بسم الله الرحمن الرحیم

در قسمت قبلی ، با مفاهیمی همچون « گره » ، « صحنه » و « نمونه سازی صحنه ها به عنوان گره » در Godot را یاد گرفتید . در این آموزش می خواهیم خیلی سریع شما را با GDScript آشنا کنم .

مقدمه

GDScript ، یک زبان برنامه نویسی اسکریپت نویسی دینامیک (dynamicly typed scripting language) است ، که سازندگان Godot ، به خاطر ساختار خاص این Game Engine (استفاده از مفهوم « گره » و « صحنه ») آنرا اختصاصا برای این Game Engine توسعه داده اند .

اصلی ترین اهدف از طراحی GDScript :

- ✓ یادگیری و استفاده از آن ساده باشد .
- ✓ کُدها خوانا و ایمن باشند . (Syntax این زبان ، با زبان Python مشابهت های زیادی دارد .)
- ✓ برای کار با گره ها ، مناسب باشد .

درست مثل سایر زبان های اسکریپت نویسی دینامیک ، کارایی را با سرعت بالای کُدنویسی و آسانی بالانس شده است ؛ ولی اکثر قسمت های حیاتی این موتور بازی سازی به زبان C++ نوشته شده اند (vector ، physics ، math ، render ، indexing و ....)

اگر در هر بازی ، کارایی و سرعت بیشتری نیاز باشد ، می توان قسمت های حیاتی را توسط زبان C++ نوشت و آنها را در اسکریپت مورد استفاده قرار داد . این قابلیت به ما اجازه می دهد که کلاس های GDScript را با کلاس های C++ بدون بر هم زدن سایر قسمت ها ، جایگزین کرد . (البته نگران نباشید ، معمولا به چنین رویکردی نیازی نیست .)

توجه : در این قسمت فرض شده است که شما آشنایی قبلی با برنامه نویسی دارید ، هدف ما در این آموزش معرفی سینتکس GDScript است .

GDScript

یک پروژه به نام GDScript بسازید و بعد گره های زیر را به شکلی که می بینید به آن اضافه کنید .

- Node2D
  - Panel

و صحنه را با نام GDScript.scn ذخیره کنید ، سپس یک اسکریپت به نام GDScript.gd به Node2D اضافه کنید .

اولین برنامه

GDScript.gd را باز کنید ، و آن را این گونه تغییر دهید .

```
extends Node2D
func _ready():
	print("Hello World")
```

تشریح کُد بالا :

- extends Node2D : این کُد به معنای ارث بری از کلاس Node2D است که باعث می شود که بتوانیم از توابع درونی آن استفاده کند .
- func _ready(): این ، یک تابع درونی Godot Engine است و به محض اجرای این « گره » ، این تابع فراخوانی می گردد .

godot.blog.ir

- ()print : یک متن را در کنسول یا بخش output ویراشگر Godot ، چاپ می کند (از این قابلیت می توان برای درک روند اجرای برنامه ها و خطایابی استفاده کرد .)

GDScript ، برای مشخص کردن بلاک کُد ها از تو رفتگی که توسط زدن کلید tab استفاده می کند و اصطلاحا به آن tab-based indetion scripting language گفته می شود . برای مثال ، در کُد بالا ، دستور ()print جزئی از بلاک کُد تابع ()_ready است چون با تورفتگی ، در محدوده ی بلاک کُد تابع ()_ready قرار گرفته است .

متغیرها

متغیرها یا شناسه ها (variable / identifers) ، مکانی در حافظه هستند که نام ، نوع ، مقدار و آدرس داشته باشند . متغیرها را با حرف کلیدی var تعریف می کنیم ، به خاطر ساختار دینامیک GDScript ، نوع و آدرس این متغیرها در اولین مقدار دهی آن ها تعیین می شود .

برای تعریف یک متغیرها از این سینتکس استفاده می کنم :

var [نام متغیر] = [مقدار]

مثال هایی از تعریف متغرها :

```
var a = 5
var characterName = “Jhon”
var array = [1,2,3]
var dictionary ={ “key” :”value” , 2:3 }
```

نام گذاری گره ها از قاعده ی زیر پیروی کند :

1. نام متغیرها نمی تواند با اعداد شروع شود . (مثال : اسم 01Label غیر مجاز است)
2. نام متغیرها نمی تواند شامل کاراکترهای غیر انگلیسی و کاراکتر های خاص باشد (یعنی نمی توان از حروف فارسی و کاراکترهای خاصی مثل &^%$#@!<> و ... در نامگذاری گره ها/متغیرها ، استفاده کرد .)
3. نام متغیرها حساس به بزرگی و کوچکی حرف است (Label برابر label نیست .)
4. در نامگذاری متغیرها می توان از علامت _ استفاده کرد .
5. نام متغیرها باید با حروف لاتین (انگلیسی) نوشته شود . (یعنی نمی توانید از حروف فارسی استفاده کنید .)
6. نمی توان از کلمات کلیدی به عنوان نام متغیرها (کلاس ها ، توابع و ...) استفاده کرد .

