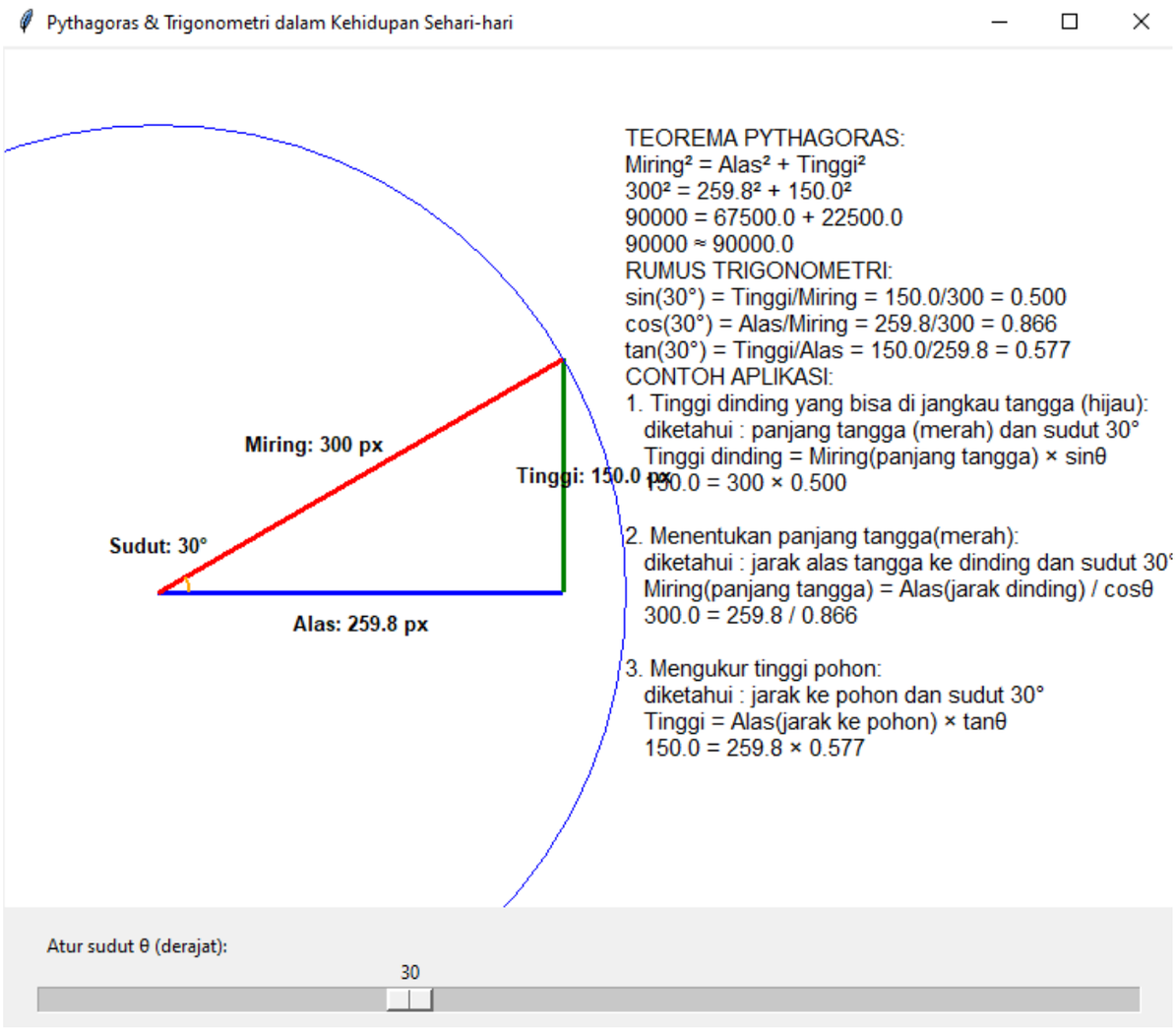
The Nature of Code

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



Berikut penjelasan **materi 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:

$\text{Miring}^2 = \text{Alas}^2 + \text{Tinggi}^2$

🧱 Contoh perhitungan:

Misalnya, panjang alas = 3, tinggi = 4:

$\text{Miring}^2 = 3^2 + 4^2$

$\text{Miring}^2 = 9 + 16 = 25$

Miring = $\sqrt{25}$ = **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

```
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()
```

Chapter 3. Oscillation

3.1 Angles

🔍 **1. Penjelasan Konsep**

Apa itu Sudut (Angle)?

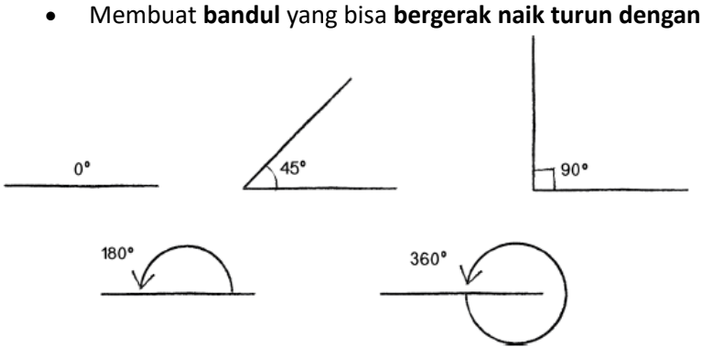
- **Sudut** adalah seberapa besar sesuatu **berputar atau berbelok**.
- Satuan sudut:
  - **Derajat**: seperti di sekolah (0°–360°)
  - **Radian**: satuan lain yang digunakan oleh komputer dan matematika tingkat lanjut.

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

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")

# Radian ke derajat
rad2 = math.pi / 3
deg2 = math.degrees(rad2)
print(f"{rad2:.2f} rad = {deg2:.1f}°")
Hasil:
60° = 1.05 rad
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:

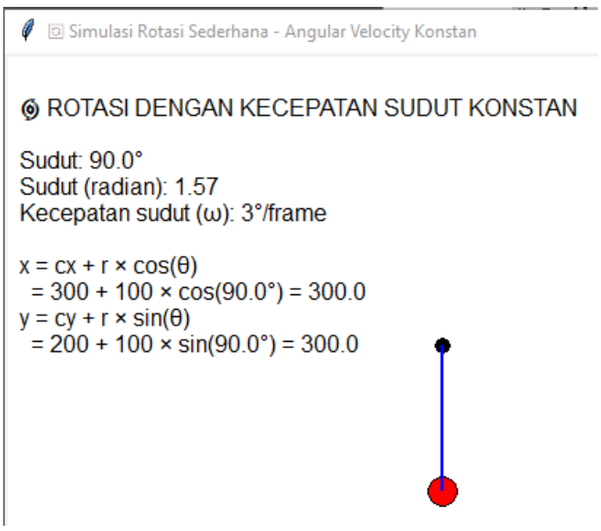
Frame 0:

- 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)

```
# 1. Simulasi Rotasi Sederhana - Angular Velocity Konstan
import tkinter as tk
import math

# Setup jendela
root = tk.Tk()
root.title("🔄 Simulasi Rotasi Sederhana - Angular Velocity Konstan")

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

# Pusat rotasi
cx, cy = 300, 200
radius = 100
angle_deg = 0
angular_velocity = 3  # derajat per frame

# Objek visual
circle = canvas.create_oval(cx - 5, cy - 5, cx + 5, cy + 5, fill="black")  # Titik pusat
dot = canvas.create_oval(0, 0, 0, 0, fill="red")  # Titik yang berputar
line = canvas.create_line(0, 0, 0, 0, fill="blue", width=2)  # Garis dari pusat ke titik
text = canvas.create_text(10, 10, anchor="nw", text="", font=("Arial", 12), fill="black")

def update():
    global angle_deg
```

```
    # Hitung posisi titik berdasarkan sudut
    angle_rad = math.radians(angle_deg)
    x = cx + radius * math.cos(angle_rad)
    y = cy + radius * math.sin(angle_rad)

    # Gambar ulang titik & garis
    canvas.coords(dot, x - 10, y - 10, x + 10, y + 10)
    canvas.coords(line, cx, cy, x, y)

    # Tampilkan informasi
    info = f"""
🧿 ROTASI DENGAN KECEPATAN SUDUT KONSTAN

Sudut: {angle_deg:.1f}°
Sudut (radian): {angle_rad:.2f}
Kecepatan sudut (ω): {angular_velocity}°/frame

x = cx + r × cos(θ)
  = {cx} + {radius} × cos({angle_deg:.1f}°) = {x:.1f}
y = cy + r × sin(θ)
  = {cy} + {radius} × sin({angle_deg:.1f}°) = {y:.1f}
"""
    canvas.itemconfig(text, text=info)

    # Tambah sudut
    angle_deg = (angle_deg + angular_velocity) % 360

    root.after(50, update)

update()
root.mainloop()
```

🔢 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)
y1 = y + L * sin(angle)
x2 = x - L * cos(angle)
y2 = y - L * sin(angle)

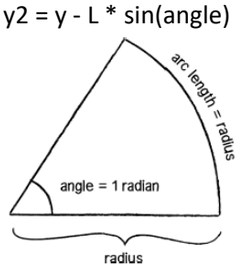
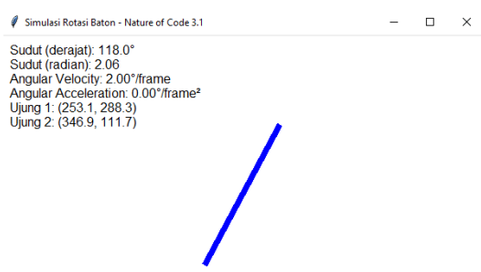


Figure 3.3

The formula to convert from degrees to radians is:

radians = 2 * PI * (degrees / 360)



📝 5. Penjelasan Fungsi dengan Komentar

```
# Membuat jendela utama Tkinter
root = tk.Tk()

# Membuat kanvas untuk menggambar
canvas = tk.Canvas(root, width=600, height=400)
```

```
# Fungsi draw_baton: menggambar tongkat berputar
# - Konversi derajat ke radian
# - Hitung ujung tongkat berdasarkan trigonometri
# - Gambar garis (tongkat)
# - Tampilkan informasi perhitungan ke layar

# Fungsi animate:
# - Menambahkan sudut rotasi
# - Memanggil draw_baton lagi
# - Loop setiap 50 milidetik
```

📚 **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 (Processing atau Python):**

- **Processing (Java-based):**
  Kamu bisa langsung pakai:
- float angle = PI / 2;
- **Python:**
  Gunakan:
- 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)
...	...	...	...
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="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))

        # 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): {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: ({end1.x:.1f}, {end1.y:.1f})
Ujung 2: ({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(50, 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
```

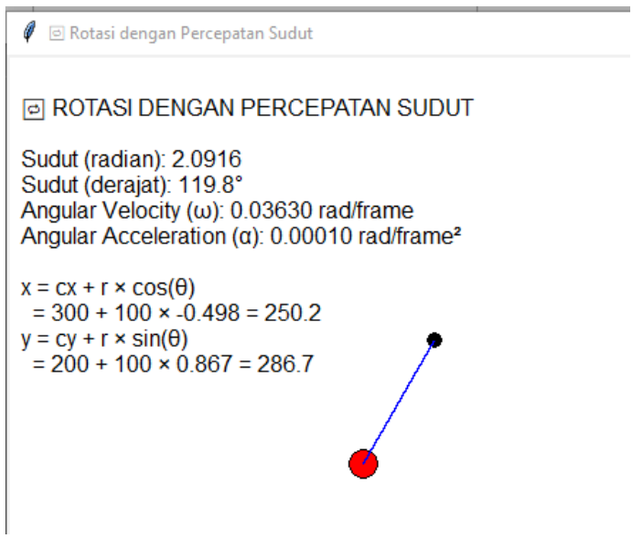
3. **Posisi Objek**:

