Lab 5: Practice programming basics

Stat 133, Fall 2016

Gaston Sanchez

Programming Basics

This document provides more examples and supporting material to help you practice writing basic functions, if-then-else statements, and loops.

Simple Math Functions

Consider the following mathematical functions:

$$f(x) = x^2, \quad g(x) = 2x + 5$$

Write two functions `f()` and `g()` based on the previous equations.

```
# your function f()
```

```
# your function g()
```

Test your functions with:

```
f(2)
f(-5)

g(0)
g(-5/2)
```

Write code to create the following composite functions:

- `fog()` for the composite function: $f \circ g(x)$
- `gof()` for the composite function: $g \circ f(x)$

```
# your function fog()
```

```
# your function gof()
```

Test your composite functions with:

```r
fog(2)
fog(-5)

gof(0)
gof(-5/2)
```

Gaussian Function

The Gaussian (Normal) function, given in the equation below, is one of the most widely used functions in science and statistics:

$$f(x) = \frac{1}{\sqrt{2\pi s}} exp \left\{ -\frac{1}{2} \left(\frac{x-m}{s} \right)^2 \right\}$$

The parameters s and m are real numbers, where s must be greater than zero.

Make a function `gaussian()` that takes three arguments: `x`, `m`, and `s`. Evaluate the function with $m = 0$, $s = 2$, and $x = 1$.

```r
# your code
```

Quartiles

You can use the `summary()` function with a numeric vector to obtain descriptive statistics:

```r
x1 <- rnorm(100)
summary(x1)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
## -2.47200 -0.68800  0.17010  0.09694  0.88280  2.24000
```

Let's write a function to compute the quartiles of a numeric vector:

```r
quartiles <- function(x) {
  quantile(x, probs = c(0.25, 0.50, 0.75, 1))
}
```

Now let's test it:

```r
quartiles(mtcars$wt)
```

```
##     25%     50%     75%    100% 
## 2.58125 3.32500 3.61000 5.42400
```

Many functions that work on vectors have a special argument: `na.rm`. This parameter is a logical value to indicate whether NAs should be removed or not. Because the `quantile()` function does come with the `na.rm` argument, we can take advantage of it and pass it to our `quartiles()` function:

```
quartiles <- function(x, na.rm = FALSE) {
  quantile(x, probs = c(0.25, 0.50, 0.75, 1), na.rm = na.rm)
}
```

Let's get the weight of cars and add some missing values:

```
weight <- mtcars$wt
weight[c(1, 10, 20)] <- NA
```

If you apply `quartiles()` on `weight` using the default call, you will get an error. To remove missing values, you can use `na.rm = TRUE`:

```
quartiles(weight, na.rm = TRUE)
```

```
##   25%   50%   75%  100%
## 2.770 3.435 3.730 5.424
```

Your turn: Descriptive Statistics

Write a function `descriptive()` that takes a numeric vector as input, and returns a named vector with the following descriptive statistics:

- `min`: minimum
- `q1`: first quartile (Q2)
- `median`: median
- `mean`: mean
- `q3`: third quartile (Q3)
- `max`: maximum
- `range`: range or span (max - min)
- `iqr`: interquartile range (IQR)
- `sd`: standard deviation

```
# your descriptive() function
descriptive <- function() {
  # fill in

}
```

If Conditionals

Write R code that will "squish" a number into the interval [0, 100], so that a number less than 0 is replaced by 0 and a number greater than 100 is replaced by 100.

```
z <- 100*pi
# Fill in the following if-else statements. You may (or may not)
# have to add or subtract else if or else statements.
if (TRUE) { # Replace TRUE with a condition.
```

```r
} else if (TRUE) { # Replace TRUE with a condition.

} else {

}
```

```
## NULL
```

Multiple If's

A common situation involves working with multiple conditions at the same time. You can chain multiple if-else statements:

```r
y <- 1 # Change this value!

if (y > 0) {
  print("positive")
} else if (y < 0) {
  print("negative")
} else {
  print("zero?")
}
```

```
## [1] "positive"
```

Even number

Write a function `is_even()` that determines whether a number is even (i.e. multiple of 2). If the input number is even, the output should be `TRUE`. If the input number is odd, the output should be `FALSE`. If the input is not a number, the output should be `NA`

For example:

```r
# even number
is_even(10)
```

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

```r
# odd number
is_even(33)
```

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

```r
# not a number
is_even('a')
```

