All about Objects –3 (70 coding exercises)

"*Skill comes from consistent and deliberate practice*." *S. Allen*.

3.2 Object creation

3.2.1.1 Object creation

Exercises

1. What other **key: value** pairs can you think of (example credit card number and a pin)?
2. What is a method?
3. What is the difference between copy by value and copy by reference?

Answers

1. Credit card and pin, lock and key, usb and port
2. A method is a function associated with a value
3. When dealing with object data types, the address in memory of an object is copied over rather than the actual value of the object (copy by value).

3.2.1 Object literal syntax

Exercises

1. Use the object literal notation to declare a variable called `author`.
    - It must have 3 property: values pairs (`name, genre, year`) and one method called `introduction` which prints out the name of the author and the book genre. You can do this with and without the **this** keyword

- Call the introduction method on the author object using the **dot notation**

Answers

1.

```
let author = {
  authorname: 'Toilken',
  genre: 'fantasy',
  year: 2000,
  introduction: function(){
    console.log(`name is ${this.authorname} and the genre is ${this.genre}`)
  }
}
author.introduction();
```

```
 "name is Toilken and the genre is fantasy"
```

3.2.1.6 Object literal enhancement

Exercises

1. Use an object literal enhancement to initialize an object using these variables:

```
let role = 'frontend';
let employed = true;
let vacation = '89';
let message = function(){
 console.log(`${role} position has ${vacation} days of vacation`);
};
```

Answers

1.

```
let employee = {
  role,
 employed,
  vacation,
  message
}
console.log(employee);
employee.message();
```

3.2.1.7 Create dynamic keys in ES6

Exercises

1. Create an object literal using an ES6 object literal enhancements that allows you to add dynamic keys to an object such that the final object is:

```
Object {
Tier-1: "90%",
Tier-2: "80%",
Tier-3: "70%"
}
```

Answers

1.

```
let tier = 'Tier';
let i = 1;
let tierRanks = {
      [tier + '-' + i++]: '90%',
      [tier + '-' + i++]: '80%',
      [tier + '-' + i++]: '70%'
```

```
}


console.log(tierRanks);
```

3.2.2 new() keyword

Exercises

1. Code the following:
    - Use the **new** keyword to instantiate a new `randomNumber` object using the in-built Object constructor function
    - Add a `luckydraw` property whose value is the boolean `true`
    - Add a method called `value` which will generate a random number between `0 -10`
    - Check if random numbers are generated by calling the `value` method on `randomNumber` object

Answers

1.

```
let randomNum = new Object();
randomNum.luckydraw = true;
randomNum.value = function(){
 console.log(Math.floor(Math.random() * 11));
}
randomNum.value();
```

3.2.2.2 User defined object constructor function

Exercises

1. Code the following constructor function:
   - Declare a constructor function called `Dessert`, which takes 4 parameters `name`, `calories`, `flavor` and `helpings`.
   - Define the value of these parameters within the object using the **this** keyword. Add a method called `totalCal` within the constructor function which will `console.log` the number of calories multiplies by the number of helpings
   - Create a `cake` object using the dessert constructor function which has the following properties:
     - `name: Bamboozle`
     - `valories: 1000`
     - `flavour: 'chocolate'`
     - `helpings: 3.5`
   - Call the `totalCal ()` method what is logged?

Answers

1.

```
function Dessert(name, calories, flavor, helpings) {
  this.name = name;
  this.calories = calories;
  this.flavor = flavor;
  this.helpings = helpings;
  this.totalCal = function() {
    console.log(this.calories * this.helpings);
  }
}
let cake = new Dessert('Bamboozle',1000, 'chocolate', 3.5);
```

```
cake.totalCal(); //3500
```

3.2.3 Object.create() method

Exercises

1. Consider the following object:

```
let department = {
  name: 'Entertainment',
  fulltime: true
}
```

- Create a child object called `musicDepartment` using the **Object.create()** method
- Add the following **key : value** pairs to it:

```
employees: 200;
remote:  true;
```

- Delete the `name` property from the `musicDepartment` child object.
- What **key : value** pairs exist inside `musicDepartment` now?

2. What must you do to completely delete the `name` property from the `musicDepartment` child object?

Answers

1.

```
let musicDepartment = Object.create(department, {});
musicDepartment.employees= 200;
musicDepartment.remote = true;
delete musicDepartment.name;
console.log(musicDepartment);
```

```
Object {
```

```
  employees: 200,
  fulltime: true,
  name: "Entertainment",
  remote: true
}
```

2. Delete it from the parent object that it inherits from:

```
delete department.name;
```

3.2.4 Object.assign() method

Exercises

1. SuperStore a company competing with Walmart has hired you! They would like to have a central source of truth for all child companies that will be part of their SuperStore. Create a `superstore` object and copy over all the key value pairs from the `food`, `hardware` and `clothing` objects using the **Object.assign()** method:

```
let food = {
  name: 'Gill SuperStore',
  locations: ['Albuquerque', 'Orlando', 'Toronto']
}
let hardware = {
  name: 'Supermax Hardware',
  locations: ['Cairo']
}
let clothing = {
  name: 'Cloth mania',
  locations: ['Vietnam', 'Jakarta']
};
```

