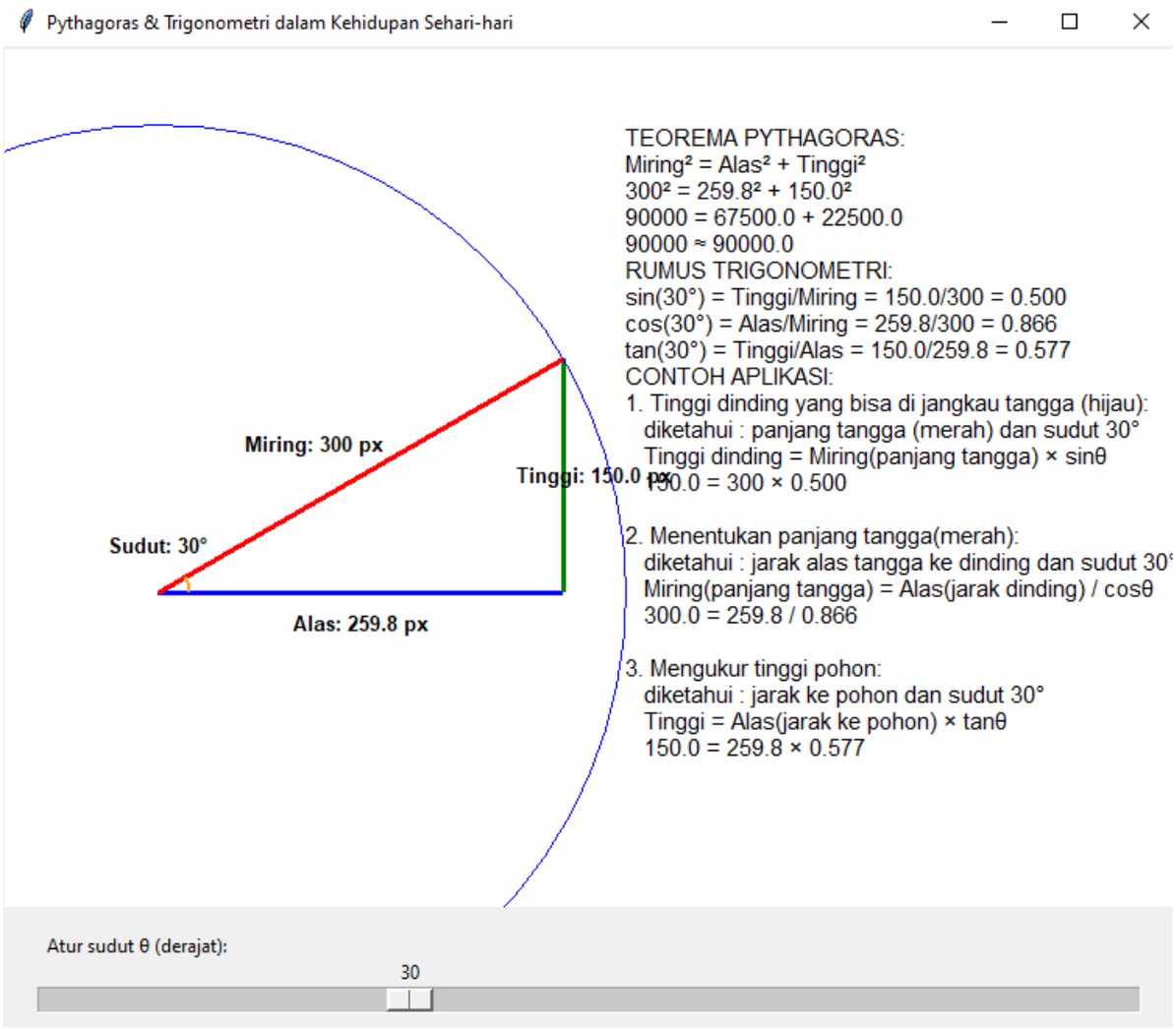
The Nature of Code

by Daniel Shiffman
https://github.com/edycoleee/nature
https://editor.p5js.org/natureofcode/collections



Pythagoras, trigonometri, dan aplikasinya

🔷 1. Teorema Pythagoras

📘 Penjelasan sederhana:

Teorema Pythagoras berlaku untuk **segitiga siku-siku** (yang punya sudut 90°).

Jika:

- Miring = sisi paling panjang (di depan sudut 90°)
- Alas = sisi bawah
- Tinggi = sisi tegak (naik ke atas)

Maka rumusnya:

Miring² = Alas² + Tinggi²

🧱 Contoh perhitungan:

Misalnya, panjang alas = 3, tinggi = 4:

Miring² = 3² + 4²

Miring² = 9 + 16 = 25

Miring = √25 = **5**

🔷 2. Hubungan dengan Trigonometri

Trigonometri membantu kita **menghitung sisi-sisi segitiga** atau **sudut**, jika kita tahu sebagian saja.

Dengan sudut θ (theta), dan sisi miring (hypotenuse), kita punya **rumus trigonometri:**

Nama	Rumus	Artinya
sin θ	tinggi / miring	untuk cari tinggi
cos θ	alas / miring	untuk cari alas
tan θ	tinggi / alas	untuk bandingkan tinggi dan alas

🔷 3. Aplikasi dalam Kehidupan Sehari-hari

1. **Tukang pasang tangga ke dinding**
    - Jika sudut tangga dan panjang tangga diketahui, kita bisa tahu **tinggi dinding** yang bisa dicapai.
    - Rumus: tinggi = miring × sin(θ)
2. **Menentukan panjang tangga**
    - Jika tahu jarak dari dinding (alas) dan sudutnya.
    - Rumus: miring = alas / cos(θ)
3. **Mengukur tinggi pohon**
    - Kita berdiri agak jauh dari pohon, ukur sudut pandang ke puncaknya.
    - Rumus: tinggi pohon = jarak ke pohon × tan(θ)

🔷 Penjelasan Simulasi Python Tkinter

🎮 Fungsi Simulasi:

Program menampilkan **segitiga siku-siku interaktif**, kamu bisa **mengubah sudut** dan langsung melihat:

- Gambar segitiga
- Hitungan sisi: alas, tinggi, dan miring
- Rumus Pythagoras
- Nilai sin, cos, tan
- Contoh aplikasi di dunia nyata

📐 Cara kerjanya:

- Panjang sisi miring ditetapkan 300 pixel.
- Saat kamu ubah sudut (angle_degrees), maka:
    - alas = cos(θ) × miring
    - tinggi = sin(θ) × miring
- Program menggambar segitiga di layar berdasarkan hasil perhitungan.
- Lalu, ditampilkan penjelasan rumus dan contoh soal.

🧑🏫 Kesimpulan

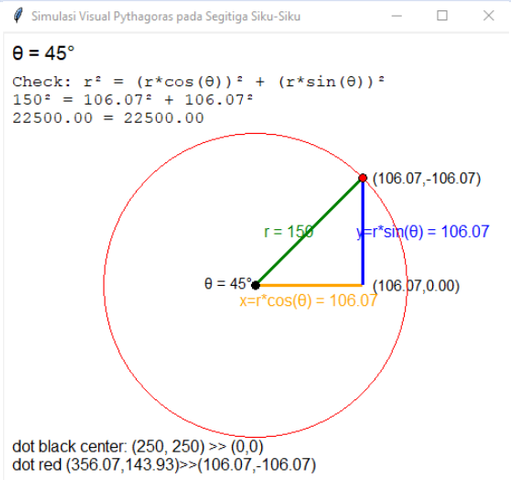
- Segitiga siku-siku itu seperti tangga: ada bawah (alas), naik (tinggi), dan tangga itu sendiri (miring).
- Pythagoras bantu menghitung satu sisi kalau dua sisi lain sudah diketahui.
- Trigonometri bantu menghitung sisi atau sudut saat hanya tahu sebagian.
- Dipakai dalam banyak hal: bangun rumah, ukur tinggi pohon, sampai bikin game atau robot!

COPY PASTE

```
#1. RUMUS TRIGONOMETRI
import tkinter as tk
import math

def update_triangle(angle_degrees):
    canvas.delete("all")

    angle = math.radians(float(angle_degrees))
    hypotenuse = 300  # Panjang sisi miring dalam pixel

    # Hitung sisi segitiga
    adjacent = hypotenuse * math.cos(angle)  # Sisi alas
    opposite = hypotenuse * math.sin(angle)  # Sisi tegak

    # ======== GAMBAR SEGITIGA ========
    # Alas (biru)
    canvas.create_line(100, 350, 100 + adjacent, 350, fill="blue", width=3)
    # Tinggi (hijau)
    canvas.create_line(100 + adjacent, 350, 100 + adjacent, 350 - opposite, fill="green",
width=3)
    # Miring (merah)
    canvas.create_line(100, 350, 100 + adjacent, 350 - opposite, fill="red", width=3)

    # Gambar sudut
    canvas.create_arc(80, 330, 120, 370, start=0, extent=angle_degrees,
                      outline="orange", width=2, style="arc")

    # ======== TEOREMA PYTHAGORAS ========
    pythagoras_text = (
        "TEOREMA PYTHAGORAS:\n"
        "Miring² = Alas² + Tinggi²\n"
        f"{hypotenuse}² = {adjacent:.1f}² + {opposite:.1f}²\n"
        f"{hypotenuse**2} = {adjacent**2:.1f} + {opposite**2:.1f}\n"
        f"{hypotenuse**2} ≈ {adjacent**2 + opposite**2:.1f}"
    )
```

```
    # ======== RUMUS TRIGONOMETRI ========
    trigono_text = (
        "\nRUMUS TRIGONOMETRI:\n"
        f"sin({angle_degrees}°) = Tinggi/Miring = {opposite:.1f}/{hypotenuse} =
{math.sin(angle):.3f}\n"
        f"cos({angle_degrees}°) = Alas/Miring = {adjacent:.1f}/{hypotenuse} =
{math.cos(angle):.3f}\n"
        f"tan({angle_degrees}°) = Tinggi/Alas = {opposite:.1f}/{adjacent:.1f} =
{math.tan(angle):.3f}"
    )

    # ======== CONTOH KASUS NYATA ========
    examples_text = (
        "\nCONTOH APLIKASI:\n"
        "1. Tinggi dinding yang bisa di jangkau tangga (hijau):\n"
        f"   diketahui : panjang tangga (merah) dan sudut {angle_degrees}°\n"
        f"   Tinggi dinding = Miring(panjang tangga) × sinθ\n"
        f"   {opposite:.1f} = {hypotenuse} × {math.sin(angle):.3f}\n\n"
        "2. Menentukan panjang tangga(merah):\n"
        f"   diketahui : jarak alas tangga ke dinding dan sudut {angle_degrees}°\n"
        f"   Miring(panjang tangga) = Alas(jarak dinding) / cosθ\n"
        f"   {hypotenuse:.1f} = {adjacent:.1f} / {math.cos(angle):.3f}\n\n"
        "3. Mengukur tinggi pohon:\n"
        f"   diketahui : jarak ke pohon dan sudut {angle_degrees}°\n"
        f"   Tinggi = Alas(jarak ke pohon) × tanθ\n"
        f"   {opposite:.1f} = {adjacent:.1f} × {math.tan(angle):.3f}"
    )

    # Gabungkan semua teks
    info_text = pythagoras_text + trigono_text + examples_text

    # Tampilkan teks
    canvas.create_text(400, 50, text=info_text, font=("Arial", 11),
                       anchor="nw", justify=tk.LEFT)

    # Label sisi-sisi segitiga
    canvas.create_text(100, 320, text=f"Sudut: {angle_degrees}°",
                       font=("Arial", 10, "bold"))
    canvas.create_text(100 + adjacent/2, 370, text=f"Alas: {adjacent:.1f} px",
                       font=("Arial", 10, "bold"))
    canvas.create_text(100 + adjacent + 20, 350 - opposite/2,
                       text=f"Tinggi: {opposite:.1f} px", font=("Arial", 10, "bold"))
    canvas.create_text(100 + adjacent/2 - 30, 350 - opposite/2 - 20,
                       text=f"Miring: {hypotenuse} px", font=("Arial", 10, "bold"))

    # Lingkaran (untuk visualisasi sudut)
    center_x, center_y = 100, 350
    radius = 300
    canvas.create_oval(center_x - radius, center_y - radius,
                    center_x + radius, center_y + radius, outline="blue")

# Membuat window
root = tk.Tk()
root.title("Pythagoras & Trigonometri dalam Kehidupan Sehari-hari")
canvas = tk.Canvas(root, width=750, height=550, bg="white")
canvas.pack()

# Slider untuk sudut
angle_slider = tk.Scale(root, from_=1, to=89, resolution=1, orient=tk.HORIZONTAL,
                        label="Atur sudut θ (derajat):",
                        command=update_triangle)
angle_slider.set(30)
angle_slider.pack(fill=tk.X, padx=20, pady=10)

update_triangle(30)
root.mainloop()
```

PERHITUNGAN PYTAGORAS



Berikut **simulasi interaktif Tkinter memahami hubungan Pythagoras pada segitiga siku-siku:**

$$r^2 = (r \times \cos(\theta))^2 + (r \times \sin(\theta))^2$$

dengan **visual garis bantu dan perubahan θ menggunakan slider interaktif** agar **benar-benar melihat bahwa panjang sisi miring tetap r**, berapa pun sudutnya.

---

1 Konsep Dasar yang Ditampilkan

- Dari pusat (x, y), gambar **garis horizontal (r × cos(θ))**.
- Dari ujung garis horizontal, gambar **garis vertikal (r × sin(θ))**.
- Hubungkan ujung garis vertikal ke pusat, membentuk **segitiga siku-siku**.
- Panjang sisi miring adalah **r**.
- Ditampilkan secara visual **perubahan nilai cos(θ), sin(θ), dan panjang sisi secara real-time**.

---

2 Script Simulasi Visual Interaktif

```
#Simulasi Visual Pythagoras pada Segitiga Siku-Siku > COPY PASTE
#Simulasi Visual Pythagoras pada Segitiga Siku-Siku > COPY PASTE
import tkinter as tk
import math

def update_canvas(angle_deg):
    canvas.delete("all")
    cx, cy = 250, 250
    r = 150
    theta = math.radians(angle_deg)

    x_cos = r * math.cos(theta)
    y_sin = r * math.sin(theta)

    # Titik ujung garis horizontal
    hx = cx + x_cos
    hy = cy

    # Titik ujung garis vertikal
    px = hx
    py = hy - y_sin

    # Gambar garis horizontal (r*cos(θ))
    canvas.create_line(cx, cy, hx, hy, fill='orange', width=3)
    canvas.create_text((cx+hx)/2, hy+15, text=f"x=r*cos(θ) = {x_cos:.2f}", fill='orange',
font=("Arial", 12))

    # Gambar garis vertikal (r*sin(θ))
    canvas.create_line(hx, hy, px, py, fill='blue', width=3)
    canvas.create_text(hx+60, (hy+py)/2, text=f"y=r*sin(θ) = {y_sin:.2f}", fill='blue',
font=("Arial", 12))

    # Gambar sisi miring (r)
    canvas.create_line(px, py, cx, cy, fill='green', width=3)
    canvas.create_text((cx+px)/2 - 20, (cy+py)/2, text=f"r = {r}", fill='green',
font=("Arial", 12))

    # Gambar titik-titik
    canvas.create_oval(cx-4, cy-4, cx+4, cy+4, fill='black')  # Pusat
    canvas.create_oval(px-4, py-4, px+4, py+4, fill='red')    # Titik ujung
```

```
    canvas.create_text(px+10, py-8, anchor='nw', text=f"({px-cx:.2f},{py-cy:.2f})",
font=("Arial", 11))
    canvas.create_text(hx+10, hy-8, anchor='nw', text=f"({hx-cx:.2f},{hy-cy:.2f})",
font=("Arial", 11))

    # Tulisan sudut
    canvas.create_text(10, 10, anchor='nw', text=f"θ = {angle_deg}°", font=("Arial", 14))
    canvas.create_text(cx-50, cy-10, anchor='nw', text=f"θ = {angle_deg}°", font=("Arial",
11))

    # Pythagoras check
    lhs = r ** 2
    rhs = x_cos ** 2 + y_sin ** 2
    check_text = f"Check: r² = (r*cos(θ))² + (r*sin(θ))²\n{r}² = {x_cos:.2f}² +
{y_sin:.2f}²\n{lhs:.2f} = {rhs:.2f}"
    canvas.create_text(10, 40, anchor='nw', text=check_text, font=("Courier", 12),
fill='black')
    check_text1 = f"dot black center: {cx, cy} >> (0,0)\ndot red
({px:.2f},{py:.2f})>>({px-cx:.2f},{py-cy:.2f}) "
    canvas.create_text(10, 400, anchor='nw', text=check_text1, font=("Arial", 12),
fill='black')

    canvas.create_oval(cx-r, cy-r, cx+r, cy+r, outline="red")

def on_slider_change(value):
    angle_deg = int(value)
    update_canvas(angle_deg)

root = tk.Tk()
root.title("Simulasi Visual Pythagoras pada Segitiga Siku-Siku")

canvas = tk.Canvas(root, width=500, height=500, bg='white')
canvas.pack()

slider = tk.Scale(root, from_=0, to=360, orient='horizontal', label='Atur θ (derajat)',
command=on_slider_change)
slider.pack()
slider.set(45)  # Default sudut 45°

update_canvas(45)

root.mainloop()
```

3️⃣ **Penjelasan Visual**
✅ **Slider untuk mengubah θ (derajat).**
✅ **Garis orange = r × cos(θ)** (sisi mendatar).
✅ **Garis biru = r × sin(θ)** (sisi tegak).
✅ **Garis hijau = r (sisi miring).**
✅ **Tampilan nilai r*cos(ϑ), r*sin(θ),**

---

segitiga sama sisi

1️⃣ **Penjelasan Singkat**

- **Segitiga sama sisi**: ketiga sisinya sama panjang.
- Menggunakan **sudut 0°, 120°, 240°** dari pusat untuk mendapatkan 3 titik.
- Digambar dengan canvas.create_polygon.

2️⃣ **Script Python Tkinter Lengkap**

```
#COPY PASTE
import tkinter as tk
import math

def get_triangle_coords(center, r, angles):
    x, y = center
    points = []
    for theta in angles:
        px = x + r * math.cos(theta)
        py = y + r * math.sin(theta)
        points.append((px, py))
    return points
```

```
root = tk.Tk()
root.title("Pemahaman Segitiga Sama Sisi: Visual dan Rumus")

canvas = tk.Canvas(root, width=500, height=500, bg='white')
canvas.pack()

center = (200, 200)
r = 100
angles = [0, 2*math.pi/3, 4*math.pi/3]  # 0°, 120°, 240°

points = get_triangle_coords(center, r, angles)

# Menggambar segitiga
canvas.create_polygon([p for point in points for p in point], fill='lightblue',
outline='black', width=2)

# Titik pusat
cx, cy = center
canvas.create_oval(cx-4, cy-4, cx+4, cy+4, fill='green')
canvas.create_text(cx+10, cy, text=f"Center ({int(cx)}, {int(cy)})", anchor='w',
font=("Arial", 10))

canvas.create_oval(cx-r, cy-r, cx+r, cy+r, outline="red")

# Tampilkan titik-titik sudut dengan hasil perhitungan
for idx, (px, py) in enumerate(points, start=1):
    canvas.create_oval(px-4, py-4, px+4, py+4, fill='red')
    canvas.create_text(px+10, py, text=f"P{idx} ({int(px)}, {int(py)})", anchor='w',
font=("Arial", 10))

# Tampilkan rumus di layar
rumus = [
    "Rumus Mencari Titik Sudut Segitiga:",
    "px = x + r * cos(theta)",
    "py = y + r * sin(theta)",
    "",
    f"x = {center[0]}, y = {center[1]}, r = {r}",
    "theta = 0°, 120°, 240° (0, 2.094, 4.188 rad)",
    "",
    f"Hasil:",
    f"P1: ({int(points[0][0])}, {int(points[0][1])})",
    f"P2: ({int(points[1][0])}, {int(points[1][1])})",
    f"P3: ({int(points[2][0])}, {int(points[2][1])})"
]

for i, line in enumerate(rumus):
    canvas.create_text(10, 10 + i*18, text=line, anchor='nw', font=("Courier", 10),
fill='black')

root.mainloop()
```

---

3 Apa yang dilihat?

✅ Segitiga **berwarna biru muda** di tengah layar.
✅ **Titik merah** pada ketiga sudut segitiga.
✅ Label koordinat **(x, y)** pada setiap sudut.
✅ Titik hijau sebagai pusat segitiga.

---

4 Penjelasan nilai nyata

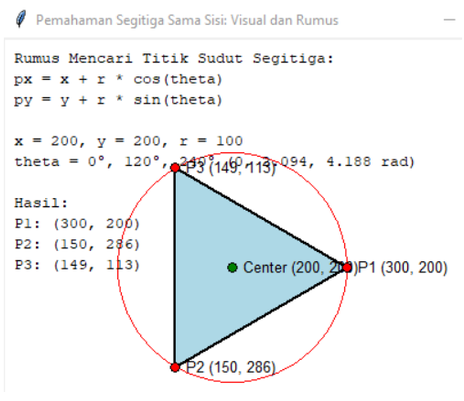
Misal:

- Pusat = (200, 200)
- size = 150 ⇒ r = 75
- angle = 0

Maka sudut:

- i = 0 ⇒ θ = 0°
- i = 1 ⇒ θ = 120° (2.094 rad)
- i = 2 ⇒ θ = 240° (4.188 rad)

Perhitungan:



Perhitungan:

✅ Titik 1 (0°):

$$x = 200 + 75 \times \cos(0) = 200 + 75 = 275$$

$$y = 200 + 75 \times \sin(0) = 200 + 0 = 200$$

(275, 200)

✅ Titik 2 (120°):

$$x = 200 + 75 \times \cos(2.094) = 200 - 37.5 = 162.5$$

$$y = 200 + 75 \times \sin(2.094) = 200 + 64.95 = 264.95$$

(163, 265)

✅ Titik 3 (240°):

$$x = 200 + 75 \times \cos(4.188) = 200 - 37.5 = 162.5$$

$$y = 200 + 75 \times \sin(4.188) = 200 - 64.95 = 135.05$$

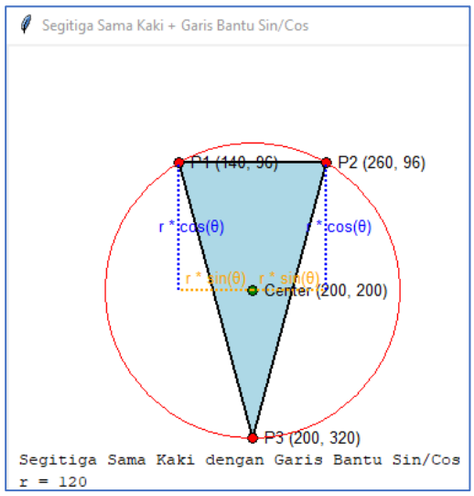
(163, 135)

MEMBUAT SEGITIGA SAMA KAKI

script segitiga sama kaki dengan garis bantu trigonometri:

📐 Rumus dan Perhitungan Dasar

1. **Parameter Utama**:
   - center = (200, 200) → Pusat koordinat (cx, cy)
   - r = 120 → Panjang kaki segitiga (radius)
   - theta = 30° → Sudut kemiringan (dikonversi ke radian) >> SUDUT LANCIP
2. **Koordinat Titik Segitiga**:
   - **P1 (Atas Kiri)**:
     px = cx - r * sin(θ)
     py = cy - r * cos(θ)
     Contoh perhitungan:
     px = 200 - 120 * sin(30°) = 200 - 120*0.5 = 140
     py = 200 - 120 * cos(30°) ≈ 200 - 120*0.866 ≈ 96.08
   - **P2 (Atas Kanan)**:
     px = cx + r * sin(θ)
     py = cy - r * cos(θ)
     Contoh:
     px = 200 + 60 = 260
     py = 96.08 (sama dengan P1)
   - **P3 (Bawah Tengah)**:
     px = cx
     py = cy + r
     Contoh:
     px = 200
     py = 200 + 120 = 320



🔍 Detail Kode Penting

1. **Konversi Sudut**:
   theta = math.radians(30) *# 30° → ~0.5236 radian*
2. **Pembuatan Segitiga**:
   canvas.create_polygon([p1, p2, p3], fill='lightblue')
3. **Garis Bantu**:
   - Garis oranye (sinus): create_line(cx, cy, hx1, hy1)
   - Garis biru (cosinus): create_line(hx1, hy1, vx1, vy1)

💡 Konsep Trigonometri yang Terlibat

1. **Sin/Cos dalam Koordinat Kartesian**:
   - sin(θ) = komponen horizontal (sumbu X)
   - cos(θ) = komponen vertikal (sumbu Y)
2. **Sistem Koordinat Layar**:
   - Sumbu Y positif ke bawah → py = cy - r*cos(θ) untuk gerak ke atas
3. **Segitiga Sama Kaki**:
   - Dua sisi sama panjang (r)
   - Sudut dasar sama (θ)

```
#COPY PASTE
import tkinter as tk
import math

root = tk.Tk()
root.title("Segitiga Sama Kaki + Garis Bantu Sin/Cos")

canvas = tk.Canvas(root, width=500, height=700, bg='white')
canvas.pack()

# Parameter segitiga
center = (200, 200)
r = 120
theta = math.radians(30)  # sudut kaki terhadap vertikal (30°)

cx, cy = center

# Hitung titik-titik segitiga sama kaki
p1 = (cx - r * math.sin(theta), cy - r * math.cos(theta))  # kiri atas
p2 = (cx + r * math.sin(theta), cy - r * math.cos(theta))  # kanan atas
p3 = (cx, cy + r)                                  # bawah

# Gambar segitiga
canvas.create_polygon([p1, p2, p3], fill='lightblue', outline='black', width=2)

# Titik pusat
canvas.create_oval(cx-4, cy-4, cx+4, cy+4, fill='green')
canvas.create_text(cx+10, cy, text=f"Center ({int(cx)}, {int(cy)})", anchor='w', font=("Arial", 10))

# Gambar titik-titik sudut
for idx, (px, py) in enumerate([p1, p2, p3], start=1):
    canvas.create_oval(px-4, py-4, px+4, py+4, fill='red')
    canvas.create_text(px+10, py, text=f"P{idx} ({int(px)}, {int(py)})", anchor='w', font=("Arial", 10))

# Garis bantu untuk P1 (kiri atas)
hx1 = cx - r * math.sin(theta)  # horizontal ke kiri
hy1 = cy                        # tetap di pusat y
vx1 = hx1
vy1 = cy - r * math.cos(theta)   # naik ke atas

canvas.create_line(cx, cy, hx1, hy1, fill='orange', dash=(4,2), width=2)  # r * sin(θ)
canvas.create_line(hx1, hy1, vx1, vy1, fill='blue', dash=(4,2), width=2)  # r * cos(θ)

canvas.create_text((cx+hx1)//2, hy1 - 10, text="r * sin(θ)", fill='orange', font=("Arial", 10))
canvas.create_text(hx1 + 10, (hy1+vy1)//2, text="r * cos(θ)", fill='blue', font=("Arial", 10))

# Garis bantu untuk P2 (kanan atas)
hx2 = cx + r * math.sin(theta)  # horizontal ke kanan
hy2 = cy
vx2 = hx2
vy2 = cy - r * math.cos(theta)

canvas.create_line(cx, cy, hx2, hy2, fill='orange', dash=(4,2), width=2)  # r * sin(θ)
canvas.create_line(hx2, hy2, vx2, vy2, fill='blue', dash=(4,2), width=2)  # r * cos(θ)

canvas.create_text((cx+hx2)//2, hy2 - 10, text="r * sin(θ)", fill='orange', font=("Arial", 10))
canvas.create_text(hx2 + 10, (hy2+vy2)//2, text="r * cos(θ)", fill='blue', font=("Arial", 10))

canvas.create_oval(cx-r, cy-r, cx+r, cy+r, outline="red")

# Penjelasan teks di layar
penjelasan = [
    "Segitiga Sama Kaki dengan Garis Bantu Sin/Cos",
    f"r = {r}",
    "P1 Titik atas kiri diperoleh dengan:",
    "  px = x - r * sin(theta)",
```

```
    f" {vx1} = {cx} - {r} * math.sin({theta:.3})",
    "  py = y - r * cos(theta)",
    f" {vy1:.3} = {cy} - {r} * math.cos({theta:.3})",
    "",
    "P2 Titik atas kanan diperoleh dengan:",
    "  px = x + r * sin(theta)",
    f" {vx2} = {cx} + {r} * math.sin({theta:.3})",
    "  py = y - r * cos(theta)",
    f" {vy2:.3} = {cy} - {r} * math.cos({theta:.3})",
    "",
    "P3 Titik bawah tengah diperoleh dengan:",
    f"  px = x = {cx}",
    f"  py = y + r ={cy + r}",
    "",
    "Garis orange: r * sin(theta) (horizontal)",
    "Garis biru: r * cos(theta) (vertikal)"
]

for i, line in enumerate(penjelasan):
    canvas.create_text(10, 330 + i*18, text=line, anchor='nw', font=("Courier", 10), fill='black')

root.mainloop()
```

SEGITIGA / VEHICLE BERGERAK

🔄 **Alur Program**

1. **Inisialisasi Vehicle**:
    - Tentukan posisi awal (x,y)
    - Set velocity awal (1, 0.5)
    - Buat polygon segitiga untuk representasi visual
2. **Loop Update** (setiap 30ms):
    - Update posisi: position += velocity
    - Hitung arah hadap berdasarkan velocity
    - Gambar ulang segitiga yang menghadap ke arah velocity

📐 **Rumus dan Perhitungan**

1. Menghitung Arah Hadap (heading())

    angle = math.atan2(velocity.y, velocity.x)

    - Menggunakan atan2 untuk mendapatkan sudut dalam radian
    - Contoh velocity (1, 0.5):

      angle = atan2(0.5, 1) ≈ 0.4636 radian ≈ 26.565°

2. Menentukan Titik Segitiga

   front = Vector(cos(angle), sin(angle)) * (r*20)
   left = Vector(cos(angle + 2.5), sin(angle + 2.5)) * (r*20)
   right = Vector(cos(angle - 2.5), sin(angle - 2.5)) * (r*20)

