

# CT5102: Programming for Data Analytics

## Week 1: Introduction to R and Vectors

<https://github.com/JimDuggan/CT5102>

Dr. Jim Duggan,  
Information Technology,  
School of Engineering & Informatics

# Overview

- R's *mission* is to enable the best and most thorough exploration of data possible (Chambers 2008).
- It is a dialect of the S language, developed at Bell Laboratories
- ACM noted that S “*will forever alter the way people analyze, visualize, and manipulate data*”

---

## ACM HONORS DR. JOHN M. CHAMBERS OF BELL LABS WITH THE 1998 ACM SOFTWARE SYSTEM AWARD FOR CREATING "S SYSTEM" SOFTWARE

New York, March 23, 1999...The Association for Computing Machinery (ACM) today named Dr. John M. Chambers of Bell Labs as the recipient of the 1998 Software System Award for developing the S System, an innovative software program that helps users to manage and extract useful information from data.

The ACM's citation notes that Dr. Chambers' work "will forever alter the way people analyze, visualize, and manipulate data . . . S is an elegant, widely accepted, and enduring software system, with conceptual integrity, thanks to the insight, taste, and effort of John Chambers."

The System Software Award recognizes those who develop software systems having a lasting influence. It will be presented on May 15, 1999 during a special ACM awards banquet in New York City, and will be accompanied by a \$10,000 prize. Financial support is provided by IBM.

### About The "S System"

The first versions of S in the 1970s pioneered the use of data visualization and interactive statistical computing. Subsequent versions provided richly enhanced modeling capability, and user extensibility, based on its functional object-based approach.

Still more recent versions provide a powerful class/method structure, new techniques to deal with large objects, extended interfaces to other languages and files, object-based documentation compatible with HTML, and powerful interactive programming techniques. The commercial version, S-Plus, is used across many disciplines where analysts must struggle with creative ways to manage and extract useful information from data. More information about S is available at <http://cm.bell-labs.com/stat/S>.

<http://www.acm.org/announcements/ss99.html>

# Open Source Software

<https://www.r-project.org>

---



[\[Home\]](#)

## Download

[CRAN](#)

## R Project

[About R](#)

[Contributors](#)

[What's New?](#)

[Mailing Lists](#)

[Bug Tracking](#)

[Conferences](#)

[Search](#)

## R Foundation

[Foundation](#)

[Board](#)

[Members](#)

