The Nature of Code

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

Chapter 1. Vectors

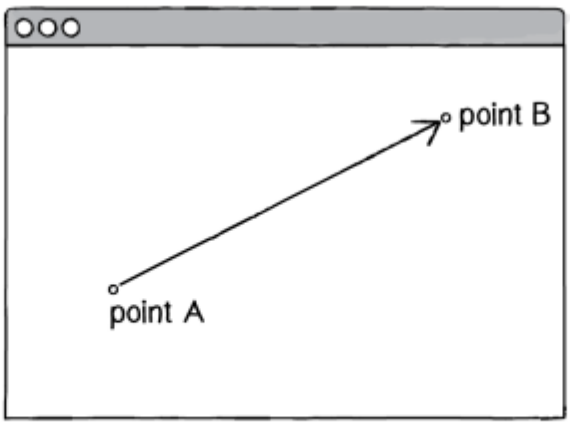


Figure 1.1: A vector (drawn as an arrow) has magnitude (length of arrow) and direction (which way it is pointing).

Bagus, kita akan menjelaskan materi **Chapter 1: Vectors** dengan cara sederhana dan menyenangkan, serta membuat contoh visualnya dengan **Python Tkinter**. Fokus kita adalah mengenalkan **konsep vektor (arah dan panjang)** dengan visualisasi interaktif.

💡 **Penjelasan Konsep "Vector"**

Apa itu Vektor?

Bayangkan kamu ingin pergi dari rumah ke sekolah. Kamu harus tahu **ke arah mana (arah)** kamu harus berjalan dan **seberapa jauh (jarak)** kamu harus melangkah. Nah, gabungan antara **arah** dan **jarak** itu disebut **vektor**.

Contoh Sederhana:

- Dari rumah (titik A) ke sekolah (titik B).
- Arah: ke kanan atas.
- Jarak: 5 langkah.

Kalau kita gambar, bentuknya seperti **panah** dari titik A ke B:

- **Ujung panah** menunjukkan arah.
- **Panjang panah** menunjukkan seberapa jauh.

🧠 **Apa yang Dipelajari Anak dari Ini?**

- Vektor = arah + panjang (jarak).
- Digambarkan sebagai panah dari satu titik ke titik lain.
- Bisa dilihat dalam kehidupan sehari-hari: jalan dari rumah ke sekolah, melempar bola, gerak benda, dll.
- Visualisasi membantu anak memahami konsep yang abstrak dengan mudah.

Kalau kamu mau lanjut, kita bisa tambahkan:

1. **Interaktifitas**: klik untuk menentukan titik A dan B.
2. **Animasi**: gerakan benda mengikuti vektor.
3. **Proyek mini**: simulasi bola yang bergerak.

Mau dilanjut ke versi interaktif atau dijelaskan cara membuat kelas Vector2D sendiri di Python?

1.1 Vectors, You Complete Me

📘 Konsep yang Perlu Dipahami

🎾 Kasus: Bola Memantul di Layar

Misalnya kita punya bola yang bergerak di layar. Awalnya kita simpan posisi dan kecepatan bola dengan 4 variabel:

```
x = 100      # posisi x
y = 100      # posisi y
xspeed = 1   # kecepatan ke kanan
yspeed = 3.3 # kecepatan ke bawah
```

Setiap kali gambar diperbarui:

- Posisi bola berubah mengikuti kecepatannya.
- Bola akan memantul jika menyentuh pinggir layar.

🔧 Masalah: Terlalu Banyak Variabel

Bayangkan kalau kita juga tambahkan angin, percepatan, gaya dorong, dan lain-lain, semua harus punya **komponen x dan y**. Jadi kodenya bisa menjadi **sangat banyak dan berantakan**.

✅ Solusi: Gunakan Vektor

Kita bisa membuat **satu objek vektor** yang menyimpan **dua nilai sekaligus: x dan y**. Jadi:

```
location = Vector(100, 100)
velocity = Vector(1, 3.3)
```

Ini lebih **rapi, sederhana**, dan nanti akan lebih **mudah dihitung** saat kita tambah fitur seperti gaya dorong atau gravitasi.

🧑‍🎨 Penjelasan Sederhana

"Bayangkan kamu punya mobil mainan. Untuk tahu gerakannya, kamu butuh tahu: ke arah mana dan seberapa cepat. Itu bisa kamu simpan dalam satu 'kotak' bernama vektor. Jadi kamu nggak perlu repot-repot simpan dua angka terpisah."

🧑‍💻 Contoh Tkinter: Bola Memantul (Tanpa Vektor vs Dengan Vektor)

1. Tanpa Vektor (Cara Lama)

Simulasi Koordinat

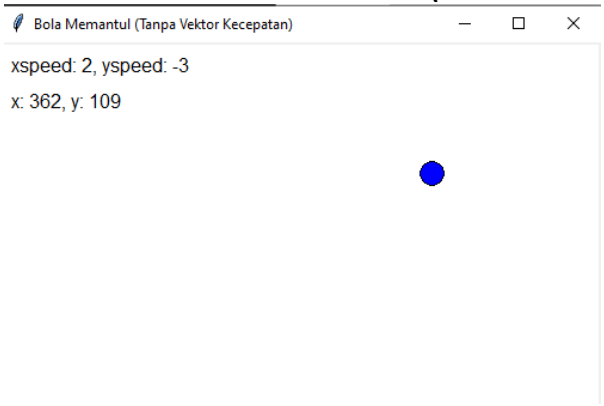
```
# SCRIPT 1 - Simulasi Pergerakan Titik 2D
# ----------------------------------------
# Simulasi pergerakan titik dalam 2D (x, y)
# Titik bergerak dengan kecepatan tetap (xspeed, yspeed)
# ----------------------------------------

# Posisi awal
x = 100
y = 100

# Kecepatan per langkah
xspeed = 2
yspeed = 3

# Cetak posisi awal dan kecepatan
print(f"Posisi awal: ({x}, {y})")
print(f"Kecepatan per langkah: ({xspeed}, {yspeed})\n")

# List untuk menyimpan koordinat tiap langkah
positions = []

# Hitung 10 langkah gerakan
for i in range(10):
```

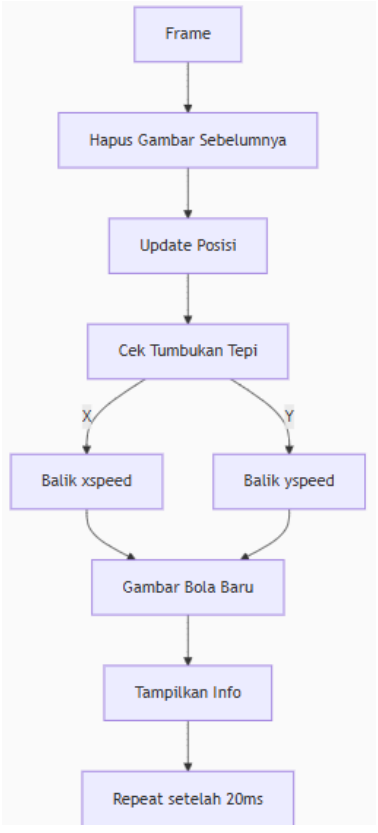
```
    # Update posisi berdasarkan kecepatan + (2, 3)
    x += xspeed
    y += yspeed

    # Simpan ke list
    positions.append((x, y))

# Cetak hasil koordinat
print("10 Koordinat hasil gerakan:")
for i, (px, py) in enumerate(positions):
    # Penjelasan: posisi ke-i = posisi sebelumnya + kecepatan
    print(f"Langkah {i+1}: posisi = ({px}, {py})")
```

Posisi awal: (100, 100)
Kecepatan per langkah: (2, 3)

10 Koordinat hasil gerakan:
Langkah 1: posisi = (102, 103)
Langkah 2: posisi = (104, 106)
Langkah 3: posisi = (106, 109)
Langkah 4: posisi = (108, 112)

BUAT SIMULASI BOLA MEMANTUL (bola ukuran 10, warna biru, posisi awal (100,100), kecepatan (2,3))





penjelasan lengkap tentang script simulasi bola memantul sederhana:

🌟 **Alur Program Utama**

1. **Inisialisasi**:
   - Posisi awal bola: (x=100, y=100)
   - Kecepatan awal: (xspeed=2, yspeed=3)
   - Ukuran canvas: 500x300 pixel
   - Ukuran bola: diameter 20 pixel (radius 10)
2. **Loop Animasi (50 FPS)**:

Diagram

📐 **Rumus Fisika Sederhana**

1. **Gerakan Bola**:

x = x + xspeed
y = y + yspeed

2. **Tumbukan Elastis Sempurna**:

Jika x ≤ 0 atau x ≥ 500: xspeed = -xspeed
Jika y ≤ 0 atau y ≥ 300: yspeed = -yspeed

🎞️ **Contoh Perhitungan Frame-by-Frame**

Frame 0:
Posisi: (100, 100)

Kecepatan: (2, 3)
Frame 1:
x = 100 + 2 = 102
y = 100 + 3 = 103
Belum sentuh tepi
Frame 2:
x = 102 + 2 = 104
y = 103 + 3 = 106
Frame N (Saat Sentuh Tepi Kanan):
Misal x = 490
Cek: 490 + 10 ≥ 500? YA
xspeed = -2
x = 490 + (-2) = 488
Frame N (Saat Sentuh Tepi Bawah):
Misal y = 290
Cek: 290 + 10 ≥ 300? YA
yspeed = -3
y = 290 + (-3) = 287

⏱️ **Karakteristik Gerakan**

- **Refresh Rate**: 50 FPS (setiap 20ms)
- **Kecepatan**: 2-3 pixel per frame
- **Waktu Tempuh**:
  - Horizontal: (500-100)/2 ≈ 200 frame (4 detik)
  - Vertikal: (300-100)/3 ≈ 67 frame (1.3 detik)

📊 **Visualisasi Tumbukan**
Sebelum tumbukan:
○→ (bergerak ke kanan)

Setelah tumbukan:
←○ (bergerak ke kiri)

💻 **Komponen Penting Kode**

1. **Variabel Global**:

```
x, y = 100, 100        # Posisi
xspeed, yspeed = 2, 3  # Kecepatan
```

2. **Fungsi** update():
   - Meng-handle logika gerakan dan tumbukan
   - Menggambar ulang seluruh scene setiap frame
3. **Tampilan Info**:
   - Menampilkan kecepatan dan posisi real-time
   - Posisi: atas kiri canvas

🔄 **Perbedaan dengan Versi Menggunakan Vector**

1. **Tanpa Kelas Vector**:
   - Operasi matematika manual
   - Variabel terpisah (x, y, xspeed, yspeed)
2. **Rendering**:
   - Menghapus dan menggambar ulang seluruh canvas
   - Versi vector hanya update posisi objek

🛠️ **Modifikasi Menarik**

```
# Tambahkan gravitasi
yspeed += 0.1  # percepatan gravitasi

# Ubah warna saat memantul
if x <= 0 or x >= 500 or y <= 0 or y >= 300:
```

```
    canvas.create_oval(x-10, y-10, x+10, y+10, fill="red")
else:
    canvas.create_oval(x-10, y-10, x+10, y+10, fill="blue")
```

📝 **Catatan Penting**

1. **Efisiensi**:
    - canvas.delete("all") kurang efisien untuk animasi kompleks
    - Lebih baik update posisi objek seperti versi vector
2. **Fisika**:
    - Tumbukan sempurna (energi tetap)
    - Untuk fisika lebih realistis bisa tambahkan:

```
xspeed *= 0.95  # Gesekan
yspeed *= 0.95
```

Script ini cocok untuk:

- Pembelajaran konsep animasi dasar
- Pengenalan fisika sederhana
- Prototyping game 2D sederhana

```
# SCRIPT 2 - Simulasi Bola Memantul (Tanpa Vektor Kecepatan)
import tkinter as tk

# 1. Kelas Vector2D >> belum pakai vector

# 2. Setup Tkinter window ##########################################
# titik awal bola
x = 100
y = 100
# kecepatan bola
xspeed = 2
yspeed = 3

window = tk.Tk()
# Judul dan ukuran window
window.title("Bola Memantul (Tanpa Vektor Kecepatan)")
canvas = tk.Canvas(window, width=500, height=300, bg="white")
canvas.pack()

# 3. Fungsi untuk update posisi mouse dan menggambar garis ##########
def update():
    # Global variabel untuk akses di dalam fungsi
    global x, y, xspeed, yspeed
    # Hapus gambar sebelumnya
    canvas.delete("all")
    # Update posisi bola
    # Penjelasan: posisi bola = posisi sebelumnya + kecepatan
    x += xspeed
    y += yspeed

    # Memantul di pinggir
    # Penjelasan: jika bola mencapai pinggir, kecepatan dibalik
    # Jika bola mencapai pinggir kiri/kanan atau atas/bawah, kecepatan dibalik
    if x <= 0 or x >= 500:
        xspeed *= -1
    if y <= 0 or y >= 300:
        yspeed *= -1

    # Gambar bola
    canvas.create_oval(x-10, y-10, x+10, y+10, fill="blue")

    # Tampilkan nilai kecepatan di layar
```

```
    canvas.create_text(10, 10, anchor="nw",
                       text=f"xspeed: {xspeed}, yspeed: {yspeed}",
                       font=("Arial", 12), fill="black")

    # Tampilkan nilai koordinat di layar
    canvas.create_text(10, 40, anchor="nw",
                       text=f"x: {x}, y: {y}",
                       font=("Arial", 12), fill="black")
    # Panggil fungsi update lagi setelah 20ms
    window.after(20, update)

# 4. Jalankan ############################################################
update()
window.mainloop()
```

Langkah	Posisi Sebelum	Tambah Speed	Setelah Update	Pantul?	Kecepatan Baru
1	(495, 295)	+ (2, 3)	(497, 298)	Tidak	(2, 3)
2	(497, 298)	+ (2, 3)	(499, 301)	y > 300 → yspeed *= -1	(2, -3)
3	(499, 301)	+ (2, -3)	(501, 298)	x > 500 → xspeed *= -1	(-2, -3)
4	(501, 298)	+ (-2, -3)	(499, 295)	Tidak	(-2, -3)
5	(499, 295)	+ (-2, -3)	(497, 292)	Tidak	(-2, -3)

---

2. Dengan Vektor (Cara Modern)

```
# SCRIPT 3 - Simulasi Pergerakan Titik 2D dengan Kelas Vektor
# ------------------------------------------
# Simulasi pergerakan titik 2D menggunakan kelas Vector2D
# Titik bergerak berdasarkan vektor kecepatan
# ------------------------------------------

# Kelas Vector2D untuk merepresentasikan posisi atau kecepatan 2D
class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    # Fungsi untuk menambahkan vektor lain ke vektor ini (penjumlahan vektor)
    def add(self, other):
        self.x += other.x
        self.y += other.y

    # Fungsi untuk mengembalikan posisi sebagai tuple
    def get_position(self):
        return (self.x, self.y)

# Inisialisasi posisi awal dan kecepatan sebagai vektor
position = Vector2D(100, 100)
velocity = Vector2D(2, 3)

# Cetak informasi awal
print(f"Posisi awal: ({position.x}, {position.y})")
print(f"Kecepatan per langkah: ({velocity.x}, {velocity.y})\n")

# Simpan semua koordinat setelah tiap langkah
positions = []

# Lakukan 10 kali update posisi
```

```
for i in range(10):
    position.add(velocity)  # tambahkan kecepatan ke posisi
    positions.append(position.get_position())  # simpan hasil posisi

# Cetak hasil
print("10 Koordinat hasil gerakan:")
for i, (px, py) in enumerate(positions):
    print(f"Langkah {i+1}: posisi = ({px}, {py})")
```

Posisi awal: (100, 100)
Kecepatan per langkah: (2, 3)

10 Koordinat hasil gerakan:
Langkah 1: posisi = (102, 103)
Langkah 2: posisi = (104, 106)
Langkah 3: posisi = (106, 109)

Kita buat kelas Vector2D sederhana dulu:

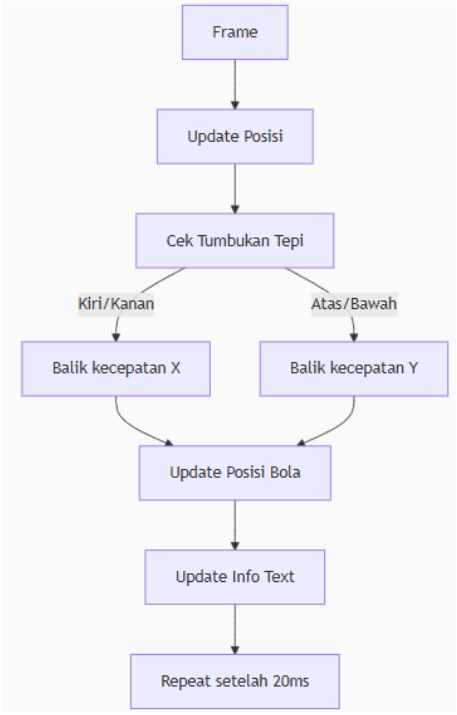
```
class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def add(self, other):
        self.x += other.x
        self.y += other.y
```

untuk lebih memudahkan dalam script kedepan simpan vector.py terpisah sehingga bisa di import

```
#vector.py
import math

class Vector:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    # ---------- Penambahan ----------
    def add(self, other):
        self.x += other.x
        self.y += other.y
        return self  # untuk chaining

    def added(self, other):
        return Vector(self.x + other.x, self.y + other.y)

    # ---------- Pengurangan ----------
    def sub(self, other):
        self.x -= other.x
        self.y -= other.y
        return self

    def subbed(self, other):
        return Vector(self.x - other.x, self.y - other.y)

    # ---------- Perkalian dengan skalar ----------
    def mult(self, scalar):
        self.x *= scalar
        self.y *= scalar
        return self
```

```
    def multed(self, scalar):
        return Vector(self.x * scalar, self.y * scalar)

    # ---------- Pembagian dengan skalar ----------
    def div(self, scalar):
        if scalar != 0:
            self.x /= scalar
            self.y /= scalar
        return self

    def dived(self, scalar):
        if scalar != 0:
            return Vector(self.x / scalar, self.y / scalar)
        return self.copy()

    # ---------- Magnitudo ----------
    def mag(self):
        return math.sqrt(self.x**2 + self.y**2)

    def magnitude(self):
        # Menghitung panjang vektor (kecepatan total)
        return math.sqrt(self.x**2 + self.y**2)
    # ---------- Normalisasi ----------
    def normalize(self):
        m = self.mag()
        if m != 0:
            self.div(m)
        return self

    def normalized(self):
        m = self.mag()
        if m != 0:
            return self.copy().div(m)
        return self.copy()

    # ---------- Limit magnitude ----------
    def limit(self, max_val):
        if self.mag() > max_val:
            self.normalize()
            self.mult(max_val)
        return self

    def limited(self, max_val):
        v = self.copy()
        if v.mag() > max_val:
            v.normalize().mult(max_val)
        return v

    # ---------- Set magnitude ----------
    def setMag(self, mag):
        self.normalize()
        self.mult(mag)
        return self

    def settedMag(self, mag):
        return self.normalized().mult(mag)

    # ---------- Heading (sudut) ----------
    def heading(self):
        if self.x == 0 and self.y == 0:
```

```
            return 0
        return math.atan2(self.y, self.x)

    def heading_rad(self):
        if self.x == 0 and self.y == 0:
            return 0
        return math.atan2(self.y, self.x)

    def heading_deg(self):
        if self.x == 0 and self.y == 0:
            return 0
        return math.degrees(self.heading_rad())  # arah dalam derajat

    # ---------- Salin vektor ----------
    def copy(self):
        return Vector(self.x, self.y)

    # ---------- Ambil posisi ----------
    def set(self, x, y):
        self.x = x
        self.y = y

    def get_position(self):
        return (self.x, self.y)

    def __repr__(self):
        return f"Vector({self.x:.2f}, {self.y:.2f})"

    def __str__(self):
        return f"({self.x:.2f}, {self.y:.2f})"
```



script bola memantul menggunakan kelas Vector:

🌟 **Alur Program Utama**

1. **Inisialisasi**:
   - Posisi awal bola: (100, 100)
   - Kecepatan awal: (2, 3) pixel per frame
   - Ukuran bola: radius 10px
   - Ukuran canvas: 500x300px
2. **Loop Animasi (50 FPS)**:

Diagram

📐 **Rumus Fisika Sederhana**

1. **Gerakan Bola**:

posisi_baru = posisi_lama + kecepatan

2. **Tumbukan Elastis Sempurna**:

jika sentuh tepi:
  kecepatan = -kecepatan *# Membalik arah*

🎞️ **Contoh Perhitungan Frame-by-Frame**

Frame 0:
Posisi: (100, 100)
Kecepatan: (2, 3)
Frame 1:
Posisi: (100+2, 100+3) = (102, 103)
Belum sentuh tepi → tidak ada pantulan
Frame 2:
Posisi: (102+2, 103+3) = (104, 106)
... (terus bergerak sampai menyentuh tepi)

Frame N (Saat Sentuh Tepi Kanan):
Misal posisi x = 490 (canvas_width - radius)
Posisi: (490, y)
Cek: 490 + 10 >= 500? YA → balik kecepatan x
Kecepatan baru: (-2, 3)
Frame N+1:
Posisi: (490-2, y+3) = (488, y+3)

📊 **Visualisasi Tumbukan**

Sebelum tumbukan:
  ○→ (bergerak ke kanan)

Setelah tumbukan:
←○ (bergerak ke kiri)

⏱️ **Karakteristik Waktu**

- Refresh rate: 50 FPS (setiap 20ms)
- Kecepatan: 2-3 pixel per frame
- Waktu mencapai tepi:
    - Horizontal: (500-100)/2 ≈ 200 frame (4 detik)
    - Vertikal: (300-100)/3 ≈ 67 frame (1.3 detik)

💻 **Komponen Kode Penting**

1. **Kelas Vector**:
    - Menyederhanakan operasi matematika
    - Method add() untuk update posisi
2. **Fungsi** update():
    - Inti animasi
    - Meng-handle logika fisika dan rendering
3. **Tampilan Info**:
    - Menampilkan posisi dan kecepatan real-time
    - Format teks: Position: (x, y)\nVelocity: (vx, vy)

🎮 **Interaksi Fisika**

1. **Koefisien Restitusi**:
    - Script ini menggunakan 1 (tumbukan elastis sempurna)
    - Untuk efek lebih realistis bisa dimodifikasi:

velocity.x *= -0.9 *# Kehilangan 10% energi saat tumbukan*

2. **Gravitasi**:
    - Bisa ditambahkan untuk efek lebih menarik:

acceleration = Vector(0, 0.1)
velocity.add(acceleration)

🛠️ **Modifikasi Menarik**

 # Ubah warna bola acak saat memantul
if terjadi_tumbukan:
  canvas.itemconfig(ball,
fill=f"#{random.randint(0,255):02x}{random.randint(0,255):02x}{random.randint(0,255):02x}")

 # Tambahkan percepatan
velocity.x *= 1.01 *# Perlahan tambah kecepatan*

Script ini cocok untuk:

- Pembelajaran fisika dasar
- Konsep game sederhana
- Animasi dasar komputer grafis

Dengan memahami alur ini, Anda bisa mengembangkannya menjadi simulasi yang lebih kompleks!

```
# SCRIPT 4 - Bola Memantul (Dengan Kelas Vektor)
```

```
import tkinter as tk

# 1. Vector2D lebih sederhana disimpan di file terpisah, jadi kita tidak perlu
mendefinisikannya lagi di sini.
from vector import Vector

# 2. Inisialisasi posisi dan kecepatan
position = Vector(100, 100)
velocity = Vector(2, 3)

# Ukuran kanvas dan bola
canvas_width = 500
canvas_height = 300
ball_radius = 10

# Setup Tkinter
window = tk.Tk()
window.title("Bola Memantul (Dengan coords())")
canvas = tk.Canvas(window, width=canvas_width, height=canvas_height, bg="white")
canvas.pack()

# Buat objek bola dan teks 1x
ball = canvas.create_oval(position.x - ball_radius, position.y - ball_radius,
                          position.x + ball_radius, position.y + ball_radius,
                          fill="green")
# Buat teks untuk menampilkan informasi posisi dan kecepatan
info_text = canvas.create_text(10, 10, anchor="nw", font=("Arial", 12), text="")

# 3. Fungsi update animasi
def update():
    # Global variabel untuk akses di dalam fungsi
    global position, velocity

    # Update posisi
    position.add(velocity)

    # Memantul di tepi
    # Jika bola mencapai pinggir kiri/kanan atau atas/bawah, kecepatan dibalik
    if position.x - ball_radius <= 0 or position.x + ball_radius >= canvas_width:
        velocity.x *= -1
    if position.y - ball_radius <= 0 or position.y + ball_radius >= canvas_height:
        velocity.y *= -1

    # Update posisi bola (pakai coords)
    canvas.coords(ball,
                  position.x - ball_radius, position.y - ball_radius,
                  position.x + ball_radius, position.y + ball_radius)

    # Update info kecepatan dan posisi
    canvas.itemconfig(info_text, text=f"Position: ({position.x:.1f},
{position.y:.1f})\n"
                                      f"Velocity: ({velocity.x:.1f},
{velocity.y:.1f})")
    # Panggil update lagi setelah 20ms
    window.after(20, update)

#4. Jalankan animasi
update()
window.mainloop()
```

Manfaat

Dengan pendekatan ini:

- Belajar **mengelompokkan data** dalam bentuk objek (OOP).
- Paham **mengapa vektor memudahkan pengkodean gerak**.
- Melihat bagaimana vektor digunakan di dunia nyata seperti game, simulasi, dan animasi.

---

1. Tanpa Vektor (Cara Lama)

Bayangkan kita punya bola di layar dengan:

- Posisi: x dan y (misal x=100, y=100)
- Kecepatan: xspeed dan yspeed (misal xspeed=2, yspeed=3)

Setiap kali program berjalan:

1. Hapus layar
2. Pindahkan bola: x ditambah xspeed, y ditambah yspeed
3. Jika bola menyentuh tepi, balikkan arahnya (xspeed atau yspeed dikali -1)
4. Gambar bola di posisi baru
5. Ulangi terus setiap 20 milidetik

Kelemahan: Harus mengurus x dan y terpisah, tidak rapi.

2. Dengan Vektor (Cara Modern)

Kita buat "kotak ajaib" bernama Vector2D yang menyimpan:

- x (posisi/kecepatan horizontal)
- y (posisi/kecepatan vertikal)

Ada 2 vector utama:

1. position = posisi bola (x,y)
2. velocity = kecepatan bola (xspeed,yspeed)

Cara kerjanya:

1. Setiap frame, tambahkan velocity ke position (otomatis x dan y diupdate)
2. Jika sentuh tepi, balikkan komponen x atau y dari velocity
3. Gambar bola di posisi baru

Keuntungan:

- Kode lebih rapi dan terorganisir
- Mirip cara kerja di dunia nyata (posisi dan kecepatan sebagai satu kesatuan)
- Lebih mudah dikembangkan untuk fitur tambahan

Perbedaan utama:

- Tanpa vektor: atur x dan y terpisah
- Dengan vektor: x dan y dikelompokkan dalam satu objek Vector2D

Chapter 1.2: Vectors for Processing Programmers

Bagus! Kita lanjut ke **Chapter 1.2: Vectors for Processing Programmers**, dan kita terjemahkan penjelasannya **secara sederhana** , lalu buat **contoh Python Tkinter** agar mereka bisa melihat langsung bagaimana vektor digunakan dalam gerakan.

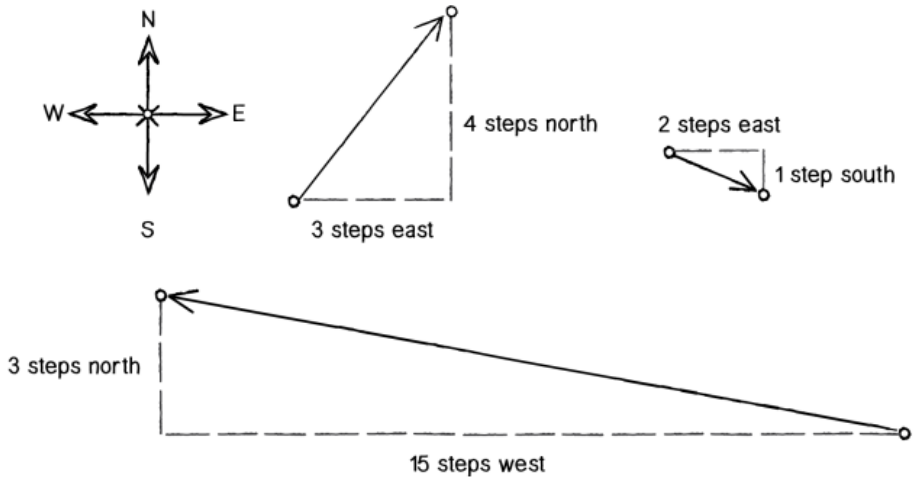


Figure 1.2

(-15, 3)	*Walk fifteen steps west; turn and walk three steps north.*
(3, 4)	*Walk three steps east; turn and walk five steps north.*
(2, -1)	*Walk two steps east; turn and walk one step south.*

🎯 **Ringkasan Materi**

📌 **Inti Konsep:**

- **Vektor bisa dianggap sebagai arah dan jarak dari satu titik ke titik lain.**
  - Contoh: (-15, 3) berarti 15 langkah ke barat dan 3 langkah ke utara.
- Dalam simulasi gerak, kita **menggeser posisi suatu objek** menggunakan vektor velocity.
- Posisi (location) juga bisa dianggap sebagai **vektor dari titik asal (0, 0)** ke titik sekarang.

