# JavaScript

## Variables and Operators

Pooja Godse
&
ALK Bilahari

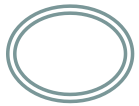
Procareer Academy

# Outline

- Identifiers – naming convention
- var , let, const
- Operators – Arithmetic , Assignment , Logical ,Relational and Conditional

Procareer Academy

# Identifiers-naming convention



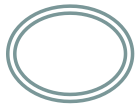
- Identifiers are names given to variables, functions, etc.
  Example: firstName, placeOfVisit, calculateGPA()
- Identifiers must begin with -
  - A letter (A-Z or a-z)
  - A dollar sign ($)
  - Or an underscore (_)
- Numbers are not allowed as the first characters.
- Hyphens are not allowed
- All JavaScript identifiers are case sensitive.

Procareer Academy

# Variables in JavaScript

Procareer Academy

# var

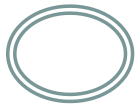


- Used to hold data that may vary
- Can redeclare variables with same name
- For variables declared inside function , Scope is ‘Function scope’
- For variables declared outside any function, Scope is ‘ Global Scope’

Note: Scope - Scope essentially means where these variables are available for use

Procareer Academy

# Var - examples

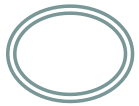


1. ***var*** *name = "William";*
2. *console.log("Welcome to JS course, Mr." + name);*
3. ***var*** *name = 123; /* Here, even though we have redeclared the same identifier, it will not throw any error.*/*
4. *console.log(name);*

Note: the multiline comments /* ---*/

Procareer Academy

# let



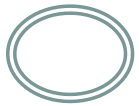
- Declared using let keyword
- It has a block scope
- It can’t be re-declared but can be re-assigned

Example:

```
1.  let name="William";
2.  console.log("Welcome to JS course, ." + name);
3.   let name = "Goth"; // throws an error
4.  console.log("Welcome to JS course, Mr.“ + name);
```

Procareer Academy

# const



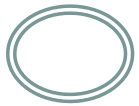
- value cannot be changed
- Cannot be re-declared
- Has a ‘block scope’

Example:

1. *const pi =3.14;*
2. *console.log("The value of Pi is: "+pi);*
3. *pi= 3.141592; // This will throw an error*
4. *console.log("The value of Pi is: "+pi);*

Procareer Academy

# Quick look at Variables



| Keyword | Scope | Re-declaration | Re-Assignment |
|---|---|---|---|
| let | Block | Not allowed | Allowed |
| const | Block | Not allowed | Not allowed |
| var | Function | Allowed | Allowed |

Procareer Academy

# Variable Hoisting

# Hoisting of var

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

This -

1. *console.log (greeter);*
2. *var greeter = "say hello“*

is interpreted as this:

1. *var greeter;*
2. *console.log(greeter); // greeter is undefined*
3. *greeter = "say hello"*

Procareer Academy

# Interesting Fact about Var

```
1. var greeter = "hey there";
2. var times = 4;
3. if (times > 3) {
4.     var greeter = "say Hi instead";
5. }
6. console.log(greeter) // "say Hi instead“
```

While this is not a problem if we knowingly want greeter to be redefined, it becomes a problem when you do not realize that a variable greeter has already been defined before.

Procareer Academy

# Let works fine

variable declared in a block with let  is only available for use within that block

```
1.  let greeting = "say Hey";
2.      let times = 4;


3.      if (times > 3) {
4.          let hi = "say Hi instead";
5.          console.log(hi);// "say Hi instead"
6.      }
7.      console.log(hi) // hi is not defined
```

Procareer Academy

# Advantage with let

If the same variable is defined in different scopes using let , there will be no error

```
let greeting = "say Hey";
    if (true) {
        let greeting = "say Hello instead";
        console.log(greeting); // "say Hello instead"
    }
    console.log(greeting); // "say Hey”
```

This is because both instances are treated as different variables since they have different scopes.

