



파이썬 소개와 세팅



파이썬



Python을 배우는 이유

이제는 차량용 임베디드 개발자도 파이썬을 배워야한다.

수많은 차량 데이터를 다루고,
AI 도 자동차 분야에 필수가 되었기 때문이다.

데이터와 AI를 다루는데
가장 많이 쓰이고, 많은 레퍼런스를 가진 언어가 Python이다.

파이썬 세팅을 해보자



◎ 파이썬 프로그래밍을 하기 위해 준비해야 할 것 두 가지

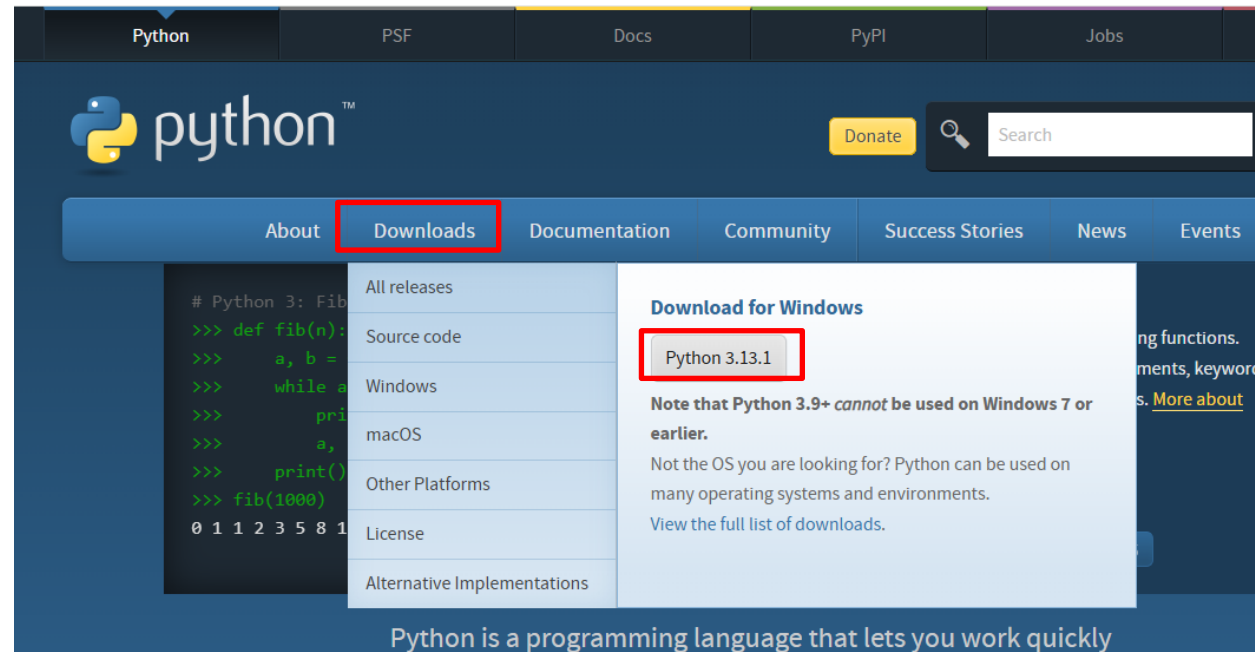
1. Python 설치 (파이썬 인터프리터 설치)
2. vscode 설치 (소스코드 작성할 수 있는 에디터)

◎ 지금부터 python과 vscode를 설치한다.



◎ 파이썬을 설치하고, 바로 동작 테스트 부터 해보자.

- python.org 에서 **Download** 버튼 클릭
- 최신버전을 다운로드 받아 설치하자.



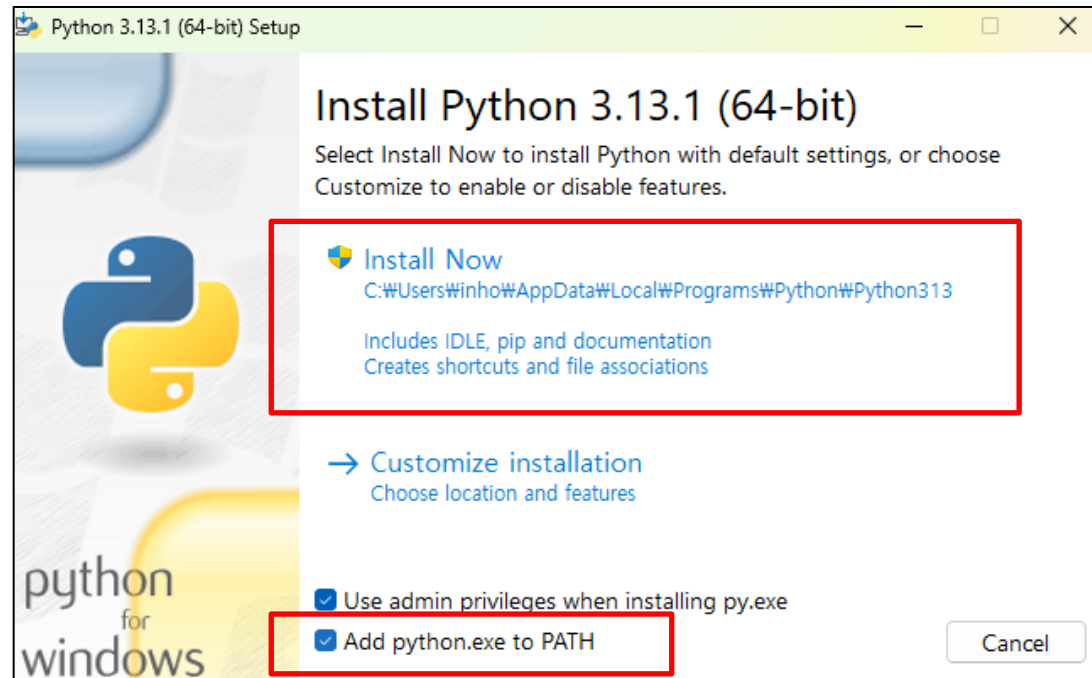
Python 설치하기



설치시 Add python.exe to PATH 를 반드시 체크할 것

Path 등록을 해야 visual studio code(우리가 사용할 툴)에서 추가 설정 없이 자동으로 Python을 인식함.

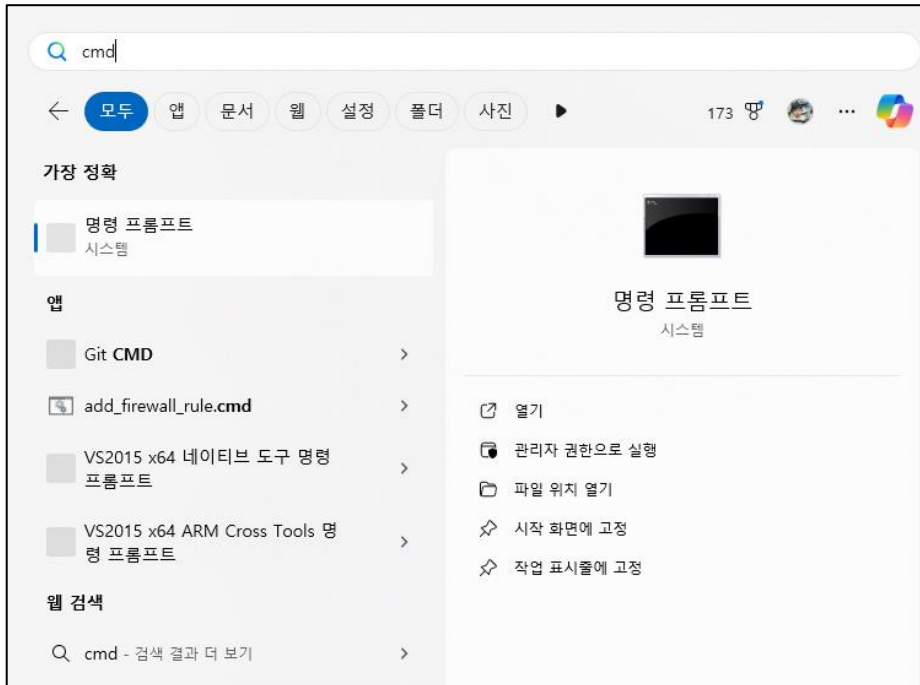
- Path 등록 안 하면 수동으로 설정해주어야하는 작업이 추가됨



Python 설치 끝



- Python 설치가 완료되었다.
- Python 설치가 잘 되었는지 확인하는 방법
 - 시작 > cmd 입력
 - python -V 입력 (V : 대문자)



```
C:\Users\inho>python -V
Python 3.13.1

C:\Users\inho>
```

