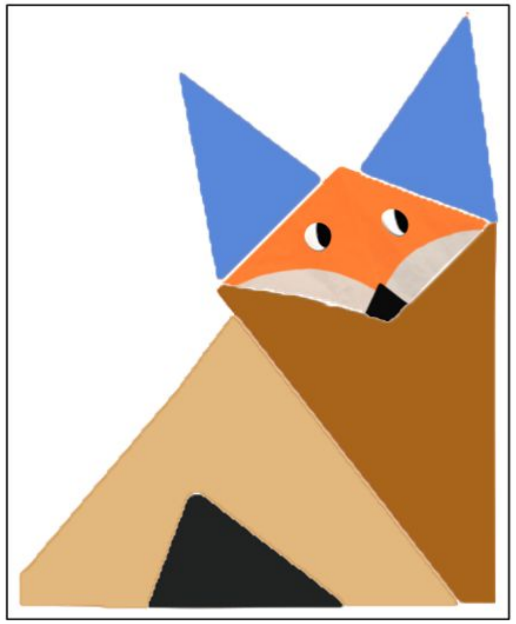
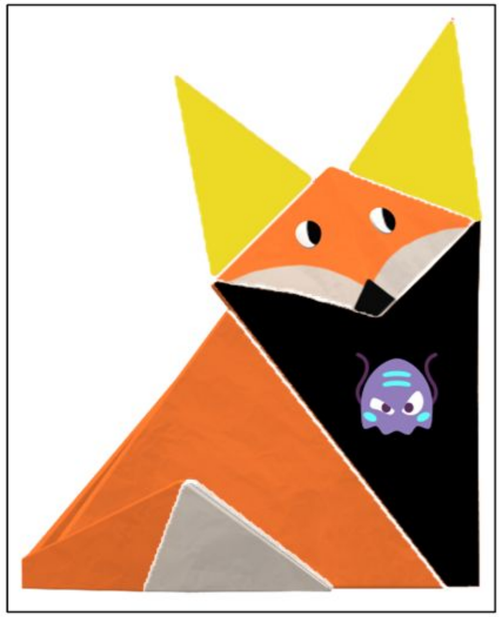
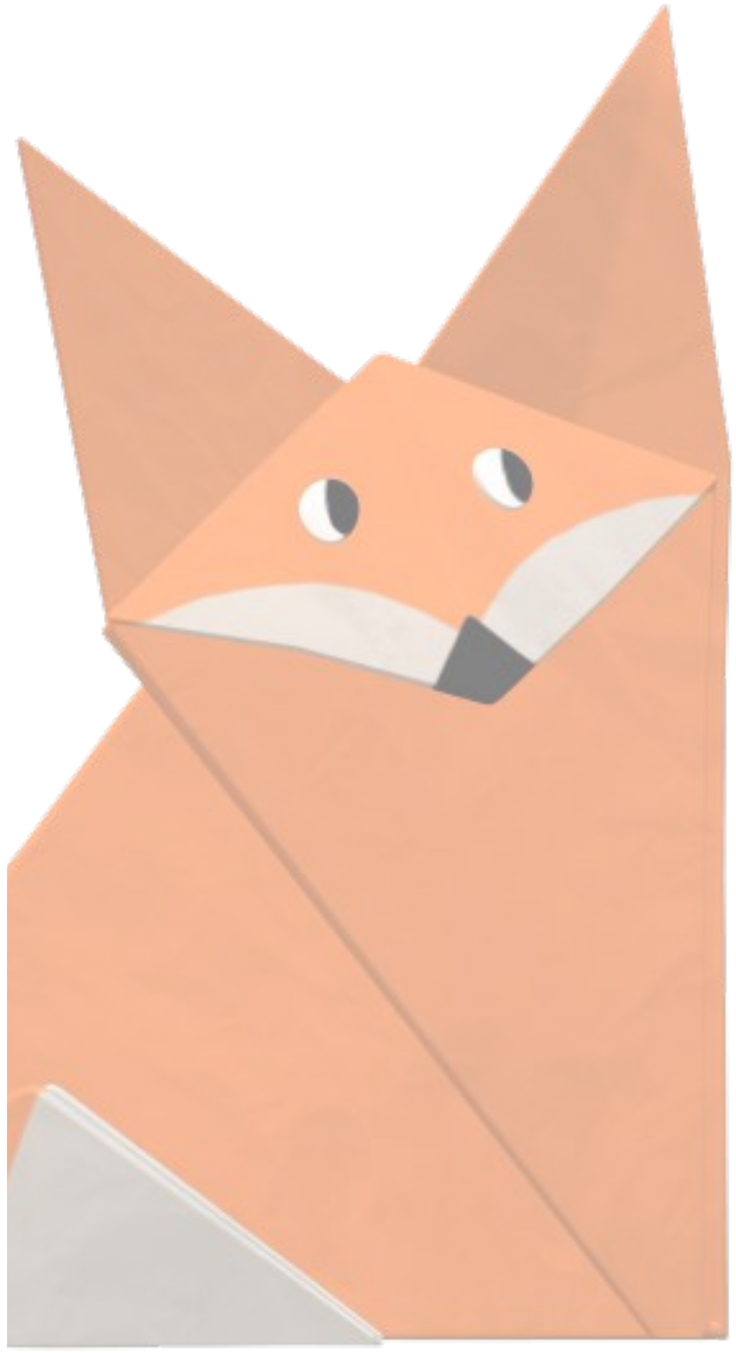
OrigamiFox NFT

@OrigamiFoxNFT
760 randomly generated NFT foxes

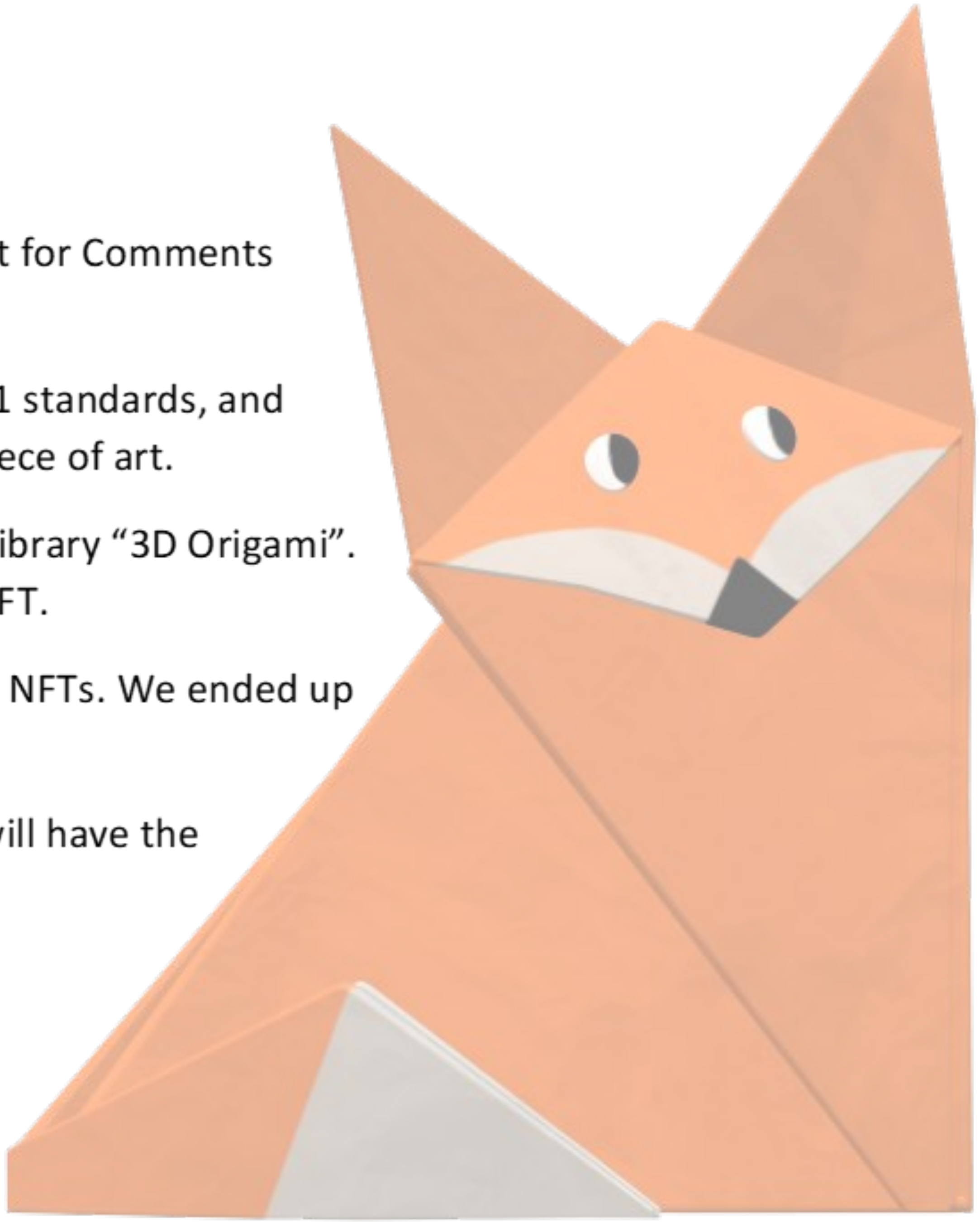
Brought to you by

AQUIBA benarroch, DARIUSH ruch-kamgar, DAVID dannenberg, & TIM moriarty

EXECUTIVE SUMMARY

- Our objective has been to tokenize an asset using Ethereum Request for Comments (ERC) standards.
- Our decision was to create a non-fungible token (NFT) using ERC-721 standards, and to build a decentralized app to facilitate the purchase of a unique piece of art.
- We decided upon an existing .png image from Microsoft's Paint 3D library "3D Origami". Our specific choice was "Origami fox" to use as a template for the NFT.
- Our initial scope was to create a series of 10,000 unique OrigamiFox NFTs. We ended up with creating an initial set of 760 distinct pieces of art.
- Members of the bootcamp cohort and instructional/support team will have the opportunity to buy their own OrigamiFox NFT.



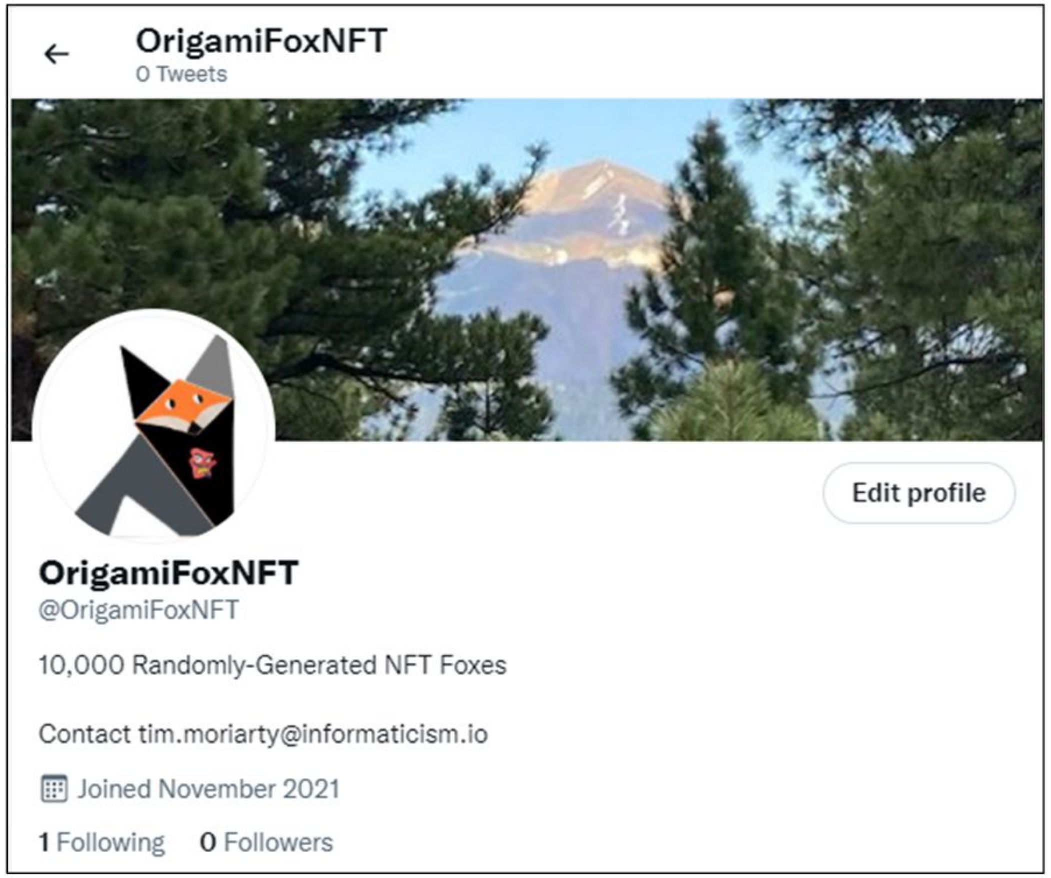
OrigamiFox NFTs:

A collection of 760 unique avatars, each with a distinctive set of traits.

The traits consist of an initial set of four trait categories:

- Ears
- Face / Tail Combo
- Upper Body
- Lower Body

Each trait category contains 5 – 9 distinct combinations of colors and designs, comprising a total of 25 images saved as .png files.


OrigamiFoxNFT
0 Tweets

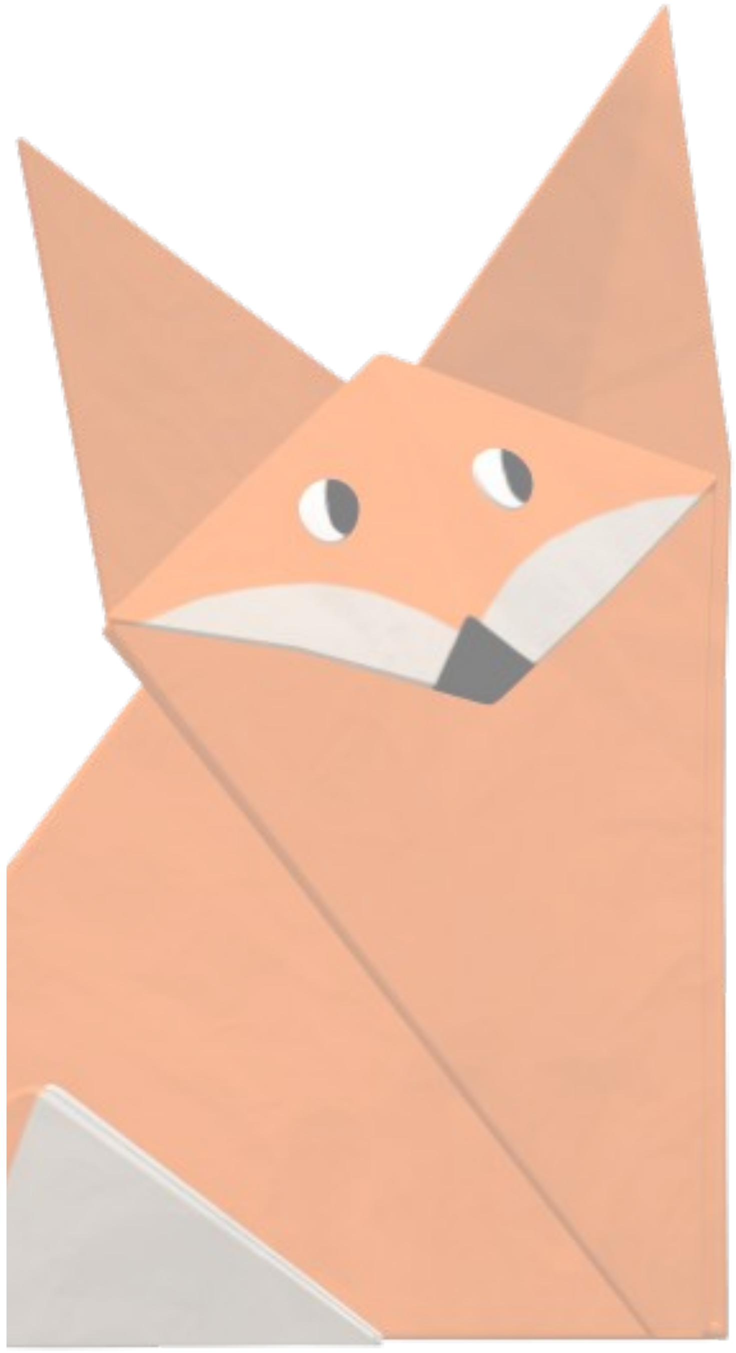
OrigamiFoxNFT
@OrigamiFoxNFT

10,000 Randomly-Generated NFT Foxes

Contact tim.moriarty@informaticism.io

Joined November 2021

1 Following 0 Followers



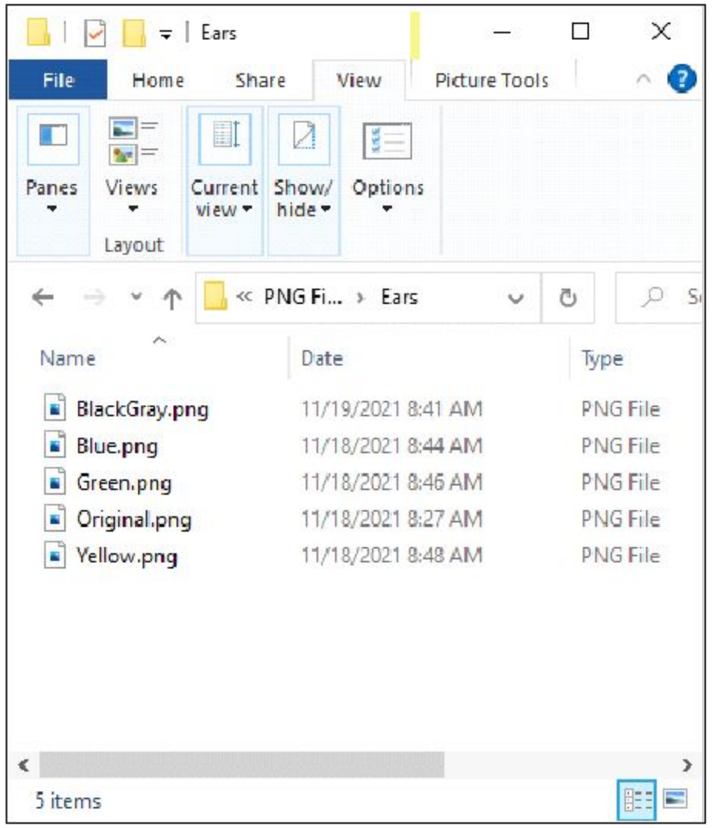
The unique NFTs are created by stacking one .png file from each of the four trait categories. Each of the 25 traits has a “rarity weight” which dictates how often a given trait will appear in an NFT. The following pages show each of the distinctive traits’ .pngs, along with related info for each.

