Default Question Block

Thank you for participating in our experiment! You have the opportunity to help shape the future of programming language design, enabling safer and easier development of high-stakes software. The study will take about an hour to complete, but you will be given opportunities for breaks in the middle. Once you start the study, please finish it within 24 hours.

You will now be asked to answer some programming questions. We are interested both in your answers and in how long you take to answer the questions. If you need a break, please take one now; afterward, please focus on the questions without getting distracted until the survey is done.

These page timer metrics will not be displayed to the recipient.

First Click: *28.008 seconds*

Last Click: *28.008 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Read the document "Ownership -- Introduction". Once you are finished, answer the following questions. You may refer to the document as needed.

These page timer metrics will not be displayed to the recipient.

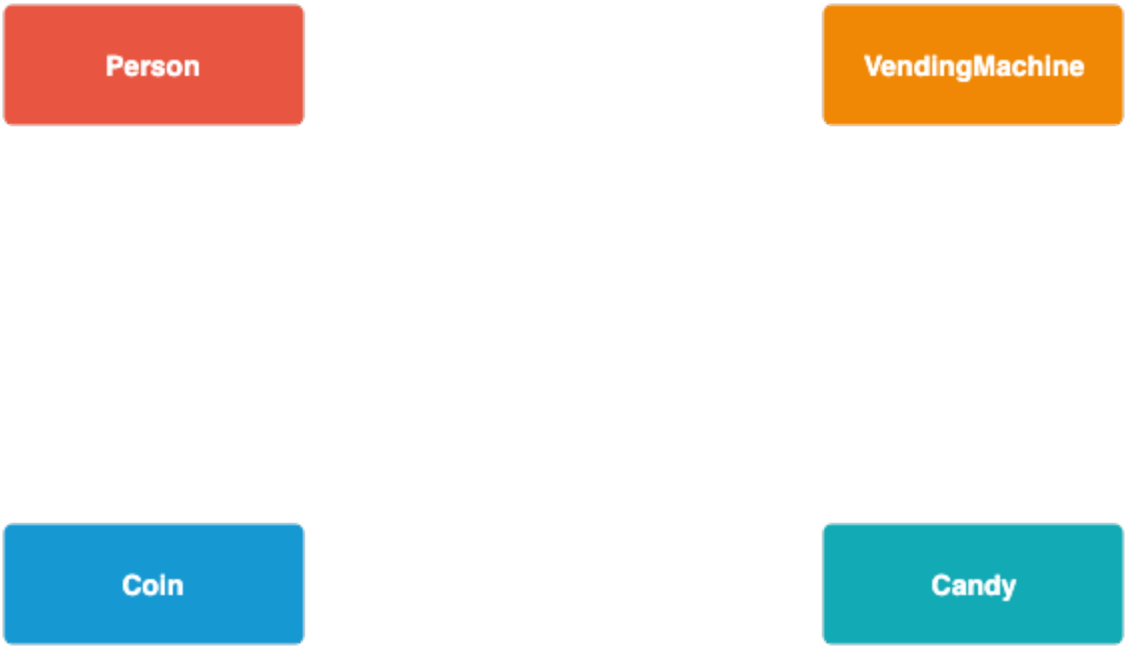
First Click: *28.001 seconds*

Last Click: *28.001 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

The following two questions refer to this diagram. A `Person` has a `Coin` which they want to use at a `VendingMachine` to acquire a `Candy`. When the `Person` places a `Coin` in the `VendingMachine`, they get a `Candy` in return. You may assume that initially, a `Person` doesn't have any `Candy`, and a `VendingMachine` doesn't have any `Coin`s.



Write the ownership references that are true BEFORE the Person places the Coin in the VendingMachine, based on the objects shown in the diagram above. One reference is given to you as an example for formatting (note it is NOT accurate); please follow this format for your references.

Example: Candy --> Coin@Owned (This means Candy has an Owned reference to a Coin object)

Write the ownership references that are true AFTER the Person places the Coin in the VendingMachine, based on the objects shown in the diagram above.

These page timer metrics will not be displayed to the recipient.

First Click: *27.995 seconds*

Last Click: *27.995 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Write a contract called `Person` that has an `Owned` reference to a `House` and a `Shared` reference to a `Park`. The `House` and `Park` contracts are given below.

```
contract House {

}

contract Park {

}
```

Please write your answer in the VSCode window (code1.obs). You may compile your code to check your answer.

These page timer metrics will not be displayed to the recipient.

First Click: *27.991 seconds*

Last Click: *27.991 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Read the document "Ownership -- Transactions". Once you are finished, answer the following questions. You may refer to the document as needed (and any previous ones).

