# Introduction to strings

INTRODUCTION TO JULIA

**James Fulton**
Climate informatics researcher

datacamp

# Strings

```
# Strings are surrounded by " "
name = "Jane" # Cannot use 'Jane'

# Strings can be any length
book = "It is a truth universally acknowledged, ..."

println(name)
println(book)
```

```
Jane
It is a truth universally acknowledged, ...
```

# Triple quotes

```
# Triple quotes
poem = """Beware the Jabberwock, my son!

      The jaws that bite, the claws that catch!"""

println(poem)
```

```
Beware the Jabberwock, my son!

The jaws that bite, the claws that catch!
```

# Triple quotes

```
greeting = """ " Well hello there " """

println(greeting)
```

```
 " Well hello there "
```

```
greeting = " " Well hello there " "
```

```
ERROR: syntax: ...
```

# Concatenating strings

```
name = "James"

greeting = "Well hello there, "

# Concatenate two strings
println(greeting*name)
```

```
Well hello there, James
```

# String interpolation

```
name = "James"

# Interpolate with $ symbol
greeting = "Well hello there, $name"

println(greeting)
```

```
Well hello there, James
```

# String interpolation

```
x_int = 10

# Insert integer into string
println("The value is $x_int")
```

```
The value is 10
```

```
x_float = 1.0

# Insert float into string
println("The value is $x_float")
```

```
The value is 1.0
```

```
x_bool = true

# Insert boolean into string
println("The value is $x_bool")
```

```
The value is true
```

```
x_char = 'A'

# Insert character into string
println("The value is $x_char")
```

```
The value is A
```

# String interpolation

```
x = 10
y = 3

# Insert x*y into string
println("The product of x and y is $(x*y)")
```

```
The product of x and y is 30
```

# String interpolation

```
x = 10
y = 3

# Insert x*y into string
println("The product of x and y is \$(x*y)")
```

```
The product of x and y is $(x*y)
```

# Indexing strings

```
# Customer's seat
seat = "E5"

# Select character
row = seat[1]        # this returns 'E'

println(row)
println(typeof(row))
```

```
E
Char
```

# Indexing strings

```
# Customer's seat
seat = "E5"

# Select characters
row = seat[1]        # this returns 'E'
number = seat[2]     # this returns '5'

println("Your seat is in row $row, seat number $number.")
```

```
Your seat is in row E, seat number 5.
```

# Indexing strings

```
# Customer's seat
seat = "E5"

# Select characters
row = seat[1]        # this returns 'E'
number = seat[end]   # this returns '5'

println("Your seat is in row $row, seat number $number.")
```

```
Your seat is in row E, seat number 5.
```

# Indexing strings

```
# Customer's seat
seat = "E5"

# Select characters
row = seat[end-1]   # this returns 'E'
number = seat[end]  # this returns '5'

println("Your seat is in row $row, seat number $number.")
```

```
Your seat is in row E, seat number 5.
```

# Slicing strings

```
receipt = "08:30 - coffee - \$3.50"




println(receipt)
```

```
08:30 - coffee - $3.50
```

# Slicing strings

```
# Index position:
#           12345...
receipt = "08:30 - coffee - \$3.50"

time = receipt[1:5]          # Select first 5 characters



println(time)
```

```
08:30
```

# Slicing strings

```
# Index position from end:
#                              4321end
receipt = "08:30 - coffee - \$3.50"

time = receipt[1:5]          # Select first 5 characters
price = receipt[end-4:end]   # Select last 5 characters

println(time)
println(price)
```

```
08:30
$3.50
```

# Let's practice!

INTRODUCTION TO JULIA

# Introduction to arrays

INTRODUCTION TO JULIA

**James Fulton**
Climate informatics researcher

# What is an array?

```
# Store run times with many variables
runtime1 = 33.1
runtime2 = 32.7
runtime3 = 34.2
runtime4 = 31.9
```

```
# Store runtimes in array
runtimes = [33.1, 32.7, 34.2, 31.9]
```

