
Aspetos da linguagem



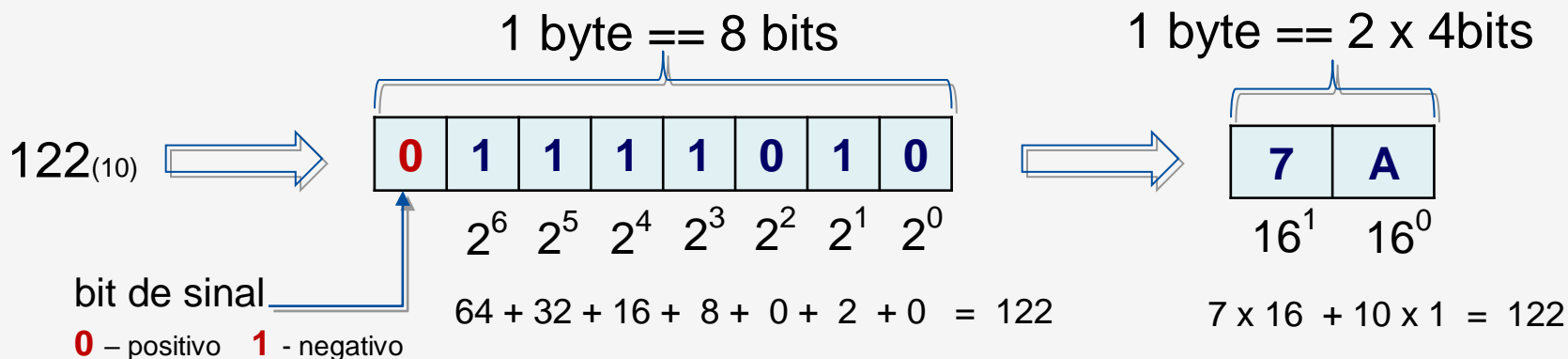
Programação (PG)

Tipos de valores inteiros e sua representação

- Inteiros (com sinal)

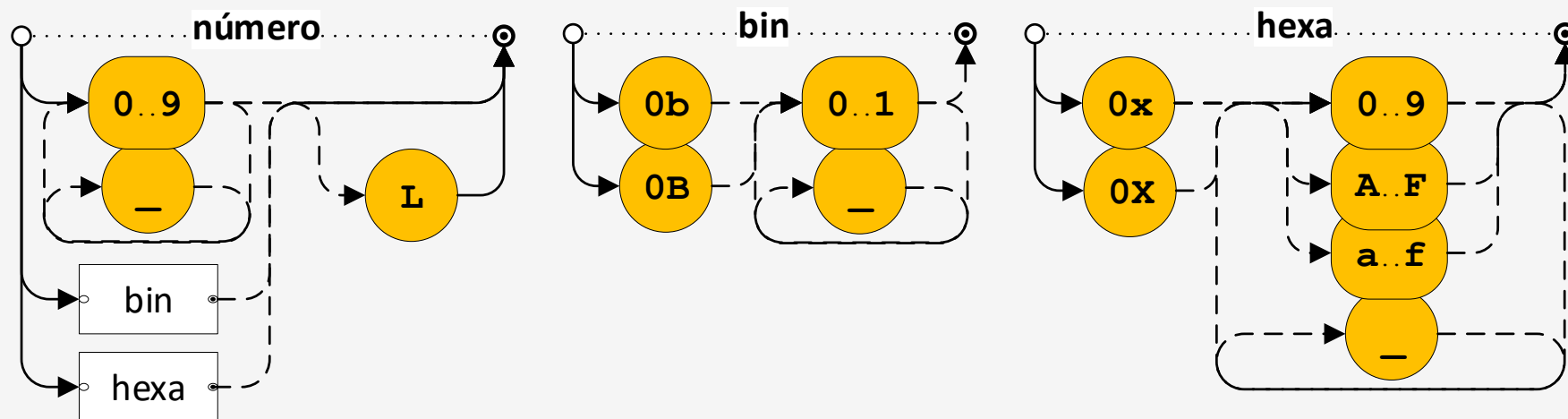
- Byte
- Short
- Int
- Long

Dimensão	Mínimo	Máximo
1 byte	-128	127
2 bytes	-32768	32767
4 bytes	-2147483648	2147483647
8 bytes	$-(2^{63})$	$2^{63}-1$



