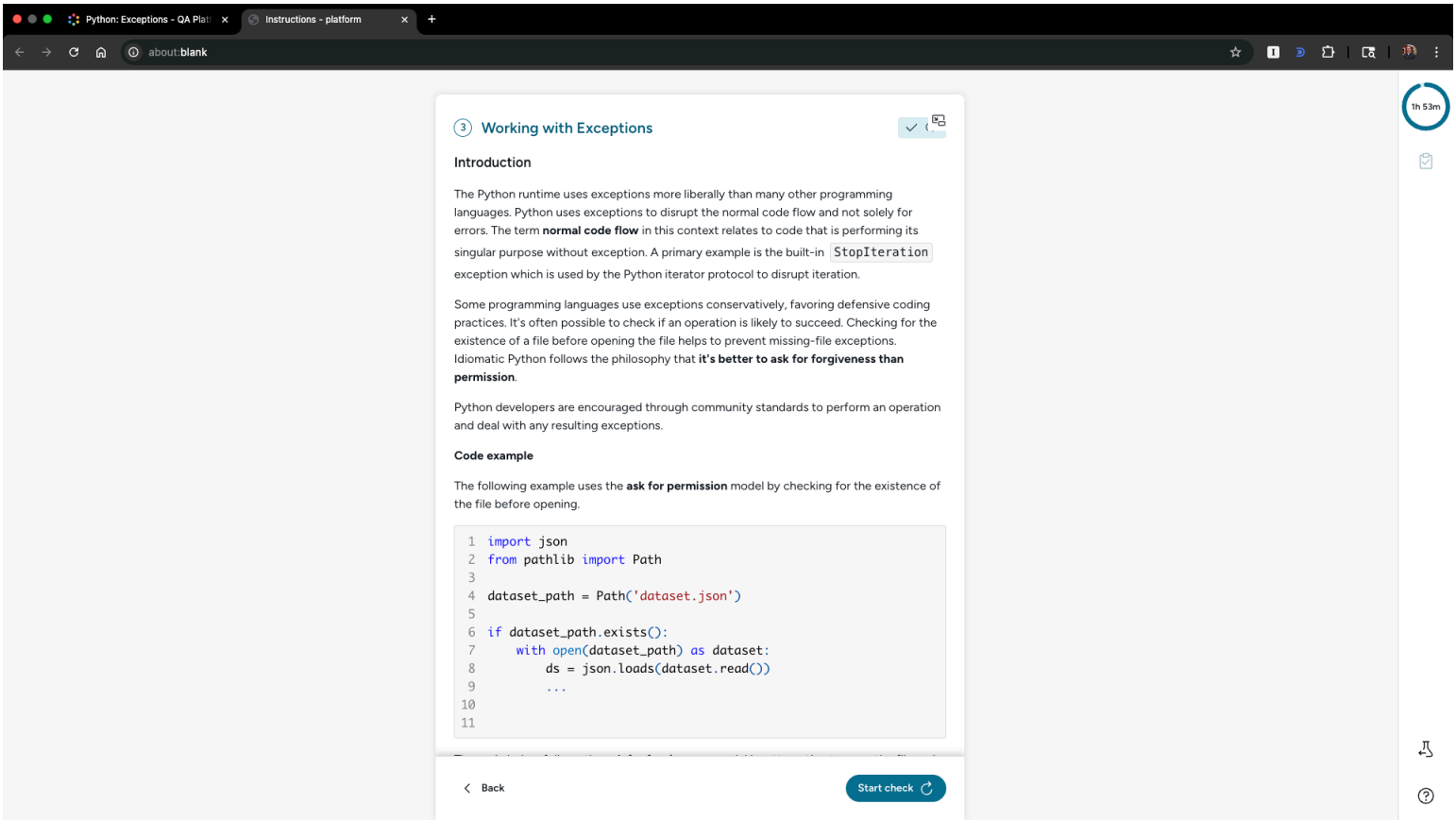
