Chrome File Edit View History Bookmarks Profiles Tab Window Help

Mon Jun 2 11:06 AM

LIConnections - VA and DoD | Log in to QA's learning platfo | Python: Introduction to Debu | Instructions - platform

about:blank

1h 59m

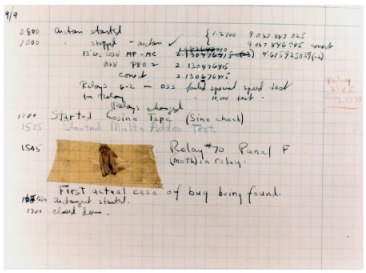
2 Caveman Debugging

Introduction

Software bugs occur when a code defect results in unexpected behaviors such as errors or incorrect results. The process of locating a bug and correcting the defect is referred to as **debugging**. The first ever computer **bug** was an actual bug. In 1946 Grace Hopper invented debugging when she traced an error in the Mark II electromechanical computer to a moth trapped in a relay.

Back

Next



(Left)
By Unknown (Smithsonian Institution) - Flickr: Grace Hopper and UNIVAC, CC BY 2.0

(Right)
Courtesy of the Naval Surface Warfare Center, Dahlgren, VA., 1988. - U.S. Naval Historical Center Online Library Photograph NH 96566-KN

The purpose of software debugging is to remove defects from a piece of software. Debugging is a core part of the development process. Going from the initial proof-of-concept code to the final application is an iterative process. Code is run, observed, and tested as it changes. This can be accomplished manually or via automated testing.

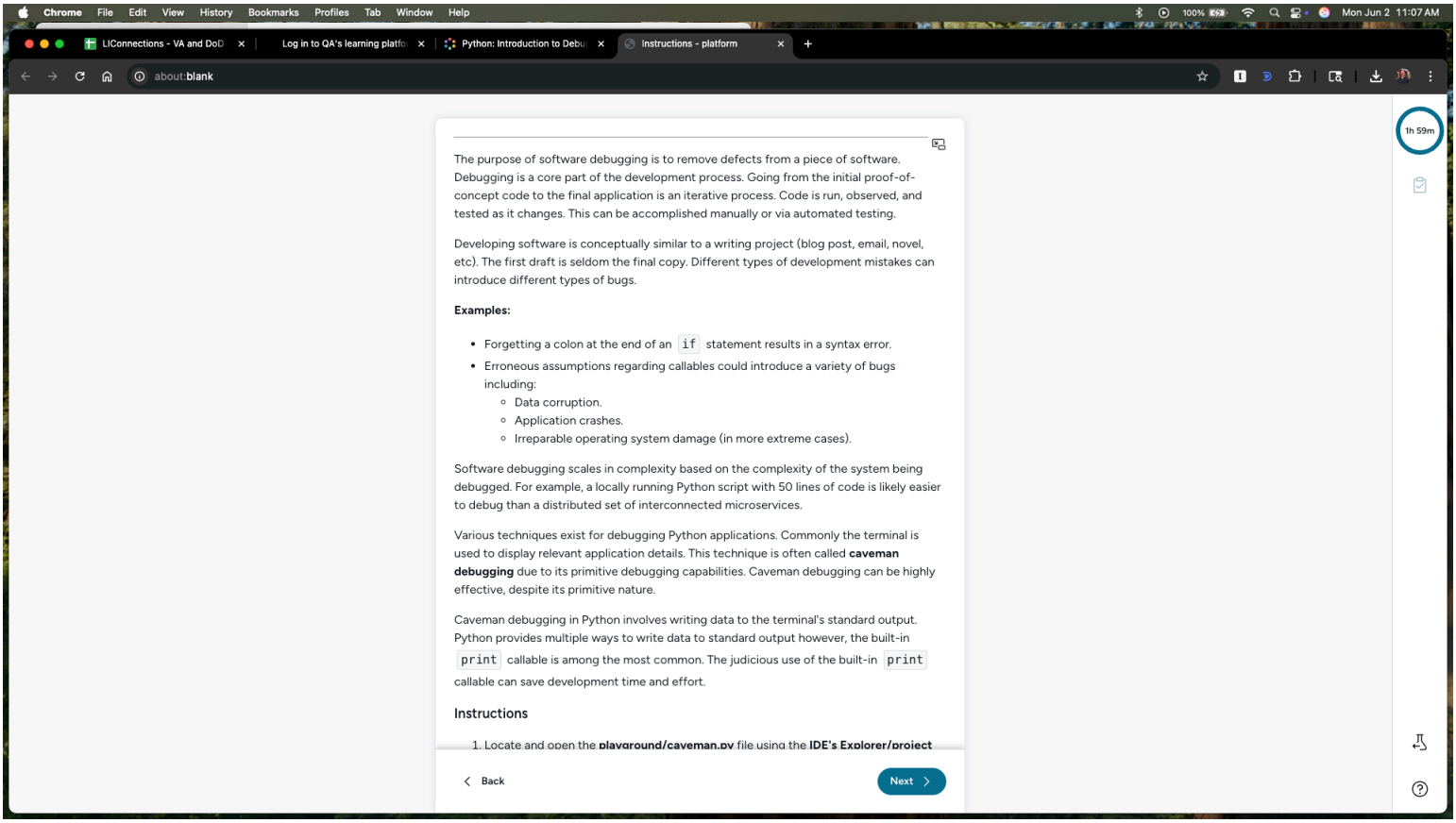
Developing software is conceptually similar to a writing project (blog post, email, novel, etc). The first draft is seldom the final copy. Different types of development mistakes can introduce different types of bugs.

