Ruby Programming

Code Blocks

- A block consists of chunks of code
- Can write the block in two ways:
  - Inline, between { and }
  - Multiline, between do and end
- Functions interact with blocks through the yield.
- Every time the function invokes yield control passes to the block.
- Control returns to the function when the block finishes.

```
$ irb --simple-prompt

>> 3.times {puts "A"; puts "B"}

A
B
A
B
A
B
=> 3
```

Here **3** is an object (of type Fixnum), it has a member function **times** which takes a block as a parameter. The function runs the block 3 times.

```
>> 3.times {|i| puts "Iteration number #{i}"}

Iteration number 0
Iteration number 1
```

```
Iteration number 2
=> 3
```

The **times** function passes a number into the block. The block gets that number in the variable **i** (as set by the **|i|**), then prints out the result.

```
>> -3.upto(2) {|i| puts (i.abs)}

3
2
1
0
1
2
=> -3
```

```
>> [3,4,7,2,9].find {|y| y%2==0}
=> 4
```

Here find function returns first element x of array such that the block returns true for x.

```
>> [3,4,7,2,9].find {|y| puts(y%2==0)}

false
true
false
true
false
=> nil
```

```
>> [3,4,7,2,9].find_all {|y| y%2==0}
=> [4, 2]
```

```
>> array = ['seabird', 'ladybug', 'seahorse',
'seashell', 'ladybird']

puts array.find_all { |item| item =~ /sea/ }
```

Output:

```
seabird
seahorse
seashell
```

```
>> person_attributes = " age; gender; education,
occupation"
person_attributes.split(/[,;]/).each {|line| puts line}
```

Output:

```
age
gender
education
occupation
```

```
population=Hash.new    #or can use {}
population["USA"] = 323
population["CA"] = 36
population.each { |key,value| puts "population of
#{key} is #{value} million" }
```

Output:

```
population of USA is 323 million
population of CA is 36 million
```

```
>> hash_store = {a: 5, b: 6, c: 7, d: 10, e: 11, f: 14}

#hash_store = {:a=> 5, :b=> 6, :c=> 7, :d=> 10, :e=>
11, :f=> 14}
```

```
puts hash_store.select{|key,value| value.odd?}
```

Output:

```
{:a=>5, :c=>7, :e=>11}
```

Exercise

Write a program to test on file.

```
# Reading lines from input file "test1.txt"

File.open("C:/Ruby24/prog/test1.txt", "r") do |f1|
f1.readlines.each {|line| puts line} end


# Copying contents from one file to another file

IO.copy_stream('C:/Ruby24/prog/test1.txt',
'C:/Ruby24/prog/test2.txt')


# Creating a file for writing

File.open("C:/Ruby24/prog/test3.txt", "w") do |f2|
f2.puts "Test file has created..." end
```

```
Input: test1.txt
This is the first line
This is the second line
This is the third line

Output: test2.txt (same as test1.txt)
Output: test3.txt
Test file has created...
```

```
# Appending contents on test1.txt
```

```
File.open("C:/Ruby24/prog/test1.txt", "a") do |line|
line.puts "\n" + "I am appending 4th line" end
```

Output (test1.txt) with the following contents

```
This is the first line
This is the second line
This is the third line
I am appending 4th line
```

yield statement

```
def yield_function
  puts "I am inside the yield function"
  yield
  puts "I am back to the yield function"
end

yield_function { puts "I am in the block" }
```

Output:

```
I am inside the yield function
I am in the block
I am back to the yield function
```

yield with parameters

```
def yield_function
  puts "I am inside the yield function"
  yield("Steve",23)
  puts "I am back to the yield function"
end

yield_function { |name,age| puts "#{name} is #{age}
years old" }
```

Output:

```
I am inside the yield function
Steve is 23 years old
I am back to the yield function
```

Introspection

- In general, it means to look inward. Humans can perform introspection, maybe questioning why we made a certain decision or finding out what makes us happy, and Ruby too can perform introspection.

- In Ruby, introspection is when our code asks questions about itself.

Exercise

```
class Person
 attr_accessor :name,:age

#The attr_accessor method is run at read time,
when ruby is constructing the class object

 @@total=0 #A class variable is shared by all
instances of a class

 def initialize (name, age)
  @@total+=1

  @name=name
  @age=age
#Instance variables are created for each class
instance and are accessible only within that
instance.
```

```
# Outside of the class definition, the value of
an instance variable can only be read or modified
via that instance's public methods
 end
 def to_person
  return "(#@name, #@age)"
 end

 def Person.total
  return "number of persons: #@@total"
 end
end

class Student < Person
 attr_accessor :school
 @@newtotal=0

 def initialize(name,age,school)
  super(name,age)

  @school=school
  @@newtotal+=1

 end

 def to_student
  return "(#@name, #@age, #@school)"
 end

 def Student.total
  return "number of students: #@@newtotal"
 end
end
```

```ruby
 p1 = Person.new("Aaron",22)
 puts p1.name
 puts p1.age
 puts p1.to_person
 puts " "

 s1 = Student.new("Matthew",27,"McGill")
 puts s1.name
 puts s1.age
 puts s1.to_student
 puts " "

 puts Person.total
 puts Student.total

# Introspection
puts p1.class
puts p1.instance_variables
puts p1.instance_of? Person
puts p1.kind_of? Person
puts Person.class_variables
puts p1.respond_to?("school")

puts " "
puts s1.class
puts s1.instance_variables
puts s1.instance_of? Student
puts Student.superclass
puts s1.respond_to?("age")
```

Results:

```
Aaron
22
(Aaron, 22)

Matthew
27
(Matthew, 27, McGill)

number of persons: 2
number of students: 1


# Introspection results

Person
@name
@age
true
true
@@total
false

Student
@name
@age
@school
true
Person
true
```