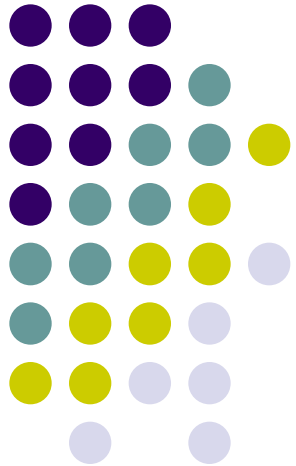
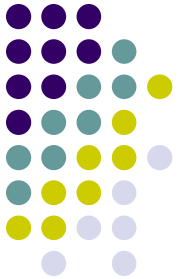
Programación en R

Estructuras de programación

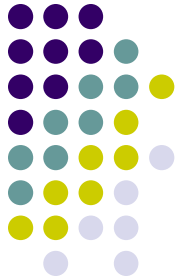


Objetivos



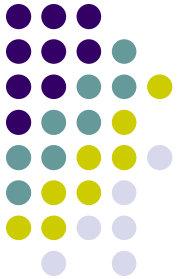
- Conocer algunas estructuras útiles de programación en R: estructuras condicionales y estructuras de repetición tipo bucle.
- Construir funciones haciendo uso de distintos elementos de programación.

Operadores de relación y aritméticos



- Operadores de relación básicos: **!** (negación), **&** (conjunción) y **|** (disyunción).
- Estos dos últimos, si se escriben repetidos (ej. `condición1 && condicion2`) tienen el mismo significado. La ventaja es que se evalúa primero la parte de la izquierda (`condición 1`) y, si ya se sabe el resultado no se sigue evaluando, por lo que pueden ser más rápidos y eliminar errores.
- Los operadores aritméticos: **<**, **>**,**<=**,**>=**,**==** (menor, mayor, menor o igual, mayor o igual, e igual, respectivamente).

Estructuras condicionales



- Son aquellas que, según el resultado de una comparación, realizan una u otra acción.
- Recuerde que un conjunto de acciones, entre llaves, se interpretan como una sola acción desde el punto de vista lógico.
- Nos centraremos en las siguientes estructuras:
  - La estructura **`if`**
    - una variante escalar:

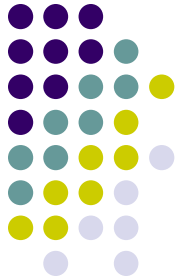
      `if (condición) acción1 [else acción2]`
    - otra vectorial:

`ifelse(condición, acción en caso cierto, acción en caso falso)`

  - La estructura **`switch`**:

`switch(expresión,[valor-1=]acción-1,....,`
`[valor-n=]acción-n)`

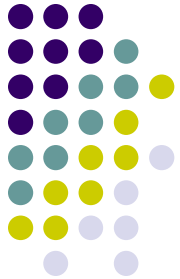
Estructuras condicionales: `if`



- La estructura **`if`** es de tipo escalar, esto es, si la condición está formada por más de un elemento, sólo considera el primero.
- Su sintaxis es: **`if (condición) acción1 [else acción2]`**
- Ejemplo:

```
raiz<-function(x)
{
   if (is.numeric(x) && min(x)>0)
   # Advierta que no se intenta aplicar
   # la función min si el argumento
   # no es numérico, previniendo un error
   {sqrt(x)} else {
      stop("O x no es numérico o no es positivo")
   }
}
```

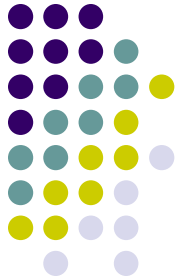
Estructuras condicionales: `if`



```
> raiz("Pepe")
Error in raiz("Pepe"): O x no es numerico o no es
   positivo
> raiz(-1)
Error in raiz(-1): O x no es numerico o no es
   positivo
> raiz(7)
[1] 2.645751
```

- Si desea que se realicen varias acciones debe incluirlas entre llaves, para que aparezcan como una sola desde el punto de vista lógico.
- Observe que en este ejemplo “acción1” y “acción2” son sendas acciones individuales, por lo que las llaves podrían eliminarse.

