

배열

배열 (Array)

- 한 번에 많은 양의 데이터를 다룰 필요가 있을 때 사용
- 한 개의 배열에는 한 개의 데이터 타입만 사용 가능

```
// 변수를 여러개를 하나의 배열로 처리
int num1, num2, num3, num4, num5, num6, num7, num8;
int[] nums = new int[8];
```

```
// 입력되는 데이터의 크기를 알 수 없을 때, 배열로 처리
int inputCount = 10; // 사용자가 입력했다고 가정
int[] inputData = new int[inputCount];
```

```
// 배열의 각 요소에 접근, Index는 0부터 시작
inputData[0] = 20;
int oneOfData = inputData[0];
```

- * **new**: 인스턴스 생성 "키워드"
- * 인스턴스는 이후 클래스에서 다룸

<https://learn.microsoft.com/ko-kr/dotnet/csharp/language-reference/keywords/>

배열의 종류

```
int[] array1 = new int[5];  
int[] array2 = { 1, 2, 3, 4, 5, 6 };  
int[,] multiDimensionalArray1 = new int[2, 3];  
int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };  
int[][] jaggedArray = new int[6][];  
jaggedArray[0] = new int[4] { 1, 2, 3, 4 };  
jaggedArray[1] = new int[2] { 5, 6 };
```

* 큰 크기의 배열은 이후 배열 **반복문**과 함께 사용

문자열

유니코드

- 전세계의 언어를 표현하기 위해 만들어진 문자 포맷
 - <https://www.unicode.org/charts/PDF/UAC00.pdf>
 - 한글 같은 경우 한 글자에 24bit를 소비
- 이전에는 영어, 숫자, 특수문자만 표현 가능한 ASCII 코드를 사용
 - <https://www.ascii-code.com/>
 - 한 글자에 8bit를 소비
- 단순히 글자-숫자 형태의 표기법인 유니코드를 실제 컴퓨터에서 사용 가능한 형태로 코드화 한 것이 UTF-8, UTF-16 같은 문자 인코딩

문자와 문자열

- 문자 (**char**)
 - 단일 문자, 유니코드 16bit 문자
 - 작은 따옴표로 표현: 'a', '+', 'G'
- 문자열 (**string**)
 - 여러 개의 문자의 집합, 유니코드 문자열
 - 큰 따옴표로 표현: "문자열 입니다.", "1234!@#\$", "A"
 - 덧셈 연산 가능: `string myName = "Leader_" + "John" + " " + 999.ToString();`
 - 다양한 문자열 제어 함수를 사용하여 다방면으로 활용 가능

문자열 관련 함수

함수명	기능
IndexOf()	찾고자 하는 지정된 문자 또는 문자열의 위치의 인덱스를 반환
LastIndexOf()	찾고자 하는 지정된 문자 또는 문자열의 위치를 뒤에서 부터 찾고 인덱스를 반환
Contains()	문자열에 지정된 문자열이 존재하면 True, 아니면 False
Replace()	지정한 문자열이 다른 지정된 문자열로 모두 바뀐 문자열을 반환
Insert()	지정된 위치에 지정된 문자열을 삽입한 문자열을 반환
Remove()	지정된 인덱스부터 지정된 수 만큼 삭제된 문자열을 반환
Split()	지정된 문자를 기준으로 분리된 문자열들을 배열로 반환
SubString()	지정된 위치로부터 지정된 수 만큼의 문자로 이루어진 문자열을 반환

<https://learn.microsoft.com/ko-kr/dotnet/csharp/programming-guide/strings/>

실습. 문자열 및 배열

- 문자열 관련 함수의 사용법을 직접 검색하여 아래 기능을 작성
 1. 10칸 크기의 문자열 배열을 생성
 2. 배열의 각 요소에 문자열 관련 함수를 하나씩 적용하여 결과값을 저장
 1. IndexOf(): "동해 물과 백두산이" 에서 "백두산"의 검색 결과를 저장
 2. LastIndexOf(): "토요일에 먹는 토마토" 에서 "토"를 검색하여 결과를 저장
 3. Contains(): "질서 있는 퇴장" 에서 "퇴"를 검색하여 결과를 저장
 4. Replace(): "그 사람의 그림자는 그랬다." 에서 "그"를 "이"로 변경
 5. Insert(): "삼성 갤럭시" 에서 "삼성" 과 "갤럭시" 사이에 "애플"을 넣기
 6. Remove(): "오늘은 왠지 더 배고프다" 에서 "더"를 삭제
 7. Split(): "이름, 나이, 전화번호" 를 ","를 기준으로 분리하여 저장 (배열 3칸 소모)
 8. SubString(): "우리 나라 만세" 에서 "나라" 만 꺼내서 저장
 3. 배열의 모든 요소를 TextBox에 출력
 4. 결과를 push 하고 Repo. 링크를 슬랙 댓글로 제출

