Lecture 4: Problem Solving

Ingmar Sturm
UCSB

2024-06-27

Special thanks to Robin Liu for select course content used with permission.

Problem Solving

Today we will tackle some possibly challenging coding problems.

These problems can all be solved using the simple techniques we have learned so far.

During a job interview, you will be asked to solve problems far more challenging than what I have here. Prepare yourself for job interviews by grinding coding problems.

You can find more coding exercises here:

https://www.kaggle.com/

https://exercism.org/tracks/r/exercises

Problem Solving

I will describe a coding problem which involves writing a function taking certain arguments.

The output of your function should match some provided **test cases**.

Remove element

You are given a numeric vector `v` and a number `target`. Create a function `remove_elt` that returns a vector containing elements of `v` but with `target` removed.

```r
remove_elt <- function(v, target) {
  # Your code here
}
```

```r
remove_elt(c(2, 3, 3, 5), 3)
```

```
## [1] 2 5
```

```r
remove_elt(c(14, 14, 7, 7, 14, 10), 14)
```

```
## [1]  7  7 10
```

05:00

Print Square

Write a function `print_square(n)`. `n` is a nonnegative integer and `print_square(n)` prints a square of `*` of dimension `n` by `n`.

```r
print_square <- function(n) {
  # Your code here
}
```

```r
print_square(2)
```

```
## [1] "*" "*"
## [1] "*" "*"
```

```r
print_square(5)
```

```
## [1] "*" "*" "*" "*" "*"
## [1] "*" "*" "*" "*" "*"
## [1] "*" "*" "*" "*" "*"
## [1] "*" "*" "*" "*" "*"
## [1] "*" "*" "*" "*" "*"
```

05:00

Print Triangle

Write a function `print_triangle(n)` that prints a triangle of side length `n`.

```r
print_triangle <- function(n) {
  # Your code here
}
```

```r
print_triangle(3)
```

```
## [1] "*"
## [1] "*" "*"
## [1] "*" "*" "*"
```

```r
print_triangle(5)
```

```
## [1] "*"
## [1] "*" "*"
## [1] "*" "*" "*"
## [1] "*" "*" "*" "*"
## [1] "*" "*" "*" "*" "*"
```

05:00

Max vector

You are given two integer vectors `nums1` and `nums2`, and an integer `n` representing the number of elements in `nums1` and `nums2`, which have the same length.

Write a function `max_vec` that returns a vector of the element-wise maximum of `nums` and `nums2`.

```r
max_vec <- function(nums1, nums2, n) {
  result <- vector(length = n) # this initializes an "empty" vector with length n
  # Your code here
  return(result)
}
```

```r
max_vec(1:4, c(0, 27, -5, 19), 4)
```

```
## [1]  1 27  3 19
```

08:00

Small Count Simple

Write a function `small_count_simple(v, target)` that takes a numeric vector `v`, a number `target`, and returns how many elements of `v` are less than `target`

```r
small_count_simple <- function(v, target) {
  # Your code here
}
```

```r
small_count_simple(c(10, 15, -2, 5), 11)
```

```
## [1] 3
```

```r
small_count_simple(c(0, 0, -2, 5), 4)
```

```
## [1] 3
```

Hint: Use a *recycling* comparison and the numeric value of logicals. Review Lecture 2 slides if necessary.

03:00

Small Count

Write a function `small_count(v)` that takes a vector `v` and returns, for each element `v[i]`, how many elements of `v` are less than `v[i]`.

```
small_count <- function(v) {
  result <- vector(length = length(v))
  # Your code here
  return(result)
}
```

```
small_count(c(12, 100, -3))
```

```
## [1] 1 2 0
```

```
small_count(c(12, 100, -3, -12))
```

```
## [1] 2 3 1 0
```

05:00

Two Sum

Write a function `two_sum` that takes a vector `v` and a number `target` and returns two *indices* of numbers in `v` that add up to `target`. The two indices cannot be equal.

```r
two_sum <- function(v, target) {
  i <- 0
  j <- 0
  # Your code here
  # Hint: one way is to use two for loops
}
```

```r
two_sum(c(2, 7, 11, 15), 9)
```

```
## [1] 2 1
```

```r
two_sum(c(3, 2, 4), 6)
```

```
## [1] 3 2
```

```r
two_sum(c(3, 3), 6)
```

```
## [1] 2 1
```

08:00

Which

This is an important function. `which` takes a logical vector and returns a vector of indices of TRUE entries.

```
which(c(T, F, F, T, T))
```

```
## [1] 1 4 5
```

Write a function `first_even` that takes a numeric vector `v` and returns the first element of `v` that is an even number.

```
first_even(c(1, 3, 5, 6, 7, 8))
```

```
## [1] 6
```

```
first_even(c(-1, -4, 0, 2))
```

```
## [1] -4
```

05:00

Last Negative

Write a function `last_negative` that takes a numeric vector `v` and returns the last element that is less than zero.

```
last_negative(c(1, -9, 0, 6, -7, 0))
```

```
## [1] -7
```

```
last_negative(c(-1, -4, 0, 2))
```

```
## [1] -4
```

05:00

any and all

These functions are usful. They take a logical vector and return a single logical:

```
any(c(F, F, T, F, F))
```

```
## [1] TRUE
```

```
all(c(F, F, T, F, F))
```

```
## [1] FALSE
```

```
all(c(T, T, T))
```

```
## [1] TRUE
```

Write an expression that tests if a vector contains all negative elements.

01:00

Vectorized Two Sum

```
two_sum <- function(v, target) {
  pairs <- combn(seq_along(v), 2)
  w <- v[pairs[1, ]] + v[pairs[2, ]] == target
  pairs[, which(w)]
}
```