Control Flow

Flow of control

- `if, unless, elsif, else`
- No parentheses or curly braces
- Use `end` to close flow control block

```
a = 5
if  a == 3
  puts "a is 3"
elsif a == 5
  puts "a is 5"
else
  puts "a is not 3 or 5"
end
```

```
unless a == 6
  puts "a is not 6"
end

# => a is 5
# => a is not 6
```

Flow of control (continued)

- `while, until`

```
a = 10
while a > 8
  puts a                          # => 10
  a -= 1                          # => 9
end

a = 9
until a > 10
  puts a                          # => 9
  a += 1                          # => 10
end
```

Flow of control – modifier form

- `if, unless, while, until` – on the same line as the statement

```
a = 5
b = 0
puts "Using modifier version" if a == 5 and b == 0
# => Using modifier version

times_2 = 2
times_2 *= 2 while times_2 < 100
puts times_2 # => 128
```

True / False

- `false` and `nil` objects are false
- EVERYTHING ELSE is true!

```
puts "0 is true" if 0 # => 0 is true
puts "false is true?" if "false" # => false is true?
puts "no way - false is false" if false # => DOES NOT PRINT ANYTHING
puts "empty string is true" if "" # => empty string is true
puts "nil is true?" if "nil" # => nil is true?
puts "no way - nil is false" if nil # => DOES NOT PRINT ANYTHING
```

for loop

- Hardly used
- `each` / `times` (covered later) preferred

```
for i in 0..2
  puts i
end
# => 0
# => 1
# => 2
```

(…Ranges are covered later…)

Blocks

- Chunks of code enclosed between either braces (`{ }`) or the keywords `do` and `end` and passed to methods as last “parameter”
- **Convention:**
  - Use `{ }` when block content is a single line
  - Use `do` and `end` when block content spans multiple lines

Blocks - Example

- Blocks are often used as iterators

```
1.times { puts "Hello World!" }
# => Hello World!

2.times do |index|
  if index > 0
    puts index
  end
end
# => 1

2.times {|index| puts index if index > 0 } # Same as above
```

Often accepts variable(s) between ||

Methods / Functions

- Parentheses are optional
- On empty parameter methods – always leave parentheses out when defining AND when calling a method
- `return` is optional
  - Value of last executed line returned
- `methods_asking_a_question?`
- `slightly_dangerous_methods!`

Methods (Continued)

```
def five? (n)
  n == 5
end
puts five? 5 # => true

def factorial (n)
  n == 0? 1 : n * factorial(n - 1)
end
puts factorial 5 # => 120

def factorial_with_default_value (n = 0)
  n == 0? 1 : n * factorial_with_default_value(n - 1)
end
puts factorial_with_default_value # => 1
puts factorial_with_default_value(3) # => 6
```

Ternary operator