Applied Statistical Programming - Spring 2022

Patrick Edwards

Problem Set 3

Due Wednesday, March 2, 10:00 AM (Before Class)

Instructions

1. The following questions should each be answered within an R script. Be sure to provide many comments in the script to facilitate grading. Undocumented code will not be graded.
2. Work on git. Fork the repository found at https://github.com/johnsontr/AppliedStatisticalProgramming2022 and add your code for Problem Set 3, committing and pushing frequently. Use meaningful commit messages because these will affect your grade.
3. You may work in teams, but each student should develop their own Rmarkdown file. To be clear, there should be no copy and paste. Each keystroke in the assignment should be your own.
4. For students new to programming, this may take a while. Get started.

Let's Make a Deal[1]

In the game show "Let's Make a Deal' ', the candidate gets to choose one of three closed doors, and receives the prize behind the door they choose. Behind one door is a new car; behind the other two doors are goats. After the contestant selects one of the 3 doors, the host opens one of the other two doors, and reveals a goat. Now, the candidate has the option of either sticking with the door they originally selected, or switching to the only other door that is still closed. What should the candidate do, and why? What are the probabilities of winning the car if they stay versus if they switch? This question is known as the Monty Hall Problem.

Your tasks

For this problem set, you will not solve the Monty Hall Problem, but you will have to code a slightly simplified version of the "Let's Make a Deal" game. More specifically, you will set up a new class, which contains information regarding the door a player chooses, and a method that simulates a modified version of the game. You will have to do this using the S3 class system. Here are the specific instructions:

1. Define a new class: `door`. Objects of this class simply take on one numeric value: 1, 2, or 3 – indicating which door a candidate chooses.
2. Create a method for `door` objects that is called `PlayGame`. This method is supposed to do the following:
    - take the numeric value that is stored in the `door` object,
    - draw a random number between 1 and 3 that presents the door behind which the car is hidden,

[1] https://en.wikipedia.org/wiki/Let's_Make_a_Deal

- compare the two numbers, and print a message congratulating a winning candidate that chose the correct door, or expressing sympathies for a losing candidate that chose the wrong door.

3. Write:
    - a construction function that allows the user to create a `door` object,
    - and a validation function that checks whether the value stored in `door` is actually an integer

ANSWERS:

Problem 1: Define a new class: `door`. Objects of this class simply take on one numeric value: 1, 2, or 3 – indicating which door a candidate chooses.

```
# (a). I create a list `choice` that consists of three objects 1, 2, & 3. I
# then assign the class `door` to this list. Crucially, this must work with
# elements that are integer and numeric. I create two objects with whole number
# elements of both integer and numeric classes.
choice1 <- 1
choice2 <- as.integer(1)
class(choice1) <- "door"
class(choice2) <- "door"

# (b). This object now has the class attribute `door`.
class(choice1)
```

```
## [1] "door"
```

```
class(choice2)
```

```
## [1] "door"
```

```
choice1
```

```
## [1] 1
## attr(,"class")
## [1] "door"
```

```
choice2
```

```
## [1] 1
## attr(,"class")
## [1] "door"
```

```
# COMPLETED.
```

Problem 2: Create a method for `door` objects that is called `PlayGame`. This method is supposed to do the following:

- take the numeric value that is stored in the `door` object,
- draw a random number between 1 and 3 that presents the door behind which the car is hidden,
- compare the two numbers, and print a message congratulating a winning candidate that chose the correct door, or expressing sympathies for a losing candidate that chose the wrong door.

```r
# (a). I create the method.
PlayGame <- function(x) {
    UseMethod("PlayGame")
}

PlayGame.door <- function(x) {
    winningDoor <- as.numeric(sample(x = 1:3, size = 1))
    if (x == winningDoor) {
        print("Congratulations! You are the owner of a BRAND. NEW. CAR!")
    }
    if (x != winningDoor) {
        print("How unfortunate, you got the goat. Our condolences :(")
    }
}

# (b). Check that this function works for `door` objects:
PlayGame(choice1)
```

```
## [1] "How unfortunate, you got the goat. Our condolences :("
```

```r
PlayGame(choice2)
```

```
## [1] "How unfortunate, you got the goat. Our condolences :("
```

```r
# (c). Check that this function does not work for non-`door` objects:
# PlayGame(unclass(choice)) Error in UseMethod('PlayGame') : no applicable
# method for 'PlayGame' applied to an object of class 'c('double', 'numeric')'

# COMPLETED.
```

