Aspetos da linguagem Kotlin

Programação (PG)

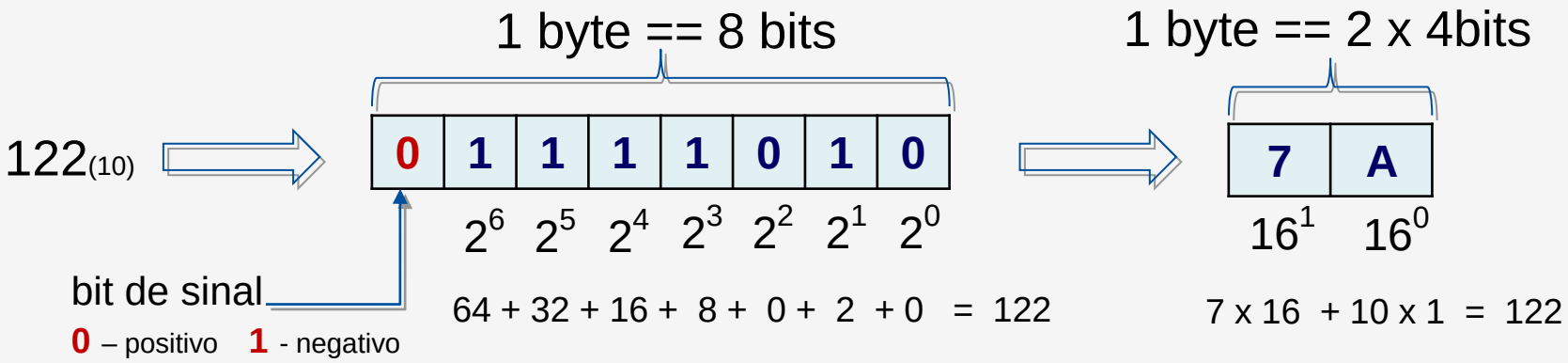
Centro de Cálculo
Instituto Superior de Engenharia de Lisboa

Pedro Alexandre Pereira (palex@cc.isel.ipl.pt)

Tipos de valores inteiros e sua representação

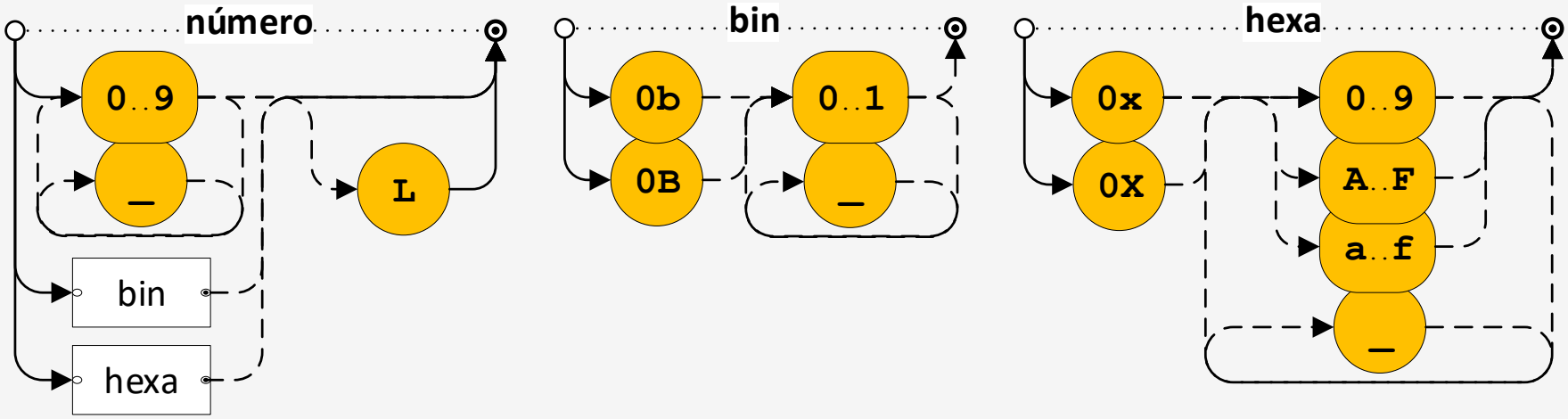
- Inteiros (com sinal)
  - **Byte**
  - **Short**
  - **Int**
  - **Long**

