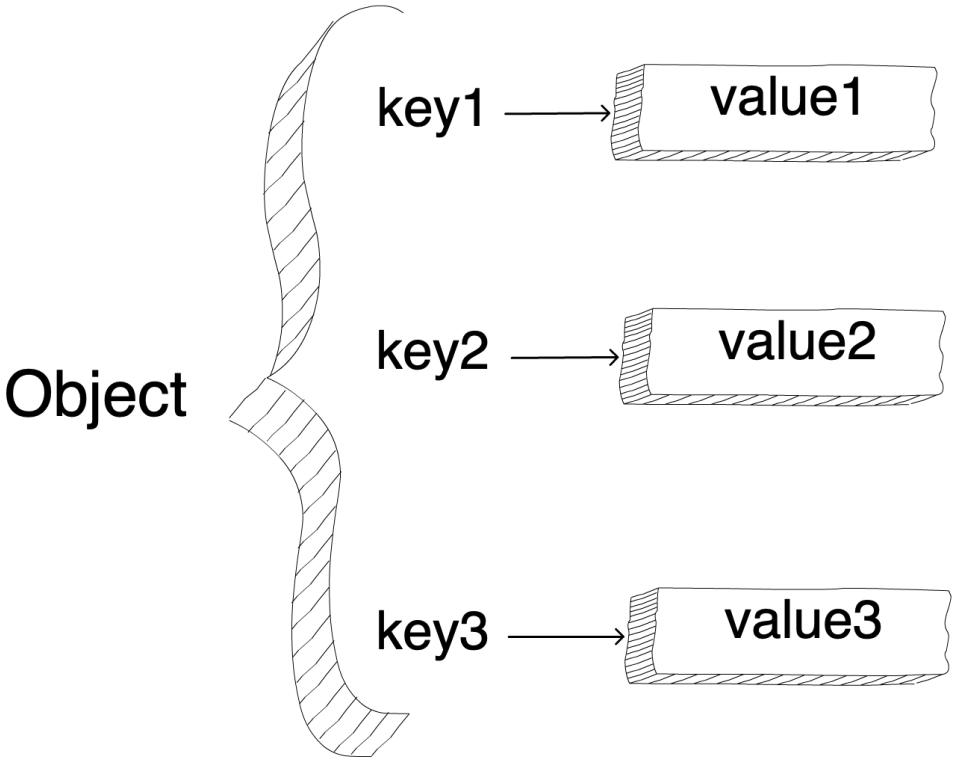
12.1. Objects and Why They Matter

So far we have learned a lot about arrays, which are data structures that can hold many values. **Objects** are also data structures that can hold many values.

Unlike arrays, each value in an object has a name or **key** for reference purposes. The pairing between a key and its value is called a **key/value pair**.

Objects store as many key/value pairs as needed, and each value needs a key. Without a key, the value cannot be accessed or modified by the programmer.



12.1.1. Initializing Objects

When defining an object, we call the initialization an **object literal**. Objects require three things for the definition: a name, a set of keys, and their corresponding values.

Note

Object literals use curly braces, `{}`, to enclose the key/value pairs.

Once we have these three things, we write an object literal like so:

```
let objectName = {key1:value1, key2:value2, key3:value3, ... };
```

If we have a lot of key/value pairs in our object, we can also put each one on a separate line!

```
let objectName = {
    key1: value1,
    key2: value2,
    key3: value3,
    .
    .
    .
};
```

Warning

When putting the key/value pairs on separate lines, it is important to pay attention to spaces and tabs! Incorrect spacing or tab usage can result in a bug.

When defining an object, keep in mind that the keys can only be valid JavaScript strings. The values can be any of the data types that we have previously discussed.

Example

Let's say that we want to create a small program for a zoo. We could create an object for storing the different data points about the animals in a zoo. We start with our first tortoise. His name is Pete! He is an 85 year old, 919 lb Galapagos Tortoise, who prefers a diet of veggies. Our object literal for all of this important data about Pete would be:

```
let tortoiseOne = {
    species: "Galapagos Tortoise",
    name: "Pete",
    weight: 919,
    age: 85,
    diet: ["pumpkins", "lettuce", "cabbage"]
};
```

12.1.2. Methods and Properties

A **property** of an object is a key/value pair of an object. The property's name is the key and the property's value is the data assigned to that key.

