

### 5-9: Information Functions

There are some supporting functions provided by VBA so that you can check the types of values inside variables or returned from functions.

Here are some functions in the Information module.

Function	Description
Err() As ErrObject	Returns the Err object of the current application
IsArray( <i>varName</i> ) As Boolean	Checks if the variable is an array
IsDate( <i>expression</i> ) As Boolean	Checks if the expression is a date
IsEmpty( <i>expression</i> ) As Boolean	Checks if the expression has the value Empty
IsNull( <i>expression</i> ) As Boolean	Checks if the expression has the value Null
IsNumeric( <i>expression</i> ) As Boolean	Checks if the expression is a number
IsObject( <i>expression</i> ) As Boolean	Checks if the expression is an object
RGB( <i>red</i> As Integer, <i>green</i> As Integer, <i>blue</i> As Integer) As Long	Returns a value representing an RGB color
TypeName( <i>varName</i> ) As String	Returns the type name of a variable

### 5-10: Number Functions

Here is a list of math functions provided in the Math module.

Function	Description								
Abs( <i>number</i> )	Absolute value of the number								
Atn( <i>number</i> As Double) As Double	Arc-tangent of a number (in radians)								
Cos( <i>number</i> As Double) As Double	Cosine of a number								
Exp( <i>number</i> As Double) As Double	The exponential function ( $e^x$ )								
Log( <i>number</i> As Double) As Double	The natural logarithm of a number								
Randomize([ <i>number</i> ])	Randomize the random number generator, with an optional argument <i>number</i> as the seed. If not supplied, the system time is used as the seed.								
Rnd([ <i>number</i> ]) As Single	Return a random number. The optional argument <i>number</i> : <table border="1" data-bbox="699 1413 1490 1641"> <thead> <tr> <th>If <i>number</i> is</th> <th>Rnd generates</th> </tr> </thead> <tbody> <tr> <td>Less than zero</td> <td>The same number every time, using <i>number</i> as the seed.</td> </tr> <tr> <td>Greater than zero</td> <td>The next random number in the sequence.</td> </tr> <tr> <td>Equal to zero</td> <td>The most recently generated number.</td> </tr> </tbody> </table>	If <i>number</i> is	Rnd generates	Less than zero	The same number every time, using <i>number</i> as the seed.	Greater than zero	The next random number in the sequence.	Equal to zero	The most recently generated number.
If <i>number</i> is	Rnd generates								
Less than zero	The same number every time, using <i>number</i> as the seed.								
Greater than zero	The next random number in the sequence.								
Equal to zero	The most recently generated number.								
Round( <i>number</i> , [ <i>numDigitsAfterDecimal</i> As Long])	Round a number to a specified number of decimal places. If <i>numDigitsAfterDecimal</i> is omitted, the number is rounded to a whole number								
Sgn( <i>number</i> )	Return a number indicating the sign of a number <table border="1" data-bbox="699 1789 1206 1944"> <thead> <tr> <th>If <i>number</i> is</th> <th>Sgn returns</th> </tr> </thead> <tbody> <tr> <td>Greater than zero</td> <td>1</td> </tr> <tr> <td>Equal to zero</td> <td>0</td> </tr> <tr> <td>Less than zero</td> <td>-1</td> </tr> </tbody> </table>	If <i>number</i> is	Sgn returns	Greater than zero	1	Equal to zero	0	Less than zero	-1
If <i>number</i> is	Sgn returns								
Greater than zero	1								
Equal to zero	0								
Less than zero	-1								
Sin( <i>Number</i> As Double) As Double	Sine of a number								
Sqr( <i>Number</i> As Double) As Double	Square root of a number								
Tan( <i>Number</i> As Double) As Double	Tangent of a number								

Here is a list of financial functions stated in the `Financial` module:

<b>Function</b>	<b>Description</b>
<code>DDB(Cost As Double, Salvage As _ Double, Life As Double, Period As _ Double, [Factor]) As Double</code>	Returns a Double specifying the depreciation of an asset for a specific time period using the double-declining balance method or some other method you specify.
<code>FV(Rate As Double, NPer As Double, _ Pmt As Double, [PV], [Due]) As Double</code>	Returns a Double specifying the future value of an annuity based on periodic, fixed payments and a fixed interest rate.
<code>IPmt(Rate As Double, Per As Double, _ NPer As Double, PV As Double, [FV], _ [Due]) As Double</code>	Returns a Double specifying the interest payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.
<code>IRR(ValueArray() As Double, [Guess]) _ As Double</code>	Returns a Double specifying the internal rate of return for a series of periodic cash flows (payments and receipts).
<code>MIRR(ValueArray() As Double, _ FinanceRate As Double, ReinvestRate _ As Double) As Double</code>	Returns a Double specifying the modified internal rate of return for a series of periodic cash flows (payments and receipts).
<code>NPer(Rate As Double, Pmt As Double, _ PV As Double, [FV], [Due]) As Double</code>	Returns a Double specifying the number of periods for an annuity based on periodic, fixed payments and a fixed interest rate.
<code>NPV(Rate As Double, ValueArray() As _ Double) As Double</code>	Returns a Double specifying the net present value of an investment based on a series of periodic cash flows (payments and receipts) and a discount rate.
<code>Pmt(Rate As Double, NPer As Double, _ PV As Double, [FV], [Due]) As Double</code>	Returns a Double specifying the payment for an annuity based on periodic, fixed payments and a fixed interest rate.
<code>PPmt(Rate As Double, Per As Double, _ NPer As Double, PV As Double, [FV], _ [Due]) As Double</code>	Returns a Double specifying the principal payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.
<code>PV(Rate As Double, NPer As Double, _ Pmt As Double, [FV], [Due]) As Double</code>	Returns a Double specifying the present value of an annuity based on periodic, fixed payments to be paid in the future and a fixed interest rate.
<code>Rate(NPer As Double, Pmt As Double, _ PV As Double, [FV], [Due], [Guess]) _ As Double</code>	Returns a Double specifying the interest rate per period for an annuity.
<code>SLN(Cost As Double, Salvage As _ Double, Life As Double) As Double</code>	Returns a Double specifying the straight-line depreciation of an asset for a single period.
<code>SYD(Cost As Double, Salvage As _ Double, Life As Double, Period As _ Double) As Double</code>	Returns a Double specifying the sum-of-years' digits depreciation of an asset for a specified period.

In addition to the above functions, you can use all the worksheet functions with the `WorksheetFunction` object. You can access the `WorksheetFunction` object by accessing `Application.WorksheetFunction`.

Example of using the `WorksheetFunction` object:

```
Set myRange = Worksheets("Sheet1").Range("A1:C10")
answer = Application.WorksheetFunction.Min(myRange)
MsgBox answer
```

**Let's try! Corner**

Calculate  $\Pr(N \leq 100)$  if  $N$  follows the Poisson distribution with mean 100. Output the result using a message box.

**5-11: String Functions**

From the `Strings` module we have the following String manipulating functions.

