Unity C# 기초 학습 가이드북

이 가이드북은 Unity에서 C# 프로그래밍을 처음 배우는 분들을 위한 기초 학습 자료입니다. 각 개념을 쉽고 구체적으로 설명합니다.

목차

---

1. 변수와 조건문 기초

변수란?

변수는 데이터를 저장하는 상자라고 생각하면 됩니다. 게임에서 플레이어의 체력, 점수, 이름 등을 저장할 때 사용합니다.

변수 타입 종류

정수 (int)

- **설명**: 정수만 저장할 수 있습니다 (소수점 없음)
- **예시**: `int hp = 100;` (체력 100)
- **사용 예**: 체력, 점수, 레벨, 개수 등

실수 (float)

- **설명**: 소수점이 있는 숫자를 저장합니다
- **예시**: `float pi = 3.14f;` (f를 붙여야 함)
- **사용 예**: 속도, 거리, 비율 등

문자열 (string)

- **설명**: 글자를 저장합니다
- **예시**: `string myName = "홍길동";` (큰따옴표 필수)
- **사용 예**: 이름, 메시지, 설명 등

불린 (bool)

- **설명**: 참(true) 또는 거짓(false)만 저장합니다
- **예시**: `bool isGameOver = false;`
- **사용 예**: 게임 종료 여부, 버튼 활성화 여부 등

접근 제한자

private (비공개)

- **설명**: 이 스크립트 안에서만 사용 가능
- **예시**: `private int hp = 100;`
- **언제 사용?**: 다른 스크립트에서 접근할 필요가 없을 때

public (공개)

- **설명**: Unity Inspector 창에서 보이고 수정 가능
- **예시**: `public int hp = 100;`
- **언제 사용?**: 게임 실행 중 Inspector에서 값을 바꾸고 싶을 때

Unity 생명주기 메서드

Start()

- **언제 실행?**: 게임이 시작될 때 딱 한 번만 실행
- **용도**: 초기 설정, 변수 초기화
- **예시**: 체력을 100으로 설정, 시작 메시지 출력

Update()

- **언제 실행?**: 매 프레임마다 계속 실행 (초당 60번 정도)
- **용도**: 지속적으로 확인해야 하는 것들
- **예시**: 키 입력 감지, 충돌 체크

조건문 (if)

조건문은 "만약 ~라면"을 표현합니다.

```
if (hp <= 0)
{
    Debug.Log("게임 오버!");
}
```

- **동작**: hp가 0 이하라면 "게임 오버!" 메시지를 출력
- **활용**: 체력이 0이 되면 게임 종료, 점수가 100점 넘으면 레벨업 등

산술 연산

변수에 숫자를 더하거나 빼는 연산입니다.

```
hp = hp - 100;  // hp에서 100을 뺀다
hp = hp + 50;   // hp에 50을 더한다
hp = hp * 2;    // hp를 2배로 만든다
hp = hp / 2;    // hp를 2로 나눈다
```

Debug.Log

코드가 제대로 실행되는지 확인하는 도구입니다.

```
Debug.Log("안녕하세요!");
Debug.Log("현재 체력: " + hp);
```

- Unity 하단의 Console 창에서 메시지를 확인할 수 있습니다
- 변수 값을 확인하거나 오류를 찾을 때 유용합니다

---

2. 비교 연산자와 논리 연산자

비교 연산자

두 값을 비교해서 참(true) 또는 거짓(false)을 반환합니다.

크기 비교

- `>` : 왼쪽이 더 크면 true
  - 예: `8 > 10` → false (8은 10보다 작음)
  - 예: `2 < 5` → true (2는 5보다 작음)

같음 비교

- `==` : 두 값이 같으면 true
  - 예: `5 == 5` → true
  - 예: `6 == 5` → false
- `!=` : 두 값이 다르면 true
  - 예: `7 != 5` → true (7과 5는 다름)
  - 예: `7 != 7` → false (7과 7은 같음)

크거나 같음, 작거나 같음

- `<=` : 왼쪽이 오른쪽보다 작거나 같으면 true
  - 예: `3 <= 5` → true
  - 예: `3 <= 3` → true
- `>=` : 왼쪽이 오른쪽보다 크거나 같으면 true

- 예: `4 >= 4` → true
- 예: `2 >= 4` → false

논리 연산자

여러 조건을 조합할 때 사용합니다.

