Documentation by Sejyoti Chakraborty
Brief documentation explaining the code:

```
import os
import sys

# Constants
TRIVIAL_COMPRESSION_MODE = "0"
INVALID_PAIR = b"\xe2\x9c\xb8\xe2\x9d\x8b"
```

The code imports the 'os' and 'sys' modules.
Two constants are defined:
'TRIVIAL_COMPRESSION_MODE' is a string constant that is used to determine if the program should use the trivial compression mode or not. If the environment variable '❯ ❙ ❊{ ▮ ❙ ■❱ ■❄ ▲{ ■▼P▲❊▼❊◆▮ ❄ ▮■❖◆' has the value '0', the program will use the trivial compression mode.
'INVALID_PAIR' is a byte string constant that is used to represent an invalid or incomplete pair. It is used in the decode_pairs() function to replace invalid or incomplete pairs with this byte string in the decoded output.

```
# Functions
def decode_pairs(pairs):
    decoded_data = bytearray()
    for p, q in pairs:
        if p == 0:
            decoded_data.append(q)
        else:
            start = len(decoded_data) - p
            end = start + q
            if start < 0 or end > len(decoded_data):
                decoded_data.extend(INVALID_PAIR)
            else:
                decoded_data.extend(decoded_data[start:end])
    return decoded_data
```

The 'decode_pairs()' function takes a sequence of pairs as input and returns the decoded data as a bytearray object.
For each pair in the input sequence, the function checks if 'p' is equal to zero. If 'p' is zero, it appends 'q' to the 'decoded_data' bytearray. If 'p' is greater than zero, it calculates the start and end positions based on the previous 'p' characters appended to decoded_data, and takes the first 'q' characters starting from the start position. If the start position is less than zero or the end position is greater than the length of 'decoded_data', it appends the 'INVALID_PAIR' byte string to 'decoded_data'.

The function returns the 'decoded_data' bytearray.

```
def encode_pairs(decoded_data):
	pairs = []
	start = 0
	while start < len(decoded_data):
		for p in range(min(start, 255), -1, -1):
			end = start + 1
			while end < len(decoded_data) and decoded_data[start:end] ==
decoded_data[end:end+p]:
				end += 1
			pairs.append((len(decoded_data)-start, end-start))
			start = end
	return pairs
```

The 'encode_pairs()' function takes a bytearray object as input and returns the encoded pairs as a list of tuples.
The function iterates over the input bytearray and for each byte 'b' at position i, it looks backwards from 'i' to find the longest sequence of bytes that is identical to the sequence of bytes starting at position 'i'. This is done by iterating over a range of 'p' values that starts from the minimum of start and 255 (to limit the maximum p value to 255), and ends at -1 (to include the case where p is zero). For each 'p' value, it checks if the sequence of bytes starting at position 'i' is equal to the sequence of bytes starting at position i+p. If they are equal, it increases end by 1 and continues to check the next byte until the sequence is.