Lesson:

Variable and its type

PW SKILLS

Topics Covered

1. Introduction to variables.
2. Creation of variables.
3. Naming variables in Javascript.
4. Assigning values to variables.

Introduction to variables.

Variables are like containers, they are used to hold the information we'll need when programming. Variables store data of any data type that can be used throughout a program.

Variable means anything that can vary. Variables hold the data value and it can be changed anytime we want.

Creating a variable is also called declaring a variable. There are three ways to create a variable in JavaScript.

1. var keyword.
2. let keyword.
3. const keyword.

```Javascript
var name = "PW Skills";
let name = "PW Skills";
const name = "PW Skills";
```

Creation of variables.

A) var

When we declare variables using the var keyword, it can be accessed within the function or globally, hence called function scoped (scope defines boundaries in the program where any variable can be accessed). We will discuss scope in detail in the next lesson.

Example 1:

```Javascript
// index.js

var fullname = "Mang Touthang";
console.log(fullname);

//Output - Mang Touthang
```

Example 2:

```Javascript
// index.js

//function block
function getFullName(){
   console.log(fullname); // 1st console
   var fullname = "Mang Touthang";
   console.log(fullname); //2nd console
}

getFullName()
console.log(fullname);  //3rd console
```

```
/*Output
undefined
Mang Touthang
Uncaught ReferenceError: fullname is not defined
*/
```

In the above example there are three console logs, let's talk about each console output one by one.

1. The 2nd console output is the value of variable **fullname** and that is quite reasonable to see.
2. But the 1st console outputs **undefined**, why? It's because, we are trying to access it before the variable initialization and javascript initializes **var** variables with value **undefined.** This concept is called hoisting in Javascript.
3. Now, let's talk on 3rd console, it throws **ReferenceError**, because as per definition, v**ar** variables are function scoped and accessible only within the function where they are defined.

B) let

when we declare a variable using the **let** keyword, it will be accessed only within the block in which it is declared, hence called block scoped (scope defines boundaries in the program where any variable can be accessed)..

Example 1:

```
Javascript
// index.js

let age = 23;
console.log(age);
//Output - 23
```

Example 2:

```
Javascript
// index.js

//block
{
   console.log(age); //1st console
   let age= 23;
   console.log(age); //2nd console
}

console.log(age); // 3rd console

/*Output
23
Uncaught ReferenceError: age is not defined
*/
```

In the above example there are three console logs, let's talk about each console output one by one.

1. The 2nd console output is the value of variable **age** and that is quite reasonable to see.
2. But the 1st console throws a Reference error, why not **undefined** like var? It's because, we are trying to access it before the variable initialization and unlike var, javascript does not initialize let variables with undefined.
3. Now, let's talk on the 3rd console, it also throws a **ReferenceError**, because as per definition, let variables are block scoped and accessible only within the block where they are defined.

C) const

Using const declaration, we define constants (which does not change later), and they are like **let, const**

variables are also block scoped.

Example :

```Javascript
// index.js

const age = 23;
console.log(age);

age = 24;

/*Output - 23
Uncaught TypeError: Assignment to constant
*/
```

If we see the above example, when we tried to reinitialize the **age**, it threw an Type Error, because unlike **var** and **let**, we cannot reinitialize **const** variables.

Another example,

```Javascript
// index.js

const alex = { name:"alex", age:23 };
console.log(alex);

alex.name = "mang";  // No Error
console.log(alex);

alex = {name:"nasik",age:33}  // Error
console.log(alex);

/*Output -
{ name:"alex", age:23 }
{ name:"mang", age:23 }
Uncaught TypeError: Assignment to constant
*/
```

In above example, we have defined an const variable alex and assigned a object to it, we have tried to explain const variable in case of object,

- **Case 1:** When we try to modify alex object properties it won't throw errors, even alex is a const variable, because alex variable is only a reference to an object not the actual object.

- **Case 2:** And when we tried to modify the alex variable with another object, we are attempting to change the reference variable itself, so it will throw an TypeError.

Similarly, with arrays also, we can modify values within arrays, but we cannot change const variables having reference to array.

Example,

```Javascript
// index.js

const nums = [1, 2, 3, 4, 5];
console.log(nums);
```

```
nums[1] = 0;  // No Error
console.log(nums);

nums = [1, 2, 3, 4, 5]; // Error

/*Output -
[1, 2, 3, 4, 5]
[1, 0, 3, 4, 5]
Uncaught TypeError: Assignment to constant
*/
```

Difference between var, let and const declaration

Here are listed differences between var, let and const variables.

var	let	const
Function Scoped: Variables accessible within the function block in which they are defined.	**Block Scoped:** Variables accessible within the block in which they are defined.	**Block Scoped:** Variables accessible within the block in which they are defined.
Initialized with value **undefined**	UnInitialized	UnInitialized
We can reassign values to variables.	We can reassign values to variables.	We cannot reassign values to variables.

Naming variables in JavaScript

When naming the variables, we must consider making the names descriptive and easily understandable. This will make our program easy to read and understand in the future when we have to refactor it.

Here are some rules one should look out for when naming variables:

- Variable names should begin with either a letter or an underscore or a dollar sign.

```
Javascript
var name = "PW SKills";
var Name = "PW SKills";
var _name = "PW SKills";
var $name = "PW SKills";
```

- Variable names should not begin with numbers or special characters except the underscore and dollar signs.
- Keywords are reserved words that have a specific meaning and cannot be used as variables. Keywords like if, else, for should not be used as variable names.
- Variable names are case-sensitive. That means name and Name are different variable names.

To ensure consistency in naming variables adopt one of the following naming conventions in naming variables. Developers mostly prefer **camel case** naming.

```
Javascript
var companyName = "PW Skills"; // Camel Case
var CompanyName = "PW Skills"; // Pascal Case
var company_name = "PW SKills"; // Snake Case
```

Assigning values to a variable.

Storing data in a variable is also called assigning a value to a variable. To store data in a variable(assign value to a variable), use the **= symbol**. Place the variable name on the left side of the = symbol and place the value to store in the variable goes on the right side of the = symbol.

The = symbol is called the assignment operator. We will look into operators in depth in further lectures.

Variables can be created before assigning values to them.

```
Javascript
var name;
name = "PW Skills";
```

Whenever we create a variable without assigning a value to it, by default javascript stores undefined [absence of the value].

Values can also be assigned to variables at the moment of creating them. Creating variables and assigning values to them at the same time is known as initializing a variable.

```
Javascript
var name = "PW Skills";
var students = 12345678;
var enrolledToFSWD = true;
var name = "PW Skills", students = 12345678, enrolledToFSWD = true;
```