Procareer Academy

# Types of Data

- refers to the type of value assigned to a variable
- JS s a loosely or dynamically typed

Example:

1. *let age = 24; //Number*
2. *let name = "Tom" //String*
3. *let qualified = true; //Boolean*

Procareer Academy

# Types of Data

We can have the same variable store different types as and when needed

Example:

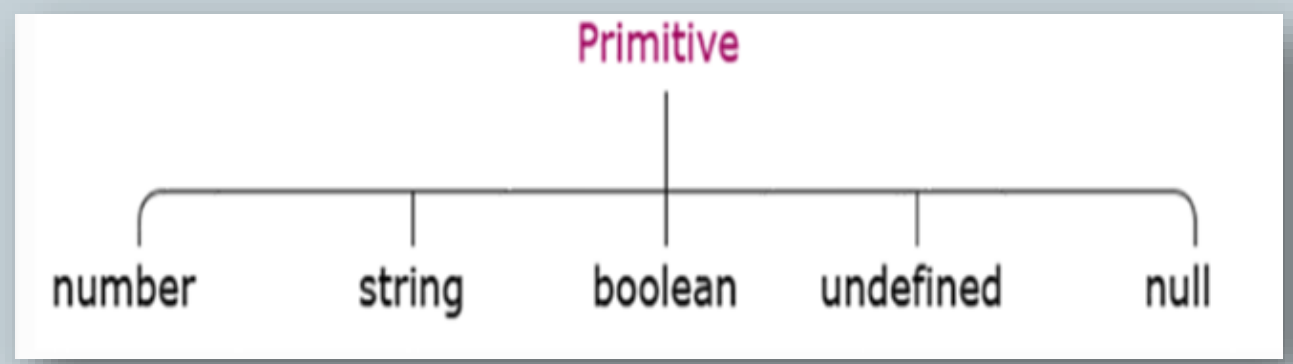
1. *let age = 24*
2. *age = "Twenty-Four“ //no error*

Procareer Academy

# Types of data

# Types of Data

Unlike few languages , JS does not expect you to mention the datatype of the variable. There are about 5 main types of data :

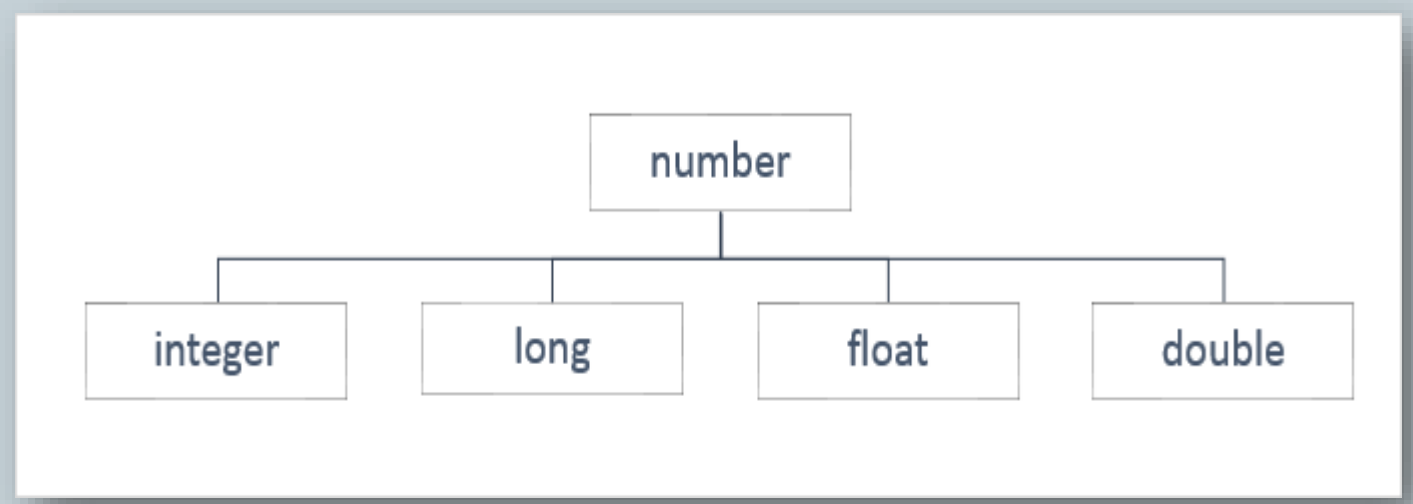



The data is said to be primitive if it contains an individual value.

