< Daml Contract Developer Cert... (DAML104) (https://daml.talentlms.com/dashboard)

You completed this exam on *26/08/2022, 10:59*
Your score is 76.92%

CORRECT

Code Analysis - I

Given the below template and script, and assuming the script returns without error, select the applicable option.

This question is part of a series of five. **The code presented in all five parts is identical.**

```
import Daml.Script

template Baz
  with
    s : Party
    c : [Party]
  where
    signatory s
    observer c

    nonconsuming choice Clone
      : ContractId Foo
      with
        ct : Party
        fooCid : ContractId Foo
      controller ct
      do
        assert (ct `elem` c)
        foo <- fetch fooCid
        create foo

    nonconsuming choice Fetch
      : Foo
      with
        ct : Party
        cid : (ContractId Foo)
      controller ct
      do
        assert (ct `elem` c)
        fetch cid

    nonconsuming choice Exercise
      : Foo
      with
        ct : Party
        p : Party
        cid : (ContractId Foo)
      controller ct
      do
        assert (ct `elem` c)
        exercise cid Bar with p


testFoo : Script () = do
  parties@[p1, p2, p3, p4] <- getParties
  let foo : Foo = getFoo p1 p2 p3 p4

  fooCid <- submit p1 do
    createCmd foo

  bazCid <- submit p4 do
    createCmd Baz with
      s = p4
      c = [p1, p2, p3]

  submitMustFail p2 do
```

```
    archiveCmd fooCid

  submit p2 do
    exerciseCmd bazCid Clone with
      ct = p2
      fooCid

  submitMustFail p3 do
    exerciseCmd bazCid Fetch with
      ct = p3
      cid = fooCid

  submit p2 do
    exerciseCmd bazCid Exercise with
      ct = p2
      p = p3
      cid = fooCid

  submit p3 do
    exerciseCmd bazCid Fetch with
      ct = p3
      cid = fooCid

  return ()
```

`Foo` is a choice type

`Foo` is a template type

`fooCid` is of type `Foo`

`foo` is of type `Bar`

CORRECT

Code Analysis - II

Given the below template and script, and assuming the script returns without error, select the applicable option.

This question is part of a series of five. **The code presented in all five parts is identical.**

```
import Daml.Script

template Baz
  with
    s : Party
    c : [Party]
  where
    signatory s
    observer c

    nonconsuming choice Clone
      : ContractId Foo
      with
        ct : Party
        fooCid : ContractId Foo
      controller ct
      do
        assert (ct `elem` c)
        foo <- fetch fooCid
        create foo

    nonconsuming choice Fetch
      : Foo
      with
        ct : Party
        cid : (ContractId Foo)
      controller ct
      do
        assert (ct `elem` c)
        fetch cid

    nonconsuming choice Exercise
      : Foo
      with
        ct : Party
        p : Party
        cid : (ContractId Foo)
      controller ct
      do
        assert (ct `elem` c)
        exercise cid Bar with p


testFoo : Script () = do
  parties@[p1, p2, p3, p4] <- getParties
  let foo : Foo = getFoo p1 p2 p3 p4

  fooCid <- submit p1 do
    createCmd foo

  bazCid <- submit p4 do
    createCmd Baz with
      s = p4
      c = [p1, p2, p3]

  submitMustFail p2 do
```

```
    archiveCmd fooCid

  submit p2 do
    exerciseCmd bazCid Clone with
      ct = p2
      fooCid

  submitMustFail p3 do
    exerciseCmd bazCid Fetch with
      ct = p3
      cid = fooCid

  submit p2 do
    exerciseCmd bazCid Exercise with
      ct = p2
      p = p3
      cid = fooCid

  submit p3 do
    exerciseCmd bazCid Fetch with
      ct = p3
      cid = fooCid

  return ()
```

p1 is the signatory of the contract referenced by fooCid

p2 is the signatory of the contract referenced by fooCid

p3 is the signatory of the contract referenced by fooCid

The contract referenced by fooCid has multiple signatories

CORRECT

Code Analysis - III

Given the below template and script, and assuming the script returns without error, select the applicable option.

