Functions

age = 40

Start

age ≥ 18

True

False

print("Y")

print("N")

Exit

Code Reuse

POP
→ **function**

OOP

association
→ Composition
→ aggregation

inheristance
→ simple
→ multiple
→ multilevel
→ hierarchical
→ hybrid
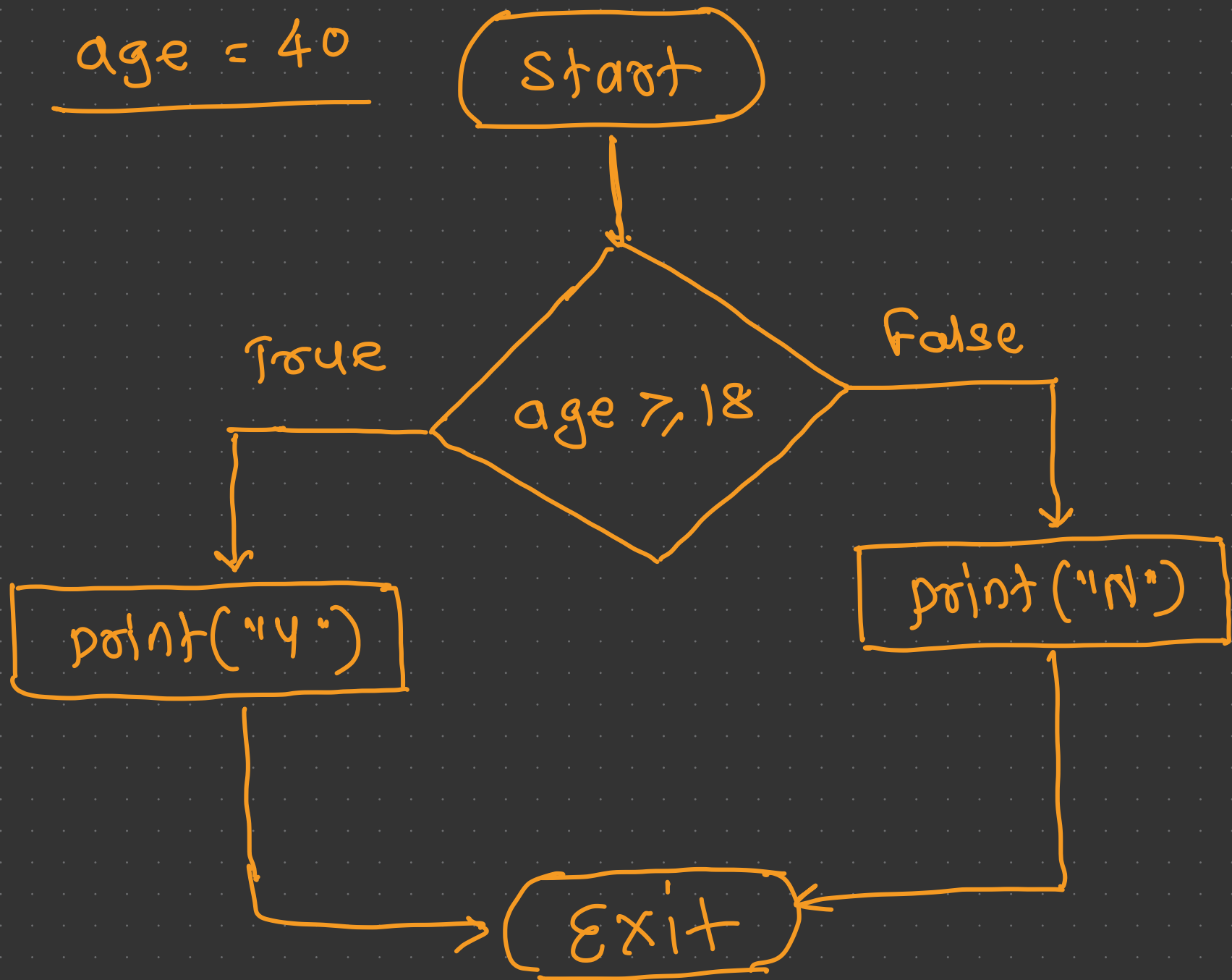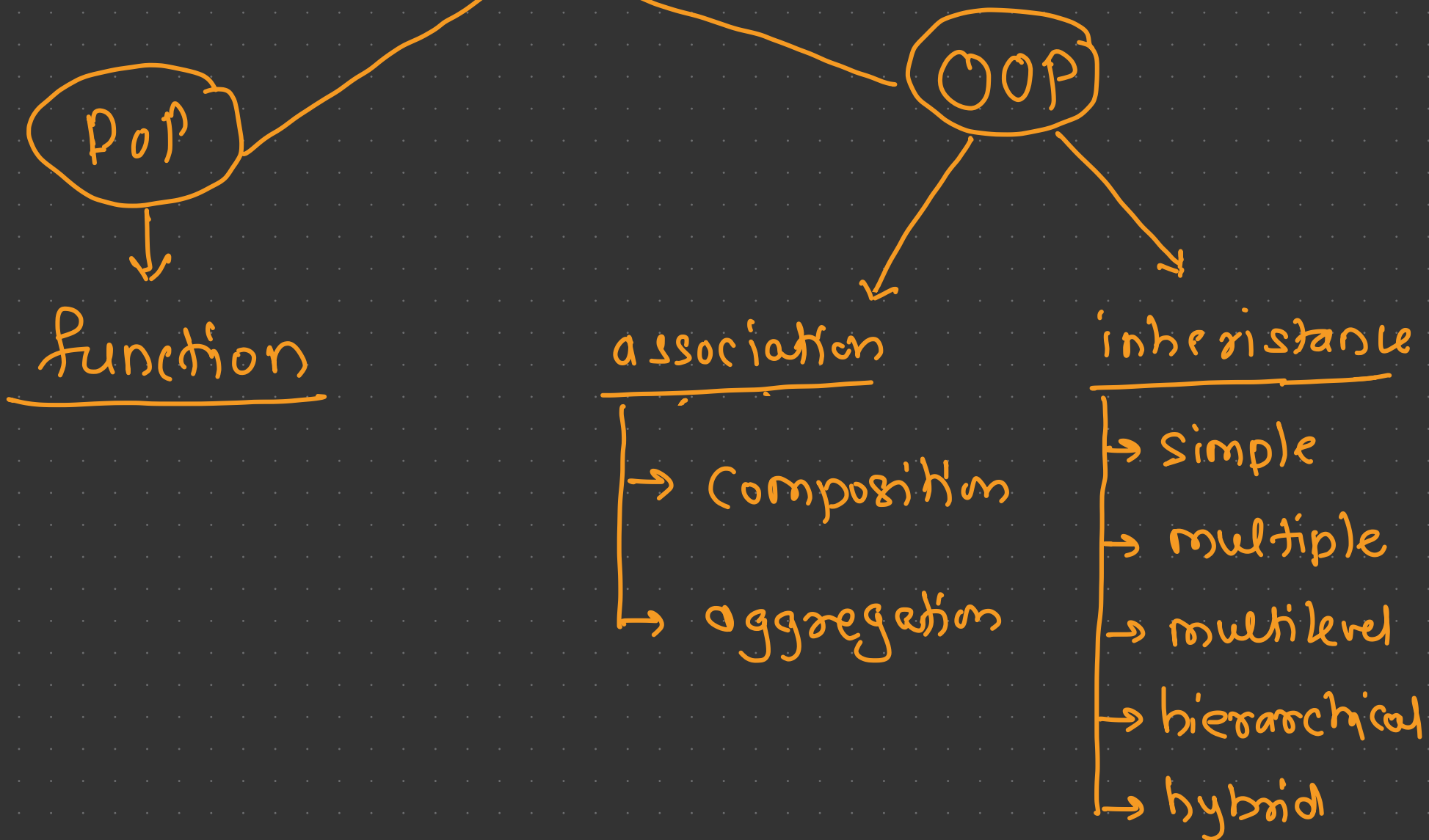
Functions

