**https://www.deachp.com/ejemplos/**

**User-Rpl.**

    El User-Rpl es el lenguaje al que tenemos acceso desde la calculadora, y es la forma mas elemental de programarla, pero la primera que debemos aprender. A continuación se presenta el contenido de este tema, el cual tiene los comandos mas usados para la programación de la calculadora: 

|  |  |
| --- | --- |
| Página 1 de 2 | |
| Página 1 | Página 2 |
| * [¿Que es un Programa?](http://www.angelfire.com/ok/hp48g/kit.html#que) * [Declaración de Variables](http://www.angelfire.com/ok/hp48g/kit.html#declaracion) * [Funciones de Comparación o Test](http://www.angelfire.com/ok/hp48g/kit.html#test) * [Funciones Lógicas](http://www.angelfire.com/ok/hp48g/kit.html#logico) * [Estructuras Condicionales](http://www.angelfire.com/ok/hp48g/kit.html#condicion)   + [IF..THEN..ELSE..END](http://www.angelfire.com/ok/hp48g/kit.html#IF)   + [CASE...END](http://www.angelfire.com/ok/hp48g/kit.html#case) * [Estructuras Iterativas (Bucles )](http://www.angelfire.com/ok/hp48g/kit.html#ciclos)   + [START...NEXT](http://www.angelfire.com/ok/hp48g/kit.html#start)   + [DO...UNTIL...END](http://www.angelfire.com/ok/hp48g/kit.html#do)   + [WHILE...REPEAT...END](http://www.angelfire.com/ok/hp48g/kit.html#while)   + [Contadores y Banderas](http://www.angelfire.com/ok/hp48g/kit.html#conta) * [Control del Teclado](http://www.angelfire.com/ok/hp48g/kit.html#teclado)   + [KEY](http://www.angelfire.com/ok/hp48g/kit.html#key)   + [WAIT](http://www.angelfire.com/ok/hp48g/kit.html#wait) | * [Entrada de datos](http://www.angelfire.com/ok/hp48g/Kit2.html#Entrada de datos)   + [INPUT](http://www.angelfire.com/ok/hp48g/Kit2.html#Input)   + [INFORM](http://www.angelfire.com/ok/hp48g/Kit2.html#Inform) * [Captura de Errores](http://www.angelfire.com/ok/hp48g/Kit2.html#error)   + [IFERR..THEN..ELSE..END](http://www.angelfire.com/ok/hp48g/Kit2.html#IFERR) * [Salida de Datos](http://www.angelfire.com/ok/hp48g/Kit2.html#Salida)   + [DISP](http://www.angelfire.com/ok/hp48g/Kit2.html#DISP)   + [MSGBOX](http://www.angelfire.com/ok/hp48g/Kit2.html#MSGBOX)   + [FONDOS](http://www.angelfire.com/ok/hp48g/Kit2.html#FONDOS) * [Salida de Datos con Gráficos](http://www.angelfire.com/ok/hp48g/Kit2.html#Gráficos)   + [®GROB](http://www.angelfire.com/ok/hp48g/Kit2.html#grob)   + [PICT](http://www.angelfire.com/ok/hp48g/Kit2.html#pict)   + [PVIEW](http://www.angelfire.com/ok/hp48g/Kit2.html#PVIEW)   + [REPL](http://www.angelfire.com/ok/hp48g/Kit2.html#REPL)   + [SUB](http://www.angelfire.com/ok/hp48g/Kit2.html#SUB)   + [®LCD y LCD®](http://www.angelfire.com/ok/hp48g/Kit2.html#LCD) |

**¿Que es un programas?**

    Un programa, para la calculadora, es todo aquello que este entre << >>. Un ejemplo de programa puede ser: << 1 2 + >>, el cual coloca en la pila de datos el numero 1, el 2 y luego los suma.   
Para editar cualquier programa hecho bajo User-Rpl solo se tiene que colocar en pila y pulsar "EDIT". Para guardarlo se coloca en pila y luego se coloca el nombre con el que se quiera grabar y se pulsa "STO".

