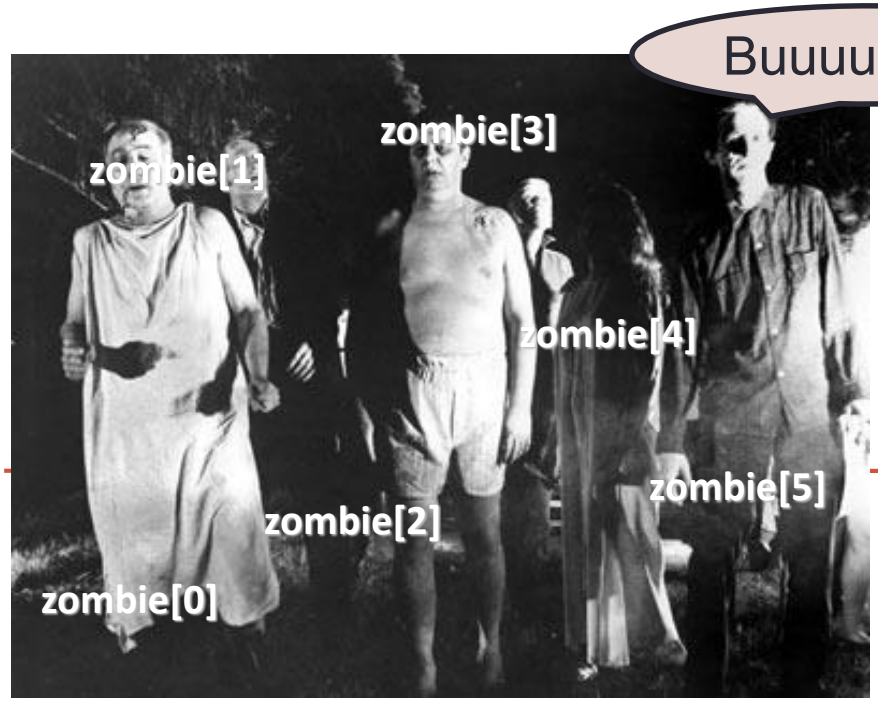
TESTING AND DEBUGGING



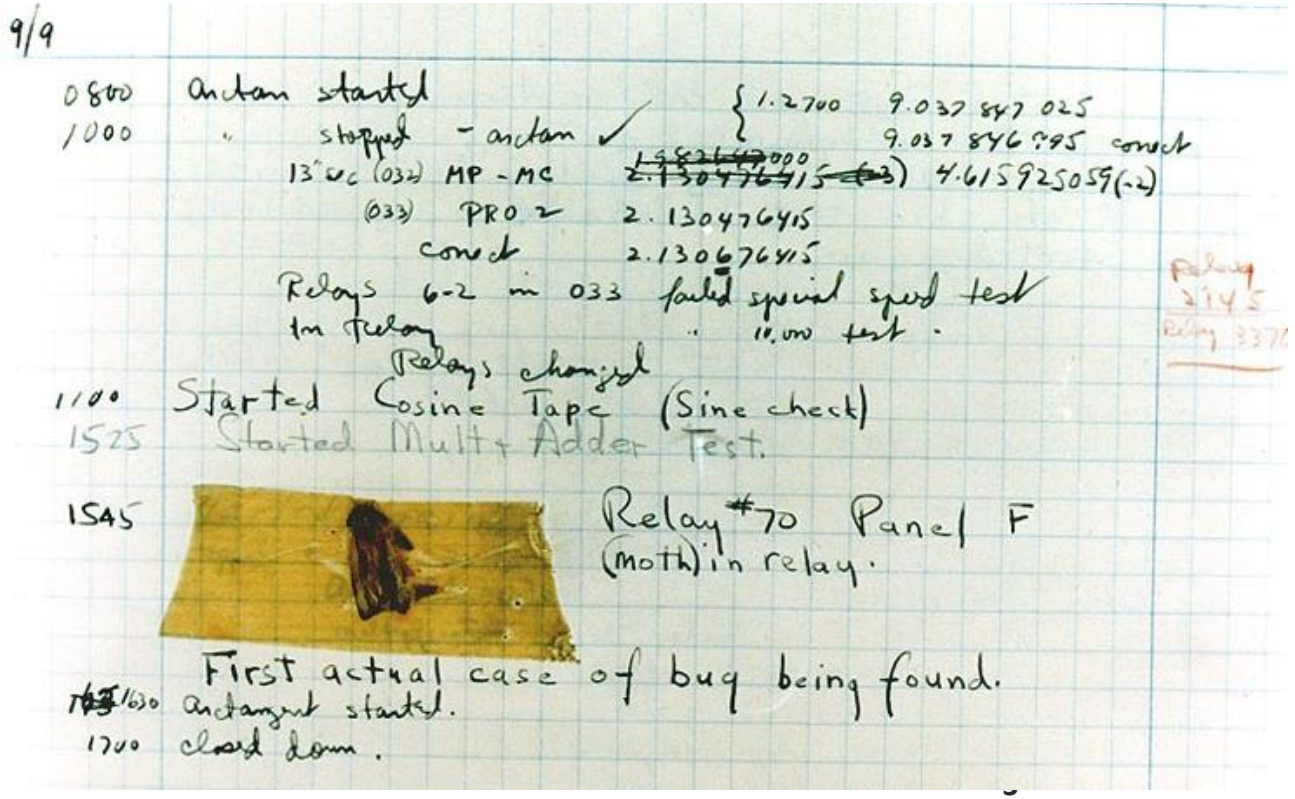
Outline

- Debugging
  - Types of Errors
    - Syntax Errors
    - Semantic Errors
    - Logic Errors

- Preventing Bugs
  - Have a plan before coding, use good style
  - Learn to trace execution
    - On paper, with print statements, using the debugger
  - Explain it to a teddy bear
  - Incremental development

Debugging

- Majority of program development time:
  - Finding and fixing mistakes! a.k.a. bugs
  - It's not just you: bugs happen to all programmers



Debugging

- Computers can help find bugs
  - But: computer can't automatically find all bugs!
- Computers do exactly what you ask
  - Not necessarily what you want
- There is always a logical explanation!
  - Make sure you saved & compiled last change

```
Roses are Red,
Violets are Blue

Unexpected '{'
on line 32.
```

"As soon as we started programming, we found out to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."
-Maurice Wilkes

"There has never been an unexpectedly short debugging period in the history of computers."
-Steven Levy

Preventing Bugs

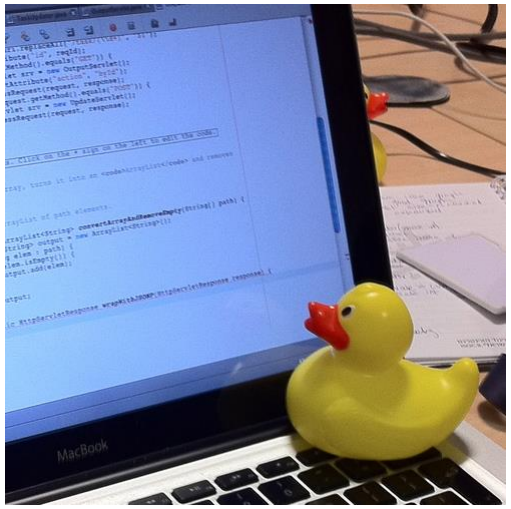
- Have a plan
  - Write out steps in English before you code
  - Write comments first particularly before tricky bits
- Use good coding style
  - Good variable names
    - "Name variables as if your first born child"
    - If variable is called `area` it should hold an area!
  - Split complicated stuff into manageable steps
  - ()'s are free, force order of operations you want
- Carefully consider loop bounds
- Listen to Idle (IDE) feedback

