Lesson 2: Computing foundation: from set theory language to R

Computer Science II
Mrs. Staffen

Computing Foundation

Foundation of computing:

- Computers (hardware) can do only very simple things (e.g., add two numbers)
- To use computers to solve a problem, we have to communicate the knowledge for solving the problem to the computer using a *precise* language.

We learned a *precise* language -- *set theory language* -- which offers *sets*, *tuples*, and *functions* for us to represent knowledge underlying a problem.

The set theory language could be directly used to convey knowledge to a computer. But we do not have a software to support that yet. The set theory language does form the basis of many programming languages. One of them is *R*.

Computing Foundation

We introduce R, following key ideas in set theory language

- R allows the use of
  - some standard *sets*, called *primitive types*,
  - ways of writing *elements* of those sets, and
  - *operations* (i.e., functions) over those sets (types)
- R allows the use of *tuples* through the constructs such as *vectors*.
- R allows the specification of (content) of *functions* and *use* of functions.

Computing Foundation

1 Primitive Types, Their Elements and Functions

2 Vector Type and c Function

3 R Program Variables

4 Vectors, and Functions on Vectors

5 Functions

1 Primitive Types

Primitive types -- each primitive type is a set we use often to model the world

- *logical* type
- *integer* type
- *double* type
- *complex* type
- *character* or *string* type
- *NULL* (a single special value, doesn’t belong to any type)

1.1 Integer Type, Its Elements and Functions

Recall that an integer is a whole number with **no** decimal or fractional component.

Which of the following are integers?

5, 27, ½, -3, 1002, 3.145, 67

1.1 Integer Type, Its Elements and Functions

Recall that an integer is a whole number with **no** decimal or fractional component.

Which of the following are integers?

5, **27**, ½, **-3**, **1002**, 3.145, **67**

1.1 Integer Type, Its Elements and Functions

In R, the type **integer** includes integers we use often but not all integers:

<u>integer</u> type: the set of 32-bit signed integer, i.e.,

{*x*: *x* >= -2147483648 and *x* <= 2147483647}

Note we use the set builder notation to precisely define what an integer type is.

How to read (or what is the meaning of) the set builder notation here?

1.1 Integer Type, Its Elements and Functions

In R, the type **integer** includes integers we use often but not all integers:

<u>**integer**</u> type: the set of 32-bit signed integer, i.e.,

$$\{x: x >= -2147483648 \text{ and } x <= 2147483647\}$$

Note we use the set builder notation to precisely define what an integer type is.

How to read (or what is the meaning of) the set builder notation here?

The set of every x that is at least -2147483648 and at most 2147483647

1.1 Integer Type, Its Elements and Functions

In R, the type **integer** includes integers we use often but not all integers:

integer type: the set of 32-bit signed integer, i.e.,

$$\{x: x >= -2147483648 \text{ and } x <= 2147483647\}$$

To represent a number in *integer type*, we have to use a special syntax:

*number***L**

where *number* is a whole number. Note there is no space between *number* and **L**

For example: 5L is a number in *integer type*.

1.1 Integer Type, Its Elements and Functions

Practice. Let's try to write some familiar *expressions* in R and see how *R* tells us the values of these expressions.

Start *R* console

- Logon to repl.it.
- In the console, type R and then press enter to start the *R console*. (The R console is software that can understand *R expressions*, i.e., sentences in the R language)
- Now you are in the *R* console

1.1 Integer Type, Its Elements and Functions

Practice. *R* is able to recognize normal integers (within integer type) and standard arithmetic expressions using functions such as +, -, *, / etc..

In the R console

- Type an integer (followed by L) and hit enter, the R console will print the value of the integer.
- Using *integer*, write *R expressions* for each of the following arithmetic expressions:

  1 - (1 -1)

  1 - 1 - 1

  2 - 2 * 2

1.2 Double Type, Its Elements and Functions

Start *R* console at repl.it

- Some numbers such as ⅓ can not be represented as exact double numbers
  - Type *R expression* 1/3, what does console print?
  - Write an *R expression* for multiplying the decimal number just printed by the console by 3. What is the value of the expression by *R* console?
  - What is the value of the *R expression* (1/3)*3? Compare your answer to the answer of the R console.

1.3 Complex Type, Its Elements and Functions

- **complex** type: the set of some *complex* numbers. There are also functions on complex numbers. We skip this topic.

1.4 Character Type, Its Elements and Functions

How do we represent words such as names of a person and address of a home besides numbers?

character or **string** type: the set of strings over some alphabet (e.g. English alphabet).

Syntax for a string (an *element* of string type):

"*stringContent*"

where *stringContent* normally can be any phrases. For example: `"Elon Musk"`

1.4 Character Type, Its Elements and Functions

Syntax for a string (an *element* of string type):

"*stringContent*"

Why do we add quotes to represent a string?

We use normal words (without quotes) for *variables* (which will be discussed later). To distinguish a string from a variable, we use quotes.

1.4 Character Type, Its Elements and Functions

Practice: Elements of Character/String type

- Write each of the following as a string in the R console
  - Lubbock Cooper High School
  - I hear and I forget. I see and I remember. I do and I understand -- Confucius

1.5 logical Type, Its Elements and Functions

Recall that we needed to use truth values, *true* or *false*, to indicate if a city is big or not. These are the values for *logical type* in R.

logical type: the set {TRUE, FALSE}. Elements of this type, TRUE and FALSE, are called **logical** or **truth values.** The values are case sensitive. `True` is NOT a logical value.

1.5 logical Type, Its Elements and Functions

Some familiar functions (operations) output logical values

- Arithmetical relations on *integer types* and *double types*: comparisons of elements of *string* type: ==, >=, <=, >, <
- String comparison operations: ==, >= (dictionary order), <=, >, <

Functions (logical connectives: and, or, not) over logical values

1.5 logical Type, Its Elements and Functions

Practice. Arithmetical relations on *integer types* and *double types* ==, >=, <=, >, < are binary functions whose value is a logical value

What are the *values* of the following R expressions? Check using the R console.

1 == 3

3 == 3

3 > 3

3 >= 3

3 < 3

3 <= 3

1.5 logical Type, Its Elements and Functions

Practice. Comparisons of elements of *string* type: ==, >=, <=, >, <

What are the *values* of the following R expressions? Check using the R console.

```
“a”  <  “a”
“a”  >=  “ab”
“a”  ==  “A”
```

1.5 logical Type, Its Elements and Functions

Logical Connectives: **&&** (and), **||** (or), **!** (not) [see comments in notes]

Recall that we need logical connectives for precise meaning:

a big US city means an *x* such that *x* is a US city and *x* is big:

big(*x*) =true **and** $x \in$ *citiesOfUSA*

1.5 logical Type, Its Elements and Functions

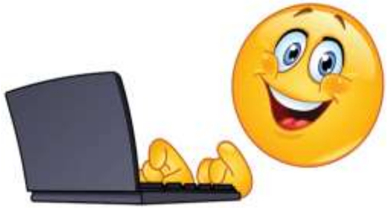
Syntax *e1* `&&` *e2*

where *e1* and *e2* are R expressions whose value are logical values.

It is **read** as *e1* and *e2*. The *value* of the expression is TRUE if **both** the values of *e1* and *e2* are TRUE. Otherwise, it is FALSE.

What is the value of each of the following R expressions? Check them using R console.

```
TRUE && FALSE
TRUE && TRUE
(2 <= 5) && (2 >= 1)
(2 <= 5) && (2 > 5)
```



1.5 logical Type, Its Elements and Functions

Syntax *e1* `&&` *e2*

where *e1* and *e2* are R expressions whose value are logical values.

It is **read** as *e1* and *e2*. The *value* of the expression is TRUE if **both** the values of *e1* and *e2* are TRUE. Otherwise, it is FALSE.

What is the value of each of the following R expressions? Check them using R console.

```
TRUE && FALSE (FALSE)
TRUE && TRUE (TRUE)
(2 <= 5) && (2 >= 1) (TRUE)
(2 <= 5) && (2 > 5) (FALSE)
```

1.5 logical Type, Its Elements and Functions

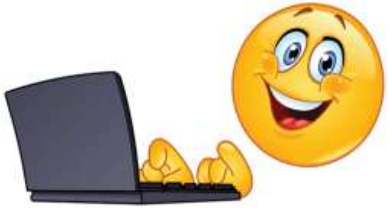
Syntax *e1* || *e2*

where *e1* and *e2* are R expressions whose value are logical values.

