

100 Advanced Vanilla JavaScript Multiple Choice Questions with Answers and Explanations

JavaScript is a versatile and powerful language used widely for web development. To truly master JavaScript, especially its advanced concepts, it's essential to understand not just the syntax but the underlying principles and behaviors of the language. Below are 100 advanced multiple-choice questions designed to challenge your knowledge and help you gain a deeper understanding of JavaScript. Each question is accompanied by detailed explanations of the correct answers.

Questions and Answers

Question: What will the following code output?

```
console.log(1 + "1" - 1);
```

- a) "10"
- b) 10
- c) 0
- d) "11"

Answer: c) 0

Explanation: In JavaScript, the + operator with a string and a number results in concatenation, making 1 + "1" equal to "11". When "11" is subtracted by 1, JavaScript implicitly converts "11" to a number, resulting in 11 - 1, which is 10.

Question: What will be the output of the following code?

```
(function() {  
  var a = b = 5;  
})();
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
console.log(b);
```

- a) ReferenceError
- b) 5
- c) undefined
- d) TypeError

Answer: b) 5

Explanation: The expression `var a = b = 5` is evaluated as `b = 5` followed by `var a = b`. Here, `b` becomes a global variable because it is not declared with `var`. Hence, `console.log(b)` outputs 5.

Question: What is the result of the following code?

```
console.log([] + []);
```

- a) ""
- b) "[]"
- c) "[object Object][object Object]"
- d) NaN

Answer: a) ""

Explanation: In JavaScript, the `+` operator between two arrays results in their string representation being concatenated. The string representation of an empty array is an empty string, so `[] + []` results in "".

Question: What will the following code output?

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

- a) "number"
- b) "NaN"
- c) "undefined"
- d) "object"

Answer: a) "number"

Explanation: NaN stands for "Not-a-Number", but its type in JavaScript is actually number.

Question: What will be the output of the following code?

```
console.log(0.1 + 0.2 === 0.3);
```

a) true

b) false

Answer: b) false

Explanation: Due to floating-point precision issues in JavaScript, $0.1 + 0.2$ does not exactly equal 0.3 , but rather a number very close to it, hence the comparison returns false.

Question: What will the following code output?

```
var foo = {n: 1};
```

```
var bar = foo;
```

```
foo.x = foo = {n: 2};
```

```
console.log(foo.x);
```

a) undefined

b) {n: 2}

c) {n: 1}

d) TypeError

Answer: a) undefined

Explanation: The assignment `foo.x = foo = {n: 2}` is evaluated from right to left. First, `foo` is assigned `{n: 2}`. However, the previous `foo` (which is referenced by `bar`) has its `x` property set to the new `foo`, which is `{n: 2}`. Since the new `foo` does not have an `x` property, `foo.x` is undefined.

Question: What is the output of the following code?

```
console.log(typeof typeof 1);
```

Learn More from Laurence Svekis <https://basescripts.com/>

- a) "number"
- b) "string"
- c) "object"
- d) "undefined"

Answer: b) "string"

Explanation: The typeof operator always returns a string. So typeof 1 returns "number", and typeof "number" returns "string".

Question: What will the following code output?

```
function foo() {  
  
  return;  
  
  {  
  
    test: 1  
  
  };  
  
}  
  
console.log(typeof foo());
```

- a) "undefined"
- b) "object"
- c) "number"
- d) "function"

Answer: a) "undefined"

Explanation: Due to automatic semicolon insertion, return; is treated as a complete statement. Therefore, the function returns undefined.

Question: What will be the output of the following code?

```
var a = 1;
```

```
function b() {  
  a = 10;  
  return;  
  function a() {}  
}  
b();  
console.log(a);
```

a) 1
b) 10
c) undefined
d) ReferenceError

Answer: a) 1

Explanation: In the function b, the declaration function a() {} is hoisted to the top of the function scope, so a inside b is a local variable. The assignment a = 10 updates this local a, not the global a. Therefore, the global a remains 1.

Question: What will the following code output?

```
var a = 5;  
  
(function() {  
  var a = 6;  
  console.log(a);  
  a = 10;  
  console.log(a);  
})();  
console.log(a);
```

Learn More from Laurence Svekis <https://basescripts.com/>

- a) 5, 6, 10
- b) 6, 10, 5
- c) 6, 10, 10
- d) 5, 6, 10, 5

Answer: b) 6, 10, 5

Explanation: The first `console.log(a)` inside the function outputs 6 because `a` is locally declared. After assigning 10 to the local `a`, the second `console.log(a)` outputs 10. The `console.log(a)` outside the function outputs the global `a`, which remains 5.

Question: What will the following code output?

```
var a = 2;  
  
var b = (function() {  
  
  delete a;  
  
  return a;  
  
})();  
  
console.log(a);  
console.log(b);
```

- a) 2, undefined
- b) undefined, undefined
- c) 2, 2
- d) undefined, 2

Answer: c) 2, 2

Explanation: The `delete` operator is used to remove properties from objects. Here, `a` is a variable, not a property of an object, so `delete a` has no effect. Thus, `a` remains 2, and the returned value `b` is also 2.

