Variables y Tipos de Datos

Función j_eval y j_plot en R

Introducción

Objetivos de aprendizaje

Variables

Table 1: Comparación de sintaxis básica entre Python, R y Julia

Lenguaje	Operador de asignación	Comando para imprimir	Descripción y Alternativas
Python	`=`	`print()`	El operador `=` es el estándar único para asignación. Para imprimir, `print()` añade automáticamente un salto de línea. En entornos interactivos, se puede usar `display()` para una representación visual más rica.
R	`<-`	`print()`	Aunque `=` funciona, se prefiere `<-`. Además de `print()`, existe `cat()`, útil para concatenar texto sin mostrar índices de vector.
Julia	`=`	`println()`	Se usa `=` para asignación. `println()` imprime con salto de línea, mientras que `print()` lo hace sin él. Al igual que en Python, existe `display()`.

Python

```
# #| eval: false
#Variable numérica
numero_dptos = 32

#Imprimir
print(numero_dptos)
```

```
32
```

```
#Ver contenido (en modo interactivo)
numero_dptos
```

```
32
```

R

```
# #| eval: false
# Variable numérica (en R se prefiere el operador <-)
numero_dptos <- 32

# Imprimir
print(numero_dptos)
```

```
[1] 32
```

```
# Ver contenido
numero_dptos
```

```
[1] 32
```

Julia

```
# #| eval: false
j_eval('
# Variable numérica
numero_dptos = 32

# Imprimir
println(numero_dptos)

# Ver contenido
numero_dptos
')
```

```
Starting Julia ...
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Va

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>nume
32

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Im

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>print
32

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Ve

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>nume
32


[1] 32
```

Asignación de variables

Table 2: Estructuras de datos y comportamiento de asignación

Lenguaje	Colección Base	Operador	Descripción y Flexibilidad
Python	Lista `[]`	`=`	Permite mezclar tipos en una lista. Es dinámico y **fuerte**: no permite operaciones inválidas entre tipos (ej. sumarle un texto a un número).
R	Vector `c()`	`<-`	El vector atómico (`c`) exige que todos los elementos sean del mismo tipo. Si se mezclan, R los convierte automáticamente (coerción).
Julia	Arreglo `[]`	`=`	Muy similar a la lista de Python pero optimizado para rendimiento. Es dinámico pero permite declarar tipos para ganar velocidad.

Conceptos clave de programación

Para entender cómo estos lenguajes manejan la información de la Table 2, es fundamental diferenciar los sistemas de tipado:

1. **Tipado Dinámico:** El tipo de la variable se define en tiempo de ejecución. No es necesario declarar que una variable es un entero o un texto antes de usarla; el lenguaje lo infiere. (Python, R y Julia son dinámicos).
2. **Tipado Estático:** El tipo de la variable debe definirse al momento de escribir el código (ej. C++ o Java). Una vez definida como "entero", no puede guardar texto. Esto previene errores antes de ejecutar el programa.
3. **Tipado Fuerte:** El lenguaje no permite operaciones entre tipos incompatibles sin una conversión explícita. Por ejemplo, en **Python**, `5 + "10"` arrojará un error.
4. **Tipado Débil:** El lenguaje intenta realizar conversiones automáticas (coerción) para que la operación funcione. En **R**, si intentas unir un número y un texto en un vector `c(1, "Bogotá")`, R convertirá el `1` en texto `"1"` silenciosamente.

Python

```python
# #| eval: false
# Iniciar con un número (Latitud de Monserrate)
datos_ubicacion = 4.6052

# Cambiar a texto (Nombre del lugar)
datos_ubicacion = "Cerro de Monserrate"

# Cambiar a una lista de coordenadas [Lat, Lon]
datos_ubicacion = [4.6052, -74.0554]

# Imprimir resultado final
print(datos_ubicacion)
```

```
[4.6052, -74.0554]
```