    - r*20 menentukan panjang segitiga (6*20=120px)
    - 2.5 adalah sudut pembukaan sayap (bisa diadjust)

3. Posisi Aktual Titik

    p1 = position + front *# Titik depan*
    p2 = position + left *# Titik kiri belakang*
    p3 = position + right *# Titik kanan belakang*

🔢 **Frame 0 (Inisialisasi)**

1. Hitung sudut hadap (θ):

angle = math.atan2(0.5, 1) ≈ 0.4636 radian (26.565°)

2. Hitung titik-titik segitiga:

front = Vector(cos(0.4636), sin(0.4636)) * (6*3) ≈ (16.1, 8.05)
left = Vector(-sin(0.4636), cos(0.4636)) * 6 ≈ (-2.68, 5.36)
right = Vector(sin(0.4636), -cos(0.4636)) * 6 ≈ (2.68, -5.36)

3. Posisi aktual:

p1 = (100,100) + (16.1,8.05) = (116.1, 108.05)
p2 = (100,100) + (-2.68,5.36) = (97.32, 105.36)
p3 = (100,100) + (2.68,-5.36) = (102.68, 94.64)

🔄 **Frame 1 (t=30ms)**

1. Update posisi:

position += velocity → (100,100) + (1,0.5) = (101, 100.5)

2. Hitung sudut hadap (θ tetap karena velocity konstan):
angle ≈ 0.4636 radian
3. Hitung titik segitiga (rumus sama):
p1 = (101,100.5) + (16.1,8.05) = (117.1, 108.55)
p2 = (101,100.5) + (-2.68,5.36) = (98.32, 105.86)
p3 = (101,100.5) + (2.68,-5.36) = (103.68, 95.14)

📈 **Frame 2 (t=60ms)**
1. Update posisi:
(101,100.5) + (1,0.5) = (102, 101)
2. Titik segitiga:
p1 = (102,101) + (16.1,8.05) = (118.1, 109.05)
p2 = (102,101) + (-2.68,5.36) = (99.32, 106.36)
p3 = (102,101) + (2.68,-5.36) = (104.68, 95.64)

🧠 **Inti Perhitungan Tiap Frame**

1. **Step 1: Update Posisi**
   position.x += velocity.x
   position.y += velocity.y
2. **Step 2: Hitung Orientasi**
   - Normalisasi velocity:

   mag = sqrt($vx^2 + vy^2$)
   heading = (vx/mag, vy/mag) *# Vektor satuan*
   - Jika velocity = (0,0), gunakan heading sebelumnya
3. **Step 3: Hitung Titik Segitiga**
   front = heading * (r*3) *# Depan lebih panjang*
   left = (-heading.y, heading.x) * r *# Tegak lurus kiri*
   right = (heading.y, -heading.x) * r *# Tegak lurus kanan*
4. **Step 4: Posisi Aktual**
   p1 = position + front
   p2 = position + left
   p3 = position + right

🛠️ **Jika Velocity Berubah**
Contoh: Velocity berubah menjadi (2, 1) di Frame 5:

1. Hitung heading baru:
   angle = atan2(1, 2) ≈ 0.4636 rad (sama, tapi kecepatan 2x)
2. Panjang depan:
   front = (2,1).normalize() * 18 ≈ (16.1,8.05) * 2 ≈ (32.2,16.1)

📊 **Visualisasi Perubahan Orientasi**

Frame	Position	Velocity	Arah Hadap (θ)	Panjang Depan
0	(100,100)	(1,0.5)	26.565°	18 px
1	(101,100.5)	(1,0.5)	26.565°	18 px
5	(105,102.5)	(2,1)	26.565°	36 px
10	(110,105)	(0,-1)	-90° (ke bawah)	18 px

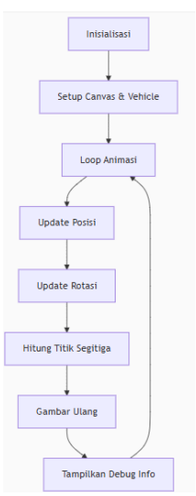
💡 **Penting!**

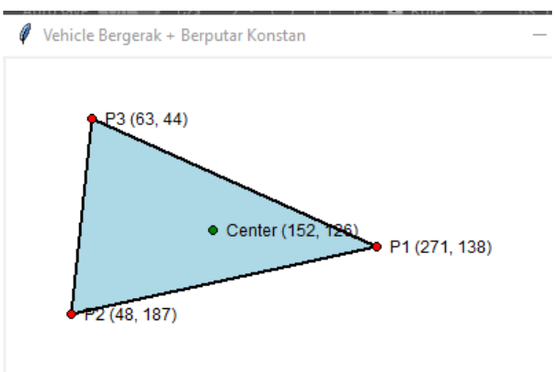
1. **Normalisasi Velocity** memastikan:
   - Bentuk segitiga konsisten walau kecepatan berubah
   - Tidak terjadi distorsi saat velocity membesar/ mengecil

```
# Vehicle Bergerak
import tkinter as tk
import math
from vector import Vector

class Vehicle:
    def __init__(self, x, y, canvas):
        self.canvas = canvas
        self.position = Vector(x, y)
        self.velocity = Vector(1, 0.5)  # bergerak pelan ke kanan dan bawah
```

```
        self.r = 6  # ukuran
        self.shape = self.canvas.create_polygon(0, 0, 0, 0, 0, 0, fill='lightblue',
outline='black', width=2)
        self.update()

    def update(self):
        # Update posisi vehicle
        self.position.add(self.velocity)
        self.show()
        self.canvas.after(30, self.update)  # panggil update setiap 30 ms

    def show(self):
        angle = self.velocity.heading()  # arah segitiga mengikuti arah velocity

        # Titik depan, kiri, kanan
        front = Vector(math.cos(angle), math.sin(angle)).multed(self.r * 20)
        left  = Vector(math.cos(angle + 2.5), math.sin(angle + 2.5)).multed(self.r * 20)
        right = Vector(math.cos(angle - 2.5), math.sin(angle - 2.5)).multed(self.r * 20)

        # Tambahkan posisi pusat vehicle
        p1 = self.position.added(front)
        p2 = self.position.added(left)
        p3 = self.position.added(right)

        # Gambar segitiga
        self.canvas.coords(self.shape, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

        # Hapus label sebelumnya
        self.canvas.delete('info')

        # Label titik-titik untuk belajar
        for p, label in zip([p1, p2, p3], ['P1', 'P2', 'P3']):
            self.canvas.create_oval(p.x-3, p.y-3, p.x+3, p.y+3, fill='red', tags='info')
            self.canvas.create_text(p.x+10, p.y, text=f"{label} ({int(p.x)}, {int(p.y)})",
anchor='w', font=("Arial", 9), tags='info')

        # Titik pusat
        cx, cy = self.position.x, self.position.y
        self.canvas.create_oval(cx-3, cy-3, cx+3, cy+3, fill='green', tags='info')
        self.canvas.create_text(cx+10, cy, text=f"Center ({int(cx)}, {int(cy)})",
anchor='w', font=("Arial", 9), tags='info')

# Membuat window
root = tk.Tk()
root.title("Vehicle Bergerak ")

# Membuat canvas
canvas = tk.Canvas(root, width=500, height=400, bg='white')
canvas.pack()

# Membuat Vehicle pada posisi awal (100, 100)
v = Vehicle(100, 100, canvas)

root.mainloop()
```

VEHICLE BERGERAK DAN BERPUTAR





📐 Rumus Utama

1. **Gerakan Linear**:

position.x += velocity.x
position.y += velocity.y

2. **Rotasi Independen**:

angle += 0.002 rad/frame

3. **Posisi Titik Segitiga**:

front = (cos(angle), sin(angle)) * (r*20)
left = (cos(angle+2.5), sin(angle+2.5)) * (r*20)
right = (cos(angle-2.5), sin(angle-2.5)) * (r*20)

🎞️ Perhitungan Tiap Frame (60ms)

Parameter Awal:

- Posisi: (100, 100)
- Velocity: (1, 0.5) pixel/frame
- Angle: 0 rad
- r: 6
- Rotasi: +0.002 rad/frame

Frame 0:

- **Posisi**: (100, 100)
- **Angle**: 0 rad
- **Titik**:
  - P1 (depan): (100 + 120*cos(0), 100 + 120*sin(0)) = (220, 100)
  - P2 (kiri): (100 + 120*cos(2.5), 100 + 120*sin(2.5)) ≈ (100 + 120*-0.801, 100 + 120*0.598) ≈ (3.88, 171.76)
  - P3 (kanan): (100 + 120*cos(-2.5), 100 + 120*sin(-2.5)) ≈ (196.12, 28.24)

Frame 1 (t=60ms):

- **Posisi**: (100 + 1, 100 + 0.5) = (101, 100.5)
- **Angle**: 0 + 0.002 = 0.002 rad ≈ 0.1146°
- **Titik**:
  - P1: (101 + 120*cos(0.002), 100.5 + 120*sin(0.002)) ≈ (221, 100.74)
  - P2: (101 + 120*cos(2.502), 100.5 + 120*sin(2.502)) ≈ (4.85, 172.47)
  - P3: (101 + 120*cos(-2.498), 100.5 + 120*sin(-2.498)) ≈ (197.15, 28.53)

Frame n:

- Posisi: (100 + n*1, 100 + n*0.5)
- Angle: n * 0.002 rad
- Pola gerakan spiral karena kombinasi:
  - Translasi konstan (velocity)
  - Rotasi konstan (angle += 0.002)

📊 Visualisasi Gerakan

Frame 0:
Posisi: (100,100)
Orientasi: 0°

Frame 60 (~3.6 detik):
Posisi: (160,130)
Orientasi: 0.002*60 ≈ 0.12 rad ≈ 6.88°

Frame 300 (~18 detik):
Posisi: (400,250)
Orientasi: 0.6 rad ≈ 34.4°

```
import tkinter as tk
import math
from vector import Vector

class App:
    def __init__(self, width=500, height=400):
        self.root = tk.Tk()
        self.root.title("Vehicle Bergerak + Berputar Konstan")

        self.canvas = tk.Canvas(self.root, width=width, height=height, bg='white')
        self.canvas.pack()

        self.vehicle = Vehicle(100, 100, self.canvas)

        self.root.mainloop()

class Vehicle:
```

```
    def __init__(self, x, y, canvas):
        self.canvas = canvas
        self.position = Vector(x, y)
        self.velocity = Vector(1, 0.5)  # bergerak ke kanan & bawah
        self.angle = 0  # sudut awal untuk rotasi independen
        self.r = 6  # ukuran
        self.shape = self.canvas.create_polygon(0, 0, 0, 0, 0, 0, fill='lightblue',
outline='black', width=2)
        self.update()

    def update(self):
        self.position.add(self.velocity)
        self.angle += 0.002  # radian per frame, rotasi konstan
        self.show()
        self.canvas.after(60, self.update)

    def show(self):
        angle = self.angle

        front = Vector(math.cos(angle), math.sin(angle)).multed(self.r * 20)
        left = Vector(math.cos(angle + 2.5), math.sin(angle + 2.5)).multed(self.r * 20)
        right = Vector(math.cos(angle - 2.5), math.sin(angle - 2.5)).multed(self.r * 20)

        p1 = self.position.added(front)
        p2 = self.position.added(left)
        p3 = self.position.added(right)

        self.canvas.coords(self.shape, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

        self.canvas.delete('info')

        for p, label in zip([p1, p2, p3], ['P1', 'P2', 'P3']):
            self.canvas.create_oval(p.x - 3, p.y - 3, p.x + 3, p.y + 3, fill='red',
tags='info')
            self.canvas.create_text(p.x + 10, p.y, text=f"{label} ({int(p.x)},
{int(p.y)})", anchor='w', font=("Arial", 9), tags='info')

        cx, cy = self.position.x, self.position.y
        self.canvas.create_oval(cx - 3, cy - 3, cx + 3, cy + 3, fill='green', tags='info')
        self.canvas.create_text(cx + 10, cy, text=f"Center ({int(cx)}, {int(cy)})",
anchor='w', font=("Arial", 9), tags='info')

if __name__ == "__main__":
    App()
```

Chapter 3. Oscillation

3.1 Angles

🔍 **1. Penjelasan Konsep**

Apa itu Sudut (Angle)?

- **Sudut** adalah seberapa besar sesuatu **berputar atau berbelok**.
- Satuan sudut:
    - **Derajat**: seperti di sekolah (0°–360°)
    - **Radian**: satuan lain yang digunakan oleh komputer dan matematika tingkat lanjut.

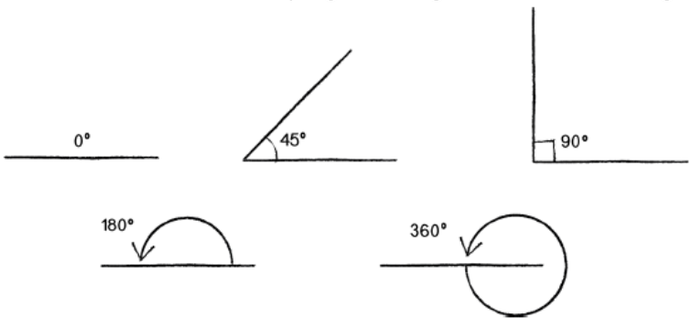
Perbandingan Derajat dan Radian

Derajat	Radian	Keterangan
0°	0	Tidak berputar
90°	π/2	Seperempat putaran
180°	π	Setengah putaran
360°	2π	Satu putaran penuh

Dengan **sin() dan cos()**, kita bisa:

- Membuat **kupu-kupu mengepakkan sayapnya**
- Membuat **planet mengorbit**
- Membuat **grafik gelombang**

- Membuat **bandul** yang bisa **bergerak naik turun dengan realistik**



perbedaan sudut derajat dan radian :

🔶 **Apa itu Sudut?**
Sudut adalah **besar putaran** antara dua garis. Bayangkan seperti kamu **memutar setir sepeda** atau **kompas**.

🟢 **1. Sudut Derajat (°)**
📏 **Penjelasan Sederhana:**
- Ini cara **paling umum** mengukur sudut, seperti di penggaris busur yang kamu pakai di sekolah.
- **1 lingkaran penuh = 360°**

📘 **Contoh:**
- Setengah lingkaran = **180°**
- Seperempat lingkaran = **90°**
- Seperdelapan lingkaran = **45°**

🎯 **Analogi:**
Bayangkan kamu memutar setir sepeda:
- Putar sedikit: 45°
- Putar setengah: 180°
- Putar balik arah: 360° (kembali ke posisi awal)

🔵 **2. Sudut Radian**
📏 **Penjelasan Sederhana:**
- Radian adalah satuan **yang dipakai di matematika & sains**, terutama **trigonometri**.
- Diukur berdasarkan **panjang busur lingkaran**.

1 lingkaran penuh = 2π radian ≈ 6.28 radian
Artinya:
- Setengah lingkaran = π radian ≈ 3.14
- Seperempat lingkaran = π/2 radian ≈ 1.57
- Seperdelapan lingkaran = π/4 radian ≈ 0.79

🔄 **Hubungan Derajat dan Radian**

Derajat (°)	Sama dengan	Radian
360°	1 lingkaran	2π radian
180°	setengah	π radian
90°	seperempat	π/2 radian
1°	kecil	π / 180 radian

🔁 **Rumus Konversi:**
- **Derajat ke radian:**
  radian = derajat × (π / 180)
- **Radian ke derajat:**
  derajat = radian × (180 / π)

🧠 **Gampangnya Ingat:**

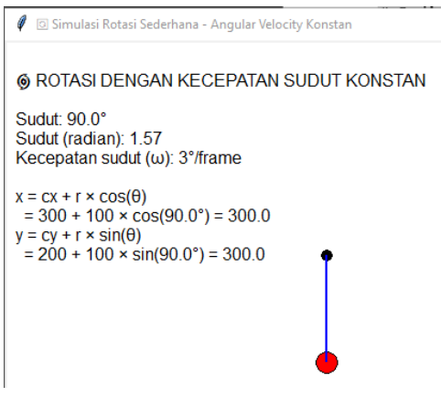
Kalau kamu pakai...	Maka...
Penggaris busur	Pakai **derajat (°)**
Kalkulator sin/cos dalam sains/matematika	Biasanya pakai **radian**
Sudut dalam program komputer	Kadang pakai **radian**, seperti di Python math.sin()

🧪 **Contoh Python:**

```
import math

# Derajat ke radian
deg = 60
```

```
rad = math.radians(deg)
print(f"{deg}° = {rad:.2f} rad") #60° = 1.05 rad

# Radian ke derajat
rad2 = math.pi / 3
deg2 = math.degrees(rad2)
print(f"{rad2:.2f} rad = {deg2:.1f}°") #1.05 rad = 60.0°
```

Simulasi Rotasi Sederhana - Angular Velocity Konstan



📝 Rumus Rotasi

1. **Konversi Sudut**:
   `radian = derajat × (π/180)`
2. **Posisi Objek**:
   x = cx + radius × cos(θ)
   y = cy + radius × sin(θ)
   Dimana:
   - (cx,cy) = pusat rotasi
   - radius = jarak dari pusat
   - θ = sudut dalam radian
3. **Update Sudut**:
   `angle_deg = (angle_deg + angular_velocity) % 360`

🎞️ Perhitungan Frame-by-Frame

Parameter:

- Pusat: (300, 200)
- Radius: 100 pixel
- Kecepatan sudut: 3°/frame
- Interval: 50ms/frame (~20 FPS)

Contoh Perhitungan 3 Frame:

Frame 0:

- Sudut: 0°
- Posisi:
  x = 300 + 100×cos(0) = 400
  y = 200 + 100×sin(0) = 200

Frame 1 (50ms):

- Sudut: 3°
- Radian: 3×π/180 ≈ 0.0524
- Posisi:
  x = 300 + 100×cos(0.0524) ≈ 300 + 99.86 ≈ 399.86
  y = 200 + 100×sin(0.0524) ≈ 200 + 5.24 ≈ 205.24

Frame 2 (100ms):

- Sudut: 6°
- Radian: 6×π/180 ≈ 0.1047
- Posisi:
  x ≈ 300 + 99.46 ≈ 399.46
  y ≈ 200 + 10.47 ≈ 210.47

Frame n:

- Sudut: n×3°
- Posisi:
  x = 300 + 100×cos(n×3°)
  y = 200 + 100×sin(n×3°)

🔍 Visualisasi Rotasi

Frame 0: (400,200) →
Frame 1: (399.86,205.24) ↗
Frame 2: (399.46,210.47) ↗
...
Frame 30: (300,300) ↑ (sudut 90°)
...
Frame 60: (200,200) ← (sudut 180°)

⚡ Komponen Penting

1. **Objek Visual**:
   - Titik pusat (lingkaran hitam)
   - Titik merah yang berputar
   - Garis biru penghubung
2. **Informasi Real-time**:
   - Sudut dalam derajat dan radian

- Perhitungan posisi x dan y
- Kecepatan sudut

3. **Parameter Kunci**:
   angular_velocity = 3 *# Derajat per frame*
   radius = 100 *# Jarak dari pusat*
   update_interval = 50 *# ms*

💡 Karakteristik Simulasi

1. **Gerak Melingkar Uniform**:
   - Kecepatan sudut konstan
   - Lintasan berbentuk lingkaran sempurna
2. **Presisi Matematis**:
   - Menggunakan fungsi trigonometri
   - Konversi satuan sudut
3. **Periodisitas**:
   - Sudut di-reset setiap 360° (% 360)

```
#Simulasi Rotasi Sederhana - Angular Velocity Konstan
import tkinter as tk
import math
from vector import Vector

class App:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("🔄 Simulasi Rotasi Sederhana - Angular Velocity Konstan")

        self.canvas = tk.Canvas(self.root, width=600, height=400, bg="white")
        self.canvas.pack()

        # Pusat rotasi
        self.center = Vector(300, 200)
        self.radius = 100
        self.angle_deg = 0
        self.angular_velocity = 3  # derajat per frame

        # Objek visual
        self.linex = self.canvas.create_line(200, 200, 400, 200, fill="green", width=2)
        self.liney = self.canvas.create_line(300, 100, 300, 300, fill="gray", width=2)
        self.circle = self.canvas.create_oval(self.center.x - 5, self.center.y - 5,
self.center.x + 5, self.center.y + 5, fill="black")
        self.dot = self.canvas.create_oval(0, 0, 0, 0, fill="red")
        self.line = self.canvas.create_line(0, 0, 0, 0, fill="blue", width=2)
        self.text = self.canvas.create_text(10, 10, anchor="nw", text="", font=("Arial",
12), fill="black")

        self.update()
        self.root.mainloop()

    def update(self):
        angle_rad = math.radians(self.angle_deg)
        offset = Vector(math.cos(angle_rad), math.sin(angle_rad)).multed(self.radius)
        point = self.center.added(offset)

        # Gambar ulang titik dan garis
        self.canvas.coords(self.dot, point.x - 10, point.y - 10, point.x + 10, point.y +
10)
        self.canvas.coords(self.line, self.center.x, self.center.y, point.x, point.y)

        # Informasi teks
        info = f"""
🧿 ROTASI DENGAN KECEPATAN SUDUT KONSTAN

Sudut: {self.angle_deg:.1f}°
Sudut (radian): {angle_rad:.2f}
Kecepatan sudut (ω): {self.angular_velocity}°/frame
{offset} + {self.center} = {point}
x = cx + r × cos(θ) {offset}
  = {self.center.x} + {self.radius} × cos({self.angle_deg:.1f}°) = {point.x:.1f}
y = cy + r × sin(θ)
  = {self.center.y} + {self.radius} × sin({self.angle_deg:.1f}°) = {point.y:.1f}
"""
        self.canvas.itemconfig(self.text, text=info)
```

```
        # Tambah sudut
        self.angle_deg = (self.angle_deg + self.angular_velocity) % 360

        self.root.after(100, self.update)

if __name__ == "__main__":
    App()
```

🔢 2. Tahapan Simulasi Python Tkinter

Kita akan:

1. **Membuat tongkat** (baton) berbentuk garis.
2. Memutar tongkat tersebut dari tengahnya.
3. Menggunakan konversi dari **derajat ke radian**.
4. Menampilkan hasil perhitungan di layar.

🧮 3. Rumus Matematika

Konversi derajat ke radian:

radian = math.radians(degree) # atau: radian = 2 * PI * (degree / 360)

Untuk memutar tongkat:

- Titik tengah (x, y)
- Panjang tongkat L
- Ujung tongkat dihitung dengan:

x1 = x + L * cos(angle) >> sumbu x >> alas segitiga
y1 = y + L * sin(angle) >> sumbu y >> tinggi segitiga
x2 = x - L * cos(angle) >> sumbu x >> alas segitiga
y2 = y - L * sin(angle) >> sumbu y >> tinggi segitiga

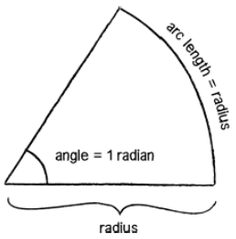


Figure 3.3

The formula to convert from degrees to radians is:

radians = 2 * PI * (degrees / 360)

📚 Kesimpulan

Hal yang Dipelajari	Penjelasan Singkat
Sudut (angle)	Digunakan untuk rotasi
Derajat & Radian	Dua satuan sudut (360° = 2π rad)
Rotasi	Dihitung dengan cos() dan sin()
Animasi	Menggunakan after() di Tkinter

Pi (π) adalah sebuah konstanta matematika yang menyatakan **perbandingan antara keliling lingkaran dan diameternya**. Artinya:

π=keliling lingkarandiameter lingkaran $\pi = \frac{\text{keliling lingkaran}}{\text{diameter lingkaran}}$

Nilai pi **tidak pernah berubah**, dan digunakan dalam banyak rumus yang berkaitan dengan lingkaran dan trigonometri.

✅ Nilai Pendekatan Pi:

π≈3.14159\pi \approx 3.14159

📏 Contoh Penggunaan Pi:

- Keliling lingkaran: $K = \pi \times d \quad (\text{d} = \text{diameter})$
- Luas lingkaran: $A = \pi \times r^2 \quad (\text{r} = \text{jari-jari})$

💻 Di dalam kode Python:

```
import math
print(math.pi)  # 3.141592653589793
```

Penjelasan singkat:

- Dalam script 1 ini **angular velocity** = 2 derajat per frame (konstan).
- Karena angular velocity tidak berubah, maka **angular acceleration = 0** (tidak ada percepatan sudut).
- Jadi rotasi bergerak dengan kecepatan sudut tetap (tidak makin cepat atau lambat).

Berikut contoh perhitungan **tiap frame** untuk simulasi rotasi baton dengan kecepatan sudut konstan, dijelaskan langkah demi langkah:

Kondisi Awal:

- **Pusat Baton**: (300, 200)
- **Panjang Baton**: 100 pixel
- **Sudut Awal (θ)**: 0°
- **Kecepatan Sudut (ω)**: 2°/frame
- **Percepatan Sudut (α)**: 0

Frame 1 (Sudut = 0°):

1. **Konversi ke Radian**:
   θ_rad = math.radians(0°) = 0
2. **Hitung Arah Unit Vector**:
   direction = Vector(cos(0), sin(0)) = Vector(1, 0) *# Mengarah ke kanan*
3. **Hitung Posisi Ujung Baton**:
   - Ujung 1 (kanan):
     end1 = center + (direction × length)
     = (300,200) + (1,0)×100
     = (400, 200)
   - Ujung 2 (kiri):
     end2 = center - (direction × length)
     = (300,200) - (1,0)×100
     = (200, 200)
4. **Update Sudut untuk Frame Berikutnya**:
   θ_baru = (0° + 2°) % 360 = 2°

Frame 2 (Sudut = 2°):

1. **Konversi ke Radian**:
   θ_rad = math.radians(2°) ≈ 0.0349 rad
2. **Hitung Arah Unit Vector**:
   direction = Vector(cos(0.0349), sin(0.0349)) ≈ Vector(0.999, 0.0349)
3. **Hitung Posisi Ujung Baton**:
   - Ujung 1:
     end1 = (300,200) + (0.999, 0.0349)×100 ≈ (399.9, 203.49)
   - Ujung 2:
     end2 = (300,200) - (0.999, 0.0349)×100 ≈ (200.1, 196.51)
4. **Update Sudut**:
   θ_baru = (2° + 2°) = 4°

Frame 3 (Sudut = 4°):

1. **Konversi ke Radian**:
   θ_rad = math.radians(4°) ≈ 0.0698 rad
2. **Hitung Arah Unit Vector**:
   direction ≈ Vector(0.998, 0.0698)
3. **Hitung Posisi Ujung Baton**:
   - Ujung 1 ≈ (300,200) + (0.998, 0.0698)×100 ≈ (399.8, 206.98)
   - Ujung 2 ≈ (300,200) - (0.998, 0.0698)×100 ≈ (200.2, 193.02)
4. **Update Sudut**:
   θ_baru = 6°

Visualisasi Pergerakan:

Frame	Sudut	Ujung 1 (x,y)	Ujung 2 (x,y)
1	0°	(400.0, 200.0)	(200.0, 200.0)
2	2°	(399.9, 203.49)	(200.1, 196.51)
3	4°	(399.8, 206.98)	(200.2, 193.02)

Frame	Sudut	Ujung 1 (x,y)	Ujung 2 (x,y)
...	...	...	...
90	180°	(200.0, 200.0)	(400.0, 200.0)

Kunci Perhitungan:

1. **Kecepatan Sudut Konstan**:
   - Setiap frame, sudut bertambah **2°** (tanpa percepatan).
   - Rumus:
     θ_baru = (θ_lama + ω) % 360
2. **Posisi Ujung Baton**:
   - Menggunakan **trigonometri (cos/sin)** untuk menghitung arah.
   - Rumus:
     ujung = pusat ± (panjang × Vector(cos(θ), sin(θ))
3. **Periodik 360°**:
   - Setelah 180 frame (360°/2°), baton kembali ke posisi awal.

Contoh Output Teks di Layar (Frame 2):
Sudut (derajat): 2.0°
Sudut (radian): 0.03
Angular Velocity: 2.00°/frame
Angular Acceleration: 0.00°/frame²
Ujung 1: (399.9, 203.5)
Ujung 2: (200.1, 196.5)
Program ini menunjukkan bagaimana **rotasi benda** dihitung dengan matematika trigonometri sederhana! 🔄

```
#SCRIpT 1 : Simulasi Rotasi Baton, angular velocity constan
import tkinter as tk
import math
from vector import Vector

# Class Baton (tongkat)
class Baton:
    def __init__(self, canvas, center, length):
        self.canvas = canvas
        self.center = center                # Vector posisi tengah
        self.length = length                # Panjang tongkat
        self.angle_deg = 0                  # Sudut awal (dalam derajat)
        self.angular_velocity = 2           # Kecepatan sudut (derajat/frame)
        self.angular_acceleration = 0       # Percepatan sudut (tetap nol)

        # Garis tongkat di canvas
        self.line = canvas.create_line(0, 0, 0, 0, width=8, fill="purple")
        self.linex = self.canvas.create_line(200, 200, 400, 200, fill="green", width=2)
        self.liney = self.canvas.create_line(300, 100, 300, 300, fill="gray", width=2)
        self.circle = self.canvas.create_oval(self.center.x - 5, self.center.y - 5,
self.center.x + 5, self.center.y + 5, fill="black")
        self.dot1 = self.canvas.create_oval(0, 0, 0, 0, fill="red")
        self.dot2 = self.canvas.create_oval(0, 0, 0, 0, fill="blue")
        # Teks informasi di sudut layar
        self.text = canvas.create_text(10, 10, anchor="nw", text="", font=("Arial", 12),
fill="black")

    def update(self):
        # Konversi sudut ke radian
        angle_rad = math.radians(self.angle_deg)

        # Buat arah unit vector berdasarkan sudut
        direction = Vector(math.cos(angle_rad), math.sin(angle_rad))

        # Hitung dua ujung tongkat berdasarkan pusat
        end1 = self.center.added(direction.multed(self.length))
        end2 = self.center.subbed(direction.multed(self.length))

        self.canvas.coords(self.dot1, end1.x - 10, end1.y - 10, end1.x + 10, end1.y + 10)
        self.canvas.coords(self.dot2, end2.x - 10, end2.y - 10, end2.x + 10, end2.y + 10)

        # Perbarui garis tongkat pada canvas
        self.canvas.coords(self.line, end1.x, end1.y, end2.x, end2.y)
```

```
        # Tampilkan informasi perhitungan pada layar
        info = f"""Sudut (derajat) red : {self.angle_deg:.1f}°
Sudut (radian): {angle_rad:.2f}
Angular Velocity: {self.angular_velocity:.2f}°/frame
Angular Acceleration: {self.angular_acceleration:.2f}°/frame²
Ujung 1 red: ({end1.x:.1f}, {end1.y:.1f})
Ujung 2 blue: ({end2.x:.1f}, {end2.y:.1f})
"""
        self.canvas.itemconfig(self.text, text=info)

        # Update sudut dengan angular velocity
        self.angle_deg = (self.angle_deg + self.angular_velocity) % 360

# Inisialisasi Tkinter dan kanvas
root = tk.Tk()
root.title("Simulasi Rotasi Baton - Nature of Code 3.1")

canvas = tk.Canvas(root, width=600, height=400, bg="white")
canvas.pack()

# Titik tengah canvas sebagai Vector
center = Vector(300, 200)

# Buat objek Baton
baton = Baton(canvas, center, length=100)

# Fungsi animasi
def animate():
    baton.update()
    root.after(100, animate)

animate()
root.mainloop()
```

Penjelasan:

- **angular_velocity**: Sudut bertambah 2° per frame → rotasi konstan.
- **angular_acceleration**: 0, karena tidak ada perubahan kecepatan sudut.
- angle_deg diperbarui dengan angle_deg + angular_velocity.
- canvas.coords() digunakan untuk menggerakkan garis tongkat sesuai perhitungan cos dan sin.
- Informasi sudut, kecepatan sudut, percepatan sudut, dan posisi ujung tongkat ditampilkan di atas canvas.