Procareer Academy

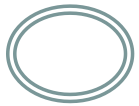
# Number

- Used to store a variable that holds a numeric value
- In almost all the programming languages a number data type gets classified as shown below:




Procareer Academy

# Number as value



1. *let pi = 3.14; // its value is 3.14*
2. *let smallestNaturalNumber = 0; // its value is 0*

Any other value that does not belong to the above-mentioned types is not considered as a legal number. Such values are represented as NaN (Not-a-Number).

1. *let result = 0/0; // its value is NaN*
2. *let result = "Ten" * 5; //its value is NaN*

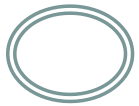
# String as value

- Used to store a textual value
- String values are written in quotes, either single or double.

*Example:*

1. *let personName= "Jane";  //OR*
2. *let personName = 'Jane';  // both will have its value as Jane*

# String as value



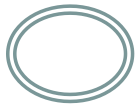
1. *let ownership= "Jane's";  //Jane's*
2. *let ownership = 'Jane"s';  // Jane"s*

This will be interpreted as Jane's and Jane"s  respectively.

1. *let ownership= "Jane"s";  //error*
2. *let ownership = 'Jane's';  //error*

Procareer Academy

# Boolean as value



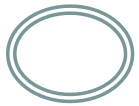
- Used to store a logical value i.e; true or false
- Values such as 100, -5, “Cat”, 10<20, 1, 10*20+30, etc. evaluates to true
- Whereas 0, “”, NaN, undefined, null, etc. evaluate to false
- Everything with a  value evaluates to True and without a value evaluates to False

Example:

*let a = true;*

Procareer Academy

# Undefined



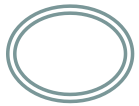
- Refers to "no value“
- undefined is assigned to indicate the absence of a value by default.

  *let custName; //here value has not been assigned , hence undefined*

- variable can be made empty by assigning the value undefined ( can be, but not recommended)
  1. *let custName = "John"; custName = undefined;*
  2. *let deptName; //deptName is now undefined*

Procareer Academy

# null



- null value represents "no object"

*let item = null;  // item is intended to be assigned with object later. Hence null is assigned during variable declaration.*

- Can check if it is pointing to a valid object or null.

    *document.write(item==null);*

- Note: 'document' is an object that represents the HTML document rendered on the browser window and write() method helps one to populate HTML expressions to the document.

# Recap

- Overview of JS
- Variables- var , let and const
- Data Types - Number , String , Boolean, null , undefined

