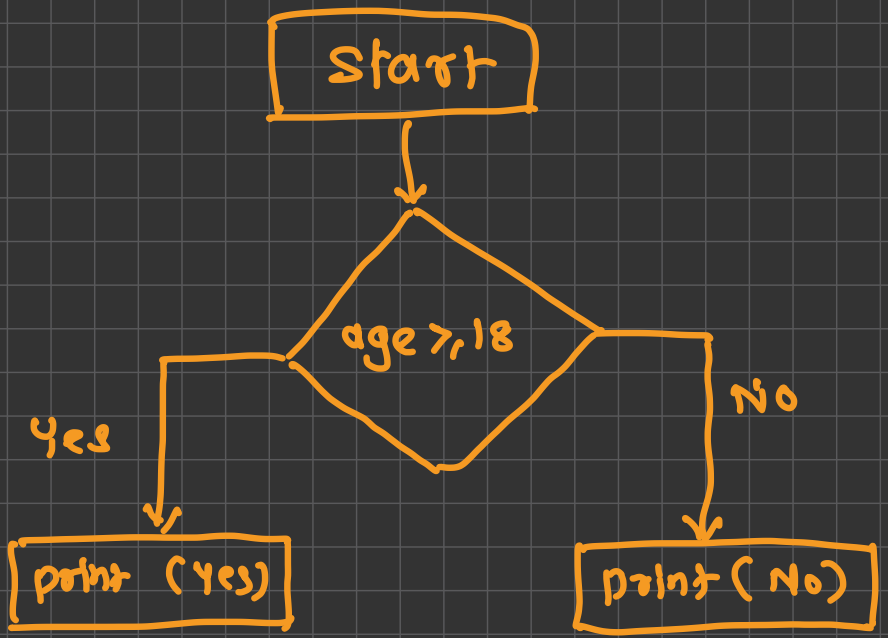
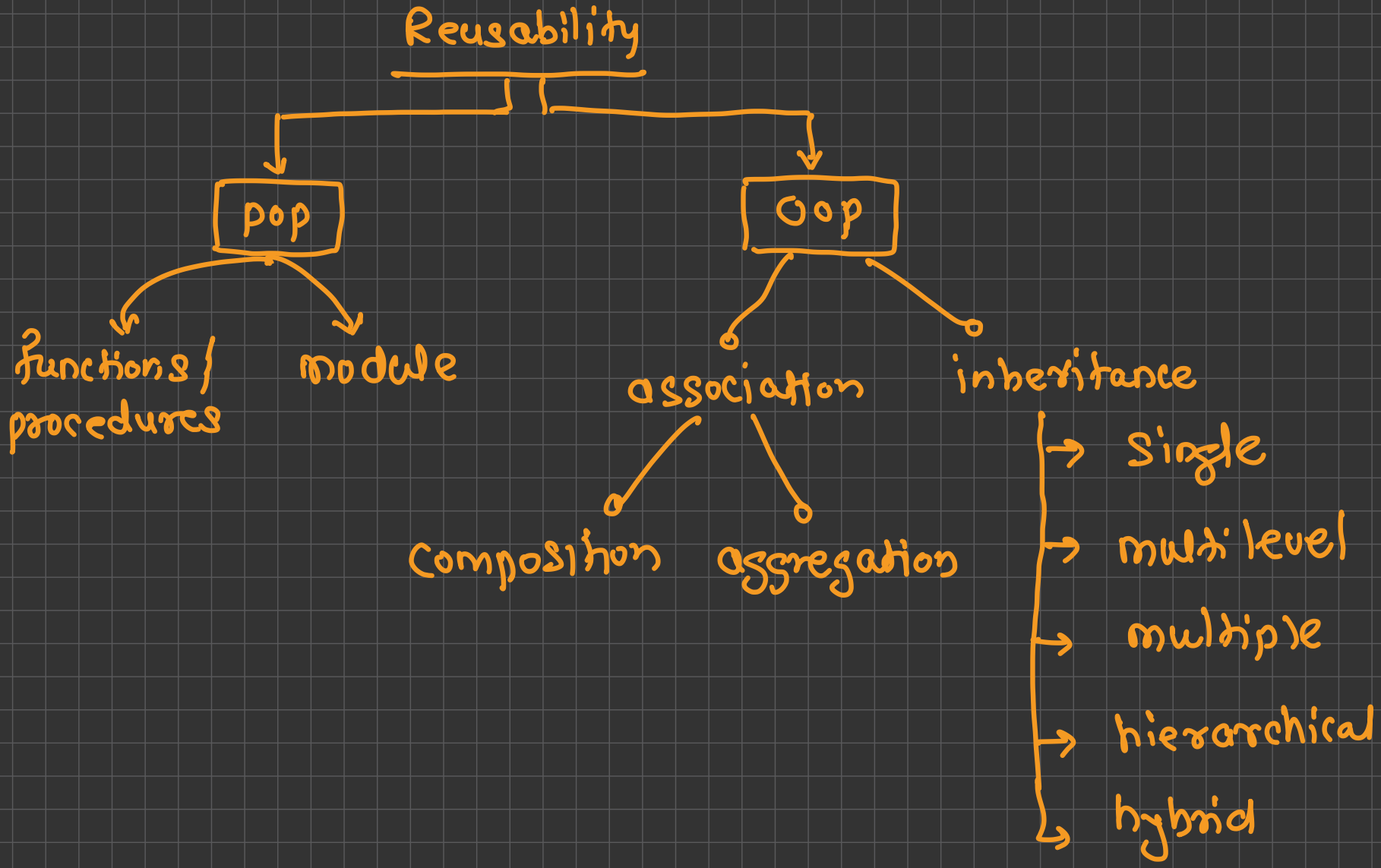
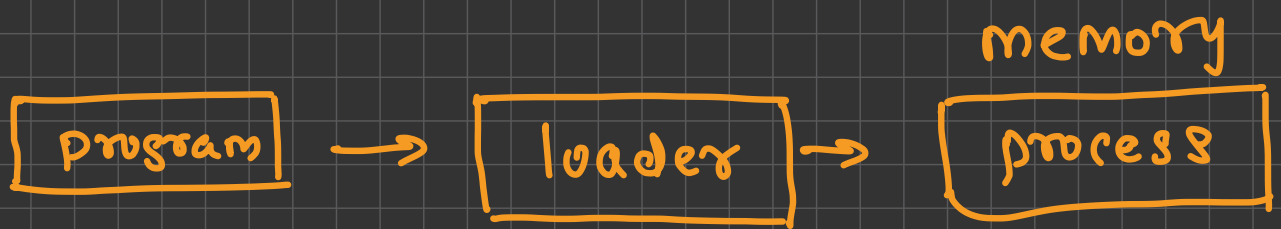
Functions