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

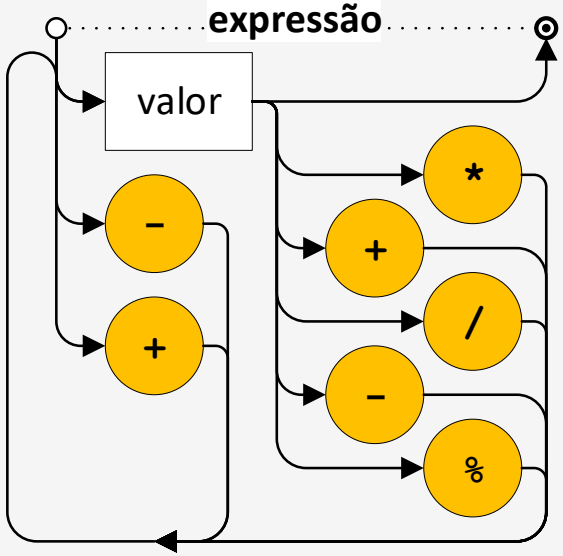


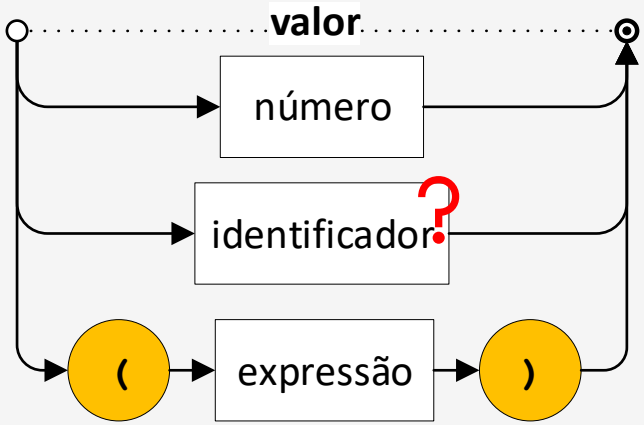

```
122                 :Int
148L                :Long
0b01111010          :Int
0x7A                :Int
1_430_800           :Int
0b0111_1010L        :Long
0x00FF00FF          :Int
122.toByte()        :Byte
400.toShort()       :Short
```

Operações aritméticas entre inteiros

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

```
27 + 3        ⇨ 30 :Int
27 - 3L       ⇨ 24 :Long
-2+2 * 7      ⇨ 12 :Int
9 / (4-2)     ⇨ 4  :Int
(3L+4) % 3    ⇨ 1  :Long
```