실습. 문자열

- "멈추지 않는 한 얼마나 천천히 가는지는 중요하지 않다. -공자"

1. 위 문구를 문자열 함수만을 이용하여 아래 과제를 수행
 1. IndexOf() 또는 LastIndexOf()를 사용하여 특수 문자를 검색하고, SubString() 또는 Remove()를 사용해 "-공자" 부분을 삭제
 2. IndexOf() 또는 LastIndexOf()를 사용하여 단어를 검색하고, Split()을 사용하여 "얼마나", "천천히", "가는지" 세 개 단어로 나누어 배열의 요소에 각각 저장
 3. "." 과 "-"를 제거하고, 모든 공백 문자를 ","로 바꾸기
2. 각 결과를 TextBox에 모두 출력
3. 결과를 push 하고 Repo. 링크를 슬랙 댓글로 제출

함수

함수, 메소드

- 함수

- 수학에서의 함수와 같이 **입력 -> 처리 -> 반환(return)** 으로 이루어진 코드 형태
- 반복적으로 사용하는 코드를 함수로 만들어서 효율적으로 처리
- 입력 자료형, 출력 자료형을 지정해야 함
- 입력 변수의 수는 제약이 없음, 0개도 가능
- 반환(**return**)은 한 개만 가능하지만, 변수, 배열, 클래스 등 다양한 형태를 출력 가능
 - 반환되는 값이 없는(void) 함수도 만들 수 있음
- 반환된 값은 같은 자료형으로 복사 가능

- 메소드

- 클래스 안에 선언된 함수
- 클래스를 배울 때 조금 더 자세히 다룰 예정

함수의 선언

반환 값(return 값)의 자료형

0 references

```
int Add(int x, int y)
{
    int a = x;
    int b = y;
    int result = a + b;

    return result;
}
```

입력 변수의 자료형

함수의 이름(식별자), 보통 첫 글자를 대문자로 작성

* 변수 a, b, result는 Add() 함수가 끝나는 시점에서 Scope를 벗어나기 때문에 함수 밖에서는 사용 불가

반환 키워드

함수의 선언

1 reference

```
void Nothing()  
{  
  ...  
}
```

`void`는 함수의 반환 값이 없다는 의미
변수의 자료형으로 사용은 불가능

입력 값이 존재하지 않을 수도 있음

- * `void` 타입 함수를 언제 사용하는 지는 클래스를 배울 때 다룸
- * 입력 값이 없는 경우는 입력 값과 상관없이 정해진 값을 출력하는 경우
(마찬가지로 클래스를 배울 때 배우기로)

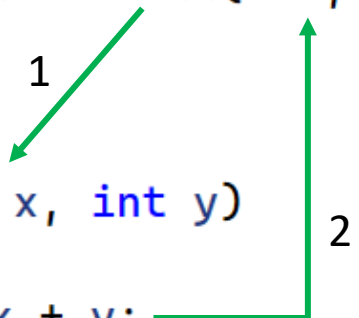
함수 사용

```
1 reference
public Form1()
{
    InitializeComponent();

    /* ... */

    int num = 200;
    int result = Add(100, num);
}

1 reference
int Add(int x, int y)
{
    return x + y;
}
```



컴파일러가 Add() 함수를 만나면
Add() 함수가 정의된 부분으로 이동하여
코드를 실행

return 된 값이 Add() 함수와 치환됨
→ Add(100, num)이 300으로 바뀜

함수 사용

1 reference

```
public Form1()
```

```
{
```

```
    InitializeComponent();
```

```
    /* ...
```

```
    int num = 200;
```

```
    int result = Add(100, num);
```

```
}
```

자료형이 같다면 값을 직접 입력하는 것도 가능

변수를 입력하는 것도 가능

Add() 함수의 출력 자료형이 int 이기 때문에 int 형 변수로 결과 값을 복사

1 reference

```
int Add(int x, int y)
```

```
{
```

```
    return x + y;
```

```
}
```

