VBA में उपयोगकर्ता द्वारा परिभाषित फंक्शन (User defined functions in VBA)

उद्देश्य : इस पाठ के अन्त में आप निम्नलिखित कार्य करने योग्य होंगे :
- **उपयोगकर्ता द्वारा परिभाषित फंक्शन बनाना**
- **फंक्शन में byval और byref से मान पास करने का वर्णन करना**
- **फंक्शन के साथ एरे के उपयोग का वर्णन करना**
- **चर के दायरे का वर्णन करना**
- **पब्लिक और प्राइवेट एक्सेस स्पेसिफिएर (विनिर्देशक) का वर्णन।**

परिचय (Introduction)

एक्सेल विसुअल बेसिक में भी (जैसे अधिकांश प्रोग्रामिंग भाषाओं में होता है) निर्देशों का एक सेट होता है तो कोई निर्दिष्ट कार्य करने के लिए एक प्रोसीजर में निहित किये जाते हैं जो एक फंक्शन या सबरूटीन दोनों हो सकता है। VBA फंक्शन और सबरूटीन में मुख्या अंतर यह है कि फंक्शन सामान्यतः एक परिणाम लौटाता है जबकि सबरूटीन कुछ नहीं लौटाता है।

इसलिए, यदि आप एक ऐसा कार्य करना चाहते हैं जो एक परिणाम लौटाए (जैसे संख्याओं के समूह को जोड़ना), तो आप सामान्य रूप से फंक्शन का उपयोग करेंगे लेकिन यदि आप चाहते हैं कि कुछ निर्देश निष्पादित होकर कुछ एक्शन करें तो सबरूटीन को उपयोग करना होगा।

यूजर द्वारा परिभाषित फंक्शन (User Defined Functions)

Excel VBA के सबसे ताकतवर विशेषताओं में से एक है कि आप अपने खुद के फंक्शन या UDFs बना सकते हैं। एक UDF (यूजर डिफाइंड फंक्शन) बस एक फंक्शन है। जिसे आप अपने खुद के कार्यों के लिए परिभाषित करते हैं। UDFs को अक्सर कस्टम फंक्शन कहा जाता है। एक UDF वर्कबुक से जुड़े कोड मॉड्यूल में रह सकता है, इस मामले में यह हमेशा उपलब्ध रहेगा जब तक वह वर्कबुक खुली होगी। वैकल्पिक रूप से आप अपना स्वयं का ऐड-इन बना सकते हैं जिसमें एक या अधिक फंक्शन शामिल हैं जिसे आप Excel में इंस्टॉल कर सकते हैं। यहाँ उपयोगकर्ता द्वारा परिभाषित फंक्शन को किसी भी सेल या स्प्रेडशीट के बिल्ट-इन (अंतर्निर्मित) सूत्रों को दर्ज करते हैं।

कस्टम फंक्शन, मैक्रोज़ की तरह, विजुअल बेसिक फॉर एप्लीकेशन (VBA) प्रोग्रामिंग भाषा का उपयोग करते हैं। यह मैक्रो से महत्त्वपूर्ण तरीके से अलग है, पहला सब प्रोसीजर के बजाए फंक्शन प्रोसीजर का उपयोग करते हैं। वे एक सब स्टेटमेंट के बजाए एक फंक्शन स्टेटमेंट से शुरू और एंड सब की बजाए एंड फंक्शन के साथ अंत होती है। दूसरा यह एक्शन लेने की बजाय गाड़ना करते हैं। कुछ प्रकार की स्टेटमेंट (जैसे वो स्टेटमेंट जो रेंज को चुनकर फॉर्मेट करती है) कस्टम फंक्शन के अंदर नहीं आती है।

एक साधारण फंक्शन ऐसा दिखेगा :

```
Function area()
Dim l, b
l = 10
b = 20
Debug.Print "area Is " & l * b
End Function
```

जब इमीडियेट विंडो से निष्पादित होगा तब यह फंक्शन एरिया प्रदर्शित करेगा।

वैकल्पि तौर पर यह फंक्शन अन्य सबरूटीन द्वारा परिभाषित किया जाता है।

```
Sub test_fn()
Call area
End Sub
```

प्रोसीजर के द्वारा मान रिटर्न करना (Returning a value from the procedures)

नीचे दिए उदाहरण में area() फंक्शन l*b की गड़ना कर रहा है।

जब सबरूटीन इस फंक्शन को कॉल कर रहा है इस मान को लौटाता है।

```
Sub test_fn()
Debug.Print "The function has returned the value " & area
End Sub
Function area()
Dim l, b, A
l = 10
b = 20
area = l * b
End Function
```

परिणाम यह होगा :The function has returned the value 200

फंक्शन में आर्ग्युमेंट पास करना (Passing Arguments to functions)

हम आर्ग्युमेंट दो तरह से पास कर सकते हैं :

1 बाय वैल्यू (ByVal): हम आर्ग्यूमेंट्स के वास्तविक मूल्य की प्रति पास करते हैं।

2 बाय रेफरेन्स (ByRef): हम आर्ग्यूमेंट के संदर्भ की प्रतिलिपि पास करते हैं।

बाय रेफरन्स मेथड आर्ग्यूमेंट को पास करने का डिफोल्ट तरीका है। यदि आप बाय रेफरन्स निर्दिष्ट नहीं कर रहे हैं तो यह बाय डिफ़ॉल्ट यह ही समझेगा। लेकिन आर्च प्रोग्रामिंग का अभ्यास है कि इसे लिखा जाये भले ही यह जरूरी न हो।

निम्नलिखित उदाहरण एक फंक्शन में चर पास करने की विधि दिखाता है byVal ।

```
Sub test_fn()
Dim a, b As Integer
a = 4
b = multiply(a)
Debug.Print "a is " & a
Debug.Print "The function has returned the value " & b
End Sub
Function multiply(ByVal a As Integer)
a = a * 10
multiply = a
End Function
```

इस प्रोग्राम का परिणाम होगा :

a is 4

फंक्शन मान 40 लौटाएगा।

a is 4

इस का अर्थ है कि चर का जो मान जो पास किया गया वह फन्कशन के कारण बाधित नहीं होता है।

निम्नलिखित उदाहरण दर्शाता है कि चर को फंक्शन बाय रेफरन्स में पास करने की विधि कैसे होती है :

```
Sub Test()
Dim A As Integer
A = 10
Debug.Print "The function has returned the value " & Modify(A)
Debug.Print "A is now  " & A
End Sub
Function Modify(ByRef A As Integer)
A = A * 2
  Modify = A
End Function
```

परिणाम यह होगा :

फंक्शन मान 20 रिटर्न करेगा।

A का मान अब 20 है।

वर्कशीट से यूजर डिफाइंड फंक्शन को कॉल करना (Calling a User Defined Function from Worksheet):

आप उसे र्देफिनेड फंक्शन को उसी तरह कॉल कर सकते हैं जैसे बिल्ट इन फंक्शन को किया जाता है। यह करने के लिए सेल में आर्ग्यूमेंट टाइप करें और फंक्शन का नाम टाइप करें जैसे आप संन्य एक्सेल एक्सेल फंक्शन के साथ करते हैं।

यूजर डिफाइंड फंक्शन में एरे पास करना (Passing Arrays to User Defined functions)

एक फ़ंक्शन एक इनुपुट पैरामीटर के रूप में एक सरणी को स्वीकार कर सकता है। ऐरे हमेशा रेफ़रन्स द्वारा पारित होते हैं (ByRef)। यदि ByVal ऐरे तो आपको एक कंपाइलर त्रुटि प्राप्त होगी। पास करने का प्रयास करते हैं इसका मतलब यह है कि कॉल किया गया प्रोसीजर यदि ऐरे पैरामीटर में कोई भी संशोधन करता है तो वह कॉल कर रहे प्रोसीजर में घोषित वास्तविक ऐरे पर भी हो जाएगा भी संशोधन को किया जाता है।

(यदि आपको ByVal ऐरे पास करने की आवश्यकता पड़ती है तो आपको वैरिएंट डाटा टाइप का उपयोग करना होगा)।

ऐरे को फंक्शन में पास करने का उदाहरण निमानुसार है :

```
Sub test()
Dim arr(1 To 10) As Integer
Dim i As Integer
'populates the array with the values 1 to 10
For i = 1 To 10
arr(i) = i
Next i
'call the function example 1 with arrIntegers as an input parameter
Call fn1(arr)
For i = 1 To 10
Debug.Print arr(i); Spc(2);
Next i
End Sub
'prints the values in arrIntegers to column A
Sub fn1(ByRef arr() As Integer)
Dim i As Integer
```

```
For i = LBound(arr) To UBound(arr)
  arr (i) = arr (i) * 2
  Cells (i,1) = arr (i)
Next i
End Sub
```

वेरिएबल्स का दायरा (Scope of variables)

स्कोप शब्द का उपयोग यह वर्णन करने के लिए किया जाता है कि एक चर को कैसे एक्सेस किया जा सकता है। यह इस पर निर्भर करता है, कि एक चर कहाँ और कैसे घोषित हुआ है। यह केवल एक ही प्रोसीजर के लिए, एक मॉड्यूल के भीतर सभी प्रोसीजर में, और इसी तरह, किसी प्रोजेक्ट या संबंधित प्रोजेक्टों के समूह के पदानुक्रम पर पहुंच योग्य हो सकता है। विसिबिलिटी शब्द, कभी-कभी स्कोप का वर्ण करने के लिए भी प्रयोग किया जाता है।

स्कोप के चार स्तर हैं :

- प्रोसीजर - स्तर का स्कोप
- मॉड्यूल - स्तर का स्कोप
- प्रोजेक्ट - स्तर का स्कोप
- वैश्विक - स्तर के स्कोप

Fig 1

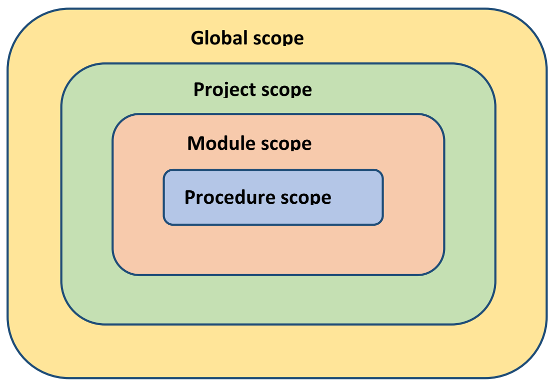


Fig 1 विभिन्न दायरे (scope) और उनके स्तर को दिखाता है।

प्रोसीजर (लोकल) स्कोप (Procedure (local) scope)

प्रोसीजर स्कोप में एक लोकल चर केवल उस प्रोसीजर के भीतर पहचाना जाता है जिसमें इसे घोषित किया जाता है। एक Dim या Static स्टेटमेंट के साथ एक लोकल चर घोषित किया जा सकता है।

जब डिम स्टेटमेंट के साथ एक स्थानीय चर घोषित किया जाता है, तो चर केवल तब तक अस्थित्व में रहा है जब तक कि प्रोसीजर जिसमें उसे घोषित किया जाता है उसका निष्पादन हो रहा हो। आमतौर पर, जब प्रोसीजर निष्पादन समाप्त हो जाता है, तो प्रोसीजर के स्थानीय चर के मान संरक्षित नहीं होते हैं, और उन चरों को आवंटित स्मृति हटा दी जाती है। अगली बार जब प्रोसीजर को निष्पादित किया जाता है, इसके सभी स्थानीय चरों को दोबारा से इनिशियलाइज़ किया जाता है।

उदाहरण के लिए, मॉड्यूल 1 कोड में निम्न सब राउटिन बनाया गया है।

```
Sub disp()
Dim s As string
s="hello"
MsgBox s
End Sub
```

मॉड्यूल 1 में सबरूटीन "disp" चलाएं और यह मेसेज बॉक्स में "Hello" संदेश प्रदर्शित करेगा।

अब, मॉड्यूज 1 से disp() सबरूटीन को कॉल करने के लिए शीट 1 कोड में निम्न सबरूटीन बनाया गया है।

```
Sub Button1_Click()
disp
End Sub
```

यह एक त्रुटि उत्पन्न उत्पन्न करेगा क्योंकि सबरूटीन disp() और वेरिएबल s मॉड्यूल 1 के लिए स्थानीय हैं और इन्हें कहीं और से एक्सेस नहीं किया जा सकता है।

स्टेटिक (Static):

स्टेटिक स्टेटमेंट के साथ घोषित एक स्थानीय चर अस्तित्व में रहता है जब पूरे समय विजुअल बेसिक चल रहा है। चर रीसेट होता है, जब निम्न में से कोई भी होता है :

- मैक्रो एक इनचाहे रन-टाइम त्रुटि उत्पन्न करता है।
- विजुअल बेसिक रूक गया है।
- आपने माइक्रोसॉफ्ट एक्सेल छोड़ दिया।
- आपने मॉड्यूल बदल दिया।

उदाहरण के लिए, FindTotal उदाहरण में, संचय चर प्रत्येक बार निष्पादित होने पर अपना मान बरकरार रखता है। पहली बार, मॉड्यूल चलाया जाता है, यदि आप नंबर 2, दर्ज करते हैं, तो संदेश बॉक्स मान "2" प्रदर्शित करेगा। अगली बार मॉड्यूल चलाया जाएगा और यदि मान 3 दर्ज किया गया है, तो संदेश बॉक्स प्रदर्शित होगा और कल मूल्य 5 होना चाहिए।

```
Sub FindTotal()
Static Total
Dim n as integer
n =InputBox("Enter a number: ")
Total = Total + n
MsgBox "The total is " &n
```

```
End Sub
```

मॉड्यूल स्कोप (Module scope)

एक चर जो मॉड्यूल शीट पर सभी प्रोसीजर के बीच पहचाना जाता है उसे "मॉड्यूल-लेवल" चर कहा जाता है। मॉड्यूल-स्तरीय चर उस मॉड्यूल की सभी प्रोसीजर के लिए उपलब्ध है, लेकिन यह अन्य मॉड्यूल में प्रोसीजर के लिए उपलब्ध नहीं है। एक मॉड्यूल-स्तरीय चर अस्तित्व में रहता है, जब तक कि Visual Basic चल रहा न हो, जब तक कि जिस मॉड्यूल में इसे घोषित नहीं किया जाता है, संपादित किया जाता है। मॉड्यूल-स्तरीय चर को पहली प्रोसीजर परिभाषा के ऊपर मॉड्यूल के शीर्ष पर Dim या Private कथन के साथ घोषित किया जा सकता है।

मॉड्यूल स्तर पर, Dim और Private के बीच कोई अंतर नहीं है। ध्यान दें कि मॉड्यूल-स्तरीय चर को किसी प्रोसीजर के भीतर घोषित नहीं किया जा सकता है।

नोट, यदि आप मॉड्यूल-स्तरीय चर के लिए Dim के बजाय private का उपयोग करते हैं, तो आपका कोड पढ़ना आसान हो सकता है। (यानी, यदि आप केवल स्थानीय चर के लिए Dim का उपयोग करते हैं, और मॉड्यूल-स्तरीय चर के लिए Private किसी विशेष चर का दायरा, अधिक स्पष्ट होगा)।

निम्नलिखित उदाहरण में, मॉड्यूल स्तर पर दो चर, A और B, घोषित किए गए हैं। ये दो चर मॉड्यूल शीट पर किसी भी प्रोसीजर के लिए उपलब्ध है। तीसरा चर, C, जिसे उदाहरण 3 मैक्रो में घोषित किया गया है, एक स्थानीय चर है और केवल उस प्रोसीजर के लिए उपलब्ध हैं।

ध्यान दें कि Test 4 में, जब मैक्रो चर C, का उपयोग करने का प्रयास करता है, तो संदेश बॉक्स खाली होता है। मेसेज बॉक्स खाली है क्योंकि C एक स्थानीय चर है और Test4, के लिए उपलब्ध नहीं है, जबकि चर A और B उपलब्ध हैं।

```
Dim A As Integer      ' Module-level variable.
Private B As Integer   ' Module-level variable.
Sub Test1()
A = 10
B = A * 10
End Sub
Sub Test2()
MsgBox "The value of A is " & A
MsgBox "The value of B is " & B
End Sub
Sub Test3()
Dim C As Integer   ' Local variable.
C = A + B
MsgBox "The value of C is " & C
End Sub
Sub Test4()
MsgBox A
MsgBox B
MsgBox C
 ' The message box is blank since C is a local variable.
End Sub
```

प्रोजेक्ट स्कोप (Project Scope)

प्रोजेक्ट स्कोप चर वे हैं जो Public कीवर्ड का उपयोग करके घोषित किए जाते हैं। इन चरों को प्रोजेक्ट में किसी भी मॉड्यूल में किसी भी से एक्सेस किया जाता है। एक्सेल में, एक प्रोजेक्ट कोड के सभी मॉड्यूल, उपयोगतार्ता फॉर्म, क्लास मॉड्यूल, और ऑब्जेक्ट मॉड्यूल (उदाहरण के लिए ThisWorkbook और Sheet1) है जो वर्कबुक में निहित हैं।

प्रोजेक्ट में कहीं से भी एक चर को एक्सेल के लिए, आपको चर की घोषणा में Public कीवर्ड का उपयोग करना होगा। हालांकि, यह वेरिएबल को प्रोजेक्ट को संदर्भित करने वाली किसी भी अन्य प्रोजेक्ट के लिए चर को एक्सेस करने योग्य बनाता है। यदि आप प्रोजेक्ट के भीतर कहीं भी एक चर को एक्सेस कर सकते हैं, तो आपको मॉड्यूल में पहली पंक्ति के रूप में आप्शन प्राइवेट मॉड्यूल का उपयोग करने की आवश्यकता है। (ऊपर और बाहर किसी भी चर की घोषणा के या प्रोसीजर के)। यह विकल्प मॉड्यूल में सब कुछ केवल प्रोजेक्ट के भीतर से एक्सेसिबल बनाता है। प्रोजेक्ट चर जो अन्य प्रोजेक्टओं के लिए एक्सेस नहीं होनी चाहिए उसे मॉड्यूल में घोषित किया जाना चाहिए जिसमें आप्शन प्राइवेट मॉड्यूल डायरेक्टिव है। वेरिएबल्स जिन्हें अन्य प्रोजेक्ट के लिए पहुँचने योग्य किया जाना चाहिए उन्हें एक अलग मॉड्यूल में घोषित किया जाना चाहिए जो आप्शन प्राइवेट मॉड्यूल निर्देश का उपयोग नहीं करता है। हालांकि, दोनों मामलों में, आपको Public keyword का उपयोग करने की आवश्यकता है।

वैश्विक स्कोप (Global Scope)

ग्लोबल स्कोप वेरिएबल वे हैं जो प्रोजेक्ट में कहीं से भी पहुँचने योग्य हैं जो उन्हें घोषित करता है और साथ ही साथ किसी भी अन्य प्रोजेक्ट को संदर्भित करता है जो पहले प्रोजेक्ट का संदर्भ देता है। वैश्विक दायरे के साथ एक चर घोषित करने के लिए, आपको उस मॉड्यूल में Public keyword का उपयोग करके घोषित करने की आवश्यकता है जो आप्शन प्राइवेट मॉड्यूल निर्देश का उपयोग नहीं करता है। किसी अन्य प्रोजेक्ट में चरों तक पहुँचने के लिए, आप बस चर के नाम का उपयोग कर सकते हैं। यदि, हांलाकि यह संभव है कि कॉलिंग प्रोजेक्ट में एक ही नाम से वेरिएबल हो, तो आपको प्रोजेक्ट नाम के साथ वेरिएबल नाम को उपसर्ग करना होगा। उदाहरण के लिए, यदि Project1 x नाम वैश्विक चर घोषित करता है, और Project2, Project1 को रेफेर करता हो तो, Project2 को कोड, निम्न कोड की पंक्तियों में से किसी एक के साथ x तक पहुँच सकता है :

```
x = 78
```

Project1.x = 78

यदि, प्रोजेक्ट 1 और प्रोजेक्ट 2 दोनों में कम से कम प्रोजेक्ट स्कोप के साथ चर हैं, तो आपको वेरिएबल के साथ प्रोजेक्ट नाम शामिल करना होगा। स्पष्टता और रखरखाव के लिए, आपको किसी अन्य प्रोजेक्ट में घोषित वेरिएबल तक पहुँचने पर हमेशा प्रोजेक्ट का नाम शामिल करना चाहिए।

यहॉ तक कि, यदि यह आवश्यक नही है, तो यह कोड को और अधिक पठनीय और रखरखाव योग्य बनाता है। कुछ चर प्रोजेक्ट को वैश्विक, स्कोप देने की कोई जरूरत नहीं है, लेकिन उसी मॉड्यूल में अन्य को वैश्विक स्कोप देना होता है। वैश्विक बनाम प्रोजेक्ट स्कोप केवल मॉड्यूल स्तर पर संभाला जाता है, न कि चर-स्तर पर।

एक्सेस विनिर्देशक (The Access Specifiers)

ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग में तकनीकों में सक एक encapsulation है। यह क्लास में डेटा छिपाने से संबंधित है और उन्हें केवल अपनी मेथड के माध्यम से उपलब्ध करा रहा है। OOPS को लागू करने वाली अधिकांश प्रोग्रामिंग भाषाएं आपको तथाकथित एक्सेस संशोधक के माध्यम से क्लासों, मेथड और फील्ड तक पहुँच नियंत्रित करने की अनुमति देती है। क्लास, कन्स्ट्रक्टर, मेथड और फील्ड तक पहुँच को एक्सेस मॉडिफायर्स को उपयोग करके विनियमित किया जाता है यानी एक क्लास नियंत्रित कर सकती है कि कौन सी जानकारी या डेटा अन्य क्लासों द्वारा एक्सेस किया जा सकता है। VBA एक्सेस विनिर्देशक है :

1 प्राइवेट (Private)

2 पब्लिक (Public)

मॉड्यूल के अंदर सभी कोड और मॉड्यूल के बाहर के सभी कोड तक एक Public प्रक्रिया तक पहुँचा जा सकता है, अनिवार्य रूप से इसे वैश्विक बनाते हैं। एक VBA Private Sub केवल मॉड्यूल में कहीं से भी बुलाया जा सकता है जिसमें यह रहा है। Public Sub in the objects, ThisWorkbook, ThisDocument, Sheet1, इत्यादि किसी प्रोजेक्ट में कहीं से भी कॉल नहीं किया जा सकता है। हांलाकि, यदि आप Public Keyword के साथ मॉड्यूल लेवल वेरिएबल घोषित करते हैं तो इसे प्रोजेक्ट में कहीं भी इस्तेमाल किया जा सकता है और इसका मूल्य बरकरार रखा जा सकता है।

यदि आप अपनी घोषणा में private शब्द को निजी रूप से बहिष्कृत करतते हैं तो डिफॉल्ट रूप से प्रोजीसर public होती है। इसलिए Sub MySub() और Public Sub MySub() एक जैसे होंगे।

Public [variable] का अर्थ है कि चर को बाहरी मॉड्यूल में सबरूटीन द्वारा उपयोग या उपयोग किया जा सकता है। इन चरों को एक सबरूटीन (आमतौर पर आपके मॉड्यूल के बहुत ऊपर) के बाहर घोषित करता है और आप उस मान को एक अलग मॉड्यूल में संग्रहित किसी अन्य सबरूटीन पर पास करना चाहते हैं।

एक प्राइवेट प्रोसीजर केवल मौजूदा मॉड्यूल के लिए उपलब्ध है। इसे किसी अन्य मॉड्यूल से या Excel workbook से एक्सेल नहीं किया जा सकता है। Private Sub स्कोप सेट करता है ताकि बाहरी मॉड्यूल से सबरूटीन उस विशेष सबरूटीन को कॉल न कर सकें। इसका मतलब है कि मॉड्यूल 1 में एक Sub मॉड्यूल 2 में एक प्राइवेट सब आरंभ करने के लिए कॉल मेर्थड का उपयोग नहीं कर सकता है।

Private [variable] का अर्थ है कि चर को अन्य मॉड्यूल में सबरूटीन द्वारा उपयोग या उपयोग नहीं किया जा सकता है। उपयोग करने के लिए, इन चरों को एक सबरूटीन (आमतौर पर आपके मॉड्यूल के बहुत ऊपर) के बाहर घोषित किया जाना चाहिए। आप इस प्रकार के चर का उपयोग कर सकते हैं, जब आपके पास एक सबरूटीन होता है जो एक मन उत्पन्न कर रहा है और आप उसी मान के अन्य मान को अन्य सबरूटीन में पास करना चाहते हैं।

Dim[variable] का उपयोग सबरूटीन के अंदर दायरे को बताने के लिए किया जाता है (आप इसके स्थान पर Private का उपयोग नहीं कर सकते हैं)। मंद को या तो सबरूटीन के अंदर या एक सबरूरूटीन के बाहर इस्तेमाल किया जा सकता है (इसे सबरूटीन के बाहर उपयोग करना Private का उपयोग करने जैसा ही होगा)।

पब्लिक, प्राइवेट और प्रोसीजर का उदहरण,

Module 1 code

```
Dim x As Integer ' This is a Private Variable since it is declared using Dim.

Public y As Integer

Sub First_Sub()

 x = 10

 y = 20

 Call Third_Sub()

End Sub

Private Sub Second_Sub()

MsgBox "Gone through First, Second and Third Subroutines !"

End Sub
```

Module 2 code

```
Sub Third_Sub()

Debug.Print x

Debug.Print y

Call Second_Sub()

End Sub
```

दो चर x और y जो सबरूटीन के बाहर घोषित किए जाते हैं। इसका मतलब है कि उनके मूल्य अन्य मैक्रोज़ में ले जा सकते हैं। चर x का एक private दायरा है, इसलिए उसी मॉड्यूल में केवल सबरूटीन ही इसके मूल्य तक पहुँच सकते हैं। चर y का public दायरा है, जिसके अर्थ है कि इसके मॉड्यूल के अंदर और बाहर सबरूटीन इसके मूल्य तक पहुँच सकते हैं।

The First_Sub() x और y को मान निर्दिष्ट करता है और फिर Third_Sub() को प्रारंभ करता है।

Third_Sub() को कॉल किया जा सकता है भले ही यह एक ही मॉड्यूल में न हो क्योंकि यह एक Public Sub है।

Third_Sub() को इमीडियेट विंडो में x और y के मान प्रदर्शित करने के लिए डिज़ाइन किया गया है। जब आप चर x को प्रिंट करने का प्रयास करते हैं तो यह कुछ भी आउटपुट नहीं करता है। ऐसा इसलिए है क्योंकि x मॉड्यूल 2 में मौजूद नहीं है। इसलिए, मॉड्यूल 2 में एक नया चर x बनाया गया है और चूंकि हमने इसे नया x को मान नहीं दिया है, तो इस स्टेटमेंट के लिए कुछ भी प्रिंट नहीं किया गया है। Debug.Print x

जब हम चर y के मान प्रिंट करते हैं, तो 12 इमीडियेट विंडो में प्रदर्शित होता है। ऐसा इसलिए है क्योंकि मॉड्यूल 2 सबरूटीन के पास मॉड्यूल1 में घोषित पब्लिक चरों तक पहुँच है। लेकिन Third_Sub() में "Call Second_Sub()" स्टेटमेंट का परिणाम एक त्रुटि होगी। ऐसा इसलिए है क्योंकि हम यहाँ से एक प्राइवेट सबरूटिन " Second_Sub" को कॉल करने का प्रयास कर रहें हैं। इससे बचने के लिए निम्नलिखित परिवर्तन किए जा सकते हैं :

1 हम Display_Message से "Private" शब्द को हटा सकते हैं।

2 हम Second_Sub() में "Private" को "Public" से प्रतिस्थापित कर सकते हैं।

3 हम एप्लीकेशन लेवल का उपयोग कर सकते हैं और Call का उपयोग करने के बजाय, हम Application.Run "Second_Sub" लिख सकते हैं। (यह मेर्थड ओवरराइड के रूप में कार्य करता है, यदि हम मॉड्यूल के बाहर सबरूटीन के लिए Second_Sub निजी रखन चाहते हैं)।

मैक्रोज़ बनाना और संपादित करना (Create and Edit Macros)

उद्देश्य : इस पाठ के अन्त में आप निम्नलिखित कार्य करने योग्य होंगे :

- **VBA में मैक्रोज़ के बारे में समझाना।**

मैक्रोज़ एक्सेल की सुविधाओं का विस्तार करने के लिए एक शक्तिशाली और लचीला तरीका प्रदान करता है। वे एक्सेल में डेटा प्रिंटिंग, स्वरूपण, कॉन्फ़िगर या बदलाव करेन जैसे दोहराए गए कार्यों के स्वचालन की अनुमति देते हैं। अपने सबसे सरल रूयप में, एक मैक्रो आपके की स्ट्रोक की रिकॉर्डिंग है। जबकि मैक्रोज़ एक्सेल में पाए जाने वाले मजबूत सुविधाओं में से एक का प्रतिनिधित्व करते हैं, लेकिन वे बनाना और उपयोग करता आसान है। छह प्रमुख बिंदु हैं जिन्हें मैं मैक्रोज़ के बारे में निम्नानुसार बनाना चाहता हूँ।

1 रिकॉर्ड करे, एक्सेल का उपयोग करें, रिकॉर्डिंग बंद करें (Record, Use Excel, Stop Recording)

मैक्रो बनाने के लिए, बस मैक्रो रिर्कार्डर चालू करें, सामान्य रूप से एक्सेल का उपयोग करें, फिर रिकॉर्डर को बंद करें। प्रोस्टो-आपने एक मैक्रो बनाया है। हांलाकि प्रक्रिया उपयोगकर्ता के दृष्टिकोण से सरल है, जबकि कवर के नीचे एक्सेल परिष्कृत विजुअल बेसिक प्रोग्रामिंग कमांड का उपयोग कर विजुअल बेसिक सबरूटीन बनाता है।

2 मैक्रो स्थान (Macro Location)

निम्नानुसार दो स्थानों में मैक्रोज़ को संग्रहित किया जा सकता है :

आप जिस वर्कबुक का उपयोग कर रहे हैं, या व्यक्तिगत मेक्रो वर्कबुक (जो डिफ़ॉल्ट रूप से दृश्य से छिपा हुआ है) यदि मैक्रो सभी वर्कबुक पर लागू होता है, तो उसे व्यक्तिगत मैक्रो वर्कबुक में संग्रहित करें ताकि यह हमेशा सभी एक्सेल वर्कबुक में उपलब्ध रहे, अन्यथा इसे वर्तमान वर्कबुत में संग्रहित करें। वर्तमान भले ही आप किसी अन्य उपयोगकर्ता को वर्कबुक ईमेल करते हैं।

3 मैक्रो को आइकन, टेक्स्ट या बटन पर असाइन करें (Assign the Macro to an Icon, Text or a Button)

अपने मेक्रो को चलाने में आसान बनाने के लिए, आपको इसे टूलबार आइकन पर असाइन करना चाहिए ताकि यह हमेशा उपलब्ध रहे चाहे इससे कोई फर्क नहीं पड़ता कि आप कौन सी वर्कबुक खोल चुके हैं। यदि मैक्रो केवल वर्तमान वर्कबुक पर लागू होता है, तो उसे टेक्स्ट या मैक्रो बटन पर असाइन करें ताकि यह वर्तमान वर्कबुक में तुरंत उपलब्ध हो सके।

4 अब्सोल्युट बनाम रिलेटिव मैक्रोज़ (Absolute versus Relative Macros)

एक “अब्सोल्यूट” मैक्रो हमेशा एक ही सेल को हमेशा प्रभावित करेगा जबकि एक “रिलेटिव” मैक्रो उन सेलों को प्रभावित करेगा जहाँ मैक्रोज़ का आह्वान करते समय कर्सर स्थित है। यह महत्त्वपूर्ण है कि अंतर को समझें।

5 मैक्रोज़ संपादन करना (Editing Macros)

एक बार यह बनने के बाद, आप व्यू मैक्रोज़ विकल्प का उपयोग करके अपने मैक्रो का देख और/या संपादित कर सकते हैं। यह एक विजुअल बेसिक प्रोग्रामिंग विंडो में मैक्रो सबरूटीन खुल जाएगा और आपको भरपूर VB टूल्स प्रदान करेगा।

6 उन्नत विजुअल बेसिक प्रोग्रामिंग (Advanced Visual Basic Programming)

वास्तव में महत्वाकांक्षी CPA के लिए, विजुअल बेसिक प्रोग्रामिंग विंडो में, आपके पास आवश्यक टूल हैं जो आपको डायलॉग बॉक्स ड्राप डाउन मेनू विकल्प, चेक बॉक्स, रेडियो बटन - के साथ बहुत परिष्कृत मैक्रोज़ बनाने के लिए आवश्यक हैं। इस को देखने के लिए, एक्सेल आप्शन में “डेवलपर टैब” ऑन करें। नीचे मैक्रोज़ बनाने के आमंत्रित करने के लिए अधिक विस्तृत टिप्पणियाँ और चरण-दर-चरण निर्देश, प्रस्तुत किया गया है इसके बाद मैक्रो पर कुछ उदाहरण बताये गए हैं।

पेज सेटअप मैक्रो (Page Setup Macro)

पेज सेटअप मैक्रो नामक एक नए मैक्रो रिकॉर्डिंग शुरू करें। सभी वर्कशीट का चयन करें और फिर पेज सेटअप चुनें और पेज नंबर, दिनांक और समय टिकटें, फाइल स्थान, टैब नाम इत्यादि शामिल करने के लिए हेडर और फूटर को कस्टमाइज़ करें। टूलबार या क्विक एक्सेस बार पर आइकन पर मैक्रो आइकन करें और हेडर डालें।

प्रिंट मैक्रोज़ (Print Macros)

क्या आपके पास एक टेम्पलेट है जो अक्सर प्रिंट करता है ? यदि ऐसा है, तो प्रत्येक रिपोर्ट को, रिपोर्ट का एक समूह, और यहाँ तक कि कई रिपोर्ट प्रिंट करने के लिए कई मैक्रो बटन डालें, और रिपोर्टिंग भविष्य में स्नैप की जाएगी।

डेटा मैक्रो हटाएं (Delete Data Macro)

डेटा मैक्रो हटाएं एक मैक्रो बनाएं जो प्रत्येक सेल पर जाता है और उस डेटा को मिटा देता है, वर्कशीट को नए मानदंडों में उपयोग के लिए रीसेट कर रहा है। मैक्रो को मैक्रो बटन पर असाइन करें और आपके नए टेम्पलेट के साथ कभी भी पुरानी मान्यताओं को मिश्रित नहीं किया जाएगा।

एक्सेल VBA में उपयोगकर्ता फॉर्म्स और कन्ट्रोल (User forms and control in Excel VBA)

उद्देश्य : इस पाठ के अन्त में आप निम्नलिखित कार्य करने योग्य होंगे :

- **VBA में फॉर्म के प्रकारों का वर्णन करना**
- **एक्सेल फॉर्म के प्रकारों को वर्णन करना**
- **फॉर्म की प्रॉपर्टीज, मेथड ओर इवेंट्स का वर्णन करना।**

फॉर्म्स और कंट्रोल्स का परिचय (Introduction to Forms and Controls)

एक फॉर्म एक मानक संरचना और प्रारूप के साथ डिज़ाइन किया गया एक दस्तावेज़ है जो जानकारी दर्ज करना, व्यवस्थित करना और संपादित करना आसान बनाता है। प्रपत्र में लेबल, टेक्स्ट बॉक्स, ड्रॉप डाउन बॉक्स और कमांड बटन भी शामिल हैं।

फॉर्म और कई कंट्रोल्स और ऑब्जेक्ट्सस का उपयोग करके जिन्हें आप उन्हें जोड़ सकते हैं, आप अपने वर्कशीट पर डेटा एंट्री को काफी बढ़ा सकते हैं अपनी वर्कशीट्स के तरीके को बेहतर बना सकते हैं।

एक्सल फॉर्म्स के प्रकार (Types of Excel forms)

कई प्रकार के फॉर्म्स हैं, जिन्हें आप एक्सेल में बना सकते हैं, डेटा फॉर्म, वर्कशीट जिनमें फॉर्म और ActiveX कंट्रोल होते हैं, और VBA UserForms शामिल हैं।

डेटा फॉर्म (Data form)

एक डेटा फॉर्म हॉरिजॉन्टल स्क्रॉल किए बिना, किसी श्रेणी या टेबल में जानकारी की एक पूरी पंक्ति दर्ज करने या प्रदिर्शित करने का एक सुविधाजनक तरीका प्रदान करता है। आप पाते हैं कि डेटा फॉर्म का उपयोग करके कॉलम से कॉलम पर जाने की अपेक्षा डेटा एंट्री आसान हो सकती है। जब आपके पास स्क्रीन पर देखे जाने वाले डेटा के अधिक कॉलम होते हैं। एक्सेल स्वचालित रूप से आपकी रेंज या टेबल के लिए बिल्टइन डेटा फॉर्म उत्पन्न कर सकता है।

फ़ॉर्म और ActiveX नियंत्रण के साथ वर्कशीट (Worksheet with Form and ActiveX controls)

वर्कशीट को एक फॉर्म माना जा सकता है, जो आपको ग्रिड पर डेटा दर्ज करने और देखने में सक्षम बनाता है।

अतिरिक्त लचीलापन के लिए, आप वर्कशीट में कंट्रोल्स और अन्य ड्राइंग ऑब्जेक्ट्स जोड़ सकते हैं, और उन्हें वर्कशीट सेल के साथ संयोजित और समन्वयित कर सकते हें। उदाहरण के लिए, आप किसी आइटम के आइटमों से चयन करने के लिए उपयोगकर्ता को आसान बनाने के लिए एक सूची बॉक्स कंट्रोल्स का उपयोग कर सकते हैं या, आप किसी उपयोगकर्ता को संख्या दर्ज करने के लिए आसान बनाने के लिएस्पिन बटन कंट्रोल्स का उपयोग कर सकते हैं।

आप अपने वर्कशीट पर ग्रिड या डेटा की तालिका के लेआउट को बदलने के बिना ग्रिड और कॉलम सीमाओं से स्वतंत्र संबंधित टेक्स्ट के साथ्म कंट्रोल्स और ऑब्जेक्ट्स को प्रदर्शित या देख सकते हैं। इन कंट्रोल्स में से कई को वर्कशीट पर सेल्स से भी जोड़ा जा सकता है और उन्हें काम करने के लिए VBA कोड की आवश्यकता नहीं है। उदाहरण के लिए, आपके पास एक चेक बॉक्स हो सकता है जिसे आप रेंज सॉर्ट करते समय इसके बिल्ट-इन सेल के साथ एक साथ स्थानांतरित करना चाहते हें। हांलाकि यदि आपके पास एक सूची बॉक्स है जिसे आप हर समय एक विशिष्ट स्थान पर रखना चाहते हैं, तो शायद आप इसे अपने बिल्ट-इन सेल के साथ एक साथ स्थानांतरित नहीं करना चाहते हैं।

VBA फार्म्स बनाने (Creating VBA Forms)

कोड विंडो से एक VBA फॉर्म बनाया जा सकता है। VBA में एक फॉर्म बनाने के लिए, कोड विंडो में इन्सर्ट मेनू पर क्लिक करें और फिर 'UserForm' पर क्लिक करें। प्रोजेक्ट विंडो में UserForm1 प्रकट होता है।

जब आप कोई फॉर्म बनाते या जोड़ते हैं, तो इसके लिए एक मॉड्यूल भी स्वचालित रूप से बनाया जाता है। किसी फ़ॉर्म से जुड़े मॉड्यूल तक पहुँचने के लिए, आप फॉर्म पर राइट-क्लिक कर सकते हैं और व्यू कोड पर क्लिक कर सकते हें। फॉर्म पर डबल क्लिकिंग या F7 दबाकर भी कोड विंडो खुल जाएगी। Shift F7 का उपयोग फिर से डिज़ाइन विंडो पर वापस ले आएगा।

फ़ॉर्म पर राइट क्लिक करके और 'प्रॉपर्टीज' चुनकर सेट किए जा सकते हैं। इसे "F4" या फॉर्म पर प्रॉपर्टीज बटन पर क्लिक करके हासिल किया जा सकता है। आवश्यकता के अनुसार टूलबॉक्स से फ़ॉर्म पर कंट्रोल्स रखा जा सकता है।

इसके अतिरिक्त, "Add" मेथड का उपयोग कर प्रोग्राम पर प्रोग्रामेटिक रूप से/रन टाइप पर कंट्रोल्स को जोड़ा जा सकता है। इसी तरह कंट्रोल्स को "Remove" मेथड का उपयोग करके रन टाइम/प्रोग्रामेटिक रूप से फॉर्म से हटाया जा सकता है। एक चेक बॉक्स कंट्रोल जोड़ने के लिए एक उदाहरण के रूप में, हम लिख सकते हैं।

```
Set cb1 = Controls.Add("Forms.CheckBox.1")
```

फॉर्म ऑब्जेक्ट से जुड़े कुछ ईवेंट और मेथड़ हैं :

Events, Activate, Deactivate, Add Control, Remove Control, Click, DblClick, Initialize, KeyPress, Resize, Scroll, Terminate, Zoom etc.

Methods Copy, Paste, Hide, Move, Print Form, Repaint, Scroll, Show etc .

फॉर्म पर विभिन्न संचालन करने के लिए आवश्यक कोड टेबल 1 में दिया जाता है।

टेबल 1

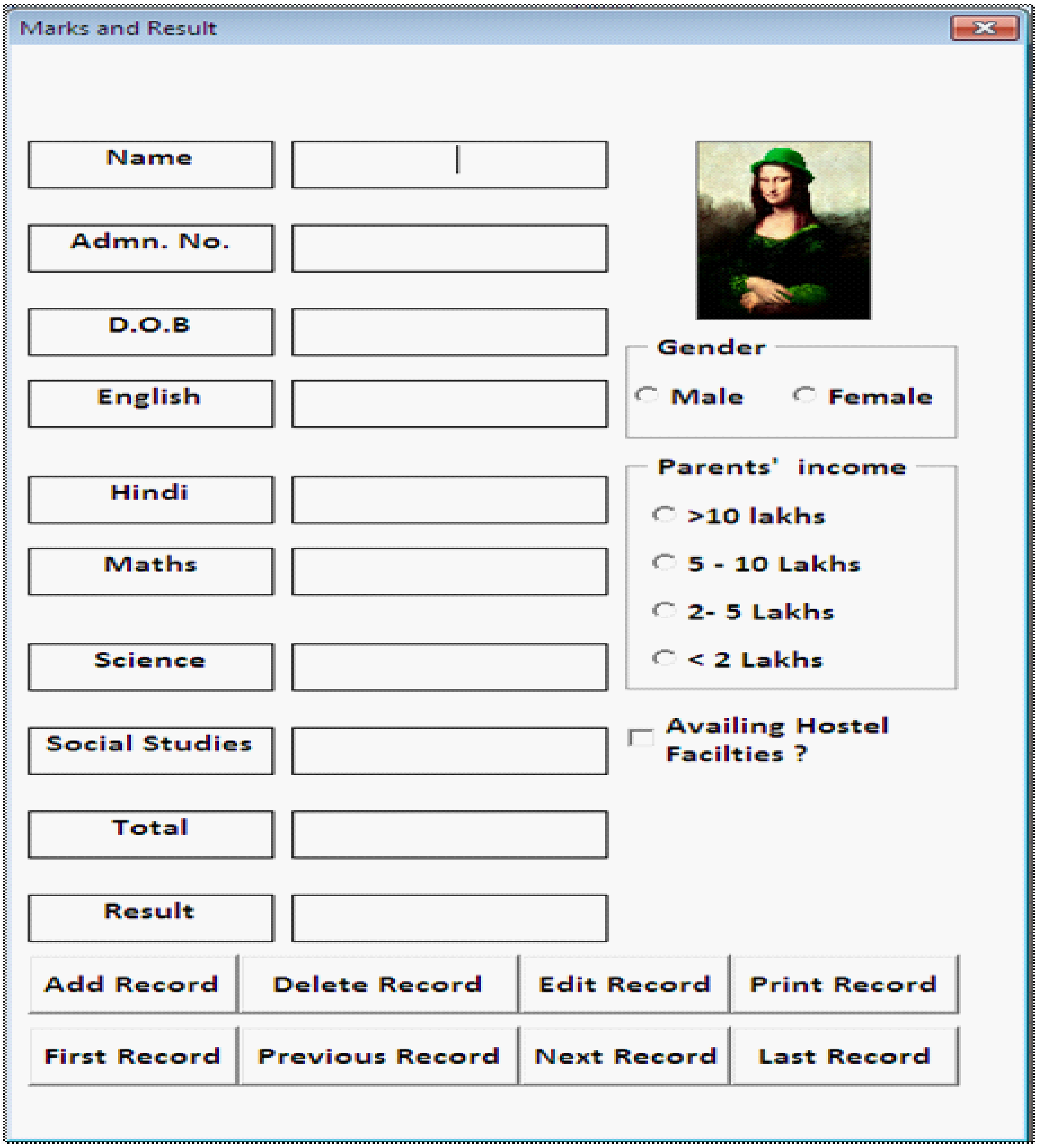
Userform एप्लीकेशन	**VBA कोण**	**एक्शन**
UserForm प्रदर्शित करने के लिए	UserForm1.Show	UserForm1 नाम के साथ UserForm प्रदर्शित करता है। यह कोड स्टेंडर्ड VBA मॉड्यूल में डाला जाना चाहिए और UserForm के कोड मॉड्यूल में नहीं होना चाहिए। आप वर्कशीट में एक बटन बना सकते हें, फिर इस बटन पर मैक्रो असाइन करने के लिए राइट क्लिक करें, और मैक्रो का चयन करे जो UserForm दिखाता है।
UserForm में मेमोरी में लोड करता है लेकिन प्रदर्शित नहीं करता	Load UserForm1	लोड कथन एक जटिल उपयोगकर्ता फॉर्म के मामले में उपयोगी हैं जिसे आप स्मृति में लोड करना चाहते हें ताकि यह शो विधि का उपयोग करने पर तेज़ी से प्रदर्शित हो सके, जो अन्यथा प्रकट होने में लंबा समय ले सकता है।
UserForm को मेमोरी से हटाता है/बंद करता है।	Unload UserForm1	नोट : Hide method (UserForm1.Hide) उपयोगकर्ता फॉर्म को स्मृति से अनलोड नहीं करता है। स्मृति से उपयोगकर्ता फॉर्म को अनलोड के लिए, unload मेथड का उपयोग किया जाना चाहिए।
	Unload Me	UserForm के कोड मॉड्यूल में किसी प्रोसीजर में Me कीवर्ड का उपयोग करें।
UserForm को छुपाता है।	UserForm1.Hide	Hide मेथड का उपयोग अस्थायी रूप से उपयोगकर्ता फॉर्म को छुपाएगा, लेकिन इसे बंद नहीं करेगा और यह समृति में लोड रहेगा।
UserForm को प्रिंट	UserForm1.PrintForm	PrintForm मेथड सीधे प्रिंटिंग के लिए उपयोगकर्ता फॉर्म भेजती है।
UserForm को मोडलेस के रूप में दिखाता है।	UserForm1.Show False	यदि उपयोगकर्ता फॉर्म को मॉडलेस के रूप में प्रदर्शित किया गया है, तो उपयोगकर्ता Excel में काम करना जारी रख सकता है जबकि उपयोगकर्ता फॉर्म दिखया जा रहा है। बूलियन आर्ग्युमेंट को छोड़कर (गलत या 0) उपयोगकर्ता फॉर्म को मॉडल के रूप में प्रदर्शित करेगा, जिस स्थिति में उपयोगकर्ता Excel में एक साथ काम नहीं कर सकता है। डिफॉल्ट रूप से UserForm को मॉडल के रूप में प्रदर्शित किया जाता है।
UserForm को बंद करता है।	Unload UserForm1	Unload मेथड निर्दिष्ट उपयोगकर्ता फॉर्म को बंद कर देती है।
	Unload Me	Unload मेथड उपयोगकर्ता फॉर्म को बंद कर देती है जिसमें कोड मॉड्यूल रहता है।
	End	फॉर्म को बंद करने के लिए "Close" कमांड बटन में एंड कथन का उपयोग करें। "End" कथन सभी फॉर्म्स को अनलोड कर देता है।
UserForm कैप्शन निर्दिष्ट करता है।	UserForm1.Caption = "Bio Data"	कैप्शन वह टेक्स्ट है जो उपयोगकर्ता फॉर्म का वर्णन करता है और पहचानता है और उपयोगकर्ता फॉर्म के हैडर में प्रदर्शित होगा।
UserForm का साइज सेट	UserForm1.Height = 250	यह पॉइंट में उपयोगकर्ता फॉर्म की ऊंचाई सेट करता है।
करता है।	UserForm1.Width = 350	यह पॉइंट में उपयोगकर्ता फॉर्म की चौड़ाई सेट करता है।
		UserForm पोजीशन सेट करना।
Left & Top properties	UserForm1.Left = 30 UserForm1.Top = 50	पिक्सेल में दूरी सेट करता है, उस विंडो के फॉर्म और बाएं या शीर्ष एज के बीच। window that contains it, in pixels.
Move मेथड	UserForm1.Move 200, 50	मूव मेथड में दो आर्ग्यूमेंट शामिल हैं जो आवश्यक हैं - बाएं दूरी और शीर्ष दूरी, उस क्रम में।

छात्रों के विवरण, अंक और परिणामों की डेटा प्रविष्टि के लिए एक नमूना फॉर्म Fig. 1 में दिखाया गया है।

कमांड बटन और दिखाए गए अन्य कंट्रोल्स के लिए आवश्यक कोड संलग्न किया जा सकता है। उपयोगकर्ता के डेटा दर्ज करने के बाद, कुल गणना की जाती है और परिणाम प्रदर्शित होता है। रिकॉर्ड को उचित रूप से संग्रहित किया जा सकता है।

कमान्ड बटन और दिखाए गए अन्य नियंत्रणों के लिए आवश्यक कोड संलग्न किया जाता है। उपयोगकर्ता डेटा दर्ज करने के बाद, कुल गणना की जाती है और परिणाम प्रदर्शित होता है। रिकॉर्ड को उचित रूप से संग्रहित किया जा सकता है।

Fig 1



VBA में मेथड और इवेंट्स (Methods and Events in VBA)

उद्देश्य : इस पाठ के अन्त में आप निम्नलिखित कार्य करने योग्य होंगे :
- **VBA मेथड और इवेंट्स की व्याख्या करना।**

मेथड्स और इवेंट्स (Methods and Events)

मेथड्स (Methods)

एक मेथड एक एक्शन है जिसे आप किसी ऑब्जेक्ट के साथ करते हैं। एक मेथड किसी ऑब्जेक्ट की प्रॉपर्टीज को बदल सकती है या ऑब्जेक्ट को कुछ कर सकती है।

उदाहरण के लिए पेंटिग एक मेथड है, एक नया कमरा बनाना एक नया घ्रर बनाने में एक मेथड है।

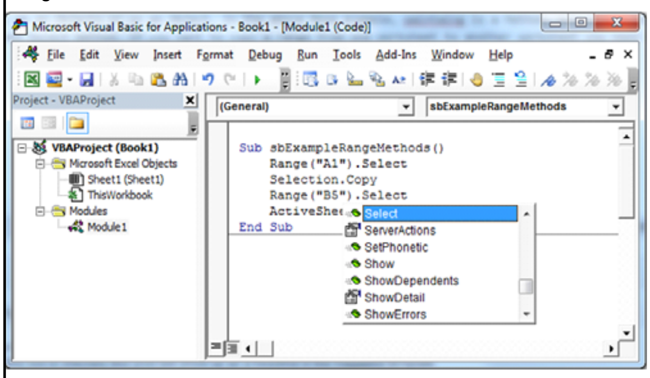
इसी प्रकार, यदि आप एक रेंज का चयन करना चाहते हैं, तो आपको Select मेथड की आवश्यकता है। यदि आप एक वर्कशीट से किसी अन्य वर्कशीट में किसी रेंज की प्रतिलिपि बनाना चाहते हैं तो आपको इसे करने के लिए Copy मेथड की आवश्यकता होगी।

उदाहरण रेंज A1 से B5 तक डेटा कॉपी करता है।

Fig 1 में दिखाए गए मॉड्यूल 1 में निम्न कोड दर्ज करें।

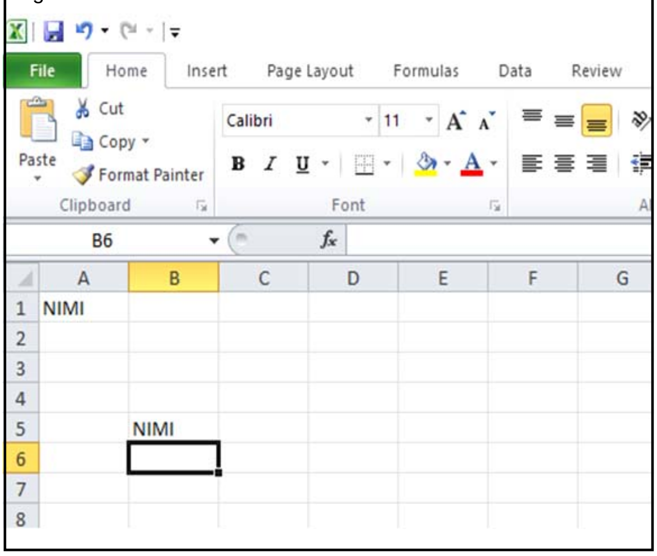
```
Sub sbExampleRangeMethods()
Range("A1").Select
Selection.Copy
Range("B5").Select
ActiveSheet.Paste
End Sub
```

Fig 1



यदि उपर्युक्त कोड निष्पादित किया गया है, तो सेल A1 की कंटेंट्स को Fig 2 में दिखाए गए अनुसार सेल B5 में कॉपी किया गया है।

Fig 2



इवेंट्स (Events)

एक इवेंट्स उपयोगकर्ता एक्शन या अन्य VBA कोड द्वारा शुरू की गई एक एक्शन है। एक इवेंट प्रोसीजर एक सब प्रोसीजर है जिसे आप लिखते हैं, इवेंट के विनिर्देश के अनुसार, जिसे किसी ईवेंट के दौरान एक्सेल द्वारा स्वचालित रूप से कहा जाता है। उदाहरण के लिए, वर्कशीट ऑब्जेक्ट में Change नामक एक ईवेंट होता है। यदि आपने चेंज इवेंट के लिए इवेंट प्रोसीजर को सही तरीके से प्रोग्राम किया है, तो एक्सेल स्वचलित रूप से उस प्रोसीजर को कॉल करेगा, हमेशा Worksheet_Change नामक और हमेशा वर्कशीट के कोड मॉड्यूल में, जब भी वर्कशीट पर किसी भी सेल का मान यूजर इनपुट या अन्य VBA कोड द्वारा बदल जाता है। (लेकिन नहीं, यदि मान में परिवर्तन फॉर्मूला गणना का परिणाम है)। कौन सा सेल बदल गया था या नए बदले गए मान के आधार पर आप कुछ क्रियाएं करने के लिए Worksheet_Change इवेंट प्रोसीजर में कोड लिख सकते हैं।

Fig 3 में दिखाए गए Worksheet_Change ईवेंट में निम्न कोड दर्ज करें।

```
Private Sub Worksheet_Change(ByVal Target As Range)
MsgBox "Changed"
End Sub
```

जब हम किसी भी सेल की सामग्री बदलते हैं तो Fig 4 में दिखाए अनुसार निम्न संदेश प्रदर्शित किया जाएगा।

Fig 3

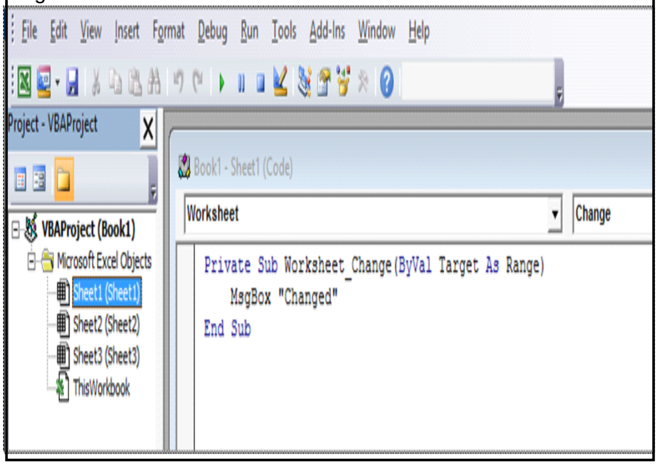
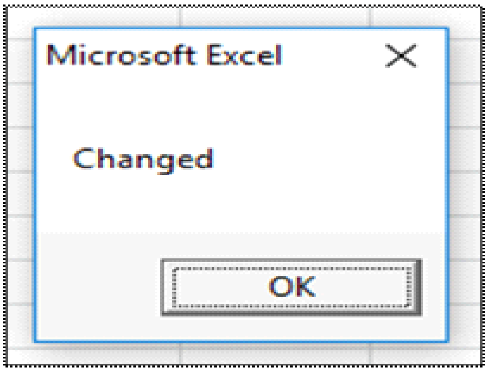

File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Project - VBAProject
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook
Book1 - Sheet1 (Code)
Worksheet
Change
Private Sub Worksheet_Change(ByVal Target As Range)
MsgBox "Changed"
End Sub

Fig 4


Microsoft Excel
Changed
OK

VBA में डीबगिंग तकनीकें (Debugging Techniques in VBA)

उद्देश्य : इस पाठ के अन्त में आप निम्नलिखित कार्य करने योग्य होंगे :
- **VBA के बारे में समझाना**
- **ब्रेक पॉइंट्स को सेट और क्लियर करने का तरीका बताना**
- **इमीडियेट विंडो के उपयोग का वर्णन करना**
- **वाच विंडो के बारे में समझाना।**

VBA डीबगिंग (VBA Debugging)

एक्सेल 2010 में, VBA के डीबगिंग एनवायरनमेंट प्रोग्रामर को VBA कोड के निष्पादन को क्षणित रूप से निलंबित करने की अनुमति देता है ताकि निम्नलिखित डीबग कार्य किए जा सकें :

1 अपने वर्तमान स्थिति में एक चर के मान की जांच करें।

2 परिणामों को देखने के लिए तत्काल विंडो में VBA कोड दर्ज करें।

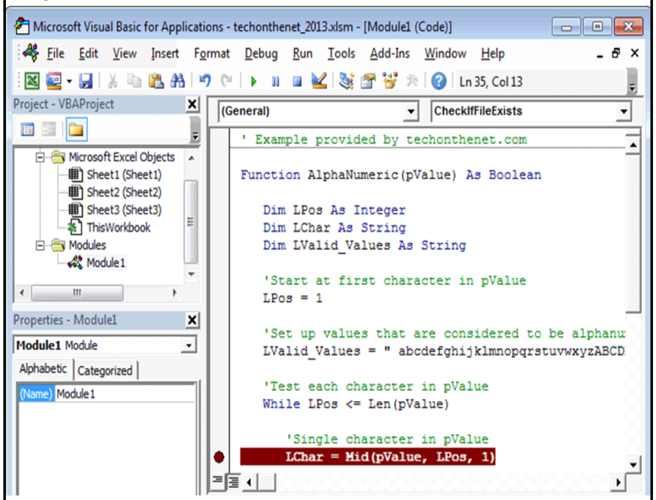
3 एक समय में कोड की प्रत्येक पंक्ति का निष्पादन करें।

4 कोड के निष्पादन जारी रखें।

5 कोड क निष्पादन को रोकें।

ये केवल कुछ कार्य हैं जिन्हें आप VBA के डिबगिंग एनवायरनमेंट में कर सकते हैं। (Fig 1)

Fig 1



VBA में ब्रेकपॉइंट (Breakpoint in VBA)

एक्सेल 2010 में, ब्रेकपॉइंटख एक बार चयनित कोड की एक चुनी लाइन है जिस पर पहुँचने के बाद प्रोग्राम क्षणिक रूप से ससपेंड हो जाएगा। एक बार ससपेंड हो जाने के बाद, और प्रोग्राम की स्थिति देखने के लिए VBA के डिबगिंग एनवायरमेंट का उपयोग करने के लिए, कोड के निष्पादन जारी रखें, या कोड के निष्पादन रोकें।

और यदि आप चाहते हें तो कोड में कई ब्रेकपॉइंट बनाएं। ब्रेकपॉइंट्स विशेष रूप से उपयोगी होते हैं जब प्रोग्राम को सस्पेंड करते हैं जहाँ आपको काई समस्या/वग मौजूद होने पर आशंका होती है।

ब्रेकपॉइंट सेट करना (Setting a Breakpoint)

सबसे पहले, आपको VBA एनवायरनमेंट खोलने की जरूरत है। ऐसा करने का सबसे तेज़ तरीका Alt+F11 दबाकर है जब एक्सेल डेटाबेस फ़ाइल खुलती है।

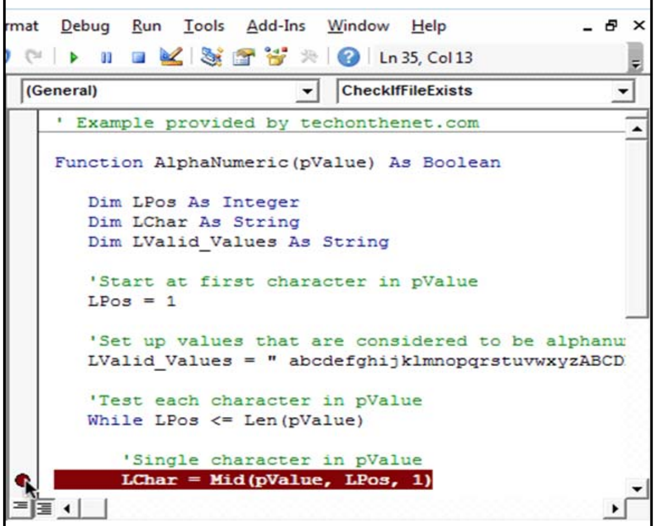
ब्रेकपॉइंट सेट करने के लिए, कोड लाइन ढूंढें जहाँ आपको प्रोग्राम को ससपेंड करना है। कोड के बाईं ओर ग्रे बार में बाएं-क्लिक करें। एक लाल बिंदु दिखाई देना चाहिए और कोड की रेखा को लाल रंग में हाइलाइट किया जाना चाहिए।

VBA में ब्रेकपॉइंट क्लियर करने के लिए (Clear Breakpoint in VBA)

VBA में एक ब्रेकपॉइंट लाला रंग में हाइलाइट किए गए कोड की एक पंक्टि के साथ रॉड डॉट द्वारा इंगित किया जाता है।

Excel 2010 में ब्रेकपॉइंट को क्लियर करने के लिए, ब्रेकपॉइंट वाले कोड की रेखा के बगल में लाल बिंदु पर बायाँ क्लिक करें। (Fig 2)

Fig 2



इस उदाहरण में, हम कोड की निम्न पंक्ति पर ब्रेकपाइंट क्लियर करना चाहते हैं :

LChar = Mid(pValue, LPos, 1) (Fig 3)

अब ब्रेकपॉइंट क्लियर हो गया है और कोड की लाइन फिर से सामान्य दिखनी चाहिए। (Fig 4)

Fig 3

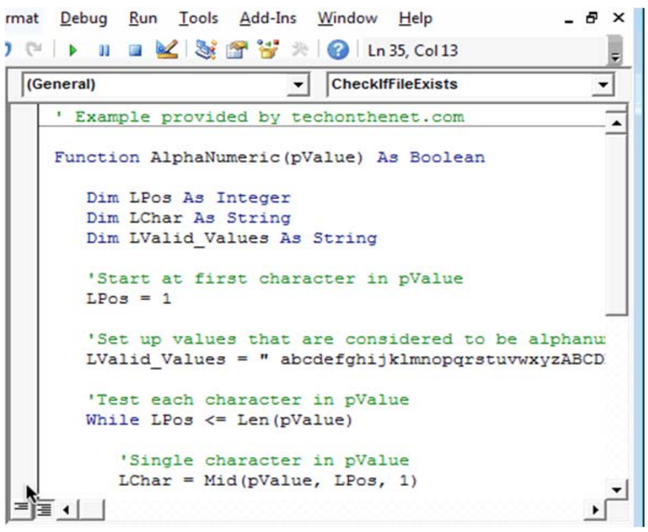
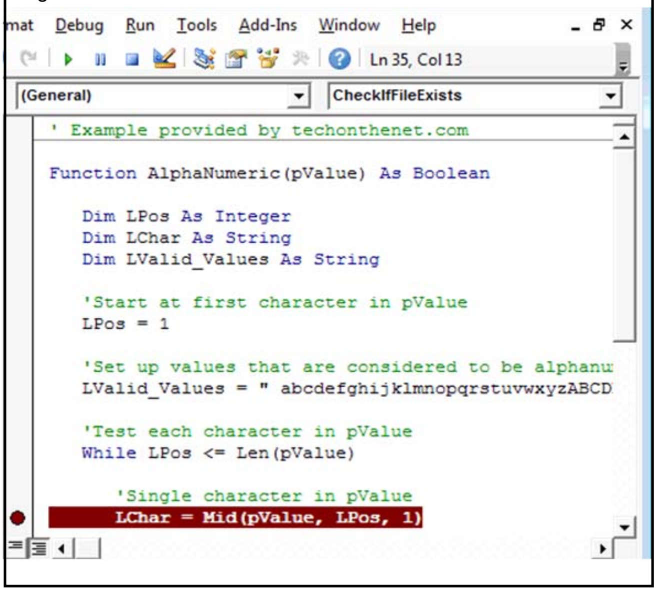


Fig 4



इस उदाहरा में, हमने कोड की निम्न पंक्ति पर ब्रेकपॉइंट बनाया है :

LChar = Mid(pValue, LPos, 1)

अब, ब्रेकपॉइंट क्लियर कर दिया गया है और कोड की लाइन फिर से सामान्य दिखनी चाहिए।

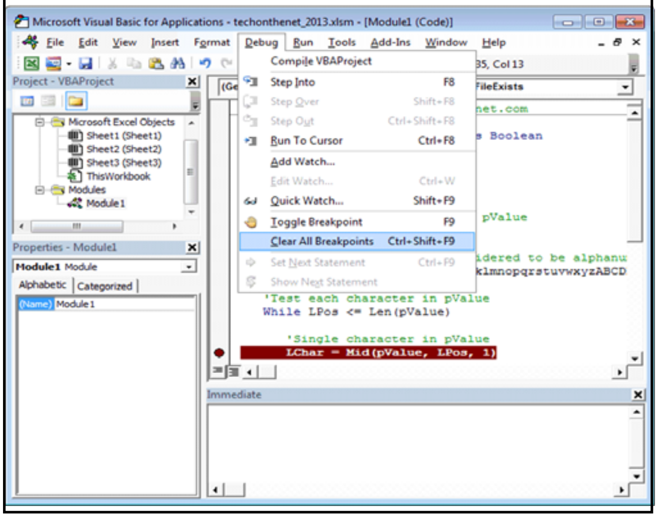
सभी ब्रेकपॉइंट क्लियर करना (Clearing all Breakpoints)

यदि उपयोगकर्ता Excel 2010 में जितना चाहें उतने ब्रेकपॉइंट्स का उपयोग कर सकते हैं, और एक बार में VBA कोड में सभी ब्रेकपॉइंट्स को क्लियर करके समय बचा सकते हैं।

कार्यक्रम में सभी ब्रेकपॉइंट्स को साफ़ करने के लिए, डीबग मेनू के तहत "क्लियर आल ब्रेकपॉइंट्स" का चयन करें। (Fig 5)

यह VBA कोड से सभी ब्रेकपॉइंट्स को हटा देगा, ताकि आपको अलग-अलग ब्रेकपॉइंट को एक-एक करके अलग-अलग करने की आवश्यकता नहीं है।

Fig 5

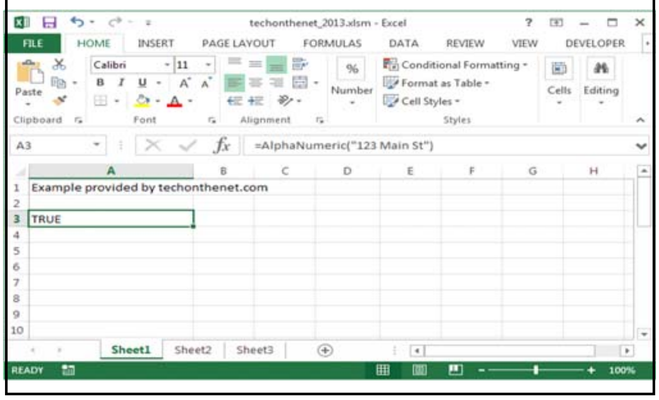


डिबग मोड (Debug Mode)

अब हम Excel 2010 में ब्रेकपॉइंट को सेट और क्लियर करने के बारे में जानते हैं, तो हम VBA में डीबग मोड पर नज़र डालें।

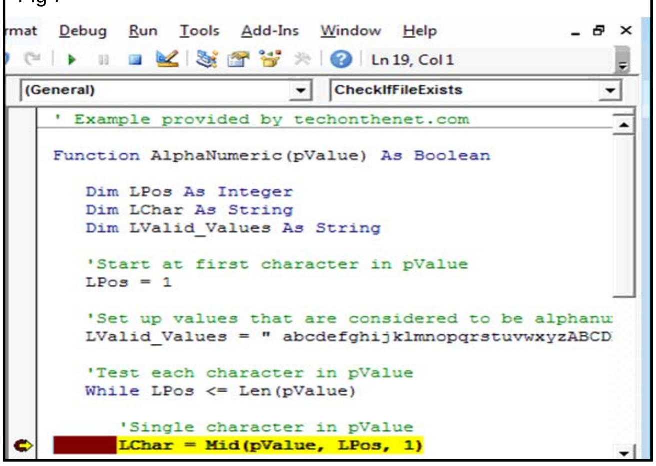
हमारे उदाहरण में, हमने अपना ब्रेकपॉइंट सेट कर किया है और हमारे अल्फा न्यूमेरिक फ़ंक्शन को सेल में सूत्र के रूप में दर्ज किया है। इससे VBA कोड निष्पादित हो जाएगा। (Fig 6)

Fig 6



जब ब्रेकपॉइंट तक पहुँच जाता है, तो एक्सेल माइक्रोसॉफ्ट विजुअल बेसिक विंडो प्रदर्शित करेगा और लाइन (पीले रंग में) को हाइलाइट करेगा जहाँ कोड ससपेंड कर दिया गया है। (Fig 7)

Fig 7



अब हम अपने एक्सेल स्प्रेडशीट में डीबग मोड में हैं। अब हम निम्न में से कोई भी कर सकते हैं :

1 अपने वर्तमान स्थिति में एक चर के मान की जांच करें।

2 परिणामों को देखने के लिए तत्काल विंडो में VBA कोड दर्ज करें।

3 एक समय में कोड की प्रत्येक पंक्ति का निष्पादन करें।

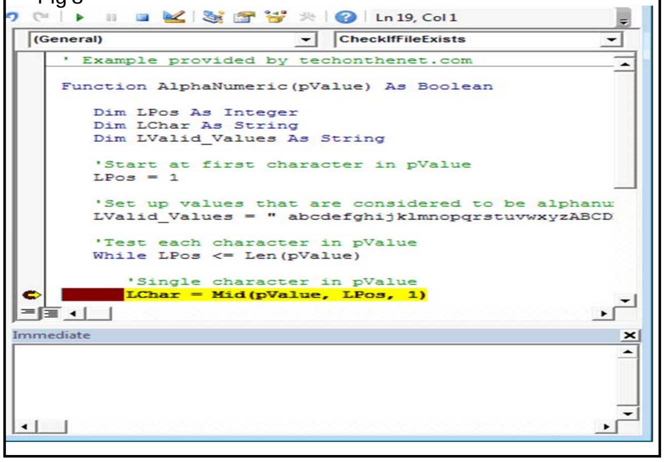
4 कोड के निष्पादन जारी रखें।

5 कोड के निष्पादन को रोकें।

इमीडियेट विंडो का उपयोग करना (Using the Immediate Window)

एक्सेल 2010 में, ससपेंड प्रोग्राम के संदर्भ में VBA कोड दर्ज करने और चलाने की अनुमति देकर इमीडियेट विंडो का उपयोग आपके प्रोग्राम की डीबग करने के लिए किया जा सकता है। (Fig 8)

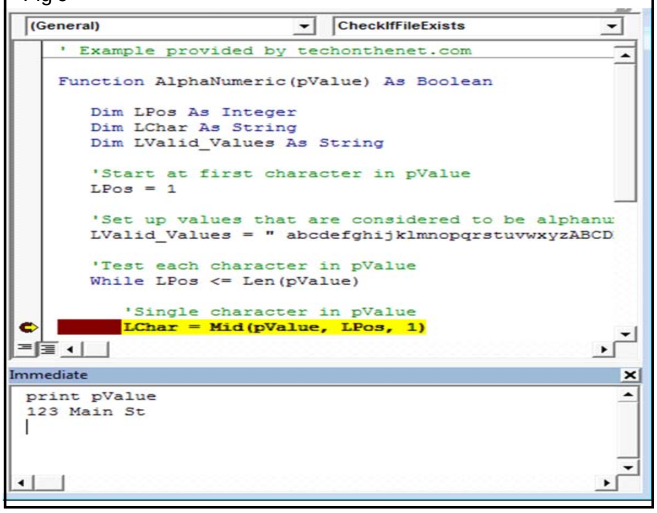
Fig 8



हमने पाया है कि तत्काल विंडो सबसे उपयोगी है, जब हमें प्रोग्राम में किसी निश्चित बिंदु पर एक चर, अभिव्यक्ति या ऑब्जेक्ट का मान पता करने की आवश्यकता होती है। यह प्रिंट कमांड का उपयोग करक किया जा सकता है।

उदाहरण के लिए, यदि आप pValue नामक चर के वर्तमान मान को देखना चाहते हैं, तो आप निम्नानुसार प्रिंट कमांड का उपयोग कर सकते हैं : (Fig 9)

Fig 9



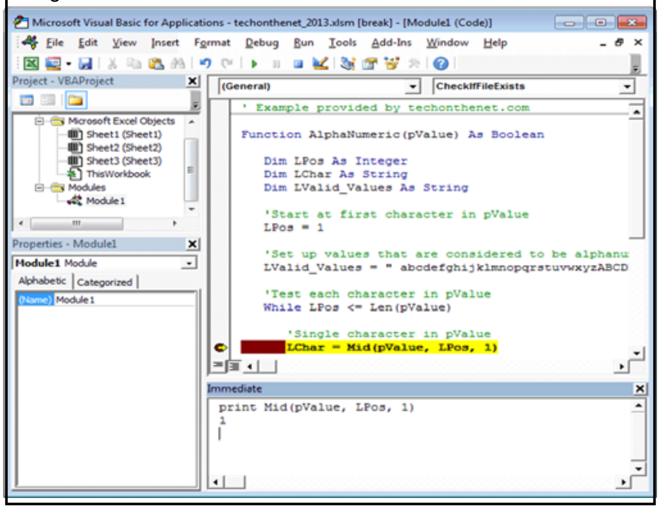
इस उदाहरण में, हमने इमीडियेट विंडो में **print pValue** टाइप किया और इंटर कुंजी को दबाया।

Print pValue

इमीडियेट विंडो ने परिणाम को अगली पंक्ति में प्रदर्शित किया है। इस मामले में, print pValue कमांड **123 Main St.** को लौटाएगा।

आप इमीडियेट विंडो में अधिक जटिल एक्सप्रेशन भी टाइप कर सकते है। (ENTER दबाना याद रखें)। उदाहरण के लिए : (Fig 10)

Fig 10



इस उदाहरण में, हम इमीडियेट विंडो में **print Mid(pValue, LPos, 1)** टाइप करके इंटर प्रेस करते हैं।

print Mid(pValue, LPos, 1)

तत्काल विंडो ने अगली पंक्ति में **1** का परिणाम प्रदर्शित किया।

तत्काल विंडो का उपयोग अन्य प्रकार के VBA कोड को चलाने के लिए किया जा सकता है, लेकिन ध्यान रखें कि तत्काल विंडो का उपयोग केवल तभी किया जा सकता है जब आपके द्वारा चलाए गए किसी भी कोड को डीबग करने के लिए केवल डीबगिंग उद्देश्यों के लिए होता है। तत्काल विंडो में दर्ज कोड सहेजा नहीं जाता है और मौजूदा VBA कोड में जोड़ा जाता है।

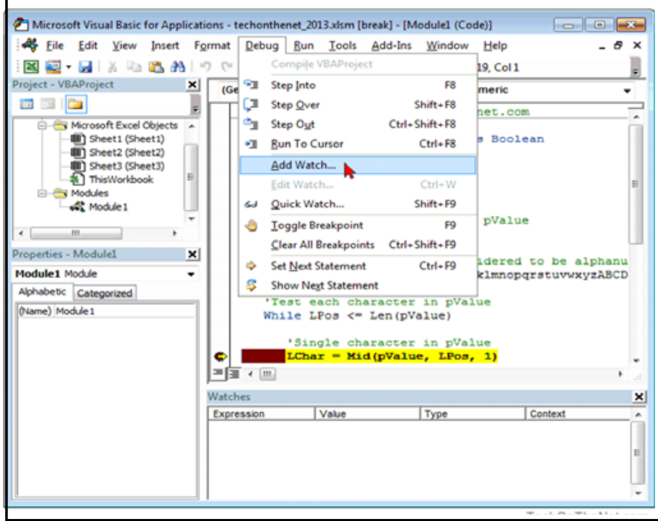
एक वॉच एक्सप्रेशन जोड़ना (Adding a Watch Expression)

Watch Window वॉच विंडो अपने वर्तमान स्थिति में एक देखी गई एक्सप्रेशन का मान प्रदर्शित करती है। VBA कोड डीबग करते समय यह बेहद उपयोगी हो सकता है। आइए देखें कि वॉच विंडो में एक्सप्रेशन कैसे जोड़े।

वॉच एक्सप्रेशन जोड़ने के लिए, डीबग मेनू के तहत ऐड वॉच का चयन करें। (Fig 11)

जब ऐड वॉच विंडो दिखाई देती है, तो देखने के लिए एक्सप्रेशन दर्ज करें और अब आप पूरा कर लें तो आके बटन पर क्लिक करें। (Fig 12)

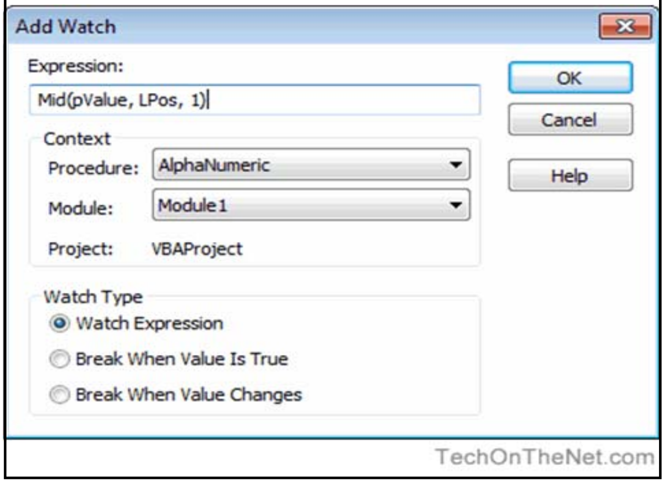
Fig 11



इस उदाहरण में, हमने अभिव्यक्ति फ़ील्ड में निम्न घड़ी अभिव्यक्ति दर्ज की है :

Mid(pValue, LPos, 1)

Fig 12



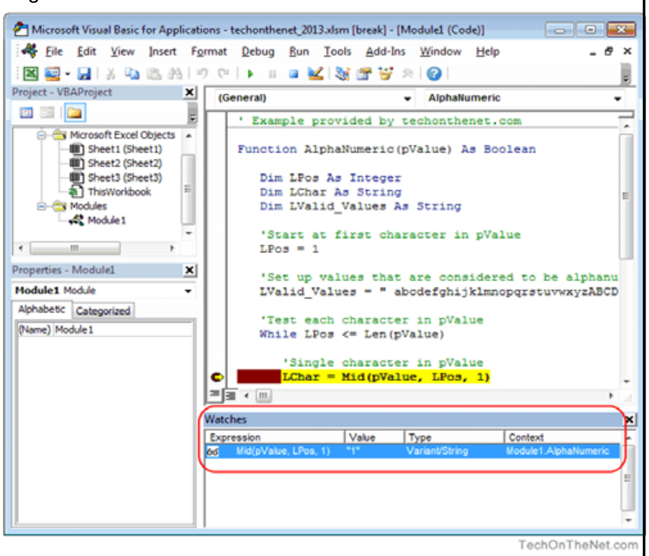
इसके बाद, हमने देखा अभिव्यक्ति के लिए संदर्भ सेट करते समय मॉड्यूल के रूप में मॉड्यूल 1 और प्रोसीजर के रूप में अल्फा न्यूमेरिक को चुना है।

अंत में, हमने वॉच एक्सप्रेशन को वॉच टाइप के रूप में चुना है लेकिन इसमें से चुनने के लिए 3 विकल्प हैं :

वॉच टाइप	विवरण
वॉच एक्सप्रेशन	वॉच एक्सप्रेशन के मान को वर्तमान स्थिति में प्रदर्शित करने के लिए।
जब मान सही है तो ब्रेक करें	कोड के निष्पादन को रोकने के लिए जब वॉच एक्सप्रेशन का मान सही है।
जब मान बदले तो ब्रेक करें	कोड को निष्पादन को रोकने के लिए जब वॉच एक्सप्रेशन का मान बदलता है।

जब VBA विंडो पर वापस आते हैं, तो वॉच विंडो स्वचालित रूप से दिखाई देगी यदि यह पहले छिपा हुआ था। वॉच विंडो के भीतर, सभी देखे गए एक्सप्रेशंस को सूचीबद्ध किया जाना चाहिए जिसमें हमने अभी जोड़ा है। (Fig 13)

Fig 13



जैसे कि आप देख सकते हैं, एक्स प्रेशन Mid(pValue, LPos, 1) अब वॉच विंडो में “1” के मान के साथ दिखाई देती है, वॉच को जोड़ना VBA कोड डीबग करते सयम चर या एक्सप्रेशन के भाव रखने का एक शानदार तरीका है।

ऑबजेक्ट ओरिएंटेड प्रोग्रामिंग अवधारणाएँ, क्लासस्, ऑब्जेक्ट्स, प्रॉपर्टीज और मेथड्स की अवधारणाएँ (Object Oriented Programming concepts, Concepts of classes, Objects, properties and Methods)

उद्देश्य : इस पाठ के अन्त में आप निम्नलिखित कार्य करने योग्य होंगे :

- **क्लास तथा ऑब्जेक्ट्स और इनकी विशेषताओं को समझाना**
- **VBA क्लास मॉड्यूल बनाम VBA सामान्य मॉड्यूल की व्याख्या करना**
- **एक क्लास मॉड्यूल के पार्ट्स और इसके प्रॉपर्टीज को सूचीबद्ध करना**
- **क्लास मॉड्यूल इवेंट्स की व्याख्या करना।**

परिचय (Introduction)

VBA क्लास मॉड्यूल उपयोगकर्ता को अपनी ऑब्जेक्ट्स को बचाने की अनुमति देता है। C# और जावा जैसे भाषाओं में, क्लासों का उपयोग ऑब्जेक्ट्स को बनाने के लिए किया जाता है। क्लास मॉड्यूल इन VBA क्लासों के समक्ष हैं। मुख्य अंतर यह है कि VBA क्लास मॉड्यूल में अन्य भाषाओं में क्लसों की तुलना में विरासत का बहुत सीमित प्रकार है। VBA में, विरासत C#\Java में इंटरफेस के समान तरीके से काम करती है।

VBA में हमारे पास क्लेक्शन, वर्कबुक, वर्कशीट आदि जैसे बिल इन वस्तुएं हैं। VBA क्लास मॉड्यूल का उद्देश्य हमें अपनी ऑब्जेक्ट्स को कस्टम बनाने के अनुमति देने है।

चलिए इस पोस्ट को शुरू करते हैं, यह देखते हुए कि हम ऑब्जेक्ट्स को पहली जगह क्यों उपयोग करते हैं।

इन्हेंरिटेस एक नई क्लास बनाने के लिए मौजूदा क्लास का उपयोग कर रही हैं।

इंटरफेस इन्हेंरिटेस का एक रूप है जो एक क्लास को विशिष्ट प्रोसीजर प्रॉपर्टीज को लागू करने के लिए मजबूर करता है।

ऑब्जेक्ट्स (Objects)

ऑब्जेक्ट्स का उपयोग करने से, हम अपने एप्लीकेशन को बनाने की अनुमति देते हैं जैसे कि हम बिल्डिंग ब्लॉक का उपयोग कर रहे हें।

विचार यह है कि प्रत्येक ऑब्जेक्ट का कोड स्वयं निहित है। यह हमारे आवेदन में किसी भी अन्य कोड से पूरी तरह से स्वतंत्र है।

ऑब्जेक्ट्स का उपयोग करने के लाभ (Advantages of Using Objects)

ब्लॉक के रूप में हमारे कोड के हिस्सों का इलाज करने से हमें बहुत सारे फायदे मिलते हैं

1 यह हमें एक समय में एक ब्लॉक, एक ब्लॉक बनाने की अनुमति देता है।

2 एप्लीकेशन के अलग-अलग हिस्सों का परीक्षण करना बहुत आसान है।

3 कोड अपडेट करने से एप्लिकेशन के अन्य हिस्सों में समस्याएं नहीं आतीं।

4 एप्लीकेशन के बीच ऑब्जेक्ट्स को जोड़ना आसान है।

ऑब्जेक्ट्स उपयोग करने के नुकसान (Disadvantages of Using Objects)

जीवन में ज्यादातर चीजों के साथ फायदे और नुकसान होते हैं। VBA क्लास मॉड्यूल का उपयोग करना अलग नहीं है। ऑब्जेक्ट्स को बनाने के लिए क्लास मॉड्यूल का उपयोग करने के नुकसान निम्नलिखित हैं।

1 एप्लीकेशन्स को बनाने के लिए शुरूआत में अधिक समय लगता है*।

2 स्पष्ट रूप से परिभाषित करना हमेशा आसान नहीं होता कि कोई ऑब्जेक्ट क्या है।

3 लोग जो क्लास और ऑब्जेक्ट्स के लिए नए हैं, उन्हें पहले समझना मुश्किल हो सकता है।

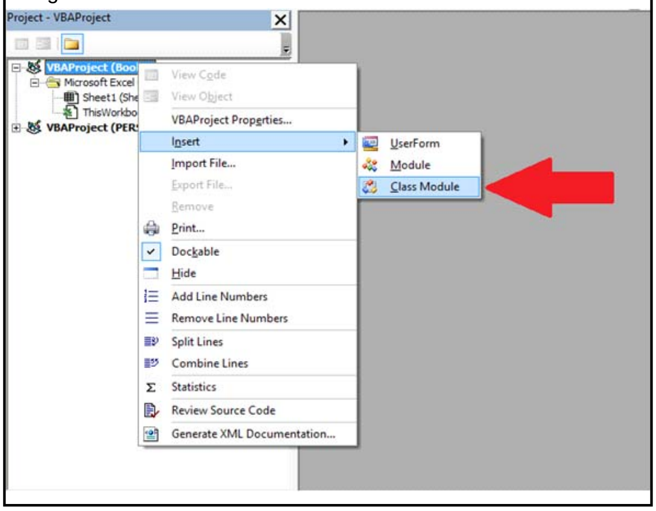
यदि ऑब्जेक्ट्स का उपयोग करके कोई एप्लिकेशन बनाते हैं तो इसे शुरू करने में अधिक समय लगेगा, इसे शुरू करने और इसे डिजाइन करने में अधिक समय व्यतीत करना होगा। हालांकि, लंबे समय तक, यह एक बड़ी माप में बचाएगा। कोड को प्रबंधित करना, अपडेट करना और पुनः उपयोग करना आसान होगा।

एक सरल क्लास मॉड्यूल बनाना (Creating a Simple Class Module)

आइए क्लास मॉड्यूल बनाने और हमारे कोड में इसका उपयोग करने का एक बहुत ही सरल उदाहरण देखें।

क्लास मॉड्यूल बनाने के लिए हम प्रोजेक्ट विंडो में राइट-क्लिक करें और फिर इंसर्ट पर क्लिक करके क्लास मॉड्यूल का चयन करें। (Fig 1)

Fig 1



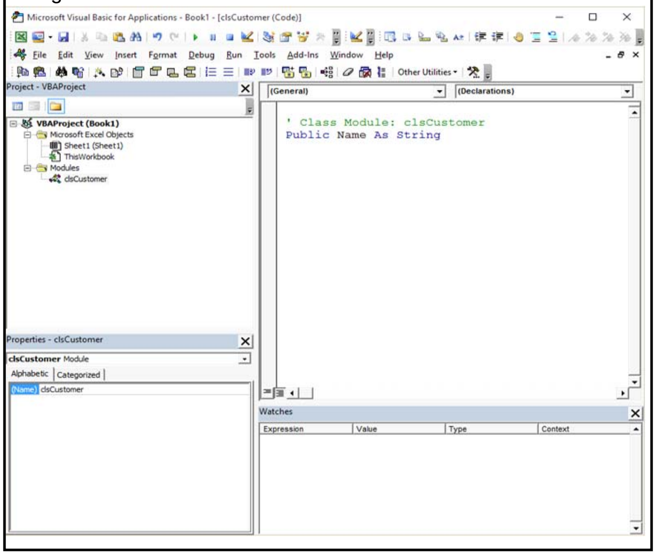
क्लास मॉड्यूल जोड़ना (Adding a Class Module)

हमारी नई क्लास को क्लास **1** कहा जाता है। हम प्रॉपर्टी विंडो में नाम बदल सकते है।

आइए क्लास मॉड्यूल का नाम **clsCustomer** में बदलें। फिर हम इस तरह के क्लास मॉड्यूल में एक चर जोड़ देंगे।

पब्लिक नाम स्ट्रिन्ग के समान (Public Name **AsString) (Fig 2)**

Fig 2



अब हम इस वर्कबुक में किसी भी मॉड्यूल (मानक या क्लास) में इस क्लास मॉड्यूल का उपयोग कर सकते है। उदाहरण के लिए

```
' Create the object from the class module
Dim oCustomer AsNew clsCustomer
' Set the customer name
oCustomer.Name = "John"
' Print the name to the Immediate Window(Ctrl + G)
Debug.Print oCustomer.Name
```

ऑब्जेक्ट्स बनाम क्लास मॉड्यूल (Class Module versus Objects)

जो लोग क्लास और VBA कक्षा मॉड्यूल का उपयोग करने के लिए नए हैं, अक्सर क्लास में क्या होता है और ऑब्जेक्ट्स क्या हैं, के बीच भ्रमित हो जाते हैं।

आइए एक वास्तविक दुनिया के उदाहरण को देखें। कॉफ़ी मग जैसे बड़े पैमाने पर उत्पादित आइटम के बारे में सोचें। मग का एक डिजाइन पहले बनाया गया है। फिर, इस डिजाइन से हजारों कॉफी मग बनाए जाते हैं।

यह क्लास मॉड्यूल और आब्जेक्ट्स कैसे काम करता है के समान है।

क्लास मॉड्यूल को डिजाइन के रूप में सोचा जा सकता है।

ऑब्जेक्ट को डिज़ाइन से बनाए गए आइटम के रूप में सोचा जा सकता है।

VBA में New कीवर्ड वह है जिसे हम क्लास मॉड्यूल से ऑब्जेक्ट बनाने के लिए उपयोग करते हैं। उदाहरण के लिए

```
' नए का उपयोग कर वस्तुओं का निर्माण
Dim oItem AsNew Class1
Dim oCustomer1 AsNewclsCustomer
Dim coll AsNew Collection
```

> **नोट : हम वर्कबुक और वर्कशीट जैसी वस्तुओं के साथ New का उपयोग नहीं करते हैं। अधिक जानकारी के लिए ''जब न्यूज की आवश्यकता नहीं है'' देखें।**

VBA क्लास मॉड्यूल बनाम VBA सामान्य मॉड्यूल (VBA Class Modules Versus VBA Normal Modules)

क्लास मॉड्यूल में कोड लिखना सामान्य मॉड्यूल में कोड लिखने जैसा ही है। हम एक ही कोड का उपयोग कर सकते हैं, जिसे हम सामान्य मॉड्यूल में उपयोग करते हैं। यह है, इस कोड का उपयोग कैसे किया जाता है जो बहुत अलग है।

आइए क्लास और सामान्य मॉड्यूल के बीच दो मुख्य अंतर देखें। ये अक्सर नए उपयोगकर्ताओं के बीच भ्रम पैदा करता है।

अंतर 1 - मॉड्यूल का उपयोग कैसे किया जाता है (Difference 1 – How the modules are used)

यदि किसी क्लास मॉड्यूल से सब/फंक्शन इत्यादि का उपयोग करना चाहते हैं तो पहले ऑब्जेक्ट बनाना होगा।

उदाहरण के लिए, उत्पना करें कि हमारे पास दो समान **PrintCustomer** sub हैं। एक क्लस मॉड्यूल में है और एक सामान्य मॉड्यूल में है।

```
' CLASS MODULE CODE - clsCustomer
Public Sub PrintCustomer()
Debug.Print "Sample Output"
End Sub

' NORMAL MODULE CODE
Public Sub PrintCustomer()
Debug.Print "Sample Output"
End Sub
```

आप ध्यान देंगे कि दोनों के लिए कोड बिल्कुल वही है।

क्लास मॉड्यूल से **PrintCustomer** सब का उपयोग करने के लिए, आपको पहले उस प्रकार का ऑब्जेक्ट बनाना होगा।

```
' Other Module
Sub UseCustomer()
Dim oCust AsNew clsCustomer
oCust.PrintCustomer
EndSub
```

सामान्य मॉड्यूल से **Print Customer** का उपयोग करने के लिए आप इसे सीधे कॉल कर सकते हैं।

‘ Other Module

Sub Use Customer()

Print Customer

End Sub

अंतर 2 - प्रतियों की संख्या (Difference 2 – Number of copies)

एक सामान्य मॉड्यूल में एक चर बनाने के दौरान इसकी केवल एक प्रति होती है। क्लास मॉड्यूल के लिए, चर की एक प्रति होती है, आपके द्वारा बनाए गए प्रत्येक ऑब्जेक्ट के लिए।

उदाहरण के लिए, कल्पना करें कि हम एक क्लास और सामान्य मॉड्यूल दोनों में एक चर **Student Name** बनाते हैं।

‘ NORMAL MODULE

Public StudentName **As String**

‘ CLASS MODULE

Public Studen tName **As String**

सामान्य मॉड्यूल चर के लिए हमारे एप्लीकेशन में केवल इस चर की एक प्रति होगी।

StudentName = “Ram”

क्लॉस मॉड्यूल के लिए चर **Student Name** की एक नई प्रति प्रत्येक बार एक नए ऑब्जेक्ट बनाये जाने पर बनाई जाती है।

Dim student1 **As New** clsStudent

Dim student 2 **As New** clsStudent

student1.Student Name = “Bill”

student2.Student Name = “Ted”

जब VBA क्लास मॉड्यूल पूरी तरह से समझते हें, तो ये अंतर स्पष्ट दिखाई देंगे।

क्लास मॉड्यूल के हिस्से (The Parts of a Class Module)

क्लास मॉड्यूल में चार अलग-अलग आइटम हैं। ये हैं :

1 **मेथड (Methods)** – functions/subs.

2 **मेम्बर वेरिएबल (Member variables)** – variables.

3 **प्रॉपर्टी (Properties)**– types of functions/subs that behave like variables.

4 **इवेंट्स (Events)** – subs that are triggered by an event.

और देख सकते हें कि वे सभी या तो फंक्शन, sub या चर है।

आइए हम उनसे निपटने से पहले कुछ उदाहरणों पर एक त्वरित नज़र डालें।

‘ CLASS MODULE CODE

‘ Member variable

Private dBalance **As Double**

‘ Properties

Property Get Balance () **AsDouble**

Balance = dBalance

EndProperty

Property Let Balance(dValue**As** Double)

dBalance = dValue

End Property

‘ Event - triggered when class created

Private Sub Class_Initialize()

dBalance = 100

EndSub

‘ Methods

Public Sub Withdraw (dAmount**As** Double)

dBalance = dBalance - dAmount

End Sub

Public Sub Deposit (dAmount**As** Double)

dBalance = dBalance + dAmount

EndSub

अब हमने उदाहरण देखें हैं, आइए इन दोनों को देखें।

क्लास मॉड्यूल मेथड्स (Class Module Methods)

मेथड्स क्लास के प्रोसीजर को संदर्भित करते हैं। VBA में, प्रोसीजर subs और फंक्शन हैं। सदस्य चर की तरह वे Public या Private हो सकते हैं।

आइए एक उदाहरण देखें

‘ CLASS MODULE CODE

‘ Class name: clsSimple

‘ Public प्रोसीजर को ऑब्जेक्ट के बाहर से कॉल किया जा सकता है।

Public Sub PrintText (sText**As** String)

Debug.PrintsText

EndSub

Public Function Calculate (dAmount**As** Double) **As Double**

Calculate = dAmount - GetDeduction

End Function

private प्रोसीजर को क्लास के अंदर से कॉल किया जा सकता है।

Private Function GetDeduction () **As Double**

GetDeduction = 2.78

EndFunction

हम **clsSimple** क्लास मॉड्यूल का उपयोग ऐसे कर सकते हैं

```
Sub Class Members ()
Dim oSimple As New clsSimple
oSimple.PrintText "Hello"
Dim dTotal As Double
dTotal = oSimple.Calculate(22.44)
Debug.Print dTotal
EndSub
```

क्लास मॉड्यूल मेम्बर चर (Class Module Member Variables)

मेम्बर चर सामान्य वेरिएबल के समान है जो हम VBA में उपयोग करते हैं। अंतर यह है कि हम **Dim** के बजाय **Public** या **Private** का उपयोग करते हैं।

```
' CLASS MODULE CODE
Private Balance AsDouble
Public AccountID As String
```

> **नोट : Dim और Private वही काम करता है लेकिन इरादा है, sub/functions में Dim का उपयोग करने के लिए और Private बाहरी sub/functions का उपयोग करने के लिए।**

Public कीवर्ड का अर्थ है, चर क्लास मॉड्यूल के बाहर से उपयोग किया जा सकता है। उदाहरण के लिए

```
Dim oAccount AsNew clsAccount
' Valid - AccountID is public
oAccount.AccountID = "499789"
' Error - Balance is private
oAccount.Balance = 678.90
```

उपर्युक्त उदाहरण में हम बैलेंस तक नहीं पहुँच सकते क्योंकि इसे **Private** के रूप में घोषित किया गया है। हम क्लास मॉड्यूल के भीतर केवल एक **Private** चर का उपयोग कर सकते हैं। हम क्लास मॉड्यूल में एक फंक्शन/सब में उपयोग कर सकते हैं। उदाहरण के लिए

```
' CLASS MODULE CODE
Private Balance As Double
Public SubSetBalance()
   Balance = 100
Debug.Print Balance
End Sub
```

पब्लिक सदस्य चर होना एक खराब अभ्यास माना जाता है। ऐसा इसलिए है क्योंकि ऑब्जेक्ट के बाहर कोड क्लास के अंदर काम करने के तरीके में हस्तक्षेप करने की इजाजत देता है। क्लास का उद्देश्य इता है कि डॉलर से क्या हाद्मे रहा है वह घुमाएँ।

सदस्य चर से सीधे उपयोगकर्ता से बात करने से बचने के लिए हम प्रॉपर्टीज का उपयोग करते हैं।

क्लास मॉड्यूल प्रॉपर्टी (Class Module Properties)

1 **गेट (Get)** – क्लास से ऑब्जेक्ट या मान देता है।

2 **लेट (Let)** – क्लास में एक मान निर्धारित करता है।

3 **सेट (Set)** – क्लास में एक ऑब्जेक्ट सेट करता है।

VBA प्रॉपर्टी के प्रारूप (Format of VBA Property)

प्रॉपर्टी के लिए सामान्य प्रारूप निम्नानुसार है :

```
Public Property Get () AsType
End Property
Public Property Let (varnameAsType )
End Property
Public PropertySet (varnameAsType )
EndProperty
```

हमने पहले ही देखा है कि प्रॉपर्टी सब का ही प्रकार है। प्रॉपर्टी का उद्देश्य कॉलर को मान प्राप्त करने और सेट करने के अनुमति देना है।

प्रॉपर्टी का उपयोग (Use of Properties)

कल्पना कीजिए कि हमारे पास एक क्लास है जो देशों की एक सूची बनाए रखता है हम सूची को एक ऐरे के रूप में स्टोर कर सकते हैं

```
' Use array to store countries
Public arrCountries As Variant
' Set size of array when class is initialized
Private Sub Class_Initialize()
ReDim arrCountries (1 To 1000)
End Sub
```

जब उपयोगकर्ता सूची में देशों की संख्या प्राप्त करना चाहता है तो वे ऐसा कर सकते हैं :

```
' NORMAL MODULE CODE
Dim oCountry As New clsCountry
' Get the number of items
NumCountries = UBound(oCountry.arrCountries) + 1
```

उपरोक्त कोड के साथ दो प्रमुख समस्याएं हैं (There are two major problems with the above code)

1 देशों की संख्या प्राप्त करने के लिए, आपको यह जानना होगा कि सूची कैसे संग्रहित की जाती है। जैसे ऐरे।

2 अगर हम कलेक्शन में ऐरे को बदलते हैं, तो हमें ऐरे को संदर्भित करने वाले सभी कोड को बदलने की आवश्यकता है।

इन समस्याओं को हल करने के लिए, हम देशों की संख्या वापस करने के लिए एक फंक्शन बना सकते हैं

```
' CLASS MODULE CODE - clsCountryList
' Array
```

Private arrCountries () **As String**

Public Function Count () **AsLong**

Count = UBound(arrCountries) + 1

End Function

We then use it like this

‘ MODULE CODE

Dim oCountries **As New** clsCountries

Debug.Print “Number of countries is “ &oCountries.Count

यह कोड ऊपर सूचीबत्रब्द्ध दो समस्याओं को हल करता है। हम अपने ऐरे को एक कलेक्शन में बदल सकते हैं और कॉलर कोड अभी भी काम करेगा। जैसे।

‘ CLASS MODULE CODE

‘ Collection

Private collCountries() **As** Collection

Public Function Count() **AsLong**

Count = collCountries.Count

End Function

कॉलर इस बात से अनजान है कि देशों को कैसे संग्रहीत किया जाता है। सभी कॉलर को यह जानने की जरूरत है कि **Count** फंक्शन देशों की संख्या वापस कर देगा।

जैसा कि हमने अभी देखा है, सब-फंक्शन उपरोक्त समस्याओं का समाधान प्रदान करता है। हालांकि, एक प्रॉपर्टी का उपयोग एक और अधिक सुरूचिपूर्ण समाधान प्रदान कर सकते हैं। एक फंक्शन/सब के बजाय **Property** का उपयोग करना

Using a Property instead of a Function/Sub

एक **Count** फंक्शन बनाने के बजाय हम एक **Count** प्रॉपर्टी बना सकते हैं। जैसे कि आप नीचे देख सकते हैं वे बहुत समान हैं।

‘ Replace this

Public Function Count() **As Long**

Count = UBound(arrCountries) + 1

End Function

‘ With this

Property Get Count () **As Long**

Count = UBound(arrCountries) + 1

End Function

इस परिदृश्य में, प्रॉपर्टी का उपयोग करने और फंक्शन का उपयोग करने में बहुत अंतर नहीं है। हालांकि, मतभेद हैं। हम आम तौर पर इस तरह गेट और लेट प्रॉपर्टी बनाते हैं।

‘ CLASS MODULE CODE - clsAccount

Privated TotalCost **As Double**

Property Get TotalCost () **As Long**

Total Cost= dTotalCost

End Property

Property Let Total Cost (dValue **As** Long)

dTotal Cost = dValue

End Property

Let का प्रयोग हमें प्रॉपर्टी को चर की तरह इस्तेमाल करने की अनुमति देता है। तो हम यह कर सकते हैं

oAccount.Total Cost = 6

दुसरा अंतर यह है कि **Let** और **Get** प्रॉपर्टी इस प्रॉपर्टी का उपयोग करते समय एक ही नाम का उपयोग करने की अनुमति प्रदान करती है। इसलिए हम इस प्रॉपर्टी को चर के रूप में उपयोग कर सकते हैं। यह इन प्रॉपर्टी का sub/function के उपर फायदा है।

oAccount.TotalCost = 6

dValue = oAccount.TotalCost

अगर हम इसे फंक्शन और सब का उपयोग करते हैं तो हम एक चर के व्यवहार को प्राप्त नहीं कर सकते हैं। इसके बजाय हमें दो अलग-अलग प्रोसीजर को कॉल करना होगा। जैसे

oAccount.SetTotalCost 6

dValue = oAccount.GetTotalCost

आप यह भी देख सकते हैं कि जब हमने Let इस्तेमाल किया। हम वैरिएबल की तरह मान असाइन कर सकते हैं। जब हम था **Set Total Cost** का उपयोग करते हैं, तो हमें इसे पैरामीटर के रूप में पास करना होता।

संक्षेप में प्रॉपर्टी (The Property in a Nutshell)

1 प्रॉपर्टी कॉलर से कार्यान्वियन के विवरण छुपाती है।

2 प्रॉपर्टी हमें वैरिएबल के समान व्यवहार प्रदान करने की अनुमति देती है।

वीबीए प्रॉपर्टी के प्रकार (Types of VBA Property)

तीन प्रकार के प्रॉपर्टी हैं। हमने पहले ही Get और Let देखा है। जो हमने देखा नहीं है वह **Set** हैं।

Set Let के समान हैं लेकिन इसका उपयोग किसी आब्जेक्ट के लिए किया जाता है (इस बारे में अधिक जानकारी के लिए VBA ऑब्जेक्ट्स को असाइन करना देखें)।

मूल रूप से विजुअल बेसिक में, **Let** कीवर्ड का प्रयोग वैरिएबल असाइन करने के लिए किया जाता है वास्तव में, अगर हम चाहें तो हम अभी भी इसका इस्तेमाल कर सकते हैं।

‘ These line are equivalent

Let a = 7

a = 7

इसलिए हम एक चर को मान निर्दिष्ट करने के लिए उपयोग करते हैं और हम ऑब्जेक्ट वैरिएबल को किसी आब्जेक्टस को असाइन करने के लिए **Set** का उपयोग करते हैं।

```
' Using Let
Dim a As Long
Let a = 7
' Using Set
Dim coll1  As Collection, coll2 As Collection
Set coll1 = New Collection
Set coll2 = coll1
```

- **Let** is मूल चर के प्रकार को मान असाइन करता है
- **Set** ऑब्जेक्ट वेरिएबल को ऑब्जेक्ट असाइन करता है।

निम्नलिखित उदाहरण में हम स्ट्रिंग वेरिएबल की **Get** और **Let** प्रॉपर्टी देखेंगे।

```
' CLASS MODULE CODE
' SET/LET PROPERTIES for a variable
Private m_sName As String
' Get/Let Properties
Property Get Name() As String
   Name = m_sName
End Property
Property Let Name (sNameAs String)
m_sName = sName
End Property
```

We can then use the **Name** properties like this

```
Sub Test Let Set()
Dim sName As String
Dim coll As New Collection
Dim oCurrency As New clsCurrency
' Let Property
oCurrency.Name = "USD"
' Get Property
sName = oCurrency.Name
End Sub
```

अगले उदाहरण में हम ऑब्जेक्ट वेरिएबल की **Get** और **Set** प्रॉपर्टी देखेंगे।

```
' CLASS MODULE CODE
Private m_collPrices As Collection
' Get/Set Properties
Property Get Prices() As Collection
Set Prices = m_collPrices
End Property
Property Set Prices (collPricesAs Collection)
Set m_collPrices = collPrices
End Property
```

हम प्रॉपर्टी का ऐसे देख सकते हैं

```
Sub Test Let Set ()
Dim coll1 As New Collection
Dim oCurrency As New cls Currency
' Set Property
Set oCurrency.Prices = coll1
' Get Property
Dim coll2 As Collection
Set Coll2 = oCurrency.Prices
EndSub
```

हम दोनों वस्तुओं के मानों को वापस करने के लिए Get प्रॉपर्टी का उपयोग करते हैं। ध्यान दें कि भले ही हम कलेक्शन को वापस करने के लिए **Get** प्रॉपर्टी का उपयोग करते है, फिर भी हमें इसे असाइन करने के लिए सेट कीवर्ड का उपयोग करने की आवश्यकता है।

क्लास मॉड्यूल इवेंट्स (Class Module Events)

एक क्लास मॉड्यूल में दो इवेंट्स होती है।

1 **इनिशियलाइज (Initialize)** – यह तब होता है जब क्लास का एक नया ऑब्जेक्ट बनाया जाता है

2 **टर्मिनेट (Terminate)** – तब होता है जब क्लास ऑब्जेक्ट हटा दिया जाता है।

ऑब्जेक्ट ओरिएंटेड भाषाओं C++, इन इवेंट्स को कन्स्ट्रक्टर और डिस्ट्रक्टर के रूप में जाना जाता है अधिकांश भाषाओं में, आप एक कन्स्ट्रक्टर को पैरामीटर पास कर सकते हैं लेकिन VBA में आप नहीं कर सकते हैं। हम इस मुद्दे को हल करने के लिए क्लास फैक्ट्री का उपयोग कर सकते हैं जैसा कि हम नीचे देखेंगे

इनिशियलाइज (Initialize)

आइए प्रारंभिक और समाप्ति इवेंट्स के साथ clsSimple नामक एक बहुत ही सरल क्लास मॉड्यूल बनाएं।

```
' CLASS MODULE CODE
Private SubClass_Initialize()
Msg Box "Class is being initialized"
End Sub
Private SubClass_Terminate()
Msg Box  "Class is being terminated"
End Sub
Public Sub Print Hello ()
Debug.Print "Hello"
End Sub
```

निम्नलिखित उदाहरण में ऑब्जेक्ट बनाने के लिए **Dim** और **New** का उपयोग करेंगे।

इस मामले में, **oSimple** तब तक नहीं बनाया जाता जब तक कि हम इसे पहली बार संदर्भित नहीं करते हैं उदा।

```
Sub Class Event sInit2 ()
Dim oSimple As New clsSimple
' Initialize occurs here
oSimple.PrintHello
EndSub
```

जब हम सेट और न्यू का उपयोग करते हैं तो व्यवहार अलग होता है। इस मामले में, ऑब्जेक्ट तब बनाया जाता है जब सेट का उपयोग किया जाता है उदा।

```
Sub Class Events Init()
Dim oSimple As clsSimple
' Initialize occurs here
Set oSimple = New clsSimple
oSimple.PrintHello
End Sub
```

नोट : New के साथ Dim का उपयोग और सेट के साथ New उपयोग करने के बीच में अंतर के बारे में अधिक जानकारी के लिए <u>Dim</u> बनाम <u>Set</u> के सूक्ष्म मतभेद देखें।

जैसा कि पहले कहा गया था, आप इनिशियलाइज में पैरामीटर पास नहीं कर सकते हैं। यदि आपको ऐसा करने की आवश्यकता है तो आपको ऑब्जेक्ट को पहले बनाने के लिए एक फंक्शन की आवश्यकता है

```
' CLASS MODULE - clsSimple
Public Sub Init (Price As Double)
EndSub
' NORMAL MODULE
PublicSubTest()
' Use CreateSimpleObject function
Dim oSimple As clsSimple
Set oSimple = CreateSimpleObject(199.99)
End Sub
Public Function CreateSimpleObject(Price As Double) As clsSimple
Dim oSimple As New clsSimple
oSimple.Init Price
Set CreateSimpleObject = oSimple
End Function
```

हम क्लास फैक्ट्री बनाए के लिए इसे CreateSimpleObject उदारहण <u>2</u> में समझायेंगे

टर्मिनेट (Terminate)

टर्मिनेट इवेंट तब होता है जब क्लास डिलीट हो जाती है। यह तब होता है जब हम इसे **Nothing** सेट करते हैं।

```
Sub Class EventsTerm ()
Dim oSimple As clsSimple
Set oSimple = NewclsSimple
' Terminate occurs here
```

```
Set oSimple = Nothing
End Sub
```

यदि हम ऑब्जेक्ट को **Nothing** पर सेट करते हैं तो VBA स्कोप से बाहर होने पर इसे स्वचालित रूप से हटा देगा।

इसका अर्थ यह है कि यदि हम किसी प्रोसीजर में ऑब्जेक्ट बनाते हैं तो जब यह प्रोसीजर समाप्त होती है तो VBA बनाए गए किसी भी ऑब्जेक्ट को हटा देगा।

```
Sub Class EventsTerm2()
Dim oSimple As New clsSimple
' Initialize occurs here
oSimple.PrintHello
' oSimple is deleted when we exit this Sub calling Terminate
EndSub
```