GDScript Programming

Complete Language Reference

What is GDScript?

- Python-inspired scripting language
- Built specifically for Godot Engine
- Dynamically typed with optional static typing
- Indentation-based syntax
- High performance (JIT compiled in Godot 4)
- Tightly integrated with Godot's node system

Why GDScript?

- Easy to learn
- Fast iteration
- Excellent IDE support
- Optimized for game development

Basic Syntax

```
# This is a comment

# Variable declaration
var health = 100
var player_name = "Hero"
var is_alive = true

# Constants
const MAX_SPEED = 500
const GRAVITY = 9.8

# Multiple assignment
var x, y = 10, 20

# Type inference
var items = []  # Array
var stats = {}  # Dictionary
```

Indentation matters!

- Use tabs or 4 spaces (tabs recommended)
- Consistent indentation required

Data Types - Primitives

Type	Example	Description
`int`	`42`, `-10`, `0xFF`	Integer numbers
`float`	`3.14`, `-0.5`, `1e6`	Floating-point
`bool`	`true`, `false`	Boolean values
`String`	`"Hello"`, `'World'`	Text strings
`StringName`	`&"node_name"`	Optimized strings
`NodePath`	`^"Path/To/Node"`	Node references

```
# Type examples
var count: int = 10
var pi: float = 3.14159
var enabled: bool = true
var message: String = "Hello Godot"
```

Data Types - Built-in (1)

Type	Description
`Vector2`	2D vector (x, y)
`Vector2i`	2D integer vector
`Vector3`	3D vector (x, y, z)
`Vector3i`	3D integer vector
`Vector4`	4D vector (x, y, z, w)

```
var position = Vector2(100, 200)
var velocity = Vector3(1.0, 0.0, -0.5)
var color = Color(1.0, 0.0, 0.0, 1.0)  # RGBA

# Vector operations
var direction = position.normalized()
var distance = position.length()
var dot_product = velocity.dot(Vector3.UP)
```

Data Types - Built-in (2)

Type	Description
`Color`	RGBA color
`Rect2`	2D rectangle
`Transform2D`	2D transformation matrix
`Transform3D`	3D transformation matrix
`Quaternion`	3D rotation
`Basis`	3D rotation/scale matrix
`Plane`	3D plane
`AABB`	Axis-aligned bounding box

Collections - Arrays

```
# Array creation
var items = []
var numbers = [1, 2, 3, 4, 5]
var mixed = [42, "text", true, Vector2(0, 0)]

# Typed arrays (Godot 4)
var integers: Array[int] = [1, 2, 3]
var nodes: Array[Node] = []

# Array operations
items.append("sword")
items.insert(0, "shield")
items.remove_at(1)
var first = items[0]
var last = items[-1]
var size = items.size()

# Iteration
for item in items:
    print(item)
```

Collections - Dictionaries

```
# Dictionary creation
var player = {
    "name": "Hero",
    "health": 100,
    "level": 5
}

# Access values
print(player["name"])
print(player.health)  # Alternative syntax

# Modify
player["health"] = 90
player.experience = 1500  # Add new key

# Check existence
if "mana" in player:
    print(player.mana)

# Iteration
for key in player:
    print(key, ": ", player[key])

for key in player.keys():
    print(key)

for value in player.values():
```

Type Hints & Static Typing

```
# Variable type hints
var speed: float = 100.0
var player_name: String = "Hero"
var items: Array[String] = ["sword", "shield"]

# Function type hints
func calculate_damage(base: int, multiplier: float) -> int:
    return int(base * multiplier)

# Return type inference
func get_position() -> Vector2:
    return Vector2(x, y)

# Type casting
var node = get_node("Player") as CharacterBody2D
var value = some_value as int

# Type checking
if value is int:
    print("It's an integer")
```

Benefits: Better autocomplete, early error detection, performance

Operators - Arithmetic

```
var a = 10
var b = 3

# Basic arithmetic
var sum = a + b          # 13
var diff = a - b         # 7
var product = a * b      # 30
var quotient = a / b     # 3.333...
var remainder = a % b    # 1
var power = a ** b       # 1000

# Integer division
var int_div = 10 // 3    # 3

# Compound assignment
a += 5    # a = a + 5
a -= 2    # a = a - 2
a *= 2    # a = a * 2
a /= 4    # a = a / 4
```

