COMP1022Q
Introduction to Computing with Excel VBA

Recursion

David Rossiter and Gibson Lam

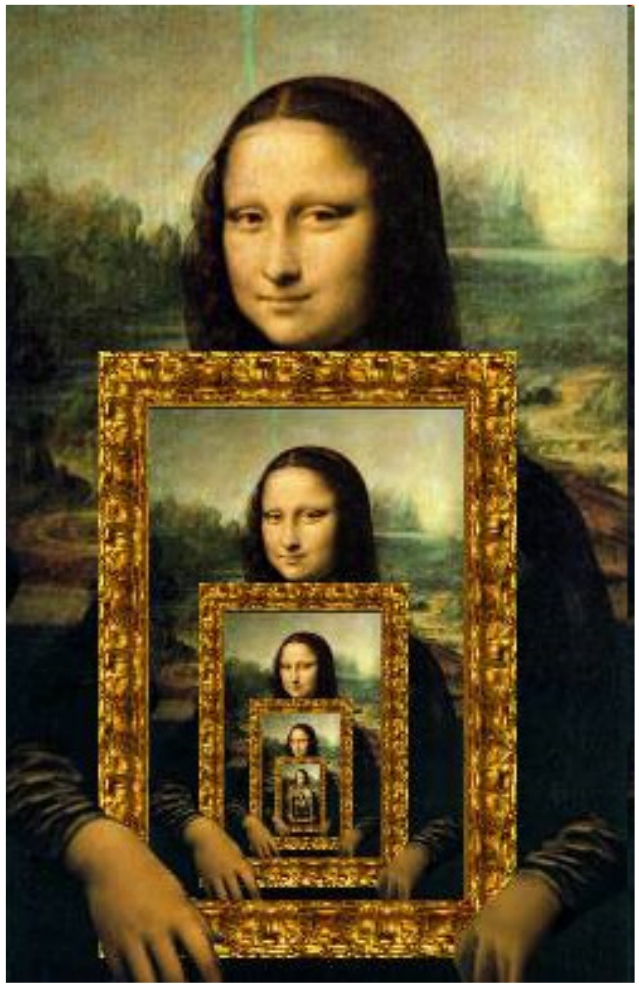
Outcomes

- After completing this presentation, you are expected to be able to:
  1. Explain how recursion works
  2. Describe the result of running any given recursive function/subroutine

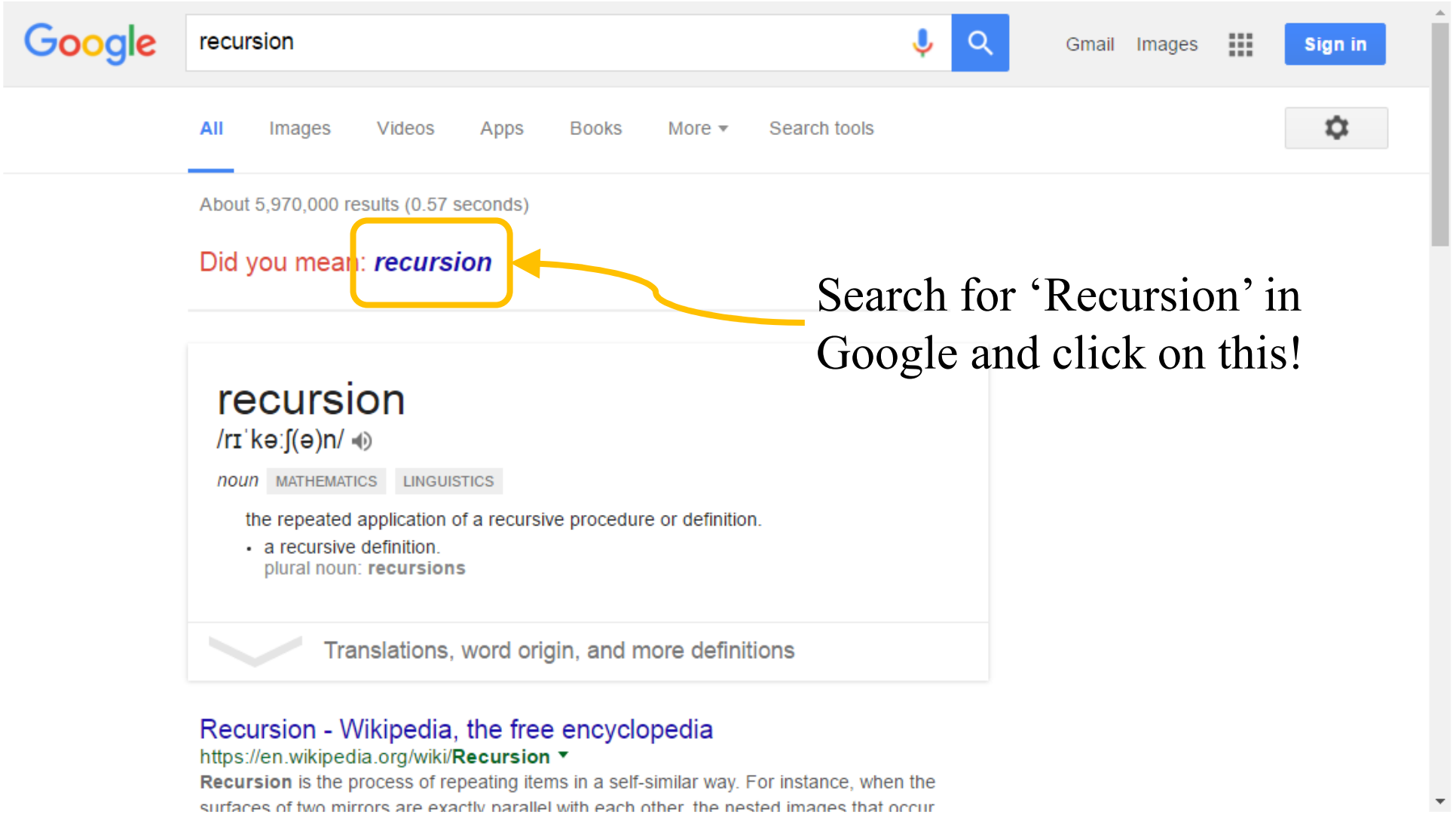
What is Recursion?