R

```r
# #| eval: false
# Iniciar con un número (Latitud de Monserrate)
datos_ubicacion <- 4.6052

# Cambiar a texto (Nombre del lugar)
datos_ubicacion <- "Cerro de Monserrate"

# Cambiar a un vector de coordenadas (c es para combinar/concatenar)
datos_ubicacion <- c(4.6052, -74.0554)
```

```
# Imprimir resultado final
print(datos_ubicacion)
```

```
[1]   4.6052 -74.0554
```

Julia

```
# #| eval: false
j_eval('
# Iniciar con un número (Latitud de Monserrate)
datos_ubicacion = 4.6052

# Cambiar a texto (Nombre del lugar)
datos_ubicacion = "Cerro de Monserrate"

# Cambiar a un arreglo (Array) de coordenadas
datos_ubicacion = [4.6052, -74.0554]

# Imprimir resultado final
println(datos_ubicacion)
')
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Ini

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>datos
4.6052

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Car

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>datos
"Cerro de Monserrate"

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Car

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>datos
2-element Vector{Float64}:
   4.6052
 -74.0554

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Imp
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>print
[4.6052, -74.0554]
```

```
[1] "[4.6052, -74.0554]"
```

Nombres para las variables

¿Por qué es clave usar nombres adecuados?

En programación, los nombres que eliges para tus variables no son solo etiquetas; son las instrucciones que permiten que otras personas (y tú mismo en el futuro) entiendan la lógica del código sin errores. Usar nombres estandarizados es vital por las siguientes razones:

1. **Evitar errores técnicos:** Los lenguajes de programación son muy estrictos. Un espacio, una tilde o una `ñ` pueden hacer que un programa que funciona bien en tu computadora falle al abrirlo en otra.
2. **Claridad en la lectura:** Es mucho más fácil entender un proceso si la variable se llama `distancia_metros` en lugar de simplemente `d`.
3. **Portabilidad:** El uso de estándares internacionales asegura que tus datos e investigaciones puedan integrarse fácilmente entre diferentes plataformas como **Python**, **R** o **Julia**.

i Resumen de Reglas de Estilo (snake_case)

Para que tus programas sean profesionales y compatibles, sigue estas reglas básicas:

- **Todo en minúsculas:** Evita mezclar mayúsculas para mantener la uniformidad.
- **Usa el guion bajo (_):** Dado que los espacios están prohibidos, el guion bajo une las palabras (ej. `precio_gasolina`).
- **Solo letras básicas y números:** Usa solo caracteres de la `a` a la `z`. Evita tildes, eñes o símbolos especiales (`!`, `#`, `$`).
- **Prohibido el guion medio (-):** El computador lo interpreta como una operación de resta.

```
#| code-summary: "Ejemplos comparativos (Python, R, Julia)"

# FORMA CORRECTA: Clara y sin errores
estacion_climatologica = "Dorado"
temperatura_celsius = 20.5
conteo_puntos_gps = 15
```

```
# FORMA INCORRECTA: Genera errores o confusión
estacion climatologica = "Dorado"  # ERROR: Los espacios no están permitidos
temperatura-celsius = 20.5         # ERROR: El guion medio intenta restar
t = 20.5                           # MAL: Es una variable muy vaga
año_inicio = 2024                  # EVITAR: La 'ñ' causa fallos de lectura
```

Python

```
# #| eval: false
# Nombres de variables recomendatos
# Variables geoespaciales claras
latitud_bogota = 4.7110
longitud_bogota = -74.0721
altura_msnm = 2640.0
municipio_nombre = "Chinchiná"
conteo_viviendas_afectadas = 150
sistema_referencia = "MAGNA-SIRGAS / Origen Nacional"


# Nombres de variables NO recomendatos
# Evitar nombres genéricos o confusos
x = 4.7110  # ¿Es latitud o un índice?
d = "Chinchiná"  # Muy vago (¿Distrito, Departamento, Dato?)
val = 2640  # Ambiguo: ¿Valor, Valencia, Variable?
coords = [4.71, -74.07]  # Lista sin etiquetas claras
```

