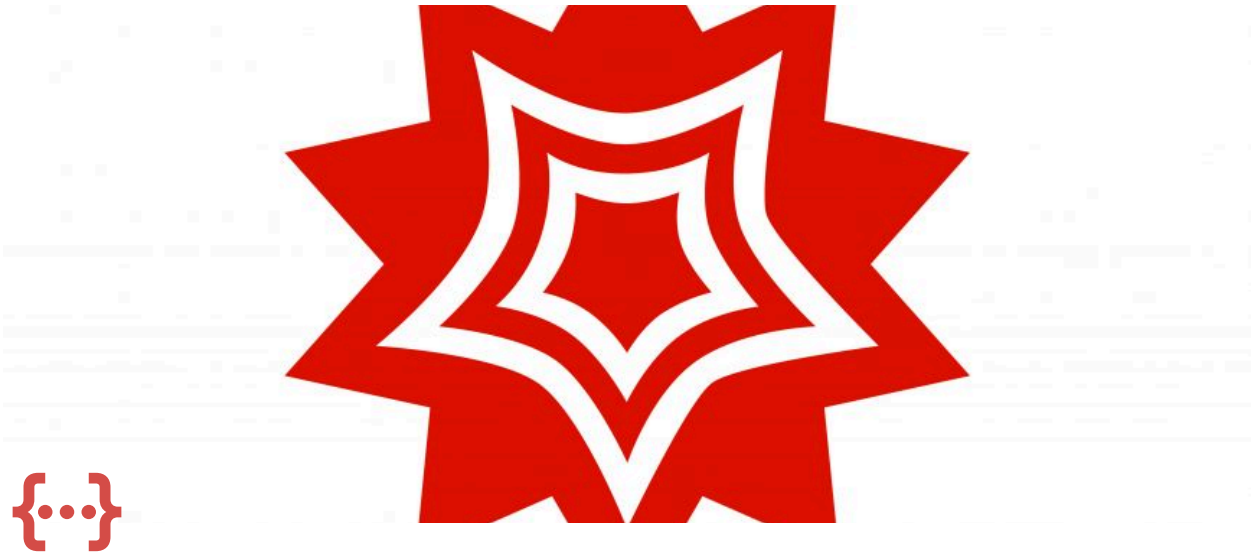

Sintaxis de Wolfram

1. Asignación de Variables

En Wolfram Mathematica, las variables se asignan usando el operador `=`.

```
x = 5
```

Esto asigna el valor 5 a la variable `x`.

Notas importantes:

- Para comprobar la igualdad se utiliza `==` (doble igual), mientras que `=` (simple) se usa para asignar.
- Las variables se pueden redefinir en cualquier momento.

---

2. Estructura Condicional: `If`

La forma general de la sintaxis de `If` es:

```
If[condición, expresiónSiVerdadero, expresiónSiFalso]
```

- `condición` es una expresión lógica que evalúa a `True` o `False`.
- `expresiónSiVerdadero` es lo que se evalúa y devuelve si la condición es verdadera.
- `expresiónSiFalso` es lo que se evalúa y devuelve si la condición es falsa. (Puede omitirse si no se necesita un caso falso.)

Ejemplo

```
x = 10;
If[x > 0,
  Print["x es positivo"],
  Print["x es negativo o cero"]
];
```

En este ejemplo, se imprime `"x es positivo"` porque `x` vale 10, que es mayor que 0.

Si no necesitas la parte falsa, podrías usar:

```
If[x > 0,
  Print["x es positivo"]
];
```

En este caso, no pasaría nada si la condición resulta falsa.

---

3. Definición de Funciones

Para definir funciones en Wolfram Mathematica, se utilizan patrones. La forma más simple es:

```
nombreFuncion[argumento_] := cuerpo
```

- `argumento_` denota un patrón que coincide con cualquier valor que se le pase a la función.
- `:=` (SetDelayed) define la función de manera que el cuerpo se evalúa cada vez que se llama.

Ejemplo

```
f[x_] := x^2 + 3x + 1
```

Aquí, `f` es una función que, dado `x`, retorna `x^2 + 3x + 1`.

Puedes llamarla así:

```
f[2]  (* Devuelve 2^2 + 3*2 + 1 = 4 + 6 + 1 = 11 *)
```

---

4. Bucle For

La forma general de un bucle `For` en Mathematica es:

```
For[inicio, prueba, incremento, cuerpo]
```

- `inicio`: inicializa variables.
- `prueba`: condición que se evalúa en cada iteración (mientras sea `True`, el bucle sigue).
- `incremento`: especifica la actualización de la variable de control.
- `cuerpo`: la(s) acción(es) que se ejecutan en cada iteración.