**3:  << 1 2 + >>**

**2:  'Nombre'**

**1:  STO**

Para ejecutarlo se puede colocar en la pila y pulsar "EVAL", o llamarlo por su nombre.[TOPE](http://www.angelfire.com/ok/hp48g/kit.html#tope)

**Declaración de variables**

    Dentro de los programas es común llamar objetos u otros programas con variables, que pueden ser Estáticas, Locales o Globales.

**Variables Estáticas:**Las Variables Estáticas son guardadas en la memoria de usuario (RAM) y son independientes del programa que las llama. La declaración de una variable de este tipo es equivalente a guardar el contenido de la variable en un directorio con un nombre, como si se tratase de un programa.

**Variables Locales:**Las variables Locales son las que solo se ejecuta en un programa y solo son reconocidas en este. Para declarar variables locales se sigue la siguiente sintaxis:

**<< obj1 obj2** **V1 V2 >>**

    Las variables locales solo serán reconocidas en el bloque que fueron declaradas, o en los bloques contenidos dentro del bloque en que se declararon. A veces se puede tener varios bloques que generen confusión, por lo que se tiene que tener cuidado con el rango de acción de las variables Locales, ya que puede que no sean reconocidas en todo el programa.

**EJEMPLO:**

**<< **** V1 V2**

**<< V1 V2 \* V1 V2 +**  **M S  (1)**

**<< M S / >> (2)**

**>>  V2 V1 \***

**>>**

    En el ejemplo anterior las variables V1 y V2 solo son reconocidas en los bloques (1) y (2), aunque se hayan declarado al principio.

**Variables Globales:**Las variables Globales son aquellas que se declaran en un programa principal y pueden ser reconocidas, en  cualquier momento, desde cualquier bloque de instrucción o desde cualquier programa que se llame y  se ejecute en el programa principal. Para declarar una variable de este estilo se sigue la sintaxis:

**<< obj1 obj2** ******V1** **V2 >>**.

 Con esta declaración no se debe tomar precaución con el rango de acción de una variable, ya que es reconocida en todo el programa, pero una declaración de este tipo hace que los programas ocupen mayor espacio en memoria, por lo que debe de usarse cuando sea "ESTRICTAMENTE NECESARIO".[TOPE](http://www.angelfire.com/ok/hp48g/kit.html#tope)

**Funciones de Comparación o Test**

    Las funciones de comparación son aquellas que comparan argumentos, devolviendo un 1 (true = Verdadero) o un 0 (false = Falso) de acuerdo a lo que se este preguntando. Estas funciones se describen en la siguiente tabla:   
  

|  |  |
| --- | --- |
| **Función de comparación** | **Descripción** |
| SAME | Pregunta si dos objetos son iguales |
| == | Pregunta si dos objetos son iguales |
|  | Pregunta si dos objetos son distintos |
| < | Pregunta si el obj2 es menor al obj1 |
| > | Pregunta si el obj2 es mayor al obj1 |
|  | Pregunta si el obj2 es mayor o igual al obj1 |
|  | Pregunta si el obj2 es menor o igual al obj1 |

    Como se puede apreciar SAME y == cumplen la misma función. [TOPE](http://www.angelfire.com/ok/hp48g/kit.html#tope)

**Funciones Lógicas**

    Las funciones Lógicas son aquellas que permiten dar a conocer la relación entre dos condiciones. Estas funciones son el AND y el OR.

**AND:** Esta función permite dar a conocer si se cumplen Simultáneamente dos condiciones, y responde a la siguiente tabla de la verdad: 

|  |  |  |
| --- | --- | --- |
| **Condición #1** | **Condición #2** | **Respuesta** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**OR:** Con esta función se puede conocer si Una de las dos condiciones se cumple, y esta dado por la tabla de la verdad:   
  

|  |  |  |
| --- | --- | --- |
| **Condición #1** | **Condición #2** | **Respuesta** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

    Adicionalmente se tiene otra función Lógica (**NOT**), pero que no es para saber el estado de dos condiciones, sino que sirve para invertir el estado de una condición; y esta dada por la siguiente tabla de la verdad:   
  

|  |  |
| --- | --- |
| **Condición** | **Condición Negada** |
| 0 | 1 |
| 1 | 0 |

