Lab 2- Distributions, histograms, and Functional

Due: 09/13/2025 23:59PM

In this lab we will become familiar with distributions, histograms, and functional programming. Do not use numpy or any other library for this lab.

Assignment Submission Guidelines

Please follow the guidelines below for submitting your assignment:

1. **Submission Deadline:**
   - All assignments must be submitted **no later than 09/13/2025 11:59 PM**.
   - Late submissions will not be accepted unless prior arrangements have been made by the instructor.
2. **Submission Platform:**
   - Submit your assignment through **Canvas**. Ensure that you upload the files to the correct assignment link.
3. **Required Files:**
   - **Jupyter Notebook file (.ipynb):** Submit the Jupyter Notebook file you used to complete the assignment. The file should contain your well-commented code.
   - **PDF Version (.pdf file):** Additionally, submit a PDF version of your Python code. This can be a printout or export of your script, showing all the code with any necessary explanations or output results included.
4. **File Naming Convention:**
   - Please name your files as follows: `Lastname_Firstname_AssignmentName`
   - Example: `Alex_John_lab2.ipynb` and `Alex_John_lab2.pdf`
5. **Technical Issues:**
   - If you encounter any technical issues with Canvas or your submission, please contact the TAs immediately **before the deadline** to avoid penalties.

Uniform Distribution

Lets start with generating some fake random data. You can get a random number between 0 and 1 using the python random module as follow:

```python
import random
x=random.random()
print("The Value of x is", x)
```

```
The Value of x is 0.8740732708171308
```

Everytime you call random, you will get a new number.

Exercise 1: Using random, write a function `generate_uniform(N, mymin, mymax)`, that returns a python list containing N random numbers between specified minimum and maximum value. Note that you may want to quickly work out on paper how to turn numbers between 0 and 1 to between other values. (10 Points)

```python
# Skeleton
def generate_uniform(N,x_min,x_max):
    out = []
    ### BEGIN SOLUTION
    for num in range(N):
        out.append(random.randint(x_min, x_max))
    print(f"Generated {N} Random numbers!")

    ### END SOLUTION
    return out

# Test your solution here
data=generate_uniform(1000,-10,10)
print ("Data Type:", type(data))
print ("Data Length:", len(data))
if len(data)>0:
    print ("Type of Data Contents:", type(data[0]))
    print ("Data Minimum:", min(data))
    print ("Data Maximum:", max(data))
```

```
Generated 1000 Random numbers!
Data Type: <class 'list'>
Data Length: 1000
Type of Data Contents: <class 'int'>
Data Minimum: -10
Data Maximum: 10
```

Exercise 2a: Write a function that computes the mean of values in a list. Recall the equation for the mean of a random variable $\mathbf{x}$ computed on a data set of n values $\{x_i\} = \{x_1, x_2, ..., x_n\}$ is $\bar{\mathbf{x}} = \frac{1}{n}\sum_i^n x_i$. (10 Points)

```python
# Skeleton
def mean(Data):
    m=0.

    ### BEGIN SOLUTION
    summ=0
    n=0
    for x in Data:
        n+=1
        summ+=x
    m=summ/n
```

```python
    ### END SOLUTION

    return m

# Test your solution here
print ("Mean of Data:", mean(data))
```

```
Mean of Data: 0.365
```

Exercise 2b: Write a function that computes the variance of values in a list. Recall the equation for the variance of a random variable **x** computed on a data set of n values $\{x_i\} = \{x_1, x_2, ..., x_n\}$ is $\langle \mathbf{x} \rangle = \frac{1}{n} \sum_i^n (x_i - \bar{\mathbf{x}})$. (10 Points)

```python
# Skeleton
def variance(Data):
    m=0.

    ### BEGIN SOLUTION

    # For computing the mean
    summ_for_mean=0
    n=0
    for x in Data:
        summ_for_mean+=x
        n+=1
    mean_for_data=summ_for_mean/n

    # For variance
    summ_for_variance=0
    for x in Data:
        summ_for_variance+=(x-mean_for_data)
    variance = summ_for_variance/n
    m=variance
    ### END SOLUTION

    return m

# Test your solution here
print ("Variance of Data:", variance(data))
```

```
Variance of Data: 3.022027073029676e-16
```

Histogramming

Exercise 3: Write a function that bins the data so that you can create a histogram. An example of how to implement histogramming is the following logic:

- User inputs a list of values `x` and optionally `n_bins` which defaults to 10.