Note: Keep in mind that that property names of the source objects must have different names, so perhaps you should differentiate between the `name` *and*

`locations` *property names for all three source objects by changing them (Hint:* `name1`*,* `name2`*..)*

Answers

1.

```
 let food = {
  name1: 'Gill SuperStore',
  locations1: ['Albuquerque', 'Orlando', 'Toronto']
}
let hardware = {
  name2: 'Supermax Hardware',
  locations2: ['Cairo']
}
let clothing = {
  name3: 'Cloth mania',
  locations3:

  ['Vietnam', 'Jakarta']
}
let superstore = Object.assign({}, food, hardware,
clothing)
console.log(superstore);
```

```
[object Object] {
  locations1: ["Albuquerque", "Orlando", "Toronto"],
  locations2: ["Cairo"],
  locations3: ["Vietnam", "Jakarta"],
  name1: "Gill SuperStore",
  name2: "Supermax Hardware",
  name3: "Cloth mania"
}
```

3.2.5 ES6 Classes

Exercises

1. A class is a special type of function declared using the **Class** keyword: True or False?
2. What is the purpose of using a class?
3. Create the following `Player` class:

    - Create a class called `Player` whose constructor takes 3 parameters `energy`, `power`, and `level`

        - Add a method called `powerControl()` to the `Player` class which will calculate subtracting energy from power and return a string `'Remaining power left is' + power`

    - Create an object called `ryu` from the `Player` class with the following values:

```
energy: 200

power: 500

level: 1
```

    - console.log `ryu`, what is displayed?

4. Use the `object.create()` method to create a new object called `chunLi` from the `ryu` object which acts as the parent object.

Answers

1. True
2. Classes introduced in ES6 allow you to create objects easily from a template function.
3.

```
class Power {
  constructor(energy, power, level) {
    this.energy = energy;
    this.power = power;
    this.level = level
  }
```

```
  powerControl() {
    let powerPoints = this.power - this.energy;
  return (`You have these many power points
${powerPoints}`);
  }
}
let ryu = new Power(200, 500, 1);
ryu.powerControl();
console.log(ryu);
```

```
Object {
  energy: 200,
  level: 1,
  power: 500
}
```

4.

```
let chunLi = Object.create(ryu, {});
console.log(chunLi);
```

```
Object {
  energy: 200,
  level: 1,
  power: 500
}
```

3.3 Object iteration

Exercises

1. Iterate through the following object using the **for-in** statement and `console.log` the value of each key:

```
let coffee = {
  roast: 'medium',
  blend: 'Ethopian',
  servings: 100,
  morningMsg: function(){
    console.log(`Ooh! The smell of an ${this.blend} blend in the mornings`)
  }
}
```

2. What method will return an array of the properties inside the `coffee` object?
3. What 3 methods can you use to convert an object to an array and then iterate through the converted array?
4. What does the **Object.values()** method do? Apply it on the `coffee` object. What is returned?
5. Consider the `coffee` object modified as below. What does the **Object.entries()** method do? Apply it on the `coffee` object. What will be the output?

```
let coffee = {
  roast: ['light','medium','dark','extra dark'],
  blend: ['Ethopian', 'Columbian', 'American'],
  servings:['small', 'medium', 'large'],
  morningMsg: function(){
    console.log(`Ooh! The smell of an ${this.blend} blend in the mornings`)
  },
```

```
}
```

6. Check if object `ob` is empty:

```
let ob = {};
```

Answers

1.

```
for(key in coffee){
  console.log(coffee[key]);
}
```

```
"medium"
"Ethopian"
100
function(){
 console.log(`Ooh! The smell of an ${this.blend} blend in the mornings`)
  }
```

2. `Object.keys()`

```
console.log(Object.keys(coffee));
```

```
["roast", "blend", "servings", "morningMsg"]
```

3. 3 methods to convert an object to an array:
    - Object.keys
    - Object.values
    - Object. entries
4. `Object.values()` will iterate through an object and return an array containing the values of the properties in an object
5. The `Object.entries()` method returns and array of arrays. Each inner array contains the property value pairs as array items

```
console.log(Object.entries(coffee));
```

```
[["roast", ["light", "medium", "dark", "extra dark"]],
["blend", ["Ethopian", "Columbian", "American"]],
["servings", ["small", "medium", "large"]],
["morningMsg", function(){
  window.runnerWindow.proxyConsole.log(`Ooh! The smell of
an ${this.blend} blend in the mornings`)
```

6. The following will return `true` for an empty object:

```
console.log(Object.keys(ob).length === 0 &&
ob.constructor === Object)
```

We use the in-built `Object.keys()` method to check if the `length` of the properties is `0`. We also check if the constructor of the object `ob` is Object this will check any wrapper instances.