Question: What will the following code output?

```
var x = [1, 2, 3];  
x[10] = 99;  
console.log(x.length);
```

- a) 3
- b) 10
- c) 11
- d) 4

Answer: c) 11

Explanation: Setting a value at index 10 increases the length of the array to 11, because JavaScript arrays are sparse, and unassigned indexes between the highest index and the array's length are treated as undefined.

Question: What will the following code output?

```
var arr = [10, 20, 30];  
arr['key'] = 'value';  
console.log(arr.length);
```

- a) 3
- b) 4
- c) 1
- d) undefined

Answer: a) 3

Explanation: Setting a property `arr['key']` does not affect the array's length property, which only counts numeric indices. Therefore, the length remains 3.

Question: What will the following code output?

```
function test() {  
  console.log(a);  
  console.log(foo());  
  var a = 1;  
  function foo() {  
    return 2;  
  }  
}  
test();
```

- a) 1, 2
- b) undefined, 2
- c) ReferenceError
- d) undefined, ReferenceError

Answer: b) undefined, 2

Explanation: Variables declared with var are hoisted to the top of the function scope but are not initialized. Thus, a is undefined when console.log(a) is called. Function declarations are also hoisted, so foo is available and returns 2.

Question: What will be the output of the following code?

```
function f() {}  
console.log(f.prototype);
```

- a) undefined
- b) {}
- c) null

Learn More from Laurence Svekis <https://basescripts.com/>

d) [object Object]

Answer: b) {}

Explanation: Every function in JavaScript has a prototype property, which is an empty object by default.

Question: What will the following code output?

```
var F = function() {};
```

```
var f = new F();
```

```
console.log(F.prototype === f.__proto__);
```

a) true

b) false

Answer: a) true

40

complete the request

ChatGPT

```
var a = {};
```

```
var b = {key: 'b'};
```

```
var c = {key: 'c'};
```

```
a[b] = 123;
```

```
a[c] = 456;
```

```
console.log(a[b]);
```

- a) 123

- b) 456

- c) undefined

- d) TypeError

Learn More from Laurence Svekis <https://basescripts.com/>

Answer: b) 456

Explanation: When objects are used as keys in JavaScript, they are converted to strings (``[object Object]``). Therefore, both ``a[b]`` and ``a[c]`` refer to the same key, and the last assignment overwrites the previous one. Thus, ``a[b]`` outputs ``456``.

Question: What will be the output of the following code?

```
var x = 10;

(function() {
  console.log(x);

  var x = 20;

})();
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: c) undefined

Explanation: The variable ``x`` is hoisted within the function scope, so the ``console.log(x)`` references the hoisted but uninitialized ``x``, which is ``undefined``.

Question: What will the following code output?

```
function foo() {
  return
  {
    message: "Hello"
  };
}
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
console.log(foo());
```

- a) { message: "Hello" }
- b) undefined
- c) null
- d) SyntaxError

Answer: b) undefined

Explanation: Due to automatic semicolon insertion, the `return` statement is terminated before the object is returned. Hence, `foo()` returns `undefined`.

Question: What will the following code output?

```
var f = function g() {  
  
return 23;  
  
};  
  
console.log(typeof g());
```

- a) "number"
- b) "undefined"
- c) "function"
- d) ReferenceError

Answer: d) ReferenceError

Explanation: The function `g` is only accessible within its own scope. Outside of this scope, it is not defined, resulting in a `ReferenceError`.

Question: What will the following code output?

```
var length = 10;  
  
function fn() {  
  
console.log(this.length);
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
}  
  
var obj = {  
  length: 5,  
  method: function() {  
    fn();  
    arguments[0]();  
  }  
};  
  
obj.method(fn);
```

- a) 10, 5

- b) 5, 10

- c) 10, 10

- d) 5, 5

Answer: c) 10, 10

Explanation: When `fn` is called in `obj.method(fn)`, the `this` keyword inside `fn` refers to the global object, where `length` is `10`. The `arguments[0]()` call also sets `this` to the global object, resulting in `10`.

Question: What will the following code output?

```
function test() {  
  var a = b = 3;  
}  
  
test();  
  
console.log(typeof a);  
  
console.log(typeof b);
```

Learn More from Laurence Svekis <https://basescripts.com/>

- a) "undefined", "number"
- b) "number", "number"
- c) "undefined", "undefined"
- d) "number", "undefined"

Answer: a) "undefined", "number"

Explanation: The statement `var a = b = 3`` declares `a`` locally within the function, but `b`` is implicitly declared as a global variable. Thus, `a`` is `undefined`` outside the function, while `b`` is `number``.

Question: What will the following code output?

```
var foo = function bar() { return 42; };  
  
console.log(typeof bar);
```

- a) "number"
- b) "function"
- c) "undefined"
- d) "object"

Answer: c) "undefined"

Explanation: The function `bar`` is only available within its own scope. Outside this scope, it is not defined, resulting in `typeof bar`` being `undefined``.

Question: What will be the output of the following code?