**5-11-1: Filter Function**

Returns a zero-based array containing subset of a string array based on a specified filter criteria.

```
Filter(sourcearray, match[, include[, compare]])
```

Part	Description		
<i>sourcearray</i>	Required. One-dimensional array of strings to be searched.		
<i>Match</i>	Required. String to search for.		
<i>include</i>	Optional. Boolean value indicating whether to return substrings that include or exclude <i>match</i> . If <i>include</i> is True, <code>Filter</code> returns the subset of the array that contains <i>match</i> as a substring. If <i>include</i> is False, <code>Filter</code> returns the subset of the array that does not contain <i>match</i> as a substring.		
<i>compare</i>	Optional <code>VbCompareMethod</code> . Numeric value indicating the kind of string comparison to use.		
	Constant	Value	Description
	<code>vbUseCompareOption</code>	-1	Performs a comparison using the setting of the <code>Option Compare</code> statement.
	<code>vbBinaryCompare</code>	0	Performs a binary comparison.
	<code>vbTextCompare</code>	1	Performs a textual comparison.
<code>vbDatabaseCompare (Advanced)</code>	2	Microsoft Access only. Performs a comparison based on information in your database.	

Remarks:

If no matches of *match* are found within *sourcearray*, `Filter` returns an empty array. An error occurs if *sourcearray* is Null or is not a one-dimensional array.

Option Compare statement:

```
Option Compare {Binary | Text | Database}
```

If used, the `Option Compare` statement must appear in a module before any procedures. It specifies the string comparison method (Binary, Text, or Database) for a module.

String comparison method	Meaning
Binary	Case-sensitive string comparison
Text	Case-insensitive string comparison
Database (Advanced)	Only be used within Microsoft Access. String comparisons based on the sort order determined by the locale ID of the database

Example:

```
Dim strArray(5) As String
strArray(0) = "Albert"
strArray(1) = "Andrew"
strArray(2) = "Andy"
strArray(3) = "Betty"
strArray(4) = "Candy"
strArray(5) = "Carmen"
outputArray Filter(strArray, "and", True, vbBinaryCompare)
'outputs "Candy"
outputArray Filter(strArray, "and", True, vbTextCompare)
'outputs "Andrew", "Andy", "Candy"
outputArray Filter(strArray, "ca", False, vbTextCompare)
'outputs "Albert", "Andrew", "Andy", "Betty"
```

### 5-11-2: FormatCurrency Function

Returns an expression formatted as a currency value using the currency symbol defined in the system control panel.

```
FormatCurrency(Expression[,NumDigitsAfterDecimal [,IncludeLeadingDigit _
[,UseParensForNegativeNumbers [,GroupDigits]]]])
```

Part	Description												
<i>Expression</i>	Required. Expression to be formatted.												
<i>NumDigitsAfterDecimal</i>	Optional. Numeric value indicating how many places to the right of the decimal are displayed. Default value is -1, which indicates that the computer's regional settings are used.												
<i>IncludeLeadingDigit</i>	Optional. Tristate constant that indicates whether or not a leading zero is displayed for fractional values. Tristate constant can be one of the followings: <table border="1" data-bbox="603 1317 1506 1509"> <thead> <tr> <th>Constant</th> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>vbTrue</td> <td>-1</td> <td>True</td> </tr> <tr> <td>vbFalse</td> <td>0</td> <td>False</td> </tr> <tr> <td>vbUseDefault</td> <td>-2</td> <td>Use the setting from the computer's regional settings.</td> </tr> </tbody> </table>	Constant	Value	Description	vbTrue	-1	True	vbFalse	0	False	vbUseDefault	-2	Use the setting from the computer's regional settings.
Constant	Value	Description											
vbTrue	-1	True											
vbFalse	0	False											
vbUseDefault	-2	Use the setting from the computer's regional settings.											
<i>UseParensForNegativeNumbers</i>	Optional. Tristate constant that indicates whether or not to place negative values within parentheses.												
<i>GroupDigits</i>	Optional. Tristate constant that indicates whether or not numbers are grouped using the group delimiter specified in the computer's regional settings.												

Example:

```
MsgBox FormatCurrency(-1234.5678, 2, , vbTrue, vbTrue) 'outputs ($1,234.57)
```

### 5-11-3: FormatDateTime Function

Returns an expression formatted as a date or time.

```
FormatDateTime(Date[,NamedFormat])
```

Part	Description																		
<i>Date</i>	Required. Date expression to be formatted.																		
<i>NamedFormat</i>	Optional. Numeric value that indicates the date/time format used. If omitted, <code>vbGeneralDate</code> is used.																		
	<table border="1"> <thead> <tr> <th>Constant</th> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>vbGeneralDate</code></td> <td>0</td> <td>Display a date and/or time. If there is a date part, display it as a short date. If there is a time part, display it as a long time. If present, both parts are displayed.</td> </tr> <tr> <td><code>vbLongDate</code></td> <td>1</td> <td>Display a date using the long date format specified in your computer's regional settings.</td> </tr> <tr> <td><code>vbShortDate</code></td> <td>2</td> <td>Display a date using the short date format specified in your computer's regional settings.</td> </tr> <tr> <td><code>vbLongTime</code></td> <td>3</td> <td>Display a time using the time format specified in your computer's regional settings.</td> </tr> <tr> <td><code>vbShortTime</code></td> <td>4</td> <td>Display a time using the 24-hour format (hh:mm).</td> </tr> </tbody> </table>	Constant	Value	Description	<code>vbGeneralDate</code>	0	Display a date and/or time. If there is a date part, display it as a short date. If there is a time part, display it as a long time. If present, both parts are displayed.	<code>vbLongDate</code>	1	Display a date using the long date format specified in your computer's regional settings.	<code>vbShortDate</code>	2	Display a date using the short date format specified in your computer's regional settings.	<code>vbLongTime</code>	3	Display a time using the time format specified in your computer's regional settings.	<code>vbShortTime</code>	4	Display a time using the 24-hour format (hh:mm).
Constant	Value	Description																	
<code>vbGeneralDate</code>	0	Display a date and/or time. If there is a date part, display it as a short date. If there is a time part, display it as a long time. If present, both parts are displayed.																	
<code>vbLongDate</code>	1	Display a date using the long date format specified in your computer's regional settings.																	
<code>vbShortDate</code>	2	Display a date using the short date format specified in your computer's regional settings.																	
<code>vbLongTime</code>	3	Display a time using the time format specified in your computer's regional settings.																	
<code>vbShortTime</code>	4	Display a time using the 24-hour format (hh:mm).																	

#### 5-11-4: FormatNumber Function

Returns an expression formatted as a number.

```
FormatNumber(Expression[,NumDigitsAfterDecimal [,IncludeLeadingDigit _
[,UseParensForNegativeNumbers [,GroupDigits]]]])
```

See `FormatCurrency` function for the descriptions of the arguments.