A **method** performs an action on the object, because it is a property that stores a function.

Example

In the case of Pete, our zoo's friendly Galapagos Tortoise, the object `tortoiseOne` has several properties for his species, name, weight, age, and diet. If we wanted to add a method to our object, we might add a function that returns a helpful statement for the general public.

```
1  let tortoiseOne = {
2      species: "Galapagos Tortoise",
3      name: "Pete",
4      weight: 919,
5      age: 85,
6      diet: ["pumpkins", "lettuce", "cabbage"],
7      sign: function() {
8          return this.name + " is a " + this.species;
9      }
10 };
```

In the example above, on line 8, we see a keyword which is new to us. Programmers use the `this` keyword when they call an object's property from within the object itself. We could use the object's name instead of `this`, but `this` is shorter and easier to read. For example, the method, `sign`, could have a return statement of `tortoiseOne.name + " is a " +tortoiseOne.species"`. However, that return statement is bulky and will get more difficult to read with more references to the `tortoiseOne` object.

12.1.3. Check Your Understanding

Question

Which of the following is NOT a true statement about objects?

a. Objects can store many values
b. Objects have properties
c. Objects have methods
d. Keys are stored as numbers

Question

Which keyword can be used to refer to an object within an object?

a. `Object`
b. `let`
c. `this`

12.2. Working with Objects

12.2.1. Accessing Properties

When using objects, programmers oftentimes want to retrieve or change the value of one of the properties. To access the value of a property, you will need the object's name and the key of the property.

Programmers have two ways to access the value of property:

1. Bracket syntax
2. Dot notation.

12.2.1.1. Bracket Syntax

To access a property with bracket syntax, the code looks like: `object["key"]`.

12.2.1.2. Dot Notation

To access a property with dot notation, the code looks like: `object.key`. Notice that the key is no longer surrounded by quotes. However, keys are still strings.

Note

Recall, the only restraint in naming a key is that it has to be a valid JavaScript string. Since a key could potentially have a space in it, bracket syntax would be the only way to access the value in that property because of the quotes.

Example

```
let tortoiseOne = {
    species: "Galapagos Tortoise",
    name: "Pete",
    weight: 919,
    age: 85,
    diet: ["pumpkins", "lettuce", "cabbage"]
};

console.log(tortoiseOne["name"]);
console.log(tortoiseOne.name);
```

Console Output

```
Pete
Pete
```

12.2.2. Modifying Properties

A programmer can modify the value of a property by using either notation style.

Warning

Recall that mutability means that a data structure can be modified without making a copy of that structure. Objects are mutable data structures. When you change the value of a property, the original object is modified and a copy is NOT made.

Example

In our zoo software, we may want to update Pete's weight as he has gained 10 lbs. We will use both bracket syntax and dot notation for our software, but that is not a requirement! Feel free to use whichever one suits your needs and is easiest for you and your colleagues to read.

```
let tortoiseOne = {
    species: "Galapagos Tortoise",
    name: "Pete",
    weight: 919,
    age: 85,
    diet: ["pumpkins", "lettuce", "cabbage"]
};

console.log(tortoiseOne.weight);

newWeight = tortoiseOne.weight + 10;

tortoiseOne["weight"] = newWeight;

console.log(tortoiseOne["weight"]);

```

Console Output

```
919
929
```

12.2.3. Check Your Understanding

All of the questions below refer to an object called `giraffe`.

```
let giraffe = {
  species: "Reticulated Giraffe",
  name: "Cynthia",
  weight: 1500,
  age: 15,
  diet: "leaves"
};
```

Question

We want to add a method after the `diet` property for easily increasing Cynthia's age on her birthday. Which of the following is missing from our method? You can select MORE than one.

```
birthday: function () {age = age + 1;}
```

a. `return`
b. `this`
c. `diet`
d. a comma

Question

Could we use bracket syntax, dot notation, or both to access the properties of `giraffe`?

12.3. Coding With Objects

12.3.1. Booleans and Objects

Objects are not stored by their properties or by value, but by *reference*. Storing something by reference means that it is stored based on its location in memory. This can lead to some confusion when comparing objects.

