# If-else Statements and Logical Expressions

**Logical Expressions**

* need to code for certain conditions
* conditions can be exact, or wide ranging
* computer will actually match up conditions with code you create
* watch for conditions that are very much alike
* prepare for ANY possible condition, even if you think it’s impossible
* types in coding
  + if/else’s
  + if/else–if’s (cond)
  + logical symbols used to compare

**if/else examples and Setups**

* if/else structure
  + **if/else** structure is called a double-selection structure, because it selects between two different actions.
* **if no “else”**
  + **use symbol ‘none in false portion**

|  |
| --- |
| **if (else) statement syntax** |
| **(if** (condition 1**)** (true statements) (false statements)**)**    (if)  (else)  (define (fib n)  (if (< n 2) n ; true  (+ (fib (- n 1)) (fib (- n 2))) ; false  )  ) |

CODE THIS!!!

* built in false statement!!!
  + MUST BE THERE!!
  + I just put a symbol ‘none

**Condition Statements – if/elseif/else**

* ***But we use a “cond” instead of an if else!!***
  + we can further expand too!!

|  |
| --- |
| **cond-else statement syntax** |
| **(cond**  [ (condition 1)  (true statements)  (true statements) (cond)  (…)  ] (else)  [else  (false statements)  (false statements)  (…)  **]**  **)** |

|  |
| --- |
| **First If-Else (using cond) Example** |
| (define score -1)  (printf "Please enter the grade you received\n" )  (set! score (read))  (cond  [(< score 70) ; highest probability should go first  (printf "THAT'S NOT GOOD\n" )  (printf "YOU FAILED\n" )  ] ; end block this branch  [else ; then passed  (printf "YOU PASSED\n" )  ] ; end of this branch  ) ; end overall conditional block  ; NOTICE ALL OF THE []s!!! BLOCK LIKE STRUCTURE!!! |

Grades Number Line

0 69 100

# ***BLOCK LIKE STRUCTURE!!***

1 line statements = [ (condition) …. statement… )] ; end block

more than 1 line = BLS

[ (condition)

… statement 1…

… statement 2…

… statement 3…

… statement 4…

…

] ; end block

*DO “IF–ELSE IFs” LATER!!!!*

* predicate conditional statement by default!!
  + uses a comparison operator that ***already*** returns #t/#f

|  |
| --- |
| Predicate if/else statement |
| 1 (define (version1 x)  2 [< x 100]  3 )  4  5 (define (main)  6 (print (version1 100))  7 (print (version2 100))  8 )  9  10 (main) |

**Evaluation of Conditional Statements**

* don’t forget the operator goes out front!!
* they themselves solve to #t/#f unless dictated to throw something else
  + do not need to “return” true or false

|  |
| --- |
| conditions evaluate |
| [slupoli@linux1 conditional]$ mzscheme  Welcome to MzScheme v4.1.1 [3m], Copyright (c) 2004-2008 PLT Scheme Inc.  > ( < 5 7)  #t |

**(< score 70)**

(using ***cond***)

   [(>= x 90 ) "A"]

   [(and(>= x 80 )(<= x 90 )) "B"]

**IF/ELSE conditions return a value!!**

* The ***computer*** will evaluate conditions and return true or false
  + you (programmer) do nothing to evaluate
    - You (the programmer) create the criteria/condition!
  + user does nothing to evaluate
* will not always reach each condition depending on structure of if

|  |
| --- |
| **Version #1** |
| **(define** setGrade 75)  **□ (if (**< setGrade 70) (printf “YOU FAILED!!\n”) (‘none) )  **□ (if (>**setGrade 69) (printf “YOU PASSED!!\n”) (‘none) ) |
| **Version #2** |
| **(define** setGrade 75)  **(cond**  **□ [(<** setGrade 70) (printf “YOU FAILED!!\n”) ]  **□ [else (**printf “YOU PASSED!!\n”) ] ; notice no condition  ) |
| **Version #3** |
| **(define** setGrade 55)  **(cond**  **□ [(<** setGrade 70) (printf “YOU FAILED!!\n”) ]  **□ [else (**printf “YOU PASSED!!\n”) ] ; notice no condition  ) |
| **Version #4 (Something you haven’t seen just yet)** |
| **(define** setGrade 75)  **(cond**  **□ [(**setGrade 70) (printf “YOU PASSED!! BARELY!!!\n”) ]  **□ [(**setGrade > 70) (printf “YOU PASSED!!\n”)]  **□ [else (**printf “YOU FAILED!!\n”)]  ) |

