SWINBURNE UNIVERSITY OF TECHNOLOGY

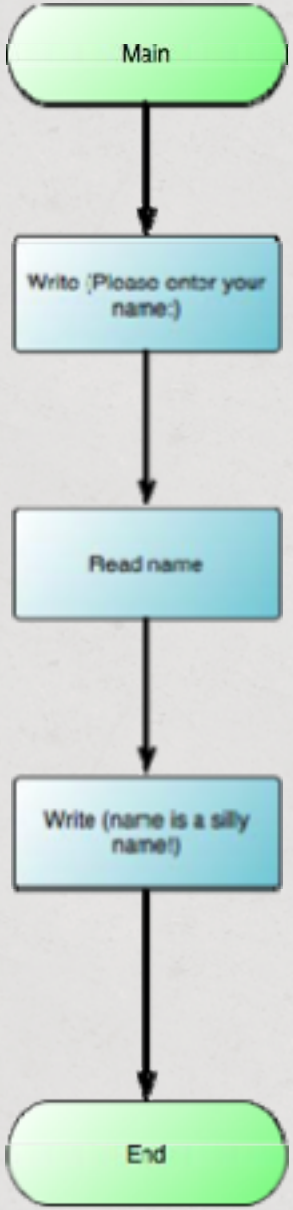
Introduction to Programming

Week 3, Selection and Iteration

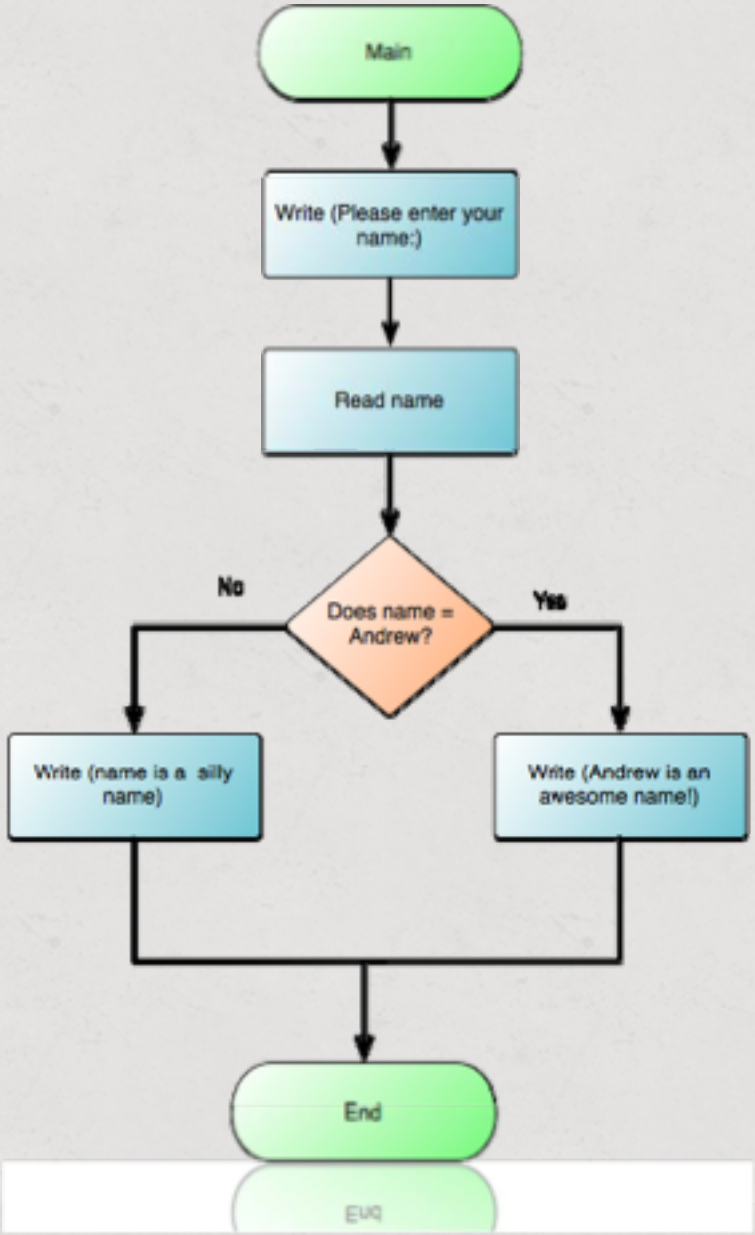
Flow Control

- This lecture we look at flow control. This makes up two of the three fundamental principles/elements of structured programming (coding aspects):
  1. Sequence (we looked at in previous weeks)
  2. Selection
  3. Iteration/repetition/looping

So far our code has been based on sequences of statements:



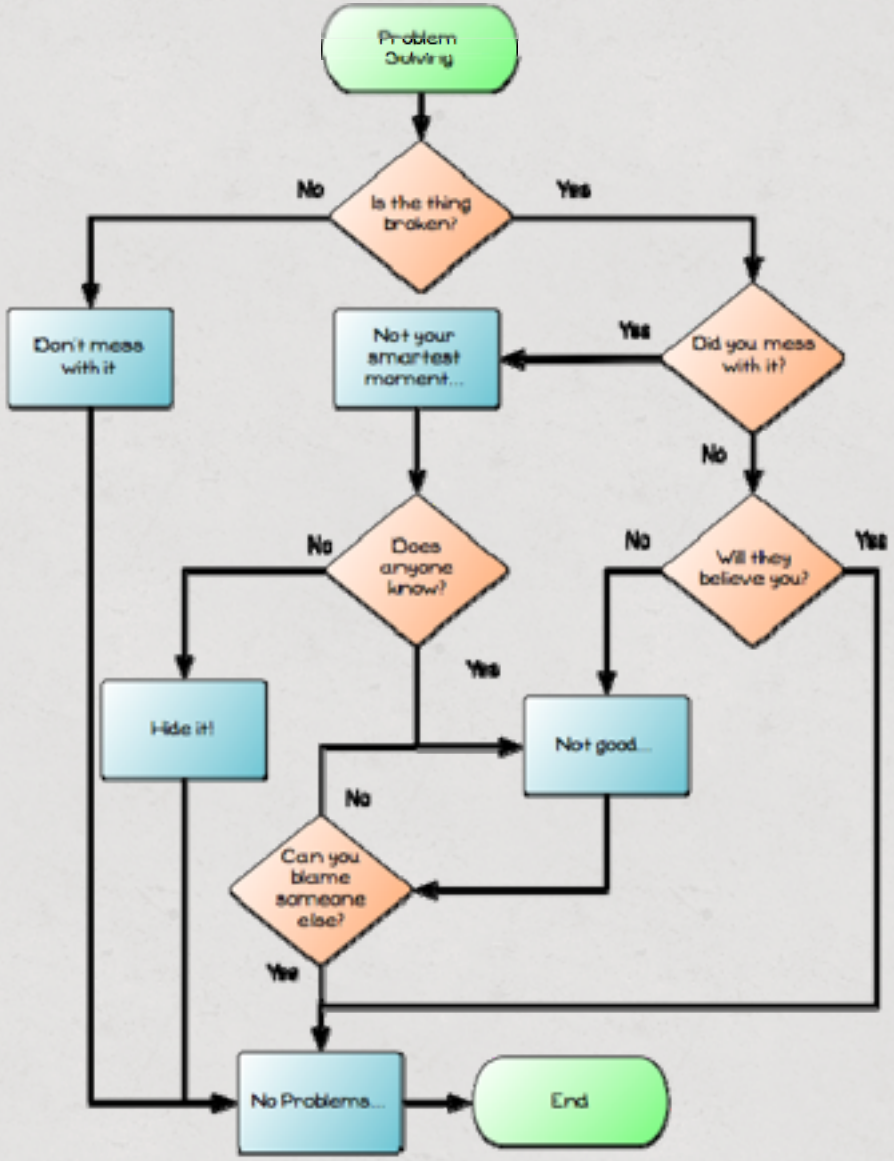
Now we are going to look at how to make our code more dynamic:



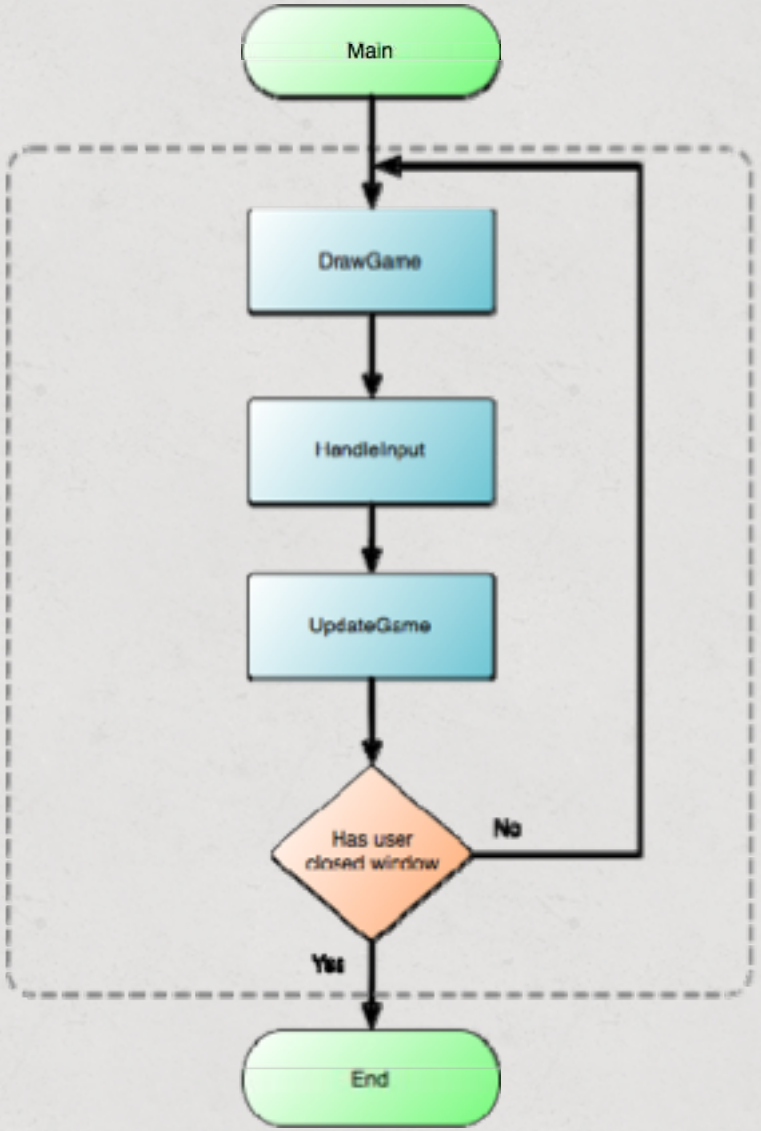
However, we need to

Carefully design our code

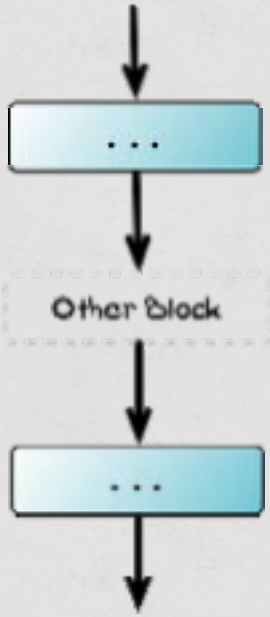
so as to avoid creating ‘spaghetti code’

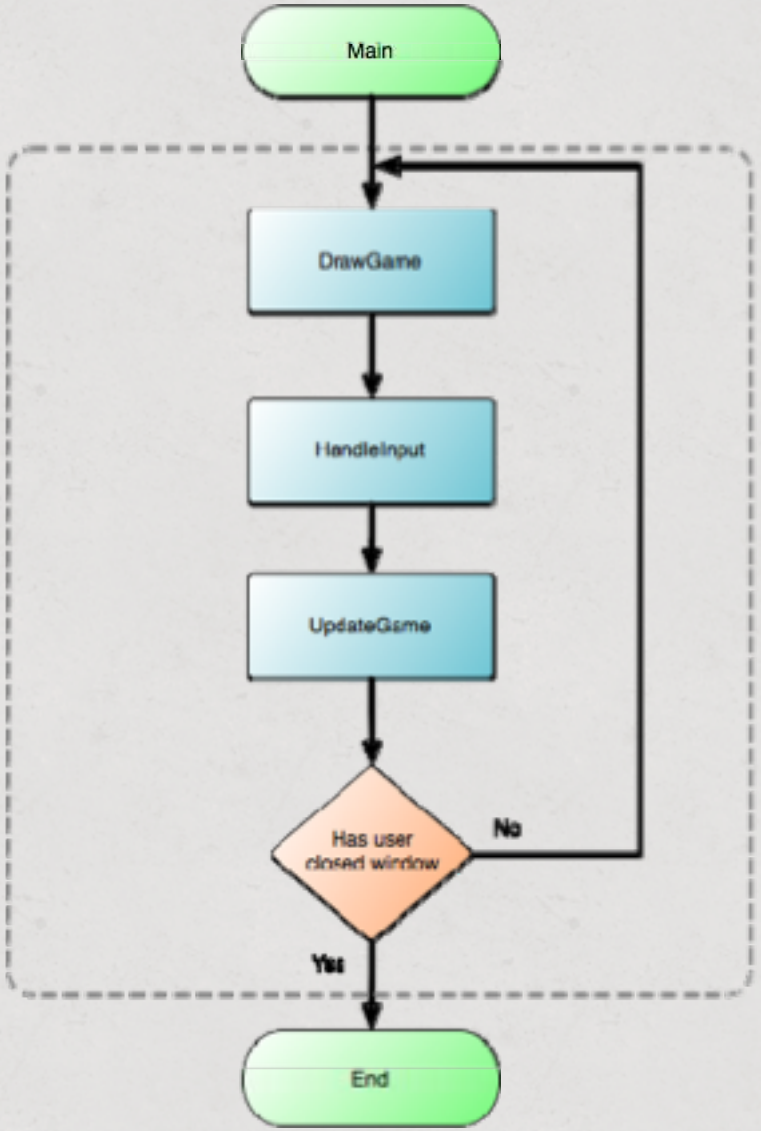


We try and use blocks of code with one entry and one exit point:

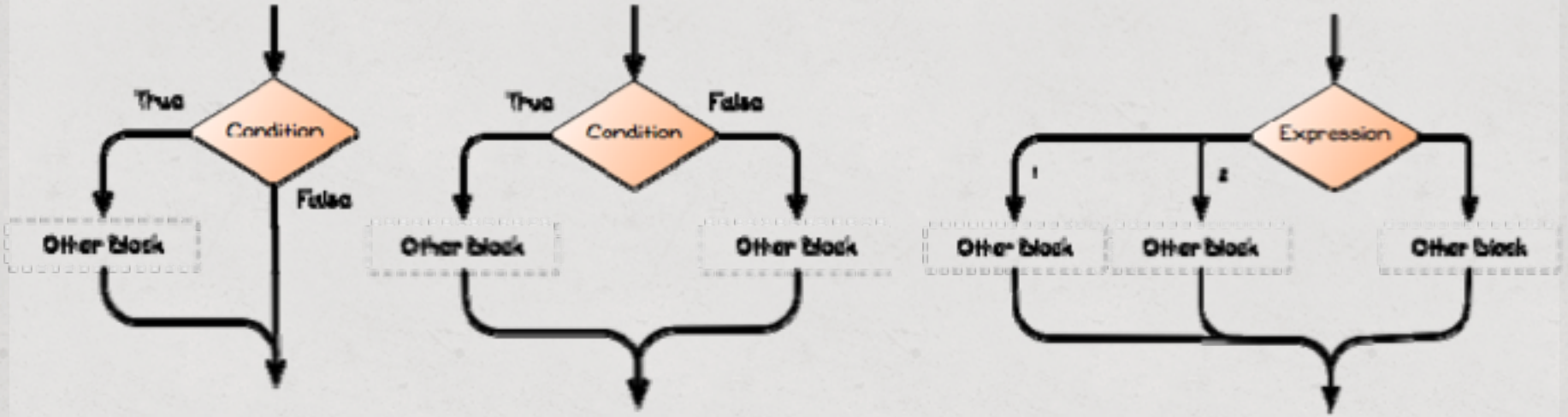


We try and use blocks of code with one entry and one exit point:

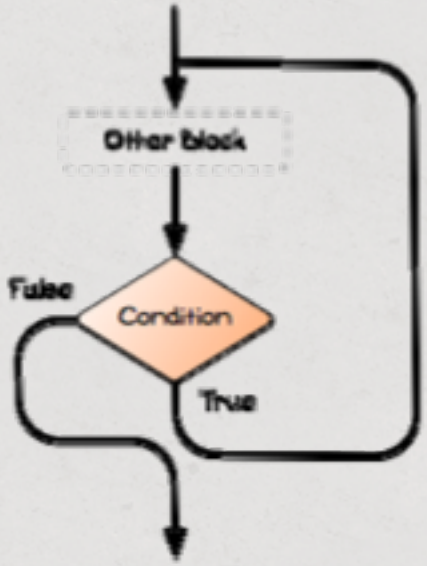


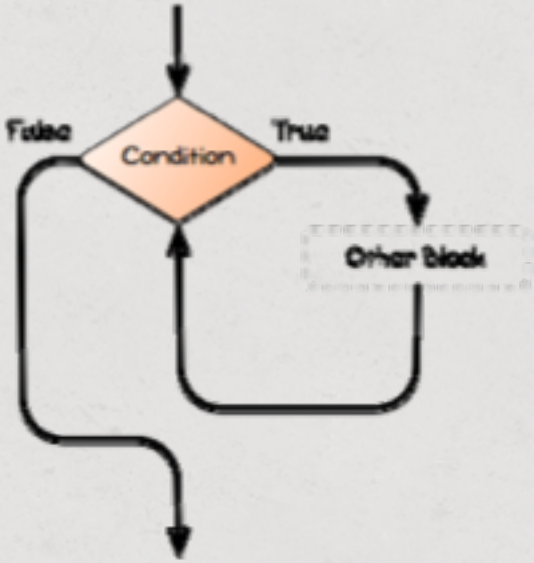


Blocks can include decisions, or branches, where one of a variety of paths is taken: this is called selection :

