Error handling

Outline

1. Introduction
2. Demonstration
3. “Begin … Rescue” block
4. Flow of handling

1. Introduction

- No matter how carefully you code your script, your program is prone to failure for reasons beyond your control. A website that your script scrapes may suddenly be down. Or someone sharing the same hard drive may delete a file your program is supposed to read from.
- Circumstances such as these will crash your program. For any kind of long continuous task that you don't want to baby-sit and manually restart, you will need to write some exception-handling code to tell the program how to carry on when things go wrong.

2. Demonstration

- The Exception class handles nearly every kind of hiccup that might occur during runtime, including syntax screwups and incorrect type handling.
- We learned early on that adding numbers and strings with no type conversion would crash a program:

```
a = 10
b = "42"
a + b
```

=> The attempted arithmetic results in this error

2. Demonstration

The **begin/rescue** block is typically used on code in which you anticipate errors. There's only one line here for us to worry about:

```
a = 10
b = "42"

begin
  a + b
rescue
  puts "Could not add variables a (#{a.class}) and b (#{b.class})"
else
  puts "a + b is #{a + b}"
End
```

=> Executing the revised code gets us error

2. Demonstration

Let's feed this simple operation with an array of values of different types to see how the else clause comes into play:

```
values = [42, 'a', 'r', 9, 5, 10022, 8.7, "sharon", "Libya", "Mars", "12", 98, rand + rand, {:dog=>'cat'}, 100, nil,
200.0000, Object, 680, 3.14, "Steve", 78, "Argo"].shuffle

while values.length > 0
  a = values.pop
  b = values.pop

  begin
    a + b
  rescue
    puts "Could not add variables a (#{a.class}) and b (#{b.class})"
  else
    puts "a + b is #{a + b}"
  end
end

=> error
```


3. Begin...Rescue block

- This is the most basic error handling technique. It starts off with the keyword begin and acts in similar fashion to an if statement in that it your program flows to an alternate branch if an error is encountered.
- The main idea is to wrap any part of the program that could fail in this block. Commands that work with outside input, such as downloading a webpage or making calculation something based from user input, are points of failure. Something like puts "hello world" or 1 + 1 is not.

3. Begin...Rescue block

```
require "open-uri"
require "timeout"

remote_base_url = "http://en.wikipedia.org/wiki"

start_year = 1900
end_year = 2000
```

```
(start_year..end_year).each do |yr|
 begin
   rpage = open("#{remote_base_url}/#{yr}")
 rescue StandardError=>e
   puts "Error: #{e}"
 else
   rdata = rpage.read
 ensure
   puts "sleeping"
   sleep 5
 end

 if rdata
   File.open("copy-of-#{yr}.html", "w"){|f|
f.write(rdata)}
 end
end
```


3. Begin...Rescue block

- **begin**: This starts off the exception-handling block. Put in the operation(s) that is at risk of failing in this clause. In the above example, the open method for retrieving the webpage will throw an exception if the website is down.
- **rescue StandardError=>e**: This is the branch that executes if an exception or error is raised. Possible exceptions include: the website is down, or that it times out during a request. The rescue clause includes the code we want to execute in the event of an error or exception (there's a difference between the Ruby Exception and Error classes, which I will get to in a later revision).
- **else**: If all goes well, this is where the program branches to. In this example, we save the contents of the open method to a variable.
- **ensure**: This branch will execute whether an error/exception was rescued or not. Here, we've decided to sleep for 3 seconds no matter the outcome of the open method.

4. Flow of handling

The retry statement redirects the program back to the begin statement. This is helpful if your begin/rescue block is inside a loop and you want to retry the same command and parameters that previously resulted in failure.

4. Flow of handling

Using **retry**

```
for i in "A".."C"
  retries = 2
  begin
    puts "Executing command #{i}"
    raise "Exception: #{i}"
  rescue Exception=>e
    puts "\tCaught: #{e}"
    if retries > 0
      puts "\tTrying #{retries} more times\n"
      retries -= 1
      sleep 2
      retry
    end
  end
end
```


4. Flow of handling

Output:

```
Executing command A
   Caught: Exception: A
   Trying 2 more times
Executing command A
   Caught: Exception: A
   Trying 1 more times
Executing command A
   Caught: Exception: A
Executing command B
   Caught: Exception: B
   Trying 2 more times
```

```
Executing command B
   Caught: Exception: B
   Trying 1 more times
Executing command B
   Caught: Exception: B
Executing command C
   Caught: Exception: C
   Trying 2 more times
Executing command C
   Caught: Exception: C
   Trying 1 more times
Executing command C
   Caught: Exception: C
```


4. Flow of handling

Using retry with OpenURI

```
require 'open-uri'
remote_base_url = "http://en.wikipedia.org/wiki"

[1900, 1910, 'xj3490', 2000].each do |yr|

retries = 3
```

```
begin
  url = "#{remote_base_url}/#{yr}"
  puts "Getting page #{url}"
  rpage = open(url)
rescue StandardError=>e
  puts "\tError: #{e}"
  if retries > 0
    puts "\tTrying #{retries} more times"
    retries -= 1
    sleep 1
    retry
  else
    puts "\t\tCan't get #{yr}, so moving on"
  end
else
  puts "\tGot page for #{yr}"
ensure
  puts "Ensure branch; sleeping"
  sleep 1
end
```


4. Flow of handling

Output

```
Getting page http://en.wikipedia.org/wiki/1900
  Got page for 1900
Ensure branch; sleeping
Getting page http://en.wikipedia.org/wiki/1910
  Got page for 1910
Ensure branch; sleeping
Getting page http://en.wikipedia.org/wiki/xj3490
  Error: 403 Forbidden
  Trying 3 more times
Ensure branch; sleeping
```

```
Getting page http://en.wikipedia.org/wiki/xj3490
  Error: 403 Forbidden
  Trying 2 more times
Getting page http://en.wikipedia.org/wiki/xj3490
  Error: 403 Forbidden
  Trying 1 more times
Getting page http://en.wikipedia.org/wiki/xj3490
  Error: 403 Forbidden
    Can't get xj3490, so moving on
Ensure branch; sleeping
Getting page http://en.wikipedia.org/wiki/2000
  Got page for 2000
```


References

- http://ruby-doc.org/
- http://ruby.bastardsbook.com/chapters/exception-handling/
- http://culttt.com/2015/07/22/using-ruby-exceptions/
- http://phocode.com/ruby/ruby-module-va-exception/

Thank you for listening!