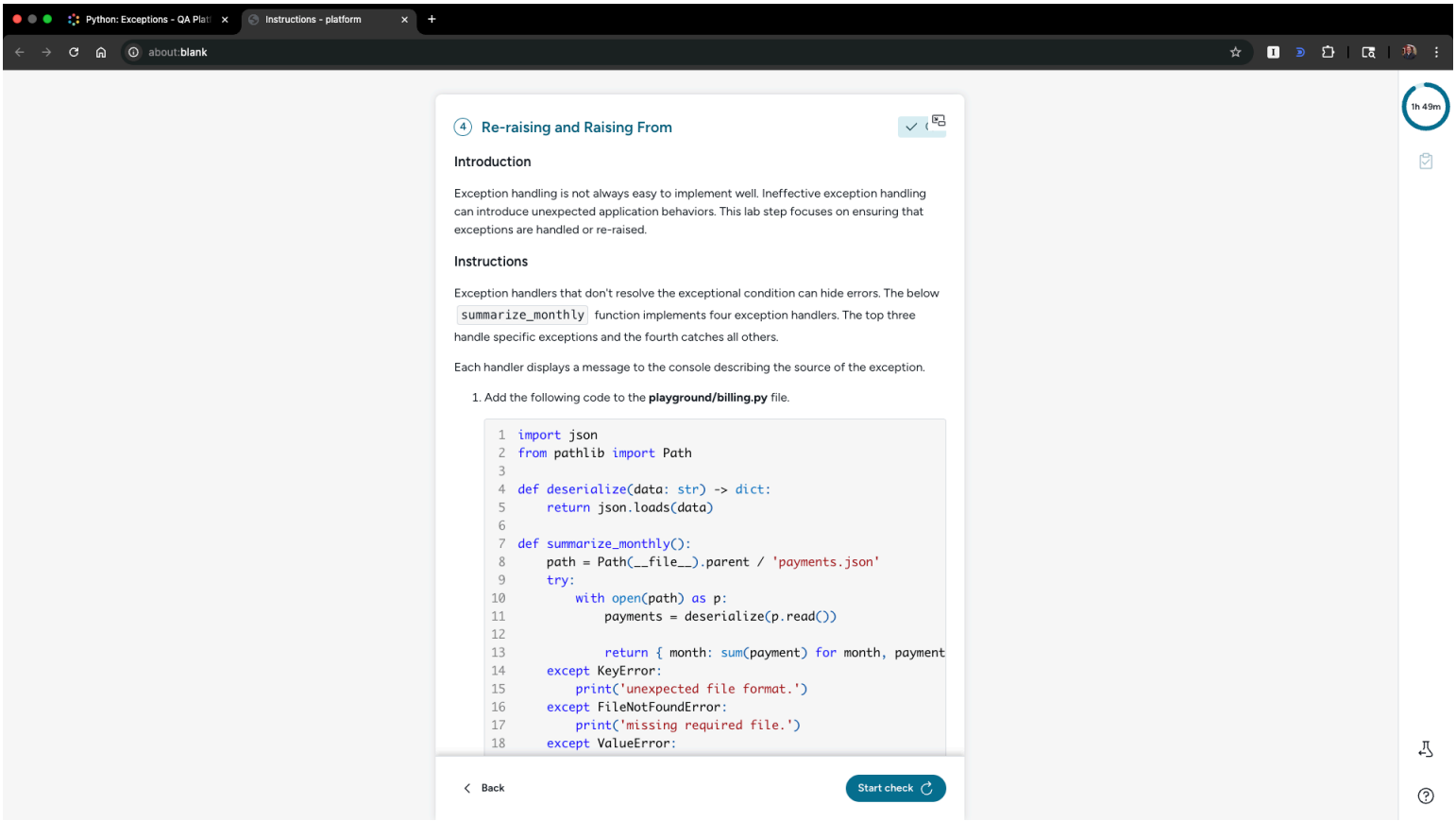
