Conditionals

INTRODUCTION TO JULIA

James Fulton
Climate informatics researcher

What are conditional expressions?

- Tell our computer to do some action if a condition is met
- Allow us to write code which makes its own decisions

```
is_raining = true

# Conditional expression
if is_raining
    println("Better get your coat")
end
```

```
Better get your coat
```

What are conditional expressions?

- Tell our computer to do some action if a condition is met
- Allow us to write code which makes its own decisions

```
is_raining = true

# Conditional expression
if is_raining
println("Better get your coat")
end
```

```
Better get your coat
```

What are conditional expressions?

```
is_raining = false


# Conditional expression
if is_raining
    println("Better get your coat")
end
```

Multiple lines of code under the if statement

```
is_raining = true

if is_raining
    # This can be many lines of code
    println("The weather is awful")
    println("Better get your coat")
end

# Code below end is always run
println("Ready to go")
```

```
The weather is awful
Better get your coat
Ready to go
```

```
is_raining = false

if is_raining
    # This can be many lines of code
    println("The weather is awful")
    println("Better get your coat")
end

# Code below end is always run
println("Ready to go")
```

```
Ready to go
```

Comparisons

When raining:

```
amount_of_rain = 1.

# Use comparison
is_raining = amount_of_rain > 0

print(is_raining)
```

```
true
```

When dry:

```
amount_of_rain = 0.

# Use comparison
is_raining = amount_of_rain > 0

print(is_raining)
```

```
false
```

Comparisons

When raining:

```
amount_of_rain = 1.

# Use comparison
is_raining = amount_of_rain > 0

# Conditional expression
if is_raining
    println("Better get your coat")
end
```

```
Better get your coat
```

When dry:

```
amount_of_rain = 0.

# Use comparison
is_raining = amount_of_rain > 0

# Conditional expression
if is_raining
    println("Better get your coat")
end
```

```

```

Comparisons

When raining:

```
amount_of_rain = 1.




# Conditional expression
if amount_of_rain>0
    println("Better get your coat")
end
```

```
Better get your coat
```

When dry:

```
amount_of_rain = 0.




# Conditional expression
if amount_of_rain>0
    println("Better get your coat")
end
```

```

```

Other comparisons

- `a == b` check if two values are equal

```
a = 1.


# Value of a is 1?
println(a==1)
```

```
true
```

```
# Data type of a is Float64?
println(typeof(a)==Float64)
```

```
true
```

Other comparisons

- `a == b` check if two values are equal
- `a != b` check if two values are not equal

```
a = 1.

# Value of a is not 1?
println(a!=1)
```

```
false
```

```
# Data type of a is not Float64?
println(typeof(a)!=Float64)
```

```
false
```

Other comparisons

- `a == b` check if two values are equal
- `a != b` check if two values are not equal
- `a > b` check if `a` greater than `b`
- `a >= b` check if greater than or equal to

```
a = 1.

# a is greater than 1?
println(a>1)
```

```
false
```

```
# a is greater than or equal to 1?
println(a>=1)
```

```
true
```

Other comparisons

- `a == b` check if two values are equal
- `a != b` check if two values are not equal
- `a > b` check if `a` greater than `b`
- `a >= b` check if greater than or equal to
- `a < b` check if `a` less than `b`
- `a <= b` check if less than or equal to

```
a = 1.

# a is less than 1?
println(a<1)
```

```
false
```

```
# a is less than or equal to 1?
println(a<=1)
```

```
true
```

When the condition is not met

```
amount_of_rain = 0.

if amount_of_rain == 0
    # Do this if conditon is met
    println("The sky looks clear")
else
    # Do this if not met
    println("Better get your coat")
end
```

```
The sky looks clear
```

```
amount_of_rain = 5.

if amount_of_rain == 0
    # Do this if conditon is met
    println("The sky looks clear")
else
    # Do this if not met
    println("Better get your coat")
end
```

```
Better get your coat
```

Additional conditions

```
amount_of_rain = 0.


if amount_of_rain == 0
    println("There is zero rain")
elseif amount_of_rain < 1
    # Add a second condition
    println("Better get your coat")
else
    println("That's a lot of rain, stay home")
end
```

```
There is zero rain
```

Additional conditions

```
amount_of_rain = 0.5


if amount_of_rain == 0
    println("There is zero rain")
elseif amount_of_rain < 1
    # Add a second condition
    println("Better get your coat")
else
    println("That's a lot of rain, stay home")
end
```

```
Better get your coat
```

Additional conditions

```
amount_of_rain = 2


if amount_of_rain == 0
    println("There is zero rain")
elseif amount_of_rain < 1
    # Add a second condition
    println("Better get your coat")
else
    println("That's a lot of rain, stay home")
end
```

```
That's a lot of rain, stay home
```

Multiple elseif's

```
amount_of_rain = 2

if amount_of_rain == 0
    println("There is zero rain")
elseif amount_of_rain < 1     # <--- many elseif conditions
    println("Better get your coat")
elseif amount_of_rain < 5     # <--- many elseif conditions
    println("You're going to need a bigger coat")
else
    println("That's a lot of rain, stay home")
end
```

```
You're going to need a bigger coat
```

Let's practice!

