

RedCrab^{PLUS}

The Calculator

Programers Manual

Version 4.20

copyright © by Redchillicrab, Singapore 2009, 2010, 2011,2012

Inhalt

- 1.1 Einleitung
- 1.2 RedCrab Program Interpreter
- 1.3 Kommentare
- 1.4 Identifikatoren
- 1.5 Gültigkeitsbereich eines Identifikators
- 1.6 String Konstante
- 1.7 Programmvariable
- 1.8 Numerische Ausdrücke
- 1.9 Boolesche Ausdrücke
- 1.10 Ausdrücke
- 1.11 Mehrzeilige Anweisungen

- 2.1 Program
- 2.2 Uses
- 2.3 Define
- 2.4 Let
- 2.5 While do
- 2.6 If Then
- 2.7 Else
- 2.8 Elseif
- 2.9 Function
- 2.10 Forward
- 2.11 Call
- 2.12 Result

- 3.1 PHP Programm Schnittstelle
- 3.2 PHP Input
- 3.3 PHP Output

1.1 Einleitung

In *RedCrab* können ein oder mehrere Datei (Module) geöffnet werden, die programmierte Funktionen oder Daten enthalten. Diese Anleitung beschreibt die Syntax des *RedCrab* Interpreters (*RCI*) und die *PHP* Schnittstelle.

1.2 RedCrab Program Interpreter

RedCrab verwendet eine eigene einfache Programmiersprache. Auch Benutzern ohne Programmier-Kenntnisse sollen in der Lage sein nach kurzer Einarbeitung einfache Funktionen zu schreiben.

Analog zum Arbeitsblatt wird bei reservierten Namen (Systemfunktionen und Anweisungen) nicht zwischen Groß- und Kleinschreibung unterschieden. Bei selbst-definierten Funktionen und Variablen wird Groß- und Kleinschreibung berücksichtigt.

Ein Programm besteht aus einer Folge von Anweisungen. Jede Anweisung beginnt mit einem Schlüsselwort. Das Zeilenende (*Linefeed*) beendet eine Anweisung. Die Schlüsselwörter *function*, *if* und *while* gelten für alle folgenden Anweisungen, bis der Anweisungsblock mit dem Schlüsselwort *end* beendet wird. Extra Leerzeichen, Tabulatoren, Linefeeds und Kommentare werden vom Programm ignoriert.

Beispiel: **Let** a = 12
 Let b = 22

Es können mehrere Anweisungen in eine Zeile geschrieben werden, wenn sie mit einem Doppelpunkt getrennt werden.

Beispiel: **Let** a = 123 : **Let** b = 22

1.3 Kommentare

Aus Gründen der Übersicht und zur Beschreibung der Funktionen, wird empfohlen, dass Sie Ihren Code dokumentieren, indem Sie Kommentare einfügen. Sie können auch mit Kommentar Symbole während der Programmentwicklung Teile des Programms deaktivieren ohne sie zu löschen. Kommentare können auf zweierlei Weise eingefügt werden:

- Ein Kommentar kann mit der runden Klammer, gefolgt von dem Multiplikation Symbol (*) beginnen und mit den Zeichen *) enden. Sie können mit den Symbolen keine Kommentare verschachteln.
- Sie können einen Kommentar mit einem doppelter Schrägstrich einleiten //. Der Kommentar endet mit dem Ende der Zeile.

1.4 Identifikatoren

RedCrab Programme enthalten Verweise auf Module, Funktionen, lokale und globale Variablen und Konstanten. Mit Ausnahme von Konstanten, wird jeder Verweis über seinen Name identifiziert. Der Name besteht aus eine Folge von Buchstaben, Ziffern und Unterstrichen. Das erste Zeichen muss ein Buchstabe oder eine Unterstrich sein.

1.5 Gültigkeitsbereich eines Identifikators

Lokale Variablen werden innerhalb einer Funktion definiert. Auf Sie kann nicht von außerhalb ihrer Funktion zugegriffen werden.