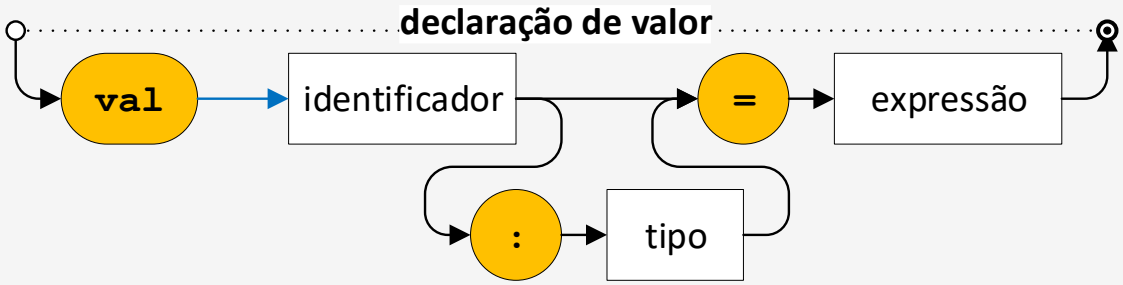


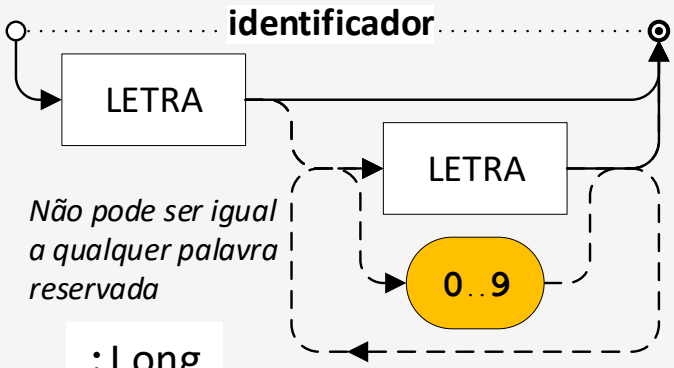


Resultado é **Long** se um dos argumentos for **Long**.

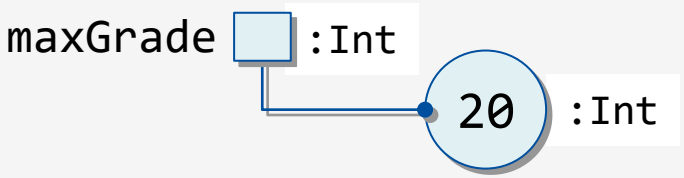
Caso contrário, o resultado é **Int**, mesmo quando todos os argumentos são **Short** ou **Byte**.

Declaração de valores

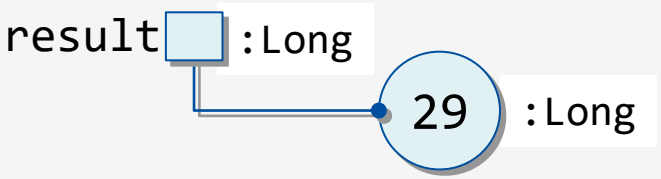





```
val maxGrade :Int = 20
```




```
          :Long
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

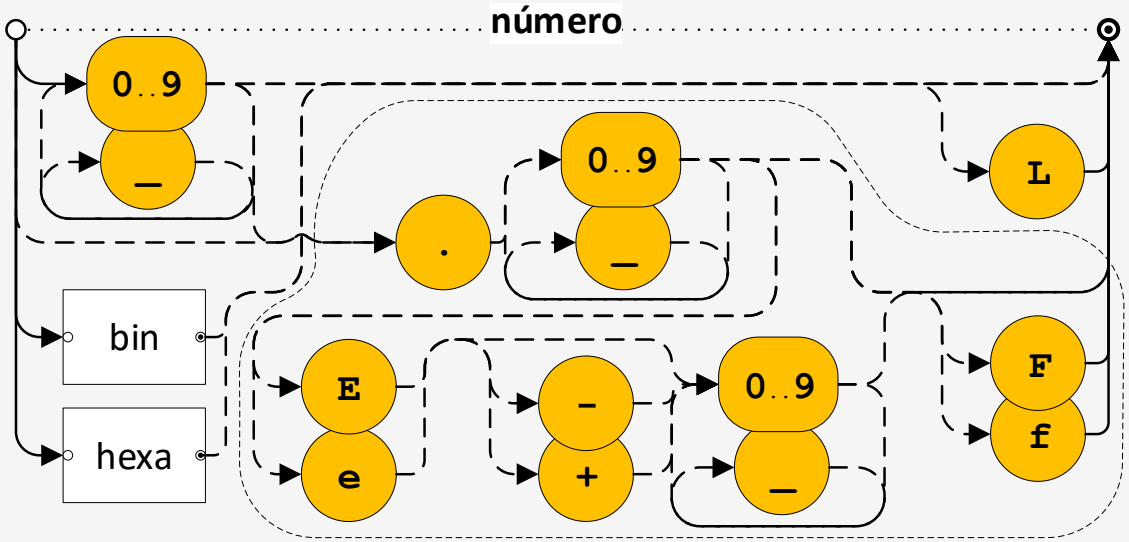
```
result / 10 ⇨ 2 :Long
```

Tipos de valores reais

- Reais (norma IEEE 754)
  - **`Float`**
  - **`Double`**

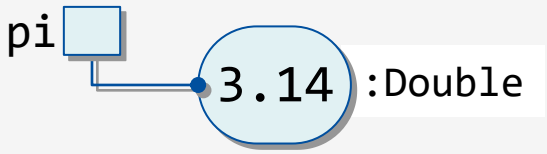
Dimensão	Precisão
4 bytes	6 a 7 dígitos
8 bytes	15 a 16 dígitos

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