Angular Motion (Gerak Sudut) dari *The Nature of Code* Bab 3.2

💡 1. Penjelasan

Di Bab 1 dan 2 kita sudah belajar:
lokasi = lokasi + kecepatan
kecepatan = kecepatan + percepatan
Sekarang kita mempelajari **versi memutar** dari pergerakan itu:
sudut = sudut + kecepatan sudut
kecepatan sudut = kecepatan sudut + percepatan sudut

$$\overrightarrow{\text{velocity}} = \overrightarrow{\text{velocity}} + \overrightarrow{\text{acceleration}}$$

$$\overrightarrow{\text{position}} = \overrightarrow{\text{position}} + \overrightarrow{\text{velocity}}$$

You can apply exactly the same logic to a rotating object:

$$\text{angular velocity} = \text{angular velocity} + \text{angular acceleration}$$

$$\text{angle} = \text{angle} + \text{angular velocity}$$

Sudut (angle) adalah seberapa banyak objek sudah berputar. Sudut dihitung dalam **radian**, bukan derajat.
Contoh: jika kita punya tongkat (baton), kita bisa memutarnya pelan-pelan seperti jarum jam.

📐 2. Rumus dan Perhitungan

Rumus Gerak Sudut:

- angle = angle + angular_velocity → posisi rotasi sekarang
- angular_velocity = angular_velocity + angular_acceleration → perubahan kecepatan putar

Misal:

- angle = 0 (awalnya belum mutar)
- angular_velocity = 0.05 (cepat mutar 0.05 radian tiap frame)
- angular_acceleration = 0.001 (kecepatan mutarnya makin cepat)

🔁 ROTASI DENGAN PERCEPATAN SUDUT

📝 Rumus Fisika Rotasi

1. **Rotasi Vektor**:

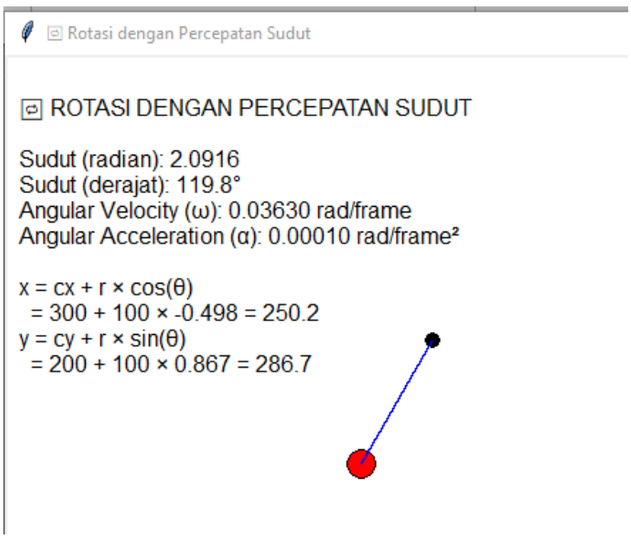
```
def rotated(self, angle_rad):
    return Vector(
        self.x*cos(θ) - self.y*sin(θ),
        self.x*sin(θ) + self.y*cos(θ)
    )
```

2. **Percepatan Sudut**:

   ω_new = ω_old + α
   θ_new = θ_old + ω_new

3. **Posisi Objek**:

   x = cx + r × cos(θ)
   y = cy + r × sin(θ)



🎞️ Perhitungan Frame-by-Frame

Parameter Awal:

- Pusat: (300, 200)
- Radius: 100 pixel
- Kecepatan sudut awal (ω): 0.03 rad/frame
- Percepatan sudut (α): 0.0001 rad/frame²
- Interval: 20ms/frame (~50 FPS)

Contoh Perhitungan 3 Frame:

Frame 0:

- θ = 0 rad
- ω = 0.03 rad/frame
- Posisi:
  x = 300 + 100×cos(0) = 400
  y = 200 + 100×sin(0) = 200

Frame 1 (20ms):

- Update ω: 0.03 + 0.0001 = 0.0301 rad/frame
- Update θ: 0 + 0.0301 = 0.0301 rad
- Posisi:
  x = 300 + 100×cos(0.0301) ≈ 399.955
  y = 200 + 100×sin(0.0301) ≈ 203.010

Frame 2 (40ms):

- ω: 0.0301 + 0.0001 = 0.0302 rad/frame
- θ: 0.0301 + 0.0302 = 0.0603 rad
- Posisi:
  x ≈ 300 + 100×cos(0.0603) ≈ 399.820
  y ≈ 200 + 100×sin(0.0603) ≈ 206.019

Frame n:

- ω = 0.03 + n×0.0001
- θ = Σω (jumlah kumulatif kecepatan sudut)

🔍 Visualisasi Percepatan

Frame 0: (400,200) →
Frame 1: (399.955,203.010) ↗
Frame 2: (399.820,206.019) ↗
...
Frame 100: ω ≈ 0.04 rad/frame (rotasi lebih cepat)

⚡ Komponen Penting

1. **Class Vector**:
   - Memiliki metode rotated() untuk transformasi
   - Menyimpan komponen x dan y
2. **Percepatan Sudut**:
   angular_velocity += angular_acceleration
   angle_rad += angular_velocity
3. **Informasi Real-time**:
   - Sudut (radian dan derajat)
   - Kecepatan dan percepatan sudut
   - Perhitungan posisi aktual

💡 Karakteristik Simulasi

1. **Gerak Rotasi Dipercepat**:
   - Kecepatan sudut bertambah secara konstan
   - Lintasan spiral (jika ada perubahan radius)
2. **Presisi Matematis**:
   - Menggunakan operasi vektor
   - Akumulasi sudut floating-point

3. **Periodisitas**:
    - Sudut di-reset setiap 2π radian

```
import tkinter as tk
import math
from vector import Vector

class Mover:
    def __init__(self, canvas, center, radius):
        self.canvas = canvas
        self.center = center
        self.radius = radius
        self.angle_rad = 0
        self.angular_velocity = 0.03
        self.angular_acceleration = 0.0001

        self.dot = self.canvas.create_oval(0, 0, 0, 0, fill="red")
        self.line = self.canvas.create_line(0, 0, 0, 0, fill="blue", width=2)
        self.text = self.canvas.create_text(10, 10, anchor="nw", text="", font=("Arial",
12), fill="black")

        self.update()

    def update(self):
        direction = Vector(math.cos(self.angle_rad), math.sin(self.angle_rad))
        point = self.center.added(direction.multed(self.radius))

        self.canvas.coords(self.dot, point.x - 10, point.y - 10, point.x + 10, point.y +
10)
        self.canvas.coords(self.line, self.center.x, self.center.y, point.x, point.y)

        info = f"""
🔁 ROTASI DENGAN PERCEPATAN SUDUT

Sudut (radian): {self.angle_rad:.4f}
Sudut (derajat): {math.degrees(self.angle_rad):.1f}°
Angular Velocity (ω): {self.angular_velocity:.5f} rad/frame
Angular Acceleration (α): {self.angular_acceleration:.5f} rad/frame²

x = cx + r × cos(θ) = {self.center.x} + {self.radius} × {direction.x:.3f} = {point.x:.1f}
y = cy + r × sin(θ) = {self.center.y} + {self.radius} × {direction.y:.3f} = {point.y:.1f}
"""
        self.canvas.itemconfig(self.text, text=info)

        self.angular_velocity += self.angular_acceleration
        self.angle_rad = (self.angle_rad + self.angular_velocity) % (2 * math.pi)

        self.canvas.after(20, self.update)

class App:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("🔁 Rotasi dengan Percepatan Sudut")

        self.canvas = tk.Canvas(self.root, width=600, height=400, bg="white")
        self.canvas.pack()

        center = Vector(300, 200)
        radius = 100

        # Garis referensi
        self.canvas.create_line(200, 200, 400, 200, fill="green", width=2)
        self.canvas.create_line(300, 100, 300, 300, fill="gray", width=2)
        self.canvas.create_oval(center.x - 5, center.y - 5, center.x + 5, center.y + 5,
fill="black")

        self.mover = Mover(self.canvas, center, radius)

        self.root.mainloop()

if __name__ == "__main__":
    App()
```

Simulasi Rotasi Baton, angular_velocity = 0.05, angular_acceleration = 0.001

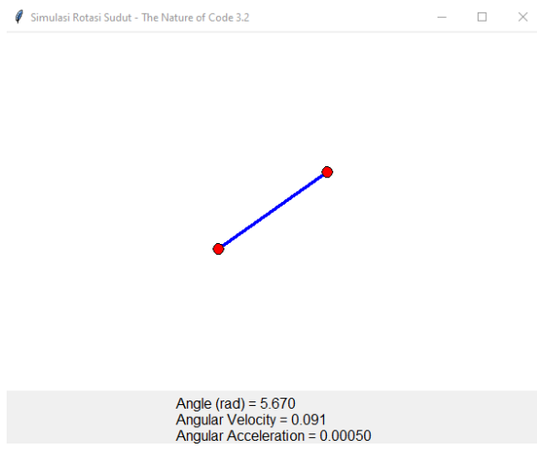
🧪 3. Simulasi Python Tkinter

Berikut adalah perhitungan untuk setiap frame dalam simulasi rotasi baton dengan parameter awal:

- Kecepatan sudut awal (angular_velocity) = 0.05 rad/frame
- Percepatan sudut (angular_acceleration) = 0.001 rad/frame²

Rumus yang digunakan setiap frame:

1. Kecepatan sudut baru: ω_new = ω_old + α
2. Sudut baru: θ_new = θ_old + ω_new



Mari kita hitung untuk 10 frame pertama:

Frame 0 (Initial State):

- Angle (θ) = 0 rad
- Angular Velocity (ω) = 0.05 rad/frame
- Angular Acceleration (α) = 0.001 rad/frame²

Frame 1:

- ω = 0.05 + 0.001 = 0.051
- θ = 0 + 0.051 = 0.051

Frame 2:

- ω = 0.051 + 0.001 = 0.052
- θ = 0.051 + 0.052 = 0.103

Frame 3:

- ω = 0.052 + 0.001 = 0.053
- θ = 0.103 + 0.053 = 0.156

Frame 4:

- ω = 0.053 + 0.001 = 0.054
- θ = 0.156 + 0.054 = 0.210

Frame 5:

- ω = 0.054 + 0.001 = 0.055
- θ = 0.210 + 0.055 = 0.265

Frame 6:

- ω = 0.055 + 0.001 = 0.056
- θ = 0.265 + 0.056 = 0.321

Frame 7:

- ω = 0.056 + 0.001 = 0.057
- θ = 0.321 + 0.057 = 0.378

Frame 8:

- ω = 0.057 + 0.001 = 0.058
- θ = 0.378 + 0.058 = 0.436

Frame 9:

- ω = 0.058 + 0.001 = 0.059
- θ = 0.436 + 0.059 = 0.495

Frame 10:

- ω = 0.059 + 0.001 = 0.060
- θ = 0.495 + 0.060 = 0.555

Dan seterusnya...

Pola umum setelah n frame:

- ω_n = 0.05 + (n × 0.001)
- θ_n = Σω_i untuk i dari 1 sampai n = 0.05n + 0.001×(n(n+1))/2

Contoh perhitungan posisi ujung baton (Frame 1):

- Panjang baton = 150 px
- Pusat di (300, 200)
- Arah vektor: (cos(0.051), sin(0.051)) ≈ (0.9987, 0.05096)
- Vektor setengah panjang: (150/2 × 0.9987, 150/2 × 0.05096) ≈ (74.90, 3.82)
- Ujung kiri: (300 - 74.90, 200 - 3.82) ≈ (225.10, 196.18)
- Ujung kanan: (300 + 74.90, 200 + 3.82) ≈ (374.90, 203.82)

Perhatikan bahwa:

1. Kecepatan sudut meningkat secara linear setiap frame
2. Sudut rotasi meningkat secara kuadratik (karena akumulasi kecepatan yang terus meningkat)
3. Posisi ujung baton dihitung menggunakan trigonometri berdasarkan sudut saat ini

Anda bisa melanjutkan perhitungan ini untuk frame-frame berikutnya dengan pola yang sama.

Berikut contoh implementasi dengan Vector2D, Baton, Canvas, dan komentar kode:

```
#SCRIPT 2 : Simulasi Rotasi Baton, angular_velocity = 0.05, angular_acceleration = 0.001
import tkinter as tk
import math
from vector import Vector
```

```
# ==========================
# Class Baton (Tongkat)
# ==========================
class Baton:
    def __init__(self, center, length):
        self.center = center                # Titik pusat rotasi (Vector)
        self.length = length               # Panjang tongkat

        self.angle = 0                     # Sudut awal (radian)
        self.angular_velocity = 0.05       # Kecepatan sudut
        self.angular_acceleration = 0.0005 # Percepatan sudut

        self.line = None                   # ID garis pada canvas
        self.circle1 = None                # Lingkaran ujung kiri
        self.circle2 = None                # Lingkaran ujung kanan

    def update(self):
        # Update sudut dan kecepatan sudut
        self.angle += self.angular_velocity
        self.angular_velocity += self.angular_acceleration

    def display(self, canvas):
        # Hitung arah rotasi sebagai unit vector
        direction = Vector(math.cos(self.angle), math.sin(self.angle))
        half = direction.multed(self.length / 2)

        # Ujung kiri dan kanan dari pusat
        end1 = self.center.subbed(half)
        end2 = self.center.added(half)

        if self.line is None:
            # Pertama kali buat objek canvas
            self.line = canvas.create_line(end1.x, end1.y, end2.x, end2.y, width=4,
fill="blue")
            self.circle1 = canvas.create_oval(end1.x-6, end1.y-6, end1.x+6, end1.y+6,
fill="red")
            self.circle2 = canvas.create_oval(end2.x-6, end2.y-6, end2.x+6, end2.y+6,
fill="red")
        else:
            # Update posisi objek
            canvas.coords(self.line, end1.x, end1.y, end2.x, end2.y)
            canvas.coords(self.circle1, end1.x-6, end1.y-6, end1.x+6, end1.y+6)
            canvas.coords(self.circle2, end2.x-6, end2.y-6, end2.x+6, end2.y+6)

        return self.angle, self.angular_velocity, self.angular_acceleration

# ==========================
# Fungsi Update Layar
# ==========================
def update():
    baton.update()
    angle, a_vel, a_acc = baton.display(canvas)

    # Tampilkan informasi di layar
    info_text = (
        f"Angle (rad) = {angle:.3f}\n"
        f"Angular Velocity = {a_vel:.3f}\n"
        f"Angular Acceleration = {a_acc:.5f}"
    )
    label_info.config(text=info_text)

    window.after(30, update)  # Loop animasi

# ==========================
# Setup Tkinter Window
# ==========================
WIDTH, HEIGHT = 600, 400
window = tk.Tk()
window.title("Simulasi Rotasi Sudut - The Nature of Code 3.2")

canvas = tk.Canvas(window, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

label_info = tk.Label(window, text="", font=("Arial", 12), justify="left")
label_info.pack()
```

```
# Buat baton di tengah layar
center = Vector(WIDTH / 2, HEIGHT / 2)
baton = Baton(center, length=150)

# Mulai animasi
update()
window.mainloop()
```

📝 4. Tahapan Script

Tahap	Penjelasan
Vector2D	Menyimpan koordinat tengah rotasi.
Baton	Menyimpan sudut, kecepatan, percepatan sudut, dan menggambar tongkat.
update()	Meningkatkan sudut dan kecepatan berdasarkan percepatan.
display()	Menggambar tongkat yang sudah diputar dengan math.cos(angle) dan math.sin(angle).
animate()	Loop Tkinter untuk memperbarui tampilan setiap 30ms.

🔢 5. Hasil Perhitungan di Layar

Contoh output yang muncul:
Angle (radian): 3.045
Angular Velocity: 0.082
Angular Acceleration: 0.001
Angle (derajat): 174.5°

📚 6. Hubungan dengan *The Nature of Code* – Bab 3.2

Simulasi ini menerapkan rumus dari buku:
angle = angle + angular velocity
angular velocity = angular velocity + angular acceleration

- Kita mengubah posisi rotasi tongkat seiring waktu.
- Konsep ini akan penting untuk simulasi seperti pendulum, gerak osilasi, dan objek berputar di game.

Apa Itu Trigonometri?

Trigonometry (trigonometri) adalah cabang matematika yang mempelajari hubungan antara sudut dan panjang sisi dalam segitiga, terutama segitiga siku-siku.

SOHCAHTOA – Kunci Utama

"SOHCAHTOA" adalah singkatan untuk membantu mengingat rumus dasar trigonometri:

1. **SOH**
   - **Sin(θ) = Opposite / Hypotenuse**
     (Sinus = Sisi Depan / Sisi Miring)
2. **CAH**
   - **Cos(θ) = Adjacent / Hypotenuse**
     (Cosinus = Sisi Samping / Sisi Miring)
3. **TOA**
   - **Tan(θ) = Opposite / Adjacent**
     (Tangen = Sisi Depan / Sisi Samping)

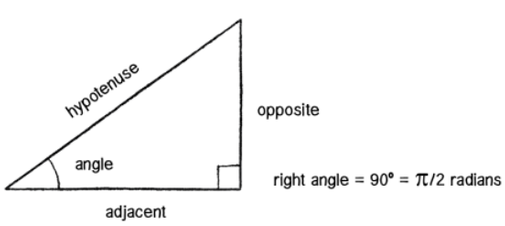
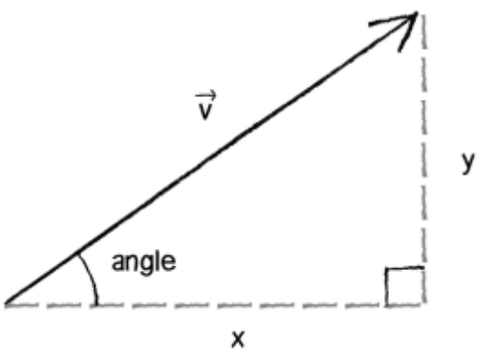


Figure 3.4

- ***soh***: sine = opposite / hypotenuse
- ***cah***: cosine = adjacent / hypotenuse
- ***toa***: tangent = opposite / adjacent



Contoh Visual (Segitiga Siku-Siku)
Bayangkan segitiga siku-siku dengan:

- **θ** (theta): salah satu sudut lancipnya.
- **Sisi Depan (Opposite)**: sisi di depan sudut θ.
- **Sisi Samping (Adjacent)**: sisi yang menyentuh sudut θ (bukan sisi miring).
- **Sisi Miring (Hypotenuse)**: sisi terpanjang, di depan sudut siku-siku.

Aplikasi dalam Vektor (Seperti di Komputer Grafis)

- Sebuah **vektor** (misal: panah) bisa dipecah jadi komponen **x** (horizontal) dan **y** (vertikal), membentuk segitiga siku-siku.
- Trigonometri membantu menghitung:
  - **Arah vektor** (sudut θ) menggunakan **tan(θ) = y/x**.
  - **Panjang vektor** (sisi miring) menggunakan Pythagoras: √(x² + y²).

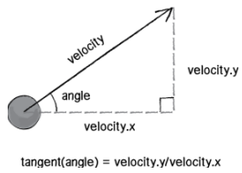
Contoh Soal Sederhana

Diketahui:

- Sisi Depan (y) = 3
- Sisi Samping (x) = 4

Hitung:

1. **Sudut θ** dengan **tangent**: $\tan(\theta) = \frac{\text{Opposite}}{\text{Adjacent}} = \frac{3}{4}$



Gunakan kalkulator: θ ≈ 36.87°.

2. **Sisi Miring (Hypotenuse)**: $\sqrt{3^2 + 4^2} = \sqrt{9 + 16} = 5$

Kenapa Penting?

Di dunia nyata, trigonometri digunakan untuk:

- Membuat game (gerakan karakter, rotasi benda).
- Menghitung jarak antar titik.
- Merancang bangunan, robotika, dan banyak lagi!

Mengarahkan Gerakan dengan Trigonometri

Bayangkan kamu ingin membuat mobil, pesawat, atau semut di layar yang bisa **berputar sesuai arah geraknya**. Ini penting karena benda-benda di dunia nyata selalu menghadap ke arah mereka bergerak!

Langkah 1: Velocity (Kecepatan) sebagai Vektor

- **Velocity** (velocity) adalah vektor yang menyimpan:
  - velocity.x: kecepatan horizontal (kiri/kanan).
  - velocity.y: kecepatan vertikal (atas/bawah).
- Contoh: Jika velocity = (3, 4), artinya benda bergerak 3 unit ke kanan dan 4 unit ke atas.

Langkah 2: Menghitung Sudut Rotasi

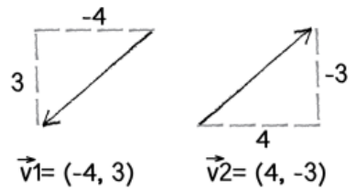
Kita butuh sudut (angle) untuk memutar benda. Gunakan **tangent** dari segitiga yang dibentuk oleh velocity:

Rumus dasar: $\tan(\text{angle}) = \frac{\text{velocity.y}}{\text{velocity.x}}$

Tapi kita butuh **sudut (angle)**-nya, bukan nilai tangennya. Solusinya: **inverse tangent** (arctangent atau atan).

Contoh:

- Jika velocity = (3, 4), maka: $\text{angle} = \arctan\left(\frac{4}{3}\right) \approx 53^\circ$



Masalah: atan() **Tidak Cukup**

Jika velocity = (-3, -4) (bergerak kiri-bawah), atan(-4/-3) juga menghasilkan 53°! Padahal seharusnya menghadap ke arah berlawanan (233°).

Penyebab: atan() hanya memberi sudut dalam rentang -90° sampai 90°, tanpa tahu kuadran asal vektor.

Solusi: Gunakan atan2()

Processing punya fungsi atan2(y, x) yang **otomatis menghitung sudut dengan benar** untuk semua kuadran.

Contoh:

1. velocity = (3, 4) → atan2(4, 3) ≈ 53°
2. velocity = (-3, -4) → atan2(-4, -3) ≈ -127° (atau 233°).

Kenapa Penting?

- **Tanpa rotasi**, mobil akan selalu menghadap kanan meski bergerak ke kiri.
- **Dengan** atan2(), benda selalu menghadap ke arah geraknya, seperti di dunia nyata!

Tips:

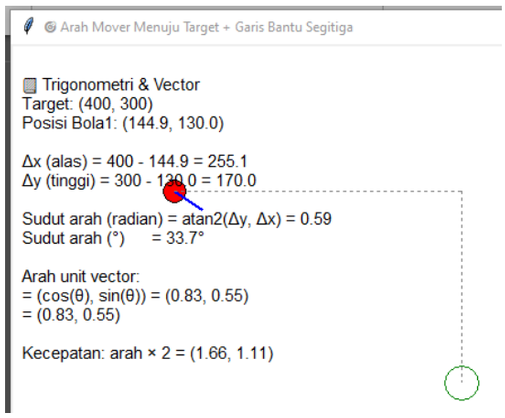
1. Selalu gunakan atan2(y, x), bukan atan(y/x).
2. Sudut di Processing menggunakan **radian**, tapi kamu bisa konversi ke derajat dengan degrees(angle).

Dengan ini, mobil, roket, atau karakter game-mu akan bergerak dengan **arah yang natural**! 🚗💨

Arah Mover Menuju Target

📝 Rumus dan Konsep Kunci

1. **Vektor Arah**:
   direction = target - posisi
2. **Normalisasi Vektor**:
   direction_unit = direction / magnitude(direction)
3. **Kecepatan**:
   velocity = direction_unit × speed
4. **Sudut Arah**:
   angle = atan2(delta_y, delta_x)
5. **Trigonometri Segitiga**:
   alas = target.x - posisi.x
   tinggi = target.y - posisi.y
   hypotenuse = √(alas² + tinggi²)



🎞️ Perhitungan Frame-by-Frame

Parameter:

- Kecepatan: 2 pixel/frame
- Interval: 30ms/frame (~33 FPS)
- Posisi awal: (100,100)
- Target: (400,300)

Contoh Perhitungan 3 Frame:

Frame 0:

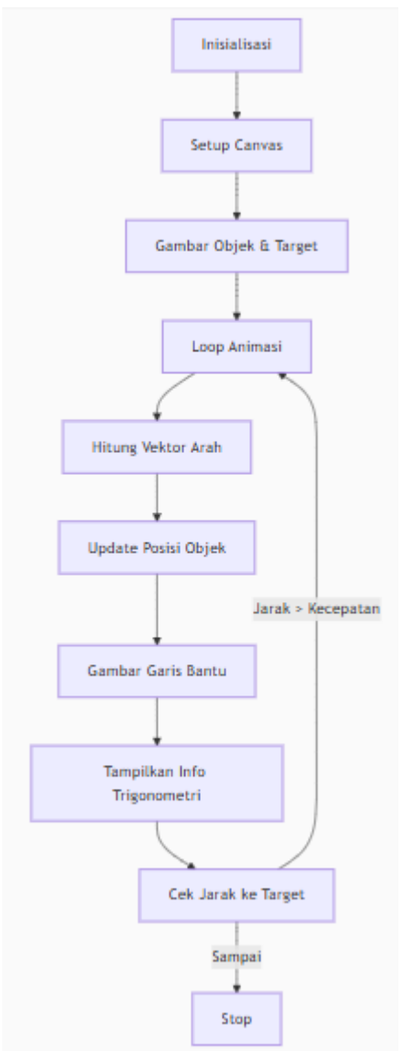
- Posisi: (100,100)
- Vektor arah: (400-100, 300-100) = (300,200)
- Magnitude: √(300²+200²) ≈ 360.56
- Vektor satuan: (300/360.56, 200/360.56) ≈ (0.832, 0.555)
- Kecepatan: (0.832×2, 0.555×2) ≈ (1.664, 1.110)

Frame 1:

- Posisi baru: (100+1.664, 100+1.110) ≈ (101.66,101.11)
- Vektor arah baru: (400-101.66, 300-101.11) ≈ (298.34,198.89)
- Proses normalisasi dan perhitungan kecepatan diulang

Frame 2:

- Posisi: (101.66+1.657, 101.11+1.102) ≈ (103.32,102.21)
- Vektor arah: (400-103.32, 300-102.21) ≈ (296.68,197.79)



🔍 Visualisasi Komponen

1. **Garis Biru**:
   - Menunjukkan arah dan besar vektor kecepatan
   - Panjang = 30 pixel (konstan untuk visualisasi)
2. **Garis Bantu Abu-abu**:
   - Garis alas: horizontal dari objek ke x target
   - Garis tinggi: vertikal dari ujung alas ke target
   - Membentuk segitiga siku-siku
3. **Informasi Trigonometri**:
   - Menampilkan perhitungan Δx, Δy, sudut, dan vektor