Examples:

- Forgetting a colon at the end of an `if` statement results in a syntax error.
- Erroneous assumptions regarding callables could introduce a variety of bugs

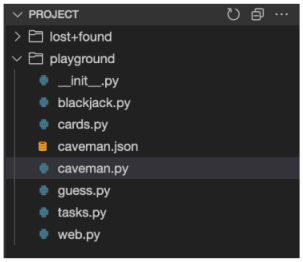
Back

Next


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 11:07 AM
LIConnections - VA and DoD
Log in to QA's learning platfo
Python: Introduction to Debu
Instructions - platform
about:blank
1h 59m
The purpose of software debugging is to remove defects from a piece of software. Debugging is a core part of the development process. Going from the initial proof-of-concept code to the final application is an iterative process. Code is run, observed, and tested as it changes. This can be accomplished manually or via automated testing.
Developing software is conceptually similar to a writing project (blog post, email, novel, etc). The first draft is seldom the final copy. Different types of development mistakes can introduce different types of bugs.
Examples:
Forgetting a colon at the end of an if statement results in a syntax error.
Erroneous assumptions regarding callables could introduce a variety of bugs including:
Data corruption.
Application crashes.
Irreparable operating system damage (in more extreme cases).
Software debugging scales in complexity based on the complexity of the system being debugged. For example, a locally running Python script with 50 lines of code is likely easier to debug than a distributed set of interconnected microservices.
Various techniques exist for debugging Python applications. Commonly the terminal is used to display relevant application details. This technique is often called caveman debugging due to its primitive debugging capabilities. Caveman debugging can be highly effective, despite its primitive nature.
Caveman debugging in Python involves writing data to the terminal's standard output. Python provides multiple ways to write data to standard output however, the built-in print callable is among the most common. The judicious use of the built-in print callable can save development time and effort.
Instructions
1. Locate and open the playground/caveman.py file using the IDE's Explorer/project
Back
Next

