Pakistan Blockchain Institute

MODULE-1

JAVASCRIPT CRASH COURSE

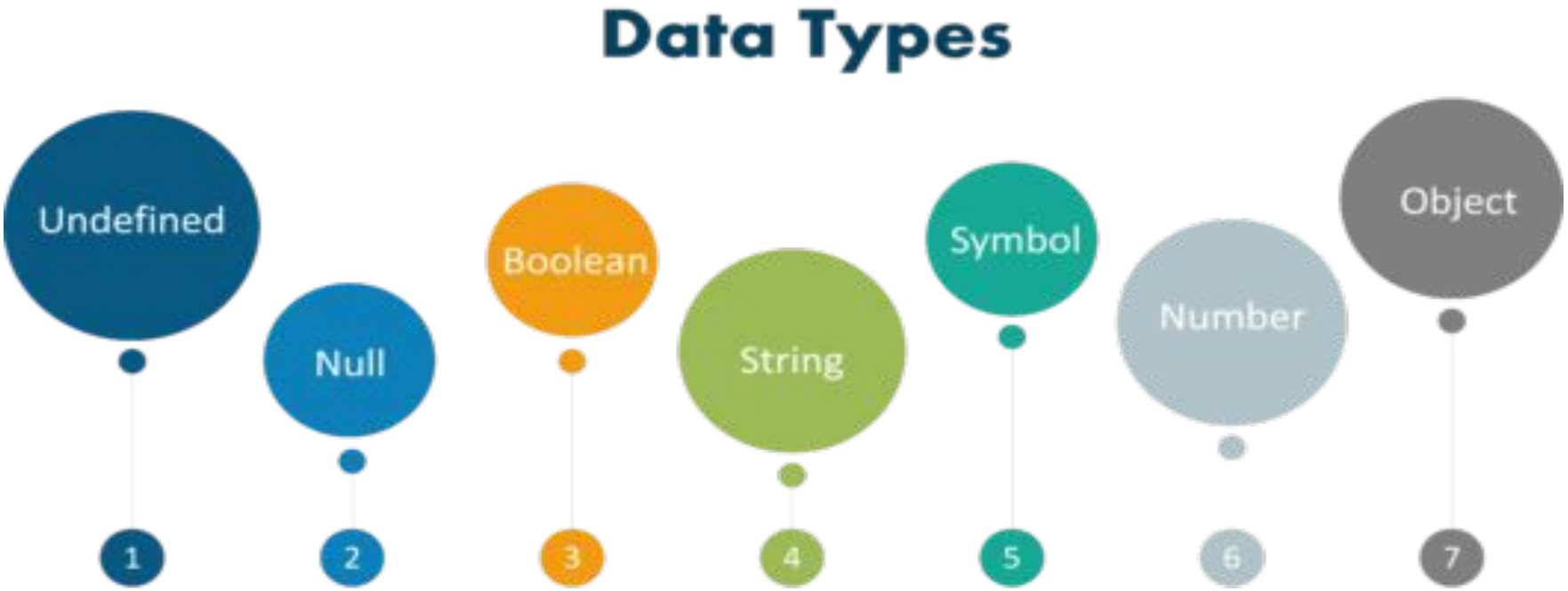
Class-2

Edversity.

Raja Rizwan Saleem
Lead Blockchain Trainer

Data Types

Data Types


Data Types
Undefined
Null
Boolean
String
Symbol
Number
Object
1
2
3
4
5
6
7

Data Types

Datatypes hold different values.
There are two types of datatypes in

JavaScript: Primitive and Non-Primitive.

Primitive defines immutable valuesand was introduced recently by ECMAScript standard.

