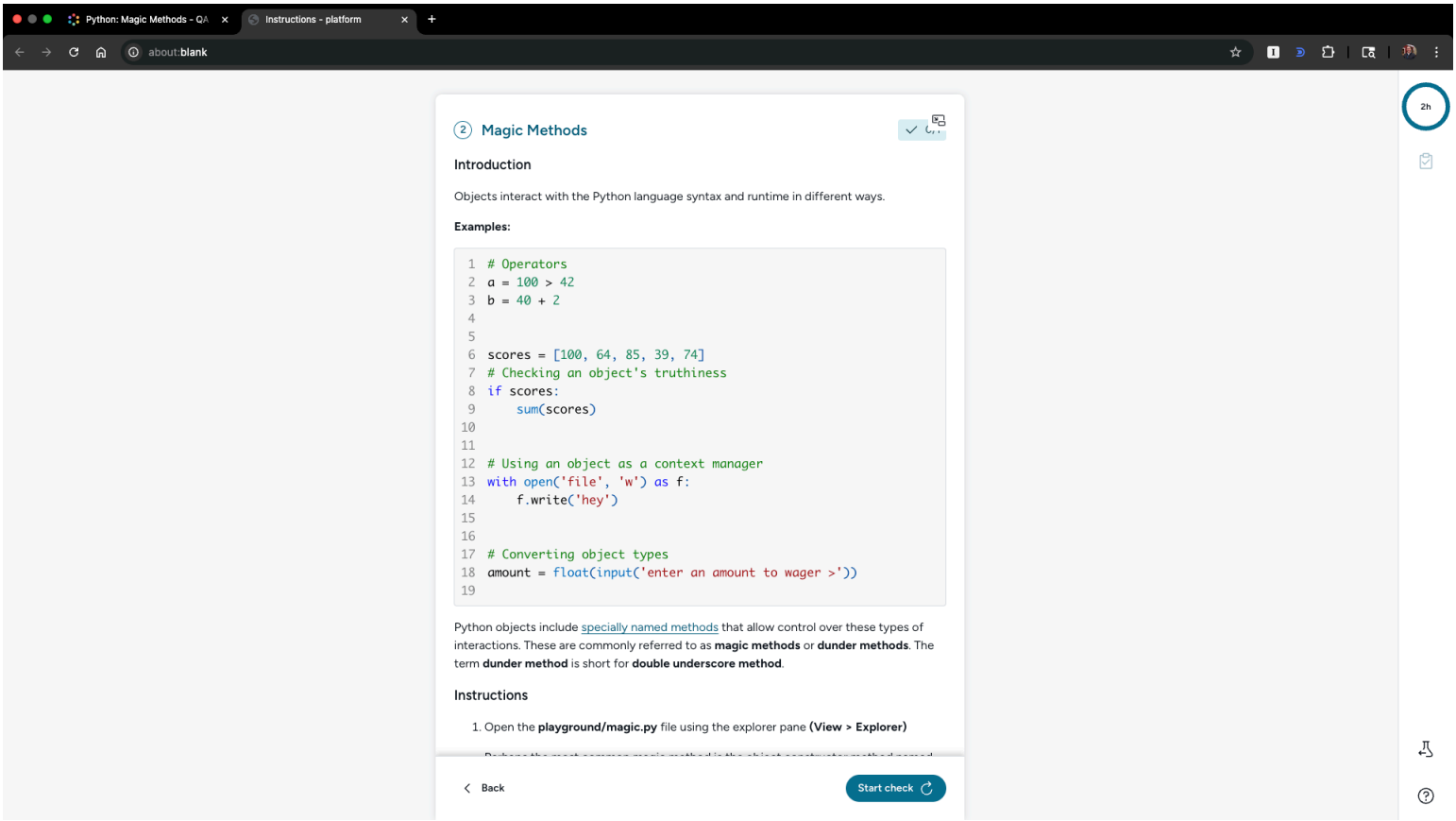

Python: Magic Methods - QA
Instructions - platform
about:blank
2h
2 Magic Methods
Introduction
Objects interact with the Python language syntax and runtime in different ways.
Examples:
# Operators
a = 100 > 42
b = 40 + 2
scores = [100, 64, 85, 39, 74]
# Checking an object's truthiness
if scores:
sum(scores)
# Using an object as a context manager
with open('file', 'w') as f:
f.write('hey')
# Converting object types
amount = float(input('enter an amount to wager >'))
Python objects include specially named methods that allow control over these types of interactions. These are commonly referred to as magic methods or dunder methods. The term dunder method is short for double underscore method.
Instructions
1. Open the playground/magic.py file using the explorer pane (View > Explorer)
Back
Start check

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

Instructions

1. Open the **playground/magic.py** file using the explorer pane **(View > Explorer)**

