Fundamental of programming languages

Python Operators

Not Available in python

Definition

Assign Operators	Arithmetic Operators	Comparison operators	Bitwise Operators	Logical Operators	Increment Operators
The basic assignment operator is equal (=), which assigns the value of its right operand to its left operand. That is, x = y assigns the value of y to x. The other assignment operators are usually shorthand for standard operations, as shown in the following definitions and examples.	Arithmetic operators take numerical values (either literals or variables) as their operands and return a single numerical value. The standard arithmetic operators are addition (+), subtraction (-), multiplication (*), and division (/).	A comparison operator compares its operands and returns a logical value based on whether the comparison is true. The operands can be numerical, string, logical, or object values. Strings are compared based on standard lexicographical ordering, using Unicode values. In most cases, if the two operands are not of the same type, JavaScript attempts to convert them to an appropriate type for the comparison.	JavaScript Uses 32 bits Bitwise Operands JavaScript stores numbers as 64 bits floating point numbers, but all bitwise operations are performed on 32 bits binary numbers. Before a bitwise operation is performed, JavaScript converts numbers to 32 bits signed integers. After the bitwise operation is performed, the result is converted back to 64 bits JavaScript numbers.	Comparison operators are used in logical statements to determine equality or difference between variables or values.	Post and pre increments
= (assign new value to var)	+ (Addition)	== (Check equality values)	& AND Sets each bit to 1 if both bits are 1	&& and (x < 10 && y > 1) is true	A=1; A++; (post increment)
+= (Assign new value with old value to var)	- (Subtraction)	=== (Check equality values with data types)	| OR Sets each bit to 1 if one of two bits is 1	|| or (x == 5 || y == 5) is false	A=1; ++a; (pre increment)
-= (Subtract new value from old value)	/ (Division)	!=, <> (inequality, not equal) !== (comparison operators)	^ XOR Sets each bit to 1 if only one of two bits is 1	! not !(x == y) is true	A=20; a--; (Post Decrement)
*= (Multiply new value with old value)	* (Multiplication)	> (Grater than)	~ NOT Inverts all the bits		A=20; --a; (pre Decrement)
/= (Divide new value with old value)	% (Modulus)	< (Less than)	<< Zero fill left shift Shifts left by pushing zeros in from the right and let the leftmost bits fall off		
%= (Modulus with old value)	** (Exponent)	>= (Grater than equal)	>> Signed right shift Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off		
**= (Power old value by new value) //= (Root old value by new value)		<= (Less than equal)	>>> Zero fill right shift Shifts right by pushing zeros in from the left, and let the rightmost bits fall off		