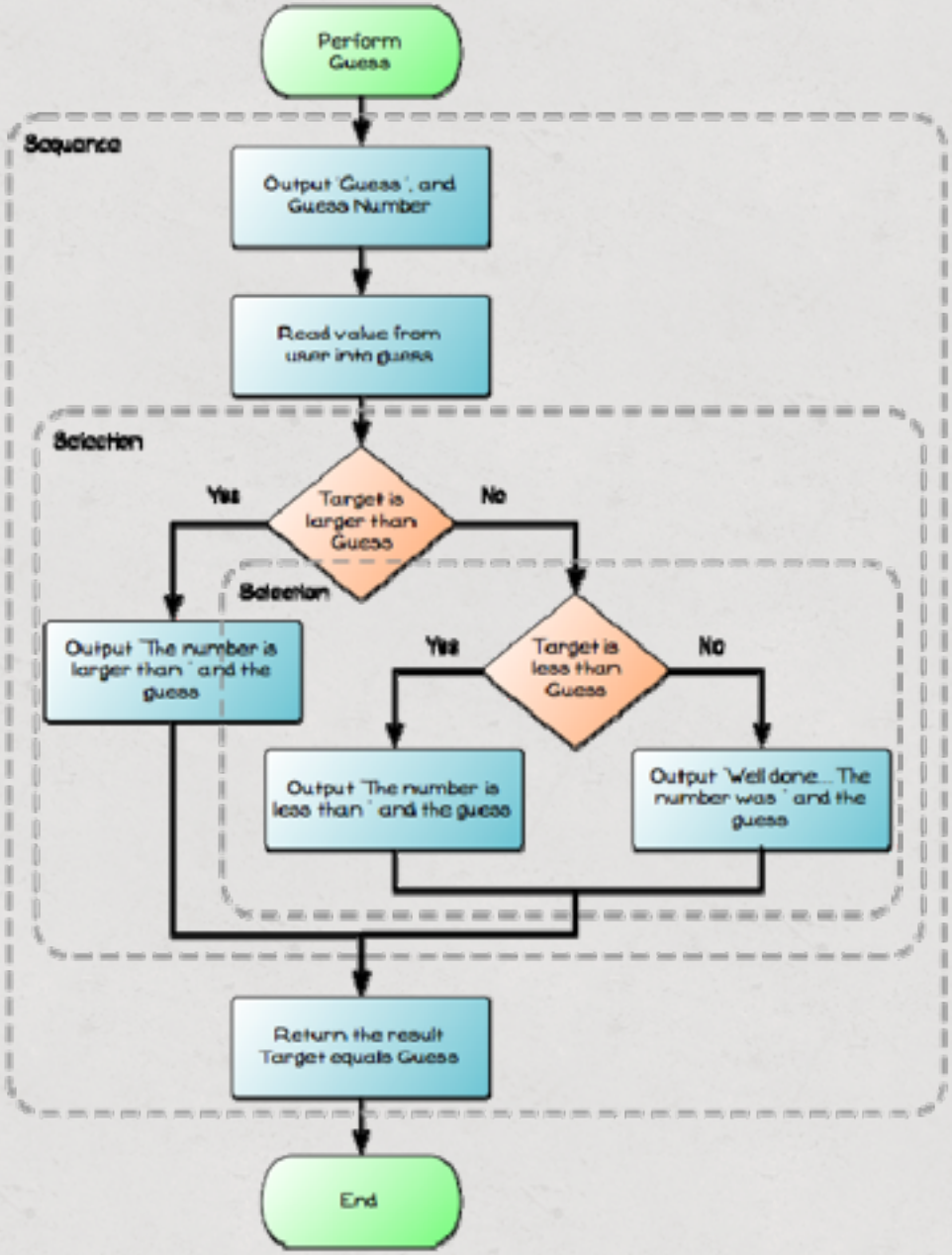


The other kind of block is a loop, where instructions are repeated a number of times: this is called repetition :



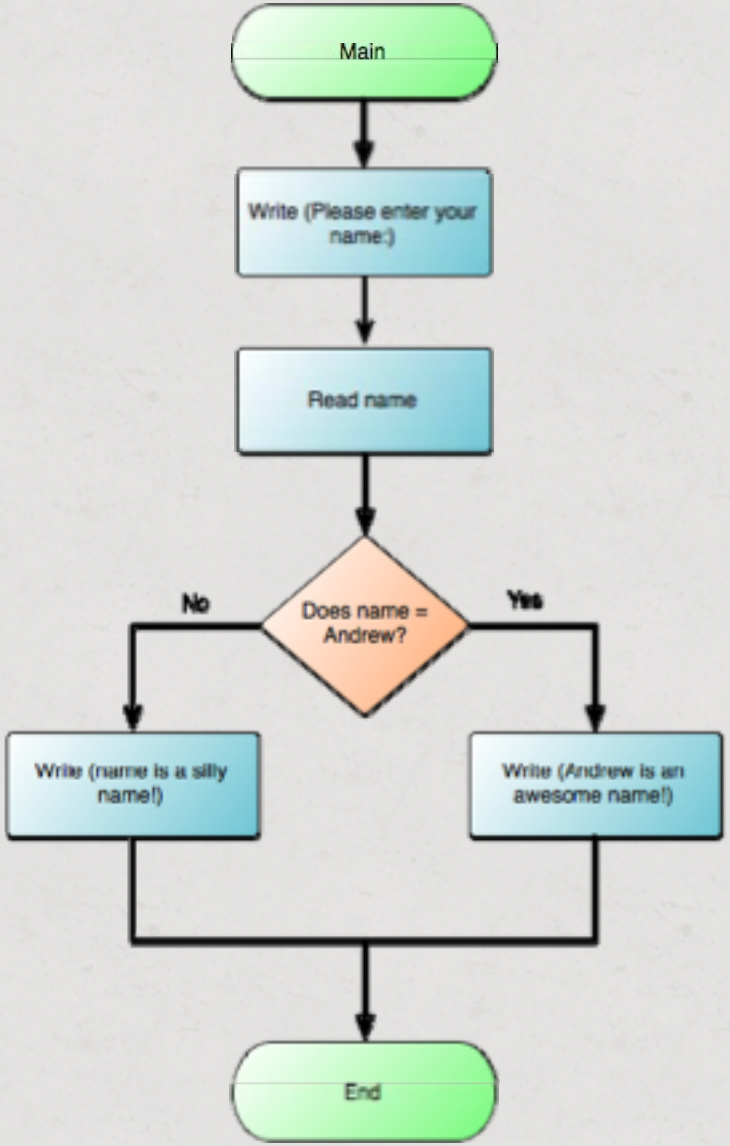


Sequence,
Selection,
and Repetition:
the key
components
of Structured
Programming



Control flow diagram for the Silly Name program:

This branching is represented using an **if** statement



If statements

- Simplest **if** statement:

```
count = 5

if (count == 5)
  puts 'It matched!'
end
```

Notice the **condition** `(count == 5)`

and the **consequent** `puts 'It matched!'`

Selection statements

Common **if** statement:

```
count = 5
if (count == 5)
  puts 'It matched!'
else
  puts 'No match!'
end
```

The condition

The consequent

The alternative

Using Booleans in conditions

- Conditional comparisons evaluate to be either *true* or *false* (i.e a Boolean value).
- Different languages have different operators:

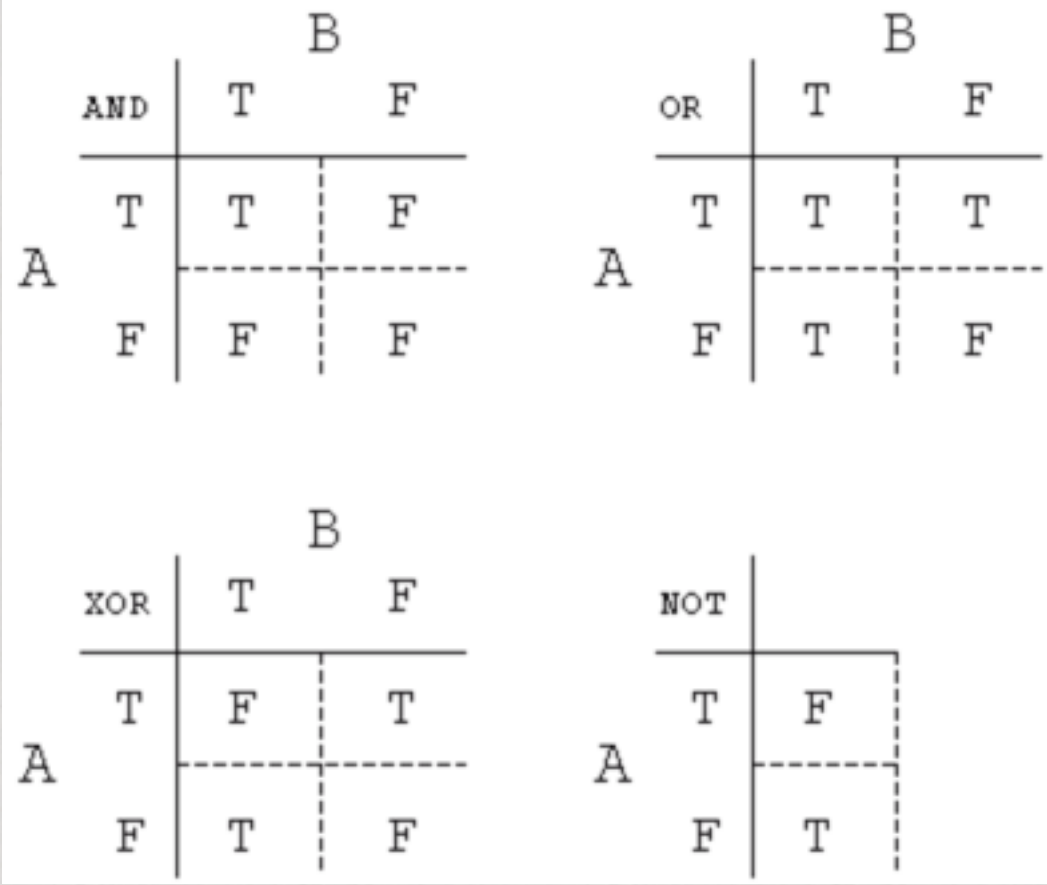
	Description	C	Pascal
Equal	Are the values the same?	a == b	a = b
Not Equal	Are the values different?	a != b	a <> b
Larger Than	Is the left value larger than the right?	a > b	
Less Than	Is the left value smaller than the right?	a < b	
Larger Or Equal	Is the left value equal or larger than the right?	a >= b	
Less Or Equal	Is the left value smaller or equal to the right?	a <= b	

