"Who looks outside, dreams; who looks inside, awakens."

Jung

Fundamentals
- We build up instructions from three types of materials
  - Constants
  - Variables
  - Expressions

Fundamentals
- Constants are just that, they are values that don’t change as our macros are executing

Fundamentals
- Variables are names assigned to locations in computer memory where we want to store things that may change as the macro executes
  - Think of them as named mailboxes
Fundamentals

- Expressions or statements are the instructions that tell the computer or EXCEL just what to do.

Fundamentals

- As a rule in this class, you will always start your code with your name, the class, the date, and the assignment number.
- For example:

![Code example]

Fundamentals – Variables

- There really aren’t that many kinds of things that we deal with so keeping them straight shouldn’t be much of a problem.
- If you get confused, check the help menu for data types.

Fundamentals – Variables

- **Boolean**
  - This is a data type that is used to store LOGICAL values.
  - True or False
  - Nothing else.
**Fundamentals – Variables**

- **Integer**
  - This one is important and used lots
  - An integer number is just a whole number
  - No decimal places
  - Notice the range of values, if you go beyond that range, you have to use another type
  - -32,768 to 32,767

- **Long**
  - If you need an integer to go beyond the range provided by the INTEGER type, you can use the LONG type
  - This is still an integer but it has a range of + or – over 2 Billion

- **Single**
  - This is the first of the real number types
  - It includes decimal fractions and will store ranges from +/− 3.4 x 10^38 down to +/- 1.2 x 10^-45
Fundamentals – Variables

- **Double**
  - If you need more precision than what is provided by the SINGLE type, you can use the DOUBLE type.
  - It ranges out from $10^{-308}$ to $10^{308}$.

- **String**
  - The only other data type we are going to worry about right now is the STRING data type.
  - This is used to hold strings made up of letters, blanks, numbers, and symbols.

Fundamentals – Constants

- Constants come in each type.
- A number or string that won’t change in the program is a constant.

```
Dim Radius As Single
Dim Height As Single
Dim Volume As Single

Volume = (3.14159 * Radius ^ 2) * Height
```

Fundamentals – Variables

- The syntax of the statement is:
  - `Dim variablename as type`
  - `variablename` is the name of the location where you want to store the information.

```
Dim Radius As Single
Dim Height As Single
Dim Volume As Single

Volume = (3.14159 * Radius ^ 2) * Height
```
Fundamentals – Variables

- The syntax of the statement is
  - Dim variablename as type
- type is the type of information that you want to store in the named location

So the statement
- Dim Radius As Single
- Sets us a storage location named Radius to hold a single precision real number

VBA allows us to dimension (type) multiple variables on the same line but you have to be careful here

Each variable name must be typed independently and separated by a comma from the other types
Fundamentals – Variables

- Three common mistakes
  - Dim Radius, Height, Volume as Single
  - Dim Radius as Single, Dim Height as Single
  - Dim as Single, Radius, Height, Volume

Every cell (as well as groups of cells, charts, and other things) is actually an object in EXCEL and so can be treated as an object in VBA

In this case, we are using what is known as a Range object

We are selecting the cells from A1 to F17 inclusive as our object

We do the selection by using the Select Method

Methods are just special ways of handling objects and their properties
Fundamentals – Variables

- We are able to set up variables to hold objects by typing the variable according to what type of object the variable will hold.
- For example to set up a range object we would use the specification statement.
- Dim WhatCells as Range

Fundamentals – Assignment statements

- An assignment statement reads
  - Place what the right side of the expression evaluates to into the location on the left side of the expression.
  - The left side and right side are separated by an equals sign (=).
  - Note that the right side is evaluated and then placed into the left side.
  - The left side is just a location.

For example the assignment statement

- X = 1.

Would evaluate the right side

- A constant evaluates to itself.

Then since there is nothing else to evaluate it would store the result in the memory location labeled X.

Since we have an evaluation, then a replacement, a statement like

- X = X + 1.

Is completely proper and meaningful.

The computer looks at the right side first

- It gets the value stored in X.
- Adds 1 to it.
- Store the result in X.
Fundamentals – Assignment statements

- If X had 21 originally stored in it and the expression
  - \( X = X + 1 \)
- Was executed, after the execution of the expression, X would contain 22
- Since X is a variable, it can change values during the execution of the program

Fundamentals – Order of Operations

- For ARITHMETIC evaluations, VBA has a number of operators
  - (uniary minus)
  - ^ exponentiation
  - * and / multiply and divide
  - \ integer division
  - Mod modulus function
  - + and – add and subtract

Fundamentals – Order of Operations

- All of the operators beside the first take two arguments
- The unary minus only take one argument
  - - (uniary minus)
  - ^ exponentiation
  - * and / multiply and divide
  - \ integer division
  - Mod modulus function
  - + and – add and subtract

Fundamentals – Order of Operations

- The order in which arithmetic expression are evaluated are
  - - (uniary minus)
  - ^ exponentiation
  - * and / multiply and divide
  - \ integer division
  - Mod modulus function
  - + and – add and subtract
Fundamentals – Order of Operations

- For operators of equal precedence, they are evaluated in left to right order

We have a way to circumvent the normal arithmetic order of operations is we choose to do so.
- Anything that we enclose in parentheses will be evaluated first.
- This order goes from inside to out.

