Data Types and Operators

Expressions

- Expressions are combinations literals values, variables or function calls
- All expressions have a type and a value

- Type:
  - Programming languages have types
  - At the lowest level they are identifiers on how something is stored or manipulated by the programming language
  - At a higher level, it signals to the programmer how a value can be used.
    - We can do arithmetic with numbers, but this would not make sense with strings.

Types in JavaScript

- Primitive types
  - Boolean
  - Number
  - String
  - Null
  - undefined
- Objects
  - Not a primitive type
  - Can be composed of primitive types

Numbers in JavaScript

Number Type In Math	Example
Integers	1
Floating point(real)	1.56

- What are valid operators?
  - Arithmetic: + - * / %
  - Comparison: < <= > >= !== ===
- JavaScript treats all numbers the same, though it’s useful to distinguish between floating point numbers and integers in programming as there are equivalent concepts in math
- Floating point numbers are approximately the real numbers and integers are the same as integers (whole numbers)

What is an operator?

- A symbol or keyword that combines, modifies, or compares one or more expressions
- The result of an operation is also an expression:

```
1;
1+1;
1+1+1;
```

- In this example, the plus symbol is the operator

Number Literals

- Base 10
  - 1234

- Floating point
  - 1.23
- Scientific Notation
  - `6.7e-11`

Special Values

Value	Description
NaN (Not A Number)	When the result of a calculation does not result in an real number. (Or when, there is not enough information to computer the result)
Infinity	When the result of the calculation is infinity
-Infinity	When the result of the calculation is negative infinity

Strings

- Strings are used to store and manipulate text
  - eg:
  - “I am in CPSC 1045”
  - “123456” (Hint: this is not a number)

- Valid string operators
  - Concatenation
    - +
  - Comparisons
    - >   >=   <   <=   !==   ===
    - Also ==   and   !=   but we’ll avoid these two for now)
    - Comparisons go by ‘alphabetical’ order relative to the Unicode encoding

Booleans

- Boolean Data Type has value of true or false
- Valid operators
  - `!  &&  ||  ===  !==`
- The **not** operator
  - `!`
  - `!true == false`
  - `!false == true`
  - `5 == 5 is true`
  - `5 != 5  is false`

Booleans

- && and || are binary operators that combine two Boolean expressions
  - && Is the **and** operator
  - || is the **or** operator
- === and !== are binary operators that compare two Boolean expressions

How boolean operators work

- Truth table
- Ex. and

a	b	Result of a && b
T	T	T
T	F	F
F	T	F
F	F	F

- Truth table
- Ex. or

a	b	Result of a || b
T	T	T
T	F	T
F	T	T
F	F	F

Variables

- Variables are storage containers
- In JavaScript they can hold pretty much anything
- Their type changes depending on what is being stored
- we use the let keyword to define a variable

- Ex. `let x;`
  - defines the variable x, with undefined type

- Ex. `let x = 5;`
  - defines the variable x, and places 5 in that container
  - Until I change the value, when I use the symbol x, it will have a value of 5

Assignment Operators

- Assignment operators assign (store) a value to a variable
- We've seen the = operator
- But there are several others
  - +=   -=   *=   /=   %=

Ex.

```
let x = 5;
x += 10;
```

This is the same as

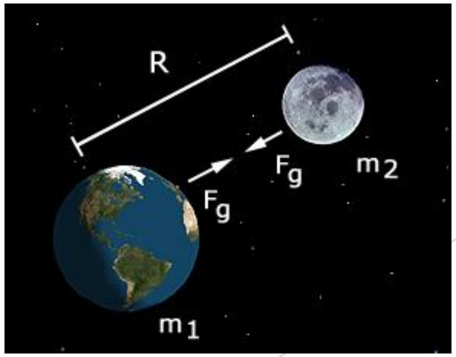
```
x = x + 15;
```

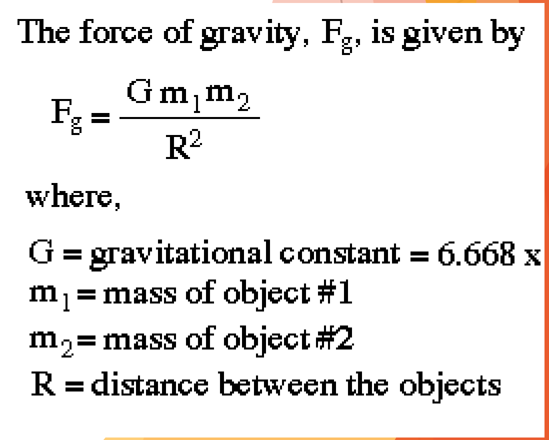
Ex. Complex Numerical Expressions in JavaScript

Example:

Newton's Law of Universal Gravitation

```
const G = 6.67e-11;  //declare gravitational constant
const m1 = 5.972e24 //declare mass(kg) of earth
const m2 = 7.347e22 //declare mass(kg) of the moon
const r = 3.84e8;   //declare distance(m) from earth to
moon
let F = G*m1*m2/Math.pow(r,2);
```




The force of gravity, F_g, is given by

$$F_g = \frac{G\, m_1 m_2}{R^2}$$

where,

G = gravitational constant = 6.668 x

m_1 = mass of object #1

m_2 = mass of object #2

R = distance between the objects

How != and == Work

- == is the comparison operator
- a == b
  - Returns true if a the same as b
  - Returns false if a is not the same as b

- != negation of the comparison operator.
- a != b
  - Returns true if a is not the same as b
  - Returns false if the a is the same as b

How !== and === Work

- === is the **exact equal to** comparison operator
- a === b
  - Returns true if a the same as b in both value AND type
  - Returns false if a is not the same as b in either value OR type

