

Module Basics



File base.js:

```
console.log('in base.js');
```

Question

What shows in the console?

Answer

in base.js

File base.js:

```
projectId = 99;
```

```
console.log('in base.js');
```

Question

What shows in the console?

Answer

Runtime Error:

Variable undefined in strict mode

File base.js:

```
import { projectId } from 'module1.js';
```

```
console.log(projectId);
```

File module1.js:

```
export let projectId = 99;
```

Question

What shows in the console?

Answer

99

File base.js:

```
import { projectId, projectName } from 'module1.js';  
console.log(`${projectName} has id: ${projectId}`);
```

File module1.js:

```
export let projectId = 99;  
export let projectName = 'BuildIt';
```



What shows in the console?



BuildIt has id: 99

File base.js:

```
import { projectId as id, projectName } from 'module1.js';  
console.log(`${projectName} has id: ${id}`);
```

File module1.js:

```
export let projectId = 99;  
export let projectName = 'BuildIt';
```



What shows in the console?



BuildIt has id: 99

File base.js:

```
import { projectId as id, projectName } from 'module1.js';  
console.log(`${projectName} has id: ${projectId}`);
```

File module1.js:

```
export let projectId = 99;  
export let projectName = 'BuildIt';
```



What shows in the console?



Runtime error:
projectId is undefined

File base.js:

```
console.log('starting in base');  
import { projectId } from 'module1.js';  
console.log('ending in base');
```

File module1.js:

```
export let projectId = 99;  
console.log('in module1');
```



What shows in the console?



in module1
starting in base
ending in base

File base.js:

```
import someValue from 'module1.js';  
console.log(someValue);
```

File module1.js:

```
export let projectId = 99;  
let projectName = 'BuildIt';  
export default projectName;
```



What shows in the console?



BuildIt

File base.js:

```
import { default as myProjectName } from 'module1.js';  
console.log(myProjectName);
```

File module1.js:

```
export let projectId = 99;  
let projectName = 'BuildIt';  
export default projectName;
```



What shows in the console?



BuildIt

File base.js:

```
import someValue from 'module1.js';  
console.log(someValue);
```

File module1.js:

```
let projectId = 99;  
let projectName = 'BuildIt';  
export { projectId, projectName };
```



What shows in the console?



undefined

File base.js:

```
import someValue from 'module1.js';  
console.log(someValue);
```

File module1.js:

```
let projectId = 99;  
let projectName = 'BuildIt';  
export { projectId as default, projectName };
```



What shows in the console?



99

File base.js:

```
import * as values from 'module1.js';  
console.log(values);
```

File module1.js:

```
let projectId = 99;  
let projectName = 'BuildIt';  
export { projectId, projectName };
```



What shows in the console?



```
{ projectId: 99,  
  projectName: 'BuildIt'}
```

Named Exports in Modules



File base.js:

```
import { projectId } from 'module1.js';  
projectId = 8000;  
console.log(projectId);
```

File module1.js:

```
export let projectId = 99;
```



What shows in the console?



Runtime error: projectId is read-only

File base.js:

```
import { project } from 'module1.js';  
project.projectId = 8000;  
console.log(project.projectId);
```

File module1.js:

```
export let project = {  
  projectId: 99  
};
```



What shows in the console?



8000

File base.js:

```
import { project, showProject } from 'module1.js';  
project.projectId = 8000;  
showProject();  
console.log(project.projectId);
```

File module1.js:

```
export let project = { projectId: 99 };  
  
export function showProject() {  
  console.log(project.projectId);  
};
```



What shows in the console?



8000
8000

File base.js:

```
import { showProject, updateFunction } from 'module1.js';  
showProject();  
updateFunction();  
showProject();
```

File module1.js:

```
export function showProject() { console.log('in original'); }  
export function updateFunction() {  
  showProject = function () { console.log('in updated'); };  
};
```



What shows in the console?



in original
in updated

Class Fundamentals



```
class Task {
```

```
}
```

```
console.log(typeof Task);
```

Question

What shows in the console?

Answer

function

```
class Task {  
  
}  
let task = new Task();  
console.log(typeof task);
```

Question

What shows in the console?

Answer

object

```
class Task {  
  
}  
let task = new Task();  
console.log(task instanceof Task);
```

Question

What shows in the console?

Answer

true

```
class Task {  
  showId() {  
    console.log('99');  
  }  
}  
let task = new Task();  
task.showId();
```

Question

What shows in the console?