- If not supplied, find the minimum and maximum (`x_min`,`x_max`) of the values in x.
- Determine the bin size (`bin_size`) by dividing the range of the function by the number of bins.
- Create an empty list of zeros of size `n_bins`, call it `hist`.
- Loop over the values in `x`
  - Loop over the values in `hist` with index `i`:
    - If x is between `x_min+i*bin_size` and `x_min+(i+1)*bin_size`, increment `hist[i]`.
    - For efficiency, try to use continue to goto the next bin and data point.
- Return `hist` and the list corresponding of the bin edges (i.e. of `x_min+i*bin_size`). (10 Points)

```
# Solution
def histogram(x,n_bins=10,x_min=None,x_max=None):
    ### BEGIN SOLUTION

    # Finding the min/max if user does not supply.
    if x_min==None and x_max==None:
        x_min=min(x)
        x_max=max(x)

    # Getting the range size.
    range_vals = (x_max - x_min)

    # Calculating the bin_size
    bin_size = (range_vals / n_bins)

    # Creating the empty list of size `n_bins` and `bin_edges`
    hist = [0]*n_bins


    # Create bin_edges =======================================
    bin_edges=[]
    for i in range(n_bins+1):
        bound = x_min+i*bin_size
        bin_edges.append(bound)

    # Looping over the values in our data set =======================================
    for value in x:
        if value < x_min or value > x_max:
            continue
        binn = int((value - x_min) / bin_size)
        if binn == n_bins:
            binn-=1
```

```
        # Add bin edges until iteration is complete
        hist[binn]+=1
    ### END SOLUTION

    return hist,bin_edges
```

```
# Test your solution here
data=generate_uniform(1000,-10,10)
h,b=histogram(data,100)
print(b)
```

```
Generated 1000 Random numbers!
[-10.0, -9.8, -9.6, -9.4, -9.2, -9.0, -8.8, -8.6, -8.4, -8.2, -8.0, -7.8, -7.6, -7.4, -7.199
```

Exercise 4: Write a function that uses the histogram function in the previous exercise to create a text-based "graph". For example the output could look like the following:

```
[  0,   1] : ######
[  1,   2] : #####
[  2,   3] : ######
[  3,   4] : ####
[  4,   5] : ####
[  5,   6] : ######
[  6,   7] : #####
[  7,   8] : ######
[  8,   9] : ####
[  9,  10] : #####
```

Where each line corresponds to a bin and the number of `#`'s are proportional to the value of the data in the bin. (10 Points)

```
# Solution
def draw_histogram(x,n_bins,x_min=None,x_max=None,character="#",max_character_per_line=20):
    ### BEGIN SOLUTION

    # Load the histogram and bin_edge data
    hist,bin_edges=histogram(x,20)

    # for every bin =======================================
    for step in range(n_bins):
        # (Stop | End | Step)
        bin_text_placeholder=[]
        bin_text_placeholder.append([f"{step}", f"{step+1}"])



        marginal_dist = round((hist[step] + hist[step+1]))
```

```
        # Create the strings that represent our bars.
        hashtags = character*marginal_dist
        if len(hashtags) > max_character_per_line:
            hashtags=character*round(len(hashtags)/max_character_per_line)
            if step < n_bins-1:
                print(f"{bin_text_placeholder}  : {hashtags}")
            else:
                print(f"{bin_text_placeholder} : {hashtags}")
        else:
            print(f"{bin_text_placeholder}:{hashtags}")


    ### END SOLUTION

    return hist,bin_edges
```

```
# Test your solution here
data=generate_uniform(1000,-10,10)
print(draw_histogram(data, 10))
```

```
Generated 1000 Random numbers!
[['0', '1']]  : #####
[['1', '2']]  : ######
[['2', '3']]  : #####
[['3', '4']]  : ####
[['4', '5']]  : #####
[['5', '6']]  : #####
[['6', '7']]  : #####
[['7', '8']]  : #####
[['8', '9']]  : ####
[['9', '10']] : ####
([43, 57, 56, 46, 40, 51, 46, 52, 39, 39, 42, 39, 55, 37, 57, 49, 46, 54, 58, 94], [-10.0, -
```