122₍₁₀₎ → 01111010₍₂₎ → 7A₍₁₆₎

Sintaxe para literais de valores inteiros



`122` :Int

`148L` :Long

`0b01111010` :Int

`0x7A` :Int

`1_430_800` :Int

`0b0111_1010L` :Long

`0x00FF00FF` :Int

`122.toByte()` :Byte

`400.toShort()` :Short

Operações aritméticas entre inteiros

Operador		Prioridade
+	Adição	baixa
-	Subtração	baixa
*	Multiplicação	alta
/	Quociente	alta
%	Resto	alta

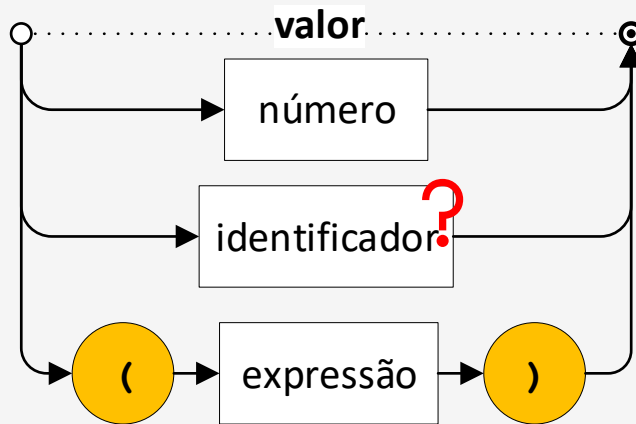
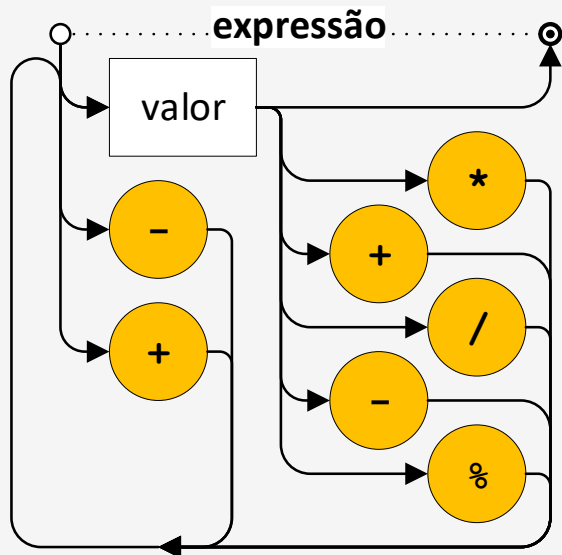
$$27 + 3 \Rightarrow 30 : \text{Int}$$

$$27 - 3L \Rightarrow 24 : \text{Long}$$

$$-2 + 2 * 7 \Rightarrow 12 : \text{Int}$$

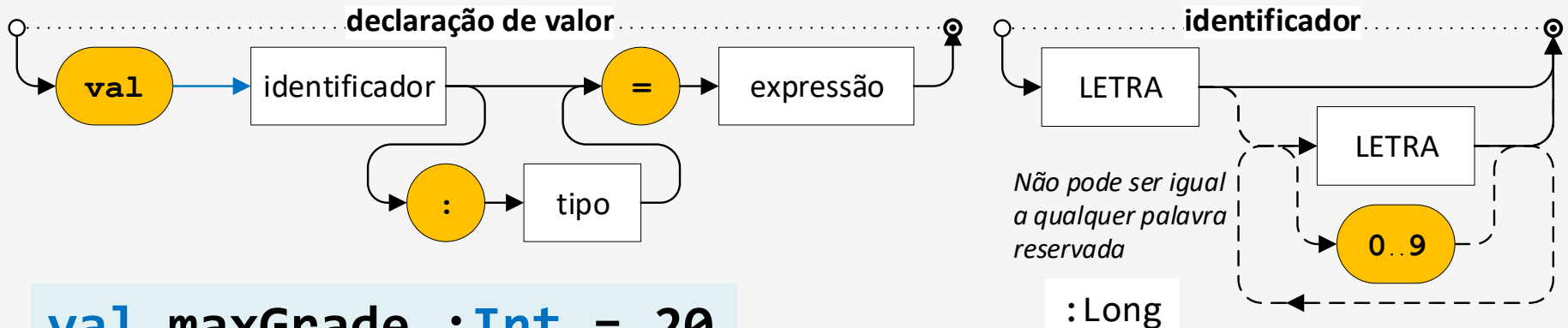
$$9 / (4 - 2) \Rightarrow 4 : \text{Int}$$