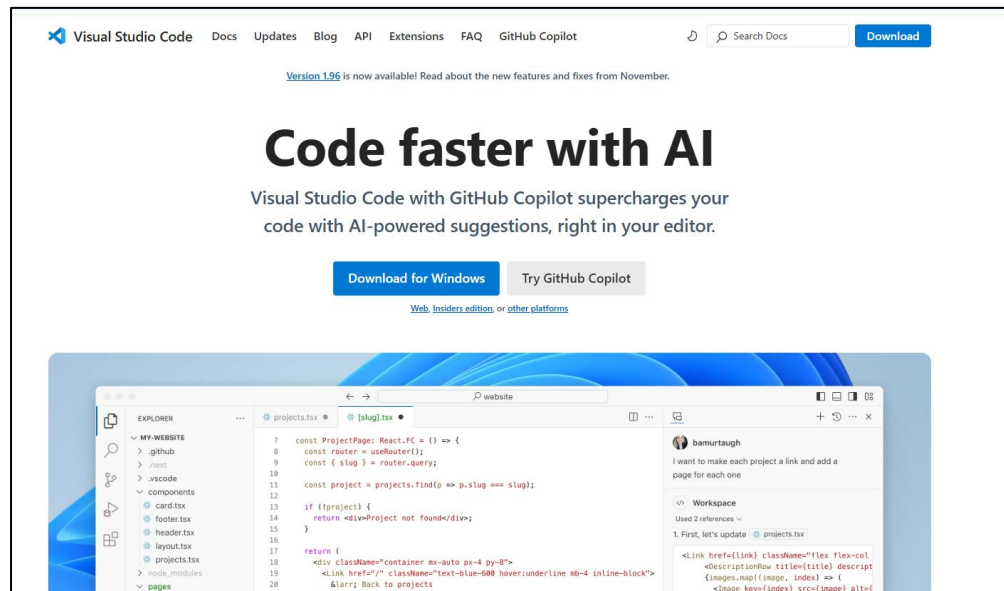
위와같이 뜨면 Python 설치가 잘 된 것이다.

Visual Studio Code 설치



Visual Studio Code를 쓰는 이유

- Python 개발자가 가장 많이 사용하는 툴 (2023년 기준)
- 마이크로소프트에서 무료로 배포
- 구글검색 : vscode 하여 다운로드 및 설치 (약 100MB)
(<https://code.visualstudio.com/>)