OR 연산자 (||)

- **의미**: "또는" - 둘 중 하나라도 true면 true
- **예시**: `2 != 4 || 6 != 6`
  - `2 != 4`는 true
  - `6 != 6`는 false
  - 결과: true (하나라도 true면 true)

실생활 예시: "비가 오거나 바람이 불면 우산을 챙긴다"

- 비가 오면 → 우산 챙김 (true)
- 바람이 불면 → 우산 챙김 (true)
- 둘 다 아니면 → 우산 안 챙김 (false)

AND 연산자 (&&)

- **의미**: "그리고" - 둘 다 true여야 true
- **예시**: `2 != 4 && 6 != 6`
  - `2 != 4`는 true
  - `6 != 6`는 false
  - 결과: false (둘 다 true가 아니면 false)

실생활 예시: "돈이 있고 시간이 있으면 영화를 본다"

- 돈 있고 시간 있으면 → 영화 봄 (true)
- 돈 없거나 시간 없으면 → 영화 안 봄 (false)

NOT 연산자 (!)

- **의미**: "아니다" - true를 false로, false를 true로 바꿈
- **예시**: `!false` → true
- **예시**: `!true` → false

실생활 예시: "게임이 끝나지 않았다면 계속 플레이"

- `!isGameOver` → 게임이 끝나지 않았으면 true

---

3. 상수와 switch-case 문

상수 (const)

상수는 한 번 정하면 절대 바뀌지 않는 값입니다.

```
const int DEAD_HP = 0;
const int DANGER_HP = 40;
const int WARNING_HP = 60;
```

- **왜 사용하나요?**
  - 코드에서 의미 있는 이름을 사용할 수 있습니다
  - 나중에 값을 바꿀 때 한 곳만 수정하면 됩니다
  - 실수로 값을 바꾸는 것을 방지합니다
- **일반 변수와 차이점**
  - 변수: `int hp = 100;` (나중에 `hp = 50;`으로 변경 가능)
  - 상수: `const int MAX_HP = 100;` (나중에 변경 불가능)

Input.GetKeyDown()

키보드 입력을 감지하는 Unity 함수입니다.

```
if (Input.GetKeyDown(KeyCode.Space))
{
    // 스페이스바를 누르면 실행되는 코드
}
```

- **주요 키 코드**
  - `KeyCode.Space`: 스페이스바
  - `KeyCode.Enter`: 엔터키
  - `KeyCode.A`, `KeyCode.B`: 알파벳 키
  - `KeyCode.LeftArrow`, `KeyCode.RightArrow`: 방향키
- **GetKeyDown vs GetKey**
  - `GetKeyDown`: 키를 누르는 순간 딱 한 번만 true
  - `GetKey`: 키를 누르고 있는 동안 계속 true

switch-case 문

여러 가지 경우를 깔끔하게 처리하는 방법입니다.

```
switch (hp)
{
    case 0:
        Debug.Log("게임 오버!");
        break;
    case 40:
        Debug.Log("위험!");
        break;
```

```
        case 60:
            Debug.Log("주의하세요!");
            break;
        default:
            Debug.Log("기본 메시지입니다.");
            break;
    }
```

- **if-else if와 비교**
    - if-else if: 조건이 복잡할 때 유용
    - switch-case: 특정 값들을 비교할 때 더 깔끔
- **각 키워드 의미**
    - `switch (변수)`: 어떤 변수를 비교할지
    - `case 값:`: 변수가 이 값이면
    - `break;`: 여기서 끝내고 나가기 (필수!)
    - `default:`: 위의 어떤 경우도 아니면
- **실전 예시**
    - 체력에 따라 다른 메시지 표시
    - 레벨에 따라 다른 보상 지급
    - 아이템 타입에 따라 다른 효과 적용

---

4. 열거형 (enum)

enum이란?

enum은 관련된 상수들을 그룹으로 묶어서 관리하는 방법입니다.

```
public enum Fruit {
    APPLE,
    BANANA,
    PEACH,
    ORANGE
}
```

- **왜 사용하나요?**
    - 관련된 값들을 한 곳에 모아서 관리
    - 오타를 방지 (APPLE을 APLE로 잘못 쓰는 실수 방지)
    - Unity Inspector에서 드롭다운 메뉴로 선택 가능

enum 사용하기

```

```

```
public Fruit myFruit;  // Inspector에서 선택 가능

switch (myFruit)
{
    case Fruit.APPLE:
        Debug.Log("사과를 먹었어!");
        break;
    case Fruit.BANANA:
        Debug.Log("바나나를 먹었어!");
        break;
}
```

- **실전 활용 예시**
  - 아이템 타입: WEAPON, ARMOR, POTION
  - 게임 상태: MENU, PLAYING, PAUSED, GAMEOVER
  - 캐릭터 직업: WARRIOR, MAGE, ARCHER

Unity Inspector에서 사용

1. 스크립트에 `public Fruit myFruit;` 선언
2. Unity 에디터에서 GameObject에 스크립트 추가
3. Inspector 창에서 드롭다운 메뉴로 Fruit 선택 가능
4. 게임 실행 중에도 값을 바꿀 수 있음

---

5. 함수 기초

함수란?

