# 부록 A
# LUA 스크립팅 레퍼런스

열한 번째 시간에서는 프로그래밍 언어 Lua를 소개하고, 기본 개념에 대한 간략한 개요를 알아봤다. 부록 A에는 Lua 언어 공부를 시작할 때 유용할 만한 몇 가지 추가 Lua 레퍼런스 표와 개념 설명을 포함하고 있다.

## 데이터 타입과 이뉴머레이션의 속성 변경

데이터 타입이란 변수가 저장할 수 있는 여러 다른 유형의 데이터를 의미한다. 표 A.1과 A.2에 나열했다.

표 A.1 기본 Lua 데이터 타입들

| 데이터 타입 설명 | |
|---|---|
| Nil | 데이터 없음 |
| Boolean | 데이터가 참 혹은 거짓 |
| number | 데이터가 실제 숫자 |
| string | 데이터가 캐릭터의 나열 |
| function | 데이터가 C*나 Lua에서 작성된 메서드 |
| userdata | C* 데이터 |
| thread | 데이터가 독립적 실행 스레드 |
| table | 어레이, 심볼 테이블, 세트, 레코드, 그래프, 트리 등 |

표 A.2 로블록스 Lua 데이터 타입들

| 데이터 타입 분류 | 커스텀 로블록스 데이터 타입들 |
|---|---|
| **색** | / BrickColor, Color3, ColorSequence, ColorSequenceKeypoint |
| **위치 혹은 관련 영역** | Axes, CFrame, UDim, UDim2, Rect, Region3, Region3int16 |
| **숫자와 나열** | NumberRange, NumberSequence, NumberSequenceKeypoint |
| **연결과 이벤트** | RBXScriptConnection, RBXScriptSignal |
| **벡터** | Vector2, Vector2int16, Vector3, Vector3int16 |
| **클래스** | Instance |
| **이뉴머레이션 관련** | Enum, EnumItem, Enums |
| **다른 타입들** | DockWidgetPluginGuiInfo, Faces, PathwayPoint, PhysicalProperties, Random, Ray, TweenInfo |

이뉴머레이션 또는 이넘enum은 해당 이넘에 속한 값의 모음을 담은 사용자 데이터를 저장하는 특수 데이터 유형이다. 이 값들은 읽기 전용이다. 스크립트에서 이넘에 접근하려면 이넘이라는 글로벌 오브젝트를 사용해야 한다. 이넘 목록은 https://developer.roblox.com/en-us/api-reference/enum에서 찾을 수 있다.

다음은 데이터 타입 또는 이넘의 속성을 변경하는 방법에 대한 몇 가지 예제이다. 벽돌 머티리얼과 빨간 벽돌색을 사용해서 redBrick이라는 새로운 파트를 생성하고 싶다고 가정해보자. 해당 파트는 기본 설정상 회색 돌 색상이므로 빨간색으로 변경하려면 다음과 같이 해야 한다.

```
redBrick.BrickColor = BrickColor.Red()
```

이제 redBrick의 머티리얼을 벽돌로 변경하려면 이넘 리스트에서 머티리얼을 가져와야 한다. Material이 이넘이기 때문이다.

```
redBrick.Material = Enum.Material.Brick
```

코드를 입력할 때 에디터가 자동 제안이나 자동 입력을 해주는 것을 눈여겨보자.

## 조건 구문

조건 구문conditional structure란 프로그램의 흐름을 제어하기 위한 방법이다. 조건이 충족하면 Lua는 true로 처리하고, 충족하지 못하면 값이 false나 nil이 된다. 조건들은 관계 연산자relational operator를 사용해 체크한다. 편의를 위해 var1이 30이고, var2가 10이라고 가정해보자.

표 A.3 관계 연산자들

| 연산자 | 설명 |
|---|---|
| + | 덧셈. 두 함수를 더한다. var1+var2는 40을 반환한다. |
| - | 뺄셈. 첫 함수에서 둘째 함수를 뺀다. var1-var2는 20을 반환한다. |
| * | 곱셈. 두 함수를 곱한다. Var1*var2는 300을 반환한다. |
| / | 나눗셈. 분모로 분자를 나눈다. var1/var2는 3을 반환한다. |
| % | 나머지. 나눗셈의 나머지를 반환한다. var1%var2는 0을 반환한다. |
| ^ | 승. 승수 값을 반환한다. var1^2는 900을 반환한다. |

표 A.4는 조건 연산자conditional operator를 담고 있다. 편의를 위해 var1이 30이고, var2가 10이라고 가정해보자.

표 A.4 조건 연산자

| 연산자 | 설명 |
|---|---|
| == | 같다. (var1 == var2)는 true가 아니다. |
| 〉 | 보다 크다. (var1 〉 var2)는 true이다. |
| 〈 | 보다 작다. (var1 〈 var2)는 false이다. |
| 〉= | 보다 크거나 같다. (var1 〉= var2)는 true이다. |
| 〈= | 보다 작거나 같다. (var1 〈= var2)는 false이다. |
| ~= | 같지 않다. (var1 ~= var2)는 true이다. |

표 A.5는 논리 연산자logical operator를 담고 있다. 편의를 위해 var1이 true를 담고 있고, var2가 false를 담고 있다고 가정해보자.

표 A.5 논리 연산자

| 연산자 | 설명 |
|---|---|
| and | 논리 and. 만일 두 변수 모두 0이 아니면 true이다. (var1 and var2)는 false이다. |
| or | 논리 or. 만일 두 변수 중 하나가 0이 아니면 true이다. (var1 or var2)는 true이다. |
| not | 논리 not. 논리 상태를 반전시킨다. !(var1)는 false이다. |

## Lua 지식 넓히기

Lua 프로그래밍 언어에 대해 더 자세히 알고 싶다면 웹에서 볼 수 있는 다음 두 개의 리소스를 활용해보자.