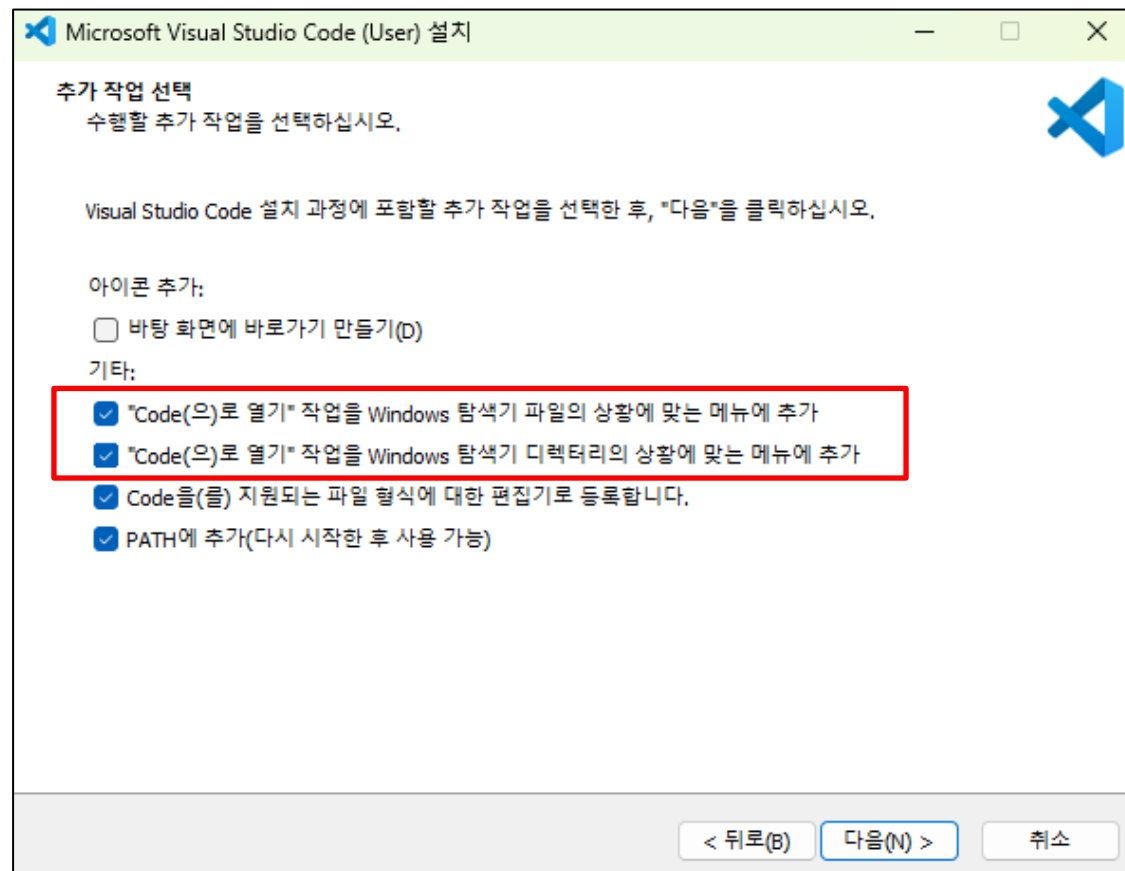


Visual Studio Code 설치하기



VSCode 설치 옵션

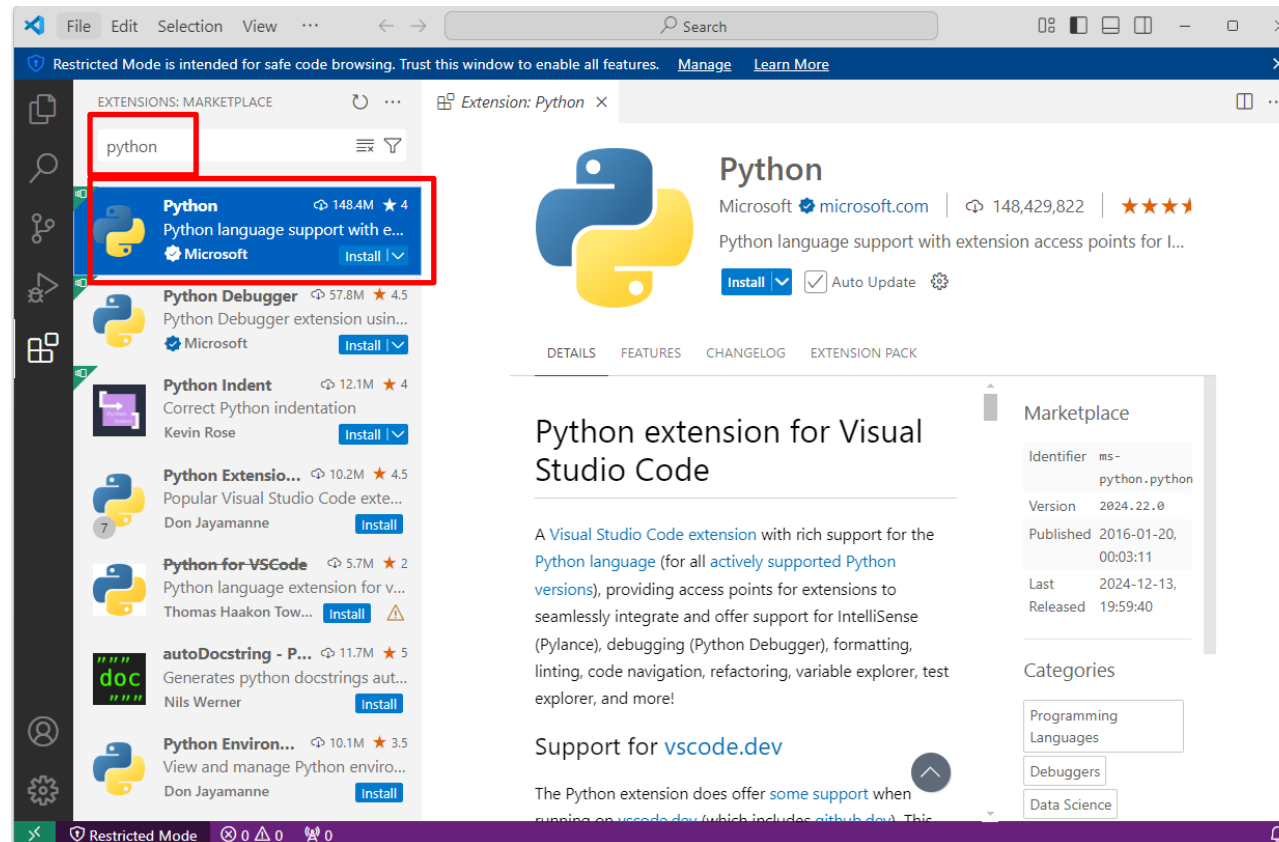
- 마우스 오른쪽 버튼을 누르면 "Code로 열기" 메뉴 추가 체크
- 이 옵션 켜면, 윈도우 화면에서 바로 마우스로 vscode 실행 가능



Python 플러그인 설치 (Extention)

◎ Extention (추가 플러그인) 설치

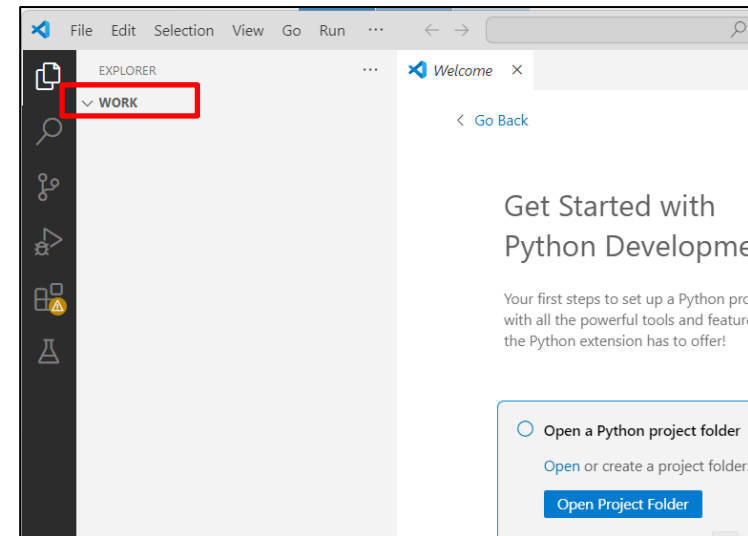
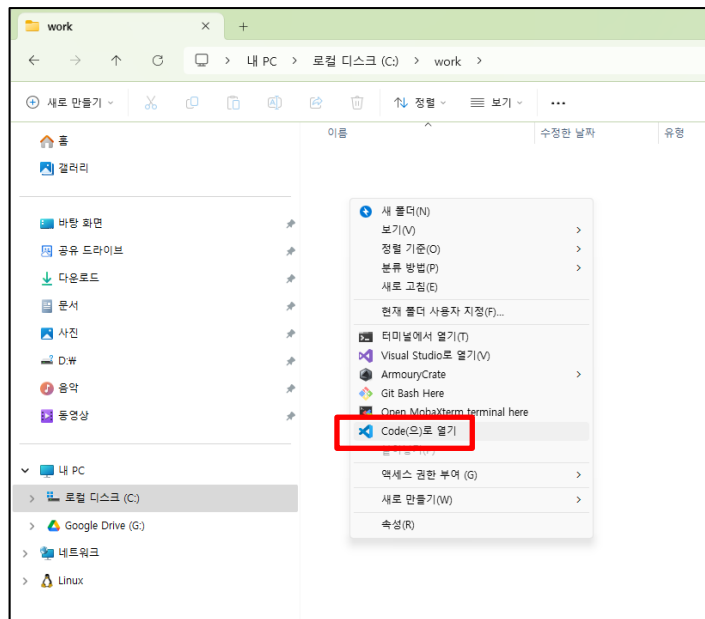
- Python 플러그인 설치 : 소스코드를 컬러로 표시 / 문법 검사 기능 / 자동완성 기능