⚡ Karakteristik Pergerakan

1. **Gerak Lurus Beraturan**:
   - Kecepatan konstan 2 pixel/frame
   - Arah selalu menuju target
2. **Presisi Matematis**:
   - Menggunakan operasi vektor
   - Perhitungan floating-point presisi tinggi
3. **Kondisi Berhenti**:
   if direction.magnitude() > speed:
   # Lanjut animasi
   else:
   # Berhenti

```
import tkinter as tk
import math
from vector import Vector

# Tkinter setup
root = tk.Tk()
root.title("🎯 Arah Mover Menuju Target + Garis Bantu Segitiga")

canvas = tk.Canvas(root, width=600, height=400, bg="white")
```

```
canvas.pack()

# Bola1 (posisi awal) dan Bola2 (target)
pos1 = Vector(100, 100)
pos2 = Vector(400, 300)

# Objek visual
radius = 10
ball = canvas.create_oval(0, 0, 0, 0, fill="red")
target = canvas.create_oval(pos2.x - 15, pos2.y - 15, pos2.x + 15, pos2.y + 15,
outline="green")
line = canvas.create_line(0, 0, 0, 0, fill="blue", width=2)  # garis arah

# Garis bantu segitiga
line_alas = canvas.create_line(0, 0, 0, 0, fill="gray", dash=(4, 2))
line_tinggi = canvas.create_line(0, 0, 0, 0, fill="gray", dash=(4, 2))

# Teks
text = canvas.create_text(10, 10, anchor="nw", font=("Arial", 11), fill="black", text="")

# Kecepatan
speed = 2

def update():
    global pos1

    # Hitung arah dari bola1 ke bola2
    direction = pos2.subbed(pos1)
    angle = math.atan2(direction.y, direction.x)  # Sudut arah (radian)
    direction_unit = direction.normalized()
    velocity = direction_unit.multed(speed)
    pos1 = Vector(pos1.x + velocity.x, pos1.y + velocity.y)

    # Update posisi bola & arah
    canvas.coords(ball, pos1.x - radius, pos1.y - radius, pos1.x + radius, pos1.y +
radius)
    canvas.coords(line, pos1.x, pos1.y, pos1.x + direction_unit.x * 30, pos1.y +
direction_unit.y * 30)

    # ===== Garis Bantu Segitiga =====
    # Alas: horizontal dari pos1 ke target.x (tetap di y = pos1.y)
    canvas.coords(line_alas, pos1.x, pos1.y, pos2.x, pos1.y)
    # Tinggi: vertical dari target.x, pos1.y ke pos2
    canvas.coords(line_tinggi, pos2.x, pos1.y, pos2.x, pos2.y)

    # Info trigonometri
    delta_x = pos2.x - pos1.x
    delta_y = pos2.y - pos1.y

    info = f"""
📘 Trigonometri & Vector
Target: ({pos2.x}, {pos2.y})
Posisi Bola1: ({pos1.x:.1f}, {pos1.y:.1f})

Δx (alas) = {pos2.x} - {pos1.x:.1f} = {delta_x:.1f}
Δy (tinggi) = {pos2.y} - {pos1.y:.1f} = {delta_y:.1f}

Sudut arah (radian) = atan2(Δy, Δx) = {angle:.2f}
Sudut arah (°)      = {math.degrees(angle):.1f}°

Arah unit vector:
= (cos(θ), sin(θ)) = ({math.cos(angle):.2f}, {math.sin(angle):.2f})
= {direction_unit}

Kecepatan: arah × {speed} = {velocity}
"""
    canvas.itemconfig(text, text=info)

    # Ulangi jika belum sampai
    if direction.magnitude() > speed:
        root.after(30, update)

update()
root.mainloop()
```

Simulasi Mover Mengejar Target Bergerak

📝 Rumus dan Konsep Kunci

1. **Vektor Arah**:

direction = target_position - mover_position

2. **Normalisasi Vektor**:

direction_unit = direction / magnitude(direction)

3. **Kecepatan Mover**:

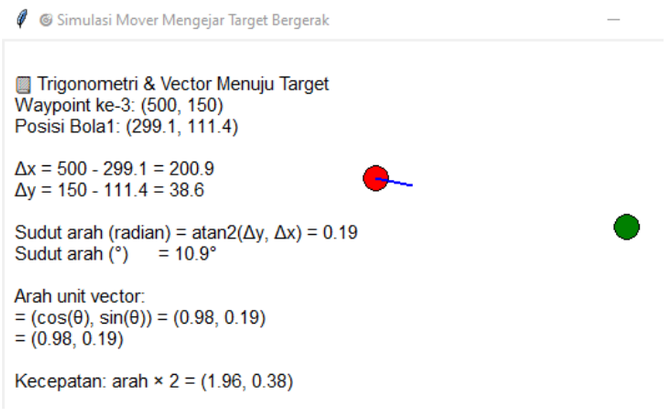
velocity = direction_unit × speed

4. **Sudut Arah**:

angle = atan2(direction.y, direction.x)

5. **Waypoint Management**:

if distance ≤ speed:
  waypoint_index = (waypoint_index + 1) % total_waypoints
  target_position = waypoints[waypoint_index]



🎞️ Perhitungan Frame-by-Frame

Parameter:

- Kecepatan mover: 2 pixel/frame
- Interval update: 30ms/frame (~33 FPS)
- Waypoints: [(250,300), (250,100), (500,150), (100,350), (400,350)]

Contoh Perhitungan:

Frame 0 (Waypoint 1: (250,300)):

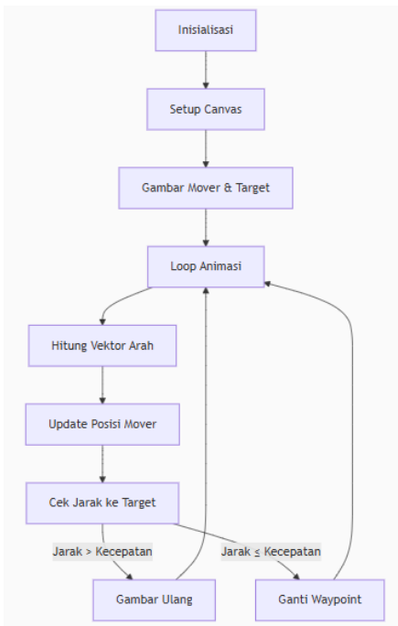
- Posisi mover: (100,100)
- Vektor arah: (250-100, 300-100) = (150,200)
- Magnitude: √(150²+200²) = 250
- Vektor satuan: (150/250, 200/250) = (0.6, 0.8)
- Kecepatan: (0.6×2, 0.8×2) = (1.2, 1.6)
- Posisi baru: (100+1.2, 100+1.6) = (101.2,101.6)

Frame 1:

- Vektor arah baru: (250-101.2, 300-101.6) ≈ (148.8,198.4)
- Normalisasi dan perhitungan kecepatan diulang

Frame n (Sampai Target):

- Ketika jarak ≤ 2 pixel:
  - Ganti waypoint ke berikutnya
  - Hitung vektor arah baru ke waypoint baru



🔍 Komponen Visual

1. **Mover (Bola Merah)**:
   - Bergerak dengan kecepatan konstan menuju target
   - Posisi diupdate setiap frame
2. **Target (Bola Hijau)**:
   - Berpindah ke waypoint berikutnya saat dicapai
   - Waypoint membentuk siklus tertutup
3. **Garis Biru**:
   - Menunjukkan arah gerakan mover
   - Panjang konstan 30 pixel

⚡ Karakteristik Pergerakan

1. **Gerak Target Diskrit**:
   - Target berpindah secara tiba-tiba antar waypoint
   - Mover akan mengikuti secara smooth
2. **Prediksi Posisi**:

 # Jika ingin prediksi waktu tempuh:
frames_needed = distance / speed

```
#4. Simulasi Mover Mengejar Target Bergerak
import tkinter as tk
import math
from vector import Vector

# Setup Tkinter
root = tk.Tk()
root.title("🎯 Simulasi Mover Mengejar Target Bergerak")

canvas = tk.Canvas(root, width=600, height=400, bg="white")
canvas.pack()
```

```
# Bola1 (mover) dan bola2 (target)
pos1 = Vector(100, 100)
waypoints = [Vector(250, 300), Vector(250, 100), Vector(500, 150), Vector(100, 350),
Vector(400, 350)]
waypoint_index = 0
pos2 = waypoints[waypoint_index]

# Visual
radius = 10
ball = canvas.create_oval(0, 0, 0, 0, fill="red")
target = canvas.create_oval(pos2.x - 10, pos2.y - 10, pos2.x + 10, pos2.y + 10,
fill="green")
line = canvas.create_line(0, 0, 0, 0, fill="blue", width=2)
text = canvas.create_text(10, 10, anchor="nw", font=("Arial", 11), fill="black", text="")

# Kecepatan bola1
speed = 2

def update():
    global pos1, pos2, waypoint_index

    # Hitung arah dari bola1 ke bola2
    direction = pos2.subbed(pos1)
    distance = direction.magnitude()
    angle = math.atan2(direction.y, direction.x)
    direction_unit = direction.normalized()
    velocity = direction_unit.multed(speed)

    # Update posisi bola1
    if distance > speed:
        pos1 = Vector(pos1.x + velocity.x, pos1.y + velocity.y)
    else:
        # Jika bola1 sudah dekat, pindah ke waypoint berikutnya
        waypoint_index = (waypoint_index + 1) % len(waypoints)
        pos2 = waypoints[waypoint_index]
        canvas.coords(target, pos2.x - 10, pos2.y - 10, pos2.x + 10, pos2.y + 10)

    # Update tampilan bola dan garis
    canvas.coords(ball, pos1.x - radius, pos1.y - radius, pos1.x + radius, pos1.y +
radius)
    canvas.coords(line, pos1.x, pos1.y, pos1.x + direction_unit.x * 30, pos1.y +
direction_unit.y * 30)

    # Informasi perhitungan
    info = f"""
📘 Trigonometri & Vector Menuju Target
Waypoint ke-{waypoint_index + 1}: ({pos2.x}, {pos2.y})
Posisi Bola1: ({pos1.x:.1f}, {pos1.y:.1f})

Δx = {pos2.x} - {pos1.x:.1f} = {pos2.x - pos1.x:.1f}
Δy = {pos2.y} - {pos1.y:.1f} = {pos2.y - pos1.y:.1f}

Sudut arah (radian) = atan2(Δy, Δx) = {angle:.2f}
Sudut arah (°)      = {math.degrees(angle):.1f}°

Arah unit vector:
= (cos(θ), sin(θ)) = ({math.cos(angle):.2f}, {math.sin(angle):.2f})
= {direction_unit}

Kecepatan: arah × {speed} = {velocity}
"""
    canvas.itemconfig(text, text=info)

    # Ulangi setiap 30ms
    root.after(30, update)

update()
root.mainloop()
```

Mover Following Mouse

📐 Rumus Utama

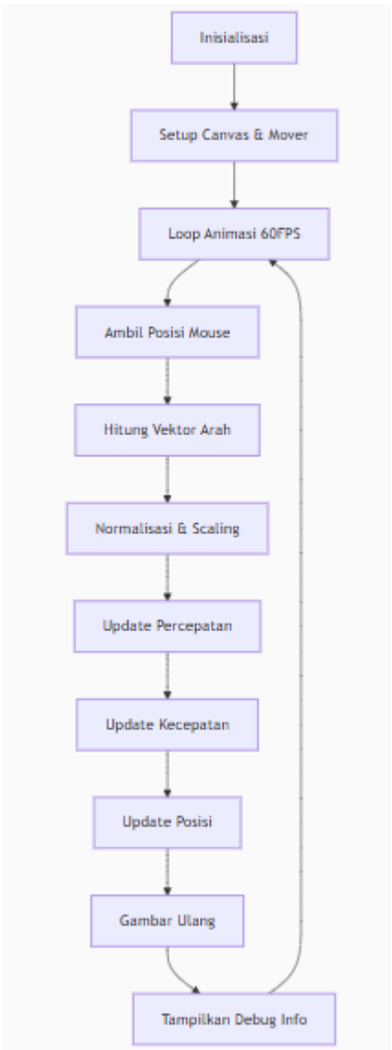
1. **Vektor Arah**:

direction = mouse_position - mover_position

2. **Normalisasi & Scaling**:

direction_normalized = direction.normalize() * 0.2

3. **Fisika Gerak**:
   acceleration = direction_normalized
   velocity += acceleration
   velocity.limit(topspeed)
   position += velocity



📊 Perhitungan Tiap Frame (16ms ≈ 60FPS)

Parameter:

- topspeed = 5 (kecepatan maksimum)
- radius = 24 (ukuran mover)
- Scaling factor: 0.2

Contoh Frame 1:

1. **Posisi Awal**:
   - Mover: (320, 240)
   - Mouse: (400, 300)
2. **Hitung Arah**:

direction = (400,300) - (320,240) = (80,60)

3. **Normalisasi**:

magnitude = sqrt(80² + 60²) = 100
normalized = (80/100, 60/100) = (0.8, 0.6)

4. **Scaling**:

scaled = (0.8, 0.6) * 0.2 = (0.16, 0.12)

5. **Update Fisika**:

acceleration = (0.16, 0.12)
velocity = (0,0) + (0.16,0.12) = (0.16,0.12) # Belum capai topspeed
position = (320,240) + (0.16,0.12) ≈ (320.16,240.12)

Contoh Frame 2:

1. **Posisi Baru**:
   - Mover: (320.16, 240.12)
   - Mouse: (405, 305) # Mouse bergerak
2. **Hitung Arah Baru**:

direction = (405,305) - (320.16,240.12) ≈ (84.84,64.88)

3. **Normalisasi & Scaling**:

magnitude ≈ sqrt(84.84² + 64.88²) ≈ 106.87
scaled ≈ (84.84/106.87, 64.88/106.87) * 0.2 ≈ (0.16,0.12)

📊 Visualisasi Gerakan

Frame 0: Mover di pusat (320,240), velocity = (0,0)
Frame 1: Mover bergerak ke (320.16,240.12)
Frame n: Mover terus mempercepat menuju mouse

⚙️ Implementasi Kode Kunci

```
def update(self):
  # 1. Ambil posisi mouse
  mouse = Vector(mouse_x, mouse_y)

  # 2. Hitung vektor arah
  dir = mouse.subbed(self.position).normalized().multed(0.2)

  # 3. Update fisika
  self.acceleration = dir
  self.velocity.add(self.acceleration)
  self.velocity.limit(self.topspeed)
  self.position.add(self.velocity)

  # 4. Update visual
  self.canvas.coords(self.shape, x-radius, y-radius, x+radius, y+radius)
```

🔍 Analisis Perilaku

1. **Gerakan Smooth**:
   - Percepatan konstan (0.2) membuat gerakan halus
   - Kecepatan dibatasi untuk menghindari gerakan instan
2. **Debug Info**:
   - Menampilkan posisi mouse dan mover
   - Menampilkan vektor arah, normalisasi, dan kecepatan
3. **Karakteristik**:
   - Mover akan terus berakselerasi selama mouse bergerak
   - Membentuk pola pursuit curve (kurva pengejaran)

```
import tkinter as tk
```

```
from vector import Vector

class Mover:
    def __init__(self, canvas, width, height):
        self.canvas = canvas
        self.width = width
        self.height = height
        self.position = Vector(width / 2, height / 2)
        self.velocity = Vector(0, 0)
        self.acceleration = Vector(0, 0)
        self.topspeed = 5
        self.radius = 24
        self.shape = self.canvas.create_oval(0, 0, 0, 0, fill='lightblue',
outline='black', width=2)
        # Info teks kosong
        self.mouse_pos = canvas.create_text(10, 10, anchor='nw', text="", font=('Arial',
10), fill='black')
        self.mover_pos = canvas.create_text(10, 30, anchor='nw', text="", font=('Arial',
10), fill='black')
        self.mover_dir = canvas.create_text(10, 50, anchor='nw', text="", font=('Arial',
10), fill='black')
        self.mover_nor = canvas.create_text(10, 70, anchor='nw', text="", font=('Arial',
10), fill='black')
        self.mover_velo = canvas.create_text(10, 90, anchor='nw', text="", font=('Arial',
10), fill='black')
        self.update()

    def update(self):
        # Dapatkan posisi mouse relatif ke canvas
        mouse_x = self.canvas.winfo_pointerx() - self.canvas.winfo_rootx()
        mouse_y = self.canvas.winfo_pointery() - self.canvas.winfo_rooty()
        mouse = Vector(mouse_x, mouse_y)
        # Update teks kosong dengan info posisi (x,y)
        self.canvas.itemconfig(self.mouse_pos, text=f"mouse: ({mouse})")
        self.canvas.itemconfig(self.mover_pos, text=f"Mover: ({self.position})")
        # Hitung arah, normalisasi, dan scaling
        dir = mouse.subbed(self.position)
        self.canvas.itemconfig(self.mover_dir, text=f"Arah: ({dir})")
        dir = dir.normalized().multed(0.2)
        self.canvas.itemconfig(self.mover_nor, text=f"Normalisi: ({dir})")

        # Percepatan dan kecepatan
        self.acceleration = dir
        self.velocity.add(self.acceleration)
        self.velocity.limit(self.topspeed)
        self.canvas.itemconfig(self.mover_velo, text=f"velocity: ({self.velocity})")
        self.position.add(self.velocity)

        # Update koordinat lingkaran
        x, y = self.position.x, self.position.y
        r = self.radius
        self.canvas.coords(self.shape, x - r, y - r, x + r, y + r)

        self.canvas.after(16, self.update)  # ~60 FPS


class App:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Mover Following Mouse - The Nature of Code")

        width, height = 640, 480
        self.canvas = tk.Canvas(self.root, width=width, height=height, bg='white')
        self.canvas.pack()

        self.mover = Mover(self.canvas, width, height)

        self.root.mainloop()

if __name__ == "__main__":
    App()
```

Sistem Koordinat Polar vs. Cartesian Coordinates

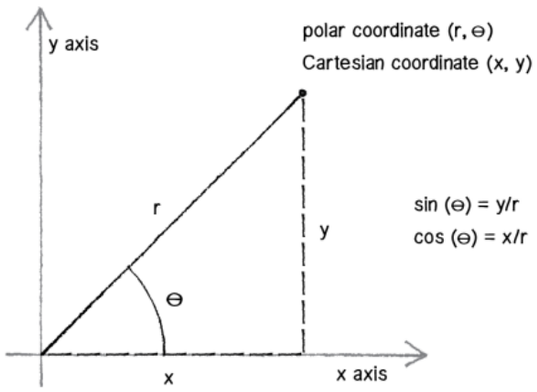
dua cara berbeda untuk menentukan posisi suatu titik, seperti yang digunakan dalam pemrograman dan matematika.

1. Koordinat Kartesius (Cartesian Coordinates)
   - Ini adalah sistem yang biasa kamu lihat di pelajaran matematika
   - Menggunakan sumbu X (horizontal) dan Y (vertikal) untuk menentukan posisi
   - Contoh: (3, 4) berarti:
     - 3 satuan ke kanan (sumbu X)
     - 4 satuan ke atas (sumbu Y)
   - Seperti alamat rumah di peta: jalan (X) dan nomor (Y)
2. Koordinat Polar (Polar Coordinates)
   - Sistem ini menggunakan sudut dan jarak untuk menentukan posisi
   - Contoh: (5, 30°) berarti:
     - 5 satuan jarak dari pusat (0,0)
     - 30° dari sumbu X positif
   - Seperti memberi petunjuk arah: "jalan 5 meter ke arah jam 1"

Perbandingan

- **Kartesius**: Bagus untuk gerakan lurus (atas/bawah, kiri/kanan)
- **Polar**: Bagus untuk gerakan melingkar atau spiral

Konversi Polar ke Kartesius



Karena komputer hanya mengerti koordinat Kartesius, kita perlu mengubah Polar ke Kartesius dengan rumus:

$x = r \times \cos(\theta)$

$y = r \times \sin(\theta)$

Dimana:

- r = jarak dari pusat
- θ (theta) = sudut dalam radian
- cos = cosinus
- sin = sinus

Contoh Praktis

Misal kita punya:

- r = 75
- θ = 45° (atau π/4 radian)

Maka:

```
x = 75 × cos(45°) ≈ 75 × 0,707 ≈ 53
y = 75 × sin(45°) ≈ 75 × 0,707 ≈ 53
```

Jadi koordinat Kartesiusnya kira-kira (53, 53)

Keuntungan Koordinat Polar

Kalau mau buat benda bergerak melingkar:

1. Dengan Kartesius: Hitung x dan y terus menerus - ribet!
2. Dengan Polar: Cukup tambah sudutnya saja (θ), r tetap
   Contoh kode sederhana:

```
theta = 0  # sudut mulai dari 0
r = 100    # jarak tetap 100
def update():
  global theta
  theta += 0.01  # tambah sudut sedikit
  x = r * cos(theta)
  y = r * sin(theta)
  # gambar benda di posisi (x,y)
```

   Jadi benda akan bergerak melingkar dengan sendirinya!

🎯 **Tujuan Simulasi**

- Menggambar lingkaran yang **berputar** mengelilingi pusat layar.

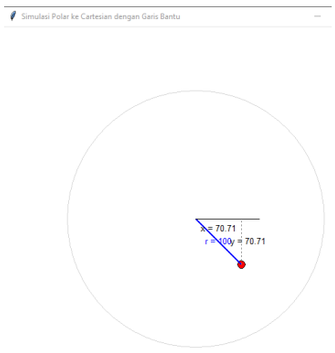
- Menggunakan **koordinat polar** dan mengonversinya ke **kartesian**: $x = r \cdot \cos(\theta) \quad \text{dan} \quad y = r \cdot \sin(\theta)$

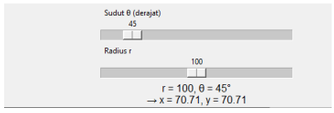
```
# SCRIPT 5 : Simulasi Konversi Koordinat Polar ke Kartesian
import tkinter as tk
import math

# === Setup Tkinter ===
root = tk.Tk()
root.title("Example 3.4: Polar to Cartesian")

WIDTH, HEIGHT = 640, 240
canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

# === Inisialisasi variabel polar ===
r = HEIGHT * 0.45     # Jari-jari (radius)
theta = 0             # Sudut awal

# Objek lingkaran dan garis
circle_id = canvas.create_oval(0, 0, 0, 0, fill="gray", outline="black", width=2)
line_id = canvas.create_line(0, 0, 0, 0, fill="black", width=2)

def animate():
    global theta

    canvas.delete("text")
    canvas.configure(bg="white")

    # Konversi polar ke kartesian
    x = r * math.cos(theta)
    y = r * math.sin(theta)

    # Pusat koordinat di tengah canvas
    center_x = WIDTH / 2
    center_y = HEIGHT / 2

    # Koordinat akhir (x, y) ditranslasi ke posisi canvas
    px = center_x + x
    py = center_y + y

    # Update garis dari pusat ke titik
    canvas.coords(line_id, center_x, center_y, px, py)

    # Update lingkaran di posisi (px, py)
    radius = 24
    canvas.coords(circle_id, px - radius, py - radius, px + radius, py + radius)

    # Tampilkan data
    canvas.create_text(10, 10, anchor="nw", text=f"theta = {theta:.2f} rad",
font=("Arial", 12), tag="text")
    canvas.create_text(10, 70, anchor="nw", text=f"theta = {math.degrees(theta):.2f}
degree", font=("Arial", 12), tag="text")
    canvas.create_text(10, 30, anchor="nw", text=f"x = {x:.1f}", font=("Arial", 12),
tag="text")
    canvas.create_text(10, 50, anchor="nw", text=f"y = {y:.1f}", font=("Arial", 12),
tag="text")

    # Update sudut theta untuk rotasi
    theta += 0.02

    root.after(30, animate)

animate()
root.mainloop()
```

Konsep	**Penjelasan Sederhana**
Koordinat Polar	Menentukan posisi berdasarkan **jarak** dari pusat dan **sudut rotasi**.
Konversi ke x, y	Rumus: x = r × cos(θ), y = r × sin(θ)
Sudut (θ)	Semakin besar, lingkaran akan **berputar mengelilingi titik tengah**.
Lingkaran	Lingkaran digambar di posisi (x, y) yang sudah dikonversi.

Konsep	Penjelasan Sederhana
Garis	Menghubungkan titik tengah ke lingkaran.

Simulasi Polar ke Cartesian dengan Garis Bantu (COPY PASTE)






```
import tkinter as tk
import math

WIDTH, HEIGHT = 600, 600
CENTER_X, CENTER_Y = WIDTH // 2, HEIGHT // 2

root = tk.Tk()
root.title("Simulasi Polar ke Cartesian dengan Garis Bantu")

canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

# ---------- Lingkaran utama ----------
circle_radius = 200
canvas.create_oval(CENTER_X - circle_radius, CENTER_Y - circle_radius,
                   CENTER_X + circle_radius, CENTER_Y + circle_radius,
                   outline="lightgray")

# ---------- Sliders ----------
theta_scale = tk.Scale(root, from_=0, to=360, label="Sudut θ (derajat)",
orient="horizontal", length=300)
theta_scale.set(45)
theta_scale.pack()

radius_scale = tk.Scale(root, from_=0, to=200, label="Radius r", orient="horizontal",
length=300)
radius_scale.set(100)
radius_scale.pack()

# ---------- Label hasil koordinat ----------
label_coords = tk.Label(root, text="", font=("Arial", 12))
label_coords.pack()

# ---------- Objek Canvas ----------
point = canvas.create_oval(0, 0, 0, 0, fill="red")
line_r = canvas.create_line(0, 0, 0, 0, fill="blue", width=2)         # garis radius r
line_x = canvas.create_line(0, 0, 0, 0, fill="gray", dash=(4, 2))     # proyeksi ke sumbu
x
line_baseline = canvas.create_line(0, 0, 0, 0, fill="black", width=1) # sumbu x (θ = 0°)

# ---------- Label garis panjang ----------
text_r = canvas.create_text(0, 0, text="", fill="blue", font=("Arial", 10))
text_x = canvas.create_text(0, 0, text="", fill="black", font=("Arial", 10))
text_y = canvas.create_text(0, 0, text="", fill="black", font=("Arial", 10))

# ---------- Fungsi Update ----------
def update():
    theta_deg = theta_scale.get()
    r = radius_scale.get()
    theta_rad = math.radians(theta_deg)
```

```
    # Hitung posisi Cartesian
    x = r * math.cos(theta_rad)
    y = r * math.sin(theta_rad)
    screen_x = CENTER_X + x
    screen_y = CENTER_Y + y

    # Update titik
    r_dot = 6
    canvas.coords(point,
                  screen_x - r_dot, screen_y - r_dot,
                  screen_x + r_dot, screen_y + r_dot)

    # Garis radius (r)
    canvas.coords(line_r, CENTER_X, CENTER_Y, screen_x, screen_y)

    # Garis proyeksi Y (vertikal ke sumbu X)
    canvas.coords(line_x, screen_x, screen_y, screen_x, CENTER_Y)

    # Garis dasar sumbu X dari center ke sudut 0° (baseline)
    canvas.coords(line_baseline, CENTER_X, CENTER_Y, CENTER_X + r, CENTER_Y)

    # Update label
    label_coords.config(
        text=f"r = {r}, θ = {theta_deg}°\n→ x = {x:.2f}, y = {y:.2f}"
    )

    # Teks panjang garis-garis
    canvas.coords(text_r, (CENTER_X + screen_x) / 2, (CENTER_Y + screen_y) / 2)
    canvas.itemconfig(text_r, text=f"r = {r}")

    canvas.coords(text_y, screen_x + 10, (CENTER_Y + screen_y) / 2)
    canvas.itemconfig(text_y, text=f"y = {y:.2f}")

    canvas.coords(text_x, (CENTER_X + screen_x) / 2, CENTER_Y + 15)
    canvas.itemconfig(text_x, text=f"x = {x:.2f}")

    root.after(33, update)

update()
root.mainloop()
```

Exercise 3.4: Spiral Path

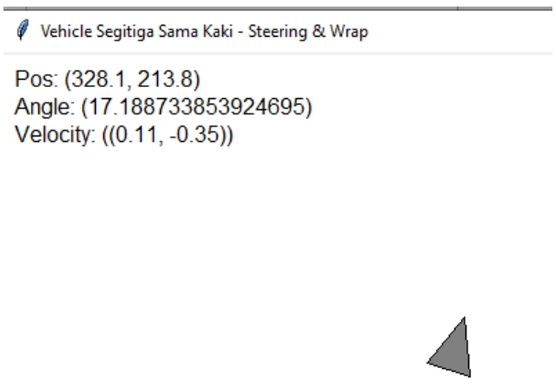
Konsep	**Penjelasan Sederhana**
Koordinat Polar	Posisi ditentukan oleh jarak (r) dari pusat dan sudut (θ) terhadap sumbu X.
Spiral	Dengan menambah **sudut** dan **radius** setiap saat, titik bergerak membentuk spiral.
Canvas Berbasis Tengah	Titik (0, 0) diubah jadi **tengah layar**, agar spiral berputar dari tengah.
Looping	animate() dipanggil terus-menerus untuk menggambar titik baru di jalur spiral.

r = 0; // radius awal
r += 0.05; // radius bertambah terus → spiral!

```
# SCRIPT 7 : Simulasi jalur spiral menggunakan koordinat polar
import tkinter as tk
import math

# Setup Tkinter
root = tk.Tk()
root.title("Exercise 3.4: Spiral Path")

WIDTH, HEIGHT = 640, 240
canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

# === Inisialisasi polar ===
r = 0             # Radius mulai dari 0 (untuk spiral)
theta = 0         # Sudut awal
dot_radius = 5    # Ukuran titik spiral

# Simpan semua titik spiral
spiral_points = []
```

```
def animate():
    global theta, r

    # Konversi polar ke kartesian
    x = r * math.cos(theta)
    y = r * math.sin(theta)

    # Geser titik ke tengah canvas
    px = WIDTH / 2 + x
    py = HEIGHT / 2 + y

    # Simpan titik spiral
    spiral_points.append((px, py))

    # Gambar semua titik spiral
    for px, py in spiral_points:
        canvas.create_oval(px - dot_radius, py - dot_radius,
                           px + dot_radius, py + dot_radius,
                           fill="black", outline="")

    # Tambah sudut dan radius
    theta += 0.1   # makin besar = makin cepat berputar
    r += 0.3        # makin besar = spiral makin melebar

    # #Info teks
    # canvas.create_text(10, 10, anchor="nw",
    #                    text=f"theta: {theta:.2f} | radius: {r:.2f}",
    #                    font=("Arial", 10), fill="blue", tag="info")

    root.after(30, animate)

animate()
root.mainloop()
```

Exercise 3.5 - Spaceship / Asteroids



📐 Rumus Utama

1. Rotasi:

heading += turn_angle # -0.05 (kiri) atau +0.05 (kanan)

2. Thrust (Propulsi):

force = Vector(cos(heading-π/2), sin(heading-π/2)) * 0.1
acceleration += force

3. Fisika Gerak:

velocity += acceleration
velocity *= damping (0.995)
velocity.limit(topspeed=6)
position += velocity

4. Wrap Around:

if position.x > width + buffer → position.x = -buffer
if position.x < -buffer → position.x = width + buffer

🎞️ Perhitungan Tiap Frame (30ms ≈ 33FPS)

Parameter:

- damping = 0.995
- topspeed = 6
- r = 16 (ukuran vehicle)
- turn_angle = ±0.05 rad/frame

Contoh Frame 1 (Tombol Up + Right ditekan):