함수는 Form1() 밖에 선언

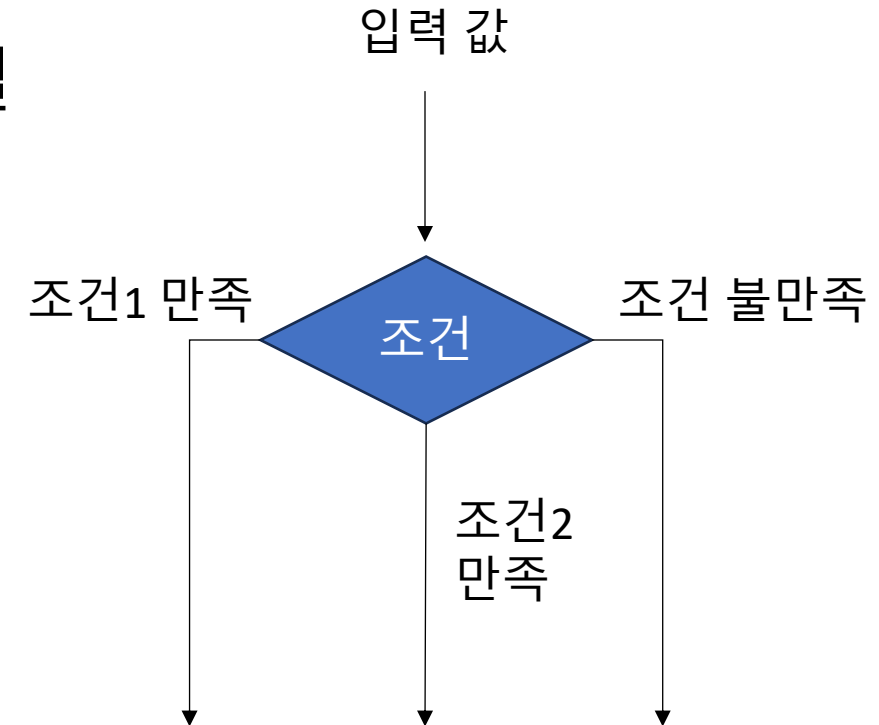
실습. 함수

- 아래 내용을 참고하여 적당한 이름의 함수를 구현
 1. int 형 숫자 두 개를 입력 받기
 2. 첫 번째 입력 값을 두 번째 입력 값으로 나눔
 3. 나뉜 값은 배열의 첫 번째 요소에 저장
 4. 나머지 값은 배열의 두 번째 요소에 저장
 - 나머지 연산은 %를 이용 (필요시 검색)
 5. 위 배열을 반환
- 1. Form1()에서 위 함수를 사용하고 TextBox에 결과 값을 출력
- 2. 결과를 push 하고 Repo. 링크를 슬랙 댓글로 제출

조건문

조건문

- 조건문은 임의의 조건을 기반으로 입력 값을 판별
- 판별 결과에 따라 코드를 동작 또는 건너뛴다
- if 문
 - 조건에 따른 분기점이 적을 때 사용 (y/n)
 - 겹겹이 사용하는 것은 코드 복잡도를 높임 (부정적)
- switch 문
 - 조건에 따른 분기점이 많을 때 사용 (1:n)
 - 보통 enum(열거형식)과 함께 사용



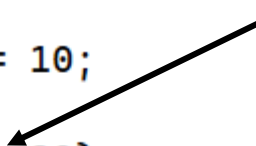
if 문

- **if (조건) { }**
 - 조건 비교 결과 true면 중괄호 내부의 소스코드를 실행
 - 조건 비교 결과 false면 중괄호를 건너뛴
- **else if (조건) { }**
 - if 문 또는 if else 문을 사용한 뒤에 바로 이어서 사용 가능
 - 앞선 if 문 또는 if else 문의 조건이 false일 경우에만 작동
- **else { }**
 - if 문 또는 if else 문을 사용 후 맨 마지막에 사용 가능
 - 위에서 사용한 if 및 if else 문의 조건 비교가 모두 false 일 경우 실행

if 문

```
int inputNum = 10;
if (inputNum > 20)
{
    // inputNum은 20보다 크지 않음: false
}
else if (inputNum < 5)
{
    // inputNum은 20보다 크지 않고, 5보다 작지 않음: false
}
else if (inputNum == 8)
{
    // inputNum은 20보다 크지 않고, 5보다 작지 않고, 8도 아님: false
}
else
{
    // 위 조건이 모두 false 일 경우 실행
}
```

비교 연산자



논리 연산자

- 조건과 조건을 비교하는 연산을 수행

```
// (조건1) OR (조건2)  
bool compared = (inputNum > 10 || inputNum < 5);
```

연산자	기능	사용법	의미
&&, &	AND	(조건A) && (조건B)	조건A, 조건B가 모두 true면 true
,	OR	(조건A) (조건B)	조건A, 조건B 중 하나라도 true면 true
!	NOT	!(조건A)	조건A가 true면 false

