USING AN IMMEDIATELY-INVOKED FUNCTION

Calling returned functions instantly instead of variable storage

```
var parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55]
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
var fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
var wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide )();
```

```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )();
```

Yep, a semicolon gives the instruction to execute the function!

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

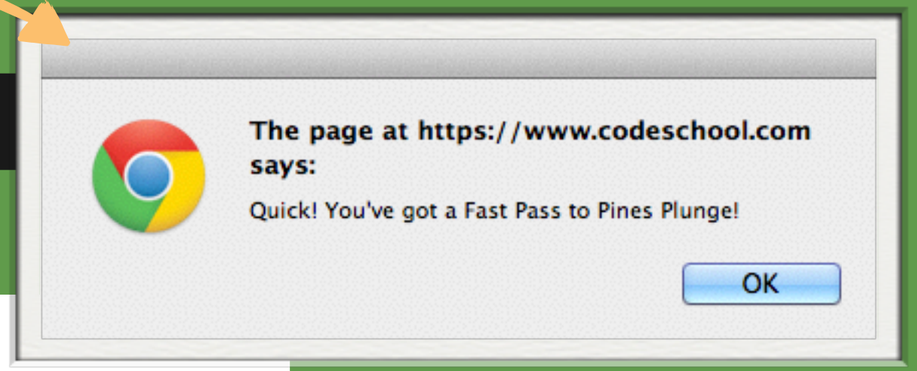
Calling returned functions instantly instead of variable storage

```
var parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55]
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
var fastPassQueue = [ "Pines Plunge", "Birch Bumpers", "Pines Plunge" ];
```

```
var wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide )();
```




```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )();
```

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

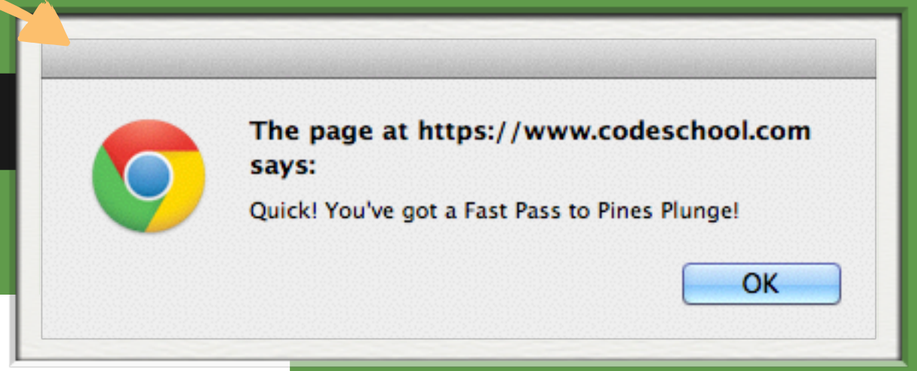
Calling returned functions instantly instead of variable storage

```
var parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55]
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
var fastPassQueue = [                    "Birch Bumpers", "Pines Plunge" ];
```

```
var wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide )();
```




```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )();
```

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

USING AN IMMEDIATELY-INVOKED FUNCTION

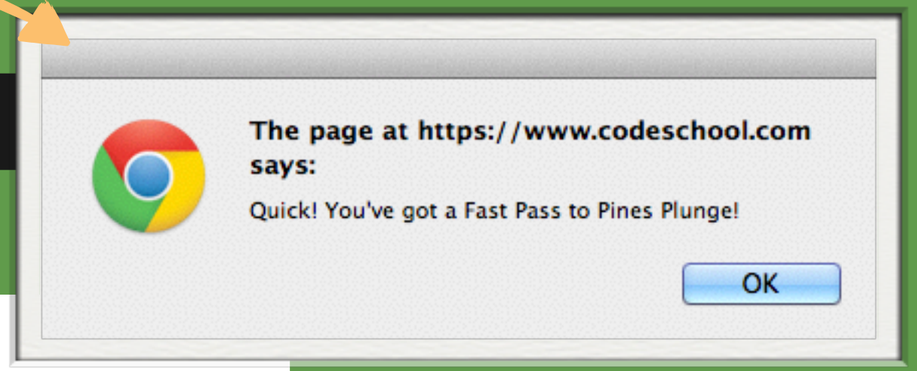
Calling returned functions instantly instead of variable storage

```
var parkRides = [ ["Birch Bumpers", 40] , ["Pines Plunge", 55]
                  ["Cedar Coaster", 20] , ["Ferris Wheel of Firs", 90]  ];
```

```
var fastPassQueue = [ "Birch Bumpers", "Pines Plunge" ];
```

```
var wantsRide = "Pines Plunge";
```

```
buildTicket( parkRides, fastPassQueue, wantsRide )();
```