- Lua 레퍼런스 매뉴얼: http://www.lua.org/manual
- Lua 프로그래밍: http://www.lua.org/pil

# 부록 B
# 휴머노이드의 속성과 함수

열두 번째 시간에는 휴머노이드Humanoid 오브젝트를 다뤘다. 휴머노이드에는 데미지를 적용하거나, 표시되는 이름을 변경하거나, 카메라 오프셋을 조작하거나, 휴머노이드의 상태(예: 오르기, 사망, 현재 생명력 등)를 가져올 때 등에 사용할 수 있는 유용한 함수와 속성들을 가지고 있다.

표 B.1는 휴머노이드의 속성들 중 일부를 담고 있다. 모든 속성을 담지 않은 이유는 몇몇은 설명이 필요 없고, 몇몇은 관련이 없기 때문이다.

표 B.1 휴머노이드 속성들과 의미

| 속성 | 의미 |
|---|---|
| Camera Offset | 캐릭터로부터 카메라 오프셋 거리를 설정한다. |
| DisplayDistanceType | Viewer: 지정한 거리에 있는 모두의 DisplayName이 보인다.<br>Subject: 모두가 각각 보이는 거리를 지정한다.<br>None: 본인 PC에서 보이지 않는다. |
| DisplayName | 캐릭터 위에 커스텀 DisplayName을 표시할 수 있는 옵션으로 기본설정은 사용자 이름이다. |
| HealthDisplayDistance | 다른 이들의 생명력을 볼 수 있는 거리. 기본 설정은 100(정수). |
| HealthDisplayType | 옵션: WhenDamaged, On, Off. |
| NameDisplayDistance | 다른 이들의 이름을 볼 수 있는 거리. 기본 설정은 100(정수). |
| NameOcclusion | OccludeAll: 모든 네임 태그를 오브젝트 뒤에 숨긴다.<br>NoOcclusion: 네임 태그들이 오브젝트를 투과해서 보인다.<br>EnemyOcclusion: 다른 팀의 이름들을 오브젝트 뒤에 숨긴다. |
| RigType | R15: 15 바디 파트 리그(rig).<br>R6: 6 바디 파트 리그. |

| | |
|---|---|
| JumpPower | UseJumpPower 사용: 점프에 적용할 힘의 양.<br>UseJumpPower 비사용: 점프 높이를 직접 선택. |
| Jump/PlatformStand/Sit | 불리언(Boolean) – true 혹은 false. 플레이어 입력을 통해 오버라이드 가능. |
| TargetPoint<br>/WalkToPart<br>/WalkToPoint | 컷씬 중에 MoveTo()를 사용해서 NPC와 플레이어를 움직일 때 주로<br>사용: WalkToPoint [첫 번째 파라미터].<br>WalkToPart [두 번째 파라미터].<br>TargetPoint와 WalkToPoint는 Vector3 입력을 사용한다.<br>WalkToPart는 Instance 입력을 사용한다. |

위 속성들을 이해하기 위해 게임 내에서 자유롭게 테스트해보면 좋다.

휴머노이드에는 도구 장착, 애니메이션 로딩, 데미지 적용, 플레이어 강제 이동, 다양한 플레이어 상태 설정 등을 위한 다양한 함수(표 B.2에 일부가 나와 있음)가 포함돼 있다. 몇몇 함수는 설명이 필요 없거나, 관련이 없기 때문에 포함하지 않았다.

표 B.2 휴머노이드 함수들과 의미

| 함수 | 의미 |
|---|---|
| GetState()/ChangeState() | HumanoidStateType을 가져오거나 설정한다. |
| EquipTool()/UnequipTools() | 특정한 도구를 장착한다. |
| LoadAnimation() | 지정한 애니메이션을 휴머노이드로 불러온 후 AnimationTrack을 반환해서 애니메이션을 재생하는데 사용한다. |
| TakeDamage() | Humanoid.Health 속성을 지정한 값만큼 낮춰 플레이어에게 데미지를 적용한다. |
| MoveTo() | 플레이어가 지정한 Vector3나 파트로 이동을 시도한다. Humanoid.WalkToPoint나 Humanoid.WalkToPart를 수정한다. |

마지막으로 휴머노이드는 휴머노이드의 속성 변경을 감지할 수 있는 RBXScriptSignal(혹은 이벤트)들이 있다. 표 B.3는 그중에서도 가장 중요한 이벤트들을 담고 있다.

표 B.3 휴머노이드 이벤트들과 의미

| 이벤트 | 의미 |
|---|---|
| AnimationPlayed() | 플레이어의 애니메이션이 시작되면 발생한다. |
| Died() | 플레이어가 사망하면(생명력이 0이 되거나 Head와 Torso가 분리되면) 발생한다. |
| HealthChanged() | 플레이어의 생명력이 변경되면 발생한다. |
| MoveToFinished() | 플레이어가 특정 위치로 이동하기 시작하면(일반적으로 MoveTo() 함수를 통해) 발생한다. |
| Seated() | 플레이어의 .Sit 속성이 활성화되면 발생한다. |
| StateChange() | HumanoidStateType이 변경되면 발생한다. Old와 New State 파라미터 사용. |

루프loop를 사용하는 대신에 이벤트를 사용해 속성의 변경을 체크하는 것이 더 나은 구성이다. 속성의 상태에 따라 쉽게 함수를 실행시킬 수 있고, 전반적으로 더 나은 성능을 꾀할 수 있다.