Code Kata

- Given 2 positive int values, return the larger value that is in the range 10..20 inclusive, or return 0 if neither is in that range.
- max1020(11, 19) → 19
- max1020(19, 11) → 19
- max1020(11, 9) → 11
- max1020(9, 9) → 0

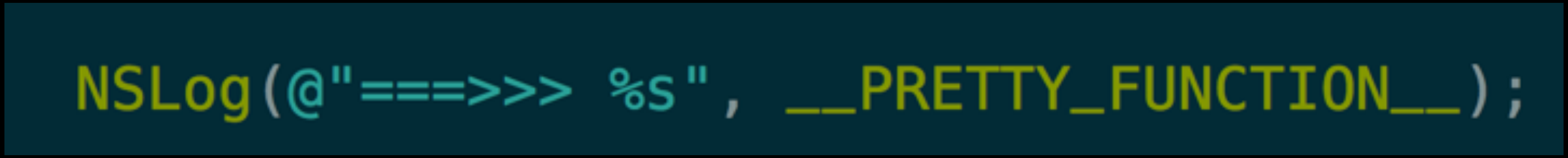
iOS Debugging

Overview

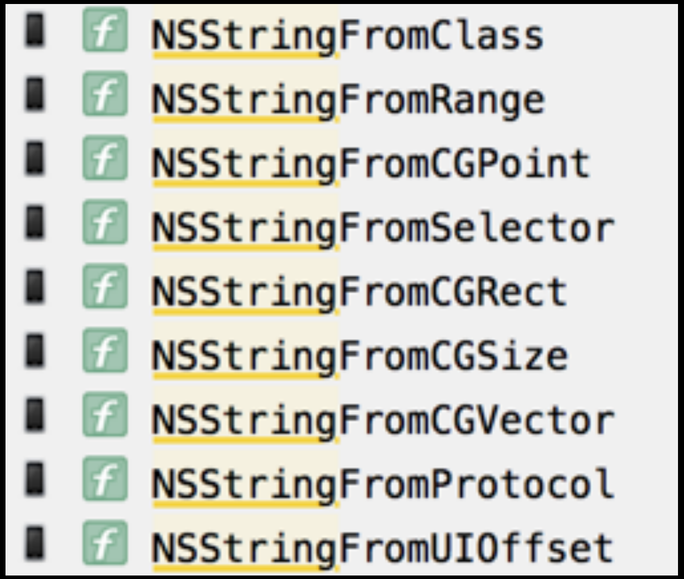
- Print statements
- Asserts
- Folding Ribbon
- Instruments
- Debugging Advice
- Practice Debugging
- Essential Tools
- Introducing LLDB commands

NSLog/Print

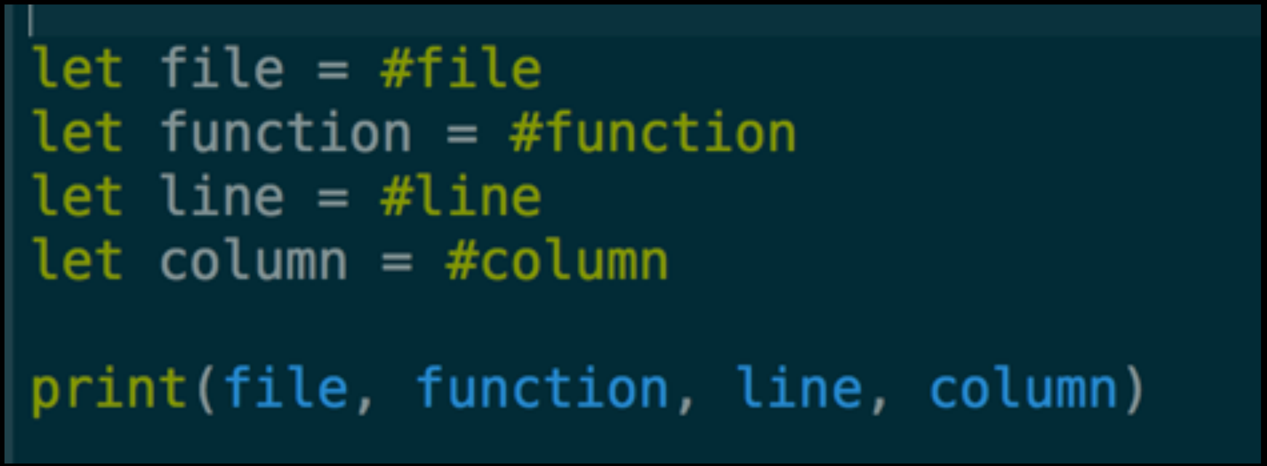
Some log tricks



```
NSLog(@"===>>> %s", __PRETTY_FUNCTION__);
```

