Thread Variables

This lesson discusses thread-local variables.

Thread Variables

A thread-local variable is a variable that has its scope localized to a single thread. We can create thread local variables using the instance method `thread_variable_set()` on the `Thread` class. And a thread local variable can be retrieved using the instance method `thread_variable_set()`.

In the code widget below, we create five threads and each thread retrieves its id stored as a thread local variable. Note that we are able to set thread local variable from outside the code block that a thread executes.

```
threads = 5.times.map do
  Thread.new do
    # wait for main thread to set-up ids
    sleep(0.5)
    myId = Thread.current.thread_variable_get("myId")
    puts "Thread with #{myId} exiting"

  end
end

# Set the id of each thread as a thread-local variable
5.times do |i|
  threads[i].thread_variable_set("myId", i)
end

# wait for all child threads to exit
threads.each(&:join)
```

Another way to declare thread-local variables is to treat a thread as if it were a hash and use square brackets with symbol keys to set and retrieve values. For instance:

```
Thread.current[:myId] = someValue
```

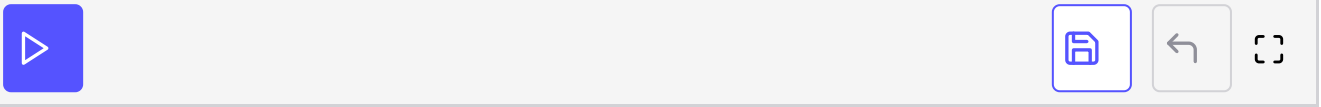
The previous example is rewritten using the hash syntax below:

```
threads = 5.times.map do
  Thread.new do
    # wait for main thread to set-up ids
    sleep(0.5)
    myId = Thread.current[:myId]
    puts "Thread with #{myId} exiting"

  end
end

# Set the id of each thread as a thread-local variable
5.times do |i|
  thread = threads[i]
  thread[:myId] = i
end

# wait for all child threads to exit
threads.each(&:join)
```

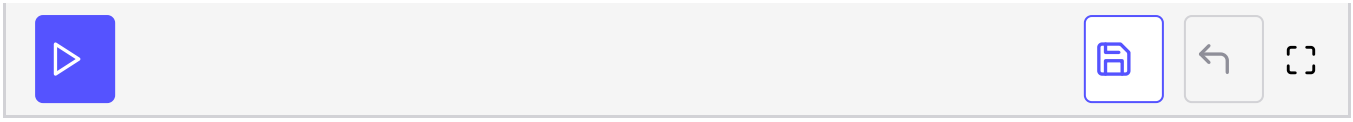


Any variables defined within a thread block are only accessible within the scope of the thread as the following example demonstrates.

```
Thread.new do

  threadLocalVar = "not_visible_outside"
  sleep(10)
end


puts threadLocalVar
```



> However variables in the outside scope can be captured within the thread block as shown by the following example:

```
varInOuterScope = 1

thread = Thread.new do
  isAlive = Thread.current.alive?
  puts("Child thread sees variable in outer scope : #{varInOuterScope}")

  # child thread mutates a variable outside the thread block
  varInOuterScope *= 10
end

thread.join()

puts("Main thread sees variable in outer scope after change by child thread : #{varInOuterSco
```