Lecture05 - Feeling Loopy

Tags

Outline

- Turn-inable adjustments
- Some notes on `print` and `return`
- Loops, loops, more loops

Turn-inable adjustments

- Assignment deadlines and release dates altered
    - We should discuss due dates…
- There will be no Assignment10 - content will be folded into final project
- `Lab04 == Lab05`
    - Lab06 will be completing Needleman-Wunsch
    - Lab07 will be implementing Smith-Waterman alignment
- Lab turn-in policy adjustment

Printing & returning confusion

- printing != returning
- BUT! The REPL prints return values (read evaluate **print** loop)

Compare:

```
function print_demo(a, b, c)
     @info a
     println(b)
     return c
 end

print_demo(1,2,3)

x = print_demo(4,5,6)

x
```

With

```
y = print_demo(7,8,9);

y

z = println(10)

z

typeof(z)
```

All julia functions `return` something

- sometimes that something is `nothing`
- Default is the last statement - that is:

```
function explicit(thing)
    return thing * 2
end

# is the same as

function implicit(thing)
    thing * 2
end
```

- Sometimes the thing `return`ed is the point ("fruitful" functions)
- Sometimes it's just a side-effect

Looping

- We often want to perform the same operation on multiple values
- One way to do this is with a loop

```
julia> for num in 0:2:10
           @info "The loop has hit $num"
       end
[ Info: The loop has hit 0
[ Info: The loop has hit 2
[ Info: The loop has hit 4
[ Info: The loop has hit 6
[ Info: The loop has hit 8
[ Info: The loop has hit 10
```

- There are many ways to loop...

`for` loops go through each item in a sequence

```
julia> for c in "Hello!"
           @info c
       end
[ Info: H
[ Info: e
[ Info: l
[ Info: l
[ Info: o
[ Info: !
```

`while` loops go until a condition is met

```
julia> counter = 0
0

julia> while counter < 5
           counter += 1
           @info "We've gotten into the loop $counter times!"
       end
[ Info: We've gotten into the loop 1 times!
[ Info: We've gotten into the loop 2 times!
[ Info: We've gotten into the loop 3 times!
[ Info: We've gotten into the loop 4 times!
[ Info: We've gotten into the loop 5 times!
```

But be careful that your condition will be met eventually...

```
julia> counter = 0
0

julia> while counter >= 0
```

```
           counter += 1
           @info "We've gotten into the loop $counter times!"
       end #...
```

`map` can apply a function to each item

- and returns a `Vector` of the results

```
julia> function my_func(thing)
           @info "Here's the thing: $thing"
           return thing ^ 2
       end
my_func (generic function with 1 method)

julia> map(my_func, [1, "Hello, World!", 2.3])
[ Info: Here's the thing: 1
[ Info: Here's the thing: Hello, World!
[ Info: Here's the thing: 2.3
3-element Vector{Any}:
 1
  "Hello, World!Hello, World!"
 5.289999999999999
```

`filter` applies a boolean function to each item, retains those that are `true`

```
julia> function my_bool(thing)
           @info "Here's the thing: $thing"
           return thing isa Int
       end
my_bool (generic function with 1 method)

julia> filter(my_bool, [1, "Hello, World!", 2.3])
[ Info: Here's the thing: 1
[ Info: Here's the thing: Hello, World!
[ Info: Here's the thing: 2.3
1-element Vector{Any}:
 1
```

What is loopable?

- anything that applies the "iterator" interface
  - OK... but what's that?
- For this class, think Arrays, Tuples, Strings
- For `Dict`s, you can iterate through keys, values, or both

Dictionary looping

```
julia> my_dict = Dict("bananas"=>2, "apples"=>5, "durian"=>0);

julia> for k in keys(my_dict) # note that dictionaries are not "ordered"
           @info "the key is $k"
       end
[ Info: the key is bananas
[ Info: the key is durian
[ Info: the key is apples

julia> for v in values(my_dict)
           @info "the value is $v"
       end
[ Info: the value is 2
[ Info: the value is 0
[ Info: the value is 5

julia> for (k,v) in my_dict
           @info "the key is $k, the value is $v"
       end
[ Info: the key is bananas, the value is 2
[ Info: the key is durian, the value is 0
[ Info: the key is apples, the value is 5
```

Count and loop at the same time with `enumerate`

```
julia> counter = 0
0

julia> for k in keys(my_dict)
           counter += 1
           @info "the counter is $counter, the key is $k"
       end
[ Info: the counter is 1, the key is bananas
[ Info: the counter is 2, the key is durian
[ Info: the counter is 3, the key is apples

julia> for (i, k) in enumerate(keys(my_dict))
           @info "the counter is $i, the key is $k"
       end
[ Info: the counter is 1, the key is bananas
[ Info: the counter is 2, the key is durian
[ Info: the counter is 3, the key is apples
```

Debugging loops

1. Go through one loop at a time (`first(iter)` and `last(iter)` can be very helpful)
2. Go through a subset of the data (eg use indexing or `enumerate`)
3. Add `println` or `@info` `@warn` statements

Lab05 (Lab04 in disguise)