Operators - Comparison & Logic

```
# Comparison
var x = 10
var is_equal = (x == 10)        # true
var not_equal = (x != 5)        # true
var greater = (x > 5)           # true
var less_equal = (x <= 10)      # true

# Logical operators
var and_result = true and false  # false
var or_result = true or false    # true
var not_result = not true        # false

# Ternary operator
var status = health > 0 if "alive" else "dead"

# Null coalescing
var name = player_name if player_name else "Unknown"
```

Control Flow - If/Else

```
# Basic if
if health > 0:
    print("Still alive")

# If-else
if score >= 100:
    print("You win!")
else:
    print("Try again")

# If-elif-else
if health > 75:
    status = "Healthy"
elif health > 25:
    status = "Wounded"
else:
    status = "Critical"

# Inline if
var result = "Pass" if score >= 60 else "Fail"
```

Control Flow - Match

```
# Match statement (like switch)
match weapon_type:
    "sword":
        damage = 10
    "axe":
        damage = 15
    "bow":
        damage = 8
    _:  # Default case
        damage = 5

# Match with multiple values
match state:
    State.IDLE, State.WALK:
        can_attack = true
    State.JUMP, State.FALL:
        can_attack = false

# Match with patterns
match value:
    [1, 2]:
        print("Array with 1 and 2")
    {"name": "Hero"}:
        print("Dictionary with name")
```

Loops - For

```
# For loop with range
for i in range(5):
    print(i)  # 0, 1, 2, 3, 4

# Range with start and end
for i in range(2, 7):
    print(i)  # 2, 3, 4, 5, 6

# Range with step
for i in range(0, 10, 2):
    print(i)  # 0, 2, 4, 6, 8

# Iterate over array
var items = ["sword", "shield", "potion"]
for item in items:
    print(item)

# Iterate with index
for i in items.size():
    print(i, ": ", items[i])
```

Loops - While

```
# While loop
var count = 0
while count < 5:
    print(count)
    count += 1

# Infinite loop with break
while true:
    process_input()
    if should_exit:
        break

# Continue to skip iteration
var i = 0
while i < 10:
    i += 1
    if i % 2 == 0:
        continue  # Skip even numbers
    print(i)
```

Functions - Basics

```
# Simple function
func greet():
    print("Hello!")

# Function with parameters
func add(a: int, b: int) -> int:
    return a + b

# Default parameters
func create_enemy(health: int = 100, damage: int = 10):
    print("Enemy: ", health, " HP, ", damage, " DMG")

# Call with named arguments
create_enemy(damage=20, health=150)

# Multiple return values
func get_position_and_velocity() -> Array:
    return [position, velocity]

var pos_vel = get_position_and_velocity()
var pos = pos_vel[0]
var vel = pos_vel[1]
```

Functions - Advanced

```
# Pass by reference (Objects, Arrays, Dictionaries)
func modify_array(arr: Array):
    arr.append("new_item")  # Modifies original

# Variadic arguments
func sum_all(numbers: Array) -> int:
    var total = 0
    for num in numbers:
        total += num
    return total

# Lambda functions (Godot 4)
var double = func(x): return x * 2
print(double.call(5))  # 10

# Higher-order functions
var numbers = [1, 2, 3, 4, 5]
var doubled = numbers.map(func(x): return x * 2)
var evens = numbers.filter(func(x): return x % 2 == 0)
```

Classes - Basics

```
# Inheriting from Node
extends Node2D

# Class variables
var health: int = 100
var max_health: int = 100

# Static typing for class
class_name Player

# Inner class
class Weapon:
    var damage: int
    var name: String

    func _init(weapon_name: String, weapon_damage: int):
        name = weapon_name
        damage = weapon_damage

var sword = Weapon.new("Excalibur", 50)
```

Classes - Lifecycle Methods

```
extends Node

# Called when node enters scene tree (once)
func _ready():
    print("Node initialized")

# Called every frame
func _process(delta):
    position.x += speed * delta

# Called every physics frame (60 FPS)
func _physics_process(delta):
    velocity.y += gravity * delta

# Called when node exits scene tree
func _exit_tree():
    print("Cleanup")

# Called when node's name changes
func _notification(what):
    if what == NOTIFICATION_RENAMED:
        print("Node renamed")
```

