Documentation

July 20, 2025

0.1 Imported Modules Documentation

0.1.1 `os` Module Functions:

- `remove(path: str) -> None`
  Remove (delete) a file at the specified path.
- `sep: str`
  The character used by the operating system to separate pathname components (e.g., '/' on Unix, '\' on Windows).

0.1.2 `secrets` Module Functions:

- `choice(sequence) -> item`
  Return a randomly chosen element from a non-empty sequence.
- `randbits(k: int) -> int`
  Return an integer with k random bits (using a cryptographically secure random number generator).

0.1.3 `gmpy2` Module Function:

- `is_prime(n, /) -> bool` (imported as `isprime`)
  Determine if a number is prime. Returns True if n is a probable prime, False otherwise.

0.1.4 `sys` Module:

- Provides access to system-specific parameters and functions.
- Used here primarily for `sys.set_int_max_str_digits()` to handle very large integers.

0.1.5 `cv2` Module (OpenCV):

- Computer vision library used for camera operations.
- Provides functions for image capture and processing.

0.1.6 `time` Module Function:

- `sleep(seconds: float) -> None`
  Suspend execution for the given number of seconds.

[1]:
```
from os import remove,sep
from secrets import choice,randbits
from gmpy2 import is_prime as isprime
import sys
import cv2
from time import sleep
```

1 `captureImage()` Function Documentation

1.1 Overview

Captures one or more images from a connected camera and saves them as JPEG files to a specified directory.

1.2 Parameters

- `path` (str): Directory path where images will be saved
- `total_images` (int, optional): Number of images to capture (default: 1)
- `camera` (int, optional): Camera device index (default: 0 for primary camera)

1.3 Behavior

1. Initializes the camera using OpenCV's `VideoCapture`
2. Adds a small random delay to ensure proper camera initialization
3. Captures the specified number of frames:
   - Creates sequential filenames (`0.jpg`, `1.jpg`, etc.)
   - Saves each frame as a JPEG in the target directory
4. Properly releases camera resources when done

1.4 Returns

List of strings containing full paths to all saved images (e.g., `['/path/0.jpg', '/path/1.jpg']`)

1.5 Error Handling

- Raises exception if:
  – Camera cannot be opened
  – Frame cannot be captured (stream ended)

1.6 Example Usage

"'python saved_images = captureImage('/photos', total_images=3) # Returns: ['/photos/0.jpg', '/photos/1.jpg', '/photos/2.jpg']

[2]:
```python
def captureImage(path,total_images=1,camera=0):
    image_names = []
    cap = cv2.VideoCapture(camera)

    if not cap.isOpened():
        raise Exception("Cannot open camera")

    ret,frame = cap.read()
    sleep((1 + 1/(randbits(16)+0.000001))) #Waits for the cam to open.
    for i in range(total_images):
        if not ret:
            raise Exception("Can't receive frame (stream end?). Exiting ...")
        else:
            ret,frame = cap.read()
            # Save the captured image
            image_name = path + sep + str(i) + ".jpg"
            image_names.append(image_name)
            cv2.imwrite(image_name, frame)
    cap.release()
    cv2.destroyAllWindows()
    return image_names
```

1.7 NoiseRandom Class Documentation

`NoiseRandom` - A class for generating random numbers using camera noise as an entropy source.

1.7.1 Attributes:

- `path` (str): Directory path where temporary images will be stored
- `images` (list): List of captured image paths
- `cameras` (list): List of camera indices to use for capture
- `strength` (int): Number of images to capture per camera (minimum 1)

1.7.2 Methods:

`__init__(path: str, strength=1, cameras=[0])` Initialize the NoiseRandom generator.

Parameters: - `path` (str): Directory path for temporary image storage - **`strength`** (int, optional): Number of images per camera. Defaults to 1. - **`cameras`** (list, optional): List of camera indices. Defaults to [0].

`randomInt(get_bytes=False) -> int | bytes` Generate a random integer from camera noise.

Parameters: - `get_bytes` (bool, optional): Return raw bytes instead of int. Defaults to False.

Returns: - `int | bytes`: Random number or its byte representation

`randomBytes(total_bytes: int, get_bytes=False) -> int | bytes` Generate random bytes of specified length.