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

Odd number

Use your function `is_even()` to write a function `is_odd()` that determines if a number is odd (i.e. not a multiple of 2). If a number is odd, the output should be `TRUE`; if a number is even the output should be `FALSE`; if the input is not a number the output should be `NA`

For example:

```r
# odd number
is_odd(1)
```

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

```r
# even number
is_odd(4)
```

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

```r
# not a number
is_odd('a')
```

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

Switch

Working with multiple chained if's becomes cumbersome. Consider the following example that uses several if's to convert a day of the week into a number:

```r
# Convert the day of the week into a number.
day <- "Tuesday" # Change this value!

if (day == 'Sunday') {
  num_day <- 1
} else {
  if (day == "Monday") {
    num_day <- 2
  } else {
    if (day == "Tuesday") {
      num_day <- 3
    } else {
      if (day == "Wednesday") {
        num_day <- 4
      } else {
        if (day == "Thursday") {
          num_day <- 5
        } else {
          if (day == "Friday") {
            num_day <- 6
          } else {
            if (day == "Saturday") {
              num_day <- 7
            }
```

```
          }
        }
      }
    }
  }
}


num_day
```

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

Working with several nested if's like in the example above can be a nigthmare.

In R, you can get rid of many of the braces like this:

```
# Convert the day of the week into a number.
day <- "Tuesday" # Change this value!

if (day == 'Sunday') {
  num_day <- 1
} else if (day == "Monday") {
  num_day <- 2
} else if (day == "Tuesday") {
  num_day <- 3
} else if (day == "Wednesday") {
  num_day <- 4
} else if (day == "Thursday") {
  num_day <- 5
} else if (day == "Friday") {
  num_day <- 6
} else if (day == "Saturday") {
  num_day <- 7
}

num_day
```

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

But still we have too many if's, and there's a lot of repetition in the code. If you find yourself using many if-else statements with identical structure for slightly different cases, you may want to consider a **switch** statement instead:

```
# Convert the day of the week into a number.
day <- "Tuesday" # Change this value!

switch(day, # The expression to be evaluated.
  Sunday = 1,
  Monday = 2,
  Tuesday = 3,
  Wednesday = 4,
  Thursday = 5,
```

```
  Friday = 6,
  Saturday = 7,
  NA) # an (optional) default value if there are no matches
```

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

Switch statements can also accept integer arguments, which will act as indices to choose a corresponding element:

```
# Convert a number into a day of the week.
day_num <- 3 # Change this value!

switch(day_num,
  "Sunday",
  "Monday",
  "Tuesday",
  "Wednesday",
  "Thursday",
  "Friday",
  "Saturday")
```

```
## [1] "Tuesday"
```

Your turn again!

Write a switch statement to determine if a given lowercase letter is a vowel. Assume that "y" is not a vowel.

```
letter <- "e" # Change this value!

switch(letter,
  a = "vowel") # Modify as necessary.
```

Loops

```
# Divide a number by 2 until it becomes odd.
val_rep <- 898128000 # Change this value!

repeat {
  print(val_rep)
  if (val_rep %% 2 == 1) { # If val_rep is odd,
    break                  # end the loop.
  }
  val_rep <- val_rep / 2 # Divide val_rep by 2 since val_rep was even.
  # When the end of the loop is reached, return to the beginning of the loop.
}
```

```
## [1] 898128000
## [1] 449064000
## [1] 224532000
## [1] 112266000
## [1] 56133000
## [1] 28066500
## [1] 14033250
## [1] 7016625
```

While Loop

```
val_while <- 898128000 # Change this value!

while (val_while %% 2 == 0) { # Continue the loop as long as val_while is even.
  print(val_while)
  val_while <- val_while / 2
}
```

```
## [1] 898128000
## [1] 449064000
## [1] 224532000
## [1] 112266000
## [1] 56133000
## [1] 28066500
## [1] 14033250
```

```
print(val_while)
```

```
## [1] 7016625
```

Write R code which multiplies a positive number by 3 and adds 1 until the result is greater than 10000. For example, 2015 -> 6046 -> 18139. Write both a `while` loop and a `repeat` loop.

```
# while loop
n_while <- 314 # Change this value!
while (FALSE) { # Replace FALSE with your stopping condition.
  # Fill in.
}
```

```
# repeat loop
n_rep <- 314 # Change this value!

repeat {
  break # Replace this with your code.
}
```