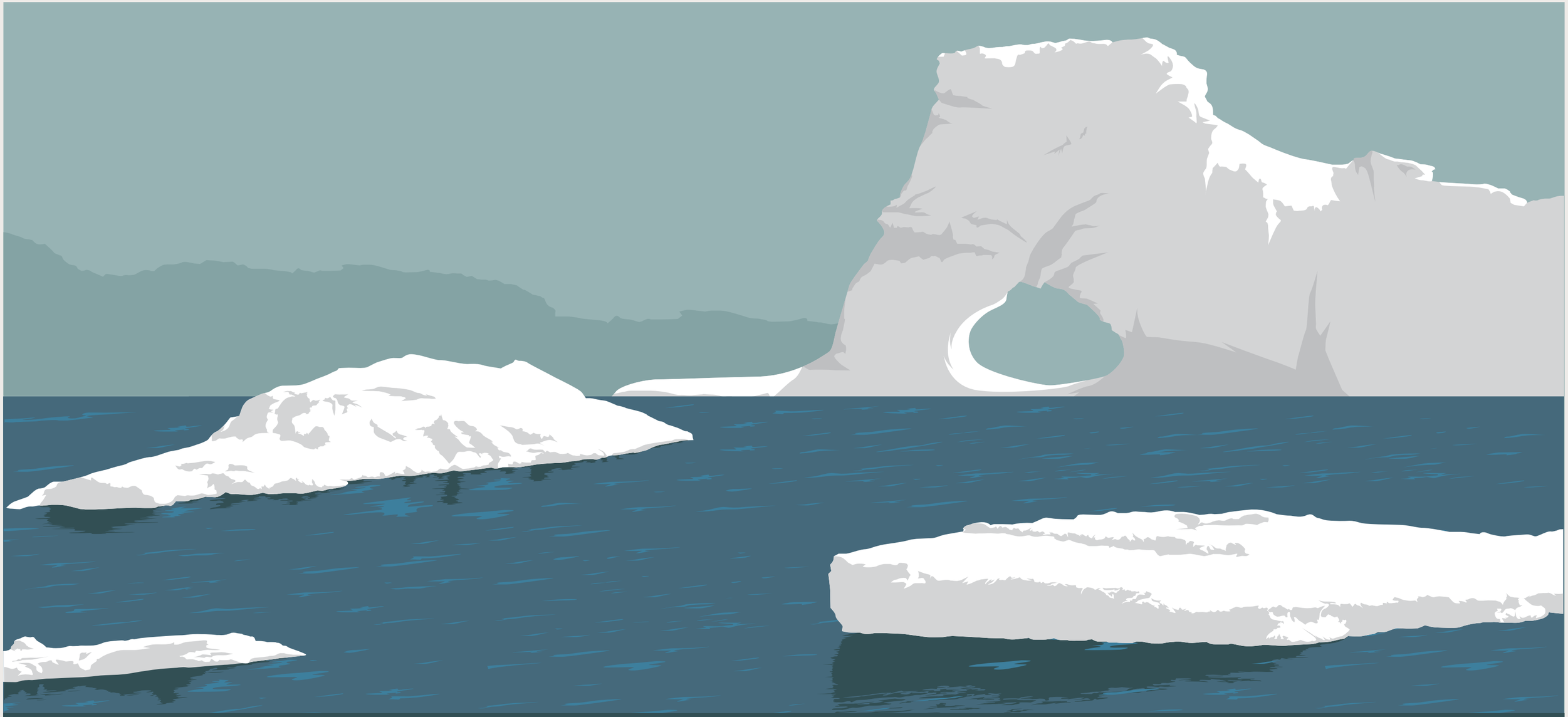



```
( function ( ) {
    alert("Quick! You've got a Fast Pass to " + pass + "!");
} )();
```

```
function buildTicket ( allRides, passRides, pick ) {
   ...
}
```

Visit the Wondrous

FOREST OF FUNCTION EXPRESSIONS



Explore

COLD CLOSURES COVE

LEVEL 2

COLD CLOSURES COVE

A PACKAGE DEAL

Guess what? Congratulations! You've already made a basic closure!

```
function buildTicket ( allRides, passRides, pick ) {
  if(passRides[0] == pick){
     var pass = fastAvail.shift();
     return function ( ) { alert("Quick! You've got a Fast Pass to " + pass + "!");
            };
  } else {
     for(var i = 0; i<allRides.length; i++){
        if(allRides[i][0] == pick){
           return function () { alert("A ticket is printing for " + pick + "!\n" +
                                "Your wait time is about " + allRides[i][1] + " minutes.");
           };
        }
     }
  }
}
```

The entire contents of one of these inner functions will still be available OUTSIDE the outermost function.

Returning a function from a function, complete with variables from an external scope, is called a closure.

A PACKAGE DEAL

A closure wraps up an entire environment, binding necessary variables from other scopes.

```
function  testClosure ( ) {

        var x = 4;
        return x;

}
```

Local Variable only!

```
testClosure();
```

→ 4

```
x;
```

→ undefined

A function's local variables aren't available once the function's scope is closed!

A PACKAGE DEAL

A closure wraps up an entire environment, binding necessary variables from other scopes.

The inner function can access the outer function's variables, because they "feel" like global variables.

```
function  testClosure ( ) {
        var x = 4;
        function closeX ( ) {
            return x;
        }
        return closeX;
}
```

Notice x does not need to be "stored" anywhere in closeX, not even as a parameter!

A PACKAGE DEAL

A closure wraps up an entire environment, binding necessary variables from other scopes.

```
function  testClosure ( ) {
        var x = 4;
        function closeX ( ) {
                return x;
        }
        return closeX;
}
```

```
var checkLocalX = testClosure();
```

```
checkLocalX();
```

→ 4

Even though `testClosure` has finished operating, its local variable is now bound within `checkLocalX`.