Instructions

1. Locate and open the **playground/caveman.py** file using the **IDE's Explorer/project** pane.



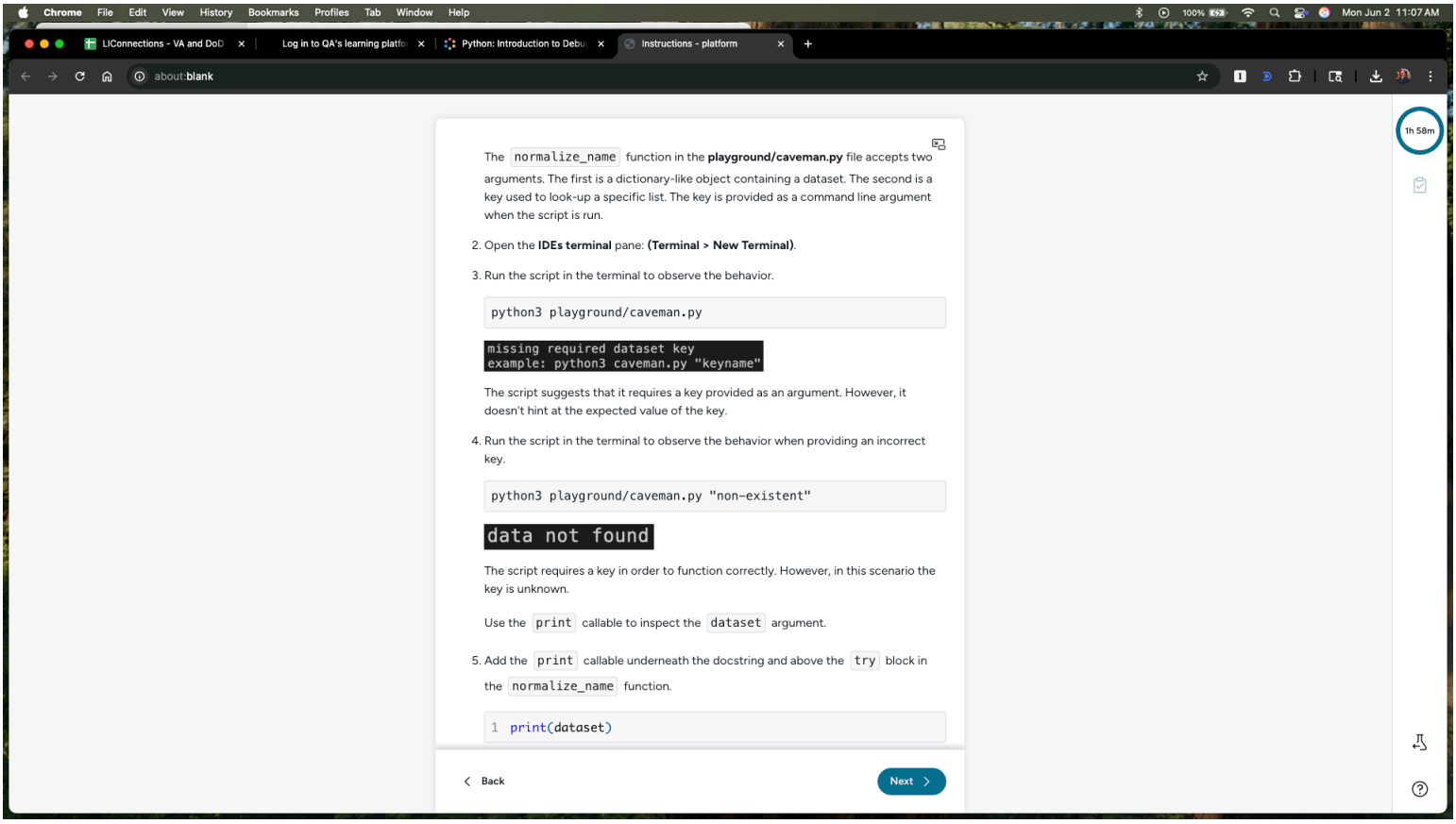
Python provides a mechanism for representing objects written to standard output via the built-in `print` callable. The `__repr__` magic method determines how objects are displayed when printed. Many of Python's built-in object types implement `__repr__` in order to provide meaningful object representations. This makes the built-in `print` callable a reasonably effective debugging tool.