```
function foo() {  
  
var x = 10;  
  
(function() {  
console.log(x);  
})();
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
x = 20;
```

```
}
```

```
foo();
```

- a) 10

- b) 20

- c) undefined

- d) ReferenceError

Answer: a) 10

Explanation: The immediately invoked function expression (IIFE) captures the initial value of `x`, which is `10`, before it is reassigned to `20`.

Question: What will the following code output?

```
var a = 1;
```

```
var obj = {
```

```
  a: 2,
```

```
  fn: function() {
```

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

```
  }
```

```
};
```

```
var fn = obj.fn;
```

```
fn();
```

- a) 1

- b) 2

- c) undefined

- d) ReferenceError

Learn More from Laurence Svekis <https://basescripts.com/>

Answer: a) 1

Explanation: When `fn`` is called outside the context of `obj``, `this`` refers to the global object, where `a`` is `1``.

Question: What will the following code output?

```
var obj = {  
  
  a: 2,  
  
  foo: function() {  
  
    var self = this;  
  
    (function() {  
  
      console.log(self.a);  
  
    })();  
  
  }  
  
};  
  
obj.foo();
```

- a) undefined
- b) 2
- c) ReferenceError
- d) null

Answer: b) 2

Explanation: The variable `self`` captures the `this`` context of `obj``, allowing the inner function to correctly reference `obj.a``.

Question: What will the following code output?

```
var obj = {  
  
  a: 1,
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
foo: function() {  
  console.log(this.a);  
}  
};  
  
setTimeout(obj.foo, 1000);
```

- a) 1
- b) undefined
- c) null
- d) 0

Answer: b) undefined

Explanation: When `obj.foo` is passed to `setTimeout`, the context (`this`) is lost, so `this.a` refers to `undefined`.

Question: What will the following code output?

```
var obj = {  
  
  a: 1,  
  
  foo: function() {  
    console.log(this.a);  
  }  
};  
  
setTimeout(function() {  
  obj.foo();  
}, 1000);
```

- a) 1
- b) undefined

Learn More from Laurence Svekis <https://basescripts.com/>

- c) null
- d) 0

Answer: a) 1

Explanation: In this case, `obj.foo` is called within a function where `obj` is correctly referenced, so `this.a` is `1`.

Question: What will the following code output?

```
function foo() {  
  console.log(this);  
}  
foo.call(null);
```

- a) null
- b) undefined
- c) global object (window in browsers)
- d) TypeError

Answer: c) global object (window in browsers)

Explanation: When `null` is passed as the context to `call`, `this` defaults to the global object.

Question: What will the following code output?

```
function foo() {  
  "use strict";  
  console.log(this);  
}  
foo.call(null);
```

- a) null
- b) undefined

Learn More from Laurence Svekis <https://basescripts.com/>

- c) global object (window in browsers)
- d) TypeError

Answer: a) null

Explanation: In strict mode, passing `null` to `call` or `apply` results in `this` being `null`.

Question: What will the following code output?

```
var obj = {  
  
  a: 1,  
  
  foo: function() {  
    return function() {  
      console.log(this.a);  
    };  
  }  
};  
  
var fn = obj.foo();  
  
fn();
```

- a) 1
- b) undefined
- c) null
- d) 0

Answer: b) undefined

Explanation: The inner function loses the `this` context of `obj` and refers to the global object, where `a` is `undefined`.

Question: What will the following code output?

```
var obj = {
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
a: 1,  
foo: function() {  
  return () => {  
    console.log(this.a);  
  };  
}  
};  
  
var fn = obj.foo();  
  
fn();
```

- a) 1
- b) undefined
- c) null
- d) 0

Answer: a) 1

Explanation: Arrow functions capture the `this` context of their surrounding scope, which in this case is `obj`.

Question: What will the following code output?

```
var a = 10;  
  
var obj = {  
  a: 20,  
  foo: function() {  
    console.log(this.a);  
    setTimeout(function() {  
      console.log(this.a);
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
}, 1000);
```

```
}
```

```
};
```

```
obj.foo();
```

- a) 20, 10

- b) 20, 20

- c) undefined, 10

- d) undefined, undefined

Answer: a) 20, 10

Explanation: The first `console.log(this.a)` refers to `obj.a`, which is `20`. The `setTimeout` callback has `this` referring to the global object, where `a` is `10`.

Question: What will the following code output?

```
var a = 10;
```

```
var obj = {
```

```
  a: 20,
```

```
  foo: function() {
```

```
    var self = this;
```

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

```
    setTimeout(function() {
```

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

```
    }, 1000);
```

```
  }
```

```
};
```

```
obj.foo();
```

Learn More from Laurence Svekis <https://basescripts.com/>

- a) 20, 10
- b) 20, 20
- c) undefined, 10
- d) undefined, undefined

Answer: b) 20, 20

Explanation: The variable `self` captures the `this` context of `obj`, allowing the `setTimeout` callback to correctly reference `obj.a`.

Question: What will the following code output?