It is **read** as *e1* or *e2*. The *value* of the expression is TRUE if one of the values of *e1* and *e2* is TRUE. Otherwise, it is FALSE.

What is the value of each of the following R expressions? Check them using R console.

```
TRUE || FALSE
FALSE || TRUE
(2 <= 5) || (2 >= 1)
(2 <= 5) || (2 > 3)
```



1.5 logical Type, Its Elements and Functions

Syntax *e1* **||** *e2*

where *e1* and *e2* are R expressions whose value are logical values.

It is **read** as *e1* or *e2*. The *value* of the expression is TRUE if one of the values of *e1* and *e2* is TRUE. Otherwise, it is FALSE.

What is the value of each of the following R expressions? Check them using R console.

```
TRUE || FALSE    (TRUE)
FALSE || TRUE    (TRUE)
(2 <= 5) || (2 >= 1) (TRUE)
(2 <= 5) || (2 > 3)  (TRUE)
```

1.5 logical Type, Its Elements and Functions

Syntax !*e1*

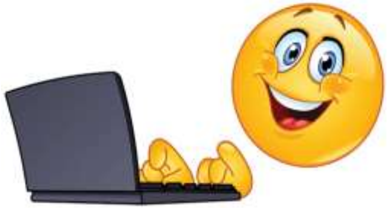
where *e1* is an R expressions whose value are logical values.

It is **read** as the negation of *e1* or not *e1*.

The *value* of the expression is TRUE if the value of *e1* is FALSE. It is FALSE if that of *e1* is TRUE.

What is the value of each of the following R expressions? Check them using R console.

```
!TRUE
!FALSE
!(2 <= 5)
!(2 > 3)
!((2 > 3) && (2 <= 5))
```



1.5 logical Type, Its Elements and Functions

Syntax ! *e1*

where *e1* is an R expressions whose value are logical values.

It is **read** as the negation of *e1*. The *value* of the expression is TRUE if the value of *e1* is FALSE. It is FALSE if that of *e1* is TRUE.

What is the value of each of the following R expressions? Check them using R console.

```
!TRUE   (FALSE)
!FALSE   (TRUE)
!(2 <= 5)  (FALSE)
!(2 > 3)  (TRUE)
!((2 > 3) && (2 <= 5))   (FALSE)
```

1.5 logical Type, Its Elements and Functions

Syntax

`NULL`

It is a special value referring to an empty tuple (). It can be used in several cases, but we will skip it for now.

1.6 Primitive Types, Their Elements and Functions - Overloaded Operators

We will talk about an interesting phenomenon where we sometimes use the same function name to represent different functions! Recall that a function is more than a function name. What is a *function signature*?

1.6 Primitive Types, Their Elements and Functions - Overloaded Operators

We will talk about an interesting phenomenon where we sometimes use the same function name to represent different functions! Recall that a function is more than a function name. What is a *function signature*?

$$functionName\text{: } set1 \text{ -> } set2$$

or more generally

$$functionName\text{: } set1, set2, \ldots, set_n \text{-> } set_{n+1}$$

1.6 Primitive Types, Their Elements and Functions - Overloaded Operators

Consider the arithmetic symbols and R symbols

+, -, *, /, %, ==, >=, <=, &&, ||, <-, ->, …

Are they functions or simply function names?

1.6 Primitive Types, Their Elements and Functions - Overloaded Operators

Consider the arithmetic symbols and R symbols

+, -, *, /, %, ==, >=, <=, &&, ||, <-, ->, …

Are they functions or simply function names?

Function names! However, we may get used to treating them as equivalent to functions, but that is not always correct. *One function name can be used for several function signatures*. When this happens, the function name is **<u>overloaded</u>**.

1.6 Primitive Types, Their Elements and Functions - Overloaded Operators

For example, + is an *overloaded* operator

- +: integer $\rightarrow$ integer

  name: +, domain: integer type, range: integer type

- +: double $\rightarrow$ double

  name: +, domain: double, range: double

These are actually two different functions! When + is used, R decides which function to use in terms of the types of the operands of +.

1.6 Primitive Types, Their Elements and Functions - Overloaded Operators

Identify which function signature is used in the following expressions:

2 + 3

4 + 22

2.3 + 4.8

3.9 + 1

1.6 Primitive Types, Their Elements and Functions - Overloaded Operators

Identify which function signature is used in the following expressions:

2 + 3	+ for integer
4 + 22	+ for integer
2.3 + 4.8	+ for double
3.9 + 1	+ for double

1.6 Primitive Types, Their Elements and Functions - Overload Operators

The example of + may seem insignificant here.

We have *overloaded* it to add a tuple and a scalar, and to add a tuple and a tuple.

Give an example of each of the three cases: add two numbers, add a tuple and a scalar, and add a tuple and a tuple:

1.6 Primitive Types, Their Elements and Functions - Overload Operators

The example of + may seem insignificant here.

We have *overloaded* it to add a tuple and a scalar, and to add a tuple and a tuple.

Give an example of each of the three cases: add two numbers, add a tuple and a scalar, and add a tuple and a tuple:

- 1 + 1 value is 2
- (1, 1) + 1 value is (2, 2)
- (1, 1) + (1, 2) value is (2, 3)

Computing Foundation

1 Primitive Types, Their Elements and Functions

2 Vector Type and c Function

3 R Program Variables

4 Vectors, and Functions on Vectors

5 Functions

2.1 Vector Type -- Motivation

We know that tuples can be used to represent complex objects such as coordinate points and even a list of students' scores.

In this part we will talk about the ways that R provides to represent tuples.

The first type of tuples in R we show is *vectors*.

2.2 Vector Type -- Formal Definition

A **vector** is a tuple whose components are of the same **primitive type** which must be one of the following

- *integer type*
- *double type*
- *string type*
- *logical type*
- *complex type* (we will skip)
- *raw* (we will skip)

2.2 Vector Type -- Formal Definition

A **vector** is a tuple whose components are all of the same **primitive type** which must be one of the following

- *integer type*
- *double type*
- *string type*
- *logical type*
- *complex type* (we will skip)
- *raw* (we will skip)

Questions:

- Is (4, 5) a vector?

2.2 Vector Type -- Formal Definition

A **vector** is a tuple whose components are all of the same **primitive type** which is one of the following

- *integer type*
- *double type*
- *string type*
- *logical type*
- *complex type* (we skip here)
- *raw* (we skip here)

Questions:

- Is (4, 5) a vector? (Yes, because 4 and 5 are both integer type)
- Is (“Elon Musk”, “Tesla”) a vector?

2.2 Vector Type -- Formal Definition

A **vector** is a tuple whose components are all of the same **primitive type** which is one of the following

- *integer type*
- *double type*
- *string type*
- *logical type*
- *complex type* (we skip here)
- *raw* (we skip here)

Questions:

- Is (4, 5) a vector? (Yes, because 4 and 5 are both integer type)
- Is (“Elon Musk”, “Tesla”) a vector? (Yes, because “Elon Musk”, “Tesla” are both string type)
- Is (“Peter”, 16, 65,150) a vector?

2.2 Vector Type -- Formal Definition

A **vector** is a tuple whose components are all of the same **primitive type** which is one of the following

- *integer type*
- *double type*
- *string type*
- *logical type*
- *complex type* (we skip here)
- *raw* (we skip here)

Questions:

- Is (4, 5) a vector? (Yes, because 4 and 5 are both integer type)
- Is (“Elon Musk”, “Tesla”) a vector? (Yes, because “Elon Musk”, “Tesla” are both string type)
- Is (“Peter”, 16, 65,150) a vector? (No. Components have different types)

2.3 Vector creation function -- c Function

Functions are a key concept in set theory language we just learned. They are at the core of computing. We use not only the arithmetic functions you are familiar with in your subjects, but also *general* functions as we introduced earlier in the set theory language.

R, provides no construct to directly write a vector (e.g., in the form of tuples). It does provide a function `c(...)` whose output is a vector and inputs are the components of the vector.