These page timer metrics will not be displayed to the recipient.

First Click: *27.986 seconds*

Last Click: *27.986 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Refer to the code below for the next five questions.

```
contract Money {
    ...
}

contract Wallet {
    Money@Owned m;

    Wallet@Owned() {
        m = new Money();
    }

    transaction spendMoney() returns Money@Owned {
        ...
    }

    transaction receiveMoney(Money@Owned >> Unowned mon) {
        ...
    }
}
```

What is **m** in the above code fragment above?

- ◯ A Money object
- ◯ An Owned reference to a Money object
- ◯ An Owned object
- ◯ All of the above
- ◯ None of the above

What is the return type of the spendMoney transaction (write "none" if one exists)?

What is the return type of the receiveMoney transaction (write "none" if one exists)?

Assuming correct implementation, what is the Ownership status of `mon` at the **beginning** of the receiveMoney transaction?

Assuming correct implementation, what is the Ownership status of `mon` at the **end** of the receiveMoney transaction?

These page timer metrics will not be displayed to the recipient.

First Click: *27.981 seconds*

Last Click: *27.981 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Refer to the following code for the next two questions.

```
contract Share {

}

contract ShareHolder {
    Share@Owned share;

    ShareHolder@Owned() {
        share = new Share();
    }

    transaction receiveShare( /* TODO */ s) {
        ...
    }

    transaction checkShareValue(Share@Owned s) {
        ...
    }
}
```

What code goes in the /* TODO */ above?
(note: s is a Share owned by another ShareHolder that is passed as a parameter to the receiveShare transaction, and will be set to the field "share")

Please write your answer in the VSCode window (code2.obs). You may compile your code to check your answer.

These page timer metrics will not be displayed to the recipient.

First Click: *27.976 seconds*

Last Click: *27.976 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

This code has been copied over for your convenience.

```
contract Share {

}

contract ShareHolder {
    Share@Owned share;

    ShareHolder@Owned() {
        share = new Share();
    }

    transaction receiveShare( /* TODO */ s) {
        ...
    }

    transaction checkShareValue(Share@Owned s) {
        ...
    }
}
```

What is an equivalent way to write the **type** of s in the checkShareValue transaction declaration?

- ◯ Share@Shared
- ◯ Share@Unowned >> Owned
- ◯ Share@Owned >> Owned
- ◯ Share@Unowned >> Unowned
- ◯ None of the above

These page timer metrics will not be displayed to the recipient.

First Click: *27.971 seconds*

Last Click: *27.971 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Read the document "Ownership -- Variables". Once you are finished, answer the following questions. You may refer to the document as needed (and any previous ones).

These page timer metrics will not be displayed to the recipient.

First Click: *27.965 seconds*

Last Click: *27.965 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Refer to the following code for the next three questions.

```
contract Money {
    int amount;

    transaction getAmount() returns int {
        return amount;
    }
}

contract Wallet {
    Money@Owned m;

    Wallet@Owned() {
        m = new Money();
    }

    transaction spendMoney() {
        ...
    }
```

```
    transaction receiveMoney(Money@Owned >> Unowned mon) returns Money@Owned {
        Money temp = m;
        m = mon;
        return temp;
    }

    transaction checkMoney() returns Money@Owned {
        return m;
    }

}
```

What must be the ownership status of m after every transaction in Wallet?

What occurs in the receiveMoney transaction with regards to ownership? Is m's ownership status before the transaction begins the same as when it ends?

Would we get a compiler error with the checkMoney function?

- ◯ Yes, you cannot return a field of a contract in a transaction
- ◯ No, the return type matches the type of m
- ◯ Yes, returning m makes it Unowned, which doesn't match the ownership status of m's declaration
- ◯ No, m is of type Owned, which matches the ownership status of m's declaration
- ◯ None of the above

These page timer metrics will not be displayed to the recipient.

First Click: *27.96 seconds*

Last Click: *27.96 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Write a constructor for the `Money` object so that it accepts an integer as a parameter and sets "amount" to that integer value.
Note: you will have to change the instantiation of m in Wallet for your code to compile.
Please write your answer in the VSCode window (code3.obs). You may compile your code to check your answer.

These page timer metrics will not be displayed to the recipient.

First Click: *27.955 seconds*

Last Click: *27.955 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Read the document "Ownership -- Miscellaneous". Once you are finished, answer the following questions. You may refer to the document as needed (and any previous ones).

These page timer metrics will not be displayed to the recipient.

