8 Basics Concepts of Programming in TLAC(Think like a computer)

Author:Usmar A. Padow (amigojapan), amigojapan's homepage

Note: This document aims to teach super beginners how to program.

0. Sequentiality
1. Variables
2. Statements
3. Conditionals/blocks
4. Functions/Types
5. Input/output
6. Loops
7. Arrays/lists

0. Sequentiality/Following instructions in order
The first thing we need to learn in order to become a programmer is that instructions always follow a regular order, usually from top to bottom.

tips

for using TLCA use a small coin to keep the current insturction. (assuming you are using the printed version of the board game)

For example:

0001	Function call BrushMyTeeth
0002	Function call CombMyHair
0003	Function call GoToWork

I will never go to or work before I brush my teeth and comb my hair...

0001	Function call BrushMyTeeth
0002	Function call GoToWork
0003	Function call CombMyHair

Well, the answer is that I would first brush my teeth, but I would get to work unkempt, which would cause trouble. That is why the order of the instructions in a program is so important... also because "computers are stupid" (I recommend watching this video) Which is another thing to keep in mind, with the previous program I just wrote, the computer will not intelligently know I should comb my hair before going to work. Computers are just not capable of such decision making at this level of programming. If you make a large program, and shuffle its instructions around, the program will surely not do what it was originally intended to do.

tips

Write variable values down on a seperate piece of paper to remember the current value of the variables (assuming you are using the printed version of the board game).

1.Variables
A variable is a place in the computer memory where some kind of value sometimes a string (some letters),

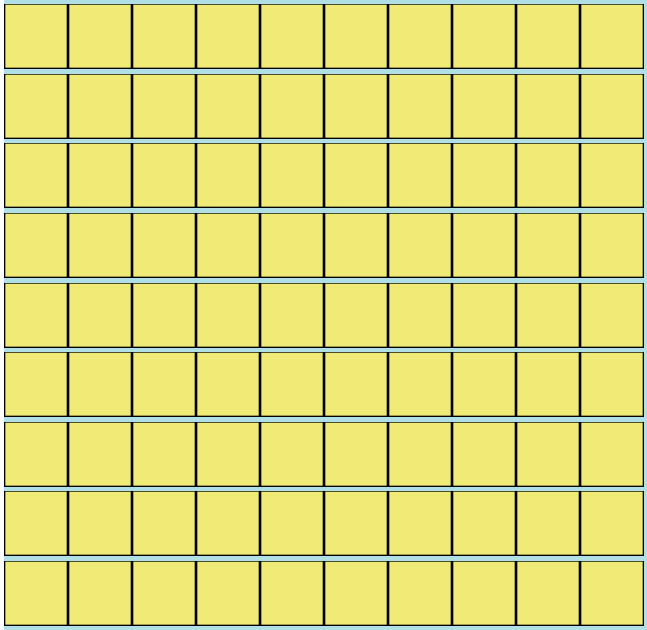
sometimes a number is stored. Like the value 5 or the string "hello”. A variable can have a simple name like just the letter x or any combination of letters like last_name or social_security_number, but not spaces, last name is not a valid variable name. If we have a variable called social_security_number we could assign it the value 123458789 as a sample social security number. Variables as their name implied can vary, which seems to be different to the mathematical view on variables, in math if you say x=5 then x is always equal to 5, but in programming we can say x=3 and then x=5 and the value would first be set to 3 and then change to 5(These are called assignments in programming). In the case of a string, the value between the double quotes will be assigned to the variable, for example if we had last_name we could assign "Einstein” to it easily as follows: last_name= "Einstein”.

2. Statements
this may be the most difficult part to explain of programming, almost everything you see in a computer program is a statement, but can also appear as keywords such as while(). Lets talk about assignment statements. For example, lets take the following program

```
0001 x=3
0002 x=5
0003 x=x+1
```

What will be the result of such a program... Well, the first statement, x=3 will set x to 3, then the next statement x=5 will (maybe counter-intuitively to some) set x to 5, then the final statement x=x+1 we need to break down into its parts. We have the x on the left of the equal sign called an "lvalue” (the variable to the left of the equal sign) which is the same as the first x=3, where the x is what stores the value, then we have x+1, the value of x at this point of the program is 5 because it comes after x=5 so when we do x+1 it is the same as doing 5+1, and then finally we assign that to the lvalue, so at the end of the program, x will contain 6 which is 5+1. Other kinds of statements we will look at later in this text are things like function calls, return statements, conditional statements and boolean statements.