    Esta función no devuelve ningún tipo de argumento o valor. [TOPE](http://www.angelfire.com/ok/hp48g/kit.html#tope)

**Estructuras Condicionales**

    Una de las características de los lenguajes de programación modernos es que son estructurados, es decir, que responden a una estructura secuencial, por lo tienen un inicio, un desarrollo (que son las instrucciones que se ejecutan una tras otra) y un final. Los programas hechos en estos tipos de lenguajes pueden diseñarse y probarse en frío previamente, mediante algoritmos que resuelvan la situación presentada.

    Las Estructuras Condicionales permiten ejecutar una acción a través de la toma de decisiones ante una o varias condiciones dadas.

**IF...THEN...ELSE...END:**

Esta estructura se conoce como partición o "si"  binario, ya que:

**SI**se cumple una **CONDICIÓN**   
**ENTONCES**se realiza una **ACCIÓN**   
**SINO**se ejecuta otra **ACCIÓN DIFERENTE**   
**FIN**

Esta estructura presenta una variante interesante, que es la "partición simple", que es:

**SI**se cumple una **CONDICIÓN**   
**ENTONCES**se realiza una  **ACCIÓN**   
**FIN**

Hay que resaltar que si se cumple la condición se ejecuta la acción, pero de resto no se realiza nada.

**EJEMPLO DE PARTICIÓN BINARIA:**

**<< - IF 0 > THEN "POSITIVO"**

**ELSE "NEGATIVO"**

**END**

**>>**

El programa que hicimos resta dos números en la pila, pregunta si el resultado es positivo o negativo, y lo indica en cada caso.

**EJEMPLO DE PARTICIÓN SIMPLE:**

**<<  IF == THEN "Elementos iguales"**

**END**

**>>**

    Este otro programa pregunta si dos objetos colocados en pila son iguales, generando un mensaje en caso afirmativo, en caso contrario no se ejecuta ninguna acción.

**CASE...END**

    Con las estructuras "IF" solo se puede tomar una decisión de entre máximo dos condiciones, por lo que si se quiere tomar una decisión de varias se tendría que colocar varios "IF" anidados unos dentro de otros o de forma consecutiva. Esto es engorroso, y de esta necesidad surgió la estructura "CASE", que sigue la siguiente sintaxis:

**<<  CASE**

**CONDICION\_1  THEN  ACCION\_1  END**

**CONDICION\_2  THEN  ACCION\_2  END**

**.**

**.**

**.**

**CONDICION\_n  THEN  ACCION\_n  END**

**ACCION\_POR\_DEFECTO**

**END >>**

**Nota:** El "CASE" solo ejecuta una condición y es la primera que encuentre como verdadera. Si no encuentra ninguna ejecuta la acción por defecto.

**EJEMPLO CON "IF"**

**<<**  **N**

**<< IF N 0 == THEN "ES IGUAL A CERO"**

**ELSE**

**IF N 0 > THEN "ES POSITIVO"**

**ELSE "ES NEGATIVO"**

**END**

**END >>**

**>>**

**MISMO EJEMPLO CON "CASE"**

**<<**  **N**

**<< CASE**

**N 0 == THEN "ES IGUAL A CERO" END**

**N 0 > THEN "ES POSITIVO" END**

**"ES NEGATIVO"**

**END >>**

**>>**

    Hay que destacar que el "CASE" pregunta primero si N es igual a cero, luego si es positivo y si no es asume la ultima condición como negativa. [TOPE](http://www.angelfire.com/ok/hp48g/kit.html#tope)

**Estructuras Iterativas**

    Las estructuras iterativas son aquellas que nos permiten repetir varias veces un proceso. Estas estructuras son de gran importancia conocerlas, ya que cuando se programa su uso en la resolución de un problema es frecuente.