2.3 c Function

Syntax:

$\mathbf{c}\,(component_1, \ldots\ldots, component_n)$

where $component_i$ (i : $1..n$) can be any expression whose *value* is of a primitive type, and the values of all *components* are of the same primitive type.

The output of the function $\mathbf{c}\,(component_1, \ldots\ldots, component_n)$ is the vector

$$(component_1, \ldots\ldots, component_n)$$

Example: To represent the vector (*5*, *6*), we use *R expression*: c(5, 6)

2.3 c Function

Syntax: **c** (*component*$_1$, …… , *component*$_n$) where *component*$_i$ (i : $1..n$) can be any expression whose *value* is of a primitive type, and the values of all *components* are of the same primitive type.

Example:

- Read the definition carefully and tell if *c(5+3, 6+3)* is a valid *R expression*.

2.3 c Function

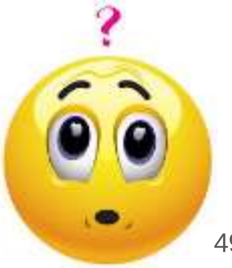
Syntax: **c** (*component*$_1$, …… , *component*$_n$) where *component*$_i$ (i : $1..n$) can be any expression whose *value* is of a primitive type, and the values of all *components* are of the same primitive type.

Example:

- Read the definition carefully and tell if c(5+3, 6+3) a valid R expression.

  (Yes. Because 5+3 is an expression whose value is 8, an integer type - a primitive type.)

- Is c("Peter", 160) a valid *R expression*? Why ?



2.3 c Function

Syntax: **c** **(***$component_1$, …… , $component_n$***)** where *$component_i$* (*i* : *1..n*) can be any expression whose *value* is of a primitive type, and the values of all *components* are of the same primitive type.

Example:

- Read the definition carefully and tell if c(5+3, 6+3) a valid R expression.

  (Yes. Because 5+3 is an expression whose value is 8, an integer type - a primitive type.)

- Is c*("Peter", 160)* a valid *R expression*? Why ?

  (No, because the values of the expressions are of different types: one is character and the other is an double.)

2.3 c Function

Syntax: **c** **(***component*$_1$, …… , *component*$_n$**)** where *component*$_i$ (i : $1..n$) can be any expression whose *value* is of a primitive type, and the values of all *components* are of the same primitive type.

Practice:

In a test Aaron got 90, Bill got 95, Cecilia got 100 and Dina got 88. Write an R expression to represent the grades of the students as a vector.

2.3 c Function

Syntax: **c** **(***component*$_1$, …… , *component*$_n$**)** where *component*$_i$ (i : $1..n$) can be any expression whose *value* is of a primitive type, and the values of all *components* are of the same primitive type.

Practice:

In a test Aaron got 90, Bill got 95, Cecilia got 100 and Dina got 88. Write an R expression to represent the grades of the students as a vector.

```
c(90,  95, 100, 88)
```

2.3 c Function

Example.

Write an R expression to represent the vector of things you feel interesting.

(You can write your favorite game names, or movies, or the scores your favorite players got in the last 5 games …)

2.3 c Function

Practice.

In R console,

```
> c(1,2)
[1] 1 2
```

After you write expression c(1,2) and hit enter in the R console, R will *evaluate* this expression (i.e., find its value) and print the value below.

[1] means that the following is a vector and 1 is the index of the first element.

The elements in the vector are printed in order.

2.4 Vector Type -- Unnamed and Named Vectors (Motivation)

We are able to represent a vector in R now. The way we write *R expressions* so far is to represent unnamed vectors.

We learned about named tuples. We also have the concept of named vectors, i.e., named tuples whose components are of the same primitive types. We will now learn how to represent named vectors in R.

2.4 Named Vectors -- Formal Definition

Named Vectors

Syntax. `c` ($name_1$ = $component_1$, …… , $name_n$ = $component_n$)

where $name_1$, …, $name_n$ are strings to represent the *named index* of the corresponding components, and *components* are defined as usual.

The output of `c` ($name_1$ = $component_1$, …… , $name_n$ = $component_n$) is

($component_1$,	$component_2$, …… ,		$component_n$)
$name_1$	$name_2$	…	$name_n$

Is the function named *c* overloaded? (i.e. same name but different output type)

2.4 Named Vectors -- Formal Definition

<u>Named Vectors</u>

Syntax. **c** **(**$name_1$ = $component_1$, …… , $name_n$ = $component_n$**)**

where $name_1$, …, $name_n$ are strings to represent the *named index* of the corresponding components, and *components* are defined as usual.

The output of **c** **(**$name_1$ = $component_1$, …… , $name_n$ = $component_n$**)** is

($component_1$,	$component_2$, …… ,		$component_n$)
$name_1$	$name_2$	…	*n*

Question: Write an R expression to represent the point (5, 6) using a named vector (with name indexes being *x-coordinate*, and *y-coordinate*)

2.4 Named Vectors -- Formal Definition

Named Vectors

Syntax. **c** ($name_1$ = $component_1$, …… , $name_n$ = $component_n$)

where $name_1$, …, $name_n$ are strings to represent the *named index* of the corresponding components, and *components* are defined as usual.

The output of **c** ($name_1$ = $component_1$, …… , $name_n$ = $component_n$) is

($component_1$,	$component_2$, …… ,		$component_n$)
$name_1$	$name_2$	…	$name_n$

Question: Write an R expression to represent the point (5, 6) using named vector (with name indexes being *x-coordinate*, and *y-coordinate*)

```
c( "xcoordinate" =5 , "ycoordinate" = 6)
```

2.4 Named Vectors

Practice.

In R console,

```
> c( "xcoordinate" =5 , "ycoordinate" = 6)
xcoordinate          ycoordinate
          5                    6
```

When R console prints the value of the expression, the named indices are put above the components. The components are not separated by "," and are not enclosed by parenthesis.

2.4 Named Vectors -- Examples

Named Vector

Syntax. `c(`$name_1$ = $component_1$, …… , $name_n$ = $component_n$`)`

The output of `c(`$name_1$ = $component_1$, …… , $name_n$ = $component_n$`)` `is`

`(component1,component2,` … `com`

$name_1$ $name_2$ …

Practice:

On a test, Aaron got 90, Bill got 95, Cecilia got 100 and Dina got 88. Using *named vectors*, write an R expression to represent the the students' scores.

2.4 Named Vectors -- Examples

Named Vector

Syntax. `c(`$name_1 = component_1, \ldots\ldots, name_n = component_n$`)`

The output of `c(`$name_1 = component_1, \ldots\ldots, name_n = component_n$`)` `is`

`(component1,component2,` ... `componentn`

$name_1$ $name_2$

$name_n$

Practice:

On a test, Aaron got 90, Bill got 95, Cecilia got 100 and Dina got 88. Using named vectors, write an R expression to represent the the students' scores.

```
c("Aaron" = 90,  "Bill" = 95, "Cecilia" = 100,  "Dina" = 88)
```

2.4 Named Vectors -- Examples

Example:

The following table displays data on several roller coasters that were opened in a recent year.

Roller coaster	Type	Height (ft)	Design	Speed (mph)	Duration (sec)
Wildfire	Wood	187.0	Sit down	70.2	120
Skyline	Steel	131.3	Inverted	50.0	90
Goliath	Wood	165.0	Sit down	72.0	105
Helix	Steel	134.5	Sit down	62.1	130
Banshee	Steel	167.0	Inverted	68.0	160
Black Hole	Steel	22.7	Sit down	25.5	75

Using named vectors, write an R expression to represent the function *Type* for the rollercoaster.

2.4 Named Vectors -- Examples

Practice:

The following table displays data on several roller coasters that were opened in a recent year.

Roller coaster	Type	Height (ft)	Design	Speed (mph)	Duration (sec)
Wildfire	Wood	187.0	Sit down	70.2	120
Skyline	Steel	131.3	Inverted	50.0	90
Goliath	Wood	165.0	Sit down	72.0	105
Helix	Steel	134.5	Sit down	62.1	130
Banshee	Steel	167.0	Inverted	68.0	160
Black Hole	Steel	22.7	Sit down	25.5	75

Using named vectors, write an R expression to represent the function *Type* for the rollercoaster.