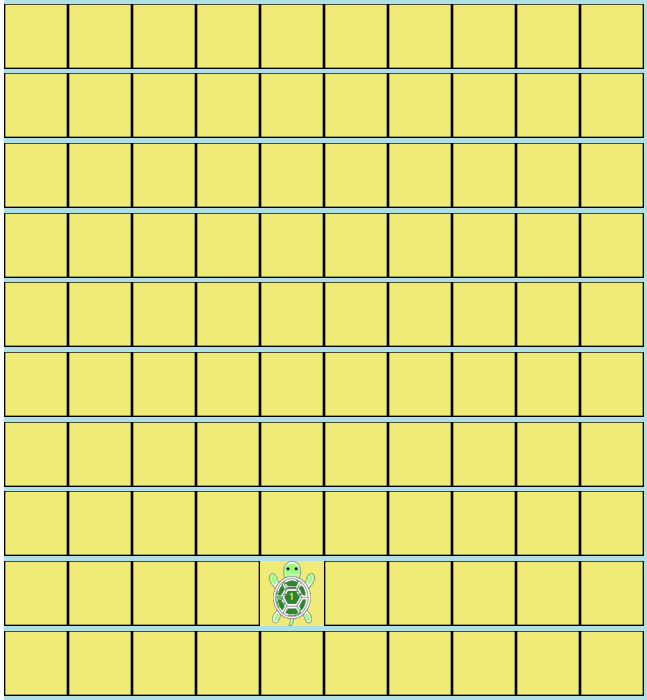
3. Conditionals/block of code
A conditional statement is a statement which give usually two different flow controls depending on wether the statement it is evaluation is true or false, for example: In order to explain conditional statements I am also forced to explain what blocks of code are. Firstly, here is an example in TLAC: take the initial state of the board as:



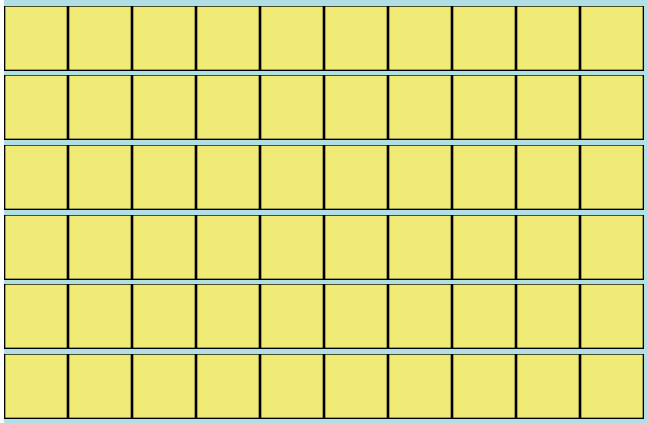
In this example, the turtle is at X=5 as you can see the turtle being on hte fifth slot fromt he leftmost slot on the board. Note: I will be introducing TALC specific statements from now on, when it is a TALC specific insturction I will let you know. TLAC fd() function(will explain functions later, but lets just say this is one for now), this function makes the turle move foward (in the direction the turtle is pointing) by one slot.

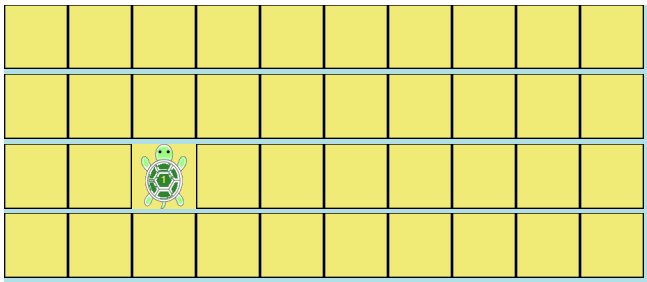
```
0001 green lines like these are called 'comments'
0002 and do not affect the execution of the program
0003 but I want you to read them for now
0004 if X>3 then
    0005 since X is more than 3,
    0006 this block(the block surrounded by the
    0007 pink rectangle) of code will be executed
    0008 Function call Turtle Move Foward
0009 if X<3 then
    0010 since X is not less than,
    0011 this block of code won't be executed
    0012 Function call Turtle Move Foward
0013 X=2
0014 this assignment statement teleports
0015 the turtle to X slot 2(two slots
0016 from the left of the board)
0017 if X<3 then
    0018 now after the last assignment statement
    0019 X is less than 3, so the following block
    0020 of code gets executed.
    0021 Function call Turtle Move Foward
```

