After reading the description, hints, and attachments we deduce a few things:
1) The probability of a single qubit rotating an angle theta in the Y - axis is measured as sin^2(theta/2).
2) Except the rotation in ry the rest of the code is majorly irrelevant as rotation in the z axis does not affect the probability, so we can completely ignore the phase parameters.
3) We need to reverse the probability and recover amplitude as hinted by the recovered_candidates.py script.
4) The flag starts with "DJSISACA{" and ends with "}".

Now with these in mind we can see exactly what needs to be done.

STEP 1 (REVERSING THE ANGLE TO OBTAIN a^b)
Since we have `amp = ((a ^ b) + 10) / 100.0` whose probability of occurring as seen by `qc.ry(amp * np.pi, 0)` is `sin^2(theta/2)` where theta is `amp * np.pi.`

We can find the value of a^b where a is `flag[i]` and b is `flag[i+1]` by running the script:

```
import numpy as np

outputs = [0.15, 0.278, 0.274, 0.274, 0.184, 0.046, 0.038, 0.808,
0.524, 0.438,
           0.816, 0.072, 0.296, 0.78, 1.0, 1.0, 0.648, 0.834,
0.982, 0.078,
           0.932, 0.546, 0.716, 0.694, 0.882, 0.994, 0.956, 0.986,
0.408, 0.746]

vals = []
for p in outputs:
    theta = 2 * np.arcsin(np.sqrt(p))
    raw = (theta / np.pi) * 100 - 10
    val = int(round(raw))
    vals.append(val)

print("Recovered a^b:", vals)
```

which gives us:

```
Recovered a^b: [15, 25, 25, 25, 18, 4, 2, 61, 42, 36, 62, 7, 27,
59, 90, 90, 50, 63, 81, 8, 73, 43, 54, 53, 68, 85, 77, 82, 34, 56]
```

STEP 2 (RECOVERING THE REAL a^b)
Now we can input these candidates in another script provided "recovered_candidates.py" and the candidates we recover from that are

```
[14, 25, 26, 26, 18, 2, 2, 58, 42, 35, 60, 9, 24, 60, 87, 90, 49,
61, 81, 7, 107, 45, 55, 51, 69, 97, 103, 80, 33, 57].
```

We have now successfully recovered the real a^b values and can run a simple script exploiting the key property of XOR in binary math which is If you have a^b=v where v is a known value then knowing one of a or b, you can always recover the other using b = a^v or a = b^v.

STEP 3 (RECOVERING THE FLAG FROM a^b)
We know the flag starts with "DJSISACA{" and ends with "}" and so the first letter of the flag is D and can run this script:

```
vals = [14, 25, 26, 26, 18, 2, 2, 58, 42, 35, 60, 9, 24, 60, 87,
90, 49, 61, 81, 7, 107, 45, 55, 51, 69, 97, 103, 80, 33, 57]

flag = [68]  # decimal for D
for v in vals:
    nxt = int(flag[-1]) ^ int(v)
    flag.append(nxt)

print(flag) # prints the decimal value of the ASCII
print(f"Recovered flag: {''.join(chr(int(x)) for x in flag)}") #
prints ASCII
```

Which finally gives us:

```
[68, 74, 83, 73, 83, 65, 67, 65, 123, 81, 114, 78, 71, 95, 99, 52,
110, 95, 98, 51, 52, 95, 114, 69, 118, 51, 82, 53, 101, 68, 125]
Recovered flag: DJSISACA{QrNG_c4n_b34_rEv3R5eD}
```