- List of values
- Surrounded by `[` `]`
- With `,` in beween values

# Arrays vs. vectors vs. matrices

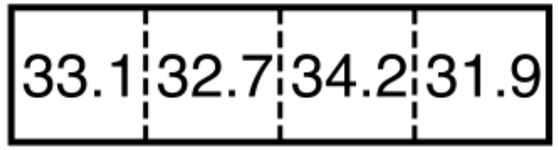
```
# Store runtimes in array
runtimes = [33.1, 32.7, 34.2, 31.9]


println(typeof(runtimes))
```

```
Vector{Float64}
```

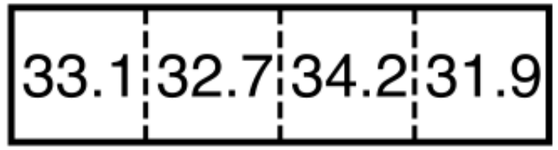
# Arrays vs. vectors vs. matrices

Vector is 1D array

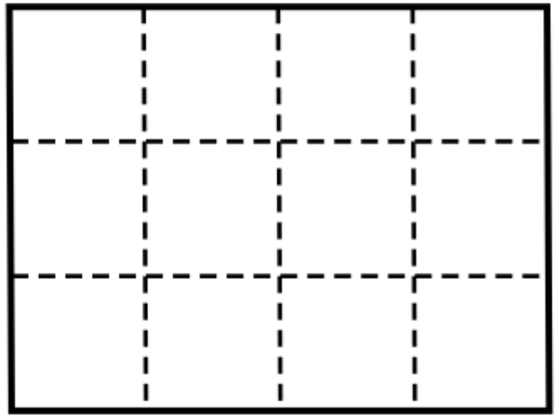


# Arrays vs. vectors vs. matrices

Vector is 1D array

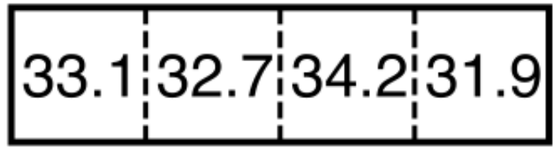



Matrix is 2D array

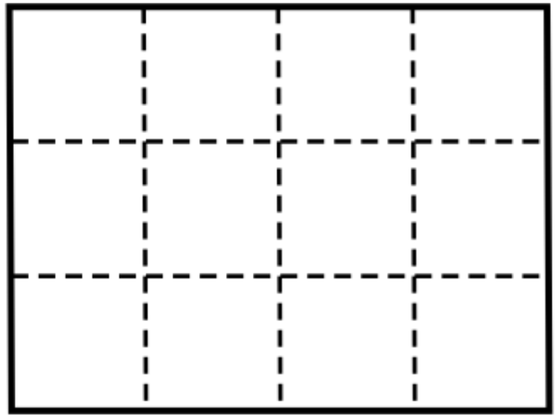


# Arrays vs. vectors vs. matrices

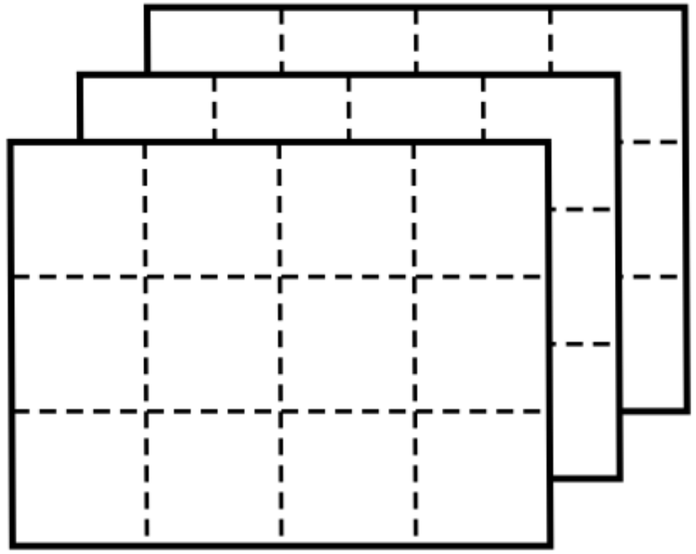
Vector is 1D array




Matrix is 2D array



Arrays can have any number of dimensions



# Array data types

```
# Store runtimes in array
runtimes = [33.1, 32.7, 34.2, 31.9]


println(typeof(runtimes))
```

```
Vector{Float64}
```

