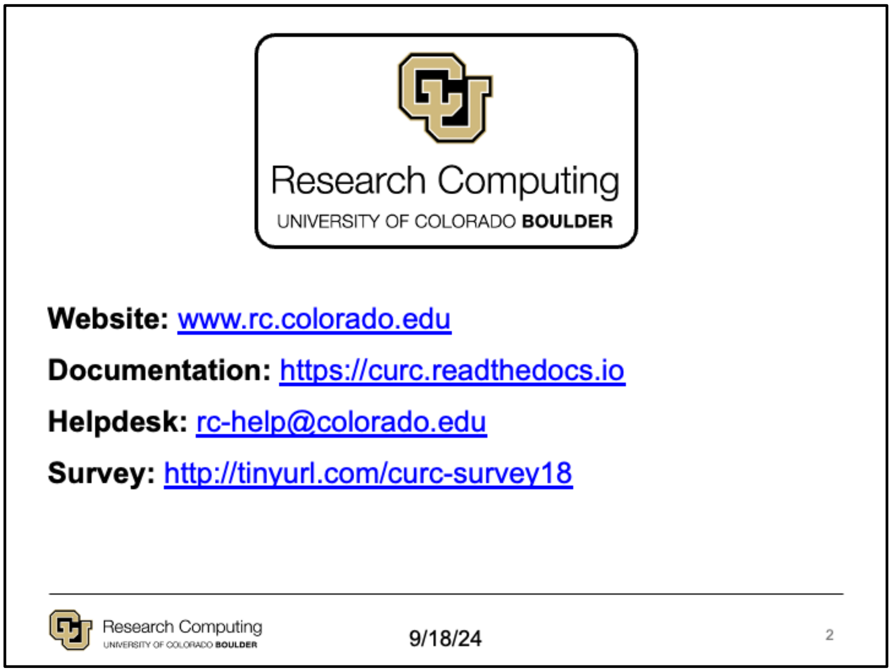
Introduction to Debugging

University of Colorado
Boulder

Artwork by @allison_horst



Research Computing
UNIVERSITY OF COLORADO **BOULDER**

Website: www.rc.colorado.edu

Documentation: https://curc.readthedocs.io

Helpdesk: rc-help@colorado.edu

Survey: http://tinyurl.com/curc-survey18

Research Computing
UNIVERSITY OF COLORADO BOULDER

Meet the User Support Team

Layla
Freeborn

Brandon
Reyes

Andy
Monaghan

Michael
Schneider

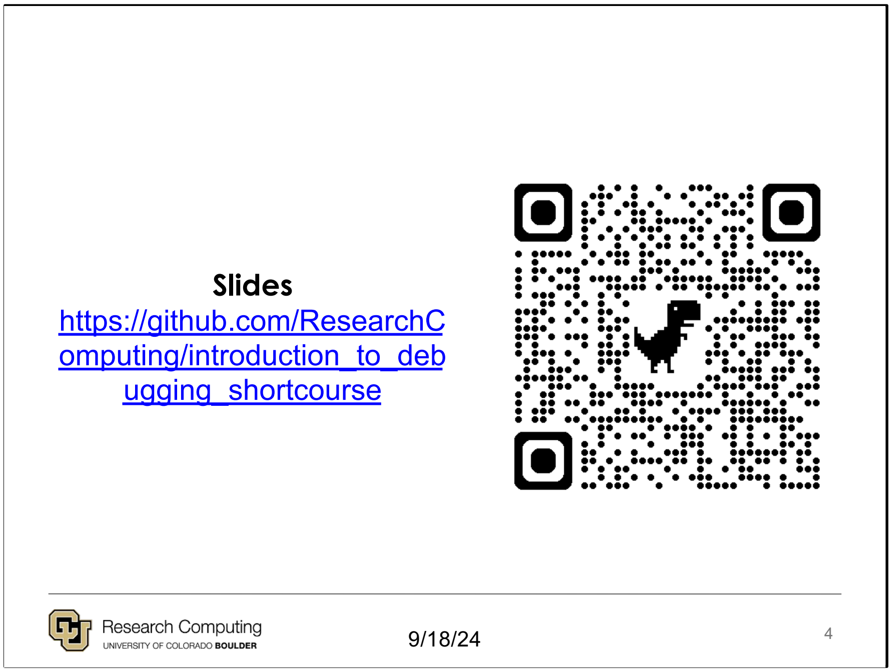
John
Reiland

Dylan
Gottlieb

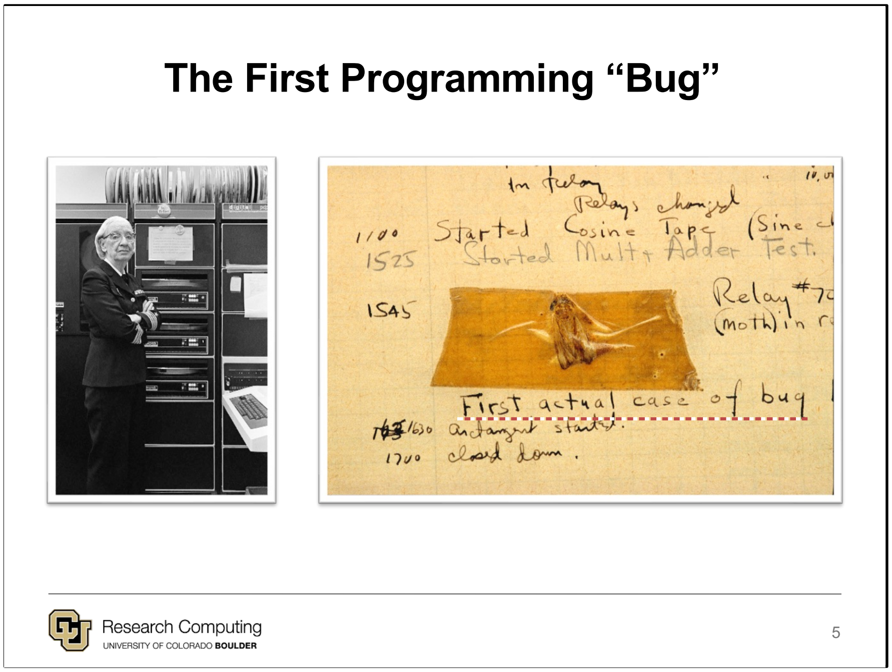
Mohal
Khandelwal

Ragan
Lee

Research Computing
UNIVERSITY OF COLORADO BOULDER



Slides
https://github.com/ResearchComputing/introduction_to_debugging_shortcourse



The First Programming "Bug"

1100 Started Cosine Tape (Sine c
1525 Started Mult+ Adder Test.
1545 Relay #70 (moth) in r
First actual case of bug
1630 antangent started.
1700 closed down.

Research Computing
UNIVERSITY OF COLORADO BOULDER

5

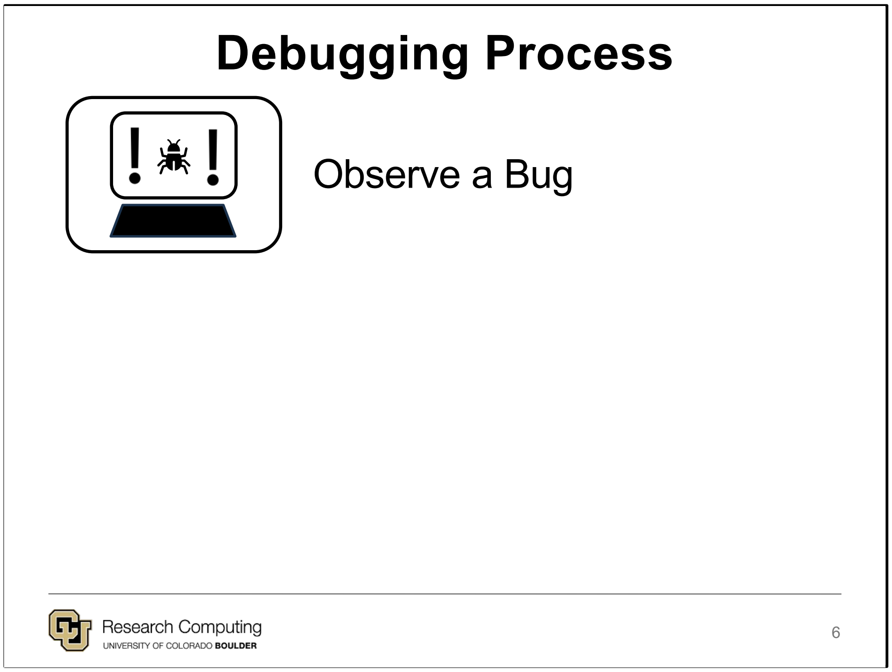
- Bugs == Errors
- The phrases (bug and debugging) are often attributed to Grace Hopper and her team – who found a literal bug (a moth) stuck in their computational machine (Mark II) – but the term is actuall an age-old engineering term dating back over century.
- In this workshop, we will discuss the process of debugging and the different techniques that you can use to better find and fix the bugs.