INTRODUCTION TO JULIA

Basic Functions

INTRODUCTION TO JULIA

James Fulton
Climate informatics researcher

What are functions?

Functions you have used:

- `println()`
- `typeof()`
- `string()`
- `push!()`
- `pop!()`
- `append!()`
- `length()`
- `sort()`

```
x = [2,1,3]

# Takes an array, returns integer
l = length(x)

# Takes an array, returns sorted array
x_sorted = sort(x)

# Takes a value, prints it to console
println(l)
```

Why use functions?

- Allows us to focus on program structure
- Can ignore irrelevant details of how a function works

Writing custom functions

```
# Declare function to convert temperatures
function fahrenheit2celsius(temp)
    # Function body
    return (temp - 32) * 5/9
end

# Use function
println(fahrenheit2celsius(212))
```

```
100.0
```

Writing custom functions

```
# Declare function to convert temperatures
function fahrenheit2celsius(temp)
    # Function body
    return (temp - 32) * 5/9
end

# Use function many times
println(fahrenheit2celsius(212))
println(fahrenheit2celsius(100))
```

```
100.0
37.77
```

Longer functions

```
# Declare function to convert temperatures
function fahrenheit2celsius(temp)
    # Function body
    temp_sub = temp - 32
    temp_c = temp_sub * 5/9
    return temp_c
end

t = fahrenheit2celsius(212)
println(t)
```

```
100.0
```

Longer functions

```
# Declare function to convert temperatures
function fahrenheit2celsius(temp)
    # Function body
    temp_sub = temp - 32        # variable inside function not available outside
    temp_c = temp_sub * 5/9
    return temp_c
end

t = fahrenheit2celsius(212)
println(temp_sub)
```

```
ERROR: UndefVarError: temp_sub not defined
```

Return keyword

```
function x_or_zero(x)
    if x>0
        return x
    else
        return 0
    end
end

println(x_or_zero(-3))
println(x_or_zero(3))
```

```
0
3
```

Return keyword

```
# Function with longer body
function check_if_raining(rain_amount)
    is_raining = rain_amount > 0
    if is_raining
        println("Better get your coat")
    else
        println("The sky looks clear")
    end
end

check_if_raining(0.2) # Function returns nothing - only prints
```

```
Better get your coat
```

Multiple arguments

```
function power(x, y)
    return x^y
end

# Use function to calculate 5*5
println(power(5, 2))
```

```
25
```

```
# Use function to calculate 2*2*2*2*2
println(power(2, 5))
```

```
32
```

Broadcasting functions

```
function fahrenheit2celsius(temp)
    return (temp - 32) * 5/9
end

temps_f = [212, 32, 100]

# Function not written to work with arrays
temps_c = fahrenheit2celsius(temps_f)
```

```
ERROR: MethodError: ...
```

Broadcasting functions

```
function fahrenheit2celsius(temp)
    return (temp - 32) * 5/9
end

temps_f = [212, 32, 100]

# Broadcast function with dot syntax
temps_c = fahrenheit2celsius.(temps_f)
println(temps_c)
```

```
[100.0, 0.0, 37.77]
```

Broadcasting functions

```
x = ["one", 2, 3.0]

# Broadcast using typeof function
println(typeof.(x))
```

```
[String, Int64, Float64]
```

Broadcasting multiple arguments

```
function power(x, y)
    return x^y
end

x_arr = [1,2,3,4,5]



# Square each element of the array
println(power.(x_arr, 2))
```

```
[1, 4, 9, 16, 25]
```

Broadcasting multiple arguments

```
function power(x, y)
    return x^y
end

x_arr = [1,2,3,4,5]
y_arr = [1,2,3,4,5]

# Use function on x_arr and y_arr
println(power.(x_arr, y_arr))
```

```
[1, 4, 27, 256, 3125]
```

Let's practice!

INTRODUCTION TO JULIA

Mutating functions and multiple dispatch

INTRODUCTION TO JULIA

James Fulton
Climate informatics researcher

Mutating functions

Some functions modify inputs

Starting with the array:

```
x = [1,2,3]
```

```
append!(x, [4,5,6])
println(x)
```

```
[1, 2, 3, 4, 5, 6]
```

```
push!(x, 4)
println(x)
```

```
[1, 2, 3, 4]
```

```
pop!(x)
println(x)
```

```
[1, 2]
```

Non-mutating functions

Starting with the array:

```
x = [3,1,2]
```

```
l = length(x)
println(x)
```

```
[3, 1, 2]
```

```
x_sorted = sort(x)
println(x)
```

```
[3, 1, 2]
```

```
x_type = typeof(x)
println(x)
```

```
[3, 1, 2]
```

Mutating and non-mutating functions

Mutating functions change their inputs

- `pop!()`
- `push!()`
- `append!()`
- ...