# Array data types

```
# Store runtimes in array
runtimes = [33.1, 32.7, 34.2, 31.9]


println(eltype(runtimes))
```

```
Float64
```

# Array data types

```
# Store integers in array
number_of_customers = [11, 19, 31, 27]

println(typeof(number_of_customers))
```

```
Vector{Int64}
```

```
# Store strings in array
names = ["Amit", "Barbara", "Carlos"]

println(typeof(names))
```

```
Vector{String}
```

```
# Store characters in array
grades = ['A', 'B', 'B', 'A']

println(typeof(grades))
```

```
Vector{Char}
```

```
# Store booleans in array
correct_answers = [true, false, true]

println(typeof(correct_answers))
```

```
Vector{Bool}
```

# Mixed data types

```
# Store multiple types in array
#         string   bool  int char float
items = ["James", true, 10, 'B', -20.3]

println(typeof(items))
```

```
Vector{Any}
```

# Mixed data types

```
# Store array of item names
item_names = ["chalk", "cheese", "eggs", "ham"]

# Store array of item prices
item_prices = [2.30, 3.50, 4.25, 2.00]

println(typeof(item_names))
println(typeof(item_prices))
```

```
Vector{String}
Vector{Float64}
```

# Indexing arrays

```
# Index:          1         2         3         4
item_names = ["chalk", "cheese", "eggs", "ham"]

# Index:          1     2     3     4
item_prices = [2.30, 3.50, 4.25, 2.00]

println(item_names[1])
println(item_prices[1])
```

```
chalk
2.30
```

# Indexing arrays

```
# Index:          1         2         3         4
item_names = ["chalk", "cheese", "eggs", "ham"]

# Index:          1     2     3     4
item_prices = [2.30, 3.50, 4.25, 2.00]

println(item_names[end])
println(item_prices[end])
```

```
ham
2.00
```

# Indexing arrays

```
# Index:          1         2         3         4
item_names = ["chalk", "cheese", "eggs", "ham"]

# Index:          1     2     3     4
item_prices = [2.30, 3.50, 4.25, 2.00]

println(item_names[end-1])
println(item_prices[end-1])
```

```
eggs
4.25
```

# Slicing arrays

```
# Index:          1         2         3         4
item_names = ["chalk", "cheese", "eggs", "ham"]

# Index:          1     2     3     4
item_prices = [2.30, 3.50, 4.25, 2.00]

println(item_names[1:2])
```

```
["chalk", "cheese"]
```

# Let's practice!

INTRODUCTION TO JULIA

# Working with arrays

INTRODUCTION TO JULIA

**James Fulton**
Climate informatics researcher

# Adding an element to the end of an array

```
# Predefine array
x = [1,2,3,4]

# Add the number 5 to end of array
push!(x, 5)

println(x)
```

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

# Adding an element to the end of an array

```
# Predefine array
x = [1,2,3,4]

# Add the float 5.0 to end of array
push!(x, 5.0)

println(x)
println(eltype(x))
```

```
[1,2,3,4,5]
Int64
```

# Adding an element to the end of an array

```
# Predefine array
x = [1,2,3,4]

# Add the float 5.2 to end of array
push!(x, 5.2)
```

```
ERROR: InexactError: Int64(5.2)
```

# Creating an array of given type

```
# Create float array
x = Float64[1,2,3,4]

println(typeof(x))

# Add the float 5.2 to end of array
push!(x, 5.2)

println(x)
```

```
Vector{Float64}
[1.0, 2.0, 3.0, 4.0, 5.2]
```

# Creating an array of given type

```
# Create empty float array
x = Float64[]


println(typeof(x))
println(x)
```

```
Vector{Float64}
Float64[]
```

# Creating an array of given type

```
# Create empty string array
x = String[]


println(typeof(x))
println(x)
```

```
Vector{String}
String[]
```

# Adding elements to the end of an array

```
# Create empty string array
x = String[]

# Add some elements to the array
push!(x, "one")
push!(x, "two")
push!(x, "three")

println(x)
```

```
["one", "two", "three"]
```

```
# Create empty string array
x = String[]

# Add some elements to the array
append!(x, ["one", "two", "three"])



println(x)
```

```
["one", "two", "three"]
```