Description (Smithsonian):
'American engineers have been calling small flaws in machines "bugs" for over a century. Thomas Edison talked about bugs in electrical circuits in the 1870s. When the first computers were built during the early 1940s, people working on them found bugs in both the hardware of the machines and in the programs that ran them.

In 1947, engineers working on the Mark II computer at Harvard University found a moth stuck in one of the components. They taped the insect in their logbook and labeled it "first actual case of bug being found." The words "bug" and "debug" soon became a standard part of the language of computer programmers.

Among those working on the Mark II in 1947 was mathematician and computer programmer Grace Hopper, who later became a Navy rear admiral. This log book was probably not Hopper's, but she and the rest of the Mark II team helped popularize the use of the term computer bug and the related phrase "debug.”’

Image & Description Source:
https://americanhistory.si.edu/collections/nmah_334663



Debugging Process

Observe a Bug

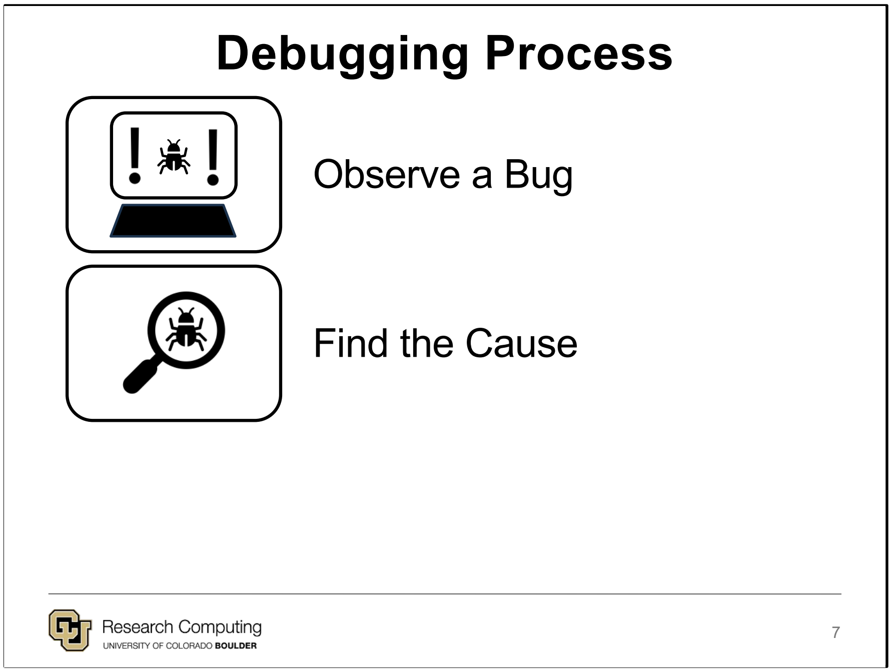
Research Computing
UNIVERSITY OF COLORADO BOULDER

6

Before we can start debugging, we must first recognize the need for debugging – i.e. observe a bug.

In the context of software programming, a bug refers to a moment when a system behaves in an unexpected or unintended manner.

This “unexpected behavior” can run the gambit of incorrect output to the full-blown system crashes!



Debugging Process

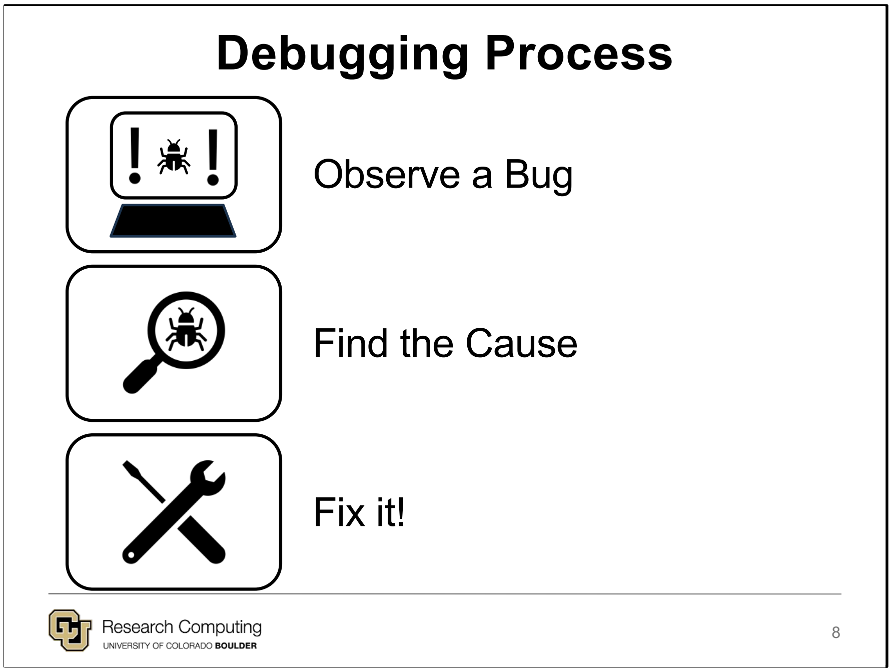
Observe a Bug

Find the Cause

After observing a bug, we must start searching for its underlying cause – which could be any one of a variety of potential errors.

It is important to note that the error, or errors, can exist across three spaces – the project's code, the system running the code, and then engineer's mind.

Today we'll be focusing on the first space, code, but it is important to always remember to check for issues in the system's hardware and your own understanding of the system and the code.



Debugging Process

Observe a Bug

Find the Cause

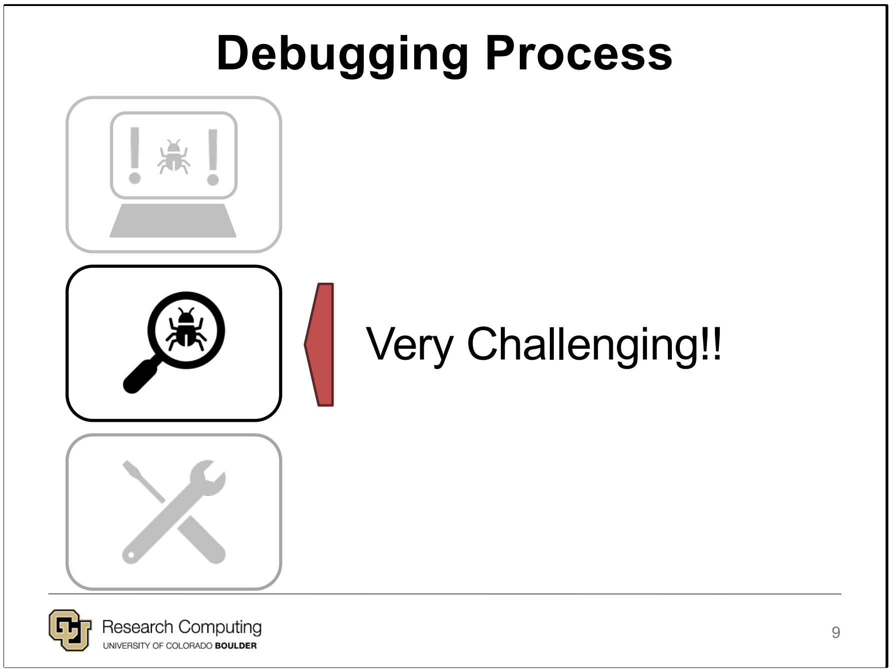
Fix it!

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

8

Once the error, or errors, has been identified it must be fixed.

