LAB 4 SOLUTIONS

Task 1

In [1]:
```
import scipy.stats as sps
```

In [2]:
```
## Part (a)
sps.binom.pmf(20, 143, 0.153)
```

Out[2]:
```
0.08687059451566365
```

In [3]:
```
## Part (b)
sps.binom.pmf(40, 143, 0.153)
```

Out[3]:
```
4.347048512074074e-05
```

Note that this is our first time seeing the notation `e-05` ! This notation is actually Python's version of scientific notation; for example, `13e-05` means 13×10^{-5}. As such, our answer to part (b) is a very small number; a number so small that many calculators would simply (but incorrectly) round it down to zero!

Task 2

In [4]:
```
## Part (a)
sps.norm.cdf(2, 3, 0.5)
```

Out[4]:
```
0.022750131948179195
```

In [5]:
```
## Part (b)
1 - sps.norm.cdf(1, -2, 1)
```

Out[5]:
```
0.0013498980316301035
```

In [6]:
```
## Part (c)
sps.norm.cdf(1, 0, 1) - sps.norm.cdf(-1, 0, 1)
```

Out[6]:
```
0.6826894921370859
```

Task 3

In [7]:
```
## Part (b)
sps.uniform.cdf(0.1532, -1, 2)
```

```
Out[7]: 0.5766
```

Note the slightly peculiar way of writing this function call (which is why we had you look up the help file for the function first!) As the help file states, a call of `sps.uniform.cdf(x, loc, scale)` corresponds to the c.d.f. of the

$$\mathrm{Unif}(\texttt{loc},\ \texttt{loc} + \texttt{scale})$$

distribution. As such, if we want a distribution uniform on the interval $[-1,\ 1]$ we need to specify `loc = -1` and `scale = 2`. As a sanity check, we know the answer is supposed to be

$$\frac{1 + 0.1532}{2} = 0.5766$$

which is precisely what we obtained above.

Task 4

```
In [8]:  ## Part (a)
         sps.norm.ppf( 1 - (0.05 / 2) )

Out[8]:  1.959963984540054

In [9]:  ## alternate Part (a)
         -sps.norm.ppf(0.05 / 2)

Out[9]:  1.9599639845400545

In [10]: ## Part (b)
         sps.norm.ppf(1 - (0.18 / 2))

Out[10]: 1.3407550336902165

In [11]: ## alternate Part (b)
         -sps.norm.ppf(0.18 / 2)

Out[11]: 1.3407550336902165
```

Task 5

```
In [12]: ## Part (a)
         x = sps.uniform.rvs(loc = 2, scale = 8, size = 100)
         x[0:10]

Out[12]: array([3.3900404 , 3.56328423, 6.47603891, 5.09647864, 8.77757225,
                3.54197677, 9.99419249, 4.38126299, 7.21970248, 9.56304892])
```

In [13]:
```
## Part (b)
y = sps.norm.rvs(98.2, 2.4, size = 150)
y[0:10]
```

Out[13]:
```
array([ 99.16789823,  98.6511934 ,  96.84418743,  96.6473626 ,
        99.60886958,  96.12687115, 100.31386748, 102.53520057,
       102.24251547,  99.95650277])
```

Task 6

In [14]:
```
import numpy.random as npr
```

In [15]:
```
npr.choice([1, 2, 3, 4, 5, 6], size = 10)
```

Out[15]:
```
array([2, 5, 4, 3, 2, 2, 2, 6, 1, 4])
```

Task 7

In [16]:
```
npr.choice([1, 2, 3], size = 4)
```

Out[16]:
```
array([1, 1, 1, 1])
```

The outcome changes each time the cell is run.

In [17]:
```
npr.seed(15)
npr.choice([1, 2, 3], size = 4)
```

Out[17]:
```
array([1, 2, 1, 2])
```

The outcome no longer changes each time the cell is run.

Task 8

In [18]:
```
x = ['success', 'failure', 'failure', 'success', 'failure', 'failure', 'fail
```

In [19]:
```
for k in x:
    print(k == 'success')
```

```
True
False
False
True
False
False
False
True
```

Task 9

FIRST ITERATION	
Start of Iteration	• `k` : `'success'`
End of Iteration	• `k` : `'success'`
SECOND ITERATION	
Start of Iteration	• `k` : `'failure'`
End of Iteration	• `k` : `'failure'`
THIRD ITERATION	
Start of Iteration	• `k` : `'failure'`
End of Iteration	• `k` : `'failure'`
FOURTH ITERATION	
Start of Iteration	• `k` : `'success'`
End of Iteration	• `k` : `'success'`
FIFTH ITERATION	
Start of Iteration	• `k` : `'failure'`
End of Iteration	• `k` : `'failure'`
SIXTH ITERATION	
Start of Iteration	• `k` : `'failure'`
End of Iteration	• `k` : `'failure'`
SEVENTH ITERATION	
Start of Iteration	• `k` : `'failure'`

End of Iteration	- `k` : `'failure'`
EIGHTH ITERATION	
Start of Iteration	- `k` : `'success'`
End of Iteration	- `k` : `'success'`

Task 10

In [20]:

```python
count = 0

for k in x:
    if k == 'success':
        count += 1

count
```

Out[20]:

```
3
```

Task 11

In [21]:

```python
import numpy as np
```

In [22]:

```python
count = 0

for k in np.arange(0, len(x)):
    if x[k] == 'success':
        count += 1

count
```

Out[22]:

```
3
```

Task 12

In [23]:

```python
## using arange
np.arange(1, 2.1, 0.1)
```

Out[23]:

```
array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. ])
```

In [24]:

```python
## using linspace
np.linspace(1, 2, 11)
```

```
Out[24]: array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. ])
```