1. Input:
    - thrust() dan turn(0.05)
2. Hitung Gaya Thrust:
    - Jika heading = 0:
      force_angle = 0 - π/2 = -π/2
      force = (cos(-π/2), sin(-π/2)) * 0.1 ≈ (0, -1) * 0.1 = (0, -0.1)
3. Update Velocity:
   velocity = (0,0) + (0,-0.1) = (0,-0.1)
   velocity *= 0.995 ≈ (0,-0.0995)
4. Update Posisi:
   position = (320,240) + (0,-0.0995) ≈ (320,239.9005)
5. Update Heading:

heading = 0 + 0.05 ≈ 0.05 rad (≈2.86°)

6. Hitung Titik Segitiga:
   front = (0, -24).rotated(0.05) ≈ (1.2, -23.99)
   left = (-16, 16).rotated(0.05) ≈ (-16.4, 15.6)
   right = (16, 16).rotated(0.05) ≈ (15.6, 16.4)

📊 Visualisasi Gerakan

Frame 0:
Posisi: (320,240)
Heading: 0 rad
Velocity: (0,0)

Frame 1:
Posisi: (320,239.9)
Heading: 0.05 rad
Velocity: (0,-0.1)

Frame 30 (~1 detik):
Posisi: (320,237)
Heading: 1.5 rad (≈85.9°)
Velocity: (0,-0.74)

⚙️ Implementasi Kode Kunci

```
def update(self):
    # Fisika
    self.velocity.add(self.acceleration)
    self.velocity.mult(self.damping)
    self.velocity.limit(self.topspeed)
    self.position.add(self.velocity)
    self.acceleration = Vector(0,0)  # Reset
def thrust(self):
    force = Vector.fromAngle(self.heading - math.pi/2).multed(0.1)
    self.applyForce(force)
    self.thrusting = True  # Untuk visual
def show(self):
    # Hitung titik segitiga
    front = Vector(0, -1.5*self.r).rotated(self.heading)
    left = Vector(-self.r, self.r).rotated(self.heading)
    right = Vector(self.r, self.r).rotated(self.heading)
    # Update visual
    self.canvas.coords(self.id, front.x,front.y, left.x,left.y, right.x,right.y)
```

🔍 Analisis Perilaku

1. Damping Effect:
   - Kecepatan berkurang 0.5% tiap frame (0.995)
   - Membuat vehicle perlahan berhenti saat tidak ada thrust
2. Gerakan Spiral:
   - Kombinasi thrust + turn menghasilkan lintasan spiral
   - Radius spiral tergantung rasio thrust vs turn rate
3. Wrap Around:
   - Vehicle muncul di sisi berlawanan saat keluar canvas
   - Buffer 2*r mencegah flicker visual

```
import tkinter as tk
import math
from vector import Vector

class Vehicle:
    def __init__(self, canvas, width, height):
        self.canvas = canvas
        self.width = width
        self.height = height

        self.position = Vector(width / 2, height / 2)
        self.velocity = Vector(0, 0)
        self.acceleration = Vector(0, 0)

        self.damping = 0.995
        self.topspeed = 6
        self.heading = 0
        self.r = 16
        self.thrusting = False

        self.id = canvas.create_polygon(0, 0, 0, 0, 0, 0, fill="gray", outline="black")
```

```
        self.text_sudut = canvas.create_text(10, 30, anchor="nw", text="",font=("Arial",
12))
        self.text_velocity = canvas.create_text(10, 50, anchor="nw",
text="",font=("Arial", 12))

    def update(self):
        self.velocity.add(self.acceleration)
        self.velocity.mult(self.damping)
        self.velocity.limit(self.topspeed)
        self.canvas.itemconfig(self.text_velocity, text=f"Velocity: ({self.velocity})")
        self.position.add(self.velocity)
        self.acceleration = Vector(0, 0)

    def applyForce(self, force):
        self.acceleration.add(force)

    def turn(self, angle):
        self.heading += angle

    def thrust(self):
        force = Vector.fromAngle(self.heading - math.pi / 2).multed(0.1)
        self.applyForce(force)
        self.thrusting = True

    def wrapEdges(self):
        buffer = self.r * 2
        if self.position.x > self.width + buffer:
            self.position.x = -buffer
        elif self.position.x < -buffer:
            self.position.x = self.width + buffer

        if self.position.y > self.height + buffer:
            self.position.y = -buffer
        elif self.position.y < -buffer:
            self.position.y = self.height + buffer

    def show(self):
        angle = self.heading
        self.canvas.itemconfig(self.text_sudut, text=f"Angle: ({math.degrees(angle) %
360})")
        cx, cy = self.position.x, self.position.y

        points = []
        front = Vector(0, -self.r * 1.5).rotated(angle).added(self.position)
        left = Vector(-self.r, self.r).rotated(angle).added(self.position)
        right = Vector(self.r, self.r).rotated(angle).added(self.position)

        points.extend([front.x, front.y, left.x, left.y, right.x, right.y])

        self.canvas.coords(self.id, *points)
        self.canvas.itemconfig(self.id, fill="red" if self.thrusting else "gray")
        self.thrusting = False

class App:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Vehicle Segitiga Sama Kaki - Steering & Wrap")

        WIDTH, HEIGHT = 640, 480
        self.canvas = tk.Canvas(self.root, width=WIDTH, height=HEIGHT, bg="white")
        self.canvas.pack()

        self.vehicle = Vehicle(self.canvas, WIDTH, HEIGHT)

        self.keys = {"left": False, "right": False, "up": False}

        self.root.bind("<KeyPress>", self.key_press)
        self.root.bind("<KeyRelease>", self.key_release)

        self.animate()
        self.root.mainloop()

    def key_press(self, event):
        if event.keysym == "Left":
            self.keys["left"] = True
        elif event.keysym == "Right":
```

```
            self.keys["right"] = True
        elif event.keysym == "Up":
            self.keys["up"] = True

    def key_release(self, event):
        if event.keysym == "Left":
            self.keys["left"] = False
        elif event.keysym == "Right":
            self.keys["right"] = False
        elif event.keysym == "Up":
            self.keys["up"] = False

    def animate(self):
        if self.keys["left"]:
            self.vehicle.turn(-0.05)
        if self.keys["right"]:
            self.vehicle.turn(0.05)
        if self.keys["up"]:
            self.vehicle.thrust()

        self.vehicle.update()
        self.vehicle.wrapEdges()
        self.vehicle.show()

        self.canvas.delete("info")
        self.canvas.create_text(10, 10, anchor="nw",
                                text=f"Pos: ({self.vehicle.position.x:.1f},
{self.vehicle.position.y:.1f})",
                                font=("Arial", 12), tag="info")

        self.root.after(30, self.animate)

if __name__ == "__main__":
    App()
```

Section 3.6: Oscillation, Amplitude, and Period

🌊 Apa itu Oscillation?

Pernah lihat senar gitar yang bergetar setelah dipetik? Atau bandul jam dinding yang terus-menerus bergerak ke kiri dan ke kanan? Nah, gerakan bolak-balik seperti itu disebut **osilasi (oscillation)**.
Contoh:

- Senar gitar bergetar → osilasi
- Ayunan bergerak maju mundur → osilasi
- Trampolin naik turun → osilasi

📈 Grafik Sine: Gerakan Naik Turun yang Lembut

Fungsi matematika sine(x) menghasilkan angka naik turun antara **-1 dan 1**, membentuk **gelombang halus** seperti ini:

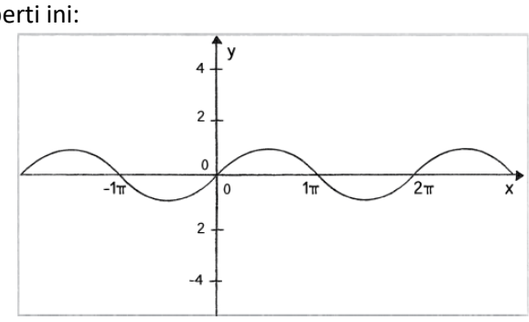


Figure 3.9: y = sine(x)

Ini disebut **gelombang sinus**. Bentuknya mirip gelombang di laut 🌊.

🔁 Periodik: Gerakan yang Terulang-ulang

Fungsi sine(x) selalu mengulang pola yang sama:

- Naik pelan
- Turun pelan
- Naik lagi, dan seterusnya...

Inilah yang disebut **gerakan periodik** — artinya **berulang** secara teratur.

🎛️ Kita Bisa Pakai sine() untuk Menggerakkan Objek!

Misalnya:

x = 100 + sin(angle) * 50

- x adalah posisi objek (misalnya bola)
- angle terus bertambah tiap frame
- sin(angle) menghasilkan angka naik-turun
- * 50 → mengatur **seberapa jauh** bola bergerak (disebut **amplitude**)
- + 100 → menentukan **posisi tengahnya** (agar tidak bolak-balik dari nol)

Hasilnya? Bola akan **bergerak pelan ke kiri dan kanan seperti ayunan!**

🧠 **Istilah Penting**

Istilah	**Penjelasan Sederhana**
Oscillation	Gerakan bolak-balik seperti ayunan
Sine (sin)	Fungsi matematika yang menghasilkan angka naik-turun lembut
Amplitude	Seberapa jauh benda bergerak dari titik tengah
Period	Waktu/kecepatan yang dibutuhkan untuk satu siklus lengkap
Periodik	Berulang secara teratur

🎮 **Contoh Gambarannya:**

Bayangkan kamu main game dan ada pesawat yang terbang naik-turun terus seperti burung. Gerakan itu bisa dibuat dengan:

y = sin(angle) * amplitude + center

- y: posisi pesawat
- angle: bertambah terus tiap frame
- amplitude: seberapa tinggi/rendah gerakannya
- center: posisi tengah di layar

🎯 **Tujuan: Membuat Lingkaran Bergerak Kiri–Kanan (Bolak-balik)**

Bayangkan kita ingin membuat **lingkaran** di layar yang bergerak ke kiri dan ke kanan **terus-menerus** — seperti bandul yang mengayun.

💡 Gerakan ini disebut **Simple Harmonic Motion**
(Fisikawan suka menyebutnya sebagai "gerakan bolak-balik sinusoidal").

🔢 **Istilah yang Perlu Kita Pahami**

Istilah	**Penjelasan Sederhana**
Amplitude	Seberapa jauh lingkaran bergerak dari tengah
Period	Berapa **lama (jumlah frame)** gerakan lengkap

🎡 **Contoh Visual: Gelombang Cosine**

```
Gelombang cos(x) seperti ini:
1.0   ---       ---      (puncak atas)
     /         \
0.0  |           |      (tengah/posisi diam)
      \         /
-1.0   ---___---         (puncak bawah)
```

- Bergerak dari **tengah → kanan → tengah → kiri → balik ke tengah**
- Ulangi terus menerus → disebut **satu siklus**

📐 **Contoh di Dunia Pemrograman (misalnya di Processing)**

Misalnya layar lebarnya 200 piksel. Kita ingin lingkaran bergerak dari:

- Tengah layar = 100 piksel
- Bergerak 100 ke kanan, 100 ke kiri → jadi bolak-balik

```
float amplitude = 100;  // Gerakan kiri-kanan sejauh 100 piksel
float period = 120;     // Satu siklus = 120 frame (misalnya 2 detik)

float x = amplitude * cos(TWO_PI * frameCount / period);
```

🧠 **Penjelasan Formula x = amplitude * cos(TWO_PI * frameCount / period)**

Bagian Formula	**Artinya**
cos(...)	Nilai naik-turun antara -1 dan 1
amplitude * cos(...)	Mengubah gerakan jadi -100 s/d 100 piksel
frameCount / period	Menentukan posisi dalam siklus (berapa jauh kita sudah bergerak)
TWO_PI	1 siklus penuh (360 derajat atau 2π radian)

Jadi,

- Ketika frameCount = 0, hasilnya cos(0) = 1 → x = +100
- Ketika frameCount = 60, hasilnya cos(π) = -1 → x = -100
- Ketika frameCount = 120, hasilnya cos(2π) = 1 → x = +100 (balik ke awal)

Gerakan akan terus berulang setiap **120 frame**.

🟢 **Gambarannya di Layar**

Bayangkan ada bola yang:

- Mulai dari kanan
- Lalu bergerak perlahan ke kiri
- Kemudian kembali ke kanan
- Dan begitu seterusnya…

Simulasi Gerak Harmonik Vertikal + Grafik

📝 Rumus Fisika GHS

1. **Persamaan Gerak Harmonik**:
   $y(t) = A \times \sin(2\pi ft + \phi)$
   - A = Amplitudo (dari slider)
   - f = Frekuensi (dari slider)
   - ϕ = Fase awal (0 dalam simulasi ini)
   - t = Waktu sejak mulai (detik)
2. **Frekuensi Angular**:
   $\omega = 2\pi f$
3. **Posisi Y**:
   y_offset = A × sin(ωt)
   y_actual = CENTER_Y + y_offset

🎞️ Perhitungan Frame-by-Frame

Parameter:

- Interval update: 20ms (~50 FPS)
- Default:
  - A = 100 px
  - f = 0.5 Hz

Contoh Perhitungan:

Frame 0 (t=0s):

y_offset = 100 × sin(2π×0.5×0) = 0

y_actual = 200 + 0 = 200

Frame 1 (t=0.02s):

ω = 2π×0.5 ≈ 3.1416 rad/s

y_offset = 100 × sin(3.1416×0.02) ≈ 100 × 0.0628 ≈ 6.28px

y_actual ≈ 200 + 6.28 ≈ 206.28

Frame 2 (t=0.04s):

y_offset ≈ 100 × sin(3.1416×0.04) ≈ 100 × 0.1253 ≈ 12.53px

y_actual ≈ 200 + 12.53 ≈ 212.53

Frame 25 (t=0.5s):

y_offset = 100 × sin(3.1416×0.5) ≈ 100 × 1 ≈ 100px (puncak)

🔍 Komponen Visual

1. **Bola Merah**:
   - Bergerak vertikal sesuai persamaan GHS
   - Posisi Y diupdate setiap frame
2. **Garis Biru**:
   - Menghubungkan pusat dengan posisi bola
   - Panjang berubah sesuai simpangan
3. **Slider Interaktif**:
   - Amplitudo: 0-150 px
   - Frekuensi: 0.1-2.0 Hz
4. **Info Real-time**:
   - Menampilkan rumus dan nilai aktual
   - Posisi Y ditampilkan di samping bola

⚡ Karakteristik Simulasi

1. **Dinamika Real-time**:
   - Perubahan slider langsung mempengaruhi gerakan
   - Waktu nyata menggunakan time.time()
2. **Presisi Matematis**:
   - Menggunakan fungsi trigonometri Python
   - Perhitungan floating-point presisi tinggi

```
# Simulasi Gerak Harmonik Vertikal + Grafik (COPY PASTE)
import tkinter as tk
import math
import time

# Setup awal
WIDTH, HEIGHT = 600, 400
CENTER_X, CENTER_Y = WIDTH // 2, HEIGHT // 2

root = tk.Tk()
root.title("Simulasi Gerak Harmonik Vertikal + Grafik")

canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

# Sliders
amp_slider = tk.Scale(root, from_=0, to=150, label="Amplitudo (px)", orient="horizontal",
length=300)
amp_slider.set(100)
amp_slider.pack()

freq_slider = tk.Scale(root, from_=0.1, to=2.0, resolution=0.1, label="Frekuensi (Hz)",
orient="horizontal", length=300)
freq_slider.set(0.5)
freq_slider.pack()

# Label rumus dan hasil
info_label = tk.Label(root, text="", font=("Arial", 12))
info_label.pack()

# Objek animasi
radius = 10
dot = canvas.create_oval(0, 0, 0, 0, fill="red")
line = canvas.create_line(CENTER_X, CENTER_Y, CENTER_X, CENTER_Y, fill="blue", width=2)
text_y = canvas.create_text(CENTER_X + 40, CENTER_Y, text="", font=("Arial", 10),
fill="black")

# Grafik waktu vs simpangan
GRAPH_WIDTH = 400
GRAPH_HEIGHT = 150
GRAPH_X = 100
GRAPH_Y = HEIGHT - GRAPH_HEIGHT - 20
graph_points = []

# Garis bantu grafik
canvas.create_line(GRAPH_X, GRAPH_Y + GRAPH_HEIGHT//2,
                   GRAPH_X + GRAPH_WIDTH, GRAPH_Y + GRAPH_HEIGHT//2,
                   fill="gray", dash=(2, 2))  # Garis nol
canvas.create_text(GRAPH_X - 30, GRAPH_Y + GRAPH_HEIGHT//2,
                   text="0", anchor="e", font=("Arial", 8))
canvas.create_text(GRAPH_X - 30, GRAPH_Y,
                   text=f"{amp_slider.get()}", anchor="e", font=("Arial", 8))
canvas.create_text(GRAPH_X - 30, GRAPH_Y + GRAPH_HEIGHT,
                   text=f"-{amp_slider.get()}", anchor="e", font=("Arial", 8))

start_time = time.time()

def update():
    global graph_points

    now = time.time()
    t = now - start_time  # waktu dalam detik

    A = amp_slider.get()
    f = freq_slider.get()
    omega = 2 * math.pi * f
    y_offset = A * math.sin(omega * t)

    # Update posisi titik
    y = CENTER_Y + y_offset
    canvas.coords(dot, CENTER_X - radius, y - radius, CENTER_X + radius, y + radius)
    canvas.coords(line, CENTER_X, CENTER_Y, CENTER_X, y)
    canvas.coords(text_y, CENTER_X + 40, y)
    canvas.itemconfig(text_y, text=f"y(t) = {y_offset:.1f}px")

    # Update grafik
```

```
    graph_points.append((t, y_offset))
    if len(graph_points) > 100:  # Batasi jumlah titik yang disimpan
        graph_points.pop(0)

    canvas.delete("graph")  # Hapus grafik sebelumnya

    # Gambar grafik baru
    for i in range(1, len(graph_points)):
        x1 = GRAPH_X + (i-1) * (GRAPH_WIDTH / 100)
        y1 = GRAPH_Y + GRAPH_HEIGHT//2 - (graph_points[i-1][1] * (GRAPH_HEIGHT/2/A))
        x2 = GRAPH_X + i * (GRAPH_WIDTH / 100)
        y2 = GRAPH_Y + GRAPH_HEIGHT//2 - (graph_points[i][1] * (GRAPH_HEIGHT/2/A))
        canvas.create_line(x1, y1, x2, y2, fill="green", width=2, tag="graph")

    # Update label skala grafik
    canvas.itemconfig(canvas.find_withtag("amplitude_text"), text=f"{A}")
    canvas.itemconfig(canvas.find_withtag("negative_amplitude_text"), text=f"-{A}")

    # Tampilkan rumus dan nilai
    info_label.config(
        text=f"Rumus: y(t) = A × sin(2πft)\nA = {A} px, f = {f} Hz, t = {t:.2f} s, y =
{y_offset:.2f} px\n"
             f"Frekuensi sudut (ω) = {omega:.2f} rad/s"
    )

    root.after(20, update)

update()
root.mainloop()
```

Simulasi Gerak Harmonik Horizontal (Sumbu X)

📝 Rumus Fisika GHS Horizontal

1. **Persamaan Gerak Harmonik**:
   x(t) = A × sin(2πft + ϕ)
   - A = Amplitudo (dari slider, 0-200 px)
   - f = Frekuensi (dari slider, 0.1-2.0 Hz)
   - ϕ = Fase awal (0 dalam simulasi ini)
   - t = Waktu sejak mulai (detik)
2. **Frekuensi Angular**:
   ω = 2πf
3. **Simpangan X**:
   x_offset = A × sin(ωt)
   x_actual = CENTER_X + x_offset

🎞️ Perhitungan Frame-by-Frame

Parameter:

- Interval update: 20ms (~50 FPS)
- Default:
  - A = 100 px
  - f = 0.5 Hz

Contoh Perhitungan:

Frame 0 (t=0s):
x_offset = 100 × sin(2π×0.5×0) = 0
x_actual = 300 + 0 = 300 (tepat di pusat)
Frame 1 (t=0.02s):
ω = 2π×0.5 ≈ 3.1416 rad/s
x_offset = 100 × sin(3.1416×0.02) ≈ 100 × 0.0628 ≈ 6.28px
x_actual ≈ 300 + 6.28 ≈ 306.28
Frame 25 (t=0.5s):
x_offset = 100 × sin(3.1416×0.5) ≈ 100 × 1 ≈ 100px (simpangan maksimum kanan)
Frame 50 (t=1.0s):
x_offset = 100 × sin(3.1416×1.0) ≈ 100 × 0 ≈ 0px (kembali ke pusat)

🔍 Komponen Visual

1. **Bola Merah**:
   - Bergerak horizontal mengikuti sumbu X
   - Posisi diupdate berdasarkan x_actual
2. **Garis Biru**:
   - Menghubungkan pusat canvas dengan posisi bola
   - Panjang berubah sesuai simpangan
3. **Slider Interaktif**:

- Amplitudo: mengontrol jangkauan gerak (0-200 px)
- Frekuensi: mengontrol kecepatan osilasi (0.1-2.0 Hz)

4. **Informasi Real-time**:
   - Menampilkan rumus dan nilai aktual
   - Nilai simpangan ditampilkan di atas bola

⚡ Karakteristik Simulasi

1. **Gerak Periodik Sempurna**:
   - Mengikuti fungsi sinus murni
   - Perioda T = 1/f
2. **Presisi Matematis**:
   - Menggunakan fungsi trigonometri Python
   - Perhitungan floating-point presisi tinggi

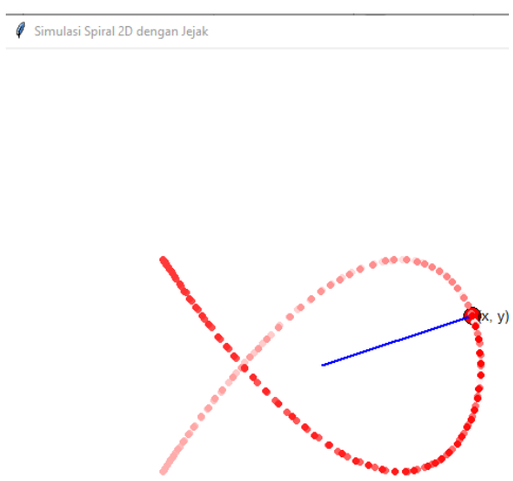
```
# Simulasi Gerak Harmonik Horizontal (Sumbu X)
import tkinter as tk
import math
import time

# Ukuran canvas dan titik pusat
WIDTH, HEIGHT = 600, 400
CENTER_X, CENTER_Y = WIDTH // 2, HEIGHT // 2

root = tk.Tk()
root.title("Simulasi Gerak Harmonik Horizontal (Sumbu X)")

canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

# Sliders
amp_slider = tk.Scale(root, from_=0, to=200, label="Amplitudo (px)", orient="horizontal",
length=300)
amp_slider.set(100)
amp_slider.pack()

freq_slider = tk.Scale(root, from_=0.1, to=2.0, resolution=0.1, label="Frekuensi (Hz)",
orient="horizontal", length=300)
freq_slider.set(0.5)
freq_slider.pack()

# Label rumus dan hasil
info_label = tk.Label(root, text="", font=("Arial", 12))
info_label.pack()

# Objek animasi
radius = 10
dot = canvas.create_oval(0, 0, 0, 0, fill="red")
line = canvas.create_line(CENTER_X, CENTER_Y, CENTER_X, CENTER_Y, fill="blue", width=2)
text_x = canvas.create_text(CENTER_X, CENTER_Y - 30, text="", font=("Arial", 10),
fill="black")

start_time = time.time()

# Fungsi update animasi
def update():
    now = time.time()
    t = now - start_time  # waktu dalam detik

    A = amp_slider.get()
    f = freq_slider.get()
    omega = 2 * math.pi * f
    x_offset = A * math.sin(omega * t)

    # Posisi titik
    x = CENTER_X + x_offset
    canvas.coords(dot, x - radius, CENTER_Y - radius, x + radius, CENTER_Y + radius)
    canvas.coords(line, CENTER_X, CENTER_Y, x, CENTER_Y)
    canvas.coords(text_x, x, CENTER_Y - 30)
    canvas.itemconfig(text_x, text=f"x(t) = {x_offset:.1f}px")

    # Update label informasi
    info_label.config(
        text=f"Rumus: x(t) = A × sin(2πft)\nA = {A}, f = {f} Hz, t = {t:.2f}s, x =
{x_offset:.2f}px"
    )
```

```
    root.after(20, update)

update()
root.mainloop()
```

↔ Perbandingan dengan Simulasi Sebelumnya

Aspek	Simulasi Harmonik Sederhana	Simulasi Gerak Melingkar
Jenis Gerak	Linear (1D) pada sumbu X	Melingkar (2D)
Persamaan	x(t) = A sin(ωt)	x(t)=r cos(θ), y(t)=r sin(θ)
Parameter Kontrol	Amplitudo & Frekuensi	Radius & Kecepatan Sudut
Energi	Energi potensial & kinetik	Energi kinetik saja
Aplikasi	Sistem pegas, bandul	Planet mengorbit, roda berputar
Visualisasi	Garis lurus berubah panjang	Garis berputar dengan panjang tetap

Gerak Spiral 2D



📝 Rumus Fisika Gerak Spiral 2D

1. **Persamaan Gerak**:
   $x(t) = A_x \times \cos(\omega_x t)$
   $y(t) = A_y \times \sin(\omega_y t)$
   - A_x, A_y = Amplitudo (50-200 px)
   - ω_x, ω_y = Kecepatan sudut (0.1-5.0 rad/s)
   - t = Waktu (detik)
2. **Posisi Aktual**:
   x_actual = CENTER_X + x(t)
   y_actual = CENTER_Y + y(t)

🎞️ Perhitungan Frame-by-Frame

Parameter:

- Interval update: 20ms (~50 FPS)
- Default:
  - A_x = 150 px, A_y = 100 px
  - ω_x = 2.0 rad/s, ω_y = 3.0 rad/s
  - Jejak maksimal: 200 titik

Contoh Perhitungan:

Frame 0 (t=0s):
$\theta_x = 2.0 \times 0 = 0$ rad
$\theta_y = 3.0 \times 0 = 0$ rad
$x = 300 + 150 \times \cos(0) = 450$
$y = 300 + 100 \times \sin(0) = 300$

Frame 1 (t=0.02s):
$\theta_x = 2.0 \times 0.02 = 0.04$ rad
$\theta_y = 3.0 \times 0.02 = 0.06$ rad
$x \approx 300 + 150 \times \cos(0.04) \approx 300 + 149.88 \approx 449.88$
$y \approx 300 + 100 \times \sin(0.06) \approx 300 + 6.00 \approx 306.00$

Frame 50 (t=1.0s):
$\theta_x = 2.0 \times 1.0 = 2.0$ rad
$\theta_y = 3.0 \times 1.0 = 3.0$ rad
$x \approx 300 + 150 \times (-0.416) \approx 237.6$
$y \approx 300 + 100 \times 0.141 \approx 314.1$

🔍 Komponen Visual

1. **Bola Merah**:
   - Posisi aktual objek
   - Bergerak membentuk pola Lissajous/spiral
2. **Garis Biru**:
   - Menghubungkan pusat dengan posisi objek
   - Panjang berubah dinamis
3. **Jejak (Trail)**:
   - 200 titik terakhir

- Gradasi warna merah→putih untuk efek fading
- Ukuran titik lebih kecil (radius 3px)

4. **Slider Interaktif**:
   - Kontrol terpisah untuk X dan Y
   - Amplitudo dan kecepatan sudut independen

⚡ Karakteristik Simulasi

1. **Pola Lissajous**:
   - Terbentuk ketika ω_x/ω_y rasio bilangan rasional
   - Spiral ketika rasio irasional
2. **Efek Visual Jejak**:

alpha = int(255 * (i + 1) / trail_length)
color = f"#ff{gray}{gray}" *# Gradasi warna*

3. **Presisi Matematis**:
   - Perhitungan floating-point 64-bit
   - Akumulasi sudut tak terbatas

↔ Perbandingan dengan Simulasi Sebelumnya

Fitur	**Sebelumnya (Gerak Melingkar)**	**Sekarang (Spiral 2D)**
Dimensi	2D (X,Y terkopel)	2D (X,Y independen)
Kontrol	Radius & ω tunggal	A_x, A_y, ω_x, ω_y terpisah
Pola Gerak	Lingkaran sempurna	Lissajous/Spiral
Visual Tambahan	-	Jejak dengan gradasi warna
Aplikasi	Gerak melingkar dasar	Sistem osilasi kompleks

```
# Simulasi Spiral 2D dengan Jejak
import tkinter as tk
import math
import time

WIDTH, HEIGHT = 600, 600
CENTER_X, CENTER_Y = WIDTH // 2, HEIGHT // 2

root = tk.Tk()
root.title("Simulasi Spiral 2D dengan Jejak")

canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

# Sliders
amp_x_slider = tk.Scale(root, from_=0, to=200, label="Amplitudo X (Aₓ)",
orient="horizontal", length=300)
amp_x_slider.set(150)
amp_x_slider.pack()

amp_y_slider = tk.Scale(root, from_=0, to=200, label="Amplitudo Y (Aᵧ)",
orient="horizontal", length=300)
amp_y_slider.set(100)
amp_y_slider.pack()

omega_x_slider = tk.Scale(root, from_=0.1, to=5.0, resolution=0.1, label="ωₓ (rad/s)",
orient="horizontal", length=300)
omega_x_slider.set(2.0)
omega_x_slider.pack()

omega_y_slider = tk.Scale(root, from_=0.1, to=5.0, resolution=0.1, label="ωᵧ (rad/s)",
orient="horizontal", length=300)
omega_y_slider.set(3.0)
omega_y_slider.pack()

# Label informasi
info_label = tk.Label(root, text="", font=("Arial", 12))
info_label.pack()

# Objek utama
radius = 8
dot = canvas.create_oval(0, 0, 0, 0, fill="red")
```

```
line = canvas.create_line(CENTER_X, CENTER_Y, CENTER_X, CENTER_Y, fill="blue", width=2)
angle_text = canvas.create_text(CENTER_X + 20, CENTER_Y - 30, text="", font=("Arial", 10),
fill="black")

# Jejak (trail)
trail_length = 200
trail_points = []

start_time = time.time()

def update():
    now = time.time()
    t = now - start_time

    A_x = amp_x_slider.get()
    A_y = amp_y_slider.get()
    ω_x = omega_x_slider.get()
    ω_y = omega_y_slider.get()

    θ_x = ω_x * t
    θ_y = ω_y * t

    x_offset = A_x * math.cos(θ_x)
    y_offset = A_y * math.sin(θ_y)

    x = CENTER_X + x_offset
    y = CENTER_Y + y_offset

    # Simpan jejak
    trail_points.append((x, y))
    if len(trail_points) > trail_length:
        trail_points.pop(0)

    canvas.delete("trail")  # Hapus jejak lama
    for i, (tx, ty) in enumerate(trail_points):
        alpha = int(255 * (i + 1) / trail_length)
        gray = hex(255 - alpha)[2:].zfill(2)
        color = f"#ff{gray}{gray}"  # Gradasi merah ke putih
        canvas.create_oval(tx - 3, ty - 3, tx + 3, ty + 3, fill=color, outline="",
tags="trail")

    # Update titik dan garis
    canvas.coords(dot, x - radius, y - radius, x + radius, y + radius)
    canvas.coords(line, CENTER_X, CENTER_Y, x, y)
    canvas.coords(angle_text, x + 20, y)
    canvas.itemconfig(angle_text, text="(x, y)")

    # Label informasi
    info_label.config(
        text=f"x(t) = {A_x}·cos({θ_x:.2f}) = {x_offset:.1f}px\n"
             f"y(t) = {A_y}·sin({θ_y:.2f}) = {y_offset:.1f}px\n"
             f"θₓ = {θ_x:.2f} rad, θᵧ = {θ_y:.2f} rad\n"
             f"ωₓ = {ω_x}, ωᵧ = {ω_y}, t = {t:.2f}s"
    )

    root.after(20, update)

update()
root.mainloop()
```

Gelombang

Bayangkan kamu sedang membuat gambar ombak di laut menggunakan komputer. Ini cara kerjanya:

1. Dasar-dasar Gelombang

- Gelombang seperti ombak yang naik turun
- Kita bisa membuatnya dengan fungsi matematika bernama *sinus*
- Seperti membuat banyak titik yang bergerak naik turun secara berurutan

2. Cara Membuat Gelombang Sederhana

Kita perlu 3 bahan utama:

1. **Sudut awal** (angle): Dimulai dari 0
2. **Kecepatan perubahan sudut** (angleVel): Seberapa cepat gelombang bergerak
3. **Tinggi gelombang** (amplitude): Seberapa tinggi rendahnya ombak

```
angle = 0
angleVel = 0.2
```

```
amplitude = 100
```
3. Membuat Titik-titik Gelombang
Kita akan meletakkan titik-titik sepanjang layar:
1. Setiap 24 piksel (jarak antar titik)
2. Hitung posisi Y menggunakan rumus:

```
y = amplitude * sin(angle)
```

3. Gambar lingkaran di posisi (x, y)
4. Tambahkan sudut untuk titik berikutnya

Contoh kode sederhana:

```
for x in range(0, lebar_layar, 24):
    y = amplitude * sin(angle)
    gambar_lingkaran(x, y + tinggi_layar/2, 48, 48)
    angle += angleVel
```

4. Hasil yang Didapat
- Semakin besar angleVel, gelombang akan lebih rapat (periodenya pendek)
- Semakin besar amplitude, gelombang akan lebih tinggi
- Jika titik-titik terlalu jarang, kita bisa hubungkan dengan garis

Analogi Sederhana
Bayangkan kamu dan beberapa teman berdiri dalam satu barisan:
- Setiap orang memegang balon di ketinggian berbeda
- Aturannya:
  - Orang pertama angkat balon setinggi sin(0)
  - Orang kedua angkat balon setinggi sin(0.2)
  - Orang ketiga angkat balon setinggi sin(0.4), dst.
- Hasilnya akan terlihat seperti gelombang!