```
var a = 10;

var obj = {

  a: 20,

  foo: function() {

    console.log(this.a);

    setTimeout(() => {

      console.log(this.a);

    }, 1000);

  }

};

obj.foo();
```

- a) 20, 10
- b) 20, 20
- c) undefined, 10
- d) undefined, undefined

Answer: b) 20, 20

Learn More from Laurence Svekis <https://basescripts.com/>

Explanation: Arrow functions capture the `this` context of their surrounding scope, which in this case is `obj`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype.bar = 10;  
  
var foo = new Foo();  
  
Foo.prototype.bar = 20;  
  
console.log(foo.bar);
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: Changing the prototype property of `Foo` after an instance has been created affects the instance, so `foo.bar` is `20`.

Question: What will the following code output?

```
function Foo() {  
  
  this.bar = 10;  
  
}  
  
Foo.prototype.bar = 20;  
  
var foo = new Foo();  
  
console.log(foo.bar);
```

- a) 10
- b) 20

Learn More from Laurence Svekis <https://basescripts.com/>

- c) undefined
- d) ReferenceError

Answer: a) 10

Explanation: The instance property `bar` of `foo` shadows the prototype property, so `foo.bar` is `10`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype = {bar: 10};  
  
var foo = new Foo();  
  
Foo.prototype.bar = 20;  
  
console.log(foo.bar);
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: Changing the property of the prototype object affects the instance, so `foo.bar` is `20`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype = {bar: 10};  
  
var foo = new Foo();  
  
Foo.prototype = {bar: 20};  
  
console.log(foo.bar);
```

Learn More from Laurence Svekis <https://basescripts.com/>

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: a) 10

Explanation: Reassigning the prototype object does not affect existing instances, so `foo.bar` remains `10`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype = {bar: 10};  
  
var foo1 = new Foo();  
  
Foo.prototype = {bar: 20};  
  
var foo2 = new Foo();  
  
console.log(foo1.bar, foo2.bar);
```

- a) 10, 10
- b) 20, 20
- c) 10, 20
- d) undefined, undefined

Answer: c) 10, 20

Explanation: `foo1` is created with the initial prototype, so `foo1.bar` is `10`. `foo2` is created after the prototype reassignment, so `foo2.bar` is `20`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype.bar = 10;
```

Learn More from Laurence Svekis <https://basescripts.com/>


```
var foo1 = new Foo();  
Foo.prototype = {bar: 20};  
var foo2 = new Foo();  
console.log(foo1.bar, foo2.bar);
```

- a) 10, 10
- b) 20, 20
- c) 10, 20
- d) undefined, undefined

Answer: c) 10, 20

Explanation: `foo1` retains the initial prototype, so `foo1.bar` is `10`. `foo2` is created with the reassigned prototype, so `foo2.bar` is `20`.

Question: What will the following code output?

```
function Foo() {}  
Foo.prototype.bar = 10;  
var foo = new Foo();  
Foo.prototype.baz = 20;  
console.log(foo.baz);
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: Adding properties to the prototype after an instance has been created affects the instance, so `foo.baz` is `20`.

Learn More from Laurence Svekis <https://basescripts.com/>

Question: What will the following code output?

```
function Foo() {  
  this.bar = 10;  
}  
  
Foo.prototype.baz = 20;  
  
var foo = new Foo();  
  
delete foo.baz;  
  
console.log(foo.baz);
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: Deleting a non-existent property has no effect, so `foo.baz` still refers to the prototype property, which is `20`.

Question: What will the following code output?

```
function Foo() {  
  this.bar = 10;  
}  
  
Foo.prototype.baz = 20;  
  
var foo = new Foo();  
  
delete foo.bar;  
  
console.log(foo.bar);
```

- a) 10

Learn More from Laurence Svekis <https://basescripts.com/>

- b) 20
- c) undefined
- d) ReferenceError

Answer: c) undefined

Explanation: Deleting an instance property removes it from the instance, so `foo.bar` is `undefined`.

Question: What will the following code output?

```
function Foo() {  
  this.bar = 10;  
}  
  
Foo.prototype.baz = 20;  
  
var foo = new Foo();  
  
delete foo.bar;  
  
foo.bar = 30;  
  
console.log(foo.bar);
```

- a) 10
- b) 20
- c) 30
- d) undefined

Answer: c) 30

Explanation: Reassigning the `bar` property on the instance sets it to `30`.

Question: What will the following code output?

```
var a = 10;  
  
var obj = {
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
a: 20,  
foo: function() {  
  var a = 30;  
  return this.a;  
}  
};  
console.log(obj.foo());
```

- a) 10
- b) 20
- c) 30
- d) undefined

Answer: b) 20

Explanation: The `this` context within `foo` refers to `obj`, so `this.a` is `20`.

Question: What will the following code output?

```
var a = 10;  
var obj = {  
  a: 20,  
  foo: function() {  
    var a = 30;  
    return function() {  
      return this.a;  
    };  
  }  
}
```

```
};  
console.log(obj.foo());
```

- a) 10
- b) 20
- c) 30
- d) undefined

Answer: a) 10

Explanation: The returned function has its `this` context set to the global object, so `this.a` is `10`.

