First steps

1. **What is the highest type of object that can exist on the Ethereum blockchain?**
   Mark only one oval.

   - Variable
   - Contract
   - Coin
   - Function

2. **What is the most basic information every contract must have in order to be deployed?**
   Mark only one oval.

   - Contract name
   - Contract owner
   - Constructor function
   - Public variable

Code related question #1

Look at code #1 and answer the following questions

Code #1:

```
contract newContract{
}
```

3. **Can the contract newContract in code #1 be deployed to the Ethereum blockchain?**
   Mark only one oval.

   - No - It does't contain any information
   - No - It doesn't contain any constractur function
   - Yes - It will deploy the contract to the blockchain, even though the contract doesn't says much.
   - It depends, It will only deploy the contract as long as there's still free space on the blockchain

4. **Who is the owner of this contract**
   Mark only one oval.

   - Since no owner variable was specified, anyone can claim this contract as his or hers own
   - The one who deployed the contract is the owner
   - The last user to interact with the contract will be the owner
   - This contract doesn't have any owner

5. **What will happen if we'll send Ethers to newContract address?**
   Mark only one oval.

   - The Ethers will be refunded to the sender account
   - The Ethers will be transferred to the newContract owner
   - The Ethers will sit in the newContract account until the owner of that contract will claim them
   - The Ethers will sit in the newContract account for all eternity

6. **Who can delete (suicide) the newCountract account?**
   Mark only one oval.

   - Only the one who deployed the contract
   - Only the current owner of the contract
   - Anyone who knows the contract ABI
   - No one can delete, destroy or suicide this contract

Code related question #2

Look at code #2 and answer the following questions

code #2:

```
contract newBrand{

    string public brand;

    function newBrand(){
        brand = "Diginomics";
    }
}
```

7. **The contract newBrand has a function that is also called newBrand(). What type of function is it?**
   Mark only one oval.

   - Constructor function
   - Exception function
   - Primitive function
   - Required function

8. **What's so special about Constructor functions?**
   Mark only one oval.

   - They will only be executed once, when the contract is first deployed.
   - They can only be executed by the owner of the contract
   - They cannot receive any arguments from the users
   - They will be called whenever someone tries to interact with the contract

9. **In the contract newBrand in code #2 we've declared one variable "brand". What type of information this variable can hold, and who can read it?**
   Mark only one oval.

   - ◯ It contains the name of the brand and can be read by the owner of the contract
   - ◯ It contains a string (any type of string) and can be read by anyone who already knows (or searching for) that brand name.
   - ◯ It contains a string (any type of string) and can be read by anyone who knows the address of the contract
   - ◯ It contains a string (any type of string) and can be read by anyone.

10. **Assume we'll remove the attribute "public" from our string "brandName". Now who can read this variable?**
    Mark only one oval.

    - ◯ No one can read it
    - ◯ Only the owner of the contract can read it
    - ◯ No one can read it, but the owner can provide a function that will return this variable
    - ◯ Anyone who knows the address of the contract can read it

11. **Assume two different people will try to deploy the same contract (newContract) onto the blockchain. What will happen?**
    Mark only one oval.

    - ◯ There will be two different contract on the blockchain, with completely independent addresses that will contain the public string "Diginomic"
    - ◯ The second contract will not be stored on the blockchain, only the first one
    - ◯ Both contract will be stored on the blockchain under the same address
    - ◯ The second contract will replace the first contract

Code related question #3

Look at code #3 and answer the following questions

code #3:

```
contract mortal{

    address public owner;

    function mortal(){
        owner = msg.sender;
    }

    modifier onlyOwner{
        if (msg.sender == owner){

        }else{
            throw;
        }
    }

    function kill() onlyOwner{
        suicide(owner);
```

```
    }
}
```

12. **When deploying the contract "mortal", what address will be stored in the variable "owner"?**
    Mark only one oval.

    - The address of the contract
    - The address of the last account that sent a message to the contract
    - The address of the owner is predefined by the creator of the contract
    - The address of the one who deployed the contract

13. **Complete the following statement: The modifier onlyOwner uses the logical statement IF[_____] THAN[_____]**
    Mark only one oval.

