Debugging Introduction

Why is this important?

Your code is going to break. It's inevitable. You're going to have to debug your code. It's a skill that you'll need to learn.

This is going to allow you to find and fix errors.

This is a great technique as well to understand the behaviour of your code at a specific line while it's running which is going to help you build code faster and fix things faster.

What are we going to do?

We're going to step through `hi_or_low_debug.py` and see how we can use `breakpoint()` to pause execution and debug our code.

Steps

1. Let's talk about what you can do when you add `breakpoint()` to a line of your code to pause execution.

- Once you pause execution you can inspect the value of the variables, change variables, and run code.
- Here's a table to show some functionality that you can refer back to so that you can see some of the commands you can do when you're debugging.

PDB (python debugger commands)	**Action**
n or next	Run the next line of code
s or step	Step into current line (usually a function call, we'll take a further look at this in the future)
r or return	Return from the current function (We'll take a further look at this in the future)
l or list	List the surrounding code lines
ll	List the more of the surrounding code lines
interact	Starts an interactive Python interpreter (REPL)
c or continue	Continue running (until a breakpoint or exit)
b or break	Set a breakpoint for a specific line or function
!	Special prefix to say "run this as Python code"
pp	Pretty-print the value of a Python expression

- source

2. Let's add a `breakpoint()` to our code and step through it.

- At the top of the `hi_or_bye.py` file add `breakpoint()` to the top of the file, put it at the same place as shown below (so we can step through it together.)

```
import random

# Generate a random integer between a range
random_number = random.randint(1, 100)
breakpoint()
# Get the user input
user_input = input("Guess a number between 1 and 100: ")
user_guess = int(user_input)
print(F"user guess {user_guess}")

# Rest of code removed for brevity
```

- Now you've run you're code you'll see that it's paused at the `breakpoint()` line and you have a `(Pdb)` prompt. This is how it should look.

```
$ python hi_or_low_debug.py
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro\hi_or_low_debug.py(7)<module>()
-> user_input = input("Guess a number between 1 and 100: ")
(Pdb)
```

- Note you'll be adding the commands in the above table as well as running some code in this prompt at a given line to see what happens.

2. Let's start by running `l` (or `list`) in the prompt

- This is going to show you the lines around what is currently being executed. Let's `l` and run it and see what happens.

```
$ python hi_or_low_debug.py
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro\hi_or_low_debug.py(7)<module>()
-> user_input = input("Guess a number between 1 and 100: ")
(Pdb) l
  2
  3     # Generate a random integer between a range
  4     random_number = random.randint(1, 100)
  5     breakpoint()
  6     # Get the user input
  7  -> user_input = input("Guess a number between 1 and 100: ")
  8     user_guess = int(user_input)
  9     print(F"user guess {user_guess}")
```

```
 10
 11     # Ask the user if they think they are higher or lower than the
number
 12     high_low_input = input("Do you think you are higher or lower than
the number? (h/l) ")
(Pdb)
```

- You can see in the above by the first `(Pdb)` prompt we've typed `l` and pressed enter.
- Let's talk about the output.
    - You see `->` on line `7`. This is the line that is currently being executed in the debugging.
    - If you take a look the `hi_or_low_debug.py` file you'll see that the line numbers add up and you're at the next "executable" line in the code.
    - Let's go to the next step.

3. Let's start by running `c` (or `continue`) in the prompt.

- This is going to just allow the code to continue running until it hits another breakpoint or the end of the file/program.
- Let's `c` and run it and see what happens.

```
$ python hi_or_low_debug.py
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro\hi_or_low_debug.py(7)<module>()
-> user_input = input("Guess a number between 1 and 100: ")
(Pdb) l
  2
  3     # Generate a random integer between a range
  4     random_number = random.randint(1, 100)
  5     breakpoint()
  6     # Get the user input
  7  -> user_input = input("Guess a number between 1 and 100: ")
  8     user_guess = int(user_input)
  9     print(F"user guess {user_guess}")
 10
 11     # Ask the user if they think they are higher or lower than the
number
 12     high_low_input = input("Do you think you are higher or lower than
the number? (h/l) ")
(Pdb) c
Guess a number between 1 and 100: 1
user guess 1
Do you think you are higher or lower than the number? (h/l) l
You are correct it's low!
the random number was 77
```

- You can see that in the second `(Pdb)` prompt we've typed `c` and pressed enter.
- Let's talk about the output.
    - There are no extra `breakpoint()`s in our file so our program just runs until it was done.