This question is part of a series of five. **The code presented in all five parts is identical.**

```
import Daml.Script

template Baz
  with
    s : Party
    c : [Party]
  where
    signatory s
    observer c

    nonconsuming choice Clone
      : ContractId Foo
      with
        ct : Party
        fooCid : ContractId Foo
      controller ct
      do
        assert (ct `elem` c)
        foo <- fetch fooCid
        create foo

    nonconsuming choice Fetch
      : Foo
      with
        ct : Party
        cid : (ContractId Foo)
      controller ct
      do
        assert (ct `elem` c)
        fetch cid

    nonconsuming choice Exercise
      : Foo
      with
        ct : Party
        p : Party
        cid : (ContractId Foo)
      controller ct
      do
        assert (ct `elem` c)
        exercise cid Bar with p


testFoo : Script () = do
  parties@[p1, p2, p3, p4] <- getParties
  let foo : Foo = getFoo p1 p2 p3 p4

  fooCid <- submit p1 do
    createCmd foo

  bazCid <- submit p4 do
    createCmd Baz with
      s = p4
      c = [p1, p2, p3]

  submitMustFail p2 do
```

```
    archiveCmd fooCid

  submit p2 do
    exerciseCmd bazCid Clone with
      ct = p2
      fooCid

  submitMustFail p3 do
    exerciseCmd bazCid Fetch with
      ct = p3
      cid = fooCid

  submit p2 do
    exerciseCmd bazCid Exercise with
      ct = p2
      p = p3
      cid = fooCid

  submit p3 do
    exerciseCmd bazCid Fetch with
      ct = p3
      cid = fooCid

  return ()
```

Parties p1 and p4 are the same

Parties p2 and p3 are the same

Parties p4 and p3 are the same

CORRECT

Code Analysis - IV

Given the below template and script, and assuming the script returns without error, select the applicable option.

This question is part of a series of five. **The code presented in all five parts is identical.**

```
import Daml.Script

template Baz
  with
    s : Party
    c : [Party]
  where
    signatory s
    observer c

    nonconsuming choice Clone
      : ContractId Foo
      with
        ct : Party
        fooCid : ContractId Foo
      controller ct
      do
        assert (ct `elem` c)
        foo <- fetch fooCid
        create foo

    nonconsuming choice Fetch
      : Foo
      with
        ct : Party
        cid : (ContractId Foo)
      controller ct
      do
        assert (ct `elem` c)
        fetch cid

    nonconsuming choice Exercise
      : Foo
      with
        ct : Party
        p : Party
        cid : (ContractId Foo)
      controller ct
      do
        assert (ct `elem` c)
        exercise cid Bar with p


testFoo : Script () = do
  parties@[p1, p2, p3, p4] <- getParties
  let foo : Foo = getFoo p1 p2 p3 p4

  fooCid <- submit p1 do
    createCmd foo

  bazCid <- submit p4 do
    createCmd Baz with
      s = p4
      c = [p1, p2, p3]

  submitMustFail p2 do
```

```
    archiveCmd fooCid

  submit p2 do
    exerciseCmd bazCid Clone with
      ct = p2
      fooCid

  submitMustFail p3 do
    exerciseCmd bazCid Fetch with
      ct = p3
      cid = fooCid

  submit p2 do
    exerciseCmd bazCid Exercise with
      ct = p2
      p = p3
      cid = fooCid

  submit p3 do
    exerciseCmd bazCid Fetch with
      ct = p3
      cid = fooCid

  return ()
```

p2 is an observer of fooCid

p3 is an observer of fooCid

CORRECT

Code Analysis - V

Given the below template and script, and assuming the script returns without error, select the applicable option.