    - IF[The sender of the message is equal to the owner] THAN[Proceed]
    - IF[The sender of the message has the same address as the contract] THAN[Proceed]
    - IF[The sender of the message has the same address as the contract] THAN[Throw]
    - IF[The contract has a defined owner] THAN[Proceed]

14. **Which of the following statements is FALSE?**
    Mark only one oval.

    - The "owner" variable can only store one address
    - The function "kill" can only be called by the owner of the contract
    - The function "mortal" can be called again to change the address stored in the "owner" variable
    - The contract "mortal" can have only one owner

15. **Is it possible to use the same modifier for other functions?**
    Mark only one oval.

    - No - Each modifier can be used only once
    - No - Each modifier has its own function. It cannot be attached to other functions
    - Yes - But since the result of the if-than statement will be the same for every function, It will run the if statements only once.
    - Yes - It will preform the same checking for each function call.

Code related question #4

Look at code #4 and answer the following questions

code #4:

```
contract mortal{

    address public owner;

    function mortal(){
        owner = msg.sender;
    }
```

```
    modifier onlyOwner{
        if (msg.sender == owner){
            _
        }else{
            throw;
        }
    }

    function kill() onlyOwner{
        suicide(owner);
    }
}

contract newBrand is mortal{

    string public brandName;

    function newBrand(string _brandName){
        brandName = _brandName;
    }

    function changeBrandName(string _newBrandName) onlyOwner {
        brandName = _newBrandName;
    }
}
```

16. **When deploying the contract "newBrand" the "mortal" contract is also deployed.**
Mark only one oval.

- ◯ True - The contract "newBrand" inherits from the contract "mortal". Therefor the "mortal" contract will also be deployed.
- ◯ True - The contract "newBrand" inherits from the contract "mortal". Therefor it needs to know the address of the contract "mortal"
- ◯ False - The contract "newBrand" is an independent contract and it doesn't depends on the contract "mortal"
- ◯ False - The contract "newBrand" inherits from the contract "mortal" and it will already contained all the variables and functions specified in the contract "mortal"

17. **Assume that the contract "mortal" was deployed by one account and the contract "newBrand" was deployed by another account. Who can access the function "changeBrandName"?**
Mark only one oval.

- ◯ The account which deployed the "newBrand" contract
- ◯ The account which deployed the "mortal" contract
- ◯ Both - The variable "owner" exists on both contract on contains both of the address
- ◯ Neither - The variable "owner" clashes so an error will be thrown

18. **How many instances of the contract "newBrand" can be deployed?**
Mark only one oval.

- Only one - Once a contract is deployed, it's already stored on the blockchain and cannot be deployed again
- Only once per address - There cannot be more then one instances of the contract per owner.
- Unlimited - The contract can be deployed again and again and again. Every time, the new contract will receive a new address
- Unlimited - As long as the address from which the contract is deployed hasn't been used to deploy a previous instance of that contract

19. **Assume that we remove the phrase "is mortal" from the "newBrand" contract, but still deploy the contract "newBrand" with the contract "mortal" still declared at the top of our code. What will happen?**
Mark only one oval.

- The contract "newBrand" won't be deployed because the function "changeBrandName" needs to declare an owner
- Everything will work just has before
- The contract wont be deployed because every contract needs to have an owner address
- Both contracts will be deployed simultaneously

20. **Can the contract "mortal" be deployed without deploying the contract "newBrand" at the same time?**
Mark only one oval.

- No - It's specifically stated in our code that the contract "newBrand" IS "mortal"
- No - If both contracts were declared, both will be deployed
- Yes - Each contract can be deployed independently
- Yes - But only the "mortal" contract can be deployed independently. The "newBrand" contract still requires us to first deploy the "mortal" contract

21. **Let's also remove the function "changeBrandName" from the contract "newBrand". Now, what will happen if we try to deploy the contract "newBrand"?**
Mark only one oval.