Example

Let's see how this affects our zoo software! Surely, the zoo has more than one tortoise. The second tortoise is named Patricia!

```
let tortoiseTwo = {
   species: "Galapagos Tortoise",
   name: "Patricia",
   weight: 800,
   age: 85,
   diet: ["pumpkins", "lettuce", "cabbage"],
   sign: function() {
      return this.name + " is a " + this.species;
   }
};
```

Because Pete and Patricia are members of the same species, are the same age, and have the same diet, you might notice that many of their properties are equal, but some are not. Pete weighs more than Patricia and of course, they have different names!

For this example, we will only keep the `species` and `diet` properties.

```
let tortoiseOne = {
   species: "Galapagos Tortoise",
   diet: ["pumpkins", "lettuce", "cabbage"]
};

let tortoiseTwo = {
   species: "Galapagos Tortoise",
   diet: ["pumpkins", "lettuce", "cabbage"]
};

console.log(tortoiseOne === tortoiseTwo);
```

Console Output

```
false
```

The objects contain properties that have the same keys and equal values. However, the output is `false`.

Even though `tortoiseOne` and `tortoiseTwo` have the same keys and values, they are stored in separate locations in memory. This means that even though you can compare the properties in different objects for equality, you cannot compare the objects themselves for equality.

12.3.2. Iterating Through Objects

We can iterate through all of the values in an object, much like we would do with an array. We will use a for loop to do that, but with a slightly different structure. `for...in` loops are specifically designed to loop through the properties of an object. Each iteration of the loop accesses a key in the object. The loop stops once it has accessed every property.

Example

```
let giraffe = {
  species: "Reticulated Giraffe",
  name: "Cynthia",
  weight: 1500,
  age: 15,
  diet: "leaves"
};

for (item in giraffe) {
   console.log(item + ", " + giraffe[item]);
}
```

Console Output

```
species, Reticulated Giraffe
name, Cynthia
weight, 1500
age, 15
diet, leaves
```

In this example, `item` is a variable that holds the string for each key. It is updated with each iteration of the loop.

Note

Inside a `for..in` loop, we can only use bracket syntax to access the property values.

Try It!

Write a `for..in` loop to print to the console the values in the `tortoiseOne` object. Try it at repl.it

12.3.3. Objects and Functions

Programmers can pass an object as the input to a function, or use an object as the return value of the function. Any change to the object within the function will change the object itself.

Example

```
let giraffe = {
  species: "Reticulated Giraffe",
  name: "Cynthia",
  weight: 1500,
  age: 15,
  diet: "leaves"
};

function birthday(animal) {
    animal.age = animal.age + 1;
    return animal;
}

console.log(giraffe.age);

birthday(giraffe);

console.log(giraffe.age);
```

Console Output

```
15
16
```

On line 16, when the `birthday` function is called, `giraffe` is passed in as an argument and returned. After the function call, `giraffe.age` increases by 1.

12.3.4. Check Your Understanding

Question

What type of loop is designed for iterating through the properties in an object?

Question

Given the following object definitions, which statement returns `true`?

```
let tortoiseOne = {
   age: 150,
   species: "Galapagos Tortoise",
   diet: ["pumpkins", "lettuce", "cabbage"]
};

let tortoiseTwo = {
   age: 150,
   species: "Galapagos Tortoise",
   diet: ["pumpkins", "lettuce", "cabbage"]
};
```

a. `tortoiseOne == tortoiseTwo`
b. `tortoiseOne === tortoiseTwo`
c. `tortoiseOne.age === tortoiseTwo.age`

12.4. The `Math` Object

JavaScript provides several built-in objects, which can be called directly by the user. One of these is the `Math` object, which contains more than the standard mathematical operations (`+, -, *, /`).

In the previous sections, we learned how to construct, modify, and use objects such as `giraffe`. However, JavaScript built-in objects cannot be modified by the user.

Unlike other objects, the `Math` object is immutable.

12.4.1. `Math` Properties Are Constants

The `Math` object has 8 defined properties. These represent *mathematical constants*, like the value for pi (π) or the square root of 2.

Instead of defining a variable to hold as many digits of pi as we can remember, JavaScript stores the value for us. To use this value, we do NOT need to create a new object. By using dot notation and calling `Math.PI`, we can access the value of pi.