به خاطر ساختار دینامیک GDScript و استفاده از ماشین مجازی (مثل هر زبان اسکریپتی دیگری) ، نوع متغیرها در هر بار مقدار دهی به آن ها دوباره تعیین می شود . برای درک این مطلب ، این تغییرات را درون GDScript.gd اعمال کنید .

```
extends Node2D

func _ready():

	var myName # null by default
	myName = "Reza"
	print(myName)
	myName = 6
	print(myName)
```

خروجی چاپ شده در Output بدین صورت خواهد بود .

```
Reza
6
```

در کُد بالا می بینید که نوشته ی « null by default » را در جلوی دستور var myName نوشتیم . در مثال بالا یک نکته ی مهم گفته شده است ، ***متغیر های مقدار دهی نشده به صورت پیش فرض مقدار null (پوچ) دارند .***

godotblog.ir

مستند سازی

نوشته های مستند سازی نوشته هایی هستند که برای نوشتن توضیحات اضافه کنار کُد به کار می روند . در علامت برای مستند سازی وجود دارد :

: GDScript هر چیزی که پس از کاراکتر هشتک # بیاید ، تا آخر خط به عنوان مستند سازی در نظر می گیرد و آن ها را نادیده می گیرد .

```
# This is a sngle line Comment
var myName # null by default
```

""" : هر چیزی که بین این علامت ها نوشته شود ، به عنوان نوشته های مستند سازی در نظر گرفته شده و در نتیجه توسط GDScript نادیده انگاشته می شوند . به صورت ساده تر ، از این علامت برای مستند سازی در چند خطی استفاده می شود .

```
""" This is Multi-Line
Comments . Do you like
This format . """
```

ثابت ها

متغیرها ی ثابت ، متغیر هایی هستند که پس از مقدار دهی دیگر نمی توان مقدار آنها را تغییر داد . متغیر ها با مقدار ثابت را با کلمه ی کلیدی const تعریف می کنیم . ثابت ها از همان قوانین نامگذاری متغیرها پیروی می کنند ولی به صورت یک عرف ، معمولا نام متغیرهای ثابت را با حروف بزرگ مشخص می شود .

```
const ANSWER_TRUE = true
const ANSWER_FALSE = false
const ANSWER_TRUE = "YES" # Error !!! you can't change value of constant varrable
```

کلمات کلید GDScript

کلمات کلیدی یک زبان برنامه نویسی ، کلماتی از پیش تعیین شده اند که معنای خاصی برای آن زبان برنامه نویسی دارند و عملکرد آن زبان برنامه نویسی وابسته به این کلمات است . جی دی اسکریپت کلمات کلیدی کمی دارد که یادگیری آن ها آسان است . در اینجا لیستی از کلمات کلیدی GDScript را می بینید .

Var	for	switch	continue	Extends
Const	do	case	pass	Tool
If	while	break	return	Signal
Else	static	tool	func	Export
Elif	setget			Break-point

همانطورکه گفته شد نمی توان از این کلمات کلیدی به عنوان نام متغیرها یا ثابت ها استفاده کرد .

عملگرها

عملگرها ، عملی را انجام می دهند ، مانند علامت + که عمل جمع کردن را انجام می دهد . در اینجا لیستی از عملگرهای GDScript را می بینید که به ترتیب اولویت لیست شده اند (توجه : بالاترین عملگرها ، بیشترین اولیت را دارند . اولیت عملگرهای سمت چپ در یک خط ، بیشتر است.)

godot.blog.ir

X[index]	(بالاترین اولیت) Subscription
x.attribute	ارجاعی به خصیصه
Extends	تست کننده ی ارث بری
~	Not بیتی
-x	عدد منفی
* / %	باقیمانده / تقسیم / ضرب
+ -	منها / به علاوه
<< >>	شیفت بیتی
&	And بیتی
^	XOR بیتی
		OR بیتی
< > == != >= <=	عملگرهای مقایسه ای
In	عملگر تست کننده ی محتوا
! not	نقیض منطقی
And &&	و منطقی
Or ||	یا منطقی
= += -= *= /= %= &= |=	عملگرهای انتساب (کمترین اولویت)