age = 19




Reusability
pop
oop
functions / procedures
module
association
inheritance
composition
aggregation
single
multilevel
multiple
hierarchical
hybrid

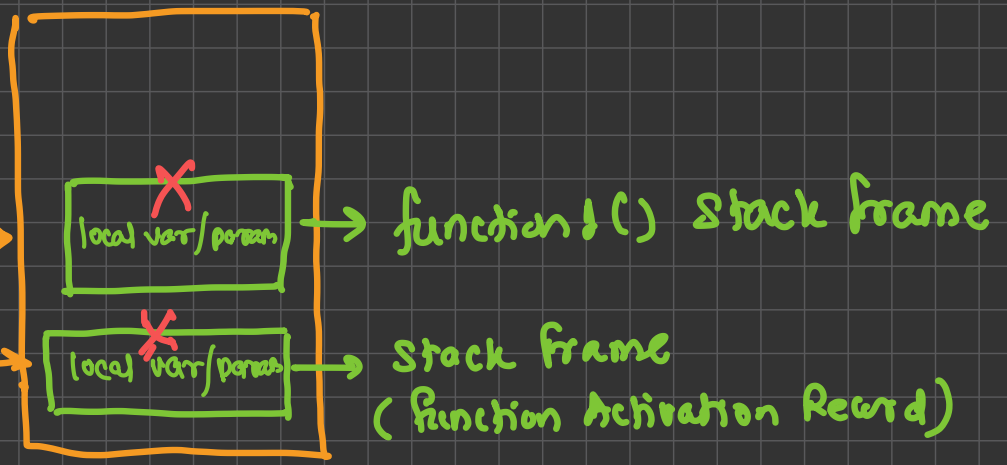


① call entry point function
→ main() / Main()

