



# Laporan Praktikum Algoritma dan Pemrograman

Semester Genap 2023/2024

<b>NIM</b>	<b>71230977</b>
<b>Nama Lengkap</b>	<b>MICHAEL HOSEA</b>
<b>Minggu ke / Materi</b>	<b>13 / TIPE DATA REKURSIF</b>

SAYA MENYATAKAN BAHWA LAPORAN PRAKTIKUM INI SAYA BUAT DENGAN USAHA SENDIRI TANPA MENGGUNAKAN BANTUAN ORANG LAIN. SEMUA MATERI YANG SAYA AMBIL DARI SUMBER LAIN SUDAH SAYA CANTUMKAN SUMBERNYA DAN TELAH SAYA TULIS ULANG DENGAN BAHASA SAYA SENDIRI.

SAYA SANGGUP MENERIMA SANKSI JIKA MELAKUKAN KEGIATAN PLAGIASI, TERMASUK SANKSI TIDAK LULUS MATA KULIAH INI.

PROGRAM STUDI INFORMATIKA  
FAKULTAS TEKNOLOGI INFORMASI  
UNIVERSITAS KRISTEN DUTA WACANA  
YOGYAKARTA  
2024

## BAGIAN 1: MATERI MINGGU INI (40%)

Pada bagian ini, tuliskan kembali semua materi yang telah anda pelajari minggu ini. Sesuaikan penjelasan anda dengan urutan materi yang telah diberikan di saat praktikum. Penjelasan anda harus dilengkapi dengan contoh, gambar/ilustrasi, contoh program (source code) dan outputnya. Idealnya sekitar 5-6 halaman.

### Materi 1: Pengantar Rekursif

Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri sebagai bagian dari definisi atau eksekusinya. Fungsi ini umumnya digunakan untuk menyelesaikan masalah yang dapat dipecah menjadi sub-masalah yang lebih kecil dari jenis yang sama. Setiap panggilan rekursif harus mendekatkan solusi ke kondisi dasar (base case) yang tidak memerlukan panggilan rekursif lebih lanjut.

Komponen fungsi rekursif dapat dibagi menjadi 2, yaitu:

1. Base Case: Kondisi yang menghentikan rekursi. Ketika kondisi ini terpenuhi, fungsi tidak lagi memanggil dirinya sendiri dan mengembalikan nilai langsung.
2. Recursive Case: Bagian dari fungsi yang memecah masalah menjadi sub-masalah yang lebih kecil dan memanggil dirinya sendiri dengan parameter yang dimodifikasi.

Berikut merupakan bentuk dasar dari fungsi rekursif

```
def rekursif_fungsi(parameter):  
    # Kasus dasar: berhenti memanggil dirinya sendiri  
    if kondisi_dasar:  
        return nilai  
    else:  
        # Kasus rekursif: memanggil fungsi dengan parameter yang dimodifikasi  
        return rekursif_fungsi(parameter_baru)
```

### Kelebihan dan kekurangan fungsi rekursif:

Beberapa kelebihan dari fungsi rekursif adalah:

1. Kesederhanaan dan Kejelasan Kode:
  - Rekursi sering kali menghasilkan kode yang lebih sederhana dan lebih mudah dipahami, terutama untuk masalah yang secara alami rekursif, seperti traversal pohon atau perhitungan deret Fibonacci.

- Contoh: Fungsi untuk menghitung faktorial atau Fibonacci dapat ditulis dengan sedikit baris kode yang lebih intuitif.

## 2. Pemecahan Masalah yang Alami:

- Beberapa masalah lebih mudah dipecahkan dengan rekursi daripada dengan iterasi. Misalnya, traversal struktur data seperti pohon biner atau graf sering kali lebih mudah diimplementasikan dengan rekursi.

## 3. Ekspresi Elegan dari Algoritma:

- Banyak algoritma terkenal, seperti algoritma pembagi dan taklukkan (divide and conquer) termasuk pencarian biner, quicksort, dan mergesort, sangat cocok untuk implementasi rekursif.