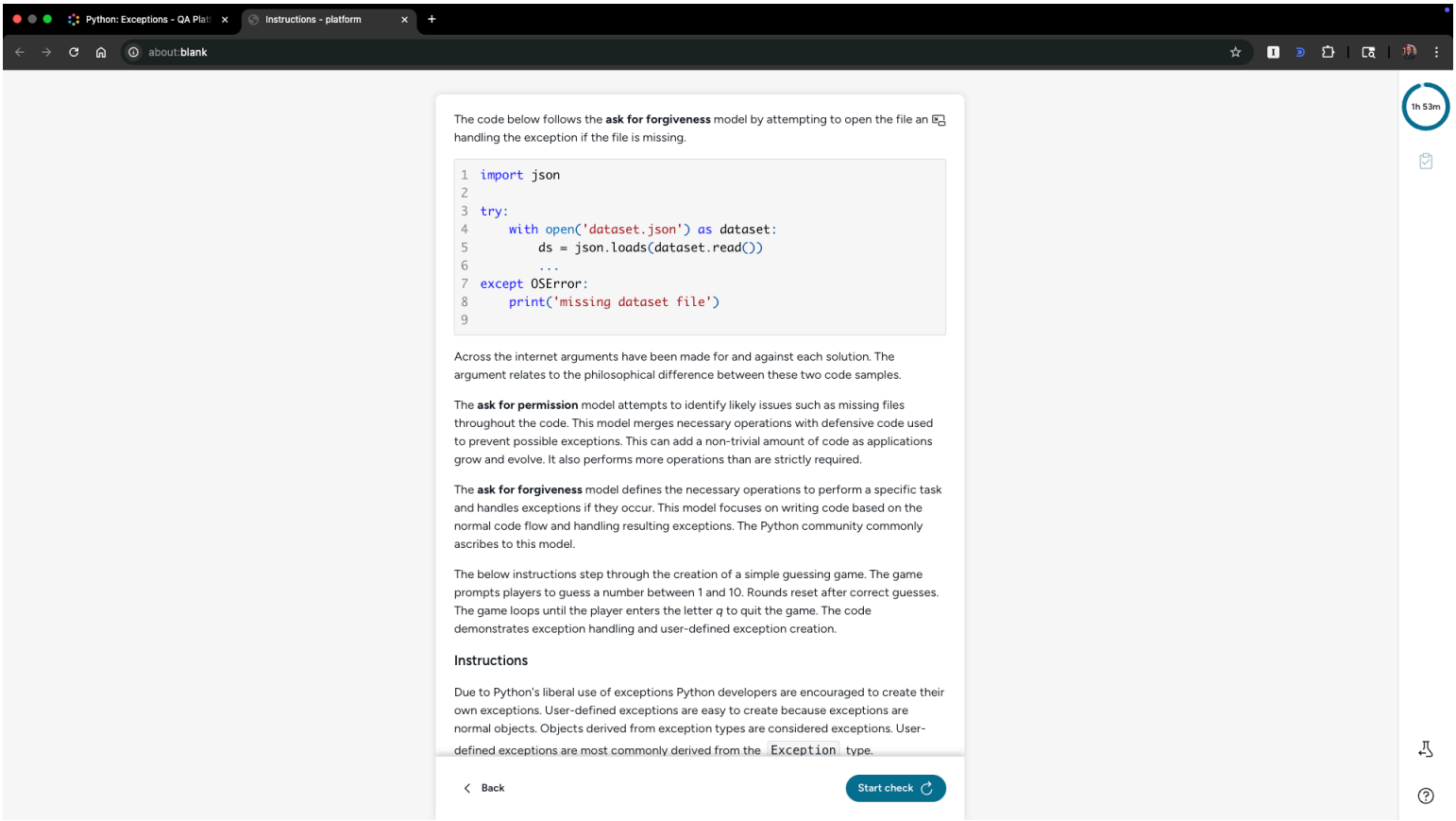

