In [1]:
```
import time
import numpy as np
from scipy.stats import norm, beta
```

Question 0 - Markdown warmup

This is *question 0* for [problem set 1 (https://jbhender.github.io/Stats507/F21/ps/ps1.html)](https://jbhender.github.io/Stats507/F21/ps/ps1.html) of [Stats 507 (https://jbhender.github.io/Stats507/F21/)](https://jbhender.github.io/Stats507/F21/).

> Question 0 is about Markdown.

The next question is about the **Fibonnaci sequence**, $F_n = F_{n-2} + F_{n-1}$. In part $\mathbf{a}$ we will define a Python function `fib_rec()`.

Below is a …

Level 3 Header

Next, we can make a bulleted list:

- Item 1
  - detail 1
  - detail 2
- Item 2

Finally, we can make an enumerated list:

1. Item 1
2. Item 2
3. Item 3

Question 1 - Fibonnaci Sequence

a.

In [2]:
```
def fib_rec(n, a = 0, b = 1):
    """
    This function generates Fibonacci numbers by using recursive metho

    Parameters
    ----------
    n : int
        the nth Fibonacci numbers will be calculated.
    a : int, optional
        the first Fibonacci numbers F_0. The default is 0.
    b : int, optional
        the second Fibonacci numbers F_1. The default is 1.
```

```
    Returns
    -------
    the nth Fibonacci numbers.

    """
    if n == 0:
        return a
    elif n == 1:
        return b
    else:
        return fib_rec(n-1) + fib_rec(n-2)
```

In [3]:
```
print(fib_rec(7))
print(fib_rec(11))
print(fib_rec(13))
```

```
13
89
233
```

b.

In [4]:
```
def fib_for(n, a = 0, b = 1):
    """
    This function generates Fibonacci numbers by using for loop.

    Parameters
    ----------
    n : int
        the nth Fibonacci numbers will be calculated.
    a : int, optional
        the first Fibonacci numbers F_0. The default is 0.
    b : int, optional
        the second Fibonacci numbers F_1. The default is 1.

    Returns
    -------
    the nth Fibonacci numbers.

    """
    x = a
    y = b
    for k in range(n):
        z = x + y
        x = y
        y = z
    return x
```

In [5]:
```
print(fib_for(7))
print(fib_for(11))
print(fib_for(13))
```

```
13
89
233
```

c.

In [6]:
```
def fib_whl(n, a = 0, b = 1):
    """
    This function generates Fibonacci numbers by using while loop.

    Parameters
    ----------
    n : int
        the nth Fibonacci numbers will be calculated.
    a : int, optional
        the first Fibonacci numbers F_0. The default is 0.
    b : int, optional
        the second Fibonacci numbers F_1. The default is 1.

    Returns
    -------
    the nth Fibonacci numbers.

    """
    x = a
    y = b
    k = 0
    while k < n:
        z = x + y
        x = y
        y = z
        k += 1
    return x
```

In [7]:
```
print(fib_whl(7))
print(fib_whl(11))
print(fib_whl(13))
```

```
13
89
233
```

d.

In [8]:
```
def fib_rnd(n):
    """
    This function generates Fibonacci numbers by using the rounding me

    Parameters
    ----------
    n : int
        the nth Fibonacci numbers will be calculated.

    Returns
    -------
    the nth Fibonacci numbers.

    """
    phi = (1 + 5 ** .5) / 2
    return round(phi ** n / 5 ** .5)
```

In [9]:
```
print(fib_rnd(7))
print(fib_rnd(11))
print(fib_rnd(13))
```

```
13
89
233
```

e.

In [10]:
```
def fib_flr(n, a = 0, b = 1):
    """
    This function generates Fibonacci numbers by using the truncation

    Parameters
    ----------
    n : int
        the nth Fibonacci numbers will be calculated.
in
    Returns
    -------
    the nth Fibonacci numbers.

    """
    phi = (1 + 5 ** .5) / 2
    return int(phi ** n / 5 ** .5 + .5)
```

In [11]:
```
print(fib_flr(7))
print(fib_flr(11))
print(fib_flr(13))
```

```
13
89
233
```

f.

In [12]:
```
def calculate_time(func, n, r):
    """
    This function calculates the median running time of each functions

    Parameters
    ----------
    func : function
        the function name.
    n : int
        the nth Fibonacci numbers will be calculated.
    r : int
        repeated number of calculating the running time.

     Returns
     -------
     the median running time of each functions.

    """
    cal_time = []
    for i in range(r):
        start = time.perf_counter()
        func(n)
        end = time.perf_counter()
        cal_time.append(end - start)
    return round(np.mean(cal_time), 4)
```

n	fib_rec	fib_for	fib_whl	fib_rnd	fib_flr
15	0.0002	0.0	0.0	0.0	0.0
20	0.0024	0.0	0.0	0.0	0.0
25	0.0261	0.0	0.0	0.0	0.0
30	0.2826	0.0	0.0	0.0	0.0
35	3.1392	0.0	0.0	0.0	0.0

Question 2 - Pascal's Triangle

a.

In [13]:
```
def pascal_row(n):
    """
    This function compute the nth row of Pascal's triangle.

    Parameters
```

```
    ----------
    n : int
        compute the nth row of Pascal's triangle.

    Returns
    -------
    the nth row of Pascal's triangle.

    """
    x = [1]
    for i in range(n - 1):
        c_i = int(x[i] * (n - i - 1) / (i + 1))
        x.append(c_i)
    return x
```

b.