CLOSURES HELP IN FUNCTION "CONSTRUCTION ZONES"

A closure can make the creation of very similar functions ultra-efficient.

```
function buildCoveTicketMaker( transport ) {
        return function ( name ) {
           alert("Here is your transportation ticket via the " + transport + ".\n" +
                 "Welcome to the Cold Closures Cove, " + name + "!");
        }
}
```

CLOSURES HELP IN FUNCTION "CONSTRUCTION ZONES"

A closure can make the creation of very similar functions ultra-efficient.

```
function buildCoveTicketMaker( transport ) {
        return function ( name ) {
           alert("Here is your transportation ticket via the " + transport + ".\n" +
                 "Welcome to the Cold Closures Cove, " + name + "!");
        }
}
```

```
var getSubmarineTicket = buildCoveTicketMaker("Submarine");
```

```
var getBattleshipTicket = buildCoveTicketMaker("Battleship");
```

```
var getGiantSeagullTicket = buildCoveTicketMaker("Giant Seagull");
```

We give `buildCoveTicketMaker` the mode of transportation, which is closed into the returned anonymous function.

CLOSURES HELP IN FUNCTION "CONSTRUCTION ZONES"

A closure can make the creation of very similar functions ultra-efficient.

```
var getSubmarineTicket = buildCoveTicketMaker("Submarine");
```

```
var getBattleshipTicket = buildCoveTicketMaker("Battleship");
```

```
var getGiantSeagullTicket = buildCoveTicketMaker("Giant Seagull");
```

CLOSURES HELP IN FUNCTION “CONSTRUCTION ZONES”

A closure can make the creation of very similar functions ultra-efficient.

```
getSubmarineTicket
```

```
getBattleshipTicket
```

```
getGiantSeagullTicket
```

BEWARE: BOUND VARIABLES WON'T BE EVIDENT IN THE STORED FUNCTION

Examining the contents of our new variables doesn't reveal closures.

```
getSubmarineTicket;
```

```
function ( name ) {
   alert("Here is your transportation ticket via the " + transport + ".\n" +
         "Welcome to the Cold Closures Cove, " + name + "!");
}
```

Holds "Submarine"

```
getBattleshipTicket;
```

```
function ( name ) {
   alert("Here is your transportation ticket via the " + transport + ".\n" +
         "Welcome to the Cold Closures Cove, " + name + "!");
}
```

Holds "Battleship"

```
getGiantSeagullTicket;
```

```
function ( name ) {
   alert("Here is your transportation ticket via the " + transport + ".\n" +
         "Welcome to the Cold Closures Cove, " + name + "!");
}
```

Holds "Giant Seagull"

BEWARE: BOUND VARIABLES WON'T BE EVIDENT IN THE STORED FUNCTION

Examining the contents of our new variables doesn't reveal closures.

```
getSubmarineTicket;
```

```
function ( name ) {
   alert("Here is your transportation ticket via the " + transport + ".\n" +
         "Welcome to the Cold Closures Cove, " + name + "!");
}
```

```
getBattleshipTicket;
```

```
function ( name ) {
   alert("Here is your transportation ticket via the " + transport + ".\n" +
         "Welcome to the Cold Closures Cove, " + name + "!");
}
```

```
getGiantSeagullTicket;
```

```
function ( name ) {
   alert("Here is your transportation ticket via the " + transport + ".\n" +
         "Welcome to the Cold Closures Cove, " + name + "!");
}
```

Until we call any of these functions with a parameter, the name variable is still undefined.

LET'S MAKE SOME TICKETS!

Passing a name to any of our ticket makers will complete our ticket-making process.

```
getSubmarineTicket;
```

```
function ( name ) {
   alert("Here is your transportation ticket via the " + transport + ".\n" +
         "Welcome to the Cold Closures Cove, " + name + "!");
}
```

```
getBattleshipTicket;
```

```
function ( name ) {
   alert("Here is your transportation ticket via the " + transport + ".\n" +
         "Welcome to the Cold Closures Cove, " + name + "!");
}
```

```
getGiantSeagullTicket;
```

```
function ( name ) {
   alert("Here is your transportation ticket via the " + transport + ".\n" +
         "Welcome to the Cold Closures Cove, " + name + "!");
}
```

LET'S MAKE SOME TICKETS!

Passing a name to any of our ticket makers will complete our ticket-making process.

```
getSubmarineTicket;
```

```
getBattleshipTicket;
```

```
getGiantSeagullTicket;
```

LET'S MAKE SOME TICKETS!