② function1() gets called

③ function1() gets exited

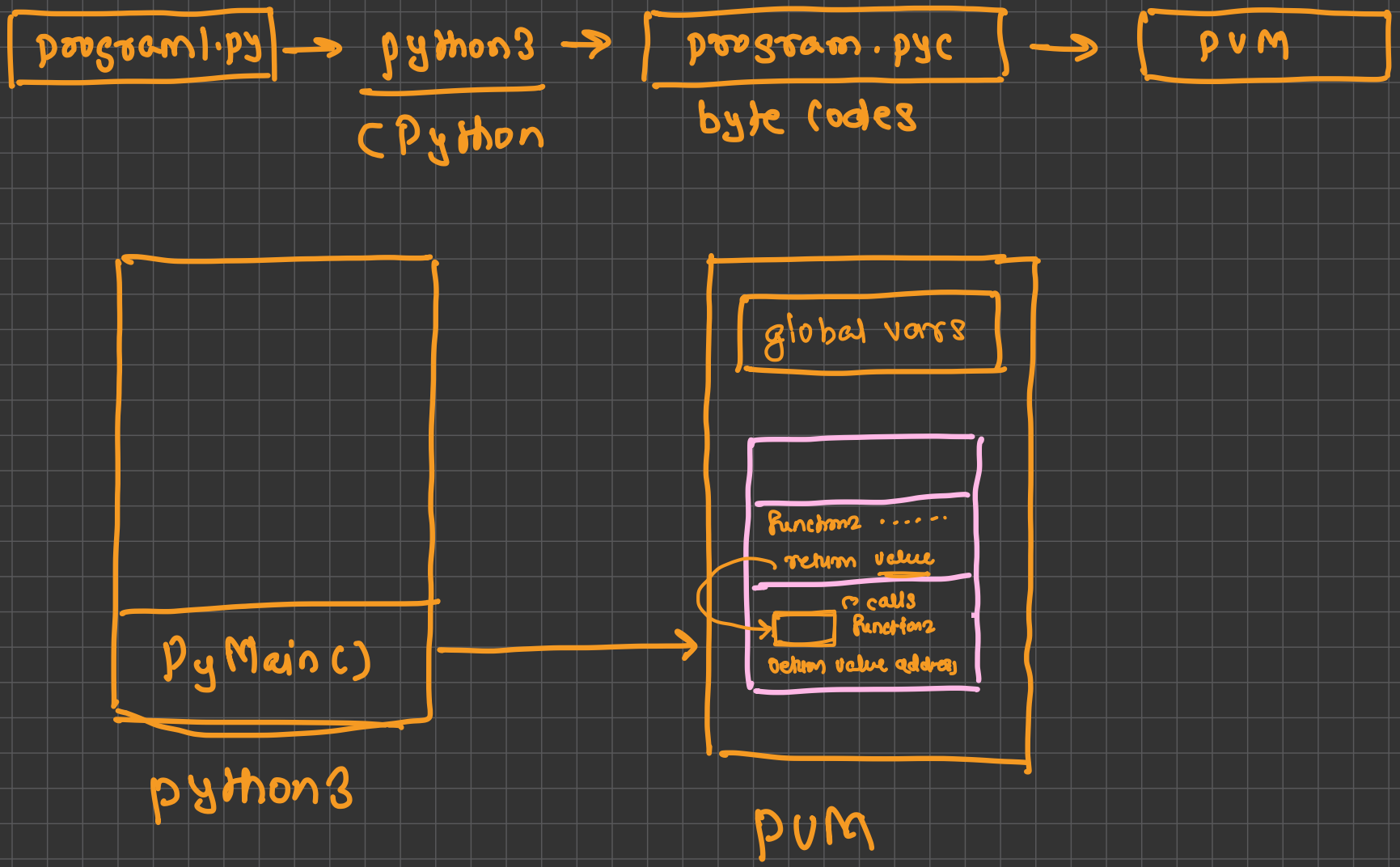
④ function1 stack frame gets deleted & removed from stack



