Open in new tab

Debugging

Lesson Duration: 60 minutes

Learning Objectives

- Understand importance of debugging
- Understand how to isolate a problem trough debugging
- Know best practice of debugging

Why Debugging Is So Important

An inescapable part of coding is that while you're trying to make something work, you come across many bugs or errors along the way. This is totally normal - if we could write perfect code off the top of our heads we'd all be billionaires! Unfortunately we can't, so coding becomes a process of solving one problem after another.

We're going to look at a few ways to firstly try to prevent us having as many errors, and then to solve the errors when they occur.

Preventative

Indentation and layout

We do nice consistent indentation and layout both for others who want to look at our code, but also for ourselves. Being consistent makes it loads easier to spot when something is out of place.

Tip - `cmd + A` and then `cmd + alt + L`can be a quick fix. `cmd + [ or ]` indents highlighted text or current line left or right.

Good variable names

Use explicit naming for variables and functions e.g. your function name should tell us exactly what it does.

Testing - unit testing, manual testing e.g. irb or console

For example:

wrong:

```
  def calculate(num1, num2, num3)
    do calculation
  end
```

right:

```
  def calculate(length, width, height)
    do calculation
  end
```

Unit testing

Is the test right? Look at test failure output - line number, does the actual value give any hints as to what's gone wrong.

Manual testing

This is things like testing one line in IRB, or using p to output values. Can be useful for a quick check.

```
def manipulate(integer)
  integer += 10
  p integer
  integer /= 2
  p integer
end
```

Small methods - single responsibility

Keeping our code in small chunks and having each method/function only responsible for one thing means that it's easier to see where things are going wrong. It has a lot of other benefits too like meaning we can reuse it more.

Error handling

Error messages

It's really important to actually read the error messages that we get. They're there to help us fix the problems after all. Don't just see that there is an error message, blank it out and go straight back to the code.

Some steps to take:

Look at the file and line number(s) Look at the message - what type of error is it? If there is one, look at the arrow or variable name to see exactly where you should check for errors - often it's before where the arrow is pointing to. Go back to the code and see what you can spot. Some common ones:

```
undefined local variable or method undefined method 'to_i' for [1, 2, 3, 4]:
Array wrong number of arguments (1 for 2) syntax error,
unexpected tIDENTIFIER, expecting '}'
```

Examples:

Undefined method

```
def defined_method()
  p "Success"
end

definedmethod()
```

Wrong number of arguments:

```
def add(num1, num2, num3)
  return num1 + num2 + num3
end

add(2, 4)
```

Missing end

```
fruits = ["apple", "orange", "pear", "plums"]
def missing(array)
  for fruit in array
    if(fruit == "grape")
      p "Plums!!"
    end
    if(fruit == "apple" || fruit == "orange" || fruit == "pear")
      p "Meh"
    # end
  end
end

missing(fruits)
```

Spotting the Bug!

This will come more with time but there are some things you can make sure to look out for. Look for spelling errors, where your brackets or opener and 'end' are opening and closing, look for missing commas, check capitalisation.

Console outputting

One method of debugging is to temporarily place console output (everyone's favourite… p!) lines in at key points in the code to see what the value of certain variables is at certain times, or to see if a section of code is being executed. Try to make these outputs meaningful so you know exactly what's going on - e.g. 'in the name checking loop' or 'value of account after deposit is: 5' are better messages than 'hello' or '5'.

Follow the flow of information

Part of a programmer's skill is in visualising the code as not just lines on a screen but as a set of objects that interact with each other. If you can learn to follow where data is being passed around and what state it should be in at any time it's easier to spot errors. Start at the beginning and follow what's happening.

Be methodical - process of elimination

It's usually worth checking every small thing is as it should be, even if it seems obvious - like is the file saved, am I in the right directory, have I called the right function. Eliminate every possible point of failure methodically - usually the error message will point you in the right direction, but not always! You will build up a list of things to check in your head (worth writing this down too!) as you go through the course.

Googling

https://realworldcoding.io/how-to-google-programming-problems-effectively-90f2a43ef982#.himftjazs

Top things for googling - include the language, start by being specific, make sure the problem the resulting page is solving is similar enough to yours to be useful. Googling the whole error message can get good results.

Rubber ducking

This can be done in your head, or to a friend/instructor - talking through what the problem is can often help you solve it much faster.

http://www.rubberduckdebugging.com/

Documentation

Make sure to find out what the recommended documentation is for the language or tool you are using, as well as looking at other options. Familiarise yourself with how that documentation is laid out, what the most useful parts of it are, and bookmark any pages you keep coming back to.

Further Tools

IDEs provide a useful tool called Breakpoint debugging… but we'll look at that later!

Finally:

If you're stuck for a while, ask a classmate or an instructor - we're all here to help each other, and your instructors literally get paid to help you learn so use this resource while you can!

Don't get upset by bugs, even if they're 'silly' ones - this is how we learn. Programming is mainly just a series of fixing one bug after another until everything is working how we want it to. Even moving from one bug to a different one is an accomplishment - one step closer to working code! And remember, each time you solve a bug, the more likely it is you will remember and not make that mistake again, or be able to solve it loads faster the next time.

Published with GitHub Pages