We can also create a function which uses `hasOwnProperty()` method:

```
function emptyObj(object) {
 for (let key in object) {
  if (object.hasOwnProperty(key))
   return false;
 }
 return true;
}
emptyObj(ob);
```

3.4 this keyword

Exercises

1. Can you explain what **this** is in JavaScript?
2. Consider the following function. What will the value of the **this** keyword be?

```
function log(){
```

```
  console.log(this);
}
log();
```

3. What will the **this** keyword refer to when in strict mode:

```
function log(){
  'use strict'
  console.log(this);
}
log();
```

4. Declare an object called `HR`. The `HR` object has 3 **key : value** pairs and 1 method. The 1st key is called `company` and its value is the string `Zimpak Software`. The second key is called `hiring` whose value is the boolean `false` and the 3rd key is called `employees` and its value is the number `100`. The 4th key is a method called `message`. The method should return a string which prints the name of the company, the number of employees and the value of the `hiring` key if the number of employees is less than or equal to `100`. Else it should return the string `Zimpack is not hiring`. You must use the **this** keyword

5. Carrying on from #4, change the value of `employees` to `20`. Outside of the object declaration. What does the message method return now? Think about what the context of the **this** keyword will be and the call-site of **this**.

6. Consider the following class called `Company`:

```
class Company {
  constructor(name, hiring, employees) {
    this.company = name;
    this.hiring = hiring;
    this.employees = employees;
  }
  message() {
```

```
    if (this.employees >= 400) {
      console.log(`${this.company} is currently not
hiring`);
    } else {
      console.log(`${this.company} is currenly  hiring`)
    }
  }
}
```

- Declare a new object called `moogle` from the `Company` class
- Pass in three arguments for the `name`, `hiring` and `employees` properties
- Call the `message` method on the your newly created object
- What is the value of `this` now?

7. What does **call()** do?

8. Consider the following code block:

```
function restaurant(stars, cuisine) {
console.log(`Welcome to ${this.name}. We have ${stars}
stars and serve ${cuisine} cuisine`);
}
```

- Declare an object and use the `call()` method in such a way that the following message will be logged to the console:

```
"Welcome to Pataya. We have 4 stars and serve Thai
cuisine"
```

9. Consider the following object:

```
let bretLee = {
  name: 'Bret Lee',
  points: 400
}
```

- Invoke the `apply()` method on a function such that the message that is logged to the screen is

```
"Bret Lee is playing in miniLeague with 400 points for this Summer"
```

10. Define the **bind()** method. What does it do?

11. Complete the following:

    - Write a function called `gymMembership` that takes a variable `fee` as a parameter. This function will calculate the total remaining amount of a gym membership fee that has been paid in full for the year. So we want to deduct a certain amount per month, which is unknown as yet. Use the **this** keyword which will refer to the total amount remaining. The function should `console.log` `this.name` and the remaining value after deducting the monthly fees.
    - Declare an object called `sukhi` with a name property whose `name` value is `'sukhi'` and a total `property` whose value is `1000`
    - Bind the `gymMembership` method to the `sukhi` object and pass as an argument the number `100` which is the monthly fees for the gym membership
    - Assign this to a variable called `getFee`
    - Call `getFee()` what is logged to the console?

Answers

1. In JavaScript, **this** is a reference to an object. The object that **this** will refer to depends on whether it is within the global context, inside an object, or within a constructor function.

2. The global window object

3. `undefined`

4.

```
let HR = {
 company: 'Zimpak Software',
 hiring: true,
```

```
 employees: 130,

 message: function() {

  if (this.employees <= 100) {

   console.log(`${this.company} hiring status is
${this.hiring} and it currently has ${this.employees}
employees`);

  } else {

   console.log(`${this.company}is not hiring`);

  }

 }

}

HR.message();
```

5. `HR.employees = 20;`

```
"Zimpak Software hiring status  is true and it currently
has 20 employees"
```

6.

```
let moogle = new Company('Moogle', true, 10)

moogle.message();
```

```
"Moogle is currenly  hiring"
```

- the value of **this** refers to the new object (`moogle`) that is created.

7. Using **call()** you can write methods that can be used on different objects

8.

```
function restaurant(stars, cuisine) {

console.log(`Welcome to ${this.name}. We have ${stars}
stars and serve ${cuisine} cuisine`);

}

let pataya = {
```

```
  name: 'Pataya'
}
restaurant.call(pataya, 4, 'Thai')
```

9.

```
function player(currentLeague, season){
  console.log((`${this.name} is playing in
${currentLeague} with ${this.points} points for this
${season}`))
}
let bretLee = {
  name: 'Bret Lee',
  points: 400
}
player.apply(bretLee, ['miniLeague', 'Summer']);
```

10. This method is similar to `call()`, however it will return a new function and set **this** to a specific object. And like `call()` it can accept comma separated arguments

11.

```
function gymMembership(fee){
    console.log(`${this.name} has a remaining balance of:
${this.total - fee}`);
}

let sukhi = {
  name:'sukhi',
  total:1000
};
let getFee = gymMembership.bind(sukhi, 90);
getFee();
```

```
"sukhi has a remaining balance of: 910"
```