#### 5-11-5: FormatPercent Function

Returns an expression formatted as a percentage (multiplied by 100) with a trailing % character.

```
FormatPercent(Expression[,NumDigitsAfterDecimal [,IncludeLeadingDigit _
[,UseParensForNegativeNumbers [,GroupDigits]]]])
```

See `FormatCurrency` function for the descriptions of the arguments.

#### 5-11-6: InStr Function

Returns a Long specifying the position of the first occurrence of one string within another.

```
InStr([start, ]string1, string2[, compare])
```

Part	Description
<i>start</i>	Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the first character position. If <i>start</i> contains Null, an error occurs. The <i>start</i> argument is required if <i>compare</i> is specified.
<i>string1</i>	Required. String expression being searched.
<i>string2</i>	Required. String expression sought.
<i>compare</i>	Optional <code>VbCompareMethod</code> . Specifies the type of string comparison. If <i>compare</i> is Null, an error occurs.

<b>If</b>	<b>InStr returns</b>
<i>string1</i> is zero-length	0
<i>string1</i> is Null	Null
<i>string2</i> is zero-length	<i>start</i>
<i>string2</i> is Null	Null
<i>string2</i> is not found	0
<i>string2</i> is found within <i>string1</i>	Position at which match is found
<i>start</i> > <i>string2</i>	0

Example:

```
Dim SearchString, SearchChar, MyPos
SearchString = "XXpXXpXXPXXP" ' String to search in.
SearchChar = "P" ' Search for "P".

' A textual comparison starting at position 4. Returns 6.
MyPos = Instr(4, SearchString, SearchChar, 1)

' A binary comparison starting at position 1. Returns 9.
MyPos = Instr(1, SearchString, SearchChar, 0)

' Comparison is binary by default (last argument is omitted).
MyPos = Instr(SearchString, SearchChar) ' Returns 9.

MyPos = Instr(1, SearchString, "W") ' Returns 0.
```

### **5-11-7: InStrRev Function**

Returns the position of an occurrence of one string within another, from the end of string.

```
InstrRev(stringcheck, stringmatch[, start[, compare]])
```

<b>Part</b>	<b>Description</b>
<i>stringcheck</i>	Required. String expression being searched.
<i>stringmatch</i>	Required. String expression being searched for.
<i>start</i>	Optional. Numeric expression that sets the starting position for each search. If omitted, -1 is used, which means that the search begins at the last character position. If <i>start</i> contains Null, an error occurs.
<i>compare</i>	Optional VbCompareMethod. Numeric value indicating the kind of comparison to use when evaluating substrings. If omitted, a binary comparison is performed.

The return values are similar to that of InStr function.

### **5-11-8: Join Function**

Returns a string created by joining a number of substrings contained in an array.

```
Join(sourcearray[, delimiter])
```

Part	Description
<i>sourcearray</i>	Required. One-dimensional array containing substrings to be joined.
<i>delimiter</i>	Optional. String character used to separate the substrings in the returned string. If omitted, the space character (" ") is used. If <i>delimiter</i> is a zero-length string (""), all items in the list are concatenated with no delimiters.

Example:

This function joins an array of strings with new lines and output the result as a text message.

```
Sub outputArray(anArray)
    MsgBox Join(anArray, vbCrLf)
End Sub
```

### **5-11-9: LCase Function**

Returns a String that has been converted to lowercase.

```
LCase(string)
```

### **5-11-10: Left Function**

Returns a String containing a specified number of characters from the left side of a string.

```
Left(string, Length)
```

Part	Description
<i>string</i>	Required. String expression from which the leftmost characters are returned. If <i>string</i> contains Null, Null is returned.
<i>Length</i>	Required; Long. Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in <i>string</i> , the entire string is returned.

Example:

```
Dim AnyString, MyStr
AnyString = "Hello World"    ' Define string.
MyStr = Left(AnyString, 1)   ' Returns "H".
MyStr = Left(AnyString, 7)   ' Returns "Hello W".
MyStr = Left(AnyString, 20)  ' Returns "Hello World".
```

### **5-11-11: Len Function**

Returns a Long containing the number of characters in a string.

```
Len(string)
```

### **5-11-12: LTrim, RTrim, and Trim Functions**

Returns a String containing a copy of a specified string without leading spaces (LTrim), trailing spaces (RTrim), or both leading and trailing spaces (Trim).

```
LTrim(string)
RTrim(string)
Trim(string)
```

### 5-11-13: Mid Function

Returns a String containing a specified number of characters from a string.

Mid(*string*, *start*[, *length*])

Part	Description
<i>string</i>	Required. String expression from which characters are returned. If <i>string</i> contains Null, Null is returned.
<i>start</i>	Required; Long. Character position in <i>string</i> at which the part to be taken begins. If <i>start</i> is greater than the number of characters in <i>string</i> , Mid returns a zero-length string ("").
<i>length</i>	Optional; Long. Number of characters to return. If omitted or if there are fewer than <i>length</i> characters in the text (including the character at <i>start</i> ), all characters from the <i>start</i> position to the end of the string are returned.

Example:

```
Dim MyString, FirstWord, LastWord, MidWords
MyString = "Mid Function Demo"      ' Create text string.
FirstWord = Mid(MyString, 1, 3)     ' Returns "Mid".
LastWord = Mid(MyString, 14, 4)     ' Returns "Demo".
MidWords = Mid(MyString, 5)        ' Returns "Function Demo".
```

### 5-11-14: Replace Function

Returns a string in which a specified substring has been replaced with another substring a specified number of times.

Replace(*expression*, *find*, *replace*[, *start*[, *count*[, *compare*]])

Part	Description
<i>expression</i>	Required. String expression containing substring to replace.
<i>find</i>	Required. Substring being searched for.
<i>replace</i>	Required. Replacement substring.
<i>start</i>	Optional. Position within <i>expression</i> where substring search is to begin. If omitted, 1 is assumed.
<i>count</i>	Optional. Number of substring substitutions to perform. If omitted, the default value is -1, which means make all possible substitutions.
<i>compare</i>	Optional VbCompareMethod. Numeric value indicating the kind of comparison to use when evaluating substrings.

If	Replace returns
<i>expression</i> is zero-length	Zero-length string ("")
<i>expression</i> is Null	An error.
<i>find</i> is zero-length	Copy of <i>expression</i> .
<i>replace</i> is zero-length	Copy of <i>expression</i> with all occurrences of <i>find</i> removed.
<i>start</i> > Len( <i>expression</i> )	Zero-length string.
<i>count</i> is 0	Copy of <i>expression</i> .

Example:

```
Dim ReplaceString, FindChar, ReplaceChar, MyPos
ReplaceString = "XXpXXpXXPXXP"
FindChar = "P"
ReplaceChar = "D"

' Returns "XXDXXDXXDXXD"
MyPos = Replace(ReplaceString, FindChar, ReplaceChar, , , vbTextCompare)

' Returns "XXDXXDXXP"
MyPos = Replace(ReplaceString, FindChar, ReplaceChar, 4, 2, vbTextCompare)

' Returns "XXpXXDXXD"
MyPos = Replace(ReplaceString, FindChar, ReplaceChar, 4, 2, vbBinaryCompare)
```

### 5-11-15: Right Function

Returns a String containing a specified number of characters from the right side of a string.