- A recursive function/subroutine is one *which calls itself*

- Recursive functions/subroutines can be very useful for some computing tasks
- For example, a cleverly written small recursive function or subroutine can sometimes replace many lines of code



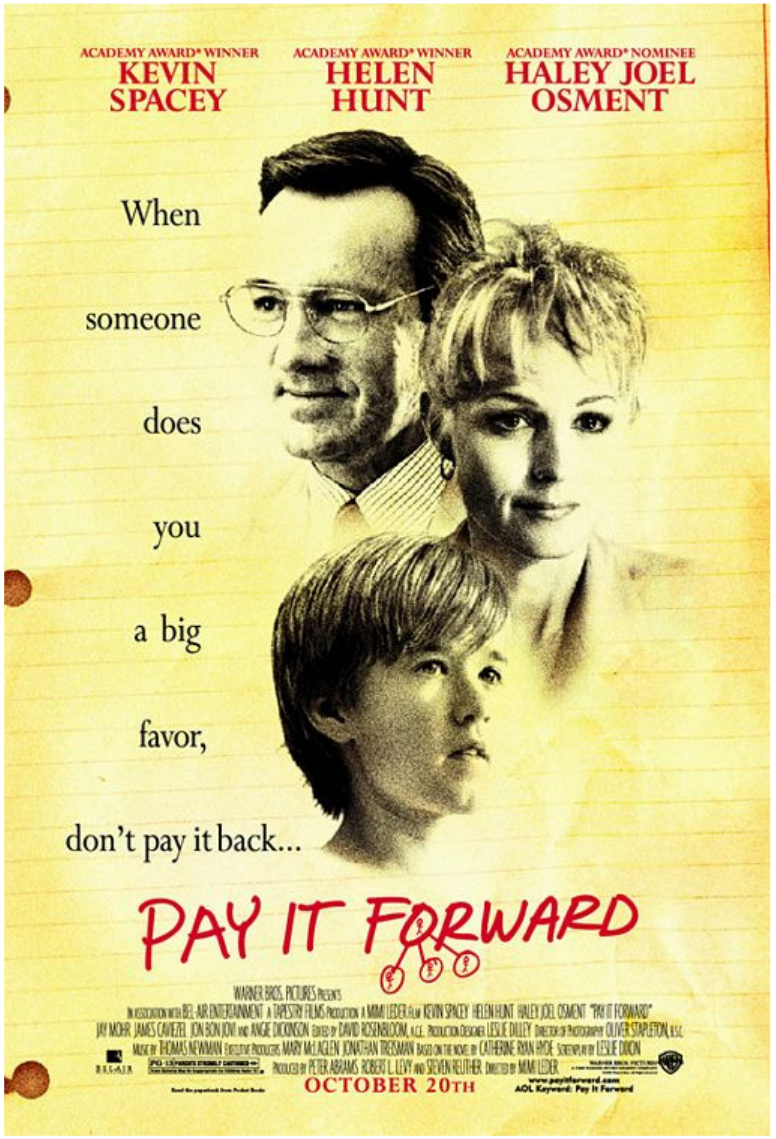
Recursion in Google



Search for ‘Recursion’ in Google and click on this!

‘Pay It Forward’

- A 2000 film about a boy who has been asked to think of a plan that will change the world
- He comes up with a plan that when someone receives a good deed, he/she helps 3 different other people



‘Pay It Forward’ Pseudo-Code

- *Pseudo-code* is used to show the general idea of a procedure

Sub Help(*Benefactor*, *Person*)

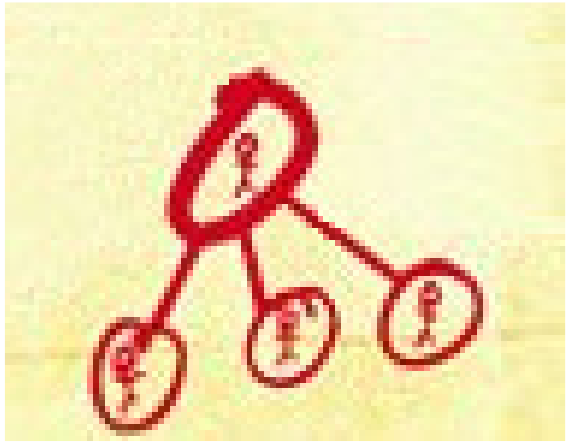
Person receives help from *Benefactor*

Help *Person*, *RandomPerson1*

Help *Person*, *RandomPerson2*

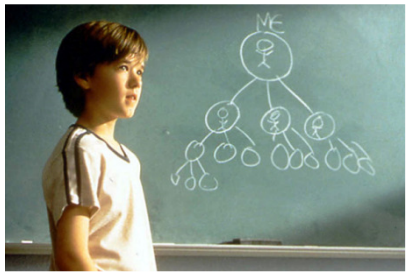
Help *Person*, *RandomPerson3*

End Sub

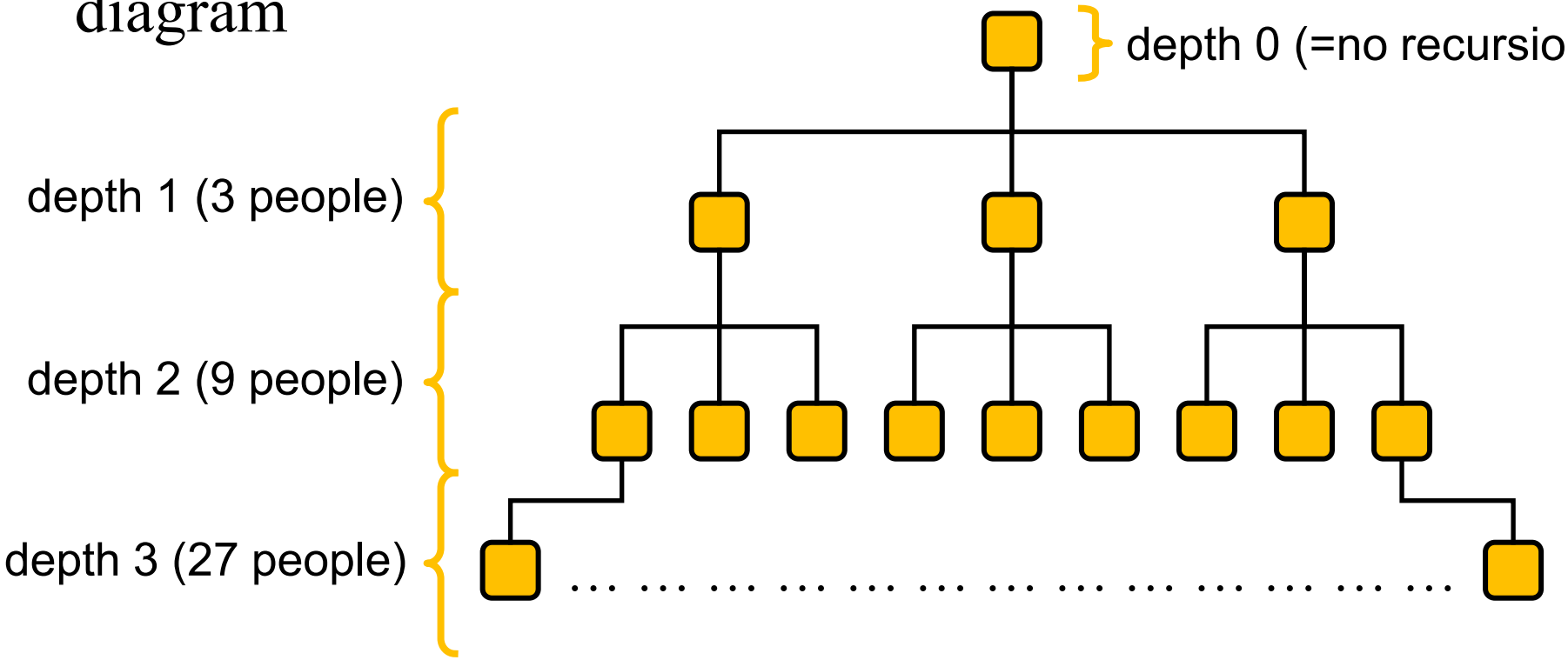


- The whole process starts with one person helping another, for example: **Help *Me*, *You***
- The above example uses pseudo-code, but the rest of this presentation uses real VBA code