Globale Variablen werden außerhalb einer Funktion definiert. Auf sie kann von allen Funktionen im Modul zugegriffen werden. Von externen Modulen und vom Arbeitsblatt können sie nur ausgelesen werden.

Funktionen können von allen Modulen und vom Arbeitsblatt aufgerufen werden.

1.6 String Konstante

String Konstante sind Sequenzen von Zeichen die in Anführungszeichen eingeschlossen sind. Strings müssen in einer Zeile geschrieben werden. Mit dem Punkt (.) können einzelne String Konstante miteinander verbunden werden.

Beispiel: `Let s = "Hallo "`
`Let t = s ."Welt"`

Die Variable *t* enthält den Text: „*Hallo Welt*“

1.7 Programmvariable

Programmvariable müssen definiert werden , bevor sie verwendet werden. Dazu wird das Schlüsselwort *define* verwendet. Optional kann in der Definition auch ein Wert zugewiesen werden. Für weitere Informationen lesen Sie unten die Beschreibung zu *define*.

1.8 Numerische Ausdrücke

Ein numerischer Ausdruck besteht aus einer Konstante, Variable, Zelle ein Datenfeldes oder einer Funktion die einen Zahlenwert liefert, oder mehrere davon die durch folgende arithmetische Operatoren verbunden sind:

- * Multiplikation
- / Division
- Mod Modulo
- Div Integer Division
- + Addition
- Subtraktion

1.9 Boolesche Ausdrücke

Ein boolescher Wert bewertet einen Ausdruck als wahr (*TRUE*) oder unwahr (*FALSE*) und hat folgendes Format:

Ausdruck Operator Ausdruck

Die Ausdrücke können numerisch oder Text Strings sein. Wenn ein numerischer Ausdruck mit einem String verglichen wird der eine Zahl enthält, wird zum Vergleich der Wert der Zahl verwendet. Werden zwei Strings miteinander verglichen, werden sie immer als Strings behandelt, unabhängig davon, ob sie Text oder Zahlen beinhalten. Die Operatoren zeigt die folgende Liste.

Operator	Operation
==	Gleich
<>	Ungleich
>	Größer als
>=	Größer oder gleich.
<	Kleiner als
<=	Kleiner oder gleich

Boolesche Ausdrücke können mit den *UND*(&) und *ODER*(/) Operatoren zusammengefaßt werden.

Beispiel: $(a \geq b) \& (c \leq d)$
 $(a == d) \mid c$

1.10 Ausdrücke

Für *if* und *while*-Anweisungen ist *TRUE* jede Zahl ungleich Null und *FALSE* ist gleich Null. Deshalb können Sie immer einen numerischen Ausdruck einsetzen, wo ein boolescher Ausdruck gefragt ist. Sie können ebenso einen booleschen Ausdruck verwenden, wo ein numerischer Ausdruck erwartet wird, was als 1 oder

0 interpretiert wird. Sie können einen String-Ausdruck, der eine Zahl darstellt einsetzen wo ein numerischer Ausdruck erlaubt ist.

1.11 Mehrzeilige Anweisungen

Eine Feld-Definition kann in der nächsten Zeile fortgesetzt werden. Die aktuelle Zeile muß mit einem Komma oder einem Semikolon beendet werden.

Ein Statement kann in der folgenden Zeile fortgesetzt werden, wenn die aktuelle Zeile mit einem Backslash (\) beendet wird.

Eine lange Zahl kann in der folgenden Zeile fortgesetzt werden, wenn die aktuelle Zeile mit einem doppelten Backslash (\\) beendet wird.