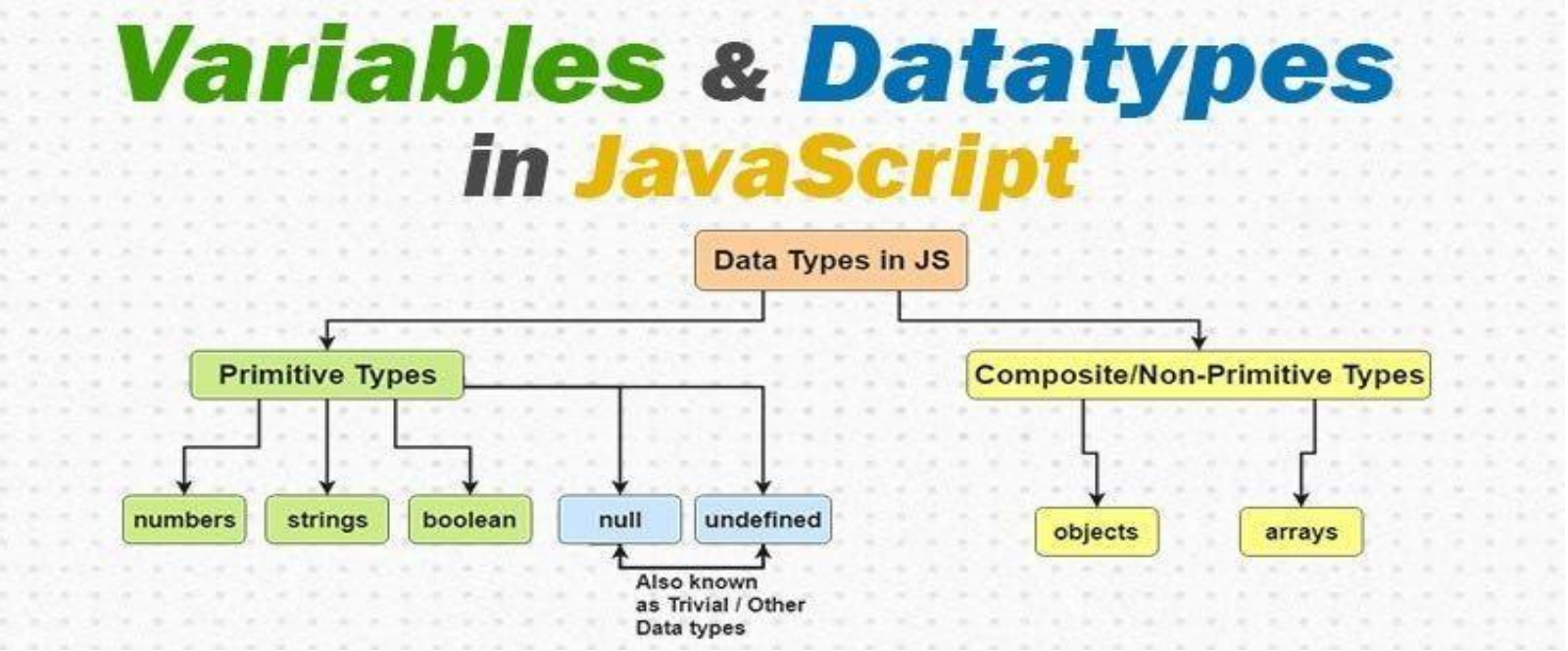
Data Types

1. Variables contains data
2. Each data has type, that’s why we call it data type
3. Variables can hold numbers like 100 and text values like "John Doe"
4. In programming, text values are called text strings.

Data Types

1. Six types are considered to be primitives.
    a. Number -- integers, floats etc
    b. String -- an array of characters
    c. Boolean -- true or false
    d. Null -- No Value
    e. Undefined --a declared variable but hasn’t been given a value
    f. Symbol -- a unique value that's not equal to any other value

Data Types


Variables & Datatypes
in JavaScript
Data Types in JS
Primitive Types
Composite/Non-Primitive Types
numbers
strings
boolean
null
undefined
objects
arrays
Also known as Trivial / Other Data types

Data Types

1. Complex Types
    a. Objects
    b. Functions

Function is callable object that executes a block of code, it lies under the object type.

Non-primitive data types is the object. The JavaScript arrays and functions are also objects.

Variable for String

The next basic data type is the string. Strings are used to represent text. They are written by enclosing their content in quotes.
To store text in variable we use strings

```
var name = "Rizwan";
name = "Saleem";;
```

Variable for String

Strings

- A string variable can store a sequence of alphanumeric characters, spaces and special characters.
- Each character is represented using 16 bit
  — You can store Chinese characters in a string.
- A string can be enclosed by a pair of single quotes (') or double quote (").
- Use escaped character sequence to represent special character (e.g.: `\"`, `\n`, `\t`)

String - Single quotes and double quotes

1. Text strings are always enclosed in quotes
2. You can use single or double quotes

```
var name = "Mansoor bhai";
var nationality = 'Pakistani';
var message = "What is your father's number";
consol.log("This is the first line\nAnd this
is the second");
```

Variable for Numbers

1. To store whole numbers or decimal values we use number data type

```
var age = 25;

var weight = 150.5;
var newWeight = weight +
20;  var originalNumber =

12 + ;
```

Variable for Boolean

It is often useful to have a value that distinguishes between only two possibilities, like "yes" and "no" or "on" and "off". For this purpose, JavaScript has a Boolean type, which has just two values, true and false, which are written as
those words.

