المثال 1 => `async/await` + `setTimeout`

```
async function test() {
  console.log("A");

  setTimeout(() => console.log("B"), 0);

  await Promise.resolve();
  console.log("C");
}

test();
console.log("D");
```

✅ **النتيجة المتوقعة:**

A
D
C
B

الشرح:

1. `A` بتنطبع مباشرة.
2. `setTimeout` بتحط الكولباك بالـ **Callback Queue**.
3. `await Promise.resolve()` بيخلق microtask ← يعني **C** بتنتظر شوي.
4. `D` بينطبع لأنه بعد الـ `test()` مباشرة (ما فيها await).
5. بعد انتهاء الكود العادي ← المايكروتاسك `C` بتشتغل.
6. وأخيراً ← الماكروتاسك `B` بتشتغل من الـ setTimeout.

المثال 2 => `async` functions المتداخلة

```
async function inner() {
  console.log("1");
  await null;
  console.log("2");
}

async function outer() {
  console.log("3");
```

```
  await inner();
  console.log("4");
}

outer();
console.log("5");
```

✅ **النتيجة المتوقعة:**

3
1
5
2
4

الشرح خطوة بخطوة:

1. `outer()` بتبدأ وتطبع 3.
2. `inner()` بتشتغل وتطبع 1.
3. توصل `await null` → تتوقف هون مؤقتاً (microtask).
4. يطلع التنفيذ ويروح يطبع 5.
5. بعد انتهاء الكود الأساسي → المايكروتاسك 2 بتشتغل.
6. بعد ما تخلص `inner()`، بيرجع التنفيذ لـ `outer()` وبيطبع 4.

المثال 3 => وعود داخل setTimeout + async/await

```
setTimeout(() => {
  console.log("A");
  Promise.resolve().then(() => console.log("B"));
  async function inner() {
    console.log("C");
    await null;
    console.log("D");
  }
  inner();
}, 0);
console.log("E");
Promise.resolve().then(() => console.log("F"));
```

✅ **النتيجة المتوقعة:**

E
F
A
C
B
D

الشرح بالتفصيل:

1. `E` أولاً (كود عادي).
2. `Promise.resolve().then(...)` → يضيف `F` للـ microtask queue.
3. ينتهي الكود العادي → المايكروتاسك `F` تشتغل.
4. بعدها يوصل دور الـ setTimeout → تطبع `A`.
5. `B` تروح للـ microtask queue
6. `C` تُطبع فوراً.
7. `await null` توقف `D` مؤقتاً (microtask).
8. بعد انتهاء الكود داخل setTimeout → تشتغل المايكروتاسكات `B` ثم `D`

المثال 4 => سلسلة Promises مع setTimeout

```
Promise.resolve()
  .then(() => {
    console.log("1");
    return new Promise((resolve) => {
      setTimeout(() => {
        console.log("2");
        resolve("3");
      }, 0);
    });
  })
  .then((value) => console.log(value));
console.log("4");
```

✅ **النتيجة المتوقعة:**

4
1
2
3

التحليل:

- الكود العادي أولاً → 4.
- أول then() بيطبع 1 مباشرة (microtask).
- بعدها بيخلق promise جديد فيه `setTimeout` → فـ 2 تتأخر.
- بعد ما تنفذ `resolve("3")` → `"setTimeout"`، فبيكمل then التالي ويطبع 3.

المثال 5 => الوحش الصغير (mix شامل)

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

setTimeout(() => console.log("2"), 0);

Promise.resolve()
  .then(() => {
    console.log("3");
    return Promise.resolve("4");
  })
  .then(console.log);

(async () => {
  console.log("5");
  await null;
  console.log("6");
})();

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

✅ **النتيجة المتوقعة:**

1
5
7
3
4
6
2

الشرح الكامل:

1. 1, 5, 7 (كود متزامن عادي).
2. بعده الـ microtasks:
   - 3 (من الـ promise الأول).
   - 4 (من then الثاني).
   - 6 (من await).
3. أخيراً الـ 2 → macrotask.