Example

```
 console.log(Math.PI);
1console.log(Math.PI*4);
2console.log(Math.PI + Math.PI);
3
```

Console Output

```
3.141592653589793
12.566370614359172
6.283185307179586
```

As stated above, the properties within `Math` *cannot* be changed by the user.

Example

```
 console.log(Math.PI);
1
2Math.PI = 1234;
3
4console.log(Math.PI);
5
```

Console Output

```
3.141592653589793
3.141592653589793
```

To use one of the other constants stored in `Math`, we replace `PI` with the property name (e.g. `SQRT2` stores the value for the square root of 2).

12.4.2. Other `Math` Properties

Besides the value of pi, JavaScript provides 7 other constants. How useful you find each of these depends on the type of project you need to complete.

More powerful uses of the `Math` object involve using *methods*, which we will examine next.

12.5. Math Methods

As with strings and arrays, JavaScript provides some built-in *methods* for the `Math` object. These allow us to perform calculations or tasks that are more involved than simple multiplication, division, addition, or subtraction.

12.5.1. Common Math Methods

The `Math` object contains over 30 methods. The table below provides a sample of the most frequently used options. More complete lists can be found here:

1. W3 Schools Math Reference
2. MDN Web Docs

To see detailed examples for a particular method, click on its name.

Ten Common Math Methods

Method	Syntax	Description
abs	`Math.abs(number)`	Returns the positive value of `number`.
ceil	`Math.ceil(number)`	Rounds the decimal `number` UP to the closest integer value.
floor	`Math.floor(number)`	Rounds the decimal `number` DOWN to the closest integer value.
max	`Math.max(x,y,z,...)`	Returns the largest value from a set of numbers.
min	`Math.min(x,y,z,...)`	Returns the smallest value from a set of numbers.
pow	`Math.pow(x,y)`	Returns the value of x raised to the power of y (x^y).
random	`Math.random()`	Returns a random decimal value between 0 and 1, NOT including 1.
round	`Math.round(number)`	Returns `number` rounded to the nearest integer value.
sqrt	`Math.sqrt(number)`	Returns the square root of `number`.
trunc	`Math.trunc(number)`	Removes any decimals and returns the integer part of `number`.

12.5.2. Check Your Understanding

Follow the links in the table above for the `floor`, `random`, `round`, and `trunc` methods. Review the content and then answer the following questions.

Question

Which of the following returns `-3` when applied to `-3.87`?

a. `Math.floor(-3.87)`
b. `Math.random(-3.87)`
c. `Math.round(-3.87)`
d. `Math.trunc(-3.87)`

Question

What is printed by the following program?

```
let num = Math.floor(Math.random()*10);

console.log(num);

```

a. A random number between 0 and 9.
b. A random number between 0 and 10.
c. A random number between 1 and 9.
d. A random number between 1 and 10.

Question

What is printed by the following program?

```
let num = Math.round(Math.random()*10);

console.log(num);

```

a. A random number between 0 and 9.
b. A random number between 0 and 10.
c. A random number between 1 and 9.
d. A random number between 1 and 10.

12.6. Combining `Math` Methods

The `Math` methods provide useful actions, but each one is fairly specific in what it does (e.g. taking a square root). At first glance, this might seem to limit how often we need to call on `Math`. However, the methods can be manipulated or combined to produce some clever results.

12.6.1. Random Selection From an Array

To select a random item from the array `happiness = ['Hope', 'Joy', 'Peace', 'Love', 'Kindness', 'Puppies','Kittens', 'Tortoise']`, we need to randomly generate an index value from 0 to 7. Since `Math.random()` returns a decimal number between 0 and 1, the method on its own will not work.

The Math.random appendix page describes how to generate random integers by combining the `random` and `floor`methods. We will use this functionality now.

Let's define a function that takes an array as a parameter. Since we might not know how many items are in the array, we cannot multiply `Math.round()` by a specific value. Fortunately, we have the `length` property…

Example

```
function randomSelection(arr){
   let index = Math.floor(Math.random()*arr.length);
   return arr[index];
}

let happiness = ['Hope','Joy','Peace','Love','Kindness','Puppies','Kittens','Tortoise'];

for (i=0; i < 8; i++){
   console.log(randomSelection(happiness));
}
```

repl.it

Console Output

```
Tortoise
Love
Kindness
Hope
Kittens
Kindness
Love
Hope
```

The `happiness` array has a length of 8, so in line 2 `Math.floor(Math.random()*arr.length)` evaluates as`Math.floor(Math.random()*8)`, which generates an integer from 0 to 7. Line 3 then returns a random selection from the array.