## The R Project for Statistical Computing

### Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

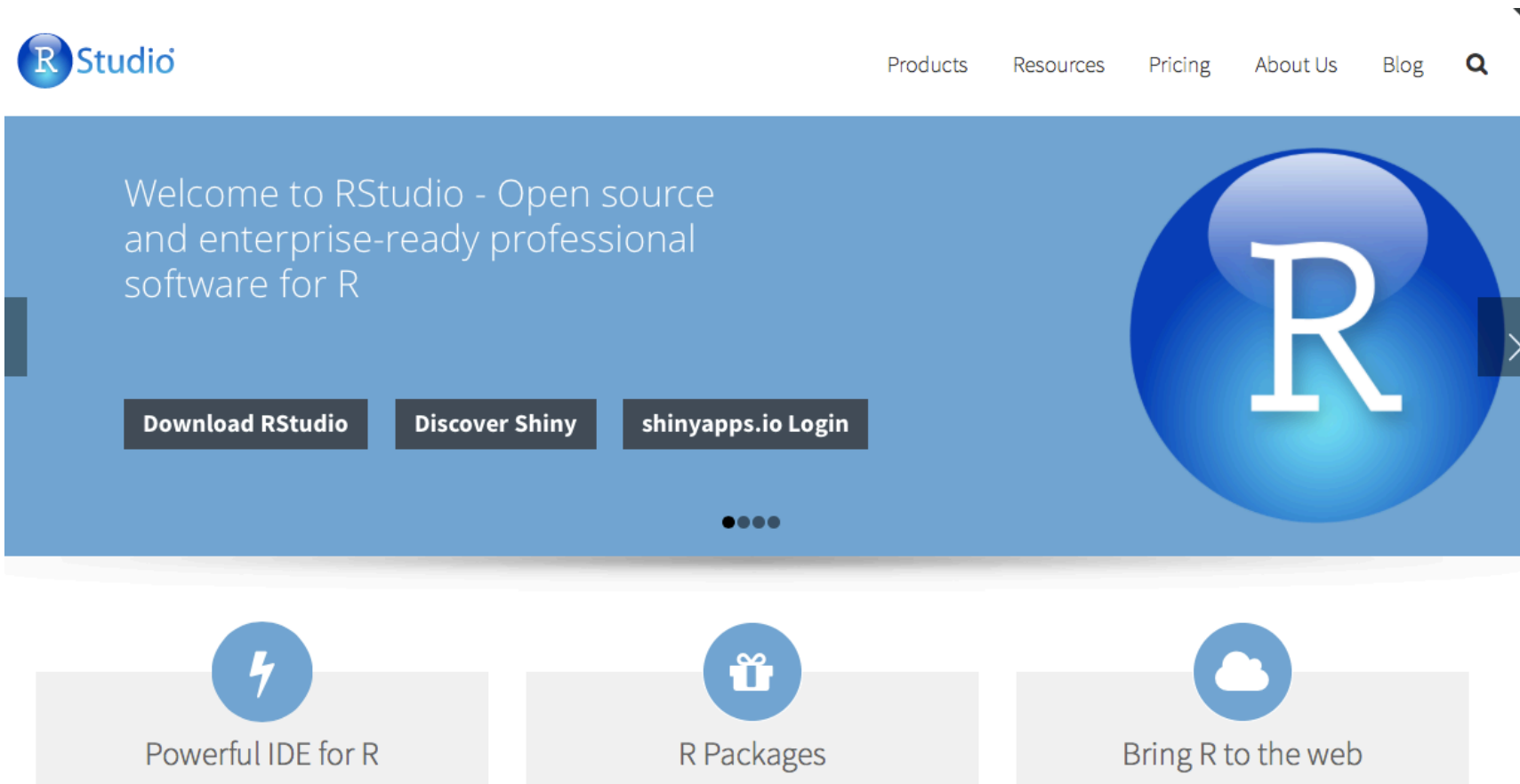
If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

### News

- [R version 3.2.2 \(Fire Safety\)](#) has been released on 2015-08-14.
- [The R Journal Volume 7/1](#) is available.
- [R version 3.1.3 \(Smooth Sidewalk\)](#) has been released on 2015-03-09.
- [useR! 2015](#), will take place at the University of Aalborg, Denmark, June 30 - July 3, 2015.
- [useR! 2014](#), took place at the University of California, Los Angeles, USA June 30 - July 3, 2014.

# R Studio for an IDE

<https://www.rstudio.com>



# Data Science Workbench

R Code

Environment/State

The screenshot displays the RStudio Data Science Workbench interface, divided into four main panels. A red border outlines the entire workspace. Blue dashed arrows point from the labels to their respective panels.

- R Code:** The top-left panel shows the script editor with the file "01 Introduction.R". The code includes:

```
83 s4<-sample(1:3,20,replace=TRUE)
84 s2=s4
85
86 # Gather data on the frequency of (whole) numbers in a vector
87 t1<-table(s1)
88 props<-prop.table(t1)
89
90
91
92
93
94
95
96
90:1 (Top Level) R Script
```
- Environment/State:** The top-right panel shows the "Global Environment" with a list of objects. The "Values" column displays the data types and values for each object.