This question is part of a series of five. **The code presented in all five parts is identical.**

```
import Daml.Script

template Baz
  with
    s : Party
    c : [Party]
  where
    signatory s
    observer c

    nonconsuming choice Clone
      : ContractId Foo
      with
        ct : Party
        fooCid : ContractId Foo
      controller ct
      do
        assert (ct `elem` c)
        foo <- fetch fooCid
        create foo

    nonconsuming choice Fetch
      : Foo
      with
        ct : Party
        cid : (ContractId Foo)
      controller ct
      do
        assert (ct `elem` c)
        fetch cid

    nonconsuming choice Exercise
      : Foo
      with
        ct : Party
        p : Party
        cid : (ContractId Foo)
      controller ct
      do
        assert (ct `elem` c)
        exercise cid Bar with p


testFoo : Script () = do
  parties@[p1, p2, p3, p4] <- getParties
  let foo : Foo = getFoo p1 p2 p3 p4

  fooCid <- submit p1 do
    createCmd foo

  bazCid <- submit p4 do
    createCmd Baz with
      s = p4
      c = [p1, p2, p3]

  submitMustFail p2 do
```

```
    archiveCmd fooCid

  submit p2 do
    exerciseCmd bazCid Clone with
      ct = p2
      fooCid

  submitMustFail p3 do
    exerciseCmd bazCid Fetch with
      ct = p3
      cid = fooCid

  submit p2 do
    exerciseCmd bazCid Exercise with
      ct = p2
      p = p3
      cid = fooCid

  submit p3 do
    exerciseCmd bazCid Fetch with
      ct = p3
      cid = fooCid

  return ()
```

The choice `Exercise` adds `p` as an observer to fooCid

The choice `Exercise` has no visible side effects

The choice `Exercise` divulges fooCid to `p`

CORRECT

Serializability

Select the type that is serializable.

`ContractId Foo -> Update Int`

`Update (Optional Int)`

`Int -> Int -> Int -> Int`

`Optional (ContractId Foo)`

CORRECT

The Decimal Type

What does the `Decimal` type represent?

A IEEE 754 binary128 quadruple-precision floating point number

A IEEE 754 decimal128 decimal floating point number

A Java `BigDecimal` immutable arbitrary precision decimal

An ANSI SQL NUMERIC(38,10) fixed point number

CORRECT

Value Immutability

Select the correct answer given the the below code excerpt

```
import Daml.Script

i : Script (Foo, Foo, Foo) = do
 let foo = -- Omitted
 let bar = foo

 let doStuffWith bar = do
       let baz = foo
       -- Omitted code
       return (baz)

 baz <- doStuffWith bar
 return (foo, bar, baz)
```

It's guaranteed that `foo` == `baz` in the returned tuple

It's guaranteed that `foo` == `bar` in the returned tuple

It's guaranteed that `bar` == `baz` in the returned tuple

CORRECT

Translating Daml to Transactions

How many actions does the below Script result in?

```
import Daml.Script
import DA.Action

template T
 with
   p : Party
 where
   signatory p
   controller p can
     nonconsuming FetchRecursively : T
       with
         n : Int
       do
         when (n > 0) do
           exercise self FetchRecursively with n = n - 1
           return ()
         fetch self
recurs : Script T = do
  p <- allocateParty "p"
  cid <- submit p do
     createCmd T with p
  submit p do
     exerciseCmd cid FetchRecursively with n = 3
```

10

9

3

7

15

8

1

CORRECT

Basic Type Classes

Match each data type with a typeclass instance for that type provided by the standard library or compiler.