Die folgende Tabelle zeigt einige Beispiele zu mehrzeiligen Statements:

Zweizeilige Statements	Interpretation
<pre>Let m = [1,2,3; 4,5,6]</pre>	<pre>Let m = [1,2,3;4,5,6]</pre>
<pre>Let m = [1,2,3,4, 5,6,7,8]</pre>	<pre>Let m = [1,2,3,4,5,6,7,8]</pre>
<pre>Let v \ = 1 + 2</pre>	<pre>Let v = 1 + 2</pre>
<pre>Let v = 12345\ 6789</pre>	<pre>Let v = 123456789</pre>
<pre>Let s = "hello " \ ."world"</pre>	<pre>Let s = "hello " ."world"</pre>

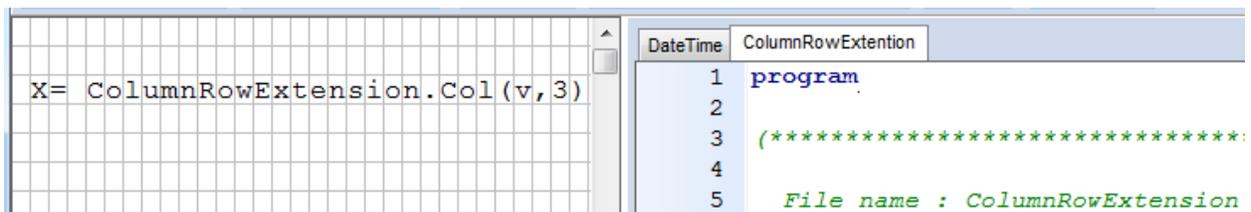
2.1 Programm

Eine Programmdatei wird immer mit dem Schlüsselwort *program* eingeleitet. Optional kann ein Name angegeben werden. Die Programmdatei wird in einer Registerseite unter ihrem Dateinamen angezeigt, oder unter dem optional angegebenen Namen.

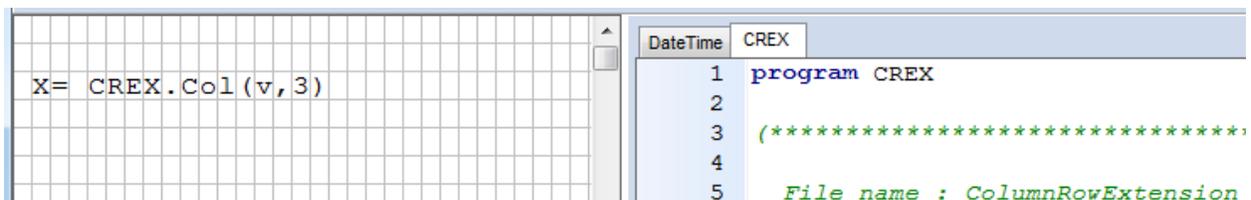
Die Syntax des Schlüsselworts *Program* ist:

```
Program  
Program name
```

In dem folgenden Beispiel wird aus dem Arbeitsblatt die Funktion *Col* des Moduls *ColumnRowExtension* aufgerufen. Die Programmdatei wird über den Namen des Registers qualifiziert.



Komfortabler ist der Aufruf der Funktion *Col* in dem folgenden Beispiel, in dem ein kurzer Programmname angegeben ist. Beim Laden des Moduls wird die Registerseite nach dem Programmnamen statt des Dateinamens benannt.



2.2 Uses

Die *uses* Anweisung enthält eine Liste externer Module (Dateinamen), die von den Funktionen in diesem Modul genutzt werden. Die *uses* Anweisung muß der *program* Anweisung unmittelbar folgen. Die einzelnen Modulnamen werden durch Komma getrennt.

Die Syntax der *uses* Anweisung ist:

```
uses module1, module2
```

2.3 Define

Die *define* Anweisung deklariert den Namen einer Variablen und weist ihr optional einen Wert zu. Wenn kein Wert zugewiesen wird, ist der Wert der Variable Null. Eine Variable kann nur verwendet werden, wenn der Name zuvor per *define* Anweisung oder in der Parameterliste einer Funktion deklariert wurde.

Die Syntax der *define* Anweisung ist:

```
define Name  
define Name = Value
```

2.4 Let

Let weist einer Variable einen Ausdruck zu. Die Syntax der *Let* Anweisung ist:

```
let variable = Ausdruck
```

Der Ausdruck bestehen aus einer oder mehreren Konstanten, Variablen oder Zellen eines Datenfeldes. Der Wert kann eine numerischer oder boolescher Ausdruck oder eine Text String sein.