3.5 Prototype and Inheritance

3.5.1 Prototype

Exercises

1. Explain Prototype inheritance in JavaScript
2. How will you query the prototype of the `book` object? And what is returned?

```
let book = {};
```

3. Consider the following object:

```
let pizza = {
  base: 'wheat',
  sauce: 'tomato',
  cheese: 'parmesan'
}
```

- Create a new object called `cheesePizza` whose prototype is the `pizza` object using the **Object.create** method.
- What will be the output of:

```
console.log(cheesePizza);
```

- Use the **Object.getPrototypeOf()** method to access the prototype of the `cheesePizza` object

Answers

1. JavaScript is a prototype based language. This means that a template object's properties and methods can be cloned and extended onto other objects. Therefore, a prototype is an object instance from which other objects can be created. This is known as prototypal inheritance.
2. Using the **Object.getPrototypeOf()** method. It returns the built in Object type as the prototype of the user created object book:.

3.

```
let cheesePizza = Object.create(pizza);

console.log(cheesePizza);
```

```
Object {
  base: "wheat",
  cheese: "parmesan",
  sauce: "tomato"
}
```

```
console.log(Object.getPrototypeOf(cheesePizza));
```

```
Object {
  base: "wheat",
  cheese: "parmesan",
  sauce: "tomato"
}
```

3.5.2 Prototype chain

Exercises

1. Explain the prototype chain

Answers

1. Each object has a prototype. When we look up a property of an object, the object itself will be queried. If the property is not found, then the JavaScript engine will look within the object's prototype for the existence of the property. If still not found, then the prototype object's prototype will be queried. Therefore, forming a chain starting at a user created object and going all the way up to the end of the chain which is **Object.Prototype**.

3.5.3 Prototype Inheritance

Exercises

1. Consider an object called `book` with the following properties and methods:

```
const book = {
  educational: true,
  diagrams: true,
  author: 'J.K',
  discount: 0,
  sale: function(){
    if(this.educational){
      this.discount = 0.5;
    }
  }
}
```

- Declare an object called `scienceFictionBook` and set its `educational` key to the boolean value `false`
- Set the `book` object to be the prototype of the `scienceFictionBook`
- What will be the value of `scienceFictionBook.discount`?

Answers

1.

```
let scienceFictionBook = {
  educational: false,
}
Object.setPrototypeOf(scienceFictionBook, book);
console.log(scienceFictionBook.discount);
```

```
0
```

3.5.3.2 Constructor functions and inheritance

Exercises

1. Consider the following and answer the questions:

```
function SchoolFranchise(accredited, teachers, online){
  this.accredited = true;
  this.teachers = false;
  this.online = true;
}
```

- Create a constructor function called `JuniorHigh` which inherits properties from the constructor function `schoolFranchise()`
- Pass in two parameters to the `JuniorHigh` constructor function: `name` and `type`
- The value of the `type` key will be the string `'Junior High'`
- Create a new object referenced by `let` `huronPublic` passing in the string `'Huron'` as a parameter which reference the school's name
- `console.log(huronPublic)` what **key : value** pairs are returned?

Answers

1.

```
function juniorHigh(name, type){
  SchoolFranchise.call(this);
  this.name = name;
  this.type = 'Junior High'
}
```

```
let huronPublic = new juniorHigh('Huron');
```

```
console.log(huronPublic);
```

```
Object {
  accredited: true,
  name: "Huron",
  online: true,
  teachers: false,
  type: "Junior High"
}
```

3.6 Classes

3.6.1 Class declarations

Exercises

1. What is a class?
2. Declare a class called `Company`

Answers

1. Classes were introduced in ES6 to mimic the class data type found in Java and other object oriented programming languages JavaScript does not have the class type, so we create functions in a way that they behave as classes in order to allow us to easily create objects. A class is a function and when invoked as a constructor will create an instance of that class. Classes are functions except that they are declared with the class keyword instead of the function keyword.

2. Class called `Company:`

```
function Company{}
```

3.6.2 Constructor method

Exercises

1. Have a good look at the following block of code. What will be logged to the console:

```
let maplesyrup = new IcecreamFlavor('Maple Syrup',
false);

class IcecreamFlavor {

  constructor(name, toppings) {

   this.name = name;

   this.toppings = toppings;

 }

}

console.log(maplesyrup);
```

2. Consider the following class `Company`:

```
class Company {

  constructor(name, funding, employees) {

   this.name = name;

   this.funding = funding;

   this.employees = employees;

 }

}
```

- Can you add another constructor to this class?
- Instantiate a new object using the `Company` class referenced by `let zimbaPay` that has the following arguments passed in to it: `'Zimba Pay', 1000000, 50`
- Use the **Object.getPrototypeOf()** method to find the prototype of the `zimbaPay` object

Answers

1. Class declarations are not hoisted. Therefore, classes must be defined before being used else it will result in a ReferenceError, as hoisting does not apply here. Also keep in mind that the body of a Class is executed in strict mode.

```
ReferenceError: can't access lexical declaration
`IcecreamFlavor' before initialization
```