# Removing the last element

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

# Remove 1 element from end
x = x[1:end-1]

println(x)
```

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

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

# Remove 1 element from end
last_element = pop!(x)

println(x)
println(last_element)
```

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

# Creating array of defined length

```
# Create integer array with 4 zeros
x = zeros(Int64, 4)


println(x)
```

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

# Replacing an element

```
# Create integer array with 4 zeros
x = zeros(Int64, 4)

# Replace element in position 3 with value 1
x[3] = 1

println(x)
```

```
[0, 0, 1, 0]
```

# Replacing many elements

```
# Create integer array with 4 zeros
x = zeros(Int64, 4)

# Replace many elements
x[2:3] = [2,3]

println(x)
```

```
[0, 2, 3, 0]
```

# Cheatsheet

- Add single element - `push!(x, 1)`
- Add many elements - `append!(x, [1,2,3])`
- Remove last element - `pop!(x)`
- Create array of given type - `Int64[1,2,3]` , `Float64[1,2,3]` , etc.
- Create empty array of given type - `Int64[]` , `Float64[]` , etc.
- Create an array full of zeros - `zeros(Int64, n)`
- Replace element - `x[index] = value`
- Replace many elements - `x[a:b] = [value1, value2, ...]`

# Let's practice!

INTRODUCTION TO JULIA

# Operating on arrays

INTRODUCTION TO JULIA

**James Fulton**
Climate informatics researcher

datacamp

# Basic array functions

```
# An array with 6 elements
x = ["a", "b", "sea", "d", "e", "f"]

# Find the length of the array
l = length(x)

println(l)
```

```
6
```

# Basic array functions

```
# An array with 6 elements
x = ["a", "b", "sea", "d", "e", "f"]

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

```
["a", "b", "d", "e", "f", "sea"]
```

```
# An array with 6 elements
x = [100, 95, 9, 22, 75, 58]

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

```
[9, 22, 58, 75, 95, 100]
```

# Vectorized operations

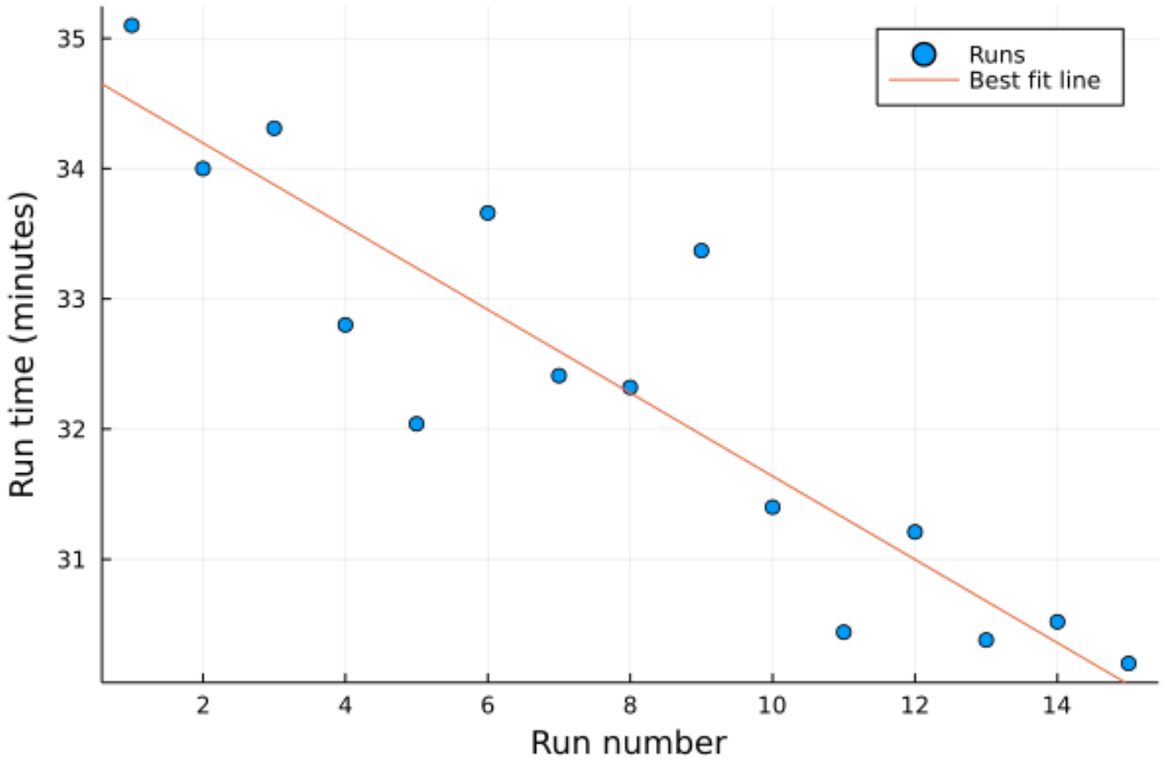
```
# Gradient and intercept
m = -0.32
c = 34.8

# Next run number is 16
x = 16

# Predict next run time
y = m * x + c

println(y)
```

```
29.68
```