Python: Exceptions - QA Platf
Instructions - platform
about:blank
1h 53m
3 Working with Exceptions
Introduction
The Python runtime uses exceptions more liberally than many other programming languages. Python uses exceptions to disrupt the normal code flow and not solely for errors. The term normal code flow in this context relates to code that is performing its singular purpose without exception. A primary example is the built-in StopIteration exception which is used by the Python iterator protocol to disrupt iteration.
Some programming languages use exceptions conservatively, favoring defensive coding practices. It's often possible to check if an operation is likely to succeed. Checking for the existence of a file before opening the file helps to prevent missing-file exceptions. Idiomatic Python follows the philosophy that it's better to ask for forgiveness than permission.
Python developers are encouraged through community standards to perform an operation and deal with any resulting exceptions.
Code example
The following example uses the ask for permission model by checking for the existence of the file before opening.
1 import json
2 from pathlib import Path
3
4 dataset_path = Path('dataset.json')
5
6 if dataset_path.exists():
7 with open(dataset_path) as dataset:
8 ds = json.loads(dataset.read())
9 ...
10
11
Back
Start check


Python: Exceptions - QA Platf
Instructions - platform
about:blank
The code below follows the ask for forgiveness model by attempting to open the file an handling the exception if the file is missing.
import json
try:
with open('dataset.json') as dataset:
ds = json.loads(dataset.read())
...
except OSError:
print('missing dataset file')
Across the internet arguments have been made for and against each solution. The argument relates to the philosophical difference between these two code samples.
The ask for permission model attempts to identify likely issues such as missing files throughout the code. This model merges necessary operations with defensive code used to prevent possible exceptions. This can add a non-trivial amount of code as applications grow and evolve. It also performs more operations than are strictly required.
The ask for forgiveness model defines the necessary operations to perform a specific task and handles exceptions if they occur. This model focuses on writing code based on the normal code flow and handling resulting exceptions. The Python community commonly ascribes to this model.
The below instructions step through the creation of a simple guessing game. The game prompts players to guess a number between 1 and 10. Rounds reset after correct guesses. The game loops until the player enters the letter q to quit the game. The code demonstrates exception handling and user-defined exception creation.
Instructions
Due to Python's liberal use of exceptions Python developers are encouraged to create their own exceptions. User-defined exceptions are easy to create because exceptions are normal objects. Objects derived from exception types are considered exceptions. User-defined exceptions are most commonly derived from the Exception type.
Back
Start check
1h 53m

Due to Python's liberal use of exceptions Python developers are encouraged to create th... own exceptions. User-defined exceptions are easy to create because exceptions are normal objects. Objects derived from exception types are considered exceptions. User-defined exceptions are most commonly derived from the `Exception` type.

Some exceptions require additional attributes to accurately represent the root cause. For example the built-in `UnicodeError` type includes an `encoding` attribute which contains the name of the encoding that raised the exception. Exceptions that don't require additional attributes can be defined in a single line of code.