🔁 **Gerakan dalam animasi/frame:**

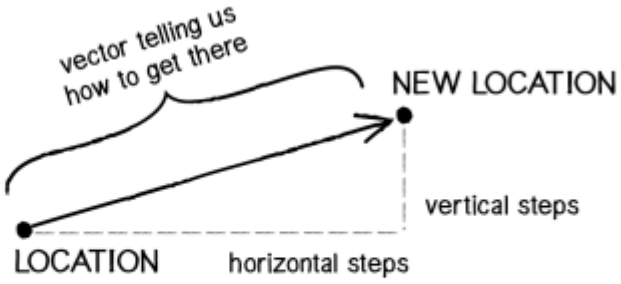


Figure 1.3

For every frame:

location = location + velocity

Artinya, setiap frame:

- Posisi bola bertambah berdasarkan kecepatannya.

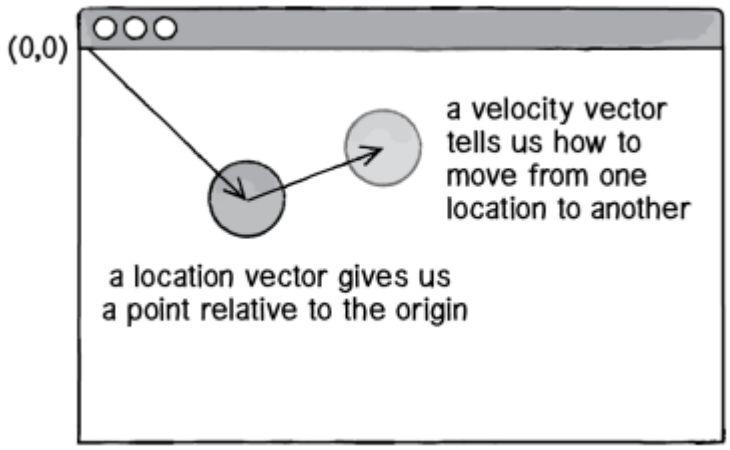
🧱 **PVector (atau Vector2D di Python) hanyalah kelas yang menyimpan x dan y, serta menyediakan fungsi-fungsi seperti penjumlahan.**

🧠 **Penjelasan**

"Bayangkan kamu berjalan dari rumah ke sekolah. Rumahmu adalah titik asal (0,0). Kalau sekolahmu ada di (5, 2), maka vektormu adalah 5 langkah ke kanan dan 2 langkah ke atas. Itu adalah **vektor posisi**.
Nah, kalau kamu jalan 1 langkah ke kanan dan 1 langkah ke atas tiap detik, maka itu adalah **vektor kecepatan**. Jadi, posisi kamu akan terus bertambah berdasarkan kecepatan."

👨‍💻 Implementasi Python Tkinter: Bola Bergerak dengan Vektor

Kita akan lanjut dari contoh sebelumnya dengan **kelas Vector2D yang mendukung penjumlahan**:



1. Membuat kelas Vector2D yang mirip PVector

```
class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def add(self, other):
        self.x += other.x
        self.y += other.y

    def copy(self):
        return Vector2D(self.x, self.y)
```

Tentu! Mari kita jelaskan kode class Vector2D dengan **cara sederhana**, seperti kamu sedang belajar tentang gerak benda di ruang 2 dimensi (misalnya posisi titik di kertas).

📦 Apa itu Vector2D?

Bayangkan kamu punya **anak panah** yang menunjuk ke suatu arah di bidang datar (seperti peta atau kertas). Anak panah ini bisa digambarkan dengan dua angka:

- x: seberapa jauh ke **kanan** (atau kiri jika negatif)
- y: seberapa jauh ke **atas** (atau bawah jika negatif)

Nah, Vector2D adalah **cara menyimpan dan menghitung arah & posisi** anak panah itu.

📦 Penjelasan Kode Baris per Baris

class Vector2D:

➡️ Kita membuat sebuah **kelas** (seperti cetakan) bernama Vector2D.
Ini adalah jenis data baru untuk menyimpan dan memproses arah/gerakan.

```
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

➡️ Ini adalah **fungsi khusus saat objek baru dibuat**.
Misalnya:
`v = Vector2D(3, 4)`
Artinya, kita membuat **vektor baru** yang menunjuk ke kanan sejauh 3 dan ke atas sejauh 4.

```
    def add(self, other):
        self.x += other.x
```

```
        self.y += other.y
```

➡️ Fungsi ini untuk **menambahkan vektor lain ke vektor kita**.

Misalnya:

```
a = Vector2D(3, 4)
b = Vector2D(1, 2)
a.add(b)  # Sekarang a menjadi (4, 6)
```

Artinya, **panah a bergeser lagi** sejauh panah b.

```
    def copy(self):
        return Vector2D(self.x, self.y)
```

➡️ Fungsi ini untuk **membuat salinan vektor**.

Kadang kita ingin **mengubah vektor baru tanpa merusak yang lama**.

Misalnya:

```
a = Vector2D(2, 3)
b = a.copy()  # b juga (2, 3)
b.add(Vector2D(1, 1))  # b jadi (3, 4), tapi a tetap (2, 3)
```

📌 **Kesimpulan**

- Vector2D adalah cara untuk menyimpan **arah dan posisi**.
- Kita bisa:
    - **membuat vektor baru**
    - **menambahkan vektor lain**
    - **menyalin vektor supaya aman diedit**

Kalau kamu suka game atau animasi, Vector2D ini **sangat penting** karena dipakai untuk posisi karakter, kecepatan gerak, arah tembakan, dan lain-lain.

```
# SCRIPT 5 -
# ------------------------------------------
# Simulasi pergerakan titik 2D dengan Vector2D dan .copy()
# Titik bergerak berdasarkan kecepatan, selama 10 langkah
# ------------------------------------------

from vector import Vector

# 2. Inisialisasi posisi dan kecepatan
location = Vector(100, 100)
velocity = Vector(2, 3)

# Cetak kondisi awal
print(f"Posisi awal: {location}")
print(f"Kecepatan per langkah: {velocity}\n")

# List untuk menyimpan posisi setiap langkah
positions = []

# 3. Simulasikan 10 langkah pergerakan
for i in range(10):
    location.add(velocity)                # Update posisi
    positions.append(location.copy())     # Simpan salinan posisi saat ini

# 4. Cetak semua posisi setelah 10 langkah
print("10 Koordinat hasil gerakan:")
for i, pos in enumerate(positions):
    print(f"Langkah {i+1}: posisi = {pos}")
```

Posisi awal: (100, 100)

Kecepatan per langkah: (2, 3)

10 Koordinat hasil gerakan:
Langkah 1: posisi = (102, 103)
Langkah 2: posisi = (104, 106)
Langkah 3: posisi = (106, 109)
Kenapa .copy() penting?
Karena kalau kamu hanya menyimpan positions.append(location), semua elemen akan menunjuk ke **objek yang sama**, dan hasil akhirnya hanya posisi terakhir diulang 10 kali. Dengan .copy(), kita menyimpan **salinan posisi pada setiap langkah**.

Bab 1.3: Penjumlahan Vektor (Vector Addition)

Bagus! Kita lanjut ke **Bab 1.3: Penjumlahan Vektor (Vector Addition)** dan implementasinya dalam **Python Tkinter**

🧠 Penjelasan Konsep: Vector Addition

📌 Apa itu penjumlahan vektor?

Misal kita punya dua vektor:

- vektor **u** = (5, 3)
- vektor **v** = (3, 3)

Kalau kita jumlahkan:

- **w = u + v = (5 + 3, 3 + 3) = (8, 6)**

Artinya: kalau kamu bergerak 5 langkah kanan dan 3 atas, lalu lanjut 3 langkah kanan dan 3 atas lagi, maka total gerakanmu = 8 kanan dan 6 atas.

📐 Dalam pemrograman:

Di Processing (dan nanti Python), penjumlahan vektor dilakukan seperti ini:

```
location.add(velocity);
```

Yang secara manual artinya:

```
location.x = location.x + velocity.x;
location.y = location.y + velocity.y;
```

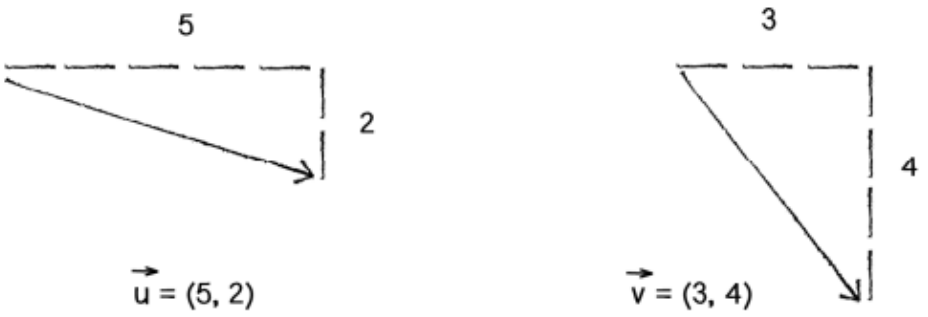


Figure 1.5

Each vector has two components, an x and a y. To add two vectors together, we simply add both x's and both y's.

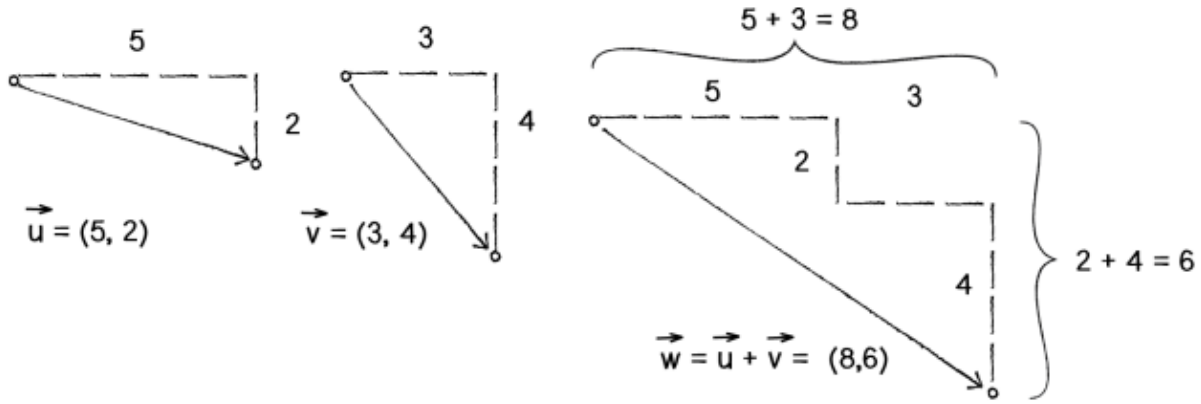


Figure 1.6

In other words:

$\vec{w} = \vec{u} + \vec{v}$

can be written as:

$w_x = u_x + v_x$
$w_y = u_y + v_y$

👦🏠 **Penjelasan**
"Vektor itu seperti arah dan jarak. Kalau kamu jalan 2 langkah ke kanan dan 1 ke atas, lalu jalan lagi 3 ke kanan dan 4 ke atas, maka kamu sudah jalan total 5 ke kanan dan 5 ke atas. Itu yang disebut penjumlahan vektor."

👦💻 **Implementasi dalam Python Tkinter**
Kita sudah punya kelas Vector2D di bab sebelumnya, sekarang kita akan *menekankan konsep penjumlahan vektor* dan implementasi add() secara eksplisit.
simulasi penjumlahan vektor menggunakan tkinter:

🎯 **Tujuan Simulasi**
Menampilkan **vektor A + B = C** dengan:

- Vektor **A** (dari pusat ke mouse X, 0)
- Vektor **B** (dari mouse X, 0 ke mouse X, Y)

- Vektor **C** (hasil penjumlahan A + B = pusat ke mouse X,Y)
- Ditampilkan panah warna berbeda dan label hasil penjumlahan
- Disertai **perhitungan numerik dan magnitude**

Warna	Vektor
🔵 Biru	Vektor A (horizontal)
🟢 Hijau	Vektor B (vertikal)
🔴 Merah	Hasil penjumlahan C = A + B

```
# SCRIPT 6 - # Simulasi Penjumlahan Titik 2D center (0,0) dengan Mouse
import tkinter as tk
from vector import Vector

# ===== Setup Tkinter =====
WIDTH, HEIGHT = 600, 400
window = tk.Tk()
window.title("Simulasi Penjumlahan Vektor")
canvas = tk.Canvas(window, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()
info_label = tk.Label(window, text="", font=("Arial", 12), justify="left")
info_label.pack()

center_x, center_y = WIDTH // 2, HEIGHT // 2

# ===== Update Fungsi Utama =====
def update(event):
    canvas.delete("all")

    # Vektor A: dari tengah ke mouse secara horizontal (X saja)
    A = Vector(event.x - center_x, 0)

    # Vektor B: dari ujung A ke posisi mouse secara vertikal (Y saja)
    B = Vector(0, event.y - center_y)

    # Vektor C = A + B
    C = A.added(B)

    # ===== Gambar Panah =====
    canvas.create_line(center_x, center_y,
                       center_x + A.x, center_y + A.y,
                       arrow=tk.LAST, fill="blue", width=2)

    canvas.create_line(center_x + A.x, center_y + A.y,
                       center_x + A.x + B.x, center_y + A.y + B.y,
                       arrow=tk.LAST, fill="green", width=2)

    canvas.create_line(center_x, center_y,
                       center_x + C.x, center_y + C.y,
                       arrow=tk.LAST, fill="red", width=3)

    # Tampilkan informasi vektor
    info = (
        f"Vektor A (horizontal): {A}\n"
        f"Vektor B (vertikal)  : {B}\n"
        f"Hasil A + B = C      : {C}\n"
        f"Magnitude C (|C|)    : {C.mag():.2f} px"
    )
    info_label.config(text=info)
```

```
    # Gambar titik akhir
    r = 4
    canvas.create_oval(center_x + C.x - r, center_y + C.y - r,
                       center_x + C.x + r, center_y + C.y + r, fill="black")

# ===== Bind Mouse Movement =====
canvas.bind("<Motion>", update)

# ===== Start Tkinter Loop =====
window.mainloop()
```

1.4: More Vector Math

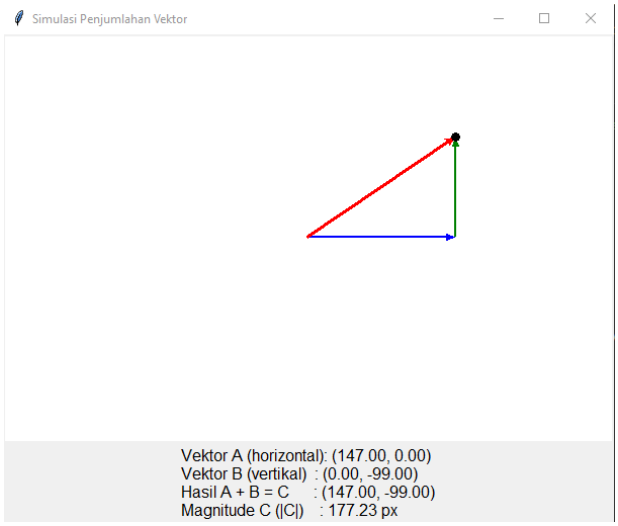
Bagus! Kita sekarang masuk ke **materi 1.4: More Vector Math**, yang membahas operasi-operasi matematika lain pada vektor. Konsep ini penting untuk simulasi gerakan objek di dunia nyata seperti gravitasi, dorongan, atau arah.

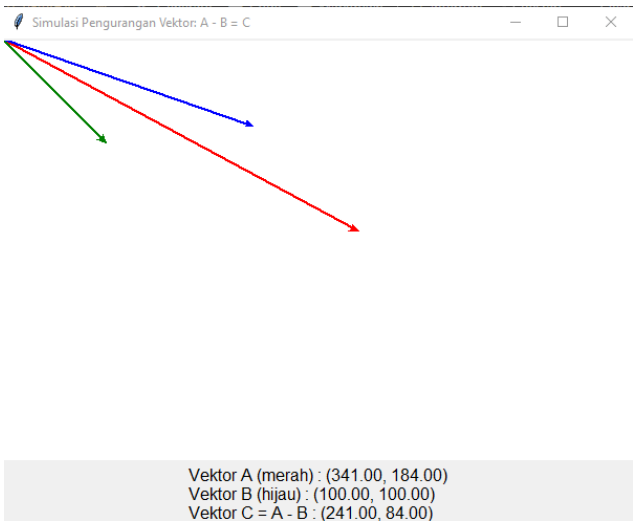
🎓 Konsep Dasar Operasi Vektor (Vector Math)

Sama seperti kita bisa menambah bilangan, kita juga bisa **menambah, mengurangi, mengalikan, membagi, atau mencari panjang (magnitude)** dari vektor.

✨ Fungsi Umum pada Vector2D (mirip PVector di Processing)

Fungsi	Penjelasan Singkat	Kegunaan dalam Simulasi
add(v)	Menambahkan vektor v ke vektor ini	Menggerakkan posisi objek sesuai kecepatan (misalnya: posisi.add(kecepatan))
sub(v)	Mengurangi vektor v dari vektor ini	Menentukan arah dari satu objek ke objek lain (contoh: arah = target.sub(posisi))
mult(s)	Kalikan vektor dengan skalar s	Mempercepat atau memperlambat gerakan (contoh: gaya.mult(0.5))
div(s)	Membagi vektor dengan skalar s	Menormalkan atau mengurangi pengaruh (misalnya: gaya.div(massa))
mag()	Menghitung panjang (magnitude) dari vektor	Mengetahui kecepatan total atau jarak tempuh dari vektor
normalize()	Ubah ke vektor satuan (panjang = 1)	Menentukan arah saja tanpa memperhitungkan besar (untuk panah arah, arah gaya, dll)
limit(max)	Membatasi panjang maksimum dari vektor	Mencegah kecepatan terlalu besar, menjaga agar gerakan tetap stabil
heading()	Mendapatkan arah vektor dalam derajat	Menentukan arah panah atau rotasi benda berdasarkan arah gerak
dist(v)	Mengukur jarak antara dua vektor	Mengukur jarak antara dua objek, misalnya untuk tabrakan atau gravitasi





Vector subtraction

$$\vec{w} = \vec{u} - \vec{v}$$

can be written as:

$$w_x = u_x - v_x$$
$$w_y = u_y - v_y$$

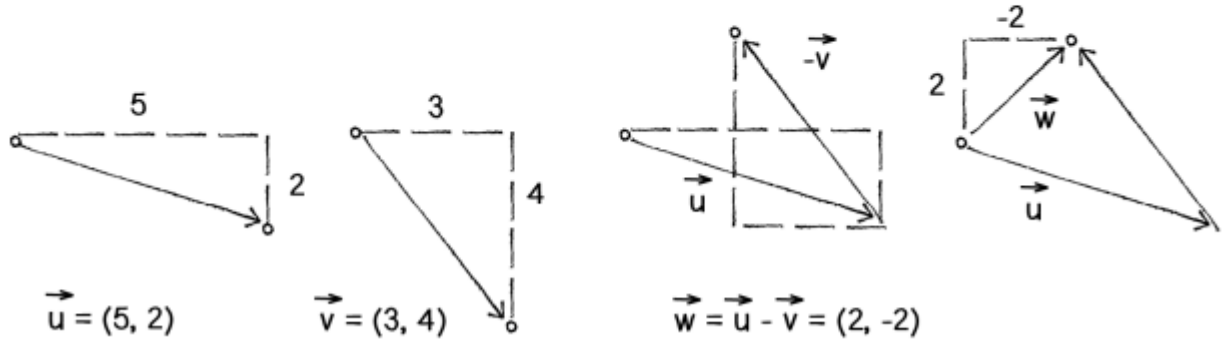


Figure 1.7: Vector Subtraction

🧮 Contoh Implementasi: Vector2D Class di Python

Kita buat versi sederhananya di Python:
simulasi pengurangan vektor (Vektor C = A - B) dalam Tkinter, lengkap dengan panah, perhitungan, dan magnitude.

🎯 Tujuan Simulasi

Menampilkan vektor:

- **A** = dari pusat (0,0) ke mouse (X, Y)
- **B** = vektor acuan tetap (contoh: kanan 100, atas 50)
- **C = A - B**
- Gambar panah vektor A, B, dan hasil pengurangan C
- Tampilkan hasil hitung dan magnitude

Warna	Vektor
🔵 Biru	A = dari pusat ke mouse
🟢 Hijau	B = vektor tetap
🔴 Merah	C = A - B (hasil)

```
# SCRIPT 7 - # Simulasi Pengurangan Vektor 2D: A - B = C
import tkinter as tk
import math

# Class Vector2D
from vector import Vector

# Setup window
WIDTH = 600
HEIGHT = 400
window = tk.Tk()
window.title("Simulasi Pengurangan Vektor: A - B = C")
canvas = tk.Canvas(window, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

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

# Titik origin di tengah
#origin_x, origin_y = WIDTH // 2, HEIGHT // 2
origin_x, origin_y = 0, 0

# Panah vektor
arrow_a = canvas.create_line(0, 0, 0, 0, arrow=tk.LAST, fill="red", width=2)
arrow_b = canvas.create_line(0, 0, 0, 0, arrow=tk.LAST, fill="green", width=2)
arrow_c = canvas.create_line(0, 0, 0, 0, arrow=tk.LAST, fill="blue", width=2)

# Fungsi update saat mouse digerakkan
def update(event):
    # Vektor A: dari origin ke mouse
    a = Vector(event.x - origin_x, event.y - origin_y)

    # Vektor B: tetap, misalnya arah kanan atas
    b = Vector(100, 100)

    # Vektor C = A - B
    c = a.copy()
    c.sub(b)

    # Koordinat ujung A
    ax = origin_x + a.x
    ay = origin_y + a.y

    # Koordinat ujung B (dari origin)
    bx = origin_x + b.x
    by = origin_y + b.y

    # Koordinat ujung C = A - B
    cx = origin_x + c.x
    cy = origin_y + c.y

    # Gambar panah A
    canvas.coords(arrow_a, origin_x, origin_y, ax, ay)
    # Gambar panah B
    canvas.coords(arrow_b, origin_x, origin_y, bx, by)
    # Gambar panah C
    canvas.coords(arrow_c, origin_x, origin_y, cx, cy)

    # Update teks info
    info_text = (
```

```
        f"Vektor A (merah) : {a}\n"
        f"Vektor B (hijau) : {b}\n"
        f"Vektor C = A - B : {c}"
    )
    info.config(text=info_text)

# Bind mouse movement
canvas.bind("<Motion>", update)

# Jalankan GUI
window.mainloop()
```

SCRIPT 8 - operasi-operasi vektor ini digunakan dalam berbagai konteks fisika, animasi, atau robotika

```
import math

# 1. Kelas Vector2D: merepresentasikan vektor 2D dengan operasi dasar
class Vector2D:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def add(self, v):
        self.x += v.x
        self.y += v.y

    def sub(self, v):
        self.x -= v.x
        self.y -= v.y

    def mult(self, scalar):
        self.x *= scalar
        self.y *= scalar

    def div(self, scalar):
        if scalar != 0:
            self.x /= scalar
            self.y /= scalar

    def mag(self):
        return math.sqrt(self.x**2 + self.y**2)

    def normalize(self):
        m = self.mag()
        if m != 0:
            self.div(m)

    def limit(self, max_val):
        if self.mag() > max_val:
            self.normalize()
            self.mult(max_val)

    def heading(self):
        return math.atan2(self.y, self.x)

    def copy(self):
        return Vector2D(self.x, self.y)

    def __str__(self):
        return f"({self.x:.2f}, {self.y:.2f})"
```

```
# 1. add(), sub(), mult(), div() — Operasi Dasar Vektor
a = Vector2D(3, 4)
b = Vector2D(1, 2)
c = 3

# Penjumlahan vektor a dan b hasilnya disimpan di a
print("# Operasi Dasar")
a.add(b)   # a = (3+1, 4+2) → (4, 6)
print("(3, 4) + (1, 2) : ", a) #(4.00, 6.00)

# Pengurangan vektor a dan b hasilnya disimpan di a
a.sub(b)   # a = (4-1, 6-2) → (3, 4)
print("(4, 6) - (1, 2) : ", a) # (3.00, 4.00)

# Perkalian vektor a dengan skalar c hasilnya disimpan di a
a.mult(c)  # a = (3*3, 4*3) → (9, 12)
print("(3, 4) * 3 : ", a) # (9.00, 12.00)

# Pembagian vektor a dengan skalar c hasilnya disimpan di a
a.div(c)   # a = (9/3, 12/3) → (3, 4)
print("(9, 12) / 3 : ", a) # (3.00, 4.00)

# 2. mag(), normalize(), limit() — Magnitudo, Normalisasi, dan Pembatasan
print("\n# Magnitudo dan Normalisasi")

# magnitudo gunanya untuk menghitung panjang vektor a
a_mag = a.mag()  # Magnitudo dari (3, 4) → 5.0
print("Magnitudo dari (3, 4) : ", round(a_mag, 2)) # 5.0

# normalisasi vektor a untuk mengubahnya menjadi panjang 1
a.normalize()  # Normalisasi (3, 4) → (0.6, 0.8)
print("Normalisasi dari (3, 4) : ", a) # (0.60, 0.80)

#limit() gunanya untuk membatasi panjang vektor
a.limit(1)  # Batasi vektor ke panjang 1 → (0.6, 0.8)
print("Batasi ke panjang 1 : ", a)

# 3. heading() — Arah Vektor
print("\n# Arah Vektor")
#heading() mengembalikan arah vektor dalam radian
a_heading = a.heading()  # Arah dari (0.6, 0.8) → sekitar 0.93 radian
print("Arah dari (0.6, 0.8) : ", round(a_heading, 2), "radian") # 0.93 radian

a_heading_deg = math.degrees(a_heading)
print("Arah dari (0.6, 0.8) : ", round(a_heading, 2), "radian /", round(a_heading_deg,
2), "derajat") #Arah dari (0.6, 0.8) :  0.93 radian / 53.13 derajat

# 4. copy() — Membuat Salinan Vektor
print("\n# Salinan Vektor")
a_copy = a.copy()  # Membuat salinan dari (0.6, 0.8)
print("Salinan dari (0.6, 0.8) : ", a_copy)

# 5. __str__() — Representasi String Vektor
print("\n# Representasi String")
print("Representasi string dari (0.6, 0.8) : ", str(a))
```

```
 # Operasi Dasar
(3, 4) + (1, 2) :  (4.00, 6.00)
(4, 6) - (1, 2) :  (3.00, 4.00)
(3, 4) * 3 :  (9.00, 12.00)
```

(9, 12) / 3 : (3.00, 4.00)

Magnitudo dan Normalisasi
Magnitudo dari (3, 4) : 5.0
Normalisasi dari (3, 4) : (0.60, 0.80)
Batasi ke panjang 1 : (0.60, 0.80)

Arah Vektor
Arah dari (0.6, 0.8) : 0.93 radian

Salinan Vektor
Salinan dari (0.6, 0.8) : (0.60, 0.80)

Representasi String
Representasi string dari (0.6, 0.8) : (0.60, 0.80)

📌 **Penutup Konsep: Properti Aritmatika**
Vektor juga mengikuti **aturan aljabar** yang biasa:

- **Komutatif**: A + B = B + A
- **Asosiatif**: (A + B) + C = A + (B + C)

Contoh cepat:
A = Vector2D(1, 2)
B = Vector2D(3, 4)
C = Vector2D(5, 6)

sum1 = A.copy() # Buat salinan A → (1, 2)
sum1.add(B)   # Tambah B → (1+3, 2+4) = (4, 6)
sum1.add(C)   # Tambah C → (4+5, 6+6) = (9, 12)

sum2 = B.copy() # Buat salinan B → (3, 4)
sum2.add(C)   # Tambah C → (3+5, 4+6) = (8, 10)
sum2.add(A)   # Tambah A → (8+1, 10+2) = (9, 12)

print(sum1) # (9, 12)
print(sum2) # (9, 12)

Vector Multiplication dan Division

Bagus, kita sekarang masuk ke materi **Vector Multiplication dan Division** dalam konteks **PVector** dari Processing, lalu kita implementasikan dalam **Python dan Tkinter** menggunakan class Vector2D.

🧠 **Konsep Inti: Perkalian dan Pembagian Vektor**
✅ **1. Multiplying a Vector by a Scalar (Skalar)**
Jika kamu punya:
u = (-3, 7)
n = 3
w = u * n = (-9, 21)
Artinya setiap komponen (x, y) dikali dengan n.

$\vec{w} = \vec{u} * n$

can be written as:

$w_x = u_x * n$
$w_y = u_y * n$

Let's look at an example with vector notation.

$\vec{u} = (-3, 7)$
$n = 3$

$\vec{w} = \vec{u} * n$
$w_x = -3 * 3$
$w_y = 7 * 3$

$\vec{w} = (-9, 21)$

Therefore, the function inside the PVector class is written as:

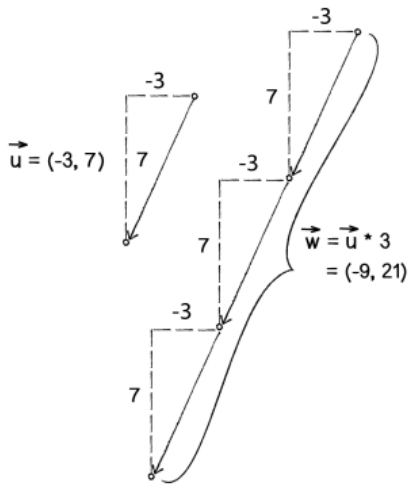


Figure 1.8: Scaling a vector