Procareer Academy

# Operators

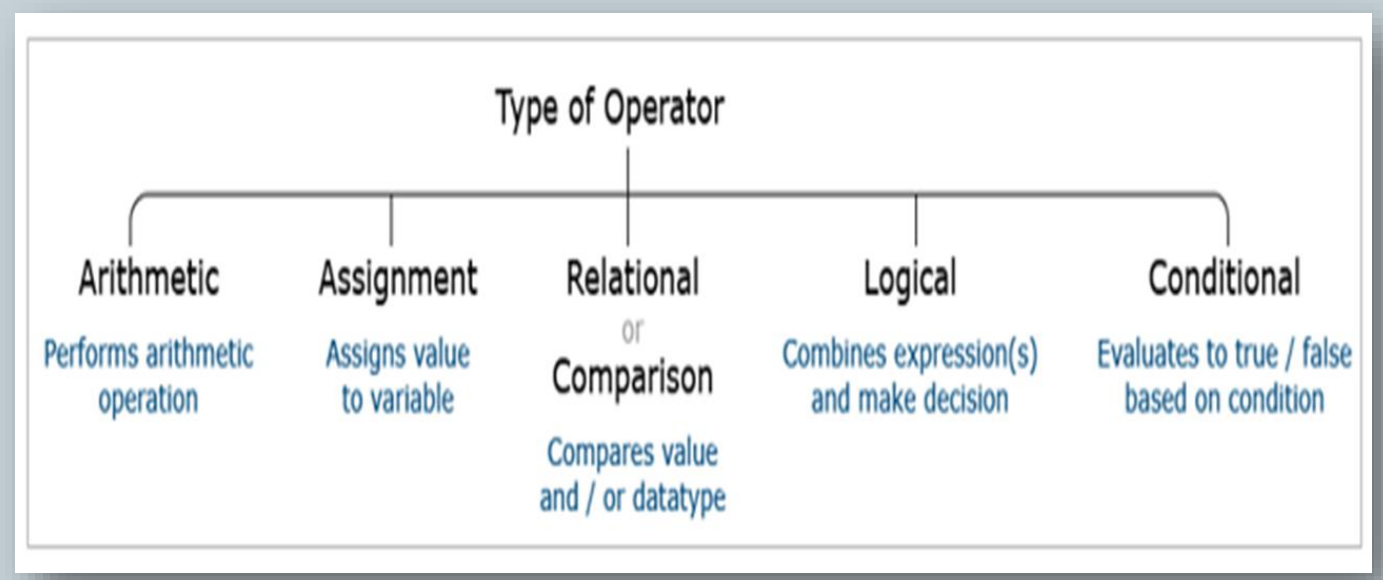
- Operators are the **symbols** used to **perform operations** on the values
- **Operands:** Represents the data
- **Operator:** which performs certain operations on the operands

Example:

*let sum = 5 + 10;*

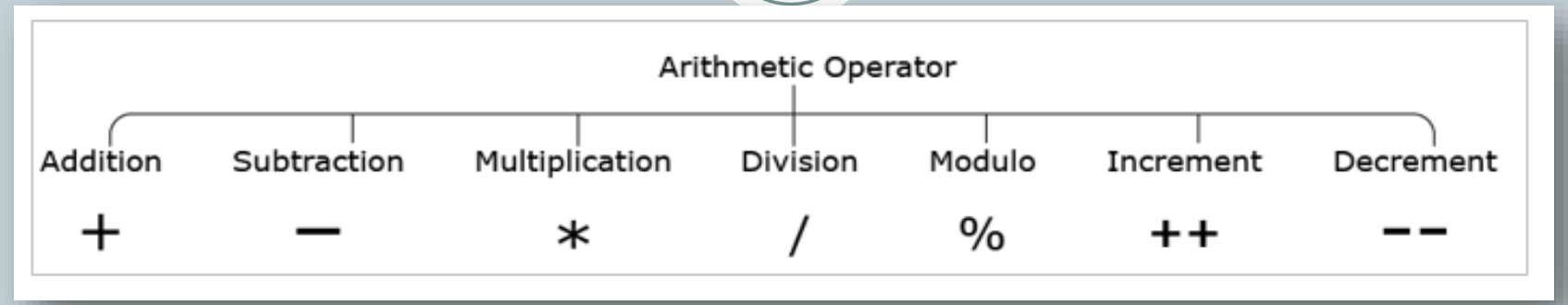
- Operators are categorized into **unary**, **binary**, and **ternary** based on the **number of operands** on which they operate in an expression.
- Based on the operation they perform they have a different set of categories

# Types of Operators




Procareer Academy

# Arithmetic Operators




```
1. let sum = 5 + 3; // sum=8
2. let difference = 5 - 3; // difference=2
3. let product = 5 * 3; // product=15
4. let division = 5/3; // division=1
5. let mod = 5%3; // mod=2
6. let value = 5;
7. value++; // increment by 1, value=6
8. let value = 10;
9. value--; // decrement by 1, value=9
```

Procareer Academy

# Arithmetic Operators : String Type Operands

Arithmetic operator '+' when used with string type results in the concatenation.

```
1. let firstName = "James";
2. let lastName = "Roche";
3. let name = firstName + " " + lastName;  // name = James Roche
```

