Flow Control

Ruby Fundamentals

Overview

- Branching with *if* and *case*
- Looping
- Exception handling
- *throw* and *catch*

Conditional Execution with *if-else*

```
if can_launch?
  launch
else
  wait
end

message = if lander_cnt > 10 then "Launching" else "Waiting" end

launch if can_launch?

if fuel_level > 50
  set_fuel_light("green")
elsif fuel_level > 25
  set_fuel_light("yellow")
else
  set_fuel_light("red")
end
```

True and False

- Only *false* and *nil* evaluate to false
- Everything else is true: *true*, 0, empty string, empty array etc.

Conditional Execution with *unless*

- *if not condition* is equivalent to *unless condition*

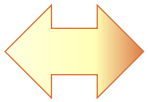
```
unless fuel_level < 25
  launch
end

launch unless fuel_level < 25
```

Ternary operator *? :*

```
can_launch? ? launch : wait
```

Conditional Initialization

```
a += 10
```



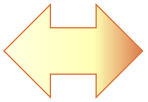
```
a = a + 10
```

```
||=        &&=
```

```
ship ||= Spaceship.new
```

```
ship = Spaceship.new unless ship
```

```
ship ||= Spaceship.new
```



```
ship || ship = Spaceship.new
```

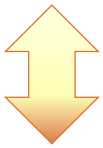
and and *or* vs. && and ||

|| && and or

- *and* and *or* have much lower precedence than && and ||
- && has higher precedence than ||
- *and* and *or* have the *same* precedence

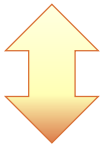
Flow Control with *and* and *or*

```
lander = Lander.locate(lander_id) and lander.recall
```



```
lander = Lander.locate(lander_id)
lander.recall if lander
```

```
if engine.cut_out?
  engine.restart or enable_emergency_power
end
```



```
if engine.cut_out?
  enable_emergency_power unless engine.restart
end
```

case Statement

```
case distance_to_dock
when "far away"
  lander.maintain_thrust
when "coasting time"
  lander.kill_thrust
when "collision imminent"
  lander.reverse_thrust
end
```

case Statement

```
thrust_power = case distance_to_dock
               when "far away"
                 100
               when "coasting time"
                 0
               when "collision imminent"
                 -100
               end
```

case Statement

```
thrust_power = case distance_to_dock
               when "far away" then 100
               when "coasting time" then 0
               when "collision imminent" then -100
               end
```

case Statement

```
case unit
when Lander
  lander.park
when Probe
  probe.retrieve_sample
  probe.transport_to_storage
else
  activate_alarm("Unidentified unit")
end
```

case Statement

```
case
when distance_to_dock > 100
  lander.maintain_thrust
when distance_to_dock > 2
  lander.kill_thrust
else
  lander.reverse_thrust
end
```

Looping: *while*

```
while high_alert?
  sound_system.play_siren_cycle
end
```

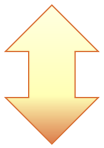
```
while high_alert? do sound_system.play_siren_cycle end
```

```
sound_system.play_siren_cycle while high_alert?
```

Looping: *until*

```
until ship.at_cruising_velocity?
  ship.accelerate
end
```

```
until ship.at_cruising_velocity? do ship.accelerate end
```



```
ship.accelerate until ship.at_cruising_velocity?
```

Looping: *begin/end*

```
begin
  lighting.start_flashing
  sound_system.play_siren_cycle
end while high_alert?


begin
  ship.accelerate
  make_fake_engine_noise
end until ship.at_cruising_velocity?
```

Looping: *for*

```
puts "Counting down to launch"

for i in [3, 2, 1]
  puts i
end



# print numbers from 1 to 10
for i in (1..10)
  puts i
end
```

Iterators and Blocks

```
[1, 2, 3].each do
  puts "This is Serenity, please respond"
end
```

```
ships = Spaceship.all
ships.each { |ship| puts ship.name }
```

Looping: *loop*

```
loop do
  go_another_light_year
  puts "This is not the edge of the universe"
end
```

Looping: Some Help from Numbers

```
10.upto(20) { |i| puts i }

20.downto(10) { |i| puts i }

3.times { puts "This is Serenity, please respond" }

1.step(10, 2) { |i| puts i }
```

Loop Flow

- *next* starts the next iteration of the loop

```
while message = comms.get_message
  next if message.type == "sync"
  message.process
end
```

Loop Flow

- *break* exits out of the loop

```
while message = comms.get_message
  message.process
  break if message.type == "voice"
end
```

```
text = while message = comms.get_message
         message.process
         break message.text if message.type == "voice"
       end
```

Loop Flow

- *redo* repeats the iteration without re-evaluating loop condition

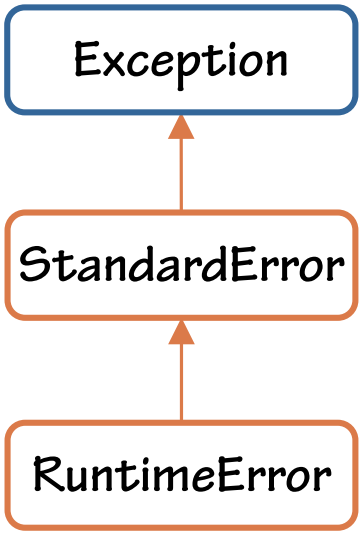
```
i = 0
while i < 3
  print "Please enter a positive number: "
  input = gets.to_i
  redo if input <= 0
  i += 1
end
```