Answer

99

```
class Task {  
  showId() {  
    console.log('99');  
  }  
}  
  
let task = new Task();  
console.log(task.showId === Task.prototype.showId);
```



What shows in the console?



true


```
class Task {  
  constructor() {  
    console.log('constructing Task');  
  }  
  showId() {  
    console.log('99');  
  }  
}  
let task = new Task();
```



What shows in the console?



constructing Task

```
class Task {  
  constructor() {  
    console.log('constructing Task');  
  },  
  showId() {  
    console.log('99');  
  }  
}  
let task = new Task();
```



What shows in the console?



Syntax error

```
class Task {  
  let taskId = 9000;  
  constructor() {  
    console.log('constructing Task');  
  }  
  showId() {  
    console.log('99');  
  }  
}  
let task = new Task();
```



What shows in the console?



Syntax error

```
let task = new Task();
```

```
class Task {  
  constructor() {  
    console.log('constructing Task');  
  }  
}
```



What shows in the console?



Error: Use before declaration

```
let newClass = class Task {  
  constructor() {  
    console.log('constructing Task');  
  }  
};  
  
new newClass();
```



What shows in the console?



constructing Task

```
let Task = function () {  
  console.log('constructing Task');  
};  
let task = {};  
Task.call(task);
```



What shows in the console?



constructing Task

```
class Task {  
  constructor() {  
    console.log('constructing Task');  
  }  
};  
let task = {};  
Task.call(task);
```



What shows in the console?



Error: class constructor cannot be called with the new keyword

```
function Project() { };  
  
console.log(window.Project === Project);
```



What shows in the console?



true


```
class Task { }
```

```
console.log(window.Task === Task);
```



What shows in the console?



false

extends and super



```
class Project {  
  constructor() {  
    console.log('constructing Project');  
  }  
}
```

```
class SoftwareProject extends Project {  
}
```

```
let p = new SoftwareProject();
```



What shows in the console?



constructing Project

```
class Project {  
  constructor(name) {  
    console.log('constructing Project: ' + name);  
  }  
}
```

```
class SoftwareProject extends Project {  
}
```

```
let p = new SoftwareProject('Mazatlan');
```



What shows in the console?



constructing Project: Mazatlan

```
class Project {  
  constructor() {  
    console.log('constructing Project');  
  }  
}  
class SoftwareProject extends Project {  
  constructor() {  
    super();  
    console.log('constructing SoftwareProject');  
  }  
}  
let p = new SoftwareProject();
```



What shows in the console?



constructing Project
constructing SoftwareProject

```
class Project {  
  constructor() {  
    console.log('constructing Project');  
  }  
}  
class SoftwareProject extends Project {  
  constructor() {  
    //super();  
    console.log('constructing SoftwareProject');  
  }  
}  
let p = new SoftwareProject();
```



What shows in the console?



ReferenceError: this is not defined

```
class Project {  
  //constructor() {  
    // console.log('constructing Project');  
  }  
}  
class SoftwareProject extends Project {  
  constructor() {  
    //super();  
    console.log('constructing SoftwareProject');  
  }  
}  
let p = new SoftwareProject();
```



What shows in the console?



ReferenceError: this is not defined

```
class Project {  
  constructor() {  
    console.log('constructing Project');  
  }  
}  
class SoftwareProject extends Project {  
  constructor() {  
    super();  
    console.log('constructing SoftwareProject');  
  }  
}  
let p = new SoftwareProject();
```



What shows in the console?



constructing Project
constructing SoftwareProject


```
class Project {  
  getTaskCount() {  
    return 50;  
  }  
}  
class SoftwareProject extends Project {  
}  
let p = new SoftwareProject();  
console.log(p.getTaskCount());
```



What shows in the console?



50

```
class Project {  
  getTaskCount() {  
    return 50;  
  }  
}  
  
class SoftwareProject extends Project {  
  getTaskCount() {  
    return 66;  
  }  
}  
  
let p = new SoftwareProject();  
console.log(p.getTaskCount());
```



What shows in the console?



66

```
class Project {  
  getTaskCount() {  
    return 50;  
  }  
}  
  
class SoftwareProject extends Project {  
  getTaskCount() {  
    return super.getTaskCount() + 6;  
  }  
}  
  
let p = new SoftwareProject();  
console.log(p.getTaskCount());
```



What shows in the console?