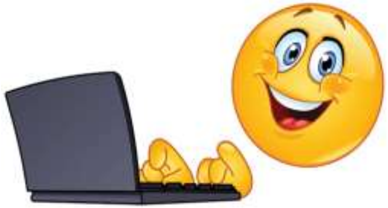
```
c( “Wildfire” = “Wood”, “Skyline” = “Steel”, “Goliath” =
“Wood”, “Helix” = “Steel”, “Banshee” = “Steel”, “BlackHole” =
“Steel”)
```

2.4 Named Vectors -- Examples

Practice:

The following table displays data on several roller coasters that were opened in a recent year.

Roller coaster	Type	Height (ft)	Design	Speed (mph)	Duration (sec)
Wildfire	Wood	187.0	Sit down	70.2	120
Skyline	Steel	131.3	Inverted	50.0	90
Goliath	Wood	165.0	Sit down	72.0	105
Helix	Steel	134.5	Sit down	62.1	130
Banshee	Steel	167.0	Inverted	68.0	160
Black Hole	Steel	22.7	Sit down	25.5	75



Using named vectors, write an R expression to represent the function *Height* for the rollercoaster.

2.4 Named Vectors -- Examples

Example:

The following table displays data on several roller coasters that were opened in a recent year.

Roller coaster	Type	Height (ft)	Design	Speed (mph)	Duration (sec)
Wildfire	Wood	187.0	Sit down	70.2	120
Skyline	Steel	131.3	Inverted	50.0	90
Goliath	Wood	165.0	Sit down	72.0	105
Helix	Steel	134.5	Sit down	62.1	130
Banshee	Steel	167.0	Inverted	68.0	160
Black Hole	Steel	22.7	Sit down	25.5	75

Using named vectors, write an R expression to represent the function *Height* for the rollercoaster.

```
c( "Wildfire" = 187.0, "Skyline" = 131.3, "Goliath" = 165.0,
"Helix" = 134.5, "Banshee" = 167.0, "BlackHole" = 22.7)
```

Computing Foundation

3.1 R Program Variables (Motivation)

Why do we use *R program variables*?

They are similar to the variables you used in math and other subjects. They are used to refer to things (so that our communication is simplified without repeating a lot of information).

They are also different from the variables you have learned in algebra or the variables we use in set builder notation. We won’t touch this difference for now.

3.1 R Program Variables

Syntax for writing an R program variable: a sequence of characters with some constraints such as

- The first character is not a number
- No character should be a space
- …

For example,

- *1x* is not a variable because first character is a number
- *first name* is not a variable because of the space in between the two words.

3.1 R Program Variables

Example:

Tell the type of each of the following R expressions

- TURE
- TRUE
- 1
- 1.2
- True
- “True”

3.1 R Program Variables

Example:

Tell the type of each of the following R expressions

- TURE - a variable
- TRUE - a logical value
- 1 - an integer
- 1.2 - a double
- True - a variable
- “True” - a string

Now we see why a string has to use quote (so that distinguishable from variables)

3.2 R Program Variables and Assignment (Motivation)

The main purpose of a variable is to refer to something.

What is the syntax for this “refer to”? R uses assignment = or <-. With this syntax, we can have, for example, *x = 3* is read as *x* refers to 3 or 3 is *assigned* to the variable *x*.

Question:

- Is = in R the same as *equals*?

3.2 R Program Variables and Assignment (Motivation)

The main purpose of a variable is to refer to something.

What is the syntax for this "refer to"? R uses assignment = or <-. With this syntax, we can have, for example, *x = 3* is read as *x* refers to 3 or 3 is *assigned* to the variable *x*.

Question:

- Is = in R the same as *equals*? (No. = in R is NO LONGER *equal*! We use == when we want to express two things are equal.)
- Write 2 = 2 in R console. What do you observe?

3.2 Assignment -- Formal Definition

Syntax of assignment (= or <-):

v **=** e or

v **<-** e

where v can be any *R program variable* and e is an R expression.

The statement is **read** as assign the value of expression e to variable v.

Once a variable is assigned to a value, the **value** of the variable is that value.

Note: For clarity, we normally use <− in this course.

3.2 R Program Variables and Assignment

Example:

Given

$$x = 3$$

How to read the R expression?

3.2 R Program Variables and Assignment

Example:

Given

$$x = 3$$

How to read the R expression? (assign the value of 3 to variable *x*)

What is the value of the R expression *x*?

3.2 R Program Variables and Assignment

Example:

Given

$$x = 3$$

How to read the R expression? (assign the value of 3 to variable *x*)

What is the value of the R expression *x*?   3

What is the value of $x + 4$?

3.2 R Program Variables and Assignment

Example:

Given

$$x = 3$$

How to read the R expression? (assign the value of 3 to variable *x*)

What is the value of the R expression *x*? 3

What is the value of $x + 4$?

To get the value of $x + 4$, we need to get the value of *x*, which is 3. Hence, this value 3 plus 4 is 7. This way of finding the value of an expression is the same as we did before by finding the value of each (sub)expression.

3.2 R Program Variables and Assignment

Variables vs **Strings**

Consider the following R expressions:

```
apples <- 10
x <- apples
y <- “apples”
```

What is the value of x? What is the value of y?

3.2 R Program Variables and Assignment

Variables vs **Strings**

Consider the following R expressions:

```
apples <- 10
x <- apples
y <- "apples"
```

What is the value of x? What is the value of y?

The value of x is 10, the value of y is "apples"

3.2 R Program Variables and Assignment

Example

Associate a name *point* to the vector (3, 4), we use assignment <-

```
point <- c(3,4)
```

read: how to read the expression?

3.2 R Program Variables and Assignment

Example

Associate a name *point* to the vector (3, 4), we use assignment <-

```
point <- c(3,4)
```

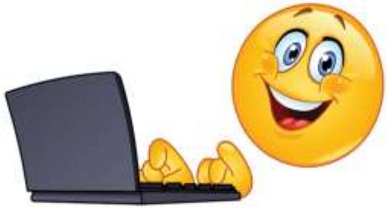
read: the value of *c(3,4)* is assigned to the variable *point*. The value/meaning of variable *point* is the tuple (3,4).

3.3 R Program Variables and Assignment--Hands on

Practice :

Open repl.it, write an R expression to assign c(3,4) to variable point? What is the value of point in R console?

Expression:



3.3 R Program Variables and Assignment--Hands on

Example：

Open repl.it, write an R expression to assign c(3,4) to variable point? What is the value of point in R console?

Expression:

```
> point <- c(3,4)
```

Computing Foundation

1 Primitive Types and Their Elements

2 Vector Type and c(...) Function

3 R Program Variables

4 Functions on Vectors

5 Functions

4.1.1 Indices of Vectors - formal definition

Given a vector, we can use indexes to access its components as we did for tuples.

Syntax is

$$v\,[\,i\,]$$

where v is an R *expression* whose value is a *vector* and i is an R *expression* whose value is an *index* (which can be a number or name).

Its **value** (meaning) is the component of v at index (the value of) i.

We **read** $v\,[\,i\,]$ as the value of vector v at index i, or the i^{th} component/element of vector v.

4.1 Indices of Vectors - formal definition

Syntax of accessing an element of a vector: *v* **[** *i* **]**

where *v* is an *R expression* whose value is a *vector* and *i* is an *R expression* whose value is an *index* (which can be a number or name).

We **read** *v* **[** *i* **]** as the value of vector *v* at index *i*.

Read definitions carefully. Does the expression below follow the syntax?

- c(1, 6)[1]?

4.1 Indices of Vectors - formal definition

Syntax of accessing an element of a vector: *v* **[** *i* **]**

where *v* is an *R expression* whose value is a *vector* and *i* is an *R expression* whose value is an *index* (which can be a number or name).

We **read** *v* **[** *i* **]** as the value of vector *v* at index *i*.

Does the following follow the syntax?

- c(1, 6)[1]?

  (Yes. the value of c(1,6) is (1,6) and 1 is an numbered index.)

- (1, 6)[1]?

4.1 Indices of Vectors - formal definition

Syntax of accessing an element of a vector: *v* **[** *i* **]**

where *v* is an *R expression* whose value is a *vector* and *i* is an *R expression* whose value is an *index* (which can be a number or name).

We **read** *v* **[** *i* **]** as the value of vector *v* at index *i*.

Does the following follow the syntax?

- c(1, 6)[1]?