Functional Programming

Exercise 5: Write a function the applies a booling function (that returns true/false) to every element in data, and return a list of indices of elements where the result was true. Use this function to find the indices of entries greater than 0.5. (10 Points)

```
def where(mylist,myfunc=None):
    out= []

    ### BEGIN SOLUTION
    if myfunc is None:
        # Returns False underhead if the function is not True.
        myfunc = lambda x: x>0.5
```

```
    # f(x) = x
    for x in mylist:
        if myfunc(x):
            answer = True
        else:
            answer = False
        out.append([x, answer])


    ### END SOLUTION

    return out

# Test your solution here
where(data)
```

```
[[7, True],
 [0, False],
 [9, True],
 [2, True],
 [5, True],
 [8, True],
 [0, False],
 [-5, False],
 [-6, False],
 [7, True],
 [-6, False],
 [-8, False],
 [-4, False],
 [8, True],
 [-3, False],
 [1, True],
 [-7, False],
 [-4, False],
 [3, True],
 [4, True],
 [-7, False],
 [4, True],
 [-10, False],
 [10, True],
 [5, True],
 [-7, False],
 [-10, False],
 [-6, False],
 [-4, False],
 [1, True],
```

```
[6, True],
[-6, False],
[2, True],
[6, True],
[-10, False],
[7, True],
[6, True],
[-6, False],
[3, True],
[4, True],
[-9, False],
[10, True],
[-3, False],
[8, True],
[4, True],
[-10, False],
[2, True],
[7, True],
[7, True],
[-3, False],
[-1, False],
[-1, False],
[3, True],
[2, True],
[5, True],
[10, True],
[-4, False],
[2, True],
[-5, False],
[-1, False],
[8, True],
[0, False],
[-3, False],
[-10, False],
[-10, False],
[-5, False],
[5, True],
[4, True],
[5, True],
[8, True],
[-8, False],
[1, True],
[-2, False],
[-5, False],
[2, True],
[-7, False],
```

```
[2, True],
[4, True],
[-10, False],
[3, True],
[-9, False],
[5, True],
[10, True],
[-5, False],
[-3, False],
[-3, False],
[-9, False],
[3, True],
[4, True],
[-3, False],
[-5, False],
[4, True],
[7, True],
[-1, False],
[-8, False],
[0, False],
[2, True],
[3, True],
[2, True],
[0, False],
[-10, False],
[-2, False],
[-3, False],
[-10, False],
[5, True],
[-9, False],
[6, True],
[10, True],
[0, False],
[-10, False],
[3, True],
[-3, False],
[-10, False],
[4, True],
[-7, False],
[3, True],
[2, True],
[10, True],
[9, True],
[-5, False],
[9, True],
[10, True],
```

```
[-2, False],
[2, True],
[8, True],
[-5, False],
[-1, False],
[-4, False],
[-6, False],
[-7, False],
[5, True],
[8, True],
[-2, False],
[-2, False],
[-7, False],
[8, True],
[3, True],
[7, True],
[-10, False],
[4, True],
[-4, False],
[-3, False],
[9, True],
[2, True],
[-8, False],
[1, True],
[5, True],
[10, True],
[-4, False],
[-10, False],
[2, True],
[7, True],
[6, True],
[-6, False],
[2, True],
[4, True],
[-3, False],
[-1, False],
[-2, False],
[1, True],
[5, True],
[8, True],
[5, True],
[-9, False],
[6, True],
[6, True],
[9, True],
[2, True],
```

```
[-10, False],
[-9, False],
[6, True],
[10, True],
[-6, False],
[-2, False],
[-4, False],
[-2, False],
[10, True],
[0, False],
[-9, False],
[-6, False],
[-5, False],
[7, True],
[0, False],
[-7, False],
[8, True],
[-1, False],
[5, True],
[5, True],
[9, True],
[-2, False],
[-10, False],
[-2, False],
[-3, False],
[3, True],
[-7, False],
[-8, False],
[5, True],
[-1, False],
[4, True],
[-9, False],
[10, True],
[-7, False],
[1, True],
[10, True],
[0, False],
[-10, False],
[8, True],
[6, True],
[-1, False],
[-6, False],
[-7, False],
[-2, False],
[-7, False],
[-9, False],
```

```
[-7, False],
[6, True],
[10, True],
[-9, False],
[3, True],
[2, True],
[7, True],
[2, True],
[5, True],
[7, True],
[-4, False],
[-4, False],
[2, True],
[4, True],
[6, True],
[-7, False],
[4, True],
[-5, False],
[-10, False],
[-3, False],
[-2, False],
[4, True],
[8, True],
[-1, False],
[-2, False],
[0, False],
[-4, False],
[3, True],
[-5, False],
[-3, False],
[7, True],
[-7, False],
[-4, False],
[5, True],
[9, True],
[-7, False],
[-5, False],
[-7, False],
[9, True],
[6, True],
[0, False],
[-7, False],
[-6, False],
[1, True],
[-4, False],
[-8, False],
```

```
[-4, False],
[5, True],
[-8, False],
[7, True],
[-1, False],
[7, True],
[-7, False],
[-5, False],
[-2, False],
[-7, False],
[-4, False],
[-6, False],
[10, True],
[9, True],
[-9, False],
[5, True],
[9, True],
[-9, False],
[7, True],
[4, True],
[8, True],
[-10, False],
[1, True],
[-10, False],
[-5, False],
[5, True],
[-8, False],
[2, True],
[8, True],
[8, True],
[-6, False],
[-8, False],
