## Tipos

| Tipo | Bit | Byte | Casas Dec |
|---|---|---|---|
| **Byte** | 8 | 1 | - |
| **Short** | 16 | 2 | - |
| **Int** | 32 | 4 | - |
| **Long** | 64 | 8 | - |
| **Float** | 32 | 4 | 6-7 |
| **Double** | 64 | 8 | 15-16 |
| **Char** | 16 | 2 | - |
| **Boolean** | 8 | 1 | - |
| **String** | - | - | - |

**Operações:**
Inteiro com Real = Real
-Bytes com + Bytes = +Byte

## Operadores

| Tipo | Operador | Simbolo |
|---|---|---|
| **Lógico** | AND | && |
| | OR | \|\| |
| | NOT | \| |
| **Aritmético** | + ou - | + ou - |
| | x ou ÷ | *ou // |
| | Resto | % |
| **Compar.** | = ou ≠ | == ou != |
| | < ou > | < ou > |
| | ≤ ou ≥ | <= ou >= |
| **Random** | Coment | // /**/ |
| | Val na Str | $ |

**Operações com char**
Char+/-Int = Char (Int+Char -> erro)
Char-Char=Int
Char+Char->erro
'Int' o código é o próprioInt

**Data class** – tipo agregado
data class Person(val name:String, var age:Int)

**Enum Class** – tipo com certo nºde objetos possíveis
enum class Dir(val dx:Int, val dy:Int){UP(0,-1), LEFT(-1, 0),…}
Dir.Up – selecionar objeto UP
UP.name – nome do objeto UP
UP.ordinal – index do objeto UP
Dir.values() – lista de objetos de Dir

## Condições

```
if(condição) {code}
else if(condição) {code}
else {code}

when{
  condição -> expressão/instrução
  … -> …
}
```
(Expressão tem de ter else)

## Intervalos

1..16 step 2 -> 1 a 16 de 2 em 2
1 until 16 -> 1 a 16(exclusive)
16 downTo 1 -> 16 a 1
(intervalo).toList() – lista do intervalo
1 in 1..16 -> true; 1 está no intervalo
0 !in 1..16 -> true; 0 não está no intervalo
val name = intervalo -> definir intervalo em valor

## Null

:Tipo? -> pode ser do tipo, ou null
ball=ball?.função() -> se ball for null, retorna null; senão for, executa a função
val pos = ball?.pos ?: 10 -> Se ball for null, retorna 10; senão for, retorna pos
!! quando se tem a certeza que não é null

## Pair

val a:Pair<T,U> = … ->os dois tipos podem ser diferentes
a.first ->primeiro valor
a.second ->segundo valor

## Listas*

```
val nums:List<Int> =listOf(1,3,2,4)
nums[3] //4 ou nums.get(3) //4
nums.first() // 1
nums.last() //4
nums.indexOf(3) // 1
nums.indices //0..3
nums.size //4
nums.sorted() //[1, 2, 3, 4]
nums.shuffled() //baralha
nums.min() //1
nums.max() //4
nums.sum() //10
nums.reversed() //[4, 2, 3, 1]
nums.toString() //string: [1, 3, 2, 4]
nums.map{num-> instrução}
nums.filter{condição}
nums.forEach{instrução}
nums.any{condição}
nums.all{condição}
```

## Funções

```
fun TipoDeExtênção.name(
parâmetro:Tipo,…):TipoDeReturn{…}

b.apply{drawCircle(pos.x,…)}
```
->"extende" a o code a b, não se tem de meter b.pos.x

## Loops

while(condição){instrução} -> repete a instrução enquanto a condição é true; pode-se utilizar break ou return para parar o loop

for(i in range/lista){instrução} -> Para cada i no range ou na lista, executa a instrução

do {intrução} while(condição) -> a condição só é avaliada dps de executar a instrução; executa sempre uma vez

repeat(times){instrução} ->repete a instrução times(Int) vezes

## CanvasLib

```
fun main() {
  print("Primeiro")
  onStart {
    val arena=Canvas(wdth, hght , CYAN)
    print("Terceiro")}
  onFinish {
    print("Quarto, depois de clicar X")}
  print("Segundo")}
```

**RGB(Int):** 0xff0000 – cor vermelha arena.
drawCircle(x, y, radius, cor, thickness)
drawLine(xFro,yFro,xTo,yTo,cor,thic)
drawRect(x,y,wdht,hght,cor,thick)
drawArc(x,y,radius,stAng,endAng,cor, thickness)
drawText(x,y,txt,cor,fontSize)
onTimeProgress(period){code}
onMouseDown{me->code}
onMouseMove{me->code}
  MouseEvent tem x/y/down
onKeyPressed{ke->code}
  KeyEvent tem code/char/texto
playSound(String)
loadSounds(String)
close() -> fecha a janela
erase() -> limpa a janela

## Outros

**Input**: readLine() -> .trim para tirar espaços
**Valores/Variáveis:**
val/var name:Tipo = valorDoTipo
const val NOME = constante

**Listas Mutáveis e Arrays**

| Nome | Elem. | Dimensão |
|---|---|---|
| **List<T>** | Imutáveis | Fixa |
| **MutableList<T>** | Mutáveis | Variável |
| **Array<T>** | Mutáveis | Fixa |

```
val lst = mutableListOf(1,2,3)
lst.add(4) / /[1,2,3,4]
lst.remove(2) //[2,3]
lst.removeIf{condição}
```
*nums.toMutableList()
(Mutable->Imutable (automático); ao contrário é necessário funcão)
arrayOf(1,2,3)
Array(size){idx->intrução}
ex:Array(10){idx->idx+10} //10,11,19
Array.forEachIndexed{idx,elem->cod}

**Recursividade**
Colocar sempre condição de paragem
Para transformar para loops, criar variável que seria recursiva e o loop é do último stack possível para o primeiro

**Operator fun**
Criar:
fun doOper(opr:Char,fx:(Int,Int)->Int){
Como chamar: doOper('+', ::add)