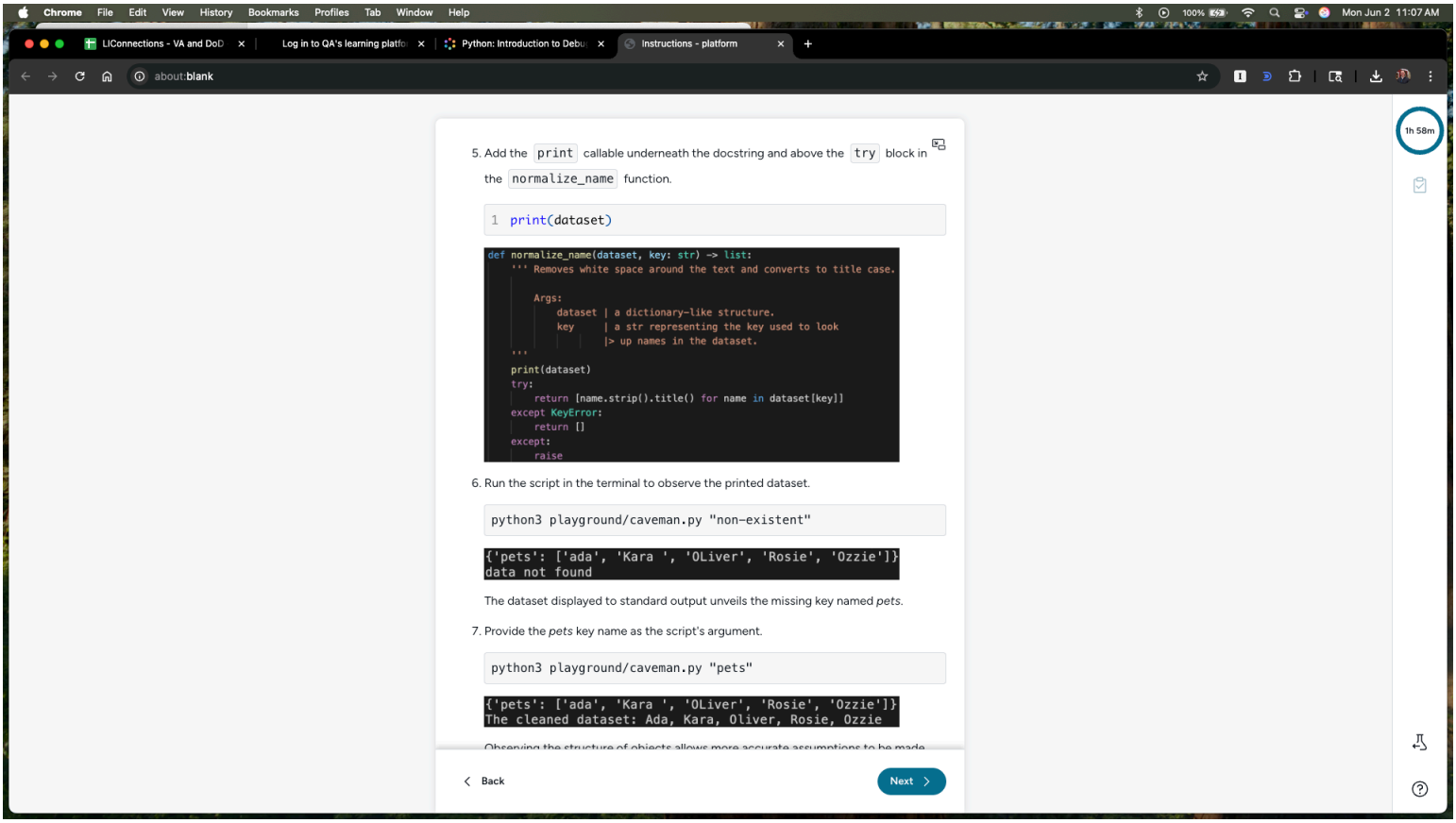
Inspecting an object's data and or structure is a good use case for caveman debugging. Imagine working with an external dataset on a project that you didn't develop. In this case displaying the dataset in the terminal using the `print` callable is an easy way to quickly visualize the data.