4. Let's go line by line running `n` (or `next`) in the prompt.

- The command `n` allows you to run the next line of code and allow you to inspect what the program is doing.
    - What we're going to do is we're going to run `n` and then `l` to see what the next line of code is and then we're going to run `n` again.
    - We're also going to see what the values of `user_input` and `user_guess` are. So we can ensure we have the right pieces going on.
- Let's take a look at the output after we run it and see what happens.

```
$ python hi_or_low_debug.py
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro\hi_or_low_debug.py(7)<module>()
-> user_input = input("Guess a number between 1 and 100: ")
(Pdb) l
  2
  3     # Generate a random integer between a range
  4     random_number = random.randint(1, 100)
  5     breakpoint()
  6     # Get the user input
  7  -> user_input = input("Guess a number between 1 and 100: ")
  8     user_guess = int(user_input)
  9     print(F"user guess {user_guess}")
 10
 11     # Ask the user if they think they are higher or lower than the
number
 12     high_low_input = input("Do you think you are higher or lower than
the number? (h/l) ")
(Pdb) n
Guess a number between 1 and 100: 25
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro\hi_or_low_debug.py(8)<module>()
-> user_guess = int(user_input)
(Pdb) user_input
'25'
(Pdb) user_guess
*** NameError: name 'user_guess' is not defined
(Pdb) c
user guess 25
Do you think you are higher or lower than the number? (h/l) h
You are wrong and this is rigged...
the random number was 71
```

- You can see here that we've run a few commands in our debugger and let's talk about it.
    - In the first `(Pdb)` prompt we've typed `l` and pressed enter. This is going to show the lines around the code (remember you can run `ll` to see even more lines!)
        - We see here that we're at line `7` but we haven't executed the line.
    - In the second `(Pdb)` prompt we've typed `n` and pressed enter. This is going to run the next line of code.

        - We see that we're now prompted for the input of our program and we enter `25` and press enter.
    - In the third `(Pdb)` prompt we're checking for the variable `user_input` and we see that it's a string.
        - Now as well we didn't do an `l` here but we know that we're on line `8` because we step through the code piece by piece.
    - In the fourth `(Pdb)` prompt we're checking for the variable `user_guess` and we see that it's not defined.
        - This is because we haven't executed the line yet! We're still on line `8` but we haven't executed it yet.
    - In the fifth `(Pdb)` prompt we've typed `c` and pressed enter. This is going to run the rest of the code until it's done.
        - The next line of code we're convert the `user_input` to an integer and assign it to `user_guess` and our code just runs until it's done.

- Great you've debugged your first file! Now let's move our `breakpoint()` further down the file to see how we can observe how we can debug `if` statements

5. Let's move our `breakpoint()` to before the `if` statement and step through it.

- right after we define `result` let's add a `breakpoint()` and run our code.

```
# other code removed for brevity.

# Ask the user if they think they are higher or lower than the number
high_low_input = input("Do you think you are higher or lower than the
number? (h/l) ")

result = ""
breakpoint()
# Compare the user input to the random number
if user_guess == random_number:
    result = "correct"
elif user_guess > random_number:
    result = "high"
elif user_guess < random_number:
    result = "low"
else:
    result = "error"

# Rest of the code removed for brevity
```