Passing a name to any of our ticket makers will complete our ticket-making process.

```
getSubmarineTicket("Mario");
```

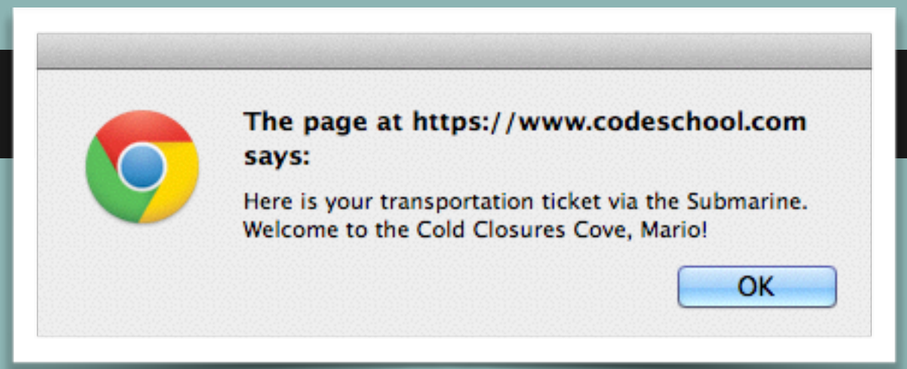
```
getBattleshipTicket("Luigi");
```

```
getGiantSeagullTicket("Bowser");
```

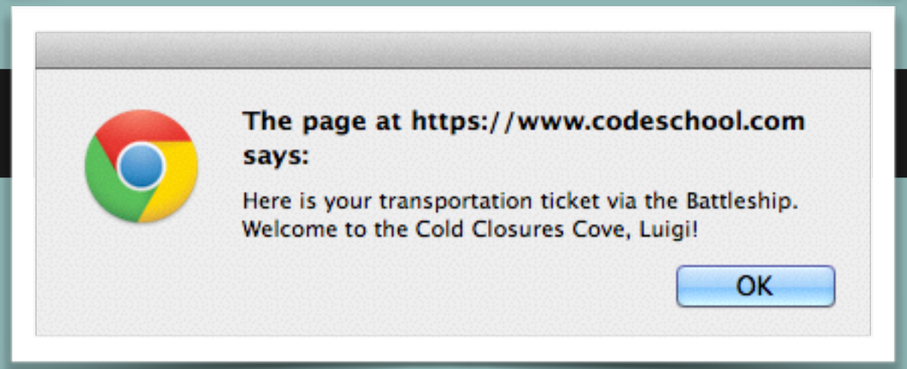
LET'S MAKE SOME TICKETS!

Passing a name to any of our ticket makers will complete our ticket-making process.

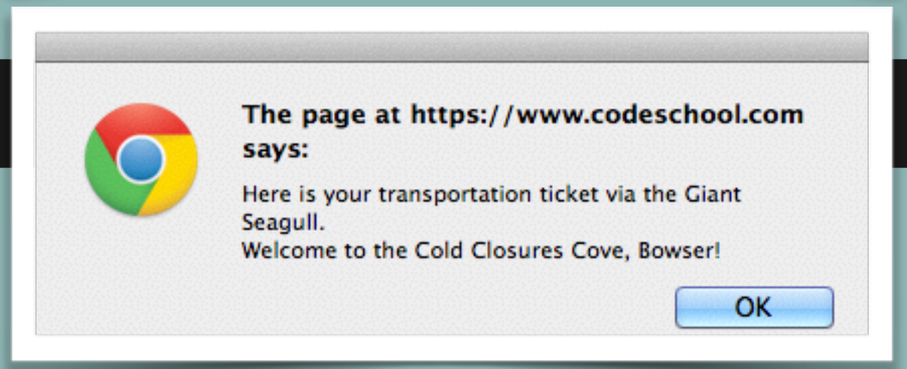
```
getSubmarineTicket("Mario");
```

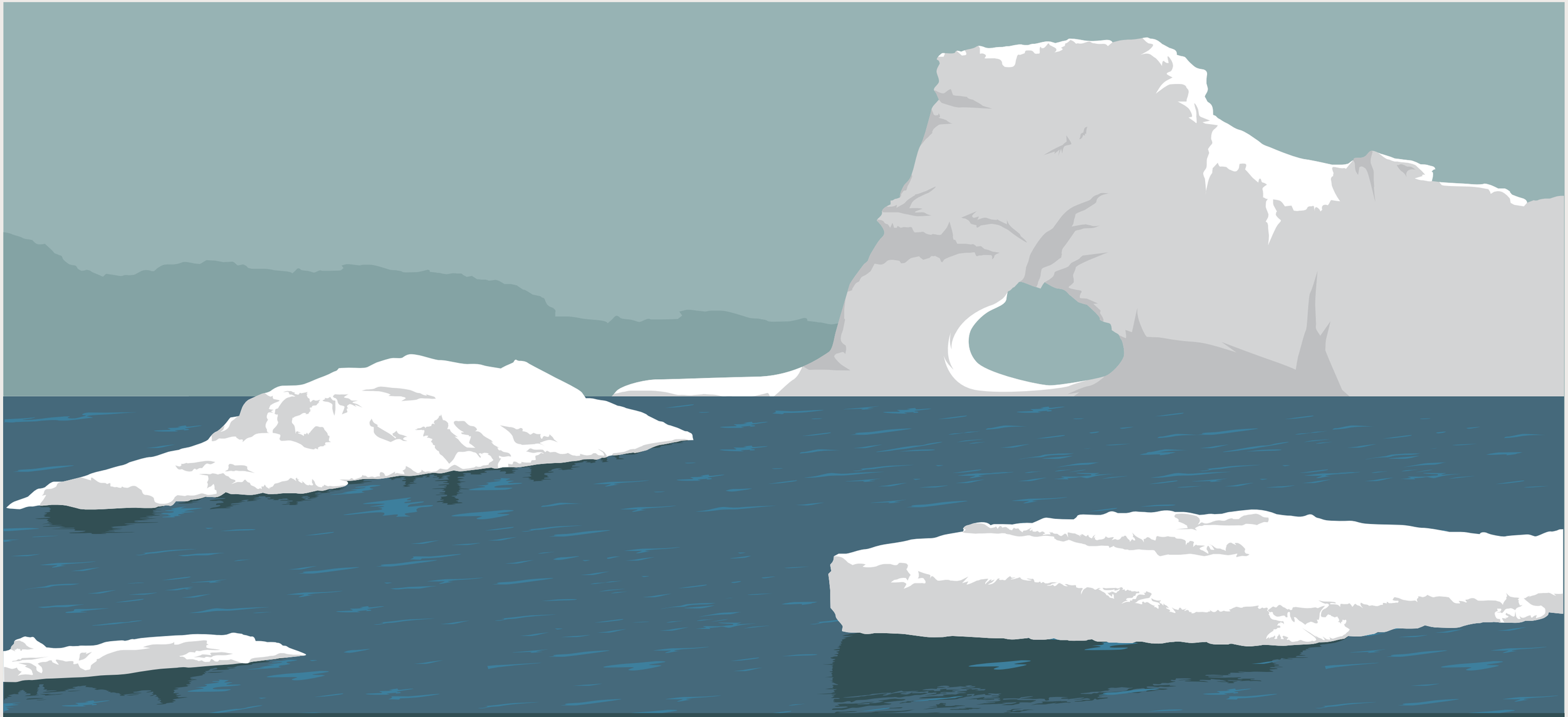


```
getBattleshipTicket("Luigi");
```



```
getGiantSeagullTicket("Bowser");
```