Lets go thru that program line by line. First lines 001 to 003 explain what comments are then line 004 checks if X of the turtle is more than 3, since it is,(from now on we will skip all comment lines) it will go into the following "block of code"(marked in pink(in this case)) which starts when you see a "pipe icon" entering our "Function call Turtle Move Foward" on line 008 at this point, you, the person playing the TLAC game, needs to move the turtle one space in the direction of the turtle is pointing twords. (twords where it's head is pointing), and since x>3 is true, the turtle is mored foward by one slot as in the following picture indicates:

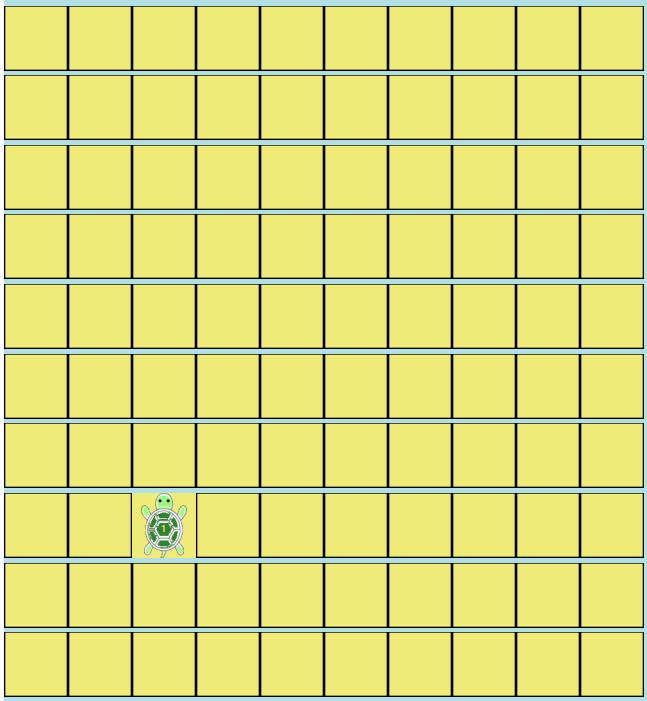


then we reach "if X<3 then" on line 0009 and as the comments say, the following "Function call Turtle Move Foward" on line 0012 will be skipped: so the turtle does not move at all then we "reach X=2" on line 0013, which will change the value of X to two, now I need to explain that values in TLAC are "indexed on 0", which only means "we count from 0"(kind of like hte joke I made at the beginning of this text calling sequenciality "chapter 0"),you should move the turtle to its X position 2 (which is the third slot counting form 0), which means puttin the turtle in the follwoing position:





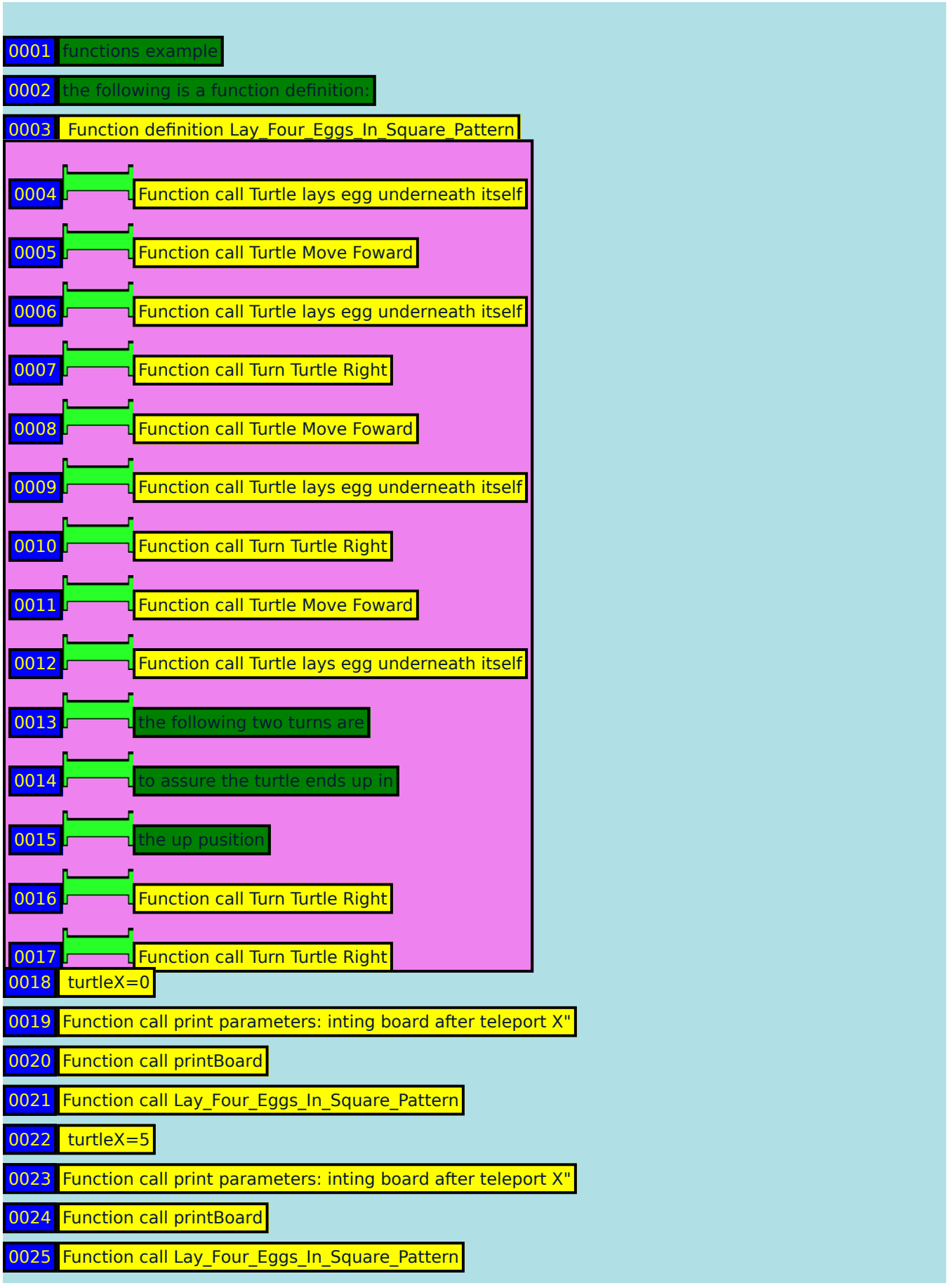
then we reach "if X<3 then" on line 0017 and as the comments say, the following "Function call Turtle Move Foward" on line 0021 will be excecuted: so the turtle neetds to be moved foward one space, making the board as follows:



4. Procedures/functions
Now functions need to be broken down into 2 separate parts, the easy part which is a function call and the hard part which is a function declaration. In the beginning of this paper I used three function calls to say what I was telling the computer to do:

```
0001 Function call BrushMyTeeth
0002 Function call CombMyHair
0003 Function call GoToWork
```

Take the following program:


0001 functions example
0002 the following is a function definition:
0003 Function definition Lay_Four_Eggs_In_Square_Pattern
0004 Function call Turtle lays egg underneath itself
0005 Function call Turtle Move Foward
0006 Function call Turtle lays egg underneath itself
0007 Function call Turn Turtle Right
0008 Function call Turtle Move Foward
0009 Function call Turtle lays egg underneath itself
0010 Function call Turn Turtle Right
0011 Function call Turtle Move Foward
0012 Function call Turtle lays egg underneath itself
0013 the following two turns are
0014 to assure the turtle ends up in
0015 the up pusition
0016 Function call Turn Turtle Right
0017 Function call Turn Turtle Right
0018 turtleX=0
0019 Function call print parameters: inting board after teleport X"
0020 Function call printBoard
0021 Function call Lay_Four_Eggs_In_Square_Pattern
0022 turtleX=5
0023 Function call print parameters: inting board after teleport X"
0024 Function call printBoard
0025 Function call Lay_Four_Eggs_In_Square_Pattern

```
0026 turtleX=8
0027 Function call print parameters: inting board after teleport X"
0028 Function call printBoard
0029 Function call Lay_Four_Eggs_In_Square_Pattern
```

Fist I think i need to introduce you to two more TLAC internal functions: the "Turtle lays egg underneath itself", as the name imlies, you should take an egg and put it inder the turtle. and then the "Turn Turtle Right" function, which means you should take the turtle piece and turn it 90 degrees tot he right. As you can see in the final state of the board in the followinf chart, the turtle lays 4 eggs in a square pattern, it does it 3 times it does it three times because there are three fucntion calls to the "Lay_Four_Eggs_In_Square_Pattern" function, which is not a TLAC function, but it is a user defined function, when you see a call to that function, you should execute all the instructions inside the user defined function. as you can see, this saved a lot of space in the program, because withought this function we woudl have had to repeat all the contents of the code inside the function 3 times over.