R

```
# #| eval: false
# Nombres de variables recomendatos
# Estilo snake_case (común en R para análisis de datos)
latitud_medellin <- 6.2442
longitud_medellin <- -75.5812
cota_terreno <- 1495
nombre_departamento <- "Antioquia"
poblacion_censo_2018 <- 6407000
proyeccion_cartografica <- "EPSG:9377"
```

```r
# Nombres de variables NO recomendatos
# Evitar abreviaturas extremas
l <- 6.2442 # ¿Latitud, Longitud, Límite, Localidad?
nom <- "Antioquia" # "nombre" es mejor
temp <- 1495 # ¿Temperatura o un archivo temporal?
p <- 6407000 # ¿Población, Perímetro, Pendiente, P-value?
```

Julia

```r
# #| eval: false
j_eval('
# Nombres de variables recomendatos
# Aprovechando el soporte Unicode de Julia
latitud_cali = 3.4516
longitud_cali = -76.5320
elevación_cauca = 995.0
nombre_vereda = "La Elvira"
densidad_poblacional = 120.5
referencia_espacial = "MAGNA-SIRGAS"
')
```

julia> # Nor

julia> # Apr

julia> latit
3.4516

julia> longi
-76.532

julia> eleva
995.0

julia> nombr
"La Elvira"

julia> densi
120.5

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>refe
"MAGNA-SIRGAS"
```

```
[1] "MAGNA-SIRGAS"
```

```
j_eval('
# Nombres de variables NO recomendatos
# Nombres que no dicen nada del contexto geográfico
var1 = 3.4516
info = "La Elvira"
tmp = 995.0 # Típico error: ¿Temperatura o Temporal?
lista = [3.45, -76.53] # Difícil de leer en modelos de optimización
')
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Nor

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Nor

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>var1
3.4516

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>info
"La Elvira"

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>tmp =
995.0

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>lista
2-element Vector{Float64}:
   3.45
 -76.53
```

```
[1] -76.53
```

Tipos de datos

Python

```
# #| eval: false

# Representa el número de elementos en un dataset geoespacial
num_elementos = 500

# Representa la latitud de un punto (Nevado del Ruiz)
latitud = 4.8920

# Representa la longitud de un punto
longitud = -75.3188

# Representa el sistema de referencia oficial de Colombia
sistema_coordenadas = "MAGNA-SIRGAS / Origen Nacional"

# Representa si el dataset está georreferenciado o no
esta_georeferenciado = True

# Una lista que representa latitud y longitud
coordenadas = [4.8920, -75.3188]

# Un diccionario con los atributos del elemento geográfico
atributos_elemento = {
    "nombre": "Nevado del Ruiz",
    "altura_msnm": 5321,
    "tipo": "Estratovolcán",
    "ubicacion": [4.8920, -75.3188],
}

print(atributos_elemento)
```

```
{'nombre': 'Nevado del Ruiz', 'altura_msnm': 5321, 'tipo': 'Estratovolcán', 'ubicacion': [4.8
```

```
# Tipo de dato
type(atributos_elemento)
```

```
<class 'dict'>
```

```
# Acceder a la primera coordenada (índice 0) dentro de la clave 'ubicacion'
latitud_ruiz = atributos_elemento["ubicacion"][0]
latitud_ruiz
```

4.892

R

```
# #| eval: false

# Representa el número de elementos (L indica tipo integer)
num_elementos <- 500L

# Representa la latitud de un punto (Nevado del Ruiz)
latitud <- 4.8920

# Representa la longitud de un punto
longitud <- -75.3188

# Representa el sistema de referencia oficial (tipo character)
sistema_coordenadas <- "MAGNA-SIRGAS / Origen Nacional"

# Representa si el dataset está georreferenciado (logical)
esta_georeferenciado <- TRUE

# Un vector que representa latitud y longitud
coordenadas <- c(4.8920, -75.3188)

# Una lista que representa los atributos del elemento geográfico
atributos_elemento <- list(
    "nombre" = "Nevado del Ruiz",
    "altura_msnm" = 5321,
    "tipo" = "Estratovolcán",
    "ubicacion" = c(4.8920, -75.3188)
)

print(atributos_elemento)
```

```
$nombre
[1] "Nevado del Ruiz"