[10, True],
[-5, False],
[-3, False],
[-5, False],
[10, True],
[-3, False],
[4, True],
[1, True],
[-1, False],
[2, True],
[8, True],
[-7, False],
[5, True],
[-5, False],
```

```
[7, True],
[7, True],
[9, True],
[6, True],
[-8, False],
[9, True],
[-5, False],
[-5, False],
[3, True],
[-4, False],
[-10, False],
[5, True],
[-9, False],
[-7, False],
[7, True],
[-7, False],
[2, True],
[-4, False],
[-9, False],
[9, True],
[-3, False],
[-3, False],
[2, True],
[-5, False],
[6, True],
[8, True],
[7, True],
[-1, False],
[-6, False],
[2, True],
[0, False],
[-7, False],
[-3, False],
[3, True],
[-5, False],
[8, True],
[-2, False],
[-9, False],
[-4, False],
[10, True],
[3, True],
[-9, False],
[-9, False],
[8, True],
[-8, False],
[-6, False],
```

```
[-4, False],
[1, True],
[-6, False],
[-4, False],
[-2, False],
[-8, False],
[-8, False],
[-5, False],
[4, True],
[8, True],
[-3, False],
[-7, False],
[-2, False],
[8, True],
[-3, False],
[-9, False],
[-8, False],
[-5, False],
[-7, False],
[-6, False],
[1, True],
[6, True],
[-10, False],
[2, True],
[-9, False],
[4, True],
[-9, False],
[7, True],
[-2, False],
[3, True],
[9, True],
[0, False],
[4, True],
[4, True],
[4, True],
[8, True],
[4, True],
[-1, False],
[-9, False],
[2, True],
[7, True],
[1, True],
[6, True],
[9, True],
[-6, False],
[1, True],
```

```
[7, True],
[3, True],
[10, True],
[-6, False],
[7, True],
[7, True],
[-2, False],
[-10, False],
[-10, False],
[-9, False],
[-8, False],
[-8, False],
[1, True],
[-9, False],
[5, True],
[-8, False],
[-4, False],
[9, True],
[7, True],
[9, True],
[4, True],
[-4, False],
[-6, False],
[-10, False],
[-8, False],
[8, True],
[-9, False],
[-3, False],
[9, True],
[-10, False],
[-8, False],
[-4, False],
[-2, False],
[-7, False],
[-4, False],
[8, True],
[6, True],
[-8, False],
[-8, False],
[-4, False],
[9, True],
[2, True],
[8, True],
[9, True],
[-7, False],
[3, True],
```

```
[-9, False],
[7, True],
[-4, False],
[4, True],
[9, True],
[10, True],
[-2, False],
[4, True],
[-2, False],
[-7, False],
[-9, False],
[-7, False],
[-4, False],
[8, True],
[7, True],
[3, True],
[-2, False],
[5, True],
[4, True],
[8, True],
[2, True],
[9, True],
[-1, False],
[2, True],
[3, True],
[-9, False],
[4, True],
[1, True],
[4, True],
[6, True],
[-4, False],
[1, True],
[-3, False],
[5, True],
[10, True],
[-3, False],
[2, True],
[-3, False],
[6, True],
[1, True],
[-9, False],
[-9, False],
[-1, False],
[-8, False],
[-5, False],
[9, True],
```

```
[-9, False],
[2, True],
[-2, False],
[7, True],
[-9, False],
[-10, False],
[10, True],
[5, True],
[4, True],
[5, True],
[-6, False],
[8, True],
[10, True],
[-5, False],
[-10, False],
[3, True],
[-5, False],
[-9, False],
[2, True],
[-1, False],
[1, True],
[-1, False],
[6, True],
[7, True],
[-1, False],
[-6, False],
[5, True],
[-7, False],
[-5, False],
[-4, False],
[10, True],
[-1, False],
[-1, False],
[-1, False],
[2, True],
[2, True],
[-3, False],
[-8, False],
[5, True],
[4, True],
[-1, False],
[9, True],
[5, True],
[6, True],
[-10, False],
[5, True],
```

```
[2, True],
[0, False],
[-2, False],
[6, True],
[7, True],
[0, False],
[-5, False],
[-8, False],
[-8, False],
[5, True],
[-3, False],
[-3, False],
[4, True],
[0, False],
[-9, False],
[9, True],
[5, True],
[0, False],
[-9, False],
[1, True],
[-1, False],
[2, True],
[-7, False],
[-3, False],
[-8, False],
[1, True],
[-4, False],
[-8, False],
[2, True],
[-5, False],
[4, True],
[-9, False],
[-3, False],
[-8, False],
[-6, False],
[5, True],
[1, True],
[10, True],
[2, True],
[1, True],
[0, False],
[-9, False],
[-3, False],
[-6, False],
[6, True],
[5, True],
```

```
[-5, False],
[0, False],
[8, True],
[0, False],
[4, True],
[2, True],
[0, False],
[10, True],
[-9, False],
[1, True],
[8, True],
[-1, False],
[-3, False],
[-6, False],
[6, True],
[-2, False],
[-5, False],
[-8, False],
[2, True],
[2, True],
[2, True],
[8, True],
[-1, False],
[9, True],
[0, False],
[-2, False],
[2, True],
[8, True],
[-8, False],
[8, True],
[9, True],
[7, True],
[-9, False],
[-5, False],
[-6, False],
[10, True],
[-7, False],
[-4, False],
[6, True],
[0, False],
[-4, False],
[2, True],
[6, True],
[-8, False],
[5, True],
[6, True],
```

```
[-3, False],
[1, True],
[0, False],
[6, True],
[0, False],
[-3, False],
[0, False],
[4, True],
[7, True],
[-6, False],
[-7, False],
[-4, False],
[9, True],
[4, True],
[7, True],
[5, True],
[1, True],
[9, True],
