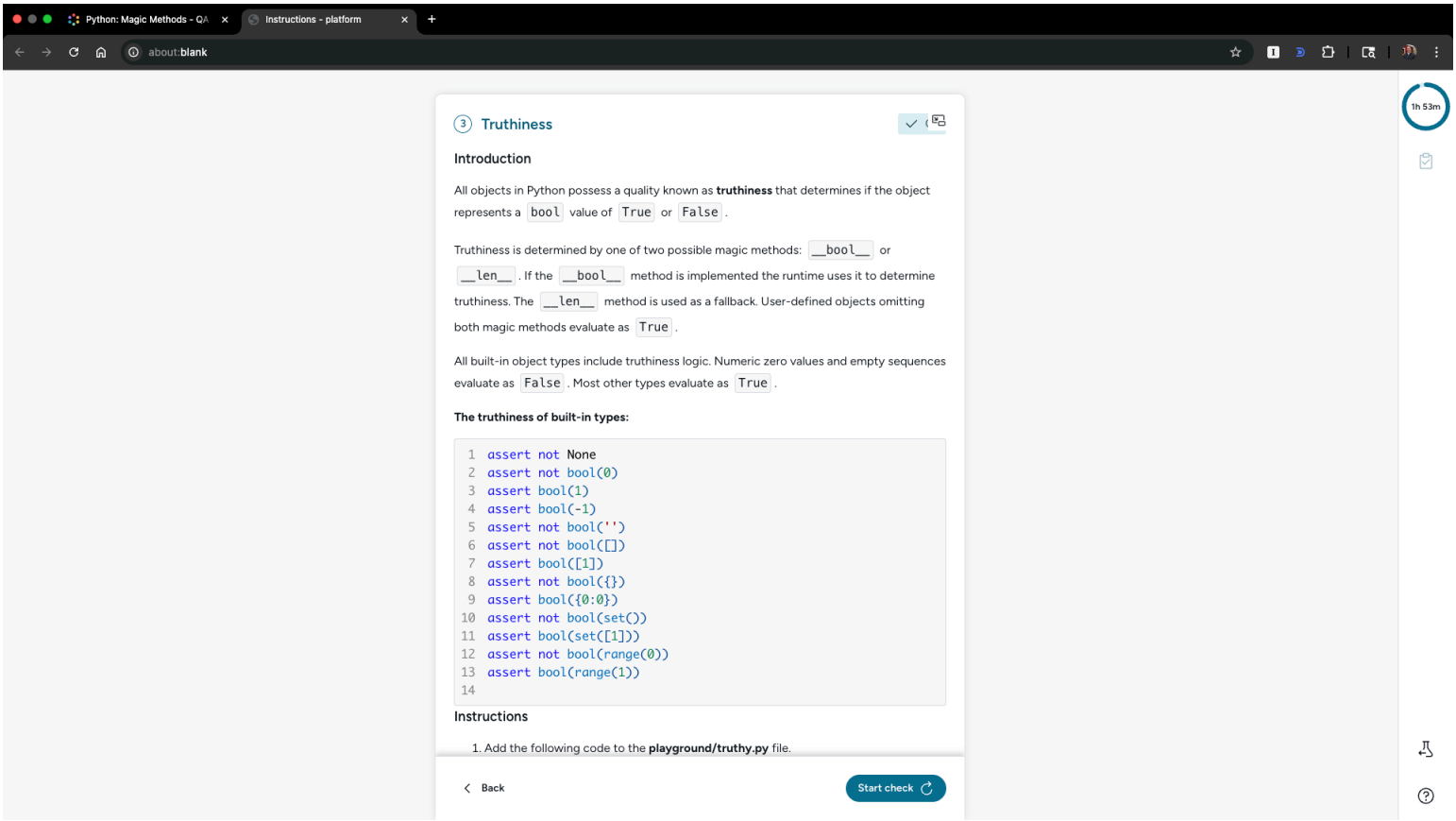