First Click: *27.95 seconds*

Last Click: *27.95 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Definitions of Money and Wallet contracts for the following questions.

```
contract Money {
    int amount;

    transaction getAmount() returns int {
        return amount;
```

```
    }
}

contract Wallet {
    Money@Owned m;

    Wallet@Owned() {
        m = new Money();
    }

    transaction spendMoney() {
        ...
    }

    transaction replaceMoney(Money@Owned >> Unowned mon) returns Money@Owned {
        Money temp = m;
        m = mon;
        return temp;
    }

}
```

Refer to the following code for the next question. Assume the definitions of Wallet and Money are as shown above.

```
transaction foo (Wallet@Owned w, Money@Owned m) {

    w.replaceMoney(m);
    w.spendMoney();
    w.replaceMoney(m);

}
```

What is the error with the transaction "foo"? Give your answer in terms of ownership.

These page timer metrics will not be displayed to the recipient.

First Click: *27.946 seconds*

Last Click: *27.946 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Write a Person contract that has an owned reference to a Wallet object. Person should also have an addMoney() transaction that takes in a Money parameter and passes it to its Wallet's replaceMoney() transaction. addMoney() should return the Wallet's initial money that was replaced.
Please write your answer in the VSCode window (code4.obs). You may compile your code to check your answer.

These page timer metrics will not be displayed to the recipient.

First Click: *27.94 seconds*

Last Click: *27.94 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Read the document "Assets". Once you are finished, answer the following questions. You may refer to the document as needed (and any previous ones).

These page timer metrics will not be displayed to the recipient.

First Click: *27.936 seconds*

Last Click: *27.936 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Refer to the following code for the next two questions.

```
asset contract Medicine {

}

asset contract Pharmacy {
    Medicine@Owned med;

    transaction getNewMedicine(Medicine@Owned >> Unowned m) {
        med = m;
    }

    transaction throwAwayMedicine() {
        disown med;
    }
}
```

What is the error (if one exists) with the getNewMedicine transaction?

- ◯ med becomes Unowned, although it should be Owned at the end of the transaction
- ◯ m is stated as becoming an Unowned reference, but actually stays Owned
- ◯ The owning reference to m is lost
- ◯ The owning reference to the original Medicine object (med) is lost
- ◯ There is no error

What is the error (if one exists) with the throwAwayMedicine transaction?

- ◯ An Owned reference cannot be disowned
- ◯ med is Unowned in the transaction, so it cannot be disowned
- ◯ med becomes Unowned after this transaction; this doesn't match its declaration
- ◯ Only a field can be disowned

◯ There is no error

How could you fix `getNewMedicine` so you don't get an error? Describe your answer in words/pseudocode. Write N/A if there is no error with the transaction.

These page timer metrics will not be displayed to the recipient.

First Click: *27.932 seconds*

Last Click: *27.932 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Read the document "States -- Introduction". Once you are finished, answer the following questions. You may refer to the document as needed (and any previous ones).

These page timer metrics will not be displayed to the recipient.

First Click: *27.927 seconds*

Last Click: *27.927 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

A `Drink` can be hot, cold, or lukewarm. If the `Drink` is cold, it has an `IceCube` object. If it is hot, it has a `CupSleeve`. The `Drink` always has an integer value to keep track of its temperature, and always starts lukewarm.

Write the `Drink` contract with the necessary states and fields as described above.
NOTE: You do NOT need to write a constructor for `Drink` (you can ignore this compiler error for now)

Please write your answer in the VSCode window (code5.obs). You may compile your code to

check your answer.

These page timer metrics will not be displayed to the recipient.

First Click: *27.923 seconds*

Last Click: *27.923 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Describe the relationship between states and ownership in your own words.

These page timer metrics will not be displayed to the recipient.

First Click: *27.919 seconds*

Last Click: *27.919 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Read the document "States -- Manipulating State". Once you are finished, answer the following questions. You may refer to the document as needed (and any previous ones).

These page timer metrics will not be displayed to the recipient.

First Click: *27.914 seconds*

Last Click: *27.914 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Refer to the following code for the next four questions.

```
contract Voter {
```

```
    string name;
    bool citizen;

    state Eligible;
    state Ineligible;
    state Registered;
    state FinishedVoting;

    Voter@Owned(string n, bool citizenship) {
        name = n;
        citizen = citizenship;
        if (!citizen) {
            -> Ineligible;
        }
        else {
            -> Eligible;
        }
    }

    transaction vote(Voter@Registered >> FinishedVoting this) {
        // TODO
    }

    transaction register(Voter@Eligible >> (Registered | Ineligible) this) {
        ...
    }
}
```

What state(s) will a Voter be in after it is first instantiated?