- !== negation of the exact comparison operator
- a !== b
  - Returns true if a is not the same as b in either value OR type
  - Returns false if the a is the same as b in both value AND type

Numbers, Comparisons and Booleans

- All of the following compare similar types and return a boolean value
  - ===
  - >
  - >=
  - <
  - <=
- Ex. 15 === 56; is false
- Ex, 15 <= 56; is true
- Ex. “Hello” === 456; is false
- Ex. “456” === 456 is also false

Operators Summary

- Binary Operator joins two expressions
  - **Arithmetic:** + - / * %
  - **Comparison:** < > <= >= === !==
  - Are operators that require two expressions
- Unary Operator modifies the value of one expression
  - ! - ++ --
  - Are operators that require one expression
- Assignment operator assigns values/references to variables
  - = += -= *= /= %=
- Dot Operator(.)
  - We will use this often, but we will explain this when we talk about objects
- Brackets/parenthesis ()
  - causes execution of a function

What's the Difference?

- Boolean, Strings and Numbers are all stored differently inside the computer for efficiency reasons
  - Numbers are stored in a way where it's efficient for computers to do calculations with them
  - Strings are stored in such a way, where it's efficient to manipulate the string
- Boolean, Strings, Numbers are **Primitive** data types in JavaScript
  - That is they are used as part of building blocks in the JavaScript Language
  - JavaScript provides built-in support for these types

Converting Strings and Numbers

- This process is called CASTING or TYPE CONVERSION

- Methods to convert strings to numbers
  - `parseInt()`
  - `parseFloat()`
  - `Number()`

- Methods to convert numbers to strings

```
Ex. let x = 5;
    x.toString();

Ex. (5.45).toString();
```

Number() example:

```
let x1 = true;
let x2 = false;
let x3 = "999";
let x4 = "999 888";

Number(x1) is 1
Number(x2) is 0
Number(x3) is 999
Number(x4) is NaN
```

How to Identify Variable Type

- The typeof method returns the type of a value
  - `typeof(1234);` returns “number”
  - `typeof(“Hello”);` returns “string”

- There are 6 types the typeof can return
  - “string” “number” “boolean”
  - “function” means code
  - “object” means a container class
  - “undefined” means no type

== VS ===

- Both == and ===, and their negation != and !== are binary comparison operators, but there is a subtle difference
  - “1”== 1, is equal to true
  - “1”===1, is equal to false
- Clearly a conversion takes place for ==
- When you are using **primitive types** stick with ===
- Manually convert to a string or number if you want the == behavior

Specifics About ==

- Different conversions take place depending on the type of the left hand side and right hand side of the statement
- Read the ECMAScript documentation for all the details.
  - The algorithm cannot fit on one slide.
- Briefly:
  - When comparing number to string the ToNumber(<string>) function is applied and then the comparison takes place
  - When comparing number to boolean the ToNumber(<string>) function is applied and then the comparison takes place

Specifics About ===

- In comparisons using === type conversion does not take place
- **Some Interesting cases**
- Result is always false when comparing different types
  - unless X is type null or undefined, then result is always true
- When comparing strings, if the strings have the same characters and length, then the result is true
- For objects, the result is true only if they are the same object.
  - This behavior, is clearly different and contradictory to how strings are treated! Equal rights for Equal types? Sadly no.

Operator Precedence

- AKA Order of Operations
- Terms:
  - Left-associativity (left-to-right)
  - Right-associativity (right-to-left)
- JavaScript Operators are ordered by precedence
  - Operators that have higher precedence will be evaluated first

Common Operator Precedence

Common Operator	Precedence	Associativity	Individual operator
Grouping	19	n/a	(…)
Multiplication, Division, Modulo	14	left-to-right	* / %
Addition, Subtraction	13	left-to-right	+ -
Equal Not equal Greater than, Less than, etc	10,11	left-to-right	=== !== > <
Logical and, or	9,7	left-to-right	&& ||
Assignment	3	right-to-left	=

Implicit Type Conversion

- Implicit type conversion is when you convert between strings/numbers without specifically using a method to do so
- The results of implicit type conversion will depend on the order of operations
- This is where making mistakes is very easy in JavaScript

Example	Result
“Dog” +1	Dog1
1+”Dog”	1Dog
1+2+”Dog”	3Dog
“Dog” + 1+2	Dog12
“Dog” + (1+2)	Dog3
“Dog” +2*3	Dog6

Expressions are Evaluated Step-by-Step

1. The highest precedence operator of evaluated first
    1. If there is more than one operator with the same precedence, the the associativity rule is used
    2. That is operators are evaluated either left to right or right to left
2. The operators with the next lower are evaluated next
3. Repeat until you are down to a single value

Example

- What will the value of x be after the expression is evaluated?

```
let x = (1+3*4) +3 > "DOG" + "CAT"
let x =(1+12)+3 > "DOG"+"CAT"
let x = 13+3 > "DOG" + "CAT"
let x = 16 > "DOG" + "CAT"
let x = 16 > "DOGCAT"
let x = false
```

Substrings

- We can obtain a portion of a string by using the substring method
- Consider the following string:

```
let testStr = “My name is unknown”;
```

- we can extract the work ‘**name**’ by using the substring method.

```
testStr.substring(3,7);
```

Optional end Index

Start Index

- We can extract the word ‘unknown’ similarly

```
testStr.substring(11);
```

Evaluating Expressions in Console

- The console window is an interactive JavaScript command prompt
- We can type expressions into the console window and get the results
  - In the browser, right click on the page and go down to “Inspect”
  - In the new area that pops up, find the “Console” tab
  - Give it a try!