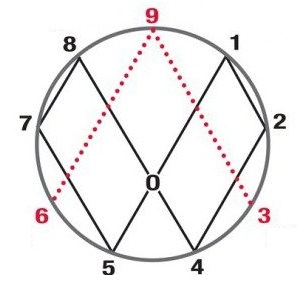
Las

Carlos A. Perez Medina

Carlitosc4@hotmail.com

NIMBUS 369

NIMBUS 369



Presentación

Nimbus es un lenguaje de programación que intenta basarse en la estructura sintáctica de Python, nace de la necesidad de hacer un lenguaje de programación el cual no requiera de muchas destrezas o tiempo para aprender a utilizar.

Nimbus está en fase de desarrollo por lo cual es carece de muchas estructuras de datos complejas que posteriormente y a como el proyecto vaya madurando estas posiblemente se irán incorporando en la pila de utilidades de este.

Esperamos que sea de su agrado el uso de Nimbus e incitamos a su persona a que nos apoye con sus conocimientos en el desarrollo de este.

El lenguaje de programación ofrece lo más básico y necesario para trabajar aplicaciones de consola, como son asignación de variables, ciclos, arboles lógicos de decisión, funciones y procedimientos.

Sin mas que decir se despide su servidor Carlos Perez.

Contenido

[Presentación 2](#_Toc10021660)

[Capítulo I: Reglas del Lenguaje 4](#_Toc10021661)

[Capítulo II: Variables, Estatutos y expresiones en Nimbus 369 (N-369) 5](#_Toc10021662)

[*Tipos de Datos* 5](#_Toc10021663)

[Comentarios 6](#_Toc10021664)

[*Palabras reservadas* 6](#_Toc10021665)

[*Variables y Constantes* 6](#_Toc10021666)

[Declaración de Variables: 6](#_Toc10021667)

[Declaración de Constantes: 6](#_Toc10021668)

[*Instrucciones* 7](#_Toc10021669)

[Función “Nimbus\_Main(){}”: 7](#_Toc10021670)

[Función “#=>()”: 7](#_Toc10021671)

[Función “#<=()”: 7](#_Toc10021672)

[Capitulo III: Estatuto de Asignación 8](#_Toc10021673)

[Asignación Simple 8](#_Toc10021674)

[Capítulo IV: Operadores Matemáticos 9](#_Toc10021675)

[Operadores Matemáticos 9](#_Toc10021676)

[Operadores de autoincremento matemático 9](#_Toc10021677)

[Capítulo V: Expresiones 10](#_Toc10021678)

[Evaluación de expresiones Matemáticas en Nimbus 369 10](#_Toc10021679)

[Capítulo VI: Condicionales en Nimbus 369 11](#_Toc10021680)

[*Expresiones Booleanas* 11](#_Toc10021681)

[*Operadores Lógicos* 11](#_Toc10021682)

[Operador Lógico “+’: 11](#_Toc10021683)

[Operador Lógico “/”: 11](#_Toc10021684)

[Operador Lógico “-”: 11](#_Toc10021685)

[Resumen de operadores lógicos: 12](#_Toc10021686)

[*Secuencias de Escape en Nimbus 369* 12](#_Toc10021687)

[Estructuras Condicionales 12](#_Toc10021688)

[*Estatuto condicional “x<condición>”:* 12](#_Toc10021689)

[Estructura condicional Simple: 12](#_Toc10021690)

[Estructura condicional doble: 13](#_Toc10021691)

[*Estatutos de repetición* 14](#_Toc10021692)

[Estructura de repetición “zc<condición>[]”: 14](#_Toc10021693)

[Estructura de repetición "zl(inicialización | condición | incremento)[]”: 14](#_Toc10021694)

[Otras palabras utilizadas en las estructuras de repetición: 14](#_Toc10021695)

[Capítulo VII: Funciones y Procedimientos en Nimbus 369 15](#_Toc10021696)

[*Creación de Funciones* 16](#_Toc10021697)

[*Creación de Procedimientos* 16](#_Toc10021698)

Capítulo I: Reglas del Lenguaje

Nimbus 369 al igual que otros lenguajes de programación es sensible a mayúsculas y minúsculas.