Procareer Academy

# Arithmetic Operators : String Type Operands

Arithmetic operator '+' when used with a string value and a numeric value, it results in a new string value.

```
1. let strValue="James";
2. let numValue=10;
3. let newStrValue= strValue + " " + numValue;  // newStrValue= James 10
```

Procareer Academy

# Arithmetic Operators

1. *1 + 2 // => 3: addition*
2. *"1" + "2" // => "12": concatenation*
3. *"1" + 2 // => "12": concatenation after number-to-string*
4. *true + true // => 2: addition after boolean-to-number*
5. *2 + null // => 2: addition after null converts to 0*

Procareer Academy

# Arithmetic Operators

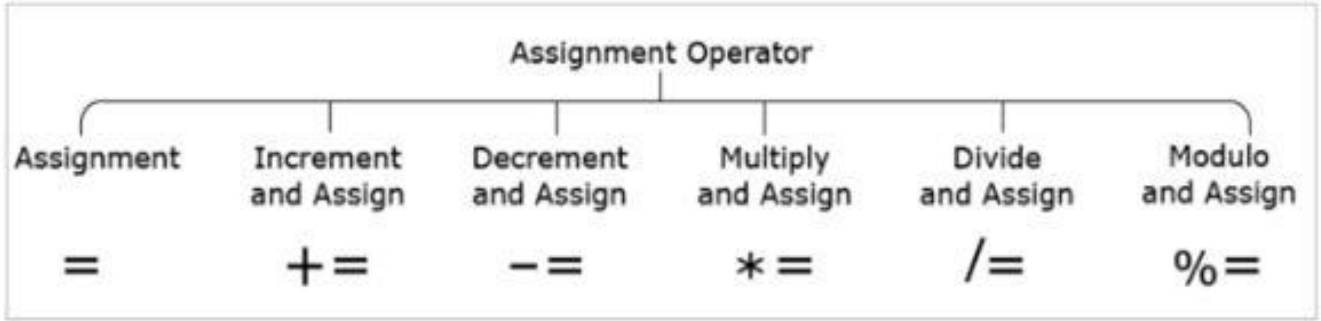
1. *2 + undefined // => NaN: addition after undefined converts to NaN*
2. *let i = 1, j = ++i; // i and j are both 2*
3. *let n = 1, m = n++; // n is 2, m is 1*
4. *1 + 2 + " blind mice" // => "3 blind mice“*
5. *1 + (2 + " blind mice") // => "12 blind mice”*

*(depends on the order in which operations are performed.)*

Procareer Academy

# Assignment Operators

Assignment operators are used for assigning values to the variables.



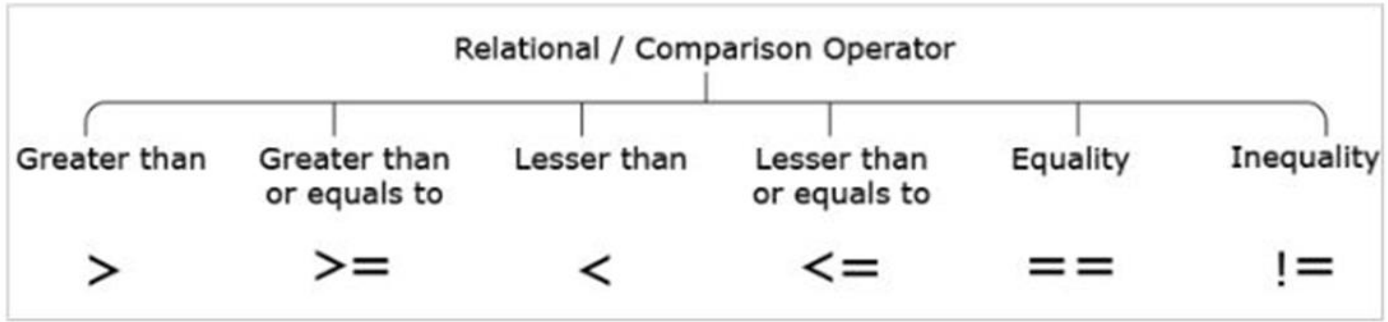

```
1. let num = 30; // num=30
2. let num += 10; // num=num+10 => num=40
3. let num -= 10; // num=num-10 => num=20
4. let num *= 30; // num=num*30 => num=900
5. let num /= 10; // num=num/10 => num=3
6. let num %= 10; // num=num%10 => num=0
```

Procareer Academy

# Relational or Comparison Operators

Relational operators are used for comparing values and the result of comparison is always either true or false.