12.6.2. Rounding to Decimal Places

The `ceil`, `floor`, and `round` methods all take a decimal value and return an integer, but what if we wanted to round 5.56789123 to two decimal places? Let's explore how to make this happen by starting with a simpler example.

`Math.round(1.23)` returns 1, but what if we want to round to one decimal place (1.2)? We cannot alter what `round` does—it *always* returns an integer. However, we CAN change the number used as the argument.

Let's multiply 1.23 by 10 (1.23*10 = 12.3) and then apply the method. `Math.round(12.3)` returns 12. Why do this? Well, if we divide 12 by 10 (12/10 = 1.2) we get the result of *rounding 1.23 to one decimal place*.

Combining these steps gives us `Math.round(1.23*10)/10`, which returns the value 1.2.

Let's return to 5.56789123 and step through the logic for rounding to two decimal places:

Step	Description
`Math.round(5.56789123*100)/100`	Evaluate the numbers in () first: 5.56789123*100 = 556.789123
`Math.round(556.789123)/100`	Apply the `round` method to 556.789123
`557/100`	Perform the division 557/100 = 5.57

The clever trick for rounding to decimal places is to multiply the original number by some factor of 10, `round` the result, then divide the integer by the same factor of 10. The number of digits we want after the decimal are shifted in front of the '.' before rounding, then moved back into place by the division.

Rounding to Decimal Places

Decimal Places In Answer	Multiply & Divide By	Syntax
1	10	`Math.round(number*10)/10`
2	100	`Math.round(number*100)/100`
3	1000	`Math.round(number*1000)/1000`
etc.	etc.	etc.

12.6.3. Check Your Understanding

Question

Which of the following correctly rounds 12.3456789 to 4 decimal places?

a. `Math.round(12.3456789)*100/100`
b. `Math.round(12.3456789*100)/100`
c. `Math.round(12.3456789*10000)/10000`
d. `Math.round(12.3456789)*10000/10000`

12.7. Exercises: Objects & Math

At our space base, it is a historic day! Five non-human animals are ready to run a space mission without our assistance! For the exercises, you will use the same five animal objects throughout.

Starter Code

12.7.1. Part 1: Create More Objects

Based on the two object literals provided in the starter code, create new object literals for three more animals:

Animal Astronauts

Name	Species	Mass (kg)	Age (years)
Brad	Chimpanzee	11	6
Leroy	Beagle	14	5
Almina	Tardigrade	0.0000000001	1

12.7.1.1. Add a New Property

For each animal, add a property called `astronautID`. Each `astronautID` should be assigned a number between 1 and 10 (including 10). However, no crew members should have the same ID.

12.7.1.2. Add a Method

Add a `move` method to each animal object. The `move` method will select a random number of steps from 0 to 10 for the animal to take. The number can be 0 as well as 10.

12.7.1.3. Store the Objects

Create a `crew` array to store all of the animal objects.

12.7.2. Part 2: Crew Reports

Upper management at the space base wants us to report all of the relevant information about the animal astronauts.

Define a `crewReports` function to accomplish this. When passed one of the animal objects, the function returns a template literal with the following
string: `'____ is a ____. They are ____ years old and ____ kilograms. Their ID is ____.'`

Fill in the blanks with the name, species, age, mass, and ID for the selected animal.

12.7.3. Part 3: Crew Fitness

Before these animal astronauts can get ready for launch, they need to take a physical fitness test. Define a `fitnessTest`function that takes an array as a parameter.

Within the function, race the five animals together by using the `move` method. An animal is done with the race when they reach 20 steps or greater. Store the result as a string: `'____ took ____ turns to take 20 steps.'` Fill in the blanks with the animal's name and race result. Create a new array to store how many turns it takes each animal to complete the race.

Return the array from the function, then print the results to the console (one animal per line).

HINT: There are a lot of different ways to approach this problem. One way that works well is to see how many iterations of the `move` method it will take for each animal to reach 20 steps.