Question: What will the following code output?

```
var a = 10;  
var obj = {  
  a: 20,  
  foo: function() {  
    var a = 30;  
    return () => {  
      return this.a;  
    };  
  }  
};  
console.log(obj.foo());
```

- a) 10
- b) 20
- c) 30

Learn More from Laurence Svekis <https://basescripts.com/>

- d) undefined

Answer: b) 20

Explanation: Arrow functions capture the `this` context of their surrounding scope, which in this case is `obj`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return () => {  
      return this;  
    };  
  }  
};  
  
console.log(obj.foo()() === obj);
```

- a) true

- b) false

Answer: a) true

Explanation: The arrow function captures the `this` context of `obj`, so it returns `obj`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return function() {  
      return this;  
    };  
  }  
}
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
};  
console.log(obj.foo>()() === obj);
```

- a) true

- b) false

Answer: b) false

Explanation: The returned function has its `this` context set to the global object, so it does not return `obj`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return this;  
  }  
};
```

```
var bar = obj.foo;
```

```
console.log(bar() === obj);
```

- a) true

- b) false

Answer: b) false

Explanation: When `bar` is called, `this` refers to the global object, not `obj`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return this;  
  }  
}
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
};  
  
console.log(obj.foo() === obj);
```

- a) true

- b) false

Answer: a) true

Explanation: When `obj.foo` is called, this` refers to obj` , so it returns obj` .`

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return this;  
  }  
};  
  
var bar = obj.foo;  
  
console.log(bar.call(obj) === obj);
```

- a) true

- b) false

Answer: a) true

Explanation: When `bar` is called with call(obj)` , this` is explicitly set to obj` , so it returns obj` .`

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return this;  
  }  
}
```

Learn More from Laurence Svekis <https://basescripts.com/>


```
};  
  
var bar = obj.foo;  
  
console.log(bar.apply(obj) === obj);
```

- a) true
- b) false

Answer: a) true

Explanation: When `bar` is called with `apply(obj)`, `this` is explicitly set to `obj`, so it returns `obj`.

Question: What will the following code output?

```
var obj = {  
  
  foo: function() {  
  
    return this;  
  
  }  
  
};  
  
var bar = obj.foo;  
  
console.log(bar.bind(obj)() === obj);
```

- a) true
- b) false

Answer: a) true

Explanation: When `bar` is called with `bind(obj)`, `this` is permanently bound to `obj`, so it returns `obj`.

Question: What will the following code output?

```
var obj = {  
  
  foo: function() {
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
return () => {  
  return this;  
};  
  
}  
  
};  
  
var bar = obj.foo;  
  
console.log(bar())() === obj);
```

- a) true

- b) false

Answer: b) false

Explanation: The arrow function captures the `this` context of `bar`, which is the global object, not `obj`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return () => {  
      return this;  
    };  
  }  
};  
  
};  
  
console.log(obj.foo())() === obj);
```

- a) true

- b) false

Answer: a) true

Learn More from Laurence Svekis <https://basescripts.com/>

Explanation: The arrow function captures the `this` context of `obj`, so it returns `obj`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return () => {  
      return this.a;  
    };  
  },  
  a: 10  
};  
  
console.log(obj.foo());
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: a) 10

Explanation: The arrow function captures the `this` context of `obj`, so it correctly references `obj.a`, which is `10`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return function() {  
      return this.a;  
    };  
  };  
};
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
},
```

```
a: 10
```

```
};
```

```
console.log(obj.foo());
```

- a) 10

- b) undefined

- c) null

- d) ReferenceError

Answer: b) undefined

Explanation: The returned function has its `this` context set to the global object, where `a` is `undefined`.

Question: What will the following code output?

```
var obj = {
```

```
  foo: function() {
```

```
    return function() {
```

```
      return this.a;
```

```
    };
```

```
  },
```

```
  a: 10
```

```
};
```

```
console.log(obj.foo.call({a: 20})());
```

- a) 10

- b) 20

- c) undefined

Learn More from Laurence Svekis <https://basescripts.com/>

- d) ReferenceError

Answer: b) 20

Explanation: The `call` method sets the `this` context for `foo`, so `this.a` is `20`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return function() {  
      return this.a;  
    };  
  },  
  a: 10  
};  
  
console.log(obj.foo.apply({a: 30})());
```

- a) 10

- b) 30

- c) undefined

- d) ReferenceError

Answer: b) 30

Explanation: The `apply` method sets the `this` context for `foo`, so `this.a` is `30`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return function() {
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
return this.a;
};
},
a: 10
};
console.log(obj.foo.bind({a: 40})());
```

- a) 10
- b) 40
- c) undefined
- d) ReferenceError

Answer: b) 40

Explanation: The `bind` method sets the `this` context for `foo` permanently, so `this.a` is `40`.

