Code:

```
#Principles of Estimation(problem c)
import numpy as np

np.random.seed(42)

n = 20
lam = 2
true_mean = 1 / lam

data = np.random.exponential(scale=true_mean, size=n)

# 2. Calculate the sample mean (X_bar)
sample_mean = np.mean(data)

# 3. Calculate the estimators T1 and T2
t1 = 1 / sample_mean
t2 = ((n - 1) / n) * t1

# Print results
print(f"Simulated Data Points:\n{data}\n")
print(f"Sample Mean (X_bar): {sample_mean:.4f}")
print(f"Estimator T1: {t1:.4f}")
print(f"Estimator T2: {t2:.4f}")
```

```
Simulated Data Points:
[0.23463404 1.50506072 0.65837285 0.45647128 0.08481244 0.08479815
 0.02991938 1.00561543 0.45954108 0.61562503 0.01039965 1.75177874
 0.89321477 0.11934381 0.10033949 0.10130571 0.18137686 0.37196392
 0.28276853 0.1721115 ]

Sample Mean (X_bar): 0.4560
Estimator T1: 2.1931
Estimator T2: 2.0835
```

```python
#question3: problem A2

import numpy as np
from scipy import stats

# 1. Set the parameters
n = 25                  # Sample size
mu = 5                  # True population mean
sigma_squared = 4       # True population variance
sigma = np.sqrt(sigma_squared)  # Standard deviation = 2 [cite: 32, 133]
confidence_level = 0.95

# 2. Simulate n = 25 samples from N(5, 4)
# We use a seed so you can reproduce the same results
np.random.seed(42)
samples = np.random.normal(mu, sigma, n)

# 3. Calculate the sample mean (x-bar)
sample_mean = np.mean(samples)

# 4. Calculate the Standard Error (SE)
# SE = sigma / sqrt(n) [cite: 54, 320]
standard_error = sigma / np.sqrt(n)

# 5. Find the z-score for 95% confidence (alpha = 0.05)
# This finds the point where 2.5% is in the tail [cite: 250, 351]
z_critical = stats.norm.ppf(1 - (1 - confidence_level) / 2)

# 6. Construct the Confidence Interval (CI)
# CI = x_bar +/- (z * SE) [cite: 50, 254]
margin_of_error = z_critical * standard_error
lower_bound = sample_mean - margin_of_error
upper_bound = sample_mean + margin_of_error

print(f"Sample Mean: {sample_mean:.4f}")
print(f"Standard Error: {standard_error:.4f}")
print(f"95% Confidence Interval: [{lower_bound:.4f}, {upper_bound:.4f}]")
```

```
Sample Mean: 4.6730
Standard Error: 0.4000
95% Confidence Interval: [3.8890, 5.4570]
```

```python
#Question 3 problemb3
import numpy as np
from scipy import stats

n = 20
mu = 10
sigma = 3 # sqrt(9)
confidence = 0.90

# Simulate samples
np.random.seed(42)
samples = np.random.normal(mu, sigma, n)

# Compute sample stats
x_bar = np.mean(samples)
s_hat_sq = np.var(samples, ddof=1) # Unbiased sample variance
s_hat = np.sqrt(s_hat_sq)

# Part B: Unknown Variance (t-distribution)
t_crit = stats.t.ppf(1 - (1 - confidence)/2, n - 1)
moe_t = t_crit * (s_hat / np.sqrt(n))
interval_t = (x_bar - moe_t, x_bar + moe_t)

# Part A comparison: Known Variance (Z-distribution)
z_crit = stats.norm.ppf(1 - (1 - confidence)/2)
moe_z = z_crit * (sigma / np.sqrt(n)) # Uses true sigma
interval_z = (x_bar - moe_z, x_bar + moe_z)

print(f"Sample Variance (s^2_n): {s_hat_sq:.4f}")
print(f"Unknown Variance (t) Interval: {interval_t}, Length:
{2*moe_t:.4f}")
print(f"Known Variance (Z) Interval: {interval_z}, Length: {2*moe_z:.4f}")
```

```
Sample Variance (s^2_n): 8.2949
Unknown Variance (t) Interval: (8.3725312980782, 10.599677333270826),
Length: 2.2271
```

```
Known Variance (Z) Interval: (8.382702958804341, 10.589505672544686),
Length: 2.2068
```

```python
#problem c3
import numpy as np
from scipy import stats

n = 100
lam = 2
mu_true = 1/lam
sigma_true = 1/lam  # For Exponential, std dev = 1/lambda
confidence = 0.95

# Simulate Exponential data
np.random.seed(42)
samples = np.random.exponential(1/lam, n)
x_bar = np.mean(samples)

# CLT Approximate CI
z_crit = stats.norm.ppf(0.975)
se = sigma_true / np.sqrt(n)
interval = (x_bar - z_crit * se, x_bar + z_crit * se)

print(f"True Mean: {mu_true}")
print(f"Sample Mean: {x_bar:.4f}")
print(f"95% Approximate CI: [{interval[0]:.4f}, {interval[1]:.4f}]")
```

```
True Mean: 0.5
Sample Mean: 0.4574
95% Approximate CI: [0.3594, 0.5554]
```

```python
#problemD2
n_b = 50
p_true = 0.6

# Simulate Bernoulli (0 or 1)
np.random.seed(42)
bernoulli_samples = np.random.binomial(1, p_true, n_b)
theta_hat = np.mean(bernoulli_samples)
```

```python
# Approximate 95% CI
se_b = np.sqrt(theta_hat * (1 - theta_hat) / n_b)
interval_b = (theta_hat - z_crit * se_b, theta_hat + z_crit * se_b)

print(f"Bernoulli CI: [{interval_b[0]:.4f}, {interval_b[1]:.4f}]")
print(f"Bernoulli Length: {2 * z_crit * se_b:.4f}")
```

```
Bernoulli CI: [0.5507, 0.8093]
Bernoulli Length: 0.2586
```