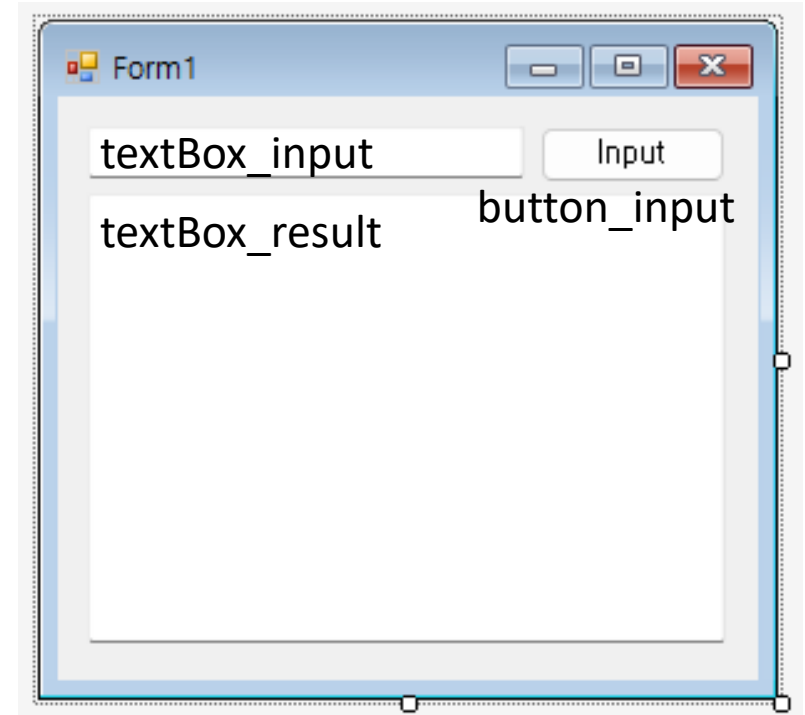
* &&는 조건A를 먼저 확인 후 결과를 결정, &는 무조건 조건A 및 조건B를 모두 확인 후 결정
ex) $(4 < 2) \ \&\& \ (5 > 1)$ 일 경우 $(4 < 2)$ 에서 이미 false가 결정되므로 $(5 > 1)$ 은 계산하지 않음

실습. if 문

- 동전 던지기(앞 면 또는 뒷 면) 함수를 작성
 1. 함수 이름을 적당하게 짓기
 2. 출력 자료형은 bool
 3. 입력은 bool 형 1개
 4. "난수 생성"을 검색하여 난수 생성 방법을 학습하고 int형 난수를 생성
 5. 생성된 난수와 % 연산을 이용, 연산 결과로 0 또는 1만 값이 나오도록 작성
 6. 입력 값과 연산 결과를 비교하여 둘이 같으면 true를 반환, 다르면 false를 반환
 - 1 = true, 0 = false 라고 가정
- 함수에 true 또는 false를 입력하고 반환된 결과에 따라 "승리" 또는 "패배"로 TextBox에 표시
- Push 후 Repo. 링크를 슬랙 댓글로 제출

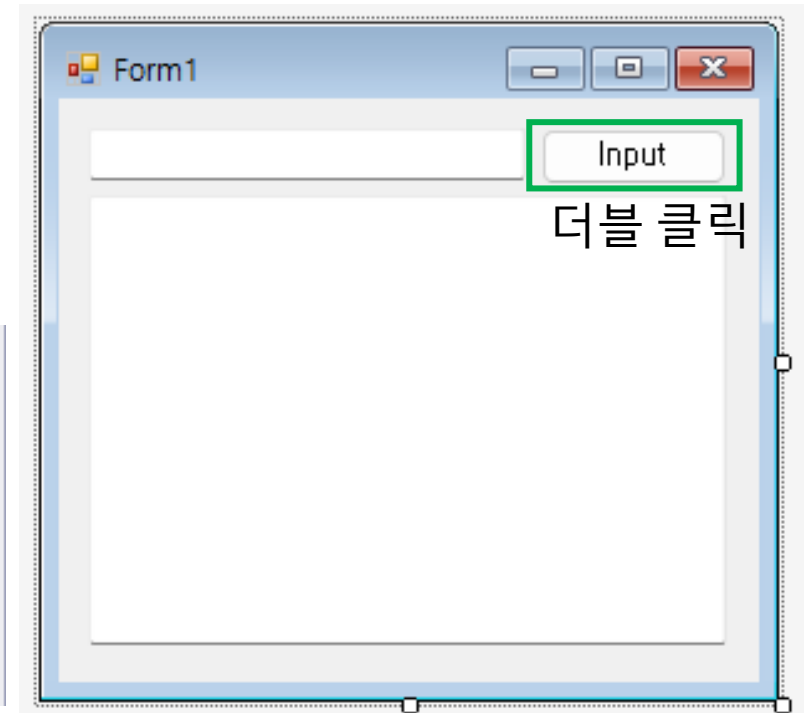
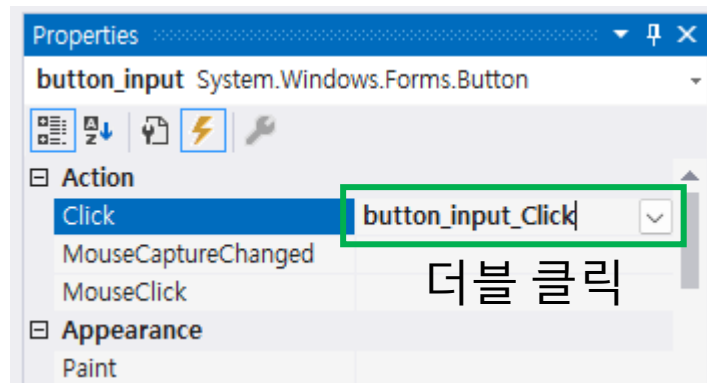
사용자 입력 받기