Berikut kekurangan dari fungsi rekursif:

### 1. Penggunaan Memori yang Tinggi:

- Setiap panggilan rekursif menambah tumpukan panggilan (call stack), yang bisa menyebabkan penggunaan memori yang besar dan potensi stack overflow jika rekursi terlalu dalam.
- Contoh: Memanggil fungsi rekursif dengan jumlah iterasi yang sangat besar dapat menyebabkan stack overflow.

### 2. Efisiensi yang Kurang:

- Fungsi rekursif tanpa optimasi (seperti memoization) dapat melakukan banyak perhitungan ulang, menyebabkan waktu eksekusi yang lebih lama.
- Contoh: Fungsi Fibonacci rekursif tanpa memoization menghitung ulang nilai-nilai Fibonacci yang sama berkali-kali.

### 3. Kesulitan dalam Debugging:

- Melacak dan debugging fungsi rekursif bisa lebih sulit dibandingkan dengan fungsi iteratif, terutama karena banyaknya tumpukan panggilan yang harus dilacak.

Tips untuk Menulis Fungsi Rekursif

### 1. Identifikasi Kasus Dasar:

- Pastikan ada kondisi yang menghentikan rekursi untuk mencegah infinite recursion.

- Kasus dasar biasanya adalah kondisi sederhana di mana hasil dapat langsung dikembalikan tanpa perlu pemanggilan rekursif.
2. Definisikan Kasus Rekursif:
    - Tentukan bagaimana masalah dapat dipecah menjadi sub-masalah yang lebih kecil.
    - Pastikan setiap panggilan rekursif mendekatkan fungsi ke kasus dasar.
  3. Uji dengan Contoh Kecil:
    - Mulailah dengan menguji fungsi rekursif dengan contoh kecil dan sederhana untuk memastikan bahwa fungsi bekerja dengan benar.
  4. Optimasi jika Diperlukan:

Jika fungsi rekursif tidak efisien (misalnya, melakukan banyak perhitungan ulang), pertimbangkan menggunakan teknik optimasi seperti memoization.

## Materi 2.1: Penerapan Fungsi Rekursif

### 1. Faktorial

Fungsi faktorial menghitung hasil perkalian dari semua bilangan bulat positif hingga  $n$ .

```
def faktorial(n):  
    if n == 0: # Kasus dasar  
        return 1  
    else: # Kasus rekursif  
        return n * faktorial(n - 1)  
  
# Contoh penggunaan  
print(faktorial(5)) # Output: 120
```

### 2. Fibonacci

Fungsi Fibonacci menghitung angka Fibonacci ke- $n$ .

```
def fibonacci(n):  
    if n == 0: # Kasus dasar  
        return 0  
    elif n == 1: # Kasus dasar  
        return 1  
    else: # Kasus rekursif
```

```

        return fibonacci(n - 1) + fibonacci(n - 2)

# Contoh penggunaan
print(fibonacci(6)) # Output: 8

```

3. Penghitungan Pangkat  
Fungsi pangkat menghitung  $x$  pangkat  $n$ .

```

def pangkat(x, n):
    if n == 0: # Kasus dasar
        return 1
    else: # Kasus rekursif
        return x * pangkat(x, n - 1)

# Contoh penggunaan
print(pangkat(2, 3)) # Output: 8

```

## Materi 3: Optimasi Fungsi Rekursif

Optimasi fungsi rekursif adalah proses memperbaiki efisiensi dari fungsi rekursif untuk menghindari masalah seperti penggunaan memori yang berlebihan dan perhitungan ulang yang tidak perlu. Berikut adalah beberapa teknik umum untuk mengoptimalkan fungsi rekursif:

1. Memoization

Memoization adalah teknik untuk menghindari perhitungan ulang dengan menyimpan hasil dari panggilan fungsi yang telah dilakukan. Ketika fungsi dipanggil dengan parameter yang sama, hasil yang tersimpan dapat langsung digunakan.

```

def fibonacci_memo(n, memo={}):
    if n in memo:
        return memo[n]
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        memo[n] = fibonacci_memo(n-1, memo) + fibonacci_memo(n-2, memo)
        return memo[n]

# Contoh penggunaan
print(fibonacci_memo(6)) # Output: 8

```

- Fungsi `fibonacci_memo` menyimpan hasil perhitungan Fibonacci dalam dictionary `memo`.
- Sebelum menghitung nilai Fibonacci, fungsi memeriksa apakah hasil sudah ada di `memo`.
- Jika ya, fungsi mengembalikan nilai yang sudah disimpan tanpa melakukan perhitungan ulang.