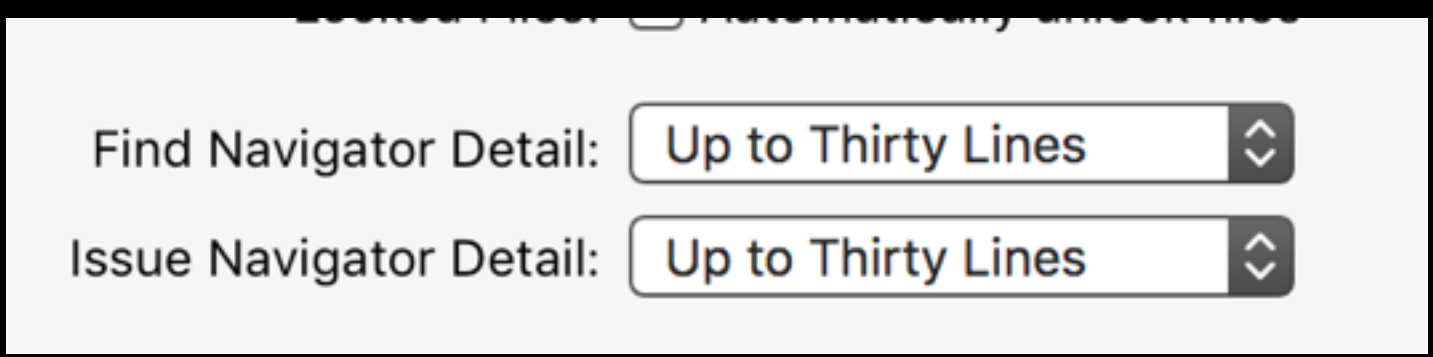


```
NSStringFromClass
NSStringFromRange
NSStringFromCGPoint
NSStringFromSelector
NSStringFromCGRect
NSStringFromCGSize
NSStringFromCGVector
NSStringFromProtocol
NSStringFromUIOffset
```



```
let file = #file
let function = #function
let line = #line
let column = #column

print(file, function, line, column)
```



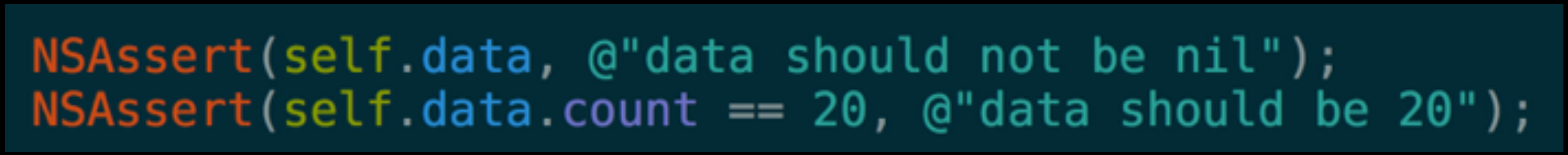
Find Navigator Detail: Up to Thirty Lines

Issue Navigator Detail: Up to Thirty Lines

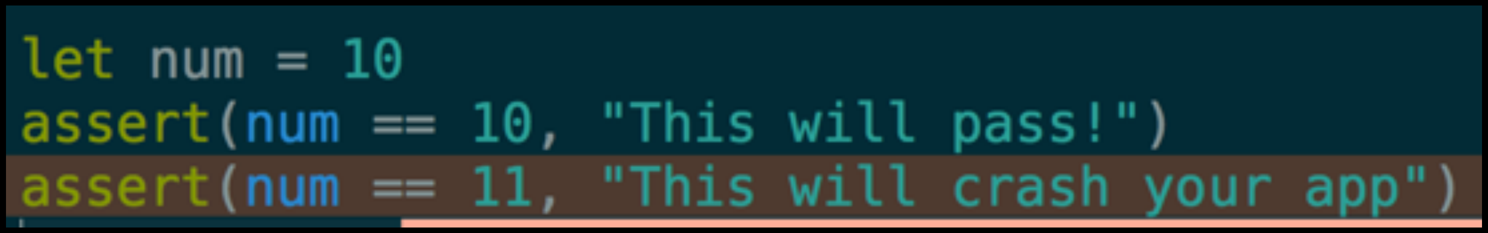
- Console logs: good
  - Easy to do
- Console logs: bad
  - Can introduce bugs
  - Need to be removed before shipping
  - Makes code harder to read
  - Busy console problem
  - DLog/ALog & other alternatives (objc)

