Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus Division
**	Exponent [ES6] – Math.pow(2,4) = 16
++	Increment x++ [x = x + 1]
--	Decrement x-- [x = x – 1]

- Exponent “**” is new to JavaScript, the early versions use “Math.pow()”
  Syntax:
  2**4 = 16
  Math.pow(2,4) = 16
- **Post increment and decrement**
  It is a mechanism where the value is assigned to reference and then incremented or decremented.
  Ex:
  let x:number = 10;
  let y:number = x++;

```
console.log(`x=${x}\ny=${y}`);
```

```
x=11
y=10
```

Ex:

```
let x:number = 10;
let y:number = x--;
console.log(`x=${x}\ny=${y}`);
```

```
x=9
y=10
```

- **Pre-Increment and decrement**
  It is a mechanism where the value is assigned to reference after increment or decrement.
  Ex:

```
let x:number = 10;
let y:number = ++x;
console.log(`x=${x}\ny=${y}`);
```

```
x=11
y=11
```

Ex:

```
let x:number = 10;
let y:number = --x;
console.log(`x=${x}\ny=${y}`);
```

```
x=9
y=9
```

Assignment Operators

Operator	Description

X=Y	Assignment
	
	Ex:
	let a:number = 10;
	let b:number = 20;
	let c:number = 30;
	let x:number = (c=(a=b));
	console.log(`x=${x}`);

X+=Y	Addition Assignment X= X + Y;
X-=Y	Subtraction Assignment X = X – Y;
X*=Y	Multiplication Assignment

	X = X * Y;
X/=y	Division Assignment
X%=Y	Remainder Assignment
X**=Y	Exponent Assignment X = X ** Y;

Comparison Operators

Operator	Description
==	Equal
!=	Not Equal
===	Strict Equal
!==	Strict Not Equal
>	Greater than
>=	Greater than or Equal
<	Less than
<=	Less than or equal

=== It can compare only the values of same type.

== It can compare values of different types.

Logical Operators

Operator	Description
&&	Logical AND

||	Logical OR
!	Logical NOT

Special Operators:

Operator	Description

typeof	It returns the data type of variable or property
	
	Ex:
	let Product:any = {
	Name: "TV",
	Price: 45000.55,
	Stock: true,
	ShippedTo: ["Delhi", "Hyd"]
	}
	console.log(`Name is ${typeof Product.Name}\nPrice is ${typeof Product.Price}\nStock is ${typeof Product.Stock}\nShipped To is ${typeof Product.ShippedTo}`);

instanceof	It is a boolean operator that returns true if the specified object is derived from the

	given class.
	
	Ex:
	class Employee
	{
	
	}
	let pic = new Image();
	let products = new Array();
	let emp = new Employee();
	
	console.log(`Pic is derived from Employee : ${pic instanceof Employee}`);
	console.log(`Pic is derived from Image : ${pic instanceof Image}`);
	console.log(`Products is dreived from Object : ${products instanceof Object}`);

new	It is dynamic memory allocating operator. It allocates memory dynamically for any specific class and loads its members into memory. Ex: let pic = new Image();

	pic.src; pic.width; let products = new Array(); products.length; products.push()

delete	It is used to delete any property from object.
	You can’t delete read-only properties.
	
	Ex:
	let product:any = {
	Name: "Samsung TV",
	Price: 45000.44
	}
	delete Math.PI; // Invalid - Read-Only
	delete product.Price;
	if(product.Price==undefined) {
	console.log("Price not Found");
	} else {
	console.log(`Name=${product.Name}\nPrice=${product.Price}`);
	}

in	It is used to verify a property from object and return true if the property is available.

	Ex:
	let product:any = {
	Name: "Samsung TV",
	Price: 45000.44
	}
	delete product.Price;
	console.log("Price" in product);
	console.log("Name" in product);
	
	Ex: for..in – to create an iterator for properties
	
	let product:any = {
	Name: "Samsung TV",
	Price: 45000.44
	}
	for(var property in product)
	{
	console.log(property);
	}

of	It is used to create an iterator for all values in collection. Ex: for..of
?:	It is a ternary operator

	Ex: (condition)?statement_if_true:statement_if_false

Statements in TypeScript

- Statements are used to control the program execution flow.
- TypeScript can use all JavaScript statements

Statement Type	Keywords
Selection Statements	If, else, switch, case, default
Looping Control	for, while, do while
Iteration Statements	for..in, for..of
Jump Statements	break, continue, return
Exception Handling	try, catch, throw, finally

Exception Handling Statements

- Errors in computer programming are classified into 2 groups

    - Compile Time Errors
    - Run Time Errors
- **Compile Time Errors:** A compiler is unable to understand the syntax or keywords or the conversion technique that you are using in program. It identifies the problems and reports them during compile time.
- **Run Time Errors:** A compile is unable to understand the instructions given to program dynamically during runtime, due that compiler terminates the application. Exception handling is required to avoid "**Abnormal Termination**".
- Exception handling is not mandatory for client-side application, as it depends of error handling. JavaScript and TypeScript can't handle the exceptions implicitly. You have to explicitly configure the exceptions.
- Exception handling statements
    - try : It is monitoring block, contains statements to execute.
    - catch : It is handler block, that can catch the exception if any statement fails to

execute.

- throw : It is used to throw any exception.
- finally : It comprises of statements executed always.

Ex:

```
try {
   var x:number = 10;
   var y:number = 2;
   if(y==0){
      throw "Can't divide by zero";
   }
   if(y>10) {
      throw "Number too large..";
   }
   if(y<0) {
      throw "Can't use -ve values";
   }
   var z = x/y;
}
```

```
catch(ex){
    console.log(ex);
}
finally {
   if(z!==undefined) {
    console.log(`z=${z}`);
   }
}
```