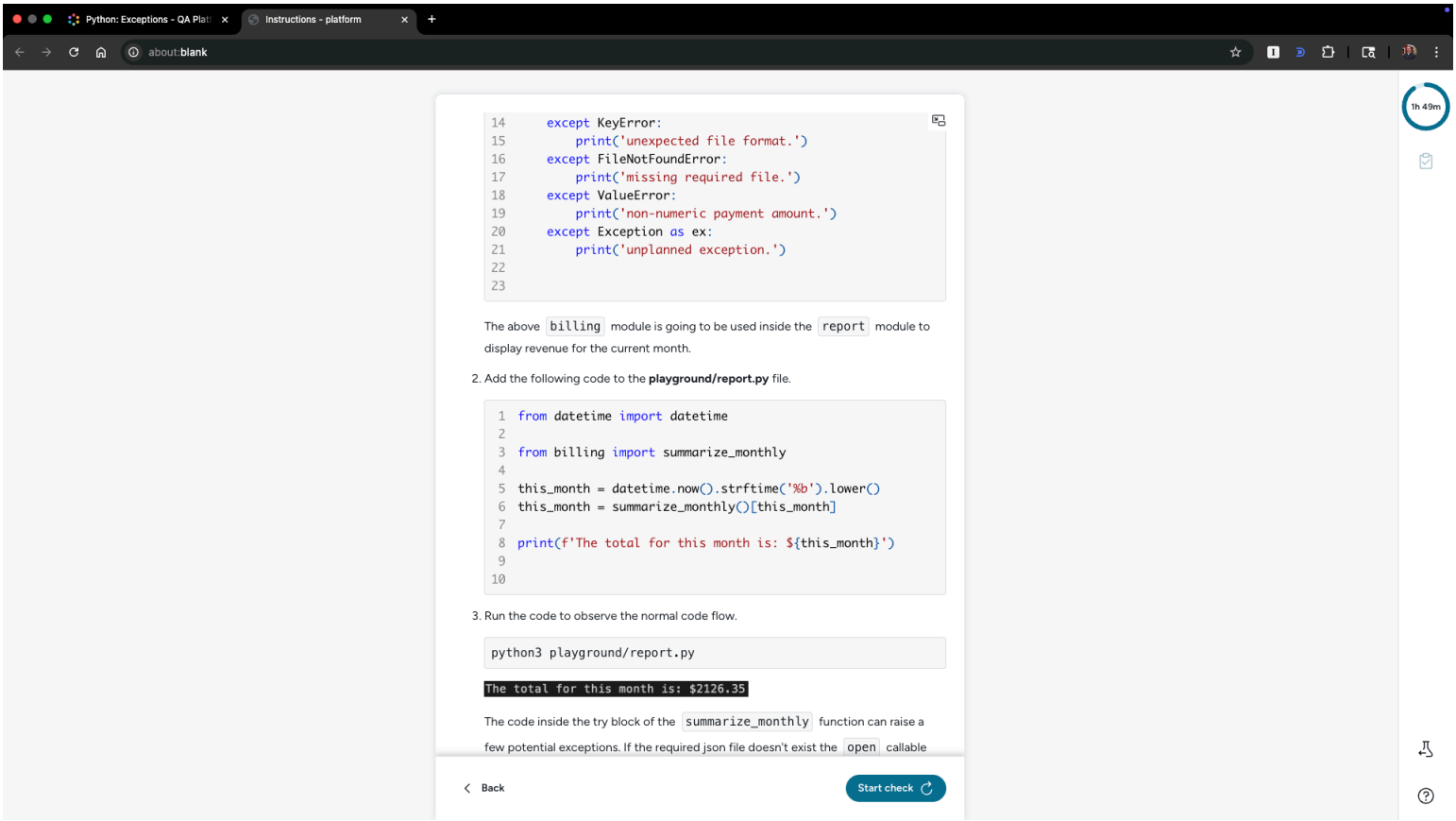