  (Yes. the value of c(1,6) is (1,6) and 1 is an numbered index.)

- (1, 6)[1]? (No, (1,6) is not an R expression)
- (c(1, 6) + c(2, 1))[1+1-2+1]?

4.1 Indices of Vectors - formal definition

Syntax of accessing an element of a vector: *v* **[** *i* **]**

where *v* is an *R expression* whose value is a *vector* and *i* is an *R expression* whose value is an *index* (which can be a number or name).

We **read** *v* **[** *i* **]** as the value of vector *v* at index *i*.

Does the following follow the syntax?

- c(1, 6)[1]?

  (Yes. the value of c(1,6) is (1,6) and 1 is an numbered index.)

- (1, 6)[1]? (No, (1,6) is not an R expression)
- (c(1, 6) + c(2, 1))[1+1-2+1]? (Yes, value of c(1, 6) + c(2, 1) is (3,7))

4.1.1 Example of Indices of Vectors

Practice 1:

Write an R expression to get the first element of the vector (3,4)?

4.1.1 Example of Indices of Vectors

Practice 1:

Write an R expression to get the first element of vector (3,4)?

c(3,4)[1]

Practice 2:

Write an R expression to get the 2nd value of vector (3,4)?

4.1.1 Example of Indices of Vectors

Practice 1:

Write an R expression to get the first element of vector (3,4)?

c(3,4)[1]

Practice 2:

Write an R expression to get the 2nd value of vector (3,4)?

c(3,4)[2]

Example 3:

`x <- c(3,4)`, what is the value of `x[1]`?

4.1.1 Example of Indices of Vectors

Example 1:

Write an R expression to get the first element of vector (3,4)?

c(3,4)[1]

Example 2:

Write an R expression to get the 2nd value of vector (3,4)?

c(3,4)[2]

Example 3:

x <- c(3,4), What is the value of x[1]?

3 can you type the R expressions (one after another) into R

4.1.2 Indices of Named Vectors

Example

How to access the *x* component of vector c(“x” = 6, “y” = 8)?

We write R expression: c(“x” = 6, “y” = 8)[“x”]

4.1.2 Indices of Named Vectors

Practice

How to access the *x* component of vector c(“x” = 6, “y” = 8)?

We write R expression: c(“x” = 6, “y” = 8)[“x”]

Write an R expression to access the *y* component of that vector?

4.1.2 Indices of Named Vectors

Example

How to access the *x* component of vector c(“x” = 6, “y” = 8)?

We write R expression: c(“x” = 6, “y” = 8)[“x”]

How to access the *y* component of that vector?

```
c("x" = 6, "y" = 8)["y"]
```

4.1.2 Indices of Named Vectors (More Examples)

Note components of a named vectors in R can always be accessed by numbered index.

The *x* component of vector c("x" = 6, "y" = 8) is also the first component. R allows R expression

c("x" = 6, "y" = 8)[1]

It is **not** recommended to use this access method for named vectors in this course.

4.1.2 Indices of Named Vectors (More Examples)

This example shows the needs of variables. We can “store” the scores of students into the variable *grades*:

```
grades <- c(“Aaron” = 90,  “Bill” = 95, “Cecilia” = 100, “Dina” = 88)
```

Practice

Then you can easily get the grade of any student. Write R expression to get the grade of Aaron.

4.1.2 Indices of Named Vectors (More Examples)

This example shows the needs of variables. We can “store” the scores of students into the variable

```
grades <- c(“Aaron” = 90,  “Bill” = 95, “Cecilia” = 100, “Dina” = 88)
```

Then you can easily get the grade of any student. Write R expression to get the grade of Aaron.

```
grades[“Aaron”]
```

Write an R expression to get Cecilia’s grade.

4.1.2 Indices of Named Vectors (More Examples)

This example shows the needs of variables. We can "store" the scores of students into the variable

```
grades <- c("Aaron" = 90,  "Bill" = 95, "Cecilia" = 100, "Dina" = 88)
```

Then you can easily get the grade of any student. Write R expression to get the grade of Aaron.

```
grades["Aaron"]
```

Write an R expression to get Cecilia's grade.

```
grades["Cecilia"]
```

4.2 Operations and Functions on Vectors

4.2.1 Vector (tuple) + (or any op) Scalar

4.2.2 Vector (Tuple) + (or any op) vector

4.2.1 Operations Functions on Vectors and Scalar (Motivation)

We know that we can have vector (1, 2, 3) + 3 whose value is (4, 5, 6). What is the *R construct* to represent the expression?

- To represent (1, 2, 3) we use R expression c(1, 2, 3)
- To represent (1,2,3) we use R expression c(1,2,3) + 3!
- So, R provides a rather straightforward representation of functions over tuples (vectors)

4.2.1 Operations Functions on Vectors and Scalar --Formal Definition

Syntax:

$e + c$

where *e* is an *R expression* whose value is a *vector* and *c* is an *R expression* whose value is of an *integer* or *double primitive* type.

Read: (connecting syntax and meaning) add vector *e* and a number *c*.

4.2.1 Application of Operations Functions on Vectors and Scalar

Syntax: $e + c$ where e is an *R expression* whose value is an vector and c is an *R expression* whose value is of an integer or double primitive type.

Example:

- Consider 3 kids John, Wilson, and Eric. John's initial pocket money is \$10, Wilson's is \$8, and Eric's is \$12. Their pocket money increases by \$3 each.
  - Write an (set theory) expression to represent the final pocket money of John, Wilson, and Eric using *tuples* and addition (+).

4.2.1 Application of Operations Functions on Vectors and Scalar

Syntax: *e* + *c* where *e* is an *R expression* whose value is an vector and *c* is an *R expression* whose value is of an integer or double primitive type.

Example:

- Consider 3 kids John, Wilson, and Eric. John's initial pocket money is $10, Wilson's is $8, and Eric's is $12. Their pocket money increases by $3 each.
  - Write an (set theory) expression to represent the final pocket money of John, Wilson, and Eric using *tuples* and addition (+).

$$(10, 8, 12) + 3$$

  - Write an R expression to represent the final pocket money?

4.2.1 Application of Operations Functions on Vectors and Scalar

Syntax: *e* + *c* where *e* is an *R expression* whose value is an vector and *c* is an *R expression* whose value is of an integer or double primitive type.

Example:

- Consider 3 kids John, Wilson, and Eric. John's initial pocket money is $10, Wilson's is $8, and Eric's is $12. Their pocket money increases by $3 each.
  - Write an (set theory) expression to represent the final pocket money of John, Wilson, and Eric using *tuples* and addition (+).

$$(10, 8, 12) + 3$$

  - Write an R expression to represent the final pocket money?

```
c(10, 8, 12) + 3
```

4.2.1 Application of Operations Functions on Vectors and Scalar

Syntax: *e* + *c* where *e* is an *R expression* whose value is an vector and *c* is an *R expression* whose value is of an integer or double primitive type.

Example:

- Consider 3 kids John, Wilson, and Eric. John’s initial pocket money is $10, Wilson’s is $8, and Eric’s is $12. Their pocket money is **doubled**.

  - Write an (set theory) expression to represent the final pocket money of John, Wilson, and Eric using tuples and scalar?

4.2.1 Application of Operations Functions on Vectors and Scalar

Syntax: *e* + *c* where *e* is an *R expression* whose value is an vector and *c* is an *R expression* whose value is of an integer or double primitive type.

Example:

- Consider 3 kids John, Wilson, and Eric. John's initial pocket money is $10, Wilson's is $8, and Eric's is $12. Their pocket money is **doubled**.

    - Write an (set theory) expression to represent the final pocket money of John, Wilson, and Eric using tuples and scalar?

(10, 8, 12) * 2

    - Write an R expression to represent to the final pocket money?

4.2.1 Application of Operations Functions on Vectors and Scalar

Syntax: *e* + *c* where *e* is an *R expression* whose value is an vector and *c* is an *R expression* whose value is of an integer or double primitive type.

Example:

- Consider 3 kids John, Wilson, and Eric. John's initial pocket money is $10, Wilson's is $8, and Eric's is $12. Their pocket money is **doubled**.
  - Are you able to **invent** a new operation (like we did for +) to represent the final pocket money of John, Wilson, and Eric using tuples and multiplication?

