Exercise 1

we are given complex numbers in Cartesian form as: [z = x + i y] where:

- (x) is the real part
- (y) is the imaginary part

To get the same number in the polar representation $z = |z|e^{i\varphi}$, we use:

$$r = |z| = \sqrt{x^2 + y^2}$$

$$\varphi = \tan^{-1}\left(\frac{y}{x}\right)$$

---

a) Multiplication

For ((1 - i)):

$$|z_1| = \sqrt{x^2 + y^2} = \sqrt{1^2 + (-1)^2} = \sqrt{2}$$

$$\varphi_1 = \tan^{-1}\left(\frac{y}{x}\right) = \tan^{-1}(-1) = -\frac{\pi}{4}$$

$$z_1 = \sqrt{2}\, e^{i(-\pi/4)}$$

For ((0 + 2i)):

$$|z_2| = \sqrt{0^2 + 2^2} = 2$$

Here we can't use our previous definition for the angle because $\tan^{-1}\left(\frac{2}{0}\right)$ is mathematically undefined.
so instead we use:

$$\cos\varphi = \frac{x}{r}, \qquad \sin\varphi = \frac{y}{r}$$

For (z = 2i):

$$\cos\varphi = \frac{x}{r} = \frac{0}{2} = 0, \qquad \sin\varphi = \frac{y}{r} = \frac{2}{2} = 1$$

which gives us:

$$\varphi_2 = \frac{\pi}{2}$$

$$z_2 = 2\, e^{i(\pi/2)}$$

The multiplication:

$$z_1 z_2 = \sqrt{2}\, e^{i(-\pi/4)}\, 2\, e^{i(\pi/2)}$$

$$z_1 z_2 = \left(\sqrt{2} \times 2\right) e^{i(-\pi/4+\pi/2)} = 2\sqrt{2}\, e^{i(\pi/4)}$$

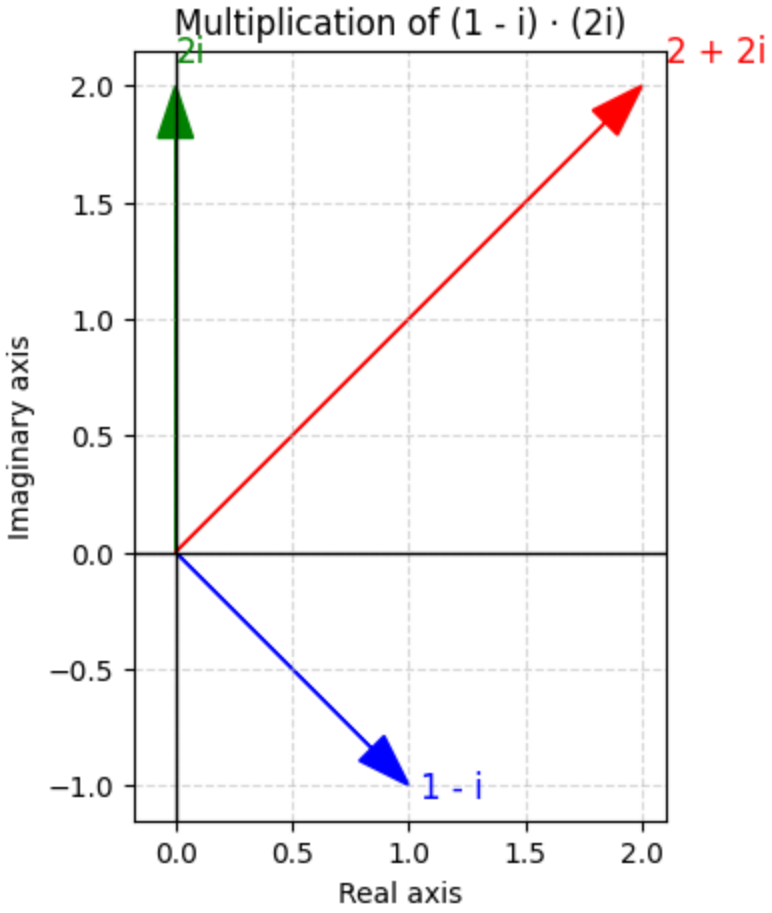
We also now convert back to cartesian form so we can use it for the plot:

$$z_1 z_2 = 2\sqrt{2}(\cos(\pi/4) + i\sin(\pi/4)) = 2\sqrt{2}\left(\frac{\sqrt{2}}{2} + i\frac{\sqrt{2}}{2}\right)$$

$$z_1 z_2 = 2 + 2i$$

From both the plot and the equations alone we can see that the magnitudes get multiplied by each other and the angles added together

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np

z1 = 1 - 1j
z2 = 0+2j
z_multi = z1 * z2

plt.figure(figsize=(5,5))
ax = plt.gca()
ax.axhline(0, color='black', lw=1)
ax.axvline(0, color='black', lw=1)
ax.set_aspect('equal')
ax.grid(True, linestyle='--', alpha=0.5)


for z, label, color in [(z1, "1 - i", "blue"), (z2, "2i", "green"), (z_multi
    plt.arrow(0, 0, z.real, z.imag, head_width=0.15, length_includes_head=Tr
    plt.text(z.real*1.05, z.imag*1.05, label, color=color, fontsize=12)

