Algorithm and Data Structures
Week 2

Endang Wahyu Pamungkas, P.hD.

If Statement

- Python uses decision-making statements using the **if** keyword.
- If the condition evaluates to True, the block of statements inside the if statement will be executed.
- If the condition evaluates to False, the next block (after the IF statement) is executed.

```
if <condition>:
    instructions
```

Else Statement

- The **else** statement can be combined with the if statement, as a branch that is executed when the condition / result evaluates to False.
- Else is optional.
- However, there can be no more than one else statement in a block of code.

```
if <condition>:
    instructions1
else:
    instructions2
```

Elif Statement

- Elif is short for else if, and is a branch of the "if" logic. With elif we can create program code that will select several possibilities that can occur.
- An IF statement can be followed by one or more elif statements (optional and not restricted).

```
if <condition>:
      instructions1
elif <condition>:
      instructions2
else:
      instructions3
```

Boolean

- The boolean data type is a data type that has only two values, namely True and False. Values with a boolean data type can be applied to a condition when making decisions. Values True and False can also be converted into the context of numbers, namely 1 for True and 0 for False.

Operasi	Untuk tipe data	Ekspresi boolean	Contoh	Hasil operasi dari contoh
AND	boolean	and	True and False	False
OR		or	True or False	True
NOT		not	not True	False

Looping

- There are two kinds of looping in Python. Looping in collection and looping in a condition.
- A collection loop is a loop that uses a collection as the basis for the loop. The number of iterations will be equal to the number of items in the collection.
- Conditional looping is a method used by computers to run commands repeatedly as long as the conditions are still met.

Looping

```
for letter in 'Informatics':
   print (letter)
```

```
number = 0
while (number < 5):
    print(number)
    number += 1
print("Program completed")
```

Function

- A function is a group of instructions made into a union. The minimum function is created with the keyword def followed by the function name and parentheses ().

```
def function_name():
      instructions
```

- If we define a set of functions in a program and run it. Nothing happens. A function should be called to make it executed.

```
def greet_bryan():
   print("Hi Bryan")
   print("Nice to see you!")
greet_bryan()
```

Return Statement

- Functions can return values. In order for the function to return a value, a return command is added in an instruction line followed by the value or variable to be returned.

```
def first():
    """Returns the first item"""
    product = ["shirt", "shoes", "pants",
    "shirt", "hat"]
    return product[0]

result = first()
print(result)

print(first())
```

Return Statement

- Functions can return values. In order for the function to return a value, a return command is added in an instruction line followed by the value or variable to be returned.

```
def first():
    """Returns the first item"""
    product = ["shirt", "shoes", "pants",
    "shirt", "hat"]
    return product[0]

result = first()
print(result)
```

Argument of Function

- Arguments are variables that hold input values to be processed inside the function. The value of the argument is entered when the function is called. Writing a function with arguments has the following syntax.

```
def area_triangle(base, height):
      """ Returns the area of a triangle with
      with input base and height """
      area = 0.5 * base * height
   return area

print(area_triangle(5, 10))
```

Recursion

- Recursion is a problem-solving method in which a solution to a problem depends on the solution of smaller problems that are part of the problem.
- An example of a well-known recursion problem is factorial computation. Factorial is a value obtained by multiplying a number by positive numbers that are smaller than that number. For example, the factorial of 4 is 4 x 3 x 2 x 1 = 24.

Recursion Function

- A recursive function is a function that calls itself. Calls inside functions are needed to solve smaller problems before they can solve the whole problem.
- When creating a recursive function, there are two important things to consider: the base case and the recursive case.
  - Base Case

    Base case is a condition in which the function will stop (no longer repeat)
  - Recursive case

    The recursive case is the condition in which the function will call itself.

Recursion Function

```
def function_recursive(arguments):
    """ recursive function """
    if (condition): # base case
       <instructions>
    else: # recursive case
       <step reduction>

def factorial(number):
     if number == 1 or number == 0: # base case
        returns 1
     else: # recursive case
        return number * factorial(number-1) #step
reduction

num = int(input("Enter Number : "))
print("The factorial of", num, "is", factorial(num))
```

Exercise

- Write a program that checks whether a number is prime or not prime!
- Example :

  Input : 5
  Output : Prime

  Input : 4
  Output : Not Prime

Rubic for Prime or Not

- Correctness
  - Program correctly identifies prime and non-prime numbers. (30 points)
  - Handles edge cases (e.g., 0, 1, negative numbers) appropriately. (20 points)
- Code Quality
  - Code can run without error. (30 points)
  - Proper use of comments for clarity and explanation. (20 points)

Exercise

- Write a program in Python to remove repetitive items from a list.

  Given : [2,2,3,4,4,4,5,5,6,7]
  Expected output : [2,3,4,5,6,7]

Rubic for Remove Repetitive Items

- Correctness
  - Program correctly remove repetitive items. (30 points)
  - Handles edge cases (e.g., empty list, list with no repetitive items) (20 points)
- Code Quality
  - Code can run without error. (30 points)
  - Proper use of comments for clarity and explanation. (20 points)

Exercise

- Write a program to calculate the sum of all number between 1 and a given number (input).

  input = 5
  output = 15

- For example, if the input is 5 so the program need to calculate 1 + 2 + 3 + 4 + 5. which is 15.

Rubic for Sum All Number

- Correctness
  - Program correctly sum all number. (30 points)
  - Handles edge cases (e.g., input of 0, negative numbers) (20 points)
- Code Quality
  - Code can run without error. (30 points)
  - Proper use of comments for clarity and explanation. (20 points)