What state(s) will a Voter be in after the transaction "register"?

What state(s) must a Voter be in to vote?

These page timer metrics will not be displayed to the recipient.

First Click: *27.91 seconds*

Last Click: *27.91 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Finish the `vote` transaction by transitioning the state of a `Voter` to the required state.
Please write your answer in the VSCode window (code6.obs). You may compile your code to check your answer.

These page timer metrics will not be displayed to the recipient.

First Click: *27.906 seconds*

Last Click: *27.906 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Refer to the following code for the next two questions.

```
asset contract Candy {
}

asset contract Coin {
}

asset contract CoinBag {

    transaction deposit(Coin @ Owned >> Unowned c) {
        // Real implementation not shown; for now, just throw c away.
        disown c;
    }
```

```
}

main asset contract TinyVendingMachine {
    CoinBag @ Owned coinBin;

    state Full {
        Candy @ Owned inventory;
    }
    state Empty;

    TinyVendingMachine@Owned () {
        coinBin = new CoinBag();
        ->Empty;
    }

    transaction restock(TinyVendingMachine @ Empty >> Full this,
                        Candy @ Owned >> Unowned c) {
        //TODO
    }

    transaction buy(/* TODO */ this,  /* TODO */ c) returns Candy @ Owned {
        coinBin.deposit(c);
        Candy result = inventory;
        ->Empty;
        return result;
    }
}
```

Write code to complete the `buy` transaction declaration above.
Please write your answer in the VSCode window (code7.obs). You may compile your code to check your answer.

Complete the `restock` transaction above by transitioning the state of `TinyVendingMachine` to `Full`.
Please write your answer in the VSCode window (code7.obs). You may compile your code to check your answer.

These page timer metrics will not be displayed to the recipient.

First Click: *27.902 seconds*

Last Click: *27.902 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Read the documents "States -- Miscellaneous" and "States and Assets". Once you are finished, answer the following questions. You may refer to the document as needed (and any previous ones).

These page timer metrics will not be displayed to the recipient.

First Click: *27.896 seconds*

Last Click: *27.896 seconds*

Page Submit: *0 seconds*

Click Count: *1 clicks*

Which of the following can be passed as a parameter to the following transaction (assume `Money` is an **asset**):

```
transaction foo(Money@Owned money) {

}
```

You may choose more than one answer.

- [ ] Money@Owned m
- [ ] Money@Unowned m
- [ ] Money@Shared m
- [ ] None of the above

Which of the following can be passed as a parameter to the following transaction (assume `Money` is an **asset**):

```
transaction foo(Money@Unowned money) {
```

```
}
```

You may choose more than one answer.

- [ ] Money@Owned m
- [ ] Money@Unowned m
- [ ] Money@Shared m
- [ ] None of the above

Which of the following can be passed as a parameter to the following transaction (assume `Money` is an **asset**):

```
transaction foo(Money@Shared money) {

}
```

You may choose more than one answer.

- [ ] Money@Owned m
- [ ] Money@Unowned m
- [ ] Money@Shared m
- [ ] None of the above

Read the document "Using Obsidian on a Blockchain." Once you are finished, answer the following questions. You may refer to the document as needed (and any previous ones).

These page timer metrics will not be displayed to the recipient.

#EditSection, TimingFirstClick#: *27.892 seconds*

#EditSection, TimingLastClick#: *27.892 seconds*

#EditSection, TimingPageSubmit#: *0 seconds*

#EditSection, TimingClickCount#: *1 clicks*

Transactions of contracts that are installed on blockchain should take a `this` parameter of type...

- ◯ Shared
- ◯ Owned

- ◯ Unowned

If an asset contract is installed on the blockchain:

- ◯ The blockchain itself will be considered the owner, preventing any asset loss, and a special exception allows clients to have Shared references to the contract instance.
- ◯ There will be an error because assets cannot be installed on the blockchain.

Read the document "Taking Advantage of Ownership." Once you are finished, answer the following questions. You may refer to the document as needed (and any previous ones).

These page timer metrics will not be displayed to the recipient.

#EditSection, TimingFirstClick#: *27.887 seconds*

#EditSection, TimingLastClick#: *27.887 seconds*

#EditSection, TimingPageSubmit#: *0 seconds*

#EditSection, TimingClickCount#: *1 clicks*

A Bank class uses owned references to Money objects to keep track of who owns which money. What are some advantages of doing this rather than using data structures that track individual account balances as numbers?