## 2. Tail Recursion

Tail recursion adalah bentuk khusus dari rekursi di mana pemanggilan rekursif adalah operasi terakhir dalam fungsi. Beberapa kompiler dan interpreter dapat mengoptimalkan tail recursion untuk mencegah penggunaan tumpukan panggilan yang berlebihan.

```
def faktorial_tail_recursion(n, accumulator=1):
    if n == 0:
        return accumulator
    else:
        return faktorial_tail_recursion(n - 1, n * accumulator)
```

# Contoh penggunaan

```
print(faktorial_tail_recursion(5)) # Output: 120
```

- `accumulator` digunakan untuk menyimpan hasil perhitungan sementara.
- Pemanggilan rekursif `faktorial_tail_recursion` adalah operasi terakhir dalam fungsi, memungkinkan optimasi tail recursion.

## 3. Dynamic Programming

Dynamic Programming (DP) adalah teknik yang menyelesaikan masalah dengan membaginya menjadi sub-masalah yang lebih kecil dan menyimpan hasil dari sub-masalah tersebut untuk digunakan kembali. DP sering digunakan untuk masalah optimasi dan kombinatoria

```
def fibonacci_dp(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    dp = [0] * (n + 1)
    dp[1] = 1
    for i in range(2, n + 1):
        dp[i] = dp[i - 1] + dp[i - 2]
    return dp[n]
```

# Contoh penggunaan

```
print(fibonacci_dp(6)) # Output: 8
```

- Fungsi fibonacci\_dp menggunakan array dp untuk menyimpan hasil perhitungan Fibonacci hingga n.
- Setiap elemen dp[i] dihitung menggunakan hasil dari elemen sebelumnya.

## BAGIAN 2: LATIHAN MANDIRI (60%)

Pada bagian ini anda menuliskan jawaban dari soal-soal Latihan Mandiri yang ada di modul praktikum. Jawaban anda harus disertai dengan source code, penjelasan dan screenshot output.

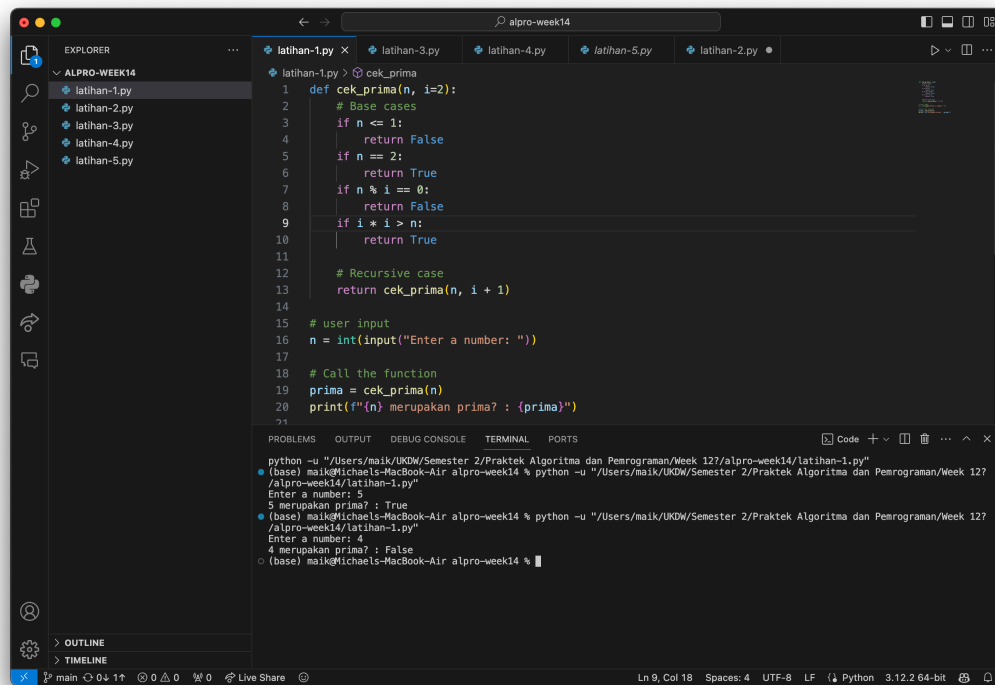
### SOAL MANDIRI 1:

Vidi adalah adik Tono yang sedang belajar bilangan prima. Vidi mengalami kesulitan untuk menentukan suatu bilangan bilangan prima atau bukan. Untuk membantu adiknya Tono kemudian membuat program untuk pengecekan bilangan prima dengan menggunakan fungsi rekursif. Bantulah Tono untuk menyelesaikan tugas tersebut.

### Source Code dan Output:

```
def cek_prima(n, i=2):  
    # Base cases  
    if n <= 1:  
        return False  
    if n == 2:  
        return True  
    if n % i == 0:  
        return False  
    if i * i > n:  
        return True  
  
    # Recursive case  
    return cek_prima(n, i + 1)  
  
# user input  
n = int(input("Enter a number: "))  
  
# Call the function  
prima = cek_prima(n)  
print(f"{n} merupakan prima?: {prima}")
```





```
1 def cek_prima(n, i=2):
2     # Base cases
3     if n <= 1:
4         return False
5     if n == 2:
6         return True
7     if n % i == 0:
8         return False
9     if i * i > n:
10        return True
11
12    # Recursive case
13    return cek_prima(n, i + 1)
14
15 # user input
16 n = int(input("Enter a number: "))
17
18 # Call the function
19 prima = cek_prima(n)
20 print(f"{n} merupakan prima? : {prima}")
```

python -u "/Users/maik/UKDW/Semester 2/Praktek Algoritma dan Pemrograman/Week 12/alpro-week14/latihan-1.py"  
• (base) maik@Michaels-MacBook-Air alpro-week14 % python -u "/Users/maik/UKDW/Semester 2/Praktek Algoritma dan Pemrograman/Week 12/alpro-week14/latihan-1.py"  
Enter a number: 5  
5 merupakan prima? : True  
• (base) maik@Michaels-MacBook-Air alpro-week14 % python -u "/Users/maik/UKDW/Semester 2/Praktek Algoritma dan Pemrograman/Week 12/alpro-week14/latihan-1.py"  
Enter a number: 4  
4 merupakan prima? : False  
• (base) maik@Michaels-MacBook-Air alpro-week14 %

## PENJELASAN:

### 1. Kasus dasar:

- Bilangan  $n \leq 1$  bukan prima.
- Bilangan  $n = 2$  adalah prima.
- Jika  $n$  habis dibagi  $i$ ,  $n$  bukan prima.
- Jika  $i \times i > n$ ,  $n$  adalah prima karena tidak ada faktor lain.

### 2. Kasus rekursif:

- Panggil kembali `cek_prima` dengan  $i + 1$  untuk memeriksa bilangan berikutnya.

Fungsi ini mulai memeriksa pembagian dari 2 dan terus naik hingga akar kuadrat dari  $n$ . Jika tidak menemukan pembagi selain 1 dan  $n$  sendiri, maka  $n$  adalah bilangan prima.

## SOAL MANDIRI 2:

Buatlah fungsi rekursif mengetahui suatu kalimat adalah palindrom atau bukan!