   Perhaps the most common magic method is the object constructor method named `__init__` . The constructor is responsible for performing object initialization. The `__init__` method is called by the runtime when an object is created.

2. Add the following code to the **playground/magic.py** file.

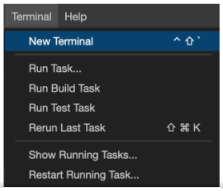
```
class Account:
    def __init__(self):
        print('new Account initialized')


def demonstrate():
    account_a = Account()


if __name__ == '__main__':
    demonstrate()

```

3. Open the built-in terminal pane **(Terminal > New Terminal)**.

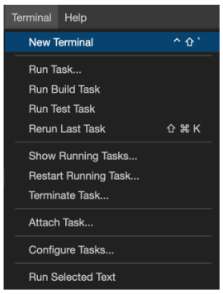


Back | Start check

2h

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

3. Open the built-in terminal pane **(Terminal > New Terminal)**.



4. Run the code to observe the result.

```
python3 -m playground.magic
```

```
new Account initialized
```

Some magic methods such as `__init__` allow for developer defined method parameters. Others such as the `__str__` method must use a fixed method signature.

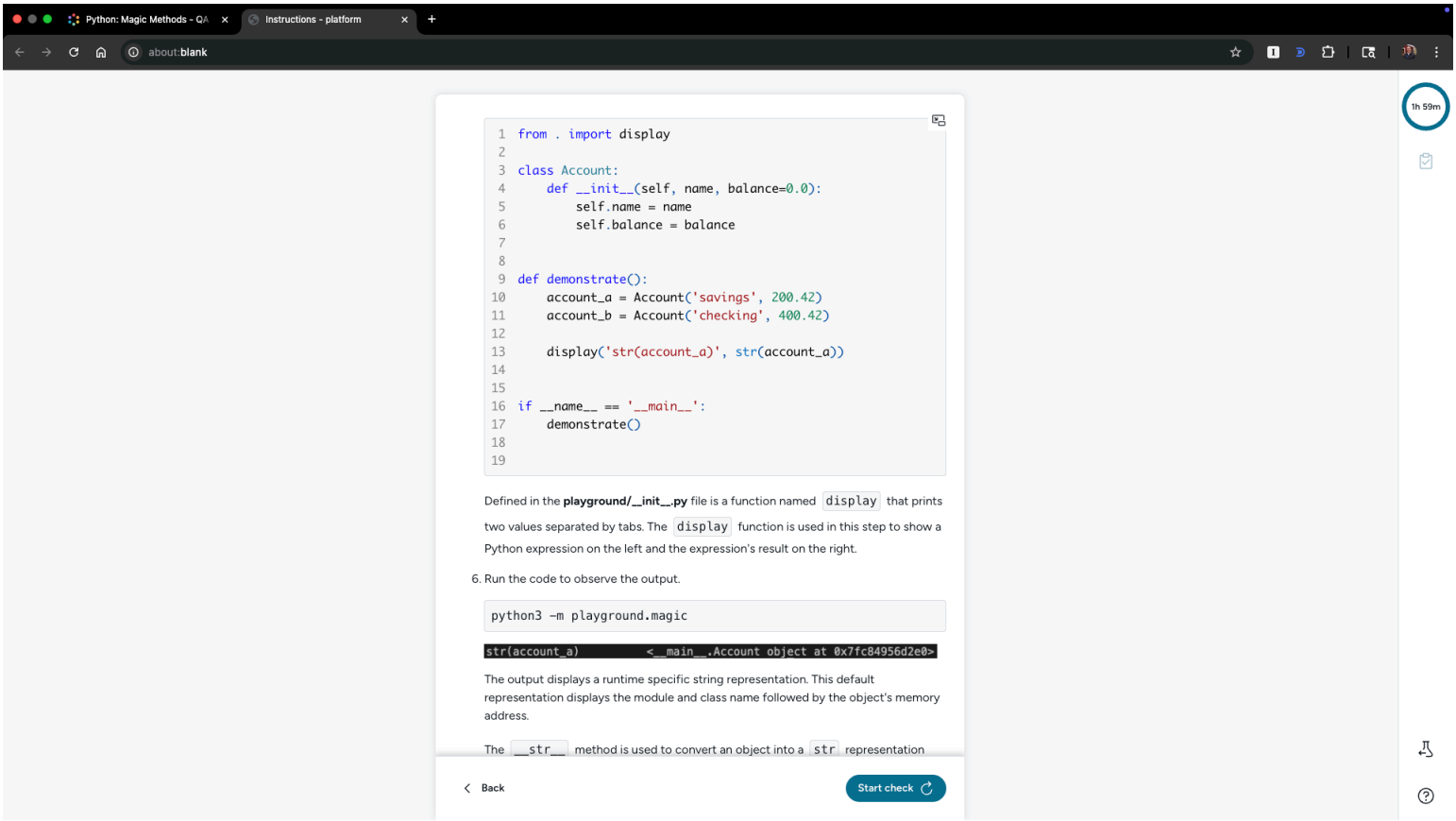
5. Replace the code in the **playground/magic.py** file with the following code.