- Let's take run our code and understand what's going on with our `if` statement in the code.

```
$ python hi_or_low_debug.py
Guess a number between 1 and 100: 97
user guess 97
Do you think you are higher or lower than the number? (h/l) l
```

```
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro\hi_or_low_debug.py(17)<module>()
-> if user_guess == random_number:
(Pdb) user_guess
97
(Pdb) random_number
60
(Pdb) l
 12     high_low_input = input("Do you think you are higher or lower than
the number? (h/l) ")
 13
 14     result = ""
 15     breakpoint()
 16     # Compare the user input to the random number
 17  -> if user_guess == random_number:
 18         result = "correct"
 19     elif user_guess > random_number:
 20         result = "high"
 21     elif user_guess < random_number:
 22         result = "low"
(Pdb) n
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro\hi_or_low_debug.py(19)<module>()
-> elif user_guess > random_number:
(Pdb) l
 14     result = ""
 15     breakpoint()
 16     # Compare the user input to the random number
 17     if user_guess == random_number:
 18         result = "correct"
 19  -> elif user_guess > random_number:
 20         result = "high"
 21     elif user_guess < random_number:
 22         result = "low"
 23     else:
 24         result = "error"
(Pdb) n
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro\hi_or_low_debug.py(20)<module>()
-> result = "high"
(Pdb) l
 15     breakpoint()
 16     # Compare the user input to the random number
 17     if user_guess == random_number:
 18         result = "correct"
 19     elif user_guess > random_number:
 20  ->     result = "high"
 21     elif user_guess < random_number:
 22         result = "low"
 23     else:
 24         result = "error"
 25
(Pdb) n
> c:\users\dmouris\sdev1001-master-course\debugging-with-
```

```
breakpoints\debugging_intro\hi_or_low_debug.py(27)<module>()
-> if result == "high" and high_low_input == "h":
(Pdb) l
 22         result = "low"
 23     else:
 24         result = "error"
 25
 26     # Is user correct?
 27  -> if result == "high" and high_low_input == "h":
 28         print("You are correct it's high!")
 29     elif result == "low" and high_low_input == "l":
 30         print("You are correct it's low!")
 31     else:
 32         print('You are wrong and this is rigged...')
(Pdb) c
You are wrong and this is rigged...
the random number was 60
```

- Let's take a look at what's going on here as we debug the code after our `breakpoint()`
  - first `(Pdb)` we're taking a look at what our `user_guess` variable is and you see that it shows `97` on the following line.
  - second `(Pdb)` we're taking a look at what our `random_number` variable is and you see that it shows `60` on the following line.
  - third `(Pdb)` we're write `l` and you can see that we're on line `17` and we haven't executed the line yet.
    - You see here that we're going to check if `user_guess` is equal to `random_number`, it isn't, so it should go to the next elif statement on line `19`.
  - fourth `(Pdb)` we write `n` to step to the next line of execution of our code.
  - fifth `(Pdb)` we write `l` and you can see that we are on line `19` of the code (we haven't executed this line yet.)
    - we're checking on the line if `user_guess` is greater than `random_number` and that is `True`, so we should go to the next indented line of code on line `20`.
  - sixth `(Pdb)` we write `n` to step to the next line of execution of our code.
  - seventh `(Pdb)` we write `l` and we can see that we're on line `20`
    - this is great! We're on the line we want and we can know that our code is going to skip the rest of the `if` statement because one path was successful.
  - eighth `(Pdb)` we write `n` to step to the next line of execution of our code.
  - ninth `(Pdb)` we write `l` and we can see that we're on line `27`
    - this means that the code has ignored the rest of the if statement because one path was successful.
  - tenth `(Pdb)` we write `c` to continue the execution of our code.
    - we see that the code has run until it's done and we see that we're wrong and the random number was `60`.

