

# Cloud Debugger

## Quickstart



This quickstart shows you how to use Cloud Debugger to debug the state of a simple Python app running on App Engine. This quickstart shows you how to accomplish the following:

- Inspect local variables and the call stack
- Generate logging statements
- Set snapshot conditions and use expressions

## Before you begin

1. If you're new to Google Cloud, [create an account](#) to evaluate how our products perform in real-world scenarios. New customers also get \$300 in free credits to run, test, and deploy workloads.
2. In the Google Cloud Console, on the project selector page, select or create a Google Cloud project.

**Note:** If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

[Go to project selector](#)

3. Make sure that billing is enabled for your Cloud project. [Learn how to confirm that billing is enabled for your project.](#)
4. [Install and initialize the Cloud SDK.](#)
5. Make sure that the following software is installed on your local system:
  - [Git](#)
  - [Cloud SDK](#)

## Deploy sample app to App Engine

Start by deploying a Python 3.7 app to App Engine.

1. Clone the app to your local computer:

```
git clone https://github.com/GoogleCloudPlatform/python-docs-samples.git
```

2. Navigate to the directory that contains the app:

```
cd python-docs-samples/appengine/standard_python3/cloud_debugger
```

3. Deploy the app to App Engine by issuing the following command:

```
gcloud app deploy
```

When prompted, select the [region](#) where you want your App Engine app located.

4. View the app by issuing the following command:

```
gcloud app browse
```

If a browser window does not automatically open displaying the app, click the URL that appears in the terminal.

The app contains a prompt to enter a string with the field already prepopulated.

Hello! Enter a string to reverse it.

**Note:** If you have problems getting the app deployed, refer to the App Engine [Quickstart for Python 3 in the App Engine standard environment](#).

5. Click **Submit**.

**Program Output:** dbca

**Correct Output:** dcba

You see two results after clicking submit. One is labeled **Program Output** and shows the output from the Reverse method in the source code. The other is labeled **Correct Output** and is the output from Python's reverse list functionality.

The **Program Output** and the **Correct Output** should be identical. However, there's a problem with the source code. Use Debugger to diagnose the issue.

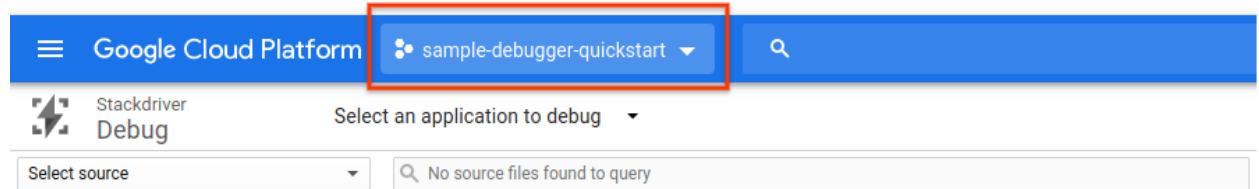
## View deployed source code

After you have deployed your app, you can view the deployed source code on the Google Cloud Console **Debug** page.

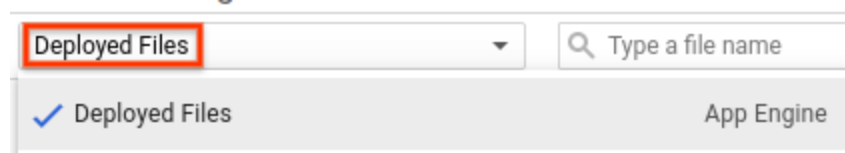
1. Navigate to the **Debug** page in the Google Cloud Console:

[Go to Debug page](#)

2. Make sure you have the correct project selected:

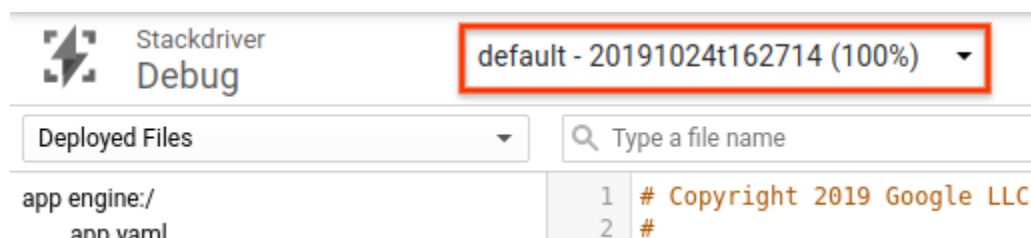


3. Confirm Debugger has access to the deployed files by verifying that **Deployed files** is selected and the app's files are present:



**Note:** For ways to upload source code to Debugger, see [Selecting source code automatically](#) or [Selecting source code manually](#).