```
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

```
# 2. ROTASI DENGAN PERCEPATAN SUDUT
import tkinter as tk
import math

# Vector class sederhana dengan metode rotate
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def rotated(self, angle_rad):
        cos_a = math.cos(angle_rad)
        sin_a = math.sin(angle_rad)
        return Vector(
            self.x * cos_a - self.y * sin_a,
            self.x * sin_a + self.y * cos_a
        )

# Setup Tkinter
root = tk.Tk()
root.title("🔁 Rotasi dengan Percepatan Sudut")

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

# Variabel rotasi
center = Vector(300, 200)
radius = 100
angle_rad = 0
angular_velocity = 0.03  # rad/frame
angular_acceleration = 0.0001  # rad/frame²
```

```
# Gambar dasar
circle = canvas.create_oval(center.x - 5, center.y - 5, center.x + 5, center.y + 5, fill="black")
dot = canvas.create_oval(0, 0, 0, 0, fill="red")
line = canvas.create_line(0, 0, 0, 0, fill="blue", width=2)
text = canvas.create_text(10, 10, anchor="nw", text="", font=("Arial", 12), fill="black")

def update():
    global angle_rad, angular_velocity

    # Arah awal (1, 0) diputar sesuai sudut
    direction = Vector(1, 0).rotated(angle_rad)
    x = center.x + radius * direction.x
    y = center.y + radius * direction.y

    # Gambar ulang
    canvas.coords(dot, x - 10, y - 10, x + 10, y + 10)
    canvas.coords(line, center.x, center.y, x, y)

    # Tampilkan info
    info = f"""
🔁 ROTASI DENGAN PERCEPATAN SUDUT

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

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

    # Update sudut & kecepatan sudut
    angular_velocity += angular_acceleration
    angle_rad = (angle_rad + angular_velocity) % (2 * math.pi)

    root.after(20, update)

update()
root.mainloop()
```

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:

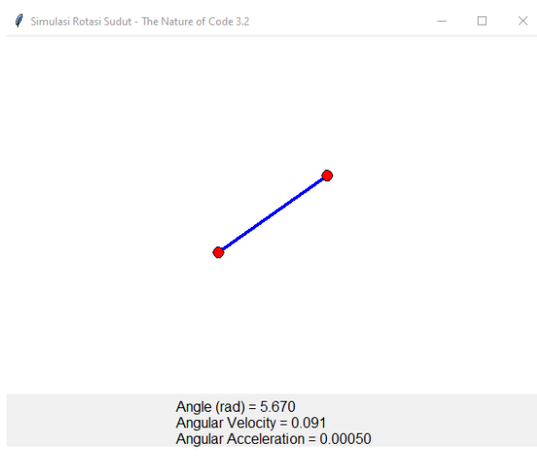
- ω = 0.05 + 0.001 = 0.051
- θ = 0 + 0.051 = 0.051

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.

Penjelasan tentang Akselerasi Sudut dalam Kode
Teks tersebut membahas mengapa sebuah objek tidak berotasi dalam simulasi dan cara mengimplementasikan akselerasi sudut. Berikut penjelasannya:
Alasan Awal Tidak Ada Rotasi

- Kode menginisialisasi akselerasi sudut ke nol (float aAcceleration = 0;)
- Dengan akselerasi sudut nol, tidak ada perubahan rotasi dari waktu ke waktu
- Objek tetap diam dalam hal rotasi

Pendekatan untuk Menambahkan Rotasi

1. **Solusi Hardcoded**:
    - Cukup beri nilai bukan nol pada akselerasi sudut
    - Ini akan membuat objek berotasi, tetapi tidak dinamis atau menarik
2. **Solusi Berbasis Fisika (Ideal tapi Kompleks)**:
    - Memodelkan torsi (torque) dan momen inersia dengan benar
    - Ini akan akurat secara fisika tetapi membutuhkan pengetahuan fisika lanjut
    - Di luar cakupan materi saat ini
3. **Solusi Cepat dan Praktis**:
    - Hubungkan akselerasi sudut dengan akselerasi linear objek
    - Contoh: aAcceleration = acceleration.x
    - Membuat hubungan antara gerakan horizontal dan rotasi
    - Akselerasi ke kanan → rotasi searah jarum jam
    - Akselerasi ke kiri → rotasi berlawanan jarum jam

Pertimbangan Penting

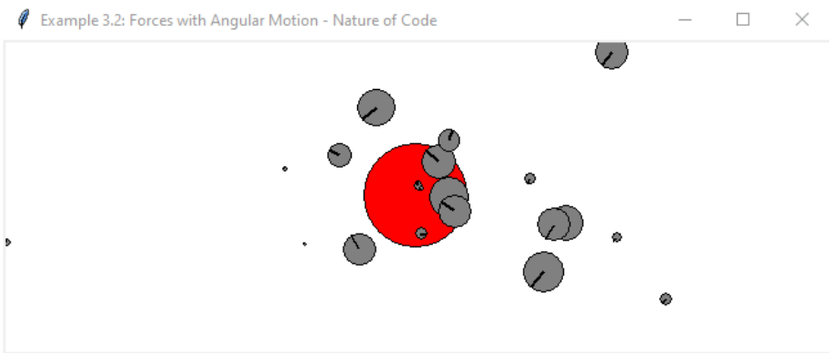
1. **Pentingnya Penskalaan (*Scaling*)**
    - Nilai percepatan linear (*x component of acceleration vector*) mungkin terlalu besar, menyebabkan rotasi terlihat tidak wajar (misalnya, objek berputar terlalu cepat).
    - Solusi:
        - **Membagi komponen *x* percepatan** dengan suatu nilai (*scaling factor*) untuk mengurangi dampaknya pada rotasi.
        - **Membatasi rentang kecepatan sudut** (*angular velocity*) agar tetap dalam batas realistis (misalnya menggunakan clamp()).
2. **Implementasi dalam Fungsi** update()
    - Fungsi ini kemungkinan memperbarui posisi dan rotasi objek setiap frame. Contoh pseudocode:

```
def update():
  # Hitung percepatan linear (contoh: dari input atau fisika)
  acceleration_x = get_acceleration_x()

  # Konversi percepatan linear ke percepatan sudut (dengan scaling)
  angular_acceleration = acceleration_x / SCALING_FACTOR

  # Update kecepatan sudut
  angular_velocity += angular_acceleration * delta_time

  # Batasi kecepatan sudut agar tidak terlalu besar
  angular_velocity = clamp(angular_velocity, -MAX_SPEED, MAX_SPEED)

  # Update rotasi objek
  rotation += angular_velocity * delta_time
```

Contoh Kasus Visual:
Bayangkan mobil dalam game:

- Saat mobil dipercepat ke kanan, bodinya sedikit miring (rotasi clockwise) seolah tertarik oleh gaya inersia.
- Tanpa scaling, percepatan tinggi bisa membuat mobil terlihat seperti "berputar liar".

Dengan penyesuaian ini, rotasi terlihat lebih alami dan terkontrol.



Gaya gravitasi antara bola merah besar (*Attractor*) dan bola-bola kecil (*Mover*).
Berikut tahapan rumusnya:

---

1. Menghitung Gaya Tarik (Attract)

$$F = G \cdot \frac{m_1 \cdot m_2}{r^2}$$

- F = Gaya tarik
- G = Konstanta gravitasi (di program = 1)
- m_1 = Massa Attractor (bola merah) = 20
- m_2 = Massa Mover (bola abu-abu) = random 0.1–2
- r = Jarak antara kedua bola

Rumus Gaya Gravitasi:
Contoh Perhitungan:
Misal:

- Bola merah di posisi **(320, 120)**
- Bola abu-abu di posisi **(300, 100)**
- Massa bola abu-abu = **1**

Langkah 1: Hitung Jarak (`r`)

$$\text{jarak} = \sqrt{(320-300)^2 + (120-100)^2} = \sqrt{400+400} = \sqrt{800} \approx 28.28$$

*Program membatasi jarak min 5, max 25, jadi dipakai **r = 25**.*

Langkah 2: Hitung Kekuatan Gaya (`strength`)

$$F = 1 \cdot \frac{20 \cdot 1}{25^2} = \frac{20}{625} = 0.032$$

Langkah 3: Tentukan Arah Gaya

- Vektor arah = posisi Attractor – posisi Mover = (320–300, 120–100) = **(20, 20)**
- Normalisasi (ubah jadi vektor panjang 1):

$$\text{vektor normal} = \left(\frac{20}{28.28}, \frac{20}{28.28}\right) \approx (0.707, 0.707)$$

- Kalikan dengan gaya:

$$\text{gaya akhir} = (0.707 \cdot 0.032, 0.707 \cdot 0.032) \approx (0.0226, 0.0226)$$

Hasil: Bola abu-abu ditarik dengan gaya **(0.0226, 0.0226)**.

---

2. Gerakan Bola Kecil (Update)

Hukum Newton:

$$F = m \cdot a \quad \Rightarrow \quad a = \frac{F}{m}$$

- `acceleration = force / mass`
- `velocity += acceleration`
- `position += velocity`

Contoh:

Jika bola abu-abu massa = 1 dan gaya = (0.0226, 0.0226):

- **Percepatan (`a`)** = (0.0226/1, 0.0226/1) = **(0.0226, 0.0226)**
- **Kecepatan (`v`)** = kecepatan lama + percepatan
- **Posisi baru** = posisi lama + kecepatan

Rotasi Garis Penunjuk:

- Garis di bola abu-abu berputar sesuai percepatan horizontal (`acceleration.x`).
- Rumus:

$$\text{sudut baru} = \text{sudut lama} + (\text{acceleration.x}/10)$$

---

3. Batasan Agar Tidak "Liar" (Constrain)

- Kecepatan sudut dibatasi antara **-0.1 sampai 0.1** agar rotasi tidak terlalu cepat.
- Jarak dibatasi **5–25** agar gaya tidak terlalu besar/kecil.

---

4. Visualisasi

- **Bola Merah**: Gambar lingkaran dengan radius = mass × 2 (40 piksel).
- **Bola Abu-Abu**:
  - Radius = mass × 8 (jika massa = 1, radius = 8 piksel).
  - Garis penunjuk = garis dari tengah ke arah sudut saat ini.

Analoginya:

- Bayangkan bola merah adalah **Matahari**, dan bola abu-abu adalah **planet-planet**.
- Semakin dekat ke Matahari, gaya gravitasi semakin kuat (tapi di program dibatasi).
- Garis di bola abu-abu seperti "penunjuk arah" yang berputar saat planet berakselerasi.

Dengan rumus-rumus ini, program bisa mensimulasikan gerakan alami seperti orbit! 🚀

Misal:

- Attractor Merah di (320,120) dengan massa 20
- Mover abu-abu di (300,100) dengan massa 1

1. Hitung jarak:
   - dx = 320-300 = 20
   - dy = 120-100 = 20
   - distance = √(20² + 20²) ≈ 28.28
   - Dibatasi jadi 25 (karena maksimum 25)
2. Hitung kekuatan:
   - strength = (1 × 20 × 1) / (25 × 25) = 20/625 = 0.032
3. Hitung arah:
   - Vektor (20,20) dinormalisasi jadi ≈ (0.707, 0.707)
   - Dikalikan strength jadi (0.707×0.032, 0.707×0.032) ≈ (0.0226, 0.0226)