### Source Code dan Output:

```
def cek_palindrom(kata):  
    # Basis: jika panjang kalimat <= 1, maka kalimat tersebut adalah palindrom
```

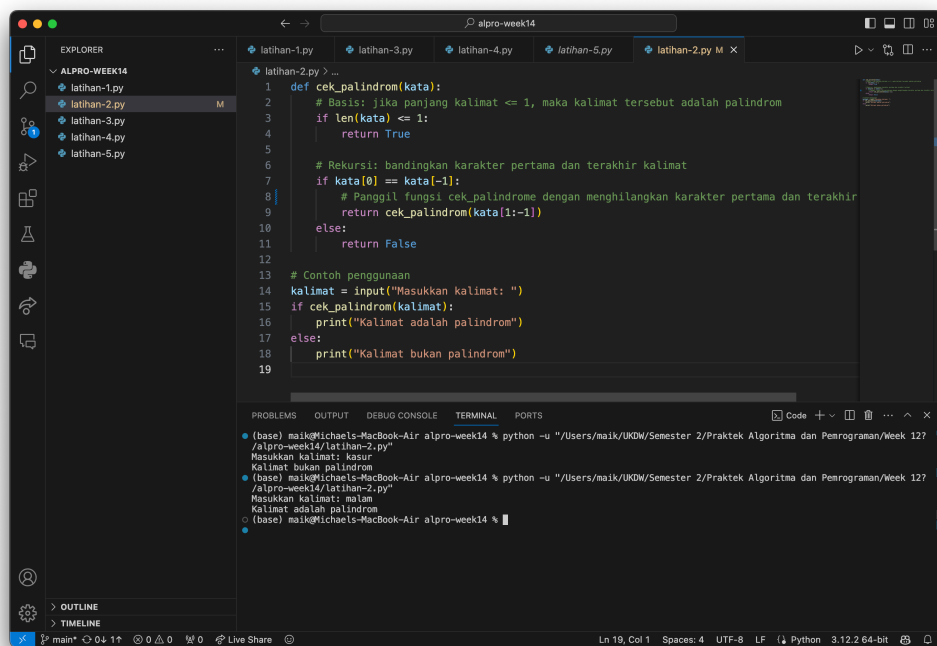
```

if len(kata) <= 1:
    return True

# Rekursi: bandingkan karakter pertama dan terakhir kalimat
if kata[0] == kata[-1]:
    # Panggil fungsi cek_palindrome dengan menghilangkan karakter pertama dan
    terakhir kalimat
    return cek_palindrom(kata[1:-1])
else:
    return False

# Contoh penggunaan
kalimat = input("Masukkan kalimat: ")
if cek_palindrom(kalimat):
    print("Kalimat adalah palindrom")
else:
    print("Kalimat bukan palindrom")

```



## PENJELASAN:

1. Kasus Dasar:
  - Jika panjang kata kurang dari atau sama dengan 1, kembalikan 'True'. Ini berarti kata tersebut adalah palindrom karena tidak ada atau hanya satu karakter.
2. Kasus Rekursif:

- Bandingkan karakter pertama ('kata[0]') dan terakhir ('kata[-1]').
- Jika sama, panggil 'cek\_palindrom' secara rekursif pada substring tanpa karakter pertama dan terakhir ('kata[1:-1]').
- Jika tidak sama, kembalikan '\*False' karena kata tersebut bukan palindrom.

