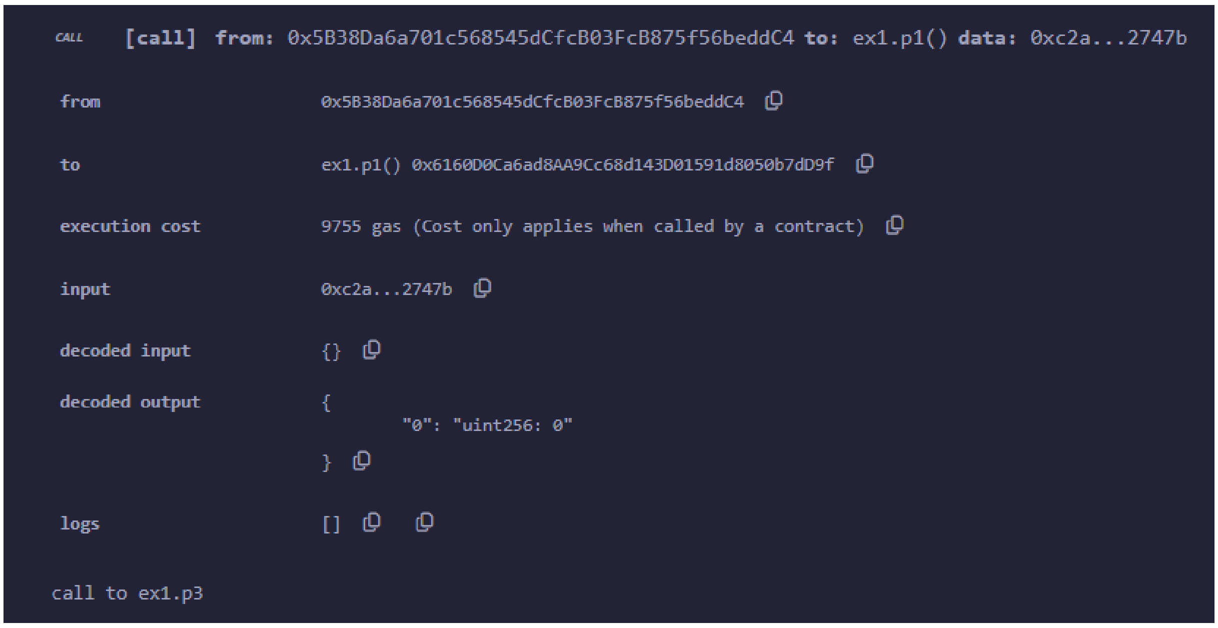
INTRO TO BLOCKCHAIN – LAB 9

1.1 – Contract 1

1. Here are the results that I had obtained for “function 1”,”function 2” and “function 3”…
    - Function 1



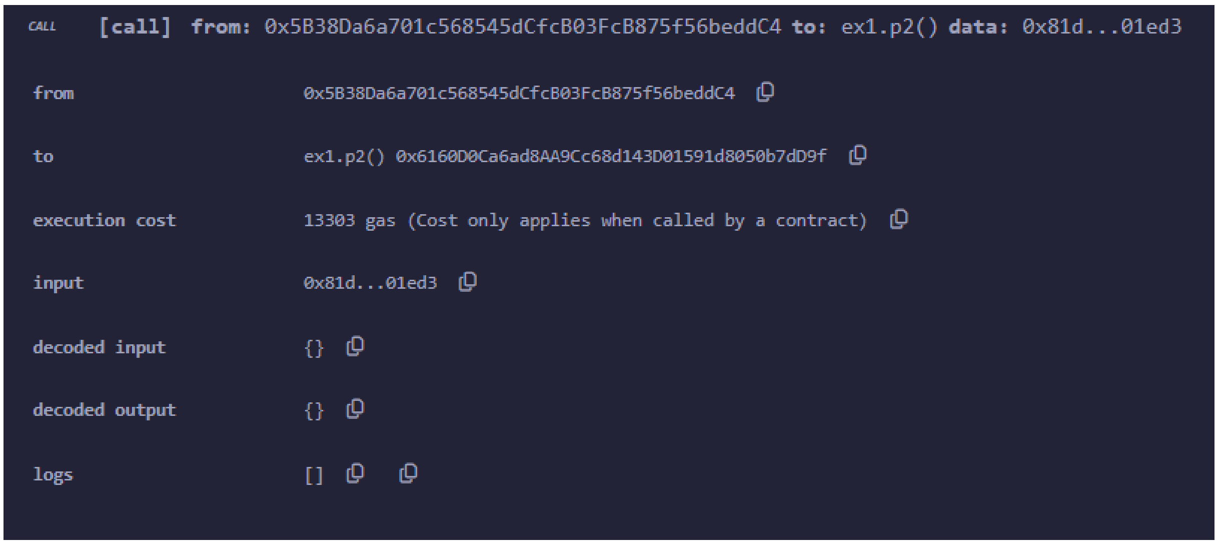
```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ex1.p1() data: 0xc2a...2747b

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                ex1.p1() 0x6160D0Ca6ad8AA9Cc68d143D01591d8050b7dD9f
execution cost    9755 gas (Cost only applies when called by a contract)
input             0xc2a...2747b
decoded input     {}
decoded output    {
                         "0": "uint256: 0"
                  }
logs              []

call to ex1.p3
```

```
Memory UB for p1(): 15


Opcodes UB for p1(): 2381+6453*nat(arr/2+1/2)
```

Explanation:

This function iterates over the array arr and sums up the even-indexed elements. The gas consumption upper bound seems reasonable, considering it's a straightforward loop with constant memory usage. The gas consumption increases linearly with the size of the array arr.