4. Make sure the correct version of your app is selected:



App Engine maintains all the deployed versions of an app. When using Debugger, verify that you have the correct version of the app selected.

After selecting the main.py file, you see the following block of code:

```
try:
    import googleclouddebugger
    googleclouddebugger.enable()
except ImportError:
    pass

import logging
```

```
logging.basicConfig(level=logging.INFO)
```

This section imports and enables the Debugger agent using a try and except block :

```
try:
    import googleclouddebugger
    googleclouddebugger.enable()
except ImportError:
    pass
```

This section configures logging:

```
import logging
logging.basicConfig(level=logging.INFO)
```

**Note:** For information on setting up Debugger, go to [Setting up Debugger](#).

You are now ready to take debug snapshots and inject debug logpoints to diagnose the problem in the source code.

## Take a debug snapshot



A debug snapshot captures the local variables and call stack that are in scope at a line location.

1. To take a debug snapshot, click the line number that contains the tmp variable. A blue arrow appears indicating that a snapshot is set, and the results panel displays "Waiting for snapshot to hit."





2. To trigger your snapshot, refresh the page.

The **Variables** pane shows the values of the variables. See that the chars array was correctly populated on the first pass through the loop. The problem is not present here because the snapshot was taken the first time the breakpoint was hit.

Variables  

self	
chars	
[0]	'a'
[1]	'b'
[2]	'c'
[3]	'd'
left	0
right	3

The **Call Stack** pane shows the results of the call stack. You can click on the functions in the **Call Stack** pane to see the local variables and parameters at that point in the code. When you click on ReverseString, you see that the input was correct.

Variables   2019-10-30 (13:58:43)

s	'abcd'
---	--------

Call Stack

StringProcessor.Reverse	main.py:47
ReverseString	main.py:66
Flask.dispatch_request	app.py:1935
Flask.full_dispatch_request	app.py:1949
Flask.wsgi_app	app.py:2446
Flask.__call__	app.py:2463
ThreadWorker.handle_request	gthread.py:328
ThreadWorker.handle	gthread.py:279
_WorkItem.run	thread.py:57
_worker	thread.py:80
Thread.run	threading.py:870
Thread._bootstrap_inner	threading.py:926
Thread._bootstrap	threading.py:890

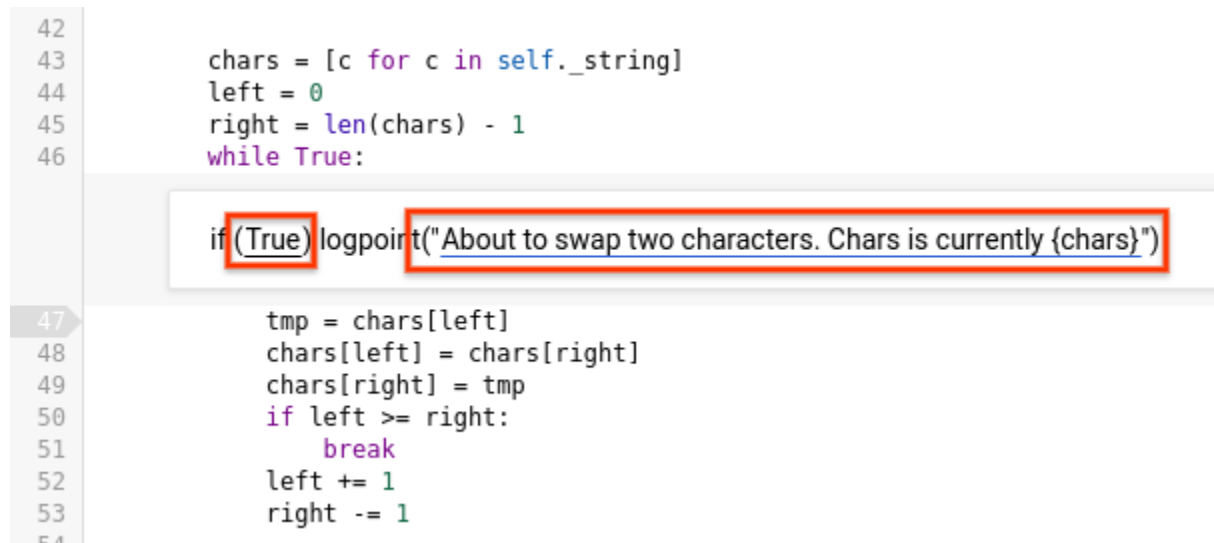
Since taking a snapshot and inspecting the variables and call stack didn't reveal the problem, use logpoints to track down the problem.