Exceptions

```
def launch
  begin
    batten_hatches
  rescue
    puts "Couldn't batten hatches"
    return false
  end
  light_seatbelt_sign
end
```

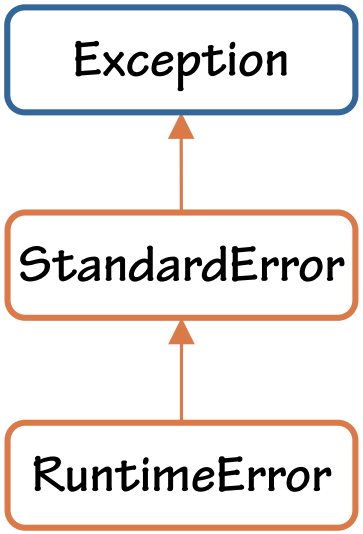
Exceptions

```
def launch
  batten_hatches
  light_seatbelt_sign
  true
rescue
  puts "Exception intercepted"
  false
end
```



Exceptions

```ruby
def launch
  batten_hatches
  light_seatbelt_sign
  true
rescue StandardError => e
  puts e.message
  false
end
```

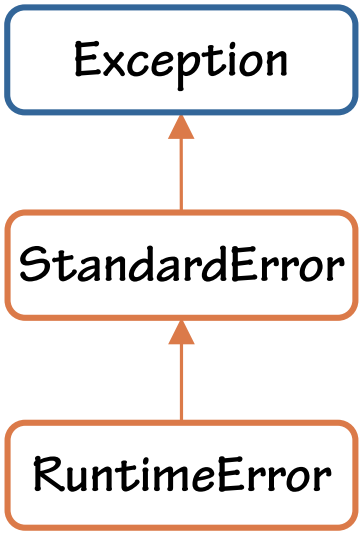



```ruby
e.backtrace
```

```
raise.rb:46:in `light_seatbelt_sign'
raise.rb:62:in `launch'
raise.rb:75:in `<top (required)>'
```

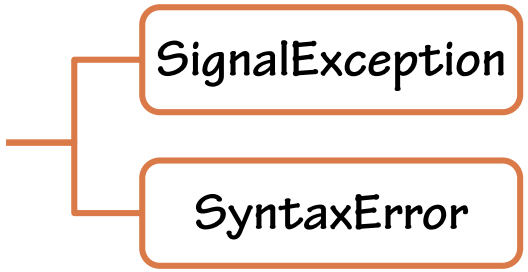
Exceptions

```
def launch
  batten_hatches
  light_seatbelt_sign
  true
rescue LightError
  puts "Lights not working, still launching"
  true
rescue StandardError => e
  puts e.message
  false
end
```



Exception Class

```
begin
  ship = Spaceship.new
  ship.launch
rescue Exception => e
  puts e.message
  puts e.backtrace
end
```



Raising Exceptions

```
def batten_hatches
  # ...
  raise "Doors jammed"
  # ...
end
```


```
def batten_hatches
  # ...
  raise HatchError, "Doors jammed"
  # ...
end
```

Re-Raising Exceptions

```
rescue HatchError => err
  puts $!.message
  raise
end
```

refers to the same exception object as *err*

Cleanup

```
def batten_hatches
  # ...
  raise "Doors jammed"
  # ...
end
```

Cleanup

```
def batten_hatches
  hatch_file = File.open("hatches.txt")
  # ...
  raise "Doors jammed"
  # ...
end
```

Cleanup

```
def batten_hatches
  hatch_file = File.open("hatches.txt")
  # ...
  raise HatchError, "Door jammed" if door.jammed?
  # ...
  true
rescue SystemCallError => e
  # handle file errors
  false
ensure
  hatch_file.close if hatch_file
end
```

Cleanup

```
def batten_hatches
  hatch_file = File.open("hatches.txt")
  # ...
  raise HatchError, "Door jammed" if door.jammed?
  # ...
  true
rescue SystemCallError => e
  # handle system call errors
  false
else
  puts "Well done, no exceptions"
ensure
  hatch_file.close if hatch_file
end
```

Retrying

```
def batten_hatches
  hatch_list = API.request("/hatches")
  # ...
end
```

Retrying

```
def batten_hatches
  1.upto (3) do |attempt|
    begin
      hatch_list = API.request("/hatches")
      break
    rescue RuntimeError => e
      puts e.message
      if attempt == 3
        puts "Request failed."
        raise
      end
    end
  end
end
```

Retrying

```
def batten_hatches
  hatch_list = API.request("/hatches")
  # ...
rescue RuntimeError => e
  attempts ||= 0
  attempts += 1
  if attempts < 3
    puts e.message + ". Retrying request."
    retry
  else
    puts "Request failed."
    raise
  end
end
```

Rescue Modifier

```
batten_hatches rescue false
```

throw/catch

```
result = catch :abort do
  probes.each do |probe|
    while sample = probe.get_sample
      result = sample.process

      throw :abort, result.message if result.code != :ok

      puts result.message
      sample.store
    end
  end
  "All samples processed"
end

puts result
```

throw/catch

```
def handle(sample)
  result = sample.process
  throw(:abort, result.message) if result.code != :ok
  puts result.message
  sample.store
end

result = catch(:abort) do
  probes.each do |probe|
    while sample = probe.get_sample
      handle(sample)
    end
  end
  "All samples processed"
end

puts result
```

Note on Scope

```
if true
  a = 10
end

puts a     # outputs 10



1.upto(10) { |i| puts i; a = i }

puts i     # error: undefined variable

puts a     # error: undefined variable
```

Summary

- Branching
- Loops
- Exception handling
- throw/catch