Beginner's Guide to Operators in START Programming

Operators are symbols or keywords that perform operations on data, such as calculations, comparisons, and logical operations. Understanding how operators work is essential for writing effective and efficient code. In this guide, we'll explore different types of operators and how to use them in your programs using your language's syntax.

Numerical Operators

Numerical operators are used to perform arithmetic operations on numbers. This can allow us to easily solve problems we face.

Addition

The addition operator adds two numbers together.

```
2 add 3
```

Subtraction

The subtraction operator subtracts one number from another.

```
5 sub 2
```

Multiplication

The multiplication operator multiplies two numbers.

```
4 mult 6
```

Division

The division operator divides one number by another.

```
15 div 3
```

Modulus

The modulus operator returns the remainder of the division.

```
5 mod 2
```

Powers

The powers operator raises the first number to the power of the second number.

```
2 pow 3
```

Numerical Operators in Use

Lets examine some examples of these operators in use, and the result they produce.

Addition and Subtraction

```
x is 5
y is 10
addition is x add y
subtraction is y sub x

write(addition) nl
write(subtraction) nl
```

This code produces the following result:

```
15
5
```

Multiplication and Division

```
w is 4
x is 6
multiplication is w mult x
write(multiplication) nl

y is 15
z is 3
division is y div z
write(division) nl
```

This code produces the following result:

```
24
5.0
```

Modulus and Powers

```
w is 5
x is 2
modulus is w mod x
write(modulus) nl

y is 2
z is 3
powers is y pow z
write(powers) nl
```

This code produces the following result:

```
1
8.0
```

Comparison Operators in Use

Comparison operators are used to compare two values and return a boolean result. These will become important later when we look at If Statements and While Loops, but for now we will gain an understanding of how they work.

In each of the below cases, `a` and `b` are variables of some value to be compared to check a condition.

Equality

The equality operator checks if two values are equal.

```
a equals b
```

Inequality

The inequality operator checks if two values are not equal.

```
a not equals b
```

Greater Than

The greater than operator checks if the first value is greater than the second value.

```
a greater than b
```

Less Than

The less than operator checks if the first value is less than the second value.

```
a less than b
```

Great Than or Equal To

The greater than or equal to operator checks if the first value is greater than or equal to the second value.

```
a greater than or equal to b
```

Less Than or Equal To

The less than or equal to operator checks if the first value is less than or equal to the second value.

```
a less than or equal to b
```

We will see working examples of these in later lessons on If Statements.

Boolean Operators in Use

Boolean operators are used to perform logical operations on boolean values. A Boolean value is either true or false. These checks can be used to confirm that more than one value is either true or false.

Logical AND

The logical AND operator returns true if both operands are true, otherwise false.

```
a and b
```

Logical OR

The logical OR operator returns true if at least one operand is true, otherwise false.

```
a or b
```

Logical NOT

The logical NOT operator returns the opposite boolean value of the operand.

```
not a
```

We will also see working examples of these in later lessons on If Statements.

All of these above operators also have alternative symbols that can be used to shorten the code, they are as follows:

- add: +
- sub: -

- mult: *
- div: /
- mod: %
- pow: ^
- equals: ==
- not equals: !=
- greater than: >
- less than: <
- greater than or equal to: >=
- less than or equal to: <=

Conclusion

Operators are powerful tools that enable you to perform various operations in your programs. By understanding how to use numerical, comparison, and boolean operators, you'll be able to write more dynamic and functional code. Practice using these operators in different scenarios to familiarize yourself with their behavior and improve your problem-solving skills in programming.