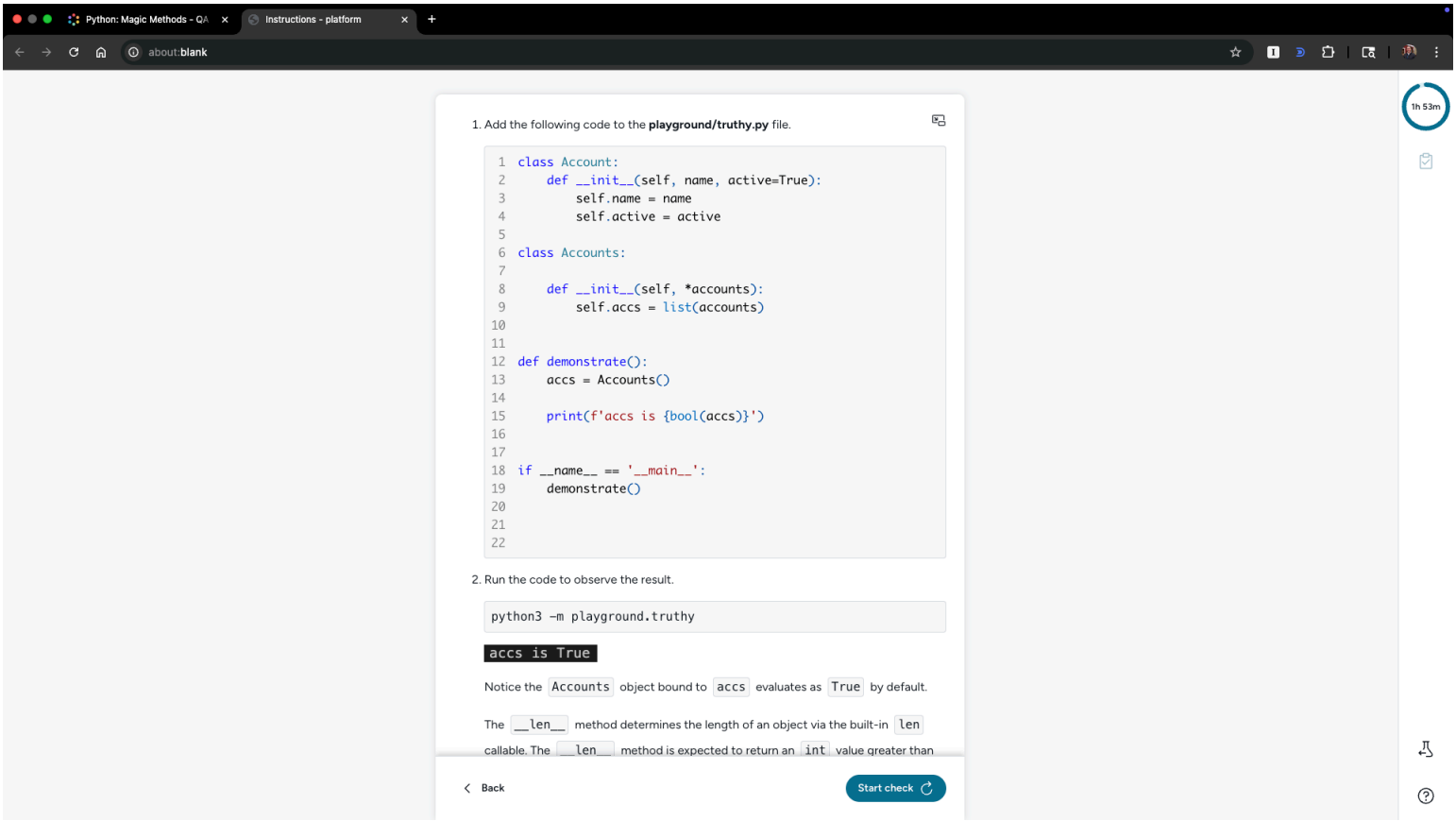