Parentheses allow us to control the order of evaluation of an arithmetic expression.
- Parentheses are evaluated from the inside out when they are nested.
- For every open parenthesis ( there must be a closed parenthesis )

Building a macro

```vba
Public Sub TryThisProcedure()
    Dim tempVal As Single
    tempVal = Range("C11").Value
    tempVal = tempVal / 2 + 5.5
    Range("G11").Value = tempVal
End Sub
```
The Next Step

- For example, if we wanted to look at what was in a cell and decide how to treat it based on if it was even or odd (integers only)
  - If the number is even, divide it by 2 and display the result in red
  - If the number is odd, add 1 to it then divide by 2 and display the result in blue

Making a Decision

We need a way to make a decision based on what the value we just stored in checkVar

```vba
Public Sub EvenOdd()
    Dim checkVar As Integer
    ActiveCell.Select
    checkVar = Selection.Value
End Sub
```

Making a Decision

Logical expression evaluation to either true or false, not to a number. This is critical to know the difference because Visual Basic sometimes tries to make up for your mistakes with tragic results.

```vba
Public Sub EventOdd()
    Dim checkVar As Integer
    ActiveCell.Select
    checkVar = Selection.Value
End Sub
```

There are actually two types of operators that are used in logical expressions:

- Relational Operators
- Logical Operators
Making a Decision

Relational operators are based on the relationship of one argument to another:
- Is \( X \) larger than \( Y \)?
- Is \( 1.2 \) not equal to \( Z \)?

In both cases, we could answer true or false to the proposed relation.
If \( X = 1 \), \( Y = 3 \), and \( Z = 5 \) then:
- Is \( X \) larger than \( Y \) → False
- Is \( 1.2 \) not equal to \( Z \) → True

While we can store the evaluation of a relational expression into a Boolean variable, what we usually do is use the results of the evaluation of the expression to control execution of the program.

```
Public Sub EvenOdd()
    Dim checkVar As Integer
    activeCell.Select
    checkVar = Selection.Value
End Sub
```
Making A Decision

The diagram is known as a flow chart and represents what action the program takes as it "flows" through execution.

You can think of it as moving along one way streets, from top to bottom, as the program executes.

In this case we get to the decision, based on the decision we can either go down the right or left path, and then eventually they converge and we go on.

In our case, the logical condition will be the evaluation of a relational expression.

Relation expressions are made up of two arguments connected by a relational operator very similar to arithmetic expressions which have an arithmetic operator and two arguments.

The relational operators are comparison operators and always evaluate to a logical value (TRUE or FALSE).

The relational operators are:

- `=` equal to
- `<>` Unequal to
- `<` Less than
- `>` Greater than
- `<=` Less than or equal to
- `>=` Greater than or equal to
Making A Decision

Here is where confusion can creep in. Now we are using the equals sign as a relational operator which is comparing the value of two arguments.

- Equal to
- Not equal to
- Less than
- Greater than
- Less than or equal to
- Greater than or equal to

Previously, we utilized it as an assignment operation to replace the left side with the evaluation of the right side. Here we use it to give the logical value based on the equality of its arguments.

Since we almost always use these relational operators within the context of process control, the context will determine which equal sign meaning we have. It isn't as bad as it seems if you are careful.

In our problem, we need to decide if a number is even or not. An even number is evenly divisible by 2 so we need to incorporate that into our programming. We can do this by taking advantage of integer arithmetic.
Remember that integers have no decimal part and that we actually set up places in memory, variables, to specifically hold integers.

If we took 3. and divided it by 2 we would get 1.5.

If we then tried to store the 1.5 into an integer, we would store a result that we are not sure of.

If we run this code, we get 2 in our selected cell.

Luckily, Visual Basic has a function which allows us to divide two integers by each other and gives us the remainder.

The function is know as MOD and takes two arguments.

Notice that we use it in the same way we used binary arithmetic operators.

Argument Operator Argument

Also, like other arithmetic operators, it evaluates to a number.

In this case, it is the remainder from the division of one integer divided by another.

You may have noticed that one of the arguments isn’t an integer.

Visual Basic made it one for this expression.

Argument Operator Argument

If you look for the Mod operator in the help section you will get the result shown here so we will know how to use the operator in our expression.

Any even number divided by 2 will have 0 as a remainder so all we have to test is what the Mod operator returns from our number and 2.

checkVar Mod 2
Making A Decision

No we can begin to put all of this together
First we need to put in the decision structure
IF-THEN-ENDIF

\[ \text{checkVar} \mod 2 \]

What we have done is to build the paths we saw earlier
We start with if then an expression that evaluates to a logical value (True or False)
We then provide instructions to execute if the logical expression evaluates to true and another set of instruction to evaluate if the logical expression is false
It will always be one or the other, never both

The form of the statement is important
I indent to know just where I am and to show that I know that there are two branches
Now we can add the code to do just what we originally set out to do.

In this code we have done a number of things
• We check to see if the value we captured from the cell is evenly divisible by 2
• We make a calculation based on that check and we change the property of the selected cell
• We display the results of our calculations in the cell
Homework

- The current calendar, call the Gregorian calendar, was introduced in 1582. Every year divisible by 4 was declared to be a leap year, with the exception of years ending in 00 and not divisible by 400. Write a macro that gets the year in four digits from a cell and in the adjacent cell (to the right) writes the number of days in that year.
DUE WED 26th January 2005