TCL scripting :-

lec (01)

→ Road Map :-

1. Introduction
2. Basics (Array, lists, strings, Hashes, loops)
3. functions (procs)
4. file Handling
5. Pattern Matching
6. Real world related tasks in TCL

---

lec (02)

TCL Basics :-

Required

puts :-

Program :-

```
puts "hello";
```

Output :- hello.

→ if we use two statements with puts we need to keep ; in between

eg:-

```
puts "Hello" ; puts "semicolon"
```

Variable declaration:-

set command is used for declaring the variables.

eg:- input:-

```
set str "Hello";
puts $str ;
```

output:-

Hello.

→ '$' is used to print the variable in command line.

Lec 03

⊙ **Square Braces :-**

"[ ]" are used for assigning value to variables from other expressions.

⊛ **example :-**

```
Set a "5";
set b $a;
```

this statement will assign 'b' to 'a'

instade of this we can use "[ ]".

```
Set b [set a "5"]
```

this is called interpolation.

⊙ Curly Braces :-

Basically, curly Braces doesn't recognize variables. it print $variable as it is.

example :-

inputs :-

```
set value 1.5;
puts {the value is $value};
puts "the value is $value";
```

Outputs :-

```
the value is $ value
the value is 1.5
```

lec(04):-

Arithmetic Operation

Expr Statement:-

→ this statement is used to evaluate the expression.

Syntax:-

Expr variable 1 operator variable 2.

example:-

```
Set a "5";
Set b "3";
Set c [expr "$a + $b"];
```

first this expression is evaluating

then the value which is evaluated and give it to 'c' variable.

Some Expression :-

Expr Statement :-

expr 10+5 ⟶ 15

expr 10.0+5 ⟶ 15.0

expr 12/7 ⟶ 1

expr 12/7+3 ⟶ 4

expr 12.0/7+3 ⟶ 4.71428

→ So, TCL compiler calculates floating pt i/p considering every thing in floating point and gives o/p like wise.

→ and for integer i/p it will consider as integer so, it will give o/p likwise.

Logical Operations :-

logical AND → &&

logical OR → ||

logical NOT → !

Bitwise Operator :-

& → Bitwise AND

| → Bitwise OR

^ → Bitwise XOR

---

lec 05 :-

Conditional Statements and Loops :-

① If conditions :-

Syntax :-

if expr1 ? then ? else if exp2 - - - -

Example :-

```
Set x = 1;
if ($x == 2) {puts "x is 2"}
else {puts "x is not 2"}
```

Here output is :-

⟶ x is not 2

② While loop :-

Example :-

```
#while loop

Set x = 0;
while ($x < 6)
{   Puts "x is $x";
    set x [expr $x + 1]
}
```

---

for loop :-

Syntax :-

```
for {initilization} {condition} {incr}
   {  statement (s);
   }
```

lec 06 :-

Arrays :-

Array Declaration :- Arrays are ordered set of values.

Syntax :-

→ set ArrayName [Index] value

Example :- **inputs :-**

```
set institute(0) ABC
set institute(1) Hello
set institute(2) Hii

puts $institute(0)
puts $institute(1)
puts $institute(2)
```