작업 폴더 만들고 Code 실행

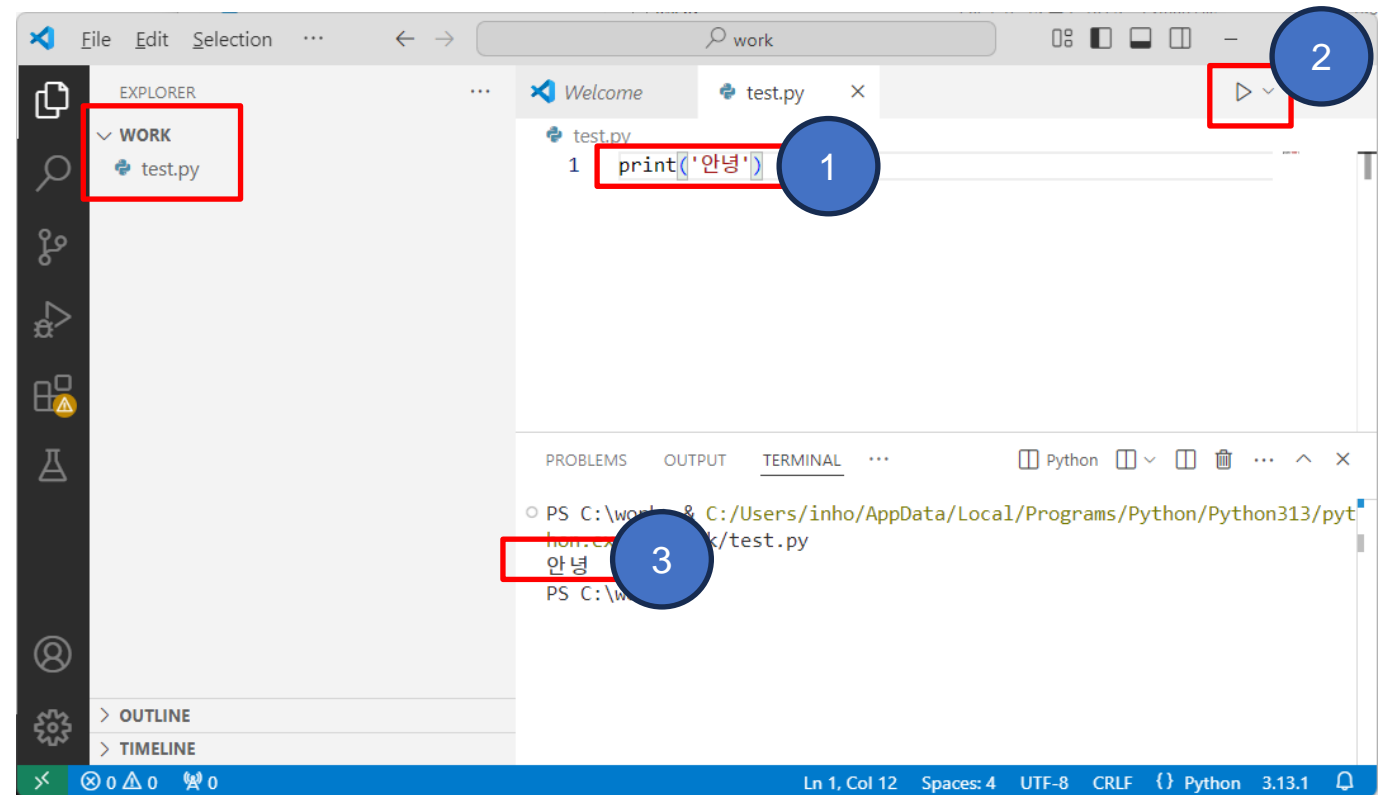
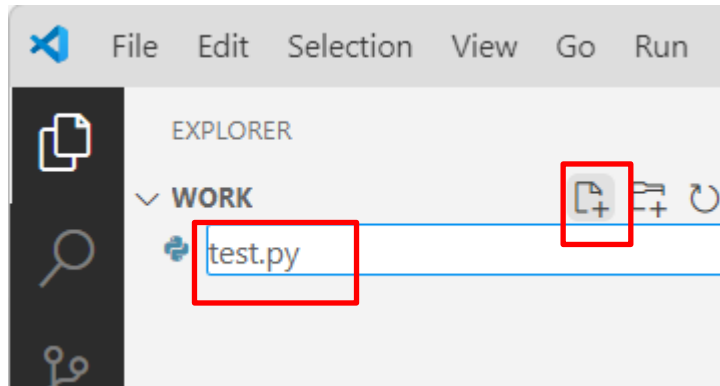


- ◎ c:\work 폴더를 하나 만든다.
 - 이 폴더는 실습 작업 폴더가 된다.
- ◎ 마우스 오른쪽 버튼을 눌러 Code로 열기를 누른다.



실행해보기

- test.py 파일을 만들고 첫 프로그램을 작성해본다.





◎ 지금까지 두 가지 프로그램을 설치하였다.

- vscode : 소스코드 작성용 프로그램, **에디터** 라고 함
- python : 파이썬 소스코드를 실행해주는 **실행기** (인터프리터라고 함)



출력





print(obj)

- 문장을 출력하고자 한다면 큰따옴표(“) 또는 작은따옴표 (')로 감싸서 표현
- 그 외 (숫자 등)을 출력하려고 한다면 그대로 작성

```
# Hello, World!  
print('Hello, World!')
```

한줄 앞에 #을 붙이면
주석이 된다.
프로그래밍 소스코드가 아닌
메모용으로 작성시 사용

print(obj1, obj2)

- 객체1과 객체2를 공백으로 분리하여 출력

```
# 1 2  
print(1, 2)
```

[도전] 본인의 이름 출력



- ◎ 본인의 이름과 나이를 출력해보시오.
 - 아래 두 가지 형태 모두 사용해서 출력해 볼것

```
# Hello, World!  
print('Hello, World!')
```

```
# 1 2  
print(1, 2)
```



한 줄에 print를 여러 번 하고 싶다면?

- `print('hello', end = '')` 을 사용
- `end` : 출력문이 끝날 때 자동으로 출력되는 문자
- `end`를 적지 않으면 Default 로 `end='\\n'`가 된다.
 '`\\n`'은 한줄 줄바꿈을 뜻한다.

```
# Hello World!
# next line
print('Hello World!')
print('next line')

# Hello World! next line
print('Hello World!', end = ' ')
print('next line')

print('hello')
```


[도전] 출력해보기



◎ 다음 그림대로 출력해보시오.

- Print함수 6개 사용
- `print('#', end=' ')` 를 사용 해볼것.

###

#7# #3#

###



특수 규칙 문자(Escape sequence)

- \n : 줄바꿈을 나타냄
- \' : 따옴표를 나타냄

```
# 안녕 나는 '컴미'야
print('안녕 나는 \'컴미\'야')

# Hello World!
# next line

print('Hello World!\nnext line')
```



변수



변수



변수(Variable) : 값을 나타내는 **이름**

변수 할당 : 변수에 값을 지정할 때 "**할당**" 이라고 한다.

```
PI = 3.14
```

할당문

"변수 PI에 값 3.14를 **할당**했다."

```
PI = '삼점일사'
```

재할당

"변수 PI에 값 '삼점일사'를 **재할당**했다."



할당문 (Assignment Statement)

```
variable = expression
```

존재하지 않는 변수라면

- 새 변수를 생성하고, 할당(assignment)함

기존에 존재했던 변수라면

- 기존 변수에 새로운 값으로 재할당(reassignment)함

[도전] 할당, 재할당 구분하기



◎ 할당과 재할당을 구분해보자

- `a = 10` #할당? 재할당?
- `b = 20` #할당? 재할당?
- `a = 50` #할당? 재할당?
- `c = 30` #할당? 재할당?

- `print(a, b, c)` 출력 결과는?



변수명 규칙

- 영문 알파벳, 언더 스코어(_), 숫자로 구성
- **대시(-) 사용 불가!**
- **숫자로 시작할 수 없음**
- 대소문자를 구분
- 파이썬 내부적으로 이미 사용중인 키워드 사용불가
- (예시 : if, True, for.. 등)

[도전] 변수 이름 가능여부 구분하기



◎ 다음 중 가능한 변수 개수를 세보자

- abc421
- 3ba
- apython
- python-3
- f7
- 9b
- _d
- show_3

변수 출력하기



변수 출력하는 두 가지 방법

```
a = 10  
print(a)
```

```
a = 20  
print(str(a) + '값이 저장되었습니다')
```

변수를 출력하는 권장 문법 : f-string



파이썬에서 변수값을 출력하는

여러가지 방법 중, f-string 이라는 방법을 가장 권장한다.