Recursive Depths



- How many good deeds are done in total after 3 depths?
- You can see what we mean by *depth* in the following diagram



- The answer is that when the maximum depth is 3, 1+3+9+27=40 good deeds in total are done

Finding the Powers of Numbers

- You know that x^y is x to the power of y, i.e.:

$$\underbrace{x \cdot x \cdot \ldots \cdot x}_{y \text{ copies of } x \text{ in the multiplication}}$$

- A simple **for** loop could be used to find x to the power of y, as shown below:

```
Result = 1
For Power = 1 To y
    Result = Result * x
Next
```

The loop runs `y` times

Thinking Recursively

- Alternatively, x^y can be expressed *recursively* as:

$$x^y = x \cdot x^{(y-1)}$$

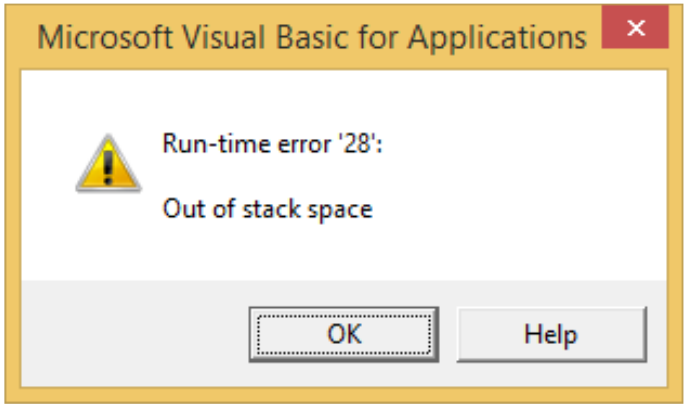
- Based on this expression, we can define a *recursive* function `Power` that calculates x^y as follows:

```
Function Power(x, y)
    Power = x * Power(x, y - 1)
End Function
```

The function runs itself to obtain $x^{(y-1)}$

Running the Recursive Function 1/2

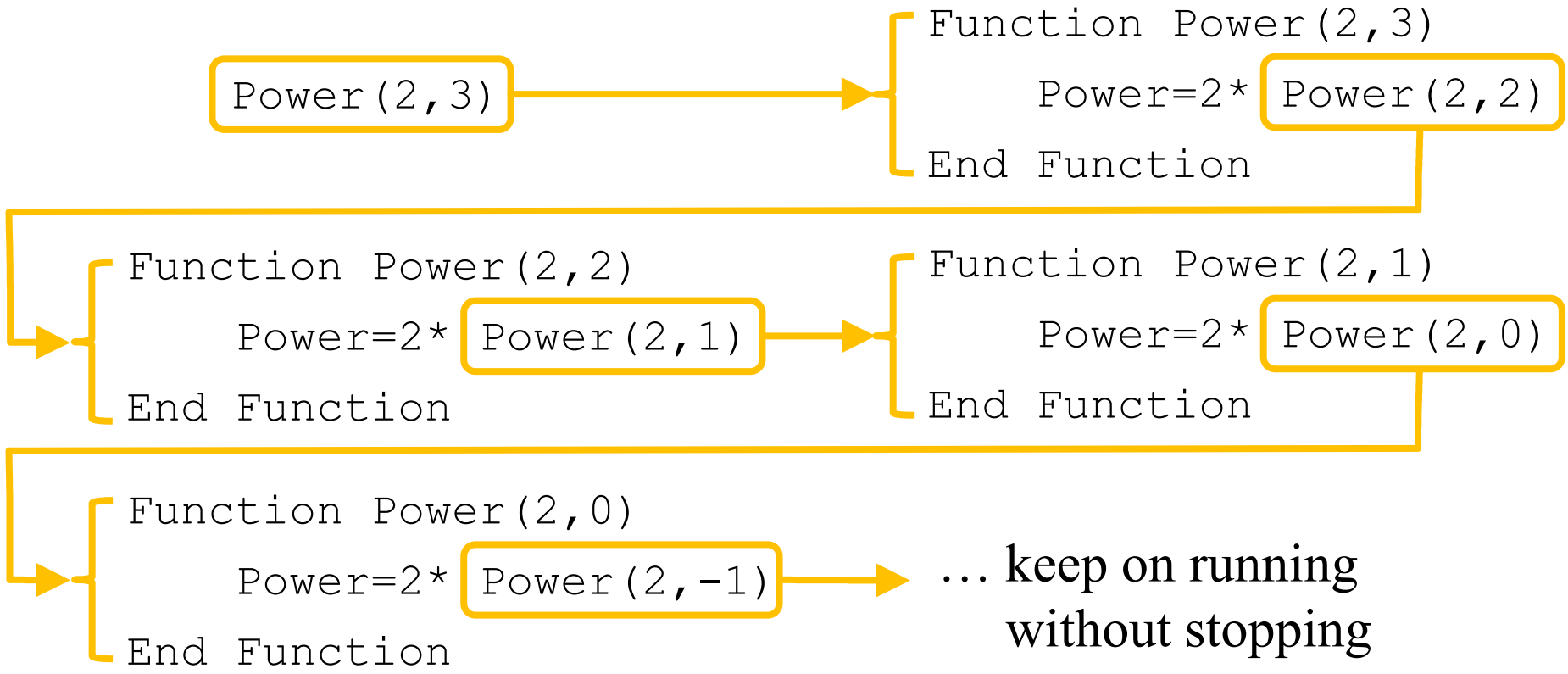
- However, if you run the function on the previous slide, it will never stop! (and eventually crash, i.e. stop running abruptly)
- The error that occurs when you do this is shown below:



- The error means that the program has to stop because it runs out of memory

Running the Recursive Function 2/2

- This is what happens if you run `Power(2,3)`:



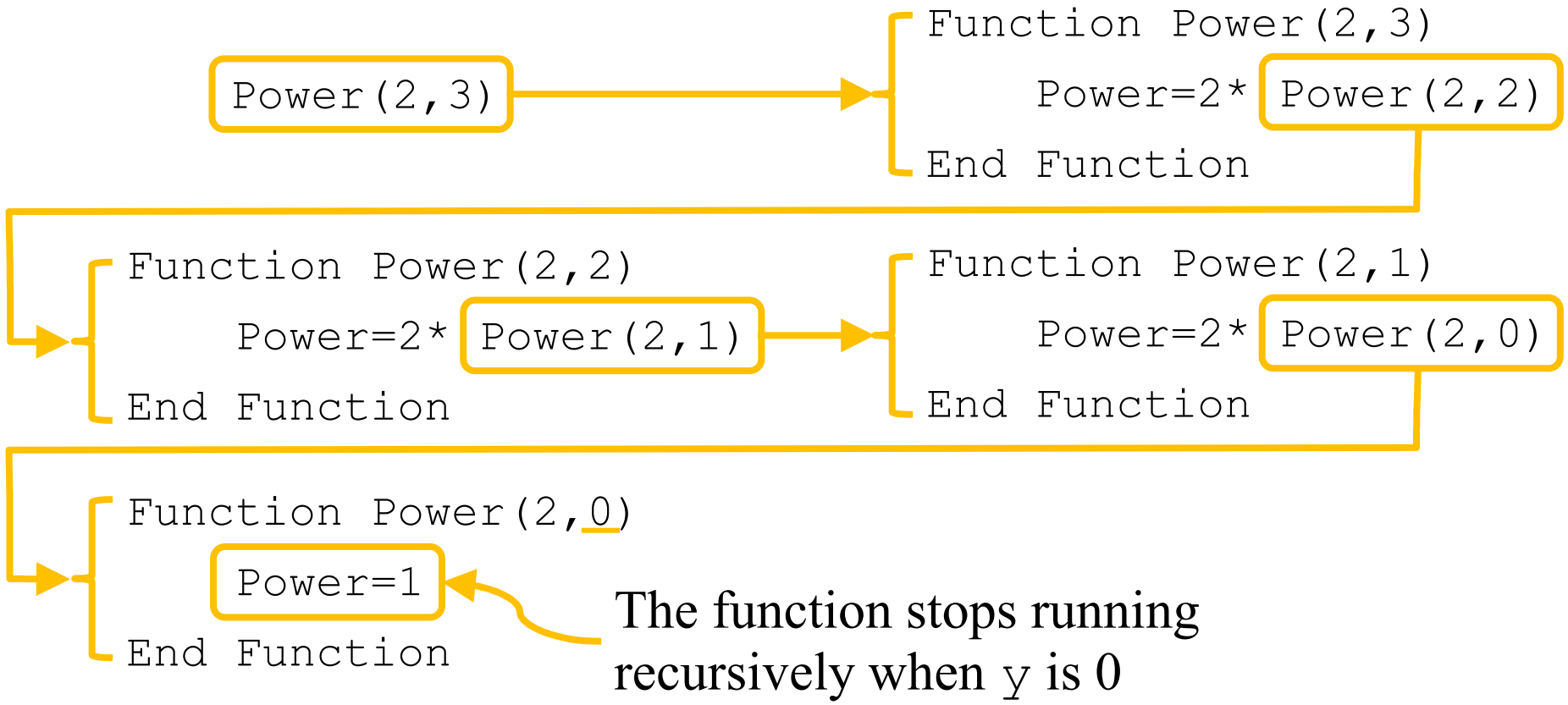
The Base Case

- When y is 0, x^0 is equal to 1, rather than $x \cdot x^{-1}$
- In recursion, this is called the *base case*
- In the example, we can include a condition to stop running the function recursively in the base case, i.e. when `y` is 0, like this:


```
Function Power(x, y)
    If y = 0 Then
        Power = 1
    Else
        Power = x * Power(x, y - 1)
    End If
End Function
```

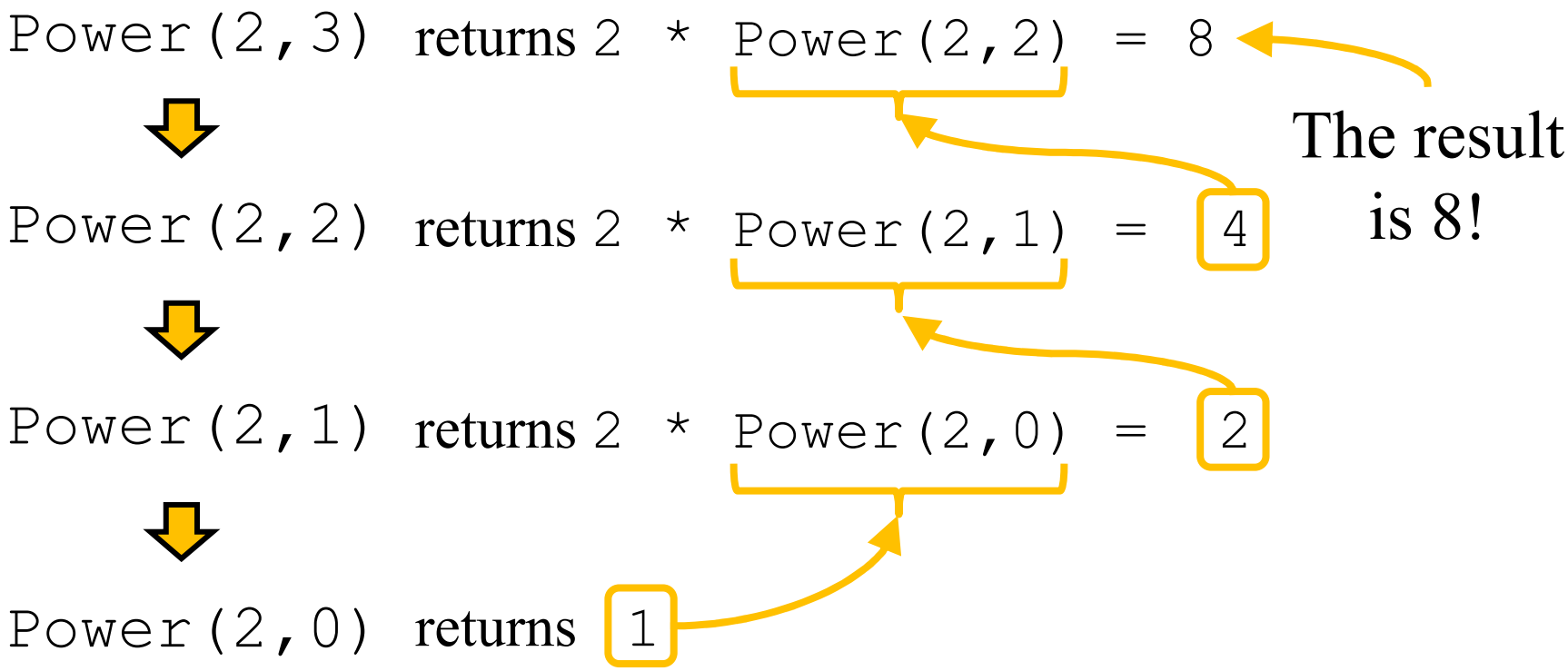
Running the Function Again

- Let's run the function again with `Power(2,3)`:



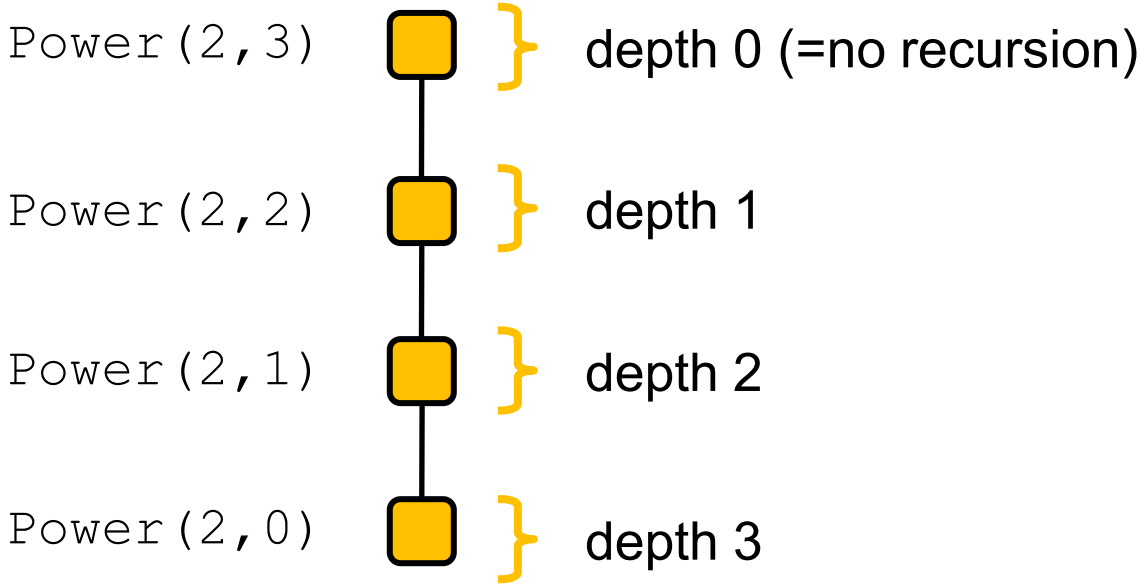
The Result of Running the Function

- We can see the return value of `Power(2,3)` by following the execution of the function:



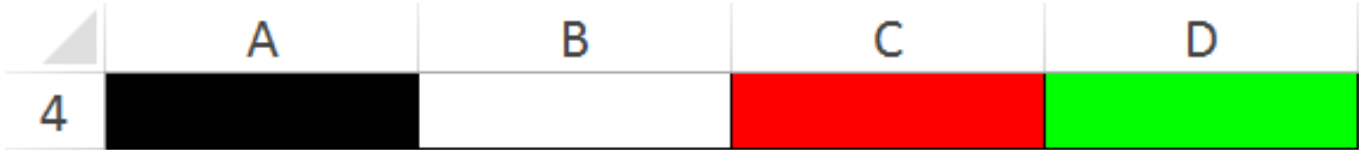
Recursive Depths

- So for the example shown on the last slide, the recursive calling pattern looks like this:



An Example Recursive Subroutine

- The previous example is a recursive **function**
- Let's look at an example which uses a recursive **subroutine** called `ChangeColour`
- The example starts the recursive subroutine using this code: `ChangeColour 1`
- The subroutine changes the colour of the first four cells of row 4 like this:



The ChangeColour Subroutine

- Here is the code of the subroutine:

```
Sub ChangeColour(Col)
    ' Change the background colour
    Cells(4, Col).Interior.ColorIndex = Col
    ' Change the borders to black
    Cells(4, Col).Borders.Color = vbBlack

    ' Recursively change the first four cells
    If Col < 4 Then
        ChangeColour Col + 1
    End If
End Sub
```

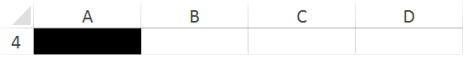
The colour of the cell is set to `Col`

The subroutine runs itself using the next column

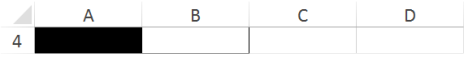
Running the Subroutine

```
ChangeColour 1
```

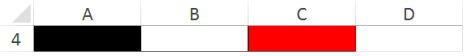
```
Sub ChangeColour(1)
    Cells(4,1).Interior.ColorIndex = 1
    ChangeColour 1 + 1
End Sub
```



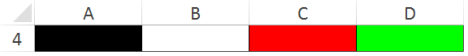

```
Sub ChangeColour(2)
    Cells(4,2).Interior.ColorIndex = 2
    ChangeColour 2 + 1
End Sub
```




```
Sub ChangeColour(3)
    Cells(4,3).Interior.ColorIndex = 3
    ChangeColour 3 + 1
End Sub
```



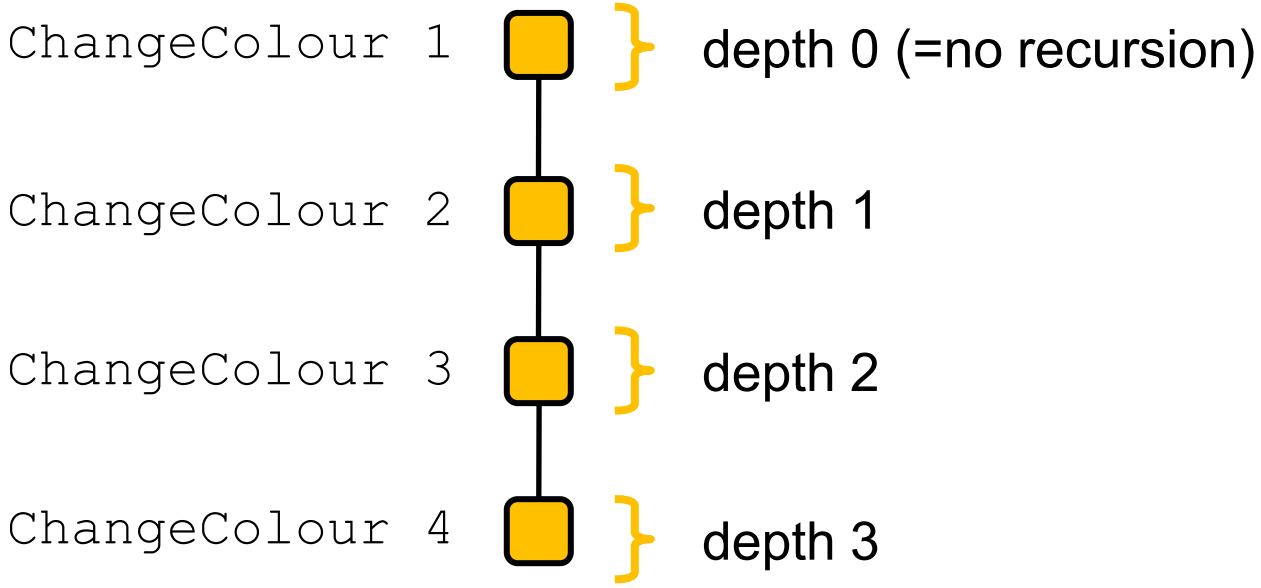

```
Sub ChangeColour(4)
    Cells(4,4).Interior.ColorIndex = 4
End Sub
```



Recursive subroutine is not run anymore because of the if statement

Recursive Depths

- For the example shown on the last slide, the recursive calling pattern looks like this:



Using a Loop

- Again, you can use a simple loop to achieve the same thing, which is shown below:

```
For Col = 1 To 4
    Cells(4, Col).Interior.ColorIndex = Col
    Cells(4, Col).Borders.Color = vbBlack
Next Col
```

- However, as we will soon see, sometimes it is easier to write things using recursion