    (10, 8, 12) * 2

  - Write an R expression to represent to the final pocket money?

    c(10, 8 , 12) * 2

4.2.2 More Applications of Operations on Vectors and Scalar

We will introduce a special type of vector such as

(FALSE, TRUE, TRUE, FALSE)

Aaron Bill Cecilia Dina

where every component is a logical value. It is called *logical vectors*.

Definition (Logical Vectors). A vector is **logical** if its components are logical values.

4.2.2 More Applications of Operations on Vectors and Scalar

Practice:

In a test Aaron got 90, Bill got 95, Cecilia got 100 and Dina got 88. Assume the grades of a set of students is represented as a named vector:

grades =	(90,	95,	100,	88)
	Aaron	*Bill*	*Cecilia*	*Dina*

We want to get a logical vector to represent if a student's grade is at least 95:

(FALSE,	TRUE,	TRUE,	FALSE)
Aaron	*Bill*	*Cecilia*	*Dina*

Question: Write an *expression* (in set theory language) whose value is the logical vector above?

4.2.2 More Applications of Operations on Vectors and Scalar

Practice: In a test Aaron got 90, Bill got 95, Cecilia got 100 and Dina got 88. Assume the grades of a set of students is represented as a named vector:

grades =	(90,	95,	100,	88)
	Aaron	*Bill*	*Cecilia*	*Dina*

We want to get a logical vector to represent if students grade is above 95:

(FALSE,	TRUE,	TRUE,	FALSE)
Aaron	*Bill*	*Cecilia*	*Dina*

Question: Write an *expression* (in set theory language) whose value is the logical vector above?

grades >= 95

4.2.2 More Applications of Operations on Vectors and Scalar

Practice.

In a test Aaron got 90, Bill got 95, Cecilia got 100 and Dina got 88.

In R console, type the following one by one

```
grades <- c(“Aaron” = 90,  “Bill” = 95, “Cecilia” = 100, “Dina” = 88)
grades >= 95
```

4.2.2 More Applications of Operations on Vectors and Scalar

Recall

grades =	(90,	95,	100,	88)
	Aaron	*Bill*	*Cecilia*	*Dina*

The value of grades >= 95?

- Since grades is (90, 95, 100, 88), grades >= 95 is?

4.2.2 More Applications of Operations on Vectors and Scalar

Recall

grades = (90, 95, 100, 88)

Aaron *Bill* *Cecilia*

The value of grades>= 95?

- Since grades is (90, 95, 100, 88), grades >= 95 is

(90, 95, 100, 88) >= 95

- What is the expression after applying >= to the above tuple and scalar?

4.2.2 More Applications of Operations on Vectors and Scalar

Recall

grades = (90, 95, 100, 88)

Aaron *Bill* *Cecilia* *Dina*

The value of grades>= 95?

- Since grades is (90, 95, 100, 88), grades >= 95 becomes

  (90, 95, 100, 88) >= 95

- What is the expression after applying >= to the above tuple and scalar?

(90>= 95, 95 >= 95, 100>= 95, 88>= 95)

Aaron *Bill* *Cecilia* *Dina*

4.2.2 More Applications of Operations on Vectors and Scalar

Get the final value of grades>= 95 step by step:

- Since grades is (90, 95, 100, 88), grades >= 95 becomes

  (90, 95, 100, 88) >= 95

- What is the expression after applying >= to the above tuple and scalar?

(90>= 95,	95 >= 95,	100>= 95,	88>== 95)
Aaron	*Bill*	*Cecilia*	*Dina*

- What is the value of the expression above?

4.2.2 More Applications of Operations on Vectors and Scalar

The value of grades>= 95?

- Since grades is (90, 95, 100, 88), grades >= 95 becomes

  (90, 95, 100, 88) >= 95

- What is the value of the expression above?

(90>= 95,	95 >= 95,	100>= 95,	88>== 95)
Aaron	*Bill*	*Cecilia*	*Dina*

- What is the value of the expression above?

(FALSE,	TRUE,	TRUE,	FALSE)
Aaron	*Bill*	*Cecilia*	*Dina*

4.2.2 More Applications of Operations on Vectors and Scalar

Practice.

In a test Aaron got 90, Bill got 95, Cecilia got 100 and Dina got 88.

Question: Using the earlier variable *grades*, write an R expression to find the logical vector indicating if a student gets a grade at most 90?

4.2.2 More Applications of Operations on Vectors and Scalar

Practice.

In a test Aaron got 90, Bill got 95, Cecilia got 100 and Dina got 88.

Question: Using the earlier variable *grades*, write an R expression to find the logical vector indicating if a student gets a grade at most 90?

```
grades <= 90
```

4.2.2 Operations Functions on Vectors and Vectors (Motivation)

We know that we can have vector (1, 2, 3) + (1, 3, 2) whose value is (2, 5, 5). What is the *R construct* to represent the expression?

- To represent (1, 2, 3) we use R expression c(1, 2, 3)
- To represent (1,2,3) + (1, 3, 2) we use R expression c(1,2,3) + c(1, 3, 2)

So, R provides a rather straightforward representation of functions over tuples (vectors)

4.2.2 Operations Functions on Vectors and Vectors

Syntax:

$$e_1 + e_2$$

where e_1 and e_2 are *R expressions* whose values are *vectors*.

Read: (connecting syntax and meaning) add vector e_1 and e_2.

4.2.2 Application of Operations Functions on Vectors and Vectors

Syntax: e_1 **+** e_2 where e_1 and e_2 are *R expressions* whose values are *vectors*.

Example:

- On Halloween Eve, in visiting the first home, Jamie got 1 piece of candy, Aaron got 2, and Eric got 3. For the second home, Jamie got 1 piece of candy, Aaron got 3, and Eric got 2, how many pieces does each child have in total after the two visits?
    - Write an (set theory) expression to represent the final number of candies of Jamie, Aaron, and Eric using *tuples* and addition (+).

4.2.2 Application of Operations Functions on Vectors and Vectors

Example: In the Halloween Eve, in visiting the first home, Jamie got 1 piece of candy, Aaron got 2, and Eric got 3. For the second home, Jamie got 1 piece of candy, Aaron got 3, and Eric got 2, how many pieces does each child have in total after the two visits?

- Write an (set theory) expression to represent the final number of candies of Jamie, Aaron, and Eric using *tuples* and addition (+).

$$(1, 2, 3) + (1, 3, 2)$$

- Write an *R expression* to represent the final number of candies?

4.2.2 Application of Operations Functions on Vectors and Vectors

Example: In the Halloween Eve, in visiting the first home, Jamie got 1 piece of candy, Aaron got 2, and Eric got 3. For the second home, Jamie got 1 piece of candy, Aaron got 3, and Eric got 2, how many pieces does each child have in total after the two visits?

- Write an (set theory) expression to represent the final number of candies of Jamie, Aaron, and Eric using *tuples* and addition (+).

  (1, 2, 3) + (1, 3, 2)

- Write an *R expression* to represent the final number of candies?

  c(1, 2, 3) + c(1, 3, 2)

4.2.2 Application of Operations Functions on Vectors and Vectors

Practice:

- In the first week, John spends $1 for a video game, and Peter spends $2 for a video game. In the second week, John spends $3 for a game, and Peter spends $2 for a game.
  - Write an R expression to represent the total cost spent by John and Peter.

4.2.2 Application of Operations Functions on Vectors and Vectors

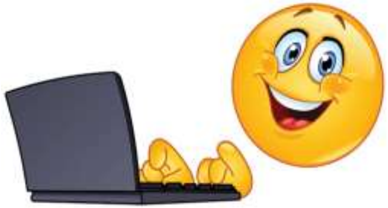
Practice:

- In the first week, John spends $1 for a video game, and Peter spends $2 for a video game. In the second week, John spends $3 for a game, and Peter spends $2 for a game.
  - Write an R expression to represent the total cost spent by John and Peter.

```
c(1, 2) + c(3, 2)
```

4.2.2 Application of Operations on Vectors and Vectors

Practice: The prices of 4 products A, B, C and D are $2, $5, $6 and $9 respectively. If there is a tax of 10% on each of these products, write an R expression to represent the final costs of A, B, C and D using named vectors, + and *. (Recall that the final cost of a product is its price plus its price times the tax rate.)



