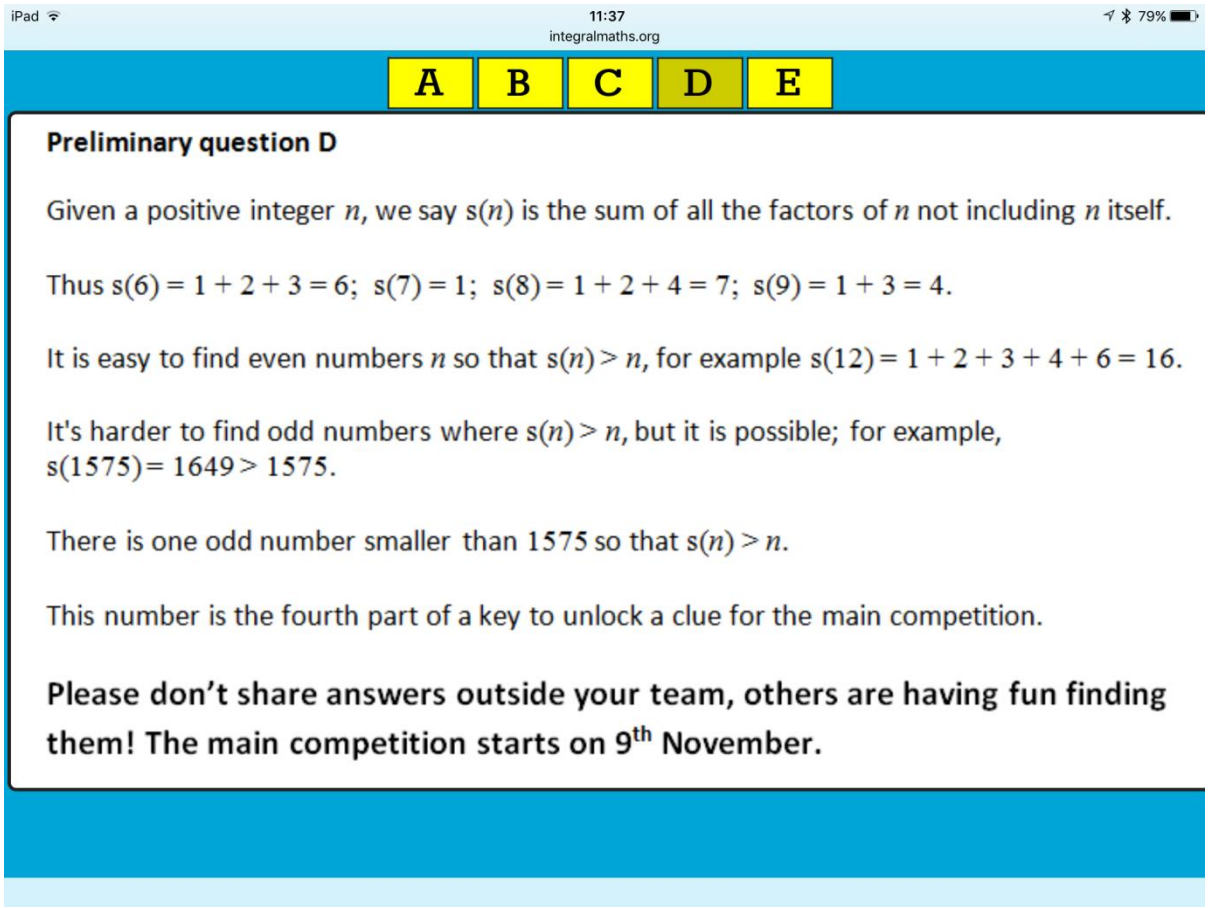
Swift in the Classroom

SPF – 9 XI 16

Today is the first day of the Ritangle Challenge (aimed at 16-18 year olds), a competition with 5 preliminary questions and 23 daily questions, organised the Integral organisation which supports maths teaching in the final two years of high school in the UK. Answers to all 28 questions are then combined to give a clue to the overall solution.

A couple of students came to me at afternoon break asking for help on Prelim Question D, below. I thought it might be of interest how Swift gave me new ways in which my class could experience the underlying math.



Discussion:

We began by teasing out why this problem was hard to handle with traditional mathematics – the definition of factor is based on multiplication/division properties, whereas the function s(n) then involves a sum – adding/subtracting.

Although we can pass between the two processes when they are separate (using logarithms and powers), when they are together they don't really mesh and this leads to difficult problems, including two open conjectures – the Twin Prime Conjecture, and Goldbach's Conjecture.

So, it makes a lot of sense we need to approach this through code.

I teased out from the class the idea of testing for divisors by checking to see if the remainder when you divide is zero and this led to the very basic code below.

The important thing here is that after 5 minutes or so the whole class was able to get some code which added up the sum of the digits. They had all typed code on their iPads, and all felt a sense of ownership.