EAR TRAITS:

TRAIT CATEGORY:
Ears

REQUIRED:
True

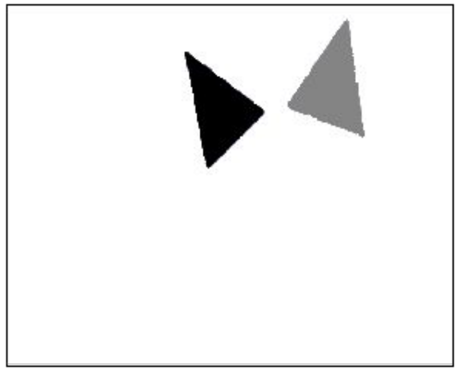
RARITY WEIGHTS:
20,
20,
20,
20,
20


Ears

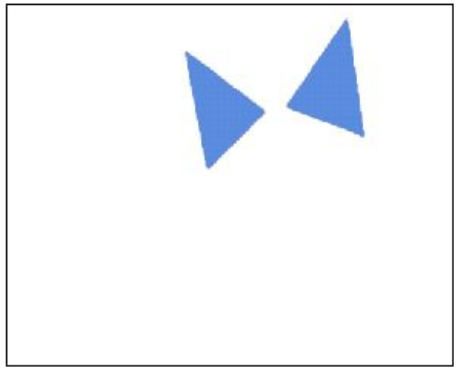
Name	Date	Type
BlackGray.png	11/19/2021 8:41 AM	PNG File
Blue.png	11/18/2021 8:44 AM	PNG File
Green.png	11/18/2021 8:46 AM	PNG File
Original.png	11/18/2021 8:27 AM	PNG File
Yellow.png	11/18/2021 8:48 AM	PNG File

5 items

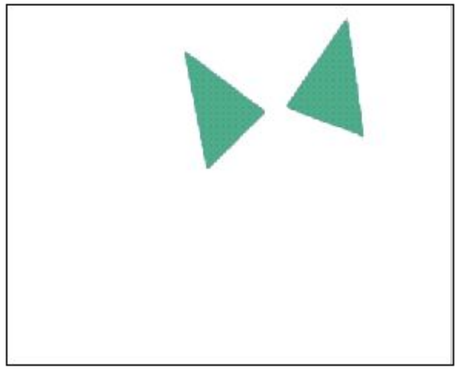
BlackGray.png



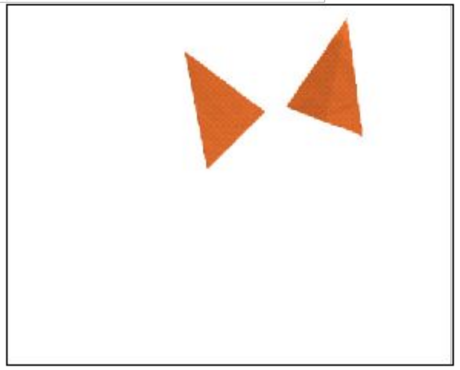
Blue.png



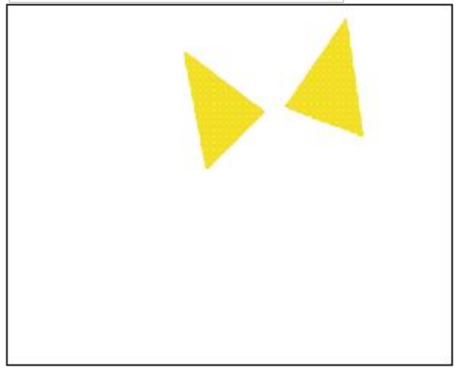
Green.png

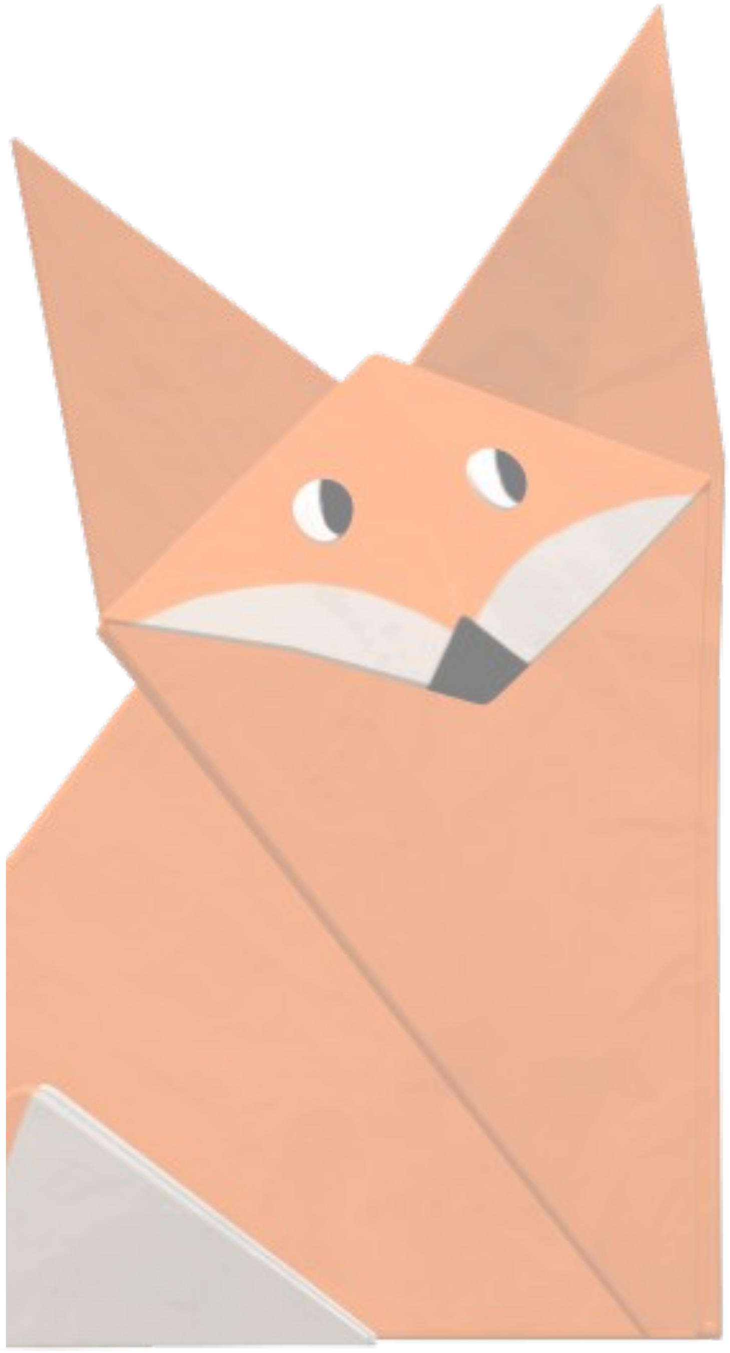


Original.png



Yellow.png



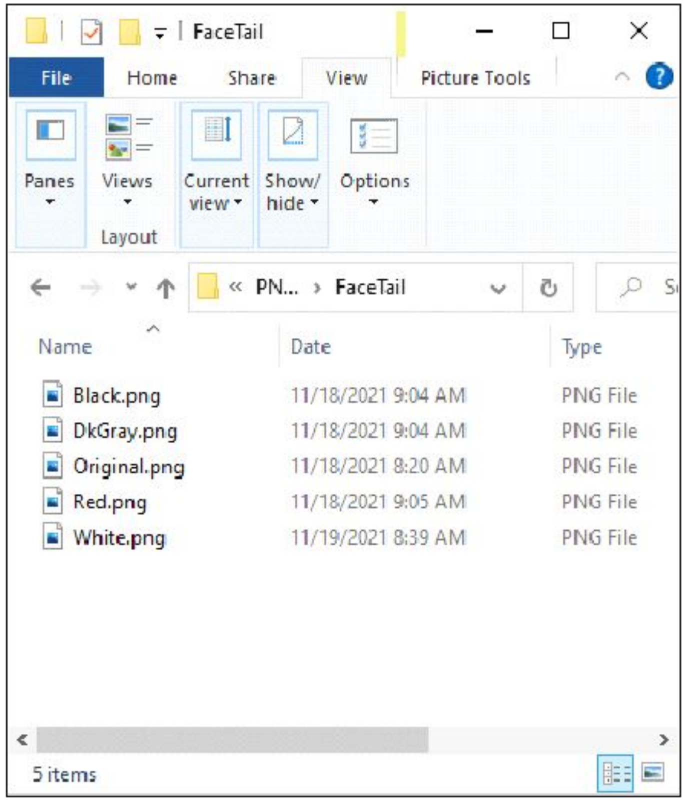


FACE/TAIL TRAITS:

TRAIT CATEGORY:
FaceTail

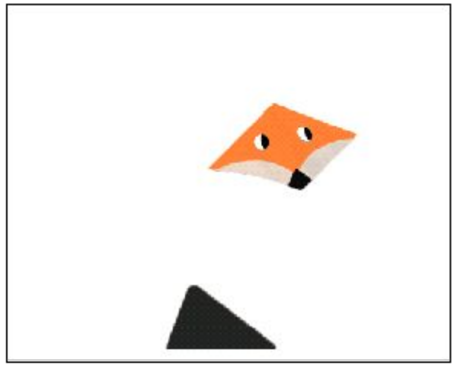
REQUIRED:
True

RARITY WEIGHTS:
22,
22,
22,
22,
12

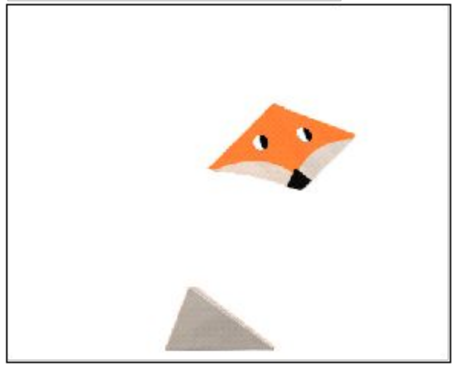


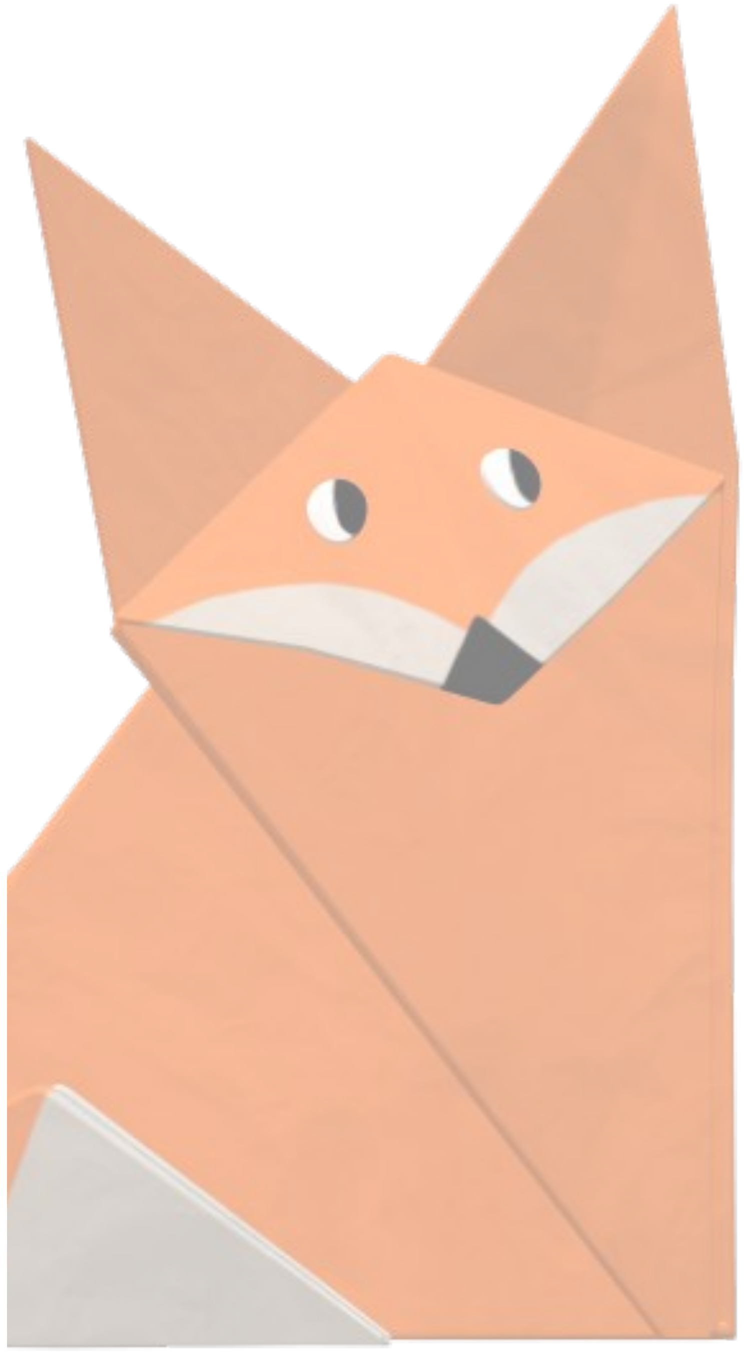
Name	Date	Type
Black.png	11/18/2021 9:04 AM	PNG File
DkGray.png	11/18/2021 9:04 AM	PNG File
Original.png	11/18/2021 8:20 AM	PNG File
Red.png	11/18/2021 9:05 AM	PNG File
White.png	11/19/2021 8:39 AM	PNG File

BlackGray.png

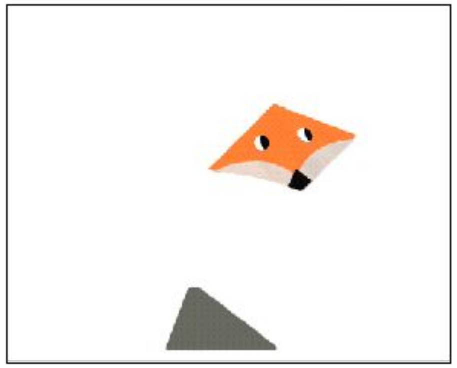


Original.png

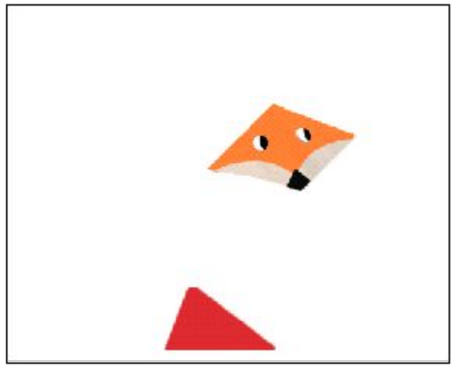




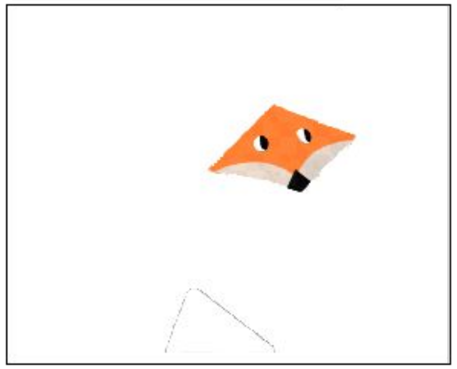
DkGray.png



Red.png



White.png

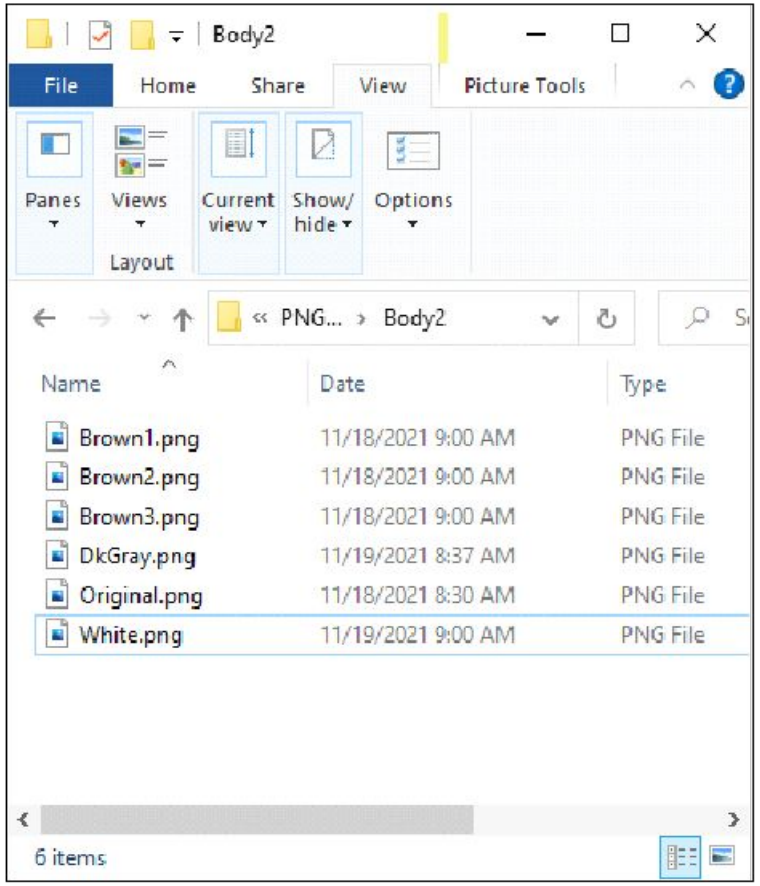


LOWER BODY TRAITS:

