For loops

INTERMEDIATE JULIA

Anthony Markham
Quantitative Developer

Course outline

- Control Structures
- Advanced Data Structures
- Advanced Functions in Julia
- DataFrame Operations and Extensibility

For loops - introduction

- A loop is a way to repeat a set of actions a known number of times.

```
for variable in iterable
    expression
end
```

```
shopping_list = ["Apples", "Bread", "Carrots", "Strawberries"]
```

```
println(shopping_list)
```

```
["Apples", "Bread", "Carrots", "Strawberries"]
```

Repeating single commands

- Accessing each element of the structure via indexing is messy and not practical.

```
println(shopping_list[1])
println(shopping_list[2])
println(shopping_list[3])
println(shopping_list[4])
```

For loops - the structure

- For loops allow us to simplify the process of extracting data from a structure
- The iterable is `shopping_list`
- The iterator is `item` , an arbitrary name

```
for item in shopping_list
    println(item)
end
```

```
Apples
Bread
Carrots
Strawberries
```

Iterating through the shopping list

- For loops allow us to simplify the process of extracting data from a structure
- The iterable is `shopping_list`
- The iterator is `item` , an arbitrary name

```
for item in shopping_list
    println(item)
    # first iteration, item = 'Apples'
end
```

```
Apples
```

Iterating through the shopping list

- For loops allow us to simplify the process of extracting data from a structure
- The iterable is `shopping_list`
- The iterator is `item` , an arbitrary name

```
for item in shopping_list
    println(item)
    # second iteration, item = 'Bread'
end
```

```
Apples
Bread
```

Iterating through the shopping list

- For loops allow us to simplify the process of extracting data from a structure
- The iterable is `shopping_list`
- The iterator is `item` , an arbitrary name

```julia
for item in shopping_list
    println(item)
    # third iteration, item = 'Carrots'
end
```

```
Apples
Bread
Carrots
```

Iterating through the shopping list

- For loops allow us to simplify the process of extracting data from a structure
- The iterable is `shopping_list`
- The iterator is `item` , an arbitrary name

```
for item in shopping_list
    println(item)
    # fourth iteration, item = 'Strawberries'
end
```

```
Apples
Bread
Carrots
Strawberries
```

Enumerate

- The enumerate function allows us to return an index and value pair when iterating over a data structure.

```
for (index, item) in enumerate(shopping_list)
    println(index, " ", item)
end
```

```
1Apples
2Bread
3Carrots
4Strawberries
```

```
shopping_list = ["Apples", "Bread", "Carrots", "Strawberries"]
```

Let's practice!

INTERMEDIATE JULIA

While loops

INTERMEDIATE JULIA

Anthony Markham
Quantitative Developer

While loops - syntax

- Less common than the `for` loop, but still useful in some cases.
- Repeat the `expression` while the `condition` is true.
- Often used to repeat an action until a certain condition is met.

```
while condition
    expression
end
```

While loops - example

- Repeat a set of actions until a condition is true

```julia
counter = 10
while counter != 0
    print(counter, " ")
    counter = counter - 1
end
```

```
10 9 8 7 6 5 4 3 2 1
```

While loops - example first iteration

```
# First iteration
counter = 10
while counter != 0  # counter = 10 here, so this is true
    print(counter, " ")  # print counter, equal to 10
    counter = counter - 1  # decrease the value of counter by 1
end
```

```
10
```

While loops - example second iteration

```
# Second iteration
counter = 10
while counter != 0  # counter now = 9 here, so this is true
    print(counter, " ")  # print counter, equal to 9
    counter = counter - 1  # decrease the value of counter by 1
end
```

```
10 9
```

While loops - example third iteration

```julia
# Third iteration
counter = 10
while counter != 0  # counter now = 8 here, so this is true
    print(counter, " ")  # print counter, equal to 8
    counter = counter - 1  # decrease the value of counter by 1
end
```

```
10 9 8
```

While loops - infinite loop

- What will happen if we forget to decrement the `counter` variable?

```
# Second iteration
counter = 10
while counter != 0  # counter now = 9 here, so this is true
    print(counter, " ")  # print counter, equal to 9
end
```

```
10 10 10 10 10 10 10 10 10
```

While loops - termination

- In the DataCamp environment, an infinite loop will cause the session to disconnect
- On your local machine, terminate the Julia program using Ctrl + C

Let's practice!

INTERMEDIATE JULIA

Ranges

INTERMEDIATE JULIA

Anthony Markham
Quantitative Developer

Ranges - UnitRange

- A `range` is an object, with its own data type.
- To create a range, we specify a start and stop, both of which are inclusive.

```
my_range = 1:10  # start:stop
```

- Printing a `range` will not reveal the sequence of values stored within the object!

```
my_range = 1:10
println(my_range)
```

```
1:10
```

- The UnitRange is the most basic type of range.

Ranges - StepRange

- To create a StepRange, we specify a step value, where each element in the range is determined by the step.

```
my_range = 0:10:50  # start:step:stop
```

- The `step` value determines what the next value in the range will be.

Ranges - StepRange definition

- Our defined StepRange has:
  - a `start` value of one
  - a `step` value of 10
  - an `end` value of 50

```julia
my_range = 1:10:50
println(my_range)
```

```
1:10:50
```

Ranges - for iteration

- Unpack the values in a range by iterating over the range
- A `for` loop can accomplish this, just as we did with a vector

```
my_range = 0:10:50
for value in my_range
    println(value)
end
```

```
0
10
20
30
40
50
```

Ranges - access

- Ranges can be accessed by element, just like vectors.
- `start` , `step` , and `stop` can be used to get the corresponding values for a range.

```
my_range = 0:10:50
println(my_range[2])
```

```
10
```

Ranges - access

- Ranges can be accessed by element, just like vectors.
- `start` , `step` , and `stop` can be used to get the corresponding values for a range.

```
println(my_range.start)
println(my_range.step)
println(my_range.stop)
```

```
0
10
50
```

Ranges - while iteration

- Iterating using a `while` loop requires accessing each individual range element.
- Ranges use the `[]` notation to access each element, just like vectors.
- Note we set `i = 1` as Julia starts indexing at 1, not 0!

```julia
i = 1

while i <= length(my_range)
    println(my_range[i])
    i = i + 1
end
```

Ranges - splat unpacking

- Splat operator `...` used to unpack an iterable.

```
my_range = 0:10:50
println([my_range...])
```

```
[0, 10, 20, 30, 40, 50]
```

- This is only one simple use case for the splat operator.

Let's practice!

INTERMEDIATE JULIA