Parameters: - `total_bytes` (int): Number of bytes to generate - `get_bytes` (bool, optional): Return bytes instead of int. Defaults to False.

Returns: - `int | bytes`: Random number or its byte representation

Raises: - `ValueError`: If total_bytes is 0 or negative

`randomPrime(total_bytes: int) -> int` Generate a random prime number of specified byte length.

Parameters: - `total_bytes` (int): Byte length of the prime number

Returns: - `int`: A probable prime number

Raises: - `ValueError`: If total_bytes is 0 or negative

Convenience Methods:

- `random1024() -> int`: Generate a random 1024-bit number
- `random2048() -> int`: Generate a random 2048-bit number
- `random4096() -> int`: Generate a random 4096-bit number
- `randomPrime1024() -> int`: Generate a random 1024-bit prime number
- `randomPrime2048() -> int`: Generate a random 2048-bit prime number
- `randomPrime4096() -> int`: Generate a random 4096-bit prime number

Private Methods:

- `__deleteImages() -> None`: Delete all captured temporary images and clear the images list
- `__captureImages() -> None`: Capture images from a random specified camera
- `__scramble(data) -> bytes`: Scramble the input data using a random swapping algorithm

[ ]:
```python
class NoiseRandom():

    def __init__(self,path:str,strength=1,cameras=[0]):
        self.path = path
        self.images = []
        self.cameras = cameras
        self.strength = strength
        if(self.strength<1):
            self.strength = 1

    def randomInt(self,get_bytes=False)->int|bytes:
        self.__captureImages()
        with open(choice(self.images), "rb") as f:
```

```
            data = f.read()
            f.close()
        self.__deleteImages()
        starting_image_index = data.find(b"\xFF\xDA")
        ending_image_index = data.find(b"\xFF\xD9")
        data = self.__scramble(data[starting_image_index+1:ending_image_index])
        if(get_bytes):
            return data
        sys.set_int_max_str_digits(len(data) * 3)
        return  int.from_bytes(data,"big",signed=False)

    def randomBytes(self,total_bytes:int,get_bytes=False) -> int|bytes:
        if(total_bytes<=0):
            raise ValueError("The value of total_bytes can't be 0 or less.")
        random_pool = self.randomInt(True)
        selected_bytes = [choice(random_pool) & 0xFF for _ in␣
↪range(total_bytes)]
        selected_bytes[0] |= (1<<7)
        if(get_bytes):
            return bytes(selected_bytes)
        return int.from_bytes(selected_bytes,"big",signed=False)

    def randomPrime(self,total_bytes:int) -> int:
        if(total_bytes<=0):
            raise ValueError("The value of total_bytes can't be 0 or less.")
        random_pool = self.randomInt(True)
        selected_bytes = [choice(random_pool) & 0xFF for _ in␣
↪range(total_bytes)]
        selected_bytes[0] |= (1<<7)
        prime_number = int.from_bytes(selected_bytes,"big",signed=False)
        while(not isprime(prime_number)):
            selected_bytes = [choice(random_pool) & 0xFF for _ in␣
↪range(total_bytes)]
            selected_bytes[0] |= (1<<7)
            prime_number = int.from_bytes(selected_bytes,"big",signed=False)
        return prime_number

    def random1024(self)->int:
        return self.randomBytes(1024//8)

    def random2048(self)->int:
        return self.randomBytes(2048//8)

    def random4096(self)->int:
        return self.randomBytes(4096//8)

    def randomPrime1024(self):
```

```
        return self.randomPrime(1024//8)

    def randomPrime2048(self):
        return self.randomPrime(2048//8)

    def randomPrime4096(self):
        return self.randomPrime(4096//8)

    def __deleteImages(self) -> None:
        for image_path in self.images:
            remove(image_path)
        self.images.clear()

    def __captureImages(self) -> None:
        self.images = self.images + captureImage(self.path,self.
↪strength,camera=choice(self.cameras))

    def __scramble(self,data):
        byte_list = list(data)
        data_size = len(byte_list)
        scrambled_data = b""
        for i in range(data_size*3):
            p1 = randbits(4*8) % data_size
            p2 = randbits(4*8) % data_size

            byte_list[p1], byte_list[p2] = byte_list[p2], byte_list[p1]

        for byte in byte_list:
            scrambled_data += byte.to_bytes()

        return scrambled_data
```