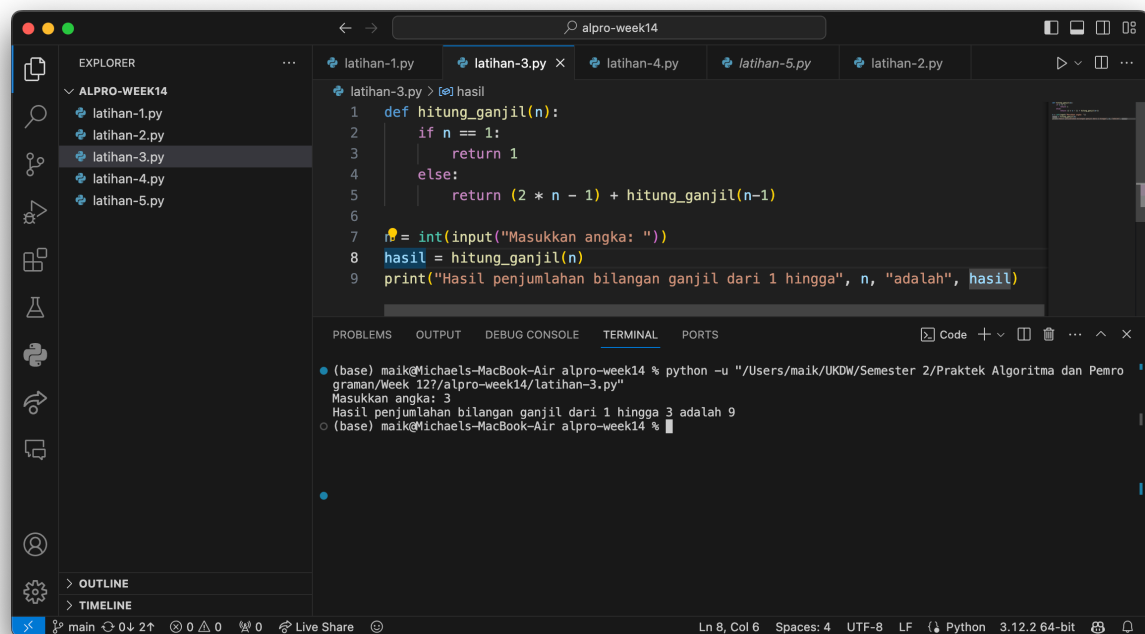
### SOAL MANDIRI 3:

Buatlah fungsi rekursif untuk menghitung jumlah deret ganjil dari  $1 + 3 + 5 + \dots + n$ !

### Source Code dan Output:

```
def hitung_ganjil(n):
    if n == 1:
        return 1
    else:
        return (2 * n - 1) + hitung_ganjil(n-1)

n = int(input("Masukkan angka: "))
hasil = hitung_ganjil(n)
print("Hasil penjumlahan bilangan ganjil dari 1 hingga", n, "adalah", hasil)
```



### PENJELASAN:

#### 1. Kasus Dasar:

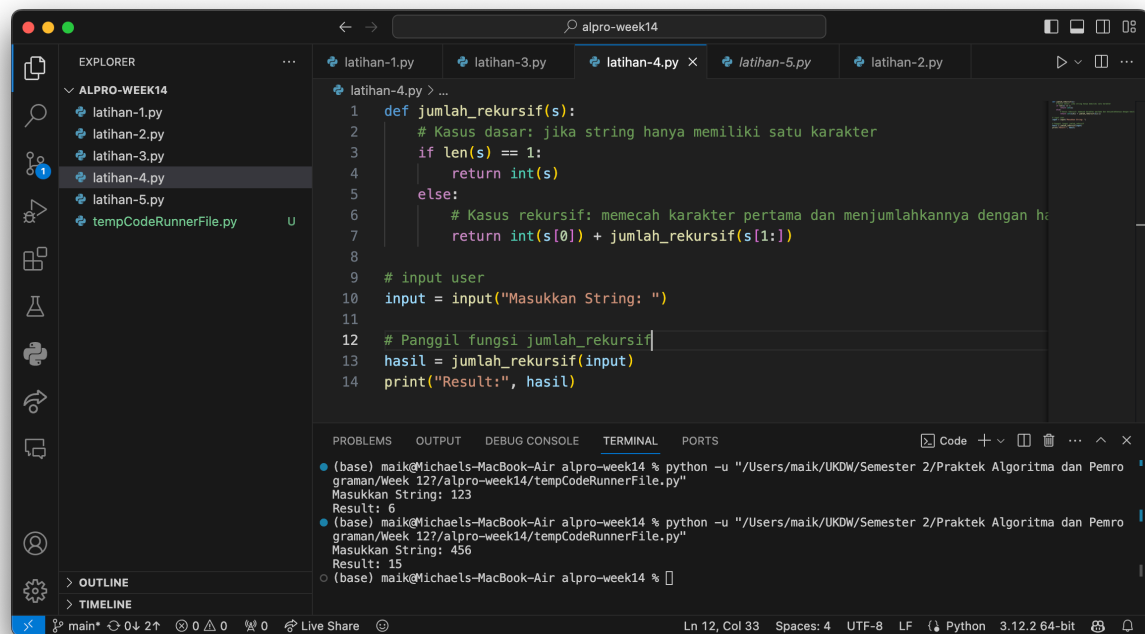
- Jika n adalah 1, kembalikan 1 karena 1 adalah bilangan ganjil pertama.
2. Kasus Rekursif:
- Jika n lebih besar dari 1, hitung bilangan ganjil ke-n yaitu  $2n - 1$  dan tambahkan dengan hasil fungsi 'hitung\_ganjil(n - 1)'.