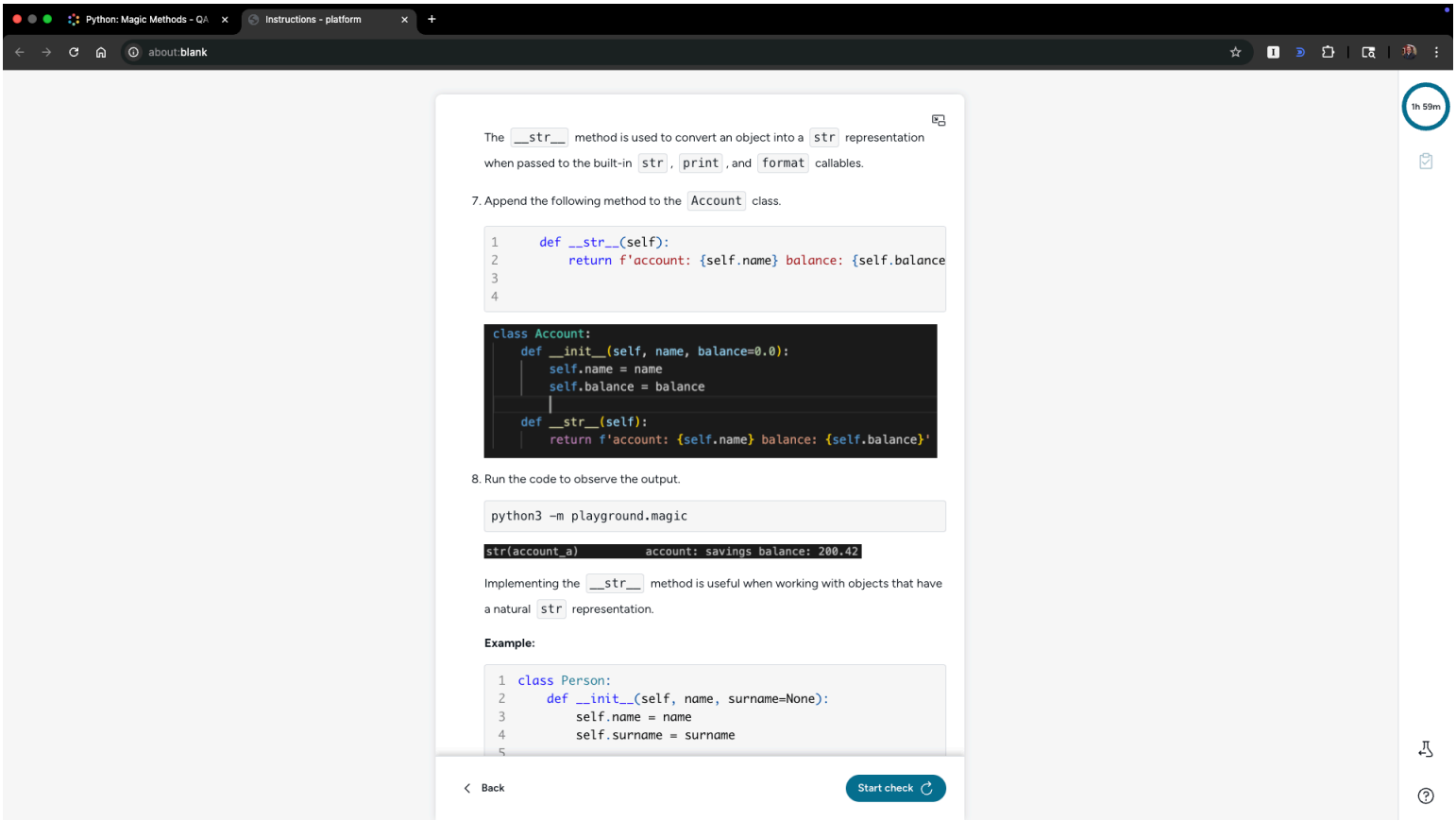
```
from . import display

class Account:
    def __init__(self, name, balance=0.0):
        self.name = name
        self.balance = balance
```

