Catto’s Programming Knowledge Base

By housecatdraws

Contents

[KNOWLEDGE PARAMETERS: 1](#_Toc178626295)

[Learn GDscript from zero – gdquest 2](#_Toc178626296)

[Lesson 1 - What Code is Like: 2](#_Toc178626297)

[Lesson 2 – Errors are Essential: 2](#_Toc178626298)

[Lesson 3 – We Stand on the Shoulders of Giants: 2](#_Toc178626299)

[Lesson 4 – Drawing a Rectangle: 4](#_Toc178626300)

[Lesson 5 – Coding your first Function: 4](#_Toc178626301)

[Lesson 6 – Coding your first Function Parameter: 5](#_Toc178626302)

[2D NODE DEFININTIONS: 5](#_Toc178626303)

[FOUNDATIONAL 2D NODES: 5](#_Toc178626304)

[Node2D: 5](#_Toc178626305)

[Camera2D: 5](#_Toc178626306)

[Sprite2D: 5](#_Toc178626307)

[AnimatedSprite2D: 5](#_Toc178626308)

[CollisionObject2D: 6](#_Toc178626309)

[PhysicsObject2D: 7](#_Toc178626310)

[StaticObject2D: 7](#_Toc178626311)

[CollisionShape2D: 7](#_Toc178626312)

[CollissionPolygon2D: 7](#_Toc178626313)

[StaticBody2D: 7](#_Toc178626314)

[AnimatableBody2D: 7](#_Toc178626315)

[PHYSICS NODES 2D: 7](#_Toc178626316)

[RigidBody2D: 7](#_Toc178626317)

[CharacterBody2D: 7](#_Toc178626318)

[Joint2D: 8](#_Toc178626319)

[Dampen Spring Joint: 8](#_Toc178626320)

[GroveJoint2D: 8](#_Toc178626321)

[PinJoint2D: 8](#_Toc178626322)

[UI NODES 2D: 8](#_Toc178626323)

[Area2D: 8](#_Toc178626324)

[Area2D: 9](#_Toc178626325)

# KNOWLEDGE PARAMETERS:

Current software in use:

Godot.4.2.2

Godot.4.3.2

Type of code covered:

GDscript ( a mix of C#, python)

# Learn GDscript from zero – gdquest

### Lesson 1 - What Code is Like:

Coding is the process of writing instructions for a program to follow. The type of code I am using is GDScript.

### Lesson 2 – Errors are Essential:

Errors (although infuriating) are a core aspect of coding in video games. Remember, it’s better to receive an error message than nothing at all!

GD quest has a built in error definition tool.

### Lesson 3 – We Stand on the Shoulders of Giants:

All code is built upon a well of knowledge made by coders before you. GDscript has “built in” code, which is a series of signals and short-cuts made by the coders that build Godot.

A bundle of code created is referred to as a “Library”. Much like a library, it’s free to use\* (unless it’s for a trademarked mechanic).

Remember: Godot is made by coders for coders! It has an expansive collection of libraries if you need it!

#### Functions (“Func”):

A function is a list of instructions with an exact name to reference and call in the rest of the code.

#### Calling Functions (“Func”):

When “calling” a function you’re telling the computer to “execute the function”.

Remember to write the name of the function exactly!

Built in function = “func run() :”

Hide()

= hides the sprite.

#### Functions (“Func”) arguments:

We use parentheses to call functions because you can give the functions arguments inside those parentheses.

Built in function = “func run() :”

rotate(0.3)

= Rotates the sprite by 0.3 radians.

#### What are radians?

Radians are a measurement of rotation used in video games and maths. You can convert them into degrees by using the formula

Degrees = radians \* 180 / PI

An angle of PI radians corresponds to 180 degrees. And 2 \* PI is a full turn: 360 degrees.

#### How do radians work?

A blue background with white text

Description automatically generated

Move\_local\_x/y:

These are two built in functions that move the sprite in pixel increments, without the influence of integers or delta, leading the sprite to seemly “teleport”.

Built in function = “func run() :”

Move\_local\_x(20)  
Move\_local\_x(20)

= moves the sprite diagonally by 20 px when executed.

### Lesson 4 – Drawing a Rectangle:

This lesson is focused on learning how to draw a rectangle using code.

This is active practice of the previous lessons where we learned how to move a sprite and rotate a sprite.

Note: some functions featured in these lessons are made specific for this tutorial, and do not apply to GDScript in Godot.

When doing several lines of code under a function, the indentation stays as 1 consistent indent, unless there is a specific function referenced in the function, defined externally.

### Lesson 5 – Coding your first Function:

Functions are sequences of instructions

When we give the sequence a name, that name is referred to as an “identifier”.

An identifier is needed to call the function as many times as we need. An example of this would be:

*Func move\_and\_rotate():*  
move\_forward(200)  
turn\_forward(90)

#### What is an Identifier?

The “Identifier” of this specific function is “move\_and\_rotate()”, which would be referenced along as the function definition (as seen the above example), is included in the code.