$altura_msnm
[1] 5321

$tipo
```

```
[1] "Estratovolcán"

$ubicacion
[1]   4.8920 -75.3188
```

```r
# Tipo de dato
class(atributos_elemento)
```

```
[1] "list"
```

```r
# Acceder a la latitud (índice 1 en R) dentro del elemento 'ubicacion'
latitud_ruiz <- atributos_elemento$ubicacion[1]
latitud_ruiz
```

```
[1] 4.892
```

Julia

```
# #| eval: false
j_eval('
# Representa el número de elementos en el dataset
num_elementos = 500

# Representa la latitud del punto (Nevado del Ruiz)
latitud = 4.8920

# Representa la longitud del punto
longitud = -75.3188

# Representa el sistema de referencia espacial
sistema_coordenadas = "MAGNA-SIRGAS / Origen Nacional"

# Representa el estado de georreferenciación (tipo Bool)
esta_georeferenciado = true

# Un arreglo que representa latitud y longitud
coordenadas = [4.8920, -75.3188]

# Un diccionario (Dict) para almacenar los atributos
atributos_elemento = Dict(
```

```
    "nombre" => "Nevado del Ruiz",
    "altura_msnm" => 5321,
    "tipo" => "Estratovolcán",
    "ubicacion" => [4.8920, -75.3188],
)

println(atributos_elemento)

# Tipo de dato
typeof(atributos_elemento)

# Acceder a la latitud (índice 1 en Julia) dentro de la clave "ubicacion"
latitud_ruiz = atributos_elemento["ubicacion"][1]
latitud_ruiz
')
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Rep

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>num_
500

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Rep

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>latit
4.892

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Rep

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>longi
-75.3188

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Rep

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>siste
"MAGNA-SIRGAS / Origen Nacional"

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Rep

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>esta
true

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Un
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>coord
2-element Vector{Float64}:
   4.892
 -75.3188

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Un

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>atrib
    "nombre" => "Nevado del Ruiz",
    "altura_msnm" => 5321,
    "tipo" => "Estratovolcán",
    "ubicacion" => [4.8920, -75.3188],
)
Dict{String, Any} with 4 entries:
  "tipo" => "Estratovolcán"
  "nombre" => "Nevado del Ruiz"
  "altura_msnm" => 5321
  "ubicacion" => [4.892, -75.3188]

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>print
Dict{String, Any}("tipo" => "Estratovolcán", "nombre" => "Nevado del Ruiz", "altura_msnm" =>

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Tip

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>typeo
Dict{String, Any}

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Acc

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>latit
4.892

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>latit
4.892


[1] 4.892
```

Table 3: Comparación detallada de sintaxis y tipos de datos

Característica / Variable	Python	R	Julia
Sufijo `L` (num_elementos)	No se usa. `500` es entero.	Se usa `500L` para `integer`. Sin `L` es `numeric`.	No se usa. `500` es `Int64`.
Asignación en Diccionario	`v = {clave: valor, ...}` (Usa `:`).	`v <- list(clave = valor, ...)`. (Usa `=`)	`v = Dict(clave => valor, ...)` (Usa `=>`).
`num_elementos`	`int`	`integer` (con el sufijo `L`)	`Int64`
`latitud` / `longitud`	`float`	`numeric` (doble precisión)	`Float64`
`sistema_coordenadas`	`str`	`character`	`String`
`esta_georeferenciado`	`bool` (`True` / `False`)	`logical` (`TRUE` / `FALSE`)	`Bool` (`true` / `false`)
`coordenadas`	`list [ , ]`: Colección mutable de objetos.	`vector c( , )`: Colección indexada de elementos del mismo tipo.	`Array [ , ]` (Vector): Arreglo indexado y optimizado para cómputo.
`atributos_elemento`	`dict { : }`: Estructura nativa de pares clave-valor.	`list list( = )`: Lista con nombres (etiquetas) para cada elemento.	`Dict Dict( => )`: Tipo de dato optimizado para mapeos clave-valor.

¿Qué es un diccionario?

Un **Diccionario** es una estructura de datos que organiza la información mediante pares de **Clave-Valor** (*Key-Value*). A diferencia de las secuencias indexadas (como las listas o vectores) donde los elementos se recuperan por su posición numérica, en un diccionario se accede a la información a través de etiquetas únicas llamadas claves.

- **Identificación:** Cada valor almacenado debe tener una clave asociada que funciona como su identificador único.
- **Heterogeneidad:** Permite agrupar diversos tipos de datos (cadenas, números, arreglos) bajo un mismo objeto.
- **Acceso Directo:** Su implementación técnica permite que la recuperación de un dato sea extremadamente eficiente, sin importar el volumen de información.
- **Estándares:** Es el concepto fundamental detrás de formatos como JSON, facilitando la organización jerárquica de los datos.

Caracteres de escape

Python