- WinForm에서 제공하는 기본 컨트롤을 이용하여 사용자 입력을 받기
- textBox_input
 - TabIndex: 0
- button_input
 - TabIndex: 1
 - Text: Input
- textBox_result
 - TabIndex: 2
 - Multiline: True
 - ReadOnly: True (결과 확인 용, 읽기 전용으로)
 - BackColor: ControlLightLight



사용자 입력 받기

- Designer에서 button_input을 더블 클릭
- 또는 Properties > Events(번개 아이콘) > Action > Click 우측 빈칸을 더블 클릭
- **button_input_click(...)** 함수가 자동 생성됨
- button_input 클릭시 위 함수 내부에 작성한 소스코드가 실행됨

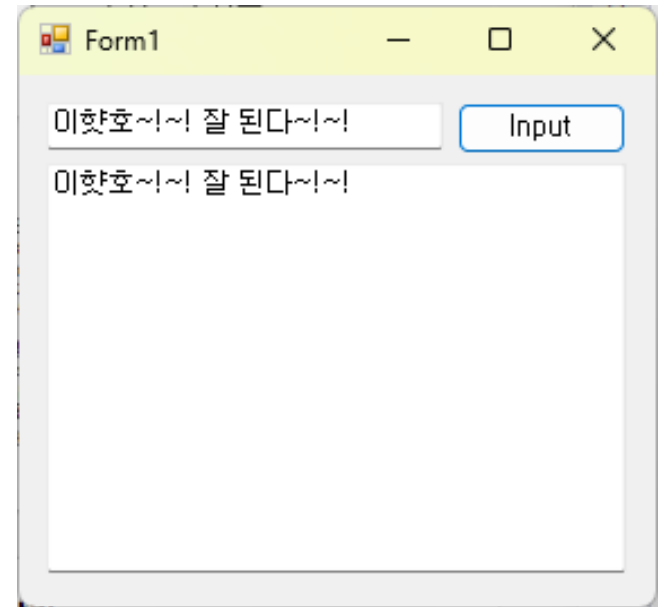


사용자 입력 받기

- textBox_input 컨트롤에 작성한 문자열을 textBox_result 로 복사

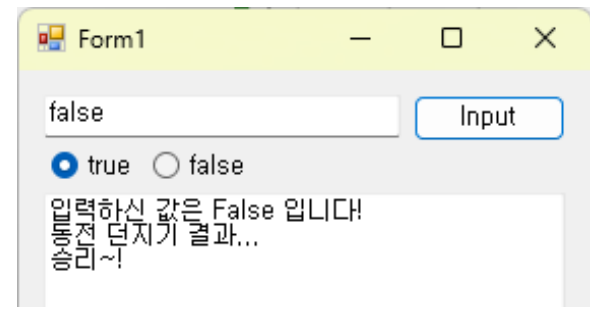
1 reference

```
private void button_input_Click(object sender, EventArgs e)
{
    // textBox_input.Text의 문자열을
    // textBox_result.Text로 복사
    textBox_result.Text = textBox_input.Text;
}
```



실습. 사용자 입력

- 실습. if 문 에서 사용했던 동전던지기 함수를 사용자 입력을 받아서 처리
 1. C# WinForm RadioButton 사용 방법을 검색하여 학습
 2. 아래 이미지와 같이 true 또는 false 라디오 버튼을 추가
 3. textBox_input에 값을 입력하지 않아도 라디오 버튼의 상태에 따라 true 또는 false 값이 입력되도록 처리
 4. 단, textBox_input에 입력된 문자열이 있다면 우선적으로 처리
 5. 사용자가 true 또는 false 이외의 값을 입력했다면 적당한 에러 메시지를 textBox_result에 출력
 6. Push 후 Repo. 링크를 슬랙 댓글로 제출



switch 문

- 조건에 따른 분기점이 많을 때 사용
- 숫자보다는 문자열을 이용
- **switch** (변수)
 - 조건을 확인할 변수를 입력
 - 조건을 만족하는 case 의 코드를 실행
 - 모든 case 및 default는 break; 로 끝나야 함
- **case** 값 :
 - switch로 입력 받은 변수가 특정한 값을 갖는지 확인
- **default** :
 - if 분의 else 와 같이 위 case에 모두 해당하지 않는다면 실행

switch 문

```
string animal = "Cat";

switch (animal)
{
    case "Dog":
        // animal == "Dog" 일때 실행되는 코드
        break;

    case "Cat":
        // animal == "Cat" 일때 실행되는 코드
        break;

    default:
        // 위 case들에 모두 해당되지 않는다면 실행
        break;
}
```

case 2 에 해당하는 코드가 실행됨



enum (열거형)