*You (the programmer) create the criteria/condition!*

*The computer will decide the outcome!!*

**If-else and Cond statements**

* those statements found within a condition deemed true
* Scheme does this very differently and can bothersome
* types of statements
  + function calls
  + basic values
* spacing is important too in distinguishing true and false portions

|  |  |
| --- | --- |
| Cond Functions (or not) within Statements | |
| Before Clean-up | After clean-up |
| (define (return-example a)  (cond  ((boolean? a) **(**"was a boolean"**)**)  ((rational? a)(/ a 2))  (else **(**#f**)**)  )  )  (display (return-example 11))  (newline)  (display (return-example 12))  (newline)  (display (return-example #t))  (newline)  (display (return-example "Lupoli")) | (define (return-example a)  (cond  ((boolean? a) "was a boolean")  ((rational? a)**(**/ a 2**)**)  (else #f)  )  )  (display (return-example 11))  (newline)  (display (return-example 12))  (newline)  (display (return-example #t))  (newline)  (display (return-example "Lupoli")) |
| \*\*\* ERROR IN return-example, "returning.scm"@9.10 -- Operator is not a PROCEDURE | ; ran perfectly |

|  |
| --- |
| if-else (kinda) functions and statements |
| ; (if (cond) [true] [false])  (define (response x)  (if (boolean? x) "was boolean" "was not boolean")  )  (define (num-response x)  (if (rational? x) (\* x 10) "not rational" )  )  (display (response #t))  (newline)  (display (response 23))  (newline)  (display (num-response 23))  (newline)  (display (num-response #t))  (newline) |
| :: gsi spacing.scm  was boolean  was not boolean  230  not rational |

**=, eq?, eqv?, or equal to compare values**

* predicate functions
* not used for comparing CHARACTERS!!
* “=” should be used when you know for certain you are comparing 2 numbers
  + will compare all number types!!
  + used most
  + will crash if one parameter is not a number
* eqv?/~~eq?~~ 🡪 compares 2 values
  + good for characters, numbers and symbols!!!
  + use eqv, eq has some very minor differences in handling outliers
  + recovers if one is not a number
  + always return #t/#f
* equal?
  + compares complex datatype such as ***Strings***, lists and Vectors

(examples on next page)

|  |
| --- |
| **“=” and eqv? Examples** |
| **=** |
| (= 42 42) => #t  (= 42 #f) **-->ERROR!!!**  (= 42 42.0) => #t  (= 10 x)  (if (= (remainder x 4) 0)  (if (= (remainder x 100) 0)  (if (= (remainder x 400) 0) #t #f)  #t )  #f) |
| eqv? |
| (eqv? 42 42) => #t  (eqv? 42 #f) => #f  (eqv? 42 42.0) => #f  (eqv? 42 score) |
| equal? |
| (cond  [(equal? x "Passed") "S"]  [(equal? x "Failed") "U"]  [(= x 3) "Three!"]  [else "Whatever"]  ) |

Logical Operators/Gates

1/0 or T/F true and false

|  |  |  |
| --- | --- | --- |
| OR- or | | |
| x | y | answer |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

A Good example of an OR would be are your keys in the ignition? (x) and are your headlights on? (y), will the alarm sound? (answer)

|  |  |  |
| --- | --- | --- |
| AND- and | | |
| x | y | answer |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

A Good example of an AND would be did your complete your projects? (x) and did you do well on your tests? (y), will you pass the class? (answer)

|  |  |
| --- | --- |
| **NOT- !** | |
| x | answer |
| 0 | 1 |
| 1 | 0 |

NOT, REVERSES each 0/1 T/F statement

So are these problems TRUE or FALSE??