```
void mult(float n) {
  x = x * n;
  y = y * n;
}
```

With multiplication, the components of the vector are multiplied by a number.

✅ **2. Dividing a Vector by a Scalar**

Jika kamu punya:
u = (8, -4)
n = 2
w = u / n = (4, -2)

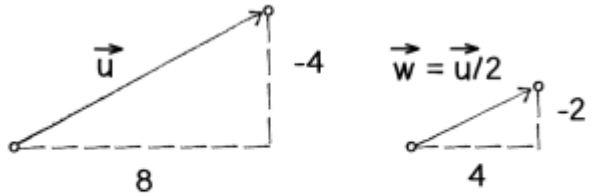


Figure 1.9

🛠️ **Implementasi dalam Python (OOP Style)**

Kita akan update class Vector2D agar punya metode:

- .mult(scalar)
- .div(scalar)

Lalu kita akan buat demo vektor dari tengah ke mouse, lalu **dikalikan dengan 0.5** (setengah panjang aslinya).

🧠 **Penjelasan Sederhana :**

Apa itu Vektor?

Vektor adalah **panah** yang menunjukkan:

- **Arah** ke mana bergerak
- **Seberapa jauh** atau cepat bergerak

Contoh: dari titik tengah (300, 200) ke arah mouse (posisi kursor kamu).

Apa itu Perkalian Skalar?

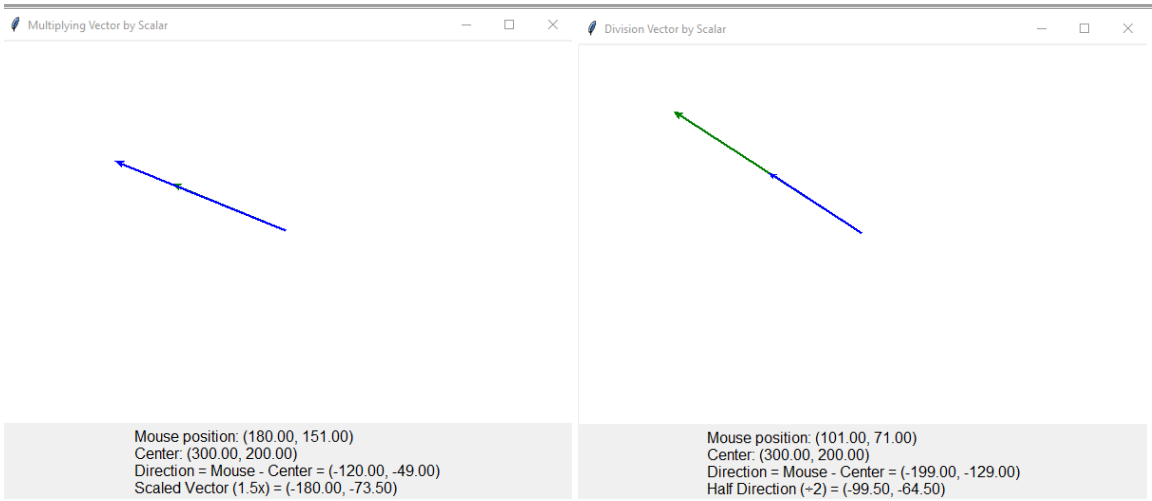
Jika kamu punya vektor arah, dan kamu **kalikan** dengan angka (skalar):

- Kalau dikalikan **0.5**, panahnya jadi **setengah lebih pendek** (lebih dekat).
- Kalau dikalikan **2**, panahnya jadi **dua kali lebih panjang** (lebih jauh).
- Ini seperti memperbesar atau mengecilkan langkah kaki kamu.

Contoh Rumus:
Misalnya:

- Mouse di (400, 300)
- Titik tengah adalah (300, 200)
- Maka:

Vektor asli = (400 - 300, 300 - 200) = (100, 100)
Setengah vektor = 0.5 * (100, 100) = (50, 50)



Script Lengkap dengan Perhitungan Ditampilkan

Tahap	Keterangan
1	Ambil posisi mouse dan hitung arah = mouse - center
2	Gambar vektor asli (hijau) menggunakan coords()
3	Kalikan arah dengan skalar (misal 1.5) → hasil vektor lebih panjang
4	Gambar vektor hasil perkalian (biru) menggunakan coords()
5	Tampilkan nilai vektor dan hasil perhitungan di bawah kanvas

```
# SCRIPT 9 - Perkalian Vektor dengan Skalar
import tkinter as tk
from vector import Vector

# Setup GUI Tkinter
window = tk.Tk()
window.title("Multiplying Vector by Scalar")
canvas_width = 600
canvas_height = 400
canvas = tk.Canvas(window, width=canvas_width, height=canvas_height, bg="white")
canvas.pack()

# Label untuk menampilkan perhitungan
info = tk.Label(window, text="", justify="left", font=("Arial", 12))
info.pack()

# Titik pusat
center = Vector(canvas_width / 2, canvas_height / 2)

# Buat dua panah garis sekali saja (tidak dihapus tiap frame)
```

```
original_vector_id = canvas.create_line(0, 0, 0, 0, arrow=tk.LAST, fill="green",
width=2)  #: Garis vektor asli
scaled_vector_id = canvas.create_line(0, 0, 0, 0, arrow=tk.LAST, fill="blue", width=2)
   #: Garis vektor hasil skalar

# Fungsi update saat mouse bergerak
def update(event=None):
    # Ambil posisi mouse
    mouse = Vector(event.x, event.y)

    # Hitung vektor dari titik tengah ke posisi mouse
    direction = mouse.copy()
    direction.sub(center)  #: Tahap 1: Hitung vektor arah (mouse - center)

    # Gambar vektor asli (hijau)
    canvas.coords(original_vector_id,
                  center.x, center.y,
                  center.x + direction.x,
                  center.y + direction.y)  #: Tahap 2: Gambar arah asli

    # Skala vektor (misal dikali 1.5)
    scaled = direction.copy()
    scaled.mult(1.5)  #: Tahap 3: Kalikan vektor dengan skalar

    # Gambar vektor hasil skalar (biru)
    canvas.coords(scaled_vector_id,
                  center.x, center.y,
                  center.x + scaled.x,
                  center.y + scaled.y)  #: Tahap 4: Gambar arah hasil skalar

    # Tampilkan informasi perhitungan di bawah
    info_text = (
        f"Mouse position: ({mouse.x:.2f}, {mouse.y:.2f})\n"
        f"Center: ({center.x:.2f}, {center.y:.2f})\n"
        f"Direction = Mouse - Center = {direction}\n"
        f"Scaled Vector (1.5x) = {scaled}"
    )
    info.config(text=info_text)  #: Tahap 5: Tampilkan teks perhitungan

# Jalankan update setiap mouse bergerak
canvas.bind("<Motion>", update)

window.mainloop()
```

PEMBAGIAN VECTOR

Tahap	Penjelasan
0	Inisialisasi vektor
1	Buat garis hijau kosong untuk vektor asli
2	Buat garis biru kosong untuk hasil pembagian
3	Hitung vektor dari pusat ke posisi mouse
4	Gambar vektor asli dengan coords()
5	Bagi vektor arah dengan skalar 2
6	Gambar vektor hasil pembagian (½ panjang)
7	Tampilkan informasi vektor dan hasil pembagian

```
# SCRIPT 10 - Pembagian Vektor dengan Skalar
import tkinter as tk
```

```
from vector import Vector

# Setup GUI Tkinter
window = tk.Tk()
window.title("Division Vector by Scalar")
canvas_width = 600
canvas_height = 400
canvas = tk.Canvas(window, width=canvas_width, height=canvas_height, bg="white")
canvas.pack()

# Label untuk menampilkan info perhitungan
info = tk.Label(window, text="", justify="left", font=("Arial", 12))
info.pack()

# Titik tengah layar sebagai pusat vektor
center = Vector(canvas_width / 2, canvas_height / 2)

# Buat dua panah garis hanya sekali (pakai coords nanti)
original_vector_id = canvas.create_line(0, 0, 0, 0, arrow=tk.LAST, fill="green",
width=2)  #: Tahap 1: Garis asli
divided_vector_id = canvas.create_line(0, 0, 0, 0, arrow=tk.LAST, fill="blue", width=2)
   #: Tahap 2: Garis hasil pembagian

# Fungsi update saat mouse bergerak
def update(event=None):
    # Ambil posisi mouse
    mouse = Vector(event.x, event.y)

    # Hitung arah dari center ke mouse
    direction = mouse.copy()
    direction.sub(center)  #: Tahap 3: Hitung vektor arah = mouse - center

    # Gambar vektor asli (hijau)
    canvas.coords(original_vector_id,
                  center.x, center.y,
                  center.x + direction.x,
                  center.y + direction.y)  #: Tahap 4: Perbarui garis asli

    # Bagi vektor dengan skalar (misal 2)
    divided = direction.copy()
    divided.div(2)  #: Tahap 5: Bagi arah dengan 2

    # Gambar hasil pembagian (biru)
    canvas.coords(divided_vector_id,
                  center.x, center.y,
                  center.x + divided.x,
                  center.y + divided.y)  #: Tahap 6: Perbarui garis hasil pembagian

    # Tampilkan informasi hasil perhitungan
    info_text = (
        f"Mouse position: ({mouse.x:.2f}, {mouse.y:.2f})\n"
        f"Center: ({center.x:.2f}, {center.y:.2f})\n"
        f"Direction = Mouse - Center = {direction}\n"
        f"Half Direction (÷2) = {divided}"
    )
    info.config(text=info_text)  #: Tahap 7: Tampilkan teks hasil

# Jalankan update saat mouse bergerak
canvas.bind("<Motion>", update)

# Jalankan aplikasi
```

```
window.mainloop()
```

1.5: Magnitude Vektor (Panjang Vektor)

Bagus! Sekarang kita akan bahas **materi 1.5: Magnitude Vektor (Panjang Vektor)** dengan **penjelasan sederhana**, lalu kita **visualisasikan dengan Python Tkinter**.

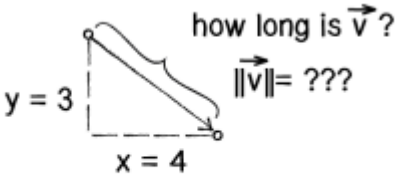


Figure 1.10: The length or "magnitude" of a vector $v\rightarrow$ is often written as: $\|v\rightarrow\|$

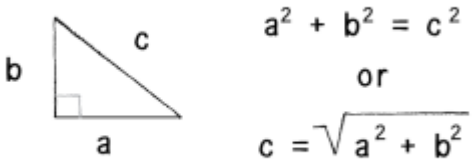


Figure 1.11: The Pythagorean Theorem

🎓 **Penjelasan Sederhana: Apa Itu Magnitude?**

Bayangkan kamu menggambar **panah dari titik tengah ke mouse**. Panah itu punya arah **dan panjang**.

🟢 **Magnitude** artinya:

"Seberapa panjang sih panah (vektor) itu?"

📐 **Menghitung Panjang Panah (Vektor)**

Kalau vektor kamu adalah:

x = 3

y = 4

Maka panjangnya dihitung pakai **Teorema Pythagoras**:

magnitude = √(x² + y²)

= √(3² + 4²)

= √(9 + 16)

= √25 = 5

🛠️ **Fungsi .mag() dalam OOP Python**

Kita tambahkan ke class Vector2D:

```
def mag(self):
  return math.sqrt(self.x ** 2 + self.y ** 2)
```

🎉 **Apa yang Terjadi?**

- Gerakkan mouse — panah dari tengah akan mengikuti.
- Kotak oranye di atas menunjukkan **panjang panah dalam piksel** (hasil dari √(x² + y²)).
- Semakin jauh kamu gerakkan mouse dari tengah, semakin panjang panah dan kotaknya.

🧠 **Penjelasan Konsep:**

Apa Itu Vektor?

Vektor itu seperti **panah** yang menunjukkan:

- **Arah** (ke mana bergerak)
- **Panjang** (seberapa jauh)

Misalnya: dari titik **tengah layar** ke posisi **mouse** kamu.

Apa Itu Pengurangan Vektor?

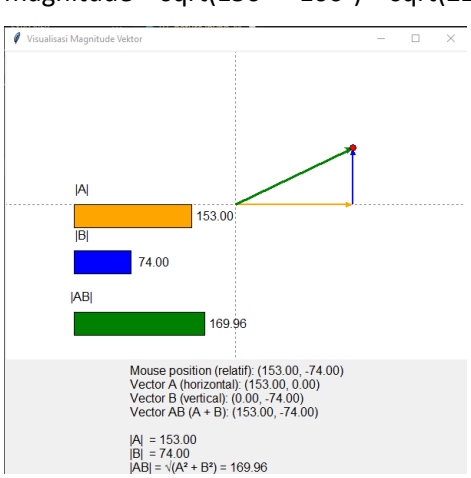
Kita hitung vektor arah dari **titik tengah** ke **mouse** dengan cara:

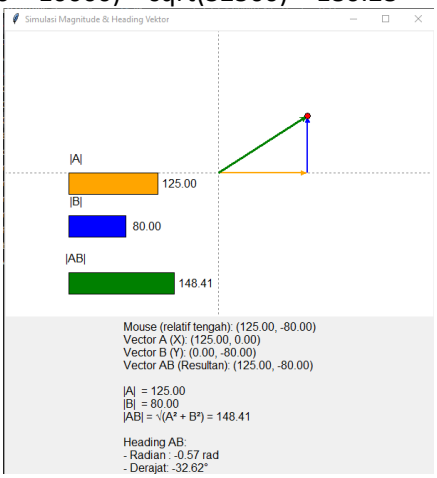
(direction.x, direction.y) = (mouse.x - center.x, mouse.y - center.y)

Contoh:

Mouse: (450, 300)
Center: (300, 200)
Direction: (450 - 300, 300 - 200) = (150, 100)

Apa Itu Magnitude?
Magnitude adalah **panjang vektor** atau **seberapa jauh** dari tengah ke mouse. Kita pakai rumus Pythagoras:
magnitude = sqrt($x^2 + y^2$)
Contoh:
magnitude = sqrt($150^2 + 100^2$) = sqrt(22500 + 10000) = sqrt(32500) ≈ 180.28





✅ **simulasi visual magnitude vektor** menggunakan Python tkinter.
Kode ini menampilkan:
✅ Panah dari titik (0,0) ke posisi mouse (vektor AB)
✅ Panah A (dari (0,0) ke mouse.x, 0)
✅ Panah B (dari mouse.x, 0 ke mouse.x, mouse.y)
✅ Panjang masing-masing vektor A, B, dan AB dalam bentuk **bar graph dan angka**
✅ Origin (0,0) berada di **tengah layar**

🧠 **Penjelasan Singkat**

- **Vector AB**: dari (0,0) ke posisi mouse.
- **Vector A**: horizontal dari (0,0) ke (mouse.x, 0)
- **Vector B**: vertikal dari (mouse.x, 0) ke (mouse.x, mouse.y)
- **Magnitude (|AB|)** dihitung dengan rumus Pythagoras: $|AB| = \sqrt{x^2 + y^2}$

```
# SCRIPT 11 - Visualisasi Magnitude Vektor
import tkinter as tk
from vector import Vector

# ====== Setup Tkinter ======
WIDTH = 600
HEIGHT = 400
window = tk.Tk()
window.title("Visualisasi Magnitude Vektor")
canvas = tk.Canvas(window, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

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

origin_x = WIDTH // 2
origin_y = HEIGHT // 2
```

```
# Bar Graph Parameters
BAR_WIDTH = 30
BAR_SCALE = 1  # skala untuk memperbesar bar
bar_y_pos = HEIGHT - 200

# ====== Update Visual ======
def update(event):
    canvas.delete("all")

    # Titik mouse relatif terhadap origin (tengah)
    mouse = Vector(event.x - origin_x, event.y - origin_y)
    vector_a = Vector(mouse.x, 0)              # A: horizontal dari origin ke (mouse.x,
0)
    vector_b = Vector(0, mouse.y)              # B: vertical dari (mouse.x, 0) ke mouse
    vector_ab = mouse.copy()                     # AB: dari origin ke mouse

    mag_a = abs(vector_a.x)
    mag_b = abs(vector_b.y)
    mag_ab = vector_ab.mag()

    # ====== Draw coordinate system ======
    canvas.create_line(0, origin_y, WIDTH, origin_y, fill="gray", dash=(2, 2))  # sumbu
X
    canvas.create_line(origin_x, 0, origin_x, HEIGHT, fill="gray", dash=(2, 2)) # sumbu
Y

    # ====== Draw vectors ======
    # vector A (horizontal)
    canvas.create_line(origin_x, origin_y,
                       origin_x + vector_a.x, origin_y,
                       arrow=tk.LAST, fill="orange", width=2)

    # vector B (vertical)
    canvas.create_line(origin_x + vector_a.x, origin_y,
                       origin_x + vector_ab.x, origin_y + vector_b.y,
                       arrow=tk.LAST, fill="blue", width=2)

    # vector AB (diagonal)
    canvas.create_line(origin_x, origin_y,
                       origin_x + vector_ab.x, origin_y + vector_ab.y,
                       arrow=tk.LAST, fill="green", width=3)

    # Titik mouse
    canvas.create_oval(origin_x + vector_ab.x - 4, origin_y + vector_ab.y - 4,
                       origin_x + vector_ab.x + 4, origin_y + vector_ab.y + 4,
                       fill="red")

    # ====== Draw Bar Graphs ======
    canvas.create_text(100, bar_y_pos - 20, text="|A|", font=("Arial", 12))
    canvas.create_rectangle(90, bar_y_pos, 90 + mag_a * BAR_SCALE, bar_y_pos +
BAR_WIDTH, fill="orange")
    canvas.create_text(90 + mag_a * BAR_SCALE + 30, bar_y_pos + BAR_WIDTH // 2,
                       text=f"{mag_a:.2f}", font=("Arial", 12))

    canvas.create_text(100, bar_y_pos + 40, text="|B|", font=("Arial", 12))
    canvas.create_rectangle(90, bar_y_pos + 60, 90 + mag_b * BAR_SCALE, bar_y_pos + 60
+ BAR_WIDTH, fill="blue")
    canvas.create_text(90 + mag_b * BAR_SCALE + 30, bar_y_pos + 60 + BAR_WIDTH // 2,
                       text=f"{mag_b:.2f}", font=("Arial", 12))
```

```
    canvas.create_text(100, bar_y_pos + 120, text="|AB|", font=("Arial", 12))
    canvas.create_rectangle(90, bar_y_pos + 140, 90 + mag_ab * BAR_SCALE, bar_y_pos +
140 + BAR_WIDTH, fill="green")
    canvas.create_text(90 + mag_ab * BAR_SCALE + 30, bar_y_pos + 140 + BAR_WIDTH // 2,
                       text=f"{mag_ab:.2f}", font=("Arial", 12))

    # ====== Info Teks ======
    info_text = (
        f"Mouse position (relatif): {vector_ab}\n"
        f"Vector A (horizontal): {vector_a}\n"
        f"Vector B (vertical): {vector_b}\n"
        f"Vector AB (A + B): {vector_ab}\n\n"
        f"|A|  = {mag_a:.2f}\n"
        f"|B|  = {mag_b:.2f}\n"
        f"|AB| = √(A² + B²) = {mag_ab:.2f}"
    )
    info_label.config(text=info_text)

canvas.bind("<Motion>", update)
window.mainloop()
```

💡 **Penjelasan Tambahan**

- **Bar oranye** di atas layar mewakili **seberapa panjang vektornya**.
- Semakin jauh mouse dari tengah, semakin **panjang panah** dan **lebar bar**.
- Informasi hitungannya ditampilkan langsung di bawah, seperti kalkulator interaktif.

---

versi lanjutan dari simulasi *Magnitude Vektor* dengan tambahan fitur:

✅ **Heading (arah vektor AB)** dalam **radian** dan **derajat (°)**

✅ Masih memuat panah A, B, dan AB

✅ Menampilkan magnitude vektor A, B, AB dalam **bar graph dan angka**

✅ Koordinat (0,0) tetap berada di **tengah layar (titik origin)**

Penjelasan Tambahan

- atan2(y, x) menghasilkan arah vektor relatif terhadap sumbu X (dalam **radian**, bisa positif/negatif)
- math.degrees(...) mengonversi radian → derajat
- Heading ini membantu memahami arah vektor bukan hanya besar (magnitude), tapi juga **kemana** vektor menunjuk.

```
# SCRIPT 12 - Heading Vektor
import tkinter as tk
from vector import Vector

# ====== Setup Tkinter ======
WIDTH = 600
HEIGHT = 400
window = tk.Tk()
window.title("Simulasi Magnitude & Heading Vektor")
canvas = tk.Canvas(window, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

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

origin_x = WIDTH // 2
origin_y = HEIGHT // 2

# Bar Graph Parameters
BAR_WIDTH = 30
```

```
BAR_SCALE = 1  # skala visual bar
bar_y_pos = HEIGHT - 200

# ====== Update Visual ======
def update(event):
    canvas.delete("all")

    # Posisi mouse relatif terhadap origin
    mouse = Vector(event.x - origin_x, event.y - origin_y)
    vector_a = Vector(mouse.x, 0)              # A: horizontal
    vector_b = Vector(0, mouse.y)              # B: vertical
    vector_ab = mouse.copy()                     # AB: dari origin ke mouse

    mag_a = abs(vector_a.x)
    mag_b = abs(vector_b.y)
    mag_ab = vector_ab.mag()

    heading_rad = vector_ab.heading_rad()
    heading_deg = vector_ab.heading_deg()

    # ====== Draw coordinate system ======
    canvas.create_line(0, origin_y, WIDTH, origin_y, fill="gray", dash=(2, 2))  # sumbu
X
    canvas.create_line(origin_x, 0, origin_x, HEIGHT, fill="gray", dash=(2, 2)) # sumbu
Y

    # ====== Draw vectors ======
    # vector A (horizontal)
    canvas.create_line(origin_x, origin_y,
                       origin_x + vector_a.x, origin_y,
                       arrow=tk.LAST, fill="orange", width=2)

    # vector B (vertical)
    canvas.create_line(origin_x + vector_a.x, origin_y,
                       origin_x + vector_ab.x, origin_y + vector_b.y,
                       arrow=tk.LAST, fill="blue", width=2)

    # vector AB (diagonal)
    canvas.create_line(origin_x, origin_y,
                       origin_x + vector_ab.x, origin_y + vector_ab.y,
                       arrow=tk.LAST, fill="green", width=3)

    # Titik mouse
    canvas.create_oval(origin_x + vector_ab.x - 4, origin_y + vector_ab.y - 4,
                       origin_x + vector_ab.x + 4, origin_y + vector_ab.y + 4,
                       fill="red")

    # ====== Draw Bar Graphs ======
    canvas.create_text(100, bar_y_pos - 20, text="|A|", font=("Arial", 12))
    canvas.create_rectangle(90, bar_y_pos, 90 + mag_a * BAR_SCALE, bar_y_pos +
BAR_WIDTH, fill="orange")
    canvas.create_text(90 + mag_a * BAR_SCALE + 30, bar_y_pos + BAR_WIDTH // 2,
                       text=f"{mag_a:.2f}", font=("Arial", 12))

    canvas.create_text(100, bar_y_pos + 40, text="|B|", font=("Arial", 12))
    canvas.create_rectangle(90, bar_y_pos + 60, 90 + mag_b * BAR_SCALE, bar_y_pos + 60
+ BAR_WIDTH, fill="blue")
    canvas.create_text(90 + mag_b * BAR_SCALE + 30, bar_y_pos + 60 + BAR_WIDTH // 2,
                       text=f"{mag_b:.2f}", font=("Arial", 12))

    canvas.create_text(100, bar_y_pos + 120, text="|AB|", font=("Arial", 12))
```

```
    canvas.create_rectangle(90, bar_y_pos + 140, 90 + mag_ab * BAR_SCALE, bar_y_pos +
140 + BAR_WIDTH, fill="green")
    canvas.create_text(90 + mag_ab * BAR_SCALE + 30, bar_y_pos + 140 + BAR_WIDTH // 2,
                       text=f"{mag_ab:.2f}", font=("Arial", 12))

    # ====== Info Teks ======
    info_text = (
        f"Mouse (relatif tengah): {vector_ab}\n"
        f"Vector A (X): {vector_a}\n"
        f"Vector B (Y): {vector_b}\n"
        f"Vector AB (Resultan): {vector_ab}\n\n"
        f"|A|  = {mag_a:.2f}\n"
        f"|B|  = {mag_b:.2f}\n"
        f"|AB| = √(A² + B²) = {mag_ab:.2f}\n\n"
        f"Heading AB:\n"
        f"- Radian : {heading_rad:.2f} rad\n"
        f"- Derajat: {heading_deg:.2f}°"
    )
    info_label.config(text=info_text)

canvas.bind("<Motion>", update)
window.mainloop()
```

1.6: *Normalizing Vectors*

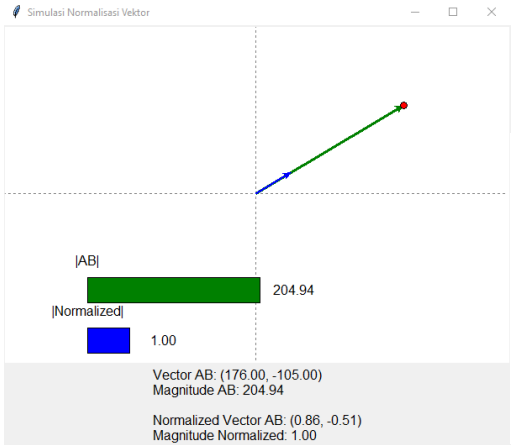
🎓 **Apa Itu Normalizing Vector?**

Bayangkan kamu punya panah (vektor) panjangnya **bebas** — bisa panjang atau pendek.

🔍 Kadang kita hanya butuh **arahnya**, bukan panjangnya. Nah, *normalizing* artinya:

"Membuat vektor punya panjang = 1, tapi tetap ke arah yang sama."

🎯 Vektor hasil normalisasi disebut **unit vector** (vektor satuan).



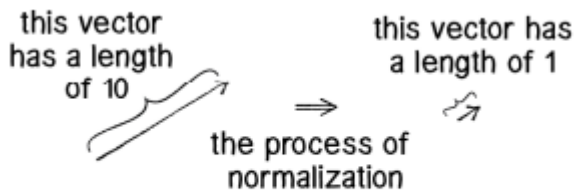


Figure 1.12

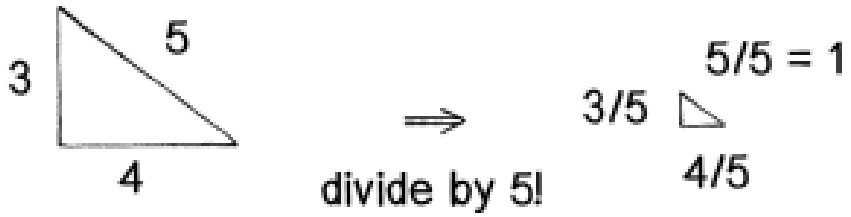


Figure 1.13

📐 Rumus Normalisasi

Kalau vektor kamu adalah:
x = 6, y = 8
Panjangnya:
magnitude = √(6² + 8²) = √100 = 10
Maka normalisasi adalah:
x = 6 / 10 = 0.6
y = 8 / 10 = 0.8
Panjang vektor sekarang = 1 → arah sama, tapi lebih pendek!

🛠 Tambahkan .normalize() ke Kelas Vector2D