Click hereto see are all the states of the board after executing this program:

Function calls help us so we don't have to write the same code many times, for example, in this program we defined the way to make a square of eggs only once, but we did three function calls. The to Lay_Four_Eggs_In_Square_Pattern function call, saving us from typing the same thing over and over, we could do 100 function calls to the same function and it would not increase the size of the program by much... It also increases readibility of the program if we don't repeat the same code over and over.

Now I will explain how a function is defined. First we have the fuction definition, followed by the function name, function names follow the same rule as variable names, in this case our function name is Lay_Four_Eggs_In_Square_Pattern, then they word parameters: or an open parenthesis (then the "parameters", which are the things that change from function call to function call. They can come in the form of variable names or just values, in this case our for example the "Turtle Move Foward" can take a parameter of how far you want the turtle to move in one move, if we have the following "Turtle Move Foward, parameters: 5", then the turtle should jump 5 spaces forward we will see examples using it like this in the future. you can also see the "print parameters: inting board after teleport X", you don't really need to worry about this function call in this example, but we added it cause we actually executed the program to generate the board states of the board and it made it easier to generate the documentation this way, since it adds the title "inting board after teleport X" in the specific place before the "printBoard" function is called. the writer of this document had to add those two functions in order to generate the state of the board properly and decided to leave them in in this example. the reason for the "printBoard" function to be added is because assignments like " turtleX=5" are not TALC internal functions and cannot generate the state of the board automatically like the other functions do, but since the turtle moves with those, the author thought it was a good idea to add it to the printout of the states of hte board. Finally there is something called a "return statement". Which tells the program to "send this back to the function call as the result". (Also note that return terminates the flow of the fucntion, no more operations occur after return, and the flow of the program goes back to the function call) in this program we omitted the return statement, but we will see examples of this later on.

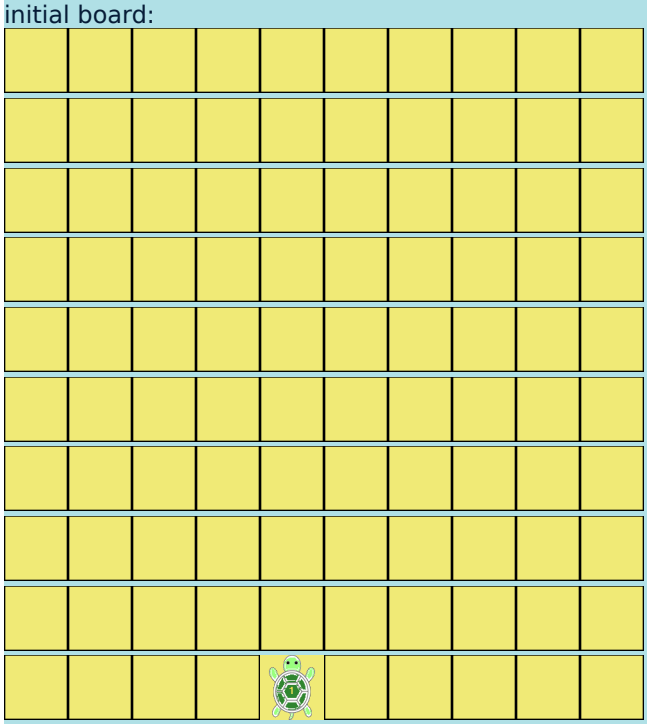
5.Input/output

Now, here is the most simple program every programmer starts out with

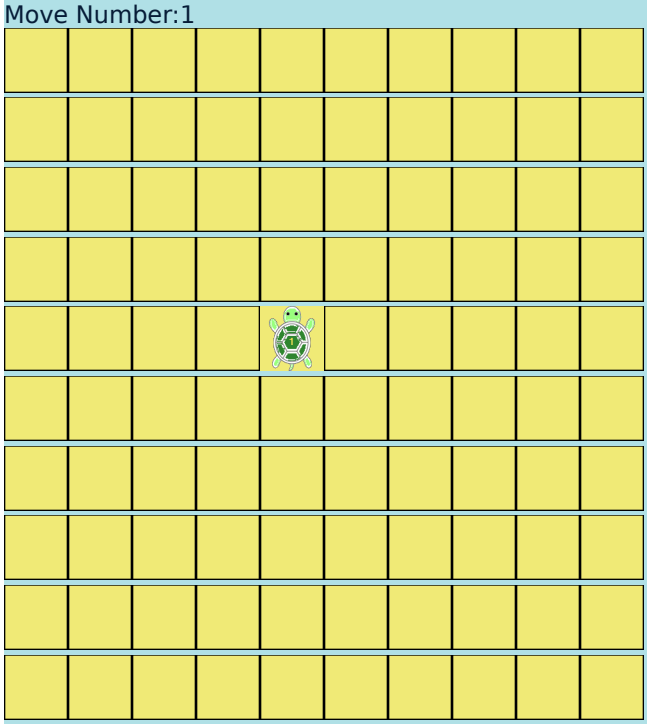
```
0001 very basic program example
0002 Function call Turtle Move Foward parameters: 5
```