TRAIT CATEGORY:
Body2

REQUIRED:
True

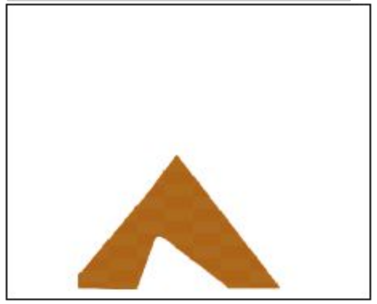
RARITY WEIGHTS:
18,
18,
18,
18,
18,
10



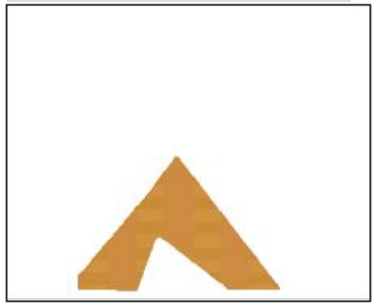
Name	Date	Type
Brown1.png	11/18/2021 9:00 AM	PNG File
Brown2.png	11/18/2021 9:00 AM	PNG File
Brown3.png	11/18/2021 9:00 AM	PNG File
DkGray.png	11/19/2021 8:37 AM	PNG File
Original.png	11/18/2021 8:30 AM	PNG File
White.png	11/19/2021 9:00 AM	PNG File

6 items

Brown1.png

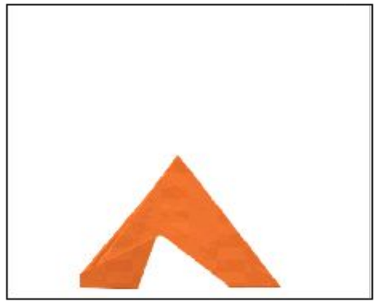


Brown2.png

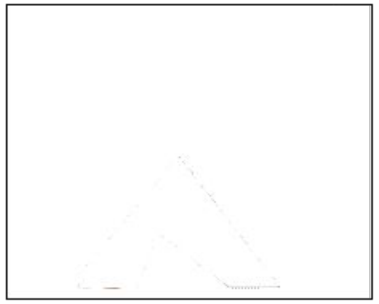


DkGray.png

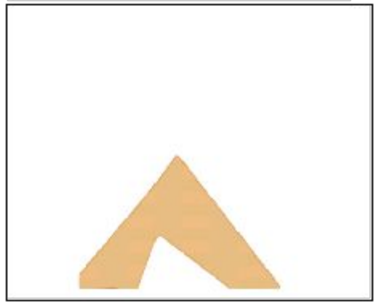
Original.png

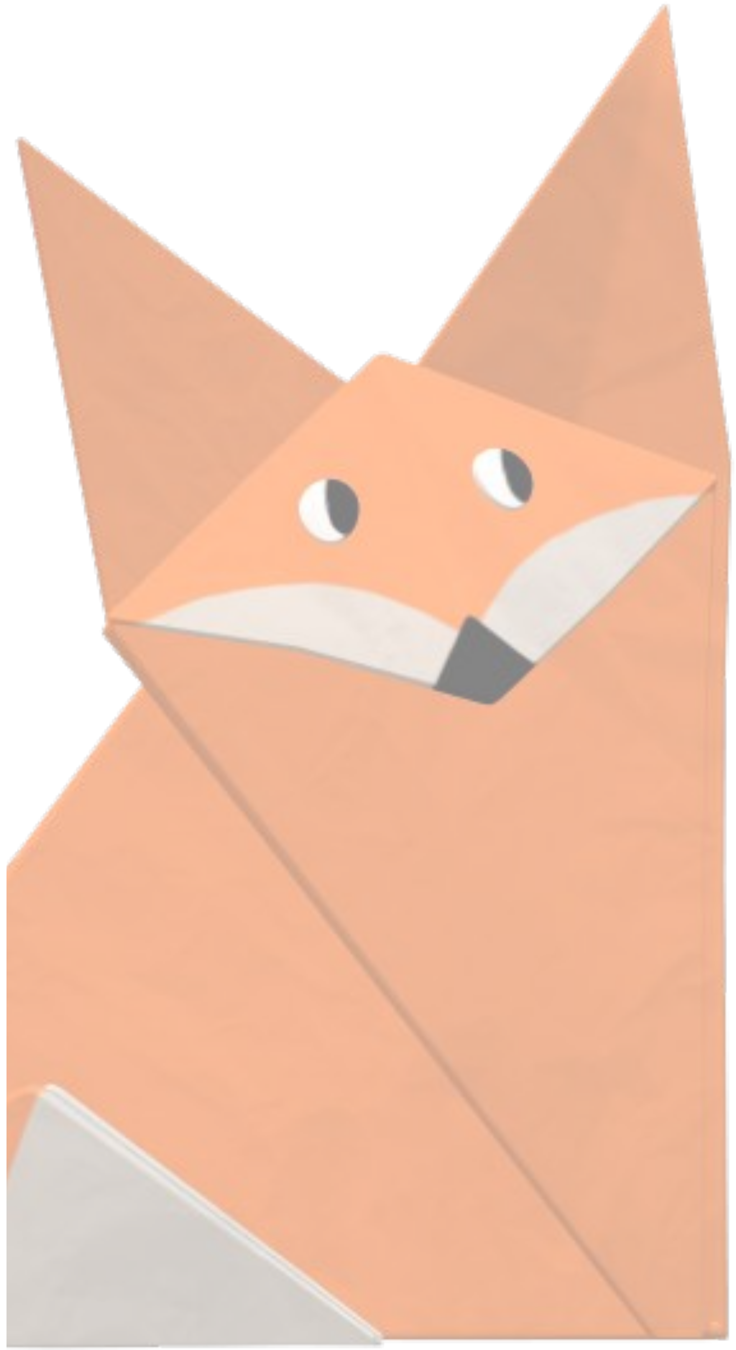


White.png



Brown3.png



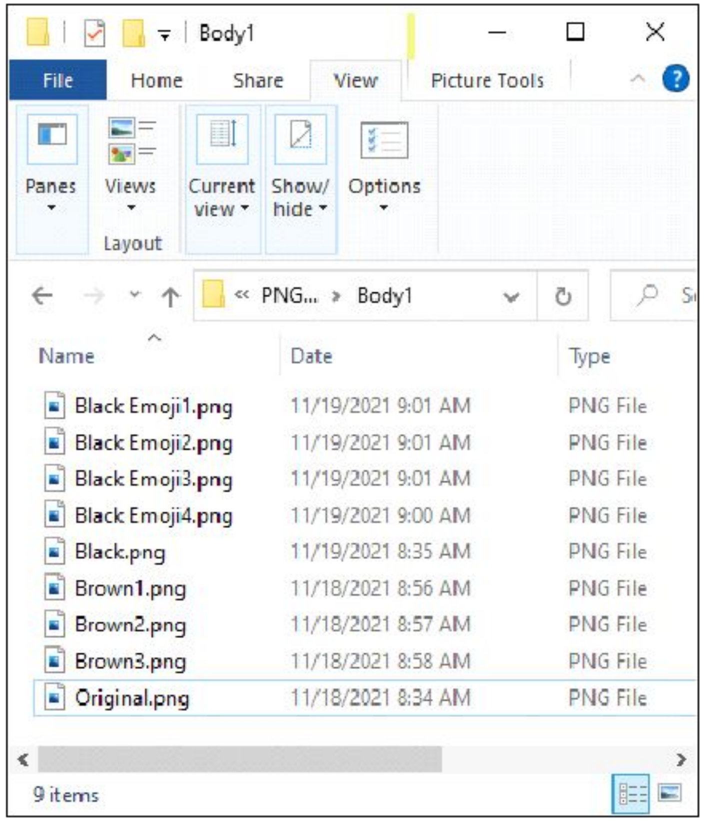


UPPER BODY TRAITS (standard):

TRAIT CATEGORY:
Body1

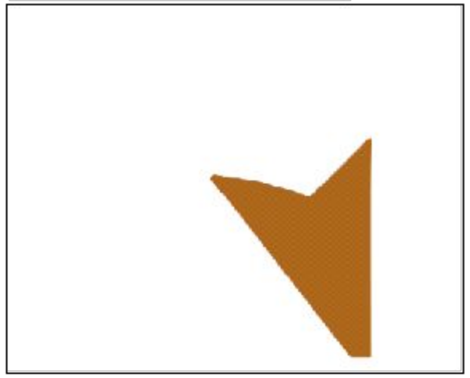
REQUIRED:
True

RARITY WEIGHTS:
5,
5,
5,
5,
16,
16,
16,
16,
16

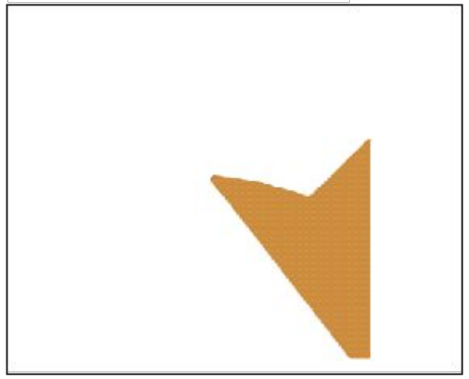


Name	Date	Type
Black Emoji1.png	11/19/2021 9:01 AM	PNG File
Black Emoji2.png	11/19/2021 9:01 AM	PNG File
Black Emoji3.png	11/19/2021 9:01 AM	PNG File
Black Emoji4.png	11/19/2021 9:00 AM	PNG File
Black.png	11/19/2021 8:35 AM	PNG File
Brown1.png	11/18/2021 8:56 AM	PNG File
Brown2.png	11/18/2021 8:57 AM	PNG File
Brown3.png	11/18/2021 8:58 AM	PNG File
Original.png	11/18/2021 8:34 AM	PNG File

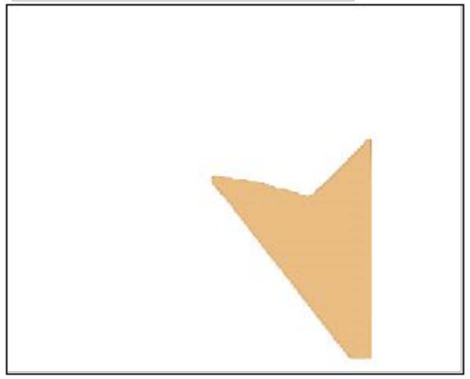
Brown1.png



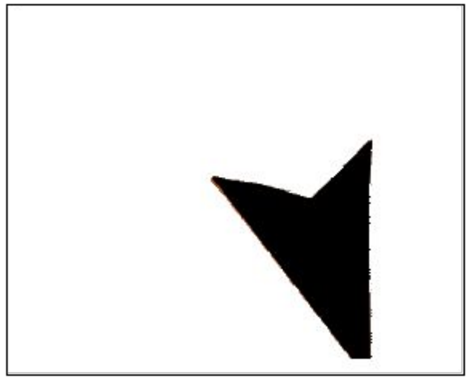
Brown2.png



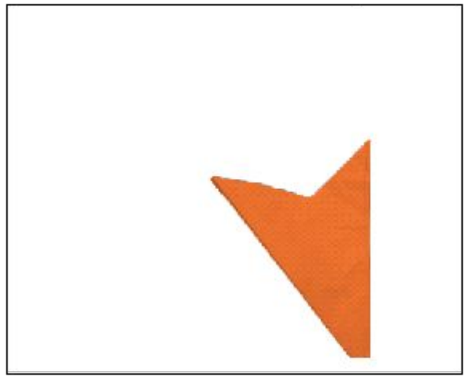
Brown3.png

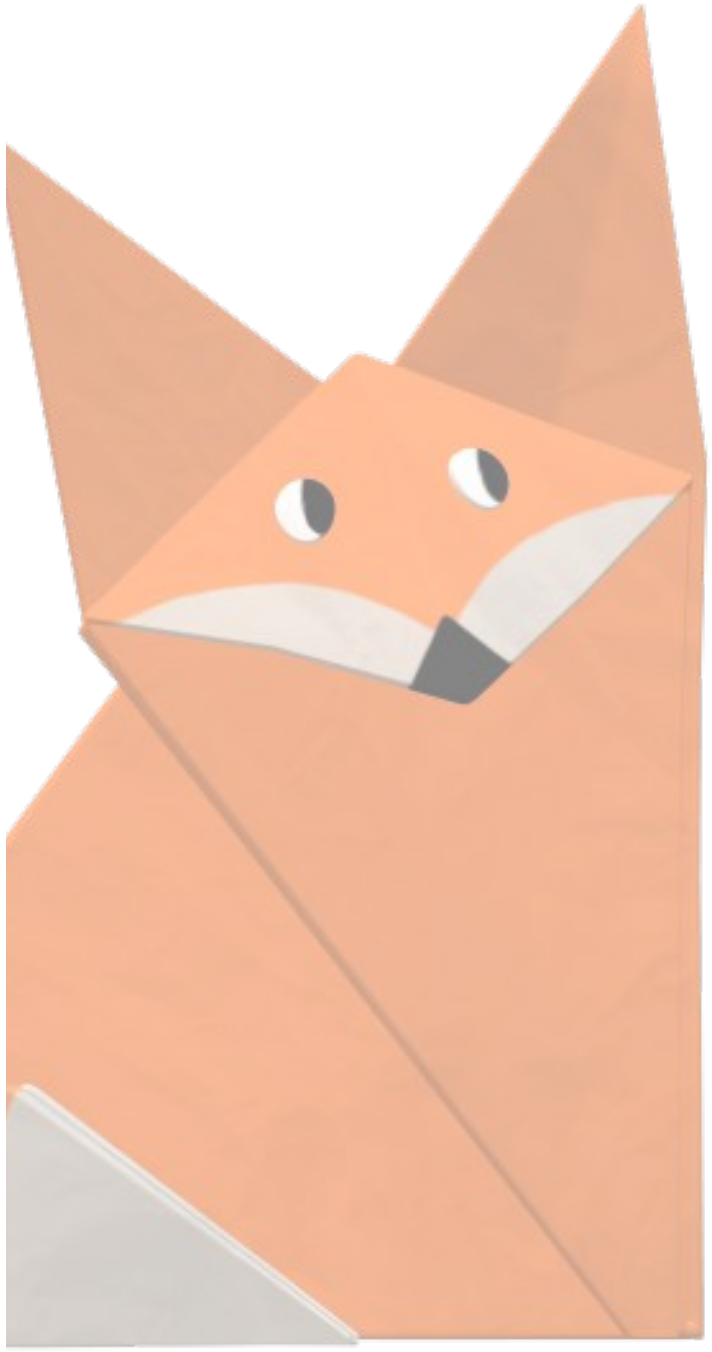


Black.png



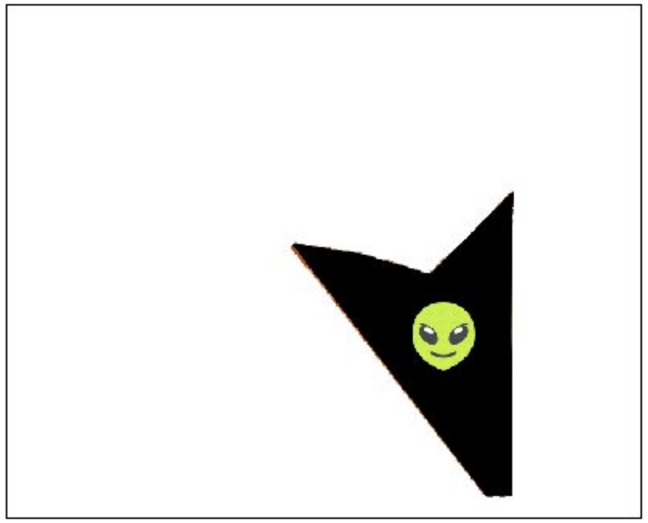
Original.png



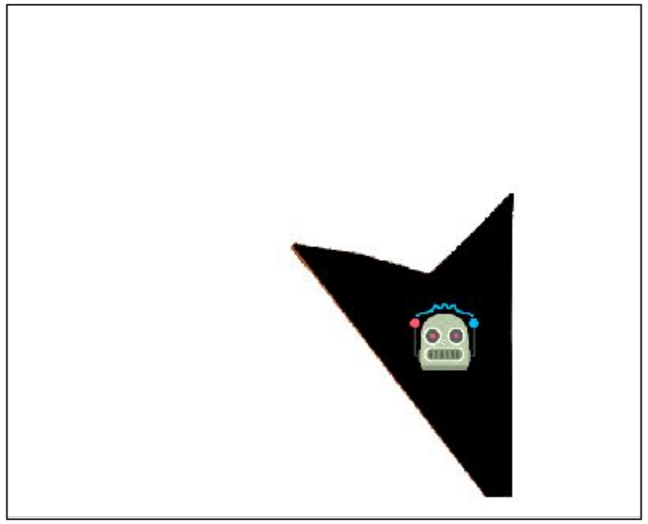


UPPER BODY TRAITS (with emoji glyphs):