[6, True],
[-3, False],
[0, False],
[-4, False],
[-6, False],
[9, True],
[9, True],
[-3, False],
[1, True],
[-5, False],
[7, True],
[-10, False],
[3, True],
[0, False],
[8, True],
[2, True],
[-2, False],
[-9, False],
[-8, False],
[3, True],
[10, True],
[10, True],
[-7, False],
[8, True],
[8, True],
[7, True],
[-8, False],
[1, True],
```

```
[6, True],
[1, True],
[2, True],
[5, True],
[-5, False],
[5, True],
[0, False],
[9, True],
[-10, False],
[-6, False],
[10, True],
[-4, False],
[8, True],
[0, False],
[10, True],
[-1, False],
[2, True],
[0, False],
[9, True],
[4, True],
[9, True],
[8, True],
[-6, False],
[2, True],
[-2, False],
[-8, False],
[9, True],
[-5, False],
[3, True],
[10, True],
[7, True],
[-8, False],
[6, True],
[5, True],
[-8, False],
[-7, False],
[-4, False],
[8, True],
[-9, False],
[4, True],
[-3, False],
[8, True],
[-8, False],
[-7, False],
[-10, False],
[-3, False],
```

```
[-9, False],
[3, True],
[3, True],
[-8, False],
[-9, False],
[6, True],
[-3, False],
[3, True],
[4, True],
[-7, False],
[-3, False],
[-5, False],
[-3, False],
[-9, False],
[-8, False],
[-1, False],
[-9, False],
[-8, False],
[3, True],
[-4, False],
[1, True],
[10, True],
[-1, False],
[-2, False],
[-4, False],
[1, True],
[-3, False],
[3, True],
[-1, False],
[-5, False],
[-10, False],
[4, True],
[10, True],
[-5, False],
[3, True],
[-5, False],
[-4, False],
[-2, False],
[-3, False],
[0, False],
[3, True],
[-5, False],
[10, True],
[8, True],
[5, True],
[5, True],
```

```
[-8, False],
[-3, False],
[-9, False],
[4, True],
[-3, False],
[-5, False],
[-9, False],
[0, False],
[-2, False],
[-1, False],
[-7, False],
[6, True],
[-8, False],
[-7, False],
[9, True],
[8, True],
[5, True],
[-1, False],
[-9, False],
[-1, False],
[6, True],
[-7, False],
[-3, False],
[9, True],
[1, True],
[7, True],
[-10, False],
[1, True],
[-5, False],
[7, True],
[7, True],
[-4, False],
[8, True],
[-8, False],
[-7, False],
[6, True],
[-2, False],
[-10, False],
[7, True],
[-4, False],
[4, True],
[-9, False],
[0, False],
[-8, False],
[7, True],
[-4, False],
```

```
[6, True],
[-3, False],
[-6, False],
[-5, False],
[0, False],
[-9, False],
[1, True],
[7, True],
[9, True],
[5, True],
[-3, False],
[9, True],
[-6, False],
[7, True],
[-9, False],
[6, True],
[0, False],
[-8, False],
[-8, False],
[6, True],
[9, True],
[8, True],
[-6, False],
[-10, False],
[-8, False],
[-10, False],
[2, True],
[-7, False],
[-8, False],
[-9, False],
[-6, False],
[10, True],
[-8, False],
[4, True],
[7, True],
[0, False],
[1, True],
[2, True],
[-2, False],
[4, True],
[4, True],
[6, True],
[2, True],
[2, True],
[-8, False],
[0, False],
```

```
[6, True],
[6, True],
[9, True],
[9, True],
[6, True],
[10, True],
[7, True],
[5, True],
[10, True],
[0, False],
[9, True],
[10, True],
[-6, False],
[-9, False],
[10, True],
[-2, False],
[2, True],
[3, True],
[3, True],
[-5, False],
[0, False],
[-2, False],
[-8, False],
[-9, False],
[4, True],
[1, True],
[7, True],
[4, True],
[7, True],
[0, False],
[5, True],
[4, True],
[9, True],
[8, True],
[4, True],
[7, True],
[-6, False],
[-4, False],
[9, True],
[-4, False],
[-1, False],
[-3, False],
[8, True],
[10, True],
[4, True],
[6, True],
```

```
[2, True],
[-6, False],
[-7, False],
[5, True],
[8, True],
[7, True],
[8, True],
[-6, False],
[5, True],
[-10, False],
[-10, False],
[7, True],
[8, True],
[9, True],
[5, True],
[8, True],
[-5, False],
[-10, False],
[-5, False],
[-8, False],
[9, True],
[-1, False],
[-4, False],
[8, True],
[3, True],
[9, True],
[4, True],
[-8, False],
[7, True],
[-3, False],
[4, True],
[1, True],
[2, True],
[8, True],
[9, True],
[-3, False],
[0, False],
[7, True],
[-4, False],
[3, True],
[-9, False],
[-10, False],
[1, True],
[-10, False],
[4, True],
[-2, False],
```

```
[7, True],
[-1, False],
[10, True],
[-1, False],
[-3, False],
[7, True],
[8, True],
[-1, False],
[-7, False],
[-6, False],
[-5, False],
[3, True],
[-9, False],
[2, True],
[4, True],
[4, True],
[-2, False],
[-5, False],
[-6, False],
[8, True],
[4, True],
[-8, False],
[6, True],
[4, True],
[-5, False],
[5, True],
[-9, False],
[-9, False],
[-8, False],
[6, True],
[-5, False],
[-8, False],
[1, True],
[-5, False],
[10, True],
[-7, False],
[3, True],
[-10, False],
[8, True],
[-1, False],
[10, True],
[8, True],
[6, True],
[9, True],
[-10, False],
[-2, False],
```

```
 [8, True],
 [10, True],
 [1, True],
 [8, True]]
```