Set	*Functor*
Update	*Action*

Int	*Ord*
ContractId Foo	*Eq*

INCORRECT

Inbuilt Typeclasses

Select the **two** true statements about typeclasses in Daml

Every template type automatically derives Eq and Show

Every template type automatically derives Eq, Show, and Ord

Every custom data type automatically derives Eq and Show

Function types automatically derive Eq

The Eq typeclass defines `==`, `<=`, and `=>`

The Show typeclass defines `toString`

To make a custom data type serializable, a `Serializable` instance needs to be defined

Instances of Eq and Show can be derived automatically in some cases

CORRECT

Declaring Variants

Fill the gaps to make the script succeed

```
Import Daml.Script

[ ✓ data ] [ ✓ Foo ] =
  [ ✓ Bar ] Int | [ ✓ Baz ] Text
t : Script (Foo, Foo) = do
  return (Baz "A", Bar 1)
```

CORRECT

Pattern Matching

Which of the following functions a, b, c, d and e can be used to turn the value of a value of type Sum into a Text?

```
data Sum
 = SInt Int
 | SText Text
 | SBool Bool

a s = case s of
 SInt i -> "SInt " <> show i
 SText t -> "SText " <> t
 SBool b -> "SBool " <> show b

b s = switch s
 case SInt i -> "SInt " <> show i
 case SText i -> "SText " <> t
 case SBool i -> "SBool " <> show b

c s = typeOf s <> " " <> show s.value

d s = case s of
 | SInt i = "SInt " <> show i
 | SText t = "SText " <> t
 | SBool b = "SBool " <> show b

e s = show s
```

a

b

c

d

e

CORRECT

If..Else Expression

Which of the following choices creates an R1 if b == True and an R2 otherwise? Select the correct answer.

```
template T
with
  p : Party
where
  signatory p

  controller p can
```

```
A
  : ()
  with
    b : Bool
  do
    if b
      then create R1 with p
      else create R2 with p
    return ()
```

```
B
  : ()
  with
    b : Bool
  do
    if b
      then do
        create R1 with p
        return ()
      else do
        create R2 with p
        return ()
```

```
C
  : ()
  with
    b : Bool
  do
    if b
      then do
        create R1 with p
        return ()
    create R2 with p
    return ()
```

```
D
  : ()
  with
    b : Bool
  do
    if b
      then
        create R1 with p
        return ()
      else
        create R2 with p
        return ()
E
  : ()
  with
    b : Bool
  do
    if b
      then
        r1 <- create R1 with p
        return (Left r1)
      else
        r2 <- create R2 with p
        return (Right r2)
```

A

B

C

D

E

CORRECT

Expressions, Values, Actions, and do

Fill the gaps in the below code to make it succeed.

```
{-# LANGUAGE ApplicativeDo #-}
module Expressions where

import Daml.Script

template T
  with
    p : Party
  where
    signatory p

comp : Script () = do
  p <- allocateParty "P"

  let
    [ ✓ c = ] createCmd T with p
    [ ✓ cmds = do ]
      cid1 [ ✓ <- ] c
      [ ✓ cid2 <- c ]
      return (cid1, cid2)
   (cid1, cid2) <- submit p cmds

  assert (cid1 /= cid2)
```

INCORRECT

The Update Action

Select the **three** applicable options.

A `create` statement always returns a `ContractId a`

A `create` statement always returns a `Update (ContractId a)`

`fetch` has type `(HasFetch a) => ContractId a -> Update a`

`lookupByKey` has type `(Key k a) => k -> Optional a`

A successfully interpreted `Update a` value in Daml corresponds to an Action in the Ledger Model

A successfully interpreted `Update a` value in Daml corresponds to a Transaction in the Ledger Model

CORRECT

Controllers and Choices

Select all that apply:

Controllers in Daml correspond to Exercise Actors in the Ledger Model

Controllers in Daml correspond to Required Authorizers in the Ledger Model

Controllers specified using the `controller c can` syntax in Daml are observers of the contract

Choices are consuming by default

INCORRECT

Function Signatures

Match function signatures with expressions

fn : ((),()) -> () -> ()	*fn (_ -> ()) [()]*
fn : [()] -> [[()]] -> [[[()]]]	*fn [()] []*
fn : (() -> ()) -> F () -> G ()	*fn ((), ()) ()*
fn : () -> ()	*fn ()*