Python: Exceptions - QA Platf
Instructions - platform
about:blank
1h 49m
4 Re-raising and Raising From
Introduction
Exception handling is not always easy to implement well. Ineffective exception handling can introduce unexpected application behaviors. This lab step focuses on ensuring that exceptions are handled or re-raised.
Instructions
Exception handlers that don't resolve the exceptional condition can hide errors. The below summarize_monthly function implements four exception handlers. The top three handle specific exceptions and the fourth catches all others.
Each handler displays a message to the console describing the source of the exception.
1. Add the following code to the playground/billing.py file.
import json
from pathlib import Path
def deserialize(data: str) -> dict:
return json.loads(data)
def summarize_monthly():
path = Path(__file__).parent / 'payments.json'
try:
with open(path) as p:
payments = deserialize(p.read())
return { month: sum(payment) for month, payment
except KeyError:
print('unexpected file format.')
except FileNotFoundError:
print('missing required file.')
except ValueError:
Back
Start check


Python: Exceptions - QA Platf
Instructions - platform
about:blank
1h 49m
14 except KeyError:
15 print('unexpected file format.')
16 except FileNotFoundError:
17 print('missing required file.')
18 except ValueError:
19 print('non-numeric payment amount.')
20 except Exception as ex:
21 print('unplanned exception.')
22
23
The above billing module is going to be used inside the report module to display revenue for the current month.
2. Add the following code to the playground/report.py file.
1 from datetime import datetime
2
3 from billing import summarize_monthly
4
5 this_month = datetime.now().strftime('%b').lower()
6 this_month = summarize_monthly()[this_month]
7
8 print(f'The total for this month is: ${this_month}')
9
10
3. Run the code to observe the normal code flow.
python3 playground/report.py
The total for this month is: $2126.35
The code inside the try block of the summarize_monthly function can raise a few potential exceptions. If the required json file doesn't exist the open callable
Back
Start check

few potential exceptions. If the required json file doesn't exist the `open` callable raises a `FileNotFoundError` exception. If the deserialized json data doesn't contain a key named *months* a `KeyError` is raised. A `ValueError` exception is raised if the `payment` list contains a non-numeric value. The final `Exception` handler catches all other exception types.

Changing the structure of the json file will result in a `KeyError`.