Output :- ABC, Hello, Hii

loop over array

inputs :-

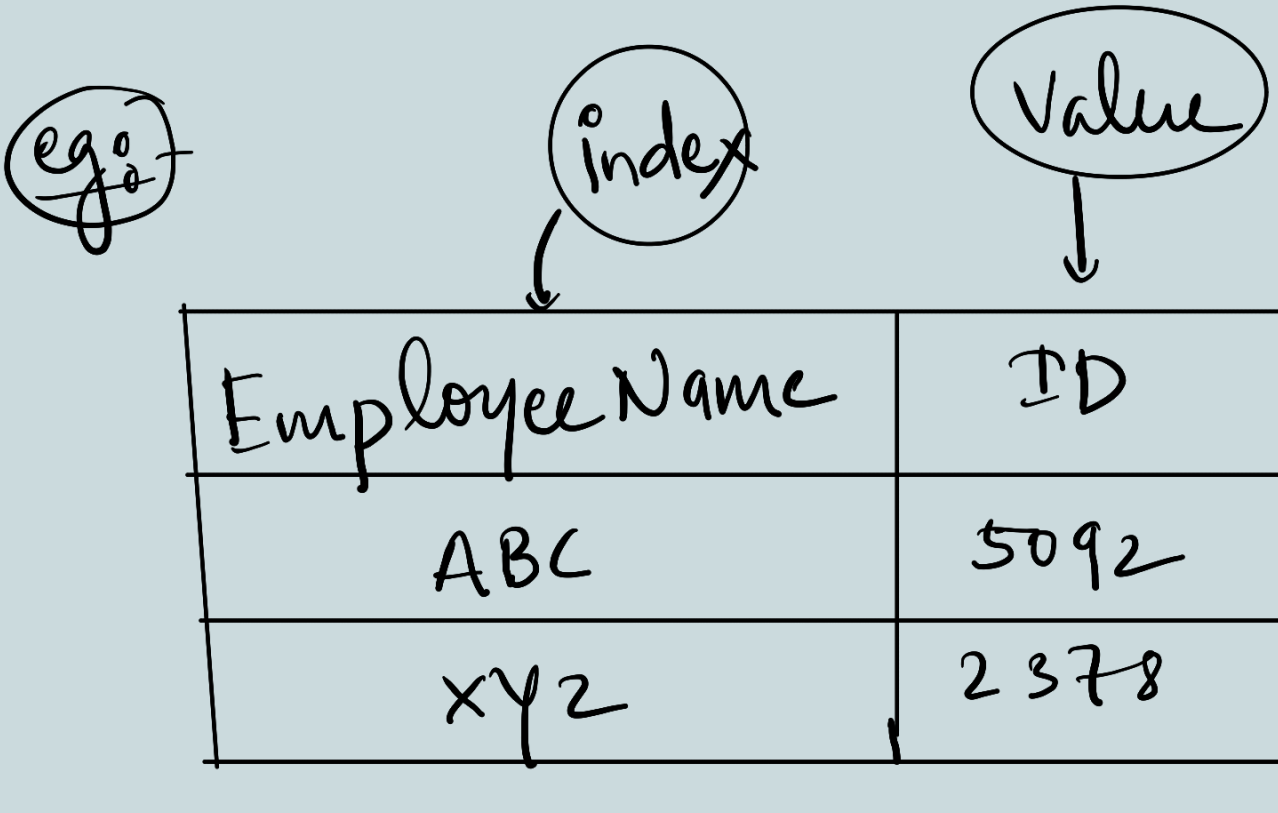
```
set institute(0) ABC
set institute(1) Hello
set institute(2) Hii

for{set index 0} {$index <[array
size institute]} {incr index}
{puts "institute ($index):
        $institute($index)}
}
```

Output :-

```
institute (0) : ABC
institute (1) : Hello
institute (0) : Hii
```

⊙ Associative Array:-



→ this is also dictionaries.

→ Associative arrays are un ordered sets of values which can be accessed with corresponding keys.

→ Associative arrays have an index which is not necessarily a no.

Example:-

```
set Employee1 (Name) "ABC"
set Employee 1 (Age) 24
Puts Employee1 (Name)
Puts Employee1 (Age)
```

Syntax :-

Retrieving indices :-

```
[array Names arrayName]
```

Eg :- (input :-)

```
set Employee 1 (Name) "ABC"
set Employee1 (Age) 24
puts [array Names Employee1]
```

Output :-

```
Name, Age
```

---

foreach loop:-

→ foreach is more useful in the event of associative arrays.

Eg:-

```
set VLSI(0) AB
set VLSI(1) CD
set VLSI(2) EF
```

```
foreach index [array Name VLSI]
    puts " VLSI($index): $ VLSI($index)
```

Basically instead of for loop we can use it.

Lec 07 exec command :-

→ it can be used to execute shell command.

example :-

exec ls → list all the directories.

exec ls *.tcl → list all .tcl files in directory

we can use any shell command in tcl script with the help of 'exec'

Lec 08

TCL Datastructures : list

→ A list is simply an ordered collection of numbers, words, strings, or other lists.

→ it can be created by several ways :-

① by setting a variable to be a list values

```
set lst [{item1} {item2} ...]
```

② with split command

```
set lst [split "item1. item2" "."]
```

③ with list command.

```
set lst [list "item1" "item2"]
```

⊙ Some important commands:-

① lindex listName index

Returns the index'th item from the list.

② llength listName

Return the no. of elements in a list.

Eg:-

```
set x "a b c"
puts "Item at index 2 is :[lindex $x 2]"
```

Lec (9) List operations :-

① lappend :- Appends new item at the last to existing list.

