4 - bindings and declaration

identifiers = given names to program entities (variable, constant, function, type)
↳ most important feature of high level languages
↳ declared once, used n times

binding = finding the corresponding binding definition/deelation (occurance) for an applied usege (occurance) of an identifier

$$\begin{tikzcd}\boxed{\begin{array}{c}\text{binding occurance}\\ \text{definition of an idetifier}\end{array}} \arrow[r, shift left] & \boxed{\begin{array}{c}\text{applied occurance}\\ \text{used position of an identifier}\end{array}} \arrow[l, shift left]\end{tikzcd}$$

☆ the scope of identifiers should be known, to access and not to use same name for other identifiers

enviroment = the set of binding occurences that are accessible at a point in a program
↳ "{ }" fields → block structure
blocks ⇒ define scope and lifetime of the variables

block structures

monolithic = whole program is a block, all identifiers' scope is global → cobol
flat block = global scope + only one single level local scope → fortran, C (partially)
nested block = multiple blocks with nested local scopes → Pascal, Java, C

hiding = identifiers defined in the inner block hides the outer block identifiers with the same name during their scope (they cannot be accessed within the inner block)

dynamic / static scope / binding

static = based on lexical structure, binding is done at compile time
dynamic = defined during the execution, the enviroment changes at run time. current stack of activated block is significant (most recent deeleration) called from which function

```
x=3
f(){x+y}
g(){x=5; f()}
main(){f()}
```

x=5
x=3
trace to find out

☆ language processor keeps track of curent environment in a symbol / identifier table usually implemented as a hash table. the table maps identifier strings to their type and binding

definition = creating a new name for an existing binding
declaration = creating a completely new binding.