함수는 특정 작업을 수행하는 코드 덩어리입니다. 같은 코드를 여러 번 반복하지 않고, 함수를 만들어서 재사용할 수 있습니다.

함수 만들기

```
void EnemyAttack()
{
    Debug.Log("Enemy Attack!");
    Debug.Log("Enemy Damaged");
    Debug.Log("Enemy HP UI Update");
}
```

- **구성 요소**
  - `void`: 이 함수는 값을 반환하지 않는다는 의미
  - `EnemyAttack`: 함수 이름 (원하는 대로 지을 수 있음)
  - `{ }`: 함수 안에 실행할 코드를 넣음

함수 호출하기

```
void Start()
{
    EnemyAttack();  // 함수 실행
    EnemyAttack();  // 또 실행
    EnemyAttack();  // 또 또 실행
}
```

- **장점**
  - 같은 코드를 여러 번 쓸 필요 없음
  - 나중에 수정할 때 한 곳만 수정하면 됨
  - 코드가 깔끔하고 읽기 쉬움

함수를 사용하는 이유

함수 없이:

```
Debug.Log("Enemy Attack!");
Debug.Log("Enemy Damaged");
Debug.Log("Enemy HP UI Update");

Debug.Log("Enemy Attack!");
Debug.Log("Enemy Damaged");
Debug.Log("Enemy HP UI Update");

Debug.Log("Enemy Attack!");
Debug.Log("Enemy Damaged");
Debug.Log("Enemy HP UI Update");
```

함수 사용:

```
EnemyAttack();
EnemyAttack();
EnemyAttack();
```

훨씬 간단하고 관리하기 쉽습니다!

---

6. 반환값이 있는 함수

return 키워드

함수에서 계산한 결과를 돌려주는 방법입니다.

```
int ThisYear()
{
```

```
    return 2025;  // 2025를 반환
}
```

- **사용 방법**

```
void Start()
{
    int year = ThisYear();  // year에 2025가 저장됨
    Debug.Log(year);        // 2025 출력
}
```

반환 타입

함수를 만들 때 어떤 타입의 값을 반환할지 정해야 합니다.

```
int ThisYear()      // int 타입 반환
{
    return 2025;
}

string GetName()    // string 타입 반환
{
    return "홍길동";
}

bool IsAlive()      // bool 타입 반환
{
    return true;
}
```

함수 내부에서 변수 사용

함수 안에서도 변수를 만들고 사용할 수 있습니다.

```
int NextYear()
{
    int lastYear = 2025;      // 함수 안에서 변수 선언
    int nextYear = lastYear + 1;  // 변수 사용
    return nextYear;          // 결과 반환
}
```

- **변수의 범위 (스코프)**
  - 함수 안에서 만든 변수는 그 함수 안에서만 사용 가능
  - 다른 함수에서는 사용할 수 없음

---

7. 매개변수가 있는 함수

매개변수란?

함수에 값을 전달해서 사용하는 방법입니다. 마치 계산기에 숫자를 입력하는 것과 같습니다.

```
int NextYear(int thisYear)  // thisYear를 받아서 사용
{
    int nextYear = thisYear + 1;
    return nextYear;
}
```

함수 호출 시 값 전달

```
void Start()
{
    int result = NextYear(2025);  // 2025를 전달
    Debug.Log(result);             // 2026 출력
}
```

- **동작 과정**
    1. `NextYear(2025)` 호출
    2. 함수 안에서 `thisYear`가 2025가 됨
    3. `thisYear + 1` 계산 → 2026
    4. 2026을 반환

매개변수의 장점

매개변수 없이 (비효율적):

```
int NextYear2025() { return 2026; }
int NextYear2026() { return 2027; }
int NextYear2027() { return 2028; }
// ... 계속 만들어야 함
```

매개변수 사용 (효율적):

```
int NextYear(int thisYear)
{
    return thisYear + 1;
}

// 어떤 연도든 사용 가능!
NextYear(2025);  // 2026
```

```
NextYear(2026);  // 2027
NextYear(2050);  // 2051
```

---

8. 여러 매개변수 사용하기

여러 매개변수 전달

함수에 여러 개의 값을 전달할 수 있습니다.

```
int PlusNumber(int a, int b)  // 두 개의 매개변수
{
    int c = a + b;
    return c;
}
```