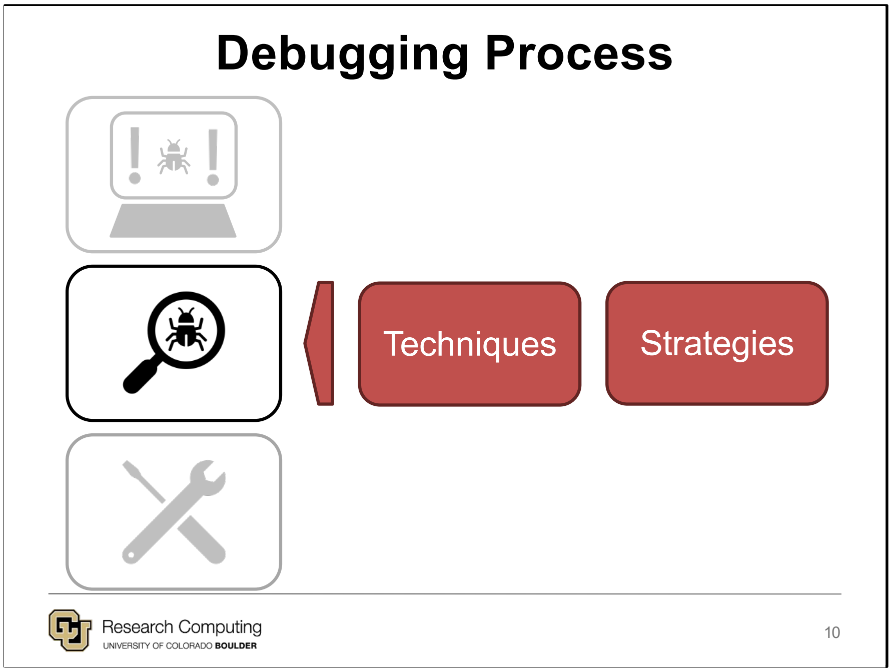
This is the final, and often easiest, step of the debugging process.



Debugging Process

Very Challenging!!

The most challenging part of debugging is this second step – finding the error.



Debugging Process

Techniques

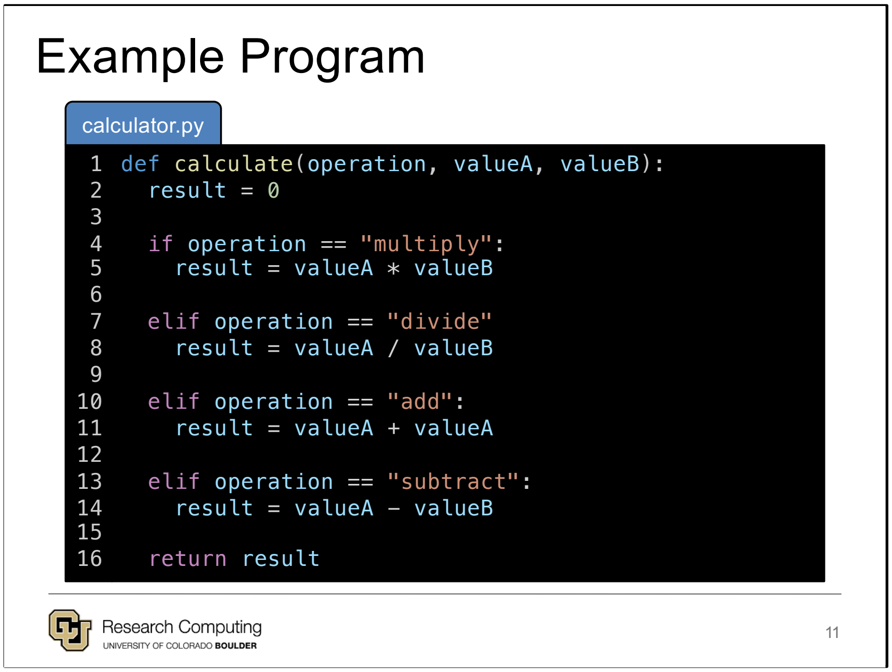
Strategies

Research Computing
UNIVERSITY OF COLORADO BOULDER

10

In this workshop, we will focus on different debugging strategies and techniques that can make it easier for you to find software errors.

While this workshop will be taught with Python, it is important to know that these strategies and techniques can be easily transferred to other programming languages and engineering domains.



Example Program

calculator.py

```python
def calculate(operation, valueA, valueB):
  result = 0

  if operation == "multiply":
    result = valueA * valueB

  elif operation == "divide"
    result = valueA / valueB

  elif operation == "add":
    result = valueA + valueA

  elif operation == "subtract":
    result = valueA - valueB

  return result
```

Research Computing
UNIVERSITY OF COLORADO BOULDER

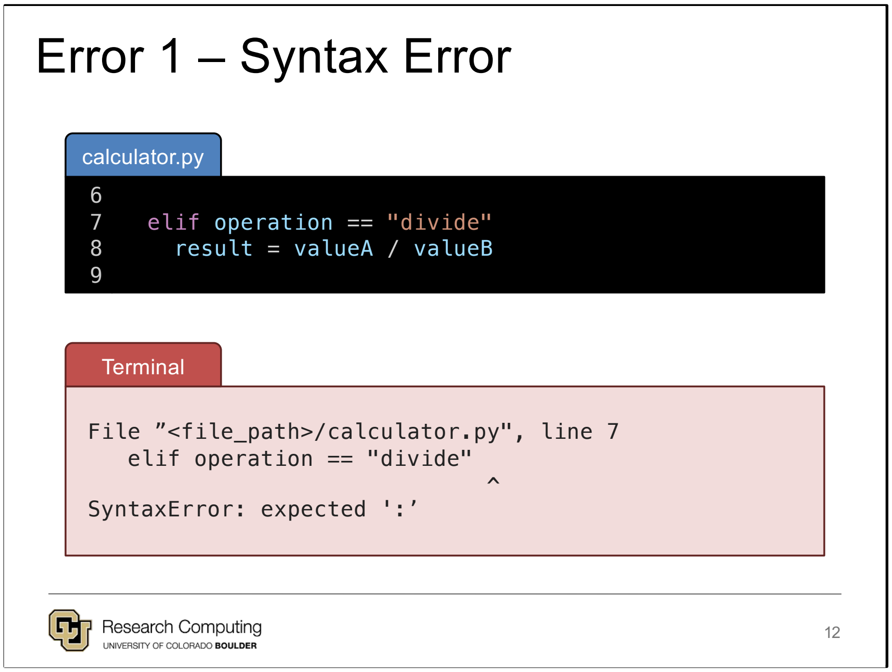
11

The example program for this workshop is called “calculator.py” and contains a simple method for calculating basic arithmetic operation.

This method contains three errors which we will work on finding and fixing together.

You can follow along by copy+pasting the calculator.py’s code into a new file on your personal workstation, VS Code (OnDemand), or use the free online IDE Trinket for Python:
https://trinket.io/python/cd7747ea1eaa

This file is also provided in this presentations Github Repo:
https://github.com/ResearchComputing/introduction_to_debugging_shortcourse



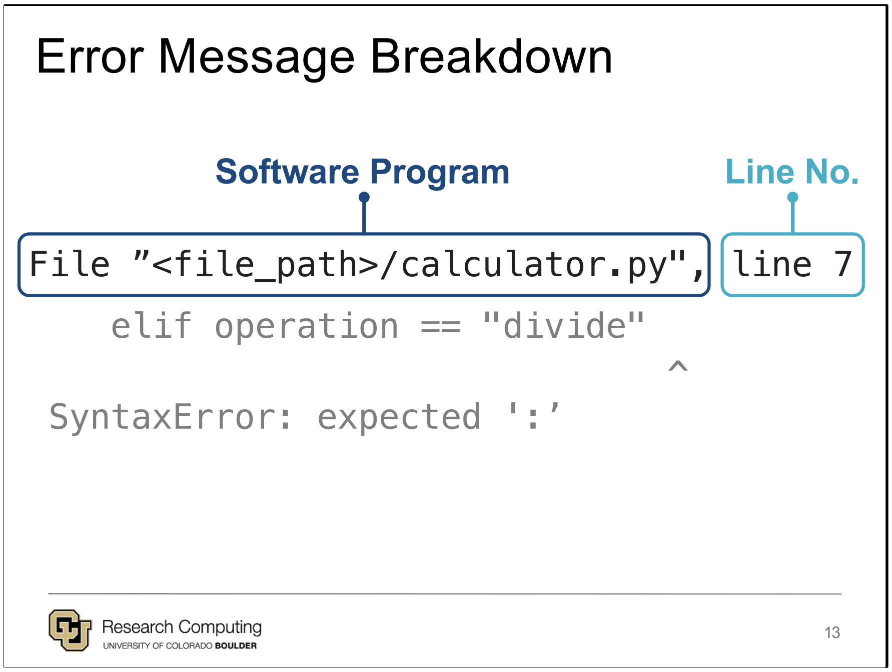
Error 1 – Syntax Error

calculator.py

```
6
7    elif operation == "divide"
8      result = valueA / valueB
9
```