4.2.2 Application of Operations on Vectors and Vectors

Practice: The prices of 4 products A, B, C and D are $2, $5, $6 and $9 respectively. If there is a tax of 10% on each of these products, write an R expression to represent the final costs of A, B, C and D using named vectors, + and *. (Recall that the final cost of a product is its price plus its price times the tax rate.)

```
c(2, 5, 6, 9) + c(2, 5, 6, 9) * 0.1
```

4.2.3 More Applications of Operations on Vectors and Vectors

Practice.

Recall the grades of a set of students is represented as a named vector:

grades = (90, 95, 100, 88)
Aaron *Bill* *Cecilia* *Dina*

Question: Reuse the variable *grades* you created before, write an R expression to find the logical vector indicating if a student gets a grade between 60 and 90?

4.2.3 More Applications of Operations on Vectors and Vectors

Practice.

Recall the grades of a set of students is represented as a named vector:

$$\text{grades} = \big(\underset{\textit{Aaron}}{90}, \underset{\textit{Bill}}{95}, \underset{\textit{Cecilia}}{100}, \underset{\textit{Dina}}{88}\big)$$

Question: Reuse the variable *grades* you created before, write an R expression to find the logical vector indicating if a student gets a grade between 60 and 90?

```
grades >= 60 & grades <= 90
```

4.2.3 More Applications of Operations on Vectors and Vectors

Recall grades = (90, 95, 100, 88)

Aaron *Bill* *Cecilia* *Dina*

What is the value of grades >= 60 & grades <= 90?

The value of R expression: `grades >= 60`?

The value of R expression: `grades <= 90`?

4.2.3 More Applications of Operations on Vectors and Vectors

Recall grades = (90, 95, 100, 88)

Aaron *Bill* *Cecilia* *Dina*

What is the value of grades >= 60 & grades <= 90?

The value of R expression: `grades >= 60`?

`(TRUE , TRUE, TRUE, TRUE)`

`Aaron Bill Cecilia Dina`

The value of R expression: `grades <= 90`?

`(TRUE, FALSE, FALSE`

4.2.3 More Applications of Operations on Vectors and Vectors

grades >= 60: (TRUE, TRUE, TRUE, TRUE)

grades <= 90: (TRUE, FALSE, FALSE, TRUE)

The value of : `grades >= 60 & grades <= 90 is`

`(TRUE, TRUE, TRUE, TRUE)` **&** `(TRUE, FALSE, FALSE, TRUE)`

What is the expression after applying & to the above tuples ?

4.2.3 More Applications of Operations on Vectors and Vectors

grades >= 60: (TRUE, TRUE, TRUE, TRUE)

grades <= 90: (TRUE, FALSE, FALSE, TRUE)

The value of : grades >= 60 & grades <= 90 is

(TRUE, TRUE, TRUE, TRUE) **&** (TRUE, FALSE, FALSE, TRUE)

What is the expression after applying & to the above tuples ?

(TRUE **&** TRUE, TRUE **&** FALSE, TRUE **&** FALSE, TRUE **&** TRUE)

whose value is

4.2.3 More Applications of Operations on Vectors and Vectors

grades >= 60: `(TRUE, TRUE, TRUE, TRUE)`

`grades <= 90: (TRUE, FALSE, FALSE, TRUE)`

The value of : `grades >= 60 & grades <= 90 is`

`(TRUE, TRUE, TRUE, TRUE)` **&** `(TRUE, FALSE, FALSE, TRUE)`

What is the expression after applying & to the above tuples ?

`(TRUE` **&** `TRUE,  TRUE` **&** `FALSE,  TRUE` **&** `FALSE,  TRUE` **&** `TRUE)`

whose value is

`(TRUE,  FALSE,  FALSE,  TRUE)`

Aaron *Bill* *Cecilia* *Dina*

4.2.3 More Applications of Operations on Vectors and Vectors

Note.

According to our use of + on tuple plus tuple, we can use && too. However R uses && for a special operation on vectors.

We recommend to use & (instead of &&) for scalars and vectors

- *(x >= 10) & (x <= 100)*
- *c(x>=10, y>=10) & c(x<=100, y <= 100)*

Similarly we use | (instead of ||) for scalars and vectors

Computing Foundation

1 Primitive Types and Their Elements

2 Vector Type and c(...) Function

3 R Program Variables

4 Functions on Vectors

5 Functions

5 Functions (Movitation)

As we have learned in set theory language, functions are an important concept for us to represent knowledge to solve problems.

In fact, we have used so many functions (e.g, +, -, >, and etc.) in the standard types (integer, double, logical etc.) so far. Without them, it is hard to imagine that we can represent any information to solve problems meaningful to the world.

We also discuss some functions that we didn't see before in regular school subjects but they are important for us to model this world (and thus help us solve problems of interest to us).

5.1 Which Function (motivation)

Recall that we have learned *set builder notation* which is very powerful in representing the world through a set of interesting things. The *interestingness* is usually represented by functions and logical connectives. Recall that an interesting city *x* is represented by

$$big(x) = \text{true } \textbf{and } x \in citiesOfUSA,$$

and an integer in the *integer type* of language R is characterized by

$$x >= -2147483648 \textbf{ and } x <= 2147483647$$

We will study an R function *which* that help us to specify a set of interesting things.

5.2 Which Function Definition and Example

Problem. Our class consists of people from different grades. Assume we have the following information: Aaron and Cecilia are 10th graders, Bill 9th grader and Dina 11th.

The grades information can be represented by a function

$$grade: students \rightarrow grades$$

where $students = \{Aaron, Celicia, Bill, Dina\ \}$ and $grades = \{9, 10, 11\}$.

Use the *grade* function and set builder notation, write the set of students of 10th grade.

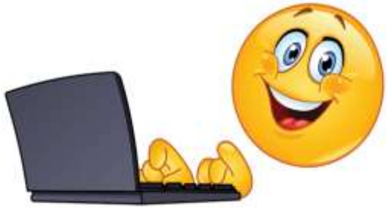
5.2 Which Function Definition and Example

Use the *grade* function and set builder notation, write the set of students of 10th grade.

$$\{x: grade(x) = 10\}$$

Problem. Write an R expression to represent such a set.

- Write an *R expression* to represent the content of the function *grade*:
- Find a logical vector indicating 10th graders
- A new function *which* select only 10th graders



5.2 Which Function Definition and Example

Assume we have the following information: Aaron and Cecilia are 10th graders, Bill 9th grader and Dina 11th. Also we assume we have function

$$grade\colon students \longrightarrow grades$$

where *students* = {*Aaron*, *Cecilia*, *Bill*, *Dina*} and *grades* = {9, 10, 11}.

- Write an *R expression* to represent the content of the function:

```
grade <- c("Aaron" = 10, "Cecilia" = 10, "Bill" = 9, "Dina" = 11)
```

5.2 Which Function Definition and Example

Use the *grade* function and set builder notation, write the set of students of 10th grade.

$$\{x: grade(x) = 10\}$$

Problem. Write an R expression to represent such a set.

- Recall the representation of a function of using *R*.
- Write an R expression to represent the logical vector indicating 10th graders

5.2 Which Function Definition and Example

Use the *grade* function and set builder notation, write the set of students of 10th grade.

$$\{x: grade(x) = 10\}$$

Problem. Write an R expression to represent such a set.

- Recall the representation of a function of using *R*.
- Write an R expression to represent the logical vector indicating 10th graders

```
is10grader <- (grade == 10)
```

5.2 Which Function Definition and Example

Now with *is10grader* being

(TRUE, FALSE, TRUE, FALSE)

Aaron *Bill* *Cecilia* *Dina*

We would like to know the names (i.e., named indices) where we have TRUE. Can you design a function (in intentional form) to achieve the above task?

5.2 Which Function Definition and Example

Now with is10grader being

(TRUE, FALSE, TRUE, FALSE)

Aaron *Bill* *Cecilia* *Dina*

We would like to know the names (i.e., named indices) where we have TRUE. Can you design a function (in intentional form) to achieve the above task?

The output of *myWhich(x)*, where *x* is a logical vector (named or unnamed), is the vector of the indices of *x* where the components are TRUE.

5.2 Which Function Definition

R provides the *which* function whose specification in intentional form is:

The output of function **`which(`*expression*`),`** where *expression* is an R expression whose value is *logical vector*, is the vector of indices of *expression* where the components are TRUE.