Beispiel:

```
Let x = 12
Let x = y
Let x = (12 + y) * z
Let x = sin(45)
Let x = "hello"
```

2.5 While do

Mit *While* wird die Folge von Anweisungen (statements) zwischen *do* und *end* wiederholt, solange die Aussage der Bedingung (expression) wahr, bzw. ungleich Null ist. Wenn die Aussage falsch bzw. Null ist, wird das Program mit der Anweisung nach *End* fortgeführt.

Die Syntax der *While do* Anweisung ist:

```
while expression do
  statements....
end
```

Beispiel:

```
Let i = 0
While i < 100 do
  Statement Sequence....
  Let i = i + 1
End
```

2.6 If Then

If Then bietet die Möglichkeit einer bedingten Programmverzweigung. Die Anweisungen (statements) zwischen *then* und *end* werden nur ausgeführt, wenn der Ausdruck (expression) einen Wert ungleich Null (*TRUE*) liefert. Sonst wird der Anweisungsblock übersprungen und das Program mit der Anweisung nach *end* fortgesetzt.

Die Syntax der *if then* Anweisung ist:

```
if expression then  
    statements....  
end
```

2.7 Else

Die *else* Anweisung ist eine Erweiterung der *if then* Anweisung. Die Syntax der *if then else* Anweisung ist:

```
if expression then  
    statements....  
else  
    statements.  
end
```

Wenn der Ausdruck zwischen *if* und *then* Null (*FALSE*) ergibt werden die Anweisungen (statements) zwischen *then* und *else* übersprungen und statt dessen die Anweisungen zwischen *else* und *end* ausgeführt. Wenn der Ausdruck zwischen *if* und *then* ungleich Null (*TRUE*) ergibt wird der Block zwischen *if* und *then* ausgeführt und der Block zwischen *else* und *end* übersprungen.

2.8 Elseif

Mit der Anweisung *elseif* können weitere Bedingungen für eine Programverzweigung programmiert werden. Der auf *elseif* folgende Ausdruck wird nur ausgewertet, wenn die vorherigen *if* und *elseif* Ausdrücke den Wert Null (*FALSE*) ergeben. Wenn der Ausdruck einen Wert ungleich Null (*True*) ergibt, werden die auf *then* folgenden Anweisungen (statements) bis zur nächsten *elseif* oder *else* Anweisung ausgeführt. Dann wird das Programm nach *end* ausgeführt.

Die Syntax der *Elseif* Anweisung ist:

```
if expression then  
  statements....  
elseif  
  statements.  
elseif  
  statements.  
else  
  statements.  
end
```

2.9 Function

Die *function* Anweisung definiert eine Funktion. Eine Funktion ist ein benannter Block von Anweisungen. Sie kann vom Arbeitsblatt, aus anderen Funktionen des Programms, oder aus jedem anderen Programm-Modul namentlich aufgerufen werden. Eine Funktion liefert als Resultat einen einzelnen Wert, ein Datenfeld oder einen Text-String.

Die Syntax der *function* Anweisung ist:

```
function Name (argument, argument.....)  
  statements...  
end
```

Die declarierten Argumente (*arguments*) in der Parameterliste können ohne *define* Anweisung in der Funktion verwendet werden.

2.10 Forward

In einem Programm können Funktionen nur aufgerufen werden, wenn sie vorher declariert sind. Der Zweck einer *forward*-Deklaration ist es, eine Funktion zu deklarieren die erst weiter unten im Quellcode implementiert ist. Dadurch können andere Funktionen, die vorwärts-deklarierte Routine aufrufen, bevor sie tatsächlich

definiert ist. Dieses ist zum Beispiel notwendig, wenn Funktionen sich gegenseitig rekursiv aufrufen.

Die Syntax der *forward*-Anweisung lautet:

```
forward Name
```

2.11 Call

Die *call*-Anweisung ruft eine Funktion auf, ohne ein Ergebnis auszuwerten.