Exercise 6(a): The `inrange(mymin,mymax)` function below returns a function that tests if it's input is between the specified values. Write corresponding functions that test: * Even * Odd * Greater than * Less than * Equal * Divisible by (10 Points)

```python
#def in_range(mymin,mymax):
#    def testrange(x):
#        return x<mymax and x>=mymin
#    return testrange

# Examples:
#F1=inrange(0,10)
#F2=inrange(10,20)

# Test of in_range
#print (F1(0), F1(1), F1(10), F1(15), F1(20))
#print (F2(0), F2(1), F2(10), F2(15), F2(20))

#print ("Number of Entries passing F1:", len(where(data,F1)))
#print ("Number of Entries passing F2:", len(where(data,F2)))

### BEGIN SOLUTION
def make_is_even():
    def is_even(x):
        return x % 2==0
    return is_even

def make_is_odd():
    def is_odd(x):
        return x % 2!=0
    return is_odd

def make_greater_than(threshold):
    def is_greater(value):
        return value > threshold
    return is_greater

def make_less_than(threshold):
    def is_less(value):
        return value < threshold
    return is_less
```

```python
def make_equal_to(target_value):
    def is_equal(value):
        return value == target_value
    return is_equal

def make_is_divisible(dividend):
    def is_divisible(divisor):
        return divisor % dividend == 0
    return is_divisible

### END SOLUTION
```

```python
# Test your solution

# returns a value, saves inside a variable

F1 = make_is_even()
even = F1(10)
print("Even: ", even)
F2 = make_is_odd()
odd = F2(12)
print("Odd: ", odd)
# returns 11 > 10
F3 = make_greater_than(10)
greater_than = F3(11)
print("Is greater than: ",greater_than)
F4 = make_less_than(10)
less_than = F4(9)
print("Is less than: ", less_than)
F5 = make_equal_to(10)
equal = F5(10)
print("Is equal: ", equal)
F6 = make_is_divisible(10)
is_divisible = F6(10)
print("Is divisible: ", is_divisible)
```

```
Even:  True
Odd:  False
Is greater than:  True
Is less than:  True
Is equal:  True
Is divisible:  True
```