Ejemplo

```
For[i = 1, i <= 5, i++,
  Print["El valor de i es ", i];
]
```

Este bucle:

1. Empieza con `i = 1`.
2. Comprueba si `i <= 5`. Si es verdad, ejecuta el cuerpo.
3. Finalmente hace `i++` (incremento de `i` en 1).
4. Se repite hasta que la prueba `i <= 5` sea falsa.

---

5. Bucle While

La estructura de `While` es:

```
While[condición, cuerpo]
```

- El cuerpo se sigue ejecutando mientras `condición` sea `True`.

Ejemplo

```
x = 0;
While[x < 3,
  Print["x vale ", x];
  x++;
]
```

Mientras `x` sea menor que 3, se imprime el valor de `x` y luego se incrementa en 1.

---

6. Uso de Table

`Table` genera una lista (o arreglo multidimensional) repitiendo una expresión para diferentes valores de variables de iteración.

La forma más simple es:

```
Table[expresión, {i, n}]
```

- Genera una lista evaluando `expresión` para `i` desde 1 hasta `n`.

También puede especificarse un rango y paso:

```
Table[expresión, {i, inicio, fin}]
Table[expresión, {i, inicio, fin, paso}]
```

Ejemplos

1. Crear una lista de cuadrados de 1 a 5:

   ```
   Table[i^2, {i, 5}]
    (* Devuelve: {1, 4, 9, 16, 25} *)
   ```

2. Crear una tabla de valores de una función en un rango específico:

   ```
   Table[x^2 - 1, {x, -2, 2}]
    (* Devuelve: {3, 0, -1, 0, 3} *)
   ```

3. Especificar un paso:

   ```
   Table[x, {x, 0, 1, 0.25}]
    (* Devuelve: {0, 0.25, 0.5, 0.75, 1.0} *)
   ```

4. Tabla bidimensional (en este caso, una matriz con las sumas i+j):

   ```
   Table[i + j, {i, 3}, {j, 3}]
    (* Devuelve:
      {
       {2, 3, 4},
       {3, 4, 5},
       {4, 5, 6}
      }
    *)
   ```

7. Uso de `Print`

El comando `Print` permite mostrar en pantalla texto, números u otras expresiones durante la evaluación de un bloque de código. Acepta múltiples argumentos separados por comas, y los concatena automáticamente.

Ejemplo básico de `Print`:

```
Print["Hola, el resultado es: ", 5 + 3]
(* Salida: Hola, el resultado es: 8 *)
```

Concatenar Strings

En **Mathematica**, el operador `<>` se usa para **concatenar cadenas de texto (strings)**. Es el equivalente a `+` para números, pero aplicado a textos.

Cuando usas `<>` en combinación con `Print`, permite unir texto, números u otros elementos en una sola cadena que luego se muestra.

Ejemplo básico:

```
Print["Hola" <> " Mundo"]
(* Salida: Hola Mundo *)
```

Concatenación de texto y números:

Si deseas combinar texto con números u otras expresiones, necesitas convertir esos números a texto usando `ToString`.

```
x = 42;
Print["El valor de x es: " <> ToString[x]]
(* Salida: El valor de x es: 42 *)
```

Sin `ToString`, Mathematica mostrará un error, ya que `<>` solo concatena cadenas, no directamente números.

Ejemplo complejo:

```
a = 5; b = 10;
Print["El valor de a + b es: " <> ToString[a + b]]
(* Salida: El valor de a + b es: 15 *)
```

El operador `<>` es ideal cuando necesitas generar salidas más legibles o personalizadas combinando texto y resultados de cálculos.

---

8. Uso de `Module`

`Module` permite definir variables locales dentro de un bloque de código para que no interfieran con variables globales. Es muy útil para encapsular funciones o cálculos.

Sintaxis básica:

```
Module[{var1, var2, ...}, expr]
```

- `{var1, var2, ...}`: Variables locales.
- `expr`: Expresión que utiliza las variables locales.

Ejemplo simple:

```
Module[{x = 5, y = 10}, x + y]
(* Salida: 15 *)
```

Ejemplo con `Print` y `Module`:

```
Module[{a = 3, b = 4},
  Print["La suma de a y b es: ", a + b];
```

```
  a * b
]
(* Salida: La suma de a y b es: 7 *)
(* Retorna: 12 *)
```

Consejos Generales

- En Mathematica, las mayúsculas importan. Las funciones internas comienzan con mayúscula (p. ej., `If`, `Table`, `For`, `While`).
- Usa `;` para separar expresiones en la misma línea o para suprimir la salida de una expresión.
- Cuando quieras imprimir algo en medio de tus cálculos, puedes usar `Print[]`.
- Para definir variables locales y evitar conflictos, se usan `Module` o `Block`. Un ejemplo simple:

  Esto asegura que la `x` de dentro no afecte a la `x` global (si existiera).

  ```
  Module[{x=0},
   x = x + 1;
   x
  ]
  ```