14.递归

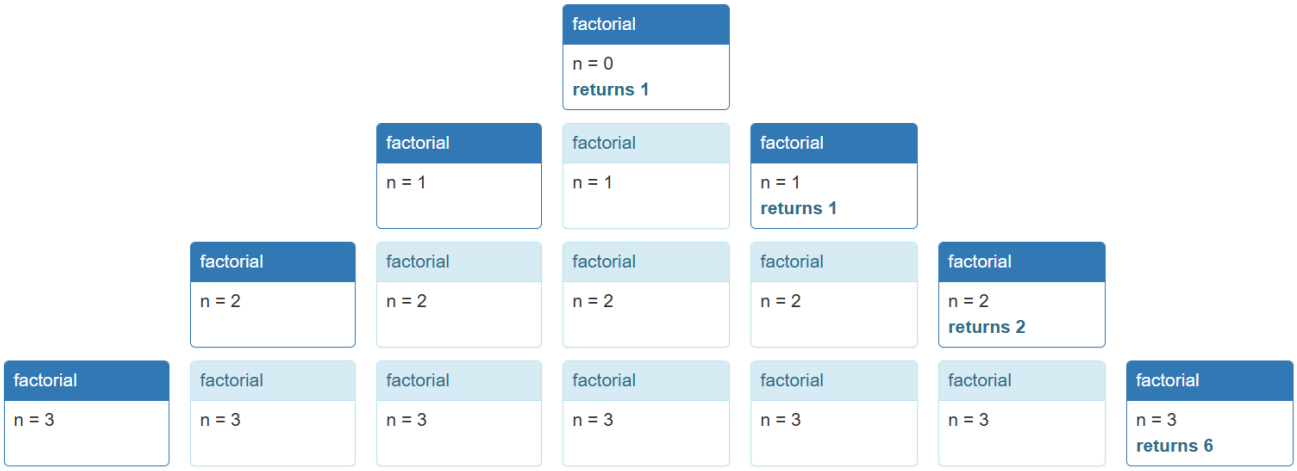
Iterative

```
function factorial(n: number): number {
  let fact = 1;
  for (let i = 1; i <= n; i++) {
    fact = fact * i;
  }
  return fact;
}
```

Recursive

```
function factorial(n: number): number {
  if (n === 0) {
    return 1;
  } else {
    return n * factorial(n-1);
  }
}
```

将递归图形化



递归包括 **基本情况** 和**迭代步骤**

相互递归

```
/**
 * @param filename 文件系统中的路径名
 */
function visitFile(filename: string): void {
    if (fs.lstatSync(filename).isDirectory()) {
        visitChildren(filename, fs.readdirSync(filename));
    }
}

/**
 * @param folder 文件系统中文件夹的路径名（例如 'ps0-bitdiddle'）
 * @param relativePathnames 相对于 folder 的路径名（例如 ['src', 'test', 'package.json', 'tsconfig.json']）
 */
function visitChildren(folder: string, relativePathnames: Array<string>): void {
    for (const pathname of relativePathnames) {
        const filename = path.join(folder, pathname);
        visitFile(filename);
    }
}
```

何时使用递归而非迭代

我们已经看到了使用递归的两个常见原因：

- 问题本身具有自然的递归性质（例如斐波那契数列）。
- 数据本身具有自然的递归结构（例如文件系统）。