파이썬에서 가장 많이 사용되는 자료형은 4가지로 분류할 수 있습니다:

**1. 숫자형 (Numeric)**

* **정수형 (int):** ..., -2, -1, 0, 1, 2, ... 와 같은 정수를 저장합니다.
* **실수형 (float):** 3.14, -10.5, 5e-2 와 같은 실수를 저장합니다.
* **복소수형 (complex):** 5+2j 와 같은 복소수를 저장합니다.
* **논리형 (bool):** True 와 False 를 저장합니다.

**2. 문자열형 (String)**

* "안녕하세요", "파이썬 자료형", 'Hello, World!' 와 같은 문자열을 저장합니다.
* 따옴표('') 또는 쌍따옴표("") 로 묶어 표현합니다.

**3. 시퀀스형 (Sequence)**

* **순서가 있는 값들의 모음**을 저장합니다.
* **인덱싱**과 **슬라이싱**을 사용하여 원소에 접근할 수 있습니다.
* **변형 가능**한 자료형과 **변형 불가능**한 자료형이 있습니다.
  + **변형 가능한 시퀀스형:**
    - **리스트 (list):** [] 로 묶어 여러 값을 순서대로 저장합니다.
    - **변경**이 가능합니다. 예) my\_list = [1, 2, 3]; my\_list[1] = 5
    - **다양한 데이터**를 저장할 수 있습니다. 예) my\_list = ["사과", 100, True]
  + **변형 불가능한 시퀀스형:**
    - **튜플 (tuple):** () 로 묶어 여러 값을 순서대로 저장합니다.
    - **변경**이 불가능합니다. 예) my\_tuple = (1, 2, 3); my\_tuple[1] = 5
    - **리스트**보다 **빠르고 안전**합니다.

**4. 딕셔너리형 (Dictionary)**

* **키-값 쌍**의 모음을 저장합니다.
* **키**를 사용하여 **값**에 접근할 수 있습니다.
* **변형 가능**한 자료형입니다.
  + my\_dict = {"name": "철수", "age": 30, "hobby": "coding"}
  + my\_dict["age"] 를 통해 30 값을 얻을 수 있습니다.

파이썬에서 자료형 형변환은 크게 두 가지 방향으로 나눌 수 있습니다.

**1. 암시적 형변환 (Implicit Type Conversion)**

자동으로 발생하는 형변환 방식입니다. 연산자나 함수 사용 시, 서로 다른 자료형 간의 연산이나 함수 호출이 필요할 때 발생합니다. 파이썬은 가능한 경우 자료형을 자동으로 변환하여 연산이나 함수 호출이 가능하도록 합니다.

**예시:**

Python

num1 = 10 # 정수형

num2 = 3.14 # 실수형

sum = num1 + num2 # 암시적 형변환 발생 (num1이 실수형으로 변환됨)

print(sum) # 출력: 13.14

**2. 명시적 형변환 (Explicit Type Conversion)**

강제적으로 자료형을 변환하는 방식입니다. 특정 자료형으로 변환해야 하는 경우, 변환 함수를 직접 사용하여 자료형을 변환합니다.

**예시:**

Python

num = "123" # 문자열형

converted\_num = int(num) # 명시적 형변환 (문자열형을 정수형으로 변환)

print(converted\_num) # 출력: 123

**자료형 형변환 방향성**

파이썬에서 자료형 형변환은 다음과 같은 방향성을 가지고 있습니다.

**1. 정수형 -> 실수형:**

정수형은 실수형보다 정밀도가 낮기 때문에, 정수형을 실수형으로 변환하는 것은 자연스러운 형변환입니다.

**예시:**

Python

num1 = 10 # 정수형

num2 = 3.14 # 실수형

sum = num1 + num2 # num1이 실수형으로 변환됨

print(sum) # 출력: 13.14

**2. 실수형 -> 문자열형:**

실수형은 문자열로 표현될 수 있기 때문에, 실수형을 문자열형으로 변환하는 것도 가능합니다.

**예시:**

Python

num = 123.45 # 실수형

str\_num = str(num) # 명시적 형변환 (실수형을 문자열형으로 변환)

print(str\_num) # 출력: '123.45'

**3. 문자열형 -> 정수형/실수형:**

문자열형은 숫자로 표현될 수 있는 경우, 정수형 또는 실수형으로 변환될 수 있습니다. 하지만, 문자열이 숫자로 변환될 수 없는 경우에는 오류가 발생합니다.