To store true/false boolean data type:

```
var isFeePaid = true;

var examPassed = false;
```

Undefined

1. The undefined data type can only have one value-the special value *undefined*
2. If a variable has been declared, but has not been assigned a value, has the value undefined

```
var name;
var age;
```

Undefined

1. You can be empty a variable by setting the value to undefined. The type will also be undefined

```
var name = "Rizwan";
name = undefined;
```

2. Now this variable contains no value, and it will cause error if we try to apply some operation on it

Null

1. This is another special data type that can have only one value-the null value.
2. A null value means that there is no value.
3. It is not equivalent to an empty string ("") or 0, it is simply nothing

```
var name = null;
var nationality = "Basit";
nationality = null;
```

Difference between null and undefined

1. undefined means a variable has been declared but has not yet been assigned a value.
2. null is an assignment value. It can be assigned to a variable as a representation of no value
3. undefined and null are two distinct types: undefined is a type itself (undefined) while null is an object.
4. Unassigned variables are initialized by JavaScript with a default value of undefined. JavaScript never sets a value to null. That must be done programmatically.

Difference between null and undefined

When we define a variable to *undefined* then we are trying to convey that the variable does not exist .
When we define a variable to *null* then we are trying to convey that the variable is empty.

JavaScript Data Types are Dynamic

1. JavaScript has dynamic types. This means that the same variable can be used to hold different data types
2. You can assign any type of value in variable any time and data type of variable will be changed

```
var name = "Basit";      It's String
/name = 25;    /   Now changed to Number
name = true;       Now changed to Boolean
/
```

typeof Operator

1. You can use the JavaScript typeof operator to find the type of a JavaScript variable.
2. The typeof operator returns the type of a variable or an expression

```
typeof "Hello"// Returns "string" var
a = 45;

typeof a;              // Returns "number"
typeof true;           // Returns "boolean"
```

Statement and Expression

Statements

1. A computer program is a list of "instructions" to be "executed" by a computer.
2. In a programming language, these programming instructions are called statements.
3. A JavaScript program is a list of programming statements.
4. A statement can set a variable equal to a value.
5. A statement can also be a function call, i.e. document.write().

Statements

1. Statements define what the script will do and how it will be done.
2. Most JavaScript programs contain many JavaScript statements.
3. The statements are executed, one by one, in the same order as they are written.

Statements

1. Statements define what the script will do and how it will be done.
2. Most JavaScript programs contain many JavaScript statements.
3. The statements are executed, one by one, in the same order as they are written.

Statements

1. Each line below is a statement

```
var a = 4;              // Statement 1
var b = 2;              // Statement 2
var c = 0;              // Statement 3
c = a + b;              // Statement 4
alert(a);               // Statement 4
console.log(c);         // Statement 4
```

End of Statement with semicolon ;

1. To end statement add semicolon at the end of each executable statement
2. Semicolons separate JavaScript statements.

```
var a = 4;
```

3. When separated by semicolons, multiple statements on one line are allowed

```
i = 3; j = 5; k = i + j;

alert(i);console.log(k);
```

End of Statement without semicolon ;

1. Typically we end statements with semicolon but JavaScript semicolon is optional, but is highly recommended to end with semicolon
2. The end of a statement is most often marked by pressing enter and starting a new line.

End of Statement without semicolon ;

```
var a = 5                    // New line will end statement
a * 4
alert(a)
console.log(a)


var a = 5a * 4               // Error, Will Not work
alert(a)console.log(a        // Error, Will not work
)
```

Expressions

1. An expression is a combination of values, variables, function calls and operators, which computes to a value.
2.

```
a + b;                                  // expression

                                        // expression

4 /  2;                                 // expression

var  a = 5;                             // expression
```

Comments

Comments

1. Comments are for the human, not the machine.
2. They help you and others understand your code when it comes time to revise, they make code more readable.
3. You can also use commenting to comment out portions of your code for testing and debugging.
4. They will prevent execution of code
5. There are two ways mark text as a comment, single line comments and multi-line comments

Single line comments

1. Single line comments start with //.
2. Any text between / and the end of the line will be ignored (will not be executed).

```
// Declare and initialize variable
var a = 6; //This is comment


//This below code will not execute
//var b = 8;
```

Multi-line comments

1. Multi-line comments start with /* and end with */.
2. Any text between /* and */ will be ignored.

```
/*
This code declared and
initialize  variable a and show
on screen

*/
var a =
6;
```

Variable Names

Variable Names Legal and Illegal

1. A variable name can't contain any spaces
2. A variable name can contain only letters, numbers, dollar signs, and underscores.
3. The first character must be a letter, or an underscore (_), or a dollar sign ($).
4. Subsequent characters may be letters, digits, underscores, or dollar signs.
5. Numbers are not allowed as the first character of variable.