plt.xlabel("Real axis")
plt.ylabel("Imaginary axis")
plt.title("Multiplication of (1 - i) · (2i)")
plt.show()
```



b) Division

For ((0 + i)):

$$|z_1| = \sqrt{0^2 + 1^2} = 1$$

similar to the multiplication case for the number (0+2i) we find the angle to be:

$$\varphi_1 = \frac{\pi}{2}$$

$$z_1 = 1\, e^{i(\pi/2)}$$

For ((1 - \sqrt{3}i)):

$$|z_2| = \sqrt{1^2 + (-\sqrt{3})^2} = \sqrt{1+3} = 2$$

$$\varphi_2 = \tan^{-1}\left(\frac{-\sqrt{3}}{1}\right) = -\frac{\pi}{3}$$

$$z_2 = 2\, e^{i(-\pi/3)}$$

The division:

$$\frac{z_1}{z_2} = \frac{(1\,e^{i(\pi/2)})}{(2\,e^{i(-\pi/3)})}$$

$$\frac{z_1}{z_2} = \frac{1}{2}\,e^{i\left(\frac{\pi}{2}-(-\pi/3)\right)} = \frac{1}{2}\,e^{i(5\pi/6)}$$

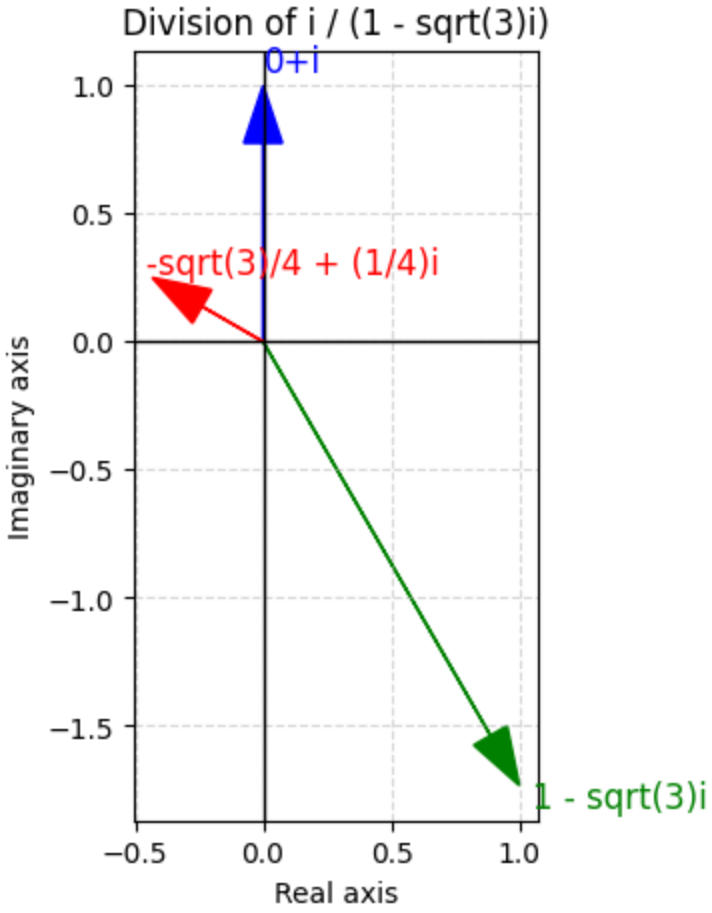
Cartesian form:

$$\frac{z_1}{z_2} = \frac{1}{2}\left(\cos\frac{5\pi}{6} + i\sin\frac{5\pi}{6}\right)$$

$$\frac{z_1}{z_2} = -\frac{\sqrt{3}}{4} + \frac{i}{4}$$

From both the plot and the equations alone we can see that the magnitudes get divided by each other and the angles subtracted from eachother.

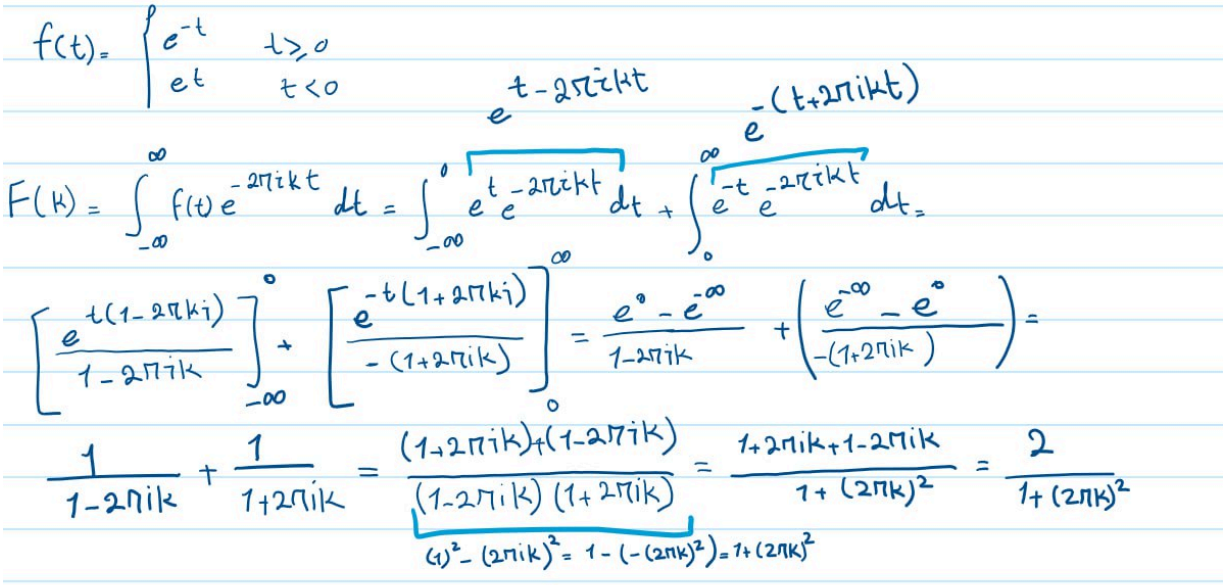
In [2]:
```
z1 = 1j
z2 = 1 - np.sqrt(3)*1j
z_div = z1 / z2