Table 5.2: Comparison Operators

For Ruby's operators see :

See: https://www.tutorialspoint.com/ruby/ruby_operators.htm

Boolean Truth Tables



AND	B: T	B: F
A: T	T	F
A: F	F	F

OR	B: T	B: F
A: T	T	T
A: F	T	F

XOR	B: T	B: F
A: T	F	T
A: F	T	F

NOT	
A: T	F
A: F	T

Source: https://users.cs.jmu.edu/bernstdh/web/common/lectures/summary_logical-operators.php

Code for the above

```ruby
a = true
b = false

if (a and b)
  puts "AND: A and B are both True"
end

if (a or b)
  puts "OR: Either one of both of A or B is true"
end

if (a ^ b) # ^ is the XOR operator in Ruby
  puts "XOR: A is true, or B is True but not both"
end

if (!a) # ! is the operator for NOT
  puts "NOT: A is False"
end
```

Boolean Conditions

Lets try and evaluate some examples:

- Single variable:

  (value > 10) or (value < 0)

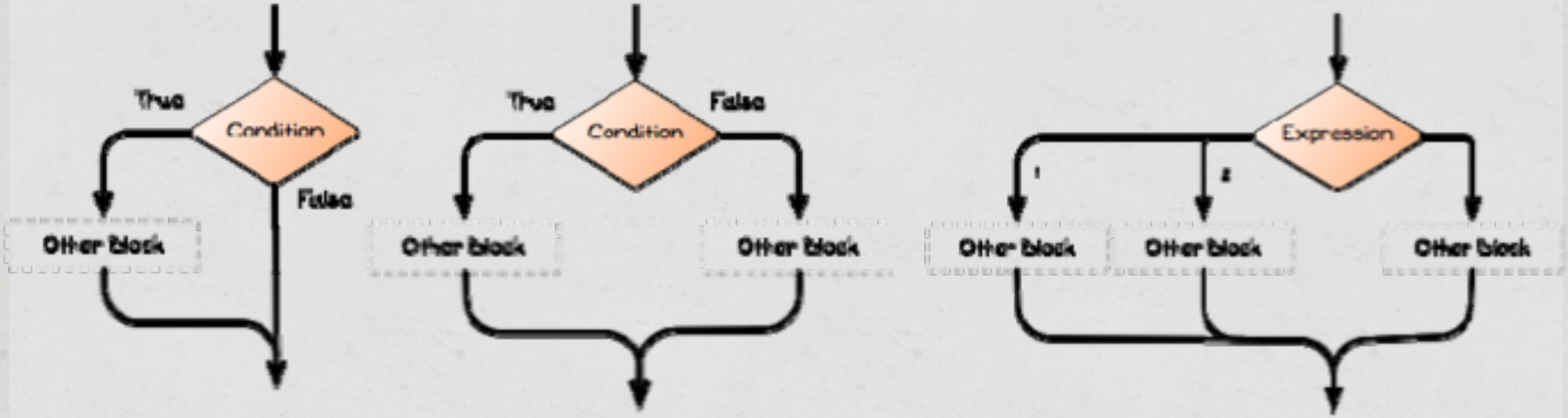
- Multiple variables:

  (age == 25) and (value != 5)

Source: https://users.cs.jmu.edu/bernstdh/web/common/lectures/summary_logical-operators.php

If statements - variations

Remember the different types we saw:



If statements - variations

```ruby
count = 5
if (count == 5)
  puts 'It matched!'
else
  puts 'No match!'
end

count = 3
if (count < 3)
  puts 'Less than 3!'
elsif (count == 3)
  puts 'Equal to 3!'
else
  puts 'Greater than 3!'
end
```

Unless Statements

```ruby
count = 5
unless (count == 5)
  puts 'Not 5!'
else
  puts 'Must be 5!'
end

puts 'It is 5!' unless count != 5
puts 'It is 5' if count == 5
```

```
MacBook-Pro:Lecture3Code mmitchell$ ruby unless_statements.rb
Must be 5!
It is 5!
It is 5
```

Case Statements

```
count = 2
case count
when 1..4
  puts 'Not 5!'
when 5
  puts 'Must be 5!'
when 5..10
  puts 'Must be between 5 and 10'
else
  puts 'Must be less than zero or more than 10'
end
```

Case Statements – For Types

```ruby
something = 10
case something
when Numeric
  puts "I'm a number.  My absolute value is #{something.abs}"
when Array
  puts "I'm an array.  My length is #{something.length}"
when String
  puts "I'm a string. In lowercase I am: #{something.downcase}"
else
  puts "I'm a #{something.class}"
end
```

```
MacBook-Pro:Lecture3Code mmitchell$ ruby type_case_statements.rb
I'm a number.  My absolute value is 10
MacBook-Pro:Lecture3Code mmitchell$ ruby type_case_statements.rb
I'm a string. In lowercase I am: fred
MacBook-Pro:Lecture3Code mmitchell$ ruby type_case_statements.rb
I'm an array.  My length is 2
MacBook-Pro:Lecture3Code mmitchell$
```

Acknowledgement: http://rosettacode.org/wiki/Conditional_structures/Ruby

Break time!

Note about Study Groups.

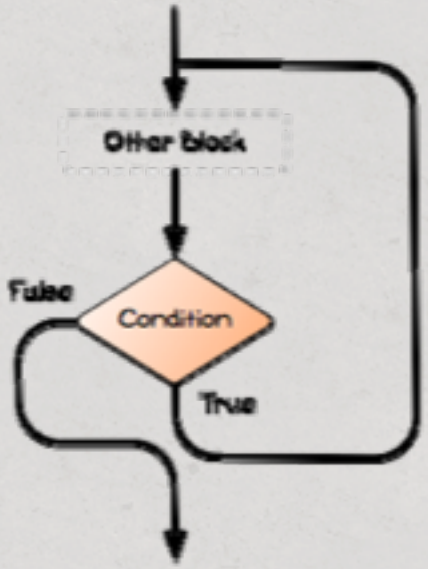
Also: Reflection Week.