- It will deploy and wont contain the variable "owner"
- It will deploy. The "owner" variable will still be declared, but we won't be able to interact with the contract
- It won't deploy - We cannot deploy a contract without specifying an owner address
- It won't deploy unless we'll first separately deploy the contract "moral"

22. **When using the function "changeBrandName" what's happening in the blockchain to the old "brandName"?**
Mark only one oval.

- All instances of the old "brandName" are replaced by the new "brandName". For both old and new blocks.
- From that point onward, all new instances of the variable "brandName" will contain the new brand name. Previous instances will remain the same
- All previous instances will be deleted. Only the new "brandName" will remain on the blockchain
- As data cannot be erased from the blockchain Both brand names will exist simultaneously on the blockchain, but on different addresses

Variables

23. **Choose the FALSE statement**
Mark only one oval.

- The variable type uint can only hold non negative numbers
- The variable type address can only hold valid Ethereum address
- The variable type int256 can only hold numbers between 0 and 256
- The variable type uint64 can only hold non negative numbers of up to 64 bytes long

24. **What is the basic denomination of 1 Ether?**
Mark only one oval.

- Wei. 1 Ether equal 1*10^18 Wei
- Satoshi. 1 Ether equal 1*10^9 Satoshis
- Cents 1 Ether equal 1*10^2 Cents
- Nano. 1 Ether equal 1*10^9 Nanos

25. **What type of variables can be stored at "address" type variable?**
Mark only one oval.

- Only accounts address
- Only valid accounts address
- Only contracts addresses
- Any Ethereum valid address

26. **The variable type "bool" is used to store what?**
Mark only one oval.

- Boolean values only (true or false)
- The integers 0 and 1 only
- The string "true" or the string "false"
- Any undefined variable

27. **When defining an array such as string[] public name, what type of index will the array contain?**
Mark only one oval.

- ◯ The array will be indexed by stings
- ◯ The array is indexed by integers
- ◯ The array can be indexed by both integers and strings
- ◯ None of the above

28. **In order to add the name "Bob" to our names array we can use:**
Check all that apply.

- ☐ name[name.length++] = "Bob";
- ☐ name.push("Bob");
- ☐ name["Bob"];
- ☐ name[name] = "Bob";

29. **Can we store more then one "Bob" in our array?**
Mark only one oval.

- ◯ No, There can only be one item of each type in the array
- ◯ No, When we'll try to add the second "Bob", it will delete the previous "Bob" entry.
- ◯ Yes, but we'll have to manually write the index number for the new "Bob" entry
- ◯ Yes, each item in the array uses unique integer as its index.

30. **Can we store a struct type variable inside an array?**
Mark only one oval.

- ◯ Yes - But first we need to declare the struct object
- ◯ Yes - But only if the struct object contains the type of variables that the array was declared to hold (int, uint, string, etc)
- ◯ No - Struct objects might contain more than one variable type
- ◯ No - There's a storage limit

31. **True or False, When using mapping we can choose our own type of index**
Mark only one oval.

- ◯ True - As long as we're using a unique index
- ◯ True - As long as we're using address
- ◯ False - Only integers can be used as indexed
- ◯ False - Only address can be used as index when using mapping

32. **What happens when we try to insert data to an already existing array cell?**
Mark only one oval.

- ◯ The previous data is saved under new index
- ◯ The previous data is replaced
- ◯ We won't be able to make the change
- ◯ The new data will be saved under new index

33. **Can we have a multilevel mapping? (mapping within mapping)?**
Mark only one oval.

- ◯ Yes and we can use which ever index we want for the inner mapping
- ◯ No each mapping needs to have its own unique index
- ◯ Only if we're using address as our index since it's known that each address is uinque
- ◯ Yes, as long as we're using different indexes for each mapping

Proxy contracts

```
contract child{

   uint256 public childAge = 1;
   string childName;

      function setChildName(string _name){
            childName = _name;
      }

}

contract parent{

   address public yourChildAddress;
   uint256 public childAge;
   string childName;

   function parent(address _addressOfTheChildContract){
      yourChildAddress = _addressOfTheChildContract;
      child myChild = child(yourChildAddress);
      childAge = myChild.childAge();
   }

      function nameYourChild(string _myChildName){
            child myChild = child(yourChildAddress);
            myChild.setChildName(_myChildName);

      }
}
```

34. **True or false - The relationship between the Child contract and the Parent contract is of inheritance.**
Mark only one oval.