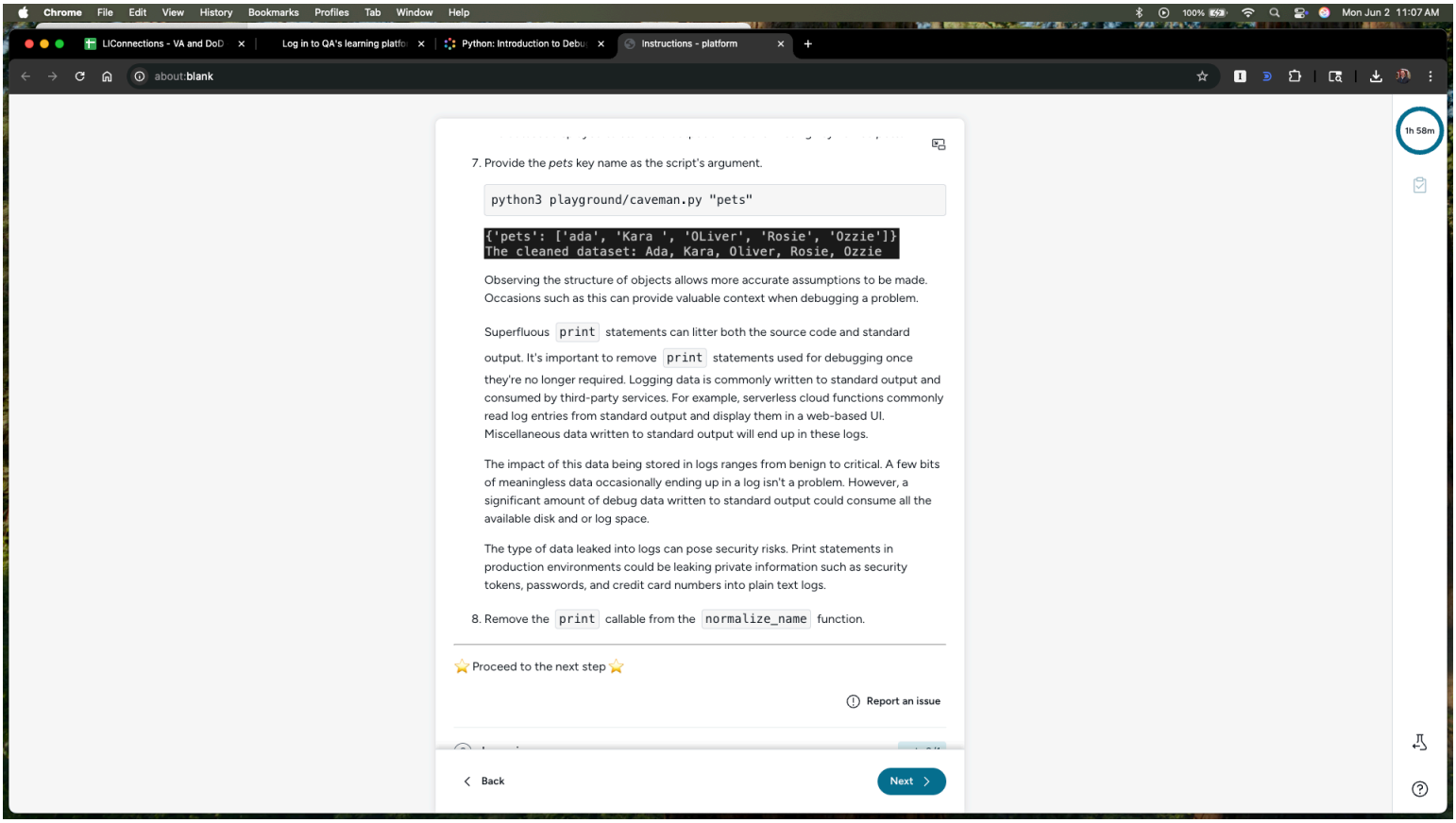
The `normalize_name` function in the **playground/caveman.py** file accepts two arguments. The first is a dictionary-like object containing a dataset. The second is a key used to look-up a specific list. The key is provided as a command line argument