- 소스코드 상에서 단순히 숫자를 텍스트로 치환
- 변수처럼 값이 저장되지는 않음
- enum 간 비교 연산도 가능

6 references

```
enum Food
{
    Pizza,          // 0
    Burger,         // 1
    Pasta,          // 2
    Kimchi = 100    // 값 지정 가능
}
```

```
Food food = Food.Kimchi;

if(food > Food.Pizza)
{
    // Pizza = 0, Kimchi = 100 이므로 true
}
```

enum (열거형)

```
Food food = Food.Kimchi;
```

```
switch (food)
```

```
{
```

```
    case Food.Pizza:
```

```
        break;
```

```
    case Food.Burger:
```

```
        break;
```

```
    case Food.Pasta:
```

```
        break;
```

```
    case Food.Kimchi:
```

```
        // 간단하게 정수로 캐스팅해서 사용 가능
```

```
        int price = (int)Food.Kimchi;
```

```
        break;
```

```
}
```

Parse처럼 자료형의 변환이 이루어지는 것은 아님
명시적으로 int 형 데이터 임을 표시해주는 것일 뿐

실습. Enum + if 문

- 가위바위보 게임
 1. 가위, 바위, 보에 대한 버튼을 3개 생성
 2. 셋 중 아무 버튼이나 “클릭하면” 컴퓨터도 가위, 바위, 보 중 하나를 랜덤하게 선택.
 3. 컴퓨터가 무엇을 선택했는지 화면에 표시하고, 사용자가 선택한 것과 비교하여 승패를 가름.
 - ❖ 승리한 쪽에는 1점 추가 / 무승부는 변화 없음.
 4. 사용자와 컴퓨터의 점수를 표시하고 먼저 3점을 얻는 쪽이 최종 승리
 5. 한쪽이 3점을 얻는 순간 점수 초기화 하기.

실습. Enum + if 문

- 가위바위보 게임

- 조건 (Hint)

1. 가위 바위 보 - enum으로 표현
2. 승부 결과 (승리 무승부 패배) - enum으로 표현
3. 승부 판단 로직(함수) 생성
4. 게임 전체 관리 로직(함수) 생성
5. 승부 완료 후, 점수 초기화 로직(함수) 생성
6. Push 후 Repo. 링크를 슬랙 댓글로 제출

실습. 결과 예시 화면

게임을 시작해보세요!

용사	마왕	
0	0	
가위	바위	보

마왕의 선택: scissors
비겼습니다

용사	마왕	
0	0	
가위	바위	보

실습. 결과 예시 화면

마왕의 선택: scissors
용사의 승리!

용사	마왕
<input type="text" value="1"/>	<input type="text" value="0"/>

가위

바위

보

마왕의 선택: scissors
마왕의 승리...! 으아아 분하다!!

용사	마왕
<input type="text" value="2"/>	<input type="text" value="1"/>

가위

바위

보

마왕의 선택: rock
마왕의 승리...! 으아아 분하다!!
마왕이 우주를 정복했다...!
(;_ಡ_)

점수 초기화

용사	마왕
<input type="text" value="0"/>	<input type="text" value="0"/>

가위

바위

보

goto 점프문

* 잘못된 사용은 건잡을 수 없는 혼란을 유발시킴 (가능한 사용 지양)

```
int selection = 0;  
textBox_print.Text += "Start\r\n";
```

Location1: // goto로 이동할 곳을 지정

```
textBox_print.Text += "Location1\r\n";
```

```
if (selection == 0)  
{
```

```
    textBox_print.Text += "if\r\n";
```

```
    selection = 1;
```

```
    goto Location1; // 소스코드에서 Location1: 을 찾아서 이동
```

```
}
```



Form1

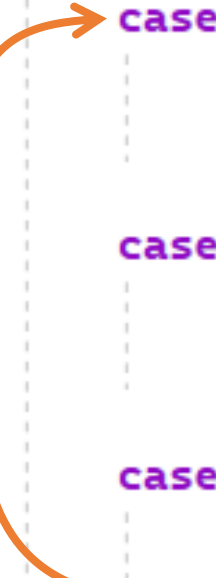
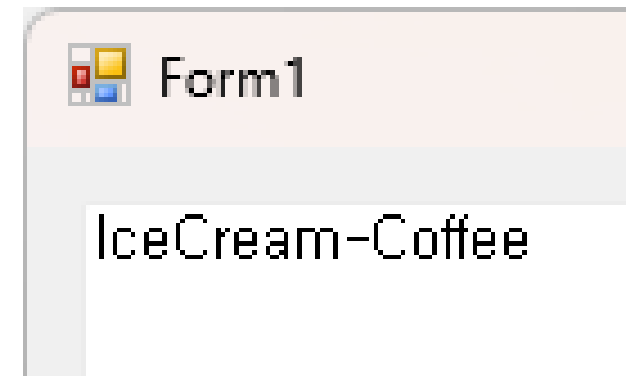
```
Start  
Location1  
if  
Location1
```

goto & switch

```
switch (choice) // 3
{
    case Plain:
        textBox_print.Text += "Coffee";
        break;

    case WithMilk:
        textBox_print.Text += "Milk-";
        goto case Plain;

    case WithIceCream:
        textBox_print.Text += "IceCream-";
        goto case Plain;
}
```

An orange arrow originates from the 'goto case Plain;' statement in the 'WithMilk' case and points to the 'case Plain:' label. Another orange arrow originates from the 'goto case Plain;' statement in the 'WithIceCream' case and also points to the 'case Plain:' label.