12.8. Studio: Objects & Math

In the exercises, you created objects to store data about the candidates for our animal astronaut corps. For this studio, we provide you with a ready-made set of candidates.

You must create code to:

A. Select the crew.
B. Perform critical mission calculations.
C. Determine the fuel required for launch.

12.8.1. Before You Start

If you are enrolled in a LaunchCode program, access this studio by following the repl.it classroom links posted in your class at learn.launchcode.org.

If you are working through this material on your own, use the repl.it links contained on this page.

12.8.2. Select the Crew

To access the code for exercise 1, open this repl.it link.

12.8.2.1. Randomly Select ID Numbers

Each candidate was assigned an ID number, which is stored in the candidate's data file and in the `idNumbers` array.

1. Write a `selectRandomEntry` function to select a random entry from the `idNumbers` array.vReview the Combining Math Methods section if you need a reminder on how to do this.
2. Use the function to select three ID numbers. Store these selections in a new array, making sure to avoid repeated numbers. No animal can be selected more than once!
3. Use a template literal to print, `'____, ____, and ____ are going to space!'` Fill in the blanks with the names of the selected animals.

Tip

`arrayName.includes(item)` can be used to check if the array already contains `item`. A `while` loop can keep the selection process going until the desired number of entries have been added to the array.

12.8.2.2. Build a `crew` Array

Design a function that takes two arrays as parameters. These hold the randomly selected ID numbers and the candidate objects.

Use one or more loops to check which animals hold the lucky ID numbers. They will be going on the space mission! Store these animals in a `crew` array, and then return that array.

12.8.3. Orbit Calculations

To access the code for the orbit calculations and first bonus mission, go to repl.it.

1. Spacecraft orbits are not circular, but we will assume that our mission is special. The animals will achieve a circular orbit with an altitude of 2000 km.
    a. Define a function that returns the circumference ($C = 2\pi r$) of the orbit. Round the circumference to an integer.
    b. Define the `missionDuration` function to take three parameters - the number of orbits completed, the orbit radius, and the orbital speed. Set the default radius to 2000 km and the default orbital speed to 28000 km/hr.
    c. Calculate how long will it take our animals to complete 5 orbits (time = distance/speed). Round the answer to 2 decimal places, then return the result.
    d. Print, `'The mission will travel ____ km around the planet, and it will take ____ hours to complete.'`
2. Time for an excursion! Code an `oxygenExpended` function to accomplish the following:
    a. Use your `selectRandomEntry` function to select one crew member to perform a spacewalk.
    b. The spacewalk will last for three orbits around the earth. Use `missionDuration` to calculate how many hours the spacewalk will take.
    c. Use the animal's `rate` method to calculate how much oxygen (O_2) they consume during the spacewalk. Round the answer to 3 decimal places.
    d. Return the string, `'__ will perform the spacewalk, which will last __ hours and require __ kg ofoxygen.'` Fill in the blanks with the animal's name, the spacewalk time, and the amount of O_2 used.

12.8.4. Bonus Missions

12.8.4.1. Conserve O_2

Instead of randomly selecting a crew member for the spacewalk, have your program select the animal with the smallest oxygen consumption rate.

12.8.4.2. Fuel Required for Launch

To access the code for this bonus mission, go to repl.it.

A general rule of thumb states that it takes about 9 - 10 kg of rocket fuel to lift 1 kg of mass into low-earth orbit (LEO). For our mission, we will assume a value of 9.5 kg to calculate how much fuel we need to launch our crew into space.

1. Write a `crewMass` function that returns the total mass of the selected crew members rounded to 1 decimal place.
2. The mass of the un-crewed rocket plus the food and other supplies is 75,000 kg. Create a `fuelRequired` function to combine the rocket and crew masses, then calculate and return the amount of fuel needed to reach LEO.
3. Our launch requires a safety margin for the fuel level, especially if the crew members are cute and fuzzy. Add an extra 200 kg of fuel for each dog or cat on board, but only an extra 100 kg for the other species. Update`fuelRequired` to account for this, then round the final amount of fuel UP to the nearest integer.
4. Print `'The mission has a launch mass of ____ kg and requires ____ kg of fuel.'` Fill in the blanks with the calculated amounts.