Memberi Otak pada Program: Logika & Keputusan!

Selamat datang di pertemuan keempat! Sejauh ini, program kita berjalan seperti rel kereta api: lurus dari awal sampai akhir, menjalankan setiap perintah secara berurutan. Tapi, program yang canggih tidak seperti itu. Program yang hebat bisa beradaptasi, bereaksi, dan membuat keputusan berdasarkan situasi yang ada. Konsep ini disebut **Control Flow**.

Bayangkan kehidupan kita sehari-hari:

- **Jika** hari ini hujan, **maka** saya akan membawa payung.
- **Jika** nilai ujian di atas 75, **maka** saya lulus, **jika tidak maka** saya harus mengulang.
- **Jika** lapar, **maka** cari makan. **Jika tidak** dan haus, **maka** cari minum. **Jika tidak keduanya**, **maka** lanjut bekerja.

Pola "jika... maka..." ini adalah inti dari **logika kondisional** atau **percabangan (branching)**. Hari ini, kita akan memberikan kemampuan ini pada program Python kita, sehingga alur eksekusinya bisa bercabang-cabang.

Fondasi dari Semua Logika: Tipe Data Boolean

Sebelum program bisa membuat keputusan, ia harus bisa mengevaluasi apakah suatu kondisi itu "benar" atau "salah". Di Python, konsep "benar" dan "salah" ini direpresentasikan oleh tipe data **Boolean (`bool`)**.

Tipe data ini sangat sederhana, ia hanya punya dua kemungkinan nilai:

1. `True` (Benar)
2. `False` (Salah)

Perhatikan, huruf `T` pada `True` dan `F` pada `False` harus kapital. Mereka adalah *keywords*, bukan string.

```
# Contoh variabel boolean
apakah_sedang_hujan = True
apakah_lampu_menyala = False

# Mengecek tipe datanya
print(type(apakah_sedang_hujan)) # Output: <class 'bool'>
print(type(False))               # Output: <class 'bool'>
```

Nilai `True` dan `False` inilah yang akan menjadi "bahan bakar" untuk semua struktur keputusan kita.

Alat Pembanding: Operator Relasional

Bagaimana cara kita mendapatkan nilai `True` atau `False` dari data yang kita punya? Jawabannya adalah dengan membandingkan dua nilai (operan) menggunakan **operator relasional (perbandingan)**. Setiap ekspresi yang menggunakan operator ini akan menghasilkan nilai boolean.

Operator	Deskripsi	Contoh Angka (x=10, y=5)	Contoh String (a="a", b="b")	Hasil
==	Sama dengan	x == 10	a == "a"	True
!=	Tidak sama dengan	x != y	a != "b"	True
>	Lebih besar dari	x > y	b > a	True
<	Lebih kecil dari	y < x	a < b	True
>=	Lebih besar dari atau sama dengan	x >= 10	a >= "a"	True
<=	Lebih kecil dari atau sama dengan	y <= x	b <= "c"	True

Peringatan Penting:

- **= vs ==**: Jangan pernah tertukar! = adalah operator *assignment* (memberi nilai), sedangkan == adalah operator *perbandingan* (mengecek kesamaan). Menggunakan = di dalam kondisi adalah kesalahan umum.
- **Tidak ada => atau =<**: Urutannya harus selalu > atau < terlebih dahulu, baru diikuti =.

```
nilai_saya = 80
nilai_kelulusan = 75

# Ekspresi boolean: apakah nilai saya lebih besar atau sama dengan nilai
kelulusan?
apakah_lulus = nilai_saya >= nilai_kelulusan
print("Apakah saya lulus?", apakah_lulus) # Output: Apakah saya lulus?
True
```

Struktur Keputusan Pertama: if

Sekarang kita punya bahan bakarnya (True/False), saatnya membangun mesin keputusannya. Struktur paling dasar adalah if. Pernyataan if akan mengecek sebuah kondisi, dan jika kondisi itu True, maka blok kode di dalamnya akan dieksekusi.