#### SOAL MANDIRI 4:

Buatlah fungsi rekursif untuk mengetahui jumlah digit dari suatu bilangan. Seperti misalnya tulisan: "234" maka jumlah digitnya adalah  $2+3+4 = 9$ !

#### Source Code dan Output:

```
def jumlah_rekursif(s):  
    # Kasus dasar: jika string hanya memiliki satu karakter  
    if len(s) == 1:  
        return int(s)  
    else:  
        # Kasus rekursif: memecah karakter pertama dan menjumlahkannya dengan hasil  
        rekursi dari sisa string  
        return int(s[0]) + jumlah_rekursif(s[1:])  
  
# input user  
input = input("Masukkan String: ")  
  
# Panggil fungsi jumlah_rekursif  
hasil = jumlah_rekursif(input)  
print("Result:", hasil)
```



## PENJELASAN:

1. Kasus Dasar:
  - Jika panjang string hanya satu karakter, kembalikan nilai integer dari karakter tersebut.
2. Kasus Rekursif:
  - Jika panjang string lebih dari satu karakter, tambahkan nilai integer dari karakter pertama dengan hasil rekursi dari sisa string (dari karakter kedua hingga akhir).

## SOAL MANDIRI 5:

Buatlah fungsi rekursif untuk menghitung kombinasi!

### Source Code dan Output:

```
def kombinasi(n, r):  
    if r == 0 or r == n:  
        return 1  
    else:  
        return kombinasi(n-1, r-1) + kombinasi(n-1, r)  
  
n = int(input("Masukkan nilai n: "))
```

```

r = int(input("Masukkan nilai r: "))

hasil = kombinasi(n, r)
print("Hasil kombinasi dari C(", n, ",", r, ") adalah", hasil)

```

```

1 def kombinasi(n, r):
2     if r == 0 or r == n:
3         return 1
4     else:
5         return kombinasi(n-1, r-1) + kombinasi(n-1, r)
6
7 n = int(input("Masukkan nilai n: "))
8 r = int(input("Masukkan nilai r: "))
9
10 hasil = kombinasi(n, r)
11 print("Hasil kombinasi dari C(", n, ",", r, ") adalah", hasil)

```

```

(base) maik@Michaels-MacBook-Air alpro-week14 % python -u "/Users/maik/UKDW/Semester 2/Praktek Algoritma dan Pemrograman/Week 12/alpro-week14/latihan-5.py"
Masukkan nilai n: 5
Masukkan nilai r: 4
Hasil kombinasi dari C( 5 , 4 ) adalah 5
(base) maik@Michaels-MacBook-Air alpro-week14 %

```

## PENJELASAN:

### 1. Kasus Dasar:

- Jika r sama dengan 0 atau sama dengan n, maka jumlah kombinasinya adalah 1 (karena tidak ada pilihan yang harus dibuat atau semua item dipilih).

### 2. Kasus Rekursif:

- Jika r tidak sama dengan 0 atau r tidak sama dengan n, maka jumlah kombinasinya adalah hasil dari kombinasi n - 1 item yang diambil r - 1 item (yaitu memilih item ke-n) ditambah dengan kombinasi n - 1 item yang diambil r item (yaitu tidak memilih item ke-n).