② Lsort :- sort list and returns a new list in the sorted order. By default, sorts the list in dictionary order.

③ Lrange :- Return a list composed of the first through last entires in the list.

Example :-

```
① set x "a b c"
  set y "d"
  Puts "old x : $x"
  lappend x $y
  Puts "updated x : $x"
```

output updated x = a b c d.

---

② lsort :-

```
set x "c d a b"
set sorted_x [lsort $x]
```

---

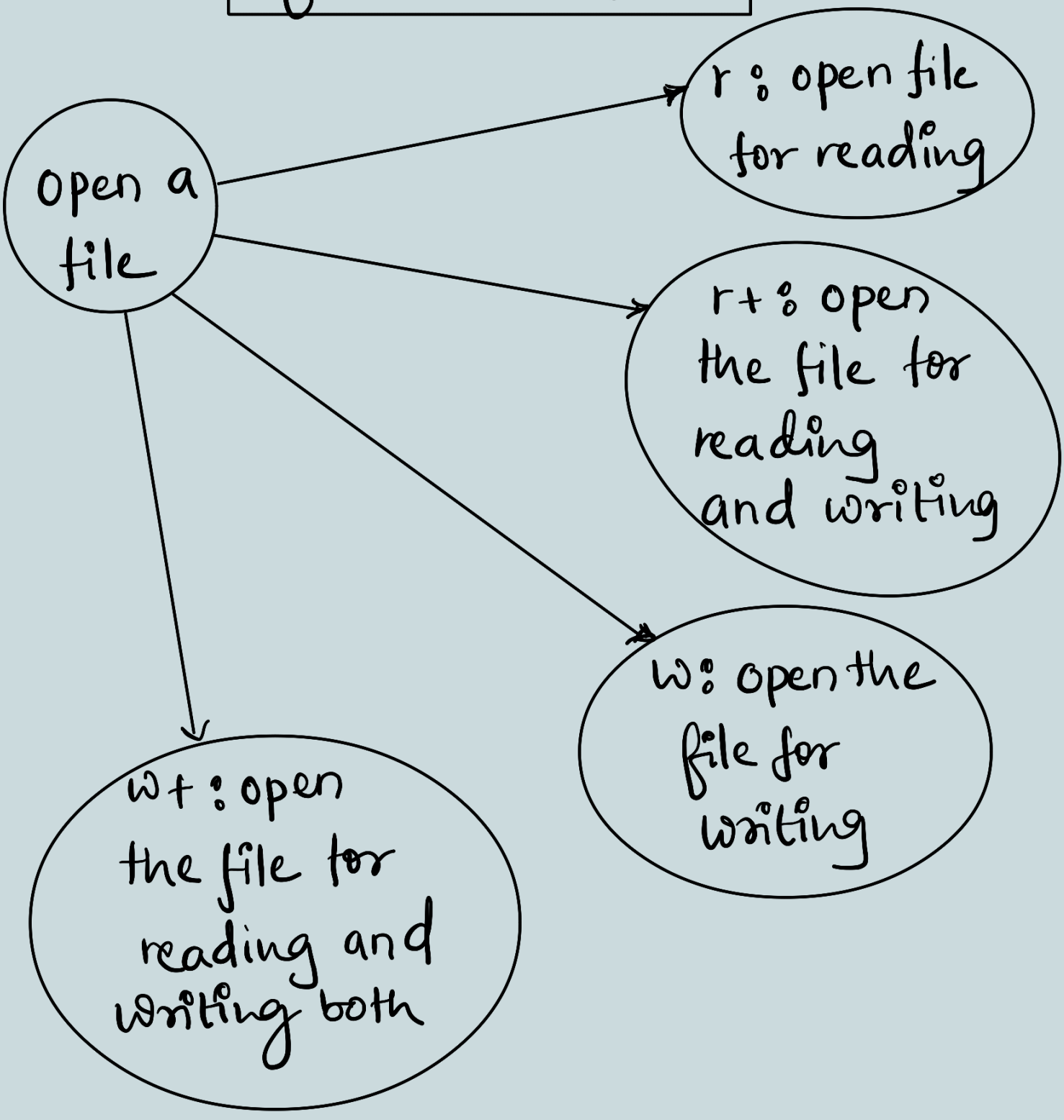
③ lrange :-

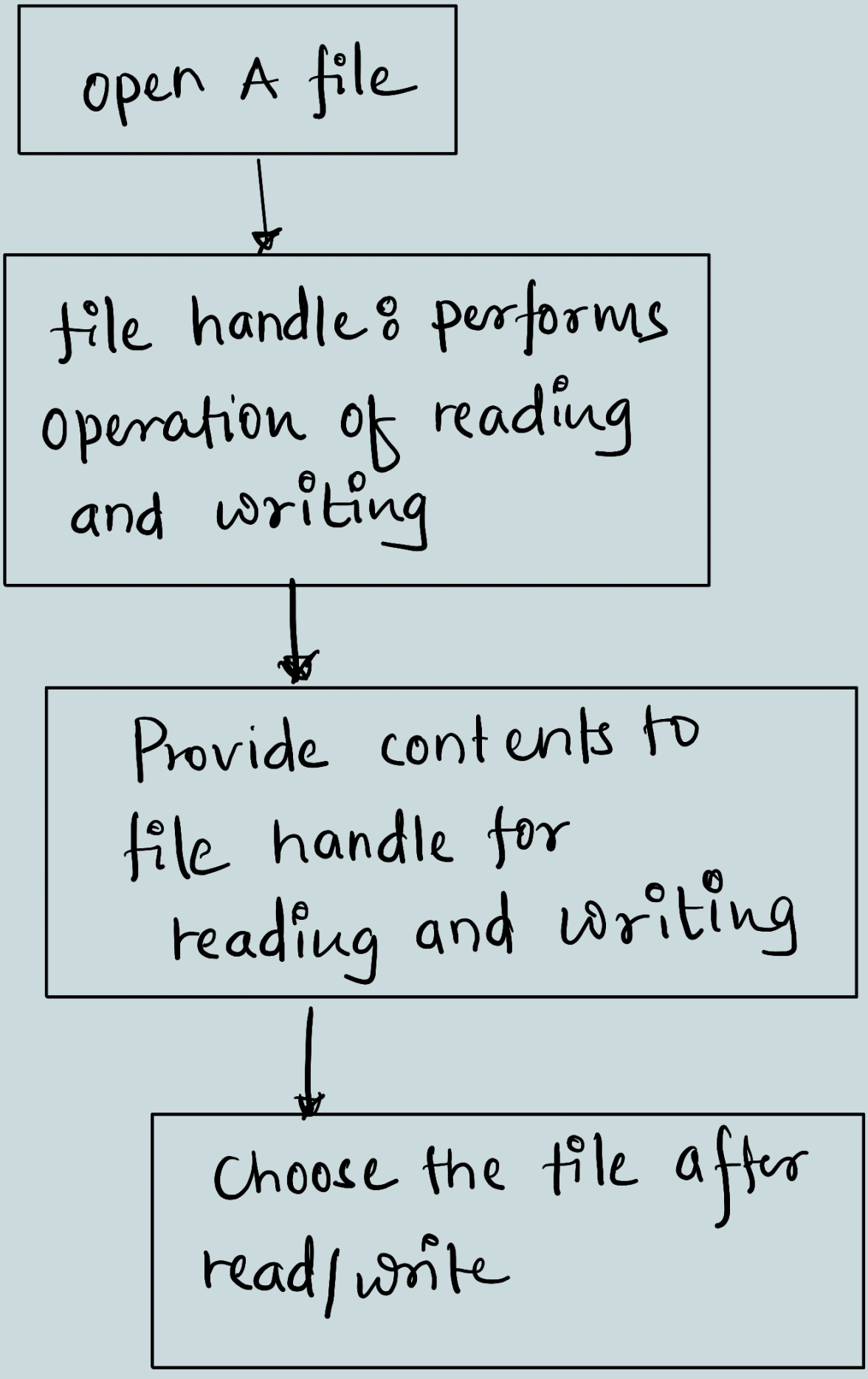
```
Set x "a b c d e"
puts [lrange $x 1 3]
```

output :- b c d.

Lec (10)

file Handling :-




open A file
file handle: performs operation of reading and writing
Provide contents to file handle for reading and writing
Choose the file after read/write

Example:-

```
set fp [open "input.txt" r]
set file_data [read fp]
puts $file_data
close $fp
```

↳ Read entire line at once

```
set fp [open "input.txt" r]
while { [get $fp data] >=0 }.{
    Puts $data
}
close fp
```

writing the content :-

```
set fp [open "input.txt" wt]
puts $fp "test"
close $fp
```