```
:Double
32.54
:Float
5_200.0F
:Double
5.2e3
```

```
val pi = 3.14
```



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
```

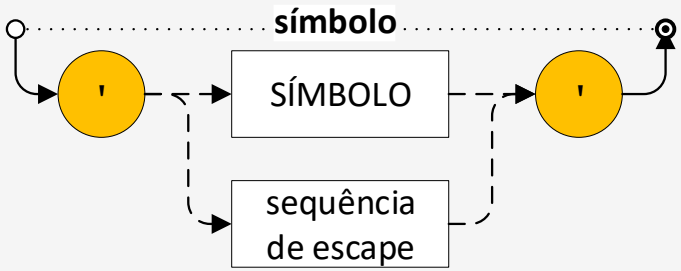
Símbolos

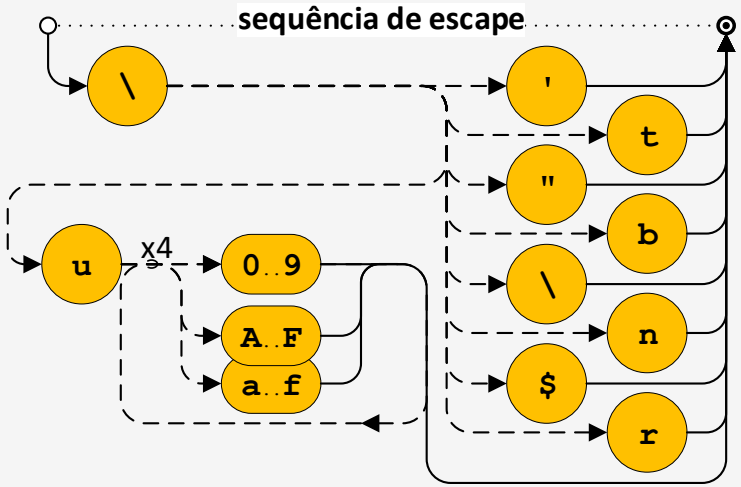
- Cada símbolo é codificado com 2 bytes usando a tabela UNICODE
- Os 256 primeiros códigos são idênticos aos do ISO 8859-1 (ASCII estendido – Latin1)
- 'A'..'Z' → 65..89 → 0x41..0x5A
- 'a'..'z' → 97..122 → 0x61..0x7A
- '0'..'9' → 48..57 → 0x30..0x39
- 'ç' → 231 → 0xE7
- 'ã' → 227 → 0xE3

- '€' → 8364 → 0x20AC
- '☎' → 9742 → 0x260E
- '☺' → 9786 → 0x263A

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	0000	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-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	007F
8-	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	008A	008B	008C	008D	008E	008F
9-	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	009A	009B	009C	009D	009E	009F
A-	00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B-	° 00B0	± 00B1	2 00B2	3 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	1 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C-	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D-	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E-	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F-	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF

Tipo **Char**






```
'K'  ⇨ K :Char
```

```
'K'.code  ⇨ 75 :Int
```

```
48.toChar()  ⇨ 0 :Char
```

```
'\''  ⇨ ' :Char
```

```
'\u20AC'  ⇨ € :Char
```

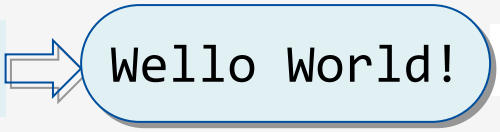
```
'a' + 2  ⇨ c :Char
```

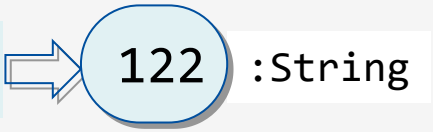
```
'c'-'a'  ⇨ 2 :Int
```

```
val letter :Char = 'f'
```

```
letter-('a'-'A')  ⇨ F :Char
```