## Inject a debug logpoint

A debug logpoint enables you to inject logging into your running app without restarting it.

1. To insert a logpoint, select the **Logpoint** tab.
2. Click the line number that contains the tmp variable. An inline text box appears. Populate the fields as follows and click **Add**:

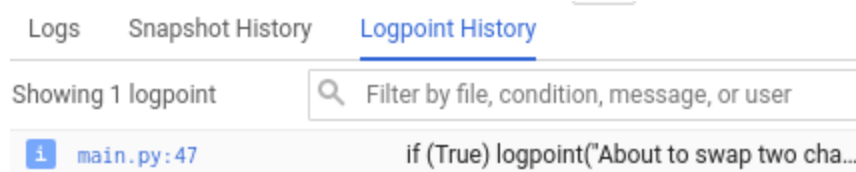
```
if (True) logpoint("About to swap two characters. Chars is currently {chars}")
```



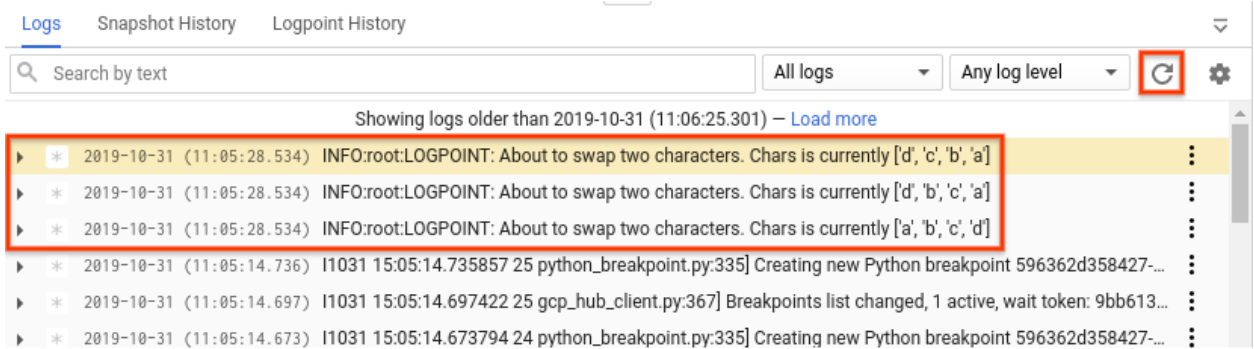
```
42
43     chars = [c for c in self._string]
44     left = 0
45     right = len(chars) - 1
46     while True:
47         tmp = chars[left]
48         chars[left] = chars[right]
49         chars[right] = tmp
50         if left >= right:
51             break
52         left += 1
53         right -= 1
54
```

A logpoint has the following structure: `if(condition)logpoint(string)`. To create a logpoint, you supply two parts:

- A condition, which must be written in the syntax of the source code.
  - A string, which can contain any number of expressions in curly braces written in the syntax of the source code.
3. To verify that the logpoint has been injected successfully, select the **Logpoint History** tab.



4. To trigger the logpoint, refresh the page.
5. To see the logs generated by the logpoint, select the **Logs** panel and click refresh **refresh**.