My Playground 3

```swift
// Sum of proper factors function
func sumf(n: Int) -> Int {
    // Check if value is zero
    if n < 2 {
        return 1
    }
    // calculation
    var result = 0
    for i in 1...(n-1) {
        if n  % i == 0 {
            result = result + i
        }
    }
    return result
}

print(sumf(n: 6))
```

123

3×

123

abc

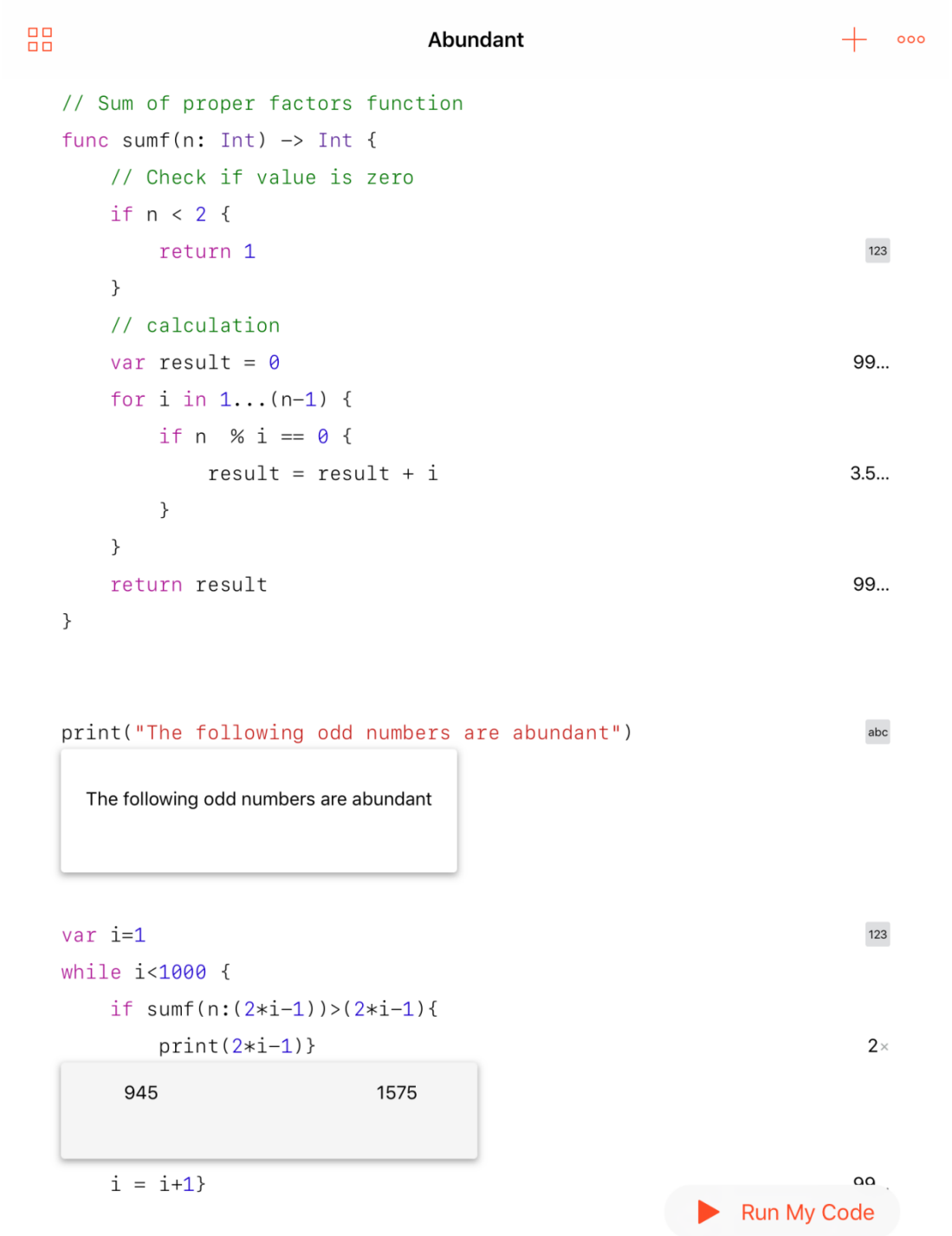
6

Run My Code

. == != < > <= >= + - * /

Development:

Next we looked at how we can cycle through odd integers using 2*i-1 as i cycles through all integers (or equivalently change our gap to 2), and we could now test all the odds up to 2000 to find the required solution.



Reflection:

At this point I shared a little theory with the class – the function is called the aliquot sum; numbers which map to themselves are perfect, if they get bigger they are abundant, if they get smaller when you apply the map then they are called deficient.

I rephrased the question as find odd abundant numbers, which of course Dr Google can do for you, but it can't do the following new exciting step.

Play:

Now I let the students loose, with a basic code in place I invited them to experiment. Can you improve the code? Change some values, what happens?

Two ideas came out:

Firstly, the students spotted a way to improve my code to check for proper factors – we only need to check up to n/2, so we could halve the calculation by changing the checking range in the original function.

Secondly, everyone continued the search, finding odd abundants up to 3,000, then 10,000, 50,000, and then with a bit of promoting from me, up to 80,000. What patterns do you spot?

Well, they all in 5, cool. Why on Earth does that work?

Back to formal maths, and I proved to them that if you start with any abundant odd, and multiply by any odd integer the answer is abundant. They then twigged that 945 being abundant will generate a whole family of odd abundants, ENDING IN A 5, 945*(2*n-1). Cool.

Then we needed to talk about necessary and sufficient conditions. We've seen that 945 generates an infinite family of odd abundants ending in 5, but are all odd abundant like this. It looks like it, I mean, we've checked up to 80,000!!!

Let's take a look now up to 90,000.

Disaster – 81,081 is an odd abundant which doesn't end in 5. The conjecture was false but so interesting to explore for ourselves rather than reading about it in a book!

Lessons:

If we need to check a proof case by case then, whether you check 1,10,100,1000, or 80000 cases, there are still infinitely many cases left to check, and we've not got very far in our checking, although our "finite intuition" tries to convince us we have.

Swift opens up possibilities for experimental mathematics, which turn mundane questions into opportunities to explore the mathematical landscape.

Code is a tool in maths, which we can learn through utility and practical (in a math sense) application.

Thank you for the opportunity to be able to share this augmented world with my students, by giving them code at their fingertips.

JSC