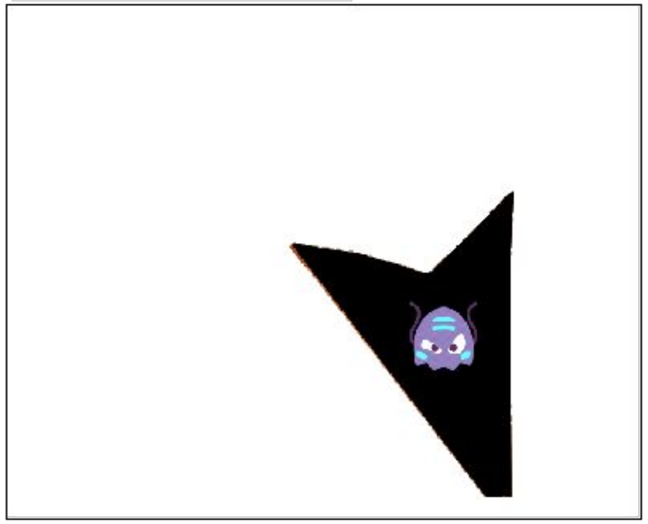
BlackEmoji1.png



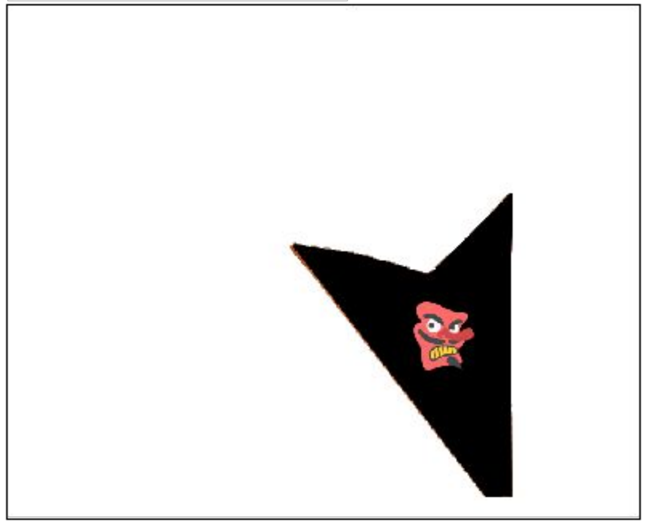
BlackEmoji2.png

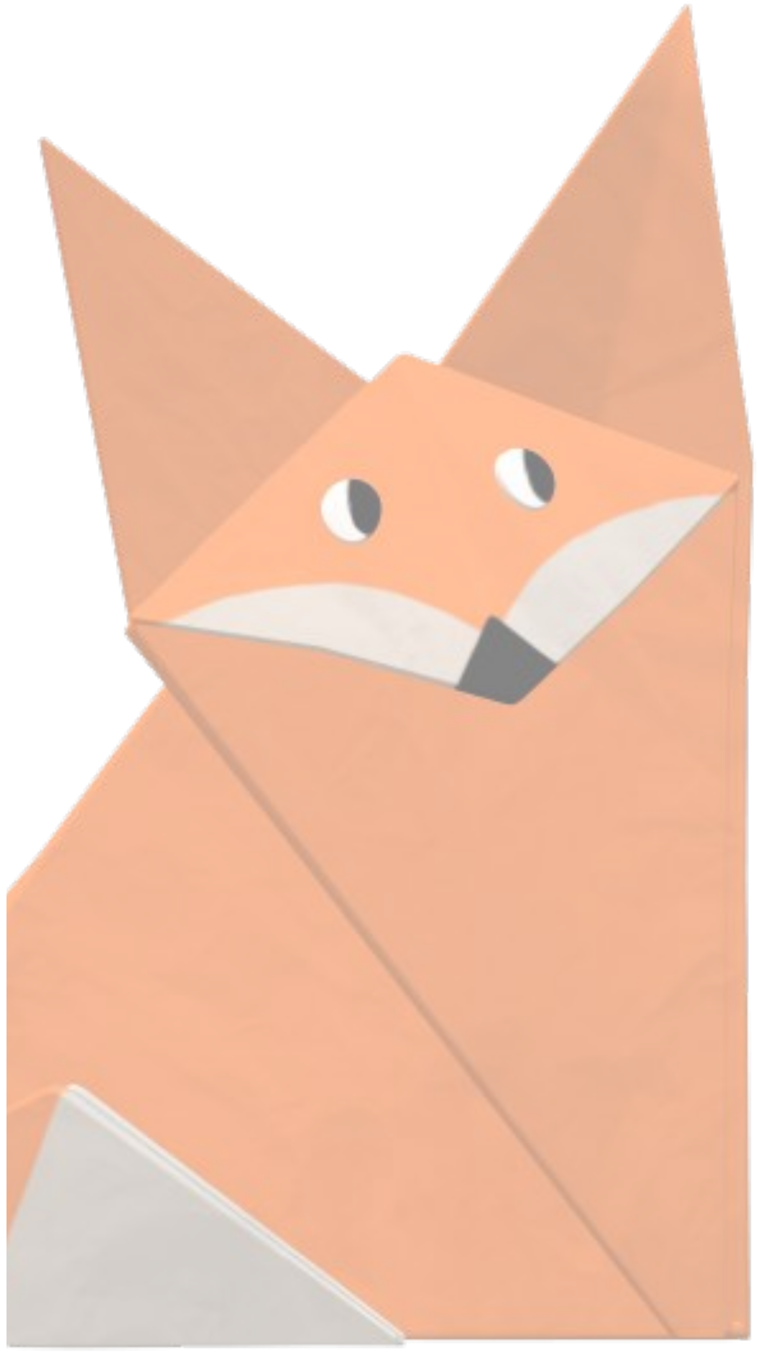


BlackEmoji3.png

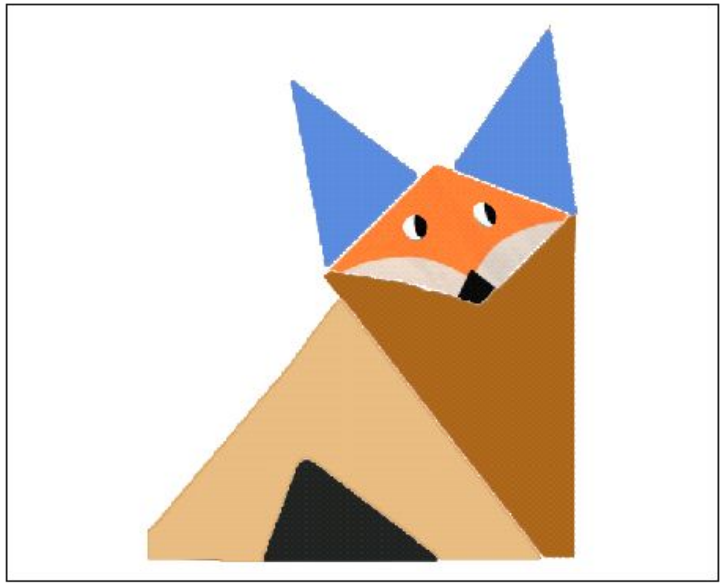


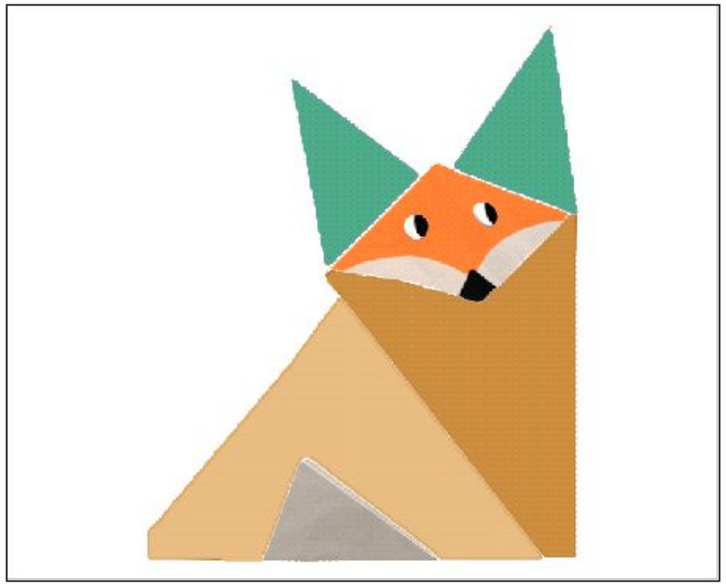
BlackEmoji4.png

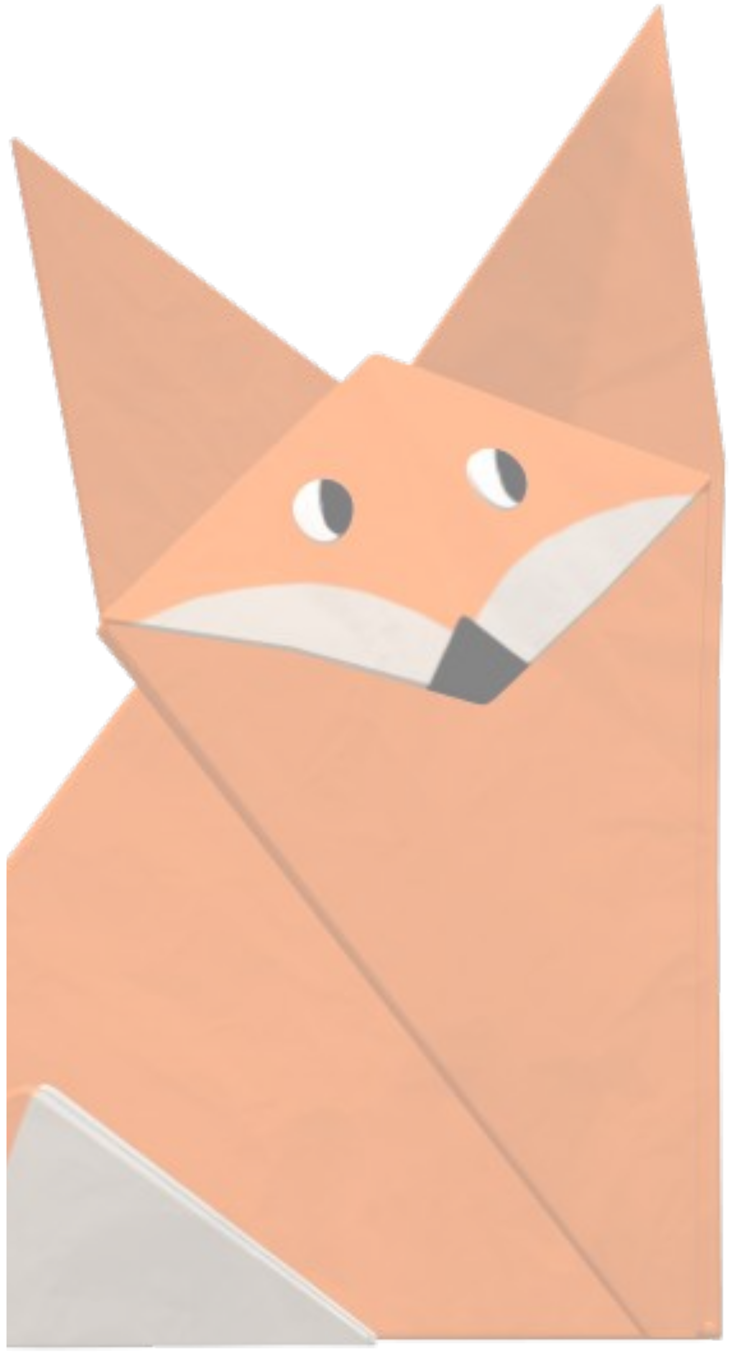


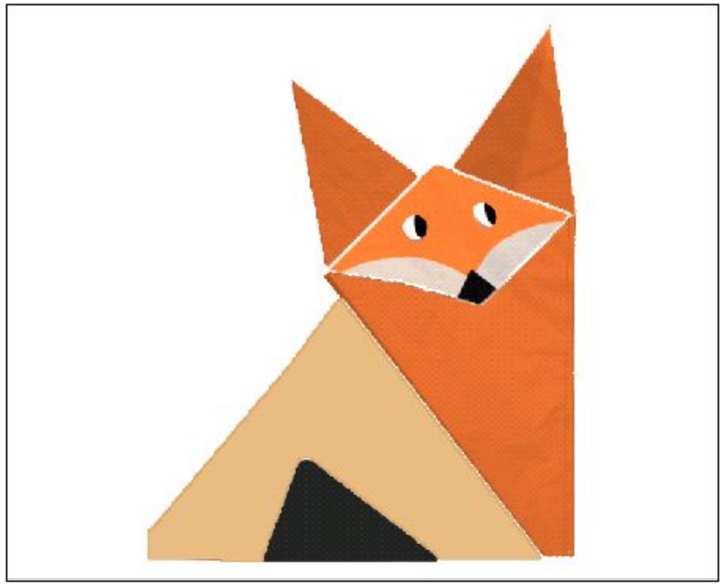


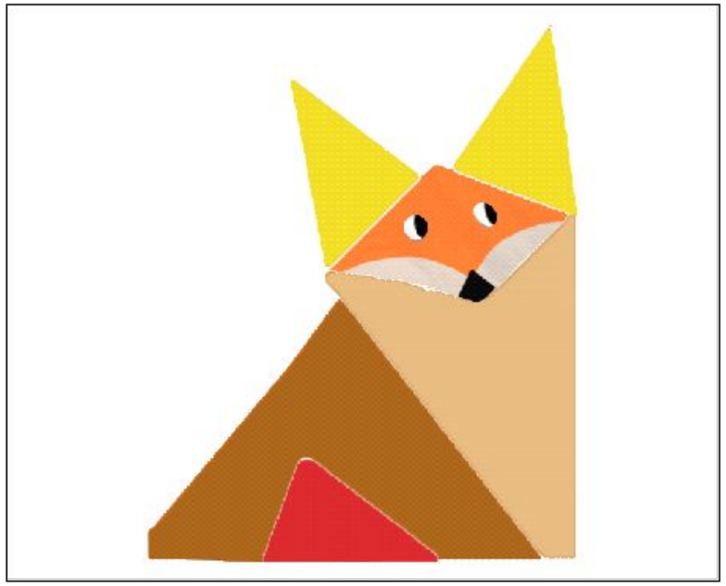
PROTOTYPES (without emoji glyphs):