Non-mutating functions do not change their inputs

- `sort()`
- `println()`
- `typeof()`
- `string()`
- `length()`
- ...

Writing a mutating function

```
function modify_array!(x)
    x[1] = 0
end

# Try to mutate y
y = [1,2,3,4,5]
modify_array!(y)

# y has changed
print(y)
```

```
[0, 2, 3, 4, 5]
```

```
function modify_array!(x)
    x = [0,2,3,4,5]
end

# Try to mutate y
y = [1,2,3,4,5]
modify_array!(y)

# y has changed
print(y)
```

```
[1, 2, 3, 4, 5]
```

Writing a mutating function

```
function modify_array!(x)
    x[1] = 0
end

# Try to mutate y
y = [1,2,3,4,5]
modify_array!(y)

print(x)
```

```
ERROR: UndefVarError: x not defined
```

```
function modify_array!(x)
    x = [0,2,3,4,5]
end

# Try to mutate y
y = [1,2,3,4,5]
modify_array!(y)

print(x)
```

```
ERROR: UndefVarError: x not defined
```

Writing a mutating function

```
function setarray2zero!(x)
    x .= 0
end

y = [1,2,3,4,5]
setarray2zero!(y)

print(y)
```

```
[0, 0, 0, 0, 0]
```

Writing a mutating function

```
function modify_array!(x)
    x .= x .- 1
end

y = [1,2,3,4,5]

modify_array!(y)

print(y)
```

```
[0, 1, 2, 3, 4]
```

Multiple dispatch

```
function double(x)
    return x*2
end
```

```
println(double(2)) # Works on integers
```

```
4
```

```
println(double(10.0)) # Works on floats
```

```
20.0
```

```
println(double("yo")) # Not on strings
```

```
ERROR: MethodError: ...
```

Multiple dispatch

```
function double(x)
    return x*2
end

function double(x::String)
    return x*x
end
```

```
println(double(2)) # Works on integers
```

```
4
```

```
println(double(10.0)) # Works on floats
```

```
20.0
```

```
println(double("yo")) # Works on strings
```

```
yoyo
```

Multiple dispatch

```
function double(x)
    return x*2
end

function double(x::String)
    return x*x
end

function double(x::Bool)
    return x
end
```

```
println(double(2)) # Works on integers
```

```
4
```

```
println(double(10.0)) # Works on floats
```

```
20.0
```

```
println(double("yo")) # Works on strings
```

```
yoyo
```

Multiple dispatch

```
function double(x::String)
    return x*x
end

function double(x::Bool)
    return x
end
```

```
println(double(2)) # Not on integers
```

```
ERROR: MethodError: ...
```

```
println(double(10.0)) # Not on floats
```

```
ERROR: MethodError: ...
```

```
println(double("yo")) # Works on strings
```

```
yoyo
```

Multiple dispatch

```
function double(x)
    return x*2
end
```

Let's practice!

INTRODUCTION TO JULIA

Using packages

INTRODUCTION TO JULIA

James Fulton
Climate informatics researcher

Packages

- A collection of Julia files from which you can import
- Popular packages include
  - `Statistics.jl` - calculating descriptive statistics
  - `DataFrames.jl` - storing and manipulating tabular data
  - `CSV.jl` - loading and saving CSV data
  - `Plots.jl` - creating visualizations
- Thousands more packages exist
- Some lists of packages found here:
  - **https://julialang.org/packages**

Installing packages

```
import MyPackage
```

```
| Package MyPackage not found, but a package named MyPackage is available from a registry.
| Install package?
|   (@v1.7) pkg> add MyPackage
|_ (y/n) [y]:
```

Importing packages

```
import Statistics

m = Statistics.mean([1,2,3])
println(m)
```

```
2.0
```

```
using Statistics

m = Statistics.mean([1,2,3])
println(m)
```

```
2.0
```

```
m = mean([1,2,3])
println(m)
```

```
ERROR: UndefVarError: mean not defined
```

```
m = mean([1,2,3])
println(m)
```

```
2.0
```

Importing packages

```
import Statistics as sts

m = sts.mean([1,2,3])
println(m)
```

```
2.0
```

```
using Statistics

m = Statistics.mean([1,2,3])
println(m)
```

```
2.0
```

```
m = mean([1,2,3])
println(m)
```

```
2.0
```

The Statistics package

- `mean()` - Calculate mean of array
- `median()` - Calculate median value of array
- `std()` - Calculate standard deviation of array values
- `var()` - Calculate variance of array values

```
mean_x = Statistics.mean(x_arr)

median_x = Statistics.median(x_arr)

std_x = Statistics.std(x_arr)

var_x = Statistics.var(x_arr)
```

Let's practice!

INTRODUCTION TO JULIA