Strukturnya adalah:

```
if <kondisi>:
    # Blok kode yang akan dijalankan JIKA kondisi bernilai True
    # (Perhatikan indentasi!)
```

Konsep Terpenting: Indentasi Di Python, blok kode didefinisikan oleh **indentasi** (jarak menjorok ke dalam), bukan kurung kurawal { }. Standarnya adalah **4 spasi**. Semua baris kode yang memiliki level indentasi yang sama di bawah if dianggap sebagai bagian dari blok if tersebut. Pernyataan seperti if yang memiliki header (diakhiri :) dan body (blok berindentasi) disebut **compound statement**.

Mari kita buat contoh: program untuk mengecek apakah seseorang boleh membuat SIM.

```
# Minta input umur dari pengguna
umur_str = input("Berapa umur Anda? ")
umur = int(umur_str)

# Syarat umur minimal untuk membuat SIM
batas_umur_sim = 17

print("Mengecek kelayakan...")

# Ini adalah struktur kondisionalnya
if umur >= batas_umur_sim:
    print("Selamat! Anda sudah cukup umur untuk membuat SIM.")
    print("Silakan datang ke kantor polisi terdekat.")

print("Pengecekan selesai.") # Baris ini tidak di-indent, jadi akan selalu dijalankan
```

Jika kondisi `umur >= batas_umur_sim` bernilai `False`, maka blok kode yang di-indent akan dilewati begitu saja.

Terkadang kita butuh *placeholder* untuk blok `if` yang kodenya belum kita tulis. Untuk ini, kita bisa gunakan `pass`.

```
if umur < 0:
    pass # Nanti kita akan tangani kasus umur negatif di sini.
```

Menangani Skenario "Jika Tidak": `else`

Bagaimana jika kondisi `if` tidak terpenuhi? Untuk menangani kasus sebaliknya, kita gunakan `else`. Blok `else` akan dieksekusi jika dan hanya jika kondisi `if` bernilai `False`.

Strukturnya adalah:

```
if <kondisi>:
    # Blok kode jika kondisi True
else:
    # Blok kode jika kondisi False
```

Karena kondisi harus `True` atau `False`, maka pasti salah satu dari dua "cabang" (`branch`) ini akan dieksekusi.

Mari kita sempurnakan program SIM kita.

```
umur_str = input("Berapa umur Anda? ")
umur = int(umur_str)
batas_umur_sim = 17

print("Mengecek kelayakan...")

if umur >= batas_umur_sim:
    print("Selamat! Anda sudah cukup umur untuk membuat SIM.")
else:
    # Blok ini akan dijalankan jika umur < 17
    selisih_umur = batas_umur_sim - umur
    print(f"Mohon maaf, Anda belum cukup umur. Anda perlu menunggu
{selisih_umur} tahun lagi.")

print("Pengecekan selesai.")
```

Lebih dari Dua Kemungkinan: `elif`

Dunia nyata seringkali punya lebih dari dua pilihan. Untuk ini, kita gunakan `elif`, singkatan dari "else if". Ini memungkinkan kita untuk membuat **rantai kondisional (chained conditionals)**.

Strukturnya:

```
if <kondisi_1>:
    # Blok kode jika kondisi_1 True
elif <kondisi_2>:
    # Blok kode jika kondisi_1 False DAN kondisi_2 True
elif <kondisi_3>:
    # Blok kode jika kondisi_1 & 2 False DAN kondisi_3 True
else:
    # Blok kode jika SEMUA kondisi di atas False
```

Python akan mengecek kondisi dari atas ke bawah. Begitu ia menemukan satu kondisi yang `True`, ia akan menjalankan blok kodenya dan **langsung keluar dari seluruh struktur `if-elif-else`**, melewati semua pengecekan sisanya.

Contoh: Program Penilaian (Grading)

```
# Minta input nilai dari pengguna
score_str = input("Masukkan skor ujian Anda (0.0 - 1.0): ")
try:
    score = float(score_str)

    # Mengecek apakah skor dalam rentang yang valid
    if score < 0.0 or score > 1.0:
        print("Error: Skor harus di antara 0.0 dan 1.0")
    else:
        # Struktur if-elif-else untuk menentukan grade
```

```
        if score >= 0.9:
            grade = "A"
        elif score >= 0.8:
            grade = "B"
        elif score >= 0.7:
            grade = "C"
        elif score >= 0.6:
            grade = "D"
        else:
            grade = "F"
        print("Grade Anda adalah:", grade)

except:
    print("Error: Masukan harus berupa angka.")
```

`else` di akhir bersifat opsional, tapi sangat berguna sebagai penangkap kasus "default" atau "jika tidak ada yang cocok sama sekali".

