Exception Handling

Exceptions are rescued in a begin/end block:

```
begin
  # code that might raise
rescue
  # handle exception
end
```

If you are inside a method you do not need to use begin or end unless you wish to limit the scope of rescued exceptions:

```
def my_method
  # ...
rescue
  # ...
end
```

The same is true for a class or module.

You can assign the exception to a local variable by using => variable_name at the end of the rescue line:

```
begin
  # ...
rescue => exception
  warn exception.message
  raise # re-raise the current exception
end
```

By default StandardError and its subclasses are rescued. You can rescue a specific set of exception classes (and their subclasses) by listing them after rescue:

```
begin
  # ...
rescue ArgumentError, NameError
  # handle ArgumentError or NameError
end
```

You may rescue different types of exceptions in different ways:

```
begin
  # ...
rescue ArgumentError
  # handle ArgumentError
rescue NameError
  # handle NameError
rescue
  # handle any StandardError
end
```

The exception is matched to the rescue section starting at the top, and matches only once. If an ArgumentError is raised in the begin section it will not be handled in the StandardError section.

You may retry rescued exceptions:

```
begin
  # ...
rescue
  # do something that may change the result of the begin block
  retry
end
```

Execution will resume at the start of the begin block, so be careful not to create an infinite loop.

Inside a rescue block is the only valid location for retry, all other uses will raise a SyntaxError. If you wish to retry a block iteration use redo. See Control Expressions for details.

To always run some code whether an exception was raised or not, use ensure:

```
begin
  # ...
rescue
  # ...
ensure
  # this always runs
end
```

You may also run some code when an exception is not raised:

```
begin
  # ...
rescue
  # ...
else
  # this runs only when no exception was raised
```

```
ensure
  # ...
end
```