Problem 3: Write:

- a construction function that allows the user to create a `door` object,
- and a validation function that checks whether the value stored in `door` is actually an integer

```r
# (ai). Make a construction function that allows the user to create a `door`
# object.
new_door <- function(choiceOfDoor = as.numeric(sample(x = 1:3, size = 1))) {
    x <- choiceOfDoor
    class(x) <- "door"
    return(x)
}
# note that the default value of choiceOfDoor is a random sample of the
# integers 1, 2, & 3. We can think of this as the game contestant randomly
# choosing between each door.

# (aii). Check that the construction function works for `choiceOfDoor inputs:
test1 <- new_door(2)
test1  # Interpretation: the game contestant chooses door 2.
```

```
## [1] 2
## attr(,"class")
## [1] "door"
```

```
test2 <- new_door()
test2  # Interpretation: the game contestant randomly chose between the three doors.
```

```
## [1] 2
## attr(,"class")
## [1] "door"
```

```
# Crucially, note that this constructor does not include validation checks: We
# can use nonsensical numbers. This is because its the validation function, not
# the construction function, that should check the validity of these values:
test3 <- new_door(5)
test3  # No sensible interpretation. Will be caught by the validity check.
```

```
## [1] 5
## attr(,"class")
## [1] "door"
```

```
# (bi). Make a validation function that checks whether the value in `door` is
# actually an integer.
validate_door <- function(x) {
    if (is.numeric(x[1]) == FALSE) {
        stop("ERROR: this object does not contain a numeric value!")
    } else if (x%%1 != 0) {
        stop("ERROR: this object does not contain the integers 1, 2, or 3!")
    } else if (x < 1 | x > 3) {
        stop("ERROR: this object does not contain the integers 1, 2, or 3!")
    }
    return(x)
}

# (bii). Check that the validation function works for `door` objects that
# contain elements with integer and numeric classes:
newObj <- validate_door(choice1)
newObj
```

```
## [1] 1
## attr(,"class")
## [1] "door"
```

```
newObj <- validate_door(choice2)
newObj
```

```
## [1] 1
## attr(,"class")
## [1] "door"
```

```r
# Both work (i.e., neither throws an error).

# (ci). Include the validation function in the original function.
PlayGame.door <- function(x) {
    x <- validate_door(x)
    winningDoor <- as.numeric(sample(x = 1:3, size = 1))
    if (x == winningDoor) {
        print("Congratulations! You are the owner of a BRAND. NEW. CAR!")
    } else if (x != winningDoor) {
        print("How unfortunate, you got the goat. Our condolences :(")
    }
}

# (cii). Use the constructor function to create valid & non-valid test objects.
# Then use these to test the new function that includes the validation
# function.
validObj1 <- new_door(1)
validObj2 <- new_door(3L)
validObj3 <- new_door()
invalidObj1 <- new_door(-5)
invalidObj2 <- new_door(1.5)
invalidObj3 <- new_door("Seashells")

PlayGame(validObj1)  # Works properly.
```

```
## [1] "Congratulations! You are the owner of a BRAND. NEW. CAR!"
```

```r
PlayGame(validObj2)  # Works properly.
```

```
## [1] "Congratulations! You are the owner of a BRAND. NEW. CAR!"
```

```r
PlayGame(validObj3)  # Works properly.
```

```
## [1] "Congratulations! You are the owner of a BRAND. NEW. CAR!"
```

```r
# PlayGame(invalidObj1) # Throws correct error.  PlayGame(invalidObj2) # Throws
# correct error.  PlayGame(invalidObj3) # Throws correct error.
```

FINISHED