**예시:**

Python

str1 = "123" # 숫자로 변환 가능한 문자열

str2 = "abc" # 숫자로 변환 불가능한 문자열

num1 = int(str1) # 명시적 형변환 (문자열을 정수형으로 변환)

print(num1) # 출력: 123

try:

num2 = int(str2) # 오류 발생

print(num2)

except ValueError:

print("문자열을 정수형으로 변환할 수 없습니다.")

**4. 기타 자료형 간의 형변환:**

위에서 언급한 자료형 외에도 다양한 자료형 간의 형변환이 가능합니다. 자세한 내용은 파이썬 공식 문서 또는 자료형 관련 도서를 참고하시기 바랍니다.

## **파이썬 조건문 구조 및 문법 (if~else)**

파이썬의 조건문은 특정 조건에 따라 코드 블록을 실행하거나 건너뛰는 데 사용됩니다. 가장 기본적인 조건문 구조는 if~else입니다.

### **1. if 문**

if 문은 조건이 참일 때 실행할 코드 블록을 정의합니다. 구문은 다음과 같습니다.

Python

if 조건:

실행할 코드 블록

**예시:**

Python

age = 20

if age >= 18:

print("성인입니다.")

코드를 사용할 때는 주의가 필요합니다.

content\_copy

위 코드는 age 변수의 값이 18 이상인지 확인합니다. 조건이 참이므로 "성인입니다."라는 문자열을 출력합니다.

### **2. else 문**

else 문은 if 문의 조건이 거짓일 때 실행할 코드 블록을 정의합니다. if 문 뒤에 작성됩니다.

Python

if 조건:

실행할 코드 블록

else:

실행할 코드 블록

**예시:**

Python

age = 17

if age >= 18:

print("성인입니다.")

else:

print("미성년자입니다.")

위 코드는 age 변수의 값이 18 이상인지 확인합니다. 조건이 거짓이므로 "미성년자입니다."라는 문자열을 출력합니다.

### **3. elif 문**

elif 문은 if 문 다음에 여러 개의 조건을 추가할 수 있도록 합니다. elif 문은 순서대로 평가되며, 처음으로 참이 되는 조건에 해당하는 코드 블록만 실행됩니다.

Python

if 조건1:

실행할 코드 블록

elif 조건2:

실행할 코드 블록

elif 조건3:

실행할 코드 블록

else:

실행할 코드 블록

**예시:**

Python

score = 85

if score >= 90:

print("A 등급입니다.")

elif score >= 80:

print("B 등급입니다.")

elif score >= 70:

print("C 등급입니다.")

else:

print("D 등급입니다.")

위 코드는 학점을 계산합니다. score 변수의 값에 따라 A, B, C, D 중 하나의 등급을 출력합니다.

### **4. 중첩 조건문**

if 문과 else 문을 혼합하여 중첩 조건문을 만들 수 있습니다. 중첩 조건문은 여러 단계의 조건을 처리하는 데 유용합니다.

Python

if 조건1:

if 조건2:

실행할 코드 블록

else:

실행할 코드 블록

else:

실행할 코드 블록

**예시:**

Python

age = 25

job = "선생님"

if age >= 18:

if job == "선생님":

print("선생님이신가요? 존경합니다!")

else:

print("성인이시군요.")

else:

print("미성년자입니다.")

위 코드는 나이와 직업에 따라 다르게 문자열을 출력합니다.

### **5. 주의사항**

* if 문, elif 문, else 문은 들여쓰기를 사용하여 코드 블록을 구분해야 합니다. 들여쓰기는 코드의 가독성을 향상시키고 구조를 명확하게 보여줍니다.
* 조건식은 True 또는 False 값을 반환하는 논리 표현식이어야 합니다.
* 여러 개의 elif 문을 사용할 수 있지만, 순서는 중요합니다. 조건은 위에서 아래 순으로 평가되므로, 더 구체적인 조건을 위에 작성하는 것이 좋습니다.
* else 문은 선택적입니다. if 문 또는 elif 문만 사용할 수도 있습니다.