Explore

COLD CLOSURES COVE

ADDING A PASSENGER TRACKER

Closure functions can even modify bound variables in the background

```
function buildCoveTicketMaker( transport ) {
        return function ( name ) {
           alert("Here is your transportation ticket via the " + transport + ".\n" +
                 "Welcome to the Cold Closures Cove, " + name + "!");
        }
}
```

ADDING A PASSENGER TRACKER

Closure functions can even modify bound variables in the background

```
function buildCoveTicketMaker( transport ) {
        var passengerNumber = 0;
        return function ( name ) {

           alert("Here is your transportation ticket via the " + transport + ".\n" +
                 "Welcome to the Cold Closures Cove, " + name + "!" +
                                                                  );
        }
}
```

We'll start every ticket maker's tracker at 0 passengers.

ADDING A PASSENGER TRACKER

Closure functions can even modify bound variables in the background

```
function buildCoveTicketMaker( transport ) {
        var passengerNumber = 0;
        return function ( name ) {
            passengerNumber++;
            alert("Here is your transportation ticket via the " + transport + ".\n" +
                  "Welcome to the Cold Closures Cove, " + name + "!" +
                                                                  );
        }
}
```

When a particular ticket maker is called, we know a new passenger should be added, so we'll increase the tracker.

ADDING A PASSENGER TRACKER

Closure functions can even modify bound variables in the background

```
function buildCoveTicketMaker( transport ) {
        var passengerNumber = 0;
        return function ( name ) {
            passengerNumber++;
            alert("Here is your transportation ticket via the " + transport + ".\n" +
                  "Welcome to the Cold Closures Cove, " + name + "!" +
                  "You are passenger #" + passengerNumber + "." );
        }
}
```

Each time a ticket is "printed," this passengerNumber will contain the precise amount of times this kind of ticket has been given.

ADDING A PASSENGER TRACKER

Closure functions can even modify bound variables in the background

```
function buildCoveTicketMaker( transport ) {
        var passengerNumber = 0;
        return function ( name ) {
            passengerNumber++;
            alert("Here is your transportation ticket via the " + transport + ".\n" +
                  "Welcome to the Cold Closures Cove, " + name + "!" +
                  "You are passenger #" + passengerNumber + "." );
        }
}
```

```
var getSubmarineTicket = buildCoveTicketMaker("Submarine");
getSubmarineTicket;
```

```
function (name) {
    passengerNumber++;
    alert("Here is your transportation ticket via the " + transport + ".\n" +
          "Welcome to the Cold Closures Cove, " + name + "!\n" +
          "You are passenger #" + passengerNumber + ".");
}
```

Notice that no initial value for passengerNumber is evident in our new function. It's value starts at 0 and is adjusted with each call to getSubmarineTicket.