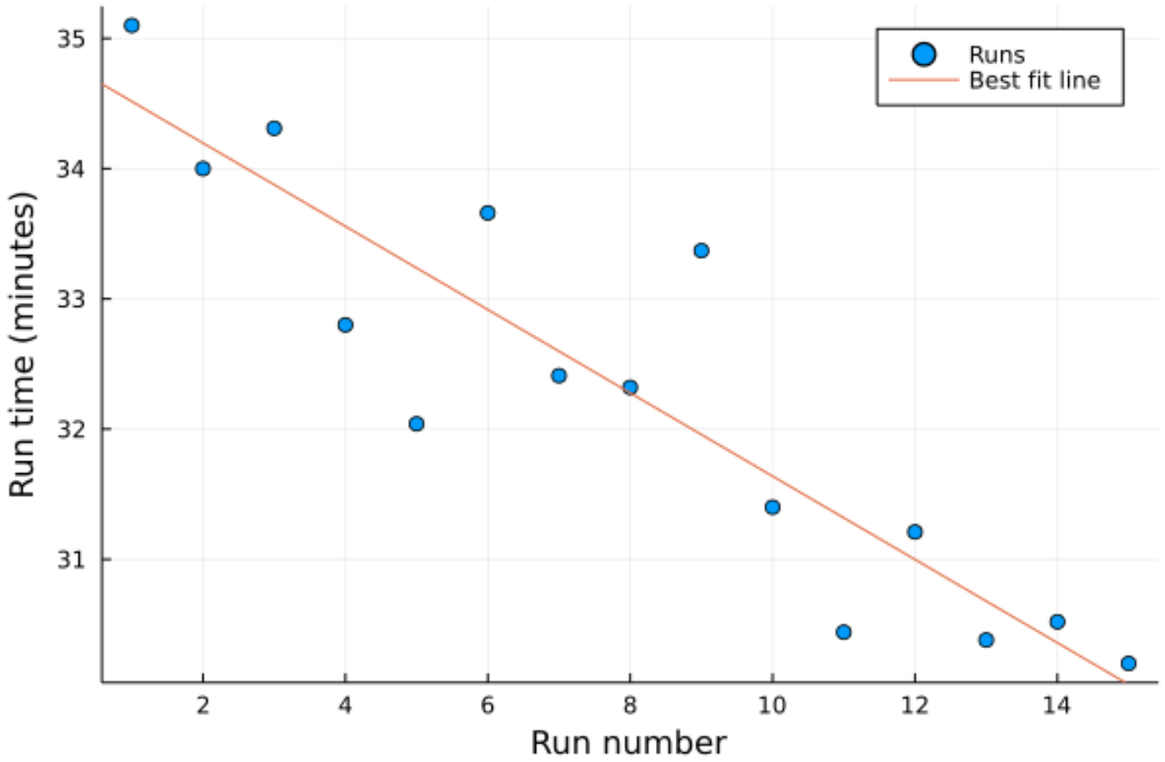


# Vectorized operations

```
# Gradient and intercept
m = -0.32
c = 34.8

# Next run numbers
x = [16, 17, 18, 19, 20]

# Predict next run time
y = m * x + c
```




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

# Array addition

Add scalar

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



# Answer we expect is [3,4,5]
println(a .+ 2)
```

```
[3, 4, 5]
```

Add array

```
a = [1,2,3]
b = [1,2,3]


# Answer we expect is [2,4,6]
println(a .+ b)
```

```
[2, 4, 6]
```

# Array addition

Add scalar

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



# Answer we expect is [3,4,5]
println(a + 2)
```

