NAME:

Control Flow

1. Write an if statment that takes an exam score like `score <- 93` and returns `"A"` if the score is greater than 90.

2. Write an if statement that takes a single integer `n` and prints "even" if `n` is divisible by 2, "odd" otherwise. Test your code with `n <- 7` and `n <- 16`. (*Hint 1: experiment with the modulo operator `%%`, e.g `3 %% 2`.)

3. Extend your previous answer to return letters grades of A, B, C, or "no pass" if the score is in the 90s, 80s, 70s, or less than 70, respectively.

4. On the lines provided below, complete the code to iterate through the vector `x` and print its contents. For this question, the iterator, `i`, should take as its values the character string elements of `x`. (*Hint: see Option 3 in the slides*)

```r
x <- c("apple", "banana", "cherry")
```

```r
for (i in _______) {

    _________(________)

}
```

5. Repeat the question but this time, the iterator, `i`, should take as its values the integers from 1 to 3. (*Hint: see Option 1 in the slides*)

```
for (i in _______) {

    _________(________)

}
```

6. Write a for loop that uses `x` and prints the following output:

```
Fruit 1: apple
Fruit 2: banana
Fruit 3: cherry
```

7. Using a for-loop (don't cheat with vectorization!), change each of the temperatures of `temps_c` one-by-one so that they are converted from degrees Celcius to Fahrenheit.

```
temps_c <- c(0, 10, 20, 30)
```

8. For-loops, like if-else statments, can be *nested*: inside the two for-loops below, your code has access to both of the iterators `i` and `j`. Fill in the code below to turn `m` from a matrix of 0s to a multiplication table showing the products of the integers 1, 2, 3 with one another.

```
m <- matrix(rep(0, 9), nrow = 3)

for (i in __________) {

    for (j in __________) {

        m[__________] <- __________
    }
}
```

Run the code in R to check your answer. As a cherry on top, make the row and column names of this multiplication table easier to read by reassigning their names with `rownames(m) <- ____` and `colnames(m) <- ____`.

9. *For fun*: Run the following code and view the result.

```r
plot(0, 0)

for(i in 1:100) {
  points(runif(1, min=0, max=1), runif(1, min=0, max=1))
}
```

`plot(0, 0)` makes a scatterplot with a single point at (0, 0). `points(x, y)` will add a point to your plot at (x, y) and `runif()` works like `rnorm()` except it generates 1 random uniform number between 0 and 1 instead of a random normal number (from a bell curve).

Modify this code so that your random points appear uniformly along the unit circle (a circle centered at (0, 0) with radius one) instead of uniformly in the upper right quadrant.

Lists

10. Create a list called `my_list` with the three following items. Name them `"char_vec"`, `"char_mat"`, and `"lst"`. Draw this list as a train with cargo in the space below.

```r
list_el_1 <- "crane"
list_el_2 <- matrix(c("a", "b", "c", "d"))
list_el_3 <- list(c(3.2, 5.7, 9), c(FALSE, TRUE))
```

For each of the following, write the R code to subset the list and then draw the corresponding train or cargo. You're welcome to subset by index, name, or logical; and use `[]`, `[[]]`, or `$`, as is appropriate.

11. The same list but without `char_mat`.

12. The character string `"crane"`.

13. The element `"d"`.

14. The element `"d"` repeated several times to form a character vector of length 4.

15. The double vector, as the sole element of a list.

16. The value `TRUE`.

Functions

17. Define a function called `f_to_c()` that converts temperature in Fahrenheit to Celcius.

18. Define a function that simulates the result of flipping a coin with sides "heads" and "tails". Generalize it to be able to simulate an *unfair* coin, i.e. a coin that does not land "heads" with probability 1/2. (see `?sample()`)

19. Define a function called `drop_n_lowest(x, n)` that takes two arguments: `x`, an atomic vector and `n`, the number of lowest elements to drop. It should return `x` with the lowest `n` elements removed. Test your function on a short numeric vector to ensure it works. (*Hint: use a function introduced at the end of the previous problem set*)

20. What are the pros and cons of defining this function with a default of `n = 1`?

21. Define a function called `check_water_temp()` that takes a temperature value and does the following:

1. Returns `"freezing"` if the supplied temp is below 32 degrees F.
2. Returns `"boiling"` if the supplied temp is above 212 degrees F[1].
3. Returns the same numerical temperature that was input if its between 32 and 212 F.
4. Allows the user to toggle between Fahrenheit or Celcius (but defaults to F) for the units of their input.

[1]Alternatively, you're welcome to look into how to return the hot face emoji (and its analog for freezing).