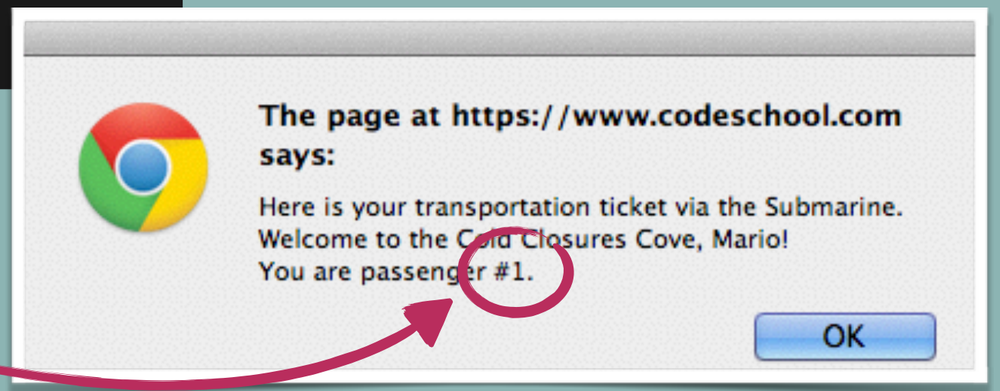
ADDING A PASSENGER TRACKER

Closure functions can even modify bound variables in the background

```
function buildCoveTicketMaker( transport ) {
        var passengerNumber = 0;
        return function ( name ) {
            passengerNumber++;
            alert("Here is your transportation ticket via the " + transport + ".\n" +
                  "Welcome to the Cold Closures Cove, " + name + "!" +
                  "You are passenger #" + passengerNumber + "." );
        }
}
```

```
var getSubmarineTicket = buildCoveTicketMaker("Submarine");
getSubmarineTicket("Mario");
```

On our first call to the new getSubmarineTicket, passengerNumber is incremented to 1.

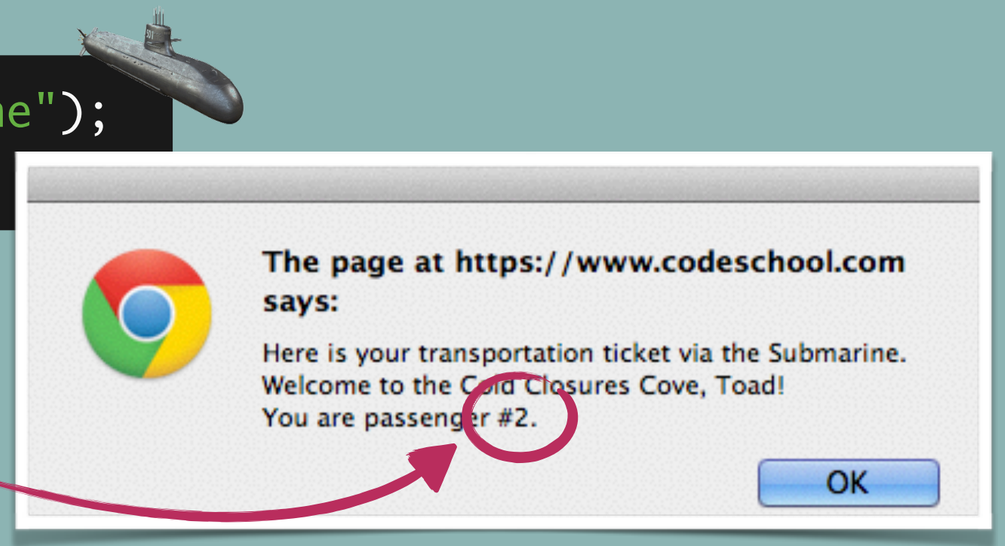


ADDING A PASSENGER TRACKER

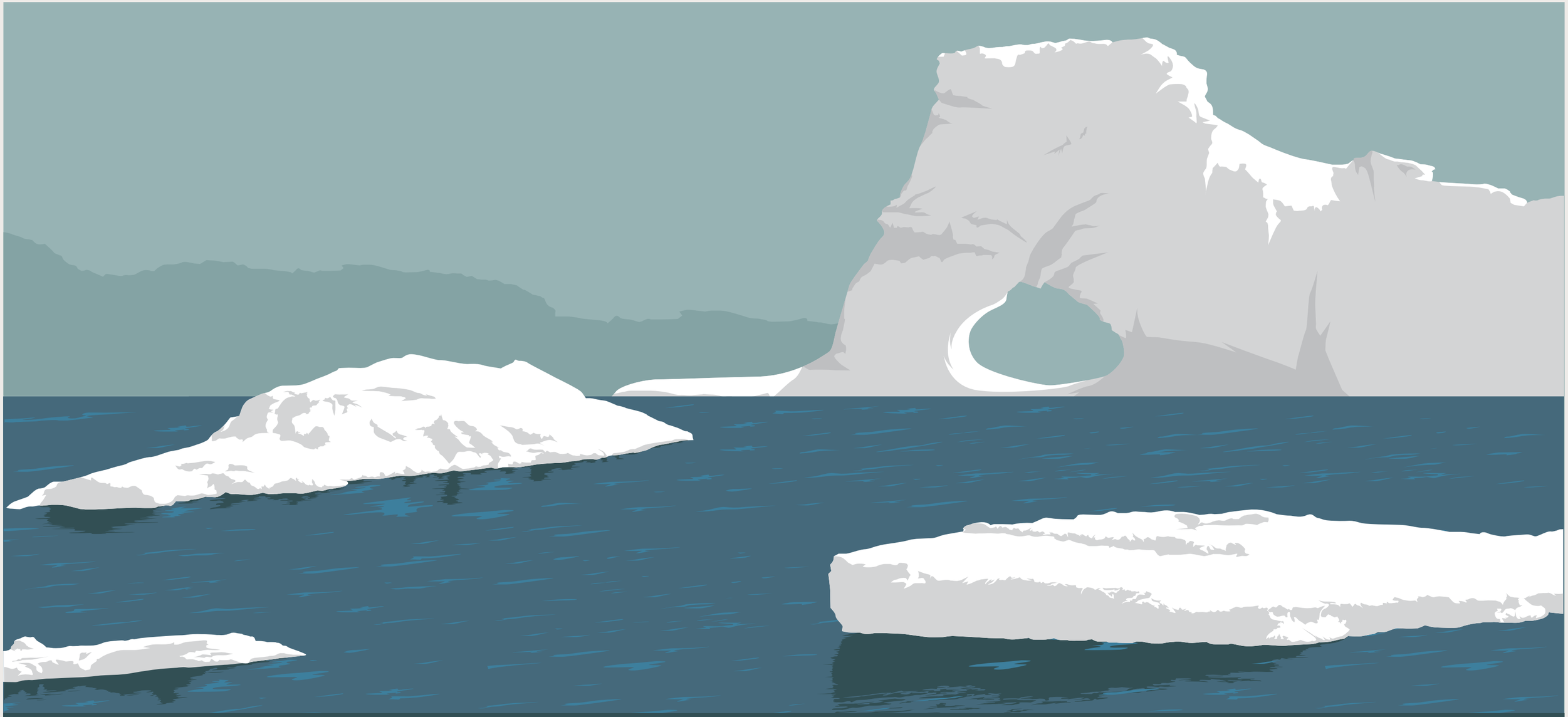
Closure functions can even modify bound variables in the background

```
function buildCoveTicketMaker( transport ) {
        var passengerNumber = 0;
        return function ( name ) {
            passengerNumber++;
            alert("Here is your transportation ticket via the " + transport + ".\n" +
                  "Welcome to the Cold Closures Cove, " + name + "!" +
                  "You are passenger #" + passengerNumber + ".");
        }
}
```

```
var getSubmarineTicket = buildCoveTicketMaker("Submarine");
getSubmarineTicket("Toad");
```