Back | Start check

2h


Python: Magic Methods - QA
Instructions - platform
about:blank
1h 59m
from . import display
class Account:
def __init__(self, name, balance=0.0):
self.name = name
self.balance = balance
def demonstrate():
account_a = Account('savings', 200.42)
account_b = Account('checking', 400.42)
display('str(account_a)', str(account_a))
if __name__ == '__main__':
demonstrate()
Defined in the playground/__init__.py file is a function named display that prints two values separated by tabs. The display function is used in this step to show a Python expression on the left and the expression's result on the right.
6. Run the code to observe the output.
python3 -m playground.magic
str(account_a) <__main__.Account object at 0x7fc84956d2e0>
The output displays a runtime specific string representation. This default representation displays the module and class name followed by the object's memory address.
The __str__ method is used to convert an object into a str representation
Back
Start check


Python: Magic Methods - QA
Instructions - platform
about:blank
1h 59m
The __str__ method is used to convert an object into a str representation when passed to the built-in str , print , and format callables.
7. Append the following method to the Account class.
1 def __str__(self):
2 return f'account: {self.name} balance: {self.balance
3
4
class Account:
def __init__(self, name, balance=0.0):
self.name = name
self.balance = balance
def __str__(self):
return f'account: {self.name} balance: {self.balance}'
8. Run the code to observe the output.
python3 -m playground.magic
str(account_a) account: savings balance: 200.42
Implementing the __str__ method is useful when working with objects that have a natural str representation.
Example:
1 class Person:
2 def __init__(self, name, surname=None):
3 self.name = name
4 self.surname = surname
5
Back
Start check

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

```
    def __str__(self):
        return f'{self.name} {self.surname or ""}'

p = Person('Ada', 'Lovelace')
print(p)
```

The `__str__` method is also useful for debugging using the built-in `print` callable.

The `__int__` and `__float__` methods are used to convert objects into `int` and `float` types.