Classes - Properties & Setters/Getters

```
extends Node

# Property with getter/setter
var health: int = 100:
    set(value):
        health = clamp(value, 0, max_health)
        health_changed.emit(health)
    get:
        return health

# Read-only property
var is_alive: bool:
    get:
        return health > 0

# Computed property
var health_percentage: float:
    get:
        return float(health) / max_health * 100.0

signal health_changed(new_health)
```

Signals - Definition & Emission

```
extends Node

# Define signals
signal health_changed(new_health: int)
signal died
signal level_up(new_level: int, skill_points: int)

# Emit signals
func take_damage(amount: int):
    health -= amount
    health_changed.emit(health)

    if health <= 0:
        died.emit()

func gain_experience(amount: int):
    experience += amount
    if experience >= next_level_exp:
        level += 1
        level_up.emit(level, 3)
```

Signals - Connecting

```
extends Node

func _ready():
    # Connect to signal
    player.health_changed.connect(_on_player_health_changed)

    # Connect with lambda
    player.died.connect(func(): print("Game Over"))

    # Connect one-shot (auto-disconnect after first call)
    timer.timeout.connect(_on_timeout, CONNECT_ONE_SHOT)

    # Disconnect
    player.health_changed.disconnect(_on_player_health_changed)

func _on_player_health_changed(new_health: int):
    health_bar.value = new_health

    if new_health < 25:
        show_low_health_warning()
```

Annotations - @export

```
extends Node2D

# Basic exports (visible in Inspector)
@export var speed: float = 100.0
@export var health: int = 100
@export var player_name: String = "Hero"

# Export ranges
@export_range(0, 100) var volume: int = 50
@export_range(0.0, 1.0, 0.1) var opacity: float = 1.0

# Export enums
@export_enum("Sword", "Axe", "Bow") var weapon_type: String
@export_flags("Fire", "Ice", "Lightning") var resistances: int

# Export file/directory
@export_file var config_path: String
@export_dir var save_directory: String

# Export nodes
@export var target: Node2D
```

Annotations - @onready & Others

```
extends Node

# @onready - initialized when _ready() is called
@onready var sprite = $Sprite2D
@onready var timer = $Timer
@onready var animation = get_node("AnimationPlayer")

# @export_category - organize in Inspector
@export_category("Movement")
@export var speed: float = 100.0
@export var acceleration: float = 500.0

@export_category("Combat")
@export var damage: int = 10
@export var attack_speed: float = 1.0

# @export_group - sub-category
@export_group("Advanced Settings")
@export var debug_mode: bool = false
```

Annotations - @tool & @rpc

```
# @tool - run in editor
@tool
extends EditorScript

func _run():
    print("Running in editor!")
    # Can modify scene here

# @rpc - multiplayer Remote Procedure Calls
extends Node

@rpc("any_peer", "call_remote")
func player_moved(position: Vector2):
    print("Player at: ", position)

# Call on remote peers
func _process(delta):
    if position_changed:
        player_moved.rpc(global_position)
```

Built-in Functions (1)

```
# Math functions
var absolute = abs(-10)          # 10
var maximum = max(5, 10)         # 10
var minimum = min(5, 10)         # 5
var clamped = clamp(15, 0, 10)   # 10
var rounded = round(3.7)         # 4
var floored = floor(3.7)         # 3
var ceiled = ceil(3.2)           # 4

# Trigonometry
var sine = sin(PI / 2)           # 1.0
var cosine = cos(0)              # 1.0
var tangent = tan(PI / 4)        # 1.0
var angle = atan2(y, x)          # Angle from origin

# Random
randomize()  # Seed RNG
var rand_int = randi() % 100     # 0-99
var rand_float = randf()         # 0.0-1.0
var rand_range = randf_range(5.0, 10.0)
```