Python: Magic Methods - QA
Instructions - platform
about:blank
1h 53m
3 Truthiness
Introduction
All objects in Python possess a quality known as truthiness that determines if the object represents a bool value of True or False.
Truthiness is determined by one of two possible magic methods: __bool__ or __len__. If the __bool__ method is implemented the runtime uses it to determine truthiness. The __len__ method is used as a fallback. User-defined objects omitting both magic methods evaluate as True.
All built-in object types include truthiness logic. Numeric zero values and empty sequences evaluate as False. Most other types evaluate as True.
The truthiness of built-in types:
assert not None
assert not bool(0)
assert bool(1)
assert bool(-1)
assert not bool('')
assert not bool([])
assert bool([1])
assert not bool({})
assert bool({0:0})
assert not bool(set())
assert bool(set([1]))
assert not bool(range(0))
assert bool(range(1))
Instructions
1. Add the following code to the playground/truthy.py file.
Back
Start check


Python: Magic Methods - QA
Instructions - platform
about:blank
1h 53m
1. Add the following code to the playground/truthy.py file.
class Account:
def __init__(self, name, active=True):
self.name = name
self.active = active
class Accounts:
def __init__(self, *accounts):
self.accs = list(accounts)
def demonstrate():
accs = Accounts()
print(f'accs is {bool(accs)}')
if __name__ == '__main__':
demonstrate()
2. Run the code to observe the result.
python3 -m playground.truthy
accs is True
Notice the Accounts object bound to accs evaluates as True by default.
The __len__ method determines the length of an object via the built-in len
callable. The __len__ method is expected to return an int value greater than
Back
Start check

Python: Magic Methods - QA | Instructions - platform | about:blank

The `__len__` method determines the length of an object via the built-in `len` callable. The `__len__` method is expected to return an `int` value greater than or equal to zero.

The `__len__` method is used as a fallback for determining truthiness if the `__bool__` method is not implemented. A value of zero evaluates as `False` otherwise `True`.

3. Append the following method to the `Accounts` class.

```
    def __len__(self):
        print('called the __len__ method.')
        return len(self.accs)


```

```
class Accounts:

    def __init__(self, *accounts):
        self.accs = list(accounts)

    def __len__(self):
        print('called the __len__ method.')
        return len(self.accs)
```