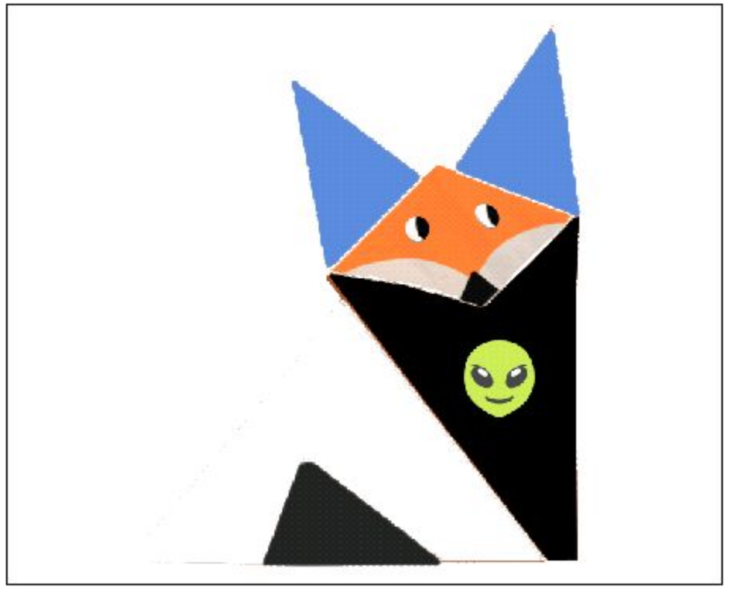


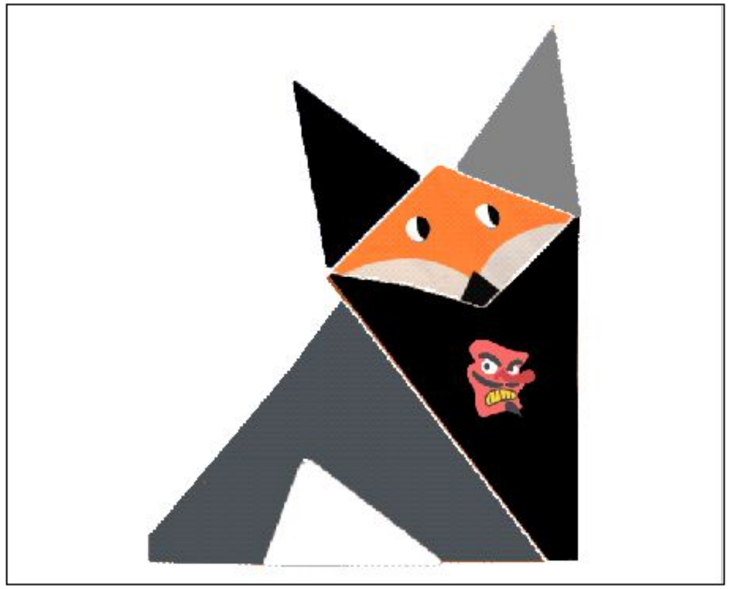


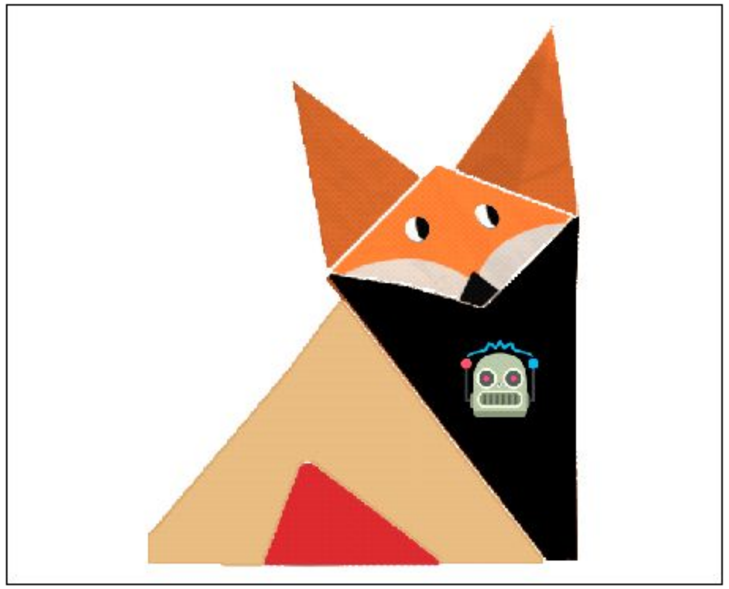


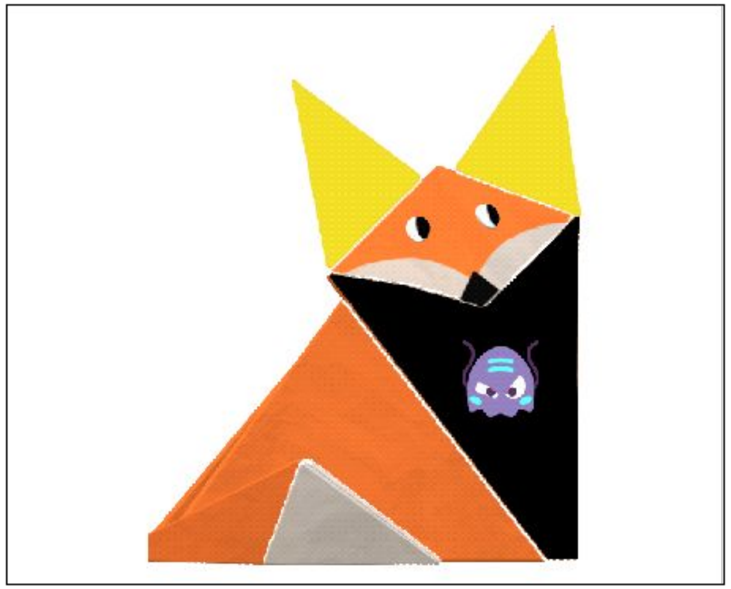


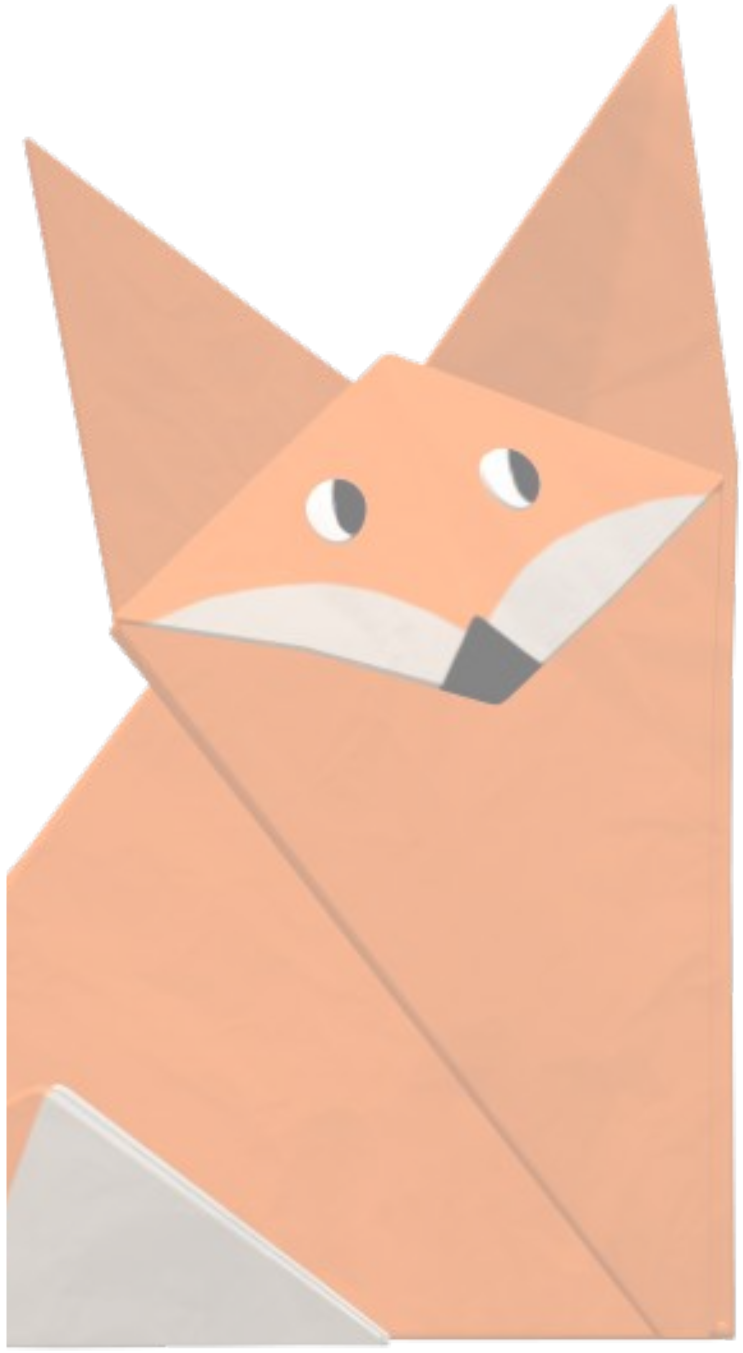
PROTOTYPES (with emoji glyphs):









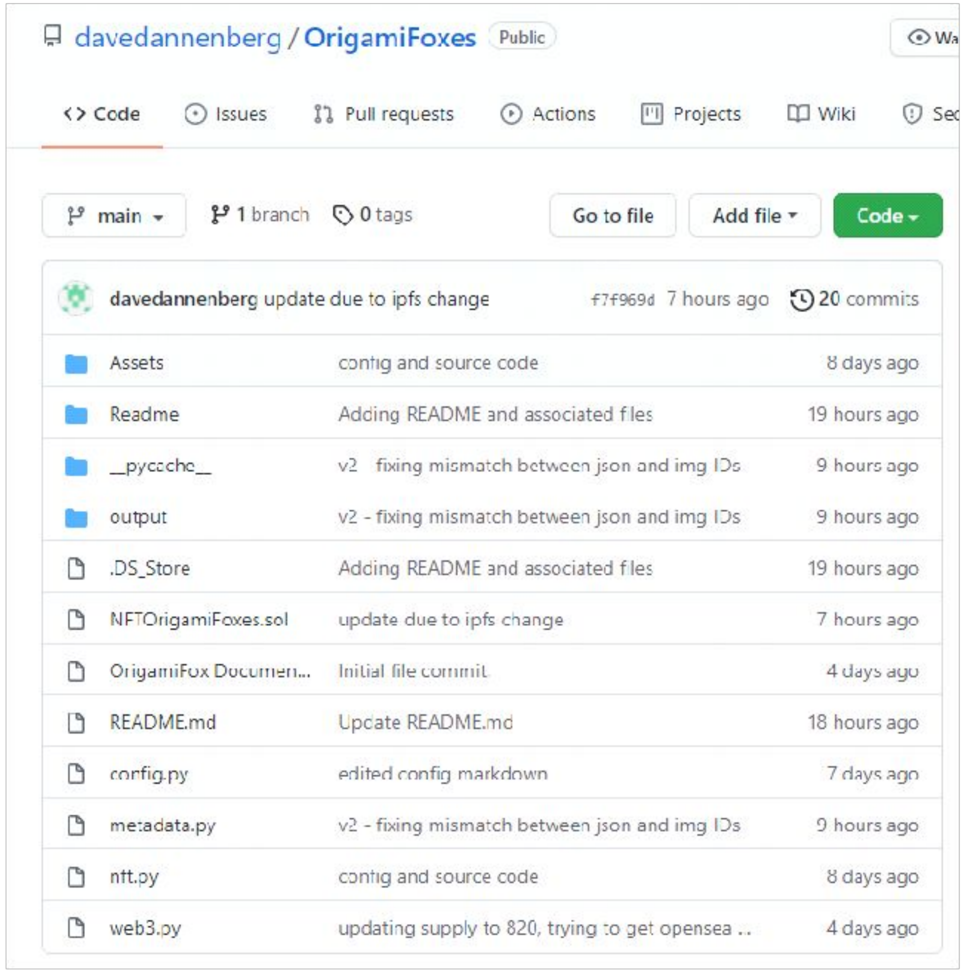


config.py

(note: The following methodology has been co-opted from an article by Rounak Barik re: Scrappy Squirrels NFTs)

We implemented a Python list variable called CONFIG which contains a dictionary for each of our four trait categories, each with five keys: id, name, directory, required, and rarity_weights.

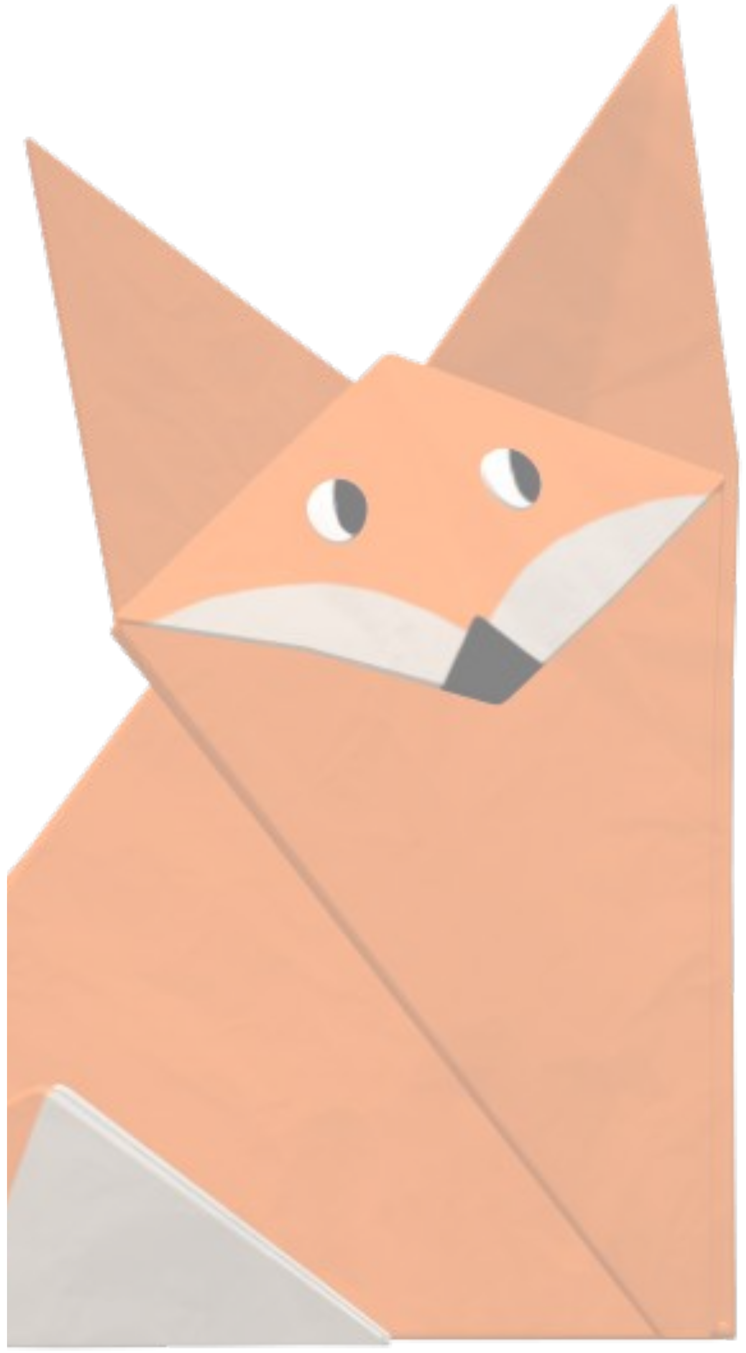
```
CONFIG = [
  {   'id': 1,
      'name': 'ear_traits',
      'directory': 'Ear',
      'required': True,
      'rarity_weights': [20, 20, 20, 20, 20]},
  {   'id': 2,
      'name': 'face_tail_traits',
      'directory': 'FaceTail',
      'required': True,
      'rarity_weights': [22, 22, 22, 22, 12]},
  {   'id': 3,
      'name': 'lower_body_traits',
      'directory': 'Body2',
      'required': True,
      'rarity_weights': [18, 18, 18, 18, 18, 10]},
  {   'id': 4,
      'name': 'upper_body_traits',
      'directory': 'Body1',
      'required': True,
      'rarity_weights': [5, 5, 5, 5, 16, 16, 16, 16, 16]},
]
```



davedannenberg / OrigamiFoxes Public

davedannenberg update due to ipfs change f7f969d 7 hours ago 20 commits

Assets	config and source code	8 days ago
Readme	Adding README and associated files	19 hours ago
__pycache__	v2 fixing mismatch between json and img IDs	9 hours ago
output	v2 - fixing mismatch between json and img IDs	9 hours ago
.DS_Store	Adding README and associated files	19 hours ago
NFTOrigamiFoxes.sol	update due to ipfs change	7 hours ago
OrigamiFox Documen...	Initial file commit	4 days ago
README.md	Update README.md	18 hours ago
config.py	edited config markdown	7 days ago
metadata.py	v2 - fixing mismatch between json and img IDs	9 hours ago
nft.py	config and source code	8 days ago
web3.py	updating supply to 820, trying to get opensea ..	4 days ago