Another call to getSubmarineTicket has passengerNumber incremented to 2! Wow, even though the function's local scope disappeared after Mario's ticket, it KEPT the progress of passengerNumber!



Explore

COLD CLOSURES COVE

LOOPS WITH CLOSURES: A CAUTIONARY TALE

Let's try to make a torpedo assigner for the Cove's Submarine

```
function assignTorpedo ( name, passengerArray ){



}
```

We'll pass in the name of a passenger, as well as a list of passengers.

LOOPS WITH CLOSURES: A CAUTIONARY TALE

Let's try to make a torpedo assigner for the Cove's Submarine

```
function assignTorpedo ( name, passengerArray ){
        var torpedoAssignment;

            This variable will hold a function that alerts name's torpedo assignment.




}
```

LOOPS WITH CLOSURES: A CAUTIONARY TALE

Let's try to make a torpedo assigner for the Cove's Submarine

```
function assignTorpedo ( name, passengerArray ){
        var torpedoAssignment;
        for (var i = 0; i<passengerArray.length; i++) {



        }

}
```

We'll loop over the list of passengers to find name.

LOOPS WITH CLOSURES: A CAUTIONARY TALE

Let's try to make a torpedo assigner for the Cove's Submarine

```
function assignTorpedo ( name, passengerArray ){
        var torpedoAssignment;
        for (var i = 0; i<passengerArray.length; i++) {
            if (passengerArray[i] == name) {
              torpedoAssignment = function ( ) {


              };
            }
        }

}
```

When we find the right name, we'll make a function that will hold our torpedo assignment closure.

LOOPS WITH CLOSURES: A CAUTIONARY TALE

Let's try to make a torpedo assigner for the Cove's Submarine

```
function assignTorpedo ( name, passengerArray ){
        var torpedoAssignment;
        for (var i = 0; i<passengerArray.length; i++) {
            if (passengerArray[i] == name) {
              torpedoAssignment = function ( ) {
                 alert("Ahoy, " + name + "!\n" +
                       "Man your post at Torpedo #" + (i+1) + "!");
              };
            }
        }

}
```

We'll close up the name variable and the loop counter i, and assign a person to the torpedo associated with their index value in the list (adjusted for zero).

LOOPS WITH CLOSURES: A CAUTIONARY TALE

Let's try to make a torpedo assigner for the Cove's Submarine

```
function assignTorpedo ( name, passengerArray ){
        var torpedoAssignment;
        for (var i = 0; i<passengerArray.length; i++) {
            if (passengerArray[i] == name) {
              torpedoAssignment = function ( ) {
                 alert("Ahoy, " + name + "!\n" +
                       "Man your post at Torpedo #" + (i+1) + "!");
              };
            }
        }
        return torpedoAssignment;
}
```