- 가장 가독성이 좋다고 평가되고

여러 출력 방법 중 성능이 가장 뛰어나게 만들어져 있기 때문이다.

```
a = 20  
print(str(a) + '값이 저장되었습니다')
```

파이썬 진영에서 권장되지 않는 출력 방식

```
print(f'{a}값이 저장되었습니다')
```

권장방법 : f-string 라는 문법 형태로 출력

변수 출력하기 (f-string)



f-string 으로 여러 변수들 값을 한꺼번에 출력하기

- 문자열에 **f 접두어**를 붙이고 표현식을 {expression} 형태로 작성하는 문법

```
name = '장상호'
age = 20
area = "Mapogu"

# 저는 장상호이고, 20세이고, Mapogu에서 살고있습니다.
print(f'저는 {name}이고, {age}세이고, {area}에서 살고있습니다.')
```

f-string 으로 소수점 출력하기



소수점 서식

- f'{variable:.소수점 자릿수f}'
- 소수점 자릿수까지 **반올림**이 된다.

```
PI = 3.14159  
  
# 3.142  
print(f'소수 3자리: {PI:.3f}')
```

[도전] 연산 후 출력하기



◎ 다음은 두 수를 더해서 출력하는 소스코드이다.

- `a = 10.25`
- `b = 20.31`
- `sum = a + b`
- `print(sum)`

◎ 위 소스코드를 다음과 같이 출력되도록 소스코드를 작성해보자.

- `a와 b를 합치면 30.6이 됩니다.`



변수의 동작원리



메모리로 저장되는 원리



$a = 10$

지금까지 a 는 10을 할당한다. 라고 학습했다.
(파이썬에서는 a 에다가 10을 **저장하는 개념이 아니다.**)

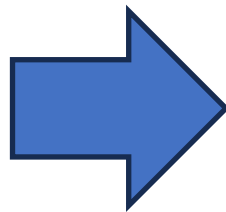
정확한 원리에 대해 알아보자.

메모리로 저장되는 원리

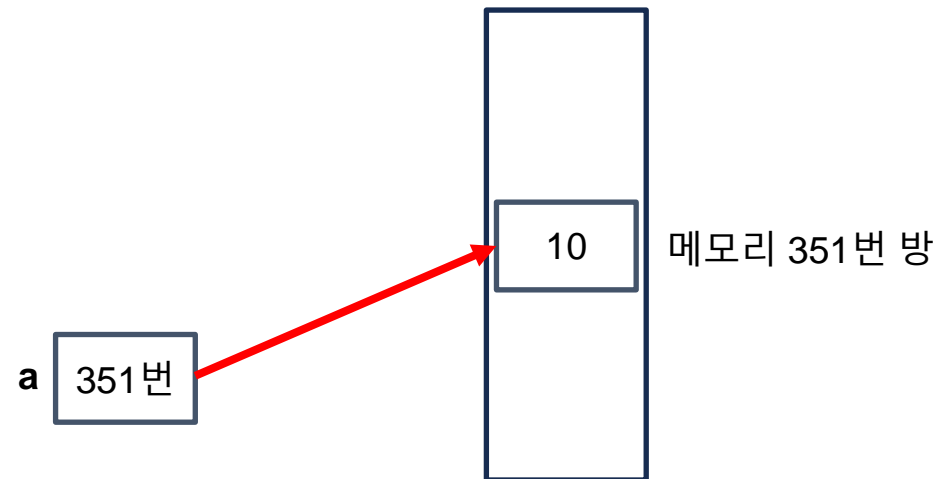


“a = 10”에 대한 동작 원리

먼저 10은 **메모리**라는
장치 어딘가에 저장된다.
메모리 351번방에 저장되었다.



a는 351번방을 가리킨다.



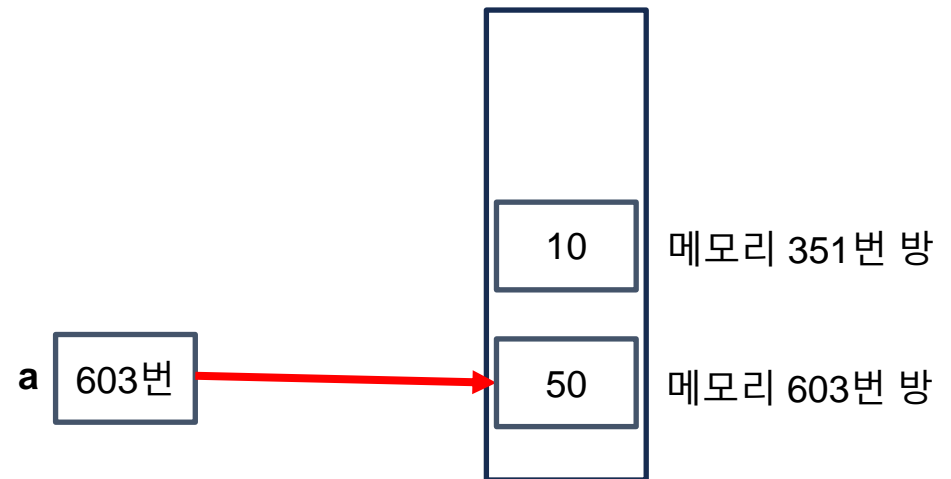
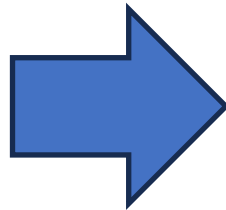
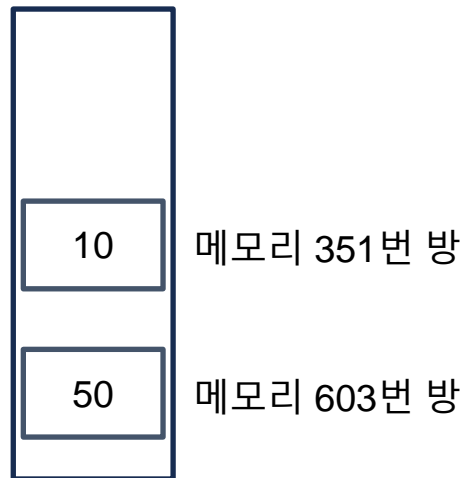
메모리로 저장되는 원리



“a = 50” 으로 다른 값이 할당된다면?

50이 **메모리** 어딘가에 저장된다.
메모리 603번방에 저장되었다.

a는 50으로 값이 수정된다.
이는 a는 603번 방을 가리킨다.



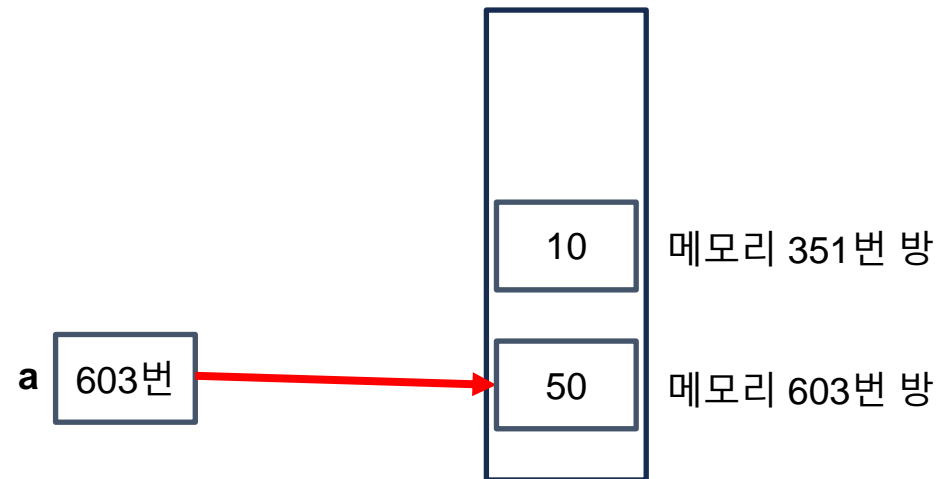
메모리로 저장되는 원리



결론

a = 50 에서, a에 50이 직접 저장되는 것이 아닌
a는 50이 저장된 위치를 가리키는 것이다.
이를 **a는 50을 참조하고 있다.** 라고 표현한다.

a는 50으로 값이 수정된다.
이는 a는 603번 방을 가리킨다.



