Assignment 3

Q.1 Write a function to get the block and its transactions.

Solution: Refer Code q1.js, q1.html in Day71

Q.2 Output for code snippet.

Q.2.1

```
console.log(x - y)
```

Answers: **0**

Solution:

`x` and `y` are type casted to number types, as they are being operated on by `-` that requires a numeric type as operands.

If it was `+` instead of `-` it would have concatenated, as `+` with a string argument is defined to do so but for `-` no operation for string operand is defined so it will try to cast operands to numeric type. If both operands cannot be casted to numeric type then will return a **NaN.**

Q.2.2

```
console.log(x || y)
```

Answers: **220**

```
console.log(x || z)
```

Answers: **220**

```
console.log(x && y)
```

Answers: **undefined**

Solution:

For `||` operation following are the two important observations:

1. If the first operand is `true` it doesn't check further and returns the first operand.
2. If either of the operands is `true` it returns the operand value that makes it true even if the other is of `undefined` type.

For `&&` operation following are the three important observations:

1. Both operands are inspected.
2. If both operands turn out to be `true`, it returns the second operand.
3. String like `true`, `false`,`0` and any string are treated as true.
4. If any operand is `undefined`, it returns an `undefined`.

Q.2.3

```
1. var obj = #8711;  //means obj value stored at #8711
2. var obj2 = obj;   //no new memory is created as in case of primitive type
3. var obj2 = #8711; //means obj2 point same as obj
4. obj1.name = "Akki"; //Assuming obj1 to be typo and is obj
5. console.log(obj2);
```

Solution:

The solution is more of theoretical kind as javascript v8 engine doesn't define such construct.

Lets say, #8711 is some address number. It could be either primitive or a non-primitive type.

By observing line 4, we can say that it's some **non-primitive** type as it looks like some user defined object.

Console would log object with name field as `**Akki**`:

```
{...,
name: `Akki`,
...}
```

Q.2.4

```
1. var obj = {
2.   name: "vivek",
3.   getName: function(){
4.   console.log(this.name);
5.   }
6. }
7. var getName = obj.getName;     //return function
8. var obj2 = {name:"akshay", getName }; //define a new obj in new memory along with copy of getName function
9. obj2.getName();
```

Answer: `**akshay**`

Solution:

Line 7: returns function. `obj2` creates a new object with new memory as the name field is different and takes the same `getName` function body. Now calling `getName` for `obj2` would log the name field for `obj2` as this is defined and will log `akshay` as output.

Q.2.5

Answer: `**3**`

Solution:

Javascript interpreter does hoisting or moving up of declaration variables which allows variables to be written first in the code and declared later which is handled implicitly by the js interpreter.