“p1.eightio”

```
func doMax: Integer (var1: Integer, var2: Integer, var3: Integer):
(
   ## this is NOT optimized, but rather a proof of reference
   max: Integer is 0;
   if (var1 > max):
   (
      max is var1;
   )
   elif (var2 > max):
   (
      max is var2;
   )
   else:
   (
      if (var3 > max):
      (
         max is var2;
      )
   )
   # return max would also work in this scenario, but you would be able to return multiple variables
simultaneously, this is what I wanted to prove
   return (max);
)

func checkPrime: Boolean(varToCheck: Integer):
(
   isPrime: Boolean is True;
   divisionChecker: Integer is 2;
   ## this is NOT optimized, but rather a proof of reference
   while(isPrime == True and divisionChecker <= varToCheck/2):
   (
      if(varToCheck % divisionChecker == 0):
      (
         isPrime is False;
      )
      divisionChecker is divisionChecker + 1;
   )
   return isPrime;
)


## Equivalent to 'void main()'
func code: Null():
```

```
(
   ## This is a commentary

   ## These are valid declarations
   var1, Var1, vAr_var, sdad_: Integer;
   x: Integer is 5;
   y, z: Integer is 2, 3;

   ## These are NOT valid declarations
   ## 1var: Integer;
   ## x: Integer is 'text';

   max_of_x_y_and_z: Integer;
   max_of_x_y_and_z is doMax(x, y, z);
)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

“p1err.eightio”
func checkPrime: Boolean(varToCheck: Integer):
(
   isPrime: Boolean is True;
   divisionChecker: Integer is 2;
   ## this is NOT optimized, but rather a proof of reference
   while(isPrime == True and divisionChecker <= varToCheck/2):
   (
      if(varToCheck % divisionChecker == 0):
      (
         isPrime is False;
      )
      divisionChecker is divisionChecker + 1;
   )
   return isPrime;
)


## Equivalent to 'void main()'
func code: Null():
(
   ## Would throw error
   1var: Integer;

   var1: Integer is 20;
   check: Boolean;
```

```
    ## Would throw error
    Boolean is checkprime(var1);
)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

func doGCD: Integer(var1: Integer, var2: Integer):
(
    ## I think every if, while, do while or for statement will require the use of (
    #IntegerMax will be a already defined function
    gcdHolder: Integer is IntegerMAX();
    gcdIsFound: Boolean is False;
    if (var1 < gcdHolder):
    (
        gcdHolder is var1;
    )
    if (var2 < gcdHolder):
    (
        gcdHolder is var2;
    )
    while (gcdHolder > 0 and gcdIsFound == False);
    (
        if (var1 % gcdHolder == 0 and var2 % gcdHolder == 0):
        (
            gcdIsFound is True;
        )
        else:
        (
            result--;
        )
    )
    return gcdHolder;
)

func doSQRTInteger: Integer(var: Float):
(
    sqrt: Integer is 1;
    while (sqrt * sqrt <= var):
    (
        sqrt is sqrt + 1;
    )
    return sqrt;
)

func do2ndOrderEquation: Float (var1: Float, var2: Float, var3: Float, var4: Float):
```

```
(
   ## This function solves the result of the equation var1*x*x + var2*x + var3 = var4;
   var3 is var3 - var4;
   delta: Float;
   delta is var2 * var2 - 4 * var1 * var3;
   sqrtDelta: Integer is doSQRTInteger(delta);
   secondOrderEquationPlus = ((-var2 + sqrtDelta) / (2 * var1));
   secondOrderEquationMinus = ((-var2 - sqrtDelta) / (2 * var1));
   return (secondOrderEquationPlus, secondOrderEquationMinus);
)

func code: Null():
(
   var1, var2: Integer;
   var1 is 100;
   var2 is 30;

   gcd: Integer is gcdHolder(var1, var2);

   var3, var4, var5, var6: Float;
   var3 is 5,4;
   var4 is 2,6;
   var5 is 7;
   var6 is 0;

   secondOrderEquationPlus, secondOrderEquationMinus: Float;
   (secondOrderEquationPlus, secondOrderEquationMinus) is do2ndOrderEquation(var3, var4, var5, var6);
)

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

func doSumOfNumbers: Float(list: List[Float]):
(
   ## len(List) will be a function that returns the nr of elements in a list
   sum: Float is 0;
   index: Integer is 1;
   for(index is 1; index <= len(list); index is index + 1):
   (
      sum is sum + list[index];
   )
   return sum;
)

func doMaxOfList: Float(list: List[Float]):
(
```

```
    max: Float is FloatMin();
    index: Integer is 1;
    for(index is 1; index <= len(list); index is index + 1):
    (
        if(max < list[index]):
        (
            max is list[index];
        )
    )
    return max;
)

func program: Null():
(
    list: List[Float] is [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    sum: Float is doSumOfNumbers(list);

    max: Float is doMaxOfList(list);
)
```