Object	Value
b1	logi [1:2] FALSE TRUE
b2	logi [1:5] FALSE TRUE FALSE TRUE FALSE
c1	chr [1:5] "Odd" "Even" "Odd" "Even" "Odd"
ind	2L
index	5L
props	table [1:3(1d)] 0.4 0.3 0.3
r	24
s	num [1:101] 51.4 55 57.2 57.3 58.6 ...
s1	int [1:20] 2 1 1 3 1 3 2 1 1 1 ...
s2	int [1:20] 2 3 2 2 2 2 1 2 2 1 ...
- Interactive console:** The bottom-left panel shows the command history and output of the executed code.

```
> s2=s4
[1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
> t1<-table(s1)
> props<-prop.table(t1)
> props
s1
 1  2  3
0.4 0.3 0.3
>
>
```
- File System:** The bottom-right panel shows the file explorer with the path "Home > Desktop > GitHub > CT5102 > 01 Vectors". It lists the files in the current directory:

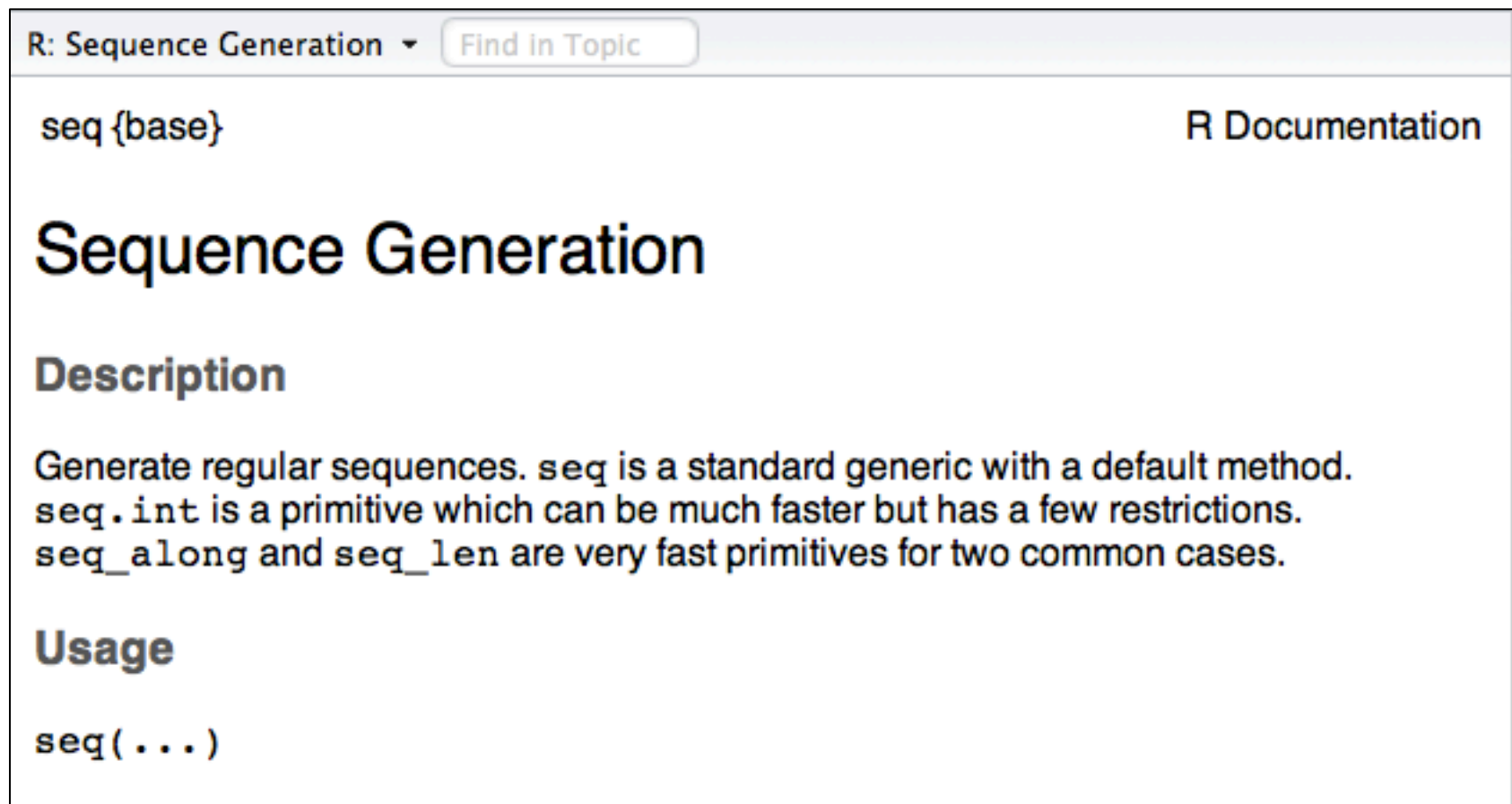
Name	Size	Modified
..		
01 Introduction.R	1.9 KB	Sep 3, 2015, 2:23 PM
01 Vectors.pdf	892 KB	Sep 3, 2015, 2:26 PM

