~~~~markdown
## Task 1: Predict Output of Async Code

### Objective
Predict the execution order of asynchronous code involving Promises,
setTimeout, and the event loop.

### Requirements
1. Predict the output order for each code block
2. Run the code and compare with your prediction
3. Explain the execution order based on the event loop

### Code Block 1: Basic Async

```javascript
console.log("1");

setTimeout(function() {
 console.log("2");
}, 0);

Promise.resolve().then(function() {
 console.log("3");
});

console.log("4");
```

**Your Prediction (order):**
```
// Write the order you expect: 1, ?, ?, ?
```

**Actual Output:**
```
// Run and write the actual order
```

**Explanation:**
```
~~~~

~~~
// Explain the execution order
```

### Code Block 2: Nested Async

```javascript
console.log("Start");

setTimeout(function() {
 console.log("Timeout 1");
  Promise.resolve().then(function() {
   console.log("Promise 1");
 });
}, 0);

Promise.resolve().then(function() {
 console.log("Promise 2");
  setTimeout(function() {
   console.log("Timeout 2");
 }, 0);
});

console.log("End");
```

**Your Prediction (order):**
```
// Write the order you expect
```

**Actual Output:**
```
// Run and write the actual order
```

**Explanation:**
```
// Explain the execution order
```
~~~

~~~markdown
### Code Block 3: Multiple Microtasks

```javascript
console.log("A");

Promise.resolve().then(function() {
 console.log("B");
  Promise.resolve().then(function() {
   console.log("C");
 });
  console.log("D");
});

Promise.resolve().then(function() {
 console.log("E");
});

setTimeout(function() {
 console.log("F");
}, 0);

console.log("G");
```

**Your Prediction (order):**
```
// Write the order you expect
```

**Actual Output:**
```
// Run and write the actual order
```

**Explanation:**
```
// Explain the execution order
```
~~~

~~~
### Code Block 4: Complex Async Chain

```javascript
console.log("1");

setTimeout(function() {
 console.log("2");
}, 0);

queueMicrotask(function() {
 console.log("3");
});

Promise.resolve().then(function() {
 console.log("4");
  queueMicrotask(function() {
   console.log("5");
 });
});

setTimeout(function() {
 console.log("6");
}, 0);

console.log("7");
```


**Your Prediction (order):**
```
// Write the order you expect
```


**Actual Output:**
```
// Run and write the actual order
```


**Explanation:**
```
~~~

~~~~
// Explain the execution order
```

### Code Block 5: Async/Await

```javascript
console.log("Start");

async function asyncFunction() {
 console.log("Async 1");
  await Promise.resolve();
 console.log("Async 2");
}

asyncFunction();

Promise.resolve().then(function() {
 console.log("Promise 1");
});

setTimeout(function() {
 console.log("Timeout");
}, 0);

console.log("End");
```

**Your Prediction (order):**
```
// Write the order you expect
```

**Actual Output:**
```
// Run and write the actual order
```

**Explanation:**
```
// Explain the execution order
~~~~

~~~
```
## Task 2: setTimeout Examples

### Objective
Complete and understand various setTimeout scenarios.

### Exercise 1: Basic setTimeout

**Task:** Create a function that logs numbers 1 to 5, with a 1-second
delay between each number.

```javascript
// TODO: Implement countWithDelay function
function countWithDelay() {
 // Your code here
 // Should output: 1 (after 1s), 2 (after 2s), 3 (after 3s), 4 (after 4s),
5 (after 5s)
}

countWithDelay();
```

**Expected Output:**
```
1  // after 1 second
2  // after 2 seconds
3  // after 3 seconds
4  // after 4 seconds
5  // after 5 seconds
```

### Exercise 2: setTimeout in Loop (Fix the Bug)

**Task:** Fix the following code so it logs 0, 1, 2 instead of 3, 3, 3.

```javascript
// BUGGY CODE - Fix this
for (var i = 0; i < 3; i++) {
 setTimeout(function() {
   console.log(i); // Currently logs: 3, 3, 3
~~~

~~~
 }, 1000);
}

// TODO: Fix the code above to log 0, 1, 2



### Exercise 3: setTimeout with Clear

**Task:** Create a countdown timer that counts from 10 to 0, then stops.

```javascript
// TODO: Implement countdown function
function countdown(start) {
 // Your code here
 // Should log numbers from start to 0, with 1 second between each
 // Should stop at 0
}

countdown(10);
// Expected output:
// 10 (immediately)
// 9  (after 1 second)
// 8  (after 2 seconds)
// ...
// 0  (after 10 seconds)
```
~~~