4. Run the code to observe the result.

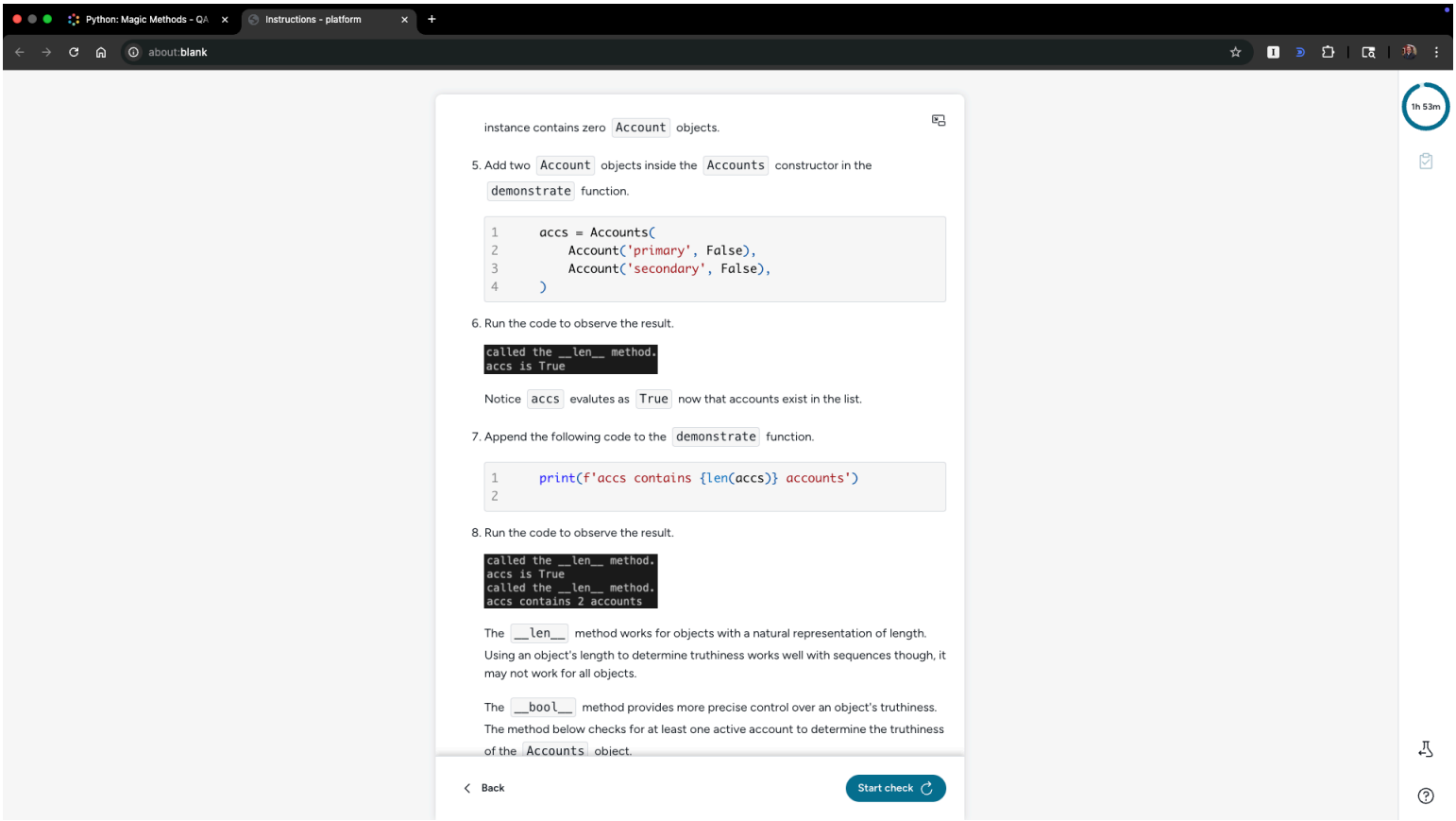
```
python3 -m playground.truthy
```

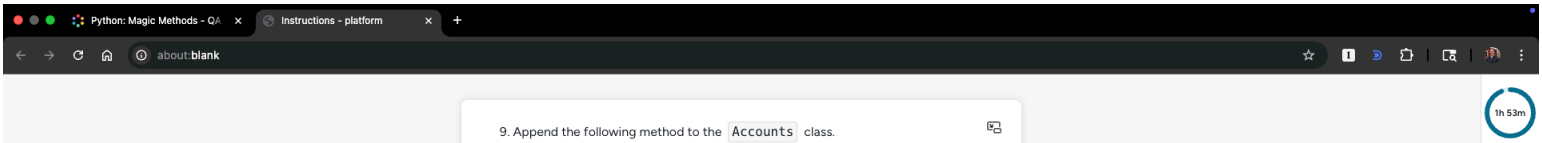
```
called the __len__ method.
accs is False
```

The object bound to `accs` now evaluates as `False` because the `Accounts` instance contains zero `Account` objects.

Back | Start check

1h 53m


Python: Magic Methods - QA
Instructions - platform
about:blank
1h 53m
instance contains zero Account objects.
5. Add two Account objects inside the Accounts constructor in the demonstrate function.
accs = Accounts(
Account('primary', False),
Account('secondary', False),
)
6. Run the code to observe the result.
called the __len__ method.
accs is True
Notice accs evalutes as True now that accounts exist in the list.
7. Append the following code to the demonstrate function.
print(f'accs contains {len(accs)} accounts')
8. Run the code to observe the result.
called the __len__ method.
accs is True
called the __len__ method.
accs contains 2 accounts
The __len__ method works for objects with a natural representation of length. Using an object's length to determine truthiness works well with sequences though, it may not work for all objects.
The __bool__ method provides more precise control over an object's truthiness. The method below checks for at least one active account to determine the truthiness of the Accounts object.
Back
Start check



9. Append the following method to the `Accounts` class.

```
    def __bool__(self):
        print('called the __bool__ method.')
        return any(a for a in self.accs if a.active)


```

10. Run the code to observe the result.

```
called the __bool__ method.
accs is False
called the __len__ method.
accs contains 2 accounts
```

With both the `__bool__` and `__len__` methods implemented the runtime uses the `__bool__` method to determine truthiness and the `__len__` method strictly for determining object length. The result is `False` because both accounts are inactive.