NSAssert/Assert

- We've seen Asserts in the tests exercise (eg. XCTest**Assert**Nil())
- Asserts can be used generally to test whether something is the case.
- Why would you want your app to crash if some condition isn't met?



```
NSAssert(self.data, @"data should not be nil");
NSAssert(self.data.count == 20, @"data should be 20");
```

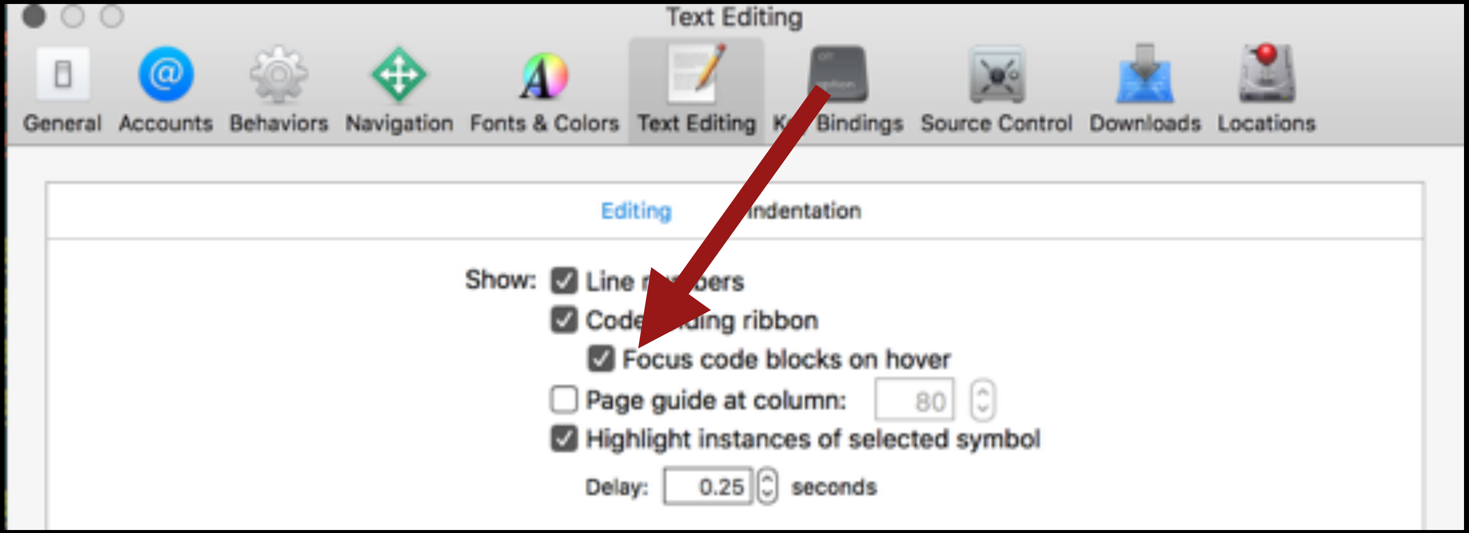


```
let num = 10
assert(num == 10, "This will pass!")
assert(num == 11, "This will crash your app")
```

- Problem with Asserts: Should be removed from production code. Might as well write unit tests instead. (ZAssert macro).