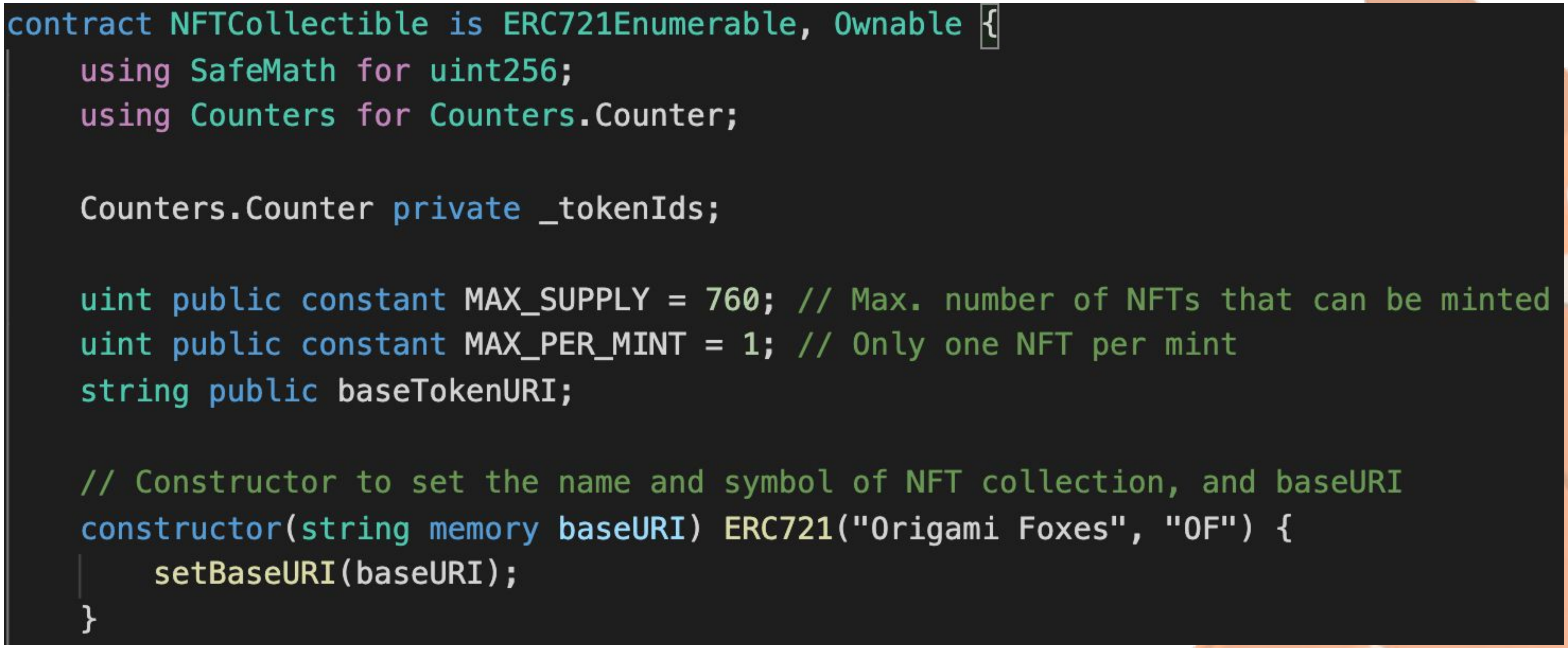
Writing the Smart Contract

Imported OpenZeppelin contracts

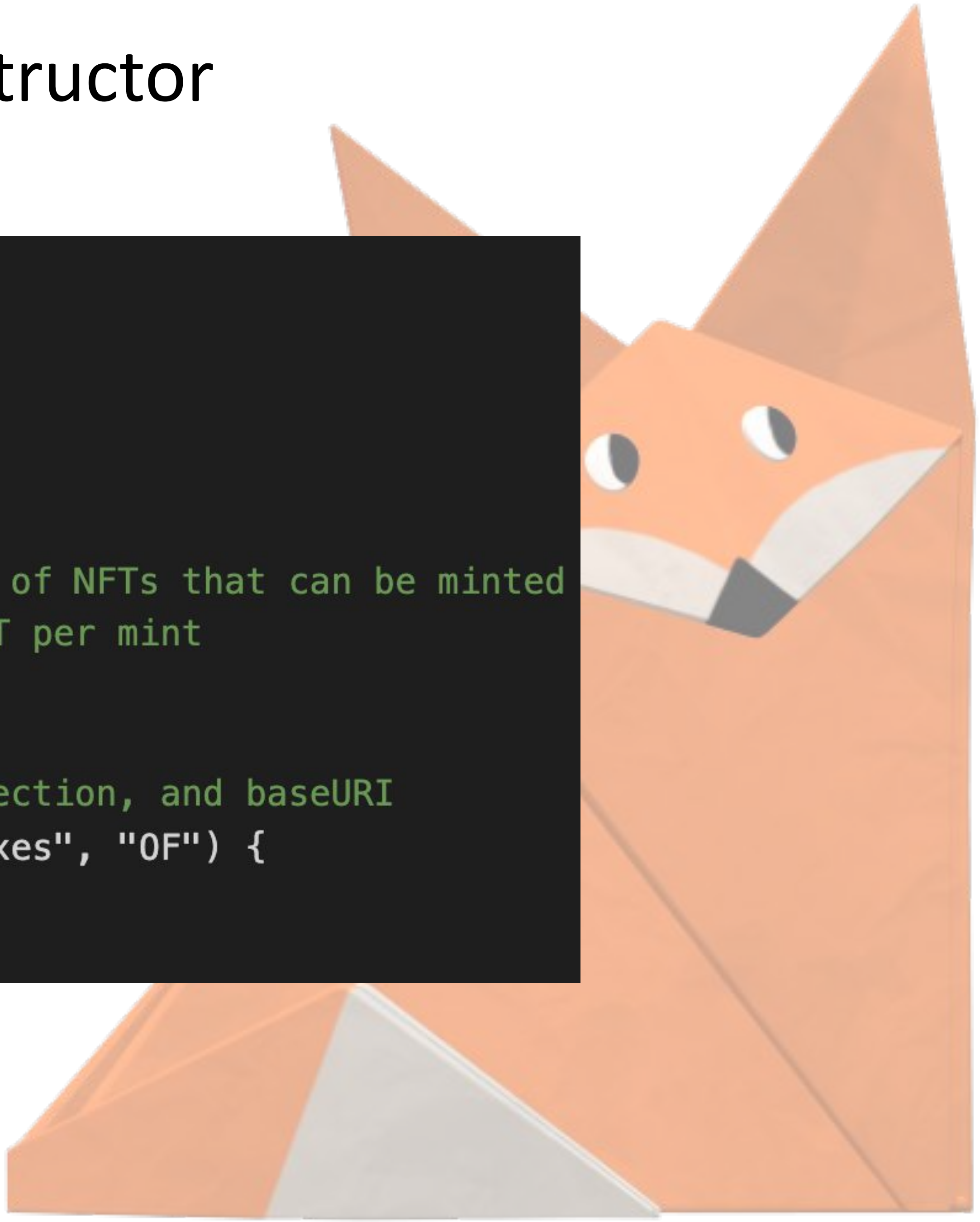
Inherited the following smart contract functionality:

- *ERC721* to import the basic NFT functions
- *IERC721Metadata* to provide NFT metadata
- *SafeMath* for robust math functions
- *Ownable* for basic authorization control functions
- *Counters* to count called tokenIds (NFTs)

Storage constants, variables and constructor

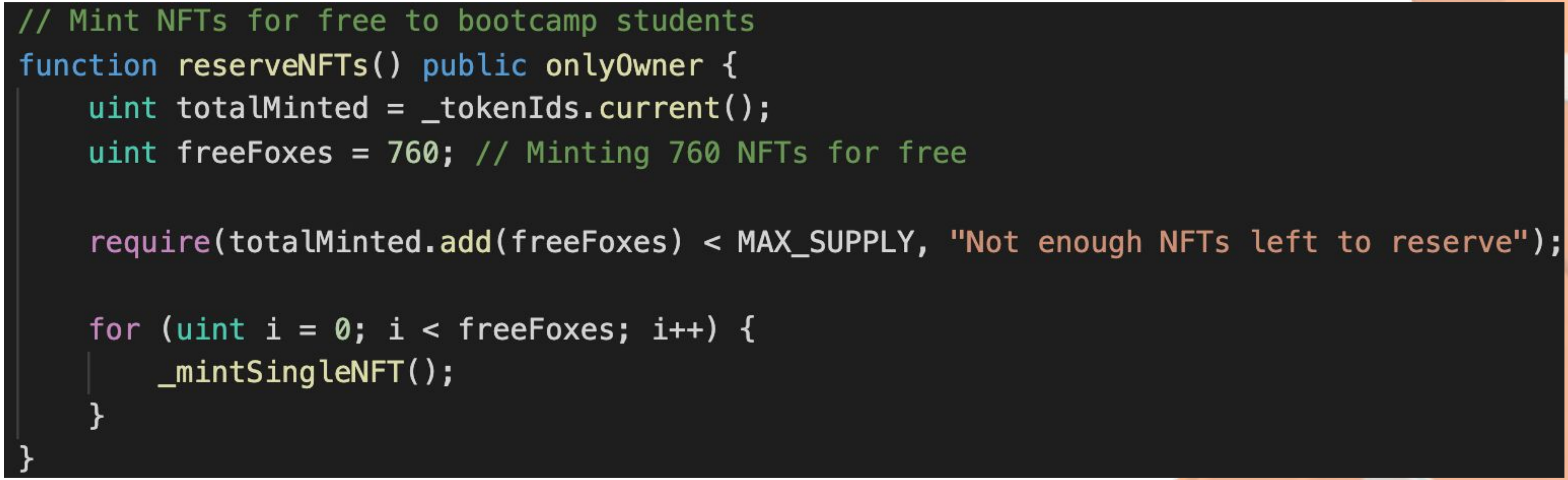


```
contract NFTCollectible is ERC721Enumerable, Ownable {
    using SafeMath for uint256;
    using Counters for Counters.Counter;

    Counters.Counter private _tokenIds;

    uint public constant MAX_SUPPLY = 760; // Max. number of NFTs that can be minted
    uint public constant MAX_PER_MINT = 1; // Only one NFT per mint
    string public baseTokenURI;

    // Constructor to set the name and symbol of NFT collection, and baseURI
    constructor(string memory baseURI) ERC721("Origami Foxes", "OF") {
        setBaseURI(baseURI);
    }
```



reserveNFT function to mint NFTs for free

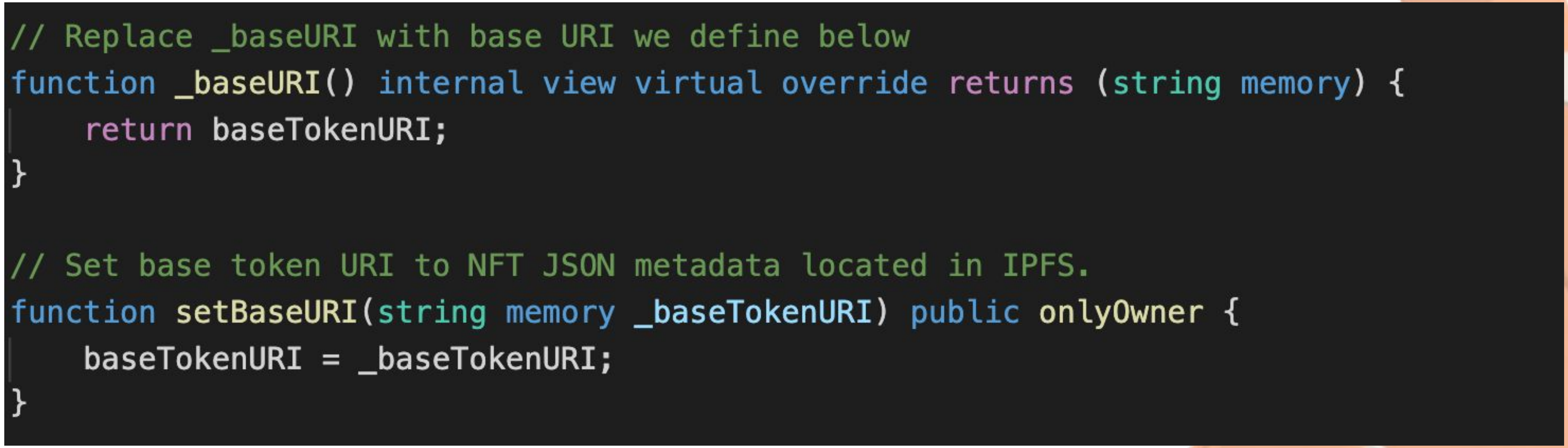
The total supply of NFTs (760) are reserved for free minting



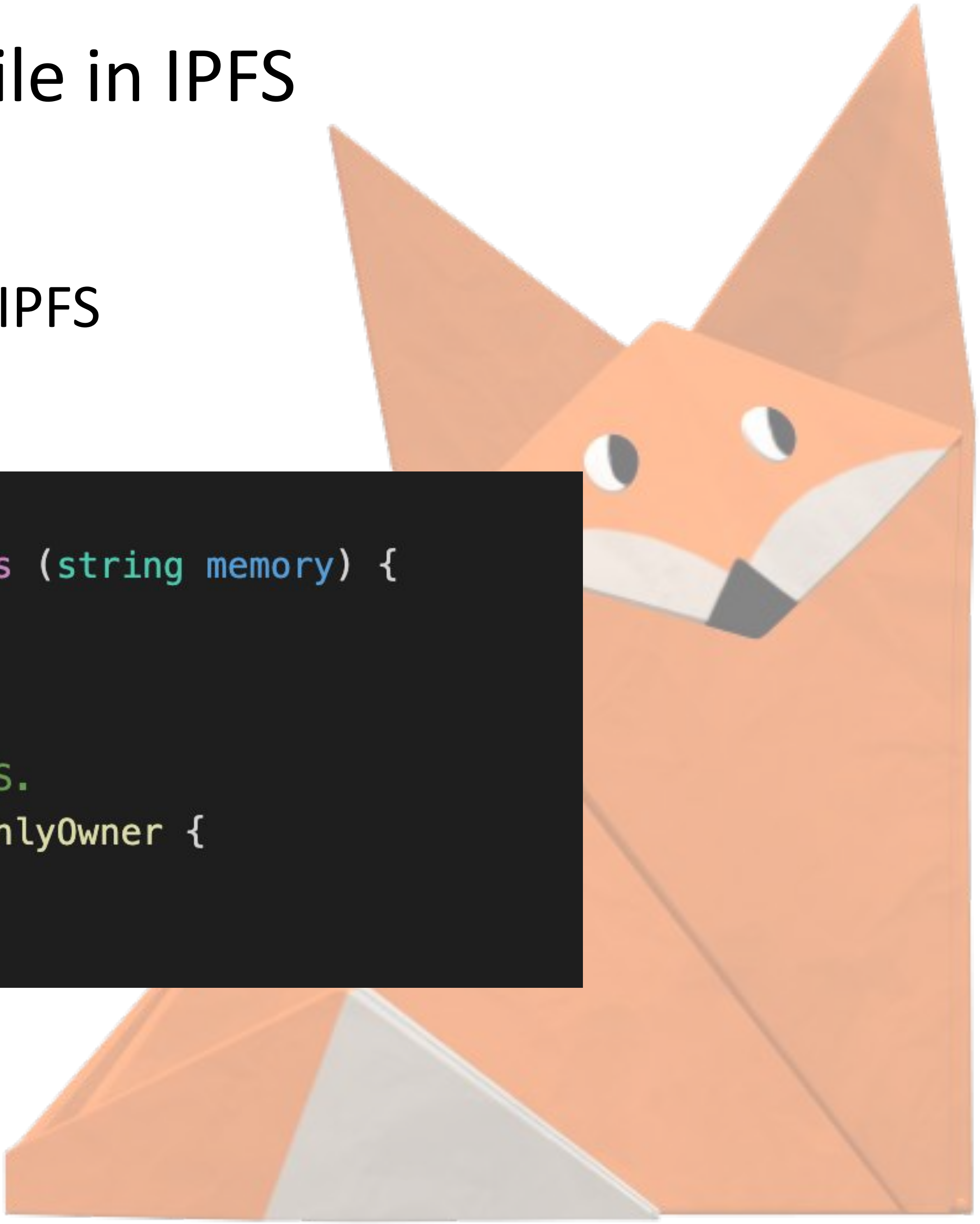
```
// Mint NFTs for free to bootcamp students
function reserveNFTs() public onlyOwner {
    uint totalMinted = _tokenIds.current();
    uint freeFoxes = 760; // Minting 760 NFTs for free

    require(totalMinted.add(freeFoxes) < MAX_SUPPLY, "Not enough NFTs left to reserve");

    for (uint i = 0; i < freeFoxes; i++) {
        _mintSingleNFT();
    }
}
```