11. Make one account active.

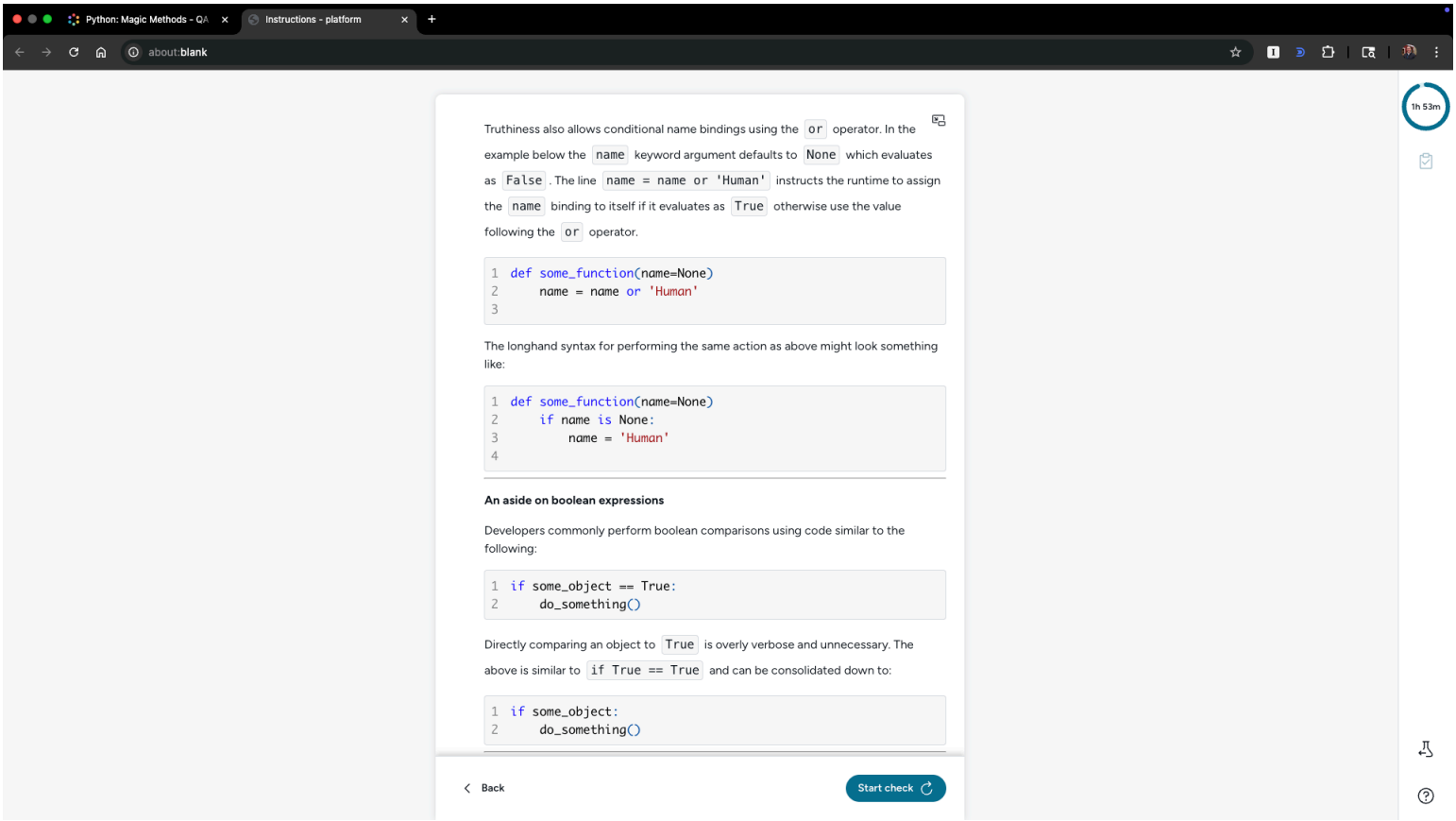
```
    accs = Accounts(
        Account('primary'),
        Account('secondary', False),
    )
```

12. Run the code to observe the result.

```
called the __bool__ method.
accs is True
called the __len__ method.
accs contains 2 accounts
```

Having finite control over an object's truthiness reduces the amount of code required to perform boolean operations.

Truthiness also allows conditional name bindings using the `or` operator. In the


Python: Magic Methods - QA
Instructions - platform
about:blank
Truthiness also allows conditional name bindings using the or operator. In the example below the name keyword argument defaults to None which evaluates as False. The line name = name or 'Human' instructs the runtime to assign the name binding to itself if it evaluates as True otherwise use the value following the or operator.
1 def some_function(name=None)
2 name = name or 'Human'
3
The longhand syntax for performing the same action as above might look something like:
1 def some_function(name=None)
2 if name is None:
3 name = 'Human'
4
An aside on boolean expressions
Developers commonly perform boolean comparisons using code similar to the following:
1 if some_object == True:
2 do_something()
Directly comparing an object to True is overly verbose and unnecessary. The above is similar to if True == True and can be consolidated down to:
1 if some_object:
2 do_something()
Back
Start check
1h 53m

Python: Magic Methods - QA | Instructions - platform | about:blank

1h 52m

13. Append the following code to the `demonstrate` function.

```
    if accs:
        print('at least one account is active')

```

14. Run the code to observe the result.

```
called the __bool__ method.
accs is True
called the __len__ method.
accs contains 2 accounts
called the __bool__ method.
at least one account is active
```

The `__len__` and `__bool__` methods allow developers to work with objects in a more Pythonic manner.

⭐ Proceed to the next step ⭐

✓ Validations

○ On-Track - Ensure The Account / Accounts Classes Are Fully Implemented

1. Ensure that you're on-track.

Python

Report an issue

④ Demonstrate Your Knowledge ✓ 0/1

Back | Start check