#1. ( or (8 > 1) (7 > 1)) =

#2. ( or (7 > 8) (8 > 1)) =

#3. ( and (8 > 9) (7 > 1)) =

**#4. ( or (8 > 9) (7 > 8)) =**

**Truth Tables**

* if we design a project with many possible conditions, we need to make sure we cover all of those possible conditions!!
* great way to error check too!!

( cond [ ( (x and y) or (y or z) )

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **x** | **y** | **z** | **x and y =A** | **y or z =B** | **Aor B** |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

1. Determine probably of True or False FINAL answer

***False*** ?? / 8 = ?? % ***True*** ?? / 8 = ?? %

Ex. (B or C) and not A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **!A** | **B or C** | **and !A** |
| 0 | 0 | 0 |  |  |  |
| 0 | 0 | 1 |  |  |  |
| 0 | 1 | 0 |  |  |  |
| 0 | 1 | 1 |  |  |  |
| 1 | 0 | 0 |  |  |  |
| 1 | 0 | 1 |  |  |  |
| 1 | 1 | 0 |  |  |  |
| 1 | 1 | 1 |  |  |  |

Try a few examples

1. (A and B) or C

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **A and B** | **or C** |
| 0 | 0 | 0 |  |  |
| 0 | 0 | 1 |  |  |
| 0 | 1 | 0 |  |  |
| 0 | 1 | 1 |  |  |
| 1 | 0 | 0 |  |  |
| 1 | 0 | 1 |  |  |
| 1 | 1 | 0 |  |  |
| 1 | 1 | 1 |  |  |

2. (A and not(C)) or not(B)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** |  |  |  |  |
| 0 | 0 | 0 |  |  |  |  |
| 0 | 0 | 1 |  |  |  |  |
| 0 | 1 | 0 |  |  |  |  |
| 0 | 1 | 1 |  |  |  |  |
| 1 | 0 | 0 |  |  |  |  |
| 1 | 0 | 1 |  |  |  |  |
| 1 | 1 | 0 |  |  |  |  |
| 1 | 1 | 1 |  |  |  |  |

3. (not(A or B) and C)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **A or B** | **!(A or B)** | **🡨 and C** |
| 0 | 0 | 0 |  |  |  |
| 0 | 0 | 1 |  |  |  |
| 0 | 1 | 0 |  |  |  |
| 0 | 1 | 1 |  |  |  |
| 1 | 0 | 0 |  |  |  |
| 1 | 0 | 1 |  |  |  |
| 1 | 1 | 0 |  |  |  |
| 1 | 1 | 1 |  |  |  |

4. (B and (C or A))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **C or A** | **🡨 and B** |
| 0 | 0 | 0 |  |  |
| 0 | 0 | 1 |  |  |
| 0 | 1 | 0 |  |  |
| 0 | 1 | 1 |  |  |
| 1 | 0 | 0 |  |  |
| 1 | 0 | 1 |  |  |
| 1 | 1 | 0 |  |  |
| 1 | 1 | 1 |  |  |

5. (((C or B) or A) and not(C))

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **!C** | **C or B** | **or A** | **and !C** |
| 0 | 0 | 0 |  |  |  |  |
| 0 | 0 | 1 |  |  |  |  |
| 0 | 1 | 0 |  |  |  |  |
| 0 | 1 | 1 |  |  |  |  |
| 1 | 0 | 0 |  |  |  |  |
| 1 | 0 | 1 |  |  |  |  |
| 1 | 1 | 0 |  |  |  |  |
| 1 | 1 | 1 |  |  |  |  |

**Using Ands, Or, etc.. in conditionals**

* as usual the operator goes out front
  + (and (e1) … (ex))
  + (or (e1) … (ex))