Terminal

```
File ”<file_path>/calculator.py", line 7
   elif operation == "divide"
                              ^
SyntaxError: expected ':’
```

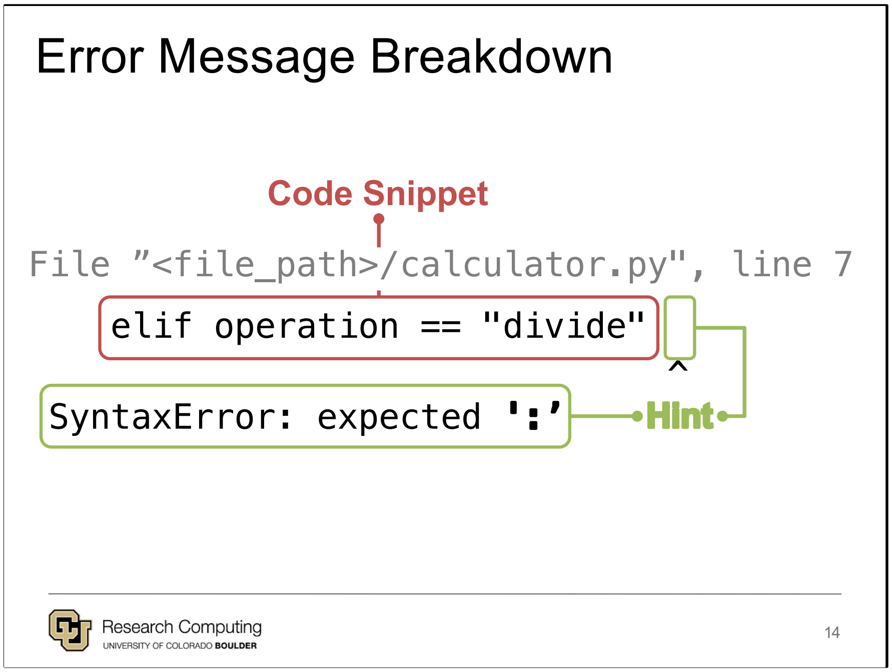


Error Message Breakdown

Software Program

Line No.

```
File ”<file_path>/calculator.py", line 7
    elif operation == "divide"
                              ^
SyntaxError: expected ':’
```



Error Message Breakdown

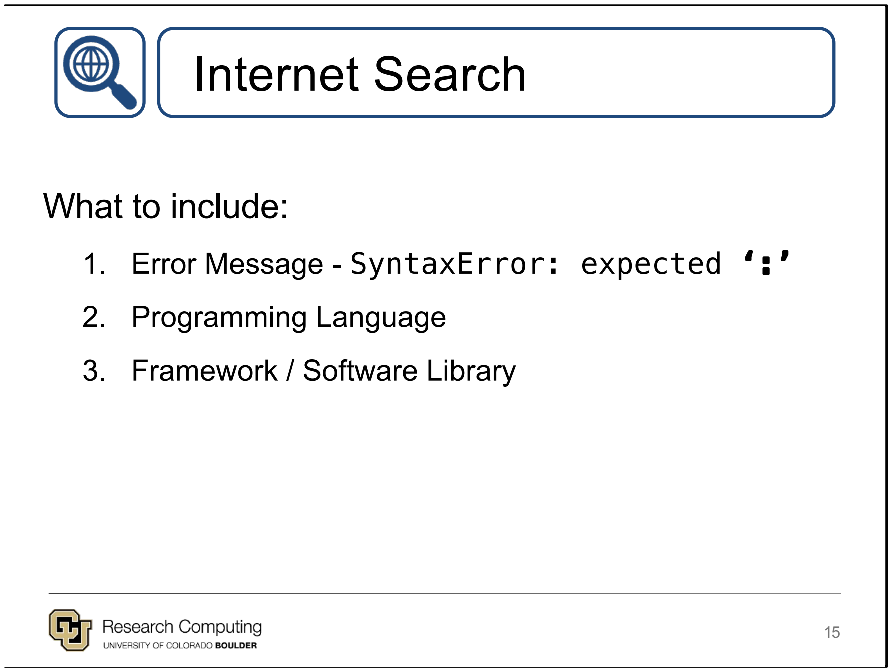
Code Snippet

```
File ”<file_path>/calculator.py", line 7
    elif operation == "divide"
                              ^
SyntaxError: expected ':'
```

Hint

Research Computing
UNIVERSITY OF COLORADO BOULDER

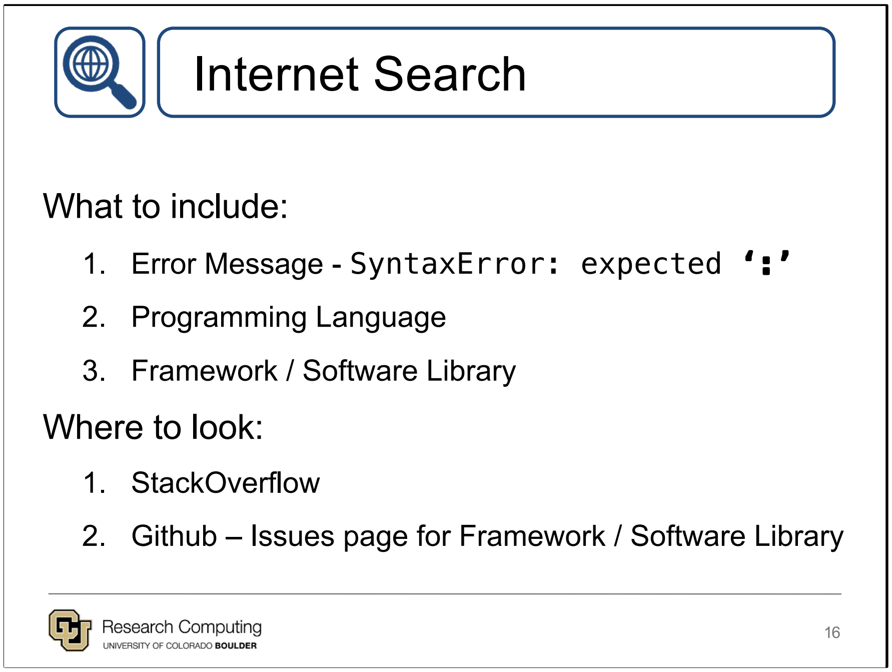
How to google Errors



Internet Search

What to include:

1. Error Message - `SyntaxError: expected ':'`
2. Programming Language
3. Framework / Software Library



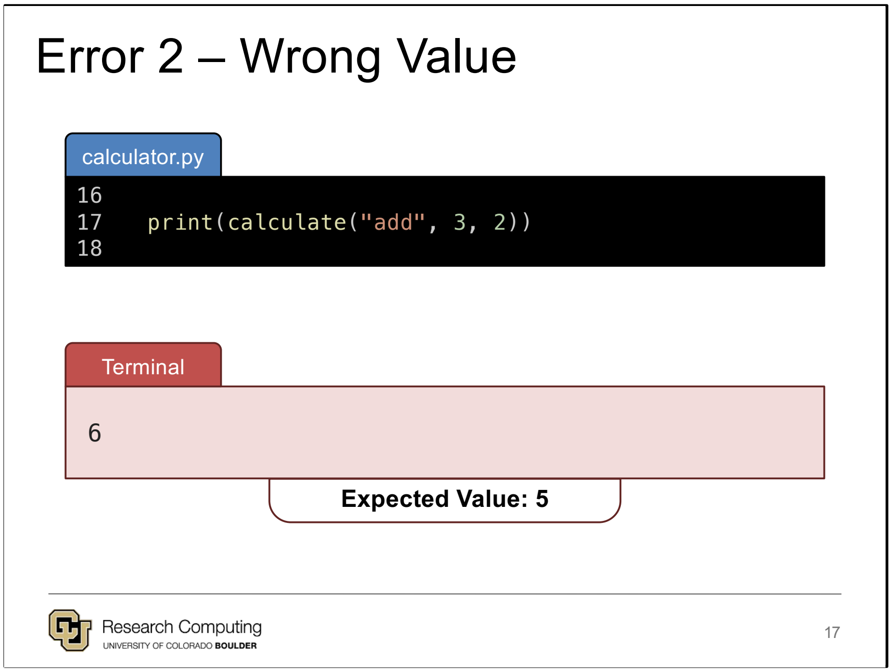
Internet Search

What to include:

1. Error Message - `SyntaxError: expected ':'`
2. Programming Language
3. Framework / Software Library

Where to look:

1. StackOverflow
2. Github – Issues page for Framework / Software Library



Error 2 – Wrong Value

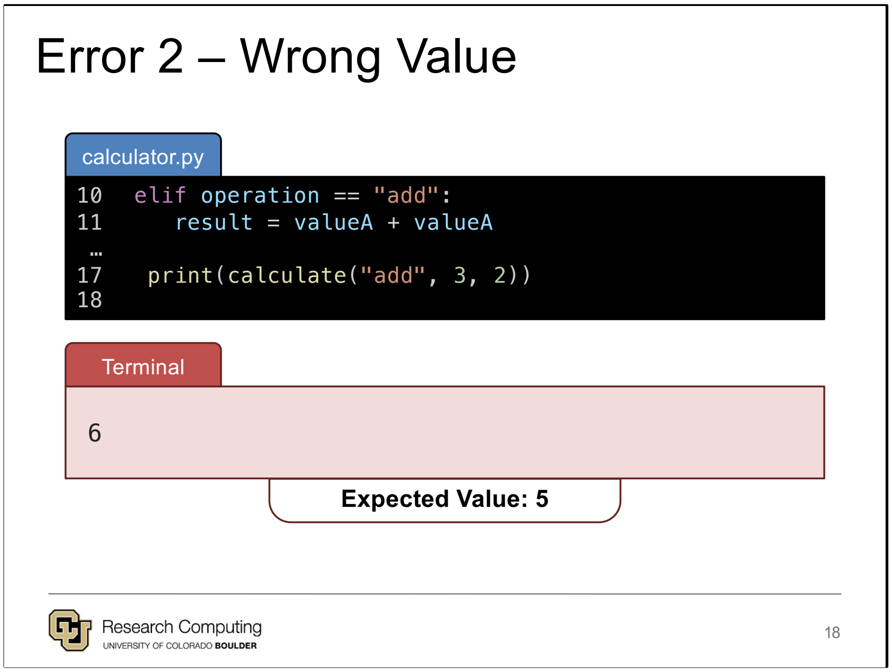
calculator.py

```python

print(calculate("add", 3, 2))

```

Terminal

```
6
```

Expected Value: 5



Error 2 – Wrong Value

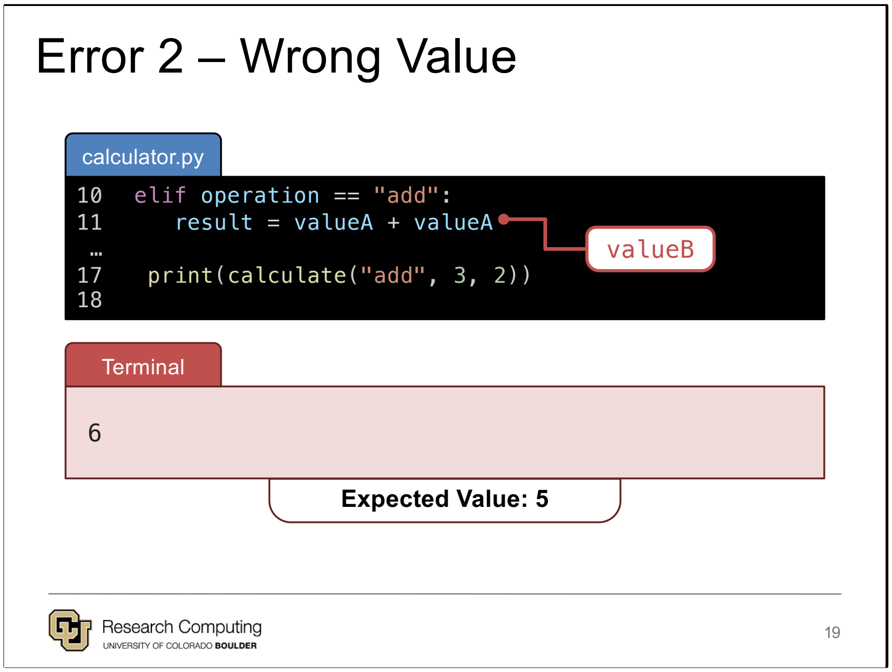
calculator.py

```
elif operation == "add":
    result = valueA + valueA
…
 print(calculate("add", 3, 2))

```

Terminal

```
6
```

Expected Value: 5



Error 2 – Wrong Value

calculator.py

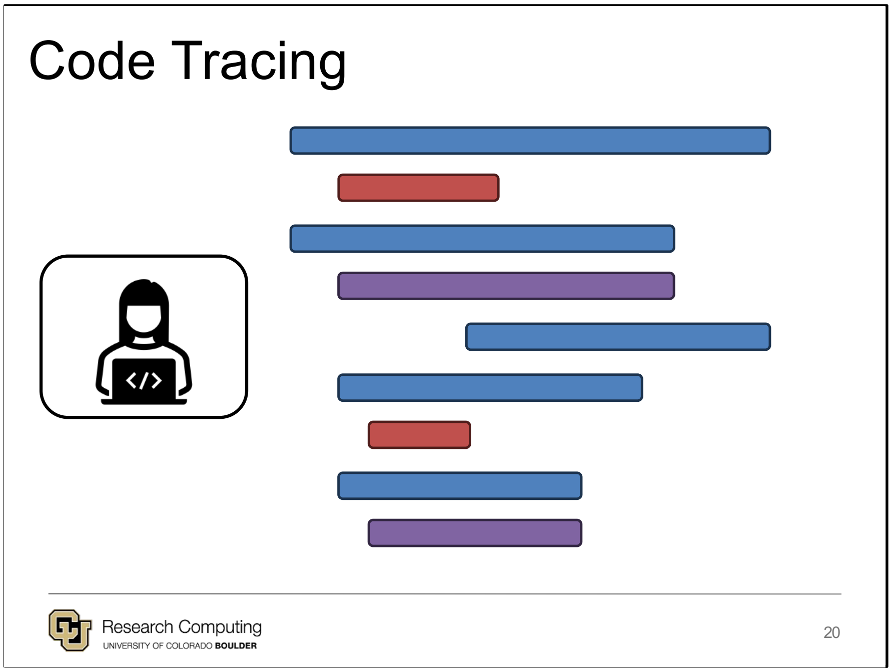
```
elif operation == "add":
    result = valueA + valueA
…
  print(calculate("add", 3, 2))

```

valueB

Terminal

```
6
```

Expected Value: 5

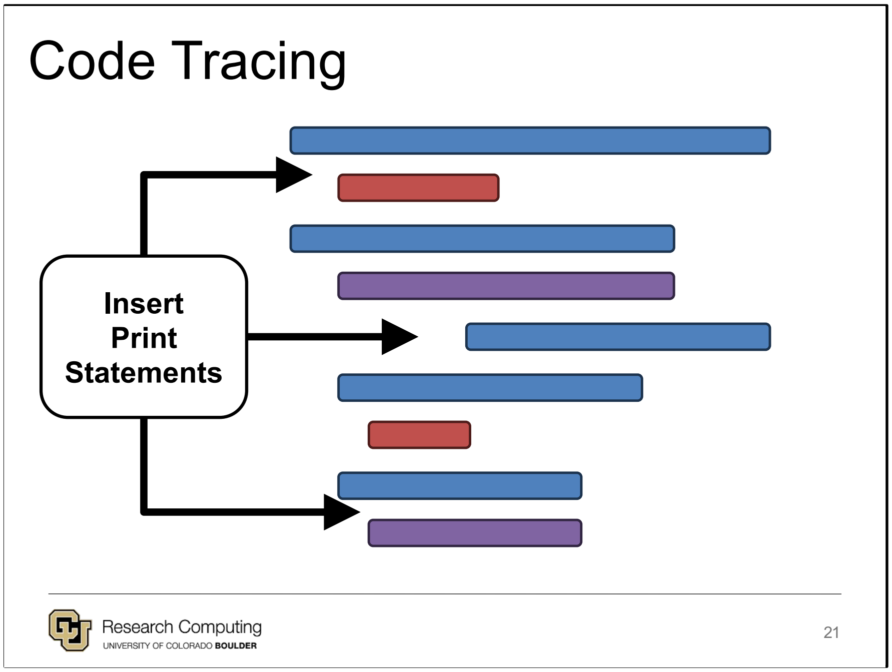


Code Tracing

Research Computing
UNIVERSITY OF COLORADO BOULDER

Mental Tracing – Reading the code line by line, helpful to read out-loud and/or add comments to explain what the system is doing.