Practice:

What is the value of `which(c(TRUE, FALSE, TRUE))` ?

5.2 Which Function Definition

R provides *which* function whose specification in intentional form is:

The output of function **`which(`***expression***`)`**, where *expression* is an R expression whose value is *logical vector*, is the vector of indices of *expression* where the components are TRUE.

Practice:

What is the value of `which(c(TRUE, FALSE, TRUE))` ?

(1, 3) - the vector of indices where value is TRUE

5.2 Which Function Definition

Practice

Study the output of function `which(`*expression*`)` when *expression* is a named vector.

Recall `is10grader <- (grade == 10)`

Write R expression `which(is10grader)` and observe what the R console prints

5.2 Which Function Definition

Practice

The output of function **`which(`***expression***`)`** when *expression* is a named vector.

Recall `is10grader <- (grade == 10)`

Write R expression `which(is10grader)` and observe what R console print.

The value is (1, 3) but NOT (Aaron, Cecilia)

	Aaron	Cecilia
	1	3

recall `is10grader`

(TRUE,	FALSE,	TRUE,	FALSE)
Aaron	*Bill*	*Cecilia*	*Ding*

5.2 Which Function Definition

The output of function `which(`*expression*`)`,

- where *expression* is an R expression whose value is a *named logical vector*, is the *named* vector with named indices of *expression* where the components are TRUE, and with the content at each named index being corresponding numbered index in *expression*.

5.2 Which Function Definition

Practice

R expression `which(is10grader)` has the value

	Aaron	Cecilia
(1,		3)

We want to get the names now.

In R, we have function names(...). The output of function **`names(`***expression***`)`**, where *expression* is an R expression whose value is a *named vector*, is a vector of the named indexes of *expression*.

Using the *names* function, write an R expression to find the names of the vector which(is10grader):

5.2 Which Function Definition

Practice

R expression `which(is10grader)` has the value

(1,		3)
	Aaron	Cecilia

We want to get the names now.

In R, we have function names(...). The output of function **`names(`***expression***`)`**, where *expression* is an R expression whose value is a *named vector*, is a vector of the named indexes of *expression*.

Using the *names* function, write an R expression to find the names of vector which(is10grader): `names(which(is10grader))`

5.2 Summary: Specifying a Set

Use the *grade* function and set builder notation, the set of students of 10th grade is

$$\{x: grade(x) = 10\}$$

Problem. Write an R expression to represent such a set. Assume grade info are in the named vector *grade*.

Get the logical vector telling us who is 10th grader: `grade == 10`

Get the vector with only 10th graders. For future easy reference, associate it with a variable:

```
is10grader <- which(grade == 10)
```

Get the names, i.e., named indices:

`names(is10grader)` or `names(which(grade == 10))`

5.3 Sum, Min, Max Functions

R provides more functions for vectors.

- The output of function `sum(`*expression*`),` where *expression* is an R expression whose value is the summation of all elements.

Practice.

Using variable *grades*, write an R expression to find the total score of all students:

5.3 Sum, Min, Max Functions

R provides more functions for vectors.

- The output of function `sum(`*expression*`)`, where *expression* is an R expression whose value is the summation of all elements.

Practice.

Using variable *grades*, write an R expression to find the total score of all students:

```
sum(grades)
```

Write an R expression to find the average score of the students. You can use function **`length(`***expression***`)`** whose output is the number of components in vector *expression*.

5.3 Sum, Min, Max Functions

R provides more function for vectors.

- The output of function **`sum(`***expression***`),`** where *expression* is an R expression whose value is the summation of all elements.

Practice.

Using variable *grades*, write an R expression to find the total score of all students:

```
sum(grades)
```

Write an R expression to find the average score of the students. You can use function **`length(`***expression***`)`** whose output is the number of elements in vector *expression*.

```
sum(grades)/length(grades)
```

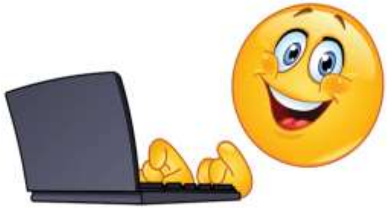
5.3 Sum, Min, Max Functions

R provides more function for vectors.

- The output of function `min(`*expression*`),` where *expression* is an R expression whose value is the element has minimum value.

Practice.

Using variable *grades*, write an R expression to find the minimal score of all students:



5.3 Sum, Min, Max Functions

R provides more function for vectors.

- The output of function **`min(`***expression***`),`** where *expression* is an R expression whose value is the element has minimum value.

Practice.

Using variable *grades*, write an R expression to find the minimal score of all students:

```
min(grades)
```

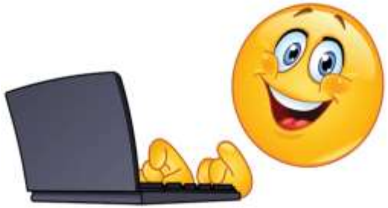
5.3 Sum, Min, Max Functions

R provides more function for vectors.

- The output of function **`max`**(*expression*), where *expression* is an R expression whose value is the element has maximum value.

Practice.

Using variable *grades*, write an R expression to find the maximal score of all students:



5.3 Sum, Min, Max Functions

R provides more function for vectors.

- The output of function **`max`**`(`*expression*`),` where *expression* is an R expression whose value is the element has maximum value.

Practice.

Using variable *grades*, write an R expression to find the maximal score of all students:

```
max(grades)
```

5.4 Practices of using functions

Roller coaster	Type	Height (ft)	Design	Speed (mph)	Duration (sec)
Wildfire	Wood	187.0	Sit down	70.2	120
Skyline	Steel	131.3	Inverted	50.0	90
Goliath	Wood	165.0	Sit down	72.0	105
Helix	Steel	134.5	Sit down	62.1	130
Banshee	Steel	167.0	Inverted	68.0	160
Black Hole	Steel	22.7	Sit down	25.5	75

Problem. The height is a function from roller coasters to their height

Practice. Write an R expression(s) to represent the height and associate to variable *height*.

5.4 Practices of using functions

Roller coaster	Type	Height (ft)	Design	Speed (mph)	Duration (sec)
Wildfire	Wood	187.0	Sit down	70.2	120
Skyline	Steel	131.3	Inverted	50.0	90
Goliath	Wood	165.0	Sit down	72.0	105
Helix	Steel	134.5	Sit down	62.1	130
Banshee	Steel	167.0	Inverted	68.0	160
Black Hole	Steel	22.7	Sit down	25.5	75

Problem. The height is a function from roller coasters to their height

Practice. Write an R expression(s) to represent the height and associate to variable *height*.

```
height <- c("Wildfire" = 187.0, "Skyline" = 131.3, "Goliath" = 165.0,
"Helix" = 134.5, "Banshee" = 167.0, "Balck Hole" = 22.7)
```

5.4 Practices of using functions

Roller coaster	Type	Height (ft)	Design	Speed (mph)	Duration (sec)
Wildfire	Wood	187.0	Sit down	70.2	120
Skyline	Steel	131.3	Inverted	50.0	90
Goliath	Wood	165.0	Sit down	72.0	105
Helix	Steel	134.5	Sit down	62.1	130
Banshee	Steel	167.0	Inverted	68.0	160
Black Hole	Steel	22.7	Sit down	25.5	75

Problem. Speed is also a function from roller coasters to speed.

Practice. Write an R expression(s) to represent the speed and associate it to variable *speed*.

5.4 Practices of using functions

Roller coaster	Type	Height (ft)	Design	Speed (mph)	Duration (sec)
Wildfire	Wood	187.0	Sit down	70.2	120
Skyline	Steel	131.3	Inverted	50.0	90
Goliath	Wood	165.0	Sit down	72.0	105
Helix	Steel	134.5	Sit down	62.1	130
Banshee	Steel	167.0	Inverted	68.0	160
Black Hole	Steel	22.7	Sit down	25.5	75

Problem. If your friend likes fast roller coasters, but is afraid of heights, how can we help them pick a good roller coaster for them? Write an R expression(s) to represent the set of roller coasters with speed greater than 40 mph but height less than 150 ft. Assume information on speed is in the named vector *speed*, and information on height is in the named vector *height*.