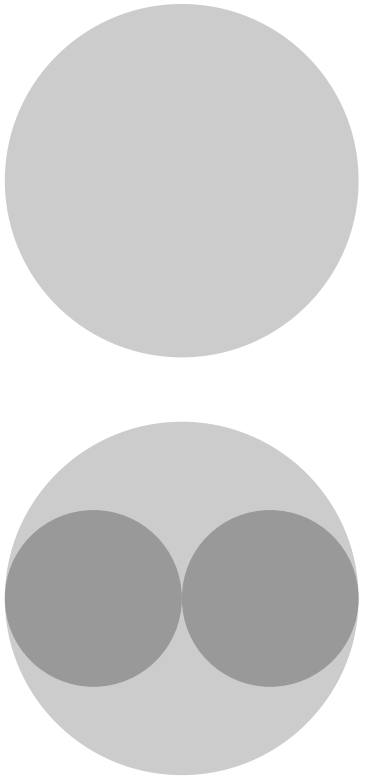
Drawing Recursive Circles

- Recursive functions/subroutines are used for many purposes
- One of them is to generate computer graphics
- The next example draws circles recursively using lots of circle shape objects in Excel
- Basically, inside one circle we draw two circles with smaller identical radiuses, and then the process repeats itself for the two smaller circles
- In this example, circles at deeper depths are darker
- To do this, we set the brightness at each level to be:

(TotalNumOfDepth – CurrentDepth) / TotalNumOfDepth

The Recursive Circle Process 1/2

- Let's look the drawing process of the circles when the maximum depth of the recursion is 4
  - At a depth of 0, a big circle is drawn
  - At a depth of 1, two smaller circles, one on the left and one on the right, are drawn on top of the big circle with the colour adjusted based on the depth of the circles

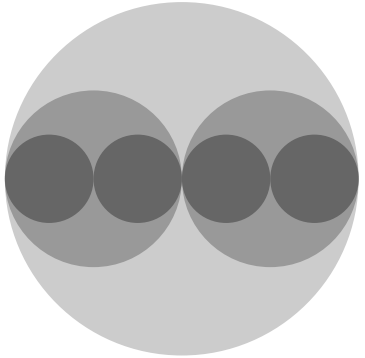


Depth = 0

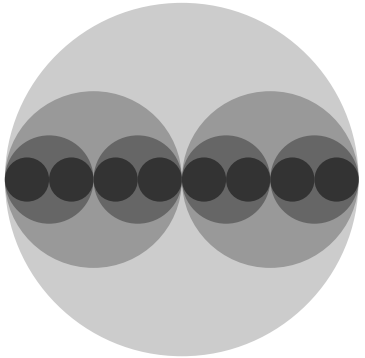
Depth = 1

The Recursive Circle Process 2/2

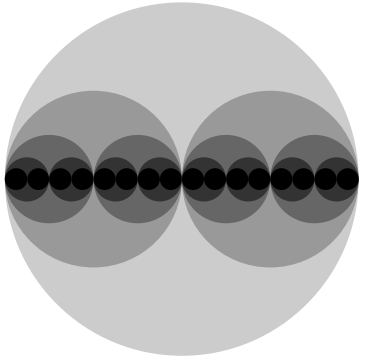
- At a depth of 2, even smaller circles are drawn recursively on each of the smaller circles
- At a depth of 3, smaller circles are drawn recursively on top of the smallest circle
- At a depth of 4, the smallest circles are drawn using black colour



Depth = 2

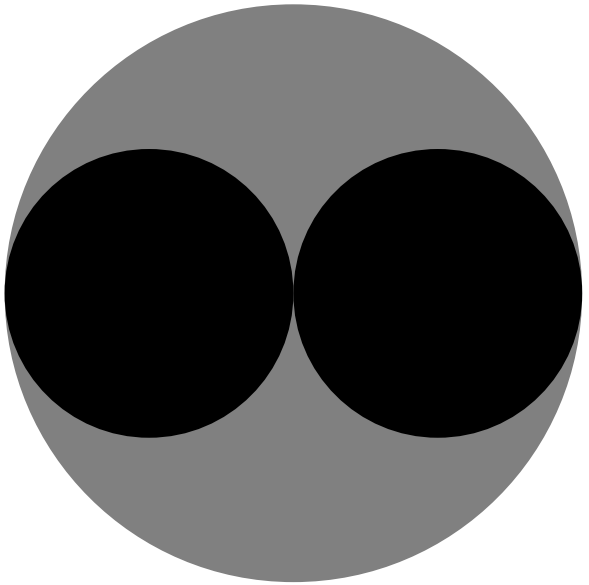


Depth = 3

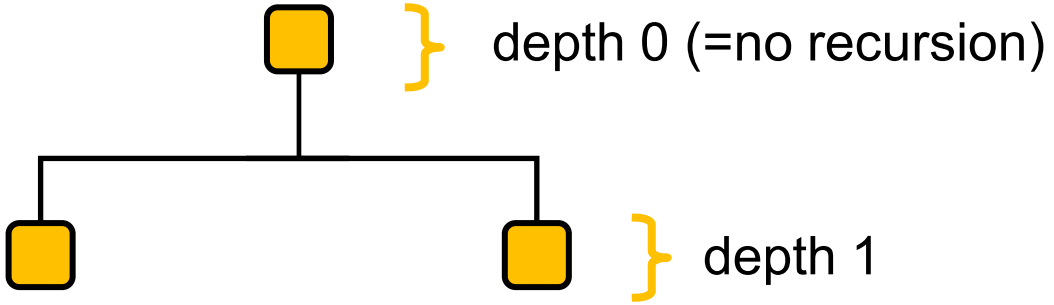


Depth = 4

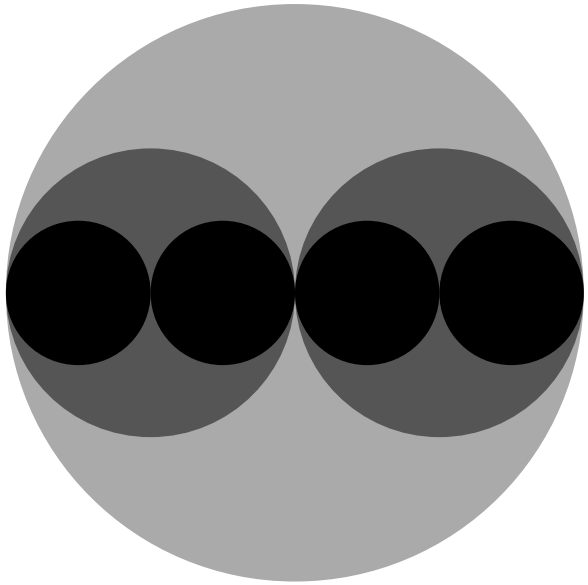
Recursive Depths



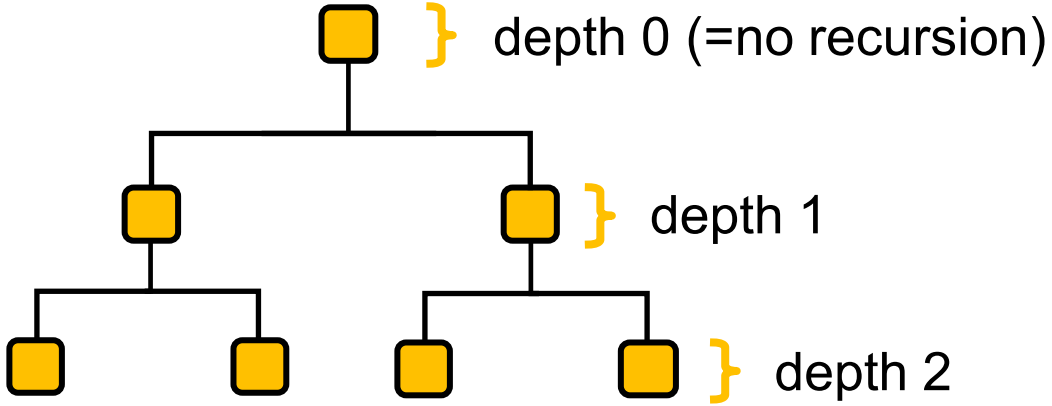
- For this example, when the maximum depth=1, this is what the depths look like:



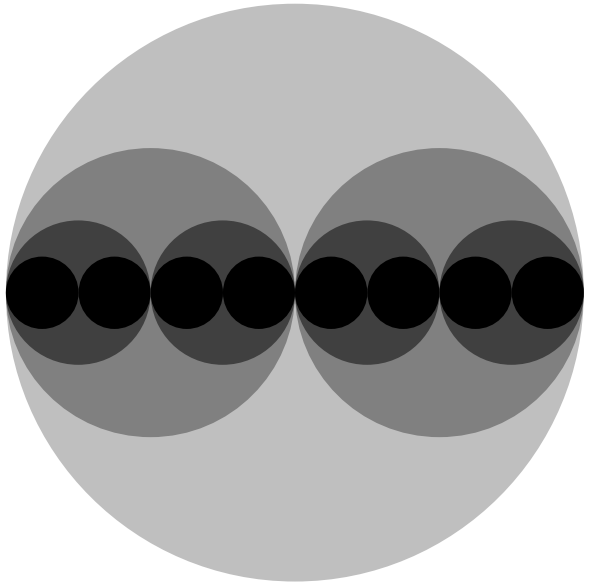
Recursive Depths



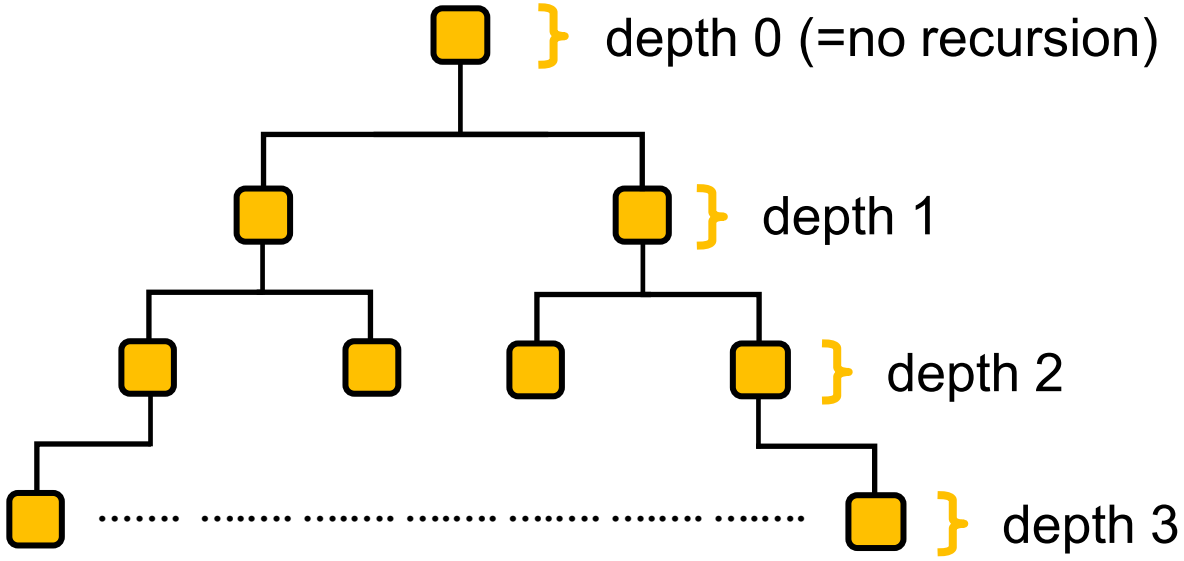
- For this example, when the maximum depth=2, this is what the depths look like:



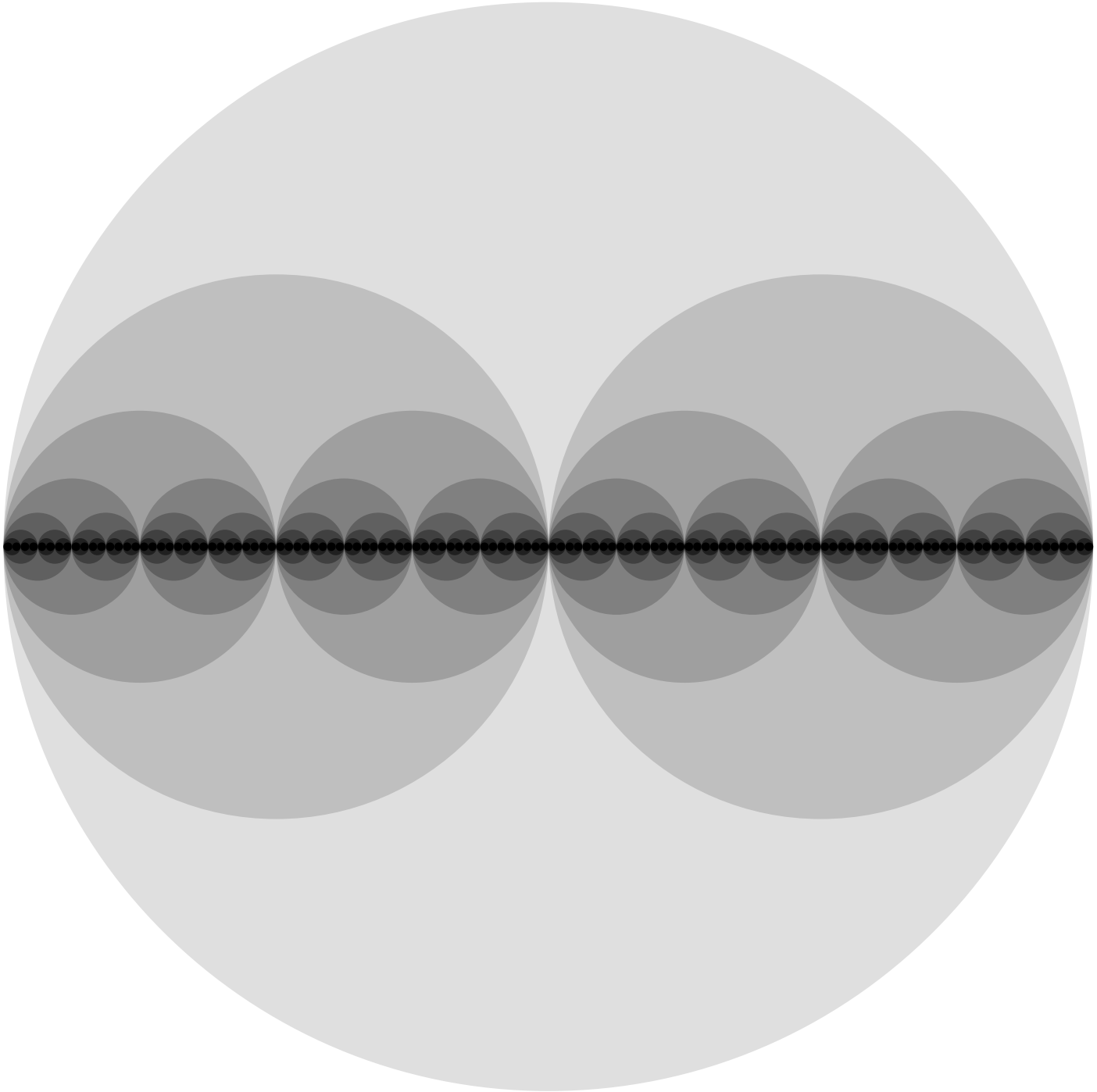
Recursive Depths



- For this example, when the maximum depth=3, this is what the depths look like:

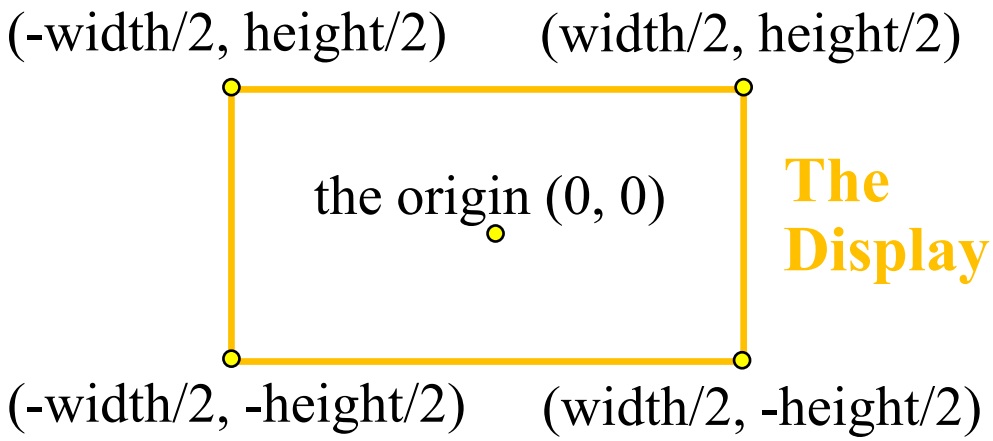


The recursive circles, after many recursive calls

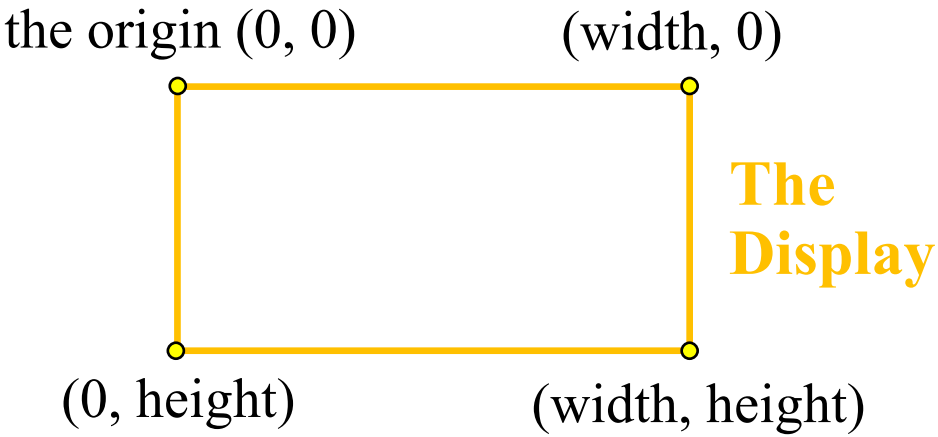


Quick Reminder of the Coordinate System

- The Cartesian coordinate system:
  - You probably used this system when you learned Math at school



- The VBA coordinate system:
  - When you draw VBA shape objects, all the x and y values are positive



Circle Recursive Code 1/4

- This slide shows the preparation code

```
Dim MaxDepth As Integer

Sub InputMaxDepth()
    Dim CenterX As Double, Radius As Double

    CenterX = 200
    Radius = 128

    MaxDepth = InputBox("Enter the maximum " & _
                        "depth for drawing the circles: ")

    DrawCircle CenterX, Radius, 0
End Sub
```

If you want to create multiple variables in a single `Dim` statement, you can list them using commas like this

Start the recursion process with the biggest circle

The Recursive Subroutine

- The recursive subroutine is shown on the next 3 slides:

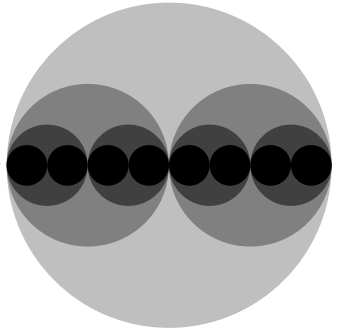
`DrawCircle` *CenterX*`,` *Radius*`,` *Depth*

CenterX is the x coordinate of the center of this circle

Radius is the radius of the current circle to be drawn

Depth is the current depth of the recursion

- In this example the value of y is always the same so we don't need to pass it/change it, we can simply use a constant value



Circle Recursive Code 2/4

- The recursive subroutine

```
Sub DrawCircle(ByVal CenterX As Double, _
        ByVal Radius As Double, ByVal Depth As Integer)
    Dim GrayColour As Double
    Dim CenterY As Double
    Dim Top As Double, Left As Double
    Dim Width As Double, Height As Double
    Dim CircleObj As Shape

    CenterY = 250

    ' Calculate the brightness
    GrayColour = CDbl(MaxDepth - Depth) / _
                 (MaxDepth + 1)
```

Calculate and set the brightness of this circle

`CDbl` means 'convert the number to a double'

Continued on the next slide

Circle Recursive Code 3/4

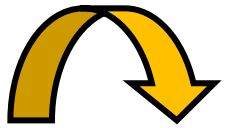
- The recursive subroutine, cont.

```
Left = CenterX - Radius
Top = CenterY - Radius

Width = Radius * 2
Height = Radius * 2

Set CircleObj = ActiveSheet.Shapes.AddShape( _
                    msoShapeOval, Left, Top, _
                    Width, Height)
CircleObj.Line.Visible = False
CircleObj.Fill.ForeColor.RGB = RGB(255 * GrayColour, _
                    255 * GrayColour, _
                    255 * GrayColour)
```

Draw a filled circle

These RGB values make a gray colour

Continued on the next slide



Circle Recursive Code 4/4

- The recursive subroutine, cont.

```
    If Depth < MaxDepth Then
        DrawCircle CenterX - Radius/2, _
                   Radius/2, Depth + 1
        DrawCircle CenterX + Radius/2, _
                   Radius/2, Depth + 1
    End If
End Sub
```

Run `DrawCircle` twice to handle the left and right areas