Incremental Development

- Split development into stages:
  - Test thoroughly after each stage
    - Don't move on until it's working!
    - Bugs are (more) isolated to the part you've just been working on
    - Prevents confusion caused by simultaneous bugs in several parts

Finding Bugs

- How to find bugs
  - Add debug print statements
    - Print out state of variables, loop values, etc.
    - Remove before submitting
  - Use debugger in your IDE
  - Talk through program line-by-line
    - Explain it to a:
      - Programming novice
      - Rubber duckie
      - Teddy bear
      - Potted plant
      - …



Debugging Example

- Problem:
  - For integer N > 1, compute its *prime factorization*
    - $98 = 2 \times 7^2$
    - $17 = 17$
    - $154 = 2 \times 7 \times 11$
    - $16{,}562 = 2 \times 7^2 \times 13^2$
    - $3{,}757{,}208 = 2^3 \times 7\ 13^2 \times 397$
    - $11{,}111{,}111{,}111{,}111{,}111 = 2{,}071{,}723 \times 5{,}363{,}222{,}357$
  - Possible application: Break RSA encryption
    - Factor 200-digit numbers
    - Used to secure Internet commerce

A Simple Algorithm

- Problem:
  - For integer N > 1, compute its *prime factorization*
- Algorithm:
  - Starting with i=2
    - Repeatedly divide N by i as long as it evenly divides, output i every time it divides
  - Increment i
  - Repeat

i	N	Output
2	16562	2
3	8281	
4	8281	
5	8281	
6	8281	
7	8281	7 7
8	169	
9	169	
10	169	
11	169	
12	169	
13	169	13 13
14	1	
…	1	

Example Run

Buggy Factorization Program

```
import sys

n = int(sys.argv[1])
for i in range (0, n)
    while n % i == 0:
    print(str(i), end=" ")
    n = n / i
```

This program has many bugs!

