Concepts in Computing with Data

Stat 133, Fall 2024

Lecture 3, 09/04/2024

1. Review

i. Logical Vector

```
a = c(TRUE, FALSE, NA)
```

ii. Integer Vector

```
b = c(2L, 4L, 6L)
```

iii. Double Vector

```
c = c(1, 2, 3)
d = c(1.1, 2.2, 3.3)
```

iv. Character Vector

```
e = c("a", "b", "c", "d")
```

2. Naming Vectors

2.1. Conceptual

In R, vectors can have names associated with each element. Naming vectors allows each element to be referenced by its corresponding name instead of its index, which can make the code more readable and easier to manage. The names can be assigned using the `names()` function.

Assigning Names to Vectors: The `names()` function allows us to associate names with vector elements. Here's how it works:

- If the number of names matches the length of the vector, each element is given a name.
- If there are more names than elements in the vector, R will throw an error, as it cannot assign names to non-existent elements.
- If there are fewer names than elements in the vector, R will assign the names to the corresponding elements and assign `NA` to the unnamed elements.

2.2. Example 1

Consider the following vectors:

```
x = c(2, 4, 6)
y = c(2, 4, 6)
```

```
abc = c("a", "b", "c")
abcd = c("a", "b", "c", "d")
```

Assigning names:

```
names(x) = abc # assigns elements in vector abc to vector x
names(y) = abcd
```

2.3. Example 2

Consider the following:

```
z = c(2, 4, 6, 8)
names(z) = abc
```

Result:

a	b	c	<NA>
2	4	6	8

Question

Explain why `names(y) = abcd` results in an error message but `names(z) = abc` does not.

Hint: Refer to section 2.1

3. Other Functions

3.1. Repeating Elements

Using the same vector `x` as in section 2:

```
rep(x, times = 2)
```

Result: Repeats each element in vector x, 2 times

a	b	c	a	b	c
2	4	6	2	4	6

```
rep(x, each = 2, times = 2)
```

Result: Repeats each element twice in vector x, 2 times

a	a	b	b	c	c	a	a	b	b	c	c
2	2	4	4	6	6	2	2	4	4	6	6

3.2. Vector Initialization

i. `vector(mode = "logical", length = 3)`
   Result: `FALSE FALSE FALSE`

ii. `vector(mode = "integer", length = 3)`
   Result: `0 0 0`

iii. `vector(mode = "double", length = 3)`
   Result: `0 0 0`

iv. `vector(mode = "character", length = 3)`
   Result: `"" "" ""`

3.3. Vector Arithmetic

R supports element-wise arithmetic operations on vectors. These include addition, subtraction, multiplication, and division, which are performed on corresponding elements of two vectors of the same length.

Example:

```
a = c(1, 2, 3)
b = c(4, 5, 6)

a + b # Result:  5 7 9
a - b # Result:  -3 -3 -3
a * b # Result:  4 10 18
a / b # Result:  0.25 0.4 0.5
```

4. Implicit Coercion

4.1. Conceptul

Implicit coercion occurs when elements of different data types are combined within the same vector. R will automatically convert (or coerce) all elements to a common data type. The coercion follows a hierarchy, converting all elements to the most flexible type to avoid loss of information.

Important

Hierarchy of Data Types: logical < integer < double < character

When coercion occurs:

- Logical values (`TRUE`, `FALSE`) are converted to integers (`TRUE` → 1, `FALSE` → 0).
- Integers are converted to doubles (numeric values with decimal points).
- Characters take precedence over all other types, meaning that if a character is present in a vector, all elements will be coerced to characters.

Implicit coercion allows R to maintain uniform data types within vectors, but it can sometimes lead to unintended results.

4.2. Example 3

Consider the following:

```
int = c(2L, 4L, 6L)
logi = c(TRUE, FALSE)

vec = c(FALSE, 1L) # Result:  0 1
```

Question

Given the output in example 3, is 0 a double or an integer?

Hint: Refer to section 4.1.

5. Atomic Vectors

Atomic vectors are the simplest data structures in R, storing elements of only a single type. Common atomic vector types include logical, integer, double (numeric), and character. R ensures that all elements within an atomic vector are of the same data type, and if necessary, performs implicit coercion to enforce this.

5.1. Coercion in Atomic Vectors

Recall the hierarchy that R uses in section 4.1. Consider the example below:

`mixed_vec = c(TRUE, 2L, "apple")`
Result: `"TRUE" "2" "apple"`

6. Explicit Coercion Functions

In R, explicit coercion functions are used when you need to manually convert elements from one data type to another. This is particularly useful when R's implicit coercion isn't suitable for your needs, or when you need to enforce a specific data type for a vector.

R provides several functions to explicitly coerce data types, such as:

- `as.logical()` – to convert values to logical (`TRUE, FALSE`).
- `as.integer()` – to convert values to integers.
- `as.double()` – to convert values to doubles (numeric).
- `as.character()` – to convert values to characters (strings).

Coercion is useful, but you need to be cautious, as converting between incompatible types (e.g., trying to convert characters to numbers) may result in warnings or NA values.

6.1. Examples

- `as.logical(c(0, 1, 2))`
  Result: `FALSE TRUE TRUE`
- `as.integer(c(1, 2.2, 3.33, 4.444))`
  Result: `1 2 3 4`
- `as.double(c("a", "b", TRUE, 133))`
  Result: Warning: `NAs introduced by coercion:  NA NA NA 133`

7. Functions to Test Data Type

i. `is.logical()`

ii. `is.integer()`

iii. `is.character()`

iv. `is.numeric()`

Example:

```
is.logical(x) # False
is.logical(logi) # True
```

8. Vectorization

Vectorized code applies the same operation to each element. This is very efficient, and should always be used when possible. It can be costly to loop through each element in an array/vector.

```
sqrt(x) # Applies square root to every element in x
mean(x) # Computes the mean (not a vectorized function)
```

9. Recycling

In R, recycling occurs when you perform operations on vectors of different lengths. If one vector is shorter than the other, R will recycle the elements of the shorter vector to match the length of the longer vector. This means the shorter vector's elements are reused repeatedly until the operation is completed.

Important

R will only issue a warning if the length of the longer vector is not a multiple of the length of the shorter vector. Otherwise, no warning will be raised.

Consider the case `length(x)` $>$ `length(y)` and `length(x)` $\%$ `length(y)` $= 0$:

```
x = c(2, 4, 6, 8)
y = c(2, 1)

x + y # Result:  4 5 8 9
```

9.2. Example with Warning

Consider the case `length(u)` $>$ `length(v)` but `length(u)` $\%$ `length(v)` $\neq 0$:

```
u = c(2, 4, 6, 8, 10)
v = c(2, 1)

u + v # Result:  4 5 8 9 12
```

10. Subsetting (to be covered in lab)