The tricky part of the question is to see that some form of iteration/recursion has to be done on the object to access nested fields.

Solution

The first step is to split up the path by the delimiter, which is a period. Then we have to recursively traverse the object given each token in the path, which can be done either with `while` / `for` loops or recursions. The looping should stop when a `null`-ish value is encountered.

Array index accessing doesn't require special handling and can be treated like accessing string-based fields on objects.

```
const arr = [10, 20, 30];
arr[1] === 20; // true
arr['1'] === 20; // true
```

JavaScript TypeScript

```
/**
 * @param {Object} objectParam
 * @param {string|Array<string>} pathParam
 * @param {*} [defaultValue]
 * @return {*}
 */
export default function get(objectParam, pathParam, defaultValue) {
  const path = Array.isArray(pathParam) ? pathParam : pathParam.split('.');

  let index = 0;
  let length = path.length;
  let object = objectParam;

  while (object != null && index < length) {
    object = object[String(path[index])];
    index++;
  }

  const value = index && index === length ? object : undefined;
  return value !== undefined ? value : defaultValue;
}
```

Edge cases

- Bad path inputs like `get(obj, 'a.b..c')` and `get(obj, '')` are unresolved and the `defaultValue` should be returned.
- The solution only works for simple objects. It doesn't work with objects with
  - `Symbol`s as keys.
  - `Map` and `Set` as values.

For these cases you can (and should) clarify the expected behavior with the interviewer.

Notes

- `null` should be differentiated from `undefined`. Hence we should not use `value != undefined` (which is `false` when `value = null`) to check whether to return the `defaultValue`.