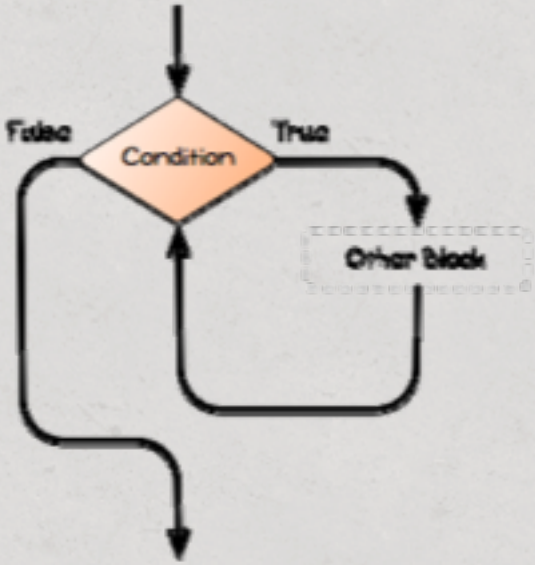
Iteration (loops)

There are two basic types of loops:

1. Pre-test
2. Post-test

We will see various forms of both.





- Loops often require *control variables* – we will see these as we go.
- When would you use a Post-test loop?

Pre-Test Loop – while

```ruby
count = 0
while count < 5
  puts 'Count is ' + count.to_s
  count = count + 1
end
```

Outputs:

```
MacBook-Pro:Lecture3Code mmitchell$ ruby while_loop.rb
Count is 0
Count is 1
Count is 2
Count is 3
Count is 4
MacBook-Pro:Lecture3Code mmitchell$
```

Pre-Test Loop – while

```ruby
count = 0
while count < 5
  puts 'Count is ' + count.to_s
  count = count + 1
end
```

Outputs:

```
MacBook-Pro:Lecture3Code mmitchell$ ruby while_loop.rb
Count is 0
Count is 1
Count is 2
Count is 3
Count is 4
MacBook-Pro:Lecture3Code mmitchell$
```

Pre-Test Loops – **for** loop

- Notice the **for** loop auto-increments the counter

  (i.e no need for count = count + 1)

```
count = 0
for count in 0..4
  puts 'Count is ' + count.to_s
end
```

Post Test Loops – Option 1

One option:

```
count = 0

begin
  puts 'my line ' + count.to_s
  count = count + 1
end until count == 3
```

Or Equivalently:

```
count = 0

begin
  puts 'my line ' + count.to_s
  count = count + 1
end while count < 2
```

Note: see: http://rosettacode.org/wiki/Loops/Do-while#Ruby

Post-Test Loop – Option 2

It is possible to use a break statement to 'jump' out of a loop or if statement.

The post test **loop:**

```
count = 0

loop do
  puts 'my line ' + count.to_s
  count += 1
  break if count == 2
end

count = 0
```

Source: http://rosettacode.org/wiki/Loops/Do-while#Ruby

Post-Test Loop – Another Alternative

- Or we could do:

```ruby
1.times do
  puts 'Enter a name: '
  info = gets.chomp
  unless info == "exit"
    puts 'You entered: ' + info
    redo
  end
end

puts "second loop"

1.times do
  puts 'Enter a name: '
  info = gets.chomp
  if info != "quit"
    puts 'You entered: ' + info
    redo
  end
end
```

Times Loop

- Or we could even do:

```
5.times do
  puts 'Enter a name: '
  name = gets.chomp
  puts 'Name is ' + name
end
```

Variable Times Loop

Or we can do:

```
count = 5
count.times do
  puts count.to_s + ' Enter a name: '
  name = gets.chomp
  puts 'Name is ' + name
end
```

Which outputs

(5 each time):

```
MacBook-Pro:Lecture3Code mmitchell$ ruby variable_times_loop.rb
5 Enter a name:
fred
Name is fred
5 Enter a name:
jill
Name is jill
5 Enter a name:
jenny
Name is jenny
5 Enter a name:
john
Name is john
5 Enter a name:
josh
Name is josh
```

Variable Times Loop – with counter

So if we want to know the iteration:

```ruby
count = 5
count.times do |i|
  puts i.to_s + ' Enter a name: '
  name = gets.chomp
  puts 'Name is ' + name
end
```

Which outputs (*i* each time):

```
MacBook-Pro:Lecture3Code mmitchell$ ruby variable_times_loop.rb
0 Enter a name:
fred
Name is fred
1 Enter a name:
sam
Name is sam
2 Enter a name:
jenny
Name is jenny
3 Enter a name:
josh
Name is josh
4 Enter a name:
jill
Name is jill
```

Variable Times Loops – **upto** loop

- The **upto** loop also auto-increments the counter

  (i.e no need for count = count + 1)

```
1.upto(5) do
  name = gets.chomp
  print 'Name is ' + name
end
```

Nested loops

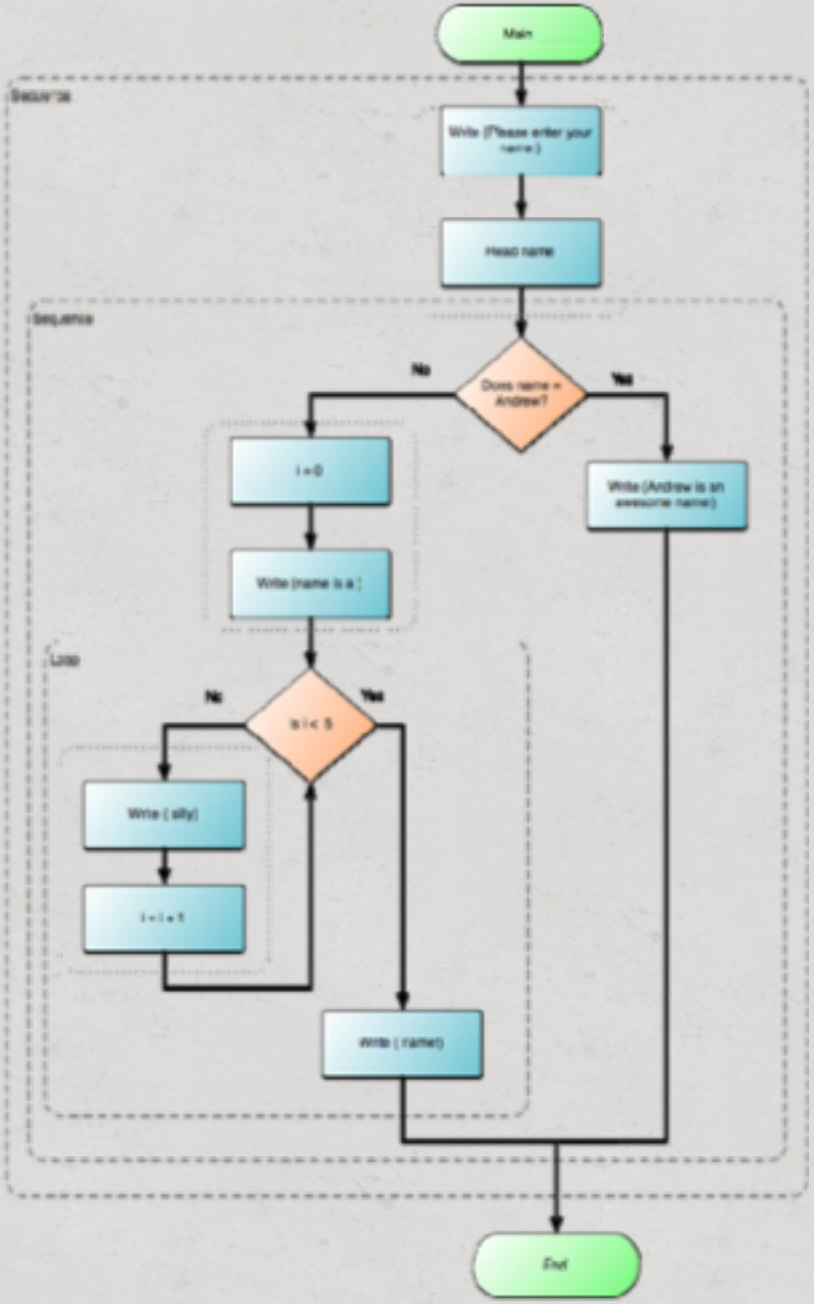
```ruby
1.upto(5) do |i|
  puts i.to_s + ' Enter runner\'s names in order of completion: '
  name = gets.chomp
  print 'Name is ' + name
  points = 0
  i.upto(3) do |j|
    points += j
  end
  puts '. You get ' + points.to_s + ' points'
end
```

