# 319.https://stackoverflow.com/questions/71645939/why-am-i-getting-an-error-on-the-array-length-in-solidity

**T:**Why am i getting an error on the array length in solidity?

**Q:**I am developing an NFT smart contract with solidity and am encountering a strange error. I use some functions to add data to mappings inside the contract, both of them loop through arrays that are passed into the contract. They look like this.  
  
function addNameDescImagePricesTowers( uint[] memory ids, string[] memory propertyNames, uint[] memory prices, string[] memory descs, string[] memory images, string[] memory towers, string[] memory districts, string[] memory neighs ) public onlyOwner() { console.log("Started to add base data"); for (uint i = 0; i < ids.length; i ++){ defaultProperties[ids[i]-1] = PropertyAttributes({ id: ids[i], propertyIndex: ids[i]-1, name: propertyNames[i], description: descs[i], image:images[i], features: Features({ tower: towers[i], district: districts[i], neighborhood: neighs[i], primary\_type: '',//primeTypes[i], sub\_type\_1: '',//subType1[i], sub\_type\_2: '',//subType2[i], structure: '',//structures[i], feature\_1: '',//feature1[i], feature\_2: '',//feature2[i], feature\_3: '',//feature3[i], feature\_4: '',//feature4[i], feature\_5: '',//feature5[i], feature\_6: ''//raritys[i] }), extensionCount: 0 }); //created minted list with all false MintedNfts[ids[i]] = false; //record the mint price MintPrice[ids[i]] = prices[i] \* (1 ether); } console.log("added base data"); } function addNftData(uint[] memory ids, string[] memory primes,string[] memory subs1, string[] memory subs2, string[] memory structures, string[] memory fets1, string[] memory fets2, string[] memory fets3, string[] memory fets4, string[] memory fets5, string[] memory fets6 ) public onlyOwner() { for (uint i = 0; i < ids.length; i ++){ uint id = ids[i]; PropertyAttributes memory x = defaultProperties[id-1]; // x.neighborhood = neighs[i]; x.features.primary\_type = primes[i]; x.features.sub\_type\_1 = subs1[i]; x.features.sub\_type\_2 = subs2[i]; x.features.structure = structures[i]; x.features.feature\_1 = fets1[i]; x.features.feature\_2 = fets2[i]; x.features.feature\_3 = fets3[i]; x.features.feature\_4 = fets4[i]; x.features.feature\_5 = fets5[i]; x.features.feature\_6 = fets6[i]; defaultProperties[i] = x; } console.log("added the core data"); }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
However when i run the contract I get an error on the second function:  
  
TypeError: Cannot read property 'length' of undefined  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
As I am passing the exactly the same ids array into the function each time I don't understand why the first one runs fine but the second gives the error?Here is the js code that runs the functions:  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
let txn; //add basic property detials txn = await propertyContract.addNameDescImagePricesTowers(ids,names,prices,descs,images, towers,districts) await txn.wait(); //add remainign detials txn = await propertyContract.addNftData( ids, neighs, primes, sub1s, sub2s, structures, fet1s, fet2s, fet3s, fet4s, fet5s, raritys); await txn.wait();  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
The whole contract looks like this:  
  