انواع درونی GDScript

اگر یادتان باشد ، هر متغیر باید یک نوع داشته باشد . هر چند می توان با استفاده از کلمه ی کلیدی var ، تعیین نوع متغیر را به انواع درونی ، را به ماشین مجازی GDScript سپرد ولی به هر حال باید بدانید که در پس زمینه ، GDScript ، نوع متغییر ها را از نوع یکی از انواع درونی GDScript تعیین می کند .

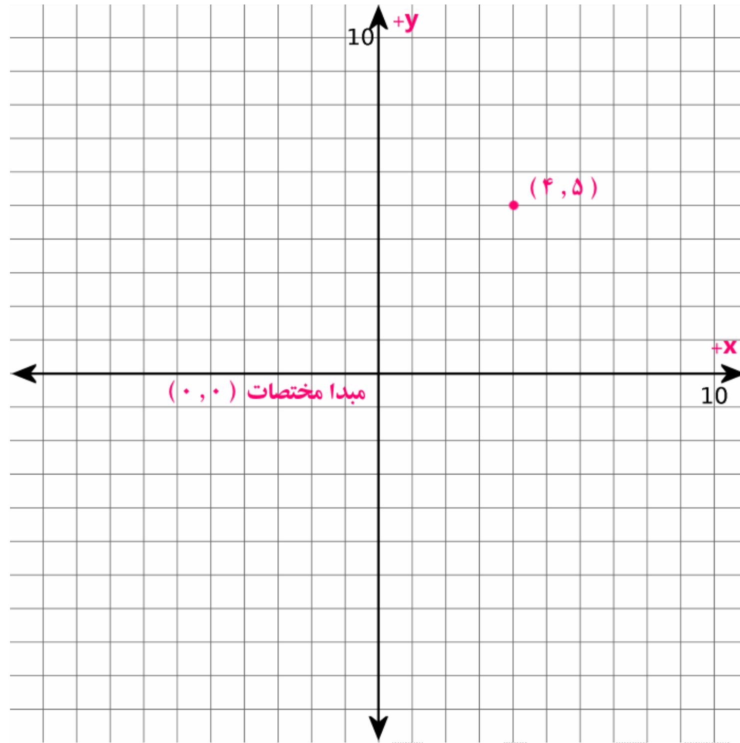
انواع درونی – انواع عادی

در اینجا لیستی از انواع درونی عادی GDScript را می بینید .

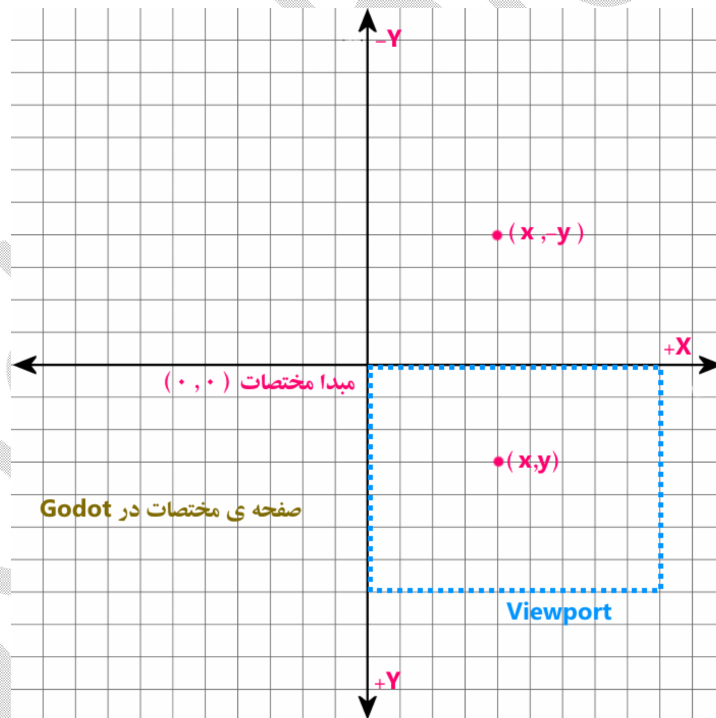
Null	پوچ ، یک نوع داده ای خاص است که هیچ مقداری ندارد و نمی توان آن را به سایر مقدارها وصل کرد
Bool	نوع بولی ، فقط می تواند مقدار true یا false برگرداند
Int	عدد صحیح که شامل مقادیر مثبت ، منفی و صفر است
Float	عدد ممیز شناور یا همان اعداد اعشار
String	رشته ، رشته ای از کاراکترها