Die Syntax der *call*-Anweisung lautet:

```
call FunctionName (argument, argument...)
```

2.12 Result

Die *result* Anweisung gibt einen Wert an die aufrufende Routine zurück. Der Rückgabewert kann eine Zahl, ein boolescher Wert, ein Text-String, ein Datenfeld oder das Resultat einer anderen Funktion sein. Es können konstante Werte oder Variable zugewiesen werden.

Die Syntax der *result* Anweisung lautet:

```
result = Value
```

3.1 PHP Programm Schnittstelle

RedCrab unterstützt die Entwicklung von *PHP* Programmen mit einer integrierten Entwicklungs Umgebung. Mehr Informationen zur Installation und Konfiguration finden Sie im *RedCrab^{PLUS}* Handbuch. Dieses Kapitel beschreibt die Programm

Schnittstelle zum Daten Transfer zwischen *RedCrab* und *PHP*. Die Schnittstelle verwendet zum Transfer die Standard Ein- und Ausgabe von *PHP* Programmen.

3.2 PHP Input

ReadCrab sendet Daten zur Standard Eingabe des *PHP* Programms mit der Methode *POST*. Die Daten enthalten ein oder mehrere benannte Argumente. Das erste Argument hat immer den Namen *func*, es enthält den Namen der Funktion die aufgerufen wird. Die nächsten Argumente enthalten die Parameter der Funktion; sie sind benannt mit *arg1*, *arg2*, *arg3* usw.

Das folgende Beispiel zeigt einen Aufruf der Funktion *Add*, mit zwei Parametern, in dem *PHP* Program *math*.

Beispiel:

Funktion Aufruf in *RedCrab*:

```
x = math.Add(a,b)
```

Handhabung des Aufrufs in *PHP*:

```
$fname = $_POST["func"]; // Funktion Name in $fname speichern  
$a1    = $_POST["arg1"]; // Erstes Argument in $a1 speichern  
$a2    = $_POST["arg2"]; // Zweites Argument in $a2 speichern  
$fname($a1,$a2);        // Aufruf der Funktion Add
```

Das Beispiel oben zeigt einen Funktions Aufruf mit zwei numerischen Parametern. Wenn die Argumente Datenfelder anhalten, werden diese von *RedCrab* in *PHP* Format konvertiert.

Das nächste Beispiel zeigt die Handhabung von Argumenten mit Datenfeldern im *PHP* Program.

Beispiel:

Eindimensionales Datenfeld

- 1.)

```
$data = $_POST["arg1"];  
$x    = $data[1];  
$y    = $data[2];
```
- 2.)

```
$x = $_POST["arg1"][1];  
$y = $_POST["arg1"][2];
```

Zweidimensionales Datenfeld

- 1.)

```
$data = $_POST["arg1"];  
$x    = $data[1][0];  
$y    = $data[2][0];
```
- 2.)

```
$x = $_POST["arg1"][0][1];  
$y = $_POST["arg1"][1][1];
```

3.3 PHP Output

Resultate werden von den *PHP* Funktionen über die Standard Ausgabe per *echo* Befehl an *RedCrab* gesendet.

Beispiel: `echo $val;`

Wenn eine Funktion ein Datenfeld als Resultat sendet, muß dieses als String formatiert werden. Das Format ist identisch mit dem Format importierter Textdateien. Das bedeutet, die Tabelle wird Zeile für Zeile gesendet. Spalten werden mit Komma, Zeilen werden mit Semikolon getrennt.

Die *PHP* Datei *RCFieldEcho.php* enthält eine Funktion, die ein Datenfeld zur Ausgabe formatiert. Sie können diese Datei in Ihr Programm importieren oder den Code kopieren.

```
1 <?PHP
2
3 function RCFieldEcho($a)
4 {
5     $cnt = count($a);
6     $mcnt = count($a[0]);
7     for ($i = 0; $i < $cnt; $i++)
8     {
9         if ($mcnt > 1 )
10        {
11            if ($i > 0) echo ' ';
12            RCFieldEcho($a[$i]);
13        }
14        else
15        {
16            if($i > 0) echo ',';
17            echo $a[$i];
18        }
19    }
20 }
21
22 ?>
```