This produces:

```
MacBook-Pro:Lecture3Code mmitchell$ ruby upto_loop_i.rb
1 Enter runner's names in order of completion:
Fred
Name is Fred. You get 6 points
2 Enter runner's names in order of completion:
Sam
Name is Sam. You get 5 points
3 Enter runner's names in order of completion:
Jill
Name is Jill. You get 3 points
4 Enter runner's names in order of completion:
Jenny
Name is Jenny. You get 0 points
5 Enter runner's names in order of completion:
Josh
Name is Josh. You get 0 points
MacBook-Pro:Lecture3Code mmitchell$
```

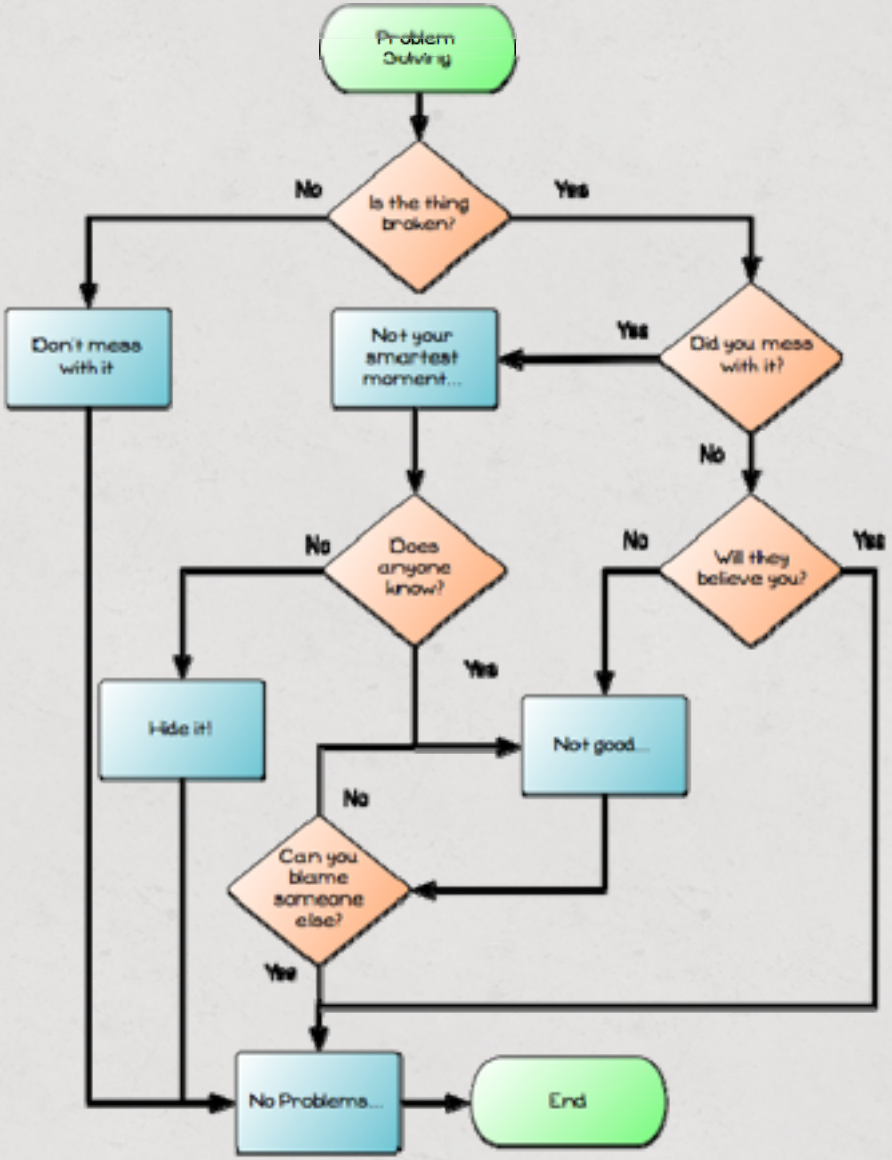
Good Code design

Good structured code design should produce code that looks like this:



Not this:

(spaghetti code)



Acknowledgement to Andrew Cain for his contribution to these notes.