Right(*string*, *length*)

Part	Description
<i>string</i>	Required. String expression from which the rightmost characters are returned. If <i>string</i> contains Null, Null is returned.
<i>length</i>	Required; Long. Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in <i>string</i> , the entire string is returned.

### 5-11-16: Space Function

Returns a String consisting of the specified number of spaces.

Space(*number*)

### 5-11-17: Split Function

Returns a zero-based, one-dimensional array containing a specified number of substrings.

Split(*expression*[, *delimiter*[, *limit*[, *compare*]])

Part	Description
<i>expression</i>	Required. String expression containing substrings and delimiters. If <i>expression</i> is a zero-length string(""), Split returns an empty array, that is, an array with no elements and no data.
<i>delimiter</i>	Optional. String character used to identify substring limits. If omitted, the space character (" ") is assumed to be the delimiter. If <i>delimiter</i> is a zero-length string, a single-element array containing the entire <i>expression</i> string is returned.
<i>limit</i>	Optional. Number of substrings to be returned; -1 indicates that all substrings are returned.
<i>compare</i>	Optional VbCompareMethod. Numeric value indicating the kind of comparison to use when evaluating substrings.

### 5-11-18: StrConv Function

Returns a String converted as specified.

StrConv(*string*, *conversion*, [*LCID*])

Part	Description												
<i>string</i>	Required. String expression to be converted.												
<i>conversion</i>	Required. Integer. The sum of values specifying the type of conversion to perform.												
	<table border="1"><thead><tr><th>Constant</th><th>Value</th><th>Description</th></tr></thead><tbody><tr><td>vbUpperCase</td><td>1</td><td>Converts the string to uppercase characters.</td></tr><tr><td>vbLowerCase</td><td>2</td><td>Converts the string to lowercase characters.</td></tr><tr><td>vbProperCase</td><td>3</td><td>Converts the first letter of every word in string to uppercase.</td></tr></tbody></table>	Constant	Value	Description	vbUpperCase	1	Converts the string to uppercase characters.	vbLowerCase	2	Converts the string to lowercase characters.	vbProperCase	3	Converts the first letter of every word in string to uppercase.
	Constant	Value	Description										
	vbUpperCase	1	Converts the string to uppercase characters.										
vbLowerCase	2	Converts the string to lowercase characters.											
vbProperCase	3	Converts the first letter of every word in string to uppercase.											
<i>LCID</i> (Advanced)	Optional. The LocaleID, if different than the system LocaleID.												

### 5-11-19: String Function

Returns a String containing a repeating character string of the length specified.

String(*number*, *character*)

Part	Description
<i>number</i>	Required; Long. Length of the returned string. If <i>number</i> contains Null, Null is returned.
<i>character</i>	Required; Variant. Character code specifying the character or string expression whose first character is used to build the return string. If <i>character</i> contains Null, Null is returned.

### 5-11-20: StrReverse Function

Returns a string in which the character order of a specified string is reversed.

StrReverse(*expression*)

### 5-11-21: UCase Function

Returns a Variant (String) containing the specified string, converted to uppercase.

UCase(*string*)

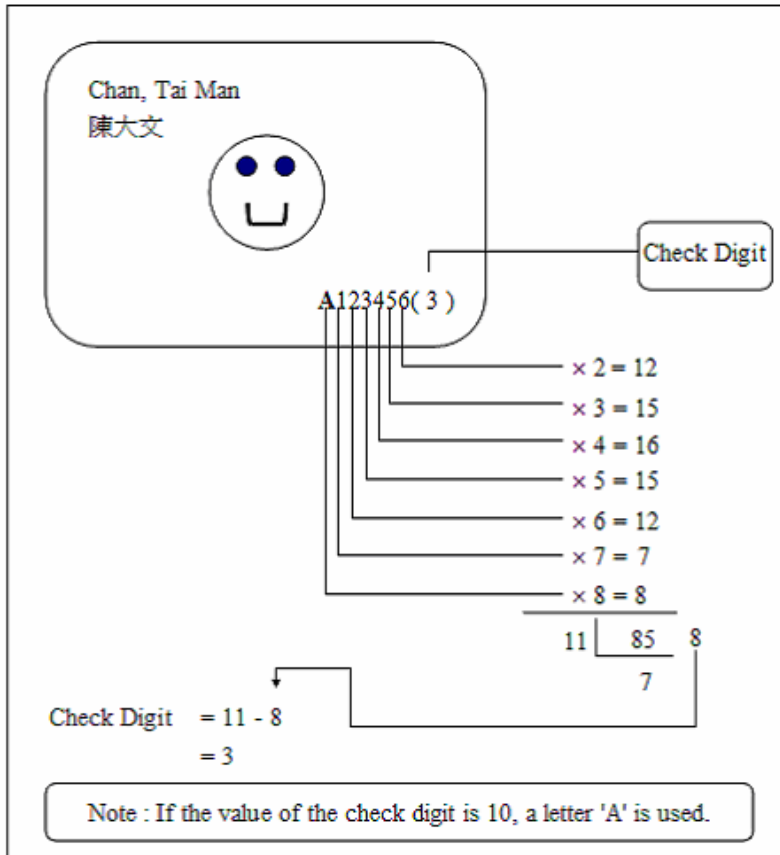
### Let's try! Corner

Create a function to validate a HKID number.

Function `isValidHKID(hkid As String) As Boolean`

The number in the bracket of an HKID is the check digit. It is calculated in the following way:

- 1 For the first letter, assign A with number 1, B with number 2, and so on.
- 2 Calculate the sum: 1<sup>st</sup> digit  $\times$  8 + 2<sup>nd</sup> digit  $\times$  7 + ... + 7<sup>th</sup> digit  $\times$  2
- 3 Divide the sum by 11 and get the remainder
- 4 The check digit is given by 11 - remainder. If it is 10, the check digit is set to be A.



### Let's try! Corner

- 1 Create a function to add leading zeros to a number so that the length of the resulting string is 4. For example, the number 12 results in "0012".
- 2 Create a sub to download the past stock data of a specific stock from Yahoo! HK finance website. You can get the file from "http://ichart.yahoo.com/table.csv?s=" + the 4-digit stock code + ".HK", using the **Open** method of the **Workbooks** collection. Add the data in this workbook to a new worksheet in an existing workbook, and then delete this workbook.
- 3 Using similar codes from section 5-8-50, add the codes to calculate the daily percentage gain to your sub. No need to sort the data.
- 4 In another sub, redo step 2-3 to download the HSI data (from <http://ichart.yahoo.com/table.csv?s=%5EHSI>) and calculate the daily gain.
- 5 Calculate the correlation between the returns of the Hang Seng Index and those of its constituent stocks, using the sheets you have created in steps 2-4, and output the results to a worksheet. You may first put the stock codes of the constituent stocks in a separated sheet. To find the stock codes, visit <http://hk.finance.yahoo.com/q/cp?s=%5EHSI>. (Another option: you may get the codes dynamically through the excel file in <http://hk.finance.yahoo.com/d/quotes.csv?s=@%5EHSI>)

## **5-12: Date and Time Functions**

Here are some variables and functions stated in `DateTime` module.