انواع درونی – انواع بُرداری (Vector built-in type)

انواع بُرداری برای مواردی همچون تعیین موقعیت یک گره در صحنه ، تعیین اندازه ی آن ، تعیین شکل آن و ... استفاده می شود . می توان گفت در ساخت بازی های دو بعدی از این انواع استفاده های بسیار زیادی می شود . قبل از ادامه ی بحث بهتر است که با موقعیت Viewport درون دنیای Godot آشنا شوید . به تصویر زیر نگاه کنید .



در ریاضیات ، صفحه ی مختصات به شکل بالاست ولی در Godot ، صفحه ی مختصات به شکل زیر است .



همانطور که می بینید ، در Godot صفحه ی مختصات با حالت عادی کمی فرق دارد و محور Y ها برعکس است . علاوه بر این ، viewport در قسمت مثبت محور x ها و y ها است . ***viewport همان صفحه ی نمایش است .***

: vector2

بُرداری دو بعدی ، دو مولفه ی X و y را در خود جا می دهد . می تواند به فیلدهای width و height هم برای خوانایی دست پیدا کند . علاوه برا این می تواند به یک آرایه نیز دسترسی پیدا کند .

به صورت ساده تر ، Vector2 ، دارای یک مقدار دو مولفه ای است که مولفه ی اول آن ، برابر X و مولفه ی دوم آن برابر y است .

به GDScript بروید . و قطعه ی کُد آن را بدین صورت عوض کنید :

```
func _ready():
var panelPos = get_node("Panel").get_pos()
print(panelPos)
```

قطعه کُد بالا ، ابتدا با استفاده از دستور ("[node path]")get_node نام گره مورد نظر را می گیرد و سپس با دستور get_pos() موقعیت این گره - در اینجا گره Button که فرزند گره Panel – است را می گیرد و درون متغیر panelPos می گذارد . موقعیت این نقطه به صورت یک vector2 است و خروجی که چاپ می شود برابر چیزی مانند : 30,280 است .

می توانید اندازه ی هر گره را بگیرید که آن هم به صورت Vector2 است . قطعه کُد زیر را به به اسکریپت اضافه کنید :

```
var panelRectangle = get_node("Panel").get_rect().size
print("Panel Rectangle size : " , panelRectangle )
```

تابع get_rect() ، مستطیل اطراف گره را می گیرد و با استفاده از property یا ویژگی size ، اندازه ی این مستطیل را به صورت vector2 بر می گرداند .

rect2:

نوع مستطیل دو بُعدی دارای 2 فیلد pos (موقعیت) و size (اندازه) است . به جای آن ها می توانید یک فیلد end که شامل موقعیت و اندازه است (pos+size) را داشته باشد .

```
Rect2(x,y,w,h) # x,y are position and w,h are size of rect
```

می توان به جای x و y ؛ و همچنین w و h از Vector2 استفاده کرد . (به مثال زیر توجه کنید– این مثال انتزاعی است)

```
var rectNewPos = vector2(50,30)
var rectNewSize = vector2(20,100)
Rect2(rectNewPos,rectNewSize)
```

vector3:

یک بُردار سه بعدی است که شامل فیلدهای x ، y و z است . می توان به این نوع به عنوان یک آرایه دسترسی پیدا کرد .

matrix32:

ماتریکس 2*3 است که برای انتقال (transform) دو بُعدی به کار می رود .

plan:

نوع سطوح سه بعدی است که در حالت نرمالیزه شده ، دارای یک فیلد بُرداری نرمال و یک d به عنوان فاصلی اسکالر هستند .

godot.blog.ir

:Quat

Quaternion یک نوع داده است که نشان دهنده ی چرخش سه بعدی است .

:AABB

Axis Aligned Bounding Box یا 3D Box ، حاوی دو فیلد pos و size است . علاوه بر این ، یک فیلد جایگزین به نام end دارد که مقدار آن برابر pos+size می باشد . می توان نام مستعار rect3 را به جای این نوع استفاده کرد .