To compound the initially defined function, you can include the Identifier in a new function definition—But the name must be different!

#### How to define your own function?

*Func name(parameters):*  
identifier(argument)

The above example is referred to as a “code block”, using the indent

The section beneath the function line is referred to as the “indent”, as it’s tabbed--indented.

#### Names in code have rules

* Identifiers cannot have spaces, so instead you can separate them with an “\_” underscore.
* You can use Capitalisation instead of underscores, but you only capitalise the second word.
* Identifiers can’t start with a number, but they can include a number.

### Lesson 6 – Coding your first Function Parameter:

The previous lessons focused on fixed variable functions.

In this lesson, the function will apply more widely. Specifically, we will be replacing the argument with parameters instead.

What is a Parameter?:

# 2D NODE DEFININTIONS:

## FOUNDATIONAL 2D NODES:

### Node2D:

* The basis for all 2D nodes. It keeps track of position, rotation, and scale.

All 2D nodes inherit from this node.

### Camera2D:

* The viewport of the game. Has rotation and position parameters.

Additional integrated features include Camera Drag, and Camera smoothing.

### Sprite2D:

* Used to render characters to a scene. It has of position and rotation parameters.

Can be used for interactable objects as well; like collectables, etc

### AnimatedSprite2D:

* Used to render sprite sheets to a scene. It has of position and rotation parameters.

Additional can be used for Characters, NPCs or Mobs.

### CollisionObject2D:

* Is not a standalone node. Is a subsection of the Collision2D node. Has the utility of separating collide-able sprites/animated sprites/ tiles/ etc into “Layers” and “Masks”.

Remember you need to understand Layers and Masks, to make sure they don’t cause a bug in animation.

### PhysicsObject2D:

* Is not a standalone node. Is a base class for the game 2D objects.

All physical bodies inherit (Inherit meaning: gain a quality) from it. PhysicsObject2D and StatitcObject2D are both classed as a “Physics” Node.

### StaticObject2D:

* Is not a standalone node. Is a base class for the game 2D objects.

Only Collision qualities inherited, as it is a stationary node. PhysicsObject2D and StatitcObject2D are both classed as a “Physics Node”. Can be used for walls, floors, ceilings.

### CollisionShape2D:

* Bounding box for the interactable area for the sprite it’s parented to.

Must be a child of a Physics Node.

### CollissionPolygon2D:

* Is not a standalone node. Is a base class for the game 2D objects.

No qualities are inherited, as it is a stationary node.

### StaticBody2D:

* Is a physics body that cannot be moved. Used for walls, floors, ceilings.

No qualities are inherited, as it is a stationary node.

### AnimatableBody2D:

* Is not a standalone node. Is a base class for the game 2D objects.

No qualities are inherited (Unaffected by gravity), as its animated properties are coded.

## PHYSICS NODES 2D:

### RigidBody2D:

* Is not a standalone node. Is affected by physics & gravity. Used for all physics body nodes.

Qualities are inherited from physics and gravity; however, it is still a static object. Interactable by player i.e.: pushing.

### CharacterBody2D:

* Is a parent node but requires CollisionShape2D and Sprite / AnimatedSprite2D nodes to make it fully functional.

CharacterBody2D inherits qualities from Velocity and Gravity. Has built in “Movement Code Template” for precise input and functionality. NOTE: The template is made to be edited & extended, as the default precision is unforgivingly precise and has no coded buffers included like “CoyoteTime” or “JumpCutting” / “JumpBuffering”.

### Joint2D:

* Is not a standalone node. Is a base class for all Physics Joints.

No qualities are inherited, joins two Physics Bodies together.

### Dampen Spring Joint:

* Is not a standalone node. This is a physics node that tethers a physics bodies together. This node has parameters for “Length”, “Stiffness”, and “Dampness”.

Physics qualities are inherited, you need to specify which object affects which in the “Joint2D” menu, in Node A and Node B menus.

### GroveJoint2D:

* Is not a standalone node. This is a physics node that tethers two physics bodies together. This node has parameters for “Length”, “Stiffness”, and “Dampness”.

Physics qualities are inherited, you need to link a rigidbody2D node to the GroveJoint2D. To prevent swinging motion from momentum, specify the RigidBody2D node as “Character”

### PinJoint2D:

* Is not a standalone node. This is a physics node that tethers two physics bodies together, RigidBody2D & PinJoint2D. Parameters for “rotation” and “location”.

Physics qualities are inherited, you need to link a rigidbody2D node to the PinJoint2D.

## UI NODES 2D:

### Area2D:

* Is a standalone node. Has 2 main functions;
  1. To act as a detection field for collision objects; including player and interactable objects.
  2. A space with it’s own physics and audio settings.

No qualities are inherited; however, you can set your own physics within this field. I.e.: Underwater areas, Windy areas.

### Area2D:

* Is a standalone node. Has 2 main functions;
  1. To act as a detection field for collision objects; including player and interactable objects.
  2. A space with it’s own physics and audio settings.