Fourier transform has a plethora of applications ranging from electronics, image processing, MP3 players, to quantum mechanics, among others. Roughly, it is the process of analyzing the data in frequency space and inverting back to original sample space.

DFT,FFT,FCT:~# DFT uses trapezoidal rule to evaluate the continuous fourier integral. FFT boasts efficiency reducing $(1/2)N^2$ operations down to $\log_2 N$. FCT further eliminates periodicity assumption of the sample.

```
for k in range (N//2+1):                    # User-defined discrete fourier transform
    for n in range (N):                     function loop to gather fourier coefficients
        c[k] += y[n]*exp(-2j*pi*k*n/N)

c = rfft(y)                                 # Numpy provides fast fourier transform

scipy.fft.dct(y)                            # Scipy provides fast cosine transform
```

! Pitfall: If the sample values are generally complex, DFT needs to process N-1 samples and not (1/2)N+1 for even or (1/2)(N+1) for odd.

noise-filtering:~# To filter out noise, take the fourier transform of the sample, remove unnecessary signal frequency, then take the inverse fourier transform.

```
c = rfft(y)                                 # Program uses fast fourier transform to
for i in range (20, len(c)):                filter out all but the first 20 data points in
    c[i] = 0                                frequency space and inverting them back to
yf = irfft(c)                               sample space
```

! Pitfall: Sometimes, the target signal and noise may be indistinguishible from each other. As such, additional algorithms must be implemented.

image-deconvolution:~# Fourier transform can be used to unblur a blurred image by dividing the fourier transform of a blurred image with the fourier transform of a point spread function.

```
def point_spread(x, y) :                    #  Gaussian point spread function is commonly
  return np.exp(-(x**2+y**2)/(2*sigma**2))  used for many cameras

if abs(psf_ft[j,i]) < epsilon :             # When the point spread function gets too
     unblurred_fourier[j, i] = blur_ft[j,i] small, we get a dividing-by-zero error. Hence,
else:                                       we need to introduce some threshold value
     unblurred_fourier[j, i] = blur_ft[j,i] / epsilon for division given by the if-else
     (psf_ft[j, i])                         statement.
```

! Pitfall: Remember not only to invoke a threshold condition but to pick an appropriate threshold value. Increasing it will make the image blurry while decreasing it will make the image look like a mosaic.