$$(3L + 4) \% 3 \Rightarrow 1 : \text{Long}$$

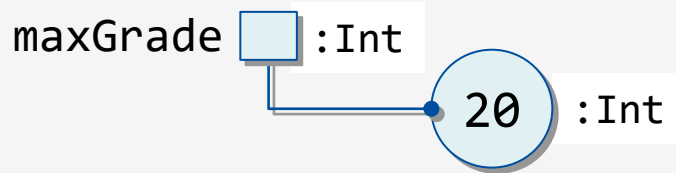


Resultado é **Long** se um dos argumentos for **Long**. Caso contrário, o resultado é **Int**, mesmo quando todos os argumentos são **Short** ou **Byte**.

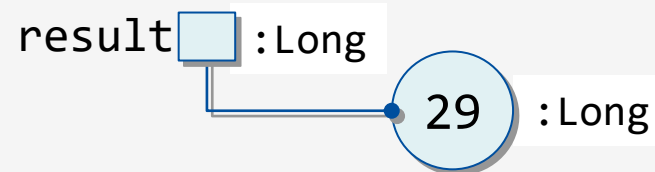
Declaração de valores



```
val maxGrade :Int = 20
```



```
val result = 27 + 2L
```



- Todos os valores têm um tipo
- O tipo é inferido quando não é indicado
- As declarações não são expressões
- Convenção *lowerCamelCase* nos identificadores

```
result / 10 => 2 :Long
```

Tipos de valores reais

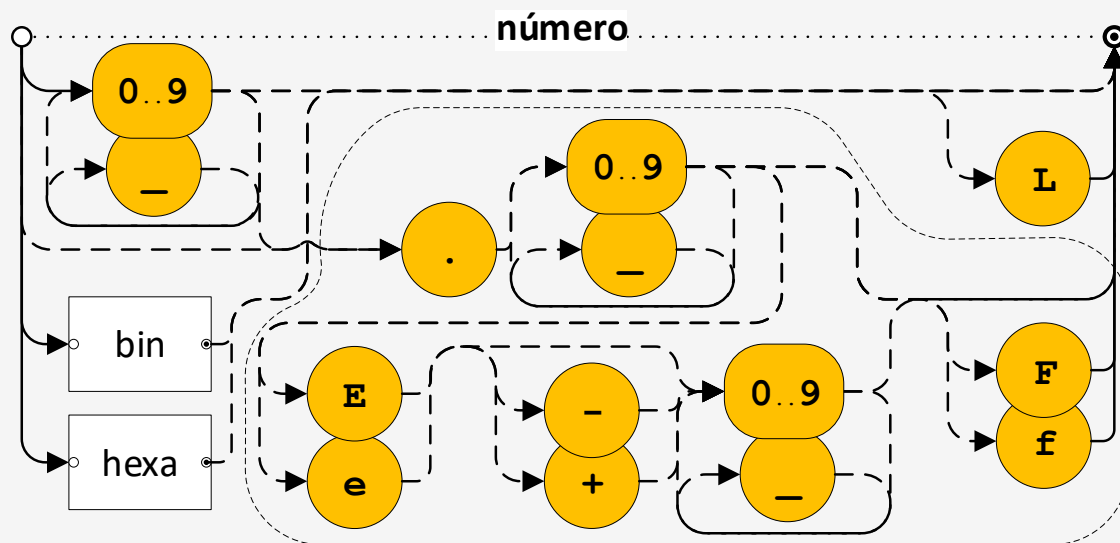
- Reais (norma IEEE 754)

- Float

- Double

Dimensão	Precisão
4 bytes	6 a 7 dígitos
8 bytes	15 a 16 dígitos

$$6,625_{(10)} \Rightarrow 4 + 2 + 0,5 + 0,125 = 2^2 + 2^1 + 2^{-1} + 2^{-3} \Rightarrow 110,101_{(2)}$$



32.54 :Double

:Float

5_200.0F

:Double

5.2e3

val pi = 3.14

pi 3.14 :Double

Operações com valores reais

- Resultado das operações aritméticas é do tipo do argumento mais abrangente.