변수



실험결과

- `id(변수)` : 변수가 참조하고 있는 객체의 메모리 주소를 나타낸다.
- 아래 그림 예시에서 **603번** 방에 해당된다.

```
a = 10
print(f'a는 메모리 {id(a)} 방에 저장되었음')

a = 50
print(f'a는 메모리 {id(a)} 방에 저장되었음')
```

```
a는 메모리 140735553209544 방에 저장되었음
a는 메모리 140735553210824 방에 저장되었음
PS C:\work>
```



이 지식을 알아야 하는 이유



◎ 미래를 위한

- 차후에 파이썬에서 Deep Copy / Shallow Copy 개념을 배울 때 알아야 하는 기본 지식

◎ 용어의 정확한 사용

- $a = 10$ 을 하는 코드는 "**a에 10을 할당한다.**" 라고 표현
 - a는 10을 가리키고 있도록 만든다 라는 의미가 된다.
- 그리고 이것은 "**변수 a는 10을 참조하고 있다**" 라고도 표현 가능하다.
- **a에 10을 저장한다.** 라는 말은 파이썬에서 잘못된 말이다.
 - a는 10을 저장된 위치를 가리키고 있는 것이기 때문



데이터 타입





참고 하세요!

Numeric Types

- int (정수), float (실수), complex (복소수)

Text Sequence Type

- str (문자열)

Sequence Types

- list, tuple, range

Non-sequence Types

- set, dict

기타

- Boolean, None, Functions

형변환(str / int)



- str → int : 형식에 맞는 숫자만 가능

```
print(int('3')) # 3
```

```
print(int('3.14')) # Error
```

```
print(int(3.5)) # 3
```

```
print(float('3.14')) # 3.14
```

- int → str : 모두 가능


```
print(str(1) + '순위') # 1순위
```

- 숫자 : 0부터 9까지
- 수 : 정수 전체, 음수 등

[도전] 데이터 타입 알아보기



◎ `type()` 함수를 사용해서 데이터 타입 5개 이상 출력해 보자.



입력(input)과 연산자



산술 연산자



연산자	의미	예시
+	더하기	$3 + 3 = 6$
-	빼기	$4 - 3 = 1$
*	곱하기	$5 * 3 = 15$
/	나누기	$4 / 2 = 2.0$
//	몫 나누기	$10 // 3 = 3$
%	나머지 나누기	$10 \% 3 = 1$
**	거듭제곱	$10 ** 3 = 1000$

문자열 연산

연산자	의미	예시
+	결합	"abc" + "def" = "abcdef"
*	반복	"abc" * 3 = "abcabcabc"

연산자 우선순위



우선 순위	연산자	예시
1	()	$(3 + 4) * 2 = 14$
2	**	$3 * 2 ** 3 = 24$
3	+, - (단항)	$-3 + 4 = 1$
4	*, /, //, %	$3 + 4 * 3 = 15$
5	+, - (이항)	$3 + 4 - 2 = 9$

◎ 어떻게 될까?

```
# ?  
print(-2 ** 2)  
  
# ?  
print(-(2 ** 2))  
  
# ?  
print((-2) ** 2)
```

복합 연산자



◎ 연산과 할당이 함께 이루어짐

기호`	연산자	예시
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
//=	a //= b	a = a // b

◎ 어떻게 될까?

```
a = 10
a += 4
print(a) # ?

b = 3
b *= 2
print(b) # ?

c = 12
c /= 4
print(c) # ?

d = 11
d //= 2
print(d) # ?
```

주석



- 프로그램 코드 내에 작성되는 설명이나 메모
- 주석은 실행되지 않는다

```
# 12월 17일 python day1 학습 내용 정리
```

```
'''
```

```
A의 아스키 코드는 65
```

```
a의 아스키 코드는 97
```

```
'''
```

◎ 주석은 왜 사용할까?





기본 입력 방법

```
# 문자열 입력
# 파이썬에서는 문자, 문자열 둘 다 string 으로 취급한다.
str = input()

# 수 하나 입력
num = int(input())

# 수 두 개 입력 ("1 5" 처럼 한 줄로 수 2개가 주어지는 경우)
num1, num2 = map(int, input().split())
```

여러 수 입력 받는 방법

- 입력 예시

1 2 3 5 6 7

```
# List에 여러숫자 입력 (한 줄로 입력 필수)
arr1 = list(map(int, input().split()))
```

[도전] 여러 숫자 입력 받아 출력하기



◎ 1 2 3 4 5 를 입력 받아 출력해 보자

- 1. 변수 1개 사용
- 2. 변수 5개 사용



**input()은 실무에서 잘 안 쓴다.
그러면 왜 배울까?**

기타 연산자



◎ 비교 연산자

연산자	의미	예시
>	보다 크다	3 > 5 (False)
<	보다 작다	"apple" < "bat" (True)
>=	보다 크거나 같다	3 * 3 >= 4 * 4 (False)
<=	보다 작거나 같다	3 + 5 <= 7 + 1 (True)
==	같다	1 * 1 == 4 / 4 (True)
!=	다르다	"coding" != "Coding" (True)

◎ 논리 연산자

연산자	의미	예시
and	두 명제가 모두 참이면 참	True and True (True) / True and False (False) / False and False (False)
or	두 명제가 모두 거짓이면 거짓	True or True (True) / True or False (True) / False or False (False)
not	참이면 거짓, 거짓이면 참	not True (False) / not False (True)

비교 연산 + 논리 연산 예시



◎ 어떻게 될까?

```
age = 25
score = 85
is_student = True

print(age >= 20 and score >= 80) # ?

print(is_student or score <= 60) # ?

print(age < 20 or not is_student) # ?
```

멤버십 연산



멤버십 연산자

- 특정 값이 속하는지 여부를 확인

기호	내용
in	왼쪽 피연산자가 오른쪽 피연산자에 속하는지 확인
not in	왼쪽 피연산자가 오른쪽 피연산자에 속하지 않는지 확인

멤버십 연산자 예시

```
name = 'sangho'

print('s' in name) # ?
print('j' in name) # ?
print('h' not in name) # ?
```

[도전] 다음 결과를 예상해 보고 이유를 생각해 보자.



◎ 어떻게 될까?

```
print(2 and 6) # ?  
print(2 and 0) # ?  
print(0 and 2) # ?  
  
print(6 or 2) # ?  
print(2 or 0) # ?  
print(0 or 2) # ?
```



파이썬 문법 정리 조건문, 반복문



조건문 if

if condition: 콜론(:)
 code

▪ 1 tab 또는 4 space로 들여쓰기

- 만약 condition의 결과가 True라면, code를 실행합니다.
- 만약 condition의 결과가 False라면, code를 실행하지 않습니다.

CODE

```
number = int(input())  
if number % 2 == 0:  
    print("짝수")
```

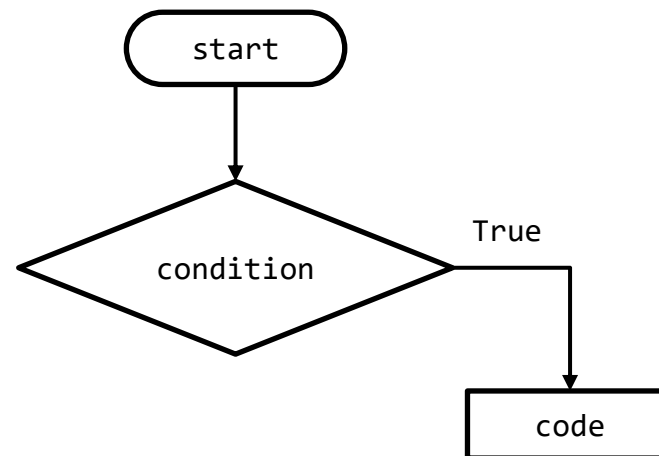
CONSOLE

10
짝수

CONSOLE

3

- Condition의 결과가 False로, “짝수”가 출력되지 않습니다.