Estructuras condicionales: `ifelse`



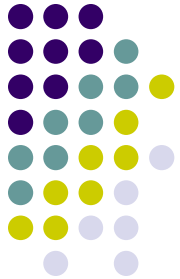
- La estructura **`ifelse`** es de tipo vectorial, esto es, si la condición está formada por más de un elemento, considera cada uno de ellos.
- Su sintaxis es:

```
ifelse(condición, acción en caso cierto, acción en caso
                                        falso)
```

- Ejemplo. Suponiendo que el argumento “x” es numérico, tendríamos:

```
> Inverso<-function(x) ifelse(x==0,NA,1/x)
> Inverso(-1:1)
[1] -1 NA 1
> Inverso(-2:2)
[1] -0.5 -1.0 NA 1.0 0.5
```

Estructuras condicionales: `switch`

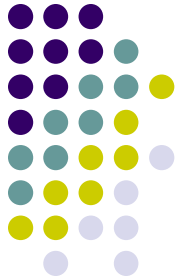


- Su sintaxis es:

```
switch(expresión,[valor-1=]acción-1,....,
                 [valor-n=]acción-n)
```

- La estructura **`switch`** es escalar y además expresión debe devolver un vector de longitud 1.
- La función se comporta de modo distinto según que la expresión evalúe a numérico o a carácter.
- Cuando la expresión devuelve un valor numérico, se transforma el resultado en un entero, i, y si está entre 1 y n, se evalúa la acción-i, tenga o no un valor asociado, y se devuelve el resultado, en caso contrario se devuelve NULL.

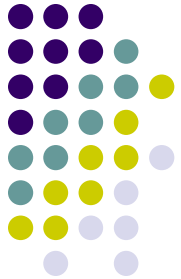
Estructuras condicionales: `switch`



```
switch(expresión,[valor-1=]acción-1,....,
                 [valor-n=]acción-n)
```

- Cuando la expresión devuelve una cadena de caracteres, si este coincide exactamente con uno de los valores, se evalúa la acción correspondiente y se devuelve el resultado. Si no coincide con ninguno y una acción (normalmente la última se reserva para ello) no tiene un valor asociado, se ejecuta esta y se devuelve su resultado; pero si todas tienen valor asociado, se devuelve NULL.
- Si hay varias acciones sin valor asociado se devuelve la primera de ellas.

Estructuras condicionales: `switch`

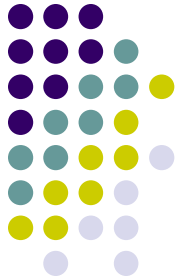


```
switch(expresión,[valor-1=]acción-1,....,
                          [valor-n=]acción-n)
```

- Ejemplos:

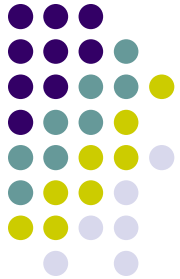
```
> n<-2
> switch(n,"Uno","Dos","Tres")
[1] "Dos"
> n<-3
> switch(n,"Uno","Dos","Tres")
[1] "Tres"
> n<-4
> switch(n,"Uno","Dos","Tres")
NULL
```

Estructuras condicionales: `switch`



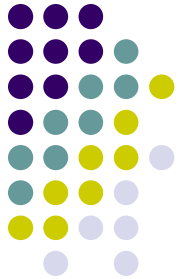
```
> Decide<-function(x)
switch(x,m=cat("Has dicho eme minúscula"),
M=cat("Has dicho eme mayúscula"),cat("Dime m o M"))
> Decide
function(x)
switch(x,
m = cat("Has dicho eme minúscula"),
M = cat("Has dicho eme mayúscula"),
cat("Dime m o M\n"))
> Decide("m")
Has dicho eme minúscula
> Decide("M")
Has dicho eme mayúscula
> Decide("P")
Dime m o M
```

Estructuras condicionales: `switch`



```
> n<-"Dos"
> switch(n,"Uno"=1,"Dos"=2,"Tres"=3,"Distinto")
[1] 2
> n<-"Cuatro"
> switch(n,"Uno"=1,"Dos"=2,"Tres"=3,"Distinto")
[1] "Distinto"
```

Estructuras de repetición definida e indefinida



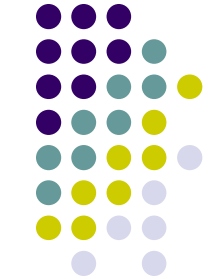
- Nos centraremos en las siguientes estructuras:
  - La estructura **`for`** que asigna a variable cada uno de los valores y realiza la acción para cada uno:

    `for (variable in valores) acción`
  - La estructura **`while`** evalúa la condición y mientras esta es cierta se evalúa la acción:

    `while (condición) acción`
  - La estructura **`repeat`** que evalúa la acción indefinidamente:

    `repeat acción`
- Las expresiones pueden contener algún condicional como **`if`** asociado con las funciones **`next`** o **`break`**.
- La estructura **`next`** indica que debe terminarse la iteración actual y pasar a la siguiente.
- La estructura **`break`** indica que debe terminarse el ciclo actual.

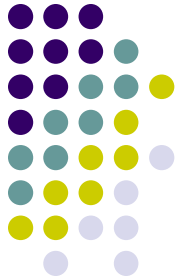
Estructuras de repetición definida e indefinida



- Ejemplos:

```
> for (i in -5:5) {cat(i,"\t",i^2,"\n")}
> for (i in -5:5) {if (i==0) next;cat(i,"\t",i^2,"\n")}
> for (i in -5:5) {if (i==0) break;cat(i,"\t",i^2,"\n")}
> i<-5;while (i>0) {print(i);i=i-1}
> i<-5;repeat{print(i);i=i-1;if(i==0) break}
```

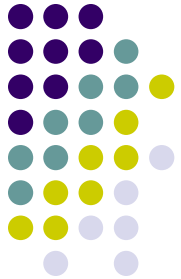
Función factorial



- A continuación, se define el factorial de un número (natural, de hecho no se ha incluido comprobación de que el argumento es correcto, **`is.integer`**, por lo que la función puede terminar con error si el argumento es de tipo carácter, por ejemplo).
- La función sufrirá ciertas depuraciones, por lo que aparecen varias versiones. Además la definiremos utilizando tanto un ciclo determinado como uno indeterminado:

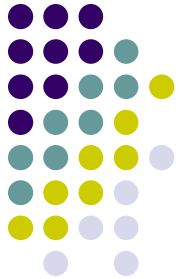
  - **`Factorial.d`**
  - **`Factorial.i`**
  - **`Factorial.r`**
  - **`Factorial.m`**

Función factorial



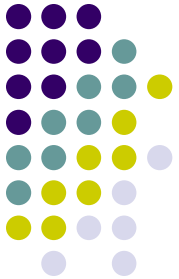
```
> Factorial.d<-function(n)
+ {
+ factorial <- 1
+ if(n>1)
+ for (i in 1:n)
+ factorial <- factorial * i
+ return(factorial)
+ }
> Factorial.d(3)
[1] 6
> Factorial.d(100)
[1] 9.332622e+157
> Factorial.d(0)
[1] 1
```

Función factorial



```
> Factorial.i<-function(n)
+ {
+ factorial <- 1
+ while(n > 0)
+ { factorial <- factorial * n
+ n <- n - 1 }
+ return(factorial)
+ }
> Factorial.i(3)
[1] 6
> Factorial.i(100)
[1] 9.332622e+157
> Factorial.i(0)
[1] 1
```

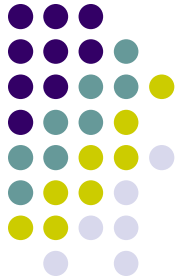
Función factorial



- Construimos ahora una función que se llama a sí misma (recursividad)

```
> Factorial.r<-function(n)
+ {
+ if(n > 1)
+ factorial <- n * Factorial.r(n - 1)
+ else factorial <- 1
+ return(factorial)
+ }
> Factorial.r(3)
> Factorial.r(300)
[1] Inf
> Factorial.r(1000)
Error: evaluation nested too deeply:
infinite recursion / options(expression=)?
```

Función factorial



```
> Factorial.m<-function(n)
+ {
+ if(n > 1)
+ return(prod(n:1))
+ else return(1)
+ }
> Factorial.m(3)
> Factorial.m(100)
```

- En este último caso no se llegan a agotar los recursos, pero se han consumido más que en los dos primeros, ya que es necesario generar el vector `n:1` y almacenarlo antes de calcular el producto de sus elementos, aunque es más eficiente desde el punto de vista de tiempo de cálculo.