4. Edit the **playground/payments.json** file.
   - Change the key *months* to *m* to cause a `KeyError`.

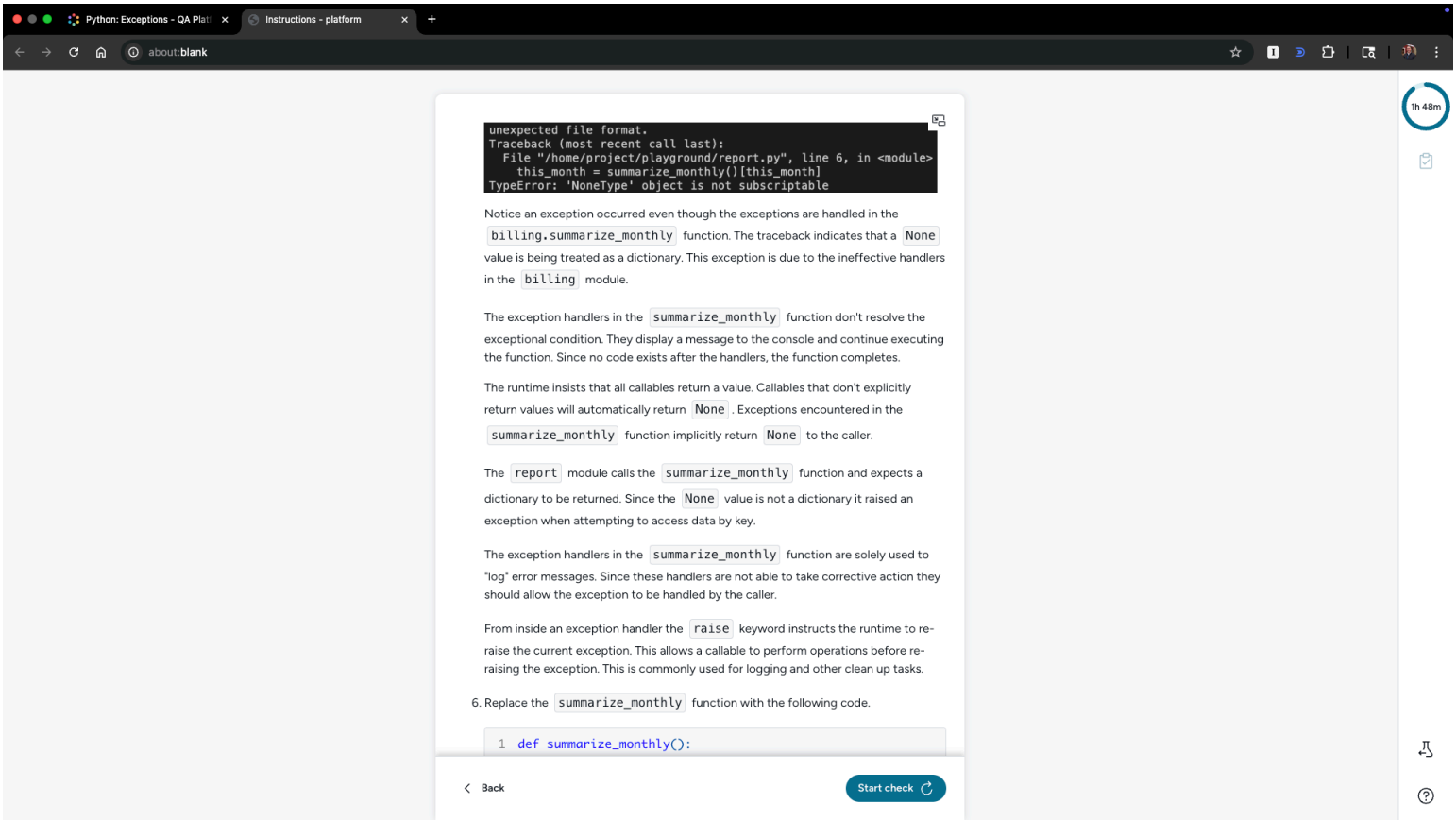
playground > payments.json

```
{
    "m": {
        "jan": [500.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "feb": [367.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "mar": [542.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "apr": [137.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "may": [753.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "jun": [794.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "jul": [125.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "aug": [974.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "sep": [372.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "oct": [278.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "nov": [384.14, 951.93, 74.50, 130.26, 130.26, 130.26],
        "dec": [709.14, 951.93, 74.50, 130.26, 130.26, 130.26]
    }
}
```

5. Run the code.

```
python3 playground/report.py
```

```
unexpected file format.
Traceback (most recent call last):
  File "/home/project/playground/report.py", line 6, in <module>
```


Python: Exceptions - QA Platf
Instructions - platform
about:blank
unexpected file format.
Traceback (most recent call last):
File "/home/project/playground/report.py", line 6, in <module>
this_month = summarize_monthly()[this_month]
TypeError: 'NoneType' object is not subscriptable
Notice an exception occurred even though the exceptions are handled in the billing.summarize_monthly function. The traceback indicates that a None value is being treated as a dictionary. This exception is due to the ineffective handlers in the billing module.
The exception handlers in the summarize_monthly function don't resolve the exceptional condition. They display a message to the console and continue executing the function. Since no code exists after the handlers, the function completes.
The runtime insists that all callables return a value. Callables that don't explicitly return values will automatically return None . Exceptions encountered in the summarize_monthly function implicitly return None to the caller.
The report module calls the summarize_monthly function and expects a dictionary to be returned. Since the None value is not a dictionary it raised an exception when attempting to access data by key.
The exception handlers in the summarize_monthly function are solely used to "log" error messages. Since these handlers are not able to take corrective action they should allow the exception to be handled by the caller.
From inside an exception handler the raise keyword instructs the runtime to re-raise the current exception. This allows a callable to perform operations before re-raising the exception. This is commonly used for logging and other clean up tasks.
6. Replace the summarize_monthly function with the following code.
1 def summarize_monthly():
Back
Start check
1h 48m

Python: Exceptions - QA Platf | Instructions - platform | about:blank

6. Replace the `summarize_monthly` function with the following code.

```
def summarize_monthly():
    path = Path(__file__).parent / 'payments.json'
    try:
        with open(path) as p:
            payments = deserialize(p.read())

            return { month: sum(payment) for month, payment
    except KeyError:
        print('unexpected file format.')
        raise
    except FileNotFoundError:
        print('missing required file.')
        raise
    except ValueError:
        print('non-numeric payment amount.')
        raise
    except Exception as ex:
        print('unplanned exception.')
        raise