Todo archivo Nimbus 369 debe empezar al inicio con la línea: <!\_Nimbus\_369\_Code\_!> para identificar que es un archivo ejecutable de tipo Nimbus.

La función “Nimbus\_Main()” es la función principal para la ejecución de un programa codificado en Nimbus 369 (N-369).

Las variables dependiendo en que parte del código son declaradas, estas serán globales o locales.

Los nombres de las variables, las funciones y los procedimientos solo pueden usar letras mayúsculas, minúsculas, números y guiones bajos.

Cada sentencia en Nimbus 369 debe estar delimitada por un “;”.

Para llamar un Procedimiento o una función en Nimbus 369 se debe utilizar la palabra reservada “inv”.

El ocultamiento tanto para las variables, las funciones y los procedimientos en Nimbus 369 está definido por la siguiente tabla. Por defecto si no se coloca ningún símbolo delante de la variable, procedimiento o función se entiende que es público.

|  |  |
| --- | --- |
| Símbolo | Visibilidad |
| @@ | Privado |
| $$ | Publico |

Capítulo II: Variables, Estatutos y expresiones en Nimbus 369 (N-369)

## *Tipos de Datos*

En Nimbus 369 (N-369) se poseen datos numéricos como los enteros y reales además de los datos no numerosos que serían los tipos carácter y lógicos

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Descripción | Tamaño | Ejemplo |
| ca | Carácter | 1 | ‘A’ |
| cc | Cadena de caracteres | 32000 | "Nimbus" |
| en | Entero | 32000 | -199 |
| flo | Número real | 32000.999999 | 3.1416 |

## *Palabras reservadas*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ca | cc | en | la | flo | divTc |  |
| sum | res | mult | div | modd | exp |  |
| eq | dif | maq | meq | maiq | meiq | inv |

## *Variables y Constantes*

Para declarar una variable o una constante en Nimbus 369 se debe colocar el tipo de dato y el nombre de la variable e inicializar en la misma línea.

Las variables y las constantes en N-369 tiene la restricción de que deben empezar con letras ya sean minúsculas y mayúsculas, seguidas de estas solo se pueden usar más letras, números o guiones bajos “\_” esto con el fin de evitar aberraciones en los nombres de estas.

### Declaración de Variables:

Declaración de variable tipo entero:

|  |
| --- |
| en Variable\_Entera < 45; |

Declaración de variable tipo carácter:

|  |
| --- |
| ca variable\_Caracter < “a”; |

### Declaración de Constantes:

Para declarar una constante en Nimbus 369 debe agregar al inicio el símbolo “~”.

|  |
| --- |
| ~ en Numero < 23; |

(declaración de una constante en Nimbus)

Por otro lado, existen variables de ámbito local y de ámbito global, en las cuales una variable de ámbito local solo existe dentro de un fragmento de código especifico y las variables de ámbito global pueden existir durante toda la ejecución del código.

¿Pero que determina el ámbito de ejecución de cada variable?

El ámbito estará determinado por el sitio donde se declare la variable, si la variable se declara al inicio del programa será considerada como global, no obstante, si se declara esa misma variable dentro de una función o método esta declaración definirá esa variable como local.

## *Instrucciones*

Algunas funciones básicas de Nimbus 369 (N-369) son la función “#=>()”, la función “#<=()” y la función “Nimbus\_Main()[]” las cuales se describen a continuación:

### Función “Nimbus\_Main()[]”:

La función “Nimbus\_Main()[]” es la encargada de ejecutar y llamar al resto de funciones o declaraciones del código. Es la función más importante, a continuación, se muestra su sintaxis:

|  |
| --- |
| Nimbus\_Main()  [  Acciones  ] |

|  |
| --- |
| Nimbus\_Main()  [  en número < #<=(“Digite un numero Entero: ”);  #=>(numero);  [ |

(Uso de la función “Nimbus\_Main()”)

### Función “#=>()”:

La función “#=>()” permite mostrar o enviar información al dispositivo estándar de salida como por ejemplo el monitor.

A continuación, se muestra el uso de la función “#=>()”:

|  |
| --- |
| #=>( 100 );  100 |

(Uso de la función “#=>()” con números)

|  |
| --- |
| #=>( “a” );  a |

(Uso de la función “#=>()” con Caracteres)

|  |
| --- |
| #=>( “a” % “b” );  ab |

(Uso de la función “#=>()” con concatenación)

### Función “#<=()”:

La función “#=>( “Mensaje” )” permite obtener datos de la entrada estándar como por ejemplo el teclado, además de brindar un mensaje que interactúe con el usuario.

A continuación se muestra el uso de la función #<=():

|  |
| --- |
| en número < #<=( “Digite un numero Entero: ” );  #=> (numero);  Digite un numero Entero: 34  34 |

(Uso de la función “#=>()” con números)

|  |
| --- |
| ca carácter < #<=( “Digite un carácter: ” );  #=>(carácter);  Digite un carácter: a  a |

(Uso de la función “#=>()” con Caracteres)

Capitulo III: Estatuto de Asignación

Los estatutos de asignación de Nimbus 369 (N-369) permiten crear nuevas variables, Nimbus permite la asignación de variables sin inicialización, si así lo desea el programador o puede inicializarlas en el momento que las crea, para asignar un valor a la variable es necesario usar el símbolo “<”.

Nimbus 369 (N-369) permite varios tipos de asignación:

## Asignación Simple

|  |
| --- |
| Numero1 < 45; |

|  |
| --- |
| en numero1;  en numero2;  numero1 < #<=(“Digite un numero Entero: ”);  numero2 < numero1;  #=>(numero2);  Digite un numero Entero: 34  34 |

Capítulo IV: Operadores Matemáticos

En Nimbus 369 los operandos matemáticos cambian en relación a otros lenguajes de programación, pero de igual forma se utilizan con el mismo propósito, sumar, restar, multiplicar, dividir, etc…

## Operadores Matemáticos

A continuación, se muestra una tabla con los operadores matemáticos en Nimbus 369:

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Ejemplo |
| sum | Adición | 2 sum 10 = 12 |
| res | Substracción | 10 res 2 = 8 |
| mult | Multiplicación | 4 mult 5 = 20 |
| div | División | 10 div 5 = 2 |
| modd | Módulo (resto) | 10 modd 2 = 0 |
| exp | Exponente | 5 exp 2 = 25 |
| divTc | División con trucado de decimal | 9 divTc 2 = 4 |

## Operadores de autoincremento matemático

Además de los estatutos de asignación anteriores Nimbus 369 también posee la siguiente tabla de asignación matemática:

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Ejemplo |
| <sum | Asigna la suma del valor de la expresión de la derecha y de la variable de la izquierda. | a <sum 5  es equivalente a:  (a < a sum 5) |
| <res | Asigna la resta del valor de la expresión de la derecha y de la variable de la izquierda. | a <rest 5 (a < a res 5) |
| <mult | Asigna la multiplicación del valor de la expresión de la derecha y de la variable de la izquierda. | a <mult 5  es equivalente a:  (a < a mult 5) |
| <div | Asigna la división del valor de la expresión de la derecha y de la variable de la izquierda. | a <div 5  es equivalente a:  (a < a div 5) |
| <modd | Asigna el modulo del valor de la expresión de la derecha y de la variable de la izquierda. | a <modd 5  es equivalente a:  (a < a modd 5) |
| <exp | Asigna la potencia del valor de la expresión de la derecha y de la variable de la izquierda. | a <exp 5  es equivalente a:  (a < a exp 5) |
| <divTc | Asigna el cociente trucado del valor de la expresión de la derecha y de la variable de la izquierda. | a <exp 5  es equivalente a:  (a < a exp 5) |

Capítulo V: Expresiones

En Nimbus las expresiones son cualquier definición que se tenga que evaluar para su ejecución.

Por ejemplo, podemos asignar valores a dos variables desde el teclado y luego imprimirlas por pantalla, esto sería un ejemplo de una expresión en Nimbus 369.

|  |
| --- |
| en numero1 < #<=( “Digite un numero Entero: ” );  en numero2 < #<=( “Digite otro número Entero: ” );  #=> (numero1 sum numero2);  Digite un numero Entero: 34  Digite otro número Entero: 32  66 |

## Evaluación de expresiones Matemáticas en Nimbus 369

Como en otros lenguajes de programación, Nimbus no es la excepción, el orden de precedencia de los operadores matemáticos es como de costumbre los paréntesis son la más alta prioridad, seguido de las expresiones de exponente, luego la multiplicación y la división, por último, quedando las sumas y las restas. En consecuencia, de encontrar mismos operadores matemáticos estos serán evaluados de izquierda a derecha.

|  |
| --- |
| #=>( (2 sum 5) res 3 exp 2 mult 2 );  -11 |

En el ejemplo anterior se muestra como Nimbus 369 evalúa la expresión matemática, sacando primero los paréntesis donde “2+5=7” luego saca el exponente que sería “3^2=9” después de haber sacado ese cálculo procede a multiplicar el resultado del exponente por dos “9\*2=18” y por ultimo aplica la resta de los paréntesis por todo el resto de la expresión en este caso sería el 18 , “7-18=-11” dando como resultado el “-11”.

Capítulo VI: Condicionales en Nimbus 369

## *Expresiones Booleanas*

Nimbus utiliza la siguiente tabla para definir los operadores de comparación:

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Ejemplo |
| eq | Devuelve verdadero si a es igual a b. | (a eq b) es falso |
| dif | Devuelve verdadero si es diferente a b. | (a dif b) es verdadero |
| maq | Devuelve verdadero si es mayor que b. | (a maq b) es falso |
| meq | Devuelve verdadero si a es menor que b. | (a meq b) es verdadero |
| maiq | Devuelve verdadero si a es mayor o igual a b. | (a maiq b) es falso |
| meiq | Devuelve verdadero si a es menor o igual a b. | (a meiq b) es verdadero |

## *Operadores Lógicos*

En Nimbus 369 también existen operadores lógicos, los cuales se representan con los símbolos “+”, “/” y “- “, a continuación, se presenta la tabla de verdad para cada uno de ellos y un cuadro resumen.

### Operador Lógico “+’:

|  |  |  |
| --- | --- | --- |
| Operando #1 | Operando #2 | Resultado |
| Verdadero | Verdadero | Verdadero |
| Verdadero | Falso | Falso |
| Falso | Verdadero | Falso |
| Falso | Falso | Falso |

### Operador Lógico “/”:

|  |  |  |
| --- | --- | --- |
| Operando #1 | Operando #2 | Resultado |
| Verdadero | Verdadero | Verdadero |
| Verdadero | Falso | Verdadero |
| Falso | Verdadero | Verdadero |
| Falso | Falso | Falso |

### Operador Lógico “-”:

|  |  |
| --- | --- |
| Operando | Resultado |
| Verdadero | Falso |
| Falso | Verdadero |

### Resumen de operadores lógicos:

|  |  |  |
| --- | --- | --- |
| Operador | Descripción | Ejemplo |
| + | Devuelve verdadero si ambos enunciados son verdaderos. | (a eq 15 + b eq 35) es verdadero. |
| / | Devuelve verdadero si uno de los enunciados es verdadero. | (a eq 15 / b eq 35) es verdadero. |
| - | Devuelve verdadero si el enunciado que sigue es falso y falso si el enunciado es verdadero. | (a eq 15 + b eq 35) es falso. |

## *Secuencias de Escape en Nimbus 369*

A continuación, se presentan una tabla con las secuencias de escape elementales que usa Nimbus 369 en sus operaciones:

|  |  |
| --- | --- |
| Secuencias de Escape | Significado |
| : [enter] | Continua en la siguiente línea |
| :\ | Barra invertida (\) |
| :/ | Barra (/) |
| :' | Comilla simple ('') |
| :" | Comilla doble ("") |
| :r | Retorno de Carro. |
| :t | Tabulador. |

## Estructuras Condicionales

En Nimbus 369 las estructuras condicionales permiten definir el flujo de ejecución de código, a continuación, se detallas las estructuras tanto de condición como de repetición.

### *Estatuto condicional “x<condición>”:*

El estatuto x<condición> es un estatuto de control condicional mediante el cual se realizan evaluaciones booleanas de falso o verdadero en Nimbus 369, a continuación, se detalla su sintaxis y variantes, seguido de un ejemplo de uso básico de los mismos.

### Estructura condicional Simple:

|  |
| --- |
| x<condición>  [  acción  ] |

|  |
| --- |
| x<5 maiq 3>  [  #=>( “5 es mayor que 3” );  ]  5 es mayor que 3 |

### Estructura condicional doble:

|  |
| --- |
| x<condición>  [  acción  ]  x!  [  acción  ] |

|  |
| --- |
| x< 5 eq 6 >  [  #=>( “5 es igual que 6” );  ]  x!  [  #=>( “5 es diferente que 6” );  ]  5 es diferente que 6 |

Recuerde que al igual que en otros lenguajes de programación en Nimbus 369 también puede hacer Condicionales anidados como por ejemplo la siguiente sintaxis:

|  |
| --- |
| x< 8 meq 5 >  [  #=>( “8 es menor que 5” );  ]  x!  [  #=>( “8 es mayor que 5 ^” );    x<(8 modd 2) eq 0>  [  #=>( “8 es un numero par” );  ]  x!  [  #=>( “8 es un número impar” );  ]  ]  8 es mayor que 5  8 es un numero par |

## *Estatutos de repetición*

Las estructuras de repetición se utilizan para ejecutar varias veces fragmentos de código, en Nimbus 369 existen dos estructuras para este fin las cuales son zc<condición>[]” y "zl()[]".

### Estructura de repetición “zc<condición>[]”:

La estructura "zc<>[]" es una estructura de repetición que mientras la condición sea verdadera esta ejecutará la repetición, no obstante, cuando la condición se torne falsa este saldrá y terminara el ciclo de ejecución.

La sintaxis de “zc<>[]” es la siguiente en sus dos variantes:

|  |
| --- |
| zc< condición >[  acciones  ] |

A continuación, se muestran ejemplos de uso básico para la estructura de repetición " zc<>[]":

|  |
| --- |
| en a < 1;  en b < 2;  zc< a meq b >[  #=>( “a es menor que b” );  a <sum 1;  ]  a es menor que b |

Capítulo VII: Funciones y Procedimientos en Nimbus 369

Como anteriormente vimos existen funciones propias de Nimbus 369 como lo son la función de mostrar información en pantalla “#=>()” y la de obtener datos del teclado “#<=(“Mensaje”)”, toda función propia del lenguaje inicia con el símbolo “#”, por otro lado el programador es libre de definir sus propias funciones.

## *Creación de Funciones*

Para crear funciones en Nimbus 369 es necesario poner “F\_” seguido del nombre de la función, este nombre debe cumplir los mismos requerimientos que se toman en cuenta al definir el nombre de las variables y para el retorno de datos se utilizara el símbolo “&”. A continuación, se muestra la sintaxis utilizada en Nimbus 369 para definir funciones. Tome en cuenta que no es necesario definir el tipo de dato de la función, pero sí de los parámetros a recibir si esta llevara parámetros.

|  |
| --- |
| F\_Nombre\_función([tipo] parámetro #1, [tipo] parámetro #N )  [  Acciones  & (valor a retornar)  ] |

(Declaración de una función en Nimbus 369)

|  |
| --- |
| <!\_Nimbus\_369\_Code\_!>  @ F\_Sumar(Uno, Dos)  [  & Uno sum Dos;  ]  Nimbus\_Main()  [  en numero1 < #<=(“Digite un numero Entero: ”);  en numero2 < #<=(“Digite otro número Entero: ”);  inv F\_Sumar(numero1, numero2);  #=>(F\_Sumar);  ] |

## *Creación de Procedimientos*

Para crear procedimientos en Nimbus 369 es necesario poner “P\_” seguido del nombre del procedimiento, este nombre debe cumplir los mismos requerimientos que se toman en cuenta al definir el nombre de las variables. Los procedimientos no retornan dato alguno, simplemente ejecutan código.

A continuación, se muestra la sintaxis utilizada en Nimbus 369 para definir procedimientos. Tome en cuenta que es necesario definir el tipo de dato de los parámetros a recibir si este llevara parámetros.

|  |
| --- |
| P\_Nombre\_Procedimiento([tipo] parámetro #1, [tipo] parámetro #N )  [  Acciones  ] |

(Declaración de un procedimiento en Nimbus 369)

|  |
| --- |
| <!\_Nimbus\_369\_Code\_!>  P\_Hola()  [  #=>(“Hola a todos esto es Nimbus 369”);  ]  Nimbus\_Main()  [  inv P\_Hola ();  ] |