**Note:** Debugger logpoints can also be viewed in the **Logs Explorer** console in Cloud Logging.

## Diagnosing the issue

The logpoints indicate that the while loop executes three times, but it only needs to execute two times. Since the logpoint is set at the beginning of the loop, it logged that it was about to swap characters on a string that was already fully reversed.

```
INFO:root:LOGPOINT: About to swap two characters. Chars is currently ['d', 'c', 'b', 'a']
INFO:root:LOGPOINT: About to swap two characters. Chars is currently ['d', 'b', 'c', 'a']
INFO:root:LOGPOINT: About to swap two characters. Chars is currently ['a', 'b', 'c', 'd']
```

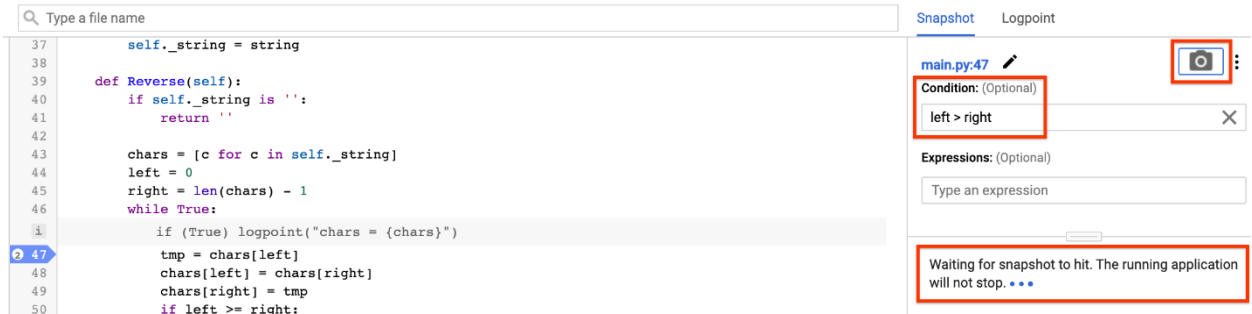
To pinpoint the problem, use a debug snapshot with a condition.

## Take a debug snapshot using a condition

The app uses the left variable and right variable to track when to stop swapping values. When the left variable is greater than the right variable, the loop should terminate.

Snapshots can be set to trigger based on a condition in the source code. Since you know when the loop should terminate, use a snapshot with a condition to isolate the problem.

1. To determine why the loop executes too many times, set the snapshot to trigger on the following condition: `left > right`. Then click the camera icon `camera_alt` to prepare Debugger for the snapshot.



2. Trigger the snapshot by refreshing the page. The **Variables** pane shows that the left variable is greater than the right variable.

Since the left variable is already greater than the right variable at this point in the loop, it means that as the loop continues, it will swap the values one more time before it reaches line 50 and exits the loop.

Variables		2019-10-27 (13:00:08)
self		
chars		
left	2	
right	1	
tmp	'b'	

## Take a debug snapshot using an expression




Debugger also lets you retrieve programming-language expressions when the snapshot is triggered. For example, you can retrieve the values in the `chars` array with expressions like `chars[1]` and `chars[2]`.

1. To get the value of an expression when a snapshot is triggered, write the expression in the **Expression** field. You can enter multiple expressions.

After the snapshot triggers, the expression values appear above the **Variables** pane.



Snapshot   Logpoint



**main.py:47**   

Condition: (Optional)

left > right

Expressions: (Optional)

chars[1]	×
chars[2]	×
Type an expression	×

Expressions   2019-10-27 (13:26:40)

chars[1]	'c'
chars[2]	'b'

Variables

## Fixing the issue

Snapshots and logpoints help diagnose the problem in the app. In this example, the while loop executes too many times because the statement that stops the loop, if left >= right: break is reached too late to stop the last iteration. To fix the problem, move if left >= right: break from line 50 to line 47.

## Cleanup

All we have to do is stop the program we have running for the profiler.

## Congratulations!

In this experiment, we worked with Cloud Profiler

Explore the how-to guides for more detail

[https://cloud.google.com/profiler/docs/how-to?\\_ga=2.243799365.-1776565748.1641784757](https://cloud.google.com/profiler/docs/how-to?_ga=2.243799365.-1776565748.1641784757)