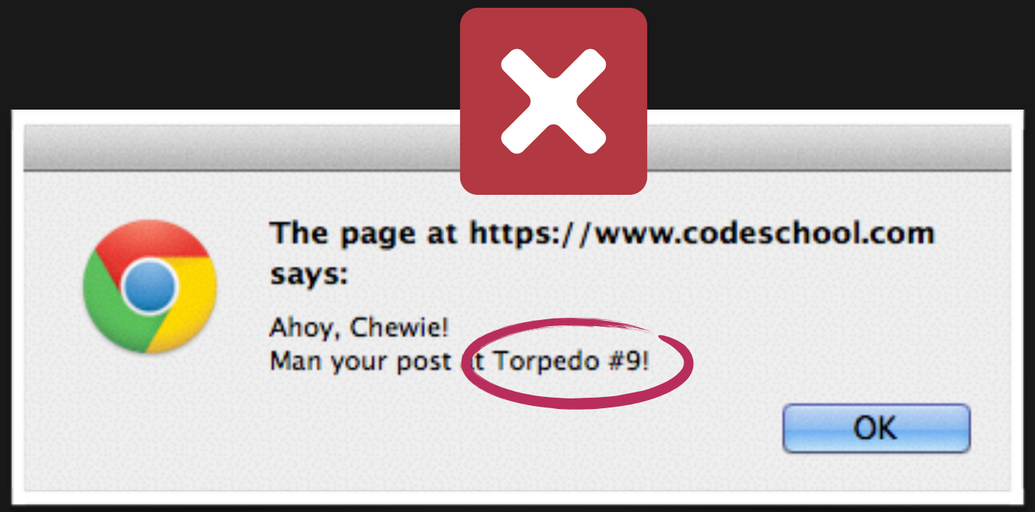
Finally, we'll hand the correct assignment back over to the global scope.

LOOPS WITH CLOSURES: A CAUTIONARY TALE

Let's try to make a torpedo assigner for the Cove's Submarine

```
function assignTorpedo ( name, passengerArray ){
        var torpedoAssignment;
        for (var i = 0; i<passengerArray.length; i++) {
            if (passengerArray[i] == name) {
              torpedoAssignment = function ( ) {
                 alert("Ahoy, " + name + "!\n" +
                        "Man your post at Torpedo #"
              };
            }
        }
        return torpedoAssignment;
}
```

Should be Torpedo #4!
What happened?




```
var subPassengers = ["Luke", "Leia", "Han", "Chewie", "Yoda", "R2-D2", "C-3PO", "Boba"];
```

```
var giveAssignment = assignTorpedo("Chewie", subPassengers);
```

```
giveAssignment();
```

CLOSURES BIND VALUES AT THE VERY LAST MOMENT

We have to pay close attention to return times and final variable states

```
function assignTorpedo ( name, passengerArray ){
        var torpedoAssignment;
        for (var i = 0; i<passengerArray.length; i++) {
            if (passengerArray[i] == name) {
              torpedoAssignment = function ( ) {
                 alert("Ahoy, " + name + "!\n" +
                       "Man your post at Torpedo #" + (i+1) + "!");
              };
            }
        }
        return torpedoAssignment;
}
```

Way before torpedoAssignment is returned, the i loop counter has progressed in value to 8 and stopped the loop.

```
var subPassengers = ["Luke", "Leia", "Han", "Chewie", "Yoda", "R2-D2", "C-3PO", "Boba"];
```

```
var giveAssignment = assignTorpedo("Chewie", subPassengers);
```

```
giveAssignment();
```

CLOSURES BIND VALUES AT THE VERY LAST MOMENT

We have to pay close attention to return times and final variable states

```
function assignTorpedo ( name, passengerArray ){
        var torpedoAssignment;
        for (var i = 0; i<passengerArray.length; i++) {
            if (passengerArray[i] == name) {
              torpedoAssignment = function ( ) {
                 alert("Ahoy, " + name + "!\n" +
                        "Man your post at Torpedo #" + (i+1) + "!");
              };
            }
        }
        return torpedoAssignment;
}
```

8+1=9

The function's actual return is the true "moment of closure," when the environment and all necessary variables are packaged up.

```
var subPassengers = ["Luke", "Leia", "Han", "Chewie", "Yoda", "R2-D2", "C-3PO", "Boba"];
```

```
var giveAssignment = assignTorpedo("Chewie", subPassengers);
```

```
giveAssignment();
```

WHAT CAN WE DO TO ENSURE THE CORRECT VALUE?

Several options exist for timing closures correctly

```
function assignTorpedo ( name, passengerArray ){
        var torpedoAssignment;
        for (var i = 0; i<passengerArray.length; i++) {
            if (passengerArray[i] == name) {
              torpedoAssignment = function ( ) {
                 alert("Ahoy, " + name + "!\n" +
                       "Man your post at Torpedo #" + (i+1) + "!");
              };
            }
        }
        return torpedoAssignment;
}
```

```
var subPassengers = ["Luke", "Leia", "Han", "Chewie", "Yoda", "R2-D2", "C-3PO", "Boba"];
```

WHAT CAN WE DO TO ENSURE THE CORRECT VALUE?

Several options exist for timing closures correctly

```
function assignTorpedo ( name, passengerArray ){

        for (var i = 0; i<passengerArray.length; i++) {
            if (passengerArray[i] == name) {
                                    function ( ) {
                alert("Ahoy, " + name + "!\n" +
                      "Man your post at Torpedo #" + (i+1) + "!");
              };
            }
        }

}
```

```
var subPassengers = ["Luke", "Leia", "Han", "Chewie", "Yoda", "R2-D2", "C-3PO", "Boba"];
```