**if-else if further extension of cond-else**

* if-else ifs are used to further down the condition into a more organized fashion

|  |
| --- |
| **3 Rules with an If-else-If structure** |
| 1. The structure will always start with an “cond(” |
| 1. The structure will always END with an “[ else” |
| 1. Everything in between is a [ ] “else-if” |
| 1. Last “[ else” has no condition, since LITERALLY everything else |

|  |  |  |
| --- | --- | --- |
| **if-else example** | **if-else if example** | |
| (cond  [(condition 1)]  [ else ]  ) | **(cond**  **[(**condition 1)  …  ] ; end block  **[ (**condition 2); this else has a couple of conditions  ] ; end block  **[ else**  ] ; end block  ); end entire condition  (define (funName parameter)  (cond  ((if condition) statement)  ((else if condition)(statement))  (else (statement))  )  ) | |
| (if)  ([ else) | (if)  ([ else -----------if)    ([ else) |  |

|  |
| --- |
| **More complex If-else if example** |
| (define getScore -1)  (printf "Please enter a score?\n")  (set! getScore (read))  ( cond  [ (< getScore 70) ; highest probability should go first  (printf "THAT'S NOT GOOD\n")  (printf "YOU FAILED\n")  ] ; end block  [ (and (> getScore 95) (<= getScore 100))  (printf "Teacher's pet!!!\n")  ]  [ (> getScore 90) (printf "Good Job!!!!\n") ] ; end block  [ else (printf "YOU PASSED\n") ]  ) ; end block |

**Mechanics of an If-else is**

* Start from top
* As soon as we find a match, structure is DONE!!

|  |
| --- |
| **Basic Example** |
| ; 100+ - 90 A  ; 89 – 80 B  ; 79 – 70 C  ; 69 – 60 D  ; 59- F  (define grade 75) ; which is a C using our scale  ( cond  [ (< grade 60) (printf “F, You failed\n”) ]  [ (< grade 70) (printf “below average\n”) ]  [ (< grade 80) (printf “average\n”) ] ; stops here!!  [ (< grade 90) (printf “above average\n”) ]  [ else (printf “teacher’s pet\n”) ]  ) |

|  |  |  |
| --- | --- | --- |
| **If-else If Mechanics Example** | | |
| Circle which condition the program will stop on | | |
| Example 1 | Example 2 | Example 3 |
| (define grade 61)  **( cond**  **[ (**< grade 60) (printf “F, You failed\n”) ]  **[ (**< grade 70) (printf “below average\n”) ]  **[ (**< grade 80) (printf “average\n”) ]  **[ (**< grade 90) (printf “above average\n”) ]  **[ else** (printf “teacher’s pet\n”) ]  ) | (define grade 49)  **( cond**  **[ (**< grade 60) (printf “F, You failed\n”) ]  **[ (**< grade 70) (printf “below average\n”) ]  **[ (**< grade 80) (printf “average\n”) ]  **[ (**< grade 90) (printf “above average\n”) ]  **[ else** (printf “teacher’s pet\n”) ]  ) | (define grade 91)  **( cond**  **[ (**< grade 60) (printf “F, You failed\n”) ]  **[ (**< grade 70) (printf “below average\n”) ]  **[ (**< grade 80) (printf “average\n”) ]  **[ (**< grade 90) (printf “above average\n”) ]  **[ else** (printf “teacher’s pet\n”) ]  ) |

Determining structure and outcomes

* terms
  + structures
    - structures start with an if, SOMETIMES end with an [ else
    - easy way, count the ifs’
  + possible outcomes
    - what grades COULD you get from this class?
  + Outcomes
    - How MANY grades will you get in this class?

Comparing and Building Strings (built-in functions)

(make-string n c)

It returns a string consisting of n of characters c. The character c can be omitted.

(string-length s)

It returns the length of a string s.

(string=? s1 s2)

It returns #t if strings s1 and s2 are the same.

(string-ref s idx)

It returns the idx-th character (counting from 0) of a string s.

(string-set! s idx c)

It sets the idx-th character of a string s to c.

(substring s start end)

It returns a substring of s consisting of characters from start to (end-1).   
(substring "abcdefg" 1 4) ⇒ "bcd"

(string-append s1 s2 ...)

It connects strings s1, s2 ....

(string->list s)

It converts a string s to a list of characters.

(list->string ls)

It converts a list of characters (ls) to a string.