```

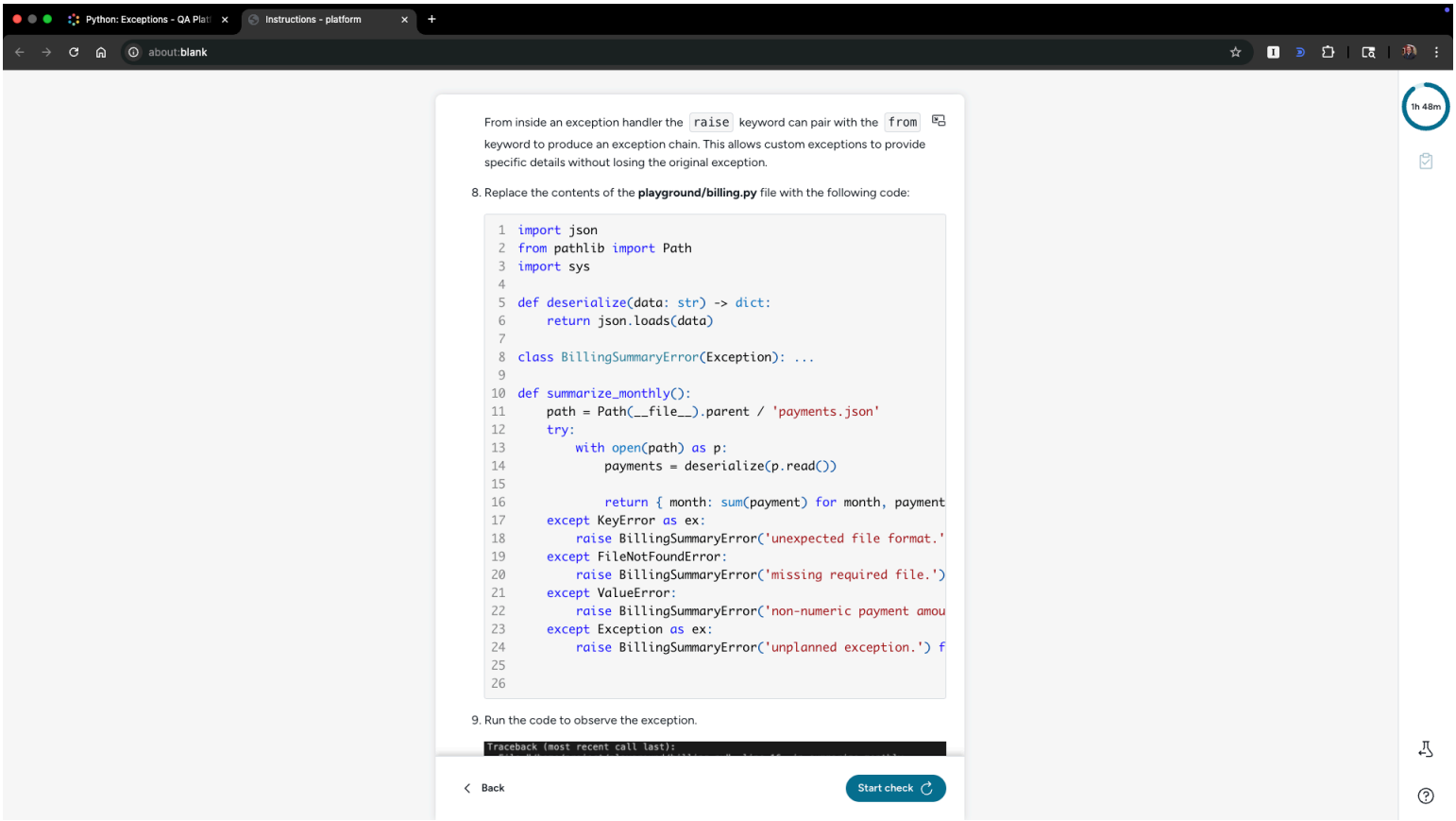
7. Run the code to observe the exception.

```
unexpected file format.
Traceback (most recent call last):
  File "/home/project/playground/report.py", line 6, in <module>
    this_month = summarize_monthly()[this_month]
  File "/home/project/playground/billing.py", line 13, in summarize_monthly
    return { month: sum(payment) for month, payment in payments['months'].items() }
KeyError: 'months'
```

Notice the error message is logged before the exception is re-raised to the caller. Since the caller (the `report` module) doesn't handle the exception it displays the traceback and exits.