**START...NEXT:**

    Una de las estructuras soportadas por la calculadora es el START, la cual es una de las estructuras no condicionales, es decir, es independiente de cualquier condición y se efectúa un numero finito de veces. La sintaxis que sigue es la siguiente: 

**<< 1 #veces**

**START ACCIÓN**

**NEXT >>**

    En este tipo de estructura se le indica el número de veces a repetir el proceso con un 1 y el número de veces a repetir, al comienzo del ciclo. El proceso a ejecutar viene seguido después del comando START, y el final del ciclo lo determina el comando NEXT.

**EJEMPLO:**

Supongamos que queramos calcular el factorial de un numero, por ejemplo 5! = 5x4x3x2x1, lo resolveríamos con el siguiente programa.

**<< 1**  **N AUX**

**<< 1 N 1 - START N AUX \* 'AUX' STO 'N' 1 STO- NEXT >>**

    Este programa efectúa el factorial, no solo del número cinco sino, de cualquier número que este en la pila. Si al verlo surge alguna duda, recomiendo correrlo con el debbuger o depurador de la calculadora. Esto se hace colocando el programa en la pila y pulsando la secuencia de teclas PRG NXT RUN DBUG STT. De esta forma se vera como funciona el programa paso a paso. Otra forma de ver si un programa está correcto es la corrida en frío, que se hace siguiendo el comportamiento del programa en una hoja de papel. Recomiendo que se copie el programa en el papel,  y las variables que intervienen en el mismo en una tabla, a medida que se va siguiendo el comportamiento del programa se ve el estado de cada variable.

**DO...UNTIL...END**

    Este tipo de estructura pertenece al bloque de los ciclos condicionales, ya que dependen de la evaluación de una condición dada. El DO ejecuta un proceso y luego pregunta por una condición, para ver si ejecuta nuevamente el proceso; y así sucesivamente. La estructura que del DO es la siguiente: 

**<< DO PROCESO**

**UNTIL CONDICIÓN**

**END >>**

**EJEMPLO:**

**<< 1** ****

**<< DO  I 1 + 'I' 2 STO+**

**UNTIL I 100 ==**

**END >>**

**>>**

En el ejemplo anterior realizamos un programa que genera los primeros 50 primeros números pares.

**WHILE...REPEAT...END**

     El WHILE tiene como característica fundamental el preguntar una condición antes de ejecutar el proceso. El WHILE sigue la sintaxis:

**<< WHILE CONDICIÓN**

**REPEAT PROCESO**

**END >>**

    La  condición puede ser el resultado de evaluar una situación dada o puede ser un contador, que no es mas que una variable que lleva el control del numero de veces que se repite el proceso.

**EJEMPLO:**

**<< -5**

**WHILE DUP 0 <**

**REPEAT 1 +**

**END >>**

    En el programa anterior se pregunta si se cumple la condición  de que el numero que se toma en pila es menor  que cero, mientras que es menor se repite el proceso de sumar uno al número de la pila. Hay que  notar que para preguntar sobre la condición (si es menor a cero) primero se duplica el numero en al pila con el comando DUP.

Contadores y Banderas en la Iteración.

    Las estructuras iterativas condicionales (DO y WHILE) repiten un proceso si ocurre una condición específica. Si se quiere repetir un proceso un número determinado de veces podemos usar lo que se denomina contadores, que son variables se incrementa a medida que se ejecutan esos procesos.

**EJEMPLO:**

**<< 5******

**<< WHILE I 0 >REPEAT**

**I  'I' 1 STO-**

**END >>**

**>>**

    El programa coloca los números del 1 al 5 en pila, usando un ciclo que es controlado por la variable I, la cual es un contador .   
    Las banderas se usa en la iteración para salirse de un ciclo sin llegar al final del mismo. Como ejemplo se va a tomar el mismo programa anterior, pero colocándole una bandera.