plt.figure(figsize=(5,5))
ax = plt.gca()
ax.axhline(0, color='black', lw=1)
ax.axvline(0, color='black', lw=1)
ax.set_aspect('equal')
ax.grid(True, linestyle='--', alpha=0.5)


for z, label, color in [(z1, "0+i", "blue"), (z2, "1 - sqrt(3)i", "green"),
    plt.arrow(0, 0, z.real, z.imag, head_width=0.15, length_includes_head=Tr
    plt.text(z.real*1.05, z.imag*1.05, label, color=color, fontsize=12)

plt.xlabel("Real axis")
plt.ylabel("Imaginary axis")
plt.title("Division of i / (1 - sqrt(3)i)")
plt.show()
```



Exercise 2

We solved this by hand so we just loaded a screenshot of our solution here so that everything is in one place.

In [3]:
```
from PIL import Image
from IPython.display import display
img = Image.open('Assignment01_EX02.jpeg')
display(img)
```



Exercise 3

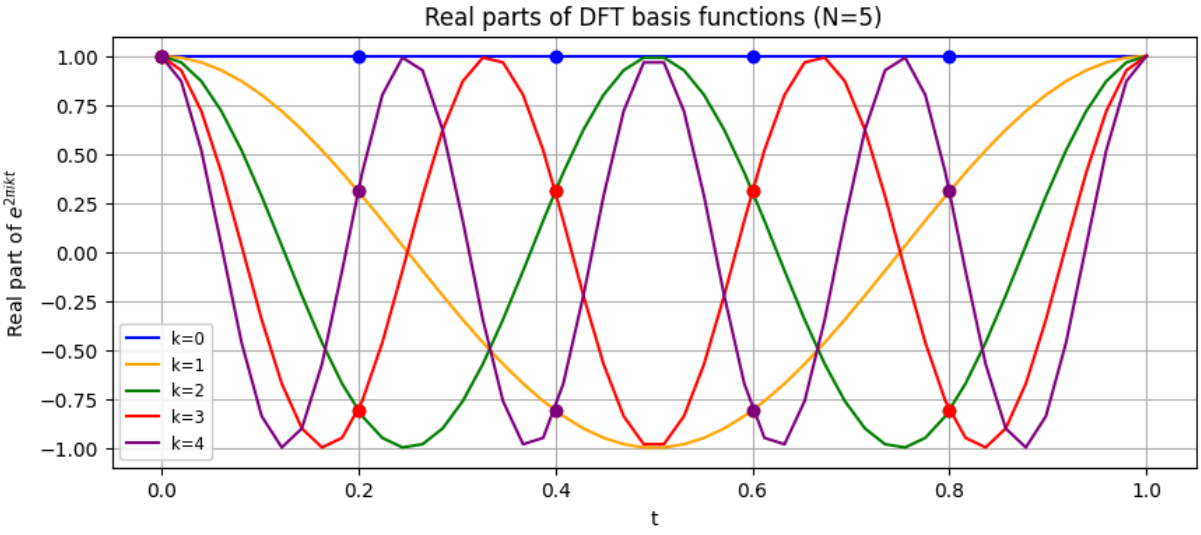
a)

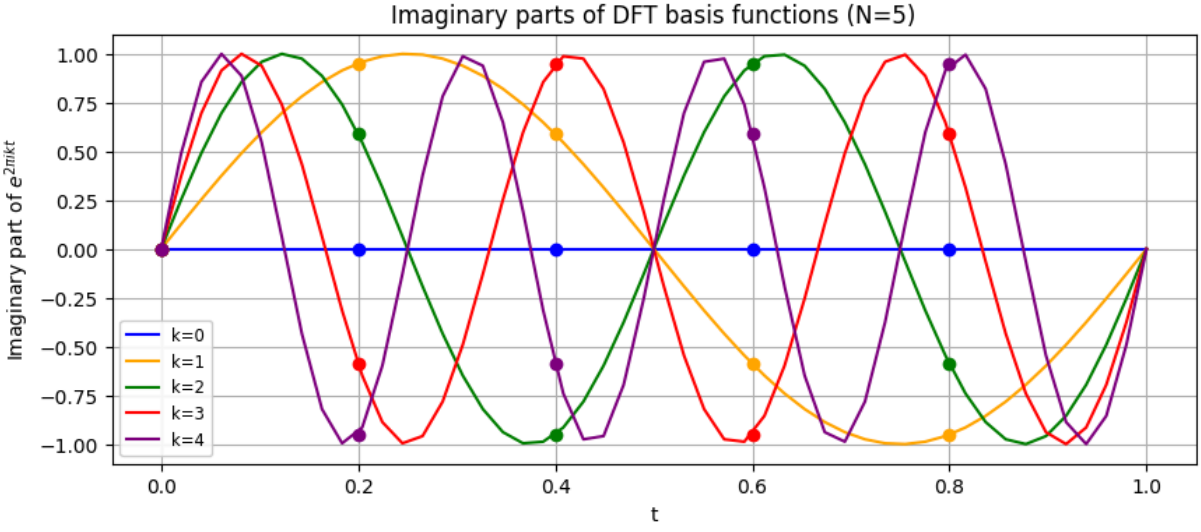
By plotting the real and imaginary parts of the DFT basis functions, we can clearly see that, as k goes higher, the representation on the graph shows a cosine and sine wave respectively, with increasing frequencies.

With $k = 0$, the representation is for the real part equal to $cos(0) = 1$, and for the imaginary part it represents $sin(0) = 0$. And for $k = 1$, both waves clearly represent a cycle of a cosine and sine wave within $t = [0, 1]$.

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt


N = 5
k_list = [i for i in range(N)]
t = np.linspace(0, 1)
t_highlight = np.array([n/N for n in range(N)])
color = ['blue','orange', 'green', 'red', 'purple']


exponentials_real = [np.exp(2*np.pi*1j*k*t).real for k in k_list]
exponentials_imag = [np.exp(2*np.pi*1j*k*t).imag for k in k_list]
expo_high_real = [np.exp(2*np.pi*1j*k*t_highlight).real for k in k_list]
expo_high_imag = [np.exp(2*np.pi*1j*k*t_highlight).imag for k in k_list]


plt.figure(figsize=(10, 4))
for k in k_list:
  plt.plot(t, exponentials_real[k], label=f'k={k}', color=color[k])
  plt.plot(t_highlight, expo_high_real[k], 'o', color=color[k])
```

```
plt.xlabel('t')
plt.ylabel('Real part of $e^{2\\pi i k t}$')
plt.title('Real parts of DFT basis functions (N=5)')
plt.legend(loc='lower left', ncol=1, fontsize='small', framealpha=0.9)
plt.grid(True)
plt.ylim(-1.1, 1.1)
plt.show()


plt.figure(figsize=(10, 4))
for k in k_list:
  plt.plot(t, exponentials_imag[k], label=f'k={k}', color=color[k])
  plt.plot(t_highlight, expo_high_imag[k], 'o', color=color[k])


plt.xlabel('t')
plt.ylabel('Imaginary part of $e^{2\\pi i k t}$')
plt.title('Imaginary parts of DFT basis functions (N=5)')
plt.legend(loc='lower left', ncol=1, fontsize='small', framealpha=0.9)
plt.grid(True)
plt.ylim(-1.1, 1.1)
plt.show()
```