Texto

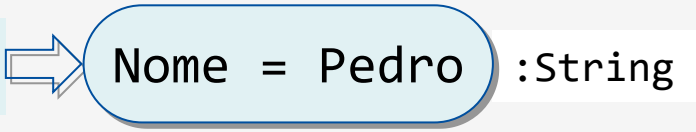
```
"Wello World!"
```



```
"122"
```


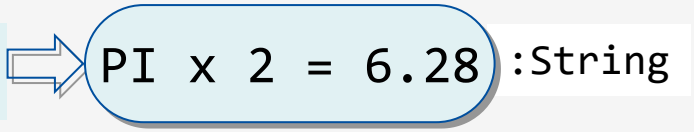

```
val empty = "" //string vazia
```

- Expressões embutidas (*template*)

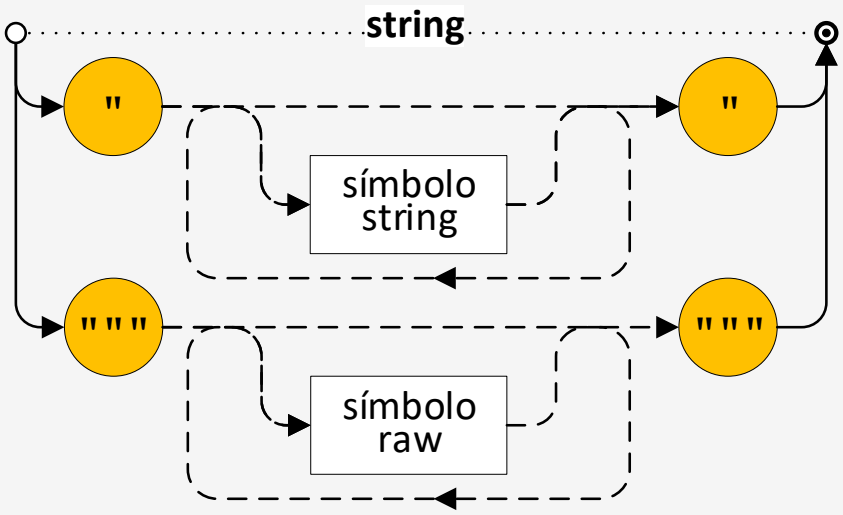
```
val name = "Pedro"
```

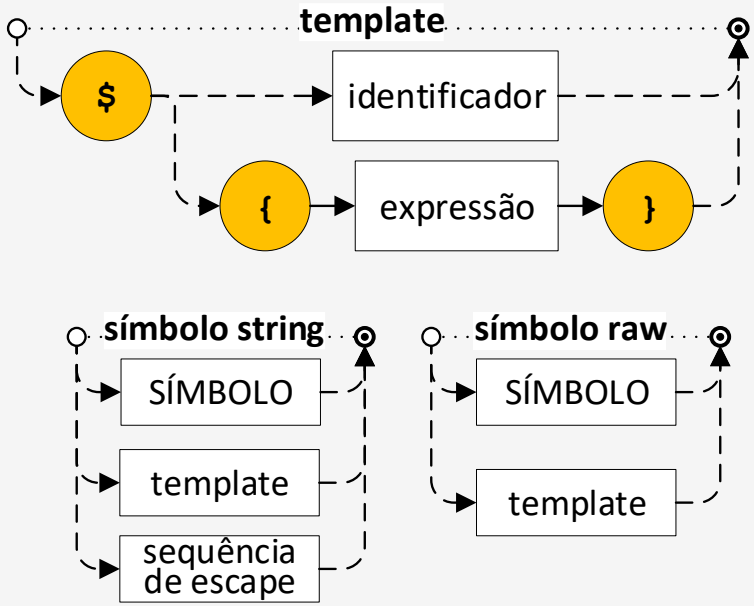
```
"Nome = $name"
```



```
val pi = 3.14
```

```
"PI x 2 = ${pi*2}"
```