Variable Names Legal and Illegal

1. Legal names:

```
var hello = 56;
var _xyz = 44;
var $work = 90;
var user2 = 56;
var i_info = 99;
var my$wor = 77;
    k
```

Variable Names Legal and Illegal

1. Illegal names:

```
var 2user = 12; // Can't start with number

var my user = 23; // Can't contains space

var hello#world = 34;

var   my-info =

44;   var my?info

=    45;   var

my*info = 45;
```

Reserved Keywords

1. Reserved Keywords cannot be used as variable name
2. Here are few reserved keywords

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Case Sensitive

1. Variable names are case sensitive.
2. So **rose** and **Rose** are two different variables

```
var rose = "Hello";

var Rose = "Hello";

alert(rose);

alert(Rose);

alert(ROSE); //

Error
```

Camel Case

1. If there's more than one word in the variable name, then it is recommended to use camel case
2. A camelCase name begins in lower case. If there's more than one word in the name, each subsequent word gets an initial cap, creating a hump.

Camel Case

```
var userResponse

var userResponseTime
var userResponseTimeLimit

var response
```

This Style of naming a variable is called camel case

Operators

Arithmetic Operators

```
var a = 5;
var b = 3;
var c = a + b;          // Addition, result 8
var d = a - b;          // Subtraction, result 2
var e = a * b;          // Multiplication, result 15
var f = a / b;          // Division, result 1.66
var g = a % b;          // Modulus, result 2
var h = a ** b;         // Exponentiation, result 125
```

Arithmetic Operators

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

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)
/	Division
%	Modulus (Remainder)
++	Increment
--	Decrement

Arithmetic Operators

Operator	Name	Purpose	Example
`+`	Addition	Adds two numbers together.	`6 + 9`
`-`	Subtraction	Subtracts the right number from the left.	`20 - 15`
`*`	Multiplication	Multiplies two numbers together.	`3 * 7`
`/`	Division	Divides the left number by the right.	`10 / 5`
`%`	Remainder (sometimes called modulo)	Returns the remainder left over after you've divided the left number into a number of integer portions equal to the right number.	`8 % 3` (returns 2, as three goes into 8 twice, leaving 2 left over).
`**`	Exponent	Raises a `base` number to the `exponent` power, that is, the `base` number multiplied by itself, `exponent` times. It was first introduced in ECMAScript 2016.	`5 ** 2` (returns `25`, which is the same as `5 * 5`).

Assignment Operators

1. Assignment operator assign value to variables
2. When you need to apply arithmetic operation and assign value to same variable then you can also

```
var a = 5; // equals = is assignment operator
a = a + 2; // Assign 7 in variable a
OR
var a = 5;
a+=2;      // Assign 7 in variable a
```

Assignment Operators (cover in upcoming slides

Example	Same as
`a += 5;`	`a = a + 5;`
`a -= 5;`	`a = a - 5;`
`a *= 5;`	`a = a * 5;`
`a /= 5;`	`a = a / 5;`
`a %= 5;`	`a = a % 5;`
`a **= 5;`	`a = a ** 5;`

Assignment Operators

Assignment operators assign values to JavaScript variables.

Operator	Example	Same As
=	x = y	x = y
+=	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
<<=	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
|=	x |= y	x = x | y
**=	x **= y	x = x ** y

Eliminating ambiguity -- BODMAS

1. Complex arithmetic expressions can pose a problem when there are multiple operators in single expression

```
var a = 5 + 2 * 3 - 2 / 2; // result 10
```

2. The evaluation of above expression is depends on BODMAS rule

Eliminating ambiguity -- BODMAS

The **BODMAS rule** states we should calculate the Brackets first (2 + 4 = 6), then the Orders (5^2 = 25), then any Division or Multiplication (3 x 6 (the answer to the brackets) = 18), and finally any Addition or Subtraction (18 + 25 = 43).

Eliminating ambiguity -- BODMAS

B Brackets first
O Orders (i.e. Powers and Square Roots, etc.)
DM Division and Multiplication

(left-to-right) AS Addition and Subtraction

(left-to-right)

Eliminating ambiguity -- BODMAS

```
var a = 5 + 2 * (3 - 2) / 2; // result 6
```

1.3 - 2 with brackets will be evaluated first, result 1

2.2 * result of (3 - 2) so 2 * 1, result 2

3.Result of 2 * (3 -2) divide by 2 so 2 / 2, result 1 4. 5 + result of 2 * (3 - 2) / 2, so 5 + 1, result 6

Eliminating ambiguity -- BODMAS

```
var a = 3 + 5 * 2;                  // result 13
var b = 8 / 2 - 1;                  // result 3
var c = 3 % 2 + 4  - 1;             // result 4
```

Thanks

End of (Module-1) Lecture-2