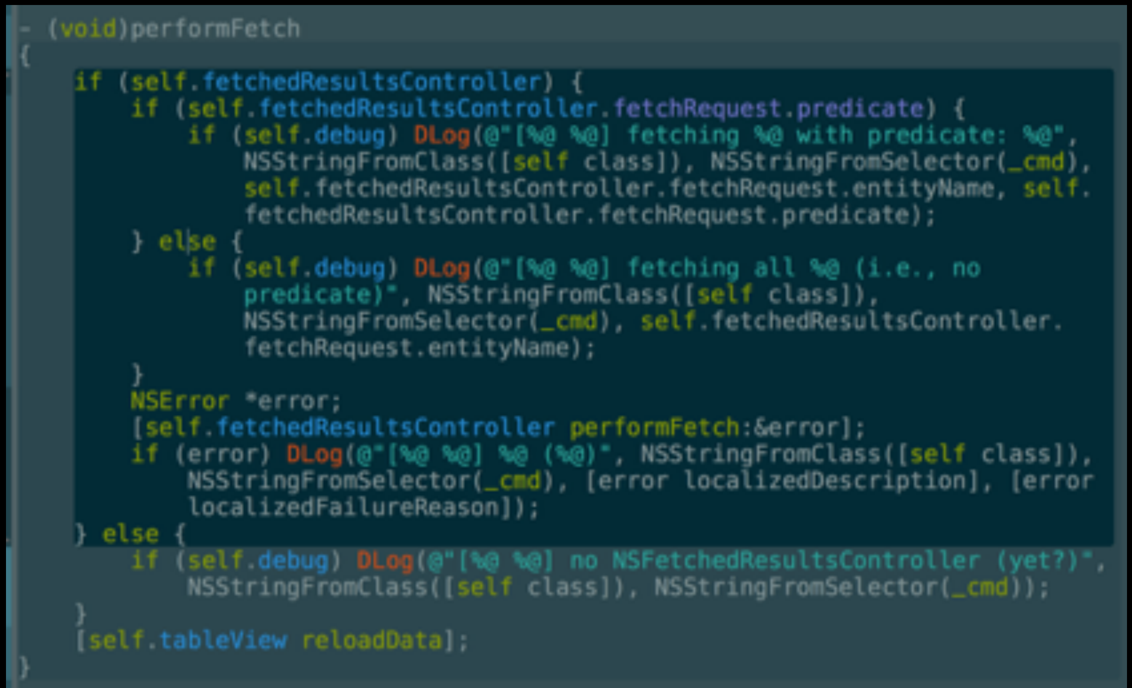
Folding Ribbon Is Your Friend

- Enable Code Folding Ribbon



Show: [✓] Line numbers
[✓] Code folding ribbon
[✓] Focus code blocks on hover
[ ] Page guide at column: 80
[✓] Highlight instances of selected symbol
Delay: 0.25 seconds

- Easily fix bugs caused by scope issues!



```
- (void)performFetch
{
    if (self.fetchedResultsController) {
        if (self.fetchedResultsController.fetchRequest.predicate) {
            if (self.debug) DLog(@"[%@ %@] fetching %@ with predicate: %@",
                NSStringFromClass([self class]), NSStringFromSelector(_cmd),
                self.fetchedResultsController.fetchRequest.entityName, self.
                fetchedResultsController.fetchRequest.predicate);
        } else {
            if (self.debug) DLog(@"[%@ %@] fetching all %@ (i.e., no
                predicate)", NSStringFromClass([self class]),
                NSStringFromSelector(_cmd), self.fetchedResultsController.
                fetchRequest.entityName);
        }
        NSError *error;
        [self.fetchedResultsController performFetch:&error];
        if (error) DLog(@"[%@ %@] %@ (%@)", NSStringFromClass([self class]),
            NSStringFromSelector(_cmd), [error localizedDescription], [error
            localizedFailureReason]);
    } else {
        if (self.debug) DLog(@"[%@ %@] no NSFetchedResultsController (yet?)",
            NSStringFromClass([self class]), NSStringFromSelector(_cmd));
    }
    [self.tableView reloadData];
}
```

Instruments

- Xcode has a massive instruments feature used for debugging and performance tuning
- We'll look at two instruments briefly: Allocations, and Time Profiler
- Allocations takes a snapshot of all of the objects your app allocates, retains and releases.
- The Time Profiler gives you data on how long your app is spending running various methods

Debugger & Instruments Tour

Debugging Strategies

- Avoid stabbing in the dark. (Lost keys analogy).
- My Technique

1. Describe problem thoroughly. Describe the precise conditions that trigger unexpected behaviour. If you need more info, gather it. THINK, don't just start stabbing into the dark (i.e. commenting out lines willy nilly).
2. Form hypothesis. Start with most obvious and easy to test.
3. Test your hypothesis.
4. Rinse & repeat
5. Document your results in **Solutions Log**.

Let’s Debug!

Important Tools

- **OpenSim** for viewing SQLite https://github.com/luosheng/OpenSim/releases
- **SQLPro** https://itunes.apple.com/ca/app/sqlpro-for-sqlite-read-only/id635299994?mt=12
- **Viewing Diff Files**:
  - **SourceTree**: https://www.sourcetreeapp.com
  - **P4Merge**: https://www.perforce.com/product/components/perforce-visual-merge-and-diff-tools
- **Networking:**
  - **Paw**: https://itunes.apple.com/ca/app/paw-http-rest-client/id584653203?mt=12

References

- https://developer.apple.com/support/debugging/
- https://developer.apple.com/library/ios/documentation/DeveloperTools/Conceptual/debugging_with_xcode/chapters/debugging_tools.html
- https://developer.apple.com/library/tvos/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/
- http://lldb.llvm.org
- http://jeffreysambells.com/2014/01/14/using-breakpoints-in-xcode