This is a simple example of a program that produces "output" the following is the output, in TLAC usualy the output is in the shape of moves on a board:

initial board:



moved foward 5 slots
Move Number:1



Now lets try some input, in TALC doing input is a bit hard to explain, well, mostly cause "input" will usually take the form of an action of the user, this can be achieved with the 'oderUser: parameters "prompt"' function this is quite different from other TALC functions, cause it means we will actually be asking the user to do something.: the following is an example of a program that uses input:

```
0001 user input example
0002
0003 initialize turtle position
0004 Function call teleportTurtleTo parameters: ,4,9
0005 Function call orderUser parameters: ,"put turtle at top left of the board"
0006 Function call simulateUserInputTeleportTurtleTo parameters: ,0,0
0007  savedX=turtleX
0008  savedY=turtleY
0009 Function call orderUser parameters: ,"put turtle at top bottom right of the board"
0010 Function call simulateUserInputTeleportTurtleTo parameters: ,9,9
0011 Function call teleportTurtleTo parameters: , savedX, savedY
```

so lets see what thiis does, first of all empty lines like line 0002 can be ingored. first we teleport the turtle to 4,9, which puts it in the normal initial position. then we order the user to do "put turtle at top left of the board". a user should obey the instructions, but remember that users may not be obedient or they can also make mistakes. next in line 0006 we have a function call used moslt to generate this documentation, it is the same as teleport to 0,0 but this is just used to make the turtle follow the instructions of the user, you can ignore this istruction too. next up we save the turtle x position to savedX by issuing savedX= turtleX then we do the same for the Y position. this should (if the user did hit thing well) with the dariable savedX and savedY being 0,0 (the top left of the board). next we tell the user to move the turtle to the bottom right of the board. you can ignore the insturction at 0010. then we have "Function call teleportTurtleTo parameters: savedX, savedY" and since savedX, savedY are 0,0, the turtle should teleport to the top left of the board. this can get interesting in games where the user gets to decide where to put the turtle. for example if th4e user initially put the turtle in another porition, the turtle would have teleported back there. so it is interesting for making like two player games, where the user interacts wit the pieces.

This is the output after following the instructions.

6. Loops
coundown loop write explanation here

```
0001 countdown example
0002 loopy library import, name it l
0003  import TLACloopyLib as l
0004
0005  Function definition loop_body parameters: n
0006      Function call Turtle Move Foward parameters: n
0007
0008 l.loop_body = loop_body
```

```
0009
0010 Function call l.countdown parameters: 3, 1
```

do while loops explanation

```
0001 loopy library import, name it l
0002 import TLACloopyLib as l
0003
0004 do_while example
0005 Function definition loop_body parameters: n
0006     Function call Turtle Move Foward parameters: n
0007 l.loop_body = loop_body
0008
0009 Function definition condition parameters: n1, n2
0010     Function call return l.conditional parameters: n1, l.op(">"), n2
0011 l.condition = condition
0012
0013 Function call l.do_while parameters: 1, 3, -1
0014 Function call promptUserToWriteDownOnPaper parameters: "n1 exceeds threshold"
```

for loops

```
0001 loopy library import, name it l
0002 import TLACloopyLib as l
0003
0004 for loop example
0005 Function definition loop_body parameters: n
0006     Function call teleportTurtleTo parameters: n,n
0007 l.loop_body = loop_body
0008
0009 Function definition condition parameters: n1, n2
0010     Function call return l.conditional parameters: n1, l.op("=="), n2
0011 l.condition = condition
0012
0013 Function call l.forloop parameters: 5, 9, 1
```