Jadi mover akan mendapat dorongan kecil ke arah (0.0226, 0.0226)
Program ini menunjukkan bagaimana gaya gravitasi bekerja di ruang angkasa dalam bentuk sederhana!

```
# SCRIPT 3 : # Simulasi Gaya Tarik dengan Gerakan Angular
import tkinter as tk
import math
import random
from vector import Vector

#1. Fungsi ini memastikan nilai tetap dalam batas tertentu. Contoh:
#constrain(10, 0, 5) akan mengembalikan 5
#constrain(-2, 0, 5) akan mengembalikan 0
```

```
def constrain(val, min_val, max_val):
    return max(min_val, min(val, max_val))
#2. Bola Merah (Attractor)
class Attractor:
    def __init__(self, canvas, x=320, y=120):
        self.canvas = canvas
        self.position = Vector(x, y)  # Posisi tengah layar
        self.mass = 20                # Massa besar
        self.G = 1                    # Kekuatan gravitasi
        r = self.mass * 2             # Ukuran bola
        # Gambar lingkaran merah
        self.id = canvas.create_oval(x-r, y-r, x+r, y+r, fill="red")
    # Attractor mengaplikasikan gaya tarik pada Mover
    def attract(self, mover):
        # Hitung jarak antara Attractor dan Mover
        force = Vector(self.position.x - mover.position.x,
                       self.position.y - mover.position.y)
        distance = force.mag()  # Panjang garis antara keduanya

        # Batasi jarak minimal 5 dan maksimal 25
        distance = constrain(distance, 5, 25)

        force.normalize()  # Buat vektor panjangnya menjadi 1

        # Hitung kekuatan tarik (seperti rumus gravitasi)
        strength = (self.G * self.mass * mover.mass) / (distance * distance)
        force.mult(strength)  # Terapkan kekuatan ke vektor

        return force

    def display(self):
        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)
#3. Bola Kecil (Mover)
class Mover:
    def __init__(self, canvas, x, y, mass):
        self.canvas = canvas
        self.mass = mass
        self.radius = self.mass * 8  # Semakin besar massa, semakin besar ukuran
        self.position = Vector(x, y) # Posisi acak
        # Kecepatan awal acak antara -1 sampai 1
        self.velocity = Vector(random.uniform(-1, 1), random.uniform(-1, 1))
        self.acceleration = Vector(0, 0)

        self.angle = 0
        self.angle_velocity = 0
        self.angle_acceleration = 0

        # Gambar bola abu-abu dengan garis penunjuk arah
        r = self.radius
        self.id_circle = canvas.create_oval(x-r, y-r, x+r, y+r, fill="gray")
        self.id_line = canvas.create_line(x, y, x+r, y, width=2, fill="black")

    def apply_force(self, force):
        f = force.copy()
        f.div(self.mass)
        self.acceleration.add(f)
    #Pergerakan:
    def update(self):
        # Update posisi berdasarkan kecepatan
        self.velocity.add(self.acceleration)
        self.position.add(self.velocity)

        # Putar garis penunjuk berdasarkan percepatan
        self.angle_acceleration = self.acceleration.x / 10.0
        self.angle_velocity += self.angle_acceleration
        self.angle_velocity = constrain(self.angle_velocity, -0.1, 0.1)
        self.angle += self.angle_velocity

        self.acceleration.mult(0)  # Reset percepatan
    #Gambar Ulang Posisi:
    def show(self):
        x, y, r = self.position.x, self.position.y, self.radius
        # Update posisi lingkaran
        self.canvas.coords(self.id_circle, x-r, y-r, x+r, y+r)
```

```
        # Hitung ujung garis penunjuk
        end_x = x + r * math.cos(self.angle)
        end_y = y + r * math.sin(self.angle)
        # Update posisi garis
        self.canvas.coords(self.id_line, x, y, end_x, end_y)

# ========================
# Main Program
# ========================
root = tk.Tk()
root.title("Example 3.2: Forces with Angular Motion - Nature of Code")

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

attractor = Attractor(canvas)

# Buat 20 bola kecil dengan posisi dan massa acak
movers = []
for _ in range(20):
    x = random.uniform(0, 640)  # Posisi X acak
    y = random.uniform(0, 240)  # Posisi Y acak
    mass = random.uniform(0.1, 2)  # Massa acak
    movers.append(Mover(canvas, x, y, mass))

def animate():
    attractor.display()
    for mover in movers:
        force = attractor.attract(mover)  # Hitung gaya tarik
        mover.apply_force(force)         # Terapkan gaya
        mover.update()                   # Update posisi
        mover.show()                     # Gambar ulang
    root.after(30, animate)             # Ulangi setiap 30ms

animate()
root.mainloop()
```

Simulasi Peluru Meriam dengan Rotasi

Program ini mensimulasikan **peluru yang ditembakkan dari meriam** dengan:

- **Kontrol sudut meriam** (kiri/kanan)
- **Gaya gravitasi** yang menarik peluru ke bawah
- **Fisika gerak parabola** seperti di kehidupan nyata

Berikut tahapan rumus dan cara kerjanya:

---

1. Komponen Utama

a. Peluru (CannonBall**)**

- **Posisi (x,y)**: Lokasi peluru di layar
- **Kecepatan (velocity)**: Arah dan cepatnya gerak
- **Percepatan (acceleration)**: Perubahan kecepatan karena gaya (misal gravitasi)
- **Radius (r)**: Ukuran peluru (8 piksel)

b. Meriam (Cannon**)**

- **Posisi (x,y)**: Tetap di (50, 300)
- **Panjang (length)**: 50 piksel
- **Sudut (angle)**: Awalnya -45° (mengarah atas-kanan)
- **Bisa diputar** dengan tombol **kiri/kanan**

---

2. Rumus Penting

a. Gerakan Peluru (update()**)**

Menggunakan **Hukum Newton**:

1. **Percepatan** mempengaruhi **kecepatan**:

velocity += acceleration

2. **Kecepatan** mempengaruhi **posisi**:

position += velocity

3. **Gravitasi** ditambahkan setiap frame:

gravity = Vector(0, 0.2) *# Tarik ke bawah (sumbu Y)*
ball.apply_force(gravity)

b. Menembakkan Peluru (shoot_force()**)**

- Gaya ditentukan oleh **sudut meriam (**angle**)** dan **kekuatan tembakan (**magnitude**)**.
- **Rumus vektor gaya**:

force_x = cos(angle) * magnitude
force_y = sin(angle) * magnitude
Contoh:

- Jika sudut = **-45°** (π/4 radian):

cos(-45°) ≈ 0.707
sin(-45°) ≈ -0.707

- Maka gaya tembak:

force_x = 0.707 * 10 ≈ 7.07 (ke kanan)
force_y = -0.707 * 10 ≈ -7.07 (ke atas)

c. Rotasi Meriam (rotate()**)**

- Sudut diubah dengan **delta_angle** (misal +0.1 atau -0.1 radian).
- **Dibatasi** antara **-90° sampai 0°** (tidak bisa melewati batas ini).

---

3. Contoh Perhitungan Gerak Peluru
Misal:

- **Sudut meriam = -45°**
- **Gaya tembak = 10**
- **Gravitasi = 0.2** (ke bawah)

Langkah 1: Hitung Gaya Awal

- force_x = cos(-45°) * 10 ≈ 7.07
- force_y = sin(-45°) * 10 ≈ -7.07

Langkah 2: Update Kecepatan & Posisi
Setiap frame (30ms), terjadi:

1. **Kecepatan awal** = (7.07, -7.07)
2. **Gravitasi** menambah acceleration.y = +0.2
3. **Kecepatan baru**:

velocity.x = 7.07 (tetap, karena gravitasi hanya mempengaruhi Y)
velocity.y = -7.07 + 0.2 = -6.87

4. **Posisi baru**:

position.x += 7.07
position.y += -6.87

Hasil:

- Peluru bergerak **parabola**:
  - Awalnya cepat ke **atas-kanan** (karena gaya tembak).
  - Semakin lama semakin **turun** (karena gravitasi).

---

4. Kontrol Program

- ⬅️ **(Panah Kiri)**: Putar meriam ke kiri.
- ➡️ **(Panah Kanan)**: Putar meriam ke kanan.
- ␣ **(Spasi)**: Tembakkan peluru.
- **Peluru otomatis di-reset** jika jatuh ke bawah layar.

---

5. Analogi Sederhana

- **Meriam** = Seseorang melempar bola.
- **Sudut meriam** = Sudut lemparan (misal 45°).
- **Gaya tembak** = Kekuatan lemparan.
- **Gravitasi** = Gaya tarik bumi yang membuat bola jatuh.

Dengan rumus ini, program bisa mensimulasikan **gerak peluru** seperti di dunia nyata! 🚀💥

```
#SCRIPT 3 : # Simulasi Peluru Meriam dengan Rotasi
import tkinter as tk
import math
from vector import Vector

class CannonBall:
    def __init__(self, canvas, x, y):
        self.canvas = canvas
        self.position = Vector(x, y)
        self.velocity = Vector(0, 0)
        self.acceleration = Vector(0, 0)
        self.r = 8
        self.topspeed = 10
        self.id = canvas.create_oval(x - self.r, y - self.r, x + self.r, y + self.r,
fill="black")

    def apply_force(self, force):
        self.acceleration.add(force)
```

```
    def update(self):
        self.velocity.add(self.acceleration)
        self.velocity.limit(self.topspeed)
        self.position.add(self.velocity)
        self.acceleration.mult(0)

    def display(self):
        x, y, r = self.position.x, self.position.y, self.r
        self.canvas.coords(self.id, x - r, y - r, x + r, y + r)

class Cannon:
    def __init__(self, canvas, x, y):
        self.canvas = canvas
        self.position = Vector(x, y)
        self.angle = -math.pi / 4  # -45 derajat
        self.length = 50
        self.id = None

    def draw(self):
        x, y = self.position.x, self.position.y
        end_x = x + self.length * math.cos(self.angle)
        end_y = y + self.length * math.sin(self.angle)
        if self.id is None:
            self.id = self.canvas.create_line(x, y, end_x, end_y, width=10, fill="gray")
        else:
            self.canvas.coords(self.id, x, y, end_x, end_y)

    def rotate(self, delta_angle):
        self.angle += delta_angle
        self.angle = max(-math.pi / 2, min(self.angle, 0))  # constrain sudut

    def shoot_force(self, magnitude=10):
        force = Vector(math.cos(self.angle), math.sin(self.angle))
        force.mult(magnitude)
        return force

# Setup Tkinter
root = tk.Tk()
root.title("Exercise 3.2 Cannon - Nature of Code")

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

cannon = Cannon(canvas, 50, 300)
ball = CannonBall(canvas, cannon.position.x, cannon.position.y)
shot = False

def animate():
    global ball, shot
    canvas.delete("background_text")

    cannon.draw()

    if shot:
        gravity = Vector(0, 0.2)
        ball.apply_force(gravity)
        ball.update()

    ball.display()

    # Reset jika jatuh
    if ball.position.y > HEIGHT:
        ball = CannonBall(canvas, cannon.position.x, cannon.position.y)
        shot = False

    # Tampilkan informasi di layar
    angle_deg = math.degrees(cannon.angle)
    canvas.create_text(10, 10, anchor="nw", text=f"Angle: {angle_deg:.1f}°",
tag="background_text", font=("Arial", 14))

    pos = ball.position
    vel = ball.velocity
    acc = ball.acceleration
    canvas.create_text(10, 30, anchor="nw", text=f"Pos: ({pos.x:.1f}, {pos.y:.1f})",
tag="background_text", font=("Arial", 12))
```

```
    canvas.create_text(10, 50, anchor="nw", text=f"Vel: ({vel.x:.2f}, {vel.y:.2f})",
tag="background_text", font=("Arial", 12))
    canvas.create_text(10, 70, anchor="nw", text=f"Acc: ({acc.x:.2f}, {acc.y:.2f})",
tag="background_text", font=("Arial", 12))

    root.after(30, animate)

def on_key_press(event):
    global shot
    if event.keysym == 'Left':
        cannon.rotate(-0.1)
    elif event.keysym == 'Right':
        cannon.rotate(0.1)
    elif event.keysym == 'space' and not shot:
        shot = True
        force = cannon.shoot_force(10)
        ball.apply_force(force)

root.bind("<KeyPress>", on_key_press)

animate()
root.mainloop()
```

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)

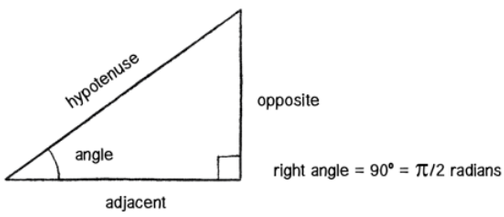
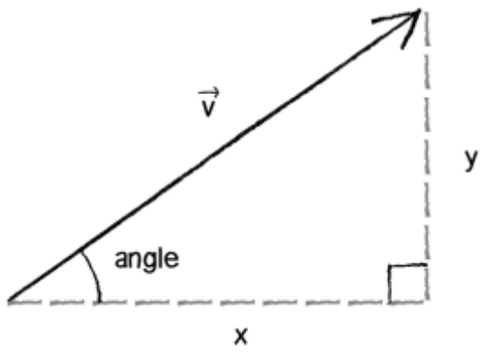


Figure 3.4

- ***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.

https://i.imgur.com/5Xc5FjN.png

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: $\sqrt{(x^2 + y^2)}$.