실습. switch 문

- 아래 요구사항을 만족하는 프로그램을 switch, enum을 사용하여 작성
 - 요구사항 이외의 요소(버튼이 꼭 필요한가? 등)는 자율에 맡김
1. 사용자는 요일을 입력할 수 있음
 2. 사용자가 요일 이외의 문자열을 입력하면 오류를 출력
 3. 각 요일별로 재미난 메시지를 화면에 출력
 - 예) 월요일 입력: "심근경색, 월요일 아침이 가장 위험! 출근을 안 해야..."
 4. Push 후 Repo. URL을 슬랙 댓글로 제출

반복문

반복문

- 특정한 조건 또는 횟수를 지정하여 원하는 만큼 코드를 반복 실행
- 반복문을 중첩하여 사용할 경우 퍼포먼스가 기하급수 적으로 떨어짐
- **for문**
 - 반복 횟수를 기준으로 함
 - 반복 횟수를 배열의 Index로 활용하는 경우가 많음
- **foreach문**
 - 배열 순차 탐색에 특화 (편리성을 위해 기능을 희생)
- **while문**
 - 반복 조건을 기준으로 함
 - 반복 조건이 허락한다면 무한히 반복하는 용도로도 사용

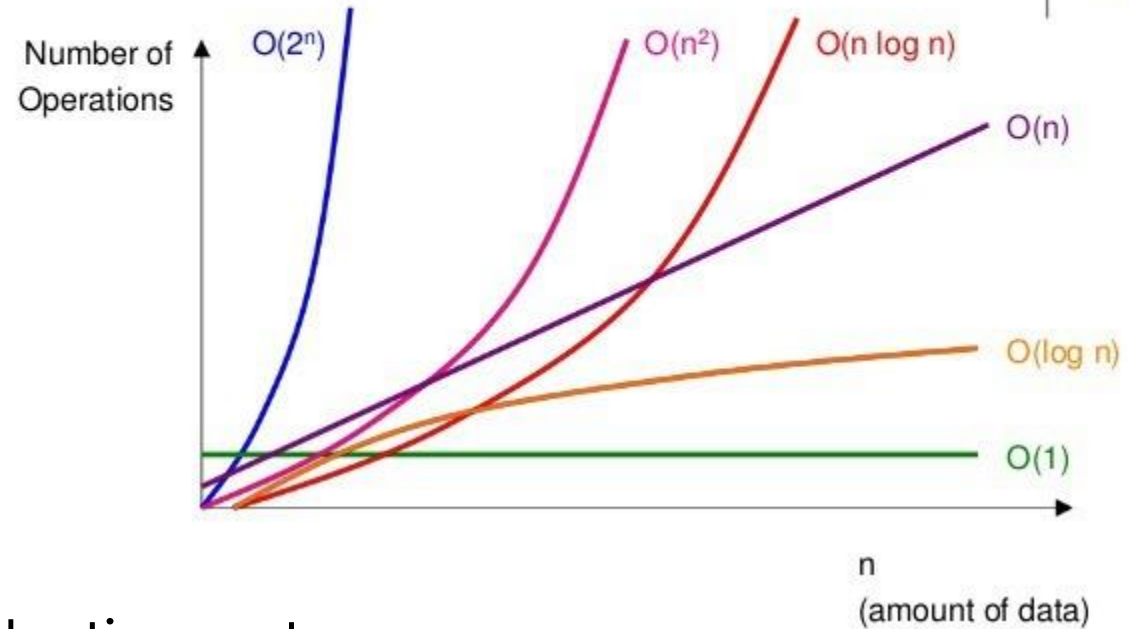
Big-O 표기법

- 알고리즘의 시간 복잡도를 판단하는 척도, $O(\text{값})$ 형태로 표기
- 알고리즘 최악의 실행 시간을 표기하므로 몇 가지 특징을 가짐
- 상수 및 계수를 무시
 - $O(5+N)$, $O(7N)$ 은 둘 다 그냥 $O(N)$ 으로 표기
- 최고차항만 표시
 - $O(4N^2 + 2N + N^3)$ 이라고 한다면 $O(N^3)$ 만 표기

Big-O 표기법

- $O(1)$
 - Stack push, pop
- $O(\log N)$
 - Binary search tree
- $O(N)$
 - for 문
- $O(N \log N)$
 - Quick/Merge/Heap sort
- $O(N^2)$
 - 이중 for 문, Insertion/Bubble/Selection sort
- $O(2^N)$
 - 피보나치 수열