```
def normalize(self):
    m = self.mag()
    if m != 0:
        self.x /= m
        self.y /= m
```

🎨 **Visualisasi Tkinter: vektor dinormalisasi ke panjang 1** dan tetap menunjuk ke arah yang sama.

✅ Tujuan

- Menampilkan:
  - **Vektor AB** (dari tengah layar ke mouse)
  - **Vektor AB normalisasi** (panjang 1 tapi arah sama)
  - Bar graph untuk membandingkan magnitudo
- Koordinat tengah layar dianggap titik (0,0)
- Saat kamu gerakkan mouse, vektor asli dan normalisasi akan ditampilkan

---

🧠 Apa itu Normalisasi?

Vektor v dinormalisasi dengan rumus:
sehingga panjangnya selalu 1, tapi arah tetap sama.

✅ Hasil Simulasi:

- **Panah Hijau**: Vektor asli (besar & arah sesuai posisi mouse)
- **Panah Biru**: Vektor yang sudah **dinormalisasi** (arah sama, panjang 1)
- **Bar Graph**: Perbandingan magnitude
- **Info Text**: Menampilkan nilai & penjelasan secara numerik

🧮 Rumus Normalisasi Vektor

1. Hitung panjang (magnitude) vektor:

Vektor `v = (x, y)`

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

Ini seperti menghitung **jarak dari pusat ke ujung panah**.

2. Buat vektor normalisasi:

$$v_{\text{norm}} = \left(\frac{x}{|v|}, \frac{y}{|v|}\right)$$

Dengan cara ini:

- Arah **tetap sama**
- Panjangnya jadi **1** (unit vector)

📜 Kode Lengkap: Visualisasi Normalisasi

```
# script13_normalisasi.py - Simulasi Normalisasi Vektor 2D
import tkinter as tk
from vector import Vector

# ====== Setup Tkinter ======
WIDTH = 600
HEIGHT = 400
window = tk.Tk()
window.title("Simulasi Normalisasi Vektor")
canvas = tk.Canvas(window, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

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

origin_x = WIDTH // 2
```

```
origin_y = HEIGHT // 2
arrow_length = 50  # Panjang tampilan vektor normalisasi

# ====== Update Visual ======
def update(event):
    canvas.delete("all")

    # Vektor AB dari pusat ke mouse
    vec_ab = Vector(event.x - origin_x, event.y - origin_y)
    vec_norm = vec_ab.normalized()

    mag_ab = vec_ab.mag()
    mag_norm = vec_norm.mag()

    # ====== Draw coordinate grid ======
    canvas.create_line(0, origin_y, WIDTH, origin_y, fill="gray", dash=(2, 2))
    canvas.create_line(origin_x, 0, origin_x, HEIGHT, fill="gray", dash=(2, 2))

    # ====== Draw vector AB (asli) ======
    canvas.create_line(origin_x, origin_y,
                       origin_x + vec_ab.x, origin_y + vec_ab.y,
                       arrow=tk.LAST, fill="green", width=3)

    # ====== Draw normalized vector (panjang tetap) ======
    canvas.create_line(origin_x, origin_y,
                       origin_x + vec_norm.x * arrow_length,
                       origin_y + vec_norm.y * arrow_length,
                       arrow=tk.LAST, fill="blue", width=2)

    # Titik mouse
    canvas.create_oval(origin_x + vec_ab.x - 4, origin_y + vec_ab.y - 4,
                       origin_x + vec_ab.x + 4, origin_y + vec_ab.y + 4,
                       fill="red")

    # ====== Draw bar graph ======
    bar_x = 100
    canvas.create_text(bar_x, HEIGHT - 120, text="|AB|", font=("Arial", 12))
    canvas.create_rectangle(bar_x, HEIGHT - 100,
                            bar_x + mag_ab, HEIGHT - 70, fill="green")
    canvas.create_text(bar_x + mag_ab + 40, HEIGHT - 85,
                       text=f"{mag_ab:.2f}", font=("Arial", 12))

    canvas.create_text(bar_x, HEIGHT - 60, text="|Normalized|", font=("Arial", 12))
    canvas.create_rectangle(bar_x, HEIGHT - 40,
                            bar_x + mag_norm * 50, HEIGHT - 10, fill="blue")
    canvas.create_text(bar_x + 50 + 40, HEIGHT - 25,
                       text=f"{mag_norm:.2f}", font=("Arial", 12))

    # ====== Info Text ======
    info = (
        f"Vector AB: {vec_ab}\n"
        f"Magnitude AB: {mag_ab:.2f}\n\n"
        f"Normalized Vector AB: {vec_norm}\n"
        f"Magnitude Normalized: {mag_norm:.2f}"
    )
    info_label.config(text=info)

canvas.bind("<Motion>", update)
window.mainloop()
```

Apa yang Terjadi?

- Gerakkan mouse ke mana pun.
- Panah selalu panjang **50 piksel**, tapi arah tetap sesuai arah mouse dari tengah.
- Karena kita **normalize dulu**, panah selalu punya panjang tetap walau posisi mouse jauh.

🔍 **Aplikasi Normalisasi**

Kita butuh normalisasi saat:

- Mengontrol arah tanpa peduli jarak (misalnya arah gerak robot).
- Menentukan arah gaya / kecepatan dalam fisika simulasi.
- Menjaga arah tetap konsisten saat panjang berubah-ubah.

---

fungsi limit() yang memotong panjang vektor jika lebih dari batas tersebut, **tanpa mengubah arah**. Konsep ini sangat berguna dalam simulasi fisika (misalnya kecepatan maksimum).

✅ **Update: Tambahkan Fungsi limit() ke Vector2D**

Berikut adalah lanjutan dari contoh sebelumnya dengan penambahan fitur *limit vector ke 100px*:

🧠 Konsep Limit

Jika `|v| > 100`, kita skalakan:

$$v = \text{normalize}(v) \times 100$$

✅ **Penjelasan Warna:**

- **Hijau** = Vektor asli
- **Biru** = Normalisasi (panjang 1, arah sama)
- **Merah** = Vektor yang dibatasi hingga maksimal 100px
- **Bar graph** = Menampilkan magnitudo masing-masing

```
# SCRIPT 14 - Simulasi Limit dan Normalisasi Vektor
import tkinter as tk
from vector import Vector

# ====== Setup Tkinter ======
WIDTH = 600
HEIGHT = 400
window = tk.Tk()
window.title("Simulasi Limit dan Normalisasi Vektor")
canvas = tk.Canvas(window, width=WIDTH, height=HEIGHT, bg="white")
canvas.pack()

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

origin_x = WIDTH // 2
origin_y = HEIGHT // 2
arrow_length = 50
max_vector_length = 100  # batas maksimum vektor

# ====== Update Visual ======
def update(event):
    canvas.delete("all")

    # Vektor dari pusat ke mouse
    vec_ab = Vector(event.x - origin_x, event.y - origin_y)
    vec_norm = vec_ab.normalized()
    vec_limited = vec_ab.limited(max_vector_length)

    mag_ab = vec_ab.mag()
    mag_norm = vec_norm.mag()
    mag_limited = vec_limited.mag()
```

```
    # ====== Garis koordinat ======
    canvas.create_line(0, origin_y, WIDTH, origin_y, fill="gray", dash=(2, 2))
    canvas.create_line(origin_x, 0, origin_x, HEIGHT, fill="gray", dash=(2, 2))

    # ====== Vektor asli ======
    canvas.create_line(origin_x, origin_y,
                       origin_x + vec_ab.x, origin_y + vec_ab.y,
                       arrow=tk.LAST, fill="green", width=3)

    # ====== Vektor dinormalisasi (panjang tetap) ======
    canvas.create_line(origin_x, origin_y,
                       origin_x + vec_norm.x * arrow_length,
                       origin_y + vec_norm.y * arrow_length,
                       arrow=tk.LAST, fill="blue", width=2)

    # ====== Vektor dibatasi (max 100 px) ======
    canvas.create_line(origin_x, origin_y,
                       origin_x + vec_limited.x, origin_y + vec_limited.y,
                       arrow=tk.LAST, fill="red", width=2)

    # Titik mouse
    canvas.create_oval(origin_x + vec_ab.x - 4, origin_y + vec_ab.y - 4,
                       origin_x + vec_ab.x + 4, origin_y + vec_ab.y + 4,
                       fill="red")

    # ====== Bar Graph Magnitude ======
    bar_x = 100
    bar_y = HEIGHT - 200
    canvas.create_text(bar_x, bar_y - 130, text="|AB|", font=("Arial", 12))
    canvas.create_rectangle(bar_x, bar_y - 110,
                            bar_x + min(mag_ab, 200), bar_y - 85, fill="green")
    canvas.create_text(bar_x + min(mag_ab, 200) + 40, bar_y - 98,
                       text=f"{mag_ab:.2f}", font=("Arial", 12))

    canvas.create_text(bar_x, bar_y - 75, text="|Normalized|", font=("Arial", 12))
    canvas.create_rectangle(bar_x, bar_y - 55,
                            bar_x + mag_norm * 50, bar_y - 30, fill="blue")
    canvas.create_text(bar_x + 50 + 40, bar_y - 43,
                       text=f"{mag_norm:.2f}", font=("Arial", 12))

    canvas.create_text(bar_x, bar_y - 20, text="|Limited|", font=("Arial", 12))
    canvas.create_rectangle(bar_x, bar_y,
                            bar_x + mag_limited, bar_y + 25, fill="red")
    canvas.create_text(bar_x + mag_limited + 40, bar_y + 13,
                       text=f"{mag_limited:.2f}", font=("Arial", 12))

    # ====== Info Text ======
    info = (
        f"Vector AB: {vec_ab}\n"
        f"Magnitude AB: {mag_ab:.2f}\n\n"
        f"Normalized AB: {vec_norm}\n"
        f"Magnitude Normalized: {mag_norm:.2f}\n\n"
        f"Limited Vector AB: {vec_limited}\n"
        f"Magnitude Limited: {mag_limited:.2f}"
    )
    info_label.config(text=info)

canvas.bind("<Motion>", update)
window.mainloop()
```

1.7 Vector Motion: Velocity

🎓 Konsep Inti: Vector Motion (Kecepatan dan Lokasi)

Sederhananya:

1. **Lokasi** = di mana benda berada.
2. **Kecepatan (Velocity)** = seberapa cepat dan ke arah mana benda bergerak.
3. **Gerakan** = Lokasi terus berubah sesuai kecepatan.

Bayangkan benda seperti bola:

- Jika bola punya kecepatan ke kanan 2 piksel per frame, maka:
- lokasi_x = lokasi_x + kecepatan_x

🧠 Kenapa Pakai OOP (Object-Oriented Programming)?

Dengan OOP, kita bisa membuat satu **class Mover**, lalu menciptakan banyak bola yang bisa bergerak sendiri-sendiri.

📘 Penjelasan Konsep: Motion 101 - Vector Velocity

1. Apa itu Lokasi dan Velocity?

Bayangkan bola bergerak di layar seperti ini:

- **Lokasi**: posisi bola sekarang (misalnya di titik x=100, y=50).
- **Velocity (kecepatan)**: arah dan kecepatan gerak bola. Misalnya:
    - velocity = (2, 1) artinya setiap waktu:
        - x bertambah 2 (bergerak ke kanan)
        - y bertambah 1 (bergerak ke bawah)

2. Bagaimana bola bergerak?

Bola bergerak karena **lokasi ditambah velocity setiap waktu (frame)**:

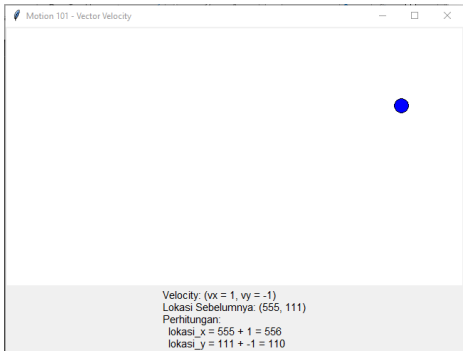
lokasi_x = lokasi_x + velocity_x

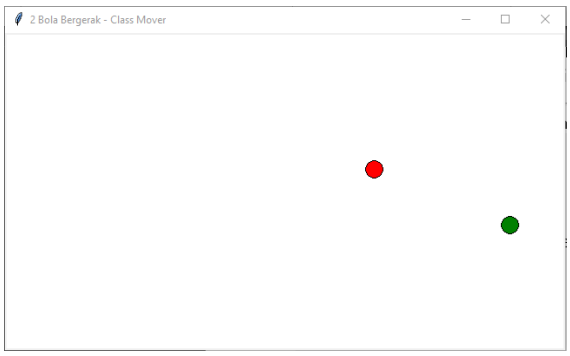
lokasi_y = lokasi_y + velocity_y

3. Kenapa pakai class Vector dan Mover?

Agar kita bisa:

- Mengatur pergerakan dengan mudah,
- Memiliki banyak bola nanti yang bergerak sendiri-sendiri.





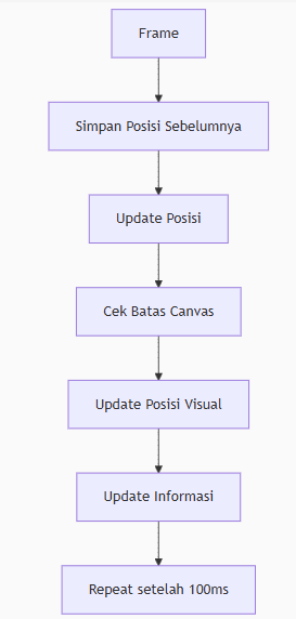
✅ Kode Python + Penampilan Perhitungan

script simulasi gerakan bola dengan vector velocity:

🌟 Alur Program Utama

1. **Inisialisasi**:
    - Membuat window tkinter dengan canvas 640x360
    - Membuat objek Mover (bola) dengan:
        - Posisi acak di dalam canvas
        - Kecepatan acak (vx dan vy antara -2 sampai 2)
    - Menyiapkan label untuk menampilkan informasi
2. **Loop Animasi (10 FPS)**:

Diagram



📐 **Rumus Fisika**

1. **Gerakan Bola**:

posisi_baru = posisi_lama + kecepatan

Dalam kode:

self.location.add(self.velocity)

2. **Batas Layar (Wrap Around)**:

if x > width: x = 0

if x < 0: x = width

if y > height: y = 0

if y < 0: y = height

🎞️ **Contoh Perhitungan Frame-by-Frame**

Asumsi Kondisi Awal:

- Posisi awal: (150, 200)
- Kecepatan: (2, -1)
- Ukuran canvas: 640x360

Frame	Sebelum Bergerak	Perhitungan	Setelah Bergerak	Cek Batas
1	(150, 200)	(150+2, 200-1)	(152, 199)	-
2	(152, 199)	(152+2, 199-1)	(154, 198)	-
...	...	...	...	...
N	(638, 5)	(638+2, 5-1)	(640, 4)	x>width → x=0
N+1	(0, 4)	(0+2, 4-1)	(2, 3)	-

💻 **Komponen Penting Class Mover**

1. **Atribut**:

self.location = Vector(x, y) *# Posisi bola*

self.velocity = Vector(vx, vy) *# Kecepatan*

self.radius = 10 *# Ukuran bola*

2. **Method Update**:
   - Menyimpan posisi sebelumnya
   - Menghitung posisi baru
   - Mengecek batas layar
   - Memperbarui tampilan visual
   - Menampilkan informasi perhitungan

📊 **Contoh Output Informasi**

Velocity: (vx = 2, vy = -1)

Lokasi Sebelumnya: (150, 200)

Perhitungan:

lokasi_x = 150 + 2 = 152

lokasi_y = 200 + -1 = 199

🔍 **Analisis Gerakan**

1. **Gerakan Linear**:
   - Bola bergerak lurus dengan kecepatan konstan
   - Tidak ada percepatan atau gesekan
2. **Efek Wrap-Around**:
   - Bola muncul di sisi berlawanan saat keluar layar
   - Menciptakan ilusi ruang tak terbatas

⏱ **Karakteristik Waktu**

- Refresh rate: 10 FPS (setiap 100ms)
- Kecepatan: 2-3 pixel per frame
- Dirancang lambat agar mudah diamati perhitungannya

```
#SCRIPT 15 - Simulasi Gerakan satu Bola dengan Vector Velocity
import tkinter as tk
import random
from vector import Vector

# Kelas Mover (bola yang bergerak)
class Mover:
    def __init__(self, canvas, width, height, info_label):
        self.canvas = canvas
        self.width = width
        self.height = height
        self.radius = 10
        self.info_label = info_label

        # Posisi dan velocity random
        self.location = Vector(random.randint(100, width-100), random.randint(100,
height-100))
        self.velocity = Vector(random.choice([-2, -1, 1, 2]), random.choice([-2, -1, 1,
2]))

        self.id = canvas.create_oval(self.location.x - self.radius, self.location.y -
self.radius,
                                     self.location.x + self.radius, self.location.y +
self.radius,
                                     fill="blue")

    def update(self):
        # Salin posisi sebelum bergerak
        prev_location = self.location.copy()

        # Tambah lokasi dengan velocity
        self.location.add(self.velocity)
        self.check_edges()

        # Update posisi bola di canvas
        self.canvas.coords(self.id,
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius)

        # Tampilkan info perhitungan
        info_text = (
            f"Velocity: (vx = {self.velocity.x}, vy = {self.velocity.y})\n"
            f"Lokasi Sebelumnya: ({prev_location.x}, {prev_location.y})\n"
            f"Perhitungan:\n"
            f"  lokasi_x = {prev_location.x} + {self.velocity.x} = {self.location.x}\n"
            f"  lokasi_y = {prev_location.y} + {self.velocity.y} = {self.location.y}"
        )
        self.info_label.config(text=info_text)

    def check_edges(self):
        # Jika keluar layar, lompat ke sisi lain
        if self.location.x > self.width:
            self.location.x = 0
        elif self.location.x < 0:
            self.location.x = self.width
```

```
        if self.location.y > self.height:
            self.location.y = 0
        elif self.location.y < 0:
            self.location.y = self.height

# Fungsi utama
def main():
    WIDTH = 640
    HEIGHT = 360

    root = tk.Tk()
    root.title("Simulasi Gerakan dengan Vector Velocity")

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

    info_label = tk.Label(root, text="", justify="left", font=("Arial", 11))
    info_label.pack()

    mover = Mover(canvas, WIDTH, HEIGHT, info_label)

    def animate():
        mover.update()
        root.after(100, animate)  # Update tiap 100 ms agar mudah dilihat

    animate()
    root.mainloop()

main()
```

🎯 **Hasil Saat Jalan:**

- Sebuah bola biru bergerak ke suatu arah.
- Di bawah layar muncul perhitungan:
    - Velocity: arah dan kecepatan bola.
    - Lokasi sebelum bergerak.
    - Rumus penambahan lokasi (lokasi + velocity).
    - Lokasi setelah bergerak.

🧠 **Kesimpulan Sederhana untuk:**

- **Velocity itu kayak dorongan terus-menerus** ke arah tertentu.
- **Setiap detik (frame)**, bola akan **bergerak sedikit demi sedikit** sesuai velocity-nya.
- Lokasi berubah = lokasi + velocity.

Kalau kamu ingin memperlambat, menambah banyak bola, atau pakai panah vektor velocity, tinggal bilang ya!

📚 **Penjelasan Sederhana**

Konsep	Penjelasan
Vector	Menyimpan posisi (x, y) dan kecepatan (vx, vy).
add	Menambahkan kecepatan ke posisi agar benda bisa bergerak.
Mover	Bola yang bisa bergerak.
update()	Memperbarui posisi berdasarkan kecepatan.
check_edges()	Kalau bola keluar dari layar, muncul lagi dari sisi berlawanan.
animate()	Fungsi yang berjalan terus untuk menghidupkan animasi.

✅ **1. Kenapa pakai velocity?**

Kita pakai velocity karena:

- **Gerakan benda tidak langsung berpindah tempat besar**, tapi **bergerak sedikit demi sedikit** dari frame ke frame (kayak animasi).
- Dengan velocity, kita bisa:
  - Mengatur arah dan kecepatan gerak.
  - Mengubah kecepatan kapan saja (misalnya saat ditabrak, kena angin, dll).
  - Menghitung posisi baru: location += velocity.

Contoh:

```
location = Vector(100, 100)
velocity = Vector(2, 3)

# Setiap frame
location.add(velocity)
```

Artinya: setiap frame, bola akan bergerak 2 ke kanan dan 3 ke bawah.

✅ **2. Kenapa pakai class Mover?**

Bayangkan kalau kita punya banyak bola:

Tanpa class:

```
loc1 = Vector(...)
vel1 = Vector(...)
loc2 = Vector(...)
vel2 = Vector(...)
# dst...
```

Ribet banget! Susah dikelola.

Dengan class Mover, kita bisa buat banyak objek:

```
mover1 = Mover(...)
mover2 = Mover(...)
```

Masing-masing bola:

- Punya posisi sendiri (self.location)
- Punya kecepatan sendiri (self.velocity)
- Bisa update dan gambar dirinya sendiri.

Jadi, class Mover bikin program **lebih rapi, lebih mudah diperluas** dan bisa punya banyak bola tanpa ngoding ulang.

❓**Lalu, apakah bisa langsung pakai vector.add(...) tanpa velocity?**

Bisa **kalau hanya 1 kali pindah tempat** (misalnya klik → langsung pindah).

Tapi kalau **benda harus terus bergerak sendiri** (seperti animasi bola), maka:

- Kita **butuh nilai perubahan posisi yang tetap** (yaitu velocity),
- Dan **update lokasinya setiap waktu**.

Jadi:

```
# ❌ Hanya langsung add ke lokasi → tidak bisa gerak terus-menerus
location.add(Vector(2, 3))  # 1x saja

# ✅ Pakai velocity → bisa terus bergerak tiap frame
velocity = Vector(2, 3)
location.add(velocity)  # dilakukan tiap frame
```

🧠 **Kesimpulan :**

Konsep	Fungsi
velocity	Menyimpan arah dan kecepatan gerak bola
location	Menyimpan posisi bola

Konsep	Fungsi
location += velocity	Dipakai supaya bola bergerak terus
class Mover	Membuat 1 bola jadi punya posisi, kecepatan, dan bisa gerak sendiri

Kalau semua digabung, bola bisa **bergerak sendiri, tidak diam, dan mudah diatur**.

✅ Versi Class Mover untuk 2 Bola Bergerak

```
#SCRIPT 16 - Simulasi Gerakan Dua Bola dengan Vector Velocity
import tkinter as tk
import random
from vector import Vector

# Kelas Mover (benda yang bergerak)
class Mover:
    def __init__(self, canvas, width, height, color="blue"):
        self.canvas = canvas
        self.width = width
        self.height = height
        self.radius = 10
        self.color = color

        # Lokasi dan kecepatan acak
        self.location = Vector(random.randint(0, width), random.randint(0, height))
        self.velocity = Vector(random.choice([-2, -1, 1, 2]), random.choice([-2, -1, 1,
2]))

        # Gambar bola di kanvas
        self.id = canvas.create_oval(
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius,
            fill=self.color
        )

    # Update posisi bola
    def update(self):
        self.location.add(self.velocity)
        self.check_edges()
        self.canvas.coords(
            self.id,
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius
        )

    # Jika keluar layar, muncul di sisi sebaliknya
    def check_edges(self):
        if self.location.x > self.width:
            self.location.x = 0
        elif self.location.x < 0:
            self.location.x = self.width
        if self.location.y > self.height:
            self.location.y = 0
        elif self.location.y < 0:
            self.location.y = self.height

# Program utama
def main():
    WIDTH = 640
    HEIGHT = 360

    root = tk.Tk()
```

```
    root.title("2 Bola Bergerak - Class Mover")
    canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
    canvas.pack()

    # Buat dua bola berbeda warna
    mover1 = Mover(canvas, WIDTH, HEIGHT, color="red")
    mover2 = Mover(canvas, WIDTH, HEIGHT, color="green")

    # Loop update animasi
    def animate():
        mover1.update()
        mover2.update()
        root.after(30, animate)

    animate()
    root.mainloop()

main()
```

✨ **Penjelasan :**

- Kita buat **class Mover** untuk mewakili **benda yang bisa bergerak sendiri**.
- Lalu kita **buat dua objek**: mover1 (bola merah) dan mover2 (bola hijau).
- Setiap bola punya **lokasi dan kecepatan sendiri**, sehingga bisa bergerak bebas.
- Setiap kali animasi berjalan, fungsi update() dari masing-masing bola dipanggil.

❓ **Kenapa class lebih bagus?**

Kalau tanpa class, kita harus nulis:
location1, velocity1
location2, velocity2
dan bikin fungsi terpisah untuk masing-masing.
Dengan class, cukup:
mover1 = Mover(...)
mover2 = Mover(...)
Lalu tinggal panggil mover1.update(), mover2.update().

🌈 **Modifikasi Script untuk 5 Bola**

```
  # Daftar warna untuk 5 bola
  colors = ["red", "green", "blue", "purple", "orange"]
  movers = [Mover(canvas, WIDTH, HEIGHT, info_label, color)
        for color in colors]
……
  def animate():
    for mover in movers:
      mover.update()
    root.after(100, animate)
```

1.8 Vector Motion: Acceleration

🧠 **Konsep Dasar: Apa itu Acceleration (Percepatan)?**

Bayangkan kamu mendorong mobil mainan pelan-pelan. Awalnya diam → mulai jalan → makin cepat.
Perubahan kecepatan itulah yang disebut **percepatan**.

📦 **Istilah Penting:**

Istilah	Artinya	Contoh
Location (Posisi)	Di mana benda berada sekarang	Titik di layar

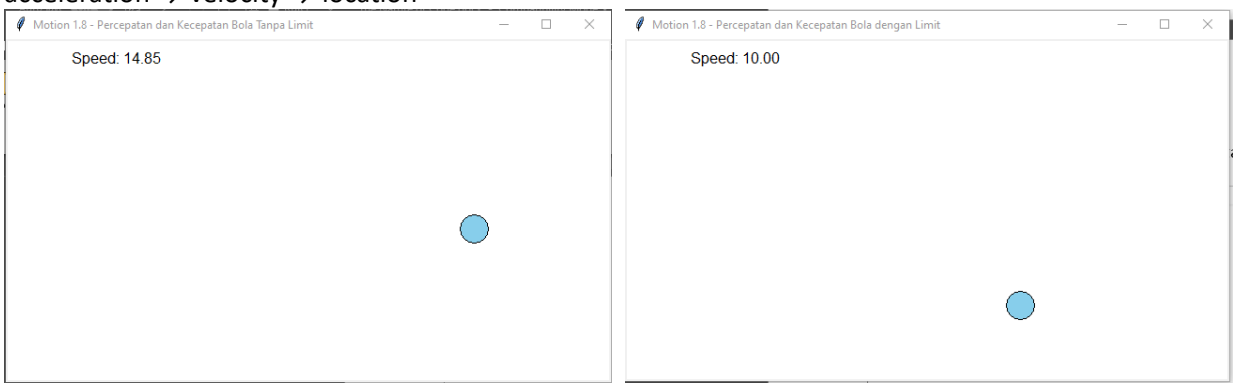
Istilah	Artinya	Contoh
Velocity (Kecepatan)	Seberapa cepat dan ke mana arahnya bergerak	Bergerak ke kanan 3px/frame
Acceleration (Percepatan)	Seberapa cepat kecepatan berubah	Awalnya diam → makin lama makin cepat

---

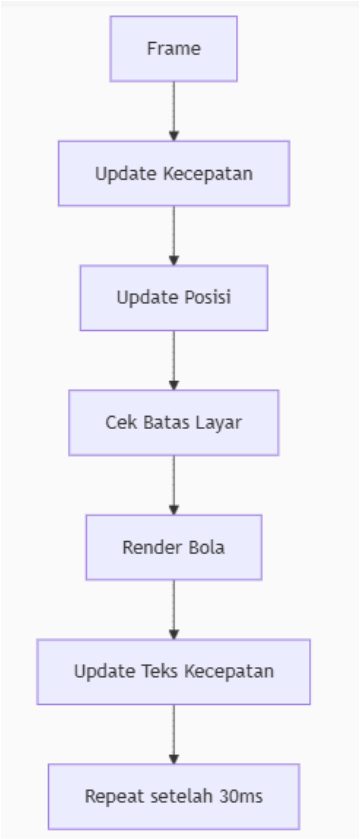
🎯 **Konsep Motion Baru: Trickle Down Motion**

"Percepatan memengaruhi kecepatan, kecepatan memengaruhi posisi."

acceleration → velocity → location



🐍 **Python Tkinter – Simulasi Bola yang Semakin Cepat**



script simulasi bola dengan percepatan konstan:

🌟 **Alur Program Utama**

1. **Inisialisasi**:
   - Bola dibuat di tengah layar (320, 180)
   - Kecepatan awal: (0, 0)
   - Percepatan konstan: (-0.05, 0.05) ke kiri atas
2. **Loop Animasi (~33 FPS)**:

Diagram

📐 **Rumus Fisika**

1. **Percepatan**:

acceleration = Vector(-0.05, 0.05) *# konstan*

2. **Update Kecepatan**:

velocity_new = velocity_old + acceleration

3. **Update Posisi**:

position_new = position_old + velocity_new

4. **Kecepatan (Magnitude)**:

speed = √(vx² + vy²)

🧮 **Contoh Perhitungan Frame-by-Frame**

Parameter Awal:

- Posisi: (320, 180)
- Kecepatan: (0, 0)
- Percepatan: (-0.05, 0.05)

Frame	Perhitungan Kecepatan	Kecepatan Baru	Perhitungan Posisi	Posisi Baru	Speed
1	(0,0) + (-0.05,0.05)	(-0.05, 0.05)	(320,180) + (-0.05,0.05)	(319.95,180.05)	0.07
2	(-0.05,0.05) + (-	(-0.10, 0.10)	(319.95,180.05) + (-	(319.85,180.15)	0.14

Frame	Perhitungan Kecepatan	Kecepatan Baru	Perhitungan Posisi	Posisi Baru	Speed
	0.05,0.05)		0.10,0.10)		
3	(-0.10,0.10) + (-0.05,0.05)	(-0.15, 0.15)	(319.85,180.15) + (-0.15,0.15)	(319.70,180.30)	0.21
10	...	(-0.50, 0.50)	...	(315.0,185.0)	0.71
20	...	(-1.00, 1.00)	...	(300.0,200.0)	1.41

💻 **Komponen Penting Class Mover**

1. **Atribut Fisika**:

```
self.location = Vector(width//2, height//2)  # Posisi tengah
self.velocity = Vector(0, 0)  # Diam awal
self.acceleration = Vector(-0.05, 0.05)  # Percepatan konstan
```

2. **Method Update**:

```
def update(self):
   self.velocity.add(self.acceleration)  # Rumus 1
   self.location.add(self.velocity)     # Rumus 2
   self.check_edges()
   # Update visual...
```

📊 **Visualisasi Gerakan**

Percepatan:
← (-x)
↑ (+y)

Lintasan bola:
↗ (bergerak ke kiri atas)
[●] mulai dari (320,180) bergerak menuju (0,360)

⚙ **Mekanisme Batas Layar**

- Jika bola keluar layar:
  - Muncul di sisi berlawanan (wrap-around)
  - Contoh: bola keluar di x=640 → muncul di x=0

📝 **Analisis Khusus**

1. **Percepatan Konstan**:
   - Kecepatan bertambah secara linear setiap frame
   - Arah diagonal kiri-atas
2. **Pola Gerakan**:
   - Frame awal: gerakan lambat
   - Semakin lama semakin cepat
   - Lintasan lurus diagonal
3. **Perhitungan Speed**:
   - Frame 1: $\sqrt{((-0.05)^2 + (0.05)^2)} \approx 0.07$
   - Frame 10: $\sqrt{((-0.5)^2 + (0.5)^2)} \approx 0.71$
   - Frame 20: $\sqrt{((-1)^2 + (1)^2)} \approx 1.41$

```
#SCRIPT 17 - Simulasi Gerakan Bola Percepatan Konstan (-0.05, 0.05)
import tkinter as tk
from vector import Vector

# ---------------------------
```

```
# Kelas Mover = Bola yang bisa bergerak
# ---------------------------
class Mover:
    def __init__(self, canvas, width, height):
        self.canvas = canvas
        self.width = width    # lebar layar
        self.height = height  # tinggi layar
        self.radius = 15      # ukuran bola

        # Posisi awal bola di tengah layar
        self.location = Vector(width // 2, height // 2)

        # Kecepatan awal = diam (0,0)
        self.velocity = Vector(0, 0)

        # Percepatan konstan ke kiri atas
        self.acceleration = Vector(-0.05, 0.05)

        # Gambar bola di canvas
        self.id = canvas.create_oval(
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius,
            fill="skyblue"
        )

        # Teks kecepatan
        self.speed_text = canvas.create_text(70, 20, text="Speed: 0.00", anchor="w",
font=("Arial", 12), fill="black")

    def update(self):
        # 1. Tambahkan percepatan ke kecepatan
        self.velocity.add(self.acceleration)

        # 2. Tambahkan kecepatan ke posisi
        self.location.add(self.velocity)

        # 3. Cek apakah bola keluar layar
        self.check_edges()

        # 4. Gambar ulang bola di posisi baru
        self.canvas.coords(self.id,
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius)

        # 5. Hitung dan tampilkan kecepatan (panjang vektor velocity)
        speed = self.velocity.magnitude()
        self.canvas.itemconfig(self.speed_text, text=f"Speed: {speed:.2f}")

    def check_edges(self):
        if self.location.x > self.width:
            self.location.x = 0
        elif self.location.x < 0:
            self.location.x = self.width
        if self.location.y > self.height:
            self.location.y = 0
        elif self.location.y < 0:
            self.location.y = self.height

# ---------------------------
# Fungsi utama (menjalankan animasi)
# ---------------------------
```

```
def main():
    WIDTH = 640
    HEIGHT = 360

    root = tk.Tk()
    root.title("Simulasi Gerakan Bola Percepatan Konstan (-0.05, 0.05)")

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

    mover = Mover(canvas, WIDTH, HEIGHT)

    def animate():
        mover.update()
        root.after(30, animate)

    animate()
    root.mainloop()

main()
```

📜 Penjelasan Mudah

Bagian Program	Penjelasan
acceleration = Vector(-0.05, 0.05)	Percepatan konstan: bola akan makin lama makin cepat
velocity.add(acceleration)	Kecepatan bertambah sedikit setiap frame (dipengaruhi percepatan)
location.add(velocity)	Posisi berpindah mengikuti kecepatan saat ini

---

🧪 Eksperimen yang Bisa Dicoba:

1. **Ubah arah percepatan** ke Vector(0.01, 0.01) → bola bergerak diagonal kanan bawah.

Konsep	Penjelasan Singkat
Posisi	Lokasi bola saat ini di layar
Kecepatan	Seberapa cepat dan ke mana bola bergerak
Percepatan	Perubahan kecepatan → membuat bola semakin cepat atau berubah arah
Edges	Saat bola keluar dari layar, ia muncul kembali dari sisi berlawanan

---

🔧 Exercise 1.4 – Menulis Fungsi limit() di PVector

📘 Apa itu limit()?

Bayangkan kamu sedang naik **mobil mainan remote control**.

- Mobil ini punya **arah dan kecepatan**, seperti vektor.
- Tapi kadang kecepatannya **terlalu tinggi!**
- Nah, fungsi limit() adalah **rem otomatis**:

Kalau kecepatannya terlalu besar, kita rem biar nggak kebablasan.

🧠 Apa itu vektor?

Vektor punya:

- **Arah** (contoh: ke kanan atas)
- **Panjang** (seberapa cepat atau kuat arah itu)

Vektor dalam gerakan disebut **kecepatan** (velocity), dan panjangnya disebut **magnitude** (dibaca: mag-ni-tud).

🔍 Apa yang dilakukan limit()?

```
def limit(self, max_val):
    # Hitung panjang vektor sekarang (pakai rumus Phytagoras)
    mag = math.sqrt(self.x**2 + self.y**2)

    # Kalau panjang vektor lebih besar dari nilai maksimum:
    if mag > max_val:
        # Kecilkan vektor agar panjangnya sama dengan max_val
        self.x = (self.x / mag) * max_val
        self.y = (self.y / mag) * max_val
```

➖ Penjelasan langkah demi langkah:

Langkah	Penjelasan
mag = ...	Hitung panjang vektor dengan rumus: √(x² + y²)
if mag > max_val	Cek apakah terlalu besar? (misalnya > 10)
self.x = ...	Kecilkan x agar panjang vektor jadi pas 10
self.y = ...	Kecilkan y juga dengan cara yang sama

🎨 **Ilustrasi Sederhana**
Bayangkan kamu punya **panah** sepanjang 30 cm yang menunjuk ke kanan atas. Tapi kamu cuma mau panahnya **maksimal 10 cm**.
Fungsi limit() akan **memendekkan panahmu** jadi 10 cm, **tapi tetap ke arah yang sama**.

🧪 **Contoh:**
vektor (6, 8) → panjangnya 10 (karena √(6² + 8²) = 10)
kalau max_val = 5 → kita ubah jadi vektor (3, 4)

📌 **Kesimpulan**

- limit() dipakai agar **kecepatan objek tidak terlalu tinggi**.
- Ini penting untuk **simulasi fisika**, biar gerakannya **lebih wajar dan aman**.
- Fungsi ini menghitung panjang vektor, lalu mengecilkannya kalau terlalu besar.

🚗 **Example 1.8: Motion 101 (Kecepatan dan Percepatan Konstan)**
Penjelasan:

- Bola dimulai diam.
- Percepatan terus menambah kecepatan setiap frame.
- Fungsi limit() menjaga agar kecepatan tidak terlalu tinggi.

✅ **Implementasi Python:**

```
#SCRIPT 18 - Simulasi Gerakan Bola Percepatan Konstan (-0.05, 0.05) dengan Limit
Kecepatan
import tkinter as tk
from vector import Vector

# ============================
# Kelas Mover (bola yang bergerak)
# ============================
class Mover:
    def __init__(self, canvas, width, height):
        self.canvas = canvas
        self.width = width
        self.height = height
        self.radius = 15

        # Posisi awal di tengah layar
        self.location = Vector(width // 2, height // 2)
```

```
        # Kecepatan awal (diam)
        self.velocity = Vector(0, 0)

        # Percepatan tetap (contoh: ke kiri atas)
        self.acceleration = Vector(-0.05, 0.05)

        # Gambar bola
        self.id = canvas.create_oval(
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius,
            fill="skyblue"
        )

        # Tampilkan nilai kecepatan
        self.speed_text = canvas.create_text(10, 10, anchor="nw",
                                             font=("Arial", 12),
                                             fill="black",
                                             text="Speed: 0.00")

    def update(self):
        # Tambahkan percepatan ke kecepatan
        self.velocity.add(self.acceleration)

        # Batasi kecepatan maksimal
        self.velocity.limit(10)

        # Tambahkan kecepatan ke posisi
        self.location.add(self.velocity)

        # Jika keluar layar, pindahkan ke sisi sebaliknya
        self.wrap_around_edges()

        # Perbarui posisi bola di canvas
        self.canvas.coords(
            self.id,
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius
        )

        # Hitung dan tampilkan kecepatan
        speed = self.velocity.magnitude()
        self.canvas.itemconfig(self.speed_text, text=f"Speed: {speed:.2f}")

    def wrap_around_edges(self):
        if self.location.x > self.width:
            self.location.x = 0
        elif self.location.x < 0:
            self.location.x = self.width
        if self.location.y > self.height:
            self.location.y = 0
        elif self.location.y < 0:
            self.location.y = self.height

# ============================
# Fungsi Utama
# ============================
def main():
    WIDTH = 640
    HEIGHT = 360
```

```
    root = tk.Tk()
    root.title("Simulasi Bola dengan Percepatan dan Limit Kecepatan")

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

    mover = Mover(canvas, WIDTH, HEIGHT)

    def animate():
        mover.update()
        root.after(30, animate)

    animate()
    root.mainloop()

# ============================
# Jalankan Program
# ============================
if __name__ == "__main__":
    main()
```

📜 Penjelasan Mudah

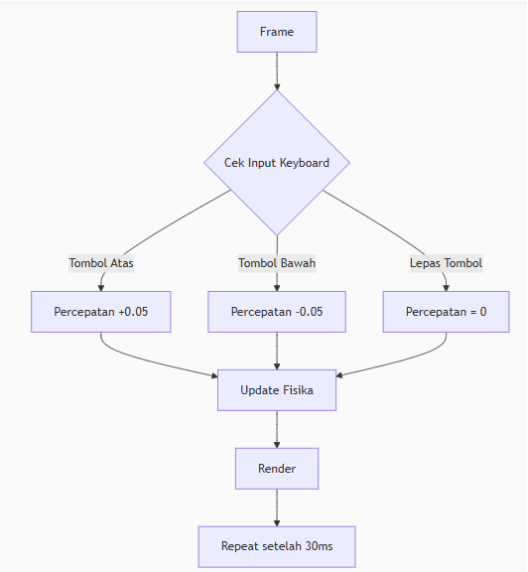
Bagian Program	Penjelasan
acceleration = Vector(-0.05, 0.05)	Percepatan konstan: bola akan makin lama makin cepat
velocity.add(acceleration)	Kecepatan bertambah sedikit setiap frame (dipengaruhi percepatan)
location.add(velocity)	Posisi berpindah mengikuti kecepatan saat ini
velocity.limit(10)	Batas maksimal kecepatan (biar tidak terlalu cepat)

🧪 Eksperimen yang Bisa Dicoba:

1. **Ubah arah percepatan** ke Vector(0.01, 0.01) → bola bergerak diagonal kanan bawah.

Konsep	Penjelasan Singkat
Posisi	Lokasi bola saat ini di layar
Kecepatan	Seberapa cepat dan ke mana bola bergerak
Percepatan	Perubahan kecepatan → membuat bola semakin cepat atau berubah arah
Limit	Membatasi kecepatan agar tidak terlalu cepat
Edges	Saat bola keluar dari layar, ia muncul kembali dari sisi berlawanan

🏃 Exercise 1.5 – Simulasi Mobil / Pelari



script simulasi mobil dengan sistem dorong dan rem:

🌟 **Alur Program Utama**

1. **Inisialisasi**:
   - Mobil dibuat di tengah layar (300, 200)
   - Kecepatan awal: (0, 0)
   - Percepatan awal: (0, 0)
   - Kecepatan maksimum (topspeed): 7
2. **Loop Animasi (~33 FPS)**:

Diagram

📐 **Rumus Fisika**

1. **Update Kecepatan**:

```
velocity += acceleration
velocity = clamp(velocity, topspeed)  # Batasi kecepatan maksimum
```

2. **Update Posisi**:

```
position += velocity
```

3. **Kontrol Mobil**:
   - Tombol Atas: acceleration = (0.05, 0)
   - Tombol Bawah: acceleration = (-0.05, 0)
   - Lepas Tombol: acceleration = (0, 0)

🎞️ **Contoh Perhitungan Frame-by-Frame**

Scenario 1: Akselerasi

Frame	Input	Percepatan	Perhitungan Kecepatan	Kecepatan Baru	Posisi Baru (x)
1	Atas	+0.05	0 + 0.05	0.05	300.05
2	Atas	+0.05	0.05 + 0.05	0.10	300.15
...	...	...	...	...	...
140	Atas	+0.05	6.95 + 0.05	7.00 (max)	390.00

Scenario 2: Pengereman

Frame	Input	Percepatan	Perhitungan Kecepatan	Kecepatan Baru	Posisi Baru (x)
1	Bawah	-0.05	7.00 - 0.05	6.95	506.95
2	Bawah	-0.05	6.95 - 0.05	6.90	513.85
...	...	...	...	...	...

💻 **Komponen Penting Class Car**

1. **Atribut Fisika**:

```
self.velocity = Vector(0, 0)  # Kecepatan
self.acceleration = Vector(0, 0)  # Percepatan
self.topspeed = 7  # Batas kecepatan
```

2. **Method Kontrol**:

```
def accelerate(self):  # Gas
  self.acceleration = Vector(0.05, 0)

def brake(self):  # Rem
  self.acceleration = Vector(-0.05, 0)
```

def stop(self): *# Netral*
  self.acceleration = Vector(0, 0)

⚙ **Mekanisme Batas Layar**

- Sistem wrap-around:

if x > width: x = 0
if x < 0: x = width

📊 **Visualisasi Gerakan**

[●] Mobil biru bergerak horizontal:
- Tombol Atas: Bergerak kanan (x meningkat)
- Tombol Bawah: Bergerak kiri (x berkurang)
- Kecepatan ditampilkan di pojok kiri atas

📘 **Penjelasan Sederhana:**

🏃 **Apa yang disimulasikan?**

Kita membuat **simulasi mobil atau pelari** yang bisa:

- **Dipercepat** (tekan tombol panah ↑)
- **Direm** (tekan tombol ↓)
- Mobil bergerak ke kanan dan **memantul ke kiri** kalau sudah sampai ujung.

⏱ **Kecepatan (speed)?**

Kita juga **menampilkan angka kecepatannya di layar**, supaya kamu bisa melihat:

- Semakin sering kamu tekan ↑, kecepatannya bertambah (tapi dibatasi maksimum).
- Kalau kamu tekan ↓, kecepatannya melambat.

✅ **Kode yang Sudah Ditambahkan:**

```
#SCRIPT 19 - Exercise 1.5 - Mobil Dorong Rem
import tkinter as tk
from vector import Vector

# ---------------------------
# Kelas Mobil / Pelari
# ---------------------------
class Car:
    def __init__(self, canvas, width, height):
        self.canvas = canvas
        self.width = width
        self.height = height
        self.radius = 15

        self.location = Vector(width // 2, height // 2)
        self.velocity = Vector(0, 0)
        self.acceleration = Vector(0, 0)
        self.topspeed = 7  # batas kecepatan maksimum

        # Gambar mobil/pelari
        self.id = canvas.create_oval(
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius,
            fill="blue"
        )

        # Tambahan: Tampilkan kecepatan
        self.speed_text = canvas.create_text(10, 10, anchor="nw", text="Speed: 0.00",
font=("Arial", 12), fill="black")

    def update(self):
        self.velocity.add(self.acceleration)       # tambahkan percepatan ke kecepatan
```

```
        self.velocity.limit(self.topspeed)         # batasi kecepatan maksimum
        self.location.add(self.velocity)           # posisi berubah
        self.check_edges()                         # cek jika keluar layar

        # Gambar ulang posisi mobil
        self.canvas.coords(self.id,
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius)

        # Tampilkan kecepatan saat ini
        speed = self.velocity.magnitude()
        self.canvas.itemconfig(self.speed_text, text=f"Speed: {speed:.2f}")

    def check_edges(self):
        if self.location.x > self.width:
            self.location.x = 0
        elif self.location.x < 0:
            self.location.x = self.width
        if self.location.y > self.height:
            self.location.y = 0
        elif self.location.y < 0:
            self.location.y = self.height

    def accelerate(self):
        self.acceleration = Vector(0.05, 0)  # dorong ke kanan

    def brake(self):
        self.acceleration = Vector(-0.05, 0)  # rem ke kiri

    def stop(self):
        self.acceleration = Vector(0, 0)

# ----------------------------
# Fungsi utama
# ----------------------------
def main():
    WIDTH, HEIGHT = 600, 400
    root = tk.Tk()
    root.title("Exercise 1.5 - Mobil Dorong Rem")

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

    car = Car(canvas, WIDTH, HEIGHT)

    # Fungsi animasi berulang
    def animate():
        car.update()
        root.after(30, animate)

    # Tangani tombol keyboard
    def on_key_press(event):
        if event.keysym == "Up":
            car.accelerate()
        elif event.keysym == "Down":
            car.brake()

    def on_key_release(event):
        car.stop()

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

```
    root.bind("<KeyRelease>", on_key_release)

    animate()
    root.mainloop()

main()
```

🔍 **Penjelasan Visual:**

Tombol	Apa yang terjadi
↑ Up	Mobil makin cepat ke kanan (dorong)
↓ Down	Mobil melambat atau mundur sedikit (rem)
👁 Di layar	Muncul "Speed: ..." sebagai angka kecepatannya

📝 **Kesimpulan supaya Bisa Ingat:**

1. **location = posisi bola.**
2. **velocity = kecepatan bergerak.**
3. **acceleration = percepatan = menambah kecepatan.**
4. **limit() = membatasi kecepatan agar tidak ngebut.**
5. Gunakan tombol untuk **mengontrol gerak dengan percepatan.**

2: Random Acceleration (Percepatan Acak)

Bagus! Sekarang kita masuk ke **Algorithm #2: Random Acceleration (Percepatan Acak)** — cocok banget untuk membuat gerakan seperti **"bola yang goyang-goyang sendiri"**, seperti **awan atau serangga yang tidak bisa diam**.

🧠 **Konsep Kunci :**

1. **Percepatan = perubahan kecepatan.**
2. Jika percepatan diacak terus, maka arah & kecepatan bola akan berubah-ubah juga.
3. Bola akan seperti punya "kehidupan" sendiri.
4. Kita tidak beri arah tetap — kita acak terus arah dorongannya!

🔁 **Perubahan pada update():**

Sebelumnya (percepatan tetap):

self.acceleration = Vector(-0.001, 0.01)

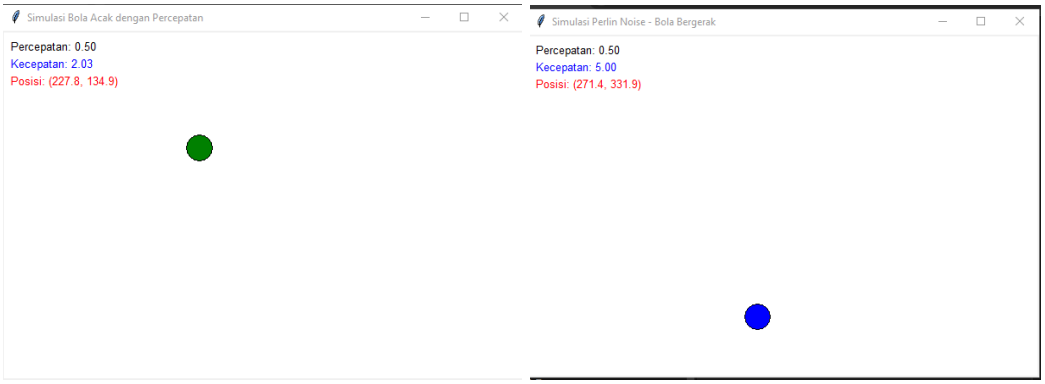
Sekarang (percepatan acak setiap frame):

self.acceleration = Vector.random2D()
self.acceleration.mult(0.5) # bisa konstan atau acak juga

🧩 **Tambahan Fungsi random2D() di Python**

Karena Python tidak punya random2D() bawaan seperti Processing, kita buat sendiri:

@staticmethod
def random2D():
  angle = random.uniform(0, 2 * math.pi)
  return Vector(math.cos(angle), math.sin(angle))



🎯 **Tujuan:**
Kita akan membuat **bola hijau yang bisa bergerak sendiri secara acak** di layar. Setiap beberapa saat:

- Bola akan mendapat **arah dorongan baru** (disebut percepatan).
- Bola akan bergerak sesuai percepatan dan kecepatan.
- Bola bisa mentok ke tepi, lalu muncul di sisi sebaliknya (seperti game Snake! 🐍).
- Kita juga **menampilkan hasil perhitungannya**: percepatan, kecepatan, dan posisi bola di layar.

✅ **Versi Lengkap Kode + Komentar**

Berikut penjelasan detail tentang alur, rumus, dan perhitungan untuk setiap frame dalam script simulasi bola acak dengan percepatan:

Alur Program Secara Umum:

1. Program membuat jendela dengan canvas putih dan sebuah bola hijau di tengah.
2. Setiap frame (30 ms), bola akan:
    - Mendapat percepatan acak baru
    - Memperbarui kecepatan berdasarkan percepatan
    - Bergerak berdasarkan kecepatan
    - Dicek apakah sudah keluar layar (jika ya, akan muncul di sisi sebaliknya)
    - Diupdate posisinya di canvas
    - Diupdate informasi teks perhitungannya

Detail Per Frame:

1. Pembuatan Percepatan Acak

- **Rumus**:

```
self.acceleration = random2D()  # Vektor dengan arah acak (magnitude = 1)
self.acceleration.mult(0.5)     # Perkalian vektor dengan skalar 0.5
```

- **Penjelasan**:
    - random2D() menghasilkan vektor satuan (magnitude = 1) dengan arah acak (0-360°).
    - Percepatan diperkecil dengan mengalikan vektor dengan 0.5, sehingga magnitude percepatan menjadi 0.5.

2. Update Kecepatan

- **Rumus**:

```
self.velocity.add(self.acceleration)  # v = v + a
```

- **Penjelasan**:
    - Kecepatan (velocity) ditambahkan dengan percepatan (acceleration).
    - Ini adalah implementasi dari hukum Newton: **Δv = a · Δt**, tetapi karena Δt dianggap 1 (per frame), cukup v += a.

3. Membatasi Kecepatan Maksimum

- **Rumus**:

```
self.velocity.limit(self.topspeed)  # topspeed = 5
```

- **Penjelasan**:

- Jika magnitude kecepatan melebihi topspeed (5), kecepatan akan dinormalisasi (dipertahankan arahnya, tetapi magnitude diubah menjadi 5).

4. Update Posisi (Gerakkan Bola)

- **Rumus**:

```
self.location.add(self.velocity)  # pos = pos + v
```

- **Penjelasan**:
  - Posisi bola diupdate dengan menambahkan kecepatan (velocity).
  - Ini adalah implementasi dari **Δx = v · Δt**, tetapi karena Δt dianggap 1, cukup pos += v.

5. Cek Batas Layar (Jika Keluar)

- **Logika**:

```
if location.x > width → location.x = 0
if location.x < 0 → location.x = width
if location.y > height → location.y = 0
if location.y < 0 → location.y = height
```

- **Penjelasan**:
  - Jika bola keluar dari kanan/kiri/atas/bawah layar, posisinya di-set ke sisi sebaliknya (efek "teleport").

6. Update Posisi Bola di Canvas

- **Rumus**:

```
canvas.coords(
   id,
   location.x - radius, location.y - radius,  # Pojok kiri atas
   location.x + radius, location.y + radius   # Pojok kanan bawah
)
```

- **Penjelasan**:
  - Menggambar ulang bola di posisi baru dengan radius yang sama.

7. Update Teks Perhitungan

- **Rumus**:

```
acc_mag = acceleration.magnitude()  # √(a.x² + a.y²)
vel_mag = velocity.magnitude()      # √(v.x² + v.y²)
pos_x, pos_y = location.x, location.y
```

- **Penjelasan**:
  - Magnitude percepatan dan kecepatan dihitung menggunakan rumus Pythagoras.
  - Nilai-nilai ini ditampilkan di layar dalam format:

```
Percepatan: [nilai]
Kecepatan: [nilai]
Posisi: (x, y)
```

Contoh Perhitungan dalam Satu Frame

Misalkan:

- **Percepatan acak**: a = (0.3, 0.4) (setelah random2D().mult(0.5))
- **Kecepatan awal**: v = (4, 3) (magnitude = 5, sudah di topspeed)
- **Posisi awal**: pos = (300, 200)

Langkah-langkah:

1. **Percepatan**:
   a = (0.3, 0.4) (magnitude = √(0.3² + 0.4²) = 0.5)
2. **Update Kecepatan**:
   v = (4 + 0.3, 3 + 0.4) = (4.3, 3.4)
   Magnitude baru = √(4.3² + 3.4²) ≈ 5.5 (melebihi topspeed)
3. **Limit Kecepatan**:
   Normalisasi: v = (4.3/5.5 * 5, 3.4/5.5 * 5) ≈ (3.91, 3.09)
   Sekarang magnitude = 5.

4. **Update Posisi**:
   pos = (300 + 3.91, 200 + 3.09) ≈ (303.91, 203.09)
5. **Tampilkan Nilai**:
   - Percepatan: 0.5
   - Kecepatan: 5.0
   - Posisi: (303.9, 203.1)

Kesimpulan

- Setiap frame, bola mendapatkan percepatan acak kecil.
- Kecepatan berubah berdasarkan percepatan, tetapi tidak melebihi topspeed.
- Posisi diupdate berdasarkan kecepatan.
- Jika bola keluar layar, muncul di sisi sebaliknya.
- Nilai percepatan, kecepatan, dan posisi ditampilkan di layar.

```
# SCRIPT 20 - Simulasi Bola Acak dengan Percepatan
import tkinter as tk
import math
from vector import Vector

def random2D():
    angle = random.uniform(0, 2 * math.pi)
    return Vector(math.cos(angle), math.sin(angle))

# ----------------------------
# Kelas Mover: Bola yang bisa bergerak
# ----------------------------
class Mover:
    def __init__(self, canvas, width, height):
        self.canvas = canvas
        self.width = width
        self.height = height
        self.radius = 15

        self.location = Vector(width // 2, height // 2)  # posisi awal
        self.velocity = Vector(0, 0)                     # kecepatan awal nol
        self.acceleration = Vector(0, 0)                 # percepatan awal nol
        self.topspeed = 5                                # kecepatan maksimum

        # Gambar bola
        self.id = canvas.create_oval(
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius,
            fill="green"
        )

        # Teks perhitungan di layar
        self.text_acc = canvas.create_text(10, 10, anchor="nw", text="", font=("Arial",
10), fill="black")
        self.text_vel = canvas.create_text(10, 30, anchor="nw", text="", font=("Arial",
10), fill="blue")
        self.text_pos = canvas.create_text(10, 50, anchor="nw", text="", font=("Arial",
10), fill="red")

    def update(self):
        # 1. Buat percepatan acak
        self.acceleration = random2D()
        self.acceleration.mult(0.5)  # percepatan acak tapi kecil

        # 2. Tambahkan percepatan ke kecepatan
```

```
        self.velocity.add(self.acceleration)

        # 3. Batasi kecepatan agar tidak terlalu cepat
        self.velocity.limit(self.topspeed)

        # 4. Tambahkan kecepatan ke posisi (gerakkan bola)
        self.location.add(self.velocity)

        # 5. Jika keluar layar, pindah ke sisi sebaliknya
        self.check_edges()

        # 6. Gambar ulang posisi bola
        self.canvas.coords(self.id,
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius)

        # 7. Tampilkan perhitungan di layar
        acc_mag = self.acceleration.magnitude()
        vel_mag = self.velocity.magnitude()
        pos_x, pos_y = self.location.x, self.location.y

        self.canvas.itemconfig(self.text_acc, text=f"Percepatan: {acc_mag:.2f}")
        self.canvas.itemconfig(self.text_vel, text=f"Kecepatan: {vel_mag:.2f}")
        self.canvas.itemconfig(self.text_pos, text=f"Posisi: ({pos_x:.1f},
{pos_y:.1f})")

    def check_edges(self):
        # Jika bola keluar layar, masuk dari sisi sebaliknya
        if self.location.x > self.width:
            self.location.x = 0
        elif self.location.x < 0:
            self.location.x = self.width
        if self.location.y > self.height:
            self.location.y = 0
        elif self.location.y < 0:
            self.location.y = self.height

# ----------------------------
# Fungsi utama untuk menjalankan animasi
# ----------------------------
def main():
    WIDTH, HEIGHT = 600, 400
    root = tk.Tk()
    root.title("Simulasi Bola Acak dengan Percepatan")
    canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
    canvas.pack()

    mover = Mover(canvas, WIDTH, HEIGHT)

    def animate():
        mover.update()          # update posisi dan kecepatan
        root.after(30, animate) # ulangi tiap 30 ms

    animate()
    root.mainloop()

main()
```

🧪 **Contoh Tampilan Hasil:**
Percepatan: 0.50

Kecepatan: 3.25
Posisi: (312.3, 200.7)

🧠 **Ringkasan Konsep Fisika:**

Istilah	**Artinya**
Percepatan	Arah dan kekuatan dorongan baru (acak)
Kecepatan	Seberapa cepat bola bergerak
Posisi	Di mana bola berada di layar
Limit	Batas maksimal kecepatan supaya tidak terlalu cepat dan sulit dikontrol

---

🧠 **Penjelasan Mudah:**

- random2D() membuat vektor dengan **arah acak** tapi panjang 1.
- Lalu kita **kalikan (mult)** dengan 0.5 agar tidak terlalu kuat dorongannya.
- Setiap frame, arah dorongan berubah. Hasilnya? Bola terlihat **bergerak liar**, seperti hidup.