Interactive console

File System

# Finding Help

- `?seq`



The screenshot shows the R documentation window for the `seq` function. The title bar at the top reads "R: Sequence Generation" with a dropdown arrow and a "Find in Topic" button. The main content area has a header "seq {base}" on the left and "R Documentation" on the right. Below this is the title "Sequence Generation" in a large font. The section "Description" follows, containing the text: "Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases." The "Usage" section is next, showing the function signature `seq(...)`.

R: Sequence Generation ▾ Find in Topic

seq {base} R Documentation

## Sequence Generation

### Description

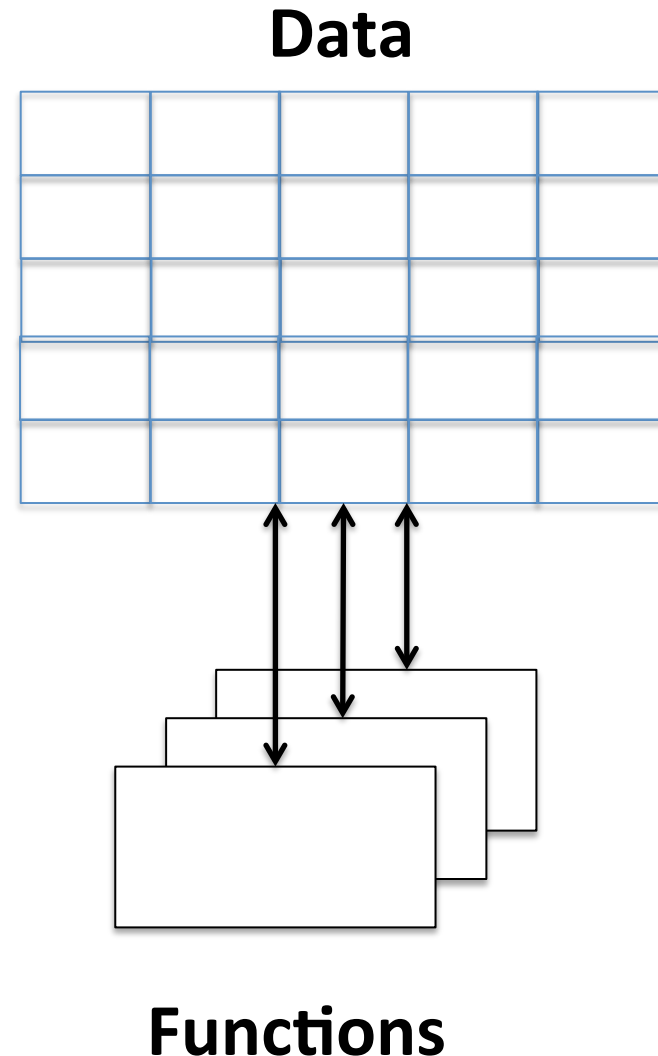
Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases.

### Usage

```
seq(...)
```

# Functional Programming

- R is a functional programming language, where software programs are organized into *functions* that can be called to transform data.
- Users of R should adopt the habit of creating simple functions which will make their work more effective and also more trustworthy (Chambers 2008).



