How to Make an App for Beginners

Module 3
Lesson 7
Cheatsheet

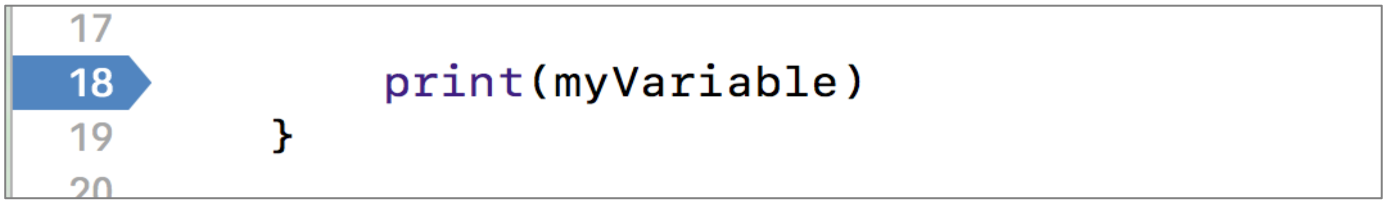
Logging:

Use the print statement to output variables into the console window. Do this pre-emptively as you're coding so that you can test your assumptions.

```
print(myVariable)
```

Breakpoints:

Click on the gutter to add or disable. To remove, click and drag it off the gutter. When stopped at a breakpoint, you can resume by clicking the "continue" button. See **Tracing**. While execution is paused, use "**po**" command in the console window to output variables and objects.

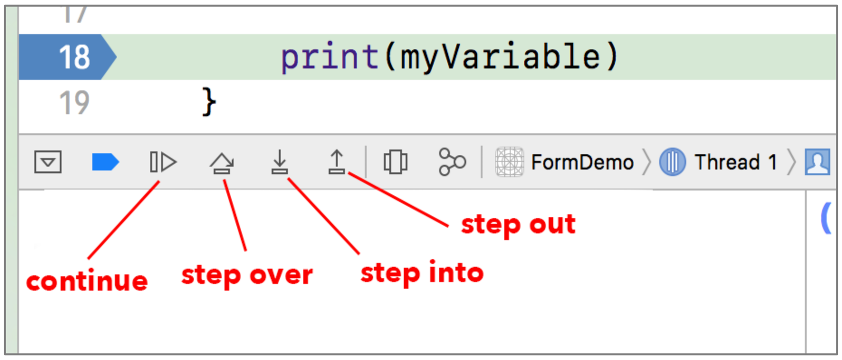


```
        print(myVariable)
    }
```

Tracing:

When execution is paused, use these controls to execute the code line by line.

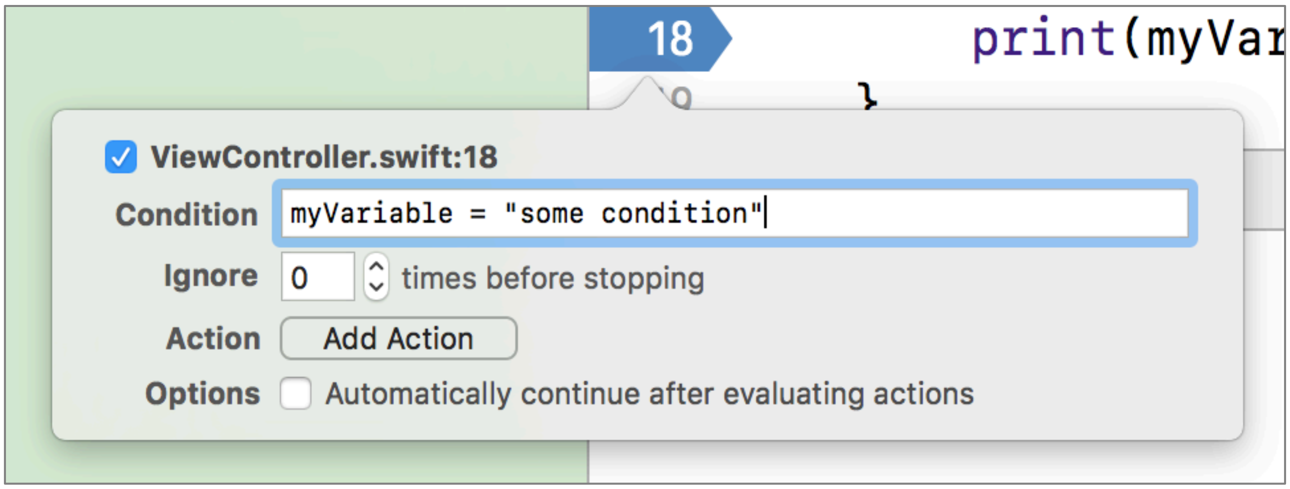
- Continue will resume execution until it hits the next breakpoint
- Step over will advance to the next line of code in the same scope
- Step into will go deeper such as INTO a method, property etc
- Step out will go back to the scope you were just at



```
        print(myVariable)
    }
```

