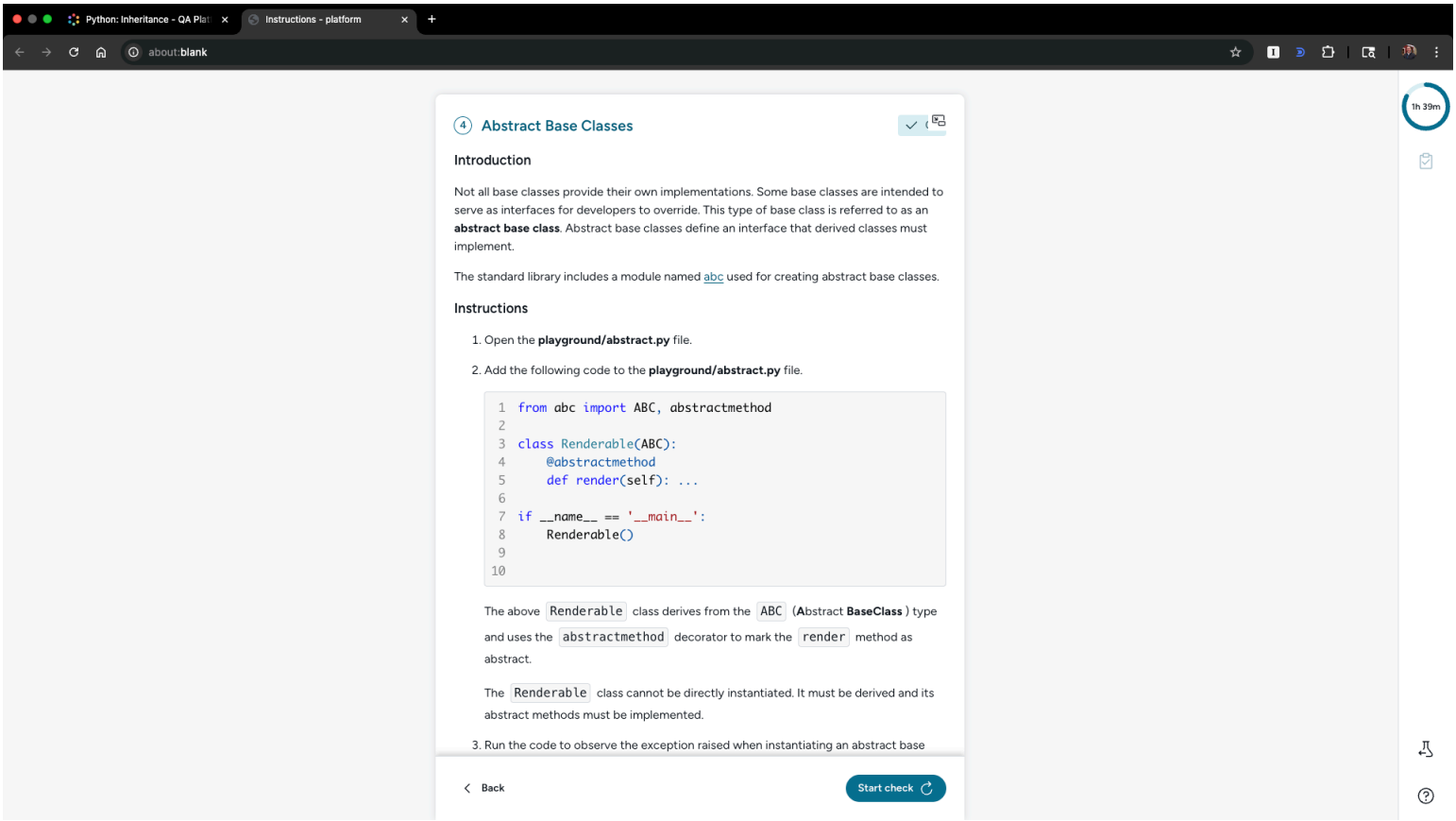
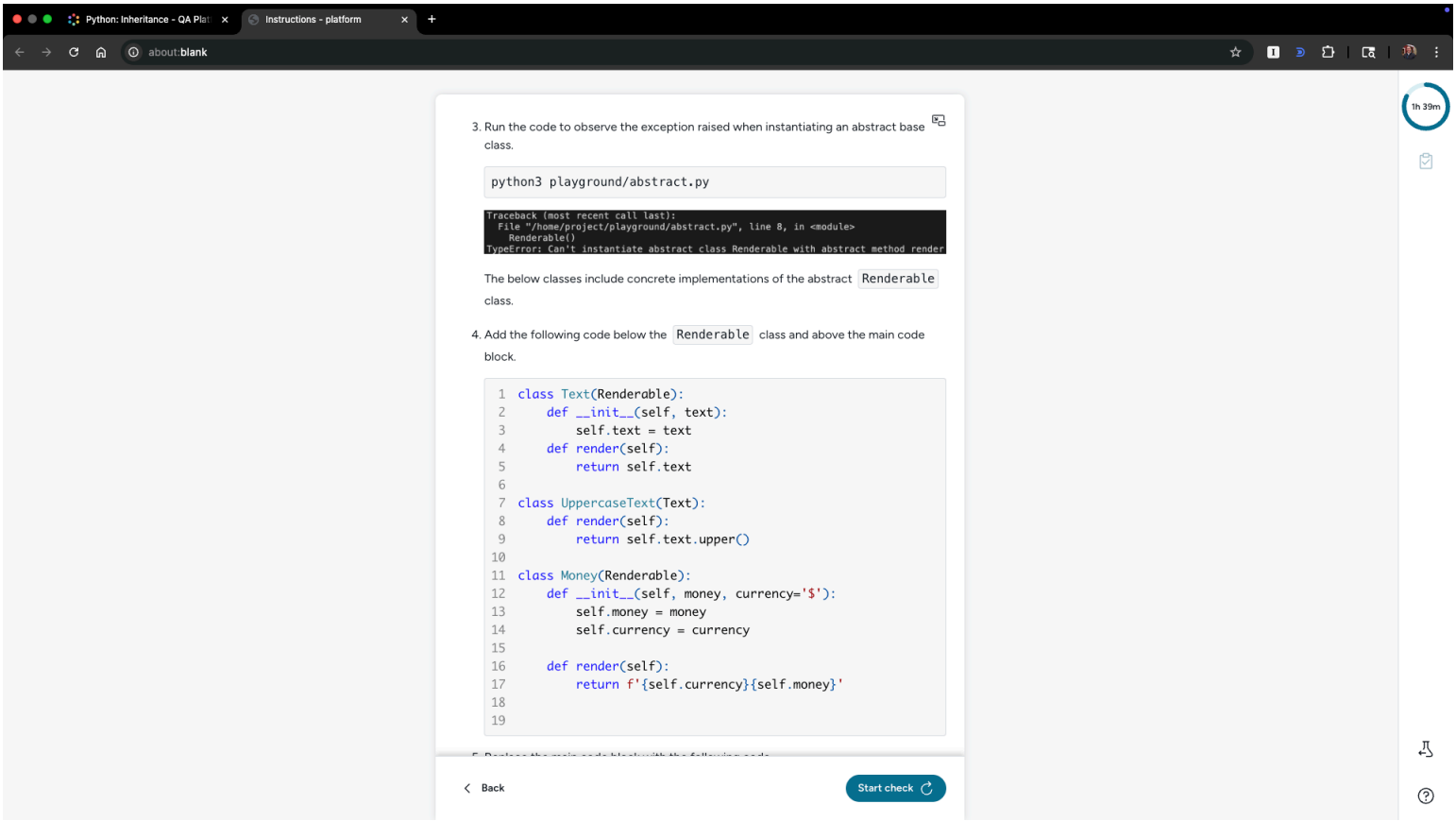
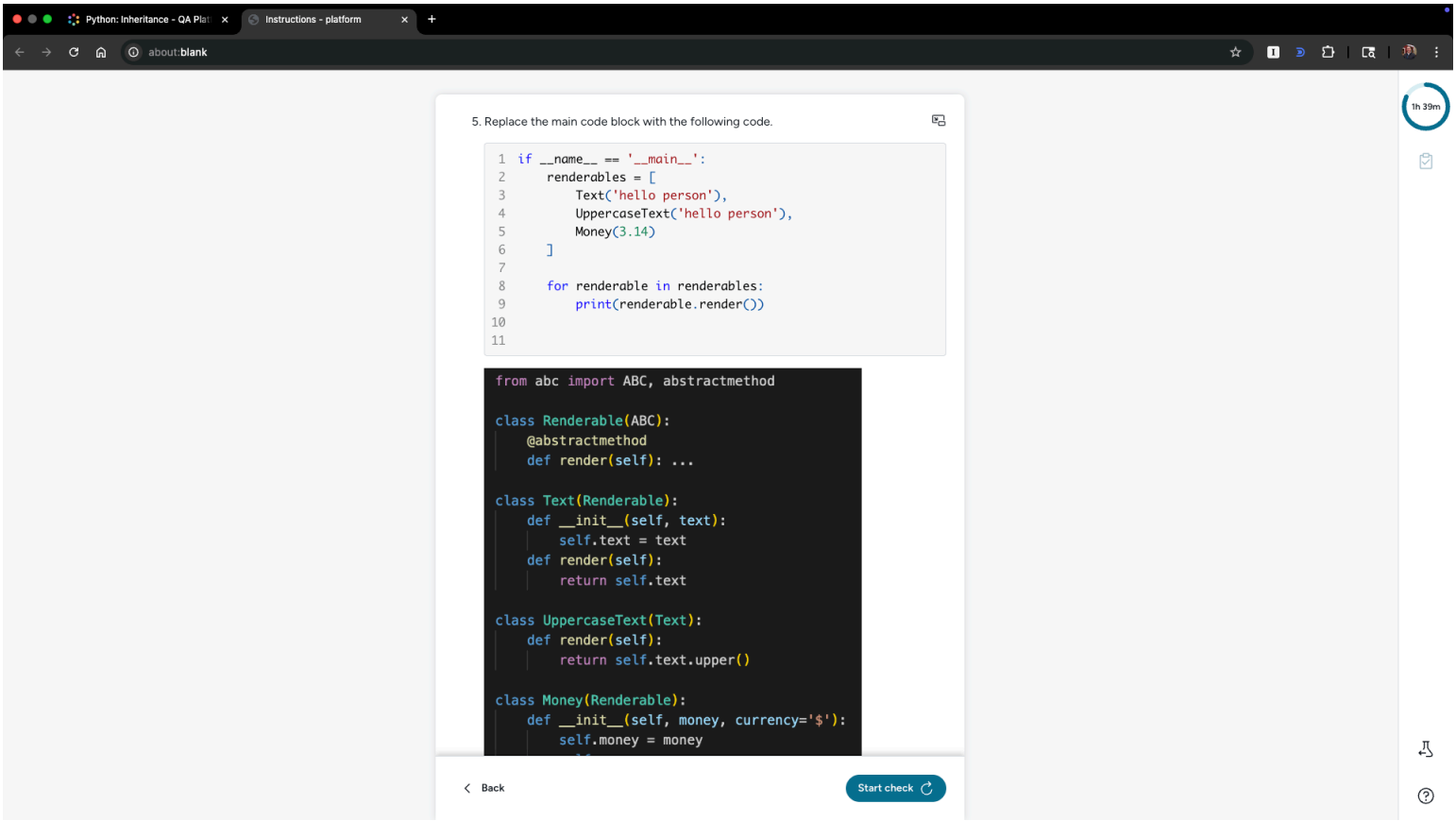
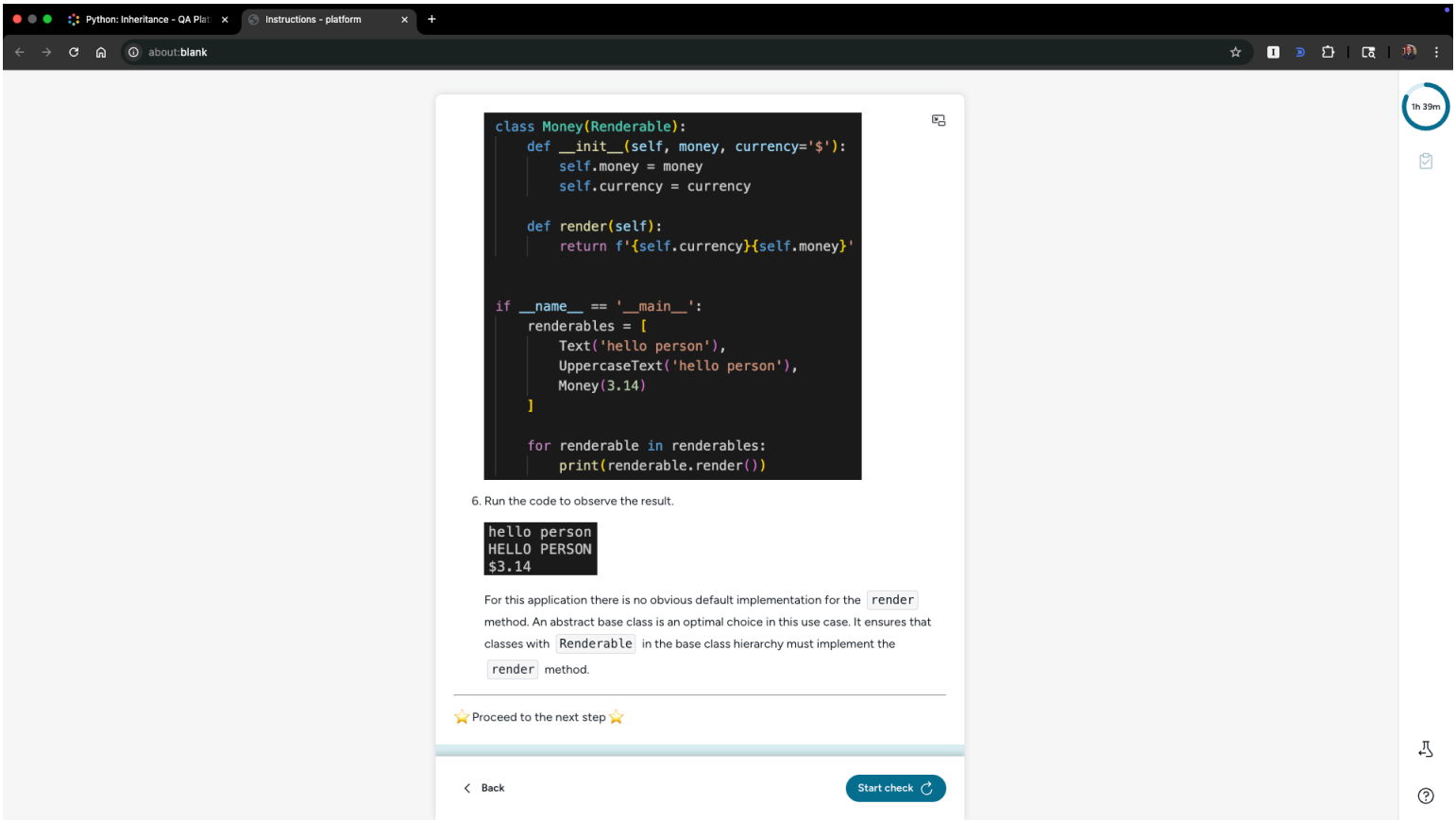

Python: Inheritance - QA Plat
Instructions - platform
about:blank
1h 39m
4 Abstract Base Classes
Introduction
Not all base classes provide their own implementations. Some base classes are intended to serve as interfaces for developers to override. This type of base class is referred to as an abstract base class. Abstract base classes define an interface that derived classes must implement.
The standard library includes a module named abc used for creating abstract base classes.
Instructions
1. Open the playground/abstract.py file.
2. Add the following code to the playground/abstract.py file.
from abc import ABC, abstractmethod
class Renderable(ABC):
@abstractmethod
def render(self): ...
if __name__ == '__main__':
Renderable()