Contoh Nyata
Jika:
- amplitude = 100
- angleVel = 0.2
- angle mulai dari 0

Perhitungan untuk 3 titik pertama:
1. Titik 1 (x=0):
   - y = 100 × sin(0) = 0
   - angle baru = 0 + 0.2 = 0.2
2. Titik 2 (x=24):
   - y = 100 × sin(0.2) ≈ 100 × 0.198 ≈ 19.8
   - angle baru = 0.2 + 0.2 = 0.4
3. Titik 3 (x=48):
   - y = 100 × sin(0.4) ≈ 100 × 0.389 ≈ 38.9

Hasilnya titik-titik akan membentuk pola naik turun seperti ombak!
Tips Tambahan
- Untuk gelombang lebih halus, kurangi jarak antar titik (misal 10 piksel)
- Untuk gelombang lebih panjang, kurangi angleVel
- Bisa juga dihubungkan dengan garis untuk membentuk satu garis ombak utuh

Simulasi Gelombang Statis

📜 **Rumus Utama**
1. **Persamaan Gelombang Sinus**:
   y = AMPLITUDE * sin(angle)
   center_y = y + HEIGHT / 2
   - AMPLITUDE: Tinggi maksimum gelombang (100 piksel)
   - angle: Sudut dalam radian yang bertambah secara konstan
   - HEIGHT / 2: Pusat vertikal layar (120 piksel)
2. **Perubahan Sudut**:
   angle += ANGLE_VELOCITY (0.2 radian per lingkaran)

🧱 **Perhitungan Tiap Lingkaran** (Bukan per frame karena statis)
Parameter:
- SPACING = 24 (Jarak antar lingkaran horizontal)
- RADIUS = 24 (Ukuran lingkaran)
- WIDTH = 640 (Lebar layar)

Contoh untuk 5 lingkaran pertama:

x	angle (rad)	sin(angle)	y	center_y
0	0.0	sin(0) = 0	100*0 = 0	0 + 120 = 120
24	0.2	sin(0.2) ≈ 0.1987	100*0.1987 ≈ 19.87	19.87 + 120 ≈ 139.87

x	angle (rad)	sin(angle)	y	center_y
48	0.4	sin(0.4) ≈ 0.3894	100*0.3894 ≈ 38.94	38.94 + 120 ≈ 158.94
72	0.6	sin(0.6) ≈ 0.5646	100*0.5646 ≈ 56.46	56.46 + 120 ≈ 176.46
96	0.8	sin(0.8) ≈ 0.7174	100*0.7174 ≈ 71.74	71.74 + 120 ≈ 191.74

Visualisasi:

```
y-axis
^
|      o       (x=96, y≈191.74)
|     o        (x=72, y≈176.46)
|    o         (x=48, y≈158.94)
|   o          (x=24, y≈139.87)
|  o           (x=0, y=120)
+------------------> x-axis
```

🔧 **Proses Pembuatan Gelombang**

1. **Loop Horizontal**:
   for x in range(0, WIDTH + 1, SPACING):
   - Membuat lingkaran setiap 24 piksel dari x=0 sampai x=640
2. **Hitung Posisi Vertikal**:
   y = AMPLITUDE * math.sin(angle)
   center_y = y + HEIGHT / 2
   - y: Posisi relatif dari tengah layar (-100 sampai +100)
   - center_y: Posisi absolut di layar (20 sampai 220)
3. **Gambar Lingkaran**:
   canvas.create_oval(
   x - RADIUS, center_y - RADIUS,
   x + RADIUS, center_y + RADIUS,
   ...
)
4. **Update Sudut**:
   angle += ANGLE_VELOCITY *# +0.2 rad setiap lingkaran*

📊 **Karakteristik Gelombang**

- **Panjang Gelombang**:
  - ANGLE_VELOCITY = 0.2 berarti 1 siklus lengkap (2π) terjadi setiap:
    2π / 0.2 ≈ 31.4 lingkaran
    31.4 * 24 ≈ 754 piksel
  - Karena lebar layar hanya 640 piksel, tidak menampilkan 1 siklus penuh
- **Amplitudo**:
  - Puncak tertinggi: HEIGHT/2 + AMPLITUDE = 120 + 100 = 220
  - Lembah terendah: HEIGHT/2 - AMPLITUDE = 120 - 100 = 20

```
# SCRIPT 11 : Simulasi gelombang statis menggunakan sinus
import tkinter as tk
import math

# === Konstanta Layar ===
WIDTH, HEIGHT = 640, 240
ANGLE_VELOCITY = 0.2
AMPLITUDE = 100
SPACING = 24
RADIUS = 24

# === Setup Tkinter ===
root = tk.Tk()
root.title("Example 3.8: Static Wave")

canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

# === Gambar lingkaran gelombang ===
angle = 0
for x in range(0, WIDTH + 1, SPACING):
    # 1) Hitung posisi y menggunakan sinus
    y = AMPLITUDE * math.sin(angle)
```

```
    center_y = y + HEIGHT / 2

    # 2) Gambar lingkaran di posisi (x, center_y)
    canvas.create_oval(
        x - RADIUS, center_y - RADIUS,
        x + RADIUS, center_y + RADIUS,
        fill="gray", outline="black", width=2
    )

    # 3) Tambahkan sudut
    angle += ANGLE_VELOCITY

root.mainloop()
```

Memahami Gelombang Dinamis dan Solusinya

Script sebelumnya menampilkan gelombang statis (tidak bergerak). Untuk membuatnya bergerak seperti gelombang nyata, kita perlu memodifikasi pendekatannya.

Masalah dengan Pendekatan Awal

Jika kita hanya menambahkan sudut (angle) secara global:

1. Gelombang akan terputus di ujung kanan layar
2. Saat animasi berlanjut, ujung kiri tidak akan menyambung dengan ujung kanan dengan mulus
3. Hasilnya akan terlihat seperti gelombang yang "terpotong" dan tidak alami

Solusi: Variabel startAngle

Untuk membuat gelombang bergerak dengan mulus, kita perlu:

1. Menyimpan sudut awal (startAngle) terpisah
2. Menambah startAngle sedikit demi sedikit setiap frame
3. Menggunakan startAngle sebagai dasar untuk menghitung posisi tiap lingkaran

Analoginya:

Bayangkan kita sedang menggulung karpet bergambar gelombang:

- startAngle seperti titik awal gulungan karpet
- Setiap kali animasi update, kita menggulung sedikit (startAngle bertambah)
- Tapi pola gelombang di startAngle tetap konsisten

Implementasi Kode yang Benar

```
startAngle = 0  # Variabel baru untuk tracking awal gelombang

def draw_wave():
    global startAngle
    canvas.delete("all")  # Hapus frame sebelumnya

    angle = startAngle  # Mulai dari startAngle

    for x in range(0, WIDTH + 1, SPACING):
        y = AMPLITUDE * math.sin(angle)
        center_y = y + HEIGHT / 2

        canvas.create_oval(
            x - RADIUS, center_y - RADIUS,
            x + RADIUS, center_y + RADIUS,
            fill="gray", outline="black", width=2
        )

        angle += ANGLE_VELOCITY

    startAngle += 0.02  # Perlahan ubah startAngle
    root.after(30, draw_wave)  # Ulangi setiap 30ms

draw_wave()  # Mulai animasi
```

Perubahan Penting:

1. startAngle: Variabel baru yang selalu bertambah sedikit setiap frame
2. angle = startAngle: Setiap lingkaran mulai menghitung dari startAngle saat ini
3. startAngle += 0.02: Membuat gelombang bergerak perlahan
4. root.after(30, draw_wave): Membuat animasi berulang setiap 30 milidetik

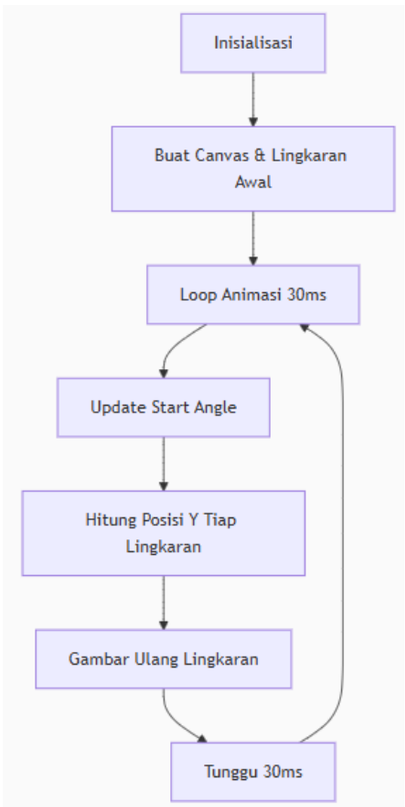
Hasilnya:

- Gelombang akan bergerak mulus dari kanan ke kiri
- Tidak ada lagi "potongan" di ujung layar

- Gerakan terlihat alami seperti ombak sungguhan

Dengan pendekatan ini, kita bisa membuat animasi gelombang yang lebih realistis! 🌊

Example 3.9: The Wave



📜 Rumus Utama

1. **Persamaan Gelombang Sinus**:
   y = map(sin(angle), -1, 1, 0, HEIGHT)
   - angle = start_angle + (i * angle_velocity)
   - start_angle bertambah 0.02 setiap frame
   - angle_velocity = 0.2 (jarak sudut antar lingkaran)
2. **Fungsi Mapping**:

```
def map_value(value, start1, stop1, start2, stop2):
  return start2 + (stop2-start2) * ((value-start1)/(stop1-start1))
```

🎞️ Perhitungan Tiap Frame (30ms ≈ 33FPS)

Parameter:

- STEP = 24 (jarak horizontal antar lingkaran)
- CIRCLE_SIZE = 48 (diameter lingkaran)
- angle_velocity = 0.2 rad/lingkaran
- start_angle bertambah +0.02 rad/frame

Contoh Frame 0:

- start_angle = 0.0
- Lingkaran ke-0 (x=0):
  angle = 0.0 + (0 * 0.2) = 0.0
  sin(0) = 0
  y = map(0, -1,1, 0,240) = 120
- Lingkaran ke-1 (x=24):
  angle = 0.0 + (1 * 0.2) = 0.2
  sin(0.2) ≈ 0.1987
  y = map(0.1987, -1,1, 0,240) ≈ 143.84
- Lingkaran ke-2 (x=48):
  angle = 0.4
  sin(0.4) ≈ 0.3894
  y ≈ 166.73

Frame 1:

- start_angle = 0.02
- Lingkaran ke-0:
  angle = 0.02
  sin(0.02) ≈ 0.0200
  y ≈ 122.40
- Lingkaran ke-1:
  angle = 0.22
  sin(0.22) ≈ 0.2182
  y ≈ 146.18

📊 Visualisasi Gelombang Bergerak

Frame 0:
o (x=0,y=120)
o (x=24,y≈143.84)
o (x=48,y≈166.73)
...

Frame 1:
o (x=0,y≈122.4)
o (x=24,y≈146.18)
o (x=48,y≈168.64)
...

⚙️ Implementasi Kode Kunci

1. **Inisialisasi Lingkaran**:

```
for x in range(0, WIDTH + 1, STEP):
  # Buat lingkaran di posisi awal
  circle_id = canvas.create_oval(...)
  self.circles.append(circle_id)
```

2. **Animasi**:

```
def animate(self):
    self.canvas.delete("all")  # Hapus frame sebelumnya
    angle = self.start_angle
```

```
        self.start_angle += 0.02

        for x in range(0, WIDTH + 1, STEP):
            sin_value = math.sin(angle)
            y = self.map_value(sin_value, -1, 1, 0, HEIGHT)
            # Gambar lingkaran baru
            canvas.create_oval(...)
            angle += self.angle_velocity
```

🔍 Analisis Perilaku

1. **Gelombang Bergerak**:
    - Perubahan start_angle membuat gelombang bergerak ke kiri
    - Kecepatan gelombang ditentukan oleh increment start_angle (0.02)
2. **Karakteristik Gelombang**:
    - Panjang gelombang: 2π / angle_velocity ≈ 31.4 lingkaran
    - Dalam piksel: 31.4 * 24 ≈ 754 piksel (lebih besar dari lebar layar)
3. **Efek Visual**:
    - Lingkaran bergerak vertikal mengikuti pola sinus
    - Gelombang tampak bergerak horizontal karena fase bertahap

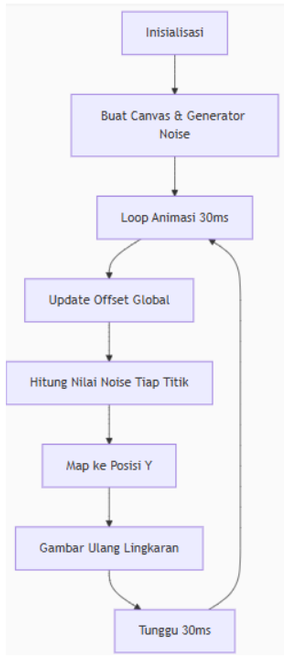
🧠 **Apa yang Terjadi?**

- Kita punya banyak lingkaran di layar (dalam satu baris).
- Posisi y dari setiap lingkaran **berubah naik-turun** sesuai gelombang sin().
- Sudut angle berubah tiap frame untuk membuat efek **bergerak seperti gelombang air**.
- map() digunakan agar sin() yang hasilnya dari -1 s/d 1 diubah jadi 0 s/d tinggi layar.

```
import tkinter as tk
import math
from vector import Vector

# Konfigurasi jendela dan kanvas
WIDTH, HEIGHT = 640, 240
STEP = 24         # Jarak antar titik gelombang
CIRCLE_SIZE = 48  # Diameter lingkaran

class WaveSimulation:
    def __init__(self, root):
        self.canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
        self.canvas.pack()

        self.start_angle = 0.0       # Mirip variabel startAngle di Processing
        self.angle_velocity = 0.2    # Kecepatan perubahan sudut antar titik

        self.circles = []  # List untuk menyimpan ID lingkaran di canvas

        # Inisialisasi lingkaran pada posisi awal
        for x in range(0, WIDTH + 1, STEP):
            y = HEIGHT // 2  # Posisi y sementara
            circle_id = self.canvas.create_oval(
                x - CIRCLE_SIZE//2, y - CIRCLE_SIZE//2,
                x + CIRCLE_SIZE//2, y + CIRCLE_SIZE//2,
                fill="gray", outline="black"
            )
            self.circles.append(circle_id)

        # Mulai animasi
        self.animate()

    def animate(self):
        # Bersihkan background dengan cara mengganti warna semua oval
        self.canvas.delete("all")

        angle = self.start_angle
        self.start_angle += 0.02  # Seiring waktu sudut awal bertambah

        # Gambar ulang semua lingkaran mengikuti pola gelombang
        for i, x in enumerate(range(0, WIDTH + 1, STEP)):
            # Ubah nilai sin menjadi nilai y (0 - HEIGHT)
            sin_value = math.sin(angle)
            y = self.map_value(sin_value, -1, 1, 0, HEIGHT)

            # Gambar ulang lingkaran
            circle_id = self.canvas.create_oval(
                x - CIRCLE_SIZE//2, y - CIRCLE_SIZE//2,
```

```
                x + CIRCLE_SIZE//2, y + CIRCLE_SIZE//2,
                fill="gray", outline="black"
            )

            angle += self.angle_velocity

        # Ulangi fungsi animate setiap 30 milidetik (~33 FPS)
        self.canvas.after(30, self.animate)

    @staticmethod
    def map_value(value, start1, stop1, start2, stop2):
        # Fungsi seperti map() di Processing: ubah nilai dari satu rentang ke rentang lain
        return start2 + (stop2 - start2) * ((value - start1) / (stop1 - start1))

if __name__ == "__main__":
    root = tk.Tk()
    root.title("Gelombang Sine dengan Tkinter")
    sim = WaveSimulation(root)
    root.mainloop()
```

Gelombang dengan Perlin Noise :



📜 Rumus Utama

1. Perlin Noise:
   noise_value = PerlinNoise(x_offset)
   - Menghasilkan nilai antara -1 sampai 1
   - x_offset bertambah secara kontinu
2. Mapping ke Posisi Y:

y = map(noise_value, 0, 1, center_y+amplitude, center_y-amplitude)

   - amplitude = HEIGHT/3 (80 piksel)
   - center_y = HEIGHT/2 (120 piksel)
3. Parameter Gerak:
   - x_offset_start bertambah +0.01 setiap frame
   - offset_increment = 0.1 (jarak sampling noise antar titik)

🎞️ Perhitungan Tiap Frame (30ms ≈ 33FPS)

Parameter:

- STEP = 24 (jarak horizontal antar lingkaran)
- CIRCLE_SIZE = 48 (diameter lingkaran)
- amplitude = 80 (variasi vertikal maksimum)

Contoh Frame 0:

- x_offset_start = 0.0
- Lingkaran ke-0 (x=0):
  noise_value = noise(0.0) ≈ 0.0
  y = map(0.0, 0,1, 200,40) = 120 (tengah)
- Lingkaran ke-1 (x=24):
  x_offset = 0.0 + 0.1 = 0.1
  noise_value = noise(0.1) ≈ 0.15
  y = map(0.15, 0,1, 200,40) ≈ 176
- Lingkaran ke-2 (x=48):
  x_offset = 0.2
  noise_value = noise(0.2) ≈ 0.25
  y ≈ 160

Frame 1:

- x_offset_start = 0.01
- Lingkaran ke-0:
  noise_value = noise(0.01) ≈ 0.02
  y ≈ 123.2
- Lingkaran ke-1:
  x_offset = 0.11
  noise_value ≈ 0.17
  y ≈ 172.8

📊 Visualisasi Gelombang Noise

```
Frame 0:
  o (x=0,y=120)
   o (x=24,y≈176)
    o (x=48,y≈160)
      ...

Frame 1:
```

```
  o (x=0,y≈123.2)
   o (x=24,y≈172.8)
    o (x=48,y≈164.8)
     ...
```

⚙️ Implementasi Kode Kunci

1. Inisialisasi Noise:
   self.noise = PerlinNoise(octaves=1) *# Noise 1 octave*
2. Animasi:
   ```
   def animate(self):
       self.canvas.delete("all")
       x_offset = self.x_offset_start
       self.x_offset_start += 0.01

       for i, x in enumerate(range(0, WIDTH+1, STEP)):
           noise_value = (self.noise(x_offset) + 1) / 2  # Normalisasi ke 0-1
           y = self.map_value(noise_value, 0,1, center_y+amplitude, center_y-amplitude)
           # Gambar lingkaran
           canvas.create_oval(...)
           x_offset += self.offset_increment
   ```

🔍 Karakteristik Perlin Noise

1. Kehalusan:
   - Nilai berubah secara gradual (tidak acak kasar)
   - Hasilkan gelombang organik
2. Parameter:
   - octaves=1: Noise sederhana
   - Bisa ditambah untuk detail lebih kompleks
3. Gerakan Alami:
   - Increment x_offset_start membuat gelombang "mengalir"
   - offset_increment mengontrol kerapatan gelombang

6. Mengapa Terlihat Lebih Alami?

- **Perlin Noise** menghasilkan pola yang halus dan acak
- Berbeda dengan gelombang sinus yang terlalu sempurna dan berulang
- Cocok untuk membuat efek alam seperti ombak, awan, atau permukaan tanah

7. Hasil Akhir

- Lingkaran-lingkaran membentuk gelombang yang bergerak perlahan
- Pergerakannya tidak teratur sempurna seperti di alam nyata
- Terlihat seperti ombak laut yang sebenarnya

Tips Bermain dengan Kode

1. Ubah amplitude untuk membuat gelombang lebih tinggi/rendah
2. Ubah offset_increment untuk mengubah kecepatan gelombang
3. Coba ganti warna lingkaran menjadi gradien untuk efek lebih keren

Program ini menunjukkan bagaimana komputer bisa meniru keacakan alam dengan matematika! 🌊✨

```
import tkinter as tk
from vector import Vector
from perlin_noise import PerlinNoise

# Konfigurasi jendela
WIDTH, HEIGHT = 640, 240
STEP = 24
CIRCLE_SIZE = 48

class WaveWithNoise:
    def __init__(self, root):
        self.canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
        self.canvas.pack()

        # Inisialisasi noise generator dan offset
        self.noise = PerlinNoise(octaves=1)
        self.x_offset_start = 0.0   # Nilai awal untuk offset
        self.offset_increment = 0.1 # Seberapa cepat noise bergerak

        self.circle_ids = []  # Menyimpan ID lingkaran

        for x in range(0, WIDTH + 1, STEP):
            y = HEIGHT // 2
            circle_id = self.canvas.create_oval(
                x - CIRCLE_SIZE//2, y - CIRCLE_SIZE//2,
                x + CIRCLE_SIZE//2, y + CIRCLE_SIZE//2,
                fill="lightblue", outline="black"
            )
```

```
            self.circle_ids.append(circle_id)

        self.animate()

    def animate(self):
        # Bersihkan canvas
        self.canvas.delete("all")

        x_offset = self.x_offset_start
        self.x_offset_start += 0.01  # Perubahan waktu global (seperti frameCount)

        for i, x in enumerate(range(0, WIDTH + 1, STEP)):
            # Ambil nilai noise untuk x_offset
            noise_value = self.noise(x_offset)

            # Mapping noise (-1 to 1) menjadi (0 to HEIGHT)
            amplitude = HEIGHT / 3  # tinggi gelombang
            center_y = HEIGHT / 2   # titik tengah vertikal
            y = self.map_value(noise_value, 0, 1, center_y + amplitude, center_y -
amplitude)

            # Gambar lingkaran
            circle_id = self.canvas.create_oval(
                x - CIRCLE_SIZE//2, y - CIRCLE_SIZE//2,
                x + CIRCLE_SIZE//2, y + CIRCLE_SIZE//2,
                fill="lightblue", outline="black"
            )

            self.circle_ids[i] = circle_id
            x_offset += self.offset_increment  # Tambah offset untuk gelombang menyebar

        self.canvas.after(30, self.animate)

    @staticmethod
    def map_value(value, start1, stop1, start2, stop2):
        return start2 + (stop2 - start2) * ((value - start1) / (stop1 - start1))

if __name__ == "__main__":
    root = tk.Tk()
    root.title("Perlin Noise Wave - Tkinter")
    app = WaveWithNoise(root)
    root.mainloop()
```

program gelombang :

1. Konsep Dasar

Program ini membuat dua gelombang berbeda menggunakan lingkaran-lingkaran kecil yang bergerak naik turun seperti ombak di laut.

2. Bagian-Bagian Penting

1. **Lingkaran Biru**: Banyak lingkaran kecil (diameter 24 pixel) yang membentuk gelombang
2. **Dua Gelombang**:
    - Gelombang kecil di atas (posisi y=75)
    - Gelombang besar di bawah (posisi y=120)

3. Cara Kerja Gelombang

Setiap gelombang punya 4 sifat utama:

1. **Amplitudo**: Seberapa tinggi gelombangnya
    - Gelombang kecil: 20 pixel
    - Gelombang besar: 40 pixel
2. **Perioda**: Seberapa panjang satu ombak
    - Gelombang kecil: sangat panjang (600)
    - Gelombang besar: lebih pendek (180)
3. **Jarak antar lingkaran**: 8 pixel (xspacing)
4. **Posisi awal**: Titik mulai gelombang (x dan y)

4. Proses Animasi

1. **Setiap 30 milidetik**:
    - Program menghitung ulang posisi semua lingkaran
    - Posisi Y dihitung dengan rumus sinus (math.sin)
    - Lingkaran digambar/digerakkan ke posisi baru
2. **Rumus Gelombang**:
   nilai_y = sin(sudut) × tinggi_gelombang
    - sudut terus bertambah sedikit setiap frame (0.02)

- Ini yang membuat gelombang terlihat bergerak

5. Analogi Sederhana

Bayangkan dua rantai panjang:

1. Rantai pertama: mata rantai kecil-kecil, gerakannya pelan
2. Rantai kedua: mata rantai lebih besar, gerakannya lebih cepat
   Kedua rantai digoyangkan naik-turun secara teratur membentuk gelombang.

6. Kenapa Ada Dua Gelombang?

Untuk menunjukkan perbedaan:

- Gelombang kecil: tinggi 20px, panjang, bergerak lambat
- Gelombang besar: tinggi 40px, pendek, bergerak cepat

7. Cara Membaca Kode

1. Wave.update(): Menghitung posisi baru tiap lingkaran
2. Wave.show(): Menggambar/menggerakkan lingkaran ke posisi baru
3. App.animate(): Mengulangi proses update dan show terus-menerus

Tips Bermain dengan Kode

1. Ubah angka amplitudo untuk membuat gelombang lebih tinggi/rendah
2. Ubah perioda untuk mengubah panjang gelombang
3. Tambah warna berbeda untuk tiap gelombang

Program ini menunjukkan bagaimana objek (lingkaran) bisa bekerja bersama membentuk pola yang indah!

```
from tkinter import *
import math

from vector import Vector

# --- Konstanta global ---
WIDTH, HEIGHT = 640, 240
CIRCLE_SIZE = 24

# --- Kelas Wave ---
class Wave:
    def __init__(self, canvas, x, y, w, amplitude, period):
        self.canvas = canvas
        self.xspacing = 8
        self.w = w
        self.origin = Vector(x, y)
        self.theta = 0.0
        self.amplitude = amplitude
        self.period = period
        self.dx = (math.tau / self.period) * self.xspacing
        self.yvalues = [0.0 for _ in range(int(self.w / self.xspacing))]
        self.circle_ids = [None for _ in range(len(self.yvalues))]

    def update(self):
        self.theta += 0.02
        x = self.theta
        for i in range(len(self.yvalues)):
            self.yvalues[i] = math.sin(x) * self.amplitude
            x += self.dx

    def show(self):
        for i in range(len(self.yvalues)):
            x_pos = self.origin.x + i * self.xspacing
            y_pos = self.origin.y + self.yvalues[i]
            if self.circle_ids[i] is None:
                self.circle_ids[i] = self.canvas.create_oval(
                    x_pos - CIRCLE_SIZE//2, y_pos - CIRCLE_SIZE//2,
                    x_pos + CIRCLE_SIZE//2, y_pos + CIRCLE_SIZE//2,
                    fill="lightblue", outline="black"
                )
            else:
                self.canvas.coords(
                    self.circle_ids[i],
                    x_pos - CIRCLE_SIZE//2, y_pos - CIRCLE_SIZE//2,
                    x_pos + CIRCLE_SIZE//2, y_pos + CIRCLE_SIZE//2
                )

# --- Aplikasi utama ---
class App:
    def __init__(self, root):
        self.canvas = Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
        self.canvas.pack()
```

```
        self.wave0 = Wave(self.canvas, 50, 75, 100, 20, 600)
        self.wave1 = Wave(self.canvas, 300, 120, 300, 40, 180)

        self.animate()

    def animate(self):
        self.wave0.update()
        self.wave0.show()

        self.wave1.update()
        self.wave1.show()

        self.canvas.after(30, self.animate)

# --- Jalankan ---
if __name__ == "__main__":
    root = Tk()
    root.title("Exercise 3.11 - OOP Wave")
    app = App(root)
    root.mainloop()
```

The Nature of Code 3.9 - Trigonometry and Forces: The Pendulum

---

🧠 **Konsep Utama**

Pendulum adalah massa (disebut **bob**) yang digantung pada titik tetap (pivot) dengan tali atau batang panjang tetap. Ketika digerakkan dari posisi setimbangnya, gaya gravitasi menarik bob ke bawah. Tetapi, karena bob terikat ke pivot, ia tidak jatuh lurus, melainkan **berayun** membentuk gerak melingkar.