Conditional Breakpoint:

Double click on a breakpoint to add a condition for it to stop the execution at that line of code. Your condition can include any variables, properties etc that have already been declared up to that line of code.



```
print(myVar
}
```

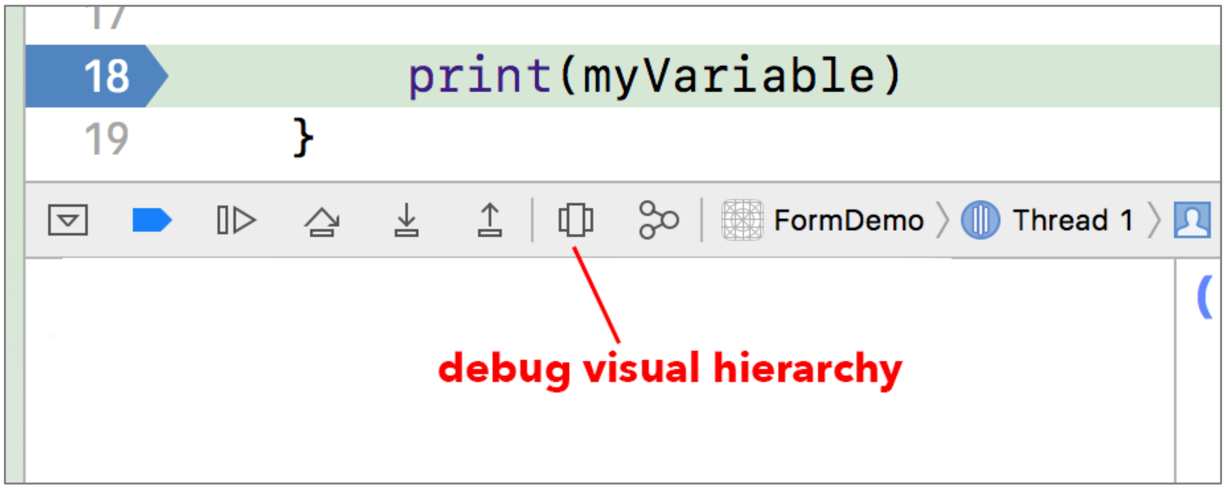
Condition: `myVariable = "some condition"`

Ignore: 0 times before stopping

Options: ☐ Automatically continue after evaluating actions

Visual Hierarchy:

When the execution is stopped at a breakpoint, you can tap on the “Debug Visual Hierarchy” button to take a look at how your views are arranged.



```
print(myVariable)
}
```

1. **Logging:** Use the print statement to output data to the console window. Use this to test your assumptions about expected variable values.

   This is helpful for determining WHERE the problem is occurring. You might not find the exact line of code but finding even a general area where the problem starts is important. This gives you a starting point to utilize breakpoints and tracing to find the EXACT line of offending code.

   For example, if you're not seeing what you're expecting on the screen, then plant print statements along the path of execution and run the app so you can see at which point the output doesn't meet your expectation.

2. **Breakpoints:** Let's you pause the execution of code at a specific line of code. Add a breakpoint by clicking on the left "gutter" beside a line of code. (this is the same gutter where the line numbers are)

   Once you identify where the problem is happening with Logging, you can use a breakpoint to stop the execution a few lines of code BEFORE where the problem is happening.

   Now as the execution is paused, you can take a deep look into various objects or variables using the "po" command in the console window.

3. **Tracing:** Once the execution is paused with breakpoints, you can trace your code line by line to see how the objects or variables are changing as each line of code. Your goal now is to find the EXACT line of code that is causing the problem.

4. **Conditional Breakpoints:** This is a useful tool when you suspect that the offending line of code is inside of a loop or if you suspect the code only happens under very specific conditions. To test your hypothesis, you can set a conditional breakpoint so that execution is only paused when those conditions are met.

5. **Inspect Visual Hierarchy:** Allows you to take a peek at how your visual elements are laid out. Useful for finding bugs with your layout where elements may not be arranged as expected.