Representing Conditions and Loops in Structure Charts

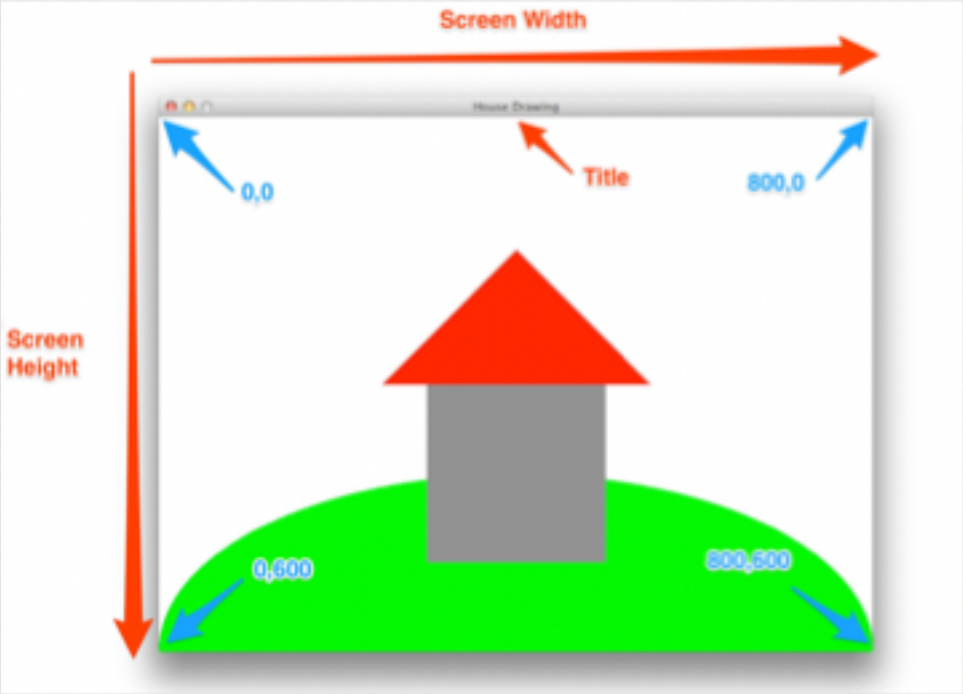
Lets see an example structure chart based on one of this week's tasks ….

Tasks for this week

- Silly name task
- Submenu task
- Shape Drawing Task

Graphical Programming (shape drawing)

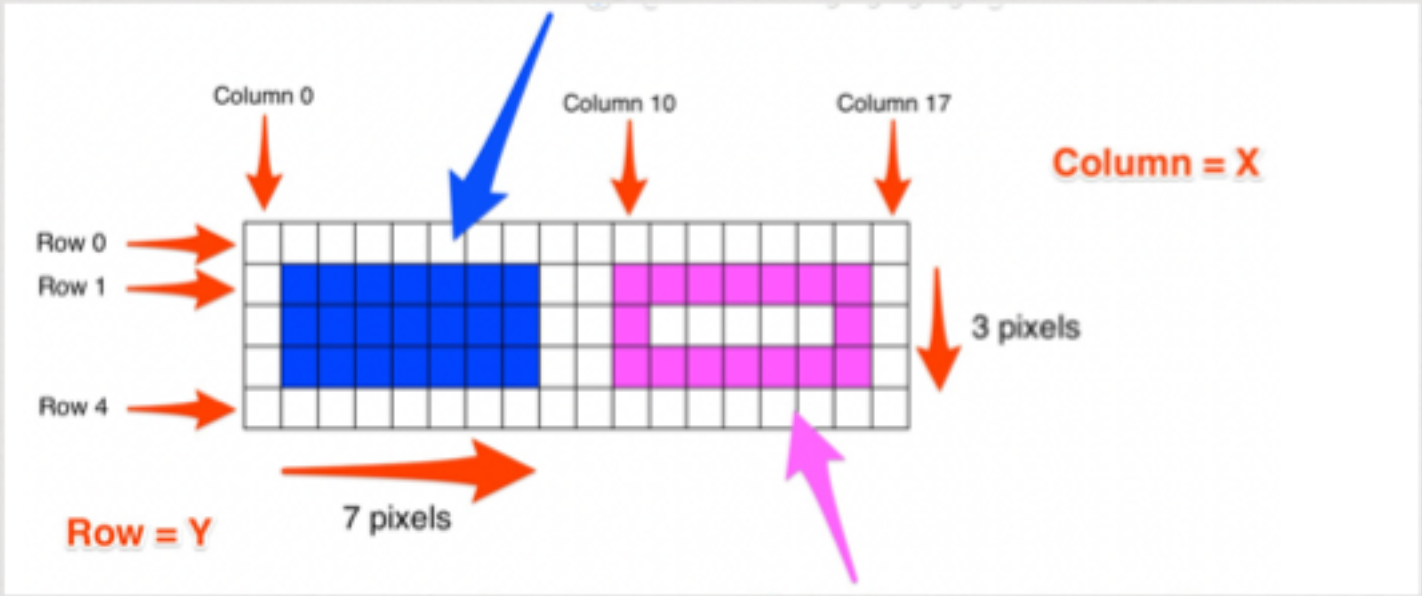
- We use a co-ordinate system to place things on the screen:



Graphical Programming

- Drawing a rectangle:

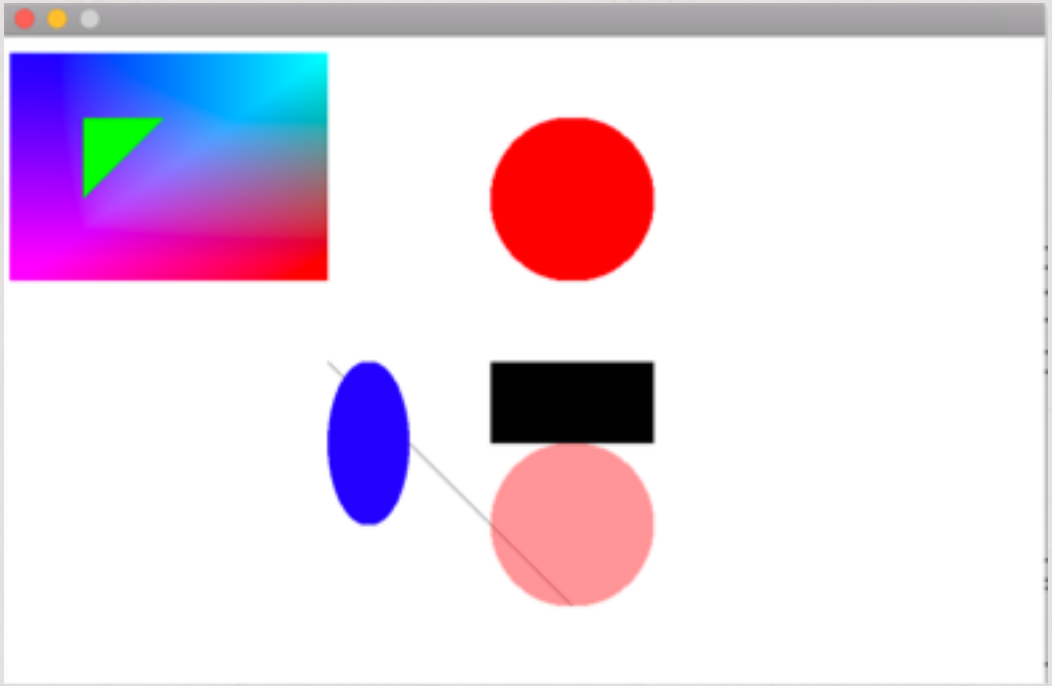
```
Gosu.draw_rect(1, 1, 7, 3, Gosu::Color::BLUE, ZOrder::TOP, mode=:default)
```



To draw this you need to draw 4 lines, a good opportunity to write a draw_open_rectangle() procedure!

Gosu Demonstration

- https://www.rubydoc.info/github/gosu/gosu/master/Gosu



Acknowledgement to Andrew Cain for his contribution to these notes.