:Matrix3

ماتریس 3*3 که برای چرخش (rotaton) و تغییر اندازه (scale) به کار می رود . این ماتریس 3 فیلد بُرداری x, y, z را داردو می توان به این ماتریس به عنوان یک آرایه از بُردارهای سه بعدی دسترسی داشت .

:Transform

انتقال سه بعدی که شامل یک فیلد basis از نوع matrix3 است و یک فیلد origin از نوع vector3 می باشد .

انواع درونی Engine

: Color

نوع داده ای رنگ که شامل فیلدهای r,g,b,a است . علاوه بر این ، می توان به این نوع داده ای به عنوان h s v (hue/saturation/value) دسترسی داشت .

: Image

حاوی یک فرمت سفارشی از تصاویر دو بعدی است که اجازه ی دسترسی مستقیم Godot را به پیکسل ها می دهد .

:NodePath

یک مسیر کامپایل شده به یک « گره» که در Scene System استفاده شده ، بر می گرداند. می توان این نوع را به یک رشته وصل کرد یا مقدار آن را از یک رشته گرفت .

```
var getPanel = get_node("Panel")
```

:(Resource ID)RID

سرورها از RID های generic برای ارجای دادن به داده های گنگ به کار گرفته می شود .

:Object

کلاس های پایه ای برای هر چیزی که از انواع درونی نیست .

:InputEvent

رویدادهایی که از دستگاه ی ورودی صادر می شوند ، در یک فرم بسیار فشرده از اشیاء ورودی (InputEvent Objects) قرار داده شده اند . به علت اینکه می توانند از هر فریم به فریم دیگری مقدارهای ورودی زیادی داشته باشند ، به صورت یک نوع داده ای جداگانه بهینه شده اند .

godot.blog.ir

انواع درونی Container :

:Array

نوع پایه ای از مجموعه مرتب از اشیاء مختلف ، این نوع می تواند آرایه های دیگر و یا نوع دیکشنری ها را در خود جا می دهد . در GDScript ، آرایه ها به صورت اتوماتیک تعیین اندازه می شوند . آرایه ها از اندیس 0 شروع می شوند .

```
var array = []

array = [1,2,3]

var b  = array[1] #  مقدار 2 در آن قرار میگیرد

array[0] = “Hi” #  Hi به جای 1 در آرایه قرار می گیرد

array.append(4) #  اضافه کردن یک مقدار به آرایه
```

آرایه های GDScrip به صورت خطی درون حافظه قرار می گیرند تا سرعت بالاتر برود . آرایه های بسیار بزرگ (بیش از هزاران المان) می توانند باعث تکه تکه شدن حافظه بشوند . در این حالت می توان از آرایه های خاص که بدین منظور آماده شده اند ، استفاده کرد . البته آرایه های خاص فقط یک نوع داده ای را قبول می کنند . این آرایه ها ، حافظه ی کمتری را اشغال می کنند ولی به علت ذات اتمکی خود ، از آرایه های generic ، کندتر اجرا می شوند ؛ برای همین ما توصیه می کنیم که از این آرایه ها فقط برای مجموعه داده های بسیار بزرگ استفاده کنید .

ByteArray	آرایه ای از بایت ها (مقدار 0 تا 255 می گیرد)
IntArray	آرایه ای از اعداد صحیح
FloatArray	آرایه ای از اعداد ممیز شناور (اعداد حقیقی یا اعداد اعشار دار)
Vector2Array	یک آرایه از اشیاء Vector2
Vector3Array	یک آرایه از اشیاء Vector3
ColorArray	آرایه ای از اشیاء Color
StringArray	آرایه از رشته ها

:Dictionary

مجموعه ای از مقادیر که توسط کلیدهایشان مشخص می شوند . (دوتایی کلید ، مقدار) . این نوع داده ای برای نگه داشتن مشخصات بازی و Save بسیار کاربُرد دارد .

```
var d={4:5, "a key":"a value", 28:[1,2,3]}

d[“Hi”] = 0

var d={

	22 : ”value” ,

	“somekey” : 2 ,
```

godot.blog.ir

```
        "otherkey "  : [2,3,4] ,
        "morekey " : "Hello"
        }
```

سینتکس به زبان Lua نیز پشتیبانی می شود . در این حالت به جای : از = استفاده می شود و نیازی نیست از علامت نقل قول برای مشخص کردن کلید استفاده شود (در حالتی که کلید از نوع رشته باشد .) با این حال ، کلید باید از قوانین نامگذاری متغیرها پیروی کند .

```
# Lua Style Dictionary
var d={
        22 = "value" ,
        somekey = 2 ,
        otherkey  = [2,3,4] ,
        morekey = "Hello"
        }
```