Relational operators shown below do implicit data type conversion of one of the operands before comparison.




```
1. 10 > 10; //false
2. 10 >= 10; //true
3. 10 < 10; //false
4. 10 <= 10; //true
5. 10 == 10; //true
6. 10 != 10; //false
```

Procareer Academy

# Relational or Comparison Operators

These operators are highly recommended to determine whether two given values are equal or not.

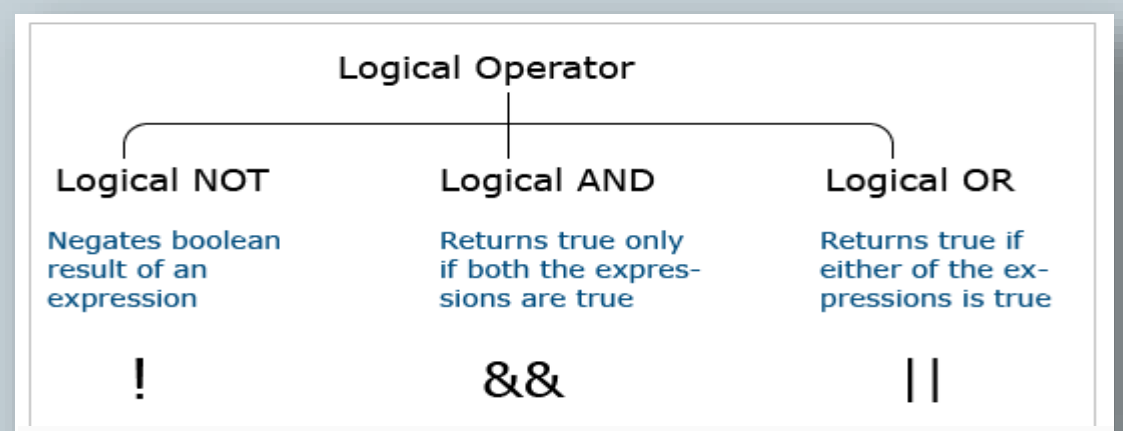
| | Strict equality | Strict inequality |
|---|---|---|
| Definition: | Returns true when value and datatype are equal | Returns true when value or datatype are inequal |
| Operator: | === | !== |
| Example: | 10 === "10" | 10 !== "10" |
| Result: | false | true |
| Explanation: | 10 and "10" have same values but 10 is a number and "10" is a string, hence returns false | 10 and "10" have same values but 10 is a number and "10" is a string, hence returns true |

# Relational or Comparison Operators

1. *1 + 2 // => 3: addition.*
2. *"1" + "2" // => "12": concatenation.*
3. *"1" + 2 // => "12": 2 is converted to "2".*
4. *11 < 3 // => false: numeric comparison.*
5. *"11" < "3" // => true: string comparison.*
6. *"11" < 3 // => false: numeric comparison, "11" converted to 11.*
7. *"one" < 3 // => false: numeric comparison, "one" converted to NaN.*

# Logical Operators

Allow a program to make a decision based on multiple conditions. Each operand is considered a condition that can be evaluated to true or false.




```
1. !(10 > 20); //true
2. (10 > 5) && (20 > 20); //false
3. (10 > 5) || (20 > 20); //true
```

Procareer Academy

# typeof Operator

- JavaScript is a loosely typed language i.e., the type of variable is decided at runtime based on the data assigned to it.
- This is also called dynamic data binding.
- It is used to find the data type of a JavaScript variable.

```
1. typeof "JavaScript World" //string
2. typeof 10.5 // number
3. typeof 10 > 20 //boolean
4. typeof undefined //undefined
5. typeof null //Object
6. typeof {itemPrice : 500} //Object
```