Exercise 1.6: **menggerakkan bola dengan percepatan berdasarkan *Perlin Noise***.

🧠 **Apa itu *Perlin Noise*?**
"Perlin Noise itu seperti *random* yang lebih halus."
Kalau random() seperti petasan — meledak ke mana-mana tiap detik — maka Perlin Noise seperti gelombang angin atau ombak — **pelan berubah, halus**, tidak kaget-kaget.
Contohnya:

- random() = arah: kanan → kiri → atas → bawah (acak banget!)
- Perlin Noise = arah: pelan belok kiri → terus pelan belok kanan → pelan belok ke bawah

---

💡 **Tujuan Exercise:**

- Menggunakan *Perlin noise* untuk menentukan arah percepatan (yang halus berubah).
- Ini membuat gerakan bola seperti **awan tertiup angin** atau **ikan berenang santai**.

Berikut ini penjelasan ulang **untuk anak SMP** beserta **versi revisi kode lengkap** yang:
✅ **Tidak memakai noise.pnoise1**, tapi menggunakan **modul perlin_noise**
✅ **Menampilkan hasil perhitungan di layar**
✅ **Disertai penjelasan per baris agar mudah dipahami**

🧠 **Penjelasan Konsep (Untuk Anak SMP)**
Apa itu Perlin noise?

- Perlin noise adalah angka acak **yang halus berubah dari waktu ke waktu**, seperti gelombang.
- Dibanding angka acak biasa (yang loncat-loncat), Perlin noise seperti perubahan cuaca yang perlahan. 🌤️🌧️

Apa yang kita lakukan?

- Kita akan buat bola yang bergerak **didorong oleh arah dari Perlin noise**.
- Arah dorongan akan berubah **perlahan**, sehingga gerakan bola terlihat **lebih alami** seperti ditiup angin.

✅ **Kode Lengkap dengan Modul perlin_noise**
Sebelum jalankan, install dulu modul:
pip install perlin-noise
🦴 **Kode:**

```
# SCRIPT 21 - Simulasi Pergerakan Bola dengan Perlin Noise
import tkinter as tk
import math
import random
from perlin_noise import PerlinNoise  # Pakai modul perlin_noise
from vector import Vector

# ----------------------------
# Kelas Mover: Bola biru yang bergerak
# ----------------------------
class Mover:
    def __init__(self, canvas, width, height, noise_gen):
        self.canvas = canvas
        self.noise = noise_gen
        self.width = width
        self.height = height
        self.radius = 15

        self.location = Vector(width // 2, height // 2)
        self.velocity = Vector(0, 0)
        self.acceleration = Vector(0, 0)
        self.topspeed = 5

        self.noise_offset = random.uniform(0, 1000)

        self.id = canvas.create_oval(
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius,
            fill="blue"
        )

        # Teks perhitungan di layar
        self.text_acc = canvas.create_text(10, 10, anchor="nw", text="", font=("Arial",
10), fill="black")
        self.text_vel = canvas.create_text(10, 30, anchor="nw", text="", font=("Arial",
10), fill="blue")
        self.text_pos = canvas.create_text(10, 50, anchor="nw", text="", font=("Arial",
10), fill="red")

    def update(self, t):
        # 1. Dapatkan angka halus dari Perlin noise (hasil 0.0 - 1.0)
        noise_val = self.noise(t + self.noise_offset)
        angle = noise_val * 2 * math.pi  # ubah jadi sudut (0 - 2π)

        # 2. Buat vektor percepatan dari sudut
        self.acceleration = Vector(math.cos(angle), math.sin(angle))
        self.acceleration.mult(0.5)

        # 3. Tambahkan percepatan ke kecepatan
        self.velocity.add(self.acceleration)
        self.velocity.limit(self.topspeed)

        # 4. Tambahkan kecepatan ke posisi
        self.location.add(self.velocity)
        self.check_edges()

        # 5. Gambar ulang bola
        self.canvas.coords(self.id,
            self.location.x - self.radius, self.location.y - self.radius,
            self.location.x + self.radius, self.location.y + self.radius)
```

```
        # 6. Tampilkan nilai-nilai di layar
        acc_mag = self.acceleration.magnitude()
        vel_mag = self.velocity.magnitude()
        pos_x, pos_y = self.location.x, self.location.y

        self.canvas.itemconfig(self.text_acc, text=f"Percepatan: {acc_mag:.2f}")
        self.canvas.itemconfig(self.text_vel, text=f"Kecepatan: {vel_mag:.2f}")
        self.canvas.itemconfig(self.text_pos, text=f"Posisi: ({pos_x:.1f},
{pos_y:.1f})")

    def check_edges(self):
        # Jika bola keluar layar, masuk dari sisi sebaliknya
        if self.location.x > self.width:
            self.location.x = 0
        elif self.location.x < 0:
            self.location.x = self.width
        if self.location.y > self.height:
            self.location.y = 0
        elif self.location.y < 0:
            self.location.y = self.height

# ----------------------------
# Fungsi utama
# ----------------------------
def main():
    WIDTH, HEIGHT = 600, 400
    root = tk.Tk()
    root.title("Simulasi Perlin Noise - Bola Bergerak")
    canvas = tk.Canvas(root, width=WIDTH, height=HEIGHT, bg="white")
    canvas.pack()

    noise = PerlinNoise(octaves=1)  # Buat generator Perlin noise
    mover = Mover(canvas, WIDTH, HEIGHT, noise)
    t = [0.0]  # Waktu sebagai input noise

    def animate():
        mover.update(t[0])
        t[0] += 0.01  # Tambah waktu perlahan agar perubahan halus
        root.after(30, animate)

    animate()
    root.mainloop()

main()
```

🧪 **Contoh Tampilan Hasil di Layar:**
Percepatan: 0.50
Kecepatan: 2.74
Posisi: (303.2, 198.1)

🧠 **Kesimpulan Sederhana:**

Konsep	**Penjelasan**
PerlinNoise	Untuk menghasilkan angka acak yang berubah **halus**
angle = noise * 2π	Mengubah nilai 0–1 jadi sudut arah gerakan
cos(angle), sin(angle)	Mengubah arah ke bentuk vektor (percepatan)
.limit()	Agar kecepatan tidak terlalu besar

Konsep	Penjelasan
.add()	Menambahkan vektor arah ke kecepatan dan posisi

✨ **Hasil:**

- Bola akan **bergerak dengan arah halus berubah**, seolah-olah seperti **awan tertiup angin lembut**.
- Gerakan ini sangat berbeda dari random, karena *tidak menyentak*.

🧠 Ringkasan Belajar Gerak Bola di Komputer (Motion 101 dengan Tkinter & Vektor)

---

1. 🎲 Apa itu *Random*?

Random = acak.
Seperti kalau kamu melempar dadu, hasilnya tidak bisa ditebak.
Di komputer, kita bisa pakai random() untuk bikin arah atau kecepatan yang acak.
Contoh:

- Bola tiba-tiba ke kanan.
- Lalu tiba-tiba ke atas.
- Gerakannya jadi **kasar atau menyentak**.

---

2. ⚙️ Custom Random

Kita bisa bikin acakan sendiri, misalnya:

- Hanya ke kanan atau ke kiri.
- Atau pakai acakan yang **lebih halus** seperti gelombang.

---

3. 🌊 Apa itu *Noise* (Perlin Noise)?

Noise itu kayak random, tapi **lebih halus dan teratur**.
Bayangkan:

- Random = badai petir 💥 (arahnya selalu berubah-ubah)
- Noise = angin sepoi-sepoi 🍃 (pelan berubah, seperti ombak)

Dengan *noise*, bola bisa:

- Bergerak santai seperti awan
- Tidak membuat kita pusing lihatnya 😄

---

4. 🎈 Apa itu *Vector* (Vektor)?

Vektor = panah yang punya **arah** dan **panjang** (besarnya).
Di dunia kita, vektor bisa:

- Menunjukkan arah angin
- Menunjukkan kecepatan mobil

Di program kita, kita pakai vektor untuk:

- **Posisi bola** → location
- **Arah dan kecepatan gerak** → velocity
- **Percepatan (perubahan arah dan kecepatan)** → acceleration

---

5. 🧭 Apa itu *Location*?

Lokasi = posisi bola sekarang (x, y).
Kalau kamu main game, karakter kamu juga punya posisi di layar. Itulah location.

---

6. 💨 Apa itu *Velocity*?

Velocity = seberapa cepat dan ke mana bola bergerak.
Misal:

- Bola punya velocity ke kanan = makin lama makin jauh ke kanan.

Kita tulis di kode:
location.add(velocity)

---

7. 🚀 Apa itu *Acceleration*?
Acceleration = perubahan kecepatan (velocity).
Kalau kamu naik sepeda dan mulai mengayuh lebih cepat → itu **accelerating**.
Kita tulis:
velocity.add(acceleration)
Lalu karena velocity berubah, posisi juga berubah!

---

8. 🔁 Trickle-Down Effect
Gerakan bola terjadi karena **rantai perubahan**:
acceleration → velocity → location
Jadi kita cukup atur **acceleration**, sisanya mengikuti otomatis!
Misalnya:

- Arah percepatan dari mouse
- Arah percepatan dari noise
- Arah percepatan dari tombol keyboard

📐 Belajar Matematika yang Terkait
Dengan belajar motion ini, kamu juga belajar **konsep matematika yang keren**:

Konsep Matematika	Penjelasan Sederhana
Koordinat (x, y)	Posisi benda di layar
Vektor	Panah arah gerak
Penjumlahan vektor	Gabung arah dan kecepatan
Magnitude (panjang vektor)	Seberapa cepat atau kuat
Trigonometri (cos, sin)	Untuk ubah sudut jadi arah
Limit	Batasi kecepatan agar tidak terlalu cepat
Noise / fungsi	Pelajari fungsi yang halus berubah

🎮 Kenapa Seru Dipelajari?

- Bisa bikin game sendiri!
- Bisa bikin simulasi bola, mobil, ikan, atau burung!
- Bisa jadi dasar untuk robot, AI, atau fisika komputer!

🚀 Lanjut Belajar Apa?

- Belajar cara gerak ke arah mouse 📍
- Buat banyak bola bergerak bersamaan ⚽⚽⚽
- Buat interaksi pakai keyboard (mobil, pelari) 🏃
- Belajar *collisions* atau tumbukan 💥
- Gabungkan dengan *game engine* atau animasi

1.9 Static vs. Non-Static Functions

Oke! Mari kita jelaskan topik **"Static vs Non-Static Functions"** ini dengan sederhana agar bisa dipahami oleh **anak SMP**, terutama jika sedang belajar **Python dengan Tkinter**.

📌 Konsep Dasar: Fungsi Biasa vs Fungsi Static
Sebelum masuk ke kodingnya, kita pahami dulu apa maksudnya.

Bayangkan kamu punya **dua buah kotak** yang berisi angka. Kadang kamu ingin **menambahkan isi kotak B ke kotak A langsung**, dan kadang kamu ingin **membuat kotak baru yang isinya gabungan dari kotak A dan B, tanpa mengubah isi A atau B**.
Nah, itu bedanya fungsi biasa (non-static) dan fungsi static!

🔍 Contoh di Dunia Nyata (dalam kode Python)

😈 Fungsi Non-Static (ubah langsung objeknya)

```
# SCRIPT 22 - Fungsi Non-Static (ubah langsung objeknya)
from vector import Vector

titik1 = Vector(0, 0)
titik2 = Vector(4, 5)

titik1.add(titik2)
print("titik1 x:",titik1.x, ", titik1 y:", titik1.y)  # Hasil: 4 5 (karena titik1
diubah)
```

📦 Fungsi Static (buat objek baru, tidak ubah yang lama)

```
# SCRIPT 23 - Fungsi Static (buat objek baru, tidak ubah yang lama)
from vector import Vector

titik1 = Vector(1, 2)
titik2 = Vector(4, 5)

titik3 = Vector.added(titik1, titik2)
print("titik3 x:",titik3.x, ", titik3 y:", titik3.y)  # Hasil: 5 7
print("titik1 x:",titik1.x, ", titik1 y:", titik1.y)  # Masih 1 2, tidak berubah
```

🧠 Kesimpulan:

Fungsi Non-Static (biasa)	Fungsi Static
Dipanggil dari objek (objek.fungsi())	Dipanggil dari class (Class.fungsi())
Mengubah isi objek itu sendiri	Tidak mengubah objek lama, hasil disimpan di objek baru
Contoh: kotak1.tambah(kotak2)	Contoh: Kotak.tambah(kotak1, kotak2)

🖼️ Dalam Proyek Tkinter (Misalnya: Simulasi Gerak)

Kamu bisa pakai konsep ini untuk **mengatur posisi objek** di layar, seperti bola yang bergerak mengikuti mouse, tanpa mengubah posisi lama.
Contoh ringkas Tkinter:

```
# SCRIPT 24
from vector import Vector

# Contoh penggunaan
v1 = Vector(1, 2)
v2 = Vector(3, 4)
v3 = Vector.added(v1, v2)

print(v3.x, v3.y)  # 4, 6
```

🧠 Soal Exercise 1.7

Kita diminta **mengubah pseudocode** ini jadi kode Python, menggunakan **static atau non-static functions** yang sesuai:
1. PVector v = new PVector(1,5);
2. PVector u = v multiplied by 2;

3. PVector w = v minus u;
4. Divide w by 3;

Langkah-langkah Penjelasan

1. Buat v dengan nilai (1, 5)

v = Vektor(1, 5)

2. Buat u = v * 2

Nah! Kita tidak ingin mengubah v, jadi kita **pakai fungsi static** untuk multiplied by 2:

u = Vektor.kali_static(v, 2)

3. Buat w = v - u

Lagi-lagi, kita ingin **menghasilkan vektor baru**, maka pakai static function:

w = Vektor.kurang_static(v, u)

4. Bagi w dengan 3

Karena kita **sudah punya w**, kita bisa ubah isinya langsung (pakai non-static):

w.bagi(3)

Kode Python Lengkap

```
# SCRIPT 25 -
from vector import Vector

# Exercise 1.7
v = Vector(1, 5)
u = Vector.multed(v, 2)    # u = v * 2
w = Vector.subbed(v, u)    # w = v - u
w.div(3)                   # w = w / 3

print("v =", v) #v = (1.00, 5.00)
print("u =", u) #u = (2.00, 10.00)
print("w =", w) #w = (-0.33, -1.67)
```

Kesimpulan :

- Kalau ingin **mengubah objek itu sendiri**, pakai **fungsi biasa (non-static)**.
- Kalau ingin **membuat objek baru**, pakai **fungsi static**.
- Di Python, kamu bisa latihan membuat class seperti Vektor untuk belajar gerak, posisi, atau koordinat.

1.10 Interactivity with Acceleration

Oke! Kita sekarang masuk ke **materi yang seru banget**: membuat **interaksi dengan mouse dan percepatan (acceleration)** di Python dengan **Tkinter**, menggunakan konsep **vektor**.

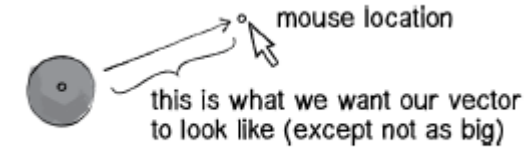


Figure 1.14

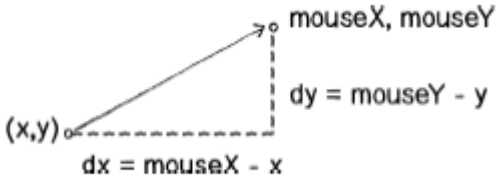
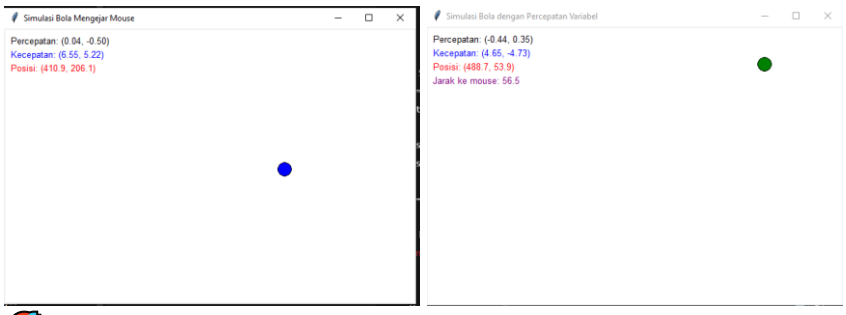


Figure 1.15

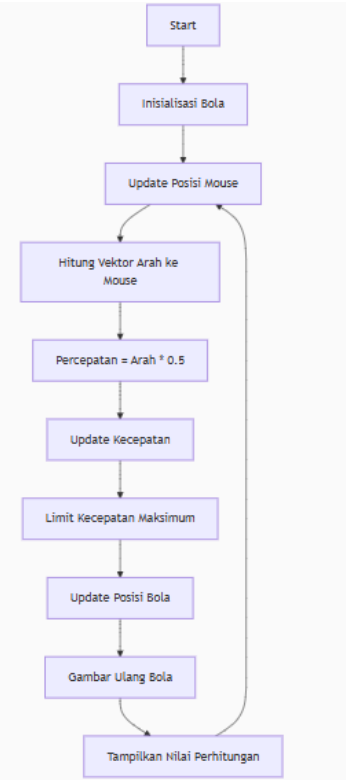


🎯 **Tujuan:**

Kita akan buat **sebuah bola** yang **bergerak menuju mouse**, menggunakan **konsep percepatan** (acceleration) yang dihitung dengan vektor.

1. Diagram Alur Program

Diagram



2. Rumus dan Perhitungan Tiap Frame

Variabel Utama:

- location: Posisi bola (Vector(x, y))
- velocity: Kecepatan bola (Vector(vx, vy))
- acceleration: Percepatan bola (Vector(ax, ay))
- topspeed: Kecepatan maksimum (10)

Langkah-Langkah Tiap Frame:

1. Hitung Arah ke Mouse (direction**)**

- **Rumus**:

direction = Vector(mouse_x, mouse_y).sub(self.location)

- **Contoh**:
  - Jika mouse di (400, 300) dan bola di (300, 200):

direction = (400-300, 300-200) = (100, 100)

2. Normalisasi Arah dan Skala Percepatan

- **Rumus**:

direction.normalize().mult(0.5) *# Normalisasi lalu kali 0.5*
self.acceleration = direction

- **Contoh**:
  - direction = (100, 100) → magnitude = √(100² + 100²) ≈ 141.42
  - Normalisasi: (100/141.42, 100/141.42) ≈ (0.707, 0.707)
  - Percepatan: (0.707*0.5, 0.707*0.5) ≈ (0.35, 0.35)

3. Update Kecepatan (velocity**)**

- **Rumus**:

self.velocity.add(self.acceleration) *# v = v + a*

- **Contoh**:
  - Jika velocity = (2, 3) dan acceleration = (0.35, 0.35):

velocity = (2 + 0.35, 3 + 0.35) = (2.35, 3.35)

4. Batasi Kecepatan Maksimum (topspeed = 10**)**

- **Rumus**:

```
if velocity.mag() > topspeed:
   velocity.normalize().mult(topspeed)
```

- **Contoh**:
  - Jika velocity = (8, 6) → magnitude = 10 (tidak diubah).
  - Jika velocity = (9, 12) → magnitude = 15 → dinormalisasi:

velocity = (9/15 * 10, 12/15 * 10) = (6, 8)

5. Update Posisi (location**)**

- **Rumus**:

self.location.add(self.velocity) *# pos = pos + v*

- **Contoh**:
    - Jika location = (300, 200) dan velocity = (2, 3):

location = (300 + 2, 200 + 3) = (302, 203)

6. Gambar Ulang Bola di Canvas

- **Rumus**:

```
canvas.coords(
   id,
   x - radius, y - radius,  # Pojok kiri atas
   x + radius, y + radius   # Pojok kanan bawah
)
```

- **Contoh**:
    - Jika location = (302, 203) dan radius = 10:

canvas.coords(id, 292, 193, 312, 213)

7. Tampilkan Nilai Perhitungan

- **Rumus**:

ax, ay = acceleration.x, acceleration.y
vx, vy = velocity.x, velocity.y
px, py = location.x, location.y

- **Contoh Output**:

Percepatan: (0.35, 0.35)
Kecepatan: (2.35, 3.35)
Posisi: (302.0, 203.0)

---

3. Contoh Perhitungan Lengkap dalam 1 Frame

Kondisi Awal:

- Posisi bola: (300, 200)
- Kecepatan bola: (2, 3)
- Posisi mouse: (400, 300)

Langkah-Langkah:

1. **Hitung** direction:

direction = (400-300, 300-200) = (100, 100)

2. **Normalisasi & Percepatan**:

direction ≈ (0.707, 0.707) → acceleration = (0.35, 0.35)

3. **Update Kecepatan**:

velocity = (2 + 0.35, 3 + 0.35) = (2.35, 3.35)

4. **Cek** topspeed:
    - Magnitude velocity = $\sqrt{(2.35^2 + 3.35^2)} \approx 4.09$ (masih di bawah 10, tidak dibatasi).
5. **Update Posisi**:

location = (300 + 2.35, 200 + 3.35) ≈ (302.35, 203.35)

6. **Output Teks**:

Percepatan: (0.35, 0.35)
Kecepatan: (2.35, 3.35)
Posisi: (302.4, 203.4)

---

4. Kesimpulan

- Bola bergerak menuju mouse dengan **percepatan konstan (**0.5**)**.
- Kecepatan bertambah searah vektor menuju mouse, tetapi **tidak melebihi** topspeed.
- Posisi diupdate setiap frame berdasarkan kecepatan.
- Nilai percepatan, kecepatan, dan posisi ditampilkan di layar.

💻 **Versi Kode Python Lengkap dengan Keterangan + Tampilan Perhitungan**

```
# SCRIPT 26 - Simulasi Bola Mengejar Mouse dengan Percepatan
import tkinter as tk
from vector import Vector
```

```
class Bola:
    def __init__(self, canvas, x, y):
        self.canvas = canvas
        self.location = Vector(x, y)       # Posisi bola sebagai Vector
        self.velocity = Vector(0, 0)       # Kecepatan awal
        self.acceleration = Vector(0, 0)   # Percepatan awal
        self.radius = 10
        self.topspeed = 10                 # Kecepatan maksimum

        self.id = canvas.create_oval(
            x - self.radius, y - self.radius,
            x + self.radius, y + self.radius,
            fill='blue'
        )

        # Teks untuk menampilkan hasil perhitungan
        self.text_acc = canvas.create_text(10, 10, anchor='nw', text="", font=('Arial',
10), fill='black')
        self.text_vel = canvas.create_text(10, 30, anchor='nw', text="", font=('Arial',
10), fill='blue')
        self.text_pos = canvas.create_text(10, 50, anchor='nw', text="", font=('Arial',
10), fill='red')

    def update(self, mouse_x, mouse_y):
        # Hitung arah dari bola ke mouse sebagai Vector
        mouse_pos = Vector(mouse_x, mouse_y)
        direction = mouse_pos.copy().sub(self.location)

        # Normalisasi arah dan skala percepatan
        direction.normalize().mult(0.5)
        self.acceleration = direction

        # Tambahkan percepatan ke kecepatan
        self.velocity.add(self.acceleration)

        # Batasi kecepatan maksimum
        speed = self.velocity.mag()
        if speed > self.topspeed:
            self.velocity.normalize().mult(self.topspeed)

        # Update posisi dengan kecepatan
        self.location.add(self.velocity)

        # Update posisi bola di canvas
        x, y = self.location.x, self.location.y
        self.canvas.coords(self.id, x - self.radius, y - self.radius, x + self.radius,
y + self.radius)

        # Tampilkan perhitungan di layar
        ax, ay = self.acceleration.x, self.acceleration.y
        vx, vy = self.velocity.x, self.velocity.y
        px, py = self.location.x, self.location.y

        self.canvas.itemconfig(self.text_acc, text=f"Percepatan: ({ax:.2f}, {ay:.2f})")
        self.canvas.itemconfig(self.text_vel, text=f"Kecepatan: ({vx:.2f}, {vy:.2f})")
        self.canvas.itemconfig(self.text_pos, text=f"Posisi: ({px:.1f}, {py:.1f})")

# Setup Tkinter
root = tk.Tk()
root.title("Simulasi Bola Mengejar Mouse")
```

