## الإجراءات والوظائف Procedures and Functions :

### 1- الإجراءات Procedure :

The essential part of structure programming is divided the program up into smaller procedure or modules, this make the program easier to develop, maintain& debug.

في البرمجة المهيكلة يقسم البرنامج إلى وحدات أو إجراءات، وهذا التشكيل يجعل البرنامج أكثر سهولة في عملية التطوير والصيانة وتصحيح الأخطاء.

### ملف الإجراءات Procedures File :

Contain modules of program codes that stay in memory for the duration of your program.

يحتوي ملف الإجراءات على رموز برمجية للوحدات والتي تبقى في الذاكرة خلال تنفيذ البرنامج.

والإجراء هو برنامج فرعي Sub-program داخل البرنامج الرئيسي Main-program. يمكن للبرنامج أن يحتوي على أكثر من إجراء، ولكن يجب أن يختلف اسم كل إجراء عن الآخر.

**الصيغة العامة للإجراء:**

```
PROCEDURE <Procedure Name>
[   LPARAMETERS parameter1 [ ,parameter2] , … ]
     Commands
[   RETURN  [ eExpression ]   ]
```

**DO (procedure name) WITH ( parameter list )**

**حيث أن:**

**Procedure Name** : يمثل اسم اختياري لهذا الإجراء.

**Parameters** : لإرسال بيانات من البرنامج الذي يستدعي الإجراء إلى المتغيرات المحلية الموجودة داخل الإجراء.

**Return** : جملة تحتوي على ما يسترجعه هذا الإجراء من بيانات.

**Do**: يستخدم لتنفيذ الإجراء مع إرسال قيم المعلمات للإجراء.

**Ex:**

**Write a procedure to add two numbers?**

```
PROCEDURE procAdd
parameters n1,n2
r=0
r=n1+n2
```

```
RETURN r
```

عند الاستدعاء :

```
Do procAdd  WITH 10, 20
```

**Ex:**

**Write a procedure to calculate the average of four numbers?**

```
PROCEDURE procAvg
parameters n1,n2, n3, n4
r=0
r=(n1+n2+n3+n4)/4
RETURN r
```

عند الاستدعاء :

```
Do procAvg  WITH 70, 40, 80, 30
```

**2- الوظائف Functions:**

الوظيفة هي برنامج فرعي Sub-program يوجد في داخل البرنامج الرئيسي Main-program وظيفتها تشابه وظيفة الأجراء مع بعض الاختلافات التي سوف نذكرها لاحقاً.

**الصيغة العامة للوظيفة :**

**FUNCTION <FunctionName >**
**[ LPARAMETERS parameter1 [ ,parameter2  ] , …]**
**Commands**
**[ RETURN [ eExpression  ]**

**Ex:**

**Write a function to add two numbers?**

```
FUNCTION funcAdd
parameters n1,n2
r=0
r=n1+n2
RETURN r
```

عند الاستدعاء :

```
s= funcAdd (10,20)
?s
```

**Ex:**

**Write a function to calculate the maximum value between two numbers?**

```
FUNCTION funcMax
parameters n1,n2
If (n1 > n2)
```

```
RETURN n1
ELSE
RETURN n2
```

عند الاستدعاء :

```
s= funcMax (10,20)
?s
```

**تلميحات:**

- يمكن عدم كتابة جملة الرجوع Return مع الوظيفة، وأن القيمة التي سيتم إرجاعها تكون منطقية ويتم استدعاء الوظيفة مباشرة بكتابة الاسم والقوسين بعدهم. مثل ( ) Av
- يمكن عدم إرسال قيم بيانية للوظيفة وعدم كتابة المعلمات، وعند الاستدعاء سوف تترك الأقواس فارغة. مثل ( )Avg = Av

**Write a function to print a message "Hello World"**

```
FUNCTION funcMsg
? "Hello World"
```

عند الاستدعاء :

```
funcMsg(  )
```

## الفرق بين الإجراءات و الوظائف Procedures Vs. Functions

Syntactically there is no difference between FUNCTION and ROCEDURE statements in VFP. In other languages, that have both statements, a function can return a value and a procedure cannot. In VFP the difference comes in the way a procedure/function is called. By default VFP passes parameters by value in call to the functions, by reference to the procedures.

من خلال بناء الجملة نلاحظ أنه ليس هناك فرق بين الإجراءات والوظائف في VFP، ولكن في اللغات الأخرى التي تحتوي على هذه الجمل يمكن للوظيفة إرجاع قيمة من خلال Return ولا يمكن للإجراء عمل ذلك. ولكن في VFP يكون الفرق من خلال الطريقة التي يتم استدعاء الإجراء أو الوظيفة. فبشكل افتراضي تمرر المعلمات من حيث القيمة By Value عند الاستدعاء في الوظائف، وتمرر المعلمات من حيث المصدر By Reference عند الاستدعاء في الإجراءات مع استخدام كلمة Do عند استدعاء الإجراء.

## الفرق بين الاستدعاء بالقيمة والمصدر Value Vs. Reference

**1- الاستدعاء بالقيمة ( By Value ) :**

وهو الوضع الذي تعمل فيه الوظائف، ويتم من خلال إرسال القيم للوظيفة عن طريق المتغيرات الوسيطة، أنظر للمثال التالي والذي يحسب متوسط عددين ( لا يعيد قيمة ) وبفرض أن x و y هما العددين الذين سنحسب متوسطهما :

---

**Ex:**

**Write a function to calculate average of two numbers?**

```
Function funcAvg
parameters a,b
r=0
r=(a+b)/2
? r
```

عند الاستدعاء :

```
x =50
y=30
funcMax (x,y)
```

عند الاستدعاء سيتم إجراء نسخة عن كل من x و y و إرسالها إلى المتغيرات الوسيطة a و b الخاصين بالوظيفة funcAvg، وبالتالي لن يتأثر المتغيرين x و y بنتائج عمليات الوظيفة ( أي لن تتغير قيمتهما حتى بعد الانتهاء) لأن المترجم قام بنسخ قيم x و y و وضعها في المتغيرين الوسيطين a و b ، و بالتالي نستنتج أيضاً أنه سيتم تحميل الذاكرة بقيمتين مشابهتين تماماً لقيم x و y .

**1- الاستدعاء بالمصدر ( By Reference ) :**

في بعض الأحيان نحتاج لإرسال بيانات كبيرة ( كالكائنات أو السجلات مثلاً ) عبر تلك المتغيرات الوسيطة، ففي الاستدعاء بالقيمة By Value سيتم إجراء نسخة أخرى لتلك القيم، وبهذا سيزداد العبء على الذاكرة و سيزداد حجمها، والحل لمعالجة هذه المشكلة هو عن طريق الاستدعاء بالمصدر، يختلف الاستدعاء بالمصدر عن الاستدعاء بالقيمة بأنه سيقوم بالتعامل مع عنوان المتغير الوسيط الفعلي في الذاكرة، وبالتالي إذا حدثت أي تغييرات على المتغير الوسيط الشكلي فإن هذا التغير سيطرأ أيضاً على المتغير الوسيط الفعلي، أي أن قيمة المتغير الوسيط

الشكلي ستعطى للمتغير الوسيط الفعلي. أنظر للمثال التالي والذي يعطي قيمة تالية للعدد الذي نعطيه:

**Ex:**
**Write a procedure to increase a number?**

```
PROCEDURE procInc
parameters a
a = a + 1
RETURN a
```

عند الاستدعاء :

```
x = 10
Do procInc  WITH  x
```

ستصبح قيمة x هي 11 متأثرة بعمليات الإجراء.