


Soal:



Functional Task

Analyze this code and explain the concept of functional programming we discussed before (Immutability, referential transparency, and higher-order functions)!

```
def increment_list(numbers):  
    return [num + 1 for num in numbers]  
  
def square(x):  
    return x * x  
  
def apply_function(func, data):  
    return [func(x) for x in data]  
  
# Sample data  
numbers = [1, 2, 3, 4, 5]  
  
# Apply functions  
incremented_numbers = increment_list(numbers)  
squared_numbers = apply_function(square, numbers)  
  
# Print results  
print("Original:", numbers)  
print("Incremented:", incremented_numbers)  
print("Squared:", squared_numbers)
```

Jawaban:

Kode yang diberikan menunjukkan prinsip **pemrograman fungsional**, yaitu:

1. Immutability (Ketidakmutabilan)

- List numbers tidak dimodifikasi secara langsung. Sebagai gantinya, fungsi `increment_list` dan `apply_function` mengembalikan list baru, menjaga data asli tetap utuh.


2. Referential Transparency (Transparansi Referensial)

- Fungsi `square(x)` selalu menghasilkan output yang sama untuk input yang sama tanpa efek samping.
- Fungsi `increment_list` dan `apply_function` juga memiliki sifat ini karena hanya mengolah data tanpa mengubah keadaan eksternal.

3. Higher-Order Function (Fungsi Orde Tinggi)

- `apply_function(func, data)` adalah contoh fungsi orde tinggi karena menerima fungsi lain (`func`) sebagai argumen dan menerapkannya ke setiap elemen dalam data.

Soal:



Fakultas Informatika
School of Computing
Telkom University

Make the source code that demonstrates Immutability, Referential Transparency and Higher Order Function in single running program.

Home Work

Jawaban:

```
from functools import reduce

# Immutability: Tidak mengubah list asli, hanya membuat yang baru
def increment_list(numbers):
    return [num + 1 for num in numbers]

# Referential Transparency: Fungsi selalu mengembalikan hasil yang sama untuk input yang sama
def square(x):
    return x * x

# Higher-Order Function: Menerima fungsi sebagai parameter
def apply_function(func, data):
    return [func(x) for x in data]

# Higher-Order Function dengan reduce untuk demonstrasi lebih lanjut
def product_of_squares(data):
    return reduce(lambda x, y: x * y, apply_function(square, data))

# Sample data
numbers = [1, 2, 3, 4, 5]

# Apply functions
incremented_numbers = increment_list(numbers)
squared_numbers = apply_function(square, numbers)
product = product_of_squares(numbers)

# Print results
print("Original:", numbers)
print("Incremented:", incremented_numbers)
print("Squared:", squared_numbers)
print("Product of Squares:", product)
```

Original: [1, 2, 3, 4, 5]
Incremented: [2, 3, 4, 5, 6]
Squared: [1, 4, 9, 16, 25]
Product of Squares: 14400

Brian Nugraha Wiyono

2311110052 (SD0401)