Contoh Soal Sederhana

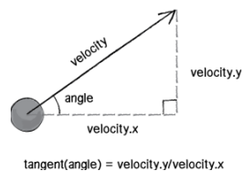
Diketahui:

- 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.

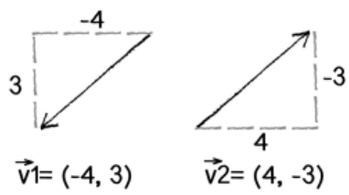
Langkah 2: Menghitung Sudut Rotasi

Kita butuh sudut (angle) untuk memutar benda. Gunakan **tangent** dari segitiga yang dibentuk oleh velocity:
https://i.imgur.com/5Xc5FjN.png

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**:

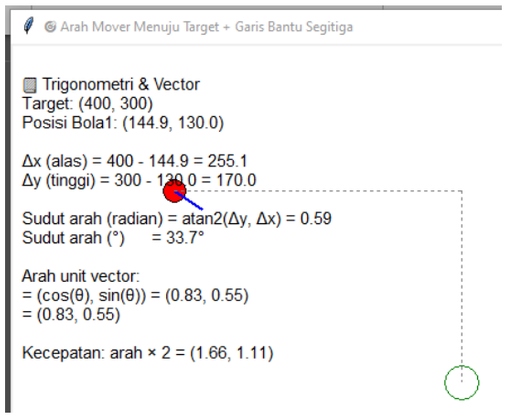
velocity = direction_unit × speed

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)

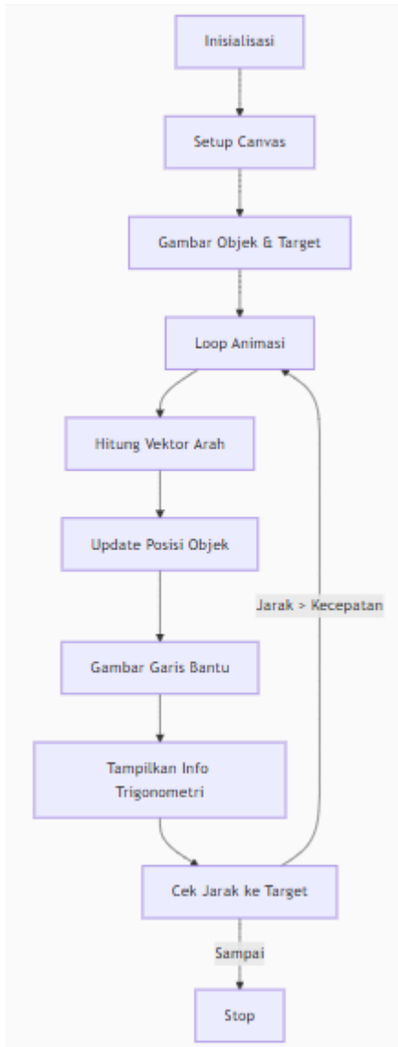
🔍 Visualisasi Komponen

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

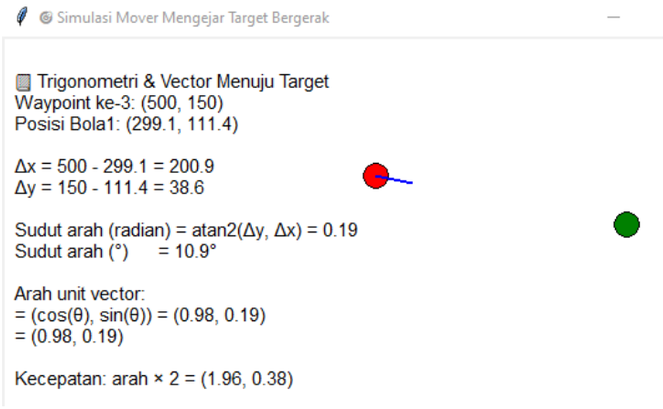
📝 Rumus dan Konsep Kunci

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)):

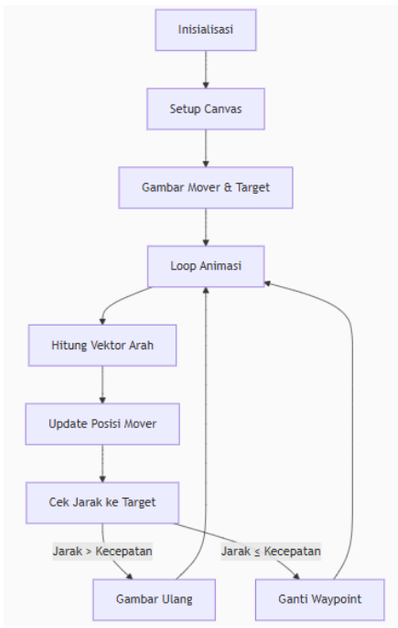
- Posisi mover: (100,100)
- Vektor arah: (250-100, 300-100) = (150,200)
- Magnitude: √(150²+200²) = 250
- Vektor satuan: (150/250, 200/250) = (0.6, 0.8)
- Kecepatan: (0.6×2, 0.8×2) = (1.2, 1.6)
- Posisi baru: (100+1.2, 100+1.6) = (101.2,101.6)

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()
```

Membuat sebuah animasi di mana sebuah mobil kotak bergerak mengikuti arah mouse

---

1. Konsep Dasar

- **Kotak (Mover)** akan bergerak menuju posisi mouse.
- Gerakan menggunakan **fisika sederhana**:
    - **Percepatan → Kecepatan → Posisi**
- Kotak akan **berputar** sesuai arah geraknya.
- Jika keluar layar, akan muncul di sisi sebaliknya (seperti game Pac-Man).

---

2. Rumus dan Cara Kerja

a. Menghitung Arah ke Mouse

```
mouse = Vector(mouse_x, mouse_y)          # Posisi mouse
```

```
dir = mouse.subbed(self.position)         # Vektor arah (mouse - posisi kotak)
dir.normalize()                           # Buat panjang vektor = 1
dir.mult(0.5)                             # Perkecil pengaruhnya
self.acceleration = dir                   # Set percepatan
```

- **Contoh**:
  - Posisi kotak = (100, 100)
  - Posisi mouse = (150, 120)
  - Vektor arah = (150-100, 120-100) = **(50, 20)**
  - Dinormalisasi (diubah jadi panjang 1):
    - Panjang vektor = $\sqrt{(50^2 + 20^2)} \approx$ **53.85**
    - Vektor normal = (50/53.85, 20/53.85) ≈ **(0.93, 0.37)**
  - Dikalikan 0.5: **(0.465, 0.185)** → Ini jadi **percepatan**

b. Update Gerakan

```
self.velocity.add(self.acceleration)   # Kecepatan += Percepatan
self.velocity.limit(self.topspeed)     # Batasi kecepatan maks (4)
self.position.add(self.velocity)       # Posisi += Kecepatan
```

- **Contoh**:
  - Jika kecepatan awal = (1, 0.5)
  - Percepatan = (0.465, 0.185)
  - Kecepatan baru = (1 + 0.465, 0.5 + 0.185) = **(1.465, 0.685)**
  - Posisi baru = (100 + 1.465, 100 + 0.685) ≈ **(101.47, 100.69)**

c. Rotasi Kotak

Kotak berputar sesuai arah kecepatan (velocity.heading()).

```
angle = self.velocity.heading()        # Sudut dalam radian
cos_a = math.cos(angle)               # Hitung komponen X
sin_a = math.sin(angle)               # Hitung komponen Y

# Hitung 4 sudut kotak setelah rotasi:
for dx, dy in [(-w/2, -h/2), (w/2, -h/2), (w/2, h/2), (-w/2, h/2)]:
    x = cx + dx * cos_a - dy * sin_a  # Rotasi sumbu X
    y = cy + dx * sin_a + dy * cos_a  # Rotasi sumbu Y
```

- **Contoh**:
  - Jika sudut = **45°** (π/4 radian):
    - cos(45°) ≈ 0.707
    - sin(45°) ≈ 0.707
  - Titik (-15, -5) diputar 45°:
    - x = 100 + (-15)*0.707 - (-5)*0.707 ≈ 100 - 7.07 ≈ 92.93
    - y = 100 + (-15)*0.707 + (-5)*0.707 ≈ 100 - 14.14 ≈ 85.86

d. Cek Tepi Layar

Jika kotak keluar layar, muncul di sisi berlawanan:

```
if position.x > width: position.x = 0      # Kanan → Kiri
if position.x < 0: position.x = width      # Kiri → Kanan
if position.y > height: position.y = 0     # Bawah → Atas
if position.y < 0: position.y = height     # Atas → Bawah
```

---

3. Alur Program

1. **Inisialisasi**:
   - Kotak muncul di tengah layar.
   - Kecepatan dan percepatan awal = 0.
2. **Setiap Frame (30ms)**:
   - Hitung arah ke mouse → ubah jadi percepatan.
   - Update kecepatan & posisi.
   - Putar kotak sesuai arah gerak.
   - Cek jika keluar layar.
3. **Output**:
   - Kotak bergerak halus ke arah mouse.
   - Tampilkan **posisi (x,y)** dan **kecepatan (vx,vy)** di layar.

---

4. Analogi Sederhana

- Bayangkan **kotak seperti mobil remote control**.
- **Mouse adalah target** yang ingin dituju.
- **Percepatan = tekanan gas**, **Kecepatan = laju mobil**.
- Semakin dekat ke mouse, mobil melambat (karena percepatan kecil).

Catatan Penting

- **Kecepatan maksimum (**topspeed = 4**)**: Agar gerakan tidak terlalu cepat.
- **Perkecil percepatan (**dir.mult(0.5)**)**: Agar gerakan lebih halus.
- **Rotasi menggunakan trigonometri (**sin/cos**)**: Untuk putaran yang smooth.

Program ini menunjukkan **konsep vektor, fisika gerak, dan rotasi** dengan cara yang menyenangkan! 🚗💨
Cara Kerja Program:

1. Program membuat window dengan kotak di tengah
2. Setiap 30 milidetik, program:
    - Mengecek posisi mouse
    - Menghitung arah menuju mouse
    - Memperbarui posisi kotak
    - Memutar kotak sesuai arah gerak
    - Jika kotak keluar layar, muncul di sisi sebaliknya
    - Menampilkan posisi dan kecepatan

Contoh Perhitungan Sederhana:
Misal:

- Posisi kotak: (100, 100)
- Posisi mouse: (150, 120)

1. Hitung arah:
    - dx = 150 - 100 = 50
    - dy = 120 - 100 = 20
    - Panjang vektor = $\sqrt{(50^2 + 20^2)} \approx 53.85$
    - Normalisasi: (50/53.85, 20/53.85) ≈ (0.93, 0.37)
    - Percepatan: (0.93*0.5, 0.37*0.5) ≈ (0.465, 0.185)
2. Update kecepatan (misal kecepatan awal (1, 0.5)):
    - Kecepatan baru: (1 + 0.465, 0.5 + 0.185) ≈ (1.465, 0.685)
    - Jika melebihi topspeed (4), akan dibatasi
3. Update posisi:
    - Posisi baru: (100 + 1.465, 100 + 0.685) ≈ (101.465, 100.685)