برای اضافه کردن یک کلید به دیکشنری ، به آن مثل کلید که وجود دارد ؛ دسترسی پیدا کنید و مقداری را به آن اضافه کنید .

```
var d = {}
d.waiting= 14 # add key "waiting" and value "14" to dictionary
d[4]= "Hello" # the is 4 and  the value is "Hello"
d["Godot"] = 3.0 # Godot is the key and 3.0 is the value
```

دستورات کنترل جریان :

دستورات کنترل جریان ، دستوراتی هستند که جریان برنامه را کنترل می کنند ،

If/elif/else

شرط های بسیار ساده را می توان با کلمات کلیدی if/else/elif ساخت . بهتر است شروط را درون پرانتز کرد تا هم برنامه خواناتر شود و هم از خطاهای احتمالی و گرامری جلوگیری شود . به خاطر طبیعت « تو رفته ی » کُد های GDScript ، می توان از elif به جای else if استفاده کرد تا ساختار تو رفته ی کُدها حفظ گردد .

```
If ( شرط ها ) :
        دستور(ها)
elif ( شرط ها ):
        دستور(ها)
else:
        دستور(ها)
```


godot.blog.ir

مثال انتزاعی زیر را در نظر بگیرید . فرض کنید می خواهیم در آخر یک مرحله به یک کاراکتر بر اساس میزان جان باقی مانده اش سکه جایزه بدهیم ، در این حالت می توانیم بدین شکل عمل کنیم :

```
export(int) var mission_health_gift = 10
If (HealthPercent == 100 ):
	mission_health_gift *= 150
	Player.coins += mission_health_gift
elif (HealthPercent >=85 && HealthPercent < 100):
	mission_health_gift *= 110
	Player.coins += mission_health_gift
elif (HealthPercent >=55 && HealthPercent < 85):
	mission_health_gift *= 90
	Player.coins += mission_health_gift
elif (HealthPercent >=25 && HealthPercent < 55):
	mission_health_gift *= 50
	Player.coins += mission_health_gift
else (HealthPercent >=1 && HealthPercent < 25):
	mission_health_gift *= 25
	Player.coins += mission_health_gift
```

می توان کُد بالا درتابعی قرار داد که در پایان مرحله با بررسی میزان جان باقی مانده ی بازیکن ، سکه هایی را به خاصیت coins بازیکن اضافه کند .

While

حلقه ی While، برای تکرار دستورات تا زمانی که شرط درون حلقه درست باشد . باید توجه داشت برای خروج از حلقه باید درون آن دستوری قرار داد که در ضمن اجرای حلقه ، باعث false شدن شرط درون حلقه شد و یا از شروط خاص استفاده شود . می توان از کلمات کلیدی continue و break درون حلقه استفاده کرد . continue برای پریدن به اجرای بعدی حلقه و break برای خروج از حلقه به کار می روند . به کُد زیر دقت کنید :

```
while(experssion):
	Statement(s)
```

For

برای مرور کردن یک محدوده (range) مشخص از داده هایی مثل آرایه ، دیکشنری ، جداول و ... از حلقه ی for استفاده می گردد .

```
for x in range():
	Statement(s)
```

godot.blog.ir

مثال های زیر را در GDSCript بنویسید و سعی کنید نحوه ی کار این کُدها را درک کنید :

```
for x in [1,2,3,4,5,6,8,5,7]:
	print(x)  # output will be 1 2 3 4 5 6 8 5 7
var dict =  { a = 0 , b = 1 , c=2}
for  i in dict:
	print(dict[i]) # کلیدها را بر پایه ی یک تعریف من در آوردی بر می گرداند ، خروجی می تواند 0 1 2 باشد یا 0 2 1 و یا ...
for i in range(3):
	print (i)
for i in range ( 1, 40 , 4 ):
	print (i)
```

توابع

توابع به تکه کُدهایی گفته می شوند که معمولا برای انجام یک کار خاص نوشته می شوند .برای تعریف تابع از کلمه ی کلیدی func استفاده می گردد .

```
func [name_of_function](function parameters):
	# body of function
```

توابع در GDScript طبیعت دینامیک دارند و نوع آرگومان های آن در هنگام اجرا مشخص می گردند (البته ، در صورت فرستادن انواع داده ای مختلف ، امکان خطا وجود دارد)

مثال تعیین جایزه ی بازیکن بر اساس جان بازیکن را در نظر بگیرید ، می توانیم آن را درون تابع زیر تعریف کنیم.

```
func calucate_player_gifts(health):
	return health *2
```

می بینید در این مثال از منطقی ساده تر و واضح تر استفاده کردیم .