Double → Float → Long → Int

3+.5*2F ⇒ 4.0 :Double

- Operador / faz a divisão real quando um dos argumentos é um valor real.

7.5F / 2 ⇒ 3.75 :Float

- Funções de conversão **toByte()**, **toShort()**, **toInt()**, **toLong()** ficam com a parte inteira do valor real.

7.9F.toInt() ⇒ 7 :Int

45.toFloat()/10 ⇒ 4.5 :Float

(45/10).toFloat() ⇒ 4.0 :Float

- A representação interna é a soma das potências de 2 mais aproximada.

val price = 5.30f // 5,30 euros

price - 0.1f ⇒ 5.2000003 :Float

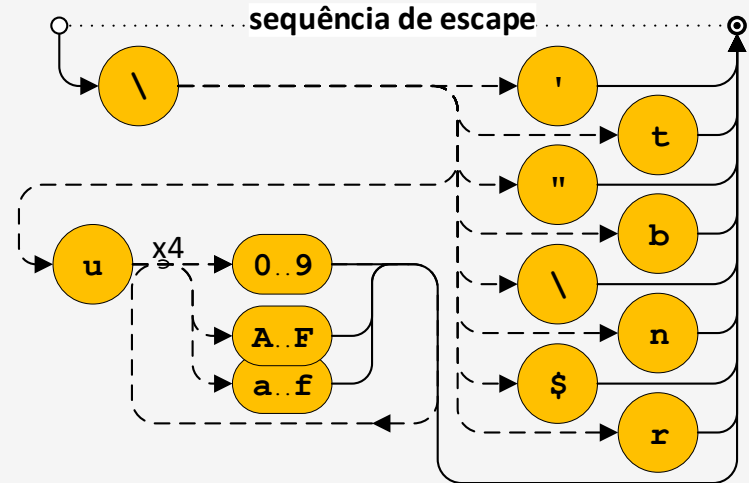
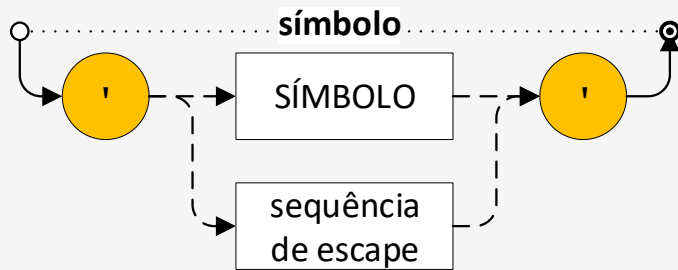
Símbolos



- Cada símbolo é codificado com 2 bytes usando a tabela
- Os 256 primeiros códigos são idênticos aos do ISO 8859-1 (ASCII estendido – Latin1)
- 'A'..'Z' → 65..89 → 0x41..0x59
- 'a'..'z' → 97..122 → 0x61..0x7A
- '0'..'9' → 48..57 → 0x30..0x39
- 'ç' → 231 → 0xE7
- 'ã' → 227 → 0xE3
- '€' → 8364 → 0x20AC
- ☎ → 9742 → 0x260E
- ☺ → 9786 → 0x263A

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-		0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{		}	~	
8-																
9-																
A-		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	®	¯	
B-	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C-	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D-	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E-	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F-	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Tipo Char



'K' \Rightarrow K :Char

'K'.toInt() \Rightarrow 75 :Int

48.toChar() \Rightarrow 0 :Char

'\\' \Rightarrow ' :Char

'\u20AC' \Rightarrow € :Char

'a' + 2 \Rightarrow c :Char

'c' - 'a' \Rightarrow 2 :Int

val letter :Char = 'f'

letter - ('a' - 'A') \Rightarrow F :Char

Texto

"Wello World!" \Rightarrow Wello World! :String

"122" \Rightarrow 122 :String

```
val empty = "" //string vazia
```

- Expressões embutidas (*template*)