```
#SCRIPT 5 : # Simulasi mobil kotak yang bergerak mengikuti arah mouse
import tkinter as tk
import math
from vector import Vector

class Mover:
    def __init__(self, canvas, width, height):
        self.canvas = canvas  # tempat menggambar
        self.width = width    # lebar layar
        self.height = height  # tinggi layar
        self.position = Vector(width / 2, height / 2)  # posisi awal di tengah
        self.velocity = Vector(0, 0)  # kecepatan awal (diam)
        self.acceleration = Vector(0, 0)  # percepatan awal
        self.topspeed = 4  # kecepatan maksimum
        self.r = 16  # ukuran
        self.id = canvas.create_polygon(0, 0, 0, 0, 0, 0, fill="gray", outline="black")  #
bentuk kotak

    def update(self, mouse_x, mouse_y):
        mouse = Vector(mouse_x, mouse_y)  # posisi mouse
        dir = mouse.subbed(self.position)    # arah menuju mouse
        dir.normalize()  # buat panjang vektor = 1
        dir.mult(0.5)    # perkecil pengaruhnya
        self.acceleration = dir  # set percepatan

        self.velocity.add(self.acceleration)  # tambah kecepatan
        self.velocity.limit(self.topspeed)    # batasi kecepatan maks
        self.position.add(self.velocity)      # update posisi

    def display(self):
        angle = self.velocity.heading()  # sudut berdasarkan arah gerak

        w, h = 30, 10  # ukuran kotak
        cx, cy = self.position.x, self.position.y  # posisi tengah

        cos_a = math.cos(angle)  # hitung komponen x
        sin_a = math.sin(angle)  # hitung komponen y

        points = []
        for dx, dy in [(-w/2, -h/2), (w/2, -h/2), (w/2, h/2), (-w/2, h/2)]:
            x = cx + dx * cos_a - dy * sin_a  # rotasi titik x
            y = cy + dx * sin_a + dy * cos_a  # rotasi titik y
            points.extend((x, y))

        self.canvas.coords(self.id, *points)  # update posisi kotak
```

```
    def check_edges(self):
        if self.position.x > self.width:  # jika keluar kanan
            self.position.x = 0           # muncul di kiri
        elif self.position.x < 0:         # jika keluar kiri
            self.position.x = self.width   # muncul di kanan

        if self.position.y > self.height:  # jika keluar bawah
            self.position.y = 0            # muncul di atas
        elif self.position.y < 0:          # jika keluar atas
            self.position.y = self.height  # muncul di bawah

# Setup Tkinter
root = tk.Tk()
root.title("Example 3.3: Pointing in the Direction of Motion")

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

mover = Mover(canvas, WIDTH, HEIGHT)

def animate():
    canvas.delete("text")
    mouse_x = canvas.winfo_pointerx() - canvas.winfo_rootx()  # posisi mouse x
    mouse_y = canvas.winfo_pointery() - canvas.winfo_rooty()  # posisi mouse y

    mover.update(mouse_x, mouse_y)  # update posisi kotak
    mover.check_edges()             # cek tepi layar
    mover.display()                 # gambar ulang

    # Tampilkan teks posisi dan kecepatan
    canvas.create_text(10, 10, anchor="nw",
                       text=f"Pos: ({mover.position.x:.1f}, {mover.position.y:.1f})",
                       font=("Arial", 12), tag="text")
    canvas.create_text(10, 30, anchor="nw",
                       text=f"Vel: ({mover.velocity.x:.2f}, {mover.velocity.y:.2f})",
                       font=("Arial", 12), tag="text")

    root.after(30, animate)  # ulangi setiap 30ms

animate()
root.mainloop()
```

Sistem Koordinat Polar vs. Cartesian Coordinates

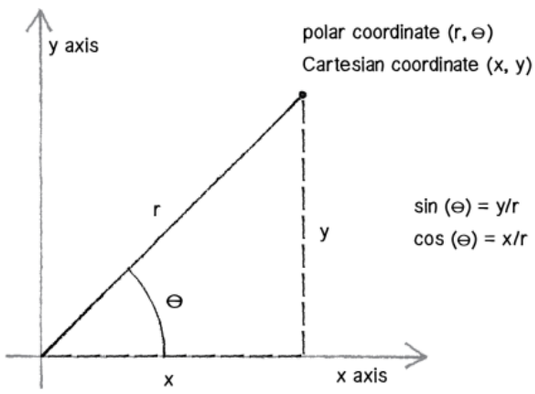
dua cara berbeda untuk menentukan posisi suatu titik, seperti yang digunakan dalam pemrograman dan matematika.

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 × cos(θ)
y = r × sin(θ)
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**

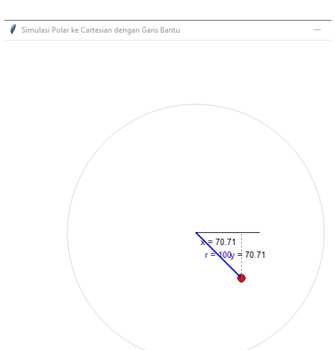
- Menggambar lingkaran yang **berputar** mengelilingi pusat layar.
- Menggunakan **koordinat polar** dan mengonversinya ke **kartesian**: $x = r \cdot \cos(\theta) \quad \text{dan} \quad y = r \cdot \sin(\theta)$

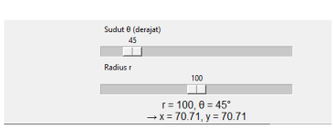
```
# SCRIPT 5 : Simulasi Konversi Koordinat Polar ke Kartesian
import tkinter as tk
import math

# === 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, 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.
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

Penjelasan Singkat:

- Tombol panah kiri dan kanan: memutar arah kapal.
- Tombol z: memberikan gaya dorong (thrust) ke arah yang dituju.
- Kapal berbentuk segitiga, bergerak di ruang dua dimensi dan membungkus jika keluar dari layar (wrap edges).

Penjelasan Simulasi Pesawat Luar Angkasa (Asteroids-style)

Program ini membuat simulasi **pesawat luar angkasa** yang bisa:

- **Berputar** (kiri/kanan)
- **Maju** dengan dorongan mesin
- **Melayang** di ruang hampa (tanpa gesekan)

---

1. Komponen Utama

a. Pesawat (Spaceship**)**

- **Posisi (x,y)**: Lokasi pesawat di layar
- **Kecepatan (velocity)**: Arah dan cepatnya gerak
- **Percepatan (acceleration)**: Perubahan kecepatan karena mesin
- **Heading (sudut)**: Arah hadap pesawat

b. Kontrol

- ⬅️ **(Panah Kiri)**: Putar pesawat ke kiri
- ➡️ **(Panah Kanan)**: Putar pesawat ke kanan
- **Z**: Nyalakan mesin (maju)

---

2. Cara Kerja Fisika

a. Gerakan Dasar

1. **Mesin menyala (**thrust()**)** → Tambahkan percepatan ke depan.

force = Vector(cos(sudut), sin(sudut)) * 0.1
(Gaya kecil = 0.1 agar tidak terlalu cepat)

2. **Update kecepatan & posisi**:

kecepatan += percepatan
kecepatan *= 0.995 *# Perlambat sedikit (efek ruang hampa)*
posisi += kecepatan

b. Putar Pesawat (turn()**)**

- Tekan panah kiri/kanan → ubah nilai heading (+/- 0.03 radian).
- **1 radian ≈ 57.3°**, jadi 0.03 radian ≈ **1.72° per frame**.

c. Rotasi Gambar Pesawat

Pesawat digambar sebagai **segitiga** yang berputar:

```
# Titik segitiga:
#   (-r, r) = kiri bawah
#   (0, -r) = ujung depan
#   (r, r)  = kanan bawah
for dx, dy in [(-r,r), (0,-r), (r,r)]:
   x = pusat_x + (dx*cos_a - dy*sin_a)  # Rotasi X
   y = pusat_y + (dx*sin_a + dy*cos_a)  # Rotasi Y
```

---

3. Contoh Perhitungan

a. Saat Mesin Menyala

- Jika sudut heading = 0 (menghadap atas):
  - force_x = cos(-90°) * 0.1 ≈ 0 * 0.1 = 0
  - force_y = sin(-90°) * 0.1 ≈ -1 * 0.1 = -0.1
    (Kecepatan bertambah ke atas (sumbu Y negatif))

b. Saat Berputar

- Setiap tekan panah kiri:
  - heading -= 0.03 (putar sedikit ke kiri)

c. Saat Melayang

- Kecepatan dikali 0.995 setiap frame:
  - Jika kecepatan = 5 → jadi 5 * 0.995 = 4.975
    (Efek seperti di ruang angkasa: tidak berhenti tiba-tiba)

---

4. Batas Layar
Jika pesawat keluar layar, muncul di sisi sebaliknya:
if posisi.x > layar_lebar + buffer: posisi.x = -buffer
if posisi.x < -buffer: posisi.x = layar_lebar + buffer
(buffer = jarak aman agar tidak muncul tiba-tiba)

---

5. Visualisasi

- **Segitiga abu-abu**: Badan pesawat.
- **Merah**: Saat mesin menyala.
- **Garis biru**: Arah hadap pesawat.

---

6. Analogi Sederhana

- **Pesawat** = Mainan hoverboard di lantai licin.
- **Mesin (Z)** = Dorongan kaki ke belakang.
- **Putar (⬅️/➡️)** = Putar badan untuk ubah arah.
- **Ruang hampa** = Tidak ada gesekan, jadi tetap meluncur.

Tips Belajar

1. **Coba ubah angka**:
    - 0.1 di thrust() → Lebih besar = lebih cepat.
    - 0.995 di update() → Ubah jadi 1 untuk tanpa perlambatan.
2. **Experiment**:
    - Tambahkan tembakan laser!
    - Buat bintang-bintang di latar.

```
# SCRIPT 8 : Simulasi kapal luar angkasa yang bergerak dan berputar
import tkinter as tk
import math
from vector import Vector

class Spaceship:
    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")

    def update(self):
        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)

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

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

    def thrust(self):
        angle = self.heading - math.pi / 2  # Kapal digambar tegak
        force = Vector.fromAngle(angle)
        force.mult(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):
        # Posisi dan rotasi segitiga
        angle = self.heading
        cx, cy = self.position.x, self.position.y

        cos_a = math.cos(angle)
        sin_a = math.sin(angle)

        # Titik segitiga (tubuh kapal)
        points = []
        for dx, dy in [(-self.r, self.r), (0, -self.r), (self.r, self.r)]:
            x = cx + dx * cos_a - dy * sin_a
            y = cy + dx * sin_a + dy * cos_a
            points.extend((x, y))

        self.canvas.coords(self.id, *points)

        # Warna berdasarkan thrust
        if self.thrusting:
            self.canvas.itemconfig(self.id, fill="red")
        else:
            self.canvas.itemconfig(self.id, fill="gray")
        self.thrusting = False

        # Garis arah dari ujung segitiga ke depan
        nose_x = cx + 0 * cos_a - (-self.r) * sin_a
        nose_y = cy + 0 * sin_a + (-self.r) * cos_a

        line_length = 20
        dir_x = math.cos(self.heading - math.pi / 2) * line_length
        dir_y = math.sin(self.heading - math.pi / 2) * line_length

        # Buat atau perbarui garis
        if not hasattr(self, 'direction_line'):
            self.direction_line = self.canvas.create_line(nose_x, nose_y,
                                                          nose_x + dir_x, nose_y + dir_y,
                                                          fill='blue', width=2)
        else:
            self.canvas.coords(self.direction_line,
                               nose_x, nose_y,
                               nose_x + dir_x, nose_y + dir_y)

# Setup Tkinter
root = tk.Tk()
root.title("Exercise 3.5: Asteroids Spaceship")

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

ship = Spaceship(canvas, WIDTH, HEIGHT)

# Event state
keys = {"left": False, "right": False, "z": False}

def key_press(event):
    if event.keysym == "Left":
        keys["left"] = True
    elif event.keysym == "Right":
        keys["right"] = True
    elif event.keysym.lower() == "z":
        keys["z"] = True

def key_release(event):
    if event.keysym == "Left":
        keys["left"] = False
```

```
    elif event.keysym == "Right":
        keys["right"] = False
    elif event.keysym.lower() == "z":
        keys["z"] = False

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

def animate():
    canvas.delete("text")

    if keys["left"]:
        ship.turn(-0.03)
    if keys["right"]:
        ship.turn(0.03)
    if keys["z"]:
        ship.thrust()

    ship.update()
    ship.wrapEdges()
    ship.show()

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

    root.after(30, animate)

animate()
root.mainloop()
```

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:

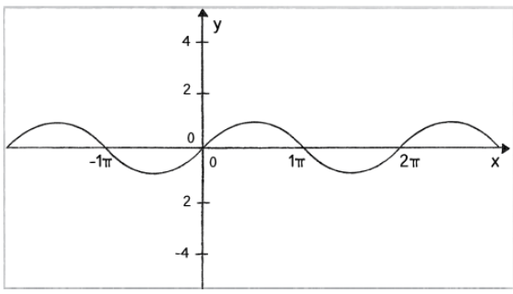


Figure 3.9: y = sine(x)

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**:

$y\_offset = A \times \sin(\omega t)$

$y\_actual = CENTER\_Y + y\_offset$

🎞️ Perhitungan Frame-by-Frame

Parameter:

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

Contoh Perhitungan:

Frame 0 (t=0s):

$y\_offset = 100 \times \sin(2\pi \times 0.5 \times 0) = 0$

$y\_actual = 200 + 0 = 200$

Frame 1 (t=0.02s):

$\omega = 2\pi \times 0.5 \approx 3.1416$ rad/s

$y_offset = 100 \times \sin(3.1416 \times 0.02) \approx 100 \times 0.0628 \approx 6.28$px

$y\_actual \approx 200 + 6.28 \approx 206.28$

Frame 2 (t=0.04s):

$y_offset \approx 100 \times \sin(3.1416 \times 0.04) \approx 100 \times 0.1253 \approx 12.53$px

$y\_actual \approx 200 + 12.53 \approx 212.53$

Frame 25 (t=0.5s):

$y_offset = 100 \times \sin(3.1416 \times 0.5) \approx 100 \times 1 \approx 100$px (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**:

$x(t) = A \times \sin(2\pi ft + \phi)$

   - 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**:

$\omega = 2\pi f$

3. **Simpangan X**:

$x_{offset} = A \times \sin(\omega t)$

$x_{actual} = CENTER\_X + x_{offset}$

🎞️ Perhitungan Frame-by-Frame

Parameter:

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

Contoh Perhitungan:

Frame 0 (t=0s):

$x_{offset} = 100 \times \sin(2\pi \times 0.5 \times 0) = 0$

$x_{actual} = 300 + 0 = 300$ (tepat di pusat)

Frame 1 (t=0.02s):

$\omega = 2\pi \times 0.5 \approx 3.1416$ rad/s

$x_{offset} = 100 \times \sin(3.1416 \times 0.02) \approx 100 \times 0.0628 \approx 6.28$px

$x_{actual} \approx 300 + 6.28 \approx 306.28$

Frame 25 (t=0.5s):

$x_{offset} = 100 \times \sin(3.1416 \times 0.5) \approx 100 \times 1 \approx 100$px (simpangan maksimum kanan)

Frame 50 (t=1.0s):

$x_{offset} = 100 \times \sin(3.1416 \times 1.0) \approx 100 \times 0 \approx 0$px (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

Example 3.5: Simple Harmonic Motion I

Harmonik Sederhana Berdasarkan Frame Count

Program ini menunjukkan gerakan bolak-balik seperti ayunan jam dinding. Berikut penjelasan sederhananya:

1. Konsep Dasar

- **Gerakan bolak-balik** seperti ayunan atau pegas
- **Perioda (waktu 1 putaran penuh)**: 120 frame
- **Jarak maksimum (amplitudo)**: 200 piksel
- **Pusat layar**: (320, 120)

2. Rumus Utama

python
Copy
Download

x = amplitudo * sin(2π * frame_count / perioda)

Penjelasan Rumus:

1. 2 * math.pi = 1 putaran penuh lingkaran (360°)
2. frame_count / perioda = seberapa jauh progres dalam 1 perioda
3. sin() = menghitung posisi naik-turun antara -1 sampai 1

3. Cara Kerja Program

a. Inisialisasi

- Membuat canvas (area gambar) berukuran 640x240 piksel
- Menentukan:
  - Perioda = 120 frame (2 detik jika 60 frame/detik)
  - Amplitudo = 200 piksel
  - Titik pusat di (320, 120)

b. Animasi (update)

1. Hitung posisi x:
   - Frame ke-0: x = 200 * sin(0) = 0 → di tengah
   - Frame ke-30: x = 200 * sin(π/2) = 200 → paling kanan
   - Frame ke-60: x = 200 * sin(π) = 0 → kembali ke tengah
   - Frame ke-90: x = 200 * sin(3π/2) = -200 → paling kiri
2. Gambar garis dan bola:
   - Garis dari pusat ke posisi bola
   - Bola berjari-jari 24 piksel
3. Naikkan frame_count dan ulangi setiap 16ms (~60fps)

4. Contoh Perhitungan

Frame	Perhitungan	Posisi x
0	200*sin(0)	0
30	200*sin(π/2)	200

Frame	Perhitungan	Posisi x
60	200*sin(π)	0
90	200*sin(3π/2)	-200
120	200*sin(2π)	0

5. Analogi Sederhana

Bayangkan:

- Sebuah bandul jam yang berayun
- 120 frame = waktu yang dibutuhkan untuk ayunan penuh (kanan-tengah-kiri-tengah)
- Amplitudo 200 = panjang ayunan maksimal

6. Percobaan Menarik

1. Ubah perioda jadi 60 frame → gerakan lebih cepat
2. Ubah amplitudo jadi 100 → ayunan lebih pendek
3. Tambahkan sumbu y untuk gerakan melingkar

```
# SCRIPT 9 : # Simulasi Gerakan Harmonik Sederhana (SHM) Berdasarkan Frame Count
import tkinter as tk
import math

class SHMFrameCountApp:
    def __init__(self, root):
        self.root = root
        self.root.title("SHM - Based on Frame Count")

        self.canvas_width = 640
        self.canvas_height = 240
        self.canvas = tk.Canvas(root, width=self.canvas_width, height=self.canvas_height,
bg="white")
        self.canvas.pack()

        self.period = 120
        self.amplitude = 200
        self.frame_count = 0

        self.center_x = self.canvas_width // 2
        self.center_y = self.canvas_height // 2

        # Buat elemen awal (garis dan lingkaran)
        self.line = self.canvas.create_line(self.center_x, self.center_y, self.center_x,
self.center_y, width=2)
        self.circle = self.canvas.create_oval(0, 0, 0, 0, fill="gray", outline="black",
width=2)

        self.update()

    def update(self):
        # Hitung posisi berdasarkan rumus SHM dengan frameCount
        x = self.amplitude * math.sin((2 * math.pi * self.frame_count) / self.period)
        y = 0  # tetap di sumbu horizontal

        # Update garis dari pusat ke titik bola
        self.canvas.coords(
            self.line,
            self.center_x, self.center_y,
            self.center_x + x, self.center_y + y
        )

        # Update lingkaran (bola)
        r = 24
        self.canvas.coords(
            self.circle,
            self.center_x + x - r,
            self.center_y + y - r,
            self.center_x + x + r,
            self.center_y + y + r
        )

        self.frame_count += 1
        self.root.after(16, self.update)  # sekitar 60 FPS
```

```
if __name__ == "__main__":
    root = tk.Tk()
    app = SHMFrameCountApp(root)
    root.mainloop()
```

Example 3.5: Simple Harmonic Motion II

Penjelasan Gerak Harmonik Sederhana

Program ini mensimulasikan **gerak bolak-balik** seperti:

- Bandul jam
- Pegas yang digetarkan
- Ayunan mainan

---

1. Konsep Dasar

- **Gerak Harmonik Sederhana** = Gerak bolak-balik teratur di sekitar titik seimbang.
- **Amplitudo** = Jarak maksimum dari titik tengah (200 piksel).
- **Kecepatan Sudut** = Seberapa cepat benda berosilasi (0.05 radian per frame).

---

2. Rumus Penting

Posisi Benda

x = amplitudo * sin(sudut)

- sin(sudut) menghasilkan nilai antara -1 sampai 1.
- Hasilnya: x bergerak dari **-200** sampai **200** piksel.

Perubahan Sudut

sudut += kecepatan_sudut

- Setiap frame, sudut bertambah **0.05 radian** (≈ 2.86°).

---

3. Cara Kerja Program

a. Inisialisasi

- **Titik Tengah**: Layar dibagi 2 (center_x = 320, center_y = 120).
- **Garis**: Menghubungkan titik tengah dengan bola.
- **Bola**: Lingkaran abu-abu yang bergerak.

b. Animasi (update()**)**

1. Hitung posisi x menggunakan sin(sudut):
    - Jika sudut = 0 → sin(0) = 0 → x = 0 (tengah).
    - Jika sudut = π/2 (90°) → sin(π/2) = 1 → x = 200 (kanan maksimum).
2. Update posisi garis dan bola:

garis: (320, 120) → (320 + x, 120)

bola: (320 + x - 24, 120 - 24) → (320 + x + 24, 120 + 24)

3. Tambah sudut untuk frame berikutnya.

c. Kecepatan Animasi

- root.after(16, update) = Update setiap **16 milidetik** (~60 frame/detik).

---

4. Contoh Perhitungan

Sudut (rad)	sin(sudut)	**Posisi** x
0.00	0.00	0
0.05	0.05	10
0.79 (≈π/4)	0.71	142
1.57 (≈π/2)	1.00	200

- **Pola Gerak**:
  0 → 200 → 0 → -200 → 0 → ... (bolak-balik terus).

---

5. Analogi Sederhana

1. **Bandul Jam**:
    - Amplitudo = Panjang tali bandul.
    - sin(sudut) = Posisi bandul saat berayun.
2. **Pegas**:
    - Bola di ujung pegas yang ditarik lalu dilepas.

---

6. Percobaan Seru

1. **Ubah Amplitudo**:

self.amplitude = 100 *# Gerakan lebih pendek*

2. **Ubah Kecepatan**:

self.angle_velocity = 0.1 *# Lebih cepat*

3. **Tambahkan Gravitasi**:

y = self.amplitude * math.cos(self.angle) *# Gerak vertikal*

Catatan:

- sin dan cos membuat gerakan **halus** seperti alam.
- Semakin besar angle_velocity, semakin **cepat** osilasinya.

```
# SCRIPT 9 : Simulasi Gerak Harmonik Sederhana (Simple Harmonic Motion)
import tkinter as tk
import math

class SimpleHarmonicMotionApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Simple Harmonic Motion - Tkinter")
        self.canvas_width = 640
        self.canvas_height = 240
        self.canvas = tk.Canvas(root, width=self.canvas_width, height=self.canvas_height,
bg="white")
        self.canvas.pack()

        self.angle = 0
        self.angle_velocity = 0.05
        self.amplitude = 200

        self.center_x = self.canvas_width // 2
        self.center_y = self.canvas_height // 2

        # Garis dan lingkaran
        self.line = self.canvas.create_line(self.center_x, self.center_y, self.center_x,
self.center_y, width=2)
        self.circle = self.canvas.create_oval(0, 0, 0, 0, fill="gray", outline="black",
width=2)

        self.update()

    def update(self):
        # Hitung posisi berdasarkan gerak harmonik sederhana
        x = self.amplitude * math.sin(self.angle)
        y = 0  # tetap di sumbu horizontal

        # Update garis
        self.canvas.coords(
            self.line,
            self.center_x, self.center_y,
            self.center_x + x, self.center_y + y
        )

        # Update lingkaran
        r = 24
        self.canvas.coords(
            self.circle,
            self.center_x + x - r,
            self.center_y + y - r,
            self.center_x + x + r,
            self.center_y + y + r
        )

        # Update sudut
        self.angle += self.angle_velocity

        # Loop
        self.root.after(16, self.update)  # sekitar 60 FPS

if __name__ == "__main__":
    root = tk.Tk()
    app = SimpleHarmonicMotionApp(root)
    root.mainloop()
```

Pertanyaan bagus! Contoh **Example 3.5.1** dan **Example 3.5.2** dari *The Nature of Code* sama-sama menggambarkan **Simple Harmonic Motion (SHM)**, tetapi mereka **berbeda dalam cara mengatur waktu** (time step) untuk menghasilkan gerakan. Berikut penjelasan rinci perbedaannya:

🧪 **Example 3.5.2:**

```
let angle = 0;
let angleVelocity = 0.05;

let x = amplitude * sin(angle);
angle += angleVelocity;
```

🔁 **Karakteristik:**

- **Menggunakan angle sebagai variabel waktu** (waktu diskret dalam bentuk sudut).
- Gerakan sinusoidal dikendalikan oleh **kecepatan sudut (angleVelocity)**.
- Lebih fleksibel karena kamu bisa mempercepat atau memperlambat gerakan dengan mengubah angleVelocity.
- Cocok untuk simulasi **gerakan osilasi dinamis** atau interaktif (misalnya, bisa diberi percepatan angular, friction, dll).

