Manual de Apoio ao Curso “Introdução ao Roblox”

2020

Índice

1. Conceitos básicos

1.1. Tipos de Dados

1.1.1. Strings

As **Strings** contêm texto. São colocados entre aspas duplas(“ ”) ou aspas simples (‘ ’). Eles também podem ser colocados entre [[e]] para que se estendam por várias linhas.

1.1.2. Tables

As **tabelas** são listas que podem ter “índices” com nomes. As tabelas são a combinação deArrays, Objetos ou Dicionários.

{"Meu array", "é bom"} – É como um objeto, mas os índices são números em ordem.
{["Meu objeto"] = "é bom"} - Posso ter índices com qualquer valor.

1.1.3. Booleans

Os booleanos são tipos de dados que podem ter os valores de verdadeiro ou falso. Os booleanos têm muito uso na programação.
Ao comparar valores em linguagens de programação, obtemos um booleano:

1 <3 - Um é menor que três, então é **verdade**
4 == 2 - Quatro não é igual a dois, então é**falso**
6> 7 - 6 é menor que 7, não maior que, então é**falso** novamente

1.1.4. Numbers

Os números podem ser armazenados de diferentes maneiras.
Floats: Podem armazenar números não inteiros de maneira eficiente.
Integer: Os inteiros são números inteiros.
Doubles: São outra forma de armazenar números semelhantes aos **floats**.

1.2. Variáveis e Funções

As variáveis são simplesmente uma maneira de armazenar um valor. Eles podem conter qualquer coisa e são um dos recursos mais importantes em todas as linguagens de programação. Estas podem ser **locais**, o que significa que podem ser usados onde são criados ou **globais**, o que significa que podem ser usados em qualquer lugar.

variable = "A minha string"
locallocalVariable = "A minha outra string"

do
print(abc) -Ainda não consigo ver o valor desta variável, pois não foi declarada.
local abc = 123 – Eu apenas existo dentrodostatemente posso ser usada apenas depois desta linha.
print(abc) –Consigo ver o valor da variável(123)
end
print(abc) –Aqui o valor da variável será null, isto é, significa que não tem valor.

As funções podem executar o mesmo código várias vezes.

local function myFunction(a, b, c)
return a, b, c
- a, b e c são como variáveis. São chamados de argumentos.

end

print(myFunction(1, 2, 3)) – Conseguimos ver os valores "1 2 3"

1.3. Instruções

As instruções constituem o código e são fundamentais para a construção da lógica na programação.
A instrução if executa o código, se uma condição for verdadeira. Para tal, faz uso dos tipos de dados booleanos.

```
if true then
print("Funcionou!")
end

if false then
print("Não vai imprimir porque a condição é falsa")
end
```

Na secção anterior foi referido o "null". Estefuncionacomofalso.Todos os valores além de false e null agem como verdadeiro (true).

```
if nil then
print("Nãovaiexecutar!")
end
if not nil then
print("Not nulléverdadeiro (true)")
end
if 2 then
print("Número!")
end
if {} then
print("Table!")
end
```

As instruções If também têm cláusulas else e elseif.
Basicamente funcionam da seguinte forma: Se algo acontecer (if) faz isto, caso contrário, se isso (elseif), faz isso, caso contrário (else) faz aquilo.

if(primeirasituação)**then**

- Executa a primeirasituação

elseif(segundasituação)**then**

- Executa a segundasituação

else

- Executa a terceirasituação

end

1.4. Loops

Os loops permitem que se selecione itens de uma matriz ou dicionário ou repita ações um determinado número de vezes. Também se podem repetir para sempre ou até que uma condição seja atendida.

- **While e loops repetidos**

while*condição***do**
    -- Açãoaexecutar
end

Estes loops repetem-se enquanto a condição for verdadeira.

repeat
    -- Ação a executar
until*condição*

Estes loops são como os whileloops mas eles repetem até a condição ser verdadeira.

whilenot*condição***do**
    -- Ação a executar
End

Este loop é exatamente igual ao anterior. Estes loops existem em duas formas diferentes e a sua escolhaé unicamente uma preferência pessoal.

- **For loops**

Estes são loops que contam o número de iterações. Podem ser usados para repetir uma ação num determinado número de vezes.

for i=1, 10 do
    -- Ação a executar
end

A variável "i"é local.Representa o índice atual. O nome a aplicar pode ser outro. Neste caso, a ação irá repetir-se 10 vezes, pois o "i" começa em 1, executa a ação, incrementa 1 (i = 2), executa a ação e assim sucessivamente até atingir o valor 10.

for i=1, 10, 2 do
    -- Ação a executar
end

Para estes loopsexiste um terceiro campo a ser considerado. Representa quanto é adicionado a cada loop. Ou seja, em vez de adicionarmos 1 em cada iteração, serão adicionados 2, o que significa que a ação a executar será feita 5 vezes.

2. Conceitos

2.1. Propriedades

As propriedades são como variáveis, mas existem numa instância. Por exemplo, as peças têm a propriedade *Anchored*, que pode ser verdadeira ou falsa.

2.2. Funções

As funções, na verdade, são como propriedades. Oferecem funcionalidade extra que as propriedades não podem.
Por exemplo:

localDestroy = part.Destroy – *Isto é uma função!*
Destroy() – *Mas porque é que não a podemos invocar? Porque é especial! Usa dois pontos (:). Mas o que é que os dois pontos fazem?*
Destroy(part) -*:dá à função um argumento extra.*

localtbl = {}
functiontbl:func()
    print(self) – *Vai imprimir o mesmo ID*
end

functiontbl.func2(self)
    print(self) – *Faz o mesmo que na função acima*
end

2.3. Eventos

Os eventos são semelhantes às instâncias.Estes têm duas funções: **Connect**e**Wait**.
Connect irá chamar uma função quando o evento é "disparado".
Por exemplo, as instâncias têm um evento ChildAdded que é disparado sempre que uma instância é adicionada a ela, também conhecida como "parent".

Waitirá aguardar até o evento ser disparado.

localmyEvent = Instance.new("BindableEvent")
myEvent.Event:Connect(print) -***Will****irá chamar a função****print****quando a dispararmos. O****Event****é um evento na instância****BindableEvent****. É muito semelhante a uma propriedade.*
myEvent:Fire("abc")

2.4. Instâncias

As instânciastêmfunções**FindFirstChild**, **WaitForChild** e **GetChildren**.

A **GetChildren** retorna uma matriz de instâncias que são os “filhos” da instância de destino.

A **FindFirstChild** procura uma instância com um determinado nome e retorna essa mesma instância. Se não existir, retorna nulo.

A **WaitForChild** é semelhante, mas espera que a instância exista.