GDScript 入门

可用的脚本语言

Godot 提供了四种游戏编程语言：GDScript、C# 以及通过 GDExtension 技术提供的 C 和 C++。还有更多社区支持的语言，但这四个是官方语言。

你可以在一个项目中使用多种语言。例如，在团队中，你可以在 GDScript 中编写游戏逻辑，因为它编写起来很快，并使用 C# 或 C++ 来实现复杂的算法并最大限度地提高其性能。或者你可以使用 GDScript 或 C# 编写所有内容。由你自己决定。

如果你是初学者，推荐从 GDScript 入手。GDScript的语法类似python，容易上手。
个人极度推荐学习UC Berkeley的CS61A课程，可以对编程语言有一个比较系统的了解。

标识符

标识符就是我们编程的时候使用的“名字“，给类、接口、**方法（函数）、变量、常量名**，包名等起名字的字符序列

任何**仅限于字母字符（a 到 z 和 A 到 Z）、 数字（0 到 9）和下划线 _ 的字符串**都可以作为标识符。此外，标识符**不能以数字开头，并且区分大小写**（foo 和 FOO 是两个不同的标识符）。

标识符现在也允许包含 UAX#31 所提供的部分 Unicode 字符，也就是说，你现在也可以将非英文字符作为标识符使用。然而， Unicode 字符中易与 ASCII 字符混淆的字符以及颜文字则是无法作为标识符使用的。
(可以使用汉字或其他合法字符命名，但是不推荐)

关键字

关键字是语言的保留字，不能用作标识符。例如 `if` ， `for` 等。
后面的内容会再提及，这里就提一下

类型与变量

类型一般分为两种，内置类型和引用类型。这里的“基本内置类型”即基本类型，除基本类型以外的所有类型都是引用类型。在函数传参时这两种类型会表现出不同的行为。

基本内置类型

null

`null` 是一个空数据类型，既不包含任何信息，也不能赋值为其他任何值。

bool

“boolean”（布尔）的缩写，只能是 `true` 或 `false` 。

int

英文“integer”（整数）的缩写，存储整数（正整数和负整数）。

float

使用浮点值存储实数，包括小数。存储的是 64 位值，等效于 C++ 中的 double。

String

Unicode 格式的字符串。

```
var string1 = "Hello, " #双引号代表这是一个字符串
var string2 = 'World!'  #单引号也可以
var result = string1 + string2
print(result)  # 输出: Hello, World!
```

在gdscript中，因为存在类型自动推断，所以我们不必记住各种类型的名称（？）。

变量创建

变量用 `var` 关键字创建，可以在初始化的时候赋一个值

```
var a # 初始化（创建）a变量，a的初始值默认是null
var b = 5 # 初始化的同时赋值
var c = 3.8
var d = b + c
```

常量

常量常用于一些在程序中一些不变的值，比如重力加速度，比如点按键人物速度的增加量，如果直接写数字，则在工程中若人物速度需要修改则需要花费很大精力，一般设为一个常量

```
const var gravity = 9.8
const var velocity = 100
```

内置向量类型

Vector2

2D 向量类型，包含 `x` 和 `y` 两个字段

```
var a = Vector2(1.1, 2.3) #创建一个二维向量并存入a变量
a.x = 1 #将a向量的x值改为1
```

Vector2i

同 Vector2，但其分量均为整型数值，在做2D网格中显示物品时非常实用。

容器内置类型

数组(Array)

任意对象类型的泛型序列

```
var arr = []
arr = [1, 2, 3]
var b = arr[1] # This is 2.
var c = arr[arr.size() - 1] # This is 3.
var d = arr[-1] # Same as the previous line, but shorter.
arr[0] = "Hi!" # Replacing value 1 with "Hi!".
arr.append(4) # Array is now ["Hi!", 2, 3, 4].
arr.pop_back() # Array is now ["Hi!", 2, 3]
```

字典(Dictionary)

键值对的集合，冒号前为键(key)，冒号后为值(value)

```
var d = {4: 5, "A key": "A value", 28: [1, 2, 3]}
d["Hi!"] = 0 #如果键中有"Hi!"，则把其对应的值改为0；如果没有则增加一对键值对
d = {
    22: "value",
    "some_key": 2,
    "other_key": [2, 3, 4],
    "more_key": "Hello"
}
```

信号(Signal)

略，在后面详细讲

基本类型和引用类型的区别

```
var b = 1
var a = b
b = b + 1
print(b) # 2
print(a) # 1
# 以上为基本类型

var c = []
var d = c
d.append(3)
print(c) # [3]
```