🧪 **Example 3.5.1:**

```
let x = amplitude * sin((TWO_PI * frameCount) / period);
```

🔁 **Karakteristik:**

- **Menggunakan frameCount** (jumlah frame sejak awal program dijalankan) sebagai penentu waktu.
- Gerakan sinusoidal **tergantung pada frame keberapa animasi berjalan**, dan period menentukan lamanya 1 siklus penuh.
- Lebih **terkontrol secara periodik**, cocok untuk visualisasi **animasi yang sinkron atau repetitif**.
- Tidak fleksibel untuk diintervensi secara fisik (misal menambahkan gaya), karena gerakannya ditentukan secara eksplisit oleh waktu.

📊 **Perbandingan Ringkas:**

Fitur	Example 3.5.2	Example 3.5.1
Penanda waktu	angle	frameCount
Kontrol kecepatan	angleVelocity	period (jumlah frame per siklus)
Lebih fleksibel untuk simulasi	Ya (bisa pakai percepatan, gaya)	Tidak (ditentukan waktu eksplisit)
Cocok untuk	Simulasi fisika, gaya, osilasi	Animasi sinkron, visualisasi periodik

---

🎯 **Kapan pakai yang mana?**

- Gunakan **Example 3.5.2** jika kamu ingin:
  - Menerapkan hukum fisika.
  - Mengubah kecepatan/arah.
  - Membuat interaktivitas dinamis.
- Gunakan **Example 3.5.1** jika kamu ingin:
  - Gerakan stabil dan sinkron.
  - Perulangan animasi yang tepat waktu (misalnya gelombang).

Penjelasan Script Osilator

Script ini membuat animasi di mana beberapa titik bergerak secara acak seperti ayunan. Mari kita bahas bagian per bagian:

1. Persiapan Awal

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

- tkinter: Untuk membuat jendela dan gambar
- math: Untuk perhitungan matematika (sinus)
- random: Untuk membuat gerakan acak
- Vector: Untuk menghitung posisi dan kecepatan

2. Kelas Oscillator (Objek yang Bergerak)

Setiap osilator adalah sebuah titik yang bergerak seperti ayunan.

Inisialisasi:

```
def __init__(self, canvas):
    self.canvas = canvas  # tempat menggambar

    # Sudut awal (0,0) - dimulai dari tengah
```

```
    self.angle = Vector(0, 0)

    # Kecepatan perubahan sudut (acak)
    self.angle_velocity = Vector(random.uniform(-0.05, 0.05),
                                 random.uniform(-0.05, 0.05))

    # Jarak maksimal ayunan (acak)
    self.amplitude = Vector(random.uniform(20, WIDTH/2),
                            random.uniform(20, HEIGHT/2))

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

Update Posisi:

```
def update(self):
    self.angle.add(self.angle_velocity)  # Tambah sudut dengan kecepatannya
```

Gambar di Layar:

```
def display(self):
    # Hitung posisi x dan y menggunakan rumus sinus
    x = math.sin(self.angle.x) * self.amplitude.x
    y = math.sin(self.angle.y) * self.amplitude.y

    # Pusat layar
    center_x, center_y = WIDTH / 2, HEIGHT / 2

    # Posisi titik osilasi
    px, py = center_x + x, center_y + y

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

    # Update lingkaran (titik osilasi)
    radius = 16
    self.canvas.coords(
        self.circle_id,
        px - radius, py - radius,  # titik kiri atas
        px + radius, py + radius   # titik kanan bawah
    )
```

3. Program Utama

Setup Layar:

```
root = tk.Tk()
root.title("Example 3.7: Oscillator Objects")

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

Buat 10 Osilator:

```
oscillators = [Oscillator(canvas) for _ in range(10)]
```

Fungsi Animasi:

```
def animate():
    for osc in oscillators:
        osc.update()    # Update posisi
        osc.display()   # Gambar ulang
    root.after(30, animate)  # Ulangi setiap 30ms
```

Cara Kerja Program:

1. Membuat 10 titik osilator dengan:
    - Kecepatan gerak berbeda-beda (acak)
    - Jarak ayunan berbeda-beda (acak)
2. Setiap 30 milidetik:
    - Setiap titik diupdate posisinya menggunakan rumus sinus
    - Garis dan lingkaran digambar ulang di posisi baru
3. Gerakan sinus membuat titik bergerak maju-mundur seperti ayunan

Contoh Perhitungan Sederhana:

Misal sebuah osilator memiliki:

- amplitude.x = 100
- angle.x = 0.5 radian
- angle_velocity.x = 0.03

Perhitungan posisi x:

$x = \sin(0.5) \times 100$
$\approx 0.479 \times 100$
≈ 47.9

Setiap frame, sudut bertambah 0.03:

text

Copy
Download
angle.x baru = 0.5 + 0.03 = 0.53
x baru = sin(0.53) × 100 ≈ 0.505 × 100 ≈ 50.5
Jadi titik bergerak dari x≈47.9 ke x≈50.5 dalam satu frame!
Analogi Sederhana:
Bayangkan kamu mengikat bola dengan tali ke langit-langit:

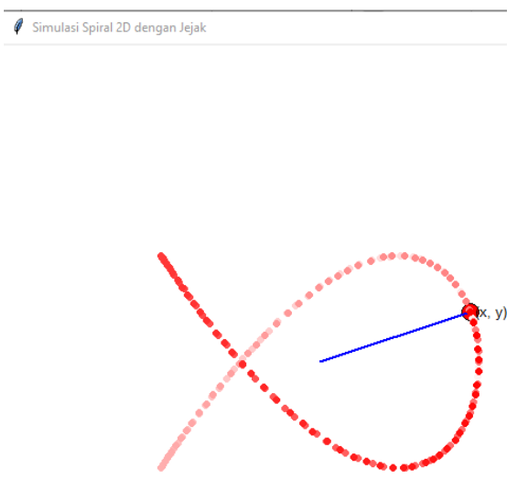
- Bola akan berayun bolak-balik
- Setiap bola memiliki panjang tali berbeda (amplitude)
- Setiap bola memiliki kecepatan ayunan berbeda (angle_velocity)
- Program ini mensimulasikan 10 bola yang berayun acak seperti itu

Semoga penjelasan ini membantu! Kalau ada yang belum jelas bisa ditanyakan lagi ya.

```
# SCRIPT 10 : Simulasi Gerakan Osilator
import tkinter as tk
import math
import random
from vector import Vector

# === Konstanta Layar ===
WIDTH, HEIGHT = 640, 240

# === Kelas Oscillator ===
class Oscillator:
    def __init__(self, canvas):
        self.canvas = canvas
        self.angle = Vector(0, 0)
        self.angle_velocity = Vector(random.uniform(-0.05, 0.05), random.uniform(-0.05,
0.05))
        self.amplitude = Vector(random.uniform(20, WIDTH / 2), random.uniform(20, HEIGHT /
2))

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

    def update(self):
        self.angle.add(self.angle_velocity)

    def display(self):
        # Konversi sudut sinusoidal ke posisi (x, y)
        x = math.sin(self.angle.x) * self.amplitude.x
        y = math.sin(self.angle.y) * self.amplitude.y

        center_x, center_y = WIDTH / 2, HEIGHT / 2
        px, py = center_x + x, center_y + y

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

        # Update lingkaran
        radius = 16
        self.canvas.coords(
            self.circle_id,
            px - radius, py - radius,
            px + radius, py + radius
        )

# === Setup Tkinter ===
root = tk.Tk()
root.title("Example 3.7: Oscillator Objects")

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

# === Buat banyak oscillator ===
oscillators = [Oscillator(canvas) for _ in range(10)]

# === Fungsi animasi utama ===
def animate():
    for osc in oscillators:
        osc.update()
        osc.display()
    root.after(30, animate)
```

```
animate()
root.mainloop()
```

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)

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

Script ini membuat animasi sederhana gelombang statis menggunakan lingkaran-lingkaran yang disusun mengikuti pola gelombang sinus. Berikut penjelasannya:

1. Konstanta Layar

```
WIDTH, HEIGHT = 640, 240       # Lebar dan tinggi layar (640px × 240px)
ANGLE_VELOCITY = 0.2           # Kecepatan perubahan sudut (mengatur kerapatan gelombang)
AMPLITUDE = 100                # Tinggi gelombang (100px)
SPACING = 24                   # Jarak antar lingkaran (24px)
RADIUS = 24                    # Jari-jari lingkaran (24px)
```

- WIDTH **&** HEIGHT: Ukuran layar animasi.
- ANGLE_VELOCITY: Mengatur seberapa cepat sudut berubah, memengaruhi kerapatan gelombang.
- AMPLITUDE: Menentukan tinggi maksimum gelombang.
- SPACING: Jarak horizontal antar lingkaran.
- RADIUS: Ukuran lingkaran yang digambar.

---

2. Setup Tkinter

```
root = tk.Tk()  # Membuat jendela utama
root.title("Example 3.8: Static Wave")  # Judul jendela

canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")  # Membuat area gambar
canvas.pack()  # Menampilkan canvas di jendela
```

- tk.Tk(): Membuat jendela utama.
- tk.Canvas(): Membuat area gambar (kanvas) dengan latar belakang putih.
- canvas.pack(): Menampilkan kanvas di jendela.

---

3. Menggambar Gelombang Statis

```
angle = 0  # Sudut awal (0 radian)
for x in range(0, WIDTH + 1, SPACING):  # Loop dari x=0 hingga x=WIDTH dengan jarak SPACING
    # 1) Hitung posisi y menggunakan sinus
    y = AMPLITUDE * math.sin(angle)  # Nilai y bergantung pada sinus sudut
    center_y = y + HEIGHT / 2  # Geser y ke tengah layar

    # 2) Gambar lingkaran di posisi (x, center_y)
    canvas.create_oval(
        x - RADIUS, center_y - RADIUS,  # Posisi kiri-atas lingkaran
        x + RADIUS, center_y + RADIUS,  # Posisi kanan-bawah lingkaran
        fill="gray", outline="black", width=2  # Warna lingkaran
    )
```

```
    # 3) Tambahkan sudut untuk lingkaran berikutnya
    angle += ANGLE_VELOCITY
```

Penjelasan Loop:

1. for x in range(0, WIDTH + 1, SPACING)
    - Loop ini menggambar lingkaran setiap SPACING piksel dari kiri (x=0) ke kanan (x=WIDTH).
2. y = AMPLITUDE * math.sin(angle)
    - Menghitung posisi vertikal (y) menggunakan fungsi sinus.
    - Semakin besar angle, semakin tinggi/nilai y berubah.
3. center_y = y + HEIGHT / 2
    - Menggeser y ke tengah layar agar gelombang tidak terpotong.
4. canvas.create_oval(...)
    - Menggambar lingkaran di posisi (x, center_y) dengan ukuran RADIUS.
5. angle += ANGLE_VELOCITY
    - Menambahkan sudut agar lingkaran berikutnya memiliki posisi y yang berbeda, membentuk gelombang.