사용 예시

```
void Start()
{
    int result = PlusNumber(3, 97);  // 3과 97을 전달
    Debug.Log(result);                // 100 출력
}
```

- **동작 과정**
    1. `PlusNumber(3, 97)` 호출
    2. `a = 3`, `b = 97`로 설정
    3. `c = 3 + 97` 계산 → 100
    4. 100 반환

실전 활용

```
// 두 점 사이의 거리 계산
float Distance(float x1, float y1, float x2, float y2)
{
    float dx = x2 - x1;
    float dy = y2 - y1;
    return Mathf.Sqrt(dx * dx + dy * dy);
}

// 플레이어 정보 출력
void ShowPlayerInfo(string name, int level, int hp)
{
    Debug.Log("이름: " + name);
    Debug.Log("레벨: " + level);
```

```
    Debug.Log("체력: " + hp);
}
```

---

9. 사칙연산 함수 만들기

덧셈, 뺄셈, 곱셈, 나눗셈 함수

```
int PlusNumber(int a, int b)
{
    return a + b;  // 더하기
}

int MinusNumber(int a, int b)
{
    return a - b;  // 빼기
}

int MultiplyNumber(int a, int b)
{
    return a * b;  // 곱하기
}

int DivideNumber(int a, int b)
{
    return a / b;  // 나누기
}
```

사용 예시

```
void Start()
{
    Debug.Log(PlusNumber(3, 97));      // 100
    Debug.Log(MinusNumber(97, 3));     // 94
    Debug.Log(MultiplyNumber(97, 3));  // 291
    Debug.Log(DivideNumber(97, 3));    // 32 (정수 나눗셈)
}
```

함수 네이밍 팁

함수 이름만 봐도 무엇을 하는지 알 수 있게 지어야 합니다.

- ☑ 좋은 예: `PlusNumber`, `CalculateDamage`, `GetPlayerHP`
- ✖ 나쁜 예: `func1`, `test`, `abc`

함수 작성 체크리스트

1. **함수 이름**: 기능을 명확히 표현하는가?
2. **매개변수**: 필요한 값들을 모두 받는가?
3. **반환값**: 결과를 제대로 반환하는가?
4. **재사용성**: 다른 곳에서도 사용할 수 있는가?

🎯 학습 순서 추천

1. **1단계**: 변수와 조건문 기초 → 프로그래밍의 기본기
2. **2단계**: 비교 연산자와 논리 연산자 → 조건을 만드는 방법
3. **3단계**: 상수와 switch-case → 고급 조건문
4. **4단계**: 열거형 (enum) → 데이터 구조화
5. **5단계**: 함수 기초 → 코드 재사용
6. **6단계**: 반환값과 매개변수 → 함수 심화

💡 핵심 개념 요약

변수 타입

- `int`: 정수 (100, -5)
- `float`: 실수 (3.14f)
- `string`: 문자열 ("Hello")
- `bool`: 불린 (true/false)

Unity 생명주기

- `Start()`: 게임 시작 시 1번 실행
- `Update()`: 매 프레임마다 실행

조건문

- `if`: 단일 조건
- `if-else`: 두 가지 경우
- `if-else if-else`: 여러 경우
- `switch-case`: 특정 값 비교

함수

- **반환 타입**: `void` (없음), `int`, `string` 등
- **매개변수**: 함수에 전달할 값들
- **return**: 결과를 반환

🛠️ 실전 팁

함수 만들기 (Visual Studio)

1. 코드 블록 선택
2. 마우스 오른쪽 클릭

3. "빠른 작업 및 리팩토링" 선택 (또는 `Ctrl + .`)
4. "메서드 추출" 선택

Debug.Log 활용

- 코드 실행 확인
- 변수 값 확인
- 오류 찾기
- 게임 진행 상황 추적

Unity Inspector 활용

- `public` 변수는 Inspector에서 보임
- 게임 실행 중에도 값 변경 가능
- `enum`은 드롭다운 메뉴로 선택 가능

코드 작성 시 주의사항

- 변수 이름은 의미 있게 지을 것
- 함수는 하나의 기능만 수행하도록
- 주석을 적절히 사용할 것
- 코드를 작은 단위로 나눌 것

연습 문제

각 개념을 배운 후 다음을 시도해보세요:

1. **변수 연습**: 플레이어의 이름, 레벨, 경험치를 변수로 만들기
2. **조건문 연습**: 경험치가 100 이상이면 레벨업 메시지 출력
3. **함수 연습**: 데미지를 계산하는 함수 만들기
4. **enum 연습**: 아이템 타입을 enum으로 만들고 switch-case로 처리하기

행운을 빕니다! 꾸준히 연습하면 프로그래밍이 쉬워집니다! 🚀