조건문 if-if

```
if condition1:  
    code1  
if condition2:  
    code2
```

- 각각의 condition1, condition2는 개별적으로 판단되고, 실행됩니다.

CODE

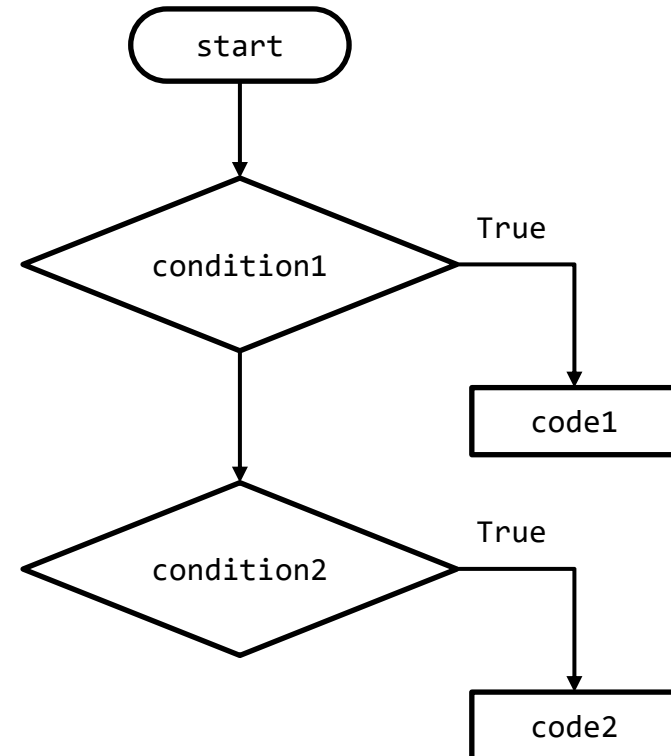
```
number = int(input())  
if number > 10:  
    print("10초과")  
if number > 100:  
    print("100초과")
```

CONSOLE

```
103  
10초과  
100초과
```

CONSOLE

```
33  
10초과
```



조건문 if-else



```
if condition:
```

```
    code1
```

```
else:
```

```
    code2
```

- 만약 condition의 결과가 True라면, code1을 실행합니다.
- 만약 condition의 결과가 False라면, code2를 실행합니다.
- else문은 바로 위에 있는 if문에만 적용됩니다.

CODE

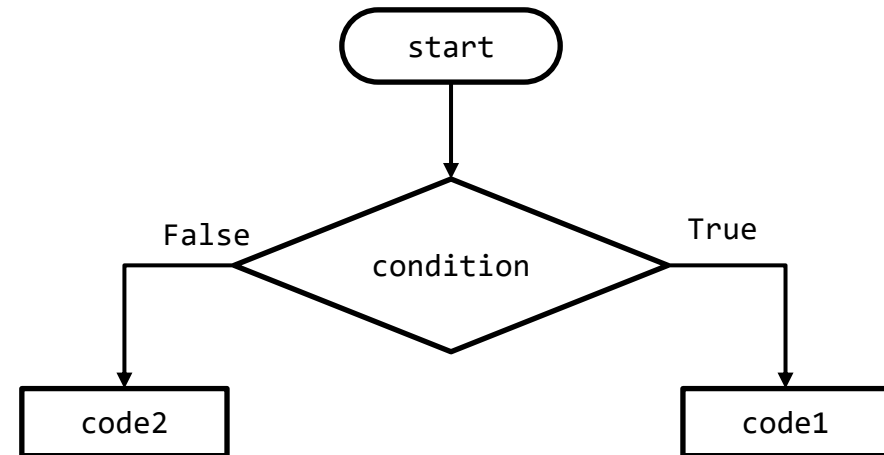
```
number = int(input())
if number % 2 == 0:
    print("짝수")
else:
    print("홀수")
```

CONSOLE

```
33
홀수
```

CONSOLE

```
44
짝수
```



조건문 if-if-else



```
if condition:
    code
if condition1:
    code1
else:
    code2
```

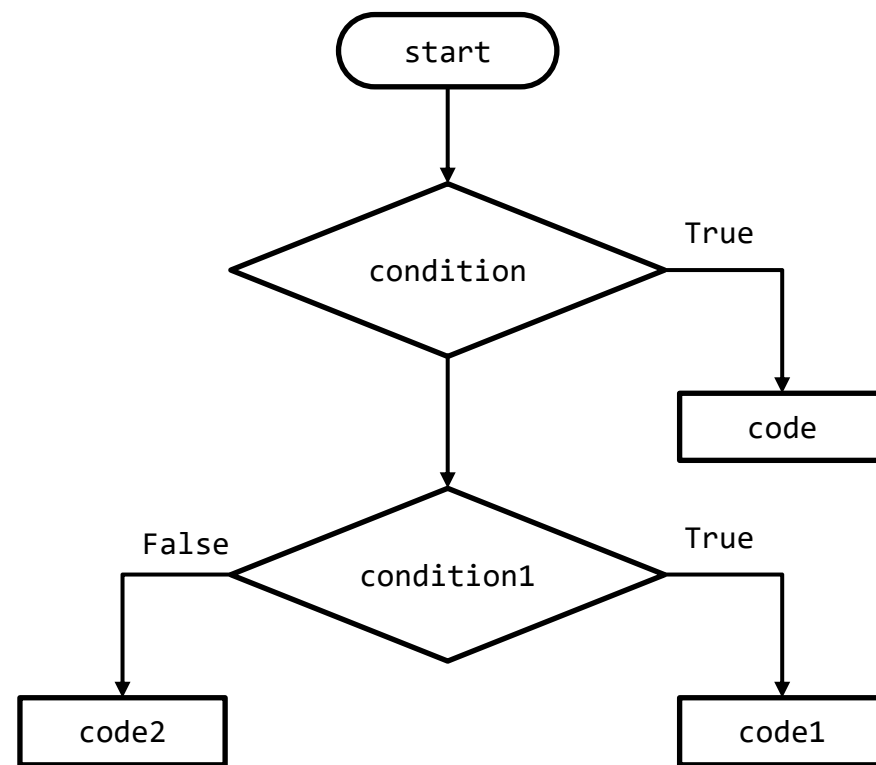
▪ condition과 condition1은 개별적으로 적용되는 조건문입니다.

CODE

```
number = int(input())
if number > 10:
    print("10초과")
if number % 2 == 0:
    print("짝수")
else:
    print("홀수")
```

CONSOLE

```
33
10초과
홀수
```



조건문 if-elif-else



```
if condition1:
    code1
elif condition2:
    code2
else:
    code3
```

- Condition2는 condition1의 결과가 False인 경우에 체크합니다.
- Else문의 code3는 위의 모든 condition이 False일때 실행됩니다.