2. No you cannot add another constructor to the class.

```
let zimbaPay = new Company('Zimba Pay', 1000000, 50);

console.log(zimbaPay);

console.log(Object.getPrototypeOf(zimbaPay));
```

```
Object {
  employees: 50,
  funding: 1000000,
  name: "Zimba Pay"
}


constructor: class Company { constructor(name, funding,
employees) }
```

3.6.3 Instance method

Exercises

1. Complete the following:
   - Add an instance method called `equity()` to the following `Company` class which will return `10%` of the total `funding` that a company receives
   - Instantiate a new object from the `Company` class called `purpleMoon`. And pass in the following parameters to it: `'Purple Moon', 5000000, 50`
   - Declare a global variable called `equity` whose value is the `equity()` method called on the `purpleMoon` object

Answers

1.

```
class Company {
  constructor(name, funding, employees) {
   this.name = name;
```

```
   this.funding = funding;
   this.employees = employees;
 }
  equity(){
    return(0.10 * this.funding);
  }
}
let purpleMoonEnterprise = new Company('Purple Moon',
5000000, 50);
let equity = purpleMoonEnterprise.equity();


console.log(equity);
```

```
500000
```

3.6.4 Static method

Exercises

1. Define a static method
2. Take as an example the following class:

```
class MusicLabel{
  constructor(name,genre) {
    this.name = 'Avocado Label';
    this.genre = genre;
  }
  static labelMotto(){
    console.log(`Gimme some ${this.name}`)
  }
}
```

A new object declared as `artist` is instantiated from this class :

```
let artist = new MusicLabel('Moozic
Records','Jazz','Richie Zoo');
```

Calling the `labelMotto()` static method on the artist object returns an error:

```
artist.labelMotto();
```

```
  TypeError: artist.labelMotto is not a function
```

- What can you do so that the `labelMotto()` static method inside the `MusicLabel` class becomes available to the artist object?

Answers

1. *A static method belongs to a class and can only be accessed by the class and not the object instance created by a Class*. The **static** keyword must be used to declare a static method
2. Remove the **static** keyword from in-front of the `labelMotto()` label:

```
class MusicLabel{

  constructor(name,genre, artist) {

    this.name = name

    this.genre = genre;

    this.artist = artist;

  }

   labelMotto(){

    console.log(`Gimme some ${this.name}`);

  }

}

let artist = new MusicLabel('Moozic
Records','Jazz','Richie Zoo');

artist.labelMotto();
```

```
"Gimme some Moozic Records"
```

3.6.5 Public and private fields

3.6.5.1 Public instance fields

Exercises

1. Describe a public instance field
2. Create a class called `Player`. The class has four public instance fields `name, score, punches, throws`. Initialize these fields to starting values of:

```
name='';

punches = 10;

throw =3;

score;
```

- The constructor method returns the sum of `punches` and `throws` and assigns the value to the `score` field
- Code a public method `startMessage()` which will console.log the message `` `Are you ready to kung-foo ${this.name}?` ``
- Instantiate a new object from this class and declare it as `fooFighter`
- Console.log the `fooFighter` instance, what do you see logged?
- Call the `startMessage()` method on the `fooFighter` instance what is logged?
- What should we change in this class so that instead of the following message from the `startMessage()` method:

```
"Are you ready to kung-foo Player?"
```

- We see a more personalized message, corresponding to a player's actual username, for example:

```
"Are you ready to kung-foo Foo Fighter?"
```

Answers

1. Public instance fields exist on every instance created from a class.
2.

```
class Player{

  name='Player';
```

```
  punches = 10;
  throw =3;
  score;
  constructor(){
   this.score = this.punches + this.throw
  }
  startMessage(){
    console.log(`Are you ready to kung-foo
${this.name}?`)
  }
}
let fooFighter = new Player();
console.log(fooFighter);
fooFighter.startMessage();
```

```
Object {
  name: "Player",
  punches: 10,
  score: 13,
  throw: 3
}
"Are you ready to kung-foo Player?"
```

```
class Player{
  name='';
  punches = 10;
  throw =3;
  score;
```

```
  constructor(name){
   this.name = name;
   this.score = this.punches + this.throw
  }
  startMessage(){
    console.log(`Are you ready to kung-foo
${this.name}?`)
  }
}
let fooFighter = new Player('Foo Fighter');
console.log(fooFighter)
fooFighter.startMessage();
```

3.6.5.2 Private instance fields

Exercises

1. Consider the preceding code block and answer the questions:

```
class InternalDetails {
  #gross_profit;
  #net_profit;
  #tax;
  #expenses;
  constructor(gross_profit, expenses, tax, net_profit) {
    this.#gross_profit  = gross_profit;
    this.#expenses = expenses;
    this.#tax = tax;
  }
  getNet() {
  this.#net_profit= this.#gross_profit - (this.#expenses
+ this.#tax)
```

```
    return(this.#net_profit)
  }
}
let mooCompany = new InternalDetails(30,1,1,0);
mooCompany.getNet();
```

