Basic 05 – Bugs and Errors

# Exercise 01: Retry

Say you have a function primitiveMultiply that in 20 percent of cases multiplies two numbers and in the other 80 percent of cases raises an exception of type MultiplicatorUnitFailure. Write a function that wraps this clunky function and just keeps trying until a call succeeds, after which it returns the result.

Make sure you handle only the exceptions you are trying to handle.

|  |
| --- |
| class MultiplicatorUnitFailure extends Error {}  function primitiveMultiply(a, b) {  if (Math.random() < 0.2) {  return a \* b;  } else {  throw new MultiplicatorUnitFailure("Klunk");  }  }  function reliableMultiply(a, b) {  // Your code here.  }  console.log(reliableMultiply(8, 8));  // → 64 |

# Exercise 01: Locked Box

Consider the following (rather contrived) object:

|  |
| --- |
| const box = {  locked: true,  unlock() {  this.locked = false;  },  lock() {  this.locked = true;  },  \_content: [],  get content() {  if (this.locked) throw new Error("Locked!");  return this.\_content;  }  }; |

It is a box with a lock. There is an array in the box, but you can get at it only when the box is unlocked. Directly accessing the private \_content property is forbidden.

Write a function called withBoxUnlocked that takes a function value as argument, unlocks the box, runs the function, and then ensures that the box is locked again before returning, regardless of whether the argument function returned normally or threw an exception.

|  |
| --- |
| const box = {  locked: true,  unlock() {  this.locked = false;  },  lock() {  this.locked = true;  },  \_content: [],  get content() {  if (this.locked) throw new Error("Locked!");  return this.\_content;  }  };  function withBoxUnlocked(body) {  // Your code here.  }  withBoxUnlocked(function() {  box.content.push("gold piece");  });  try {  withBoxUnlocked(function() {  throw new Error("Pirates on the horizon! Abort!");  });  } catch (e) {  console.log("Error raised:", e);  }  console.log(box.locked);  // → true |