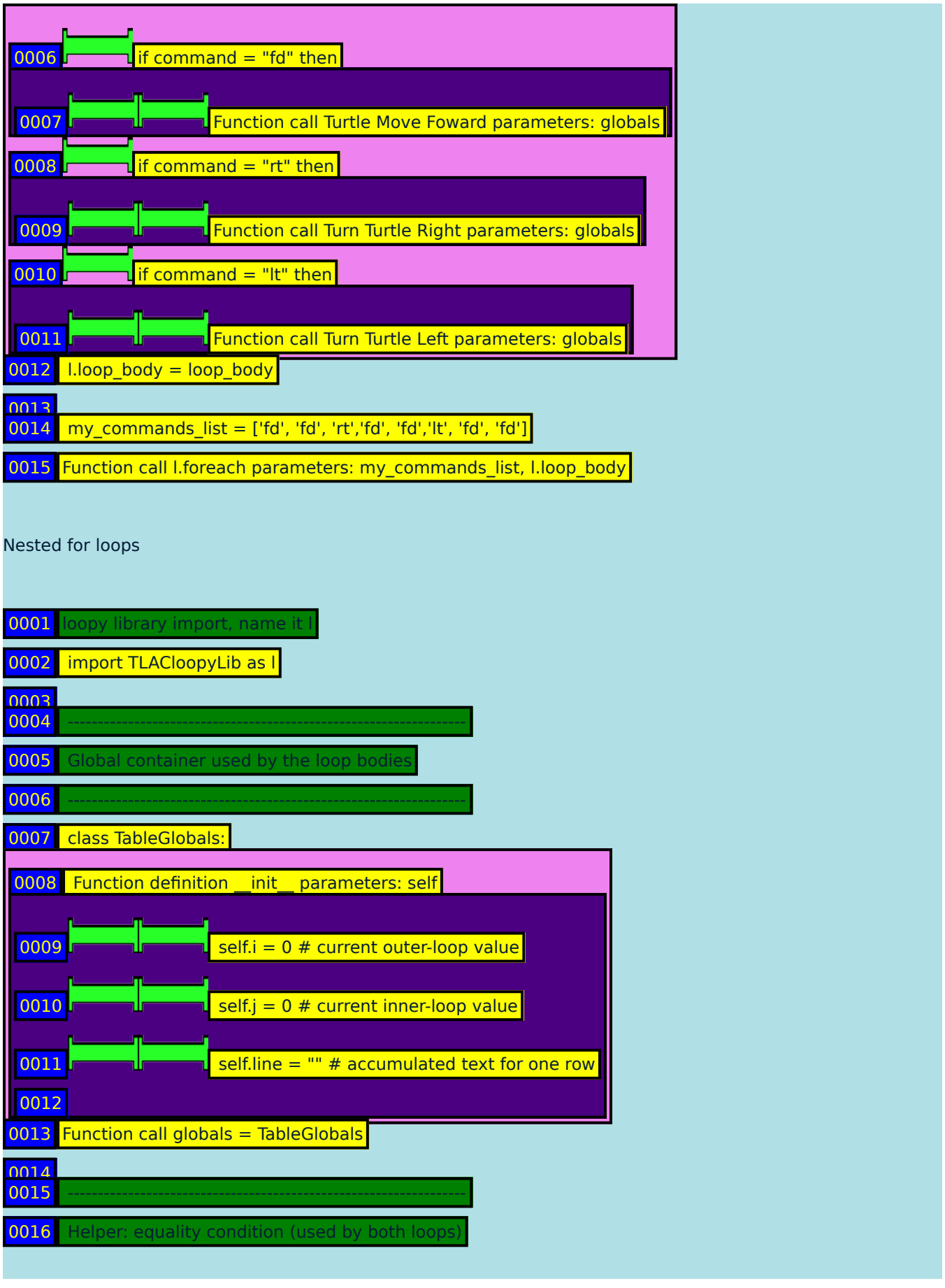
Make a pyramid using for multiple for loops

```
0001 loopy library import, name it l
0002 import TLACloopyLib as l
0003
0004 The following makes a pyramid, it is an example of how to use several loopbodies for a forloop in the code
0005 class initGlobals:
0006  Function definition __init__ parameters: self
0007      self.output = ""
0008      self.ch = "*"
0009
0010 Function call globals = initGlobals
0011
0012 Function definition loop_body1 parameters: n
0013     output = output + ' '
0014
0015 Function definition loop_body2 parameters: n
0016     output = output + ch
0017
0018 Function definition condition_eq parameters: n1, n2
0019     Function call return l.conditional parameters: n1, l.op("=="), n2
0020 l.condition=condition_eq
0021
0022 Function definition print_many_christmas parameters: ch, times, triangle_width
```

```
0023        output = ''
0024        ch = ch
0025        repeat_end = triangle_width - times
0026        Save outer loop state and set inner loop state for spaces
0027        l.loop_body=loop_body1
0028        Function call l.forloop parameters: 0, repeat_end, 1
0029        Set inner loop state for stars
0030        repeat_end2 = times * 2 - 2
0031        l.loop_body=loop_body2
0032        Function call l.forloop parameters: 0, repeat_end2, 1
0033        Function call print parameters: output
0034
0035    Function definition loop_body3 parameters: triangle_width
0036        Function call print_many_christmas parameters: '*', triangle_width, 5
0037
0038    l.condition = condition_eq
0039    l.loop_body = loop_body3
0040    Function call l.forloop parameters: 1, 5, 1
```