Code Stepping w/ Print Statements – Use carefully placed print statements to observe the state of variables.



Code Tracing

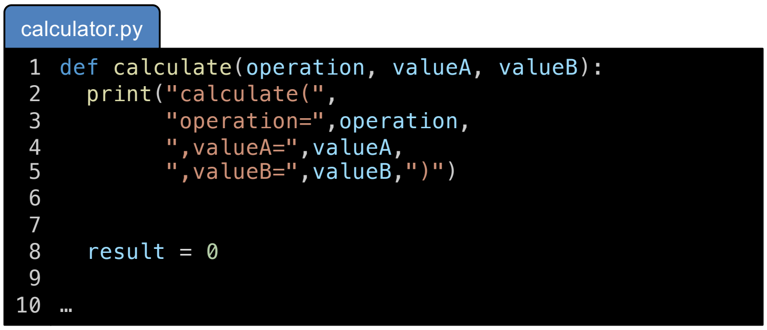
Insert Print Statements

Research Computing
UNIVERSITY OF COLORADO BOULDER

Mental Tracing – Reading the code line by line, helpful to read out-loud and/or add comments to explain what the system is doing.

Code Stepping w/ Print Statements – Use carefully placed print statements to observe the state of variables.

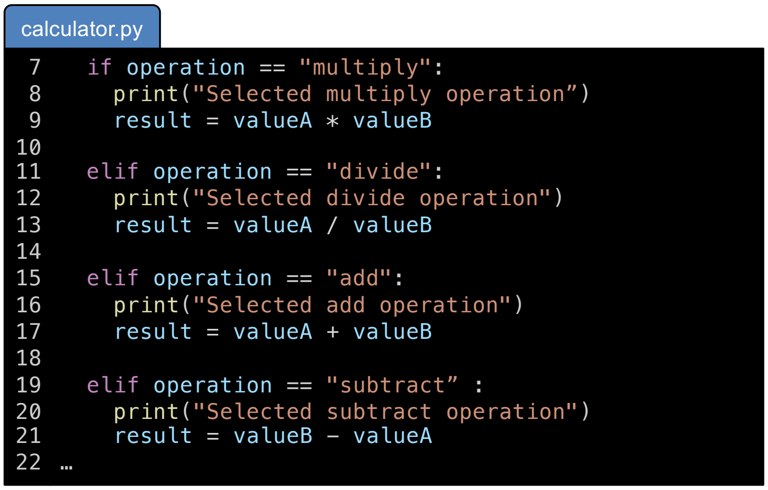
Strategic Print Statements (1)



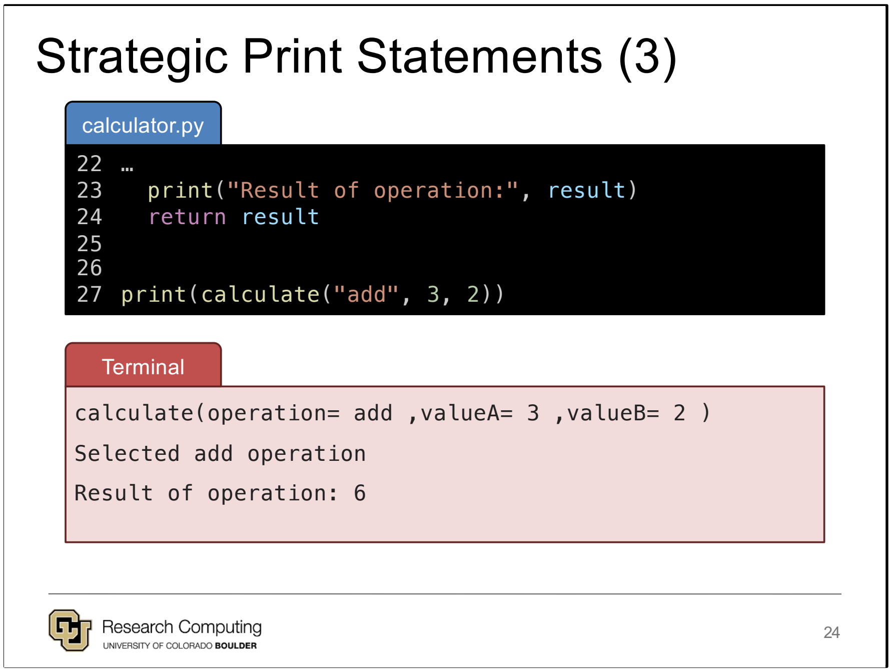
calculator.py

```python
def calculate(operation, valueA, valueB):
  print("calculate(",
        "operation=",operation,
        ",valueA=",valueA,
        ",valueB=",valueB,")")


  result = 0

…
```

Strategic Print Statements (2)



```
 if operation == "multiply":
  print("Selected multiply operation”)
  result = valueA * valueB

 elif operation == "divide":
  print("Selected divide operation")
  result = valueA / valueB

 elif operation == "add":
  print("Selected add operation")
  result = valueA + valueB

 elif operation == "subtract” :
  print("Selected subtract operation")
  result = valueB - valueA
…
```



Strategic Print Statements (3)

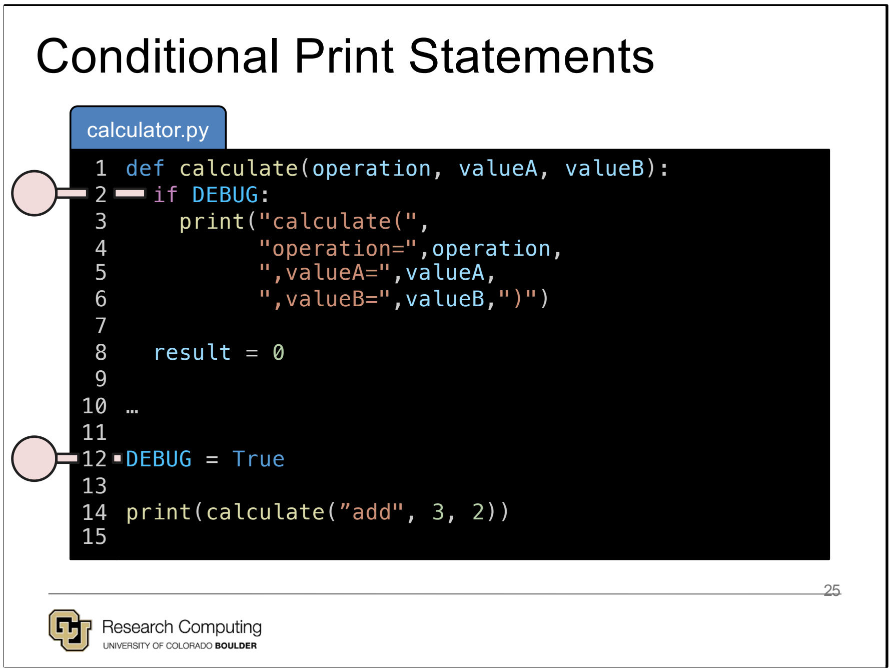
calculator.py

```python
…
    print("Result of operation:", result)
    return result


print(calculate("add", 3, 2))
```

Terminal

```
calculate(operation= add ,valueA= 3 ,valueB= 2 )
Selected add operation
Result of operation: 6
```

Research Computing
UNIVERSITY OF COLORADO **BOULDER**



Conditional Print Statements

calculator.py

```
def calculate(operation, valueA, valueB):
  if DEBUG:
    print("calculate(",
          "operation=",operation,
          ",valueA=",valueA,
          ",valueB=",valueB,")")

  result = 0

…

DEBUG = True

print(calculate(”add", 3, 2))