Menggabungkan Kondisi: Operator Logika (`and`, `or`, `not`)

Terkadang, satu keputusan bergantung pada **beberapa** kondisi. Untuk menggabungkan ekspresi boolean, kita gunakan operator logika.

`and` (dan)

Menghasilkan `True` hanya jika **KEDUA** kondisi di kiri dan kanannya bernilai `True`.

```
# Contoh: Pendaftaran kursus lanjutan
sudah_lulus_dasar = True
umur = 20

# Kondisi: harus sudah lulus DAN umurnya di atas 18
if sudah_lulus_dasar and umur > 18:
    print("Anda memenuhi syarat untuk mendaftar kursus lanjutan.")
else:
    print("Maaf, Anda belum memenuhi syarat.")
```

`or` (atau)

Menghasilkan `True` jika **SALAH SATU** (atau kedua) kondisi di kiri atau kanannya bernilai `True`.

```
# Contoh: Diskon tiket akhir pekan
hari = "Sabtu"
punya_kartu_member = False

# Kondisi: hari adalah Sabtu ATAU hari adalah Minggu ATAU punya kartu
member
if hari == "Sabtu" or hari == "Minggu" or punya_kartu_member:
    print("Anda berhak mendapatkan diskon akhir pekan!")
```

`not` (bukan/negasi)

Membalikkan nilai boolean. `not True` menjadi `False`, dan `not False` menjadi `True`.

```
sedang_hujan = False

if not sedang_hujan: # jika TIDAK sedang hujan
    print("Cuaca cerah, ayo pergi keluar!")
```

Evaluasi Sirkuit Pendek (Short-Circuit Evaluation)

Python sangat efisien. Saat mengevaluasi ekspresi logika, ia akan berhenti begitu hasilnya sudah pasti.

- Pada `A and B`: Jika `A` adalah `False`, Python tidak akan pernah mengecek `B`, karena hasilnya sudah pasti `False`.
- Pada `A or B`: Jika `A` adalah `True`, Python tidak akan pernah mengecek `B`, karena hasilnya sudah pasti `True`.

Ini melahirkan teknik canggih bernama **Guardian Pattern**. Kita bisa mencegah error dengan menempatkan "penjaga" di depan.

```
# Contoh tanpa guardian, bisa error
x = 6
y = 0
# if (x/y) > 2: # Ini akan menyebabkan ZeroDivisionError

# Dengan guardian pattern
if y != 0 and (x/y) > 2:
    print("Hasilnya lebih dari 2")
else:
    print("Kondisi tidak terpenuhi atau tidak bisa dibagi.")
```

Di sini, `y != 0` adalah sang penjaga. Jika `y` adalah 0, kondisi pertama menjadi `False`, dan Python tidak akan pernah mencoba `(x/y)`, sehingga error bisa dihindari.

Apa yang Dianggap `False` di Python?

Secara umum, Python menganggap nilai yang "kosong" atau "nol" sebagai `False` saat dievaluasi dalam sebuah kondisi. Nilai-nilai berikut dianggap `False`:

1. `None` (tipe data null)
2. `False` itu sendiri
3. Angka nol dari semua tipe: `0`, `0.0`, `0j`
4. Urutan (sequence) atau koleksi (collection) yang kosong: `""` (string kosong), `[]` (list kosong), `()` (tuple kosong), `{}` (dictionary kosong).

Semua nilai lain akan dianggap `True`.

```
nama = ""
if nama:
    print(f"Halo, {nama}")
else:
    print("Nama belum diisi.") # Blok ini yang akan jalan
```

Struktur Kondisional Lanjutan

Kondisi Bersarang (Nested Conditionals)

Kita bisa meletakkan `if` di dalam `if` yang lain. Ini berguna untuk alur pengecekan yang lebih rinci.

```
umur = 25
punya_sim = True

if umur >= 17:
    print("Umur memenuhi syarat.")
    # 'if' yang bersarang di dalam 'if' pertama
    if punya_sim:
        print("Boleh mengemudi.")
    else:
        print("Tidak boleh mengemudi karena belum punya SIM.")
else:
    print("Umur tidak memenuhi syarat.")
```

Perhatian: Hindari nesting yang terlalu dalam (lebih dari 2-3 level), karena membuat kode sulit dibaca. Ingat: *Flat is better than nested*.