CORRECT

Basic StdLib Functions

Match signatures to functions

elem :	*(Eq a) => a -> [a] -> Bool*
filter :	*(a -> Bool) -> [a] -> [a]*
foldl :	*(a -> b -> a) -> a -> [b] -> a*
const :	*a -> b -> a*
fmap :	*(Functor F) => (a -> b) -> F a -> F b*

CORRECT

Defining Custom Functions

Which of the following are correct definitions for a custom `const` function?

```
constA = const
```

```
constB : a -> b -> a
constB x y = x
```

```
constC (x : a) (y : b) : a = x
```

```
constD : (a -> b -> a) = (\x y -> x)
```

```
constE x _ = x
```

constA

constB

constC

constD

constE

INCORRECT

Looping and Recursion

Which of the following are working alternatives for the `filter` function?

```
filterRecurs : (a -> Bool) -> [a] -> [a]
filterRecurs  f xs = case xs of
    [] -> []
    x::xss -> if f x
        then x::yss
        else  yss
        where yss = (filterRecurs f xss)
```

```
filterFold : (a -> Bool) -> [a] -> [a]
filterFold f xs = foldr work [] xs
    where work x acc = if f x then x::acc else acc
```

```
filterFor : (a -> Bool) -> [a] -> [a]
filterFor f xs = ys
    where
        ys = []
        for xs (\x -> if f x
                then ys = x::ys
                else return ())
```

```
filterMap : (a -> Bool) -> [a] -> [a]
filterMap f xs = map work xs
    where work x = if f x then x else ()
```

filterRecurs

filterFold

filterFor

filterMap

CORRECT

Errors and Aborts

Which of these scripts will succeed?

```
a : Script () = do
  abort "Foo"
  return ()
```

```
b : Script () = do
  let a : Script () = abort "Foo"
  return ()
```

```
c : Script () = do
  let x : Decimal = 1.0 / 0.0
  return ()
```

```
d : Script () = do
  let e : () = error "foo"
  assert (e == e)
  return ()
```

```
e : Script () = do
  let e : Either Text () = abort "foo"
  assert (e == e)
  return ()
```

a

b

c

d

e

INCORRECT

Packaging, Modularization and Identifiers

A Daml Project is configured using a [✓ ***daml.yaml***] file. The `daml build` command picks up that file and compiles all referenced source files into a single [✓ ***package***]. The unique identifier of a compiled template is usually represented in the format X:Y:Z format, where X is the [*template name*], Y is the module name, and Z is the [✓ ***template name***].

CORRECT

Searching the StdLib by Signature

Name a standard library function with signature `(Applicative m) => Int -> m a -> m [a]`

replicateA

CORRECT

Know your Dev Tools

Match tools with a statement that applies to them

Daml REPL	*can be used with or without connecting to a Ledger*
Daml Script	*Has both Daml IDE integration and runs against the Ledger API*
Daml Scenarios	*only works in the Daml IDE and Sandbox*
Navigator	*has a customizable web-based user interface*

CORRECT

The IDE Transaction View

Select **two** options that **DO NOT** apply to the below Transaction View of a Script in the IDE

```
Transactions:
 TX 0 1970-01-01T00:00:00Z (Main:30:14)
 #0:0
 │   consumed by: #1:0
 │   referenced by #1:0
 │   known to (since): 'Alice' (0)
 └─> create Main:Asset
     with
       issuer = 'Alice'; owner = 'Alice'; name = "TV"

 TX 1 1970-01-01T00:00:00Z (Main:36:12)
 #1:0
 │   known to (since): 'Alice' (1)
 └─> 'Alice' exercises Give on #0:0 (Main:Asset)
             with
               newOwner = 'Bob'
     children:
     #1:1
     │   consumed by: #2:0
     │   referenced by #2:0
     │   known to (since): 'Alice' (1), 'Bob' (1)
     └─> create Main:Asset
         with
           issuer = 'Alice'; owner = 'Bob'; name = "TV"

 TX 2 1970-01-01T00:00:00Z (Main:39:3)
 #2:0
 │   known to (since): 'Bob' (2), 'Alice' (2)
 └─> 'Bob' exercises Give on #1:1 (Main:Asset)
           with
             newOwner = 'Alice'
     children:
     #2:1
     │   known to (since): 'Bob' (2), 'Alice' (2)
     └─> create Main:Asset
         with
           issuer = 'Alice'; owner = 'Alice'; name = "TV"

Active contracts:  #2:1

Return value: #2:1
```