---

4. Hasil Akhir

- **Gelombang sinus statis** terbentuk dari lingkaran-lingkaran yang posisinya diatur oleh fungsi sin(angle).
- Semakin besar ANGLE_VELOCITY, semakin rapat gelombangnya.
- Semakin besar AMPLITUDE, semakin tinggi gelombangnya.

```
# 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
Berikut penjelasan sederhana tentang cara kerja program gelombang ini:
1. Persiapan Awal

- Program membuat jendela putih berukuran 640x240 pixel
- Kita akan menggambar banyak lingkaran kecil yang membentuk gelombang
- Jarak antar lingkaran = 24 pixel
- Ukuran tiap lingkaran = 48 pixel

2. Cara Membuat Gelombang Bergerak
Program ini menggunakan 3 bahan utama:

1. start_angle: Sudut awal untuk memulai gelombang (seperti titik start lomba lari)
2. angle_velocity: Kecepatan perubahan sudut (seperti seberapa cepat pelari bergerak)
3. **Fungsi sinus (**math.sin**)**: Untuk menghitung naik-turunnya gelombang

3. Alur Kerja Program

1. **Pertama kali jalan**:
    - Membuat banyak lingkaran sejajar horizontal di tengah layar
    - Lingkaran-lingkaran ini disimpan dalam daftar circles
2. **Saat animasi berjalan**:
    - Setiap 30 milidetik, program menghitung ulang posisi semua lingkaran

- Posisi Y tiap lingkaran dihitung menggunakan rumus sinus:

y = sin(sudut) → hasilnya antara -1 sampai 1
lalu diubah menjadi posisi pixel di layar (0-240)

- Sudut untuk lingkaran berikutnya selalu ditambah sedikit (angle_velocity)

3. **Agar gelombang bergerak**:
   - Setiap frame animasi, start_angle ditambah 0.02
   - Ini seperti menggeser awal gelombang sedikit demi sedikit
   - Efeknya terlihat seperti gelombang bergerak ke kiri

4. Rumus Penting

1. **Menghitung posisi Y**:

sin_value = math.sin(angle) *# Hasil antara -1 sampai 1*
y = map_value(sin_value, -1, 1, 0, HEIGHT) *# Diubah ke posisi layar*
Fungsi map_value mengubah angka dari rentang -1..1 menjadi 0..240 (tinggi layar)

2. **Pergerakan gelombang**:

self.start_angle += 0.02 *# Geser sedikit titik awalnya*

5. Analogi Sederhana

Bayangkan teman-teman berbaris membentuk gelombang:

- Kalian semua memegang tali panjang
- Saat diberi aba-aba, kalian bergerak naik-turun mengikuti irama (sinus)
- Setiap anak di barisan mulai bergerak sedikit lebih lambat dari sebelahnya
- Guru terus menggeser titik start-nya, sehingga seluruh gelombang terlihat bergerak

6. Hasil Akhir

- Terlihat rangkaian lingkaran membentuk gelombang yang bergerak mulus
- Gelombang akan terus bergerak ke kiri tanpa putus
- Kecepatan gelombang bisa diatur dengan mengubah nilai angle_velocity dan penambahan start_angle

Program ini menunjukkan bagaimana matematika (khususnya fungsi sinus) bisa digunakan untuk membuat animasi yang indah!

🧠 **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 :

1. Konsep Dasar

Program ini membuat animasi gelombang yang terlihat lebih alami (tidak terlalu sempurna seperti gelombang sinus) menggunakan teknik yang disebut **Perlin Noise**.

2. Bahan-Bahan yang Digunakan

- **Kanvas/Tempat Gambar**: Layar putih ukuran 640x240 pixel
- **Lingkaran**: Banyak lingkaran biru muda (diameter 48 pixel) berjarak 24 pixel
- **Perlin Noise**: Seperti "mesin pembuat angka acak yang halus" untuk membuat pola alam

3. Cara Kerja Program

1. **Persiapan Awal**:
    - Membuat jendela dan tempat gambar (canvas)
    - Menyiapkan "mesin noise" (PerlinNoise)
    - Menggambar lingkaran-lingkaran di tengah layar
2. **Saat Animasi Berjalan** (setiap 30 milidetik):
    - Program bertanya ke "mesin noise": "Berapa nilai untuk posisi X sekarang?"
    - Mesin noise memberi angka antara -1 sampai 1
    - Angka ini diubah menjadi posisi Y lingkaran
    - Posisi X tetap, posisi Y berubah-ubah mengikuti nilai noise
    - Geser sedikit nilai offset agar gelombang terlihat bergerak

4. Rumus Penting

1. **Mendapatkan Nilai Noise**:

noise_value = self.noise(x_offset) *# Hasil antara -1 sampai 1*

2. **Mengubah Nilai Noise ke Posisi Y**:

y = map_value(noise_value, 0, 1,
center_y + amplitude, *# Paling bawah*
center_y - amplitude) *# Paling atas*

5. Analogi Sederhana

Bayangkan ini seperti permainan "telepon rusak":

1. Setiap lingkaran adalah seorang anak dalam barisan
2. Guru bisikkan angka ke anak pertama, yang meneruskan ke anak berikutnya dengan sedikit perubahan
3. Angka ini menentukan seberapa tinggi tangan anak harus diangkat
4. Hasilnya membentuk pola naik-turun yang alami

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**:

python
Copy
Download
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 seperti di JS ---
```

```
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.

---

program simulasi pendulum (bandul) :
1. Apa Itu Program Ini?
Program ini membuat simulasi bandul yang bisa:
- Berayun sendiri seperti bandul jam
- Bisa ditarik dengan mouse untuk mengubah posisinya
- Menampilkan informasi sudut, kecepatan, dan percepatan

2. Bagian-Bagian Penting

1. **Titik Gantung**: Titik di atas tempat tali bandul digantung (posisi x=320, y=50)
2. **Tali**: Garis yang menghubungkan titik gantung dengan bola bandul
3. **Bola Bandul**: Lingkaran abu-abu yang berayun (radius 24 pixel)
4. **Panjang Tali**: 175 pixel

3. Cara Kerja Bandul

1. **Gerakan Alami**:
    - Bola akan berayun karena pengaruh gravitasi (nilai 0.4)
    - Semakin lama ayunan akan berkurang karena ada redaman (damping 0.995)
    - Perhitungan menggunakan rumus fisika sederhana tentang gerak bandul
2. **Rumus Penting**:

percepatan = (-gravitasi / panjang_tali) × sin(sudut)
kecepatan += percepatan
sudut += kecepatan

3. **Saat Ditarik Mouse**:
    - Bisa klik dan tarik bola bandul ke posisi baru
    - Saat dilepas, bandul akan berayun dari posisi baru tersebut

4. Analogi Sederhana

Bayangkan mainan bandul yang terbuat dari:

- Seutas benang (tali)
- Sebuah bola besi di ujungnya
- Bisa kita tarik ke samping lalu dilepas untuk berayun
- Perlahan-lahan ayunannya akan berkurang karena ada gesekan udara

5. Informasi yang Ditampilkan

Di pojok kiri atas ada teks yang menunjukkan:

1. Sudut bandul (dalam derajat)
2. Percepatan bandul (rad/s²)
3. Kecepatan bandul (rad/s)

6. Proses Animasi

1. Setiap 16 milidetik (~60 frame per detik):
    - Hitung posisi baru bandul
    - Gambar ulang tali dan bola di posisi baru
    - Update informasi teks
2. Saat mouse bergerak sambil menekan:
    - Bandul mengikuti posisi mouse
    - Sudut dihitung berdasarkan posisi mouse

7. Coba Sendiri!

1. Klik dan tarik bola bandul ke berbagai posisi
2. Lihat bagaimana bandul berayun berbeda tergantung:
    - Seberapa jauh ditarik (sudut awal)
    - Kecepatan saat dilepaskan

Program ini menggabungkan konsep fisika sederhana dengan pemrograman untuk membuat simulasi interaktif yang menyenangkan! ⏳

SIMULASI PENDULUM

```
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

3. Cara Kerja Program

1. **Setiap Frame (60 kali/detik)**:
   - Hitung percepatan sudut kedua bandul menggunakan rumus fisika
   - Update posisi bandul berdasarkan percepatan dan kecepatan
   - Gambar ulang posisi bandul dan jejaknya
2. **Rumus Fisika Sederhana**:
   Program menggunakan persamaan gerak pendulum ganda yang mempertimbangkan:
   - Gravitasi (g = 1)
   - Panjang tali (r1, r2)
   - Massa bola (m1, m2)
   - Sudut dan kecepatan sudut (a1, a2, a1_v, a2_v)
3. **Efek Redaman**:

a1_v *= 0.999 *# Perlahan mengurangi kecepatan*
a2_v *= 0.999 *# Agar gerakan lebih stabil*

4. Proses Gambar

1. **Hitung Posisi**:

x1 = cx + r1 * sin(a1) *# Posisi bandul 1*
y1 = cy + r1 * cos(a1)
x2 = x1 + r2 * sin(a2) *# Posisi bandul 2*
y2 = y1 + r2 * cos(a2)

2. **Gambar Komponen**:
   - Garis penghubung dari titik gantung ke bandul 1
   - Bola bandul 1 (lingkaran hitam)
   - Garis penghubung dari bandul 1 ke bandul 2
   - Bola bandul 2 (lingkaran hitam)
   - Jejak biru dari gerakan bandul 2

5. Informasi yang Ditampilkan

Di pojok kiri atas ada:

- Sudut bandul 1 (a1 dalam radian)
- Sudut bandul 2 (a2 dalam radian)

6. Analogi Sederhana

Bayangkan dua buah bola yang diikat dengan tongkat:

- Bola pertama diikat ke langit-langit
- Bola kedua diikat ke bola pertama
- Ketika digoyang, gerakannya akan sangat menarik dan tidak terduga!

7. Fakta Menarik

- Pendulum ganda menunjukkan gerakan kacau (chaos)
- Posisi awal sedikit berbeda bisa hasilkan gerakan sangat berbeda
- Jejak biru akan membentuk pola-pola indah yang tidak berulang

8. Coba Sendiri!

1. Ubah parameter di awal program:

r1, r2 = 150, 100 *# Ubah panjang tali*
m1, m2 = 20, 10 *# Ubah massa bola*
g = 0.5 *# Ubah gravitasi*

2. Lihat bagaimana perubahan mempengaruhi gerakan pendulum

Program ini menunjukkan bagaimana hukum fisika bisa menciptakan gerakan yang indah dan 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:

🌟 **Apa Itu Program Ini?**

Program ini membuat simulasi bola biru yang digantung dengan pegas:

- Bola bisa ditarik dengan mouse dan akan bergerak seperti pegas
- Ada gaya gravitasi yang menarik bola ke bawah
- Pegas akan menarik bola kembali ke posisi semula

🧱 **Bagian-Bagian Utama**

1. **Bola Biru (Bob)**:
   - Massa: 20
   - Ukuran: radius 20 pixel
   - Bisa ditarik dengan mouse
2. **Pegas (Spring)**:
   - Titik gantung: di tengah atas layar (320,10)
   - Panjang normal: 100 pixel
   - Kekakuan pegas: 0.2 (semakin besar semakin kaku)
3. **Gaya yang Bekerja**:
   - Gravitasi: menarik bola ke bawah (0.98)
   - Gaya pegas: menarik bola kembali ke titik gantung

🔄 **Cara Kerja Program**

1. **Setiap Frame (60 kali/detik)**:
   - Hitung semua gaya yang bekerja pada bola
   - Update posisi bola berdasarkan kecepatan dan percepatan
   - Gambar ulang posisi bola dan pegas
2. **Rumus Fisika Sederhana**:

python

Copy

Download

Gaya pegas = -k × (panjang_sekarang - panjang_normal)

   - k: kekakuan pegas
   - Semakin jauh ditarik, semakin kuat gaya tariknya kembali
3. **Saat Bola Ditarik**:
   - Bisa klik dan tarik bola ke posisi baru
   - Saat dilepas, pegas akan menarik bola kembali sambil berosilasi

🖱️ **Interaksi Mouse**

1. **Klik kiri**: Cek apakah klik mengenai bola
2. **Tahan dan drag**: Pindahkan bola
3. **Lepas klik**: Bola akan mulai bergerak dengan gaya pegas

📐 **Perhitungan Posisi**

1. Posisi bola diupdate dengan:

python
Copy
Download
kecepatan_baru = kecepatan + percepatan
posisi_baru = posisi + kecepatan_baru

2. Ada batasan panjang pegas:
    - Tidak boleh lebih pendek dari 30 pixel
    - Tidak boleh lebih panjang dari 200 pixel

🌈 **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)**:

python
Copy
Download

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**:

python
Copy
Download

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()
```