حالا می توانیم از این تابع استفاده کنیم .

```
Player.coins= calculate_player_gifts(health)
```

در GDScript ، نمی توان توابع را به عنوان آرگومان به توابع دیگر ارسال کرد .

برای ارجاع دادن یک تابع در حالت اجرا (مثلا برای ذخیره ی آن در یک متغیر یا فرستادن آن به تابع دیگری به عنوان یک آرگوان) باید از helper های call یا funcref زیر استفاده کرد .

godot.blog.ir

فراخوانی تابع با استفاده از نام آن در یک مرحله

```
mynode.call("myFunction",args)
```

ذخیره تابع در یک متغیر یا ارجاع

```
var myFunc = funcref(mynode , "myFunction")
```

فراخوانی تابع ذخیره شده در متغیر

```
myFunc.call_func(args)
```

به یاد داشته باشید که توابع پیش فرض مثل _init ، و بیشتر اعلان ها مثل _enter_tree ، _exit_tree ، _process ، _fixed_process و ... به صورت خودکا ردر کلاس پایه فراخوانی می گردند . بنابراین فقط در زمانی overload کردن آن ها (به هر صورتی) نیاز داریم که به صورت صریح آن ها را فراخوانی کنیم .

با استفاده از کلمه ی کلیدی Static ، می توانیم یک تابع را به صورت ایستا تعریف کنیم .وقتی یک تابع به صورت ایستا تعریف شده باشد ، دیگر هیچ دسترسی به « متغیر های عضو نمونه » یا selft ندارد . توابع ایستا بیشتر برای ساخت کتابخانه های از توابع helper کاربرد دارند .

```
static func sum(a,b):
	return a+b
```

کلاس ها

به صورت پیش فرض ، بدنه ی یک اسکریپت یک کلاس بدون است و می توان به صورت خارجی به عنوان یک resource یا file به آن ارجاع داد . مثلا در اینجا کلاسی به نام GDScript.gd خواهیم داشت .

سینتکس کلاس باید خیلی فشرده باشد و فقط می تواند اعضای کلاس و توابع را در خود داشته باشد . می توان در کلاس ها ، توابع ایستا داشت ولی نمی توان اعضای استاتیک داشت   و دلیل این امن به خاطر thread saftey است ، چون Script ها را می توان بدون اطلاع برنامه نویس در چند thread جداگانه راه بیاندازد . به همین صورت ، متغیرهای عضو (از جمله آرایه ها و دیکشنری ها) در هر باری که یک کلاس نمونه سازی می شود ؛ دوباره مقداردهی و راه انداخته میشود . در زیر مثالی از یک class file می بینید .

```
# saved as file named myClass.gd
var 5
Func print_value _of_a():
	Print(a)
```

ارث بری / Inheritance :

یک کلاس می تواند از موارد زیر ارث بری داشته باشد :

- ✓ یک Global Class
- ✓ از یک کلاس فایل دیگر (که به عنوان یک فایل با پسوند gd . ذخیره شده است)
- ✓ از یک کلاس درونی که داخل یک کلاس فایل دیگر است .

باید توجه داشته باشید که در GDScript ، ارث بری چندگانه مجاز نیست .

برای مشخص کردن ارث بری از کلمه ی کلیدی extends استفاده می کنیم ، به مثال های زیر توجه کنید

godot.blog.ir

```
# ارث بری از یک کلاس جهانی
extends Node2D

# ارث بری از یک کلاس فایل دیگر
extends "AnotherClassFile.gd"

# ارث بری از یک کلاس درونی که داخل کلاس فایل دیگر است
extends "SomeClassFile.gd".InnerClass
```

کلمه ی کلید Extends علاوه بر کاربرد اصلی خود که برای مشخص کردن ارث بری است ، کاربُرد ثانویه ای دارد . برای اینکه چک کنیم که آیا یک نمونه (شی) ، از یک کلاس خاص ارث بری دارد یا نه ؟ می توان از کلمه ی کلیدی extends به عنوان یک عملگر شرطی استفاده کرد . به مثال زیر تجه کنید :

```
const enemy_class = preload("enemy.gd")
if ( entity extends enemy_class):
        entity.apply_damage();
```

در کُد بالا ابتدا کلاس Enemy را با دست preload ، در ثابتی به نام enemy_class ، پیش بارگذاری کردیم . سپس در شرط گفتیم که اگر شی entity از کلاس Enemy ارث بری داشته باشه ، متد Apply_damage را برای شی entity اجرا کن و به شی Entity مقداری صدمه اعمال کن .