Tipo `String`



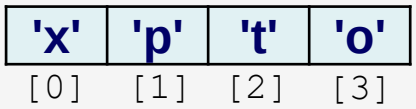



```
"\$\"\'"   ⇨  $"'   :String
```

```
"""\$\"\'\\"""   ⇨  \$\"\'\\   :String
```

Operações com Texto

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



- 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:

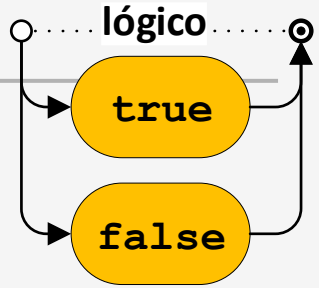
```
"7A".toLong(16)    ⇨ 122 :Long
```

```
"3.5e2".toFloat()  ⇨ 350 :Float
```

```
9.toString(2)      ⇨ 101 :String
```

```
27.toString()      ⇨ 27 :String
```

Valores lógicos




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

- Comparação de igualdade:

```
val x = 27
```

```
x == 10          false:Boolean
ok != true       false:Boolean
"abc"!="cba"     true:Boolean
```

Oper	Descrição
==	Igual a
!=	Diferente de

- Comparação relativa:

```
x > 10           true:Boolean
25.5 < x         true:Boolean
```

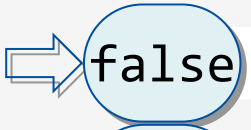
Oper	Descrição
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

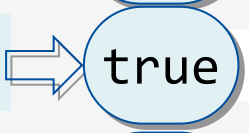
Operações lógicas

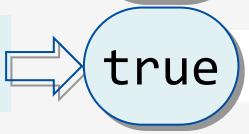
```
val ok = true
```

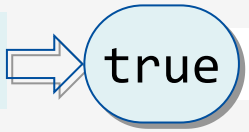
```
val x = 27
```

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

```
! ok
```



```
x>10 && ok
```



```
ok || x==10
```



```
sym>='A' && sym<='Z'
```


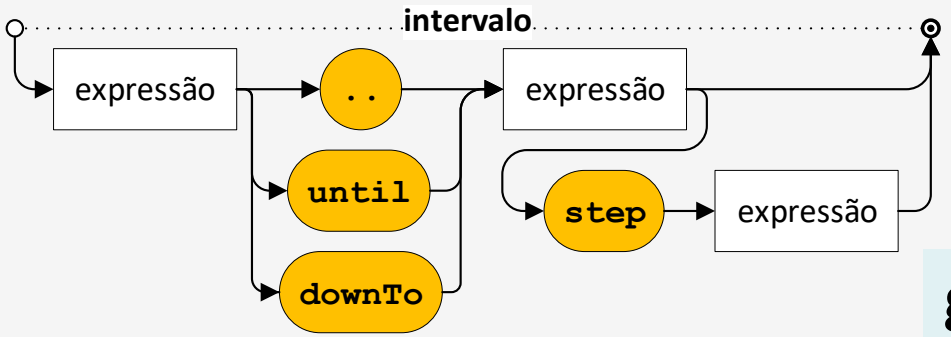
Oper	Descrição
&&	E (and)
||	OU (or)
!	Negação (not)

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

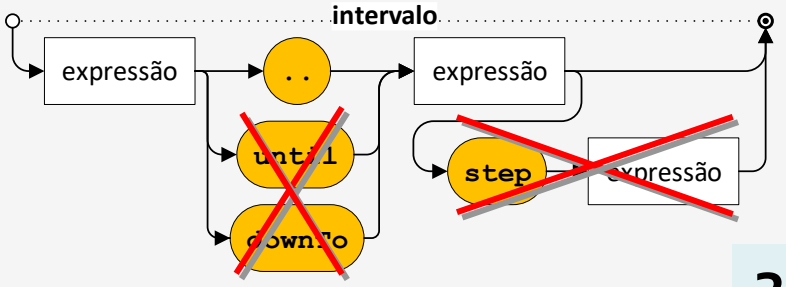
```
grades.count()
```
⇨ `21` :Int

```
'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