- True - The Parent contract inherent the childName and childAge variables from the Child contract
- True - The Child contract inherent the childName and childAge variables from the Parent contract
- False - Only contracts that contains the keyword "is" will have inheritance relations
- False - Only contracts with fixed size variable can inherent from other contracts

35. **Could the Parent contract be deployed without first deploying the Child contract?**
Mark only one oval.

- No - The constructor function in the Parent contract requires the address of a valid Child contract.
- Yes - There's no inheritance relationship between the two contracts.
- Yes - In fact, The Parent contract needs to be deployed before the Child contract
- Both contracts depends on each other, therefor they need to be deployed simultaneously

36. **Can the current Ethereum virtual machine read variables from other contracts?**
Mark only one oval.

- No - Contract can only access variables stored under its own address
- No - All variables must be declared in advance
- Yes - It can read every type of variable
- Yes - As long as the ABI of the contract is known and the variable size is pre determind

37. **The Parent contract can change the variable "childName" that was declared on the Child contract. Still, it cannot read it. Why?**
Mark only one oval.

- Sting variables have different sizes. The current EVM cannot read variables of unfixed size
- The Child contract was deployed before the Parent contract, therefor the Parent contract cannot access its variables
- The variable childName isn't a public variable, therefor it cannot be accessed
- The variable childName can only be declared once

38. **Who can access the function setChildName?**
Mark only one oval.

- Any user who knows the ABI of the Child contract
- Only the corresponding Parent contract
- Only the owner of the Child contract
- Only the owner of the Parent contract

39. **Who can access the function "nameYourChild"?**
Mark only one oval.

- ◯ Any user who knows the ABI of the Child contract
- ◯ Only the owner of the Parent contract
- ◯ Only the owner of the Child contract
- ◯ Any user who knows the ABI of the Parent contract

40. **How many Child contract can our Parent contract interact with**
Mark only one oval.

- ◯ Limitless amount
- ◯ Only 2
- ◯ It can interact with as many Child contracts as it wants as long as we change the childAddress variable
- ◯ It can only interact with 1 Child contract. The one we've inserted its address when first deploying the Parent contract

41. **What will happen if we removed the "public" attribute from the variable "childAge"?**
Mark only one oval.

- ◯ The Parent contract won't be allowed to read this variable
- ◯ The "childAge" variable will not be visible to other users
- ◯ The Parent contract will not deploy
- ◯ All of the above

42. **Changing the variable "childAge" from uint256 to uint8 can be done in 3 different ways. In only 2 ways the contracts will continue to work. Choose the 2 right options**
Check all that apply.

- ☐ Change the size for both contracts
- ☐ Change the size to uint8 only for the Child contract and leave it as uint256 for the Parent contract
- ☐ Change the size to uint8 only for the Parent contract and leave it as uint256 for the Child contract
- ☐ The contracts cannot be deployed after the variable size was change

General questions

43. **When can a function call an event?**
Mark only one oval.

- ◯ At any giving time
- ◯ Only once it competed its execution
- ◯ When new block is mined
- ◯ Functions can't call an event, only users can

44. **Once event was emitted, who can watch it?**

Mark only one oval.

- ◯ Anyone who's watching the contract
- ◯ Each event have a list or recipients
- ◯ Anyone who's address is part of the event message
- ◯ Only the own called the event

45. **How many events can be called?**

Mark only one oval.

- ◯ There's no limitation (except for gas price limitations)
- ◯ Only one event per function execution
- ◯ Only one event per block
- ◯ The number of events to be called is equal to the number of clients who watched the contract