```

25

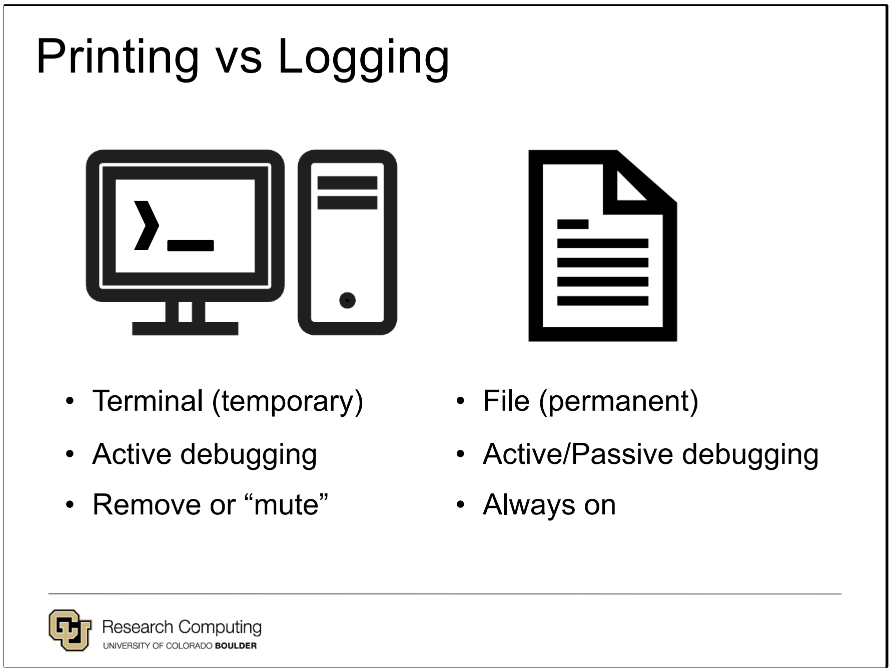
Research Computing
UNIVERSITY OF COLORADO BOULDER

Discuss benefits of different types of print statements/approaches

Show how we can make a DEBUG Boolean, which can turn debug statements on/off – great for long term coding projects.

Provide a general format for debug statements and log statements.
- Make sure to provide enough but not too much information
- If dealing with confidential or sensitive data – Be very careful! Discuss with domain experts and the Secure Research Computing team for guidance to ensure you are not leaking provide information!

Format:
TimeStamp [CATEGORY/TAG] : ERROR MESSAGE



Printing vs Logging

- Terminal (temporary)
- Active debugging
- Remove or “mute”

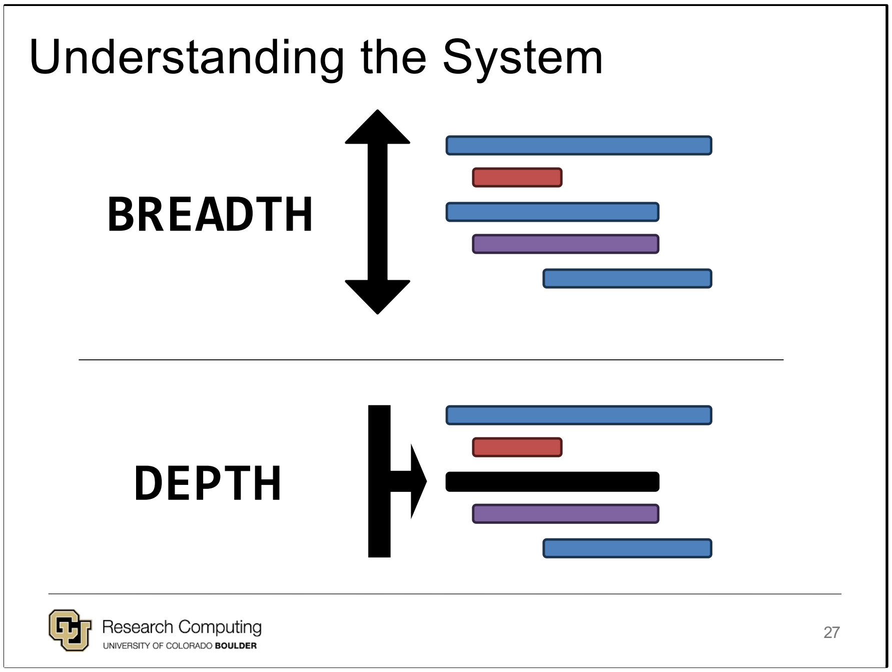
- File (permanent)
- Active/Passive debugging
- Always on

Research Computing
UNIVERSITY OF COLORADO BOULDER

Create a debugging method

Create 3-4 potential categories for your log messages

Write them down in a readme file or as comment in the method header.



Understanding the System

BREADTH

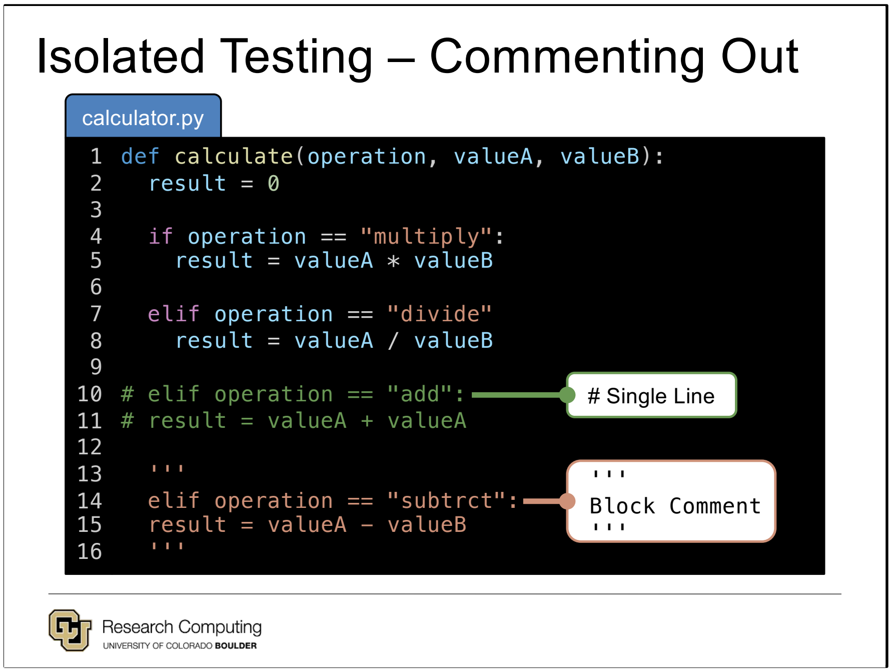
DEPTH

Research Computing
UNIVERSITY OF COLORADO BOULDER

27

Update this note to highlight the pitfall of diving into debugging (i.e. depth search). Try to always first familiarize yourself with the system with a breadth search where you consider the organization and structure of your project's different components.

Combine in discussion of forward/backward reasoning.



Isolated Testing – Commenting Out

calculator.py

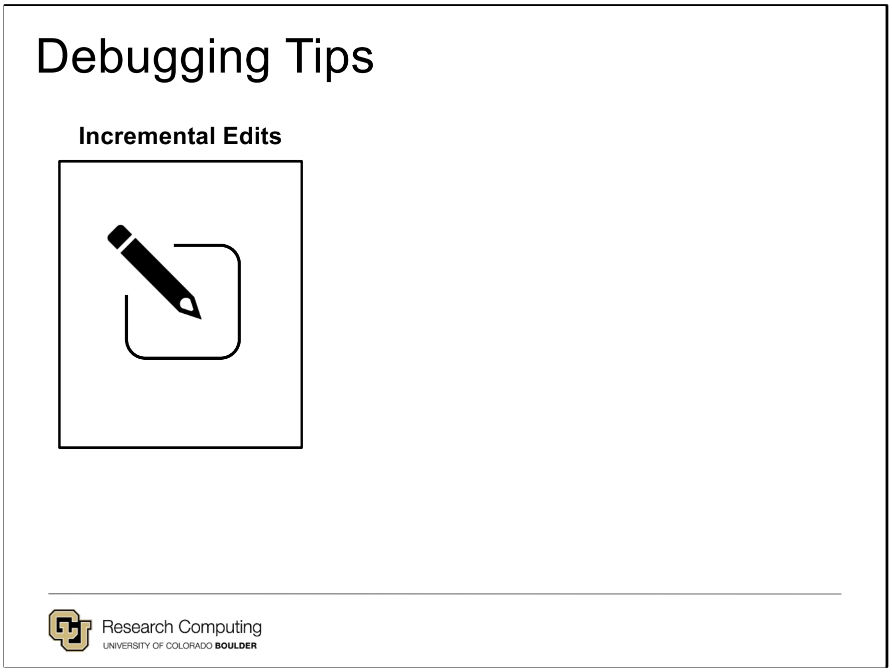
```python
def calculate(operation, valueA, valueB):
  result = 0

  if operation == "multiply":
    result = valueA * valueB

  elif operation == "divide"
    result = valueA / valueB

# elif operation == "add":
# result = valueA + valueA

  '''
  elif operation == "subtrct":
  result = valueA - valueB
  '''
```

```
# Single Line
```

```
'''
Block Comment
'''
```

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

Note: Be mindful of indentation when using triple quotes as a “block comment” - this is creating a String literal which need to follow the same indentation rules of the surrounding code chunk.



