various module systems like
  - Common JS
  - AMD [Asynchronous Module Distribution]
  - UMD [Universal Module Distribution]

- Node.js provides a module system called UMD

- Javascript can use the existing module system and implement in your application

Steps:

① You have to create a module, which is javascript file with set of functions, classes and variables

② You have to mark members with export

Note: Only exported members can be imported

Syntax:

```
export function name()
{

}
```

PAGE No.
DATE / /

③ You have to import the module and members of module in any page

Syntax:

```
import {member} from 'module_file_path';
```

④ You can implement the member in same location or in any another script source.

Note:

Script type for ~~mod~~ module system must be defined as "module".

```
<script type="module"></script>
```

- Every module can have only one default member to export

ex. export default function Name() {}

- The default member is directly imported

```
import DefaultMember, {OtherMembers}
from 'path';
```

↑ Non Default members

- You can't define directly constant values in a module if you want further implementation

- Always use formal declarations

Steps:

① Add new folder "Library"
② Add new file into folder "Product.js"
③ Add new HTML file

PAGE No.
DATE / /

Class in OOP

- In computer programming class is a program template.
- A class comprises of sample data and logic which you can implement and customize according to the requirements
- A class should be mutable i.e. it should be changable
- We can not have any immutable element in class
- All members in class should be mutable.

- Class have various behaviours
  a) Model
  b) Entity
  c) Blue print

- Class is refered as model when it is representing data.
- Class is refered as Entity when it is representing buisness requirements.
- Class is always blue print as it contains set of data and logic which you can implement at any location.

PAGE No.
DATE / /

Configuring Class:

1. Class Declaration
2. Class Expression

1. Class Declaration:

- Class Declaration is used for classes that are requested according to requirement

```
class ClassName
{
}
```

2. Class Expression:

- Class Expression is used for classes that are loaded according to state and situation.

```
var refName = class { };
```

PAGE NO.
DATE / /

Class Members

- A class comprises of following members

a) Properties
b) Methods
c) Accessors
d) Constructors.

FAQ: Can we define Variable in Class?
Ans: No

FAQ: Can we define Function in Class?
Ans: No

FAQ: Why we can not define variables and functions in class?
Ans: They are immutable. Class can contain only mutable members.

FAQ: What is difference between Variable and Property.
Ans: Variables are immutable. Properties are mutable i.e. Properties can change their behaviour according to the condition.

FAQ: What is difference between function and method?

Ans: Functions are immutable
Methods are mutable.

a) Properties:

- Properties are references in class memory where you can store data.
- Data is ~~state~~ stored in property
- You can store any type of data in property [Primitive or Non-Primitive]

Syntax:

```
class className
{
    Property = value;
}
```

- In order to access members of a class, you have to create an instance of class.

ex. let instance = new className

- "new" is dynamic memory allocating operator.

- "Constructor" is responcible for creating ~~and on~~ an object for class

- Properties are mutable, you can control the behaviour of property using "Accessors".

b) Accessors

- Accessors will provide a fine grained control over properties.

[grained control ≠ full control]

- Accessors define the restrictions for reading and writing data into properties

- Accessors are used for reading and writing data into properties, at the same time they can define restrictions.

- The accessors used in class are:

a) get() [Getter]
b) set() [Setter]

a) get() accessor is used for reading values from a property.

b) set() accessor is used for writing data into property

Syntax:

```
- propertyName;
get Property ()
{
    return this._propertyName;
}
Set Property (newName)
{
    this._propertyName = newName;
}
```

Note: For get Accessor return ~~statement~~ statement is mandatory.

PAGE No.
DATE / /

c) Methods :

- Methods define the actions to perform
- All method specifications are similar to functions

FAQ: What is difference between Methods, Functions, Procedures

Ans:
- Functions are intended to return value
- Methods are not intended to return value
- Procedures may or may not return value. Changes according to situation

In JS procedures are not present.

Syntax:

```
class className {

    methodName() {
        // Logic
    }
}

let obj = new className();
obj.methodName();
```

d) Constructors:

- It is special type of method in class
- It is used for instantiation (Creation of object)
- Constructor is a subroutine that executes automatically for every object
- It initializes memory to store values, at the time of allocating memory for class
- In Javascript constructor is anonymous [without name]
- Class name is used for constructor implisitly
- It is defined by using "constructor" keyword
- Every class have default constructor
- You can configure explicit constructor to define explicit actions.

Syntax:

```
class className
{
    constructor ()
    {
    }
}
```

let obj1 = new className ;

Constructor is refered implicitly

let obj2 = new className () ;

Constructor is refered explicitly by using ()

FAQ.: If '()' mandatory beside class name?
Ans: No. It is required only when constructor is parameterized.

- Java Script class can't overload constructor
- Java Script class can't have private constructor.
- Java Script class can't have static constructor.
- Java Script class can't have constructor polymorphism.

- A derived class constructor must have super call

Note:

According to rules of OOP, if classes are configure with relationship then a derived class constructor must call base class constructor

In other OOP languages it is done implicitly. But in JS you need explicit calling

PAGE No.
DATE / /

* Code Reusability

- It is one of the key feature of OOP
- You can configure code reusability by using 2 techniques
  a) Aggregation
  b) Inheritance

a) Aggregation :

- It is a techinique used to access the code of one class in another by using instance of class
- It is refered as "Has-A Relation".
- Without configuring any relation beth classes you can access the members of one one class in another class
- It is known as "Object-to-Object" Communication

PAGE No.
DATE / /

* Inheritance :

- In this technique a class can extend another class.
- You can configure relation between classes
- Members of base classes are accessible in derived class by using "super" keyword.
- Without creating instance of base class you can access its members using "super" keyword
- Inheritance is code extensibility.
- This is refered as "Is-A-Relation".

PAGE No.
DATE / /

* Types of inheritance

1) Single inheritance:
- A base class is extended by derived class

2) Multilevel inheritance:
- Derived class is extented by another derived class

3) Multiple inheritance:
- Single derived class implements multiple super classes
- It is not supported by JS
- It is supported by interfaces in OOP
- But JS does n't have interface

FAQ: Why multiple inheritance is not supported for classes ?
Ans: Because of Constructor dead lock.

Deadlock is a situation where every constructor waits for each other

Interface in OOP supports multiple inheritance as it doesn't have constructor.