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

```
    def __float__(self):
        return self.balance

    def __int__(self):
        return int(self.balance)


```

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

```
    display('int(account_a)', int(account_a))
    display('float(account_a)', float(account_a), indent=1)
```

```
def demonstrate():
    account_a = Account('savings', 200.42)
    account_b = Account('checking', 400.42)

    display('str(account_a)', str(account_a))
    display('int(account_a)', int(account_a))
    display('float(account_a)', float(account_a), indent=1)
```

Back | Start check

1h 59m

```
def demonstrate():
    account_a = Account('savings', 200.42)
    account_b = Account('checking', 400.42)

    display('str(account_a)', str(account_a))
    display('int(account_a)', int(account_a))
    display('float(account_a)', float(account_a), indent=1)
```

11. Run the code to observe the output.

```
str(account_a)          account: savings balance: 200.42
int(account_a)          200
float(account_a)        200.42
```

The `__repr__` method is similar to the `__str__` method except that `__repr__` is intended for developers. The `__repr__` method commonly displays a string representing the code required to recreate the object. Passing an object to the built-in `repr` function calls the object's `__repr__` method.

The `__repr__` method is called by other built-in callables. The `print` and `format` callables may call the `__repr__` method as a fallback if no `__str__` method exists.

12. Append the following method to the `Account` class.

```
    def __repr__(self):
        return f'{self.__class__.__name__}({self.balance})'


```

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

```
    display('repr(account_a)', repr(account_a))
```

14. Run the code to observe the output.

14. Run the code to observe the output.

```
str(account_a)          account: savings balance: 200.42
int(account_a)          200
float(account_a)        200.42
repr(account_a)         Account(200.42)
```

Notice the displayed string represents the code required to instantiate the class. If possible use `__repr__` to return the code required to reproduce an object. Otherwise return a description of the object that would be useful for debugging.

Comparison methods allow objects to be compared using operators. Implementing these methods allow objects to be compared using a more natural syntax.

15. Append the following method to the `Account` class.

```
    def __eq__(self, other):
        return self.balance == other.balance

    def __lt__(self, other):
        return self.balance < other.balance

    def __le__(self, other):
        return self.balance <= other.balance

    def __gt__(self, other):
        return self.balance > other.balance

    def __ge__(self, other):
        return self.balance >= other.balance


```

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

```
    display('account_a > account_b', account_a > account_b,
```

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

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

```
    display('account_a > account_b', account_a > account_b,
    display('account_a < account_b', account_a < account_b,
    display('account_a >= account_b', account_a >= account_b
    display('account_a <= account_b', account_a <= account_b
    display('account_a == account_b', account_a == account_b
    display('account_a != account_b', account_a != account_b
```

17. Run the code to observe the output.

```
str(account_a)          account: savings balance: 200.42
int(account_a)          200
float(account_a)        200.42
repr(account_a)         Account(200.42)
account_a > account_b   False
account_a < account_b   True
account_a >= account_b  False
account_a <= account_b  True
account_a == account_b  False
account_a != account_b  True
```

The above code uses numeric comparisons however, these comparison methods are not limited to numbers. These methods are intended to allow objects to be compared in a natural manner. The below example implements the equality method to compare `str` values.

Example:

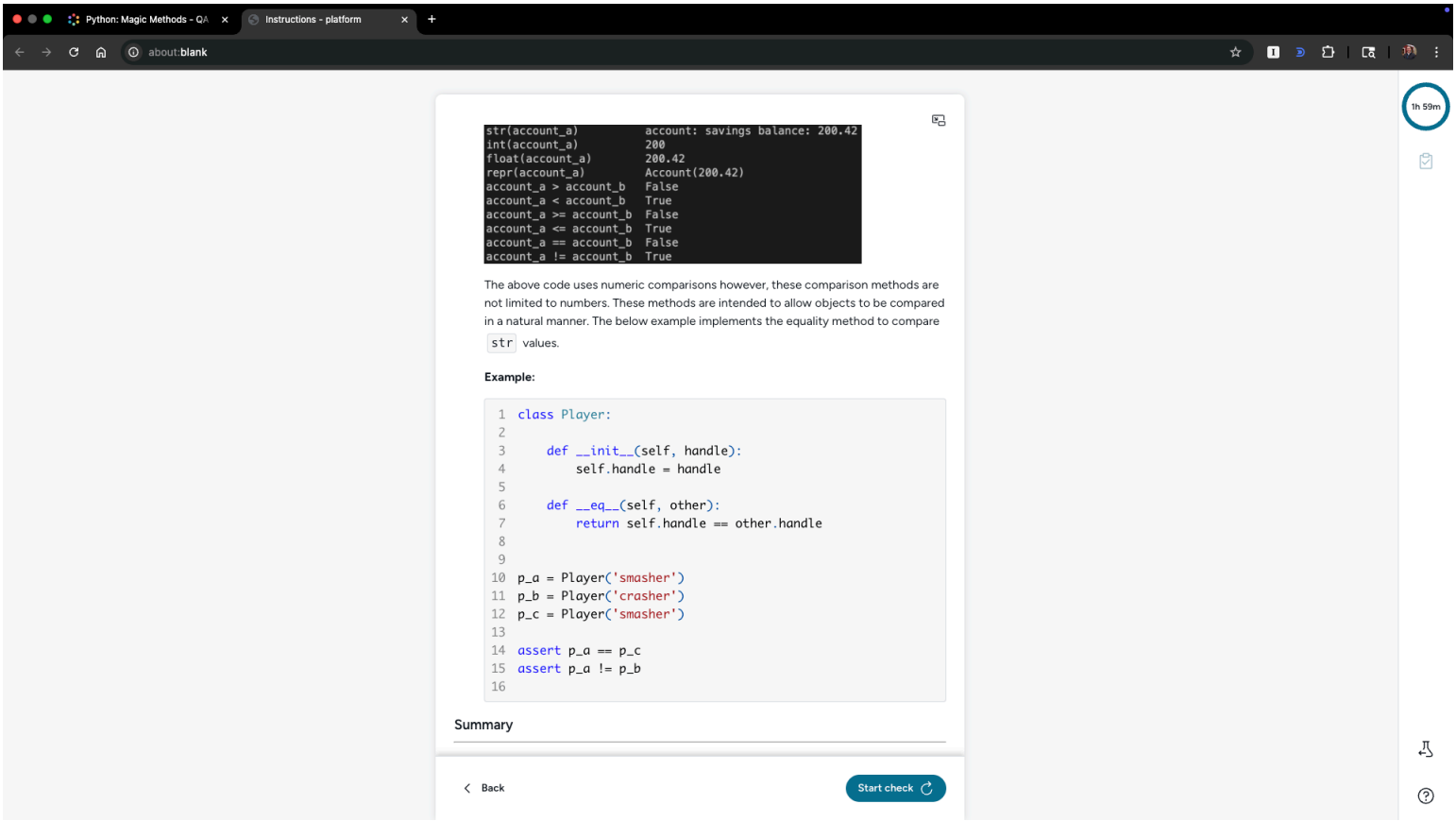
```
class Player:

    def __init__(self, handle):
        self.handle = handle

    def __eq__(self, other):
        return self.handle == other.handle
```

Back | Start check

1h 59m


Python: Magic Methods - QA
Instructions - platform
about:blank
str(account_a) account: savings balance: 200.42
int(account_a) 200
float(account_a) 200.42
repr(account_a) Account(200.42)
account_a > account_b False
account_a < account_b True
account_a >= account_b False
account_a <= account_b True
account_a == account_b False
account_a != account_b True
The above code uses numeric comparisons however, these comparison methods are not limited to numbers. These methods are intended to allow objects to be compared in a natural manner. The below example implements the equality method to compare str values.
Example:
class Player:
def __init__(self, handle):
self.handle = handle
def __eq__(self, other):
return self.handle == other.handle
p_a = Player('smasher')
p_b = Player('crasher')
p_c = Player('smasher')
assert p_a == p_c
assert p_a != p_b
Summary
Back
Start check
1h 59m