Procareer Academy

# Ternary/Conditional Operator

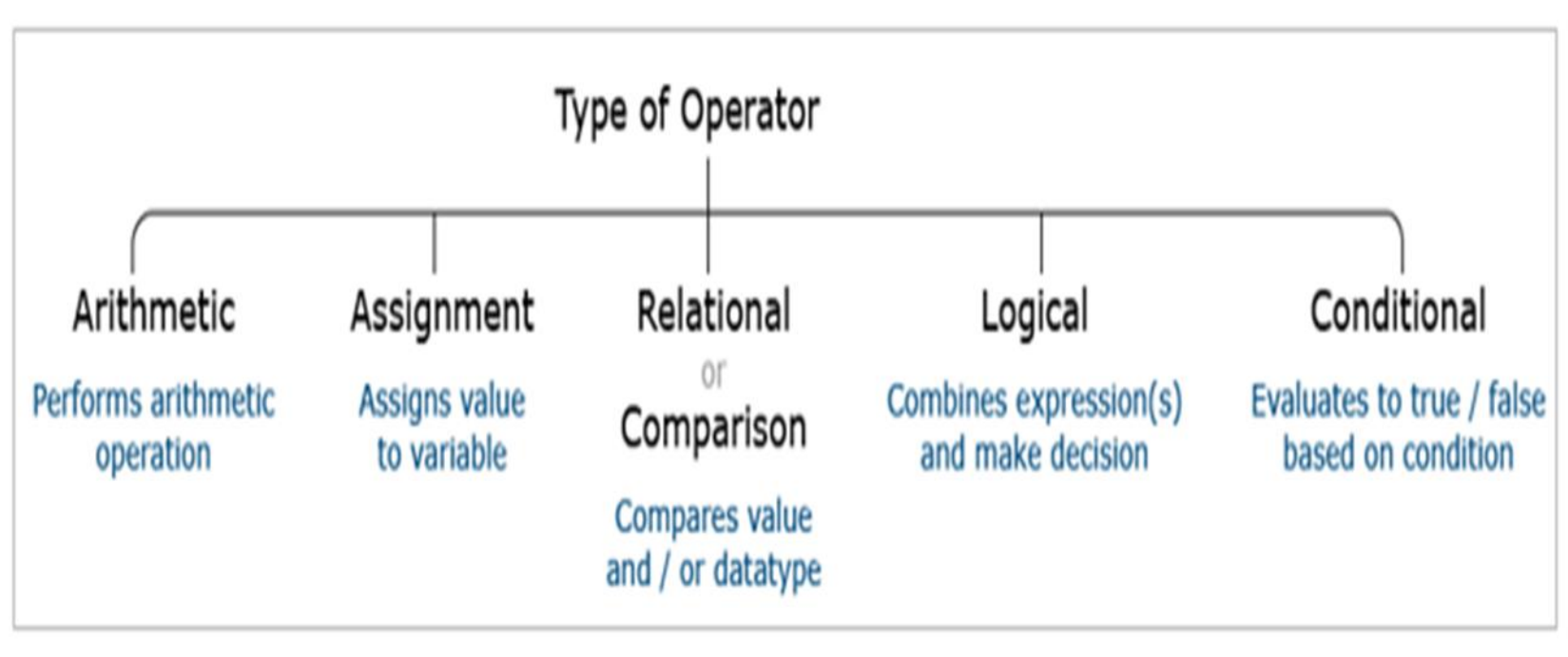
It is a conditional operator that evaluates to one of the values based on whether the condition is true or false.

```
greeting = "hello " + (username ? username : "there");
```

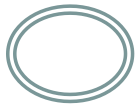
```
1. greeting = "hello ";
2. if (username) {
3. greeting += username;
4. } else {
5. greeting += "there";
6. }
```

Procareer Academy

# Recap




Procareer Academy



Questions?

Procareer Academy

Thank you.

Procareer Academy