(string-copy s)

It copies a string s.

|  |
| --- |
| **Example 1** |
| ; Name: example  ; This function determines if a string matches a condition  ; This function will display the result internally  ; input: Single String  ; output: None  (define (example x)  (if [string=? x “A”] (printf “matched”)  (printf “did not match”)  )  ; (if [string=? X “A”] (“matched”)  ; (“did not match”)  ; )  ) |

|  |
| --- |
| **Example 2** |
| ; Name: example 2  ; This function determines if a string matches a condition  ; input: Single String  ; output: String  (define (example2 x)  (if [string=? x “A”] (string-append “” “matched”)  (string-append “” “did not match”)  )  ) |

|  |
| --- |
| **Example 3** |
| ; Name: example 3  ; This function determines if a string matches a condition  ; input: Single String  ; output: Boolean  (define (example3 x)  (if [string=? x “A”] #t #f)  ; (if [string=? x “A”] (#t) (#f))  ) |

|  |
| --- |
| **Example 4** |
| ; Name: example 4  ; This function determines if a string matches a condition  ; input: Single String  ; output: Boolean/String  (define (example4 x)  (if [string=? x “A”] #t  (string-append “” “did not match”)  )  ) |

­­­­­­

**Multiple Conditions**

* conditions in if/else-cond statements don’t exactly follow spoken English
* Common errors popped up
* each condition requires ***2*** values!!
  + x < y
  + 234 > value
  + value1 <= value2

|  |  |
| --- | --- |
| **Common Condition Errors** | |
| Incorrect | Correct |
| **(** value1 > value2 > value3) | **(** and (> value1 value2) (> value1 value3)) |
| **(** value1 < value2 > value3) | **( and** (< value1 value2) (> value2 value3)) |
| **(**value1 > value2 > value3 > value4) | **( and** (value1 value2) (> value2 value3) (> value3 value4))  ; yes, one and since all needed ands!! |
| **(** value1 > value2 > value3) or (value3 < value2) | ???? |

**Switch (but case in Scheme) Statements**

* The **case** structure is called a multiple-selection structure, because it selects one of many possible actions. (to be discussed later in semester)
* USE FOR EXACT VALUES ONLY!!!

|  |  |
| --- | --- |
| **When to use what** | |
| **If/else** | **Switch/case** |
| (if (set< grade 90) | case #\A’ |
| (if (and (score > 90) (= attendance 0)) | case 0 |

* Used for ***Menus***
* ALSO USED TO BE USER (NOT PROGRAMMER) FRIENDLY!!!
* When to use
  + if-else  1-3 actions
  + case( )  3 or more actions

Expression

(variable)  char or (define value **(ONLY!!! NOT TO BE USED WITH STRINGS!!!)**

**=, eq?, eqv?, or equal to compare values**

* covered in Variables too
* predicate functions
* “=” should be used when you know for certain you are comparing 2 numbers
  + will compare all number types!!
  + used most
  + will crash if one parameter is not a number
* eqv?/~~eq?~~ 🡪 compares 2 values
  + good for characters, numbers and symbols!!!
  + use eqv, eq has some very minor differences in handling outliers
  + recovers if one is not a number
  + always return #t/#f
* equal?
  + compares complex datatype such as ***Strings***, lists and Vectors
  + we go over again in Strings

|  |  |
| --- | --- |
| **“=” and eqv? Examples** | |
| **=** | **eqv?** |
| (= 42 42) => #t  (= 42 #f) **-->ERROR!!!**  (= 42 42.0) => #t  (= 10 x) | (eqv? 42 42) => #t  (eqv? 42 #f) => #f  (eqv? 42 42.0) => #f  (eqv? 42 score) |

**Final word on () ; end block’s**

* They need to line up!!!
* I will be particular in grading!!

|  |
| --- |
| **() ; end block Setup Example** |
| stdio.h>  (define main()  (  ( cond [ ( i < 7)  [  (printf “bad Score\n”)  ]  [( ( i > 9)  [  (printf “good Score\n”)  ]  [ else  [  (printf “ very good\n”)  ]  ( cond [ ( i < 3) (printf “wow you stink!!\n”) ]  char choice  (printf “Press X or Q for Exit\n”)  (printf “Press L to turn Left\n”)  (printf “Press R to turn Right\n”)  (read)“%d”, &choice)  switch(choice)  [  ;case #\3 when coding for a NUMERIC choice  case #\L whatever code you need here ; will take you out of case  case #\R whatever code you need here ; will take you out of case  case #\Q ; no , case will continue  case #\X (printf “Exiting Robot Program\n”) ; will take you out of case  default (printf “Invalid\n”) ; if entry was anything but #\L’, #\R’, or #\X’ this is what  ]  return 0  ] |