setBaseURI function to specify JSON file in IPFS

Allows changing the URI location of JSON file in IPFS

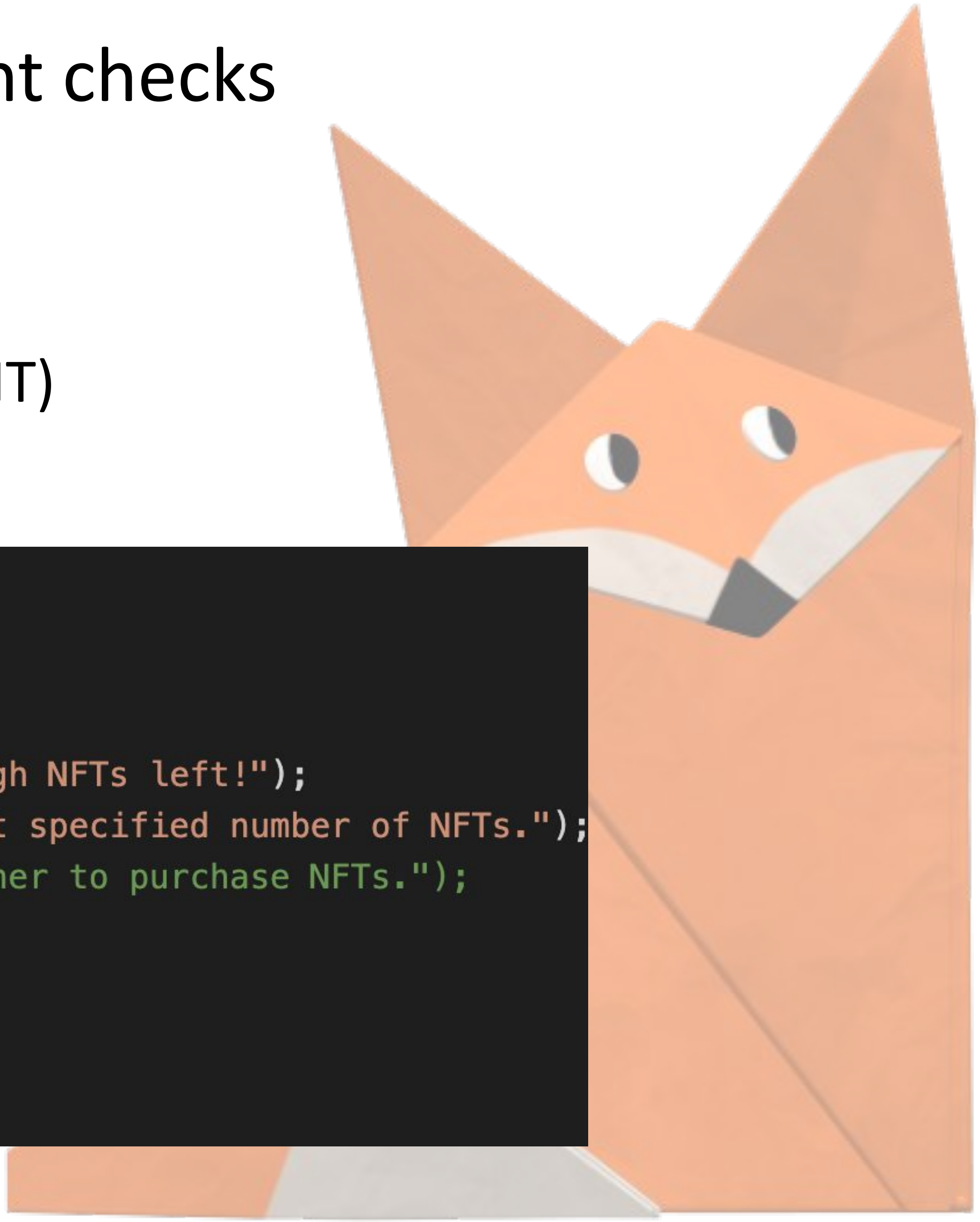


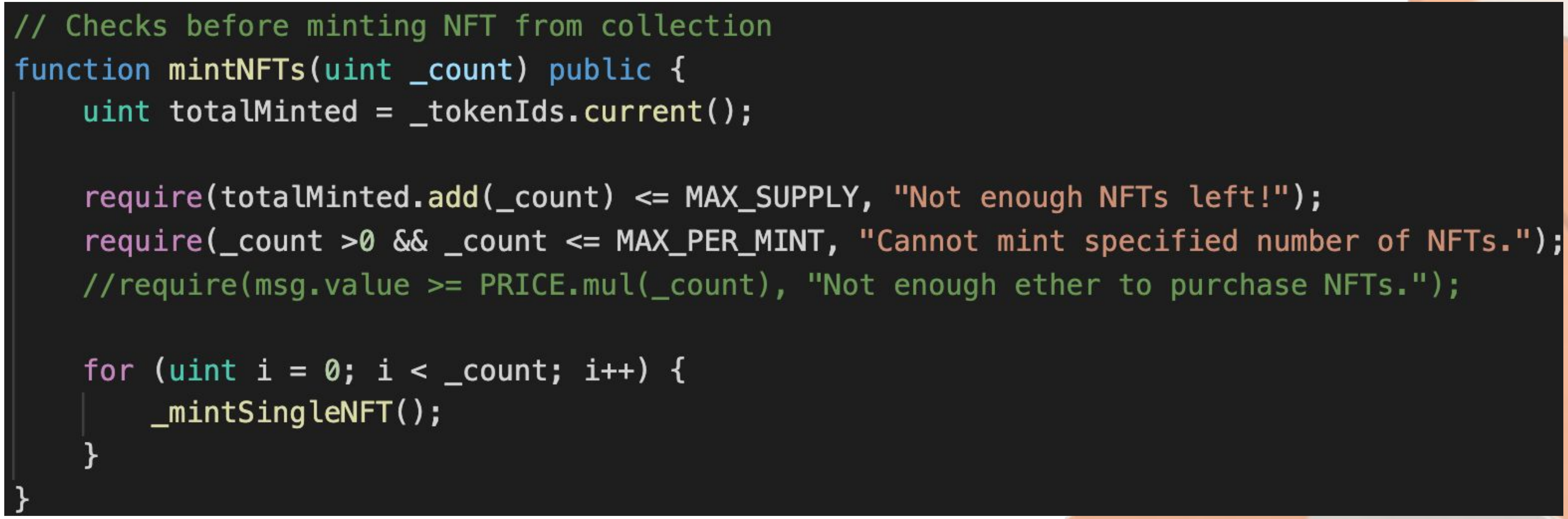
```
// Replace _baseURI with base URI we define below
function _baseURI() internal view virtual override returns (string memory) {
    return baseTokenURI;
}

// Set base token URI to NFT JSON metadata located in IPFS.
function setBaseURI(string memory _baseTokenURI) public onlyOwner {
    baseTokenURI = _baseTokenURI;
}
```



mintNFTs function to perform pre-mint checks

- Checks there are enough NFTs left to mint.
- Checks only 1 NFT is minted (MAX_PER_MINT)

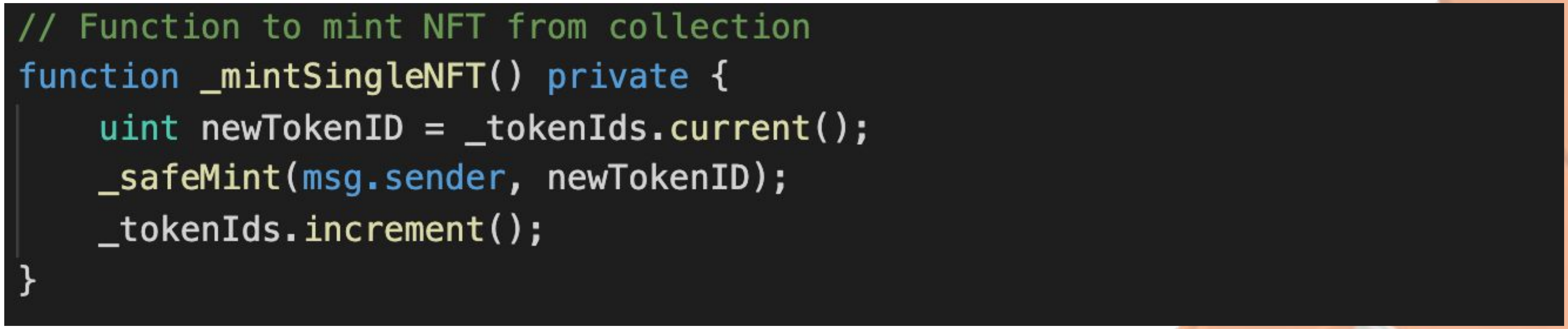




```
// Checks before minting NFT from collection
function mintNFTs(uint _count) public {
    uint totalMinted = _tokenIds.current();

    require(totalMinted.add(_count) <= MAX_SUPPLY, "Not enough NFTs left!");
    require(_count >0 && _count <= MAX_PER_MINT, "Cannot mint specified number of NFTs.");
    //require(msg.value >= PRICE.mul(_count), "Not enough ether to purchase NFTs.");

    for (uint i = 0; i < _count; i++) {
        _mintSingleNFT();
    }
}
```

mintSingleNFT function to mint NFT

1. Get current the token ID not yet minted.
2. Assign NFT ID to the address that called the function
3. Increment tokenID's counter by 1



```
// Function to mint NFT from collection
function _mintSingleNFT() private {
    uint newTokenID = _tokenIds.current();
    _safeMint(msg.sender, newTokenID);
    _tokenIds.increment();
}
```

Uploading generated images and creating metadata for NFT marketplaces

Upload Images to a hosting solution

- We needed to decide where to store our generated images that would later be minted into NFTs via our smart contract:
  - Store on a blockchain
  - Store in a standard public cloud-hosted environment (AWS/Google Cloud/Microsoft Azure)
  - Store on a decentralized platform such as IPFS/Arweave
- We found problems with the first two options:
  - Blockchain storage is expensive. Our ~50mb folder of images would cost roughly $75K to store on Ethereum
  - Public clouds are not decentralized, creating principal risk
- We settled on IPFS as it's decentralized, immutable, and cheap. Every image stored on the network has a unique address. If the image changes, so does its url address, making it tamper-resistant. From here, all we have to store on the blockchain is metadata *about* the NFT, reducing costs.

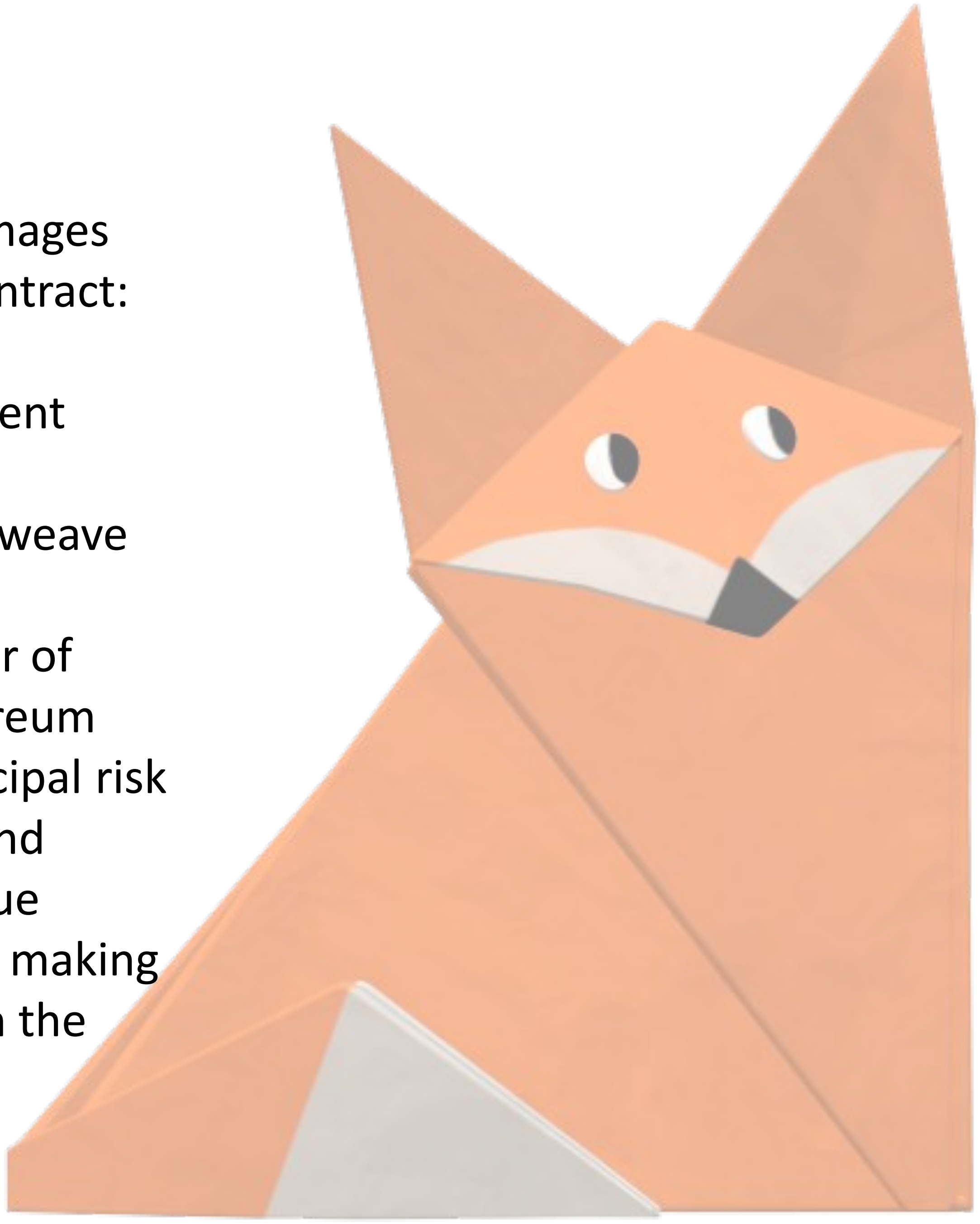
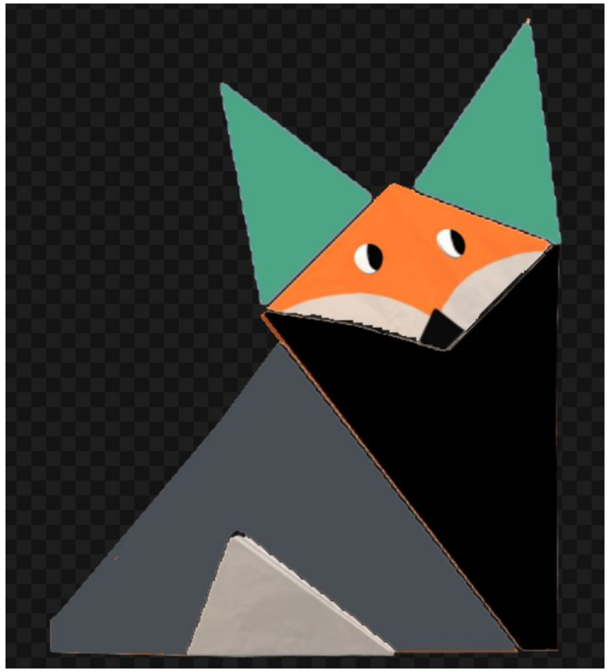
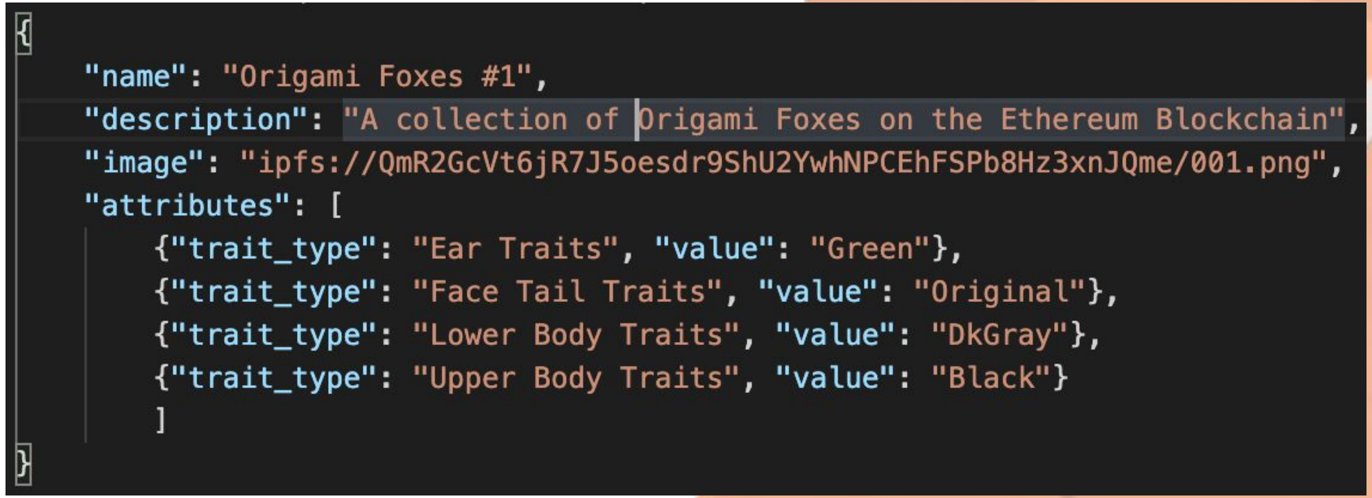