```
# #| eval: false

# Salto de línea para separar información de capas geográficas
print("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")
```

```
Capa: Departamentos_Colombia
Estado: Procesada exitosamente.
```

```
# Salto de línea y tabulación para jerarquizar metadatos
print("Metadatos del proyecto:\n\tAutor: IGAC\n\tEscala: 1:100.000")
```

```
Metadatos del proyecto:
    Autor: IGAC
    Escala: 1:100.000
```

```
# Uso de comillas simples dentro de dobles para nombres propios
print("Nombre del archivo: 'Mapa_Relieve_Colombia.shp'")
```

```
Nombre del archivo: 'Mapa_Relieve_Colombia.shp'
```

R

```
# #| eval: false

# Salto de línea para separar información de capas geográficas
print("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")
```

```
[1] "Capa: Departamentos_Colombia\nEstado: Procesada exitosamente."
```

```
# Salto de línea y tabulación para jerarquizar metadatos (cat interpreta mejor los caracteres
cat("Metadatos del proyecto:\n\tAutor: IGAC\n\tEscala: 1:100.000")
```

```
Metadatos del proyecto:
    Autor: IGAC
    Escala: 1:100.000
```

```
# Uso de comillas simples dentro de dobles para nombres propios
print("Nombre del archivo: 'Mapa_Relieve_Colombia.shp'")
```

```
[1] "Nombre del archivo: 'Mapa_Relieve_Colombia.shp'"
```

Julia

```
# #| eval: false
j_eval('
# Salto de línea para separar información de capas geográficas
println("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")

# Salto de línea y tabulación para jerarquizar metadatos
println("Metadatos del proyecto:\n\tAutor: IGAC\n\tEscala: 1:100.000")

# Uso de comillas simples dentro de dobles para nombres propios
println("Nombre del archivo: \'Mapa_Relieve_Colombia.shp\'")
')
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Sal

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>print
Estado: Procesada exitosamente.")
Capa: Departamentos_Colombia
Estado: Procesada exitosamente.

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Sal

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>print
    Autor: IGAC
    Escala: 1:100.000")
Metadatos del proyecto:
    Autor: IGAC
    Escala: 1:100.000

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Uso
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>print
Nombre del archivo: 'Mapa_Relieve_Colombia.shp'
```

```
[1] "Nombre del archivo: 'Mapa_Relieve_Colombia.shp'"
```

Caracteres especiales de escape

Table 4: Caracteres de escape comunes en lenguajes de programación

Carácter	Nombre	Descripción Técnica
`\n`	Salto de línea (Newline)	Mueve el cursor al inicio de la siguiente línea para organizar texto.
`\t`	Tabulación (Tab)	Inserta un espacio horizontal para crear jerarquías o sangrías.
`\\`	Barra invertida	Permite imprimir el carácter `\` (crucial para rutas de archivos en sistemas locales).
`\"`	Comilla doble	Permite insertar comillas dobles sin cerrar prematuramente la cadena de texto.
`\'`	Comilla simple	Permite insertar comillas simples; en Julia/R dentro de `j_eval` suele requerir escape extra.

Comentarios

Python

