Class Prep 1.3.1

Riley Primeau

2022-08-29

Section 1.1.3 Efficiency

```
count <- 0
m <- 10
for(i in 1:m)
  count <- count + 1
count
```

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

```
count <- 0
m <- 10
n <- 7
for(i in 1:m)
  for(j in 1:n)
    count <- count + 1
count
```

```
## [1] 70
```

```
count = 0
m = 10
for(i in 1:m)
  for(j in 1:i)
    count = count + 1
count
```

```
## [1] 55
```

```
isPrime = function(n) {
  if(n == 2)
    return(TRUE)

  for(i in 2:sqrt(n))
    if(n %% i ==0)
      return(FALSE)

  return(TRUE)
}
isPrime(5)
```

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

```
count = 0
m = 10
x = 1:100
y = rep(1, 100)
for(i in 1:m) {
  y = y * x
  count = count + 1
}
count

## [1] 10
```

##Section 1.2.1 Data Types

```
TRUE == -1; TRUE == 0; TRUE == 1; TRUE == 3.14

## [1] FALSE

## [1] FALSE

## [1] TRUE

## [1] FALSE

FALSE == -1; FALSE == 0; FALSE == 1; FALSE == 3.14

## [1] FALSE

## [1] TRUE

## [1] FALSE

## [1] FALSE

TRUE < FALSE; TRUE > FALSE

## [1] FALSE

## [1] TRUE

x = c(TRUE, FALSE, TRUE, FALSE, TRUE)
sum(x); mean(x); length(x)

## [1] 3

## [1] 0.6

## [1] 5

x = c(1, 2, 3, 4, 3, 1, 2, 3, 3, 4, 1, 3, 4, 3)
sum(x==3)

## [1] 6

sum(x<3)

## [1] 5

x = 3.14
is.numeric(x)

## [1] TRUE

is.integer(x)

## [1] FALSE
```

```
is.integer(3)

## [1] FALSE

x = 3.14
as.numeric(x)

## [1] 3.14

as.integer(x)

## [1] 3
```

##Section 1.2.2 Data Structures

```
(x = c(1, 0, 1 , 0))
```

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

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

```
## [1] 1 0 1 0 2 4 6
```

```
(z = c(x, y))
```

```
##  [1] 1 0 1 0 1 0 1 0 2 4 6
```

```
z[10]
```

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

```
z[c(10, 9, 1)]
```

```
## [1] 4 2 1
```

```
(z1 = list(a = 3, b = 4))
```

```
## $a
## [1] 3
##
## $b
## [1] 4
```

```
(z2 = list(s = "test", nine = 9))
```

```
## $s
## [1] "test"
##
## $nine
## [1] 9
```

```
(z = list(z1, z2))

## [[1]]
## [[1]]$a
## [1] 3
##
## [[1]]$b
## [1] 4
##
##
## [[2]]
## [[2]]$s
## [1] "test"
##
## [[2]]$nine
## [1] 9

(A = matrix(1:12, 3, 4))

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

A[2,3]

## [1] 8

A[2,]

## [1]  2  5  8 11

A[,3]

## [1] 7 8 9

x1 = 1:3
x2 = 4:6
cbind(x1, x2)

##      x1 x2
## [1,]  1  4
## [2,]  2  5
## [3,]  3  6

rbind(x1, x2)

##    [,1] [,2] [,3]
## x1    1    2    3
## x2    4    5    6
```

```
(A)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
t(A)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
```

```
t(x)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    1    0
```

```
t(t(x))
```

```
##      [,1]
## [1,]    1
## [2,]    0
## [3,]    1
## [4,]    0
```

```
NA == NA
```

```
## [1] NA
```

```
NA == 1
```

```
## [1] NA
```

```
natest = c(1, 2, NA, 4, 5)
is.na(natest)
```

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

##Section 1.3.1 Summation Algorithms

```
naivesum <- function(x) {
    s <- 0
    n <- length(x)

    for(i in 1:n)
        s <- s + x[i]
    return(s)
}

x = c(1, 2, 3, 4.5, -6)
naivesum(x)

## [1] 4.5

pwisesum <- function(x) {
    n <- length(x)

    if(n == 1)
        return(x)
    m = floor(n / 2)
    return(pwisesum(x[1:m]) + pwisesum(x[(m + 1):n]))
}

pwisesum(x)

## [1] 4.5

kahansum <- function(x) {
    comp <- s <- 0
    n <- length(x)

    for(i in 1:n) {
        y <- x[i] - comp
        t <- x[i] + s
        comp <- (t - s) - y
        s <- t
    }
    return(s)
}

kahansum(x)

## [1] 4.5

sum(c(1, 2, 3, 4, NA, 5))

## [1] NA
```

```
sum(c(1, 2, 3, 4, NA, 5), na.rm = TRUE)

## [1] 15
```