// SPDX-License-Identifier: MITpragma solidity ^0.8.1;import "hardhat/console.sol";import "@openzeppelin/contracts/token/ERC721/ERC721.sol";//import '@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol';//access controlimport "@openzeppelin/contracts/access/Ownable.sol";// Helper functions OpenZeppelin provides.import "@openzeppelin/contracts/utils/Counters.sol";import "@openzeppelin/contracts/utils/Strings.sol";import "./libraries/Base64.sol";contract MetropolisWorldGenesis is ERC721, Ownable { // The tokenId is the NFTs unique identifier, it's just a number that goes // 0, 1, 2, 3, etc. using Counters for Counters.Counter; Counters.Counter private \_tokenIds; string private \_contractURI; struct Extension { string catergory; string name; string contractId; string tokenId; } struct PropertyAttributes { uint id; uint propertyIndex; string name; string description; string image; Features features; uint extensionCount; } struct Features { string tower; string district; string neighborhood; string primary\_type; string sub\_type\_1; string sub\_type\_2; string structure; string feature\_1; string feature\_2; string feature\_3; string feature\_4; string feature\_5; string feature\_6; } //store a list of all the NFT's available to mint. //this is built on when the constructor is called. mapping(uint => PropertyAttributes) defaultProperties; //PropertyAttributes[] defaultProperties; //store which has been minted. mapping(uint => bool) public MintedNfts; //store mint prices mapping(uint => uint) public MintPrice; //map the nft tokenid to the atributes mapping(uint256 => PropertyAttributes) public nftHolderAttributes; //map extensions to the token id. mapping(uint256 => mapping(uint => Extension)) extensions; // A mapping from an address => the NFTs tokenId. Gives me an ez way // to store the owner of the NFT and reference it later. mapping(uint256 => address) public nftHolders; constructor() ERC721("Metropolis World Genesis", "METGEN") { console.log("OK I am making the contract, just this once mind"); // I increment \_tokenIds here so that my first NFT has an ID of 1. \_tokenIds.increment(); } function contractURI() public view returns (string memory) { return \_contractURI; } function addNameDescImagePricesTowers( uint[] memory ids, string[] memory propertyNames, uint[] memory prices, string[] memory descs, string[] memory images, string[] memory towers, string[] memory districts, string[] memory neighs ) public onlyOwner() { console.log("Started to add base data"); for (uint i = 0; i < ids.length; i ++){ defaultProperties[ids[i]-1] = PropertyAttributes({ id: ids[i], propertyIndex: ids[i]-1, name: propertyNames[i], description: descs[i], image:images[i], features: Features({ tower: towers[i], district: districts[i], neighborhood: neighs[i], primary\_type: '',//primeTypes[i], sub\_type\_1: '',//subType1[i], sub\_type\_2: '',//subType2[i], structure: '',//structures[i], feature\_1: '',//feature1[i], feature\_2: '',//feature2[i], feature\_3: '',//feature3[i], feature\_4: '',//feature4[i], feature\_5: '',//feature5[i], feature\_6: ''//raritys[i] }), extensionCount: 0 }); //created minted list with all false MintedNfts[ids[i]] = false; //record the mint price MintPrice[ids[i]] = prices[i] \* (1 ether); } console.log("added base data"); } function addNftData(uint[] memory ids, string[] memory primes,string[] memory subs1, string[] memory subs2, string[] memory structures, string[] memory fets1, string[] memory fets2, string[] memory fets3, string[] memory fets4, string[] memory fets5, string[] memory fets6 ) public onlyOwner() { for (uint i = 0; i < ids.length; i ++){ uint id = ids[i]; PropertyAttributes memory x = defaultProperties[id-1]; // x.neighborhood = neighs[i]; x.features.primary\_type = primes[i]; x.features.sub\_type\_1 = subs1[i]; x.features.sub\_type\_2 = subs2[i]; x.features.structure = structures[i]; x.features.feature\_1 = fets1[i]; x.features.feature\_2 = fets2[i]; x.features.feature\_3 = fets3[i]; x.features.feature\_4 = fets4[i]; x.features.feature\_5 = fets5[i]; x.features.feature\_6 = fets6[i]; defaultProperties[i] = x; } console.log("added