```python
# #| eval: false

# Comentario de bloque: Información de procesamiento
print("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")
```

```
Capa: Departamentos_Colombia
Estado: Procesada exitosamente.
```

```python
# Comentario en línea: RMSE en metros
rmse = 0.05  # Tolerancia para precisión submétrica

"""
Este es un comentario multilínea en Python.
Se utilizan triples comillas (aunque técnicamente son cadenas
de texto no asignadas, se usan para documentar bloques extensos).
"""
```

```
'\nEste es un comentario multilínea en Python.\nSe utilizan triples comillas (aunque técnicam
```

```python
print("Nombre del archivo: 'Mapa_Relieve_Colombia.shp'")
```

```
Nombre del archivo: 'Mapa_Relieve_Colombia.shp'
```

R

```r
# #| eval: false

# Comentario de bloque: Información de procesamiento
print("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")
```

```
[1] "Capa: Departamentos_Colombia\nEstado: Procesada exitosamente."
```

```r
# Comentario en línea: RMSE en metros
rmse <- 0.05  # Tolerancia para precisión submétrica

# R no tiene un operador nativo para comentarios multilínea.
# Se deben usar múltiples numerales consecutivos para
# documentar bloques de código extensos.
cat("Metadatos del proyecto:\n\tAutor: IGAC\n\tEscala: 1:100.000")
```

```
Metadatos del proyecto:
    Autor: IGAC
    Escala: 1:100.000
```

Julia

```
# #| eval: false
j_eval('
# Comentario de bloque: Información de procesamiento
println("Capa: Departamentos_Colombia\nEstado: Procesada exitosamente.")

# Comentario en línea: RMSE en metros
rmse = 0.05  # Tolerancia para precisión submétrica

#=
Este es un comentario multilínea nativo en Julia.
Permite comentar grandes bloques de código o explicaciones
extensas sin necesidad de colocar numerales en cada línea.
=#
println("Nombre del archivo: \'Mapa_Relieve_Colombia.shp\'")
')
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Co

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>print
Estado: Procesada exitosamente.")
Capa: Departamentos_Colombia
Estado: Procesada exitosamente.

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Co

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>rmse
0.05

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>#=</s
Este es un comentario multilínea nativo en Julia.
Permite comentar grandes bloques de código o explicaciones
extensas sin necesidad de colocar numerales en cada línea.
=#

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>print
Nombre del archivo: 'Mapa_Relieve_Colombia.shp'


[1] "Nombre del archivo: 'Mapa_Relieve_Colombia.shp'"
```

Documentación y caracteres especiales

Table 5: Sintaxis de documentación y escape

Carácter	Nombre	Función Técnica
`#`	Numeral	Inicia comentarios de una sola línea en los tres lenguajes.
`\n`	Newline	Inserta un salto de línea dentro de una cadena de texto.
`\t`	Tab	Inserta una tabulación horizontal (sangría).
`"""`	Triple comilla	Usado en Python para documentación de bloques (*docstrings*).
`#= =#`	Block comment	Delimitador nativo de Julia para comentarios multilínea.

Nota técnica: En el desarrollo de herramientas geomáticas automatizadas, los comentarios multilínea son la ubicación estándar para definir el **encabezado del script**, detallando la licencia, el sistema de referencia de coordenadas (CRS) esperado por el algoritmo y los requerimientos de librerías. Mientras que Julia ofrece una sintaxis limpia con `#= =#`, en Python es vital recordar que las triples comillas `"""` son técnicamente *strings* literales que el intérprete descarta si no están asignadas.

Trabajando con variables y tipos de datos

Python

```python
# #| eval: false

import math

# Inicialización de variables para asegurar la ejecución
num_elementos = 500

# Incrementar el número de elementos existentes
num_elementos += 20
num_elementos
```

520

```python
# Convertir la latitud de grados decimales a radianes
latitud_ruiz = 4.8920
latitud_radianes = math.radians(latitud_ruiz)

# Agregar nuevas coordenadas a la lista [Lat, Lon] de Bogotá
coordenadas = [4.8920, -75.3188]
coordenadas.append(4.6097)  # Latitud de Bogotá
coordenadas.append(-74.0817) # Longitud de Bogotá

# Acceso y formateo de cadenas usando el diccionario de atributos
nombre_volcan = atributos_elemento["nombre"]
altura_volcan = atributos_elemento["altura_msnm"]
mensaje = f"{nombre_volcan} tiene una altura de {altura_volcan} metros."

