# Cases, Numbers, Arrays, Ranges, File IO

# case expressions

- 2 “flavors”
  1. Similar to a series of “`if`” statements
  2. (more common) specify a target next to `case` and each `when` clause is compared to target
     - Can use regular expressions!
- No fall-through logic !!!

# case expressions (Continued)

```
name = "Joey"
case # 1st flavor
  when name == "Joe"
    puts "Not exactly!"
  when Time.now.hour > 23
    puts "It's past your bed time!"
  else
    puts "Welcome back #{name}"
end # => Welcome back Joey

# 2nd flavor
case name
  when /[oye]{3}/ then puts "Joey?! Is it you?"
end
# => Joey?! Is it you?
```

# Numbers

- Full-fledged objects
  - Not primitives!
  - Have many methods
- Automatically convert between small and really large numbers
- Even simple operations are really calling methods in disguise

# Numbers

```
a = 23
b = 0
puts b.zero? # => true
puts a.even? # => false

1.upto(3) {puts "Hello there..."} # => Hello there
                                  # => Hello there
                                  # => Hello there
puts a.class # => Fixnum
puts "#{a**10} #{(a**10).class}" # => 41426511213649 Bignum

# syntactic sugar
puts "#{a + b} is the same as #{a.+(b)}" # => 23 is the same as 23

puts a.between?(20, 30) # => true
```

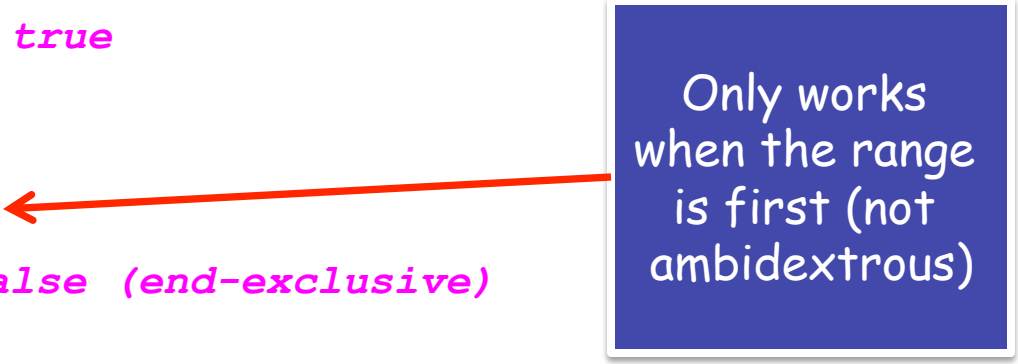
# Ranges

- Used to express natural sequences
  - `1..20`, `'a'...'z'`
- 2 dots all-inclusive, 3 dots end-exclusive
- Efficient! (only start and end stored)
- Can be converted to an array with `to_a`
- Also used for conditions and intervals

# Ranges

```
some_range = 1..3
puts some_range.max # => 3
puts some_range.include? 2 # => true

# Intervals
puts (1...10) === 5.3 # => true
puts ('a'...'r') === "r" # => false (end-exclusive)
age = 13
case age
  when 0..12 then puts "You are still a baby"
  when 13..99 then puts "You are such a teenager!"
  else puts "You are getting older..."
end
# => You are such a teenager!
```



# Arrays

- Collection of object refs (auto-expandable)
- Indexed using `[]` operator (method)
- Can be indexed with negative numbers or ranges
- Heterogeneous types allowed in the same array!
- Can use `%w{one two}` for string array creation

# Arrays

```
het_arr = [1, "two", :three] # heterogeneous types
puts het_arr[1] * 2 # => twotwo (array indices start at 0)

arr_words = %w{ pretty cool stuff going on here}
puts arr_words[-2] # => on
puts "#{arr_words.first} - #{arr_words.last}" # => pretty - here
p arr_words[-3, 2] # => ["going", "on"] (go back 3 and take 2)
p arr_words[2..4] # => ["stuff", "going", "on"] (range)
p arr_words[1...3] # => ["cool", "stuff"] (end-exclusive range)

# Make a String out of array elements separated by ';'
puts arr_words.join(';') # => pretty;cool;stuff;going;on;here
```

# Arrays

- Modifying arrays
  - Append with `push` or `<<`
  - Remove with `pop` or `shift`
  - Set with `[]=` (method)
    - Single `[index]` - replace with right side
    - Multiple – see examples in PickAxe, page 49
  - Sort or reverse with `sort!` and `reverse!`
  - Randomly pull an element out with `sample`

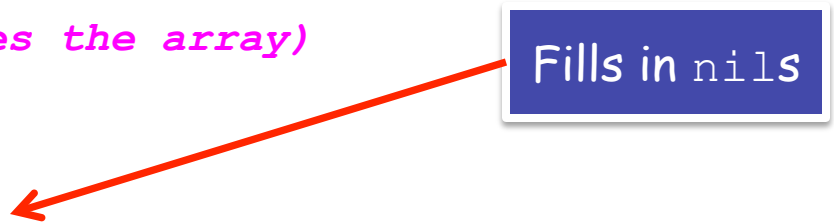
# Arrays

```
# You want a stack (LIFO)? Sure
stack = []; stack << "one"; stack.push ("two")
puts stack.pop # => two

# You meant a queue (FIFO)? We have those too...
queue = []; queue.push "one"; queue.push "two"
puts queue.shift # => one

a = [5,3,4,2].sort!.reverse!
p a # => [5,4,3,2] (actually modifies the array)

a[6] = 33
p a # => [5, 4, 3, 2, nil, nil, 33]
```



# Arrays

- Some useful array methods
  - `each`
  - `select`
  - `reject`
  - `map`
  - `inject`
- Many, many others…
- Another very important API to master!

# Array

```
a = [1, 3, 4, 7, 8, 10]
a.each { |num| print num } # => 1347810 (no new line)
puts # => (print new line)

new_arr = a.select { |num| (4..8) === num }
p new_arr # => [4,7,8]
new_arr = a.select { |num| (4..8) === num }.reject{ |num| num.even? }
p new_arr # => [7]

new_arr = a.map {|x| x * 3}
p new_arr # => [3, 9, 12, 21, 24, 30]

# Inject: 1. no args, 2. 1 arg and 3. no block
p a.inject { |sofar, next_one| sofar + next_one} # => 33 (sum)
puts a.inject(1) {|sofar, next_up| sofar * next_up} # => 6720
puts a.inject (:+) # => 33 (sum up all array members)
```

# File IO

```
# Write to file (w+ - open for writing and reading)
File.open("test.txt","w+") do |file|
  file.puts "Hello there"
  file.puts "no... really"
end # The file is automatically closed after the block executes

# Read from file
line_arr = []
File.open("test.txt", "r") do |file|
  file.each_line { |line| line_arr << line.chomp }
end
p line_arr # => ["Hello there", "no... really"]

# Read everything into a word array
words = IO.read("test.txt").split(/\s+/)
p words # => ["Hello", "there", "no...", "really"]
```