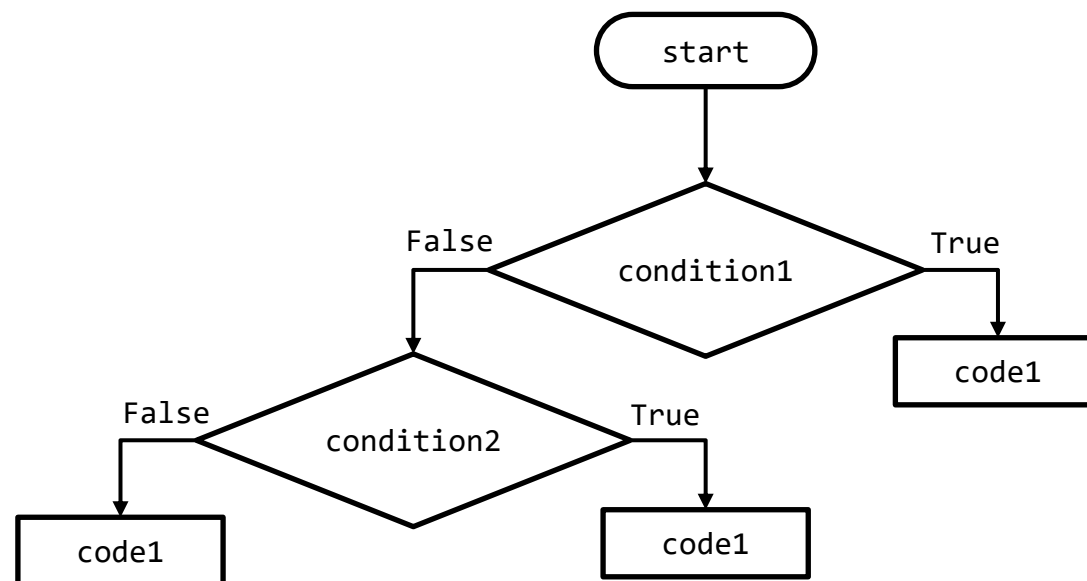
CODE

```
grade = int(input())
if grade > 80:
    print("상")
elif grade > 50:
    print("중")
else:
    print("하")
```

CONSOLE

```
66
중
```

- grade > 80은 False이므로 elif문의 grade > 50을 체크, 해당 명제는 True이므로 “중” 이 출력됩니다.



반복문 while



`while condition:`
`code`

- `condition`의 결과가 “참”일 동안 `code`가 실행됩니다.

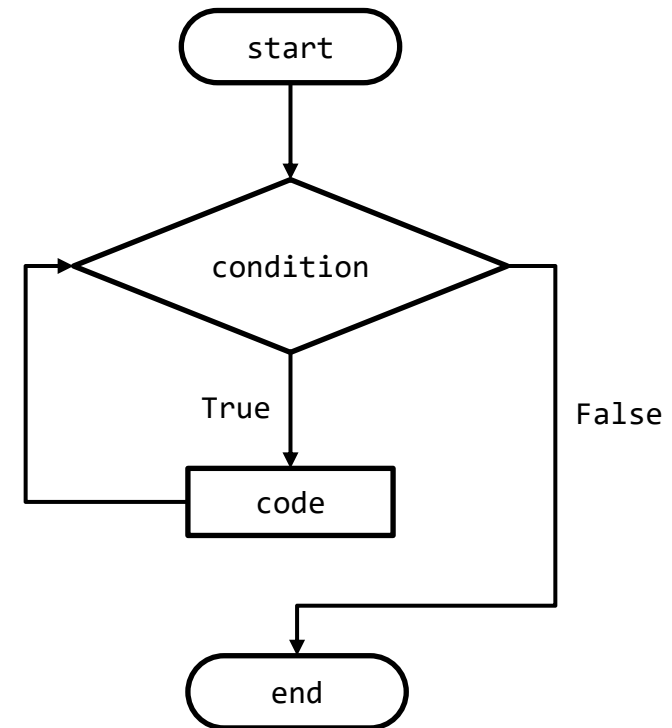
CODE

```
while True:  
    print(1)
```

CONSOLE

```
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
..
```

- `Condition`이 항상 `True`이기 때문에, 멈추지 않고 무한 반복됩니다. → 무한 루프 (infinite loop)
- 그렇기에 `while`문은 언젠가 `condition`이 `False`로 바뀔수 있는 명제가 필요합니다.



반복문 while



```
iterator
while condition:
    code
    move iterator
```

CODE

```
i = 0
while i < 3:
    print("Hello World!")
    i += 1
```

CONSOLE

```
Hello World!
Hello World!
Hello World!
```

- `i += 1`은 `i = i + 1`과 동일합니다.

반복문 for



```
for iterator in iterable:  
    code
```

- iterator: 반복자 (값을 차례대로 꺼낼수 있는 객체)
- iterable : 반복 가능한 객체 (요소를 하나씩 반환할 수 있는 객체)

CODE

```
for letter in "python":  
    print(letter)
```

CONSOLE

```
p  
y  
t  
h  
o  
n
```

- letter는 "python"의 요소들을 순차적으로 접근하여 출력합니다.

iterable



데이터 타입	의미	불/가변	Type	예시
string	문자열	immutable	sequence	"string"
list	여러 요소들이 포함될 수 있는 collection	mutable	sequence	[1, 2, 3, 3]
set	고유의 값들이 포함될 수 있는 collection	mutable	set	{1, 2, 3, 4}
tuple	변화가 불가능한 요소들이 포함될 수 있는 collection	immutable	sequence	(1, 2, 3)
range	특정 정수의 범위를 생성	immutable	sequence	range(1, 5) → [1, 2, 3, 4]
dictionary	key, value 페어로 이루어진 collection	mutable	map	{"one" : 1, "two" : 2}

- iterable : 반복 가능한 객체 (요소를 하나씩 반환할 수 있는 객체)
- **immutable** : collection의 요소가 변할 수 없는 데이터
- **mutable** : collection의 요소가 변할 수 있는 데이터
- **sequence** : 순서가 존재하여 index로 접근할 수 있는 데이터
- set : 집합형 데이터 (순서가 보장되지 않음)
- map: 매핑형 데이터 (순서가 보장되지 않음)

- 위 내용들은 뒷 부분에서 심화적으로 다룰 예정

range()



- 연속된 숫자들로 반복을 할 필요가 있다면 range()를 활용할 수 있습니다.

함수	의미	예시
range(en)	0부터 en-1까지의 숫자 시퀀스를 생성	range(5) → [0, 1, 2, 3, 4]
range(st, en)	st부터 en-1까지의 숫자 시퀀스를 생성	range(2, 5) → [2, 3, 4]
range(st, en, step)	st부터 en-1까지의 숫자 시퀀스를 step간격으로 생성	range(1, 5, 2) → [1, 3]

CODE

```
for i in range(5):  
    print(i, end=" ")
```

CONSOLE

```
0 1 2 3 4
```

CODE

```
for i in range(10, 0, -1):  
    print(i, end=" ")
```

CONSOLE

```
10 9 8 7 6 5 4 3 2 1
```

break



break

- 반복문 내에서 break를 만나는 순간, 반복을 종료합니다.

CODE

```
counter = 0
while True:
    if counter == 3:
        break
    print("counter: ", counter)
    counter += 1
print("반복 종료")
```

CONSOLE

```
counter: 0
counter: 1
counter: 2
반복 종료
```

- counter가 3이 될 때, break의 condition을 충족하여 반복문을 종료합니다.

continue



continue

- 반복문 내에서 continue를 만나는 순간, 반복문의 condition으로 즉각 돌아갑니다.

CODE

```
counter = 0
while True:
    print("counter: ", counter)
    if counter == 3:
        continue
    counter += 1
print("반복 종료")
```

CONSOLE

```
counter: 0
counter: 1
counter: 2
counter: 3
counter: 3
...
```

- counter가 3이 되면, continue를 만나 아래 counter += 1을 만나지 못하고 condition 부분인 while True로 돌아갑니다.
- 그렇기에 3에서 무한 반복이 발생합니다.