# Mostrar el resultado final
mensaje
```

```
'Nevado del Ruiz tiene una altura de 5321 metros.'
```

R

```r
# #| eval: false

# Inicialización de variables necesarias
num_elementos <- 500
# Incrementar el número de elementos
num_elementos <- num_elementos + 20
num_elementos
```

```
[1] 520
```

```r
# Convertir latitud a radianes
latitud_ruiz <- 4.8920
latitud_radianes <- latitud_ruiz * (pi / 180)

# Agregar elementos a un vector (creando una copia combinada)
coordenadas <- c(4.8920, -75.3188)
coordenadas <- c(coordenadas, 4.6097, -74.0817)
```

```r
# Acceso a lista y formateo de cadenas
nombre_volcan <- atributos_elemento$nombre
altura_volcan <- atributos_elemento$altura_msnm
mensaje <- paste(nombre_volcan, "tiene una altura de", altura_volcan, "metros.")

# Mostrar el resultado final
mensaje
```

```
[1] "Nevado del Ruiz tiene una altura de 5321 metros."
```

Julia

```julia
# #| eval: false
j_eval('
# Inicialización de variables necesarias
num_elementos = 500
# Incrementar el número de elementos
num_elementos += 20
num_elementos

# Convertir latitud a radianes usando la función nativa deg2rad
latitud_ruiz = 4.8920
latitud_radianes = deg2rad(latitud_ruiz)

# Agregar coordenadas a un arreglo
coordenadas = [4.8920, -75.3188]
push!(coordenadas, 4.6097)
push!(coordenadas, -74.0817)

# Acceso e interpolación de texto
nombre_volcan = atributos_elemento["nombre"]
altura_volcan = atributos_elemento["altura_msnm"]
mensaje = "$nombre_volcan tiene una altura de $altura_volcan metros."

# Mostrar el resultado final
mensaje
')
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Ini
```

```
<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>num_
500

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># In

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>num_
520

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>num_
520

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Co

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>lati
4.892

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>lati
0.08538150700756261

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Ag

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>coor
2-element Vector{Float64}:
   4.892
 -75.3188

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>push
3-element Vector{Float64}:
   4.892
 -75.3188
   4.6097

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>push
4-element Vector{Float64}:
   4.892
 -75.3188
   4.6097
 -74.0817

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Ac

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>nomb
```

```
"Nevado del Ruiz"

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>altu
5321

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>mensa
"Nevado del Ruiz tiene una altura de 5321 metros."

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'># Mos

<span style='color:green;font-weight:bold;'>julia> </span><span style='color:darkcyan;'>mensa
"Nevado del Ruiz tiene una altura de 5321 metros."


[1] "Nevado del Ruiz tiene una altura de 5321 metros."
```

Table 6: Comparación de operaciones comunes con variables

Operación	Python	R	Julia
Incremento	`+=`	`x <- x + n`	`+=`
Grados a Radianes	`math.radians()`	`x * (pi/180)`	`deg2rad()`
Agregar a lista	`.append()` (Mutable)	`c(x, n)` (Inmutable)	`push!()` (Mutable)
Interpolación de texto	`f"{var}"`	`paste()` / `sprintf()`	`"$var"`

Nota técnica: En el desarrollo de algoritmos geográficos, la mutabilidad es un concepto clave. Mientras que Python y Julia permiten modificar una lista de coordenadas directamente (`append` / `push!`), en R los vectores son inmutables; cada vez que "agregamos" un dato, R crea una copia nueva del vector en memoria.

Operaciones básicas con texto

Python

R

Julia

Aprendizajes principales

Python

R

Julia

Ejercicios

Ejercicio 1: Asignación de variables y operaciones básicas

Ejercicio 2: Trabajando con strings