Kita akan simulasikan:

- **Sudut** pendulum terhadap vertikal (θ / theta)
- **Kecepatan sudut** (angular velocity = ω / omega)
- **Percepatan sudut** (angular acceleration = α / alpha)

⚖️ **Gaya yang Bekerja**

Pendulum terkena gaya gravitasi. Tetapi gaya ini tidak langsung mempercepat bob dalam lintasan melingkar.
Kita butuh komponen gaya gravitasi **yang sejajar dengan lintasan melingkar**.
Kita gunakan **trigonometri** untuk menghitung gaya gravitasi pada arah ini.

🧮 **Perhitungan Simulasi Pendulum**

1. Sudut

Sudut θ menunjukkan posisi pendulum dari vertikal:
theta = sudut (dalam radian)

---

2. Gaya Gravitasi sebagai Torsi

Gaya yang mendorong pendulum berayun adalah **komponen tangen dari gaya gravitasi**, yaitu:
`F_tangen = -g * sin(θ)`
Tanda negatif karena arah gaya mengarah untuk mengembalikan ke posisi setimbang.

---

3. Hukum Newton 2 dalam bentuk sudut (rotasi)

Ingat:
`Torsi (τ) = m * a_tangen * r = m * g * sin(θ) * r`
Tapi: Torsi = I * α
Jadi:
`α = - (g / L) * sin(θ)`
Keterangan:

- α: percepatan sudut
- g: gravitasi (misalnya 9.8 m/s²)
- L: panjang tali pendulum
- θ: sudut dari vertikal

---

4. Update Gerakan (Euler Integration)

Simulasi gerakan dilakukan secara bertahap (step by step):

```
angular_acceleration = - (g / length) * sin(theta)
angular_velocity += angular_acceleration
theta += angular_velocity
```

Kita juga bisa menambahkan **peredaman** (friction/drag rotasi):

```
angular_velocity *= damping  # contoh: 0.99
```

📈 **Visualisasi**
Pendulum akan berayun seperti bandul jam. Besarnya sudut (theta) berubah setiap waktu karena percepatan sudut (alpha), dan kecepatan sudut (omega) menentukan perubahan sudut.

🧪 **Contoh Nilai**
Misal:

- length = 100 (piksel atau meter)
- g = 1 (disederhanakan)
- theta = π/4 (45 derajat)

Maka:
angular_acceleration = - (1 / 100) * sin(π/4)
≈ - 0.0071

🧭 **Ringkasan Tahapan**

1. Tentukan panjang pendulum dan sudut awal.
2. Hitung percepatan sudut: α = - (g / L) * sin(θ)
3. Update kecepatan sudut: ω += α
4. Update posisi sudut: θ += ω
5. Gambar posisi bob: (x = origin_x + L * sin(θ), y = origin_y + L * cos(θ))
6. Ulangi setiap frame.

SIMULASI PENDULUM

Rumus Fisika yang Digunakan

a. **Percepatan Sudut (a_acceleration)**:
`a_acceleration = (-gravity / r) * sin(angle)`

- gravity: Konstanta gravitasi (0.4 dalam kode)
- r: Panjang tali pendulum
- angle: Sudut aktuall dalam radian

b. **Kecepatan Sudut (a_velocity)**:
`a_velocity += a_acceleration * delta_time`
(Dalam kode, delta_time dianggap 1 karena loop berjalan cukup cepat)

c. **Redaman (Damping)**:
`a_velocity *= damping_factor (0.995)`

d. **Posisi Bob**:
`bob_x = origin_x + r * sin(angle)`
`bob_y = origin_y + r * cos(angle)`

3. **Perhitungan Setiap Frame**
Misalkan untuk frame ke-n dengan:

- angle = π/4 (45°)
- a_velocity = 0.1 rad/s
- r = 175
- gravity = 0.4

Frame n:

1. Hitung a_acceleration:

a_acceleration = (-0.4 / 175) * sin(π/4) ≈ -0.00162

2. Update a_velocity:

a_velocity = 0.1 + (-0.00162) ≈ 0.09838

3. Apply damping:

a_velocity *= 0.995 ≈ 0.09789

4. Update angle:

angle = π/4 + 0.09789 ≈ 0.8824 rad (~50.56°)

5. Hitung posisi bob:

bob_x = 320 + 175 * sin(0.8824) ≈ 320 + 175 * 0.773 ≈ 455.28
bob_y = 50 + 175 * cos(0.8824) ≈ 50 + 175 * 0.634 ≈ 160.95

Frame n+1:

- Proses diulang dengan nilai angle dan a_velocity yang baru.

4. **Visualisasi Tkinter**

- **Garis/Tali**:

canvas.coords(line, origin_x, origin_y, bob_x, bob_y)

- **Bob (Lingkaran)**:

canvas.coords(circle, bob_x - radius, bob_y - radius, bob_x + radius, bob_y + radius)

- **Teks Info**:
Menampilkan sudut (dalam derajat), percepatan, dan kecepatan sudut.

5. **Timing Loop**

- root.after(16, update): Loop berjalan setiap ~16ms (~60 FPS).

- Perhitungan fisika dilakukan setiap frame tanpa dependensi waktu nyata (delta time).

6. **Konsep Penting**
   - **Gerak Harmonik Sederhana**: Pendulum dianggap sebagai sistem osilasi dengan redaman.
   - **Koordinat Polar ke Cartesian**: Konversi sudut & panjang tali ke posisi (x,y).
   - **Euler Integration**: Metode sederhana untuk mengupdate kecepatan dan posisi.

```
import tkinter as tk
import math
from vector import Vector

# Kelas Pendulum
class Pendulum:
    def __init__(self, origin_x, origin_y, r, canvas):
        self.origin = Vector(origin_x, origin_y)
        self.r = r
        self.angle = math.pi / 4  # Sudut awal 45 derajat
        self.a_velocity = 0.0     # Kecepatan sudut
        self.a_acceleration = 0.0 # Percepatan sudut
        self.damping = 0.995      # Redaman
        self.ball_radius = 24
        self.dragging = False
        self.canvas = canvas

        # Posisi bob
        self.bob = Vector()
        self.update_bob()

        # Objek canvas
        self.line = canvas.create_line(0, 0, 0, 0, width=2)
        self.circle = canvas.create_oval(0, 0, 0, 0, fill='gray')
        self.text = canvas.create_text(10, 10, anchor='nw', font=("Arial", 12), fill="black")

    def update_bob(self):
        self.bob.set(self.r * math.sin(self.angle), self.r * math.cos(self.angle))
        self.bob.add(self.origin)

    def update(self):
        gravity = 0.4
        self.a_acceleration = (-gravity / self.r) * math.sin(self.angle)
        self.a_velocity += self.a_acceleration
        self.angle += self.a_velocity
        self.a_velocity *= self.damping
        self.update_bob()

    def show(self):
        self.canvas.coords(self.line, self.origin.x, self.origin.y, self.bob.x, self.bob.y)
        self.canvas.coords(self.circle,
                           self.bob.x - self.ball_radius, self.bob.y - self.ball_radius,
                           self.bob.x + self.ball_radius, self.bob.y + self.ball_radius)
        info = (
            f"Sudut: {math.degrees(self.angle):.2f}°\n"
            f"Percepatan: {self.a_acceleration:.4f} rad/s²\n"
            f"Kecepatan: {self.a_velocity:.4f} rad/s"
        )
        self.canvas.itemconfig(self.text, text=info)

# Tkinter setup
root = tk.Tk()
root.title("Simulasi Pendulum")

canvas = tk.Canvas(root, width=640, height=360, bg='white')
canvas.pack()

pendulum = Pendulum(320, 50, 175, canvas)

# Loop animasi
```

```
def update():
  pendulum.update()
  pendulum.show()
  root.after(16, update)

update()
root.mainloop()
```

SIMULASI PENDULUM DENGAN INTERVENSI MOUSE

```
import tkinter as tk
import math
from vector import Vector

# Kelas Pendulum
class Pendulum:
    def __init__(self, origin_x, origin_y, r, canvas):
        self.origin = Vector(origin_x, origin_y)
        self.r = r
        self.angle = math.pi / 4  # Sudut awal 45 derajat
        self.a_velocity = 0.0     # Kecepatan sudut
        self.a_acceleration = 0.0 # Percepatan sudut
        self.damping = 0.995      # Redaman
        self.ball_radius = 24
        self.dragging = False
        self.canvas = canvas

        # Posisi bob
        self.bob = Vector()
        self.update_bob()

        # Objek canvas
        self.line = canvas.create_line(0, 0, 0, 0, width=2)
        self.circle = canvas.create_oval(0, 0, 0, 0, fill='gray')
        self.text = canvas.create_text(10, 10, anchor='nw', font=("Arial", 12),
fill="black")

    def update_bob(self):
        self.bob.set(self.r * math.sin(self.angle), self.r * math.cos(self.angle))
        self.bob.add(self.origin)

    def update(self):
        if not self.dragging:
            gravity = 0.4
            self.a_acceleration = (-gravity / self.r) * math.sin(self.angle)
            self.a_velocity += self.a_acceleration
            self.angle += self.a_velocity
            self.a_velocity *= self.damping
        self.update_bob()

    def show(self):
        self.canvas.coords(self.line, self.origin.x, self.origin.y, self.bob.x,
self.bob.y)
        self.canvas.coords(self.circle,
                           self.bob.x - self.ball_radius, self.bob.y - self.ball_radius,
                           self.bob.x + self.ball_radius, self.bob.y + self.ball_radius)
        info = (
            f"Sudut: {math.degrees(self.angle):.2f}°\n"
            f"Percepatan: {self.a_acceleration:.4f} rad/s²\n"
            f"Kecepatan: {self.a_velocity:.4f} rad/s"
        )
        self.canvas.itemconfig(self.text, text=info)

    def clicked(self, mx, my):
        d = math.hypot(mx - self.bob.x, my - self.bob.y)
        if d < self.ball_radius:
            self.dragging = True

    def stop_dragging(self):
        self.a_velocity = 0
        self.dragging = False

    def drag(self, mx, my):
        if self.dragging:
            dx = mx - self.origin.x
            dy = my - self.origin.y
```

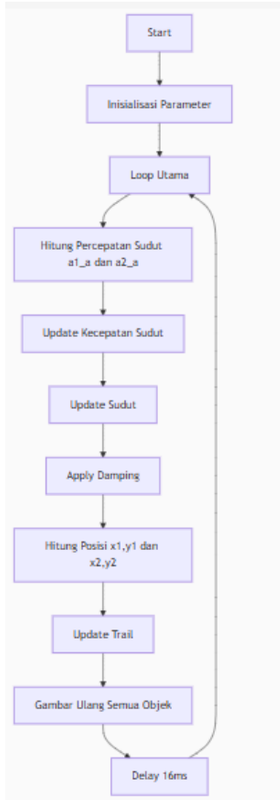
```
            self.angle = math.atan2(dx, dy)  # Sudut dari sumbu vertikal, dibalik supaya
arah mouse benar

# Tkinter setup
root = tk.Tk()
root.title("Simulasi Pendulum")

canvas = tk.Canvas(root, width=640, height=360, bg='white')
canvas.pack()

pendulum = Pendulum(320, 50, 175, canvas)

# Loop animasi
def update():
    pendulum.update()
    pendulum.show()
    root.after(16, update)

# Mouse events
def on_mouse_press(event):
    pendulum.clicked(event.x, event.y)

def on_mouse_release(event):
    pendulum.stop_dragging()

def on_mouse_drag(event):
    pendulum.drag(event.x, event.y)

# Bind mouse
canvas.bind("<ButtonPress-1>", on_mouse_press)
canvas.bind("<B1-Motion>", on_mouse_drag)
canvas.bind("<ButtonRelease-1>", on_mouse_release)

update()
root.mainloop()
```

DOUBLE PENDULUM

1. Apa Itu Pendulum Ganda?

Ini adalah simulasi dua bandul yang saling terhubung:

- Bandul pertama digantung di titik tetap
- Bandul kedua digantung di ujung bandul pertama
- Gerakannya sangat menarik dan tidak terduga!



2. Bagian-Bagian Utama

1. **Titik Gantung**: Di tengah atas layar (x=320, y=100)
2. **Bandul Pertama**:
    - Panjang tali: 100 pixel
    - Massa bola: 10
3. **Bandul Kedua**:
    - Panjang tali: 100 pixel
    - Massa bola: 10
4. **Jejak Biru**: Garis biru menunjukkan lintasan bandul kedua

2. Rumus Fisika Pendulum Ganda

Rumus utama yang digunakan berasal dari persamaan Lagrangian untuk sistem pendulum ganda:

Percepatan Sudut Bola 1 (a1_a):

```
num1 = -g*(2*m1 + m2)*sin(a1)
num2 = -m2*g*sin(a1 - 2*a2)
num3 = -2*sin(a1 - a2)*m2
num4 = a2_v²*r2 + a1_v²*r1*cos(a1 - a2)
den = r1*(2*m1 + m2 - m2*cos(2*a1 - 2*a2))
a1_a = (num1 + num2 + num3*num4) / den
```

Percepatan Sudut Bola 2 (a2_a):

```
num1 = 2*sin(a1 - a2)
num2 = a1_v²*r1*(m1 + m2)
num3 = g*(m1 + m2)*cos(a1)
num4 = a2_v²*r2*m2*cos(a1 - a2)
den = r2*(2*m1 + m2 - m2*cos(2*a1 - 2*a2))
a2_a = (num1*(num2 + num3 + num4)) / den
```

Damping (Redaman):

```
a1_v *= 0.999
a2_v *= 0.999
```

---

3. Perhitungan Tiap Frame

Misalkan pada frame ke-n dengan parameter:

- a1 = π/2 (90°), a2 = π/2
- a1_v = 0.1 rad/s, a2_v = 0.05 rad/s
- r1 = r2 = 100, m1 = m2 = 10
- g = 1

Frame n:

1. Hitung a1_a:
    - num1 = -1*(2*10 + 10)*sin(π/2) = -30
    - num2 = -10*1*sin(π/2 - 2*π/2) = -10*sin(-π/2) = 10
    - num3 = -2*sin(0)*10 = 0
    - num4 = (0.05²*100 + 0.1²*100*cos(0)) = 0.25 + 1 = 1.25
    - den = 100*(20 + 10 - 10*cos(0)) = 100*20 = 2000
    - a1_a = (-30 + 10 + 0) / 2000 ≈ -0.01 rad/s²
2. Hitung a2_a:
    - num1 = 2*sin(0) = 0 → Hasil a2_a = 0 (karena sin(a1 - a2) = 0)
3. Update Kecepatan & Sudut:
    - a1_v = 0.1 + (-0.01) = 0.09 rad/s
    - a2_v = 0.05 + 0 = 0.05 rad/s
    - a1 = π/2 + 0.09 ≈ 1.6708 rad (~95.7°)
    - a2 = π/2 + 0.05 ≈ 1.6208 rad (~92.9°)
4. Posisi Bola:
    - x1 = 100*sin(1.6708) ≈ 99.5
    - y1 = 100*cos(1.6708) ≈ -8.7
    - x2 = 99.5 + 100*sin(1.6208) ≈ 99.5 + 99.9 = 199.4
    - y2 = -8.7 + 100*cos(1.6208) ≈ -8.7 + (-3.8) = -12.5
    - Koordinat layar: (x1+320, y1+100) dan (x2+320, y2+100)
5. Gambar Ulang:
    - Garis dari (cx,cy) ke (x1,y1) dan (x1,y1) ke (x2,y2).
    - Lingkaran di (x1,y1) dan (x2,y2).

---

4. Visualisasi Tkinter

- Trail/Jejak: Menyimpan 1000 titik terakhir dari (x2,y2) untuk menggambar garis biru.
- Batang & Bola:
    - Batang 1: (cx,cy) ke (x1,y1)
    - Batang 2: (x1,y1) ke (x2,y2)
    - Bola: Lingkaran dengan radius m1 dan m2.
- Teks Info: Menampilkan sudut a1 dan a2 dalam radian.

---

5. Contoh Output Visual

a1 = 1.67 rad

a2 = 1.62 rad

- Gerakan kacau (chaotic) khas pendulum ganda.
- Trail biru menunjukkan lintasan bola kedua.

---

6. Konsep Penting

1. Sistem Dinamis Non-Linear: Persamaan kompleks dengan ketergantungan pada sin(a1 - a2).
2. Damping: Faktor 0.999 untuk mencegah osilasi tak terhingga.
3. Koordinat Relatif: Posisi bola kedua dihitung relatif terhadap bola pertama.
4. Chaos Theory: Perubahan kecil di sudut awal menghasilkan pola gerakan yang sangat berbeda.

Simulasi ini menunjukkan bagaimana sistem fisika sederhana bisa menghasilkan perilaku kompleks!

```
import tkinter as tk
import math

# Window dimensions
WIDTH = 640
HEIGHT = 480

# Pendulum parameters
r1, r2 = 100, 100     # Panjang tali
m1, m2 = 10, 10       # Massa bola
a1, a2 = math.pi / 2, math.pi / 2  # Sudut awal (radian)
a1_v, a2_v = 0.0, 0.0 # Kecepatan sudut awal
g = 1             # Gravitasi

# Center point
cx, cy = WIDTH / 2, 100

# Koordinat sebelumnya untuk jejak
```

```
px2, py2 = -1, -1
trail = []

# Fungsi utama update setiap frame
def update():
    global a1, a2, a1_v, a2_v, px2, py2

    # Rumus percepatan sudut a1_a (dari physics pendulum ganda)
    num1 = -g * (2 * m1 + m2) * math.sin(a1)
    num2 = -m2 * g * math.sin(a1 - 2 * a2)
    num3 = -2 * math.sin(a1 - a2) * m2
    num4 = a2_v * a2_v * r2 + a1_v * a1_v * r1 * math.cos(a1 - a2)
    den = r1 * (2 * m1 + m2 - m2 * math.cos(2 * a1 - 2 * a2))
    a1_a = (num1 + num2 + num3 * num4) / den

    num1 = 2 * math.sin(a1 - a2)
    num2 = a1_v * a1_v * r1 * (m1 + m2)
    num3 = g * (m1 + m2) * math.cos(a1)
    num4 = a2_v * a2_v * r2 * m2 * math.cos(a1 - a2)
    den = r2 * (2 * m1 + m2 - m2 * math.cos(2 * a1 - 2 * a2))
    a2_a = (num1 * (num2 + num3 + num4)) / den

    # Perbarui kecepatan dan sudut
    a1_v += a1_a
    a2_v += a2_a
    a1 += a1_v
    a2 += a2_v

    # Damping agar gerakan lebih stabil
    a1_v *= 0.999
    a2_v *= 0.999

    # Hitung posisi bola
    x1 = r1 * math.sin(a1)
    y1 = r1 * math.cos(a1)
    x2 = x1 + r2 * math.sin(a2)
    y2 = y1 + r2 * math.cos(a2)

    # Pindah koordinat ke layar
    x1 += cx
    y1 += cy
    x2 += cx
    y2 += cy

    # Tambah trail
    if px2 != -1:
        trail.append((px2, py2, x2, y2))
        if len(trail) > 1000:
            trail.pop(0)
    px2, py2 = x2, y2

    # Gambar ulang semua
    canvas.delete("all")

    # Gambar trail jejak
    for x0, y0, x1t, y1t in trail:
        canvas.create_line(x0, y0, x1t, y1t, fill="blue")

    # Gambar batang dan bola
    canvas.create_line(cx, cy, x1, y1, width=2)
    canvas.create_oval(x1 - m1, y1 - m1, x1 + m1, y1 + m1, fill="black")

    canvas.create_line(x1, y1, x2, y2, width=2)
    canvas.create_oval(x2 - m2, y2 - m2, x2 + m2, y2 + m2, fill="black")

    # Tampilkan informasi
```

```
    canvas.create_text(10, 10, anchor="nw", text=f"a1 = {a1:.2f} rad")
    canvas.create_text(10, 30, anchor="nw", text=f"a2 = {a2:.2f} rad")

    # Jadwalkan frame berikutnya
    canvas.after(16, update)

# Setup Tkinter window
root = tk.Tk()
root.title("Double Pendulum - Python Tkinter")

canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

update()
root.mainloop()
```

gaya pegas (spring forces)

Apa Itu Gaya Pegas? (Spring Force)
Pernah main **yoyo**, **slinki**, atau **karet gelang**?
Ketika kamu **tarik** pegas (atau karet), pegas akan **menarik balik** ke posisi semula. Nah, gaya yang mendorong benda kembali ke posisi awal ini disebut **gaya pegas**.

Hukum Hooke (Hooke's Law)
Seorang ilmuwan bernama **Robert Hooke** menemukan bahwa:
Semakin jauh kamu menarik pegas, semakin besar pegas akan menarik balik.
Dalam bahasa matematika:
F = -k * x
Penjelasan:

- **F** = gaya pegas (arahnya ke dalam, makanya ada tanda minus)
- **k** = kekuatan pegas (pegas yang kaku punya nilai k besar, pegas lembek nilainya kecil)
- **x** = seberapa jauh kamu menarik atau menekan pegas dari panjang awalnya (rest length)

Contoh Sehari-Hari:
Bayangkan kamu menggantungkan bola pada karet gelang dari langit-langit.

- Ketika diam, panjang karet = **rest length**
- Ketika kamu tarik bola ke bawah, karet **memanjang**
- Semakin panjang tarikanmu, karet akan semakin ingin **balik ke posisi semula**
- Gaya ini yang kita sebut **gaya pegas**

Simulasi di Komputer
Di komputer, kita pakai **vektor** untuk menunjukkan arah dan jarak.
1. Titik Anchor
Tempat ujung atas pegas (misalnya langit-langit)
PVector anchor;
2. Titik Bob (Bola)
Lokasi bola yang tergantung
PVector location;
3. Panjang Normal Pegas
Panjang pegas saat tidak ditarik atau ditekan
float restLength;

Langkah-Langkah Menghitung Gaya Pegas

1. **Cari arah dari anchor ke bola**

PVector dir = PVector.sub(location, anchor);

2. **Hitung panjang saat ini**

float currentLength = dir.mag(); // jarak antara anchor dan bola

3. **Hitung seberapa banyak pegas berubah (x)**

float x = restLength - currentLength;

4. **Hitung besar gaya pegas**

float k = 0.1; // konstanta kekakuan pegas
float force = -k * x;

5. **Tentukan arah gaya**

Kita pakai dir.normalize() untuk mendapatkan arah 1 satuan (unit vector).

6. **Buat gaya pegas sebagai vektor**

dir.normalize(); // arah
dir.mult(force); // dikali besar gaya

Kesimpulan

- Gaya pegas selalu **menarik kembali** ke panjang normal.
- Semakin jauh kamu tarik, semakin besar gaya tarik baliknya.
- Kita bisa menghitung ini dengan **vektor** dan hukum Hooke.
- Dengan pemrograman, kita bisa membuat simulasi bola yang tergantung pada pegas dan bergerak naik-turun seperti nyata!

---

simulasi pegas:

2. Rumus Fisika yang Digunakan

Gaya Pegas (Hukum Hooke):

```
F_spring = -k * Δx
```

- k: Konstanta pegas (0.2 dalam kode)
- Δx: Selisih panjang pegas dari panjang istirahat (current_length - rest_length)

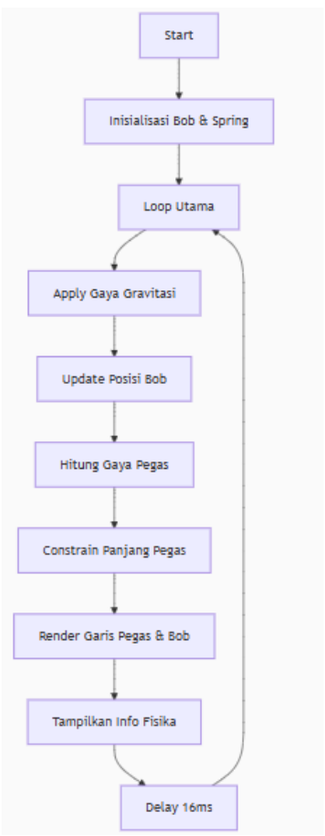
Pergerakan Bob (Hukum Newton II):

```
a = F_total / m
v = v_prev + a * Δt
s = s_prev + v * Δt
```

- Δt: Dianggap 1 (karena loop berjalan cepat)

Gaya Gravitasi:

`F_gravity = m * g → (0, 0.98 * m) dalam kode`



3. Perhitungan Tiap Frame

Misalkan pada frame ke-n dengan parameter:

- bob.position = (320, 100)
- bob.velocity = (0, 0)
- spring.rest_length = 100
- spring.anchor = (320, 10)
- bob.mass = 20

Frame n:

1. **Apply Gaya Gravitasi**:

gravity = Vector(0, 0.98)
bob.apply_force(gravity) *# F = (0, 0.98*20) = (0, 19.6)*

2. **Hitung Gaya Pegas**:
   - Vektor dari anchor ke bob:

force = bob.position - spring.anchor = (320-320, 100-10) = (0, 90)

   - Panjang pegas saat ini:

$d = \sqrt{(0^2 + 90^2)} = 90$

   - Selisih panjang:

stretch = 90 - 100 = -10

   - Gaya pegas:

F_spring = -0.2 * (-10) * (0,90).normalized() = 2 * (0,1) = (0, 2)

3. **Total Gaya pada Bob**:

F_total = F_gravity + F_spring = (0, 19.6) + (0, 2) = (0, 21.6)

4. **Update Percepatan**:

a = F_total / m = (0, 21.6) / 20 = (0, 1.08)

5. **Update Kecepatan & Posisi**:
   - Kecepatan baru:

v = (0,0) + (0,1.08) = (0,1.08)

   - Posisi baru:

s = (320,100) + (0,1.08) = (320,101.08)

6. **Constrain Panjang Pegas** (jika melewati batas min/max):
   - Jika d < 30 atau d > 200, posisi dipaksa ke batas terdekat.

---

4. Visualisasi Tkinter

- **Bob**: Lingkaran biru dengan radius 20 piksel.
- **Pegas**: Garis hitam dari anchor ke bob.
- **Anchor**: Titik hitam kecil di ujung pegas.
- **Info Teks**:
  - Gaya pegas (F_spring)
  - Kecepatan bob (velocity)
  - Percepatan bob (acceleration)

---

6. Fitur Interaktif

1. **Drag & Drop**:

- Klik pada bola untuk menggerakkannya.
- Saat didrag, kecepatan direset ke nol.

2. **Fisika Realistis**:
   - Bola akan berosilasi seperti sistem pegas-massa nyata.
   - Gravitasi dan konstanta pegas dapat diubah.

---

7. Konsep Penting

1. **Hukum Hooke**: Gaya pegas proporsional dengan perubahan panjang.
2. **Integrasi Euler Sederhana**:
   v += a
   s += v
3. **Vektor 2D**: Semua perhitungan menggunakan operasi vektor.
4. **Energy Loss**: Tidak ada redaman eksplisit, tetapi constrain panjang membatasi energi.

Simulasi ini menunjukkan dasar fisika untuk sistem seperti:

- Permainan dengan tali/pegas
- Simulasi cloth physics
- Sistem partikel elastis

🌈 **Contoh Gerakan**

1. Tarik bola ke bawah dan lepaskan → akan naik-turun seperti pegas
2. Tarik bola ke samping → akan berayun seperti bandul
3. Gerakan akan perlahan berkurang karena ada efek redaman

💡 **Fakta Menarik**

- Semakin besar nilai k (0.2 di program), pegas semakin kaku
- Semakin besar massa bola, gerakan akan semakin lambat
- Coba ubah nilai gravitasi untuk melihat efek berbeda!