Debugging Tips

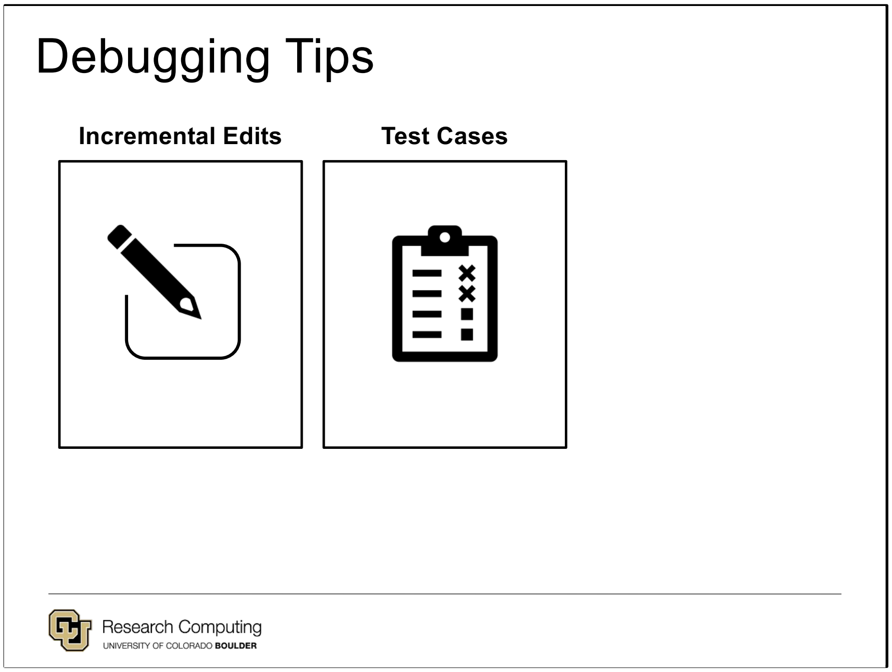
Incremental Edits

Research Computing
UNIVERSITY OF COLORADO BOULDER

Incremental Edits
Tracking changes can be challenging – especially if you make a large number within a short period of time (remember there are limits to “undo, ctrl+Z”!)

So, whenever possible, only make one change at a time, test it, and then decide if you want to roll it back or keep it.

Software like Github can help, but isn’t really intended for tracking the constant back-and-forth updates often made while debugging. So it’s best not to rely on a tool, but rather focus on having a methodical process for keeping your debugging changes in check.



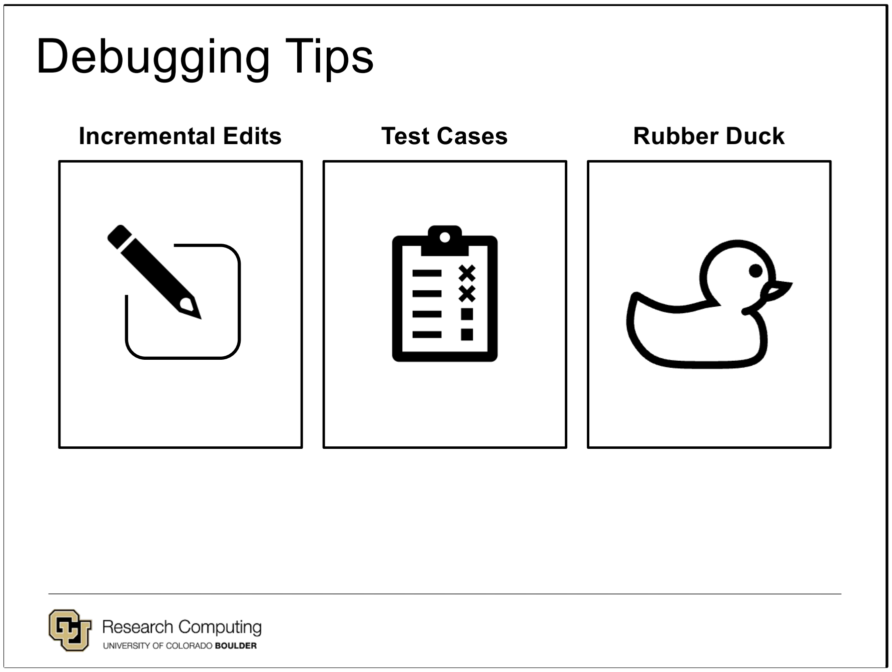
Debugging Tips

Incremental Edits

Test Cases

Research Computing
UNIVERSITY OF COLORADO BOULDER

Test Cases: **add notes**



Debugging Tips

Incremental Edits

Test Cases

Rubber Duck

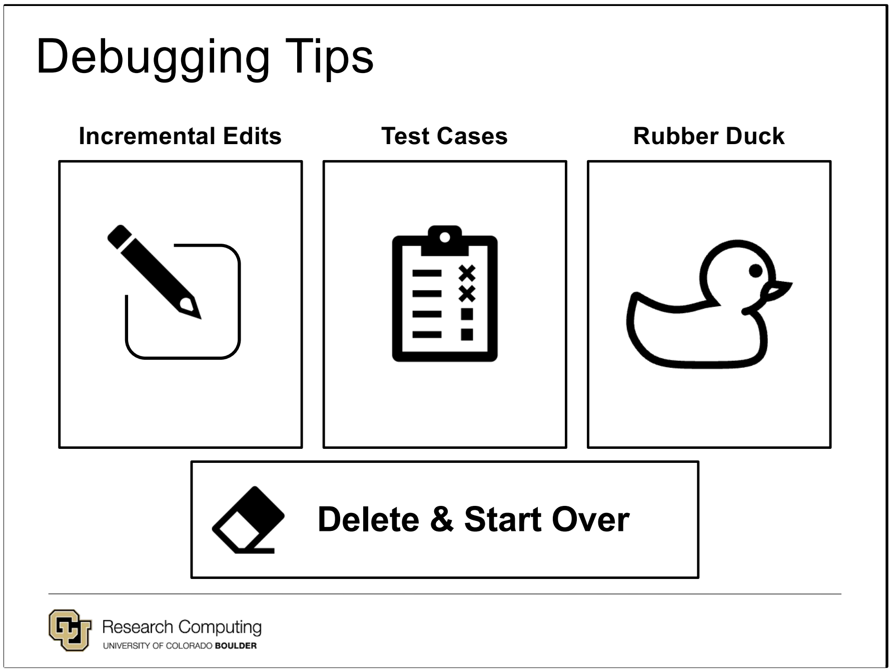
Research Computing
UNIVERSITY OF COLORADO **BOULDER**

Rubber Duck Debugging:
Wikipedia Article: https://en.wikipedia.org/wiki/Rubber_duck_debugging

Explain, out loud, to your rubber duck friend what your program is intended to do and the bug you are experiencing.

In detail, explain what each part and/or line-of-code in your program does to your duck.

By discussing your work out-loud, you may reach a “Eureka!” moment, where the issue you were overlooking before become crystal clear.



Debugging Tips

Incremental Edits

Test Cases

Rubber Duck

Delete & Start Over

Research Computing
UNIVERSITY OF COLORADO BOULDER

Delete & Start Over:
Last Resort – helpful but can be dangerous, since you never identified the true cause of the bug. This means the bug could still pop-up after re-writing your code or arise later-on leading to a frustrating cycle of write-rewrite

Survey and feedback

http://tinyurl.com/curc-survey18