**EJEMPLO:**

**<< 5******

**<< WHILE I 0 >REPEAT**

**I  'I' 1 STO-**

**IF I 3 == THEN 0 'I' STO END**

**END >>**

**>>**

    La bandera está en el IF, ya que al llegar I a el valor de 3 lo coloca con  valor de cero, luego el WHILE analizar la condición  y confirmará que I no es mayor que cero, así termina el ciclo.[TOPE](http://www.angelfire.com/ok/hp48g/kit.html#tope)

**Control del Teclado.**

    Un programa en ejecución puede tomar una decisión de acuerdo a una tecla que es oprimida. Esto es posible debido a que el programa reconoce cualquier tecla, ya que  que cada botón del teclado está identificado. Para reconocer un botón, se asume que el teclado de la calculadora es una  de matriz, que tendría la siguiente forma   
    
  

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Identificación de las teclas** | | | | | |
| 11.x | 12.x | 13.x | 14.x | 15.x | 16.x |
| 21.x | 22.x | 23.x | 24.x | 25.x | 26.x |
| 31.x | 32.x | 33.x | 34.x | 35.x | 36.x |
| 41.x | 42.x | 43.x | 44.x | 45.x | 46.x |
| 51.x | | 52.x | 53.x | 54.x | 55.x |
| 61.x | 62.x | 63.x | 64.x | 65.x | - |
| 71.x | 72.x | 73.x | 74.x | 75.x | - |
| 81.x | 82.x | 83.x | 84.x | 85.x | - |
| -ON- | 92.x | 93.x | 94.x | 95.x | - |

    Donde "x" puede tomar 7 valores (0,1, 2, 3, 4, 5 ó 6). El valor de x depende de si se usa una tecla de asignación de función, es decir, si se pulsó antes la tecla con la flechita morada, la flechita verde, la tecla de asignación de caracter (alfa) ó una combinación de estas tres. Si se ordena en una tabla resulta:   
    
  

|  |  |  |
| --- | --- | --- |
| **Tecla de asignación de función** | **Tecla en la HP** | **Valor** |
| Ninguna | --- | 0 ó 1 |
| Desplazamiento a la Derecha | [Desp. Der] | 2 |
| Desplazamiento a la Izquierda | [Desp Izq.] | 3 |
| Alfa | [Alfa] | 4 |
| Alfa + Desplazamiento a la Derecha | [Alfa] + [Desp. Der] | 5 |
| Alfa + Desplazamiento a la Izquierda | [Alfa] + [Desp Izq.] | 6 |

    Entonces si tenemos como identificación de una tecla el valor de 82.2 significa que pulsamos primero la tecla de asignación de función morada y luego pulsamos el número 1. (ver tabla de asignación de teclas, y la de funciones).

    Los comandos usados para el reconocimiento del teclado son "KEY" Y "WAIT", los cuales generan una respuesta con el formato explicado anteriormente.

KEY

    Este comando es usado para identificar a cualquier tecla en el momento de ejecución de un programa. La forma como reconoce una tecla es de manera instantánea, es decir, si se ejecuta devuelve el número de identificación de la tecla que se estaba pulsando en ese instante. Si no se está pulsando ninguna devuelve el valor cero.

**EJEMPLO:**

    Como este comando devuelve el valor de la tecla que se esté pulsando se incluirá en un bloque de iteración para ver su efecto, pero es válido ejecutar << KEY >> para ver su funcionamiento.

**<< DO  KEY DUP**

**UNTIL 0 **

**END >>**

     Este programa ejecuta la instrucción KEY hasta que se pulsa una tecla. Si el valor devuelto es 0 se sigue repitiendo el ciclo.

WAIT

    A diferencia del KEY, el comando WAIT devuelve el número de identificación de la tecla que se halla pulsado en un período de tiempo especificado. Si no se pulsa en ese tiempo ninguna tecla no aparecerá nada en la pila de la calculadora.

**EJEMPLO:     << 3 WAIT >>**

La ventaja del WAIT sobre el KEY es que se puede utilizar para esperar que se pulse una tecla en un período indeterminado de tiempo sin usar una estructura iterativa adicional. Esto se logra colocando como período de tiempo el valor de cero al comando WAIT.

**EJEMPLO:     << 0 WAIT >>**

    Una posible desventaja de usar WAIT es que cuando se oprima la tecla ON, con este último ejemplo, se devolverá el valor cero en vez de la posición de la tecla correspondiente. [TOPE](http://www.angelfire.com/ok/hp48g/kit.html#tope)