# Data Structures - Vector

- The fundamental data type in R is the vector,
- A variable that contains a sequence of elements that have the same data type (Matloff 2009).
- Create using  $c(e_1, \dots, e_n)$
- Assignment statement  $\leftarrow$

**v1**

1
4
9
16
25

```
> v1<-c(1,4,9,16,25)
> v1
[1] 1 4 9 16 25
```



# Index

- The concept of an index is powerful in R, as it allows access to individual data elements of a vector, using the square brackets notation.
- In R, unlike programming languages such as C and Java, the index for a vector starts at 1.

```
> v1  
[1] 1 4 9 16 25  
  
> v1[1]  
[1] 1  
  
> v1[2]  
[1] 4  
  
> v1[5]  
[1] 25
```

# Creating sequences

- The colon operator (:) generates regular sequences within a specified range.

```
> v1<-1:10  
> v1  
[1] 1 2 3 4 5 6 7 8 9 10  
  
> v2<-3:13  
> v2  
[1] 3 4 5 6 7 8 9 10 11 12 13
```

# Using **seq()** function

- Useful function to create sequences, and allows for further detail via the **by** argument

```
> v6<-seq(1,5)
> v6
[1] 1 2 3 4 5

> v7<-seq(1,5,by=.5)
> v7
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

> v8<-seq(1,100,by=20)
> v8
[1] 1 21 41 61 81
```

# Sequences as indices

- Sequences can be used as indices for vectors, with the minus sign used for exclusion

```
> v1  
[1] 1 4 9 16 25
```

```
> v1[1:2]  
[1] 1 4
```

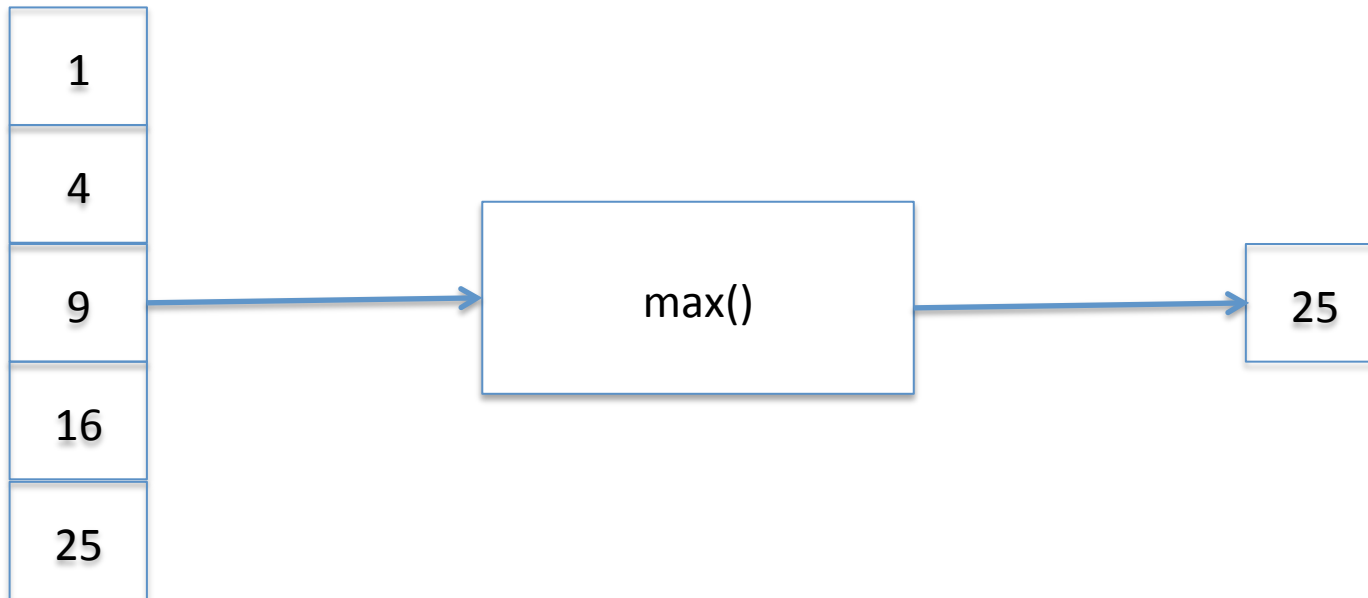
```
> v1[-1]  
[1] 4 9 16 25
```

```
> v1[-(1:3)]  
[1] 16 25
```