Program ini menunjukkan hubungan antara gaya pegas dan gerakan benda dengan cara yang menyenangkan dan interaktif! 🚀

```
import tkinter as tk
from vector import Vector

# ==== Bob (bola) ====
class Bob:
    def __init__(self, x, y, canvas):
        self.position = Vector(x, y)
        self.velocity = Vector(0, 0)
        self.acceleration = Vector(0, 0)
        self.mass = 20
        self.radius = 20
        self.dragging = False
        self.canvas = canvas
        self.id = canvas.create_oval(x - self.radius, y - self.radius,
                                     x + self.radius, y + self.radius,
                                     fill="blue")

    def apply_force(self, force):
        # F = m * a → a = F / m
        f = force.dived(self.mass)
        self.acceleration = self.acceleration.added(f)

    def update(self):
        if not self.dragging:
            # v = v + a
            self.velocity = self.velocity.added(self.acceleration)
            # s = s + v
            self.position = self.position.added(self.velocity)
        self.acceleration = Vector(0, 0)
        self.canvas.coords(self.id,
            self.position.x - self.radius,
            self.position.y - self.radius,
            self.position.x + self.radius,
            self.position.y + self.radius)

    def handle_click(self, mx, my):
        d = Vector(mx, my).subbed(self.position).mag()
        if d < self.radius:
            self.dragging = True

    def stop_dragging(self):
        self.dragging = False
```

```
    def handle_drag(self, mx, my):
        if self.dragging:
            self.position = Vector(mx, my)
            self.velocity = Vector(0, 0)

# ==== Spring ====
class Spring:
    def __init__(self, x, y, rest_length, canvas):
        self.anchor = Vector(x, y)
        self.rest_length = rest_length
        self.k = 0.2  # konstanta pegas
        self.canvas = canvas
        self.line_id = canvas.create_line(x, y, x, y)
        self.anchor_id = canvas.create_oval(x - 5, y - 5, x + 5, y + 5, fill="black")

    def connect(self, bob):
        # hitung gaya pegas
        force = bob.position.subbed(self.anchor)  # arah gaya: dari anchor ke bob
        d = force.mag()  # panjang saat ini
        stretch = d - self.rest_length  # selisih panjang

        # Fspring = -k * x
        force = force.normalized().multed(-self.k * stretch)
        bob.apply_force(force)
        return force  # dikembalikan agar bisa ditampilkan

    def constrain_length(self, bob, minlen, maxlen):
        direction = bob.position.subbed(self.anchor)
        d = direction.mag()
        if d < minlen:
            direction = direction.normalized().multed(minlen)
            bob.position = self.anchor.added(direction)
            bob.velocity = Vector(0, 0)
        elif d > maxlen:
            direction = direction.normalized().multed(maxlen)
            bob.position = self.anchor.added(direction)
            bob.velocity = Vector(0, 0)

    def show_line(self, bob):
        self.canvas.coords(self.line_id,
                           bob.position.x, bob.position.y,
                           self.anchor.x, self.anchor.y)

# ==== Main App ====
def main():
    root = tk.Tk()
    root.title("Spring Simulation")
    canvas = tk.Canvas(root, width=640, height=300, bg="white")
    canvas.pack()

    spring = Spring(320, 10, 100, canvas)
    bob = Bob(320, 100, canvas)

    text_id_force = canvas.create_text(10, 260, anchor="w", text="", font=("Consolas",
12))
    text_id_velocity = canvas.create_text(10, 280, anchor="w", text="", font=("Consolas",
12))
    text_id_accel = canvas.create_text(300, 280, anchor="w", text="", font=("Consolas",
12))

    mouse = {"x": 0, "y": 0}

    def update():
        # gaya gravitasi konstan ke bawah
        gravity = Vector(0, 0.98)
        bob.apply_force(gravity)

        bob.update()
        bob.handle_drag(mouse["x"], mouse["y"])

        # hitung dan aplikasikan gaya pegas ke bob
        spring_force = spring.connect(bob)

        # batasi panjang pegas
        spring.constrain_length(bob, 30, 200)
```

```
        # update garis pegas
        spring.show_line(bob)

        # tampilkan gaya, kecepatan, percepatan
        canvas.itemconfig(text_id_force,
                          text=f"F_spring = ({spring_force.x:.2f}, {spring_force.y:.2f})")
        canvas.itemconfig(text_id_velocity,
                          text=f"velocity  = ({bob.velocity.x:.2f},
{bob.velocity.y:.2f})")
        canvas.itemconfig(text_id_accel,
                          text=f"accel     = ({bob.acceleration.x:.2f},
{bob.acceleration.y:.2f})")

        root.after(16, update)  # ~60 FPS

    def on_mouse_press(event):
        bob.handle_click(event.x, event.y)

    def on_mouse_release(event):
        bob.stop_dragging()

    def on_mouse_motion(event):
        mouse["x"] = event.x
        mouse["y"] = event.y

    canvas.bind("<ButtonPress-1>", on_mouse_press)
    canvas.bind("<ButtonRelease-1>", on_mouse_release)
    canvas.bind("<Motion>", on_mouse_motion)

    update()
    root.mainloop()

if __name__ == "__main__":
    main()
```

simulasi pegas ganda:

Apa Itu Program Ini?

Program ini membuat simulasi 3 bola yang saling terhubung dengan pegas:

- Ada 3 bola abu-abu (b1, b2, b3) yang digantung berjajar
- Setiap bola dihubungkan dengan pegas ke bola lainnya
- Bola paling atas (b1) bisa ditarik dengan mouse
- Sistem akan bergerak mengikuti hukum fisika pegas

Bagian-Bagian Utama

1. **3 Bola Abu-abu**:
    - b1: Posisi atas (y=100)
    - b2: Posisi tengah (y=200)
    - b3: Posisi bawah (y=300)
    - Semua bola punya massa 24 dan radius 24 pixel
2. **3 Pegas**:
    - s1: Menghubungkan b1 dan b2
    - s2: Menghubungkan b2 dan b3
    - s3: Menghubungkan b1 dan b3
    - Panjang normal setiap pegas: 100 pixel
    - Kekakuan pegas (k): 0.2

Cara Kerja Program

1. **Setiap Frame (60 kali/detik)**:
    - Hitung semua gaya pegas yang bekerja
    - Update posisi semua bola
    - Gambar ulang posisi bola dan pegas
2. **Hukum Hooke (Hukum Pegas)**:

Gaya pegas = -k × (panjang_sekarang - panjang_normal)

    - Semakin jauh ditarik, semakin besar gaya tariknya kembali
3. **Gerakan Bola**:
    - Bola paling atas (b1) bisa ditarik dengan mouse
    - Saat dilepas, semua bola akan bergerak saling mempengaruhi
    - Ada efek redaman (damping) sehingga gerakan perlahan berhenti

Interaksi Mouse

1. **Klik kiri** pada b1: Mulai menarik bola
2. **Gerakkan mouse**: Bola akan mengikuti
3. **Lepas klik**: Bola akan bergerak alami dengan gaya pegas

📊 **Informasi Fisika**
Di pojok kiri atas ditampilkan:

- **F**: Gaya total pada b1 (dalam x dan y)
- **v**: Kecepatan b1
- **a**: Percepatan b1

🔄 **Contoh Gerakan**

1. Tarik b1 ke kiri → semua bola akan bergerak ke kiri
2. Tarik b1 ke atas → pegas akan menariknya kembali ke bawah
3. Gerakan akan membentuk pola yang kompleks karena saling terhubung

💡 **Fakta Menarik**

- Sistem ini disebut "massa-pegas" dalam fisika
- Semua bola saling mempengaruhi gerakannya
- Coba ubah nilai k jadi lebih besar (misal 0.5) untuk pegas lebih kaku
- Ubah damping mendekati 1 untuk gerakan lebih lama

Program ini menunjukkan bagaimana benda-benda saling terhubung di dunia nyata dengan cara yang menyenangkan! 🌈
Berikut contoh perhitungan gerakan sederhana untuk bola pertama (b1) dalam satu frame simulasi, dijelaskan langkah demi langkah dengan angka sederhana:
Kondisi Awal:

- Posisi b1: (320, 100)
- Posisi b2: (320, 200)
- Panjang normal pegas (s1): 100
- Kekakuan pegas (k): 0.2
- Massa bola: 24
- Gravitasi: (0, 0.98)

Langkah 1: Hitung Gaya Pegas (s1) antara b1 dan b2

1. Hitung jarak aktual b1 ke b2:

jarak_aktual = √[(320-320)² + (200-100)²] = 100
(Karena posisi vertikal saja, jarak = 200-100 = 100)

2. Hitung perpanjangan pegas:

stretch = jarak_aktual - panjang_normal = 100 - 100 = 0
(Pegas tidak meregang/stretch = 0)

3. Gaya pegas (Hukum Hooke):

gaya_pegas = -k × stretch = -0.2 × 0 = 0
(Tidak ada gaya karena pegas tidak meregang)

Langkah 2: Hitung Gaya Gravitasi

- Gaya gravitasi pada b1:

gaya_gravitasi = Vector(0, 0.98 × massa) = (0, 0.98 × 24) = (0, 23.52)

Langkah 3: Total Gaya pada b1
total_gaya = gaya_pegas + gaya_gravitasi = (0, 0) + (0, 23.52) = (0, 23.52)

Langkah 4: Hitung Percepatan (a = F/m)
percepatan = total_gaya / massa = (0, 23.52) / 24 = (0, 0.98)

Langkah 5: Update Kecepatan dan Posisi

1. **Kecepatan baru** (dengan redaman 0.98):

kecepatan_baru = (kecepatan_lama + percepatan) × damping
Misal kecepatan_lama = (0, 0):
kecepatan_baru = (0 + 0, 0 + 0.98) × 0.98 = (0, 0.96)

2. **Posisi baru**:

posisi_baru = posisi_lama + kecepatan_baru
posisi_baru = (320, 100) + (0, 0.96) = (320, 100.96)

Hasil Akhir dalam 1 Frame:

- **Posisi b1** bergerak dari (320, 100) → (320, 100.96)
- **Penyebab gerakan**: Gaya gravitasi menarik bola ke bawah
- **Pegas** belum bekerja karena belum ada perpanjangan

Contoh Kasus dengan Tarikan Mouse:
Jika b1 ditarik ke (320, 50) lalu dilepas:

1. Jarak b1-b2 = 200 - 50 = 150
2. stretch = 150 - 100 = 50
3. gaya_pegas = -0.2 × 50 = -10 (ke atas)

4. Bola akan bergerak naik karena gaya pegas > gravitasi (10 vs 0.98×24=23.52).

(Note: Perhitungan disederhanakan dengan mengabaikan pegas s3 dan efek bola b3 untuk memudahkan pemahaman.)

Ini menunjukkan bagaimana program menghitung gerakan alami pegas!

```
# spring_simulation.py
import tkinter as tk
from vector import Vector

class Bob:
    def __init__(self, canvas, x, y, mass=24):
        self.canvas = canvas
        self.position = Vector(x, y)
        self.velocity = Vector()
        self.acceleration = Vector()
        self.mass = mass
        self.damping = 0.98
        self.drag_offset = Vector()
        self.dragging = False
        self.radius = mass
        self.id = canvas.create_oval(x - mass, y - mass, x + mass, y + mass, fill="gray")

    def apply_force(self, force):
        # F = m * a => a = F / m
        f = force.dived(self.mass)
        self.acceleration = self.acceleration.added(f)

    def update(self):
        if not self.dragging:
            self.velocity = self.velocity.added(self.acceleration)
            self.velocity = self.velocity.multed(self.damping)
            self.position = self.position.added(self.velocity)
        self.acceleration = Vector()  # Reset percepatan
        self.canvas.coords(self.id,
            self.position.x - self.radius,
            self.position.y - self.radius,
            self.position.x + self.radius,
            self.position.y + self.radius)

    def handle_click(self, mx, my):
        dx = mx - self.position.x
        dy = my - self.position.y
        if dx * dx + dy * dy < self.radius * self.radius:
            self.dragging = True
            self.drag_offset = Vector(self.position.x - mx, self.position.y - my)

    def stop_dragging(self):
        self.dragging = False

    def handle_drag(self, mx, my):
        if self.dragging:
            self.position = Vector(mx, my).added(self.drag_offset)

class Spring:
    def __init__(self, canvas, a, b, length, k=0.2):
        self.canvas = canvas
        self.a = a
        self.b = b
        self.length = length
        self.k = k
        self.id = canvas.create_line(0, 0, 0, 0)

    def update(self):
        force = self.a.position.subbed(self.b.position)  # arah
        distance = force.mag()
        stretch = distance - self.length
        force = force.normalized().multed(-1 * self.k * stretch)  # Hooke’s Law: F = -k *
x

        # Terapkan gaya pada masing-masing bob
        self.a.apply_force(force)
        self.b.apply_force(force.multed(-1))

        # Perbarui visual garis pegas
        self.canvas.coords(self.id,
```

```
            self.a.position.x, self.a.position.y,
            self.b.position.x, self.b.position.y)

def main():
    root = tk.Tk()
    root.title("Exercise 3.16 - Multiple Springs")
    canvas = tk.Canvas(root, width=640, height=360, bg="white")
    canvas.pack()

    b1 = Bob(canvas, 320, 100)
    b2 = Bob(canvas, 320, 200)
    b3 = Bob(canvas, 320, 300)

    s1 = Spring(canvas, b1, b2, 100)
    s2 = Spring(canvas, b2, b3, 100)
    s3 = Spring(canvas, b1, b3, 100)

    mouse = {"x": 0, "y": 0, "pressed": False}

    def on_mouse_press(event):
        mouse["pressed"] = True
        b1.handle_click(event.x, event.y)

    def on_mouse_release(event):
        mouse["pressed"] = False
        b1.stop_dragging()

    def on_mouse_motion(event):
        mouse["x"], mouse["y"] = event.x, event.y

    def update():
        canvas.delete("info")

        b1.handle_drag(mouse["x"], mouse["y"])
        b1.update()
        b2.update()
        b3.update()

        s1.update()
        s2.update()
        s3.update()

        # Tampilkan informasi gaya, kecepatan, dan percepatan b1
        info = f"F: ({b1.acceleration.x * b1.mass:.2f}, {b1.acceleration.y * b1.mass:.2f})
| "
        info += f"v: {b1.velocity} | a: {b1.acceleration}"
        canvas.create_text(10, 10, anchor="nw", text=info, fill="black", font=("Arial",
10), tags="info")

        root.after(16, update)

    canvas.bind("<ButtonPress-1>", on_mouse_press)
    canvas.bind("<ButtonRelease-1>", on_mouse_release)
    canvas.bind("<Motion>", on_mouse_motion)

    update()
    root.mainloop()

if __name__ == "__main__":
    main()
```

simulasi deretan pegas :

Apa Itu Program Ini?

Program ini membuat simulasi 5 bola abu-abu yang saling terhubung dengan pegas seperti rantai:

- Ada 5 bola yang tersusun vertikal
- Setiap bola dihubungkan ke bola di bawahnya dengan pegas
- Bola bisa ditarik dengan mouse dan akan bergerak seperti pegas
- Ada gaya gravitasi yang menarik bola ke bawah

Bagian-Bagian Utama

1. **5 Bola Abu-abu**:
    - Posisi awal: (320, 50), (320, 90), (320, 130), (320, 170), (320, 210)
    - Massa setiap bola: 24
    - Ukuran: radius 12 pixel (setengah dari massa)

2. **4 Pegas Hitam**:
    - Menghubungkan bola 1-2, 2-3, 3-4, dan 4-5
    - Panjang normal: 40 pixel
    - Kekakuan pegas (k): 0.2

⚙️ **Cara Kerja Program**

1. **Setiap Frame (60 kali/detik)**:
    - Hitung semua gaya pegas antara bola-bola
    - Update posisi semua bola
    - Gambar ulang posisi bola dan pegas
2. **Hukum Pegas (Hooke's Law)**:

Gaya pegas = -k × (panjang_sekarang - panjang_normal)

    - Contoh: Jika pegas diregangkan jadi 50 pixel (awal 40):

Gaya = -0.2 × (50 - 40) = -2 (tarik ke atas)

🖱️ **Interaksi Mouse**

1. **Klik kiri** pada bola: Bisa mulai menarik bola
2. **Gerakkan mouse**: Bola yang diklik akan mengikuti
3. **Lepas klik**: Bola akan bergerak alami dengan gaya pegas

📉 **Gaya yang Bekerja**

1. **Gravitasi**:
    - Setiap bola ditarik ke bawah dengan gaya (0, 0.98 × massa)
    - Contoh: Untuk massa 24 → (0, 23.52)
2. **Gaya Pegas**:
    - Akan menarik bola kembali ke posisi normalnya

🔄 **Contoh Gerakan Sederhana**

Misal kita tarik bola ke-3 ke posisi (320, 100):

1. Pegas 2-3 akan meregang:
    - Jarak baru = 100 - 130 = -30 (harusnya dihitung dengan rumus jarak)
    - Gaya = -0.2 × (-30 - 40) = 14 (ke atas)
2. Pegas 3-4 akan menekan:
    - Jarak baru = 170 - 100 = 70
    - Gaya = -0.2 × (70 - 40) = -6 (ke bawah)
3. Bola akan bergerak mencari titik seimbang baru

💡 **Fakta Menarik**

- Semua bola saling mempengaruhi gerakannya
- Coba ubah nilai k jadi lebih besar (misal 0.5) untuk pegas lebih kaku
- Ubah damping mendekati 1 untuk gerakan lebih lama

Program ini menunjukkan bagaimana benda-benda saling terhubung di dunia nyata dengan cara yang menyenangkan! 🌈

🎮 **Coba Sendiri!**

1. Tarik bola di tengah ke samping → lihat efeknya ke bola lain
2. Tarik bola paling bawah → hanya pegas di atasnya yang berpengaruh
3. Amati bagaimana gerakan perlahan berhenti karena redaman

```
from tkinter import *
from vector import Vector
import math

class Bob:
    def __init__(self, x, y, canvas):
        self.position = Vector(x, y)
        self.velocity = Vector(0, 0)
        self.acceleration = Vector(0, 0)
        self.mass = 24
        self.damping = 0.98
        self.dragging = False
        self.drag_offset = Vector(0, 0)
        self.canvas = canvas
        self.id = canvas.create_oval(x - self.mass / 2, y - self.mass / 2,
                                     x + self.mass / 2, y + self.mass / 2,
                                     fill="gray")

    def apply_force(self, force):
        f = force.dived(self.mass)
        self.acceleration = self.acceleration.added(f)

    def update(self):
        if not self.dragging:
            self.velocity = self.velocity.added(self.acceleration)
```

```
            self.velocity = self.velocity.multed(self.damping)
            self.position = self.position.added(self.velocity)
            self.acceleration = Vector(0, 0)

        r = self.mass / 2
        self.canvas.coords(self.id,
                           self.position.x - r, self.position.y - r,
                           self.position.x + r, self.position.y + r)

    def handle_click(self, mx, my):
        d = math.dist([mx, my], [self.position.x, self.position.y])
        if d < self.mass:
            self.dragging = True
            self.drag_offset = Vector(self.position.x - mx, self.position.y - my)

    def stop_dragging(self):
        self.dragging = False

    def handle_drag(self, mx, my):
        if self.dragging:
            self.position = Vector(mx, my).added(self.drag_offset)

class Spring:
    def __init__(self, a, b, length, canvas):
        self.a = a
        self.b = b
        self.length = length
        self.k = 0.2
        self.canvas = canvas
        self.id = canvas.create_line(a.position.x, a.position.y,
                                     b.position.x, b.position.y,
                                     fill="black", width=2)

    def update(self):
        force = self.a.position.subbed(self.b.position)
        stretch = force.mag() - self.length
        force = force.normalized().multed(-1 * self.k * stretch)
        self.a.apply_force(force)
        self.b.apply_force(force.multed(-1))
        self.canvas.coords(self.id,
                           self.a.position.x, self.a.position.y,
                           self.b.position.x, self.b.position.y)

def main():
    root = Tk()
    root.title("Spring Array Simulation")
    canvas = Canvas(root, width=640, height=360, bg="white")
    canvas.pack()

    bobs = [Bob(320, i * 40 + 50, canvas) for i in range(5)]
    springs = [Spring(bobs[i], bobs[i+1], 40, canvas) for i in range(4)]
    mouse = {"x": 0, "y": 0}

    def on_mouse_press(event):
        for b in bobs:
            b.handle_click(event.x, event.y)

    def on_mouse_release(event):
        for b in bobs:
            b.stop_dragging()

    def on_mouse_motion(event):
        mouse["x"], mouse["y"] = event.x, event.y

    def update():
        for b in bobs:
            b.handle_drag(mouse["x"], mouse["y"])
        for s in springs:
            s.update()
        for b in bobs:
            b.update()
        canvas.after(16, update)

    canvas.bind("<ButtonPress-1>", on_mouse_press)
    canvas.bind("<ButtonRelease-1>", on_mouse_release)
    canvas.bind("<Motion>", on_mouse_motion)
```

```
    update()
    root.mainloop()

if __name__ == "__main__":
    main()
```

Berikut penjelasan sederhana tentang program simulasi gravitasi

💥 **Apa Itu Program Ini?**

Program ini membuat simulasi:

- 6 "robot kecil" (crawler) abu-abu yang bergerak acak
- 1 "magnet" besar (attractor) di tengah yang bisa ditarik dengan mouse
- Robot-robot akan tertarik ke magnet seperti gaya gravitasi

🧱 **Bagian-Bagian Utama**

1. **Robot Kecil (Crawler)**:
    - Bentuk: Lingkaran abu-abu (ukuran 8-16 pixel)
    - Punya "antena" yang bergetar sesuai kecepatannya
    - Bergerak acak di awal
2. **Magnet Besar (Attractor)**:
    - Bentuk: Lingkaran abu-abu/hitam di tengah layar
    - Bisa ditarik dengan mouse
    - Menarik robot-robot kecil seperti magnet
3. **Gaya Gravitasi**:
    - Kekuatan: 0.4 (bisa diubah)
    - Semakin dekat robot ke magnet, semakin kuat tarikannya

⚙️ **Cara Kerja Program**

1. **Setiap Frame (60 kali/detik)**:
    - Hitung gaya tarik magnet ke setiap robot
    - Update posisi semua robot
    - Gambar ulang semua objek
2. **Rumus Gaya Tarik**:

Gaya = (G × massa_magnet × massa_robot) / (jarak²)

    - G = 0.4 (seperti kekuatan magnet)
    - Jarak minimal 5 pixel, maksimal 25 pixel

🖱️ **Interaksi Mouse**

1. **Klik kiri** pada magnet: Bisa mulai menarik magnet
2. **Gerakkan mouse**: Magnet akan mengikuti
3. **Lepas klik**: Magnet akan berhenti di posisi baru

🌀 **Contoh Gerakan**

1. Saat magnet di tengah:
    - Robot-robot akan berputar mengelilingi magnet seperti planet
2. Saat magnet ditarik ke samping:
    - Robot-robot akan mengikuti magnet seperti ditarik tali
3. Antena robot:
    - Akan bergetar lebih cepat jika robot bergerak cepat
    - Getarannya menggunakan rumus sinus (math.cos)

💡 **Fakta Menarik**

- Semakin besar massa robot, semakin lambat gerakannya
- Coba ubah nilai G jadi lebih besar (misal 1.0) untuk magnet super kuat
- Robot yang jauh dari magnet akan bergerak lebih bebas

🧲 **Analog Sederhana**

Bayangkan:

- Magnet besar seperti Matahari
- Robot-robot seperti planet kecil
- Gaya tarik magnet seperti gaya gravitasi
- Antena robot seperti sensor yang bergetar

Program ini menunjukkan bagaimana gaya tarik menarik bekerja di alam semesta dengan cara yang menyenangkan! 🚀

🎮 **Coba Sendiri!**

1. Tarik magnet dan lihat robot-robot mengikutinya
2. Amati bagaimana antena robot bergetar berbeda saat bergerak cepat/lambat
3. Coba ubah jumlah robot dengan mengubah angka 6 di range(6)

Berikut contoh perhitungan sederhana untuk **1 robot (crawler)** dan **magnet (attractor)** dalam **1 frame** (16ms), dijelaskan langkah demi langkah dengan angka yang mudah dipahami:

Kondisi Awal:

- **Magnet (Attractor)**:
  - Posisi: (320, 180)
  - Massa: 20
  - Gaya tarik (G): 0.4
- **Robot (Crawler)**:
  - Posisi: (200, 150)
  - Massa: 10
  - Kecepatan awal: (1, 0)

Langkah 1: Hitung Gaya Tarik Magnet ke Robot

1. Hitung jarak robot ke magnet:

dx = 320 - 200 = 120
dy = 180 - 150 = 30
jarak = √(120² + 30²) = √(14400 + 900) ≈ 123.7 pixel

2. Batasi jarak (min 5, max 25):

jarak_efektif = 25 (karena 123.7 > 25)

3. Hitung gaya tarik:

Gaya = (G × massa_magnet × massa_robot) / (jarak²)
= (0.4 × 20 × 10) / (25²)
= 80 / 625
≈ 0.128

4. Arah gaya (vektor dari robot ke magnet):

arah_x = 120 / 123.7 ≈ 0.97
arah_y = 30 / 123.7 ≈ 0.24
Gaya_x = 0.128 × 0.97 ≈ 0.124
Gaya_y = 0.128 × 0.24 ≈ 0.031

Langkah 2: Update Posisi Robot

1. **Percepatan** (a = F/m):

a_x = 0.124 / 10 ≈ 0.0124
a_y = 0.031 / 10 ≈ 0.0031

2. **Kecepatan Baru**:

v_x_baru = 1 (kecepatan lama) + 0.0124 ≈ 1.0124
v_y_baru = 0 + 0.0031 ≈ 0.0031

3. **Posisi Baru**:

pos_x_baru = 200 + 1.0124 ≈ 201.012
pos_y_baru = 150 + 0.0031 ≈ 150.003

Langkah 3: Hitung Getaran Antena Robot

- Kecepatan robot = √(1.0124² + 0.0031²) ≈ 1.012
- Getaran antena (sinus):

theta_antena += 1.012 / 10 ≈ 0.1012 radian
x_antena = (cos(0.1012) + 1) / 2 × (massa × 2)
≈ (0.995 + 1) / 2 × 20
≈ 19.95 pixel

Hasil Akhir dalam 1 Frame:

Parameter	Nilai Awal	Nilai Baru
Posisi Robot (x,y)	(200, 150)	(201.01, 150.00)
Kecepatan Robot	(1, 0)	(1.012, 0.003)
Posisi Antena	-	19.95 pixel dari robot

Visualisasi Gerakan:

Frame 0: Robot di (200,150) → Frame 1: Robot di (201.01,150.00)
△ △
| (antena panjang 20) | (antena panjang 19.95)

- Robot bergerak **sedikit** mendekati magnet karena jarak masih jauh.
- Antena bergetar **hampir tidak terlihat** karena perubahan kecil.

Apa yang Terjadi Jika Robot Lebih Dekat?

Misal jarak = 15 pixel:

- Gaya = (0.4×20×10)/(15²) ≈ 0.356 (lebih besar!)
- Robot akan bergerak lebih cepat ke magnet.

Program ini menghitung ini untuk **semua robot** setiap 16ms (60fps)! 🔄

```
import tkinter as tk
import math
import random
from vector import Vector

class Oscillator:
    def __init__(self, canvas, amplitude):
        self.canvas = canvas
        self.theta = 0
        self.amplitude = amplitude
        self.line = canvas.create_line(0, 0, 0, 0, fill='black')
        self.dot = canvas.create_oval(0, 0, 0, 0, fill='black')

    def update(self, vel_mag):
        self.theta += vel_mag

    def display(self, pos):
        x = math.cos(self.theta) * self.amplitude
        self.canvas.coords(self.line, pos.x, pos.y, pos.x + x, pos.y)
        r = 4
        self.canvas.coords(self.dot, pos.x + x - r, pos.y - r, pos.x + x + r, pos.y + r)

class Crawler:
    def __init__(self, canvas, width, height):
        self.canvas = canvas
        self.pos = Vector(random.uniform(0, width), random.uniform(0, height))
        self.vel = Vector(random.uniform(-1, 1), random.uniform(-1, 1))
        self.acc = Vector()
        self.mass = random.uniform(8, 16)
        self.radius = self.mass
        self.id = canvas.create_oval(0, 0, 0, 0, fill='gray')
        self.osc = Oscillator(canvas, self.mass * 2)

    def apply_force(self, force):
        f = force.dived(self.mass)
        self.acc = self.acc.added(f)

    def update(self):
        self.vel = self.vel.added(self.acc)
        self.pos = self.pos.added(self.vel)
        self.acc = Vector()
        self.osc.update(self.vel.mag() / 10)

    def display(self):
        r = self.radius
        self.canvas.coords(self.id, self.pos.x - r, self.pos.y - r, self.pos.x + r,
self.pos.y + r)
        self.osc.display(self.pos)

class Attractor:
    def __init__(self, canvas, pos, mass, g):
        self.canvas = canvas
        self.pos = pos
        self.mass = mass
        self.G = g
        self.radius = mass
        self.dragging = False
        self.rollover = False
        self.drag_offset = Vector()
        self.id = canvas.create_oval(0, 0, 0, 0, fill='lightblue')

    def attract(self, crawler):
        force = self.pos.subbed(crawler.pos)
        d = max(5.0, min(25.0, force.mag()))
        force = force.normalized()
        strength = (self.G * self.mass * crawler.mass) / (d * d)
        return force.multed(strength)

    def render(self):
        fill = 'gray' if self.dragging else 'lightgreen' if self.rollover else 'lightblue'
```

```
        self.canvas.itemconfig(self.id, fill=fill)
        r = self.radius
        self.canvas.coords(self.id, self.pos.x - r, self.pos.y - r, self.pos.x + r,
self.pos.y + r)

    def clicked(self, mx, my):
        d = math.hypot(mx - self.pos.x, my - self.pos.y)
        if d < self.radius:
            self.dragging = True
            self.drag_offset = Vector(self.pos.x - mx, self.pos.y - my)

    def stop_dragging(self):
        self.dragging = False

    def drag(self, mx, my):
        if self.dragging:
            self.pos.x = mx + self.drag_offset.x
            self.pos.y = my + self.drag_offset.y

    def rollover_check(self, mx, my):
        d = math.hypot(mx - self.pos.x, my - self.pos.y)
        self.rollover = d < self.radius

    def go(self):
        self.render()

# --- Main Application ---
def main():
    root = tk.Tk()
    root.title("Attraction Array with Oscillation")
    width, height = 640, 360
    canvas = tk.Canvas(root, width=width, height=height, bg='white')
    canvas.pack()

    attractor = Attractor(canvas, Vector(width / 2, height / 2), 20, 0.4)
    crawlers = [Crawler(canvas, width, height) for _ in range(6)]

    mouse = {'x': 0, 'y': 0}

    def draw():
        attractor.rollover_check(mouse['x'], mouse['y'])
        attractor.drag(mouse['x'], mouse['y'])
        attractor.go()

        for c in crawlers:
            f = attractor.attract(c)
            c.apply_force(f)
            c.update()
            c.display()

        root.after(16, draw)

    def on_mouse_move(event):
        mouse['x'], mouse['y'] = event.x, event.y

    def on_mouse_down(event):
        attractor.clicked(event.x, event.y)

    def on_mouse_up(event):
        attractor.stop_dragging()

    canvas.bind("<Motion>", on_mouse_move)
    canvas.bind("<ButtonPress-1>", on_mouse_down)
    canvas.bind("<ButtonRelease-1>", on_mouse_up)

    draw()
    root.mainloop()

if __name__ == "__main__":
    main()
```