**FYI – Section**

**Cases**

|  |
| --- |
| **Case Example** |
| (case month  ((sep apr jun nov) 30)  ((feb) 28)  (else 31)  ) |

FIX caseExample.scm**Cases**

* Handle each action, can handle more than one

|  |
| --- |
| **One line case statement** |
| case #\I setCount++ ; for a char choice  case 0 setCount 0 ; for a (define choice |
| **Two or more line case statement** |
| case #\D  (printf “Displaying answer\n”)  total + 1  (printf “ %d\n”, total)  ; should ALWAYS HAVE THIS A AT END OF CASE!!! |
| **Two or more CASES for the same outcome** |
| case #\Q  case #\X (printf “Exiting Robot Program\n”)  ; As there was no discontinuation both X or Q will display “Exiting…” |

|  |
| --- |
| **Overall Switch/Case structure and example** |
| ~~(define main()~~  ~~(~~  ~~(define choice ‘z’)~~  ~~(printf “Press X or Q for Exit\n”)~~  ~~(printf “Press L to turn Left\n”)~~  ~~(printf “Press R to turn Right\n”)~~  ~~(read)“%d”, &choice)~~  ~~switch(choice)~~  ~~(~~  ~~;case #\3 when coding for a NUMERIC choice~~  ~~case #\L whatever code you need here ; will take you out of case~~  ~~case #\R whatever code you need here ; will take you out of case~~  ~~case #\Q ; no , case will continue~~  ~~case #\X (printf “Exiting Robot Program\n”) ; will take you out of case~~  ~~default (printf “Invalid\n”) ; if entry was anything but #\L’, #\R’, or #\X’ this is what~~  ~~]~~  ~~return 0~~  ] |

**Final Word on if-else efficiency**

* In the examples I have above, notice no redundancy
  + no duplicate variables (answer1, answer2, …)
  + no duplicate displays (all were different in minor ways)

|  |  |
| --- | --- |
| **Building Efficient If-else Code** | |
| **inefficient if-else** | **inefficient if-else** |
| ( cond [ (hours<=40)  (  (define salary1=8\*hours  (printf "salary is %lf dollars\n", salary1)  ]  [ else  (  (define salary2=320+12\*(hours-40)  (printf "salary is %lf dollars\n", salary2)  ] | (define salary 0  ( cond [ (hours<=40)  (  salary =8\*hours  ]  [ else  (  salary=320+12\*(hours-40)  ]  (printf "salary is %f dollars\n", salary1) |

; Why do both so the same work, but one is shorter in length??

**Errors that drove me to insanity:**

|  |
| --- |
| Error Message #1 |
| procedure application: expected procedure, given: 10 (no arguments) |
| Fix:  the |

I thought this meant the function I was in!!! It was the if statement I had incorrectly coded.

1. (if ( < aVar bVar) (aVar) (bVar))

All I wanted to do was to return aVar or bVar. While the general syntax was correct, the t/f statements inside an if/else require more.

|  |
| --- |
| Error Message #2 |
| default-load-handler: expected a `module' declaration for `ifElseIfExample', found: something else in: #<path:/afs/umbc.edu/users |
| Fix:  Forgot the “–r” when running in command line!! |

**Wait, now it fails!!!**

At runtime are type mismatches are detected!!!! So minimal code needed for datatypes. Behind the scenes it is only breaking things into tokens, THEN when interpreting does it do type check

if(zero? 0) ( 4) ( ( (+ “Lupoli” 3.14159 ‘(1 2) ) ; the + will fail at the string

**but if it is CONSTANTLY true, it never checks the fail portion!! That could be wrong!!! A compiler would check the entire thing first before running.**

**You should always check each condition!!!**