b)

The equality is clear, as we can rewrite the first term as follows:

$$e^{2\pi i(N-k)\frac{n}{N}} = e^{2\pi in}e^{-2\pi ik\frac{n}{N}}$$

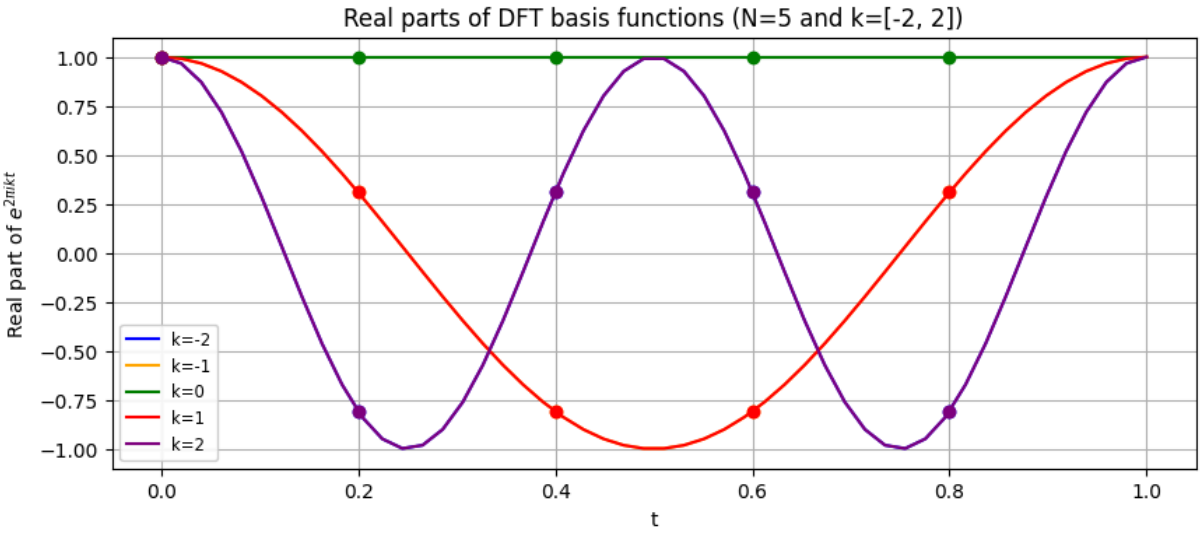
And, as $e^{2\pi in} = 1$ for any integer n, as it represents a cycle within the unit circle, we get the equality:

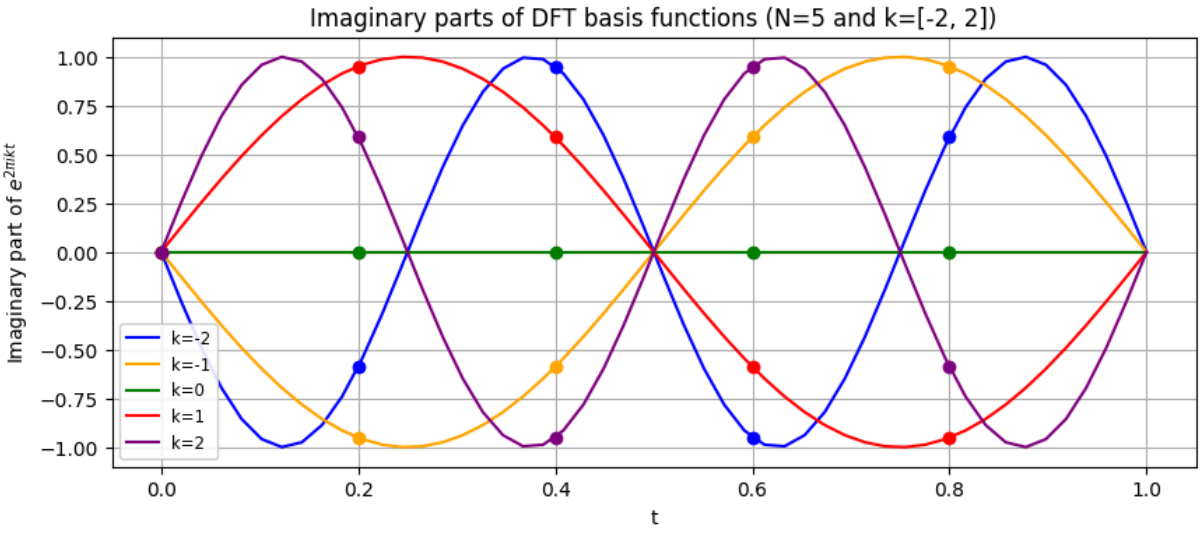
$$e^{2\pi i(N-k)\frac{n}{N}} = e^{-2\pi ik\frac{n}{N}}$$

We can now plot the previous graph with $k \in -2, -1, 0, 1, 2$, where we can clearly see that the real part is the same for the positive and negative part, while the imaginary part is a mirrored version. It's clear that the highlighted points in the previous and new graph are equal.

In [ ]:
```
N = 5
k_list = [-2, -1, 0, 1, 2]
t = np.linspace(0, 1)
t_highlight = np.array([n/N for n in range(N)])
color = ['blue','orange', 'green', 'red', 'purple']


exponentials_real = [np.exp(2*np.pi*1j*k*t).real for k in k_list]
exponentials_imag = [np.exp(2*np.pi*1j*k*t).imag for k in k_list]
expo_high_real = [np.exp(2*np.pi*1j*k*t_highlight).real for k in k_list]
expo_high_imag = [np.exp(2*np.pi*1j*k*t_highlight).imag for k in k_list]


plt.figure(figsize=(10, 4))
for i in range(len(k_list)):
  k = k_list[i]
  plt.plot(t, exponentials_real[i], label=f'k={k}', color=color[i])
  plt.plot(t_highlight, expo_high_real[i], 'o', color=color[i])

plt.xlabel('t')
plt.ylabel('Real part of $e^{2\\pi i k t}$')
plt.title('Real parts of DFT basis functions (N=5 and k=[-2, 2])')
plt.legend(loc='lower left', ncol=1, fontsize='small', framealpha=0.9)
plt.grid(True)
plt.ylim(-1.1, 1.1)
plt.show()


plt.figure(figsize=(10, 4))
for i in range(len(k_list)):
  k = k_list[i]
  plt.plot(t, exponentials_imag[i], label=f'k={k}', color=color[i])
  plt.plot(t_highlight, expo_high_imag[i], 'o', color=color[i])


plt.xlabel('t')
plt.ylabel('Imaginary part of $e^{2\\pi i k t}$')
```

```
plt.title('Imaginary parts of DFT basis functions (N=5 and k=[-2, 2])')
plt.legend(loc='lower left', ncol=1, fontsize='small', framealpha=0.9)
plt.grid(True)
plt.ylim(-1.1, 1.1)
plt.show()
```

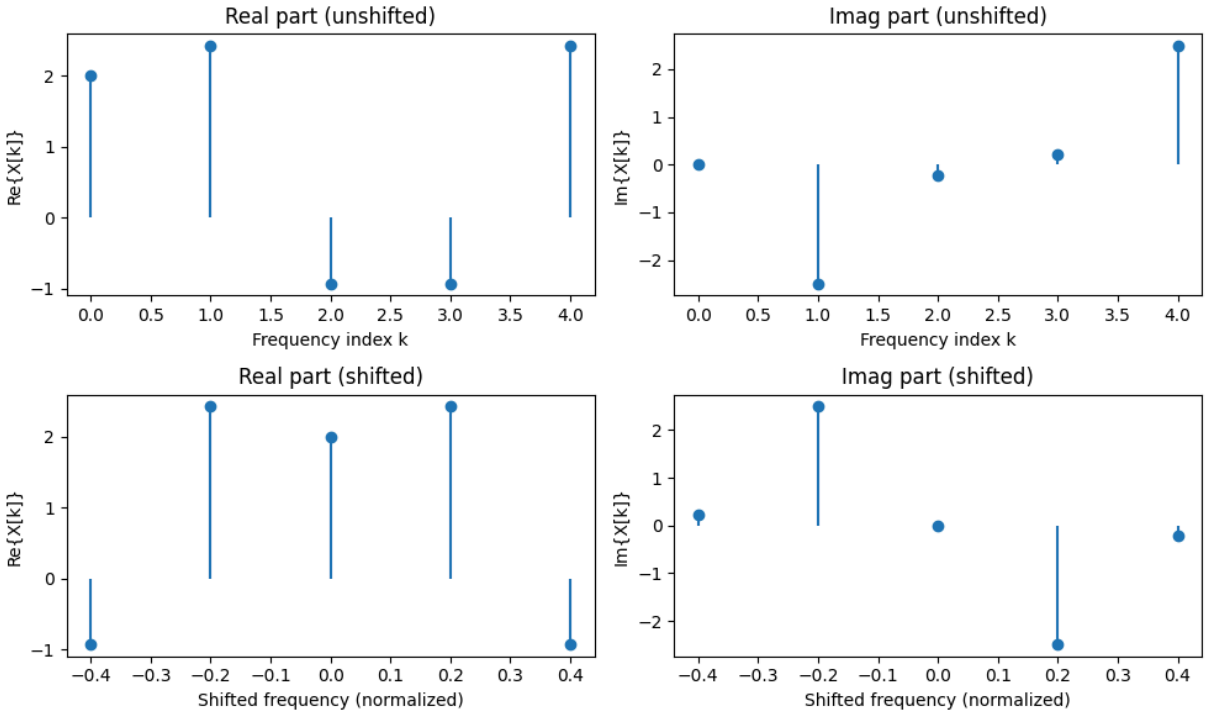




c)