Back | Start check

1h 48m


Python: Exceptions - QA Platf
Instructions - platform
about:blank
1h 48m
From inside an exception handler the raise keyword can pair with the from keyword to produce an exception chain. This allows custom exceptions to provide specific details without losing the original exception.
8. Replace the contents of the playground/billing.py file with the following code:
import json
from pathlib import Path
import sys
def deserialize(data: str) -> dict:
return json.loads(data)
class BillingSummaryError(Exception): ...
def summarize_monthly():
path = Path(__file__).parent / 'payments.json'
try:
with open(path) as p:
payments = deserialize(p.read())
return { month: sum(payment) for month, payment
except KeyError as ex:
raise BillingSummaryError('unexpected file format.'
except FileNotFoundError:
raise BillingSummaryError('missing required file.')
except ValueError:
raise BillingSummaryError('non-numeric payment amou
except Exception as ex:
raise BillingSummaryError('unplanned exception.') f
9. Run the code to observe the exception.
Traceback (most recent call last):
Back
Start check

9. Run the code to observe the exception.

```
Traceback (most recent call last):
  File "/home/project/playground/billing.py", line 16, in summarize_monthly
    return { month: sum(payment) for month, payment in payments['months'].items() }
KeyError: 'months'

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/home/project/playground/report.py", line 6, in <module>
    this_month = summarize_monthly()[this_month]
  File "/home/project/playground/billing.py", line 18, in summarize_monthly
    raise BillingSummaryError('unexpected file format.') from ex
billing.BillingSummaryError: unexpected file format.
```

Notice that the exception at the bottom displays the custom exception with a specific error message. Above is the exception directly responsible for the `BillingSummaryError`.

Not all exceptions require the original traceback to be chained. For example, third-party modules commonly replace lower-level exceptions with higher-level exceptions. An example can be found in the source code for the Flask web application framework.

Passing a `None` value to the `from` keyword causes the original exception to be suppressed.