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

Add array

```
a = [1,2,3]
b = [1,2,3]

# Answer we expect is [2,4,6]
println(a + b)
```

```
[2, 4, 6]
```

# Array subtraction

```
# Subtract scalar
println(a .- 1)
```

```
[0, 1, 2]
```

```
# Subtract array
println(a .- b)
```

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

```
# Subtract scalar
println(a - 1)
```

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

```
# Subtract array
println(a - b)
```

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

# Array multiplication

Multiply by scalar

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



# Answer we expect is [5,10,15]
println(a .* 5)
```

```
[5, 10, 15]
```

Multiply by array

```
a = [1,2,3]
b = [1,2,3]

# Answer we expect is [1,4,9]
println(a .* b)
```

```
[1, 4, 9]
```

# Array multiplication

Multiply by scalar

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



# Answer we expect is [5,10,15]
println(a * 5)
```

```
[5, 10, 15]
```

Multiply by array

```
a = [1,2,3]
b = [1,2,3]

# Answer we expect is [1,4,9]
println(a * b)
```

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

# Array division

```
# Divide by scalar
println(a ./ 2)
```

```
[0.5, 1.0, 1.5]
```

```
# Divide by array
println(a ./ b)
```

```
[1.0, 1.0, 1.0]
```

```
# Divide by scalar
println(a / 2)
```

```
[0.5, 1.0, 1.5]
```

```
# Divide by array
println(a / b)
```

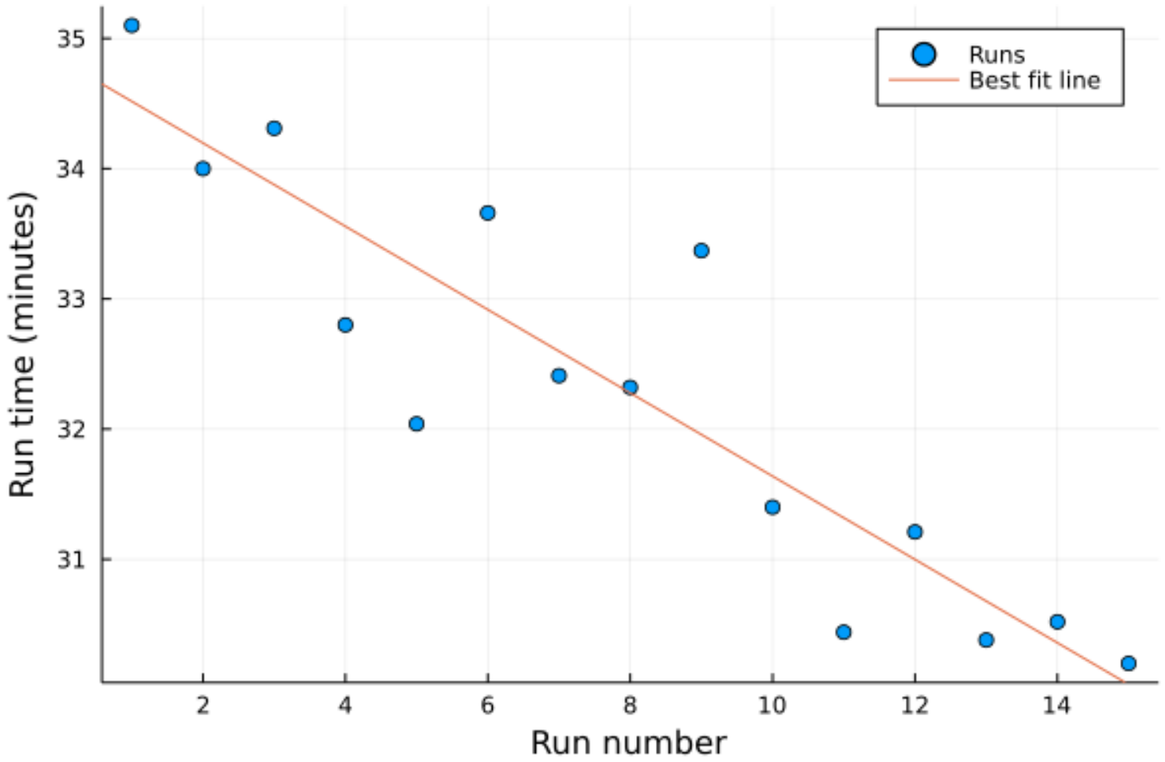
```
 0.071   0.142  0.214
 0.142   0.285  0.428
 0.214   0.428  0.642
```