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftshift, fftfreq

# --- Parameters ---
N = 5                       # signal length
n = np.arange(N)            # sample indices
x = np.array([1, 2, 0, -1, 0])  # example real signal

# --- Compute FFT ---
X = fft(x)

# --- Frequency axes ---
freqs = np.arange(N)            # unshifted frequency indices (0 ... N-1)
```

```
freqs_shifted = fftshift(fftfreq(N))  # shifted frequencies (-0.5 ... 0.5 cy

# --- Apply fftshift ---
X_shifted = fftshift(X)

# --- Plot ---
fig, axs = plt.subplots(2, 2, figsize=(10,6))
axs[0,0].stem(freqs, X.real, basefmt=" ")
axs[0,0].set_title("Real part (unshifted)")
axs[0,0].set_xlabel("Frequency index k")
axs[0,0].set_ylabel("Re{X[k]}")

axs[0,1].stem(freqs, X.imag, basefmt=" ")
axs[0,1].set_title("Imag part (unshifted)")
axs[0,1].set_xlabel("Frequency index k")
axs[0,1].set_ylabel("Im{X[k]}")

axs[1,0].stem(freqs_shifted, X_shifted.real, basefmt=" ")
axs[1,0].set_title("Real part (shifted)")
axs[1,0].set_xlabel("Shifted frequency (normalized)")
axs[1,0].set_ylabel("Re{X[k]}")

axs[1,1].stem(freqs_shifted, X_shifted.imag, basefmt=" ")
axs[1,1].set_title("Imag part (shifted)")
axs[1,1].set_xlabel("Shifted frequency (normalized)")
axs[1,1].set_ylabel("Im{X[k]}")