- What will `mooCompany.getNet()` return
- What happens when you try and access `mooCompany.#gross_profit`?
- Can you set the value of the private `#net_profit` field of the `mooCompany` class instance

Answers

1. `mooCompany.getNet()` returns

```
28
```

Cannot access `mooCompany.#gross_profit`:

```
Uncaught SyntaxError: Private field '#gross_profit' must be declared in an enclosing class
```

- Setting the value of the private `#net_profit` field will return the following:

```
Uncaught SyntaxError: Private field '#net_profit' must be declared in an enclosing class
```

3.6.5.3 Public static fields

Exercises

1. How is a static public field declared?
2. What purpose do static fields have?
3. Declare a class called `WeddingPlanner`. The class has 2 `static` public fields:

```
llc = 'Wedding Gee LLC'
tax_number ='319000'
```

4. The class has 3 private `instance` fields:

```
company = 'Wedding Gee';

office ='101 Plum Street, Chicago';

planner = 'Keanna Rose';
```

5. The class will have 2 public instance fields:

```
client_name;

client_budget;
```

- Code a constructor function that sets the `client_name` and `client_budget` fields to an instance of an object created from the `WeddingPlanner` class using the **this** keyword
- Code an instance function called `welcomeMessage` that will `console.log` the following message:

```
console.log(`Hi, ${this.client_name}! Welcome to
${this.#company}, I am your planner ${this.#planner}.
Please confirm that your budget is
${this.client_budget}`)
```

- Instantiate a new object which will represent a client using the `WeddingPlanner` class. Call it `missSpadina`. Pass in the parameters `( 'J. Spadina', 50000)` to this new instance
- Call the `welcomeMessage()` method on the `missSpadina` instance
- Query the prototype of the `missSpadina` instance

Answers

1. With the `static` keyword.
2. Static fields reference values that will be consistent across all instances of the class.
3. *Answer 3-5*

```
class WeddingPlanner {

  #company = 'Wedding Gee';

  #office = '101 Plum Street, Chicago';

  #planner = 'Keanna Rose';
```

```
  static llc = 'Wedding Gee LLC'

  static tax_number = '319000'

  client_name;

  client_budget;

  constructor(client_name, client_budget) {

    this.client_name = client_name;

    this.client_budget = client_budget;

  }

  welcomeMessage(company, planner) {

    console.log(`Hi, ${this.client_name}! Welcome to
${this.#company}, I am your planner ${this.#planner}.
Please confirm that your budget is
${this.client_budget}`)

  }

}

let missSpadina = new WeddingPlanner('J. Spadina',
50000);

missSpadina.welcomeMessage();
```

```
Hi, J. Spadina! Welcome to Wedding Gee, I am your planner
Keanna Rose. Please confirm that your budget is 50000
```

```
console.log(Object.getPrototypeOf(missSpadina));
```

```
constructor: class WeddingPlanner
```

3.6.5.4 Private static fields

1. What function do private static fields serve?
2. Using the `DonutGiveaway` class instantiate 2 instances (`donut1` and `donut2`). Log the object instances, to the console what is logged? Also query the `keys` of the `donut1` object

```
class DonutGiveaway {
  static #max_instances = 5;
  static #instances = 0;
  flavor;
  constructor(flavor) {
    DonutGiveaway.#instances++;
    if (DonutGiveaway.#instances >
      DonutGiveaway.#max_instances) {
      throw new Error(
        'Unable to create a new donut instance'
      )
    } else {
      this.flavor = flavor;
    }
  }
}
```

Answers

1. Private static fields are useful in order to hide implementation details that you would like to remain unchanged.
2.

```
let donut1 = new DonutGiveaway('Java JavaScript');
let donut2 = new DonutGiveaway('Snappy Semicolon');
console.log(donut1);
console.log(donut2);
console.log(Object.keys(donut1));
```


```
["flavor"]
```

3.6.6 Inheritance with extends

Exercises

1. Have a look at the following class and answer the questions:

```
class Dog{

  constructor(legs, tail){

    this.legs = 4;

    this. tail = 1;

  }

}
```

- Construct child class which extends `Dog`, called `Breed`
- The `Breed` child class has the `breed` property set to the string `'Boston Terrier'`
- Create a new instance of the `Breed` class called `rocko`
- Console.log `rocko`, what all **key : value** pairs are logged:

Answers

1.

```
class Breed extends Dog{

breed = 'Boston Terrier';

}

let rocko = new Breed();

console.log(rocko);
```

```
Object {
  breed: "Boston Terrier",
  legs: 4,
  tail: 1
}
```

3.6.7 Inheritance with super keyword

Exercises

1. Consider the following parent class `Dog` and answer the questions:

```
class Dog{
  constructor(legs, tail){
    this.legs = 4;
    this. tail = 1;
  }
}
```

- Create a child class called `Breed` which extends the `Dog` class
- Use the appropriate fields such that when an instance is logged to the console, the following is returned:

```
Object {
  breed: "Boston terrier",
  legs: 4,
  tail: 1
}
```