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

bola = Bola(canvas, 300, 200)

def animasi():
    x = canvas.winfo_pointerx() - canvas.winfo_rootx()
    y = canvas.winfo_pointery() - canvas.winfo_rooty()
    bola.update(x, y)
    root.after(16, animasi)  # sekitar 60 fps

animasi()
root.mainloop()
```

📺 **Contoh Hasil Tampilan di Layar:**
Percepatan: (0.35, 0.28)
Kecepatan: (4.57, 3.84)
Posisi: (358.2, 243.5)

🧠 **Kesimpulan Sederhana**

Langkah	Penjelasan
1 Hitung dx, dy	Arah dari bola ke mouse
2 Normalisasi	Biar arah punya panjang 1
3 Kali 0.5	Supaya percepatannya kecil
4 Tambah percepatan ke kecepatan	Biar makin cepat
5 Batasi kecepatan	Biar gak ngebut kebablasan
6 Update posisi	Bola bergerak sedikit demi sedikit ke arah mouse

---

💡 **Penjelasan Sederhana**

- Bola tahu **di mana letak mouse**.
- Dia akan **bergerak sedikit demi sedikit ke arah mouse**, makin cepat karena percepatannya ditambahkan tiap frame.
- Tapi kecepatannya **dibatasi**, biar tidak terbang terlalu cepat.

📚 **Apa yang Dipelajari dari Ini?**

- Konsep **vektor dan arah**.
- Percepatan dan kecepatan dalam **fisika sederhana**.
- **Interaksi mouse**.
- Dasar **animasi dan simulasi gerak**.
- Cara **membuat aplikasi GUI** dengan Python Tkinter.

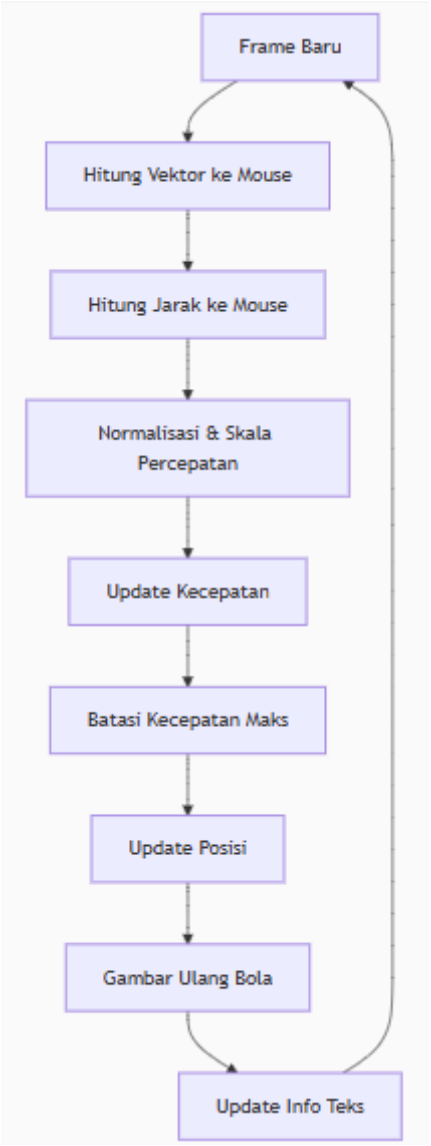
Keren! Sekarang kita akan **lanjut ke Exercise 1.8**, yaitu membuat **percepatan bola berubah-ubah** tergantung jaraknya ke mouse:

🎯 **Tujuan Exercise 1.8:**
Buat bola yang tetap **bergerak ke arah mouse**, **tapi kecepatan perubahannya (percepatan) lebih kuat saat dekat atau jauh** dari mouse.
Kita akan mengubah magnitude dari acceleration berdasarkan **jarak** bola ke mouse.

🧠 **Konsep Fisika:**
Percepatan tergantung jarak:
Kita akan eksperimen dua gaya:

- **Lebih cepat saat DEKAT ke mouse** (seperti gaya magnet kecil).
- **Lebih cepat saat JAUH dari mouse** (seperti gravitasi kuat dari jauh).

Rumus dan Perhitungan Tiap Frame



1. Variabel Utama

- location: Posisi bola (Vector)
- velocity: Kecepatan bola (Vector)
- acceleration: Percepatan bola (Vector)
- topspeed: 10 (batas kecepatan maksimum)
- radius: 10 (ukuran bola)

2. Mekanisme Utama

A. Hitung Vektor Arah ke Mouse
direction = Vector(mouse_x, mouse_y) - self.location
Contoh:

- Jika mouse di (400, 300) dan bola di (300, 200):

direction = (400-300, 300-200) = (100, 100)
B. Hitung Jarak ke Mouse
distance = direction.mag() *# √(x² + y²)*
Contoh:

- distance = √(100² + 100²) ≈ 141.42

C. Skala Percepatan Berdasarkan Jarak
direction.normalize().mult(distance * 0.01)
Perhitungan:

1. Normalisasi:

direction = (100/141.42, 100/141.42) ≈ (0.707, 0.707)

2. Skala:

acceleration = (0.707, 0.707) * 141.42 * 0.01 ≈ (1.0, 1.0)

D. Update Kecepatan
self.velocity.add(self.acceleration)
Contoh:

- Jika velocity = (2.0, 1.5) dan acceleration = (1.0, 1.0):

velocity = (2.0+1.0, 1.5+1.0) = (3.0, 2.5)
E. Batasi Kecepatan Maksimum

```
if speed > topspeed:
  velocity.normalize().mult(topspeed)
```

Contoh:

- Jika velocity = (8, 6) (magnitude=10):
  - Tidak diubah karena sudah tepat topspeed
- Jika velocity = (9, 12) (magnitude=15):
  - Normalisasi: (9/15, 12/15) = (0.6, 0.8)
  - Dikali topspeed: (0.6*10, 0.8*10) = (6, 8)

F. Update Posisi
self.location.add(self.velocity)
Contoh:

- Jika location = (300, 200) dan velocity = (3.0, 2.5):

location = (300+3.0, 200+2.5) = (303.0, 202.5)

Contoh Perhitungan Frame-by-Frame
Frame 1:

- Posisi bola: (300, 200)

- Posisi mouse: (400, 300)

1. direction = (100, 100)
2. distance ≈ 141.42
3. acceleration ≈ (1.0, 1.0)
4. Jika velocity awal (0,0):
    - velocity = (1.0, 1.0)
5. Posisi baru: (301.0, 201.0)

Frame 2:

- Posisi bola: (301.0, 201.0)
- Mouse bergerak ke (410, 310)

1. direction = (109, 109)
2. distance ≈ 154.15
3. acceleration ≈ (1.09, 1.09) (karena 154.15*0.01≈1.54; 1.54*0.707≈1.09)
4. velocity = (1.0+1.09, 1.0+1.09) = (2.09, 2.09)
5. Posisi baru: (303.09, 203.09)

Visualisasi Gerakan

```
Layar (600x400):
       (0,0)                    (600,0)
         +-----------------------+
         |                       |
         |   ●  →→→ mengejar 🖱  |
         |                       |
         +-----------------------+
       (0,400)                  (600,400)
```

Karakteristik unik:

1. **Percepatan Dinamis**: Semakin jauh dari mouse, semakin besar percepatan (distance*0.01)
2. **Gerakan Smooth**: Kecepatan dibatasi untuk menghindari gerakan tiba-tiba
3. **Fisika Realistis**: Mengimplementasikan konsep vektor fisika sebenarnya

---

Persamaan Matematika Kunci

1. **Vektor Arah**:

direction = mouse_pos - bola_pos

2. **Percepatan**:

acceleration = normalize(direction) * distance * 0.01

3. **Update Kecepatan**:

velocity = velocity + acceleration

4. **Limit Kecepatan**:

if |velocity| > topspeed:
  velocity = (velocity/|velocity|) * topspeed

5. **Update Posisi**:

position = position + velocity

Output Teks Informasi

Setiap frame menampilkan:

1. Percepatan: (ax, ay)
2. Kecepatan: (vx, vy)
3. Posisi: (x, y)
4. Jarak ke mouse: distance

Contoh Output:

Percepatan: (1.09, 1.09)
Kecepatan: (2.09, 2.09)
Posisi: (303.1, 203.1)
Jarak ke mouse: 154.2

Dengan mekanisme ini, bola akan:

- Bergerak lebih cepat saat jauh dari mouse
- Bergerak lebih lambat saat mendekati mouse
- Berhenti tepat saat mencapai posisi mouse

```
# SCRIPT 27 - Simulasi Bola dengan Percepatan Variabel
import tkinter as tk
from vector import Vector

class Bola:
    def __init__(self, canvas, x, y):
        self.canvas = canvas
        self.location = Vector(x, y)
        self.velocity = Vector(0, 0)
        self.acceleration = Vector(0, 0)
        self.radius = 10
        self.topspeed = 10

        self.id = canvas.create_oval(
            x - self.radius, y - self.radius,
            x + self.radius, y + self.radius,
            fill='green'
        )

        # Teks info
        self.text_acc = canvas.create_text(10, 10, anchor='nw', text="", font=('Arial',
10), fill='black')
        self.text_vel = canvas.create_text(10, 30, anchor='nw', text="", font=('Arial',
10), fill='blue')
        self.text_pos = canvas.create_text(10, 50, anchor='nw', text="", font=('Arial',
10), fill='red')
        self.text_dist = canvas.create_text(10, 70, anchor='nw', text="",
font=('Arial', 10), fill='purple')

    def update(self, mouse_x, mouse_y):
        mouse_pos = Vector(mouse_x, mouse_y)
        direction = mouse_pos.copy().sub(self.location)

        distance = direction.mag()
        if distance == 0:
            return  # Sudah di mouse, tidak bergerak

        # Normalisasi dan skala percepatan sesuai jarak
        direction.normalize().mult(distance * 0.01)
        self.acceleration = direction

        self.velocity.add(self.acceleration)

        speed = self.velocity.mag()
        if speed > self.topspeed:
            self.velocity.normalize().mult(self.topspeed)

        self.location.add(self.velocity)

        x, y = self.location.x, self.location.y
        self.canvas.coords(self.id, x - self.radius, y - self.radius, x + self.radius,
y + self.radius)

        # Update teks info
```

```
        ax, ay = self.acceleration.x, self.acceleration.y
        vx, vy = self.velocity.x, self.velocity.y
        px, py = self.location.x, self.location.y

        self.canvas.itemconfig(self.text_acc, text=f"Percepatan: ({ax:.2f}, {ay:.2f})")
        self.canvas.itemconfig(self.text_vel, text=f"Kecepatan: ({vx:.2f}, {vy:.2f})")
        self.canvas.itemconfig(self.text_pos, text=f"Posisi: ({px:.1f}, {py:.1f})")
        self.canvas.itemconfig(self.text_dist, text=f"Jarak ke mouse: {distance:.1f}")

# Setup Tkinter
root = tk.Tk()
root.title("Simulasi Bola dengan Percepatan Variabel")

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

bola = Bola(canvas, 300, 200)

def animasi():
    x = canvas.winfo_pointerx() - canvas.winfo_rootx()
    y = canvas.winfo_pointery() - canvas.winfo_rooty()
    bola.update(x, y)
    root.after(16, animasi)

animasi()
root.mainloop()
```

🧪 **Contoh Hasil di Layar**
Percepatan: (0.34, 0.22)
Kecepatan: (2.54, 1.68)
Posisi: (310.5, 206.7)
Jarak ke mouse: 34.2

💡 **Simpulan Konsep**

Langkah	**Penjelasan Anak SMP**
🧭 Arah	Bola tahu ke mana arah mouse
📏 Jarak	Bola tahu seberapa jauh mouse dari dia
⚡ Percepatan	Semakin jauh, bola makin cepat mengejar
🌀 Kecepatan	Bola tambah cepat tiap frame
🚫 Batasan	Tapi kecepatannya dibatasi
📍 Posisi	Bola pindah sedikit demi sedikit ke arah mouse

Kalau kamu ingin coba versi lain seperti **percepatan lebih kuat saat DEKAT** (bukan jauh), tinggal ganti rumus di bagian:
# magnitude = distance * 0.01 ← lebih kuat saat jauh
# atau:
magnitude = max(10 - distance * 0.05, 0) # ← lebih kuat saat dekat

---

Script 24 - Simulasi Bola Mengejar Mouse dengan Percepatan Konstan

- **Percepatan Tetap**
  Percepatan bola selalu memiliki **besar tetap (0.5)** dan hanya berubah arah mengikuti posisi mouse. Jadi, walau jarak bola ke mouse jauh atau dekat, percepatan yang diberikan sama besar, hanya arah yang disesuaikan.

- **Proses**
    1. Hitung arah dari bola ke mouse (vektor arah unit).
    2. Kalikan arah dengan percepatan konstan (0.5).
    3. Tambahkan percepatan ke kecepatan.
    4. Batasi kecepatan maksimal (topspeed).
    5. Update posisi bola.
- **Perilaku bola**
  Bola bergerak ke arah mouse dengan percepatan tetap, sehingga bola cenderung bergerak halus dan stabil menuju posisi mouse.

Script 25 - Simulasi Bola dengan Percepatan Variabel

- **Percepatan Variabel (Bergantung Jarak)**
  Percepatan bola **tidak tetap**, tapi **berbanding lurus dengan jarak bola ke mouse**.
  Jadi, semakin jauh bola dari mouse, semakin besar percepatan yang diterima bola.
- **Proses**
    1. Hitung arah dari bola ke mouse (vektor arah unit).
    2. Hitung jarak bola ke mouse.
    3. Kalikan arah unit dengan distance * 0.01 sebagai besar percepatan (magnitude percepatan berubah-ubah sesuai jarak).
    4. Tambahkan percepatan ke kecepatan.
    5. Batasi kecepatan maksimal (topspeed).
    6. Update posisi bola.
- **Perilaku bola**
  Bola akan bergerak sangat cepat saat jauh dari mouse karena percepatan besar, dan melambat saat sudah dekat mouse karena percepatan mengecil. Ini menciptakan gerakan yang lebih "dinamis" dan realistis, seperti bola yang mengejar target dengan tenaga lebih besar saat jauh.

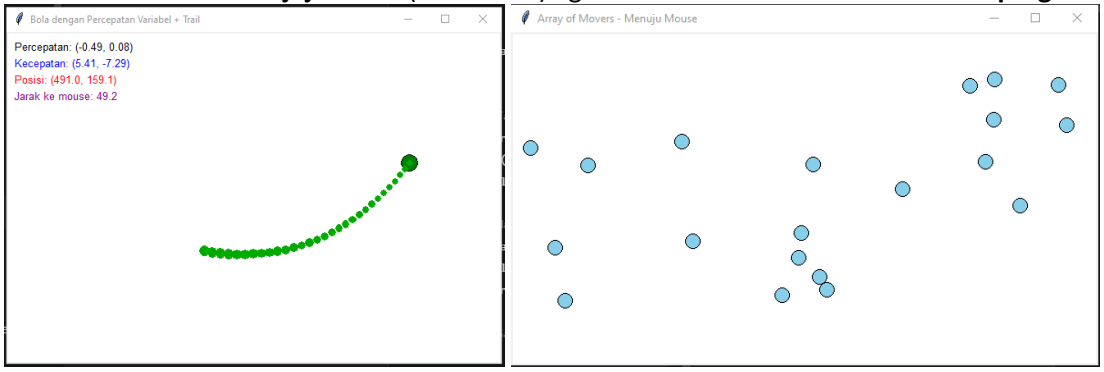
Kesimpulan Perbedaan Utama:

Aspek	Script 24	Script 25
Percepatan	Konstan (0.5), hanya arah berubah	Variabel, tergantung jarak bola ke mouse
Dampak kecepatan	Kecepatan bertambah stabil	Kecepatan bisa besar saat jauh, kecil saat dekat
Gerakan bola	Halus, stabil	Lebih dinamis, cepat saat jauh, lambat saat dekat
Kompleksitas	Lebih sederhana	Lebih realistis dan natural

📌 **Catatan Tambahan:**

- **Normalisasi vektor** bikin arah tetap, tapi panjangnya 1.
- Kita bisa **mengontrol percepatan** dengan mengatur panjang vektor (magnitude).

Tambahkan **efek visual jejak bola** (trail effect) agar anak-anak bisa **melihat lintasan pergerakan bola**.



🎨 **Konsep Jejak Bola:**

Jejak bisa dibuat dengan dua cara:

1. **Membuat transparan background** (tidak bisa langsung di Tkinter)
2. **Menyimpan jejak posisi sebelumnya**, lalu menggambar **lingkaran kecil** di posisi lama

---

✅ **Cara Kita (cocok untuk Tkinter):**

Kita akan:

- Simpan posisi bola sebelumnya dalam **list**
- Gambar titik kecil untuk setiap posisi itu
- Batasi panjang list agar jejak tidak terlalu panjang

🧠 **Perubahan di Kode:**

Tambahkan di dalam kelas Bola:

- List trail = []
- Tambahkan self.trail.append(...) setiap update
- Gambar jejak dengan canvas.create_oval(...)

Diagram Alur Program

Diagram

Rumus dan Perhitungan Tiap Frame

1. Variabel Utama

- location: Posisi bola (Vector)
- velocity: Kecepatan bola (Vector)
- acceleration: Percepatan bola (Vector)
- topspeed: 10 (batas kecepatan maksimum)
- trails: List menyimpan 30 posisi terakhir
- trail_ids: List ID canvas untuk jejak

2. Mekanisme Utama

A. Hitung Vektor Arah ke Mouse

arah = Vector(mouse_x - bola_x, mouse_y - bola_y)

Contoh:

- Mouse di (400, 300), bola di (300, 200):

arah = (100, 100)

B. Hitung Jarak & Percepatan Dinamis

distance = arah.mag() *# √(x² + y²)*

arah.normalize().mult(distance * 0.01) *# percepatan = 1% jarak*

Perhitungan:

1. Normalisasi:

arah = (100/141.42, 100/141.42) ≈ (0.707, 0.707)

2. Skala:

acceleration = (0.707, 0.707) * 141.42 * 0.01 ≈ (1.0, 1.0)

C. Update Kecepatan & Posisi

self.velocity.add(self.acceleration)

self.velocity.limit(self.topspeed)

self.location.add(self.velocity)

Contoh:

- Jika velocity awal (2.0, 1.5):

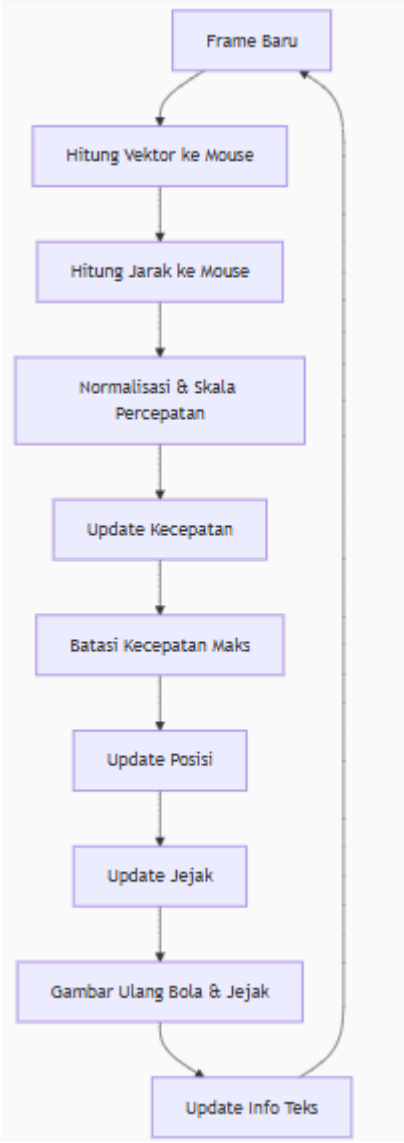
velocity baru = (2.0+1.0, 1.5+1.0) = (3.0, 2.5)

posisi baru = (300+3.0, 200+2.5) = (303.0, 202.5)

D. Sistem Jejak (Trail)

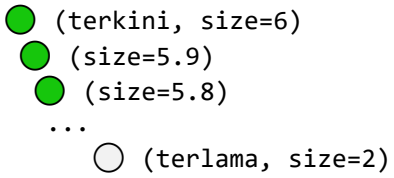
self.trails.append((x, y)) *# Simpan posisi saat ini*

if len(self.trails) > 30:



```
    self.trails.pop(0)  # Hapus jejak terlama

# Gambar jejak dengan ukuran mengecil
for i, (tx, ty) in enumerate(self.trails):
    size = max(2, 6 - (30 - len(self.trails) + i) * 0.1)
```

Ilustrasi Jejak:



Contoh Perhitungan Frame-by-Frame

Frame 1:

- Posisi bola: (300, 200)
- Mouse: (400, 300)

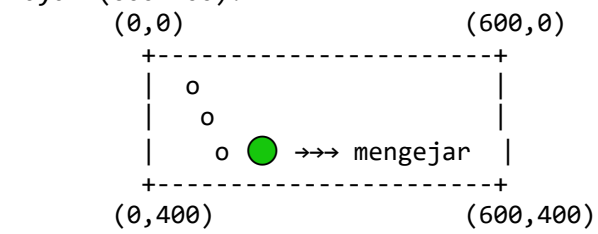
1. arah = (100, 100)
2. distance ≈ 141.42
3. acceleration ≈ (1.0, 1.0)
4. velocity = (0+1.0, 0+1.0) = (1.0, 1.0)
5. Posisi baru: (301.0, 201.0)
6. Jejak: [(301.0, 201.0)]

Frame 2:

- Posisi bola: (301.0, 201.0)
- Mouse: (410, 310)

1. arah = (109, 109)
2. distance ≈ 154.15
3. acceleration ≈ (1.09, 1.09)
4. velocity = (1.0+1.09, 1.0+1.09) = (2.09, 2.09)
5. Posisi baru: (303.09, 203.09)
6. Jejak: [(301.0, 201.0), (303.09, 203.09)]

Visualisasi Gerakan



Karakteristik:

1. **Jejak Dinamis**: 30 titik dengan ukuran mengecil
2. **Percepatan Adaptif**: Semakin jauh dari mouse, semakin cepat
3. **Efek Visual**: Jejak membentuk gradien transparansi implisit

---

Persamaan Matematika Kunci

1. **Vektor Arah**:

```
arah = mouse_pos - bola_pos
```

2. **Percepatan Dinamis**:

```
acceleration = normalize(arah) * distance * 0.01
```

3. **Limit Kecepatan**:

```
if |velocity| > topspeed:
    velocity = normalize(velocity) * topspeed
```

4. **Ukuran Jejak**:

size = max(2, 6 - (max_trails - current_index)*0.1)

Output Teks Informasi

Setiap frame menampilkan:

1. Percepatan: (ax, ay)
2. Kecepatan: (vx, vy)
3. Posisi: (x, y)
4. Jarak ke mouse: distance

Contoh Output:

Percepatan: <1.09, 1.09>
Kecepatan: <2.09, 2.09>
Posisi: (303.1, 203.1)
Jarak ke mouse: 154.2

Fitur Khusus

1. **Efek Jejak**:
    - Menyimpan 30 posisi terakhir
    - Jejak terbaru lebih besar (size=6)
    - Jejak lama mengecil hingga size=2
    - Warna hijau transparan (#00aa00)
2. **Optimasi Performa**:
    - Menghapus dan menggambar ulang jejak tiap frame
    - Membatasi jumlah jejak untuk menghindari memory leak

Dengan mekanisme ini, bola akan:

- Bergerak dengan percepatan proporsional terhadap jarak mouse
- Meninggalkan jejak visual yang indah
- Memberikan informasi fisika real-time

Kode Lengkap + Efek Jejak Bola

```
# SCRIPT 28 - Simulasi Bola dengan Percepatan Variabel dan Jejak
import tkinter as tk
from vector import Vector

# Kelas Bola
class Bola:
    def __init__(self, canvas, x, y):
        self.canvas = canvas
        self.location = Vector(x, y)
        self.velocity = Vector(0, 0)
        self.acceleration = Vector(0, 0)
        self.radius = 10
        self.topspeed = 10
        self.id = canvas.create_oval(x - self.radius, y - self.radius,
                                     x + self.radius, y + self.radius,
                                     fill='green')

        self.trails = []
        self.trail_ids = []

        # Info teks
        self.text_acc = canvas.create_text(10, 10, anchor='nw', text="", font=('Arial',
10), fill='black')
        self.text_vel = canvas.create_text(10, 30, anchor='nw', text="", font=('Arial',
10), fill='blue')
        self.text_pos = canvas.create_text(10, 50, anchor='nw', text="", font=('Arial',
10), fill='red')
```

```
        self.text_dist = canvas.create_text(10, 70, anchor='nw', text="",
font=('Arial', 10), fill='purple')

    def update(self, mouse_x, mouse_y):
        target = Vector(mouse_x, mouse_y)
        arah = Vector(target.x - self.location.x, target.y - self.location.y)

        distance = arah.mag()
        if distance == 0:
            return

        arah.normalize()
        arah.mult(distance * 0.01)  # percepatan variabel
        self.acceleration = arah

        self.velocity.add(self.acceleration)
        self.velocity.limit(self.topspeed)

        self.location.add(self.velocity)

        # Update posisi bola dengan canvas.coords
        x, y = self.location.x, self.location.y
        self.canvas.coords(self.id, x - self.radius, y - self.radius,
                                    x + self.radius, y + self.radius)

        # Tambahkan ke trail
        self.trails.append((x, y))
        if len(self.trails) > 30:
            self.trails.pop(0)

        for trail_id in self.trail_ids:
            self.canvas.delete(trail_id)
        self.trail_ids.clear()

        for i, (tx, ty) in enumerate(self.trails):
            size = max(2, 6 - (30 - len(self.trails) + i) * 0.1)
            color = f'#00aa00'
            trail = self.canvas.create_oval(tx - size, ty - size, tx + size, ty + size,
fill=color, outline="")
            self.trail_ids.append(trail)

        # Tampilkan info
        self.canvas.itemconfig(self.text_acc, text=f"Percepatan: {self.acceleration}")
        self.canvas.itemconfig(self.text_vel, text=f"Kecepatan: {self.velocity}")
        self.canvas.itemconfig(self.text_pos, text=f"Posisi: ({x:.1f}, {y:.1f})")
        self.canvas.itemconfig(self.text_dist, text=f"Jarak ke mouse: {distance:.1f}")

# Setup tkinter
root = tk.Tk()
root.title("Bola dengan Vector + Trail")

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

bola = Bola(canvas, 300, 200)

def animasi():
    x = canvas.winfo_pointerx() - canvas.winfo_rootx()
    y = canvas.winfo_pointery() - canvas.winfo_rooty()
    bola.update(x, y)
    root.after(16, animasi)
```

```
animasi()
root.mainloop()
```

👁️ **Efek Visual yang Terlihat**

- Bola hijau mengejar mouse.
- Meninggalkan **maksimal 30 jejak hijau kecil** di belakangnya.
- Jejak ini memperlihatkan **lintasan gerak bola**.
- Bisa mengamati: “Oh, bola tadi belok ke sana!” 🎯

1.11 - Array of Movers

Kita lanjut ke **materi 1.11 - Array of Movers** dalam Python dan Tkinter, Di sini, kita akan membuat **banyak bola (movers)** yang semuanya **bergerak menuju mouse**, bukan cuma satu bola.

🧠 **Konsep Inti**

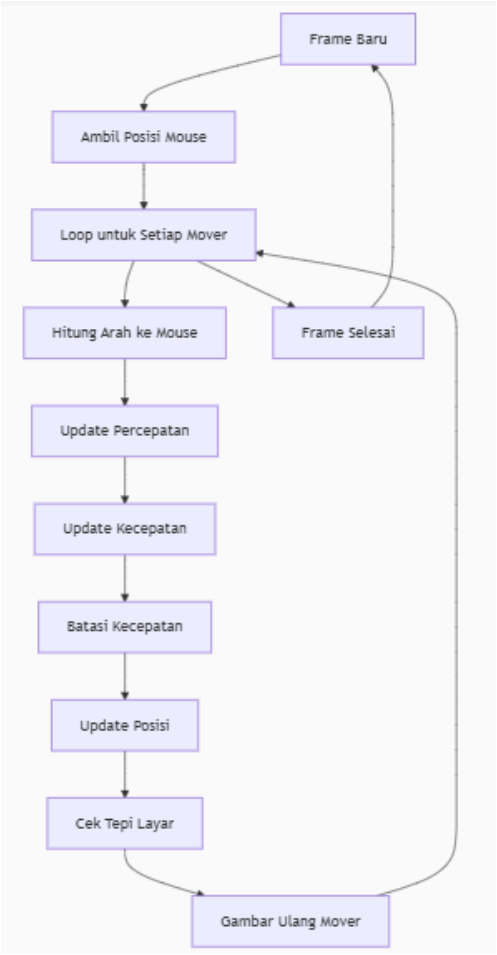
Di Processing:
Mover[] movers = new Mover[20]; // 20 bola
Artinya: kita ingin **banyak objek** Mover yang disimpan dalam **array (atau list)**, dan semuanya bergerak sesuai **aturan percepatan ke arah mouse**.

🧩 **Tujuan Belajar**

1. Anak paham apa itu **list dari objek**.
2. Anak bisa ulangi kode **1 bola → banyak bola**.
3. Anak lihat semua bola bergerak ke arah mouse.
4. Anak mengerti edge wrapping (batas layar).

🔁 **Rencana Langkah**

1. Buat kelas Mover (mirip Bola sebelumnya).
2. Di main, buat list berisi 20 Mover.
3. Dalam animasi:
    - Update semua bola
    - Gambar ulang semua bola
4. Implementasi check_edges() agar bola **muncul kembali dari sisi berlawanan** jika keluar layar.



Rumus dan Perhitungan Tiap Frame

1. Variabel Utama (per Mover)

- location: Posisi mover (Vector)
- velocity: Kecepatan mover (Vector)
- acceleration: Percepatan mover (Vector)
- topspeed: 4 (batas kecepatan maksimum)
- radius: 8 (ukuran mover)

2. Mekanisme Utama per Mover

A. Hitung Vektor Arah ke Mouse

direction = Vector(mouse_x - mover_x, mouse_y - mover_y)
direction.normalize().mult(0.5) *# Percepatan konstan 0.5*

Contoh:

- Mouse di (400, 300), mover di (300, 200):

direction = (400-300, 300-200) = (100, 100)
normalize = (100/141.42, 100/141.42) ≈ (0.707, 0.707)
acceleration = (0.707*0.5, 0.707*0.5) ≈ (0.35, 0.35)

B. Update Kecepatan

self.velocity.add(self.acceleration)

self.velocity.limit(self.topspeed) *# Batasi ke 4*

Perhitungan:

1. Tambahkan percepatan:
    - Jika velocity awal (1.0, 1.5):

velocity = (1.0+0.35, 1.5+0.35) = (1.35, 1.85)

2. Cek batas kecepatan:
    - Magnitude = $\sqrt{(1.35^2 + 1.85^2)} \approx 2.29$ (di bawah 4, tidak diubah)

C. Update Posisi

self.location.add(self.velocity)

Contoh:

- Jika location (300, 200) dan velocity (1.35, 1.85):

location = (300+1.35, 200+1.85) = (301.35, 201.85)

D. Cek Tepi Layar (Wrap-Around)

if x > width: x = 0
if x < 0: x = width
if y > height: y = 0
if y < 0: y = height

Contoh Perhitungan 3 Mover Sekaligus

Frame 1:

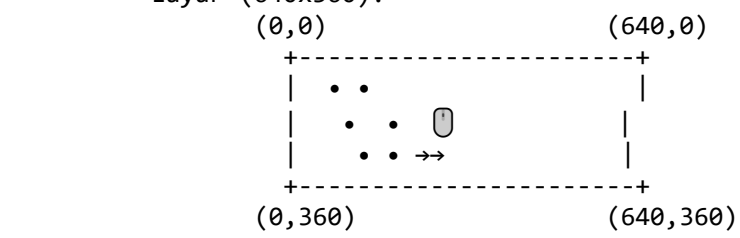
Mover	Posisi Awal	Arah ke Mouse	Percepatan	Kecepatan Baru	Posisi Baru
1	(100,100)	(300,200)→(0.6,0.8)	(0.3,0.4)	(0.3,0.4)	(100.3,100.4)
2	(500,300)	(-100,0)→(-1,0)	(-0.5,0)	(1.0,-0.5)+(-0.5,0)=(0.5,-0.5)	(500.5,299.5)
3	(320,180)	(80,120)→(0.55,0.83)	(0.28,0.42)	(0.28,0.42)	(320.28,180.42)

Frame 2:

(Mouse bergerak ke (420, 320))

Mover	Posisi	Arah Baru	Percepatan	Kecepatan Baru	Posisi Baru
1	(100.3,100.4)	(319.7,219.6)→(0.82,0.57)	(0.41,0.29)	(0.3+0.41,0.4+0.29)=(0.71,0.69)	(101.01,101.09)
2	(500.5,299.5)	(-80.5,20.5)→(-0.97,0.25)	(-0.49,0.13)	(0.5-0.49,-0.5+0.13)=(0.01,-0.37)	(500.51,299.13)
3	(320.28,180.42)	(99.72,139.58)→(0.58,0.81)	(0.29,0.41)	(0.28+0.29,0.42+0.41)=(0.57,0.83)	(320.85,181.25)

Visualisasi Gerakan



Karakteristik:

1. **Gerakan Kolektif**: 20 mover bergerak bersama menuju mouse
2. **Percepatan Seragam**: Nilai percepatan konstan (0.5)
3. **Efek Wrap-Around**: Mover yang keluar layar muncul di sisi berlawanan

---

Persamaan Matematika Kunci

1. **Vektor Arah**:

direction = normalize(mouse_pos - mover_pos)

2. **Percepatan Konstan**:

acceleration = direction * 0.5

3. **Update Kecepatan**:

velocity = velocity + acceleration

if |velocity| > topspeed:

velocity = normalize(velocity) * topspeed

4. **Update Posisi**:

position = position + velocity

Parameter Penting

1. **Kecepatan Maks (**topspeed=4**)**:
   - Mencegah mover bergerak terlalu cepat
   - Menciptakan efek "inertia" alami
2. **Percepatan Konstan (**0.5**)**:
   - Nilai kecil untuk gerakan halus
   - Cukup besar untuk responsif
3. **Jumlah Mover (**20**)**:
   - Cukup untuk efek visual menarik
   - Tidak terlalu banyak untuk performa

Perbedaan dengan Script Sebelumnya

1. **Percepatan Konstan** (bukan berdasarkan jarak)
2. **Multi-Mover** (bukan single object)
3. **Tidak Ada Jejak** (fokus pada gerakan kelompok)

Dengan mekanisme ini, semua mover akan:

- Bergerak menuju mouse dengan kecepatan terbatas
- Memiliki percepatan yang seragam
- Menciptakan efek "kawanan" yang organik

💻 Kode Lengkap Python Tkinter: Array of Movers