56

```
let project = {  
  getTaskCount() { return 50; }  
};  
let softwareProject = {  
  getTaskCount() {  
    return super.getTaskCount() + 7;  
  }  
}  
Object.setPrototypeOf(softwareProject, project);  
  
console.log(softwareProject.getTaskCount());
```



What shows in the console?



57

Properties for Class Instances



```
class Project {  
  constructor() { this.location = 'Mazatlan'; }  
}  
class SoftwareProject extends Project {  
  constructor() {  
    super();  
  }  
}  
let p = new SoftwareProject();  
console.log(p.location);
```



What shows in the console?



Mazatlan

```
class Project {  
  constructor() { let location = 'Mazatlan'; }  
}  
class SoftwareProject extends Project {  
  constructor() {  
    super();  
  }  
}  
let p = new SoftwareProject();  
console.log(p.location);
```



What shows in the console?



undefined

```
class Project {  
  constructor() { this.location = 'Mazatlan'; }  
}  
class SoftwareProject extends Project {  
  constructor() {  
    super();  
    this.location = this.location + ' Beach';  
  }  
}  
let p = new SoftwareProject();  
console.log(p.location);
```



What shows in the console?



Mazatlan Beach

Static Members



```
class Project {  
    static getDefaultId() {  
        return 0;  
    }  
}  
console.log(Project.getDefaultId());
```



What shows in the console?



0

```
class Project {  
    static getDefaultId() {  
        return 0;  
    }  
}  
  
var p = new Project();  
console.log(p.getDefaultId());
```



What shows in the console?



Error: Object doesn't support
property or method
getDefaultId

```
class Project {  
  static let id = 0;  
}  
console.log(Project.id);
```



What shows in the console?



Syntax Error: (expected

```
class Project {  
  
}  
Project.id = 99;  
console.log(Project.id);
```



What shows in the console?



99

new.target



```
class Project {  
  constructor() {  
    console.log(typeof new.target);  
  }  
}  
  
var p = new Project();
```



What shows in the console?



function

```
class Project {  
  constructor() {  
    console.log(new.target);  
  }  
}  
  
var p = new Project();
```



What shows in the console?



```
constructor() {  
  console.log(new.target); }
```



```
class Project {  
    constructor() {  
        console.log(new.target);  
    }  
}  
  
class SoftwareProject extends Project {  
    constructor() {  
        super();  
    }  
}  
  
var p = new SoftwareProject();
```



What shows in the console?



```
constructor() {  
    super();  
}
```

```
class Project {  
    constructor() {  
        console.log(new.target);  
    }  
}  
class SoftwareProject extends Project {  
}  
var p = new SoftwareProject();
```



What shows in the console?



```
constructor(...args) {  
    super(...args);  
}
```

```
class Project {  
    constructor() {  
        console.log(new.target.getDefaultId());  
    }  
}  
class SoftwareProject extends Project {  
    static getDefaultId() { return 99; }  
}  
var p = new SoftwareProject();
```



What shows in the console?



99

Summary



Module Basics

File base.js:

```
import { projectId } from 'module1.js';  
console.log(projectId);
```

File module1.js:

```
export let projectId = 99;
```



Summary



Named Exports

File base.js:

```
import { project } from 'module1.js';  
project.projectId = 8000;  
console.log(project.projectId);
```

File module1.js:

```
export let project = {  
  projectId: 99  
};
```



Summary



Classes

```
class Task {  
  constructor() {  
    console.log('constructing Task');  
  }  
  showId() {  
    console.log('99');  
  }  
}  
let task = new Task();
```



Summary



extends and super

```
class Project {  
    constructor() {  
        console.log('constructing Project');  
    }  
}  
class SoftwareProject extends Project {  
    constructor() {  
        super();  
        console.log('constructing SoftwareProject');  
    }  
}  
let p = new SoftwareProject();
```



Summary



Constructor Function Properties

```
class Project {  
    constructor() { this.name = 'Mazatlan'; }  
}  
class SoftwareProject extends Project {  
    constructor() {  
        super();  
    }  
}  
let p = new SoftwareProject();  
console.log(p.name);
```



Summary



Static Members

```
class Project {  
    static getDefaultId() {  
        return 0;  
    }  
}  
console.log(Project.getDefaultId());
```



Summary



new.target

```
class Project {  
    constructor() {  
        console.log(new.target);  
    }  
}  
class SoftwareProject extends Project {  
}  
var p = new SoftwareProject();
```