In [14]:
```
def print_pascal_trl(n):
    """
    This function prints the first n rows of Pascal's triangle.

    Parameters
    ----------
    n : int
        print the first n rows of Pascal's triangle.

    Returns
    -------
    None.

    """
    largest_elm = pascal_row(n)[n // 2]
    elm_width = len(str(largest_elm)) + 1
    trl_width = elm_width * n

    for i in range(n):
        x = pascal_row(i + 1)
        y = "".join([str(x[j]).center(elm_width) for j in range(i + 1)
        print(y.center(trl_width))
```

In [15]:
```
print_pascal_trl(11)
```

```
In [15]: print_pascal_tri(11)
```

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  70  56  28  8   1
  1   9   36  84 126 126  84  36  9   1
1   10  45 120 210 252 210 120  45  10  1
```

Question 3 - Statistics 101

a.

```
In [16]: def estimate_ci(data, level, output = "string"):
             """
             This function estimates a point and interval based on Normal theor

             Parameters
             ----------
             data : array
                 input data.
             level : int
                 confidence interval.
             output : string

             Returns
             -------
             string or dictionary.

             """
             mean = np.mean(data)
             std = np.std(data)
             z = norm.ppf(1 - (1 - level * .01) / 2) / (len(data) ** .5)
             est = round(mean, 3)
             lwr = round(mean - z * std, 3)
             upr = round(mean + z * std, 3)

             if output == "string":
                 return str(est) + '[' + str(level) + '%CI:(' + str(lwr) + ','
             elif output == None:
                 return {'est': est, 'lwr': lwr, 'upr': upr}
```

b.

```
In [17]: def binomial_ci(data, method, level, output = "string"):
             """
             This function estimates a point and interval using four different
```

```
    Parameters
    ----------
    data : array
        input data.
    method : string
        indicate which method will be used.
        "normal_appx" means normal approximation will be used.
        "clopper_pearson" means Clopper-Pearson interval will be compu
        "Jeffrey" means Jeffrey’s interval interval will be computed.
        "Agresti-Coull" means Agresti-Coull interval will be computed.
    level : int
        confidence interval.
    output : string

    Returns
    -------
    string or dictionary.

    """
    x = np.sum(data)
    n = len(data)
    p_hat = x / n
    alpha = 1 - level * .01

    if method == "normal_appx":
        est = round(p_hat, 3)
        if np.min([n * p_hat, n * (1 - p_hat)]) > 12:
            z = norm.ppf(1 - alpha / 2)
            lwr = round(p_hat - z * ((p_hat * (1 - p_hat) / n) ** .5),
            upr = round(p_hat + z * ((p_hat * (1 - p_hat) / n) ** .5),
        else:
            print("The condition is not satisfied.")
            return

    if method == "clopper_pearson":
        est = round(p_hat, 3)
        lwr = round(beta.ppf(alpha / 2, x, n - x + 1), 3)
        upr = round(beta.ppf(1 - alpha / 2, x + 1, n - x), 3)

    if method == "Jeffrey":
        est = round(p_hat, 3)
        lwr = round(np.max([0, beta.ppf(alpha / 2, x + 0.5, n - x + 0.
        upr = round(np.min([1, beta.ppf(1 - alpha / 2, x + 0.5, n - x

    if method == "Agresti_Coull":
        z = norm.ppf(1 - alpha / 2)
        n_tilde = n + z ** 2
        p_tilde = (x + z ** 2 / 2) / n_tilde
        est = round(p_tilde, 3)
        lwr = round(p_tilde - z * ((p_tilde * (1 - p_tilde) / n) ** .5
        upr = round(p_tilde + z * ((p_tilde * (1 - p_tilde) / n) ** .5

    if output == "string":
        return str(est) + '[' + str(level) + '%CI:(' + str(lwr) + ','
    elif output == None:
        return {'est': est, 'lwr': lwr, 'upr': upr}
```

c.

In [18]:
```
data = np.append(np.repeat(1, 42), np.repeat(0, 48))
```

confidence level	standard	normal approximation	clopper-pearson	Jeffrey's	Agresti-Coull
90%	0.467[90%CI: (0.38,0.553)]	0.467[90%CI: (0.38,0.553)]	0.467[90%CI: (0.376,0.559)]	0.467[90%CI: (0.382,0.553)]	0.468[90%CI: (0.381,0.554)]
95%	0.467[95%CI: (0.364,0.57)]	0.467[95%CI: (0.364,0.57)]	0.467[95%CI: (0.361,0.575)]	0.467[95%CI: (0.366,0.569)]	0.468[95%CI: (0.365,0.571)]
99%	0.467[99%CI: (0.331,0.602)]	0.467[99%CI: (0.331,0.602)]	0.467[99%CI: (0.331,0.606)]	0.467[99%CI: (0.336,0.601)]	0.469[99%CI: (0.333,0.604)]

The length of the different methods.

confidence level	standard	normal approximation	clopper-pearson	Jeffrey's	Agresti-Coull
90%	0.173	0.173	0.183	0.171	0.173
95%	0.206	0.206	0.214	0.203	0.206
99%	0.271	0.271	0.275	0.265	0.271

The Jeffrey’s intervals have smallest width.