2. Create a class called `MusicLabel`:

- `MusicLabel` has a constructor method which takes one parameter called `label_name`
- Set the value of the `label_name` field to `'Avocado Label'`

- The `MusicLabel` class has a static method called `LabelMotto()` which will `console.log` the string:

```
'Hello and welcome!';
```

- Create a sub-class called `Jazz` which extends the `MusicLabel` class
- It's `constructor()` method takes `artist`, `label_name` and `genre` as parameters
- Access the `genre` field which is present in the parent `MusicLabel` class
- Using `this` set the values of the `artist` field to the value passed in to the instance
- Set `this.genre` to the string `'Jazz';`
- Create an instance of the Jazz class which is called `richieZoo`, using the `new` keyword and pass in the string `'Richie Zoo'` as a parameter which is the artist's name
- `console.log` the `richieZoo` instance

Answers

1.

```
class Dog{
  constructor(legs, tail){
    this.legs = 4;
    this. tail = 1;
  }
}
class Breed extends Dog{
  constructor(breed){
    super();
   this.breed = breed;
  }
}
```

```
let doggie = new Breed('Boston terrier');
console.log(doggie);
```

2.

```
class MusicLabel{
  constructor(label_name) {
    this.label_name = 'Avocado Label';
  }
  static LabelMotto(){
    console.log('Hello and welcome!')
  }
}
class Jazz extends MusicLabel {
constructor(artist, label_name, genre) {
  super(label_name);
  this.artist = artist;
  this.genre ='Jazz';
  }
}
let richieZoo = new Jazz('Richie Zoo');
console.log(richieZoo);
```

```
Object {
  artist: "Richie Zoo",
  genre: "Jazz",
  label_name: "Avocado Label"
}
```

3.6.8 InstanceOf

Exercises

1. What does the **instanceof** operator do?

Answers

1. In order to check if an object is an instance of a specific class use the **instanceOf** operator. This will check if an object is an instance of that particular class.

3.6.9 Constructor property

Exercises

1. What does the **constructor** property of an object indicate?

Answers

1. It determines the exact class of an object instance

3.6 Accessors: Getters and Setters

Exercises

1. What are accessor properties?
2. What are the 2 types of accessor properties:
3. Consider the code below:

```
let donut ={
  units: 100,
  flavors:['strawberry','oreo','java'],
  price: 5.99,
}
```

- Write a getter called `getDonutFlavor` using the default method syntax
- Inside the method define a variable called `flavorList` which references the array in `flavors` property

- For each element in the `flavors` array log to the console the flavor on a new line
- Call `donut.getDonutFlavor()` what is returned?

4. Using the default method syntax write a setter called `setDonutFlavor` for the same `donut` object in question #3. The `setDonutFlavor` function will receive a string parameter and push it into the array of the `donut.flavors` property. Call the `setDonutFlavor()` method and pass in the string `'chocolate'`. The value of `donut.flavors` property should be

```
['strawberry','oreo','java', 'chocolate']
```

5. Replace the `setDonutFlavor` method created in question #4 via the default method syntax with the `set` keyword. And pass in the string `'chocolate'` to the *donutFlavor* setter. Console.log `donut.flavors`. It should be:

```
['strawberry','oreo','java', 'chocolate']
```

6. Replace the `getDonutFlavor` method created in question #3 via the default method syntax with the `get` keyword.
7. Consider the following empty `gamingPC` object:

```
let gamingPC ={};
```

The `graphicsCard` property is defined using the `Object.defineProperty()` method:

```
Object.defineProperty(gamingPC, 'graphicsCard', {
  configurable: false,
  enumerable: false,
  writable: true,
  value: 'RTX2060'
});
```

- Using `Object.defineProperty` method, define a setter for the `gamingPC` object called `newCard`. This setter will assign a new value to the `graphicsCard` property
- Assign `'GeForce RTX 2070 Super'` to the `newCard` setter

- Console.log `gamingPC.graphicsCard`. The output should be:

```
"GeForce RTX 2070 Super"
```

8. Write a getter function which will retrieve the value of the `graphicsCard` property. Call the getter `getCard`

Answers

1. Accessor properties are functions that will execute on either getting or setting a value. They are computed *properties*
2. Getters and setters.
3.

```
let donut ={
  units: 100,
  flavors:['strawberry','oreo','java'],
  price: 5.99,
  getDonutFlavor: function(){
  let flavorList= donut.flavors;
flavorList.forEach(function(flavor){
     console.log(flavor);
   })
  },
};
```

```
"strawberry"
"oreo"
"java"
```

4.

```
let donut ={
  units: 100,
  flavors:['strawberry','oreo','java'],
```

```
  price: 5.99,
  setDonutFlavor(x){
    donut.flavors.push(x)
  }
};
donut.setDonutFlavor('chocolate');
console.log(donut.flavors);
```

```
["strawberry", "oreo", "java", "chocolate"]
```