### **5-12-1: Date Variable**

This holds the system date. The type is `Date`. To change it into a string in the form *mm-dd-yyyy*, use the type-cast operator `$`, i.e, `Date$`.

Note: The format of the return value of this variable, as with most other return values with type `Date`, is determined by the setting in the Control Panel.

Example:

```
'Suppose today is 7/18/2008
MsgBox Date 'Outputs 7/18/2008
MsgBox Date$ 'Outputs 07-18-2008
```

### **5-12-2: Now Variable**

This holds the current date and time with type `Date`. Read only.

Example:

```
'Suppose today is 7/18/2008, now is 8:00:00 PM
MsgBox Now 'Outputs 7/18/2008 8:00:00 PM
```

### **5-12-3: Time Variable**

This holds the system time. The type is `Date`.

Example:

```
'Suppose now is 8:00:00 PM
MsgBox Time 'Outputs 8:00:00 PM
MsgBox Time$ 'Outputs 20:00:00
```

### **5-12-4: Timer Variable**

Returns a `Single` representing the number of seconds elapsed since midnight.

Example:

This example uses the `Timer` function to pause the application. The example also uses `DoEvents` to yield to other processes during the pause.

```

Dim PauseTime, Start, Finish, TotalTime
If (MsgBox("Press Yes to pause for 3 seconds", 4)) = vbYes Then
    PauseTime = 3      ' Set duration.
    Start = Timer      ' Set start time.
    Do While Timer < Start + PauseTime
        DoEvents      ' Yield to other processes.
    Loop
    Finish = Timer     ' Set end time.
    TotalTime = Finish - Start      ' Calculate total time.
    MsgBox "Paused for " & TotalTime & " seconds"
End If

```

\* Note: You cannot run this code at midnight.

### 5-12-5: DateAdd Function

Returns a Date containing a date to which a specified time interval has been added.

DateAdd(*interval*, *number*, *date*)

Part	Description																		
<i>interval</i>	Required. String expression that is the interval of time you want to add. <table border="1"> <thead> <tr> <th>Setting</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>yyyy</td> <td>Year</td> </tr> <tr> <td>q</td> <td>Quarter</td> </tr> <tr> <td>m</td> <td>Month</td> </tr> <tr> <td>y, d, w</td> <td>Day</td> </tr> <tr> <td>ww</td> <td>Week</td> </tr> <tr> <td>h</td> <td>Hour</td> </tr> <tr> <td>n</td> <td>Minute</td> </tr> <tr> <td>s</td> <td>Second</td> </tr> </tbody> </table>	Setting	Description	yyyy	Year	q	Quarter	m	Month	y, d, w	Day	ww	Week	h	Hour	n	Minute	s	Second
Setting	Description																		
yyyy	Year																		
q	Quarter																		
m	Month																		
y, d, w	Day																		
ww	Week																		
h	Hour																		
n	Minute																		
s	Second																		
<i>number</i>	Required. Numeric expression that is the number of intervals you want to add. It can be positive (to get dates in the future) or negative (to get dates in the past).																		
<i>date</i>	Required. Date or literal representing date to which the interval is added.																		

Example:

```

Dim FirstDate As Date, IntervalType As String, Number As Integer, Msg As String
IntervalType = "m"      ' "m" specifies months as interval.
FirstDate = InputBox("Enter a date")
Number = InputBox("Enter number of months to add")
Msg = "New date: " & DateAdd(IntervalType, Number, FirstDate)
MsgBox Msg

```

### 5-12-6: DateDiff Function

Returns a Long specifying the number of time intervals between two specified dates.

DateDiff(*interval*, *date1*, *date2*[, *firstdayofweek*[, *firstweekofyear*]])

Part	Description																											
<i>interval</i>	Required. String expression that is the interval of time you use to calculate the difference between <i>date1</i> and <i>date2</i> . See the DateAdd function for the list of possible expressions.																											
<i>date1</i> , <i>date2</i>	Required; Date. Two dates you want to use in the calculation.																											
<i>firstdayofweek</i>	Optional. A constant that specifies the first day of the week. If not specified, Sunday is assumed. <table border="1"> <thead> <tr> <th>Constant</th> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>vbUseSystem</td> <td>0</td> <td>Use the NLS API setting.</td> </tr> <tr> <td>vbSunday</td> <td>1</td> <td>Sunday (default)</td> </tr> <tr> <td>vbMonday</td> <td>2</td> <td>Monday</td> </tr> <tr> <td>vbTuesday</td> <td>3</td> <td>Tuesday</td> </tr> <tr> <td>vbWednesday</td> <td>4</td> <td>Wednesday</td> </tr> <tr> <td>vbThursday</td> <td>5</td> <td>Thursday</td> </tr> <tr> <td>vbFriday</td> <td>6</td> <td>Friday</td> </tr> <tr> <td>vbSaturday</td> <td>7</td> <td>Saturday</td> </tr> </tbody> </table>	Constant	Value	Description	vbUseSystem	0	Use the NLS API setting.	vbSunday	1	Sunday (default)	vbMonday	2	Monday	vbTuesday	3	Tuesday	vbWednesday	4	Wednesday	vbThursday	5	Thursday	vbFriday	6	Friday	vbSaturday	7	Saturday
Constant	Value	Description																										
vbUseSystem	0	Use the NLS API setting.																										
vbSunday	1	Sunday (default)																										
vbMonday	2	Monday																										
vbTuesday	3	Tuesday																										
vbWednesday	4	Wednesday																										
vbThursday	5	Thursday																										
vbFriday	6	Friday																										
vbSaturday	7	Saturday																										
<i>firstweekofyear</i>	Optional. A constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs. <table border="1"> <thead> <tr> <th>Constant</th> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>vbUseSystem</td> <td>0</td> <td>Use the NLS API setting.</td> </tr> <tr> <td>vbFirstJan1</td> <td>1</td> <td>Start with week in which January 1 occurs (default).</td> </tr> <tr> <td>vbFirstFourDays</td> <td>2</td> <td>Start with the first week that has at least four days in the new year.</td> </tr> <tr> <td>vbFirstFullWeek</td> <td>3</td> <td>Start with first full week of the year.</td> </tr> </tbody> </table>	Constant	Value	Description	vbUseSystem	0	Use the NLS API setting.	vbFirstJan1	1	Start with week in which January 1 occurs (default).	vbFirstFourDays	2	Start with the first week that has at least four days in the new year.	vbFirstFullWeek	3	Start with first full week of the year.												
Constant	Value	Description																										
vbUseSystem	0	Use the NLS API setting.																										
vbFirstJan1	1	Start with week in which January 1 occurs (default).																										
vbFirstFourDays	2	Start with the first week that has at least four days in the new year.																										
vbFirstFullWeek	3	Start with first full week of the year.																										