Question: What will the following code output?

```
var obj = {
  foo: function() {
    return () => {
      return this.a;
    };
  },
  a: 10
};
console.log(obj.foo.call({a: 50})());
```

- a) 10

- b) 50
- c) undefined
- d) ReferenceError

Answer: a) 10

Explanation: The arrow function captures the `this` context of `obj`, so `this.a` is `10`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return () => {  
      return this.a;  
    };  
  },  
  a: 10  
};  
  
console.log(obj.foo.apply({a: 60})());
```

- a) 10
- b) 60
- c) undefined
- d) ReferenceError

Answer: a) 10

Explanation: The arrow function captures the `this` context of `obj`, so `this.a` is `10`.

Question: What will the following code output?

```
var obj = {
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
foo: function() {  
  return () => {  
    return this.a;  
  };  
},  
a: 10  
};  
  
console.log(obj.foo.bind({a: 70})());
```

- a) 10
- b) 70
- c) undefined
- d) ReferenceError

Answer: a) 10

Explanation: The arrow function captures the `this` context of `obj`, so `this.a` is `10`.

Question: What will the following code output?

```
var a = 10;  
  
var obj = {  
  a: 20,  
  foo: function() {  
    console.log(this.a);  
  }  
};  
  
var bar = obj.foo;
```

Learn More from Laurence Svekis <https://basescripts.com/>


```
bar.call(obj);
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: The `call` method sets the `this` context to `obj`, so `this.a` is `20`.

Question: What will the following code output?

```
var a = 10;
```

```
var obj = {
```

```
  a: 20,
```

```
  foo: function() {
```

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

```
  }
```

```
};
```

```
var bar = obj.foo;
```

```
bar.apply(obj);
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: The `apply` method sets the `this` context to `obj`, so `this.a` is `20`.

Learn More from Laurence Svekis <https://basescripts.com/>

Question: What will the following code output?

```
var a = 10;

var obj = {

  a: 20,

  foo: function() {

    console.log(this.a);

  }

};

var bar = obj.foo;

bar.bind(obj)();
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: The `bind` method sets the `this` context to `obj`, so `this.a` is `20`.

Question: What will the following code output?

```
var obj = {

  foo: function() {

    return () => {

      return this.a;

    };

  },

};
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
a: 10
};
var bar = obj.foo;
console.log(bar());
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: b) undefined

Explanation: The arrow function captures the `this` context of the global object, where `a` is `undefined`.

Question: What will the following code output?

```
var obj = {
  foo: function() {
    return () => {
      return this.a;
    };
  },
  a: 10
};
console.log(obj.foo());
```

- a) 10
- b) undefined
- c) null

Learn More from Laurence Svekis <https://basescripts.com/>

- d) ReferenceError

Answer: a) 10

Explanation: The arrow function captures the `this` context of `obj`, so `this.a` is `10`.

Question: What will the following code output?

```
function Foo() {  
  this.bar = 10;  
}  
  
Foo.prototype.baz = function() {  
  return this.bar;  
};  
  
var foo = new Foo();  
console.log(foo.baz());
```

- a) 10

- b) undefined

- c) null

- d) ReferenceError

Answer: a) 10

Explanation: The method `baz` on the prototype correctly references the instance property `bar`, so `foo.baz()` returns `10`.

Question: What will the following code output?

```
function Foo() {  
  this.bar = 10;  
}  
  
Foo.prototype.baz = function() {
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
return this.bar;
};
var foo = new Foo();
var baz = foo.baz;
console.log(baz());
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: b) undefined

Explanation: The `this` context of `baz` is the global object, where `bar` is `undefined`.

Question: What will the following code output?

```
function Foo() {
  this.bar = 10;
}
Foo.prototype.baz = function() {
  return this.bar;
};
var foo = new Foo();
var baz = foo.baz.bind(foo);
console.log(baz());
```

- a) 10
- b) undefined

Learn More from Laurence Svekis <https://basescripts.com/>

- c) null
- d) ReferenceError

Answer: a) 10

Explanation: The `bind` method sets the `this` context of `baz` to `foo`, so it correctly references `foo.bar`.

Question: What will the following code output?

```
function Foo() {  
  
  this.bar = 10;  
  
}  
  
Foo.prototype.baz = function() {  
  
  return this.bar;  
  
};  
  
var foo = new Foo();  
  
setTimeout(foo.baz, 1000);
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: b) undefined

Explanation: The `this` context of `baz` is the global object when called inside `setTimeout`, where `bar` is `undefined`.

Question: What will the following code output?