5.

```
let donut ={
  units: 100,
  flavors:['strawberry','oreo','java'],
  price: 5.99,
  set donutFlavor(x){
    donut.flavors.push(x)
  }
};
donut.donutFlavor = 'chocolate';
console.log(donut.flavors);
```

```
["strawberry", "oreo", "java", "chocolate"]
```

6.

```
let donut ={
  units: 100,
  flavors:['strawberry','oreo','java'],
  price: 5.99,
```

```
  get flavor(){
    let flavorList= donut.flavors;
    flavorList.forEach(function(flavor){
     console.log(flavor);
   })
  }
};
donut.flavor;
```

```
"strawberry"
"oreo"
"java"
```

7.

```
Object.defineProperty(gamingPC, "newCard", {
  set: function(newCard) {
    this.graphicsCard = newCard;
  }
});
gamingPC.newCard ='GeForce RTX 2070 Super';
console.log(gamingPC.graphicsCard);
```

```
"GeForce RTX 2070 Super"
```

8.

```
Object.defineProperty(gamingPC, "getCard", {
  get: function() {
    console.log(this.graphicsCard);
  }
```

```
});

gamingPC.getCard;
```

```
"GeForce RTX 2070 Super"
```

3.8 Copying objects with shallow and deep copies

Exercises

1. What is the difference between shallow and deep copying?
2. What are some ways to create a shallow copy?
3. Create a copy of the following object using the **spread** operator. Name the copied object: `toronto_clone`:

```
let city ={
  name: 'Toronto',
  coordinates: '43.6532° N, 79.3832° W',
  streets: {
  North: 'Bathurst',
  South: 'Queens',
  West: 'Bathurst',
  East: 'Spadina'
  },
  population: 3190000,
};
```

- Once you create a copy of the `city` object, check whether your clone contains all the key: value pairs from the `city` object by `console.logging` it
- Change the value of the `North` property in the nested streets object to the string `'St.George'` in the copied `toronto_clone` object
- Change the value of `name` property in your copy object to `'Toronto clone'`

- Console.log both the `city` and `toronto_clone` objects
- Can you explain why the two objects have the same value for the `North` property in nested `streets` object but not for the name property?

4. Have a look at the following code and answer the question:

```
let a = {};

let b = {};

console.log(a === b); // true or false
```

5. For the following object, complete the following tasks:

```
let a ={
  one: 'one',
  two: 'two',
  three: 'three'
}
```

   - Using `Object.assign()` copy over the contents of object `a` to a another object declared as `b`
   - Change the value of `b.one` to the string `'zoo'`
   - What is `a.one` ?

Answers

1. In a shallow copy, a new object is created which has the exact values as the source object. Primitive values are copied by value whereas if a field is of a reference type (an array or another object) then the reference/memory address is copied over to the newly created target object. *In a shallow copy the source object and target object share the same memory address.*
   In a deep copy, all the values (primitive and reference types) are duplicated into a source object and allocated new memory locations. Therefore, *the memory address of the target and source objects is different*. Changing the source object after a deep copy will not affect the target object.

2. Some ways to shallow copy are:

   - Iteration using a for-in loop

- Spread operator
- Object.assign()

3.

```
let toronto_clone = {...city};

//console.log(toronto_clone);

toronto_clone.streets.North = 'St.George';

toronto_clone.name ='Toronto clone';

console.log(toronto_clone)

console.log(city)
```

`toronto_clone` when cloned (no changes as yet)

```
[object Object] {
  coordinates: "43.6532° N, 79.3832° W",
  name: "Toronto",
  population: 3190000,
  streets: [object Object] {
    East: "Spadina",
    North: "Bathurst",
    South: "Queens",
    West: "Bathurst"
  }
}
```

`toronto_clone` after the changes to the `North` and `name` properties

```
[object Object] {
  coordinates: "43.6532° N, 79.3832° W",
  name: "Toronto clone",
  population: 3190000,
  streets: [object Object] {
    East: "Spadina",
    North: "St.George",
    South: "Queens",
    West: "Bathurst"
  }
}
```

`city` object after the changes to the `North` and `name` properties in the `toronto_clone` object

```
[object Object] {
  coordinates: "43.6532° N, 79.3832° W",
  name: "Toronto",
  population: 3190000,
  streets: [object Object] {
    East: "Spadina",
    North: "St.George",
    South: "Queens",
    West: "Bathurst"
  }
}
```

4. `false`. Both objects are empty however, they point to different locations in memory.
5.

```
let b = Object.assign({},a);

b.one = 'zoo';

console.log(a.one);
```

```
"one"
```

3.8.2 Copying objects with deep copy

Exercises

1. Make a deep copy called `truck_copy` of the following object:

```
let peterbilt = {
  company: 'Peterbilt Motors Company',
  type: 'on-highway',
  class_number:8,
  load:{
    light: '10 tonne',
    medium: '20 tonne',
    heavy: '30 tonne'
```

```
  },
}
```

Answers

1.

```
let truck_copy = JSON.parse(JSON.stringify(peterbilt));
```