There are three Commits

There are five actions

`owner` is a signatory of `Asset`

`issuer` is a signatory of `Asset`

`owner` is a controller of the choice `Give`

Bob is an observer of the final contract

Bob witnessed the final contract

CORRECT

Testing for Failure

What's the keyword used in Script to test that a transaction cannot be submitted?

submitMustFail

CORRECT

Security Analysis and Testing I

The below code shows an Iou with a Merge choice and a script to test that choice. Select the applicable statement about this model.

This question is part of a two-part series, and **the code presented in both parts is identical**.

```
import Daml.Script

template Iou
with
  issuer : Party
  owner : Party
  amount : Decimal
where
  signatory issuer
  controller owner can
    Merge
      : ContractId Iou
      with
        cid : ContractId Iou
      do
        asset <- fetch cid
        assert (asset.owner == owner)
        archive cid
        create asset with
          amount = this.amount + asset.amount
testAsset : Script () = do
 [issuer, owner] <- mapA allocateParty ["Alice", "Bob"]
 asset1 <- submit issuer do
   createCmd Iou with amount = 1.0; ..
 asset2 <- submit issuer do
   createCmd Iou with amount = 1.0; ..
 mergedAsset <- submit owner do
   exerciseCmd asset1 Merge with cid = asset2
  [(_, mergedIou)] <- query @Iou owner

 assert (mergedIou.issuer == issuer)
 assert (mergedIou.owner == owner)
 assert (mergedIou.amount == 2.0)
```

The owner could merge in an Iou from another owner, thus stealing money

The owner could create a new Iou thus creating money from nowhere

This model is safe

The owner could manipulate the amount of an existing Iou thus creating money from nowhere

---

CORRECT

Security Analysis and Testing - II

What modifications would you make to this code to make it safe, and test that it’s safe? Select the **two** best options.

This question is part of a two-part series, and **the code presented in both parts is identical**.

```
import Daml.Script

template Iou
with
  issuer : Party
  owner : Party
  amount : Decimal
where
  signatory issuer
  controller owner can
    Merge
      : ContractId Iou
      with
        cid : ContractId Iou
      do
        asset <- fetch cid
        assert (asset.owner == owner)
        archive cid
        create asset with
          amount = this.amount + asset.amount
testAsset : Script () = do
 [issuer, owner] <- mapA allocateParty ["Alice", "Bob"]
 asset1 <- submit issuer do
   createCmd Iou with amount = 1.0; ..
 asset2 <- submit issuer do
   createCmd Iou with amount = 1.0; ..
 mergedAsset <- submit owner do
   exerciseCmd asset1 Merge with cid = asset2
  [(_, mergedIou)] <- query @Iou owner

 assert (mergedIou.issuer == issuer)
 assert (mergedIou.owner == owner)
 assert (mergedIou.amount == 2.0)
```

None, it’s already safe

Change the assert clause in the choice from `asset.owner == owner` to `asset.issuer == issuer`

Add another assert clause to the choice, checking that `asset.issuer == issuer`

Change the assert to check that `archive` and `create` statements in the choice are well authorized

Add `submitMustFail` statements to the script to check that `Merge` can’t be called in any degenerate cases

CORRECT

Daml Ledger Structure

A Ledger is a sequence of Commits, each of which is a [✓ ***Transaction***] annotated with a party called the requester, or sometimes submitter. A Transaction is a list of [✓ ***Actions***], of which there are four types. [✓ ***Exercise***] Actions can have consequences, which are again a Transaction.