plt.tight_layout()
plt.show()
```



Without fftshift: The zero frequency X0(DC term) appears on the left, and higher frequencies proceed to the right. This is the default FFT order.

With fftshift: The spectrum is rearranged so that zero frequency is in the center. The negative frequencies appear on the left, and the positive frequencies on the right.

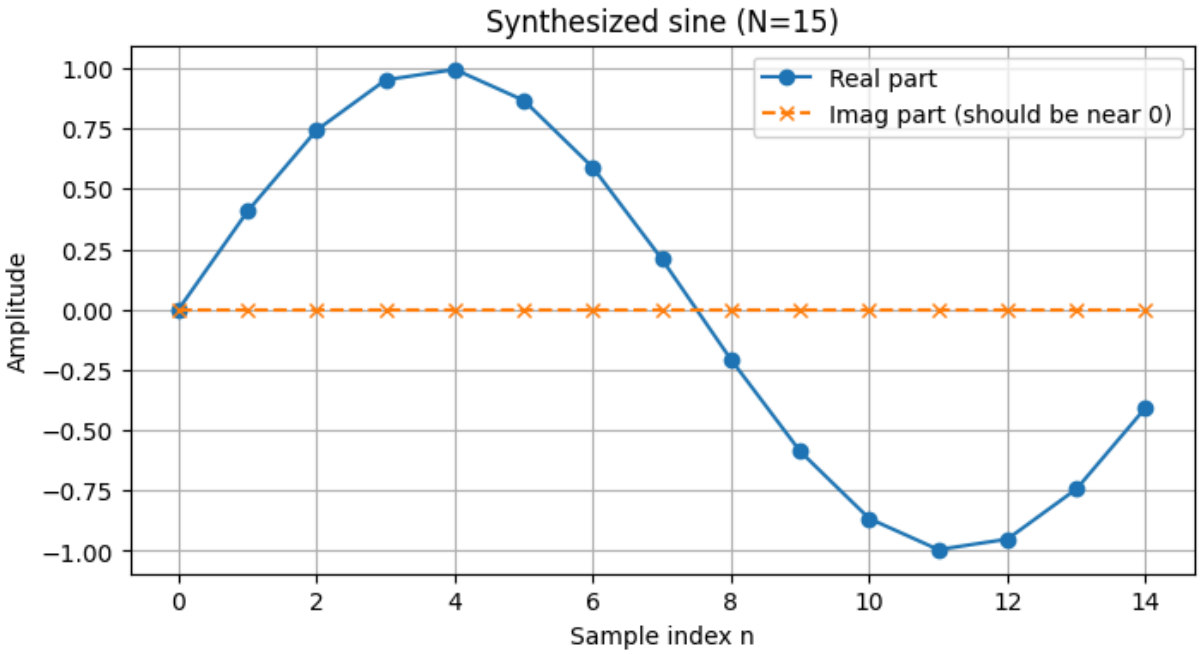
As evident from the plots, the DFT is conjugate symmetric: The real part is even symmetric and the imaginary part is odd symmetric.

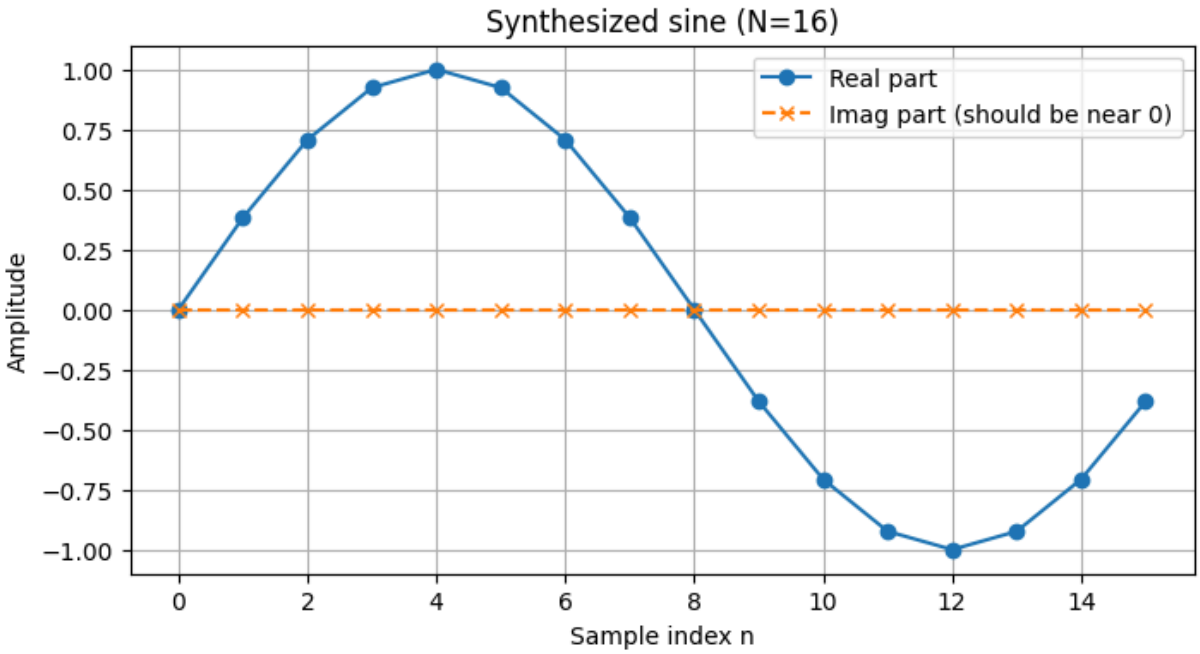
When N is odd, the center bin corresponds exactly to frequency 0. When N is even, there's no single center bin. fftshift moves the Nyquist frequency to the leftmost position, and the zero frequency goes to the center-left. In other words: For odd N, the shift is by (N-1)/2 positions. For even N, the shift is by N/2 positions.

d)

In [4]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import ifft, fftshift, ifftshift

def synthesize_sine(N):
    # --- Create frequency-domain coefficients ---
    X = np.zeros(N, dtype=complex)

    # Fundamental frequency index (one period)
    k0 = 1  # fundamental frequency component

    # Conjugate-symmetric coefficients for real sine
    X[k0] = -1j * N / 2
    X[-k0] =  1j * N / 2

    # --- Inverse FFT (reconstruct time-domain signal) ---
    x = ifft(X)

    # --- Plot ---
    n = np.arange(N)
    plt.figure(figsize=(8,4))
    plt.plot(n, x.real, 'o-', label='Real part')
    plt.plot(n, x.imag, 'x--', label='Imag part (should be near 0)')
    plt.title(f"Synthesized sine (N={N})")
    plt.xlabel("Sample index n")
    plt.ylabel("Amplitude")
    plt.legend()
    plt.grid(True)
    plt.show()

    return x
```

In [6]:
```python
#Test with odd and even N values
for N in [15, 16]:
    synthesize_sine(N)
```





For a real sinusoid, its DFT has two nonzero coefficients. One at +f_0 and one at -f_0.

In the DFT array,

- the coefficient for +f_0 corresponds to index $k_0 = f_0 N$
- the coefficient for −f_0 corresponds to $k = N - k_0$

To get a real-valued sine, the coefficients must satisfy conjugate symmetry:

$$X[k_0] = -j/2, X[N - k_0] = j/2$$

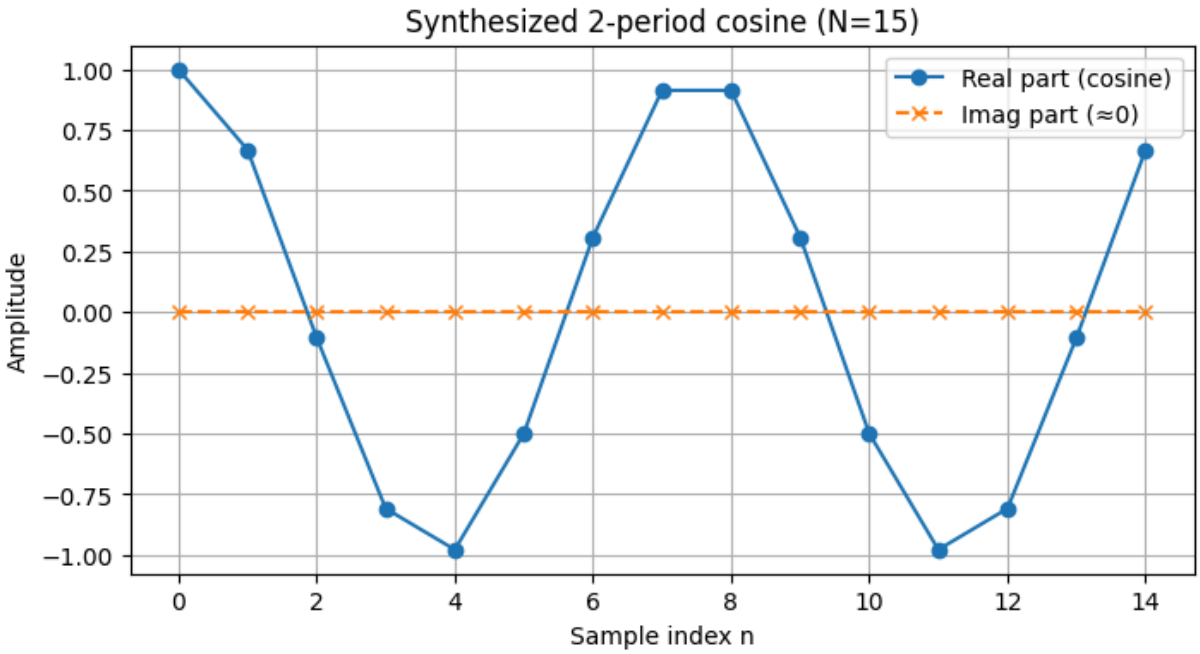
so that the imaginary part cancels out properly whereas for a cosine, they are real.

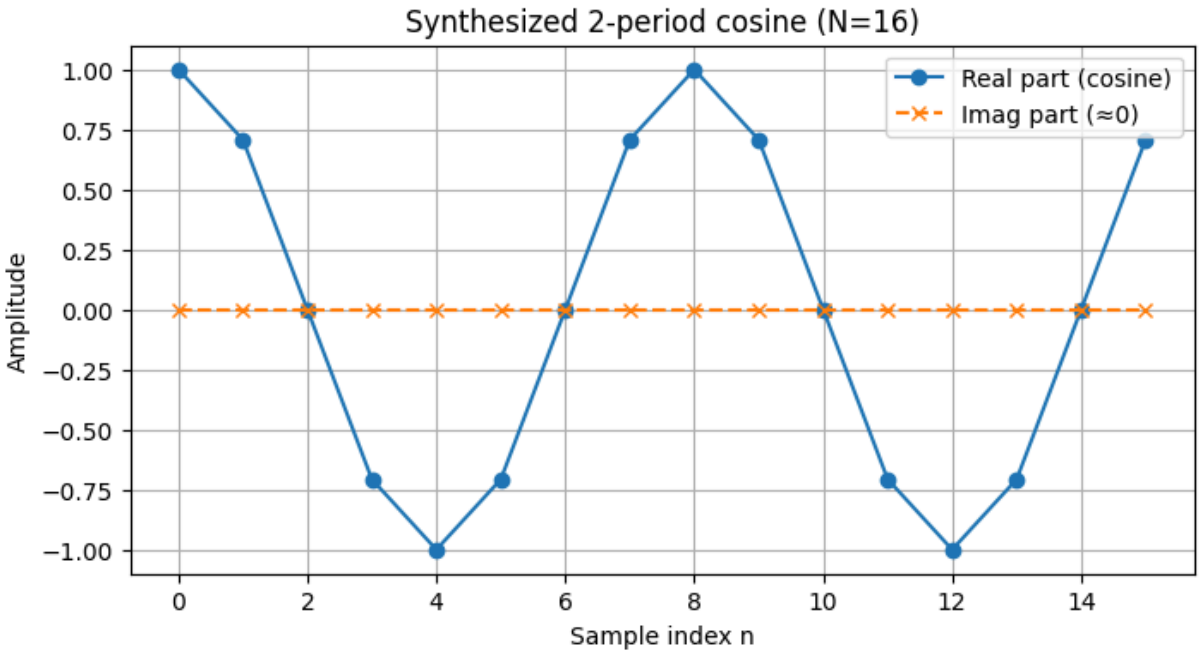
By default, NumPy's ifft does not divide by N, it produces the unnormalized inverse. Therefore, to get a signal with unit amplitude, we must ensure our coefficients are scaled correctly.

No explicit fftshift or ifftshift is required if we place the coefficients directly in the DFT order. We would need fftshift / ifftshift only if we had constructed our frequency spectrum in centered form (negative frequencies on the left, positive on the right), as shown in the previous task. In that case, before calling ifft(), we would apply ifftshift(X_centered) to convert it back to standard FFT order.

In [7]:
```
def synthesize_cosine_two_periods(N):
    X = np.zeros(N, dtype=complex)
    k0 = 2  # two cycles per N samples

    X[k0] = N / 2
    X[-k0] = N / 2

    x = ifft(X)

    n = np.arange(N)
    plt.figure(figsize=(8,4))
    plt.plot(n, x.real, 'o-', label='Real part (cosine)')
    plt.plot(n, x.imag, 'x--', label='Imag part (≈0)')
    plt.title(f"Synthesized 2-period cosine (N={N})")
    plt.xlabel("Sample index n")
    plt.ylabel("Amplitude")
    plt.legend()
    plt.grid(True)
    plt.show()

    return x
```

In [8]:
```
for N in [15, 16]:
    synthesize_cosine_two_periods(N)
```


Synthesized 2-period cosine (N=15)
Real part (cosine)
Imag part (≈0)
Amplitude
Sample index n
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14


Synthesized 2-period cosine (N=16)
Real part (cosine)
Imag part (≈0)
Amplitude
Sample index n
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
12
14

In [ ]: