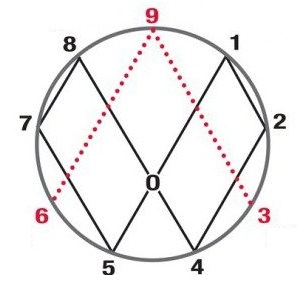
Las

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](#_Toc11865587)

[Capítulo I: Reglas del Lenguaje 4](#_Toc11865588)

[Capítulo II: Variables, Estatutos y expresiones en Nimbus 369 (N-369) 5](#_Toc11865589)

[*Tipos de Datos* 5](#_Toc11865590)

[*Palabras reservadas* 5](#_Toc11865591)

[*Variables y Constantes* 5](#_Toc11865592)

[Declaración de Variables: 5](#_Toc11865593)

[Declaración de Constantes: 5](#_Toc11865594)

[*Instrucciones* 6](#_Toc11865595)

[Función “Nimbus\_Main()[]”: 6](#_Toc11865596)

[Función “#=>()”: 6](#_Toc11865597)

[Función “#<=()”: 7](#_Toc11865598)

[Capitulo III: Estatuto de Asignación 8](#_Toc11865599)

[Capítulo IV: Operadores Matemáticos 8](#_Toc11865600)

[Operadores Matemáticos 9](#_Toc11865601)

[Capítulo V: Expresiones 9](#_Toc11865602)

[Evaluación de expresiones Matemáticas en Nimbus 369 9](#_Toc11865603)

[Capítulo VI: Condicionales en Nimbus 369 10](#_Toc11865604)

[*Expresiones Booleanas* 10](#_Toc11865605)

[*Operadores Lógicos* 10](#_Toc11865606)

[Operador Lógico “+’: 10](#_Toc11865607)

[Operador Lógico “/”: 10](#_Toc11865608)

[Resumen de operadores lógicos: 11](#_Toc11865609)

[Estructuras Condicionales 11](#_Toc11865610)

[*Estatuto condicional “x<condición>”:* 11](#_Toc11865611)

[Estructura condicional: 11](#_Toc11865612)

[*Estatutos de repetición* 12](#_Toc11865613)

[Estructura de repetición “zc<condición>[]”: 12](#_Toc11865614)

[Capítulo VII: Funciones y Procedimientos en Nimbus 369 13](#_Toc11865615)

[*Creación de Funciones* 13](#_Toc11865616)

[*Creación de Procedimientos* 14](#_Toc11865617)

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 y guiones bajos.

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

Todos los enunciados declarados deben ir expuestos en la misma línea.

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

Las funciones no pueden llevar el mismo nombre de las variables ya que las funciones comparten ámbito con la función principal.

Todas las variables que se deseen utilizar en nimbus se deben declarar antes de la función principal o en dado caso abrir un ámbito “[]” para realizar las operaciones matemáticas o lógicas correspondientes.

Las operaciones matemáticas con números decimales se debe de realizar asignando el valor a una variable de tipo “flo” y no de forma directa, ya que el compilador dará error.

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

|  |  |
| --- | --- |
| Símbolo | Visibilidad |
| @ | Privado |

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 | Un caracter | ‘A’ |
| en | Entero | 32000 | -199 |
| flo | Número real | 32000.999999 | 3.1416 |

## *Palabras reservadas*

|  |  |  |  |
| --- | --- | --- | --- |
| ca | en | flo | inv |
| sum | res | mult | div |
| eq | dif | maq | meq |

## *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 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  ] |

|  |
| --- |
| en número ;  Nimbus\_Main()  [  número < #<=( 5 );  [  #=>(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, no obstante este mensaje no puede estar separado por blancos, sino que debe separar las palabras con un “\_”.

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

|  |
| --- |
| en número;  [  número < #<=( ‘Digite\_un\_numero’ ) ;  ]  [  #=>( número) ;  ]  34 |

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

|  |
| --- |
| ca carácter ;  [  carácter < #<=( ‘Digite\_un\_caracter‘ ) ;  ]  [  #=>(carácter) ;  ]  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, para asignar un valor a la variable es necesario usar el símbolo “<”.

Nimbus 369 (N-369) permite solo un tipo de asignación que es la asignación simple la cual se detalla a continuación:

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

(asignación simple)

|  |
| --- |
| en numeroP ;  [  numeroP < #<=( ‘Digite\_numero:’ ) ;  ]  [  #=>( numeroP ) ;  ]  Digite\_numero: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 |

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 numeroP ;  en numeroS ;  [  numeroP < #<=( ‘Digite\_un\_numero\_Entero:‘ ) ;  numeroS < #<=( ‘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

En Nimbus las operaciones matemáticas deben hacerse declaración por declaración, asignando cada resultado a una variable e ir posteriormente utilizando el resultado anterior para la nueva operación matemática, esto con la salvedad de no exceder dos operandos como máximo por operación. A continuación se muestra un ejemplo de lo planteado:

|  |
| --- |
| en numP < 0 ;  en numS < 0 ;  en numT < 0 ;  en numC < 0 ;  [  num1 < 2 sum 5 ;  num2 < num1 res 3 ;  num3 < num2 exp 2 ;  num4 < num3 mult 2 ;  ]  [  #=>( num4 );  ]  32 |

Ejemplos de sentencias no válida para Nimbus:

|  |
| --- |
| num1 < 2 sum 5 mult 4 ;  num1 < ( 2 sum 5 ) mult 4 ;  en num1 < 2 sum 5 ; |

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 |

## *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 |

### 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. |

## 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 al igual que la asignación de variables, el condicional booleano y todas sus sentencias se deben definir una tras otra en la misma línea, a continuación, se detalla su sintaxis, seguido de un ejemplo de uso básico.

### Estructura condicional:

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

|  |
| --- |
| x<5 maiq 3>  [  #=>( ‘5\_es\_mayor\_que\_3’ ) ;  ]  5\_es\_mayor\_que\_3’ |

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 maq 5 >[  #=>( ‘8\_es\_mayor’ ) ;  x< 8 modd 2 eq 0>  [  #=>( ‘8\_es\_par’ ) ;  ]  ]  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 “zc<condición>[]”

### 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 ();  ] |