Stack → last in first out

⑤ main() returns & stack frame get removed

⑥ the process exits (everything will be removed from memory)


program1.py
python3
CPython
program.pyc
byte codes
PVM
PyMain()
python3
global vars
function ....
return value
calls function
return value address
PVM

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)
```

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

↓

dunder doc

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** (custom 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

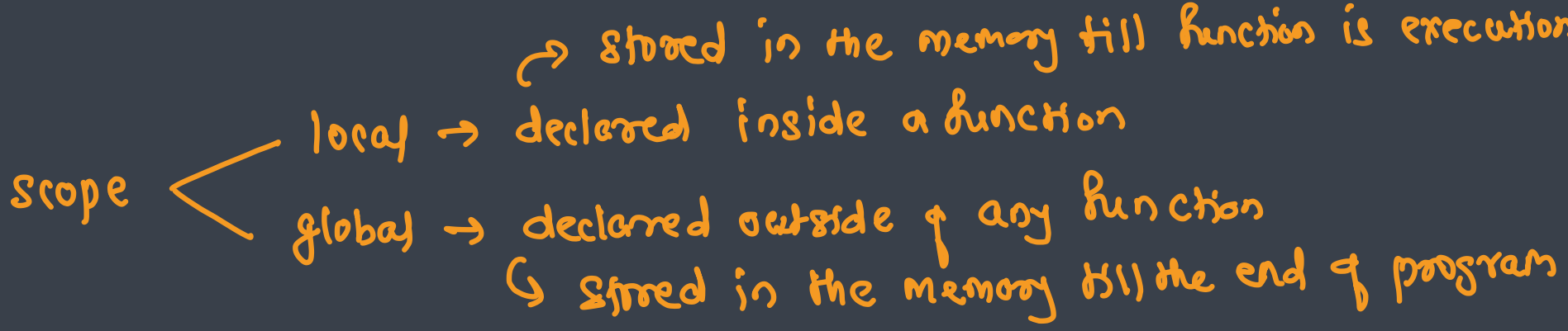
Returning a value

- 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

- a function can return one and only one value

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

```
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

```
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 → local function / inner function

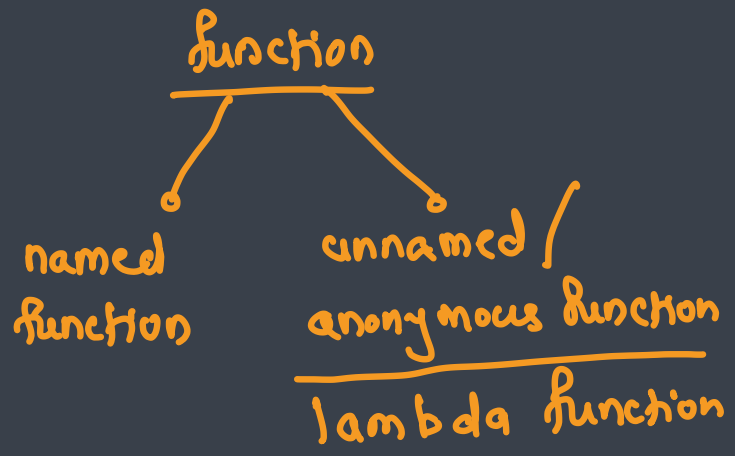
- Function within a function is called as nested function or inner function
- E.g.

```
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



num1 = 200

0x100

500

num1

num1 = 500

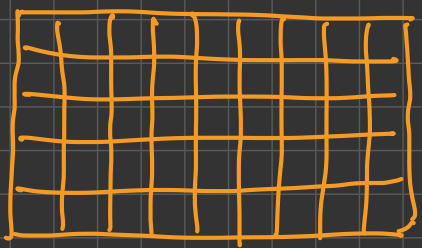
num2 = num1

0x200

200

num2

```
def function1():
    print("inside function1")
```



memory (RAM)

0x600

0x500

function1 reference

0x500

```
function body:
print("...")
```

function

```
my_function1 = function1
```

0x800

0x500

my_function1 reference

python is a functional programming language

→ function is considered as a first class citizen
  ↳ function is treated as a variable of type function

→ a function can be passed as a parameter to another function
  ↳ map(), filter(), reduce()

→ a function can be returned as a return value of another function
  ↳ closure → decorator