# max function

Input Vector

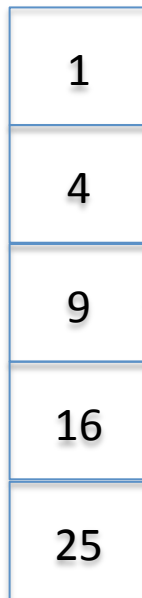
Output Vector



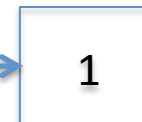
```
> v1  
[1] 1 4 9 16 25  
> max(v1)  
[1] 25
```

# min function

Input Vector



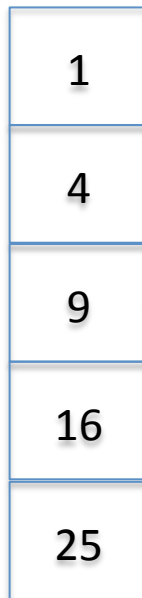
Output Vector



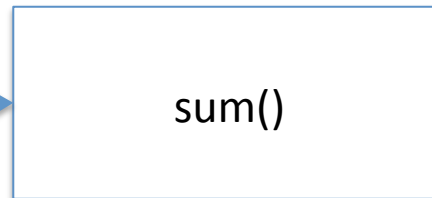
```
> v1  
[1] 1 4 9 16 25  
> min(v1)  
[1] 1
```

# sum function

Input Vector



Output Vector



```
> v1  
[1] 1 4 9 16 25  
> sum(v1)  
[1] 55
```

# Challenge 1.1

- Create an R vector of squares of 1 to 10
- Find the minimum
- Find the maximum
- Calculate the range
- Repeat the above using the **range()** function



# Modes

R variable types are known as *modes*, and these can include integer, numeric, character, and logical types. The mode of a variable can be examined using the **mode(x)** function call.

```
> v1<-c(1,2,3)
> mode(v1)
[1] "numeric"
```

```
> v2<-c(TRUE,FALSE)
> mode(v2)
[1] "logical"
```

```
> v3<-c("CT5102","CT561")
> mode(v3)
[1] "character"
```

# Displaying the object structure

- `str()` function
- Compactly display the internal structure of an R object

```
> v1<-c(1,2,3)
```

```
> str(v1)  
num [1:3] 1 2 3
```

```
> v2<-c(TRUE,FALSE)
```

```
> str(v2)  
logi [1:2] TRUE FALSE
```

```
> v3<-c("CT5102","CT561")
```

```
> str(v3)  
chr [1:2] "CT5102" "CT561"
```

# Arithmetic Operator

- Arithmetic operations can also be applied to vectors in an element-wise manner

```
> v1  
[1] 1 2 3 4 5  
> v2<-3*v1  
  
> v2  
[1] 3 6 9 12 15  
  
> v3<-v1+v2  
> v3  
[1] 4 8 12 16 20
```

# Vectorization

- A powerful feature of R is that it supports *vectorization*
- Functions can operate on every element of a vector, and return the results of each individual operation in a new vector.

```
> v1  
[1] 1 2 3 4 5  
  
> r<-sqrt(v1)  
  
> r  
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

# Key Idea

**Input Vector**

1
4
9
16
25

**Output Vector**

1
2
3
4
5

**sqrt()**

```
graph LR; Input[Input Vector] --> sqrt[sqrt()]; sqrt --> Output[Output Vector]
```

The diagram illustrates a data transformation process. An input vector containing the values [1, 4, 9, 16, 25] is passed through a function labeled 'sqrt()'. The output vector, resulting from the square root operation, contains the values [1, 2, 3, 4, 5]. Arrows indicate the flow of data from the input vector to the function and then to the output vector.

# Conditional Expressions on Vectors

- Conditional expressions can be applied to vectors, and this operation returns a boolean vector

```
> v1  
[1] 1 4 9 16 25
```

```
> v1 %% 2==0  
[1] FALSE TRUE FALSE TRUE FALSE
```

# Boolean vectors used as indices

- A target the vector can be filtered by the TRUE locations in the boolean vector.