```
function Foo() {  
  
  this.bar = 10;
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
}  
Foo.prototype.baz = function() {  
  return this.bar;  
};  
var foo = new Foo();  
setTimeout(foo.baz.bind(foo), 1000);
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: a) 10

Explanation: The `bind` method sets the `this` context of `baz` to `foo`, so it correctly references `foo.bar`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype.bar = 10;  
  
var foo1 = new Foo();  
  
Foo.prototype = {baz: 20};  
  
var foo2 = new Foo();  
  
console.log(foo1.bar, foo2.bar);
```

- a) 10, undefined
- b) undefined, undefined
- c) 10, 10
- d) 10, 20

Learn More from Laurence Svekis <https://basescripts.com/>

Answer: a) 10, undefined

Explanation: `foo1` retains the initial prototype, so `foo1.bar` is `10`. `foo2` is created with the reassigned prototype, where `bar` is `undefined`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype.bar = 10;  
  
var foo1 = new Foo();  
  
Foo.prototype.bar = 20;  
  
var foo2 = new Foo();  
  
console.log(foo1.bar, foo2.bar);
```

- a) 10, 10
- b) 20, 20
- c) 10, 20
- d) undefined, undefined

Answer: b) 20, 20

Explanation: Changing the prototype property affects all instances, so both `foo1.bar` and `foo2.bar` are `20`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype.bar = 10;  
  
var foo1 = new Foo();  
  
Foo.prototype = {bar: 20};  
  
var foo2 = new Foo();  
  
console.log(foo1.bar, foo2.bar);
```

Learn More from Laurence Svekis <https://basescripts.com/>

- a) 10, 10
- b) 20, 20
- c) 10, 20
- d) undefined, undefined

Answer: c) 10, 20

Explanation: `foo1` retains the initial prototype, so `foo1.bar` is `10`. `foo2` is created with the reassigned prototype, so `foo2.bar` is `20`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype.bar = 10;  
  
var foo1 = new Foo();  
  
Foo.prototype = {baz: 20};  
  
var foo2 = new Foo();  
  
console.log(foo1.baz, foo2.baz);
```

- a) 20, 20
- b) undefined, 20
- c) undefined, undefined
- d) ReferenceError

Answer: b) undefined, 20

Explanation: `foo1` retains the initial prototype, so `foo1.baz` is `undefined`. `foo2` is created with the reassigned prototype, so `foo2.baz` is `20`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype.bar = 10;
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
var foo1 = new Foo();  
Foo.prototype.bar = 20;  
var foo2 = new Foo();  
console.log(foo1.bar, foo2.bar);
```

- a) 10, 10
- b) 20, 20
- c) 10, 20
- d) undefined, undefined

Answer: b) 20, 20

Explanation: Changing the prototype property affects all instances, so both `foo1.bar` and `foo2.bar` are `20`.

Question: What will the following code output?

```
function Foo() {}  
Foo.prototype.bar = 10;  
var foo1 = new Foo();  
foo1.bar = 30;  
Foo.prototype.bar = 20;  
var foo2 = new Foo();  
console.log(foo1.bar, foo2.bar);
```

- a) 30, 10
- b) 30, 20
- c) 10, 20
- d) undefined, undefined

Answer: b) 30, 20

Learn More from Laurence Svekis <https://basescripts.com/>

Explanation: The instance property `foo1.bar` shadows the prototype property, so `foo1.bar` is `30`. The prototype property `foo2.bar` is `20`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype.bar = 10;  
  
var foo1 = new Foo();  
  
Foo.prototype = {baz: 20};  
  
var foo2 = new Foo();  
  
console.log(foo1.bar, foo2.bar);
```

- a) 10, undefined
- b) undefined, undefined
- c) 10, 10
- d) 10, 20

Answer: a) 10, undefined

Explanation: `foo1` retains the initial prototype, so `foo1.bar` is `10`. `foo2` is created with the reassigned prototype, where `bar` is `undefined`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype.bar = 10;  
  
var foo1 = new Foo();  
  
Foo.prototype.bar = 20;  
  
var foo2 = new Foo();  
  
console.log(foo1.bar, foo2.bar);
```

- a) 10, 10

Learn More from Laurence Svekis <https://basescripts.com/>

- b) 20, 20
- c) 10, 20
- d) undefined, undefined

Answer: b) 20, 20

Explanation: Changing the prototype property affects all instances, so both `foo1.bar` and `foo2.bar` are `20`.

Question: What will the following code output?

```
function Foo() {}  
  
Foo.prototype.bar = 10;  
  
var foo1 = new Foo();  
  
Foo.prototype = {baz: 20};  
  
var foo2 = new Foo();  
  
console.log(foo1.baz, foo2.baz);
```

- a) 20, 20
- b) undefined, 20
- c) undefined, undefined
- d) ReferenceError

Answer: b) undefined, 20

Explanation: `foo1` retains the initial prototype, so `foo1.baz` is `undefined`. `foo2` is created with the reassigned prototype, so `foo2.baz` is `20`.

Question: What will the following code output?

```
var a = 10;  
  
var obj = {  
  
a: 20,
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
foo: function() {  
  console.log(this.a);  
}  
};  
  
var bar = obj.foo;  
  
bar();
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: a) 10

Explanation: The `this` context of `bar` is the global object, so `this.a` is `10`.

Question: What will the following code output?

```
var a = 10;  
  
var obj = {  
  a: 20,  
  foo: function() {  
    console.log(this.a);  
  }  
};  
  
var bar = obj.foo;  
  
bar.call(obj);
```

- a) 10

- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: The `call` method sets the `this` context to `obj`, so `this.a` is `20`.