Built-in Functions (2)

```
# Type conversion
var text = str(42)                  # "42"
var number = int("42")              # 42
var decimal = float("3.14")         # 3.14

# String operations
var lower = "HELLO".to_lower()      # "hello"
var upper = "hello".to_upper()      # "HELLO"
var trimmed = "  text  ".strip_edges()  # "text"
var parts = "a,b,c".split(",")      # ["a", "b", "c"]
var joined = ",".join(["x", "y"]) # "x,y"

# Type checking
var is_num = typeof(value) == TYPE_INT
var is_node = value is Node
```

Built-in Functions (3)

```
# Node tree functions
var parent = get_parent()
var children = get_children()
var node = get_node("Path/To/Node")
var node_short = $Path/To/Node

# Scene management
var scene = load("res://scenes/enemy.tscn")
var instance = scene.instantiate()
add_child(instance)
instance.queue_free()  # Deferred deletion

# Group operations
add_to_group("enemies")
remove_from_group("enemies")
var enemies = get_tree().get_nodes_in_group("enemies")
get_tree().call_group("enemies", "take_damage", 10)
```

String Formatting

```
# String interpolation
var name = "Hero"
var level = 10
var text = "Player: %s, Level: %d" % [name, level]

# Format with dictionary
var msg = "{name} has {hp} HP".format({
    "name": player_name,
    "hp": health
})

# Multiple formats
var formatted = "%s scored %d points" % [name, score]
var decimal = "Pi: %.2f" % PI  # "Pi: 3.14"
var padded = "%5d" % 42        # "   42"

# Multi-line strings
var long_text = """
This is a
multi-line
string
"""
```

Error Handling

```
# Assert (debug only)
assert(health > 0, "Health must be positive")

# Print errors
push_error("Something went wrong!")
push_warning("This is deprecated")

# Check for null
if node != null:
    node.do_something()

# Safe navigation (Godot 4)
var value = node?.get_value()  # null if node is null

# Error codes
var file = FileAccess.open("path", FileAccess.READ)
if file == null:
    var error = FileAccess.get_open_error()
    match error:
        ERR_FILE_NOT_FOUND:
            print("File not found")
        ERR_CANT_OPEN:
            print("Cannot open file")
```

Enums

```
# Define enum
enum State {
    IDLE,
    WALK,
    RUN,
    JUMP
}

# Enum with custom values
enum Element {
    FIRE = 1,
    WATER = 2,
    EARTH = 4,
    AIR = 8
}

# Use enum
var current_state = State.IDLE

func change_state(new_state: State):
    match new_state:
        State.IDLE:
            animation.play("idle")
        State.WALK:
            animation.play("walk")
        State.RUN:
```

Preload vs Load

```
# Preload - loads at compile time (faster, must be constant)
const PlayerScene = preload("res://player.tscn")
const PlayerScript = preload("res://player.gd")

func spawn_player():
    var player = PlayerScene.instantiate()
    add_child(player)

# Load - loads at runtime (dynamic paths allowed)
var scene_path = "res://enemies/" + enemy_type + ".tscn"
var enemy_scene = load(scene_path)
var enemy = enemy_scene.instantiate()

# ResourceLoader (async loading)
ResourceLoader.load_threaded_request(scene_path)
# ... later
var scene = ResourceLoader.load_threaded_get(scene_path)
```

Coroutines & Await

```
# Await signal
func shoot():
    animation.play("shoot")
    await animation.animation_finished
    spawn_bullet()

# Await multiple frames
func delayed_action():
    await get_tree().process_frame  # Wait 1 frame
    print("One frame later")

    await get_tree().create_timer(2.0).timeout
    print("2 seconds later")

# Chained awaits
func sequence():
    play_animation("intro")
    await animation.animation_finished

    await get_tree().create_timer(1.0).timeout

    play_animation("outro")
    await animation.animation_finished

    queue_free()
```

Yield Pattern (Multiple Signals)

```
# Wait for first signal to fire
func wait_for_input():
    await any([
        button_a.pressed,
        button_b.pressed
    ])
    print("Button pressed!")

# Custom async function
func fade_out(duration: float):
    var tween = create_tween()
    tween.tween_property(self, "modulate:a", 0.0, duration)
    await tween.finished

# Use it
func game_over():
    await fade_out(1.0)
    get_tree().reload_current_scene()
```