10. Replace the `summarize_monthly` function with the following:

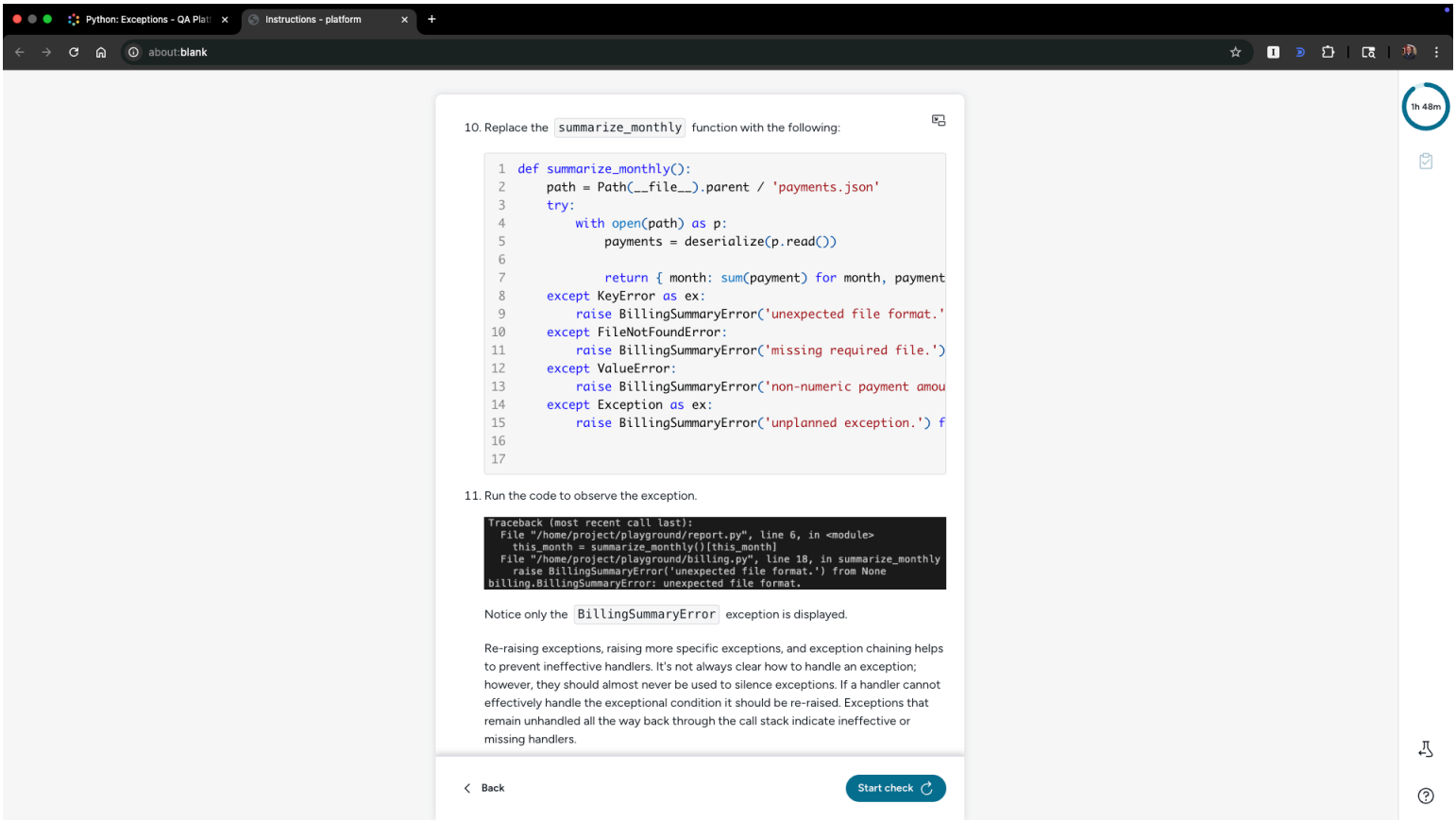
```
def summarize_monthly():
    path = Path(__file__).parent / 'payments.json'
    try:
        with open(path) as p:
            payments = deserialize(p.read())

            return { month: sum(payment) for month, payment
    except KeyError as ex:
        raise BillingSummaryError('unexpected file format.'
    except FileNotFoundError:
        raise BillingSummaryError('missing required file.')
    except ValueError:
```

Back

Start check

1h 48m


Python: Exceptions - QA Platf
Instructions - platform
about:blank
1h 48m
10. Replace the summarize_monthly function with the following:
def summarize_monthly():
path = Path(__file__).parent / 'payments.json'
try:
with open(path) as p:
payments = deserialize(p.read())
return { month: sum(payment) for month, payment
except KeyError as ex:
raise BillingSummaryError('unexpected file format.'
except FileNotFoundError:
raise BillingSummaryError('missing required file.')
except ValueError:
raise BillingSummaryError('non-numeric payment amou
except Exception as ex:
raise BillingSummaryError('unplanned exception.') f
11. Run the code to observe the exception.
Traceback (most recent call last):
File "/home/project/playground/report.py", line 6, in <module>
this_month = summarize_monthly()[this_month]
File "/home/project/playground/billing.py", line 18, in summarize_monthly
raise BillingSummaryError('unexpected file format.') from None
billing.BillingSummaryError: unexpected file format.
Notice only the BillingSummaryError exception is displayed.
Re-raising exceptions, raising more specific exceptions, and exception chaining helps to prevent ineffective handlers. It's not always clear how to handle an exception; however, they should almost never be used to silence exceptions. If a handler cannot effectively handle the exceptional condition it should be re-raised. Exceptions that remain unhandled all the way back through the call stack indicate ineffective or missing handlers.
Back
Start check