Question: What will the following code output?

```
var a = 10;

var obj = {
  a: 20,
  foo: function() {
    console.log(this.a);
  }
};

var bar = obj.foo;
bar.apply(obj);
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: The `apply` method sets the `this` context to `obj`, so `this.a` is `20`.

Question: What will the following code output?

```
var a = 10;
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
var obj = {  
  a: 20,  
  foo: function() {  
    console.log(this.a);  
  }  
};  
  
var bar = obj.foo;  
bar.bind(obj)();
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: The `bind` method sets the `this` context to `obj`, so `this.a` is `20`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return () => {  
      return this.a;  
    }  
  }  
};  
  
a: 10  
};
```

```
var bar = obj.foo;  
console.log(bar());
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: b) undefined

Explanation: The arrow function captures the `this` context of the global object, where `a` is `undefined`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return () => {  
      return this.a;  
    };  
  },  
  a: 10  
};  
console.log(obj.foo());
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: a) 10

Learn More from Laurence Svekis <https://basescripts.com/>

Explanation: The arrow function captures the `this` context of `obj`, so `this.a` is `10`.

Question: What will the following code output?

```
function Foo() {  
  
  this.bar = 10;  
  
}  
  
Foo.prototype.baz = function() {  
  
  return this.bar;  
  
};  
  
var foo = new Foo();  
  
console.log(foo.baz());
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: a) 10

Explanation: The method `baz` on the prototype correctly references the instance property `bar`, so `foo.baz()` returns `10`.

Question: What will the following code output?

```
function Foo() {  
  
  this.bar = 10;  
  
}  
  
Foo.prototype.baz = function() {  
  
  return this.bar;  
  
};
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
var foo = new Foo();  
var baz = foo.baz;  
console.log(baz());
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: b) undefined

Explanation: The `this` context of `baz` is the global object, where `bar` is `undefined`.

Question: What will the following code output?

```
function Foo() {  
  this.bar = 10;  
}  
Foo.prototype.baz = function() {  
  return this.bar;  
};  
var foo = new Foo();  
var baz = foo.baz.bind(foo);  
console.log(baz());
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: a) 10

Explanation: The `bind` method sets the `this` context of `baz` to `foo`, so it correctly references `foo.bar`.

Question: What will the following code output?

```
function Foo() {  
  
  this.bar = 10;  
  
}  
  
Foo.prototype.baz = function() {  
  
  return this.bar;  
  
};  
  
var foo = new Foo();  
  
setTimeout(foo.baz, 1000);
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: b) undefined

Explanation: The `this` context of `baz` is the global object when called inside `setTimeout`, where `bar` is `undefined`.

Question: What will the following code output?

```
function Foo() {  
  
  this.bar = 10;  
  
}  
  
Foo.prototype.baz = function() {
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
return this.bar;
};
var foo = new Foo();
setTimeout(foo.baz.bind(foo), 1000);
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: a) 10

Explanation: The `bind` method sets the `this` context of `baz` to `foo`, so it correctly references `foo.bar`.

Question: What will the following code output?

```
var a = 10;
var obj = {
  a: 20,
  foo: function() {
    console.log(this.a);
  }
};
var bar = obj.foo;
bar();
```

- a) 10
- b) 20
- c) undefined

Learn More from Laurence Svekis <https://basescripts.com/>

- d) ReferenceError

Answer: a) 10

Explanation: The `this` context of `bar` is the global object, so `this.a` is `10`.

Question: What will the following code output?

```
var a = 10;

var obj = {

  a: 20,

  foo: function() {

    console.log(this.a);

  }

};

var bar = obj.foo;
```

```
bar.call(obj);
```

- a) 10

- b) 20

- c) undefined

- d) ReferenceError

Answer: b) 20

Explanation: The `call` method sets the `this` context to `obj`, so `this.a` is `20`.

Question: What will the following code output?

```
var a = 10;

var obj = {

  a: 20,
```

Learn More from Laurence Svekis <https://basescripts.com/>

```
foo: function() {  
  console.log(this.a);  
}  
};  
  
var bar = obj.foo;  
bar.apply(obj);
```

- a) 10
- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: The `apply` method sets the `this` context to `obj`, so `this.a` is `20`.

Question: What will the following code output?

```
var a = 10;  
  
var obj = {  
  a: 20,  
  foo: function() {  
    console.log(this.a);  
  }  
};  
  
var bar = obj.foo;  
bar.bind(obj)();
```

- a) 10

- b) 20
- c) undefined
- d) ReferenceError

Answer: b) 20

Explanation: The `bind` method sets the `this` context to `obj`, so `this.a` is `20`.

Question: What will the following code output?

```
var obj = {  
  foo: function() {  
    return () => {  
      return this.a;  
    };  
  },  
  a: 10  
};  
  
var bar = obj.foo;  
console.log(bar());
```

- a) 10
- b) undefined
- c) null
- d) ReferenceError

Answer: b) undefined

Explanation: The arrow function captures the `this` context of the global object, where `a` is `undefined`.