Comparing Big O Functions

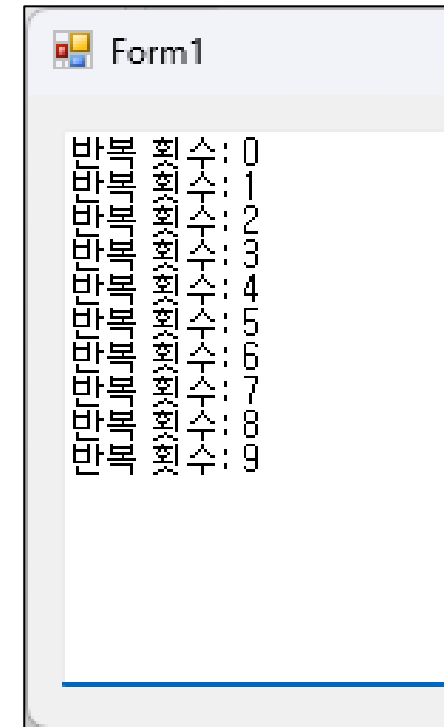


for 문

- 반복 횟수를 기준으로 함
- for(첫 시작 값; 조건; 값의 증감)

```
string message = "";  
for (int i = 0; i < 10; i++)  
{  
    message += "반복 횟수: " + i.ToString() + "\r\n";  
}  
textBox_result.Text = message;
```

++ 기호: 원래 값 +1
-- 기호: 원래 값 -1



Form1

반복 횟수	값
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

for 문

- 배열과 함께 사용할 경우

```
int array_size = 10;
int[] loopCount = new int[array_size];
for (int i = 0; i < array_size; i++)
{
    loopCount[i] = i; // index로 i를 사용
}
textBox_result.Text = loopCount[loopCount.Length - 1].ToString();
```

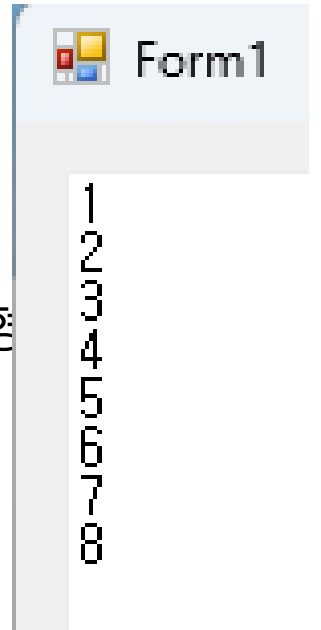
foreach 문

* 배열 전체를 순차적으로 탐색할 경우에 편리하게 사용가능

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8 };
```

배열 1개의 요소 배열 또는 인덱스로 접근 가능한 자료형
-> 열거형

```
foreach(int num in numbers)
{
    textBox_print.Text += num.ToString() + "\r\n";
}
```



실습. for문

- 가짜 성적표 만들기 🐼
 1. 학생수를 입력
 2. 입력된 학생 수 만큼 0~100점 사이의 랜덤한 점수를 생성하고 각 학생에게 점수를 할당
 3. 학생의 이름은 "학생1", "학생2", ... 와 같이 숫자만 붙여서 표기
 4. 모든 학생에 대해 "학생1의 점수: 42점" 과 같은 형태로 결과를 표시
 - 이름과 성적을 입력하면 위와 같은 문자열을 만들어주는 함수를 작성하여 사용
 5. Push 후 Repo. 링크를 슬랙 댓글로 제출

while 문

- 조건을 기준으로 반복
- 무한 반복을 수행할 경우 자주 이용됨
- while(조건)

```
int count = 0;
while(count < 100)
{
    // 반복할 소스코드
    count++;
}
```

```
bool run = true;
while (run)
{
    // 반복할 소스코드

    if("반복을 종료하고 싶다면" == "")
    {
        run = false;
    }
}
```

break, continue

- break: 반복문 안에서 탈출
- continue: 현재 반복만 스킵하고 다음 반복 진행

```
for (int i = 0; i < 10; i++)
{
    if (i == 5)
    {
        break;
    }
    textBox_result.Text += i.ToString() + "\r\n";
}
// 출력 결과 : 1, 2, 3, 4
```

```
for (int i = 1; i <= 5; i++)
{
    if (i == 3)
    {
        continue;
    }
    textBox_result.Text += i.ToString() + "\r\n";
}
// 출력 결과 : 1, 2, 4, 5
```

while + if

```
int count = 0;
while (true) ————— 반복 조건
{
    if(count == 0)
    {
        count++;
        textBox_print.Text = "Loop Start!\r\n";
    }
    else if (count < 5)
    {
        textBox_print.Text += count.ToString();
        textBox_print.Text += "/r/n";
        count++;
    }
    else
    {
        break; // continue도 가능
    }
}
```