The above Renderable class derives from the ABC (Abstract BaseClass) type and uses the abstractmethod decorator to mark the render method as abstract.
The Renderable class cannot be directly instantiated. It must be derived and its abstract methods must be implemented.
3. Run the code to observe the exception raised when instantiating an abstract base
Back
Start check


Python: Inheritance - QA Plat
Instructions - platform
about:blank
1h 39m
3. Run the code to observe the exception raised when instantiating an abstract base class.
python3 playground/abstract.py
Traceback (most recent call last):
File "/home/project/playground/abstract.py", line 8, in <module>
Renderable()
TypeError: Can't instantiate abstract class Renderable with abstract method render
The below classes include concrete implementations of the abstract Renderable class.
4. Add the following code below the Renderable class and above the main code block.
class Text(Renderable):
def __init__(self, text):
self.text = text
def render(self):
return self.text
class UppercaseText(Text):
def render(self):
return self.text.upper()
class Money(Renderable):
def __init__(self, money, currency='$'):
self.money = money
self.currency = currency
def render(self):
return f'{self.currency}{self.money}'
Back
Start check


Python: Inheritance - QA Plat
Instructions - platform
about:blank
5. Replace the main code block with the following code.
1 if __name__ == '__main__':
2 renderables = [
3 Text('hello person'),
4 UppercaseText('hello person'),
5 Money(3.14)
6]
7
8 for renderable in renderables:
9 print(renderable.render())
10
11
from abc import ABC, abstractmethod
class Renderable(ABC):
@abstractmethod
def render(self): ...
class Text(Renderable):
def __init__(self, text):
self.text = text
def render(self):
return self.text
class UppercaseText(Text):
def render(self):
return self.text.upper()
class Money(Renderable):
def __init__(self, money, currency='$'):
self.money = money
Back
Start check
1h 39m


Python: Inheritance - QA Plat
Instructions - platform
about:blank
class Money(Renderable):
def __init__(self, money, currency='$'):
self.money = money
self.currency = currency
def render(self):
return f'{self.currency}{self.money}'
if __name__ == '__main__':
renderables = [
Text('hello person'),
UppercaseText('hello person'),
Money(3.14)
]
for renderable in renderables:
print(renderable.render())
6. Run the code to observe the result.
hello person
HELLO PERSON
$3.14
For this application there is no obvious default implementation for the render method. An abstract base class is an optimal choice in this use case. It ensures that classes with Renderable in the base class hierarchy must implement the render method.
Proceed to the next step
Back
Start check
1h 39m