# Vectorized operations

```
# Gradient and intercept
m = -0.32
c = 34.8

# Next run numbers
x = [16, 17, 18, 19, 20]

# Predict next run time
y = m * x + c
```



# Vectorized operations

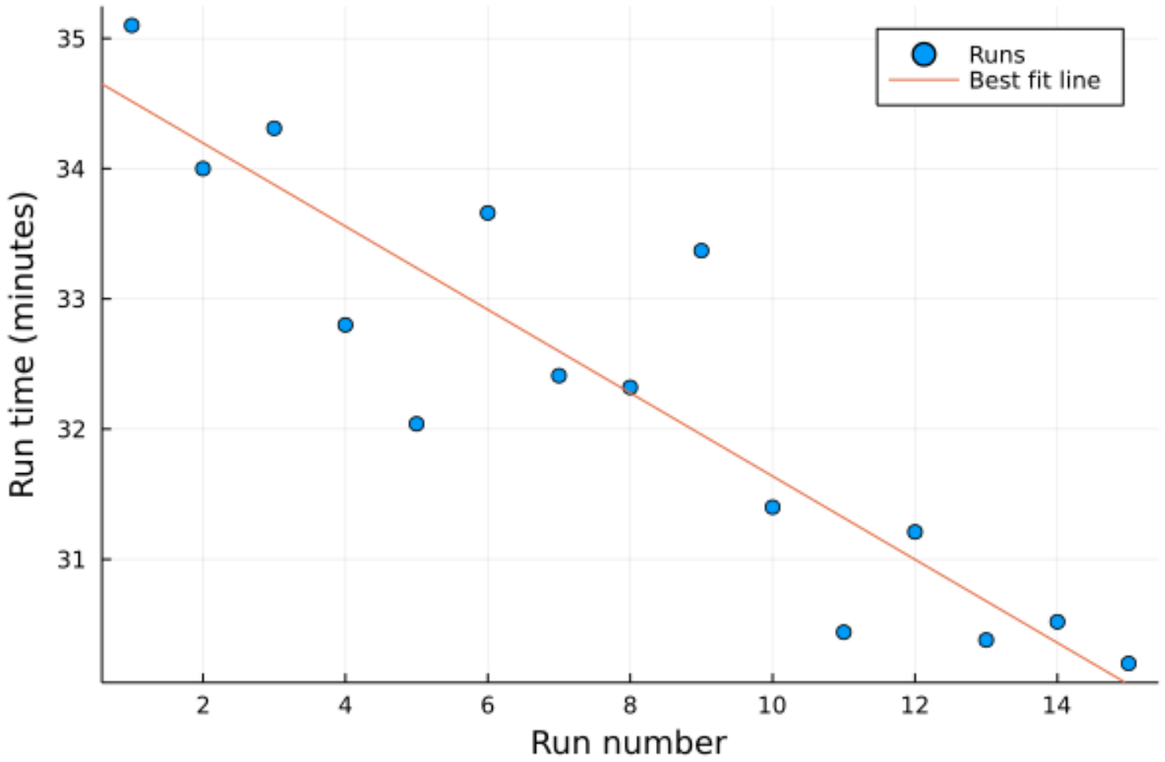
```
# Gradient and intercept
m = -0.32
c = 34.8

# Next run numbers
x = [16, 17, 18, 19, 20]

# Predict next run time
y = m .* x .+ c

println(y)
```

```
[29.68, 29.36, 29.04, 28.72, 28.40]
```



# Cheatsheet

For arrays `a` and `b`

| Operation | Scalar example | Array example |
| --- | --- | --- |
| Addition | `a .+ 1` | `a .+ b` or `a + b` |
| Subtraction | `a .- 1` | `a .- b` or `a - b` |
| Multiplication | `2 .* a` or `2 * a` | `a .* b` |
| Division | `a ./ 2` or `a / 2` | `a ./ b` |

# Let's practice!

INTRODUCTION TO JULIA