1. Add the following code to the **playground/guess.py** file.

```
1
2 class GameOver(Exception): ...
3
4
```

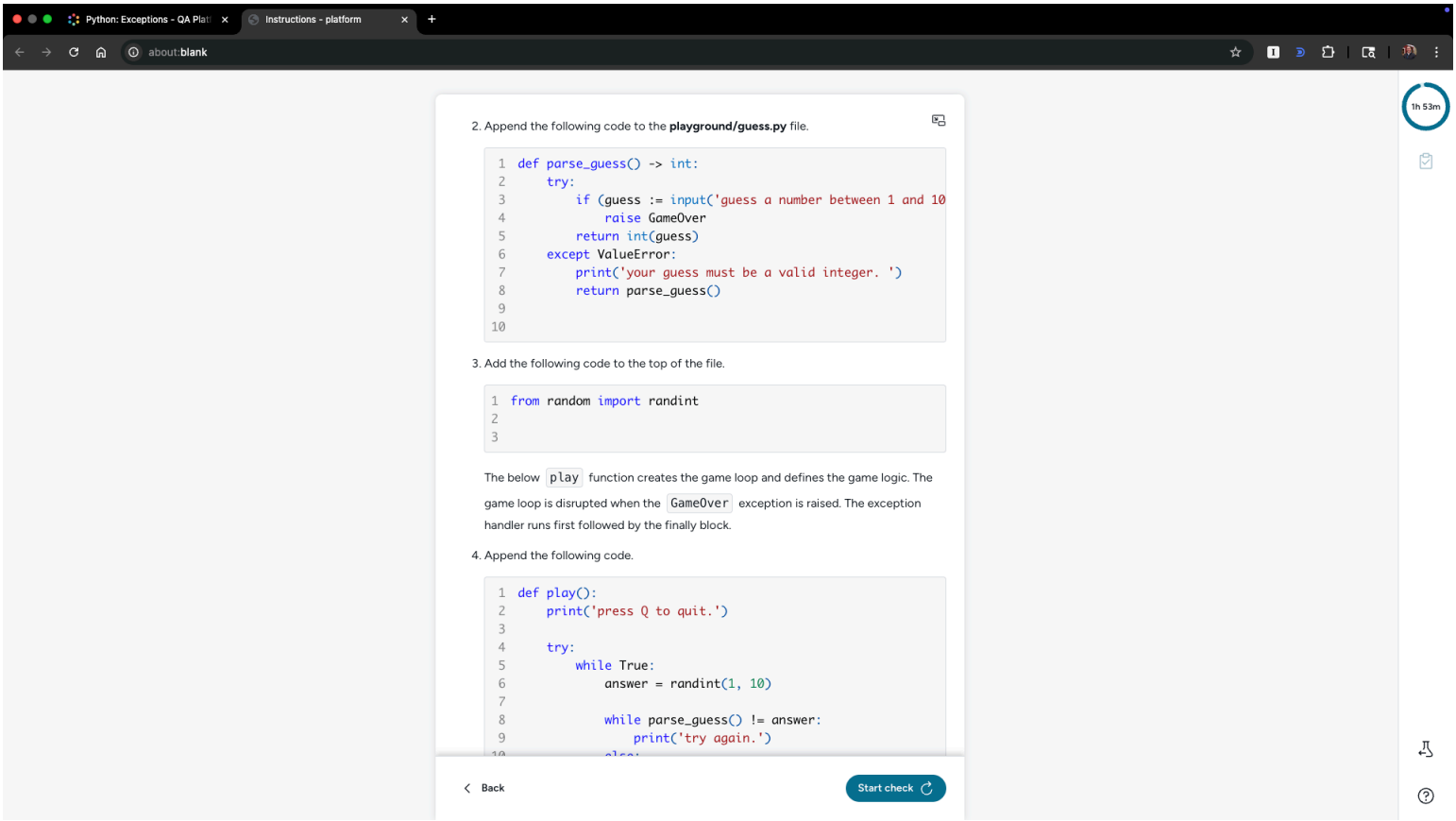
NOTE: The ellipsis (`...`) is synonymous with the `pass` keyword.

The `GameOver` exception is raised when the player quits the game. The `parse_guess` function in the below code is responsible for prompting the player for guesses. If the player enters the letter *q* the `GameOver` exception is raised. Otherwise the player's entry is converted to an `int` type and returned.

The below code handles `ValueError` exceptions caused when the player's entry cannot be converted to an `int` type. If a `ValueError` exception is raised then the player entered a non-numeric entry and must guess again. The except block recursively calls itself in order to prompt the user again. This function forms an implicit loop that will continue until a number or the letter *q* is entered.

2. Append the following code to the **playground/guess.py** file.

```
1 def parse_guess() -> int:
```


Python: Exceptions - QA Platf
Instructions - platform
about:blank
2. Append the following code to the playground/guess.py file.
def parse_guess() -> int:
try:
if (guess := input('guess a number between 1 and 10
raise GameOver
return int(guess)
except ValueError:
print('your guess must be a valid integer. ')
return parse_guess()
3. Add the following code to the top of the file.
from random import randint
The below play function creates the game loop and defines the game logic. The game loop is disrupted when the GameOver exception is raised. The exception handler runs first followed by the finally block.
4. Append the following code.
def play():
print('press Q to quit.')
try:
while True:
answer = randint(1, 10)
while parse_guess() != answer:
print('try again.')
Back
Start check
1h 53m