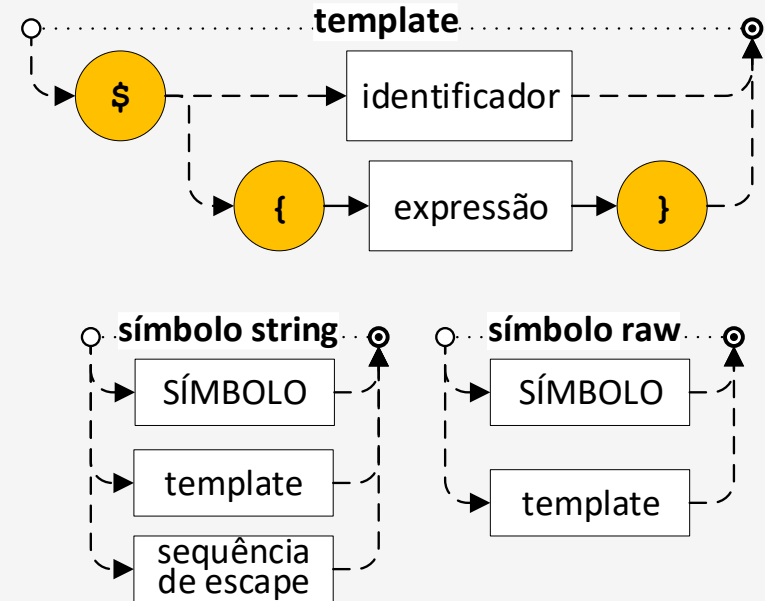
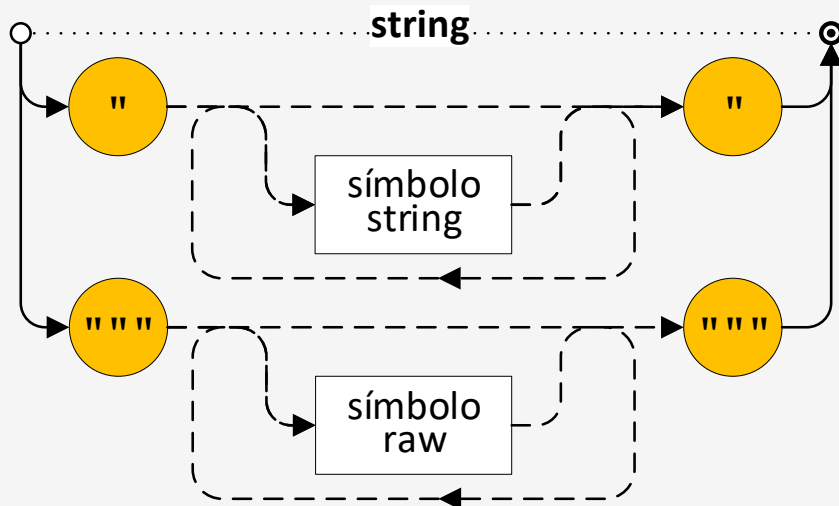
```
val name = "Pedro"
```

"Nome = \$name" \Rightarrow Nome = Pedro :String

```
val pi = 3.14
```

"PI x 2 = \${pi*2}" \Rightarrow PI x 2 = 6.28 :String

Tipo String



"\\$\"\'" ⇒ "\$"':String

""\"\$\"\'\\\" ⇒ \"\\$\"\'\\\":String

Operações com Texto

```
val str = "xpto"
```

'x'	'p'	't'	'o'
[0]	[1]	[2]	[3]

- Concatenação:

"abc" + str ⇒ abcxpto :String

str + 27 ⇒ xpto27 :String

- Indexação e comprimento :

str[0] ⇒ x :Char

str[2] ⇒ t :Char

str.length ⇒ 4 :Int

str[str.length-1] ⇒ o :Char

- Conversões:

"3.5e2".toFloat() ⇒ 350 :Float

"7A".toLong(16) ⇒ 122 :Long

9.toString(2) ⇒ 101 :String

27.toString() ⇒ 27 :String

Valores lógicos

```
val ok :Boolean = true
```

```
val x = 27
```

- Comparação de igualdade:

```
x == 10 ⇒ false :Boolean
```

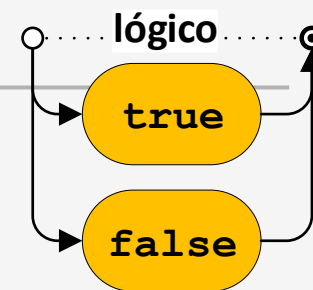
```
ok != true ⇒ false :Boolean
```

```
"abc" != "cba" ⇒ true :Boolean
```

- Comparação relativa:

```
x > 10 ⇒ true :Boolean
```

```
25.5 < x ⇒ true :Boolean
```



Oper	Descrição
==	Igual a
!=	Diferente de

Oper	Descrição
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Operações lógicas

```
val ok = true
```

```
val x = 27
```

```
val sym = 'S'
```

```
! ok ⇒ false : Boolean
```

```
x > 10 && ok ⇒ true : Boolean
```

```
ok || x == 10 ⇒ true : Boolean
```

```
sym >= 'A' && sym <= 'Z' ⇒ true : Boolean
```

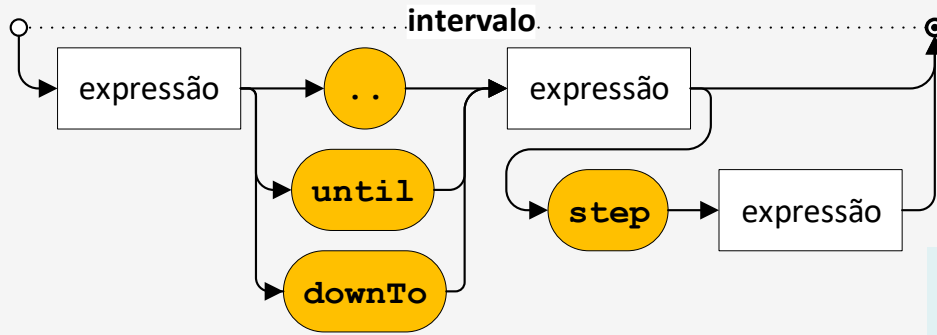
Oper	Descrição
&&	E (and)
	OU (or)
!	Negação (not)

A && B	false	true
false	false	false
true	false	true

A B	false	true
false	false	true
true	true	true

! A	
false	true
true	false

Intervalos de inteiros ou de símbolos (enumeráveis)



```
val grades = 0..20
```

```
grades.first => 0 :Int
```

```
15 in grades => true :Boolean
```

```
grades.last => 20 :Int
```

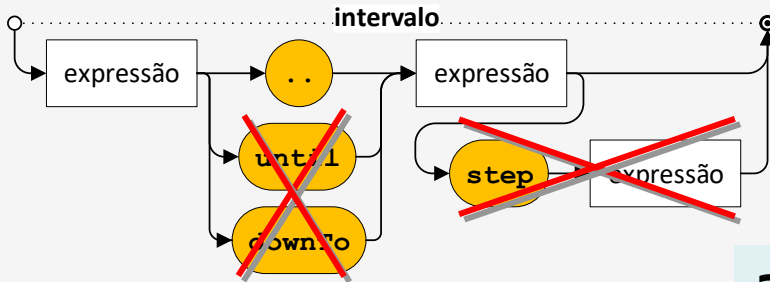
```
grades.count() => 21 :Int
```

```
'B' in 'a'..'z' => false :Boolean
```

```
1 until 8 == 1..7 => true :Boolean
```

```
(12 downTo 1 step 2).last => 2 :Int
```

Intervalos de valores comparáveis não enumeráveis



```
val values = 2.3 .. 4.2
```

3.5 in values ⇒ true : Boolean

values.start ⇒ 2.3 : Double

~~values.count()~~

```
val words = "FIM".."ISEL"
```

"GATO" in words ⇒ true : Boolean

"ABC" !in words ⇒ true : Boolean