CORRECT

Daml's Time Model

The Daml Ledger Model distinguishes between two important types of time: Ledger Time which is set by [✓ ***the submitting node during interpretation***], and Record Time which is set by [✓ ***the commit/synchronisation protocol during commit***]. The difference between the two is bounded by the Skew parameters of the ledger. Daml's getTime function gives access to the [✓ ***Ledger***] Time.

CORRECT

Party and Contract Relationships

[✓ ***Signatories***] of a contract are parties which control their existence. They are [✓ ***required***], and the authority of [✓ ***all***] such party(/ies) is needed to create or archive the contract. They are guaranteed to be informed of [✓ ***all***] actions on the contract.

INCORRECT

Informees and Witnesses

Select all **three** applicable options

An informee of an action is also a witness of that action.

An informee of a create action of a contract c is always a stakeholder of c.

A witness of a create action of a contract c is always a stakeholder of c.

An informee of an exercise action of a contract c is always a stakeholder of c.

An informee of an action is a witness of all its subactions.

Witnesses of the create action of c are guaranteed to also witness the consuming exercise on c.

All stakeholders on a contract c are informees on all actions on c.

CORRECT

The Authorization Model

The consequences of an `Exercise Action a` on a contract `c` are authorized by [✓ ***the signatories of c and the actors of a***]. Every action has a set of required authorizers. A transaction is well-authorized if the [✓ ***required authorizers***] of every action are a subset of the [✓ ***authorizers***].

CORRECT

Consistency

The Daml Ledger Model guarantees that for every contract c, there is at most one [✓ ***Create***] action which comes before all other actions on c and moves c into the [✓ ***Active***] state, and that there is at most one [✓ ***Consuming Exercise***] action, which comes after all other actions on c and moves c into the archived state.

CORRECT

Ledger Immutability

Select the correct answer

Once an action has been written to the ledger as part of a commit, it can never be changed.

The set of active contracts is immutable.

Active contracts can be modified using an Update Action.

INCORRECT

Execution Phases and Exceptions

Assuming the submitting participant node is honest, functional, and up to date, during which phase do exceptions occur?

1. An error due to an invalid JWT token happens during [*Validation*]
2. An error due to an incorrect key lookup happens during [*Confirmation*]
3. A division by zero error happens during [✓ ***Interpretation***]

INCORRECT

Contention on Contract Keys

```
template T
  with
    s : Party
    o : Party
 where
    signatory s
    observer o
    key s : Party
    maintainer key
```

There are exactly two live contracts of type `T`:

```
`T with s = alice; o = bob`
`T with s = bob; o = bob`
```

Assuming transactions are otherwise well-authorized and correct, select all that apply

If `alice` submits a transaction containing `fetchByKey @T bob`, it [✓ ***will fail during interpretation***].
If `bob` submits a transaction containing `lookupByKey @T alice`, it [*will fail during validation*].
If `bob` submits a transaction containing `fetchByKey @T alice`, but `alice` archives instance 1 of T before validation, the transaction [✓ ***will fail during validation***].
If `alice` submits a transaction containing `lookupByKey @T bob`, but bob archives instance 2 of T before validation, the transaction [✓ ***will succeed***].

INCORRECT

One Script, Many Uses

Match a way of running Scripts with what it does

Opening a Daml file with a Script in the IDE	*runs the script on a special in-memory ledger emulator (the Script Service)*
Running `daml test`	*runs all scripts in the project or given DAR file and runs them in a special in-memory ledger emulator (the Script Service)*
Running `daml script --all`	*takes a DAR file, a script identifier, and a ledger connection and runs the script against that ledger*
Running `daml script`	*starts a Sandbox and runs all scripts in a given DAR file against that Sandbox*

Exam completed!