以上为引用类型

指定类型

```
var a: Array[int]
var b: int = 5
```

常用运算符

运算符

函数

允许复用的一段代码。

```
func my_function(a, b):
    return a + 2b
```

调用函数

```
var x = my_function(1,2)
print(x) # 5
```

若函数体只含一行语句，则可以将函数及其函数体缩在同一行语句内编写：

```
func square(a): return a * a
```

可以在参数列表之后使用箭头标记（ -> ）来指定函数的返回值类型：

```
func my_int_function() -> int:
    return 0
```

函数的作用域

全局变量和局部变量

基本类型

```
var a = 5
var b = 5

func my_func(a,b):
    a = 6
    return a + b

print(a) # 5
```

不要这样写代码!
在这里，a和b相当于是被复制进去了

引用类型

```
var a = [1,2,3]

func my_func(b):
    b.push_back(4)
    return b

var c = my_func(a)
print(c)
```

引用函数

就 Callable 对象而言，函数是其第一类对象。通过名称引用函数而不调用它，会自动生成指向该函数的 Callable。这可用于将函数作为参数传递。

```
func map(arr: Array, function: Callable) -> Array:
    var result = []
    for item in arr:
        result.push_back(function.call(item))
    return result
```

可调用体**必须**使用 `call` 方法进行调用。你不能直接使用 () 运算符。

语句与流程控制

表达式和语句

表达式是运算符和操作数的有序排列

```
2 + 2 # Binary operation.
-5 # Unary operation.
x # Identifier representing variable or constant.
x[4] # Subscript access.
x > 2 or x < 5 # Comparisons and logic operators.
x == y + 2 # Equality test.
do_something() # Function call.
[1, 2, 3] # Array definition.
{A = 1, B = 2} # Dictionary definition.
preload("res://icon.png") # Preload builtin function.
```

有效的表达式被写到程序里就成为语句。

条件句

条件句通过使用 `if` / `else` / `elif` 语法创建。条件中的括号可写可不写。

```
if (expression):
    statement(s)
elif (expression):
    statement(s)
else:
    statement(s)
```

这里的expression一般是bool类型，一般是表判断含义的表达式

```
if (a > 2):
    a += 2
elif (a < = -2):
    a -= 2
else:
    pass
```

短的语句可以与条件句写在同一行内

```
if 1 + 1 == 2: return 2 + 2
else:
    var x = 3 + 3
    return x
```

三元表达式

类似于C语言中 `a ? b : c` 的语法

```
var x = (value1) if (expression) else (value2)# 如果expression是true，那表达式的值为value1，否则为
y += 3 if y < 10 else -1
```

可以写的更长

```
var fruit = (
        "apple" if count == 2
        else "pear" if count == 1
        else "banana" if count == 0
        else "orange"
)
```

但是可读性会降低所以不建议这么写

in

想要检查某个值是否包含在某些容器之中时，可以通过 if 语句与 in 操作符组合来实现：

```
var text = "abc"
if 'a' in text:
    print(There is an 'a' in the text.)

var arr = ['apple','banana','peach','shxt']
if 'shxt' in arr:
    arr.erase('shxt') # remove 'shxt' from the array
```

(array官方文档)[./https://docs.godotengine.org/zh-cn/4.x/classes/class_array.html#class-array-method-erase]

循环句

while

一般的循环通过 while 语法创建，可以使用 break 来跳出整个循环，或者使用 continue 来跳出当前批循环并进入下一轮的循环当中（会在该轮循环将该关键字下方所有在该循环体内的语句全部跳过）：

```
while (expression):
    statements

while (i > 1 and i < 9):
    i += 1
```

for

要遍历一个容器，例如数组或表，使用 `for` 循环。遍历字典时，键被存储在循环变量中。

```
for x in [5, 7, 11]:
    statement # Loop iterates 3 times with 'x' as 5, then 7 and finally 11.
var dict = {"a": 0, "b": 1, "c": 2}
for i in dict:
    print(dict[i]) # Prints 0, then 1, then 2.

# range 左闭右开

for i in range(3):
    statement # Similar to [0, 1, 2] but does not allocate an array.
for i in range(1, 3):
    statement # Similar to [1, 2] but does not allocate an array.
for i in range(2, 8, 2):
    statement # Similar to [2, 4, 6] but does not allocate an array.
for i in range(8, 2, -2):
    statement # Similar to [8, 6, 4] but does not allocate an array.
for c in "Hello":
    print(c) # Iterate through all characters in a String, print every letter on new line.
for i in 3:
    statement # Similar to range(3).
for i in 2.2:
    statement # Similar to range(ceil(2.2)).
```

match

类似于在许多其他语言中出现的 `switch` 语句，可以称为大号的 `if` 语句

```
match x:
    1:
        print("It's one!")
    2:
        print("It's one times two!")
    _:
        print("It's not 1 or 2. I don't care to be honest.")
```

如果x=1，则控制台会打印"It's one!"，然后不再执行之后的语句。
_代表通配符，也就是说如果上面的都不匹配，就执行_下的语句