7. Arrays lists Lists are a kind of variable that can hold miltiple values, or lists of values. foreach example

```
0001    loopy library import, name it l
0002    import TLACloopyLib as l
0003
0004    foreach example
0005    Function definition loop_body parameters: command
```



Nested for loops

```
0001 loopy library import, name it l
0002 import TLACloopyLib as l
0003
0004 ------------------------------------------------------------
0005 Global container used by the loop bodies
0006 ------------------------------------------------------------
0007 class TableGlobals:
0008  Function definition __init__ parameters: self
0009     self.i = 0 # current outer-loop value
0010     self.j = 0 # current inner-loop value
0011     self.line = "" # accumulated text for one row
0012
0013 Function call globals = TableGlobals
0014
0015 ------------------------------------------------------------
0016 Helper: equality condition (used by both loops)
```

```
0017 ----------------------------------------------------------
0018 Function definition eq parameters: n1, n2
0019     Function call return l.conditional parameters: n1, l.op("=="), n2
0020 l.condition = eq # termination condition
0021
0022 ----------------------------------------------------------
0023 Inner-loop body – compute i*j and append to the current line
0024 ----------------------------------------------------------
0025 Function definition inner_body parameters: j
0026     product = i * j
0027     Function call line += "\t" + str parameters: i) + " * " + str(j) + " = " + str(product
0028
0029 ----------------------------------------------------------
0030 Outer-loop body – run the *inner* loop and then print the line
0031 ----------------------------------------------------------
0032 Function definition outer_body parameters: i
0033     i = i
0034     line = ""
0035
0036     ---- inner loop (j from 1 to 3) ----------------------------
0037     l.loop_body = inner_body # body that fills the line
0038     l.forloop(1, 3, 1) # 1 ... 3 # restore outer state
0039     ----------------------------------------------------------
0040
0041     print(line) # row finished
0042     print() # newline after row
0043
0044 ----------------------------------------------------------
0045 Run the outer loop (i from 1 to 3)
```

```
0046 ------------------------------------------------------------
0047 l.loop_body = outer_body # body that runs inner loop
0048 l.forloop(1, 3, 1) # 1 ... 3
```

-Operations on lists
Here are a few examples of operations you can do on lists:

```
0001 operations on lists example
0002 loopy library import, name it l
0003 import TLACloopyLib as l
0004
0005 Function definition loop_body parameters: item
0006     Function call promptUserToWriteDownOnPaper parameters: item
0007 l.loop_body = loop_body
0008 fruits=["banana","apple","peach","pear"] # create a list
0009 print first element of list
0010 Function call promptUserToWriteDownOnPaper parameters: fruits[0]
0011 print last element
0012 Function call promptUserToWriteDownOnPaper parameters: fruits[3]
0013 Function call promptUserToWriteDownOnPaper parameters: "now re-writing all fruits:"
0014 Function call l.foreach parameters: fruits, l.loop_body
0015 Function call promptUserToWriteDownOnPaper parameters: "now re-writing all fruits in reverse:"
0016 reverses the list
0017 Function call fruits.reverse
0018 Function call l.foreach parameters: fruits, l.loop_body
0019 Function call promptUserToWriteDownOnPaper parameters: "now writing fruits in alphabetical order:"
0020 sorts the list in alphabetical order
0021 Function call fruits.sort
0022 Function call l.foreach parameters: fruits, l.loop_body
```

download fruits.py
The output for this code will look like this:

```
banana
```

```
  pear
  now reprinting all fruits
  banana
  apple
  peach
  pear
  now reprinting all fruits in reverse
  pear
  peach
  apple
  banana
  now printing fruits in alphabetical order
  apple
  banana
  peach
  pear
```

An "index" in programming list is a number that points to the element of the list that we want to retreieve. One thing I want to point out is that fruits[0] points to "banana" cause lists in python are "zero indexed" which means the list starts at 0, not at 1(just like the index of this tutorial). Ok, I think if you grasped this you have grasped the very basics of python, you should now be able to go on to taking the tutorials at the python pages or other sites. Here is the link to the python official tutorial.

This document was originally written in 2017 for python2 and C. Modified in 2022 to be python3 only.