Example:

```
Dim TheDate As Date, Msg As String
TheDate = InputBox("Enter a date")
Msg = "Days from today: " & DateDiff("d", Now, TheDate)
MsgBox Msg
```

### 5-12-7: DatePart Function

Returns a Variant (Integer) containing the specified part of a given date.

DatePart(*interval*, *date*[,*firstdayofweek*[, *firstweekofyear*]])

Part	Description
<i>interval</i>	Required. String expression that is the interval of time you want to return.
<i>date</i>	Required. Date value that you want to evaluate.
<i>firstdayofweek</i>	Optional. A constant that specifies the first day of the week. If not specified, Sunday is assumed.
<i>firstweekofyear</i>	Optional. A constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs.

Example:

```
Dim TheDate As Date, Msg As String
TheDate = InputBox("Enter a date:")
Msg = "Quarter: " & DatePart("q", TheDate)
MsgBox Msg
```

### **5-12-8: DateSerial Function**

Creates a `Date` value for a specified year, month, and day.

```
DateSerial(year, month, day)
```

### **5-12-9: DateValue Function**

Creates a `Date` value representing the inputted String expression of a date.

```
DateValue(date)
```

Example:

```
MyDate = DateValue("February 12, 1969")
```

### **5-12-10: Day Function, Hour Function, Minute Function, Month Function, Second Function, Year Function**

Returns an Integer representing the day of the month, the hour, the minute, the month, the second and the year, respectively.

```
Day(date)
Hour(time)
Minute(time)
Month(date)
Second(time)
Year(date)
```

### **5-12-11: TimeSerial Function**

Returns a `Date` containing the time for a specific hour, minute, and second.

```
TimeSerial(hour, minute, second)
```

### **5-12-12: TimeValue Function**

Creates a `Date` value from a String representation of a time.

```
TimeValue(time)
```

Example:

```
MyTime = TimeValue("4:35:17 PM")
```

### 5-12-13: Weekday Function

Returns a Integer containing a whole number representing the day of the week.

`Weekday(date, [firstdayofweek])`

<b>Part</b>	<b>Description</b>
<i>date</i>	Required. Variant, numeric expression, string expression, or any combination, that can represent a date. If <i>date</i> contains Null, Null is returned.
<i>firstdayofweek</i>	Optional. A constant that specifies the first day of the week. If not specified, <code>vbSunday</code> is assumed. See <code>DateDiff</code> function for a list of possible values for this argument.

Example:

```
Dim MyDate, MyWeekDay
MyDate = #February 12, 1969#      ' Assign a date.
MyWeekDay = Weekday(MyDate)      ' MyWeekDay contains 4 because
                                  ' MyDate represents a Wednesday.
```

### 5-13: Cell Formatting

You have already learnt some cell formatting techniques through the `Range` object. For more complicated formatting techniques, we have to do so through mainly `Borders` collections, `Characters` objects, `ColorFormat` objects, `Font` objects, and `Interior` objects. You can look through these classes through the object browser.

Remember that we can record macros to obtain VBA codes. So if we want a specific format of a cell, we can easily do so by recording the format. For example if we want a yellow background cell, just record the action while we are adding a yellow background to a cell.

#### Let's try! Corner

Using the HSI table, make the header of the table bold-font and with a blue background. Make the background of the odd rows yellow and even rows pink.

**Let's try! Corner**

We want to create an amortization schedule in Excel. Practically this can be done by Excel functions, but as an exercise we use VBA to generate the table.

First there is a bit recap for an amortization schedule.

We have a loan of amount  $L$  which needs to be repaid with equal amount  $R$  at the end of each year for  $n$  years. The interest rate on the loan is  $i$  per annum. With  $L$ ,  $n$  and  $i$  given, we can then calculate  $R$ . Also the balance at the end of year  $t$ ,  $B_t$ , the interest paid for year  $t$ ,  $I_t$ , and the principal repaid for year  $t$ ,  $P_t$ , can also be calculated. These equations help us solve all the unknowns.

$$R = \frac{L}{\text{PVIFA}(i;n)}, \text{ where } \text{PVIFA}(i;n) = a_{\overline{n}|i} = \frac{1 - (1+i)^{-n}}{i}$$

$$B_t = \begin{cases} B_{t-1} - P_t & t > 0 \\ L & t = 0 \end{cases}$$

$$I_t = B_{t-1}i$$

$$P_t = R - I_t$$

We then have recursive relationships with the unknowns. First we determine the balance of the previous year, we then calculate the interest paid for the current year and consequently the principal repaid. Finally we get the balance of the current year.

- 1 Use 3 input boxes or a user form to ask for user's input of  $L$ ,  $n$  and  $i$ .
- 2 Write your program so that it generates the following sample table. You can add your own formatting to the table so that it looks nicer.

	A	B	C	D	E
1	Input				
2	L = 1000				
3	n = 20				
4	i = 0.06				
5					
6	Output				
7	t	R	I_t	P_t	B_t
8	0				\$1,000.0000
9	1	\$87.1846	\$60.0000	\$27.1846	\$972.8154
10	2	\$87.1846	\$58.3689	\$28.8156	\$943.9998
11	3	\$87.1846	\$56.6400	\$30.5446	\$913.4552
12	4	\$87.1846	\$54.8073	\$32.3772	\$881.0780
13	5	\$87.1846	\$52.8647	\$34.3199	\$846.7581
14	6	\$87.1846	\$50.8055	\$36.3791	\$810.3791
15	7	\$87.1846	\$48.6227	\$38.5618	\$771.8172
16	8	\$87.1846	\$46.3090	\$40.8755	\$730.9417
17	9	\$87.1846	\$43.8565	\$43.3281	\$687.6137
18	10	\$87.1846	\$41.2568	\$45.9277	\$641.6859
19	11	\$87.1846	\$38.5019	\$48.6824	\$593.0035

## 5-14: File Input and Output (Advanced Topic)

Sometimes you do not want to simply import a file into Excel, or simply save a workbook. You may want to do some data manipulation before saving it. You can do so with custom file input and output (File I/O).

### 5-14-1: Open a File

You must open a file before any I/O operation can be performed on it. The `Open` statement lets you open a file for input and/or output.

There are three types of `Open` statements. The three statements are designed for sequential file I/O, random file I/O and binary file I/O. For basic file I/O with text data, use sequential one. If the data are organized in a way that each record is of fixed length, use the random one. Finally, if the data contains binary data (e.g. an image file), use the binary mode.

Here we only learn the sequential file I/O. For other two methods see the online help:

[http://msdn.microsoft.com/en-us/library/aa733671\(VS.60\).aspx](http://msdn.microsoft.com/en-us/library/aa733671(VS.60).aspx) (random file I/O)

[http://msdn.microsoft.com/en-us/library/aa231223\(VS.60\).aspx](http://msdn.microsoft.com/en-us/library/aa231223(VS.60).aspx) (binary file I/O)

When you open a file for sequential access, you open it to perform one of the following operations:

- Input characters from a file (Input)
- Output characters to a file (Output). If you open the file with this mode, all existing data in the file will be destroyed.
- Append characters to a file (Append). Existing data will not be destroyed; new data will be stored at the end of this file.

To open a file for sequential access, use the following syntax for the `Open` statement:

```
Open pathname For [Input | Output | Append] As [#]filenumber
```

Note:

- *filenumber* is an integer in the range 1 to 511, inclusive used to represent the file. Use the `FreeFile` function to obtain the next available file number.
- When you open a sequential file for Input, the file must already exist; otherwise, an error occurs. When you try to open a nonexistent file for Output or Append, however, the `Open` statement creates the file first and then opens it.
- After opening a file for an Input, Output, or Append operation, you must close it, using the `Close` statement, before reopening it for another type of operation.

Example:

```
filenumber = FreeFile  
Open "C:\file1.txt" For Input As #filenumber
```

### 5-14-2: Read from a File

To retrieve the contents of a text file, open the file for sequential Input. Then use the `Line Input #` statement to copy the file into program variables.

```
Line Input #filenumber, varname
```

Each call to this statement writes a single line of contents in the file specified by *filenumber* into *varname*.

Example of copying data into variable:

```
Do Until EOF(filenumber)
  Line Input #filenumber, nextLine
  linesFromFile = linesFromFile + nextLine + vbCrLf
Loop
```

Although `Line Input #` recognizes the end of a line when it comes to the new line characters (`vbCrLf`), it does not include the new line characters when it reads the line into the variable. If you want to retain the new line characters, your code must add it.

There are two other methods of reading a file: using the `Input` function; and using the `Input #` statement. See the online help for these two methods.

### **5-14-3: Writing to a File**

To store the contents of variables in a sequential file, open it for sequential `Output` or `Append`, and then use the `Print #` statement.

```
Print #filenumber, [outputlist]
```

This writes the expression specified in the *outputlist* to the file, with the new line characters appended to the expression.

Example:

```
For Each str In strArray
  Print #filenumber, str
Next str
```

You can also use the `Write #` statement, which you can learn in the online help.

### **5-14-4: Close the File**

Use the `close` statement to close the file. Otherwise you may not be able to open the file next time you run your codes again.

```
Close #filenumber
```

Examples of opening a file for reading and writing:

```
Sub ch5_14_4_1()  
    Dim nextLine As String, linesFromFile As String, filenumber As Long  
    filenumber = FreeFile  
  
    Open "C:\test1.txt" For Input As #filenumber  
  
    Do Until EOF(filenumber)  
        Line Input #filenumber, nextLine  
        linesFromFile = linesFromFile + nextLine + vbCrLf  
    Loop  
    MsgBox linesFromFile  
  
    Close #filenumber  
End Sub
```

```
Sub ch5_14_4_2()  
    Dim strArray(5) As String, str, filenumber As Long  
    strArray(0) = "Albert"  
    strArray(1) = "Andrew"  
    strArray(2) = "Andy"  
    strArray(3) = "Betty"  
    strArray(4) = "Candy"  
    strArray(5) = "Carmen"  
  
    filenumber = FreeFile  
    Open "C:\test1.txt" For Output As #filenumber  
  
    For Each str In strArray  
        Print #filenumber, str  
    Next str  
  
    Close #filenumber  
End Sub
```

#### **5-14-5: FileSystem module**

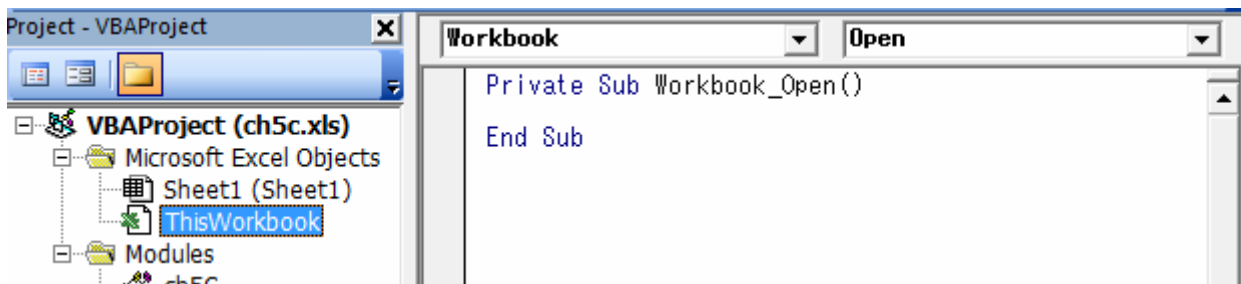
The FileSystem module provides several useful functions for file access. Explore the module yourself through the object browser.

#### **5-15: Excel Events (Advanced Topic)**

In Excel, there are a lot of events you can program. Here, we will only discuss briefly some Workbook events and Worksheet events.

#### **5-15-1: Workbook Events**

You will program your events in the ThisWorkbook object module. As usual, you can automatically create an empty event handling procedure with the two list boxes at the top of the code editing area.



For example, if you want to add `Workbook_Open` event handling procedure, you can choose `Workbook` in the left list and `Open` in the right list.

Now, here is a list of useful `Workbook` events you may want to explore in the object browser.

Event	Event Handling Procedure	Description
Activate	<code>Workbook_Activate ()</code>	Occurs when a workbook is activated.
BeforeClose	<code>Workbook_BeforeClose (Cancel As Boolean)</code>	Occurs before the workbook closes. If the workbook has been changed, this event occurs before the user is asked to save changes. <ul style="list-style-type: none"> <li>● <i>Cancel</i>: False when the event occurs. If the event procedure sets this argument to True, the action is cancelled.</li> </ul>
BeforePrint	<code>Workbook_BeforePrint (Cancel As Boolean)</code>	Occurs before the workbook (or anything in it) is printed.
BeforeSave	<code>Workbook_BeforeSave (ByVal SaveAsUi As Boolean, Cancel As Boolean)</code>	Occurs before the workbook is saved. <ul style="list-style-type: none"> <li>● <i>SaveAsUi</i>: True if the Save As dialog box will be displayed.</li> </ul>
Deactivate	<code>Workbook_Deactivate ()</code>	Occurs when the workbook is deactivated.
NewSheet	<code>Workbook_NewSheet (ByVal Sh As Object)</code>	Occurs when a new sheet is created in the workbook. <ul style="list-style-type: none"> <li>● <i>Sh</i>: The new sheet. Can be a Worksheet or Chart object.</li> </ul>
Open	<code>Workbook_Open()</code>	Occurs when the workbook is opened.
SheetActivate	<code>Workbook_SheetActivate (ByVal Sh As Object)</code>	Occurs when any sheet is activated.
SheetBeforeDoubleClick	<code>Workbook_SheetBeforeDoubleClick (ByVal Sh As Object, ByVal Target As Range, ByVal Cancel As Boolean)</code>	Occurs when any worksheet is double-clicked, before the default double-click action. <ul style="list-style-type: none"> <li>● <i>Target</i>: The cell nearest to the mouse pointer when the double-click occurred.</li> </ul>
SheetBeforeRightClick	<code>Workbook_SheetBeforeRightClick (ByVal Sh As Object, ByVal Target As Range, ByVal Cancel As Boolean)</code>	Occurs when any worksheet is right-clicked, before the default right-click action.
SheetCalculate	<code>Workbook_SheetCalculate (ByVal Sh As Object)</code>	Occurs after any worksheet is recalculated or after any changed data is plotted on a chart.
SheetChange	<code>Workbook_SheetChange (ByVal Sh As Object, ByVal Source As Range)</code>	Occurs when cells in any worksheet are changed by the user or by an external link. <ul style="list-style-type: none"> <li>● <i>Source</i>: The changed range.</li> </ul>
SheetDeactivate	<code>Workbook_SheetDeactivate (ByVal Sh As Object)</code>	Occurs when any sheet is deactivated.
SheetSelectionChange	<code>Workbook_SheetSelectionChange (ByVal Sh As Object, ByVal Target As Range)</code>	Occurs when the selection changes on any worksheet. <ul style="list-style-type: none"> <li>● <i>Target</i>: The new selected range.</li> </ul>

### Let's try! Corner

Design an event handling procedure to disable right-clicking. When the user right-clicks a cell in the workbook, a message box appears showing that the user cannot right-click in this workbook.

### 5-15-2: Worksheet Events

You write event handling procedures of each worksheet in the corresponding sheet module. Here are some events of the Worksheet class.

<b>Event</b>	<b>Event Handling Procedure</b>	<b>Description</b>
Activate	Worksheet_Activate ()	Occurs when a worksheet is activated.
BeforeDouble-Click	Worksheet_BeforeDoubleClick (ByVal <i>Target</i> As Range, <i>Cancel</i> As Boolean)	Occurs when a worksheet is double-clicked, before the default double-click action.
BeforeRight-Click	Worksheet_BeforeRightClick (ByVal <i>Target</i> As Range, <i>Cancel</i> As Boolean)	Occurs when a worksheet is right-clicked, before the default right-click action.
Calculate	Worksheet_Calculate ()	Occurs after the worksheet is recalculated.
Change	Worksheet_Change (ByVal <i>Target</i> As Range)	Occurs when cells on the worksheet are changed by the user or by an external link.
Deactivate	Worksheet_Deactivate ()	Occurs when the chart, worksheet, or workbook is deactivated.
SelectionChange	Worksheet_SelectionChange (ByVal <i>Target</i> As Range)	Occurs when the selection changes on a worksheet.

### Exercise 5

- Write a program to make the background of a sheet “checked” like the following:

The image shows a screenshot of an Excel spreadsheet. The columns are labeled J through T, and the rows are numbered 1 through 20. The cells alternate between red and green in a checkerboard pattern, starting with a red cell in the top-left corner (J1).

- Write an event handling procedure to automatically change the column width of all columns to 2 when a new sheet is added.
- Ask the user to input a range of cells. Add a thin border to around all the cells in the range.
- Write a program to calculate the duration and convexity of a series of cashflows. The procedures to calculate them, in short, is first to treat the cashflows into a series of zero coupon bonds. Next we use the

fact that the duration of a zero coupon bond is its maturity, and the convexity is given by  $\frac{T \times (T + 1)}{(1 + i)^2}$ ,

where  $T$  is the maturity. Finally the duration and the convexity of the whole series of cashflows are respectively the weighted average of the duration and the convexity of each of the cashflow with the

price of each cashflow as its weight. Specifically, the weight is given by  $w_t = \frac{P_t(i)}{\sum_t P_t(i)}$ , where  $P_t(i)$  is

the price of the cashflow generated at time  $t$ , and the duration is given by  $\sum_t w_t D_t$ , and the convexity is

$\sum_t w_t C_t$ , with  $D_t$  and  $C_t$  being the duration and convexity of the cashflow at time  $t$  respectively.

The inputs will be inputted by the user in advance in a worksheet. You are going to produce a table similar to this one, with the cells in blue are inputted by the user.

	A	B	C	D	E	F	G	H
1	Input						Output	
2	i	0.05					Price	1243.117
3							Duration	7.327666
4	Time	Cashflow	PV	Duration	Convexity		Convexity	65.44737
5	(Total)	1800	1243.117	7.327666	65.44737			
6	0.25	20	19.75753	0.25	0.283447			
7	0.5	20	19.518	0.5	0.680272			
8	0.75	20	19.28138	0.75	1.190476			
9	1	20	19.04762	1	1.814059			
10	1.25	20	18.8167	1.25	2.55102			
11	1.5	20	18.58857	1.5	3.401361			
12	1.75	20	18.36322	1.75	4.365079			
13	2	20	18.14059	2	5.442177			
14	2.25	20	17.92066	2.25	6.632653			
15	2.5	20	17.7034	2.5	7.936508			
16	2.75	20	17.48878	2.75	9.353741			
17	3	20	17.27675	3	10.88435			
18	3.25	20	17.0673	3.25	12.52834			
19	3.5	20	16.86038	3.5	14.28571			
20	3.75	20	16.65598	3.75	16.15646			
21	4	20	16.45405	4	18.14059			

5. Using the data in <http://www.ssa.gov/OACT/STATS/table4c6.html>, create a life table for each of male and female. In the life table, you should reproduce the life expectancy  $e_x = E[K(x)] = \sum_{k=0}^{120-x} k {}_k p_x q_{x+k}$ .

Also, add  $A_x = E[v^{K(x)+1}] = \sum_{k=0}^{120-x} v^{k+1} {}_k p_x q_{x+k}$  and  $\ddot{a}_x = E[a_{\overline{K(x)|}}] = \sum_{k=0}^{120-x} v^k {}_k p_x$  using an interest rate  $i =$

5%. (Reference: [http://en.wikipedia.org/wiki/Life\\_table](http://en.wikipedia.org/wiki/Life_table);

[http://en.wikipedia.org/wiki/Actuarial\\_present\\_value](http://en.wikipedia.org/wiki/Actuarial_present_value))

6. Ask the user to input a stock code from an input box. Use the Yahoo! HK Finance page to download past data of the stock price. Then, calculate the 5-day, 30-day and 90-day moving average while use your code to filter only this year's data. Plot a graph including the stock price and the average lines using your code. (Try recording a macro while you are adding a graph.)
7. Design a calendar for a year. First, ask the user to input a year. Then insert 12 worksheets. With each worksheet, design a monthly calendar.
8. For all the constituent stocks of Hang Seng Index, search the days within the past 365 days where the percentage gain of the day is higher than 5%. Output all the records in one sheet, including the date, the name of the stock, the code of the stock, the closing price, and the percentage gain.