Exercise 6(b): Repeat the previous exercise using `lambda` and the built-in python functions sum and map instead of your solution above. (10 Points)

```python
def make_is_even():
    def is_even(x):
```

```python
        return x % 2==0
    return is_even

def make_is_odd():
    def is_odd(x):
        return x % 2!=0
    return is_odd

def make_greater_than(threshold):
    def is_greater(value):
        return value > threshold
    return is_greater

def make_less_than(threshold):
    def is_less(value):
        return value < threshold
    return is_less

def make_equal_to(target_value):
    def is_equal(value):
        return value == target_value
    return is_equal

def make_is_divisible(dividend):
    def is_divisible(divisor):
        return divisor % dividend == 0
    return is_divisible
### BEGIN SOLUTION
# All lambdas are in a variable or function, nested in `function_list`.
def function_list(a, number_list):
    # `a` is the number we want to test against
    is_even = lambda x: x % 2==0
    is_odd = lambda x: x % 2!=0
    def greater_than(a):
        return lambda x: x > a
    def less_than(a):
        return lambda x: x < a
    def equal_to(a):
        return lambda x: x == a
    def is_divisible(a):
        return lambda x: x % a == 0

    # Hard-coded the sum(map())'s
    Are_they_even = sum(map(is_even, num_list))
    Are_they_odd = sum(map(is_odd, num_list))
```

```
    Who_is_greater = sum(map(greater_than(a), num_list))
    Who_is_lesser = sum(map(less_than(a), num_list))
    Who_is_equal = sum(map(equal_to(a), num_list))
    Is_it_divisible = sum(map(is_divisible(a), num_list))

    # returns a dictionary
    return {
        "even numbers": Are_they_even,
        "odd numbers": Are_they_odd,
        f"greater than {a}": Who_is_greater,
        f"lesser than {a}": Who_is_lesser,
        f"equal to {a}": Who_is_equal,
        f"divisible by {a}": Is_it_divisible
    }
### END SOLUTION

# Test your solution
# Prints the freq of even numbers ===============================
num_list = [1,2,3,4,2]
result = function_list(2, num_list)
result
```

```
{'even numbers': 3,
 'odd numbers': 2,
 'greater than 2': 2,
 'lesser than 2': 1,
 'equal to 2': 2,
 'divisible by 2': 3}
```