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

```
            while parse_guess() != answer:
                print('try again.')
            else:
                print('you win!')

    except GameOver:
        print('thanks for playing!')
    finally:
        print('like and subscribe!')

if __name__ == '__main__':
    play()
```

playground > guess.py

```
from random import randint


class GameOver(Exception): ...

def parse_guess() -> int:
    try:
        if (guess := input('guess a number between 1 and 10 > ')).lower() == 'q':
            raise GameOver
        return int(guess)
    except ValueError:
        print('your guess must be a valid integer. ')
        return parse_guess()

def play():
    print('press Q to quit.')

    try:
        while True:
            answer = randint(1, 10)

            while parse_guess() != answer:
                print('try again.')
            else:
                print('you win!')

```

Back | Start check

1h 53m

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

1h 53m

playground > guess.py

```python
from random import randint


class GameOver(Exception): ...

def parse_guess() -> int:
    try:
        if (guess := input('guess a number between 1 and 10 > ')).lower() == 'q':
            raise GameOver
        return int(guess)
    except ValueError:
        print('your guess must be a valid integer. ')
        return parse_guess()

def play():
    print('press Q to quit.')

    try:
        while True:
            answer = randint(1, 10)

            while parse_guess() != answer:
                print('try again.')
            else:
                print('you win!')

    except GameOver:
        print('thanks for playing!')
    finally:
        print('like and subscribe!')

if __name__ == '__main__':
    play()

```

5. Run the code to play the game.

```
python3 playground/guess.py
```

```
press Q to quit.
guess a number between 1 and 10 >
```

There are multiple paths through the code two of which include exceptions.

6. Enter invalid input: *x*

Back

Start check

6. Enter invalid input: *x*

```
press Q to quit.
guess a number between 1 and 10 > x
your guess must be a valid integer.
guess a number between 1 and 10 >
```

The runtime raises a `ValueError` exception attempting to convert the letter *x* into a number. The `ValueError` exception handler informs the player of the problem before reattempting the operation. Exception handlers provide opportunities to resolve correctable exceptions.

7. Enter numbers 1 - 10 until the answer is found.

```
press Q to quit.
guess a number between 1 and 10 > x
your guess must be a valid integer.
guess a number between 1 and 10 > 1
try again.
guess a number between 1 and 10 > 2
you win!
guess a number between 1 and 10 >
```

This game loop is the **normal code flow** for this code. Raising an exception to disrupt the flow is used in this example because quitting is an exception to the normal code flow

8. Enter the letter: *q*

```
press Q to quit.
guess a number between 1 and 10 > x
your guess must be a valid integer.
guess a number between 1 and 10 > 1
try again.
guess a number between 1 and 10 > 2
you win!
guess a number between 1 and 10 > q
thanks for playing!
like and subscribe!
```

The `parse_guess` function raises the `GameOver` exception when the letter q is encountered. Notice the `parse_guess` function doesn't handle the `GameOver` exception. The handler resides in the `play` function. When an exception occurs the

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

8. Enter the letter: *q*

```
press Q to quit.
guess a number between 1 and 10 > x
your guess must be a valid integer.
guess a number between 1 and 10 > 1
try again.
guess a number between 1 and 10 > 2
you win!
guess a number between 1 and 10 > q
thanks for playing!
like and subscribe!
```

The `parse_guess` function raises the `GameOver` exception when the letter q is encountered. Notice the `parse_guess` function doesn't handle the `GameOver` exception. The handler resides in the `play` function. When an exception occurs the runtime goes back through the call stack and checks for suitable handlers. The exception handler in the `play` function is used because the exception remained unhandled by the `parse_guess` function.

Exceptions remaining unhandled through the entire call stack cause the runtime to report the traceback before exiting the application.

⭐Proceed to the next step⭐

Validations

On Track - Working With Exceptions

Ensure your lab environment is on-track

Python

Report an issue

Back

Start check

1h 53m