Image vs. JSON Metadata file for a sample NFT

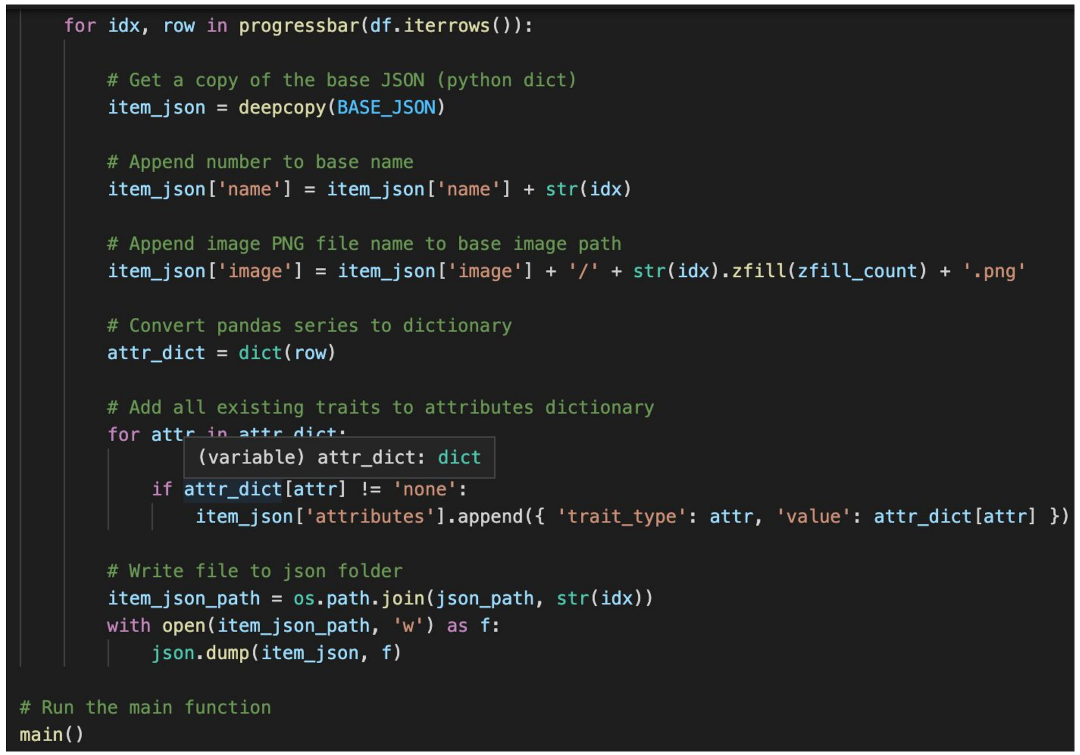




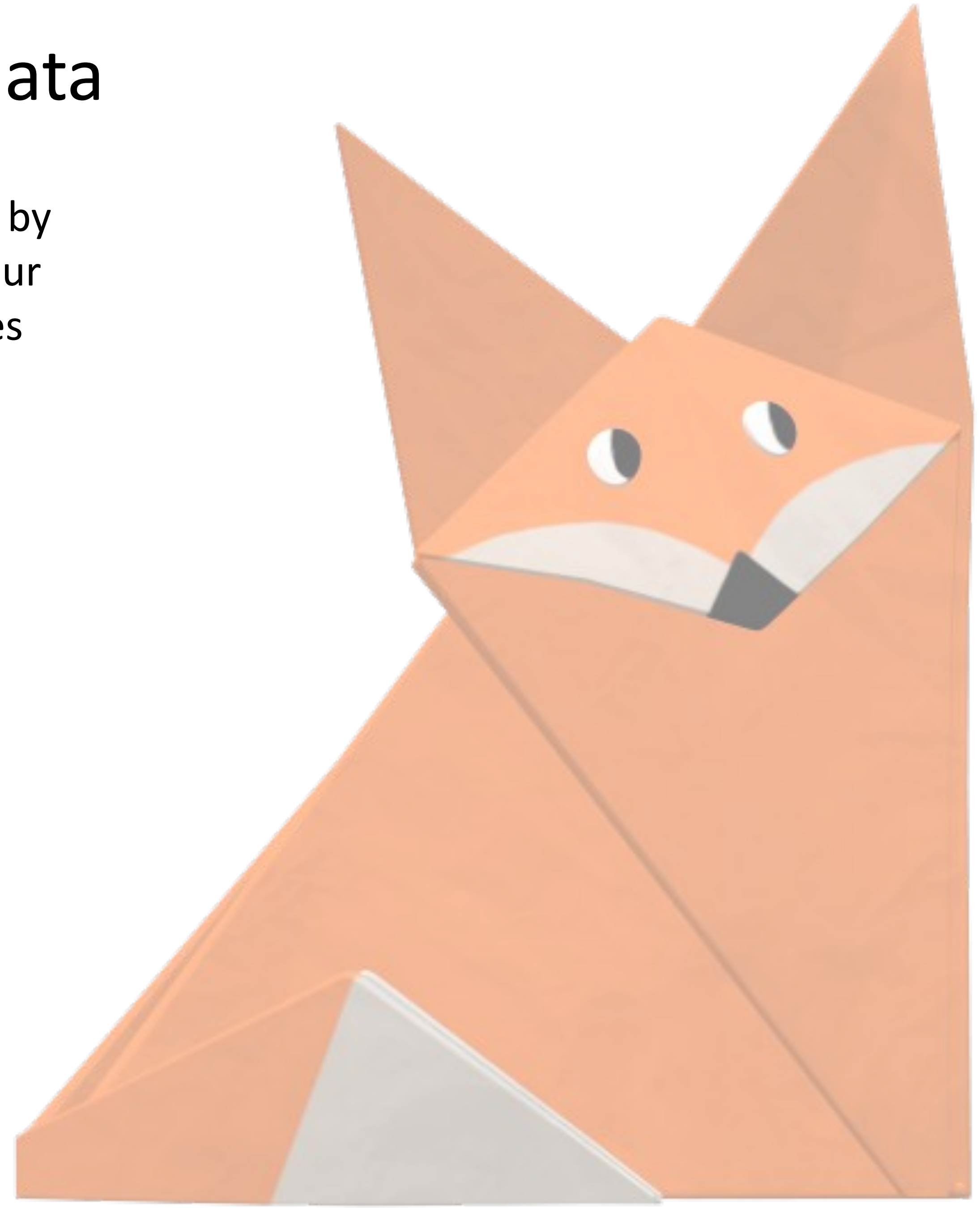
```
{
    "name": "Origami Foxes #1",
    "description": "A collection of Origami Foxes on the Ethereum Blockchain",
    "image": "ipfs://QmR2GcVt6jR7J5oesdr9ShU2YwhNPCEhFSPb8Hz3xnJQme/001.png",
    "attributes": [
        {"trait_type": "Ear Traits", "value": "Green"},
        {"trait_type": "Face Tail Traits", "value": "Original"},
        {"trait_type": "Lower Body Traits", "value": "DkGray"},
        {"trait_type": "Upper Body Traits", "value": "Black"}
        ]
}
```

Generate Compliant JSON NFT Metadata

The JSON metadata file must be compliant and understandable by NFT marketplaces. We created a metadata.py file that links to our folder of images on IPFS using a generated URL (CID) and creates metadata that meets NFT marketplace standards.



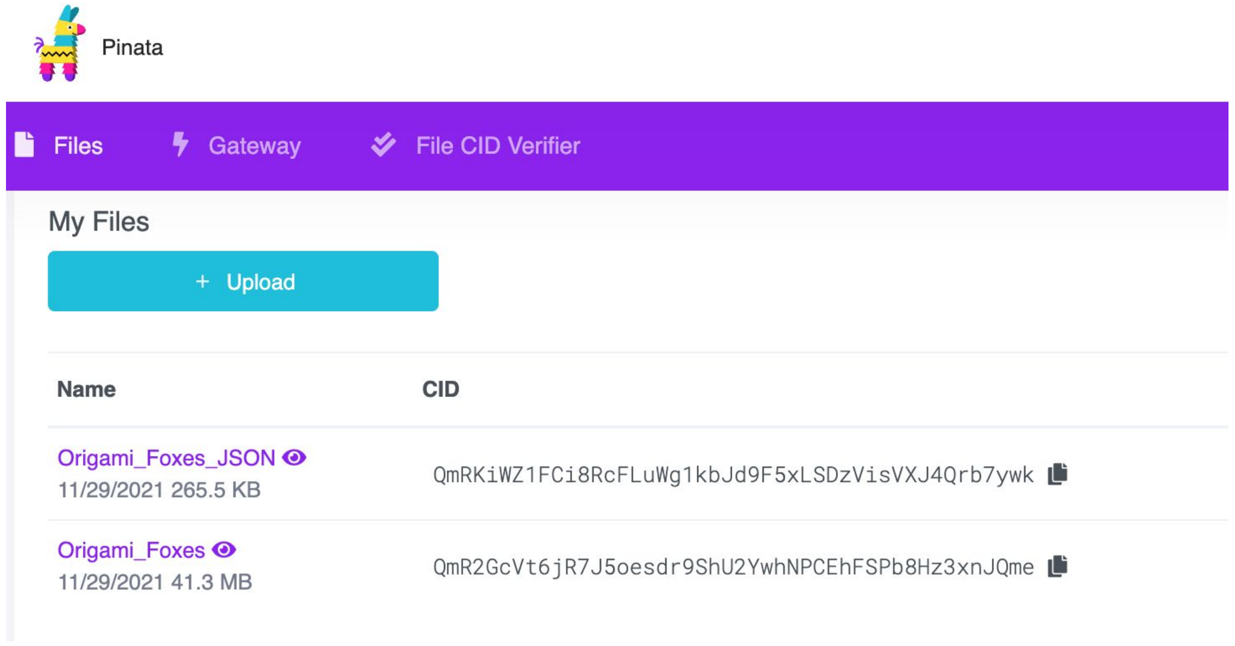
```python
    for idx, row in progressbar(df.iterrows()):

        # Get a copy of the base JSON (python dict)
        item_json = deepcopy(BASE_JSON)

        # Append number to base name
        item_json['name'] = item_json['name'] + str(idx)

        # Append image PNG file name to base image path
        item_json['image'] = item_json['image'] + '/' + str(idx).zfill(zfill_count) + '.png'

        # Convert pandas series to dictionary
        attr_dict = dict(row)

        # Add all existing traits to attributes dictionary
        for attr in attr_dict:
            if attr_dict[attr] != 'none':
                item_json['attributes'].append({ 'trait_type': attr, 'value': attr_dict[attr] })

        # Write file to json folder
        item_json_path = os.path.join(json_path, str(idx))
        with open(item_json_path, 'w') as f:
            json.dump(item_json, f)

# Run the main function
main()
```



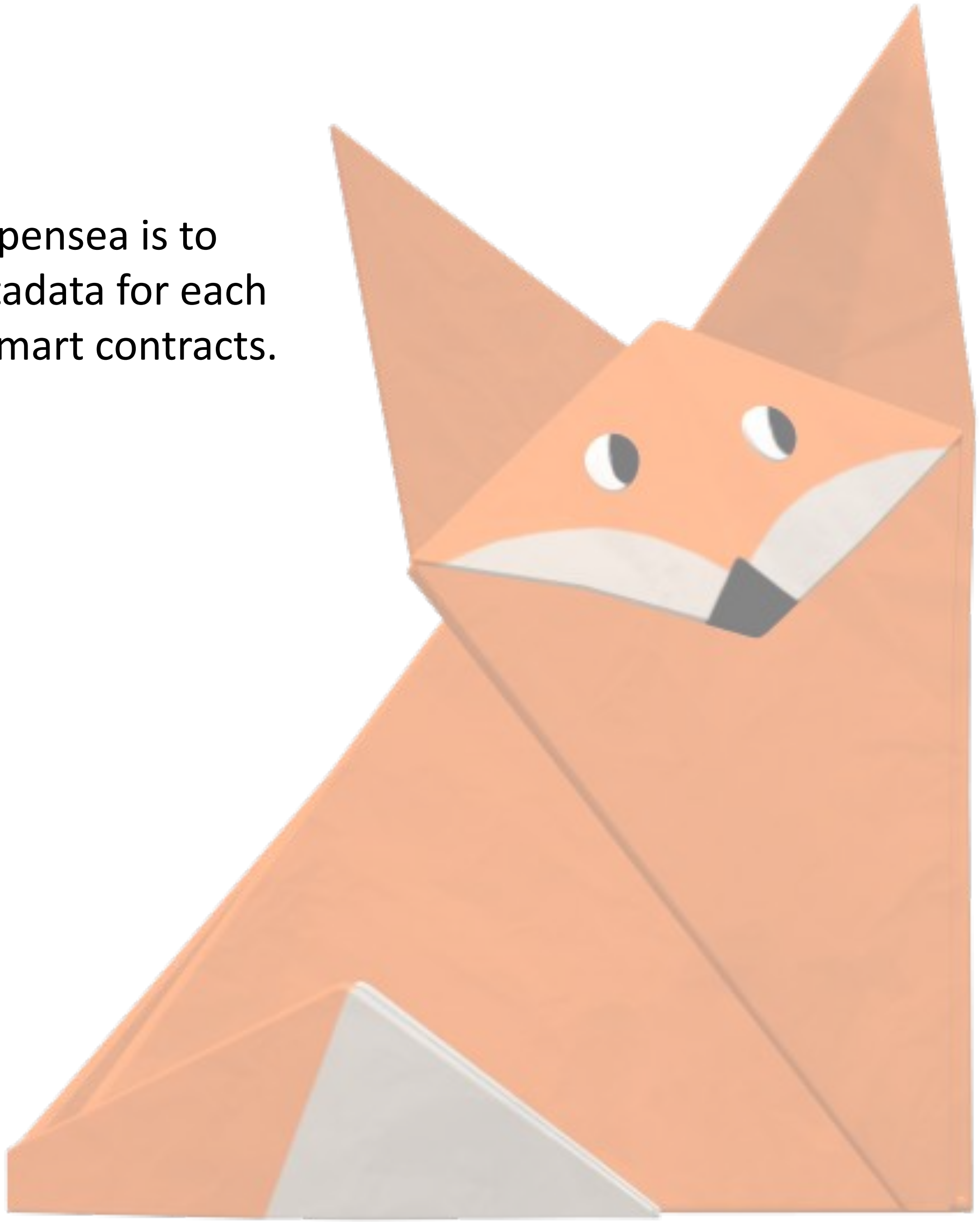
Upload Metadata Folder to IPFS

The last step before deploying on an NFT marketplace such as Opensea is to upload our metadata folder to IPFS. This folder contains the metadata for each one of our generated NFTs to IPFS. This is what gets utilized by smart contracts.

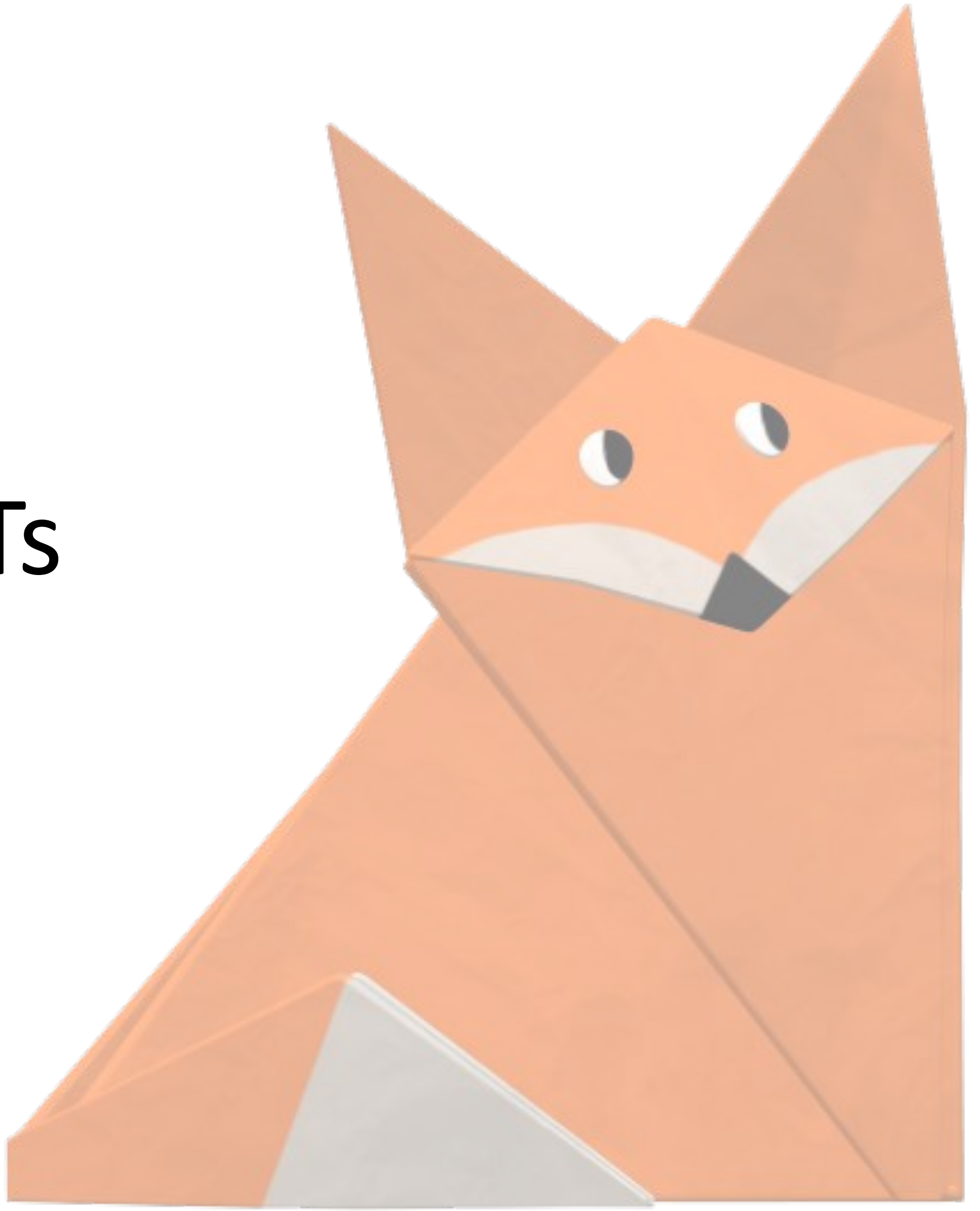
We use Pinata for our IPFS uploads.



Name	CID
Origami_Foxes_JSON 11/29/2021 265.5 KB	QmRKiWZ1FCi8RcFLuWg1kbJd9F5xLSDzVisVXJ4Qrb7ywk
Origami_Foxes 11/29/2021 41.3 MB	QmR2GcVt6jR7J5oesdr9ShU2YwhNPCEhFSPb8Hz3xnJQme



Deploying OrigamiFox NFTs to OpenSea’s Testnet



Conclusions and next steps