Debugging: Syntax Errors

```
import sys

n = int(sys.argv[1])
for i in range (0, n)
    while n % i == 0:
      print(str(i), end=" ")
      n = n / i
```

- Syntax errors
  - Illegal Python program
  - Usually easily found and fixed

Debugging: Semantic Errors

```
import sys

n = int(sys.argv[1])
for i in range (0, n):
    while n % i == 0:
        print(str(i), end = " ")
        n = n / i
```

Need to start at 2 since 0 and 1 cannot be factors.

```
% python Factors1.py 98
Traceback (most recent call last):
  File "Factors1.py", line 5, in <module>
    while n % i == 0:
ZeroDivisionError: integer division or
modulo by zero
```

- Semantic error
  - Legal but wrong Python program
  - Run program to identify problem

Debugging: Even More Problems

```
import sys

n = int(sys.argv[1])
for i in range (2, n):
    while n % i == 0:
        print(str(i), end = " ")
        n = n / i
```

```
% python Factors2.py 5
```

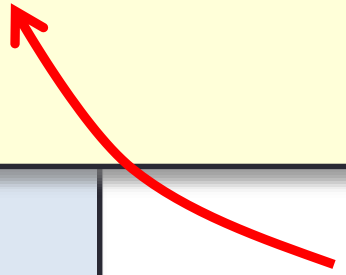
No output???

Debugging: Adding Trace Print Statement

```
import sys

n = int(sys.argv[1])
for i in range (2, n):
    while n % i == 0:
        print(str(i), end = " ")
        n = n / i
    print("TRACE " + str(i) + " " + str(n))
```

```
% python Factors3.py 5
TRACE 2 5
TRACE 3 5
TRACE 4 5
```



i in for-loop should go up to n

Success?

```
import sys

n = int(sys.argv[1])
for i in range (2, n+1):
    while n % i == 0:
        print(str(i), end = " "
        n = n / i
```

Fixes the "off-by-one" error in the loop bounds.

```
% python Factors4.py 5
5

% python Factors4.py 6
2 3

% python Factors4.py 98
2 7 7

% python Factors4.py 3757208
2 2 2 7 13 13 397
```

Correct, But Too Slow

```
import sys

n = int(sys.argv[1])
for i in range (2, n+1):
    while n % i == 0:
        print(str(i), enc
        n = n / i
```

```
% python Factors4.py 11111111
11 73 101 137

% python Factors4.py 1111111111
21649 51329

% python Factors4.py 11111111111111111
2071723 5363222357
```

Fixed Faster Version

```
import sys

n = int(sys.argv[1])
i = 2
while i^2 <= n:
    while n % i == 0:
        print(str(i), en
        n = n / i
    i += 1
```

```
% python Factors5.py 98
2 7 7

% python Factors5.py 11111111
11 73 101 137

% python Factors5.py 1111111111
21649 513239

% python Factors5.py 11111111111111
11 239 4649 909091

% python Factors5.py 1111111111111111
2071723 5363222357
```

Factors: Analysis

- How large an integer can I factor?

```
% python Factors.py 3757208
2 2 2 7 13 13 397

% python Factors.py 92011111169755555703
92011111169755555703
```

digits	i <= n	i*i <= n
3	instant	instant
6	0.15 seconds	instant
9	77 seconds	instant
12	21 hours *	0.16 seconds
15	2.4 years *	2.7 seconds
18	2.4 millennia *	92 seconds

* estimated

Preventing Bugs

- Have a plan
  - Write out steps in English before you code
  - Write comments first particularly before tricky bits
- Use good coding style
  - Good variable names
    - "Name variables as if your first born child"
    - If variable is called `area` it should hold an area!
  - Split complicated stuff into manageable steps
  - ()'s are free, force order of operations you want
- Carefully consider loop bounds
- Listen to Idle (IDE) feedback

Incremental Development

- Split development into stages:
  - Test thoroughly after each stage
    - Don't move on until it's working!
    - Bugs are (more) isolated to the part you've just been working on
    - Prevents confusion caused by simultaneous bugs in several parts

Summary

- Debugging
  - Types of Errors
    - Syntax Errors
    - Semantic Errors
    - Logic Errors
- Preventing Bugs
  - Have a plan before coding, use good style
  - Learn to trace execution
    - On paper, with print statements, using the debugger
  - Explain it to a teddy bear
  - Incremental development
- Test, Test, Test!!