- In Python, a function is a group of related statements that performs a specific task

- Functions help break our program into smaller and modular chunks

- As our program grows larger and larger, functions make it more organized and manageable

- Furthermore, it avoids repetition and makes the code reusable

- Syntax

```
def function_name(parameters):
    """docstring"""
    statement(s)
```

← document string

→ help of the block

→ optional

→ ignored by compiler

Docstrings

- The first string after the function header is called the docstring and is short for documentation string

- It is briefly used to explain what a function does

- Although optional, documentation is a good programming practice

- We generally use triple quotes so that docstring can extend up to multiple lines

- This string is available to us as the __doc__ attribute of the function

Function Types

- Functions can divided into following two types:
    - **Built-in functions**
        - Functions that readily come with Python are called built-in functions
        - If we use functions written by others in the form of library, it can be termed as library functions
        - E.g., str(), int(), float() etc.

    - **User defined functions**
        - Functions that we define ourselves to do certain specific task are referred as user-defined functions
        - User-defined functions help to decompose a large program into small segments which makes program easy to understand, maintain and debug
        - If repeated code occurs in a program. Function can be used to include those codes and execute when needed by calling that function.
        - Programmers working on large project can divide the workload by making different functions

Returning a value

- The return statement is used to exit a function and go back to the place from where it was called

- This statement can contain an expression that gets evaluated and the value is returned

- If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object

Scope of variables

- Scope of a variable is the portion of a program where the variable is recognized

- Parameters and variables defined inside a function are not visible from outside the function. Hence, they have a local scope.

- The lifetime of a variable is the period throughout which the variable exists in the memory

- The lifetime of variables inside a function is as long as the function executes

- They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Local Scope

- A variable declared inside the function's body or in the local scope is known as a local variable

```python
def foo():
    local_var = "local"


foo()
# error
print(local_var)
```

Global Scope

- In Python, a variable declared outside of the function or in global scope is known as a global variable
- This means that a global variable can be accessed inside or outside of the function

```python
g_var = "global"
def foo():
    print("inside foo")
    print(g_var)


foo()
```

Global Keyword

- In Python, global keyword allows you to modify the variable outside of the current scope

- It is used to create a global variable and make changes to the variable in a local context

- Rules of global Keyword
  - When we create a variable inside a function, it is local by default
  - When we define a variable outside of a function, it is global by default. You don't have to use global keyword
  - We use global keyword to read and write a global variable inside a function
  - Use of global keyword outside a function has no effect

Nested Function

- Function within a function is called as nested function or inner function

- E.g.

```python
def outer():
    print("inside outer")
    def inner():
        print("inside inner")
    inner()


outer()
# error
inner()
```

Anonymous/Lambda Function

- In Python, an anonymous function is a function that is defined without a name
- While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword
- Hence, anonymous functions are also called lambda functions
- Syntax
    - lambda arguments: expression
- Characteristics
    - It can only contain expressions and can't include statements in its body
    - It is written as a single line of execution
    - It does not support type annotations
    - It can be immediately invoked