Array & Dictionary Methods (1)

```
# Array methods
var items = ["sword", "shield", "potion"]

items.append("bow")                    # Add to end
items.push_back("arrow")               # Same as append
items.push_front("helmet")             # Add to start
items.insert(1, "armor")               # Insert at index

var removed = items.pop_back()         # Remove from end
var first = items.pop_front()          # Remove from start
items.remove_at(2)                     # Remove by index
items.erase("shield")                  # Remove by value

print(items.size())                    # Length
print(items.is_empty())                # Check if empty
items.clear()                          # Remove all
```

Array & Dictionary Methods (2)

```
# Array searching & sorting
var numbers = [5, 2, 8, 1, 9]

numbers.sort()                         # [1, 2, 5, 8, 9]
numbers.reverse()                      # [9, 8, 5, 2, 1]

var has_five = numbers.has(5)         # true
var index = numbers.find(8)           # 2 (or -1 if not found)
var max_val = numbers.max()           # 9
var min_val = numbers.min()           # 1

# Dictionary methods
var dict = {"a": 1, "b": 2}

dict.erase("a")                       # Remove key
dict.clear()                          # Remove all
var keys = dict.keys()                # ["b"]
var values = dict.values()            # [2]
print("a" in dict)                    # false
```

Functional Programming

```
var numbers = [1, 2, 3, 4, 5]

# Map - transform each element
var doubled = numbers.map(func(x): return x * 2)
# Result: [2, 4, 6, 8, 10]

# Filter - select elements
var evens = numbers.filter(func(x): return x % 2 == 0)
# Result: [2, 4]

# Reduce - accumulate
var sum = numbers.reduce(func(acc, x): return acc + x, 0)
# Result: 15

# Any - check if any element matches
var has_even = numbers.any(func(x): return x % 2 == 0)
# Result: true

# All - check if all elements match
var all_positive = numbers.all(func(x): return x > 0)
# Result: true
```

Object-Oriented Programming

```
# Inheritance
class_name Enemy
extends CharacterBody2D

var health: int = 100

func take_damage(amount: int):
    health -= amount

# Subclass
class_name Boss
extends Enemy

var shield: int = 50

func take_damage(amount: int):
    if shield > 0:
        shield -= amount
    else:
        super.take_damage(amount)  # Call parent method

# Virtual methods
func _to_string():
    return "Boss(health=%d, shield=%d)" % [health, shield]
```

Static Members

```
class_name GameManager
extends Node

# Static variable (shared across all instances)
static var instance: GameManager
static var score: int = 0
static var high_score: int = 0

func _ready():
    instance = self

# Static method
static func add_score(amount: int):
    score += amount
    if score > high_score:
        high_score = score

# Access from anywhere
func _on_enemy_killed():
    GameManager.add_score(100)
    print("Score: ", GameManager.score)
```

Memory Management

```
# Manual deletion
var node = Node.new()
add_child(node)
node.queue_free()  # Deferred (safe)
# node.free()      # Immediate (dangerous in _process)

# Reference counting (Objects)
var resource = SomeResource.new()  # refcount = 1
var ref2 = resource                # refcount = 2
ref2 = null                        # refcount = 1
# Automatically freed when refcount = 0

# WeakRef for avoiding circular references
var weak_player = weakref(player)
if weak_player.get_ref():
    weak_player.get_ref().do_something()

# Object pools (reuse objects)
var bullet_pool = []
func get_bullet():
    if bullet_pool.is_empty():
        return Bullet.new()
    return bullet_pool.pop_back()
```

Debugging & Print Functions

```
# Basic printing
print("Hello")
print("Multiple", "arguments", 123)

# Formatted printing
print("Player: %s, HP: %d" % [name, health])

# Debug printing (with stack trace)
print_debug("Debug info")

# Rich printing (with colors/formatting in console)
print_rich("[color=red]Error![/color]")
print_rich("[b]Bold text[/b]")

# Conditional printing
if OS.is_debug_build():
    print("Debug mode only")

# Performance profiling
var start = Time.get_ticks_usec()
heavy_function()
var elapsed = Time.get_ticks_usec() - start
print("Time: ", elapsed, " microseconds")
```

Best Practices

Naming Conventions:

- Variables/functions: `snake_case`
- Constants: `UPPER_CASE`
- Classes: `PascalCase`
- Private members: `_leading_underscore`

Code Organization:

- Group related variables
- Signals first, then exports, then regular vars
- Public methods before private (`_method`)
- Keep functions small and focused

Performance:

Common Patterns - Singleton

```
# Global autoload (game_manager.gd)
extends Node

static var instance: GameManager

var score: int = 0
var game_state: String = "menu"

signal score_changed(new_score)

func _ready():
    instance = self

func add_score(amount: int):
    score += amount
    score_changed.emit(score)

# Access from anywhere
func _on_pickup():
    GameManager.add_score(10)
```

Common Patterns - State Machine

```
class_name StateMachine
extends Node

var current_state: State
var states: Dictionary = {}

func add_state(name: String, state: State):
    states[name] = state
    add_child(state)

func change_state(new_state: String):
    if current_state:
        current_state.exit()

    current_state = states[new_state]
    current_state.enter()

func _process(delta):
    if current_state:
        current_state.update(delta)
```

Common Patterns - Observer

```
# Observable subject
class_name Subject
extends Node

signal value_changed(new_value)

var _value: int:
    set(v):
        if _value != v:
            _value = v
            value_changed.emit(_value)
    get:
        return _value

# Observer
class_name HealthBar
extends ProgressBar

func _ready():
    player.value_changed.connect(_on_health_changed)

func _on_health_changed(new_health: int):
    value = new_health
```

Common Patterns - Factory

```
class_name EnemyFactory
extends Node

const GOBLIN = preload("res://enemies/goblin.tscn")
const ORC = preload("res://enemies/orc.tscn")
const DRAGON = preload("res://enemies/dragon.tscn")

func create_enemy(type: String) -> Enemy:
    match type:
        "goblin":
            return GOBLIN.instantiate()
        "orc":
            return ORC.instantiate()
        "dragon":
            return DRAGON.instantiate()
        _:
            push_error("Unknown enemy type: " + type)
            return null
```

Useful One-Liners

```
# Random element from array
var random_item = items[randi() % items.size()]

# Toggle boolean
is_enabled = not is_enabled

# Clamp position to screen
position = position.clamp(Vector2.ZERO, screen_size)

# Linear interpolation
position = position.lerp(target_position, 0.1)

# Convert bool to int
var int_value = int(is_active)  # 1 or 0

# Safe division
var result = numerator / max(denominator, 0.001)

# Modulo wrap-around
var wrapped = (index + 1) % array.size()
```

Performance Tips

Use signals instead of polling:

```
# Bad
func _process(_delta):
    if player.health <= 0:
        game_over()

# Good
func _ready():
    player.died.connect(game_over)
```

Cache node references:

```
# Bad
func _process(_delta):
    $Sprite2D.position.x += speed

# Good
@onready var sprite = $Sprite2D
func _process(_delta):
    sprite.position.x += speed
```

Godot 4 New Features

Typed Arrays:

```
var numbers: Array[int] = [1, 2, 3]
var nodes: Array[Node] = []
```

await instead of yield:

```
# Godot 3
yield(get_tree().create_timer(1.0), "timeout")

# Godot 4
await get_tree().create_timer(1.0).timeout
```

Lambda improvements:

```
var double = func(x): return x * 2`
items.map(func(x): return x * 2)
```

Resources & Further Learning

Official Documentation:

- docs.godotengine.org/en/stable/tutorials/scripting/gdscript/
- GDScript reference manual
- Built-in class documentation

Practice Projects:

- 2D platformer
- Top-down shooter
- Puzzle game
- Your own game jam entry!

Community:

- /r/godot

Key Takeaways

✅ **Python-like syntax** - Easy transition if you know Python
✅ **Optional static typing** - Better performance and IDE support
✅ **Signals** - Powerful event system
✅ **Built-in types** - Vector2, Color, Transform, etc.
✅ **Functional features** - map, filter, reduce, lambdas
✅ **Memory management** - Reference counting for objects
✅ **Coroutines** - await for async operations
✅ **Integration** - Tight coupling with Godot's node system

Start simple, practice regularly, read the docs!