HOMEWORK 2

DUE FRI. SEP. 4

Submission: Submit your solutions to gradescope. Note that I am not providing code snippets here; you should read the instructions (carefully) to see what functions you need to write (if unclear, feel free to ask).

Exercises. Nothing to submit here.

E1 (no submission). Suppose I have the tuples

```
tup = (1, 2)
foo = ([1, 2], [3, 4])
```

Verify that you can’t change the contents of `tup` using `tup[0]=...` and so on, and that

```
tup = (1, 2)
a = tup[0]
a = 3
```

doesn’t change the first element of `tup`. Can the contents of `foo` change (without re-defining `foo`)?

E2 (a slicing example). The \( k \)-th principal minor \( A_k \) of an \( n \times n \) matrix \( A \) is the upper left \( k \times k \) submatrix. For instance,

\[
A = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}, \quad A_1 = [1], \quad A_2 = \begin{bmatrix}1 & 2 \end{bmatrix}.
\]

Write a function `set_minor(mat, k, new)` that takes in a square matrix `mat` and a \( k \times k \) matrix `new` and sets the \( k \)-th principal minor of `mat` equal to `new`, e.g.

\[
k = 2, \quad \text{mat} = A, \quad \text{new} = \begin{bmatrix} 11 & 12 \\
13 & 14 \end{bmatrix} \implies \text{mat} = \begin{bmatrix} 11 & 12 & 3 \\
13 & 14 & 6 \\
7 & 8 & 9 \end{bmatrix}
\]

This should modify `mat` and leave `new` unchanged. Do this using one for loop (over the rows of \( A \)) and slices to set each row.

Assume that `mat` is represented by a list of rows.

b) Can you avoid using for loops entirely and without creating any new lists?

*Hint:* What does `mat[0:2][0:2]` do? Be careful with this!

# example of slicing:
a = [0, 1, 2, 3, 4]
b = [10, 11, 12]
a[1:4] = b # now a is [0, 10, 11, 12, 4]
Programming problems.

Q1 (integration by random sampling). Here is a simple method for computing a definite integral
\[ I = \int_{a}^{b} f(x) \, dx \]
where \( f(x) \) is a positive function.

a) Write a function that estimates \( I \) as follows (Monte Carlo integration)
   
   i) Consider a rectangle \( R \) with sides along the \( x \) and \( y \) axes large enough to contain the area under the curve \((x, f(x))\) in the given interval.
   
   ii) Generate \( N \) uniformly distributed points\(^1\) \((x, y)\) in \( R \) (note: \( x \) and \( y \) can be drawn from independent uniform distributions separately).
   
   iii) Estimate \( I \) by assuming that the ratio of the number of points under the curve to \( N \) is the ratio of \( I \) to the area of the rectangle.

Note that the function \( f(x) \) should be an input.

b) Estimate \( \pi \) by using (b) on the integral
\[ \frac{\pi}{4} = \int_{0}^{1} f(x) \, dx \quad \text{where } f(x) = \sqrt{1-x^2}. \]
How many points \( N \) do you need to get to \( 3.14 \cdots \)?

c) Create a table of the error in the estimate for \( \pi \) vs. \( N \) for \( N = 1000, 2000, \cdots, 10000 \). Rather than calculate the error once for each \( N \), you should have your code do a fairly large number of trials and then average the result.

Your code should output this result when run (via ‘main’).

\(^1\)Consult the documentation at https://docs.python.org/3/library/random.html to figure out how to generate a uniformly distributed real number in an interval \([a, b] \).
Q2 (binary search). Suppose I have a sorted list of values

\[ a_0 \leq a_1 \leq \cdots \leq a_{n-1} \]

and I want to know if the value \( x \) is in the list. The binary search algorithm proceeds in the following way:

- First, check that \( x \) is between \( a_0 \) and \( a_{n-1} \)
- Start by setting \( \ell = 0 \) and \( r = n - 1 \) (left and right bounds)
- While not done:
  - Let \( c = (\ell + r)/2 \) (rounded) be the midpoint index.
  - if \( x \) is greater than \( a_c \), set \( \ell = c + 1 \)...
  - ...and if \( x \) is less than \( a_c \), (you figure this out)

The value \( x \), if it is in the list at all, must be between \( \ell \) and \( r \) (the ‘search interval’) at each step. The algorithm stops when the interval has a size of one.

a) (optional) Consider trying to find \( x = 1 \) in the list \([0,1,2,3,4,5]\). Write down, explicitly the steps taken by the algorithm. (This is a useful exercise when writing code - do a small case ‘by hand’ to both understand the process and have an example).

b) Write a function \textbf{search(vals, x)} that implements this algorithm. It should return the index of \( x \) if it is found, and either \(-1\) or \textit{None} if it is not found.

- Your algorithm, at each step, should print \([\ell, r]\).
- Try to keep the algorithm elegant by making the ‘base case’ (the step where the algorithm stops) as simple as possible.

- Style note: Don’t name the left bound \( 1 \) - it’s bad style (is it \( \ell \) or one?).

c) Write a ‘main’ that creates a list of 100 elements (the numbers 0 to 99 for simplicity) and searches for some value (your choice), so that it will show the steps taken by the algorithm.