```
> v1
[1] 1 4 9 16 25

> b1<-v1 %% 2 == 1

> b1
[1] TRUE FALSE TRUE FALSE TRUE

> v1[b1]
[1] 1 9 25
```

# 1 minute Quiz

- What will the following return?

```
> v1  
[1] 1 4 9 16 25  
  
> b1<-c(FALSE,TRUE)  
  
> v1[b1]
```



# which()

- Give the TRUE indices of a logical object, allowing for array indices.

```
> v1
[1] 1 4 9 16 25
> ind<-which(v1==4)
> ind
[1] 2
> v1[ind]
[1] 4
```

# Challenge 1.2

- Write an R script to find range of a vector.
- Write 2 “parallel” vectors that store a course code in one, and the number of students in the other. Write a script that returns the number of students in a course.

# Naming vector elements

- The elements of a vector can also be given names, and this can be useful in defining parameters for further analysis. The name can be used as an index.

```
> names(v1)<-  
c("Var1","Var2","Var3","Var4","Var5")  
  
> v1  
Var1 Var2 Var3 Var4 Var5  
  1   4   9  16  25  
  
> v1["Var5"]  
Var5  
 25
```

# Vectorized if/else

- Vectors can also be processed using the vectorized **ifelse(b,u,v)** function, which accepts a boolean vector **b** and allocates the element-wise results to be either **u** or **v**.

```
> v1
[1] 1 4 9 16 25

> c1<-ifelse(v1%%2==0,"Even","Odd")

> c1
[1] "Odd" "Even" "Odd" "Even" "Odd"
```

# sample() function

- Sample takes a sample of the specified size from the elements of x using either with or without replacement.

```
> s1<-sample(1:3,20,replace=TRUE)
> s1
[1] 3 3 2 3 3 3 1 2 2 2 3 3 1 3 2 3 2 3 2 3

> s2<-sample(1:42,6)
> s2
[1] 29 21 37 17 11 32
```

# Replicating results from sample()

- Use **set.seed(n)** function

```
> sample(1:3,20,replace=TRUE)
[1] 2 3 2 2 2 2 1 2 2 1 2 2 1 1 1 2 1 3 1 2
> sample(1:3,20,replace=TRUE)
[1] 2 1 2 2 3 1 2 1 3 2 1 2 2 2 2 3 1 3 2 3
```

```
> set.seed(111)
> sample(1:3,20,replace=TRUE)
[1] 2 3 2 2 2 2 1 2 2 1 2 2 1 1 1 2 1 3 1 2
> sample(1:3,20,replace=TRUE)
[1] 2 1 2 2 3 1 2 1 3 2 1 2 2 2 2 3 1 3 2 3
> set.seed(111)
> sample(1:3,20,replace=TRUE)
[1] 2 3 2 2 2 2 1 2 2 1 2 2 1 1 1 2 1 3 1 2
```

# table() function

- Can be used to perform counts on data (factors) in a vector. Works over many dimensions (e.g. data frame).

```
> s1  
[1] 3 3 2 3 3 3 1 2 2 2 3 3 1 3 2 3 2 3 2 3  
  
> t1<-table(s1)  
  
> t1  
s1  
 1  2  3  
2  7 11  
> names(t1)  
[1] "1" "2" "3"
```

```
> prop.table(t1)  
s1  
 1  2  3  
0.10 0.35 0.55
```

# Finding frequency counts

- The name can be used directly, or the which statement can find the position in the vector using %in%

```
> t1["3"]  
3  
11  
> t1[which(names(t1) %in% "3")]  
3  
11
```

```
> which(c(1,2) %in% c(2,3,4,1))  
[1] 1 2
```



# References

- Chambers, J. 2008. Software for data analysis: programming with R. Springer Science & Business Media. Chicago.
- Chang, W. 2013. R Graphics Cookbook. "O'Reilly Media, Inc." Sebastapol, CA.
- Matloff, N. 2009. The Art of R Programming. No Starch Press, San Francisco, CA.