6. Let's change the value of `user_guess` to exactly the value of `random_number` and see how our program behaves!

- You can change the value of variables in our debugger and it's really useful if you want to check a condition that is hard to reproduce. Let's take a look at how we can do that.

```
$ python hi_or_low_debug.py
Guess a number between 1 and 100: 60
user guess 60
Do you think you are higher or lower than the number? (h/l) h
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro\hi_or_low_debug.py(17)<module>()
-> if user_guess == random_number:
(Pdb) random_number
89
(Pdb) user_guess = random_number
(Pdb) l
 12     high_low_input = input("Do you think you are higher or lower than
the number? (h/l) ")  13
 14     result = ""
 15     breakpoint()
 16     # Compare the user input to the random number
 17  -> if user_guess == random_number:
 18         result = "correct"
 19     elif user_guess > random_number:
 20         result = "high"
 21     elif user_guess < random_number:
 22         result = "low"
(Pdb) n
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro\hi_or_low_debug.py(18)<module>()
-> result = "correct"
(Pdb) l
 13
 14     result = ""
 15     breakpoint()
 16     # Compare the user input to the random number
 17     if user_guess == random_number:
 18  ->     result = "correct"
 19     elif user_guess > random_number:
 20         result = "high"
 21     elif user_guess < random_number:
 22         result = "low"
 23     else:
(Pdb) c
You are wrong and this is rigged...
the random number was 89
```

- Let's take a look at the `(Pdb)` prompts to see what's going on.
  - Before the `(Pdb)`
    - We set the input `user_input` to `"60"` which `user_guess` is going to be the integer `60`.
  - first `(Pdb)` we're taking a look at what the value of `random_number` is and you see that it shows `89` on the following line.

    - second `(Pdb)` we're assigning the value of `random_number` to `user_guess` so that we can test the `if` statement if the values are equal.
    - third `(Pdb)` we're just checking which line we're on and we're at the `if` statement.
    - fourth `(Pdb)` we write `n` to step to the next line of execution of our code.
    - fifth `(Pdb)` we write `l` and see that the value of the variable `result` is now `"correct"` and we know that the `if` statement was `True`.
    - sixth `(Pdb)` we write `c` so that our code can continue running until it's done.
    - After the `(Pdb)`
        - We see that our code tells us it's wrong! We should fix this bug!

7. Fix the bug in the previous step!

- In our `high_low_debug.py` file, let's add a condition to give the user a message if they are correct.

```
# other code removed for brevity.

# Is user correct?
if result == "correct":
    print("You are correct! You must be cheating or something!")
elif result == "high" and high_low_input == "h":
    print("You are correct it's high!")
elif result == "low" and high_low_input == "l":
    print("You are correct it's low!")
else:
    print('You are wrong and this is rigged...')

print(F"the random number was {random_number}")
```

- Let's use our debugger one last time to see how we can ensure we get this message.

```
$ python hi_or_low_debug.py
Guess a number between 1 and 100: 56
user guess 56
Do you think you are higher or lower than the number? (h/l) h
> c:\users\dmouris\sdev1001-master-course\debugging-with-
breakpoints\debugging_intro_start\hi_or_low_debug.py(17)<module>()
-> if user_guess == random_number:
(Pdb) random_number
36
(Pdb) user_guess = random_number
(Pdb) c
You are correct! You must be cheating or something!
the random number was 36
```

- Let's take a look at the `(Pdb)` prompts to see what's going on.
    - Before the `(Pdb)`
        - We set the input `user_input` to `"56"` which `user_guess` is going to be the integer `56`.

- first `(Pdb)` we're taking a look at what the value of `random_number` is and you see that it shows `36` on the following line.
- second `(Pdb)` we're assigning the value of `random_number` to `user_guess` so that we can test the `if` statement if the values are equal.
- third `(Pdb)` we're just checking which line we're on and we're at the `if` statement.
- fourth `(Pdb)` we write `c` so that our code can continue running until it's done.
- After the `(Pdb)`
    - We see that our code tells us it's correct! We should fix this bug!

Conclusion

You can see already that using `breakpoint()` has helped us debug a bit in our code.

We're going to use this extensively in the future, and this is a great tool to see and understand what your program is doing if you're stuck.