Formatting and parsing

Formatting and Parsing

- Additionally, for strings you can format strings with formatters.
- Best used for string interpolation that has many variables
- One liner usage

```
# Use two formatting codes.
format = "Number is %d, type is %s" % [13, "cat"]
puts format
```

- Multiple Lines

```
irb(main):001:0> s = "format %d"
=> "format %d"
irb(main):002:0> s % 3
=> "format 3"
irb(main):003:0>
```

Flow of Control – if-elsif-else

- Allows to execute the first condition that is met (that returns true)
- Following are the two ways to declare it: with and without then, it is not a good practice to include the *then* keyword

```
if 10 < 20 then
    print "10 is less than 20"
end
```

```
if 10 < 20
    print "10 is less than 20"
end
```

Not good practice to include “then” keyword
You can write too one line conditions
do something if (condition)

Flow of Control – if-elsif-else

- Using else: happens if none of the conditions above were met

```
if customerName == "Fred"
      print "Hello Fred!"
else
      print "You're not Fred! Where's Fred?"
end
```

This if and else should be preferred when there is more than one line inside the block. If not, it is preferable to use ternary operator.

Flow of Control – if-elsif-else

- Elsif constructs
  - Multiline as a best practice
  - Can go as an assignment, but include the else (best-practice) if you don't a nil value as well as project-based styles

```
if customerName == "Fred"
      print "Hello Fred!"
elsif customerName == "John"
      print "Hello John!"
elsif customername == "Robert"
      print "Hello Bob!"
end
```

Flow of Control – case-when statement

```
car = "Patriot"

manufacturer = case car
   when "Focus" then "Ford"
   when "Navigator" then "Lincoln"
   when "Camry" then "Toyota"
   when "Civic" then "Honda"
   when "Patriot" then "Jeep"
   when "Jetta" then "VW"
   when "Ceyene" then "Porsche"
   when "Outback" then "Subaru"
   when "520i" then "BMW"
   when "Tundra" then "Nissan"
   else "Unknown"
end

puts "The " + car  + " is made by "  + manufacturer
```

- Remember it accepts ranges

```
score = 70

result = case score
   when 0..40 then "Fail"
   when 41..60 then "Pass"
   when 61..70 then "Pass with Merit"
   when 71..100 then "Pass with Distinction"
   else "Invalid Score"
end

puts result
```

- It's not a good practice to include then, go to next line

```
1.  case capacity
2.  when 0
3.    "You ran out of gas."
4.  when 1..20
5.    "The tank is almost empty. Quickly,
    find a gas station!"
6.  when 21..70
7.    "You should be ok for now."
8.  when 71..100
9.    "The tank is almost full."
10. else
11.   "Error: capacity has an invalid
    value (#{capacity})"
```

Flow of Control – While and Until Loops

- While is a loop, something that keeps executing while the condition is met

```
i = 0
while i < 5 do
   puts i
   i += 1
end
```

Flow of Control – Unless and until

- Unless an if statement that executes only when the condition is false

```
[irb(main):003:0> unless false
[irb(main):004:1>   puts "i will execute"
[irb(main):005:1> end
 i will execute
 => nil
[irb(main):006:0> "i will execute" unless false
 => "i will execute"
 irb(main):007:0>
```

It is like an inverse while.
You can include an else to the unless (you cant on while)

Flow of Control – Loops and Ranges

- Using ranges in loops

```
for i in 1..8 do
    puts i
end
```

Flow of Control – .times method

- Click to ac

```
5.times { |i| puts i }

0
1
2
3
4
```


Flow of Control – downto

- Subtracts the value until the end is reached

```
irb(main):018:0> 5.downto(1) { |x| puts x }
5
4
3
2
1
=> 5
```

```
997
1000
 => ((1..1000).step(3))
2.7.5 :009 > (1..10).step(2).each { |x| puts x }
1
3
5
7
9
 => ((1..10).step(2))
```

Step on ranges

Flow of Control – redo

- Executes the block again with the same value
  - Be careful, may enter into infinite loops

Flow of Control – next

- Stops executing the current block at the current value and takes the next value. Please note the block will keep iterating until finished.

Useful Links

https://www.rubyguides.com/2019/09/ruby-next-break-keywords/
https://blog.appsignal.com/2018/06/05/redo-retry-next.html
https://www.rubyguides.com/2015/10/ruby-case/
https://www.techotopia.com/index.php/Ruby_Essentials