the core data"); } //Public payable mint //check enough eth sent function publicMint(uint id) public payable { //get current tokenId (starts at 1) uint256 newItemId = \_tokenIds.current(); uint \_propertyIndex = id - 1; //check if minted require(MintedNfts[\_propertyIndex]==false, "Already Minted"); //check if paid enough require(MintPrice[\_propertyIndex] <= msg.value, "Not enough eth paid"); \_safeMint(msg.sender, newItemId); nftHolderAttributes[newItemId] = PropertyAttributes({ id: defaultProperties[\_propertyIndex].id, propertyIndex: \_propertyIndex, name: defaultProperties[\_propertyIndex].name, description: defaultProperties[\_propertyIndex].description, image: defaultProperties[\_propertyIndex].image, features: Features({ tower: defaultProperties[\_propertyIndex].features.tower, district: defaultProperties[\_propertyIndex].features.district, neighborhood: defaultProperties[\_propertyIndex].features.neighborhood, primary\_type: defaultProperties[\_propertyIndex].features.primary\_type, sub\_type\_1: defaultProperties[\_propertyIndex].features.sub\_type\_1, sub\_type\_2: defaultProperties[\_propertyIndex].features.sub\_type\_2, structure: defaultProperties[\_propertyIndex].features.structure, feature\_1: defaultProperties[\_propertyIndex].features.feature\_1, feature\_2: defaultProperties[\_propertyIndex].features.feature\_2, feature\_3: defaultProperties[\_propertyIndex].features.feature\_3, feature\_4: defaultProperties[\_propertyIndex].features.feature\_4, feature\_5: defaultProperties[\_propertyIndex].features.feature\_5, feature\_6: defaultProperties[\_propertyIndex].features.feature\_6 }), extensionCount: 0 }); console.log("Minted NFT w/ tokenId %s and characterIndex %s", newItemId, \_propertyIndex); // Keep an easy way to see who owns what NFT. Maps token id to address nftHolders[newItemId] = msg.sender; // update the minted list MintedNfts[\_propertyIndex] = true; // Increment the tokenId for the next person that uses it. \_tokenIds.increment(); } //free owner only mint function mintPropertyNft(uint id, address toWallet) public onlyOwner(){ //payable //get current tokenId (starts at 1) uint256 newItemId = \_tokenIds.current(); uint \_propertyIndex = id - 1; //check if minted require(MintedNfts[\_propertyIndex]==false, "Already Minted"); \_safeMint(toWallet, newItemId); nftHolderAttributes[newItemId] = PropertyAttributes({ id: defaultProperties[\_propertyIndex].id, propertyIndex: \_propertyIndex, name: defaultProperties[\_propertyIndex].name, description: defaultProperties[\_propertyIndex].description, image: defaultProperties[\_propertyIndex].image, features: Features({ tower: defaultProperties[\_propertyIndex].features.tower, district: defaultProperties[\_propertyIndex].features.district, neighborhood: defaultProperties[\_propertyIndex].features.neighborhood, primary\_type: defaultProperties[\_propertyIndex].features.primary\_type, sub\_type\_1: defaultProperties[\_propertyIndex].features.sub\_type\_1, sub\_type\_2: defaultProperties[\_propertyIndex].features.sub\_type\_2, structure: defaultProperties[\_propertyIndex].features.structure, feature\_1: defaultProperties[\_propertyIndex].features.feature\_1, feature\_2: defaultProperties[\_propertyIndex].features.feature\_2, feature\_3: defaultProperties[\_propertyIndex].features.feature\_3, feature\_4: defaultProperties[\_propertyIndex].features.feature\_4, feature\_5: defaultProperties[\_propertyIndex].features.feature\_5, feature\_6: defaultProperties[\_propertyIndex].features.feature\_6 }), extensionCount: 0 }); console.log("Minted NFT w/ tokenId %s and characterIndex %s", newItemId, \_propertyIndex); // Keep an easy way to see who owns what NFT. Maps token id to address nftHolders[newItemId] = msg.sender; // update the minted list MintedNfts[\_propertyIndex] = true; // Increment the tokenId for the next person that uses it. \_tokenIds.increment(); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Any suggestions gratefully received.

**C1:**Can you share all your smart contract code?

**C2:**@Kerry99 added the whole contract code.

0 **Answer**