Monte Carlo

Exercise 7: Write a "generator" function called `generate_function(func,x_min,x_max,N)`, that instead of generating a flat distribution, generates a distribution with functional form coded in `func`. Note that `func` will always be > 0.

Use the test function below and your histogramming functions above to demonstrate that your generator is working properly.

Hint: A simple, but slow, solution is to a draw random number `test_x` within the specified range and another number `p` between the `min` and `max` of the function (which you will have to determine). If `p<=function(test_x)`, then place `test_x` on the output. If not, repeat the process, drawing two new numbers. Repeat until you have the specified number of generated numbers, `N`. For this problem, it's OK to determine the `min` and `max` by numerically sampling the function. (10 Points)

```
# Why not use a class
def generate_function(func,x_min,x_max,N=1000):
```

```
    out = list()
    ### BEGIN SOLUTION

    # Fill in your solution here

    ### END SOLUTION

    return out
```

```
# A test function
def test_func(x,a=1,b=1):
    return abs(a*x+b)
print("Did not solve.")
```

```
Did not solve.
```

Exercise 8: Use your function to generate 1000 numbers that are normal distributed, using the `gaussian` function below. Confirm the mean and variance of the data is close to the mean and variance you specify when building the Gaussian. Histogram the data. (10 Points)

```
import math

def gaussian(mean, sigma):
    def f(x):
        return math.exp(-((x-mean)**2)/(2*sigma**2))/math.sqrt(math.pi*sigma)
    return f

# Example Instantiation
g1=gaussian(0,1)
g2=gaussian(10,3)
```

Exercise 9: Combine your `generate_function`, `where`, and `in_range` functions above to create an integrate function. Use your integrate function to show that approximately 68% of Normal distribution is within one variance. (10 Points)

```
def integrate(func, x_min, x_max, n_points=1000):

    return integral
print("Did not solve.")
```

```
Did not solve.
```