- Function 2



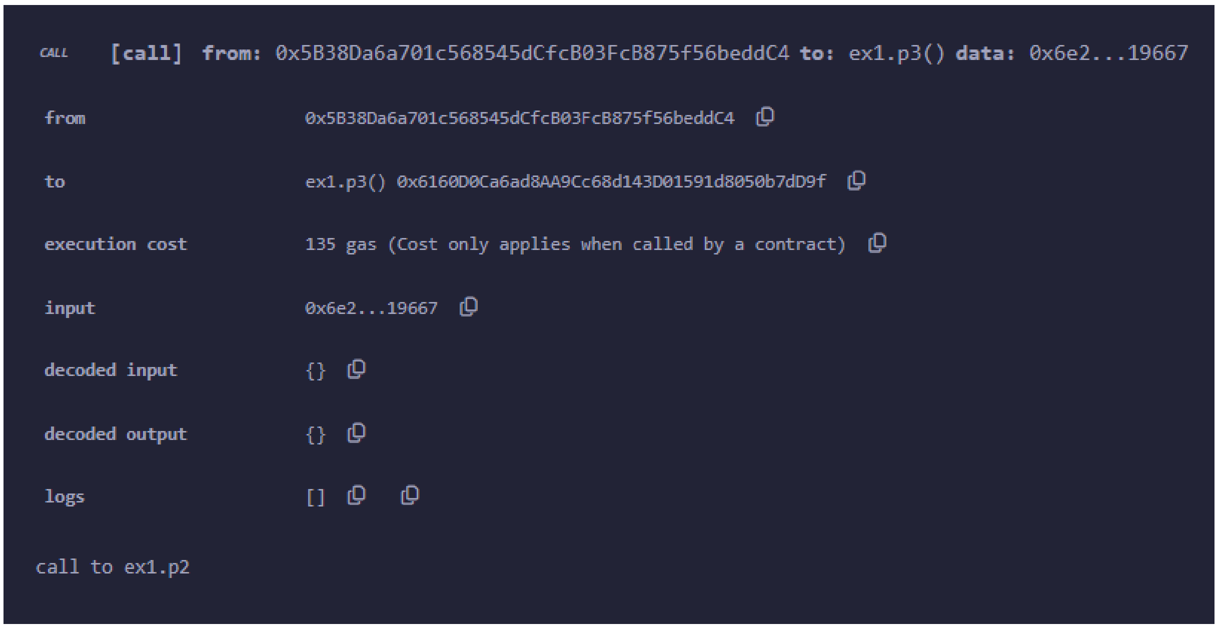
```
CALL    [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ex1.p2() data: 0x81d...01ed3

from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                  ex1.p2() 0x6160D0Ca6ad8AA9Cc68d143D01591d8050b7dD9f

execution cost      13303 gas (Cost only applies when called by a contract)

input               0x81d...01ed3

decoded input       {}

decoded output      {}

logs                []
```

```
Memory UB for p2():
3*max([nat(arr)+4+2])+pow(max([nat(arr)+4+2]),2)/512



Opcodes UB for p2(): 6673+2150*nat(arr-1/32)
```

Explanation:

This function simply assigns the storage array arr to a memory array local. The gas consumption upper bound from GASTAP seems to take into account the size of the array arr and the operations involved in the assignment. It shows a linear increase in gas consumption with the size of the array arr.

- Function 3



```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: ex1.p3() data: 0x6e2...19667

from               0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                 ex1.p3() 0x6160D0Ca6ad8AA9Cc68d143D01591d8050b7dD9f
execution cost     135 gas (Cost only applies when called by a contract)
input              0x6e2...19667
decoded input      {}
decoded output     {}
logs               []

call to ex1.p2
```

```
Memory UB for p3(): 9



Opcodes UB for p3(): 126
```

Explanation:

This function assigns the storage array arr to a storage array local. Since storage pointers are used, the gas consumption is minimal, as evident from both Remix and GASTAP results.

2. Yes, GASTAP will be really useful tool because it is showing the Upper Bounds and makes us change how we design our contracts, which will lead us to a more effective design

1.2 - Contract 2

1. Here is the result I obtained for “power” function of the ex2

- Since at the assignment only “GASTAP” had been mentioned to analyze the code, I hadn’t used remix

```
Memory UB for powers(): 9



Opcodes UB for powers(): 2261+26471*nat(arr)
```

The powers function's gas cost per loop iteration scales with the size of array arr, specifically by 26,471 gas per element. With an initial cost of 2,261 gas, the function's overall gas consumption grows linearly with the array size.

2. This is the optimized code which reduces Access

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.5.0;

contract ex2 {
  uint[] arr = new uint[](5);

  function powers() external {
    uint len = arr.length; // Cache array length locally

    uint[] memory local = arr;

    for (uint i = 0; i < len; i++) {
      local[i] = i**i;
    }
  }
}
```

- I used another array called “local” which is “memory”

```
Opcodes UB for powers(): 8843+2150*nat(arr-1/32)+134*nat(arr)
```

3. The reason behind this efficency is, “local” uses “memory” but “arr” uses “storage”… when we use “memory” it costs less to reach a data….