Back

Next


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 11:07 AM
LIConnections - VA and DoD
Log in to QA's learning platfo
Python: Introduction to Debu
Instructions - platform
about:blank
1h 58m
The normalize_name function in the playground/caveman.py file accepts two arguments. The first is a dictionary-like object containing a dataset. The second is a key used to look-up a specific list. The key is provided as a command line argument when the script is run.
2. Open the IDEs terminal pane: (Terminal > New Terminal).
3. Run the script in the terminal to observe the behavior.
python3 playground/caveman.py
missing required dataset key
example: python3 caveman.py "keyname"
The script suggests that it requires a key provided as an argument. However, it doesn't hint at the expected value of the key.
4. Run the script in the terminal to observe the behavior when providing an incorrect key.
python3 playground/caveman.py "non-existent"
data not found
The script requires a key in order to function correctly. However, in this scenario the key is unknown.
Use the print callable to inspect the dataset argument.
5. Add the print callable underneath the docstring and above the try block in the normalize_name function.
1 print(dataset)
Back
Next


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 11:07 AM
LIConnections - VA and DoD
Log in to QA's learning platfo
Python: Introduction to Debu
Instructions - platform
about:blank
1h 58m
5. Add the print callable underneath the docstring and above the try block in the normalize_name function.
1 print(dataset)
def normalize_name(dataset, key: str) -> list:
''' Removes white space around the text and converts to title case.
Args:
dataset | a dictionary-like structure.
key | a str representing the key used to look
|> up names in the dataset.
'''
print(dataset)
try:
return [name.strip().title() for name in dataset[key]]
except KeyError:
return []
except:
raise
6. Run the script in the terminal to observe the printed dataset.
python3 playground/caveman.py "non-existent"
{'pets': ['ada', 'Kara ', 'OLiver', 'Rosie', 'Ozzie']}
data not found
The dataset displayed to standard output unveils the missing key named pets.
7. Provide the pets key name as the script's argument.
python3 playground/caveman.py "pets"
{'pets': ['ada', 'Kara ', 'OLiver', 'Rosie', 'Ozzie']}
The cleaned dataset: Ada, Kara, Oliver, Rosie, Ozzie
Observing the structure of objects allows more accurate assumptions to be made
Back
Next


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon Jun 2 11:07 AM
LIConnections - VA and DoD
Log in to QA's learning platfo
Python: Introduction to Debu
Instructions - platform
about:blank
1h 58m
7. Provide the pets key name as the script's argument.
python3 playground/caveman.py "pets"
{'pets': ['ada', 'Kara ', 'OLiver', 'Rosie', 'Ozzie']}
The cleaned dataset: Ada, Kara, Oliver, Rosie, Ozzie
Observing the structure of objects allows more accurate assumptions to be made. Occasions such as this can provide valuable context when debugging a problem.
Superfluous print statements can litter both the source code and standard output. It's important to remove print statements used for debugging once they're no longer required. Logging data is commonly written to standard output and consumed by third-party services. For example, serverless cloud functions commonly read log entries from standard output and display them in a web-based UI. Miscellaneous data written to standard output will end up in these logs.
The impact of this data being stored in logs ranges from benign to critical. A few bits of meaningless data occasionally ending up in a log isn't a problem. However, a significant amount of debug data written to standard output could consume all the available disk and or log space.
The type of data leaked into logs can pose security risks. Print statements in production environments could be leaking private information such as security tokens, passwords, and credit card numbers into plain text logs.
8. Remove the print callable from the normalize_name function.
Proceed to the next step
Report an issue
Back
Next