Your turn! Now generalize the above code to create a function `reduce()` which performs the same operation. (You should change very little.)

```r
# your reduce() function
reduce <- function(x) {
  # Fill in.
  while(x %% 2 == 0) {
    x <- x / 2
  }
  return(x)
}

reduce(898128000)
```

```
## [1] 7016625
```

For loop

Write a for loop to add 1 to every element of a vector.

```r
vec <- c(3, 1, 4) # Change this value!

for (j in c()) { # Replace c() with an appropriate sequence.
  # Fill in.
}
```

Summation Series

Write a for loop to find the sum or show that the series has no sum:

$$1 + \frac{1}{9} + \frac{1}{81} + \ldots$$

Sine Function

Consider the following series that is used to approximate the cuntion $sin(x)$:

$$sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Write a function `sin_aprox()` that uses a `for` loop to approximate $sin(x)$ with the first N terms. Your function should be executed like this:

```r
sin_approx(x = 1, N = 25)
```

Solutions

```r
# functions f() and g()
f <- function(x) {
  x * x
}

g <- function(x) {
  (2 * x) + 5
}


# composite functions fog() and gof()
fog <- function(x) {
  f(g(x))
}

gof <- function(x) {
  g(f(x))
}


# gaussian function
gaussian <- function(x = 1, m = 0, s = 1) {
  constant <- 1 / (sqrt(2*pi))
  exponent <- exp(-0.5 * ((x - m) / s)^2)
  # output
  constant * (1/s) * exponent
}


# descriptive statistics function
descriptive <- function(x, na.rm = FALSE) {
  c(
    "min" = min(x, na.rm = na.rm),
    "q1" = quantile(x, probs = 0.25, na.rm = na.rm),
    "median" = median(x, na.rm = na.rm),
    "mean" = mean(x, na.rm = na.rm),
    "q3" = quantile(x, probs = 0.75, na.rm = na.rm),
    "max" = max(x, na.rm = na.rm),
    "range" = max(x, na.rm = na.rm) - min(x, na.rm = na.rm),
    "iqr" = IQR(x, na.rm = na.rm),
    "sd" = sd(x, na.rm = na.rm)
  )
}


# "squish" a number into the interval [0, 100]
z <- 100*pi

if (z < 0) {
  z <- 0
} else if (z > 100) {
```

```
  z <- 100
}
print(z)
```

```
## [1] 100
```

```
# test if a number is even
is_even <- function(x) {
  if (is.numeric(x)) {
    return(x %% 2 == 0)
  } else {
    return(NA)
  }
}


# test if a number is odd
is_odd <- function(x) {
  !is_even(x)
}


# switch vowels
letter <- "e" # Change this value!

switch(letter,
  a = ,
  e = ,
  i = ,
  o = ,
  u = "vowel",
  "not a vowel") # Modify as necessary.
```

```
## [1] "vowel"
```

```
# multiply a positive number by 3 and add 1 until the result is > 10000
n_rep <- 314 # Change this value!
repeat {
  n_rep <- (n_rep * 3) + 1
  print(n_rep)
  if (n_rep > 10000) break
}
```

```
## [1] 943
## [1] 2830
## [1] 8491
## [1] 25474
```

```
n_while <- 314 # Change this value!
while (n_while <= 10000) {
  n_while <- (n_while * 3) + 1
  print(n_while)
}
```

```
## [1] 943
## [1] 2830
## [1] 8491
## [1] 25474
```

```
# reduce() function
reduce <- function(x) {
  # Fill in.
  while(x %% 2 == 0) {
    x <- x / 2
  }
  return(x)
}

reduce(898128000)
```

```
## [1] 7016625
```

```
# summation of first series up to 20 terms
series1 <- 0

for (i in 0:20) {
  series1 <- series1 + (1 / (9^i))
}
series1
```

```
## [1] 1.125
```

```
# summation of second series, trying with various number of terms
series2 <- 0

# up to 10 terms
for (i in 0:10) {
  series2 <- series2 + (1 / (9^i))
}
series2
```

```
## [1] 1.125
```

```
# sine function
sine_approx <- function(x = 1, N = 25) {
  k = 1
  s = x
  sign = 1
  while (k < N) {
    sign = -1 * sign
    k = k + 2
    term = sign * (x^k) / factorial(k)
    s = s + term
  }
  return(s)
}

sine_approx(x = 1, N = 25)
```

```
## [1] 0.841471
```