1) The confidence level in the context of χ^2 statistics refers to the probability that the data observed fits the model proposed, assuming the model is correct. For $\chi^2 = 16$ with the degree of freedom $\nu = 10$, I am going to use Python.

```python
import scipy.stats as stats

chi2_stat = 16
degrees_of_freedom = 10

confidence_level = 1 - stats.chi2.cdf(chi2_stat, degrees_of_freedom)

print(f"Confidence Level: {confidence_level * 100:.2f}%")
```

```
Confidence Level: 9.96%
```

```python
chi2_per_dof = 0.5
degrees_of_freedom_v2 = 30

alt_chi2_value = chi2_per_dof * degrees_of_freedom_v2

alt_confidence_level = 1 - stats.chi2.cdf(alt_chi2_value, degrees_of_freedom_v2)

print(f"Alternative Confidence Level: {alt_confidence_level * 100:.2f}%")
```

```
Alternative Confidence Level: 98.97%
```

2) To solve this question, again I use Python.

```python
import numpy as np

group1 = np.array([71, 59, 50, 51, 63, 60])
group2 = np.array([46, 41, 68, 53, 47, 49, 35, 49, 54, 64])


t_stat, p_value = stats.ttest_ind(group1, group2, equal_var=False)

print(f"T-statistic: {t_stat:.2f}")
print(f"P-value: {p_value:.4f}")

if p_value < 0.05:
    print("The two groups are distinguishable at the 95% confidence level.")
else:
    print("The two groups are NOT distinguishable at the 95% confidence level.")
```

```
T-statistic: 1.88
P-value: 0.0830
The two groups are NOT distinguishable at the 95% confidence level.
```

3) The Python code is applied by myself to solve this question.

```python
observed_events = np.array([0, 1, 2, 3, 4, 5, 6])
frequencies = np.array([320, 387, 189, 76, 22, 4, 2])
total_intervals = frequencies.sum()

mean_events = np.sum(observed_events * frequencies) / total_intervals

poisson_probs = stats.poisson.pmf(observed_events, mean_events)

expected_frequencies = poisson_probs * total_intervals
expected_frequencies *= total_intervals / expected_frequencies.sum()

chi2_stat, p_value = stats.chisquare(frequencies, expected_frequencies)

print(f"Estimated Mean (μ): {mean_events:.3f}")
print(f"Chi-squared Statistic: {chi2_stat:.3f}")
print(f"P-value: {p_value:.3f}")

if p_value < 0.05:
    print("The Poisson model does NOT provide a good fit to the data.")
else:
    print("The Poisson model provides a good fit to the data.")
```

```
Estimated Mean (μ): 1.113
Chi-squared Statistic: 4.125
P-value: 0.660
The Poisson model provides a good fit to the data.
```

4) We already know some values such as:

$$P(\text{Covid}) = \frac{36,4}{100,000} = 0.000364$$

$P(\text{Positive} \mid \text{Covid}) = 0.58$ $\qquad$ $P(\text{Negative} \mid \neg\text{Covid}) = 1 - P(\text{Positive} \mid \neg\text{Covid}) = 0.989$

$P(\text{Positive} \mid \neg\text{Covid}) = 0.011$ $\qquad$ $P(\neg\text{Covid}) = 1 - P(\text{Covid}) = 0.999636.$

Now, I am going to implement a Python code.

```python
P_COVID = 36.4 / 100000
P_not_COVID = 1 - P_COVID

sensitivity = 0.58  # P(Positive | COVID)
false_positive_rate = 0.011  # P(Positive | not COVID)
specificity = 1 - false_positive_rate  # P(Negative | not COVID)

P_Positive = (sensitivity * P_COVID) + (false_positive_rate * P_not_COVID)

P_COVID_given_Positive = (sensitivity * P_COVID) / P_Positive

P_Negative = (specificity * P_not_COVID) + ((1 - sensitivity) * P_COVID)

P_COVID_given_Negative = ((1 - sensitivity) * P_COVID) / P_Negative

print(f"Probability of COVID given Positive test: {P_COVID_given_Positive * 100:.2f}%")
print(f"Probability of COVID given Negative test: {P_COVID_given_Negative * 100:.4f}%")

✓ 0.0s
```

```
Probability of COVID given Positive test: 1.88%
Probability of COVID given Negative test: 0.0155%
```

5) The CDF of the desired power-law function is:

$$F(y) = \int_0^y P(y')\,dy'$$

The normalized probability function $\Rightarrow P(y) = (n+1)y^n$ for $0 \le y < 1$

↳ $\therefore F(y) = \int_0^y (n+1)(y')^n\,dy' = y^{n+1}$

↳ a uniformly distributed random variable $x \in U(0,1) \Rightarrow F(y) = x$

- $y^{n+1} = x \Rightarrow y = x^{\frac{1}{n+1}}$

6) To solve this question I implement a Python code.

```python
measured_flux = -0.4
sigma_flux = 0.5

true_flux = np.linspace(0, 5, 1000)

cdf_at_zero = stats.norm.cdf(0, loc=measured_flux, scale=sigma_flux)

normalization_factor = 1 - cdf_at_zero

cumulative_probability = (stats.norm.cdf(true_flux, loc=measured_flux, scale=sigma_flux) - cdf_at_zero) / normalization_factor

upper_limit_index = np.searchsorted(cumulative_probability, 0.95)
upper_limit_flux = true_flux[upper_limit_index]

rounded_upper_limit = round(upper_limit_flux, 3)

print(f"95% upper limit to the true flux: {rounded_upper_limit} mJy")
```

✓ 0.0s

```
95% upper limit to the true flux: 0.756 mJy
```

7)

PDF of an exponential distribution:

$$p(x|a) = \frac{1}{a} e^{-x/a} \quad , \text{ where } x > 0$$

Our goal is to find the MLE of 'a' under two priors:

1) Uniform prior: $p(a>0) =$ constant
2) Scale-free prior: $p(a>0) \propto \frac{1}{a}$

→ Likelihood function:

$$L(a) = \prod_{i=1}^{N} p(x_i|a) = \prod_{i=1}^{N} \frac{1}{a} e^{-x_i/a}$$

$$= \ln(L(a)) = -N \ln(a) - \frac{1}{a} \sum_{i=1}^{N} x_i$$

→ Uniform prior

$$p(a|x_1, x_2, \dots, x_n) \propto L(a)$$

- Maximizing $\ln(L(a))$

$$\frac{d}{da} \ln(L(a)) = -\frac{N}{a} + \frac{1}{a^2} \sum_{i=1}^{N} x_i = 0$$

$$a = \frac{1}{N} \sum_{i=1}^{N} x_i = \hat{a}$$

→ Scale-free prior

$$p(a \mid x_1, x_2, \dots, x_n) \propto \frac{1}{a} L(a) \propto \frac{1}{a^{N+1}} e^{-\frac{\sum_{i=1}^{N} x_i}{a}}$$

- log posterior:

$$\ln\left[p(a \mid x_1, x_2, \dots, x_n)\right] = -(N+1)\ln a - \frac{1}{a}\sum_{i=1}^{N} x_i$$

- Maximizing this and setting to 0.

$$\frac{d}{da}\ln\left[p(a \mid x_1, x_2, \dots, x_n)\right] = -\frac{N+1}{a} + \frac{1}{a^2}\sum_{i=1}^{N} x_i = 0$$

$$a = \frac{1}{N+1}\sum_{i=1}^{N} x_i = \hat{a}$$