```
# SCRIPT 29 - Simulasi Array of Movers yang Bergerak Menuju Mouse
import tkinter as tk
import random
from vector import Vector

# 🎯 Class untuk bola yang bergerak
class Mover:
    def __init__(self, canvas, width, height):
        self.canvas = canvas
        self.width = width
        self.height = height
        self.radius = 8
        self.topspeed = 4

        # Posisi acak, kecepatan awal nol
        self.location = Vector(random.randint(0, width), random.randint(0, height))
        self.velocity = Vector()
        self.acceleration = Vector()

        self.id = canvas.create_oval(0, 0, 0, 0, fill='skyblue', outline='black')

    def update(self, target_x, target_y):
        # 🎯 Hitung arah menuju mouse
        target = Vector(target_x, target_y)
        direction = Vector(target.x - self.location.x, target.y - self.location.y)
```

```
        direction.normalize()
        direction.mult(0.5)  # percepatan tetap

        self.acceleration = direction

        # 💨 Update velocity dan batasi kecepatan
        self.velocity.add(self.acceleration)
        self.velocity.limit(self.topspeed)

        # 📍 Update posisi
        self.location.add(self.velocity)

        # 🌍 Cek tepi layar
        self.check_edges()

        # 🔄 Gambar ulang posisi bola
        x, y = self.location.x, self.location.y
        self.canvas.coords(self.id, x - self.radius, y - self.radius,
                                    x + self.radius, y + self.radius)

    def check_edges(self):
        if self.location.x > self.width:
            self.location.x = 0
        elif self.location.x < 0:
            self.location.x = self.width

        if self.location.y > self.height:
            self.location.y = 0
        elif self.location.y < 0:
            self.location.y = self.height

# 🧠 Setup Tkinter
root = tk.Tk()
root.title("Array of Movers - Menuju Mouse")

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

# 🔵 Buat beberapa Mover
movers = [Mover(canvas, canvas_width, canvas_height) for _ in range(20)]

# 🎥 Loop animasi
def animate():
    mouse_x = canvas.winfo_pointerx() - canvas.winfo_rootx()
    mouse_y = canvas.winfo_pointery() - canvas.winfo_rooty()
    for mover in movers:
        mover.update(mouse_x, mouse_y)
    root.after(16, animate)  # 60 FPS

animate()
root.mainloop()
```

📦 **Apa itu class Mover?**

Seperti cetakan kue: semua mover punya bentuk dan aturan gerak yang sama, tapi posisinya berbeda.

🔍 **Penjelasan Fisik yang Sederhana**

- **Percepatan = arah ke mouse**
- **Kecepatan = makin lama makin cepat, tapi dibatasi**

- **Lokasi = berubah sesuai kecepatan**
- **Jika keluar layar = pindah ke sisi lain** (wrap around)

🎓 **Ilustrasi**

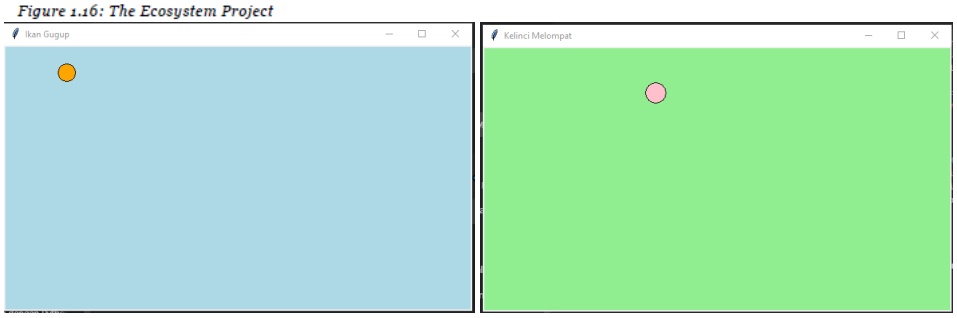
Konsep	**Penjelasan**
class Mover	Seperti resep atau blueprint bola
self.location	Posisi bola sekarang
self.velocity	Seberapa cepat bola bergerak
self.acceleration	Dorongan ke arah mouse
update()	Menghitung ulang gerak bola ke arah mouse
check_edges()	Kalau bola keluar layar, dia "teleport" ke sisi seberangnya

---

✨ **Yang Terjadi:**

- Ada **20 bola** muncul acak di layar.
- Setiap bola bergerak menuju **kursor mouse**.
- Jika bola keluar dari kanan, muncul dari kiri (dan sebaliknya), sama juga atas–bawah.
- Semua bola punya **kecepatan maksimum (topspeed)** agar tidak terlalu cepat.

Figure 1.16: The Ecosystem Project



Bagus! Sekarang kita memasuki **proyek besar pertama** : **The Ecosystem Project** 🐟🐇🐍, yaitu simulasi ekosistem digital di mana ada makhluk-makhluk bergerak secara alami seperti ikan berenang, kelinci melompat, ular melata, dll.

🎯 **Tujuan Proyek:**

- Anak belajar **membuat aturan gerakan makhluk hidup** (bukan cuma bola biasa).
- Menggunakan **acceleration dan velocity** sebagai cara utama untuk mengontrol gerakan.
- Memberi "kepribadian" pada makhluk lewat **perilaku (kode)**, bukan hanya tampilan.

🧠 **Konsep Dasar:**

Setiap makhluk:

- Punya posisi (location)
- Punya kecepatan (velocity)
- Punya percepatan (acceleration)
- Punya perilaku yang beda-beda (misalnya melompat, berputar, dll)

Kita tidak langsung mengatur posisi. Kita hanya mengatur **percepatan (acceleration)**, lalu biarkan physics (perhitungan kecepatan dan lokasi) mengatur sisanya.

✅ **Tahapan Step-by-Step**

🔷 **Step 1: Rancang "Makhluk"**

Misalnya kita mulai dari **seekor ikan yang berenang gugup (nervous fish)**.

Sifat:

- Terus bergerak.
- Kadang-kadang ganti arah secara acak.
- Tidak diam di tempat.
- Saat mendekati tepi, ikan cemas → langsung belok menjauh.
-

Diagram Alur Program

Diagram

Rumus dan Perhitungan Tiap Frame

1. Variabel Utama

- position: Posisi ikan (Vector)
- velocity: Kecepatan ikan (Vector)
- acceleration: Percepatan ikan (Vector)
- max_speed: 3 (batas kecepatan)
- size: 12 (ukuran ikan)

2. Mekanisme Utama

A. Gerakan Acak (5% Chance)

```
if random.random() < 0.05:  # 5% probability
   angle = random.uniform(0, 2π)
   force = Vector(cos(angle), sin(angle)).mult(0.5)
   self.acceleration = force
```

Contoh Perhitungan:

- Jika random angle = 1.2 radian (≈68.75°)
- force = (cos(1.2), sin(1.2)) ≈ (0.36, 0.93)
- Setelah dikali 0.5 → acceleration ≈ (0.18, 0.47)

B. Cek Tepi Layar

```
safe_zone = 20
edge_force = 1.5

# Contoh jika di tepi kiri (x < 20)
if self.position.x < safe_zone:
   self.acceleration = Vector(edge_force, self.acceleration.y)
```

Logika:

- Jika ikan mendekati tepi (dalam 20px), dapatkan percepatan menjauh
- Tepi kiri: percepatan +x (1.5)
- Tepi kanan: percepatan -x (-1.5)
- Atas/bawah: sama untuk sumbu y

C. Update Kecepatan dan Posisi

1. **Update Kecepatan**:

self.velocity.add(self.acceleration)
Contoh:

- Jika velocity = (1.0, 0.5) dan acceleration = (0.18, 0.47)
- Hasil: (1.18, 0.97)

2. **Batasi Kecepatan**:

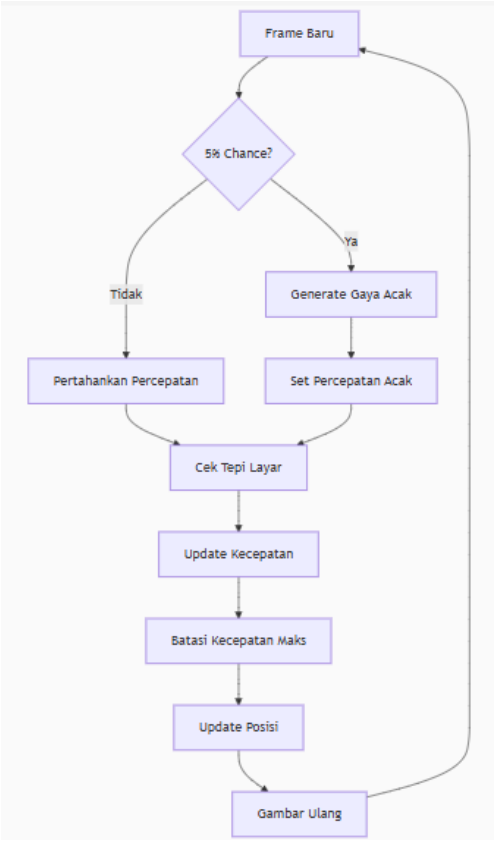
self.velocity.limit(self.max_speed) *# max_speed = 3*

- Jika magnitude > 3, pertahankan arah tapi set magnitude = 3
- Contoh: (4.0, 3.0) → magnitude=5 → dinormalisasi jadi (2.4, 1.8)

3. **Update Posisi**:

self.position.add(self.velocity)
Contoh:

- Jika position = (100, 200) dan velocity = (1.18, 0.97)
- Hasil: (101.18, 200.97)



Contoh Perhitungan Frame-by-Frame
Frame 1: Gerakan Normal

- Posisi awal: (320, 180)
- Kecepatan awal: (1.0, 0.5)
- Tidak trigger gerakan acak (random > 0.05)
- Tidak di tepi layar → acceleration = (0, 0)
- **Hasil akhir:**
  - Kecepatan tetap: (1.0, 0.5)
  - Posisi baru: (321.0, 180.5)

Frame 2: Trigger Gerakan Acak

- Posisi: (321.0, 180.5)
- Trigger 5% → acceleration = (0.18, 0.47)
- Update kecepatan: (1.0+0.18, 0.5+0.47) = (1.18, 0.97)
- Posisi baru: (322.18, 181.47)

Frame 3: Menghindari Tepi

- Posisi: (15, 100) → x < 20 (tepi kiri)
- Dapat percepatan: (1.5, 0)
- Kecepatan baru: (1.18+1.5, 0.97+0) = (2.68, 0.97)
- Jika magnitude > 3: dinormalisasi
- Posisi baru: (17.68, 100.97)

Visualisasi Gerakan

```
Layar (640x360):
       (0,0)                    (640,0)
         +------------------------+
         |   ●                    |
         |      ●→                |
         |         ●  (hindari tepi)|
         +------------------------+
       (0,360)                  (640,360)
```

Karakteristik gerakan:

1. **Gugup**: Gerakan tak terduga (5% chance perubahan arah)
2. **Menghindari Tepi**: Percepatan otomatis saat dekat tepi
3. **Gerakan Halus**: Kecepatan dibatasi untuk gerakan realistis

---

Persamaan Fisika yang Digunakan

1. **GLBB (Gerak Lurus Berubah Beraturan)**:

$v = v_o + a \cdot \Delta t$ # $\Delta t = 1$ (per frame)

$p = p_o + v \cdot \Delta t$

2. **Limit Kecepatan**:

if |v| > max_speed:
  v = (v/|v|) * max_speed

Dengan parameter:

- Δt implisit = 1 (setiap frame)
- Percepatan (a) berasal dari:
  - 5% chance: gaya acak
  - Saat dekat tepi: gaya tolak

```
# SCRIPT 30 - Simulasi Ikan Gugup yang Bergerak Acak
import tkinter as tk
import random
from vector import Vector

# ----------------------------
# Class Fish: Ikan yang Gugup
# ----------------------------
class Fish:
    def __init__(self, canvas, width, height):
        self.canvas = canvas
        self.width = width
        self.height = height
        self.size = 12

        self.position = Vector(random.randint(0, width), random.randint(0, height))
        self.velocity = Vector(random.uniform(-2, 2), random.uniform(-2, 2))
        self.acceleration = Vector(0, 0)
        self.max_speed = 3

        self.body = canvas.create_oval(0, 0, 0, 0, fill='orange', outline='black')

    def update(self):
        # 🌀 Kadang-kadang bergerak acak (5%)
        if random.random() < 0.05:
            angle = random.uniform(0, 2 * math.pi)
            force = Vector(math.cos(angle), math.sin(angle))
            force.mult(0.5)
            self.acceleration = force

        # 🧠 Cek tepi layar dan beri gaya balik
        self.check_edges()

        # 🧭 Update kecepatan dan batasi
        self.velocity.add(self.acceleration)
        self.velocity.limit(self.max_speed)

        # 🚀 Update posisi
        self.position.add(self.velocity)

        # 🎨 Gambar ulang di posisi baru
        x, y = self.position.x, self.position.y
        self.canvas.coords(self.body,
                           x - self.size, y - self.size,
                           x + self.size, y + self.size)

    def check_edges(self):
        safe_zone = 20
```

```
        edge_force = 1.5

        if self.position.x < safe_zone:
            self.acceleration = Vector(edge_force, self.acceleration.y)
        elif self.position.x > self.width - safe_zone:
            self.acceleration = Vector(-edge_force, self.acceleration.y)

        if self.position.y < safe_zone:
            self.acceleration = Vector(self.acceleration.x, edge_force)
        elif self.position.y > self.height - safe_zone:
            self.acceleration = Vector(self.acceleration.x, -edge_force)

# ----------------------------
# Main: Buat Layar dan Animasi
# ----------------------------
root = tk.Tk()
root.title("Ikan Gugup - Versi Vector")

lebar = 640
tinggi = 360
canvas = tk.Canvas(root, width=lebar, height=tinggi, bg="lightblue")
canvas.pack()

ikan = Fish(canvas, lebar, tinggi)

def animate():
    ikan.update()
    root.after(20, animate)  # 50 FPS
animate()
root.mainloop()
```

🔷 Cara Kerja Program:

1. **Inisialisasi**:
    - Ikan muncul di posisi acak dengan kecepatan acak
    - Bentuknya lingkaran orange
2. **Setiap Frame**:
    - Ada 5% kemungkinan ikan berubah arah tiba-tiba
    - Kecepatan diupdate berdasarkan percepatan
    - Posisi diupdate berdasarkan kecepatan
    - Jika dekat tepi, ikan akan menghindar
3. **Visualisasi**:
    - Lingkaran orange bergerak acak di layar biru
    - Saat mendekati tepi, ikan akan berbalik arah

🐇 **Kelinci yang lompat-lompat** di ekosistem digital kita.

Tujuannya adalah membuat gerakan **melompat secara acak** seperti kelinci asli — tidak halus seperti ikan yang berenang, tapi **bergerak cepat lalu berhenti sebentar**, lalu lompat lagi.

🎯 **Tujuan:**

- Mengontrol gerakan kelinci hanya dengan **acceleration** dan **velocity**
- Membuat **"perilaku khas kelinci"**: lompat → berhenti → lompat lagi
- Visualisasinya tetap pakai **Tkinter** sederhana

🧠 **Konsep "Kelinci Lompat"**

Sifat kelinci:

- Tidak jalan terus-terusan
- Gerakannya berupa **lonjakan cepat** lalu **diam sebentar**

- Sering ganti arah
- Lompatannya pendek dan cepat

Class Bunny dengan Gerakan Lompat

Diagram Alur Program
Diagram
Rumus dan Perhitungan Tiap Frame
1. Variabel Utama

- posisi: Vector(x, y) - Posisi kelinci
- kecepatan: Vector(vx, vy) - Kecepatan kelinci
- percepatan: Vector(ax, ay) - Percepatan kelinci
- kecepatan_max: 7 (batas kecepatan)
- waktu_tunggu: Counter untuk jeda antar lompatan

2. Alur Per Frame (dengan Contoh Perhitungan)
Contoh Kasus Awal:

- Posisi awal: (320, 180)
- Kecepatan awal: (0, 0)
- Waktu tunggu: 0 (siap lompat)

Frame 1: Lompat Pertama

1. **Generate Lompatan Acak** (karena waktu_tunggu ≤ 0)

sudut = random.uniform(0, 2π) *# Misal: 1.2 radian (≈68.75°)*
kekuatan = random.uniform(3, 6) *# Misal: 4.5*
Percepatan:
ax = cos(1.2) * 4.5 ≈ 1.68
ay = sin(1.2) * 4.5 ≈ 4.18
percepatan = (1.68, 4.18)

2. **Update Kecepatan**:

kecepatan = (0 + 1.68, 0 + 4.18) = (1.68, 4.18)

3. **Perlambatan** (dikalikan 0.85):

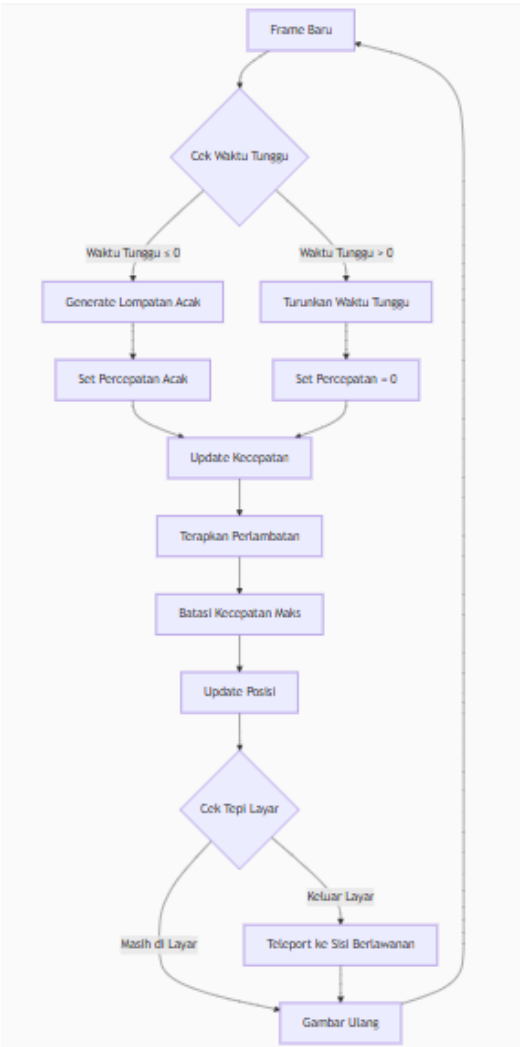
kecepatan = (1.68*0.85, 4.18*0.85) ≈ (1.43, 3.55)

4. **Batasi Kecepatan** (max = 7):
    - Magnitude: √(1.43² + 3.55²) ≈ 3.83 (di bawah 7, tidak diubah)
5. **Update Posisi**:

posisi = (320 + 1.43, 180 + 3.55) ≈ (321.43, 183.55)

6. **Output Console**:

Posisi baru: (321.43, 183.55)
Kecepatan: (1.43, 3.55)



Frame 2: Dalam Waktu Tunggu

1. **Waktu Tunggu** (misal di-set 25 frame):

waktu_tunggu -= 1 *# Sekarang 24*
percepatan = (0, 0)

2. **Update Kecepatan** (hanya perlambatan):

kecepatan = (1.43*0.85, 3.55*0.85) ≈ (1.22, 3.02)

3. **Update Posisi**:

posisi = (321.43 + 1.22, 183.55 + 3.02) ≈ (322.65, 186.57)

Frame N: Keluar Layar
Kasus: Posisi y > tinggi_layar (360)

1. **Teleport ke Atas**:

if posisi.y > tinggi:
  posisi.y = 0
Contoh:
Posisi sebelum: (330, 362) → setelah: (330, 0)

Mekanisme Kunci

1. **Sistem Waktu Tunggu**:
   - Setelah lompat, kelinci "beristirahat" selama 20-40 frame
   - Selama waktu tunggu: percepatan = (0,0)
2. **Fisika Gerakan**:
   - **Percepatan**: Di-set acak saat lompat (arah + kekuatan)
   - **Perlambatan**: Kecepatan dikali 0.85 tiap frame (efek gesekan)
   - **Kecepatan Maks**: Dibatasi 7 untuk gerakan realistis
3. **Perhitungan Vektor**:

 # Contoh normalisasi vektor
def normalize(v):
  mag = sqrt(v.x² + v.y²)
  return Vector(v.x/mag, v.y/mag)

Contoh Output Runtime

--- Perhitungan Frame Baru ---
Posisi awal: (320.0, 180.0)
Kecepatan awal: (0.0, 0.0)
Waktunya lompat!
Percepatan baru: (1.68, 4.18)
Set waktu tunggu: 28 frame
Kecepatan setelah percepatan: (1.68, 4.18)
Kecepatan setelah perlambatan: (1.43, 3.55)
Kecepatan dibatasi: (1.43, 3.55)
Posisi baru: (321.43, 183.55)

Visualisasi Gerakan

```
Layar (640x360):
       (0,0)                    (640,0)
         +------------------------+
         |                        |
         |   🐇 (bergerak ↗)      |
         |                        |
         +------------------------+
       (0,360)                  (640,360)
```

Setiap lompat menciptakan lintasan parabola karena:

1. Awal frame: percepatan besar
2. Frame berikutnya: perlambatan progresif
3. Saat waktu_tunggu habis: lompat baru acak

```
# SCRIPT 31 - Simulasi Kelinci Melompat
import tkinter as tk
import random
import math
from vector import Vector

# ----------------------------
# 🐇 Kelas Kelinci Pakai Vector
```

```
# ----------------------------
import tkinter as tk
import random

class Kelinci:
    def __init__(self, kanvas, lebar, tinggi):
        self.kanvas = kanvas
        self.lebar = lebar
        self.tinggi = tinggi

        # Vektor posisi, kecepatan, percepatan
        self.posisi = Vector(random.randint(0, lebar), random.randint(0, tinggi))
        self.kecepatan = Vector(0, 0)
        self.percepatan = Vector(0, 0)

        self.ukuran = 14
        self.kecepatan_max = 7
        self.waktu_tunggu = 0

        self.bentuk = kanvas.create_oval(0, 0, 0, 0, fill='pink')

    def update(self):
        print("\n--- Perhitungan Frame Baru ---")
        print("Posisi awal:", self.posisi)
        print("Kecepatan awal:", self.kecepatan)

        if self.waktu_tunggu <= 0:
            print("Waktunya lompat!")
            sudut = random.uniform(0, 2 * math.pi)
            kekuatan = random.uniform(3, 6)
            self.percepatan = Vector(math.cos(sudut) * kekuatan, math.sin(sudut) *
kekuatan)
            print("Percepatan baru:", self.percepatan)

            self.waktu_tunggu = random.randint(20, 40)
            print(f"Set waktu tunggu: {self.waktu_tunggu} frame")
        else:
            self.waktu_tunggu -= 1
            self.percepatan.set(0, 0)
            print(f"Menunggu... sisa {self.waktu_tunggu} frame")

        # Tambahkan percepatan ke kecepatan
        self.kecepatan.add(self.percepatan)
        print("Kecepatan setelah percepatan:", self.kecepatan)

        # Perlambatan
        self.kecepatan.mult(0.85)
        print("Kecepatan setelah perlambatan:", self.kecepatan)

        # Batasi kecepatan maksimum
        self.kecepatan.limit(self.kecepatan_max)
        print("Kecepatan dibatasi:", self.kecepatan)

        # Update posisi
        self.posisi.add(self.kecepatan)
        print("Posisi baru:", self.posisi)

        # Cek tepi layar
        self.cek_tepi()

        # Gambar ulang
```

```
        x, y = self.posisi.x, self.posisi.y
        self.kanvas.coords(self.bentuk,
                           x - self.ukuran, y - self.ukuran,
                           x + self.ukuran, y + self.ukuran)

    def cek_tepi(self):
        if self.posisi.x < 0:
            self.posisi.x = self.lebar
            print("Kelinci keluar kiri, muncul di kanan")
        elif self.posisi.x > self.lebar:
            self.posisi.x = 0
            print("Kelinci keluar kanan, muncul di kiri")

        if self.posisi.y < 0:
            self.posisi.y = self.tinggi
            print("Kelinci keluar atas, muncul di bawah")
        elif self.posisi.y > self.tinggi:
            self.posisi.y = 0
            print("Kelinci keluar bawah, muncul di atas")

# ----------------------------
# 💻 Program Utama
# ----------------------------
root = tk.Tk()
root.title("Kelinci Melompat")

lebar_layar = 640
tinggi_layar = 360

kanvas = tk.Canvas(root, width=lebar_layar, height=tinggi_layar, bg="lightgreen")
kanvas.pack()

kelinci = Kelinci(kanvas, lebar_layar, tinggi_layar)

def animasi():
    kelinci.update()
    root.after(30, animasi)

print("=== Program Dimulai ===")
animasi()
root.mainloop()
```

📝 Cara Kerja Program:

1. **Inisialisasi**:
    - Kelinci muncul di posisi acak
    - Awalnya diam (kecepatan 0)
2. **Setiap Frame**:
    - Jika waktu tunggu habis, kelinci akan melompat:
        - Tentukan arah acak (0-360 derajat)
        - Tentukan kekuatan lompatan acak (3-6)
    - Update kecepatan berdasarkan percepatan
    - Perlambatan sedikit demi sedikit (efek gesekan)
    - Pastikan tidak terlalu cepat
    - Update posisi berdasarkan kecepatan
3. **Visualisasi**:
    - Lingkaran pink bergerak dengan pola lompatan acak
    - Saat mencapai tepi, muncul di sisi seberang

4. **Output Console**:
    - Akan menampilkan perhitungan detail setiap frame
    - Bisa dilihat perubahan posisi, kecepatan, dan percepatan

Lanjut ke:

-  Ular yang melata bergelombang
-  Burung yang terbang melengkung
-  AI predator yang mengejar kelinci