Ternary Operators (Conditional Expressions)

Ini adalah cara singkat untuk menulis `if-else` dalam satu baris. Tujuannya bukan hanya untuk mempersingkat, tapi membuat *assignment* berdasarkan kondisi menjadi lebih jelas.

Strukturnya: `<nilai_jika_benar> if <kondisi> else <nilai_jika_salah>`

```
# Cara biasa
umur = 22
if umur >= 17:
    status = "Dewasa"
else:
    status = "Anak-anak"
print(status)

# Dengan Ternary Operator
status_ternary = "Dewasa" if umur >= 17 else "Anak-anak"
print(status_ternary)
```

Menangani Error dengan `try-except`

Saat kita meminta input angka dari pengguna dan mengonversinya dengan `int()` atau `float()`, apa yang terjadi jika pengguna mengetik "halo"? Program kita akan *crash* dengan `ValueError`.

Untuk menangani error yang bisa diprediksi ini secara elegan, kita gunakan `try` dan `except`.

- **`try`**: Tempatkan kode yang berpotensi error di dalam blok ini.
- **`except`**: Blok ini hanya akan dijalankan jika terjadi error di dalam blok `try`.

```
inp = input('Enter Fahrenheit Temperature: ')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print("Celsius Temperature:", cel)
except:
    print('Error, please enter a numeric input')
```

Dengan ini, program kita tidak akan *crash*, melainkan akan menampilkan pesan error yang ramah dan melanjutkan eksekusi (jika ada kode setelahnya).

Latihan untuk Menguji Pemahaman

Waktunya mempraktikkan kemampuan baru program kita untuk berpikir!

Latihan 1: Program Perhitungan Upah dengan Lembur **Buatlah sebuah program** untuk menghitung total upah karyawan. Program harus meminta input jam kerja dan tarif per jam. Aturannya: jam kerja di atas 40 jam dihitung sebagai waktu lembur dengan tarif 1.5 kali lipat dari tarif normal.

- Contoh Input: Jam = 45, Tarif = 10
- Perhitungan: (40 jam normal _ 10) + (5 jam lembur _ 10 * 1.5) = 400 + 75 = 475
- Contoh Output: `Pay: 475.0`

Latihan 2: Program Upah yang Aman dari Error **Modifikasi program dari Latihan 1**. Gunakan blok `try` dan `except` untuk menangani kasus di mana pengguna memasukkan teks (non-numerik) sebagai input jam atau tarif. Jika terjadi error, program harus menampilkan pesan `Error, please enter numeric input` dan berhenti dengan elegan tanpa *crash*.

Latihan 3: Program Penilaian (Grading) yang Lengkap **Buatlah sebuah program** yang meminta input skor antara 0.0 dan 1.0.

1. Gunakan `try-except` untuk menangani input non-numerik.
2. Jika input numerik, periksa apakah skor berada dalam rentang 0.0 sampai 1.0. Jika tidak, tampilkan pesan error yang sesuai (misal: `Bad score`).
3. Jika skor valid, tampilkan grade berdasarkan tabel berikut:
    - `>= 0.9` -> `A`

- `>= 0.8` -> `B`
- `>= 0.7` -> `C`
- `>= 0.6` -> `D`
- `< 0.6` -> `F`

Latihan 4: Logika Pintu Masuk Wahana **Buatlah sebuah program** yang mensimulasikan aturan masuk ke sebuah wahana. Aturannya: "Pengunjung boleh masuk jika tingginya minimal 150 cm **DAN** (usianya di atas 12 tahun **ATAU** didampingi orang tua)".

1. Buat tiga variabel: `tinggi_cm`, `usia`, `didampingi_ortu` (berisi `True` atau `False`).
2. Isi variabel-variabel tersebut dengan nilai apa pun untuk pengujian.
3. Terapkan logika di atas menggunakan `if` dan operator `and`/`or` untuk mencetak "Boleh Masuk" atau "Tidak Boleh Masuk".
4. Ubah-ubah nilai variabel untuk menguji semua kemungkinan skenario.