سازنده ها / Constructors :

تابع درونی ()_init به عنوان سازنده ی کلاس عمل می کند . وقتی از یک کلسا ارث بری می شود ، سازنده های کلاس های والد به صورت خودکار فراخوانی می گردند ، بنابریان ، به صورت معمول دلیلی برای فراخوانی ()_init ، نیست .

اگر سازنده های کلاس های والد ، آرگومان دریافت کنند ، می توان بدین گونه عمل کرد .

```
Func _init(آرگومان های کلاس فعلی).(آرگومان های کلاس والد):
        # pass
```

کلاس های درونی

کلاس های درونی ، با استفاده از کلمه ی کلیدی Class تعریف می گردند . این کلاس ها با استفاده از تابع ()new نمونه سازی می شوند .

```
# تعریف یک کلاس درونی داخل یک کلاس فایل دیگر
```

godot.blog.ir

```
class MyMath:
	var num1 = 2
	var num2 = 4
	func printSum():
		print(str("Sum is : " , num1+num2))
```

این سازنده ی کلاس فایل اصلی است

```
Func _init():
	var c = MyMath.new()
	c.printSum
```

godot.blog.ir

:setters / getters

اگر نخواهیم که سایر کلاس ها و قسمت های مختلف برنامه به صورت مستقیم به یک تابع دسترسی داشته باشند ، می توانید از کلمه ی کلیدی setget برای ساختن توابع setters و getters استفاده کنیم . در ساده ترین حالت ممکن تابع setter ، مقداری را درون یک متغیر قرار می دهد و تابع getter مقدار یک متغیر را بر می گرداند . دلیل استفاده از setter و getter ، مدیریت دسترسی و مقدار دهی به یک متغیر یا فیلد کلاس است .

برای تعریف setter و getter از این سینتکس استفاده می کنیم .

var نام متغیر = [مقدار] setget تابع سِتر , تابع گِتر

مثال :

```
var text = setget setText , getText

func setText( txt ) :
	text = txt
func getText():
	return text
```

بارگذاری منابع :

برای بارگذاری منابع (گره ها ، صحنه ها ، کلاس ها و ...) از توابع ی load و preload استفاده می کنیم .

- ✓ Load : هر بار که یک منبع درخواست داده شد ، آن را بارگذاری می کند .
- ✓ Preload : فقط یک باردر زمان کامپایل و اصطلاحا اجرای بازی منبع مورد نظر را بارگذاری می کند .

باید توجه داشته باشید که Godot ، هر منبع را فقط و فقط یکبار در حافظه بارگذاری می کند و نه بیشتر ، یعنی اگر از دستور های load یا preload ، در چند جای برنامه یک منبع را بارگذاری کرده باشید ، Godot فقط یکبار آن را در حافظه قرار می دهد ودر سایر موارد از همان مقادیر درون حافظه استفاده می کند .

```
var myClass = load("MyClass.gd")
var nextScene = preload(str("Level",n))
```

پایان

این یک مقدمه ی کوتاه به GDScript است . حتی ساده ترین زبان های برنامه نویسی ، پیچیدگی های خاص خود را دارند که یادگیری آنها مستلزم کار کردن عملی و برنامه نویسی با آن زبان برنامه نویسی بود ، برای همین من بیش از این وارد جزئیات نمی شوم و همین جا این آموزش را به پایان می برم . هدفم از این آموزش فقط آشنا کردن شما نسبت به سینتکس GDScript بود ، نه اینکه یک کتاب مرجع در مورد زیر و بم این زبان اسکریپتی که فقط در موتور بازی سازی Godot کاربُرد دارد ، بنویسم . در ادامه و در حین اجرای پروژه ها با جنبه های کاربُردی این زبان آشنا می شوید .

اعتقاد دارم که یادگیری یک زبان برنامه نویسی در حین اجرای پروژه و به صورت عملی راحت تر و موثر تر است .

نوشته شده در تاریخ 1395/03/27
نسخه ی مورد استفاده Godot 2.0.2
نویسنده : رضا پویا

godot.blog.ir

godot.blog.ir