## **파이썬 반복문 구조 및 문법 (for)**

**1. 기본 구조**

Python

for 변수명 in 반복 가능 객체:

실행할 코드 블록

코드를 사용할 때는 주의가 필요합니다.

content\_copy

* **변수명:** 반복 가능 객체의 각 요소를 순서대로 저장하는 변수
* **반복 가능 객체:** 리스트, 튜플, 문자열, 딕셔너리 등 반복될 수 있는 자료 구조
* **실행할 코드 블록:** 반복될 코드들이 들여쓰기되어 작성되는 부분

**2. 예시**

**2.1 리스트 순회**

Python

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

for number in numbers:

print(number)

**결과:**

1

2

3

4

5

**2.2 범위 반복**

Python

for i in range(10):

print(i)

**결과:**

0

1

2

3

4

5

6

7

8

9

**2.3 문자열 순회**

Python

for char in "Hello":

print(char)

**결과:**

H

e

l

l

o

**3. 응용**

**3.1 특정 조건에 맞는 요소만 출력**

Python

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

for number in numbers:

if number % 2 == 0:

print(number)

**결과:**

2

4

**3.2 반복 횟수 제한**

Python

for i in range(3, 6):

print(i)

**결과:**

3

4

5

**3.3 두 개의 리스트를 동시에 순회**

Python

names = ["Alice", "Bob", "Charlie"]

ages = [30, 25, 22]

for name, age in zip(names, ages):

print(f"{name} is {age} years old.")

**결과:**

Alice is 30 years old.

Bob is 25 years old.

Charlie is 22 years old.

**4. 참고**

* for 문은 반복 가능 객체의 모든 요소를 순서대로 한 번씩만 방문합니다. 특정 요소를 건너뛰거나 순서를 바꿀 수는 없습니다.
* 반복 횟수를 정확히 알고 있을 때는 range() 함수를 사용하여 반복 가능 객체를 만드는 것이 좋습니다.
* 리스트, 튜플, 문자열과 같은 기본 자료 구조 외에도 딕셔너리, 파일 객체 등 다양한 자료 구조를 for 문에서 사용할 수 있습니다.
* 더 복잡한 반복에는 중첩 반복문, break 문, continue 문 등을 활용할 수 있습니다.

## **파이썬 while 반복문 구조 및 문법**

**1. 기본 구조**

파이썬의 while 반복문은 조건이 \*\*참(True)\*\*인 동안 코드 블록을 반복적으로 실행하는 구조입니다. 조건이 \*\*거짓(False)\*\*가 되면 반복문이 종료됩니다.

Python

while 조건:

코드 블록

**2. 주요 구성 요소**

* **조건:** while 키워드 다음에 오는 표현식으로, 반복 여부를 결정합니다.
* **코드 블록:** 들여쓰기된 코드 블록으로, 조건이 참일 때마다 반복적으로 실행됩니다.

**3. 작동 방식**

1. while 반복문은 먼저 조건을 평가합니다.
2. 조건이 참이면 코드 블록을 실행하고, 다시 조건을 평가합니다.
3. 조건이 거짓이 될 때까지 2번 과정을 반복합니다.

**4. 예시**

Python

# 1부터 10까지 출력

i = 1

while i <= 10:

print(i)

i += 1

**5. 주의점**

* 조건이 항상 참인 경우, 무한 루프에 빠질 수 있으므로 주의해야 합니다.
* 반복 조건을 변경하는 코드를 반드시 코드 블록 안에 작성해야 합니다.
* break 키워드를 사용하여 반복문을 강제로 종료할 수 있습니다.

**6. for 루프와 비교**

while 루프는 반복 횟수가 미리 정해지지 않은 경우에 유용하며, for 루프는 시퀀스를 순환하는 경우에 적합합니다.

## **파이썬 반복문 구조 및 문법: break**

파이썬에서 반복문은 특정 코드 블록을 반복적으로 실행하는 데 사용되는 제어 흐름 문입니다. 두 가지 기본적인 반복문 유형은 for 루프와 while 루프입니다.

**1. for 루프**

for 루프는 반복 가능한 시퀀스의 각 항목에 대해 코드 블록을 반복적으로 실행합니다. 시퀀스는 리스트, 튜플, 문자열, 범위 등이 될 수 있습니다.

Python

for i in range(10):

print(i)

위 코드는 0부터 9까지의 숫자를 순서대로 출력합니다.

**2. while 루프**

while 루프는 조건식이 참인 동안 코드 블록을 반복적으로 실행합니다. 조건식이 거짓이 되면 루프가 종료됩니다.

Python

i = 0

while i < 10:

print(i)

i += 1

위 코드는 0부터 9까지의 숫자를 순서대로 출력합니다.

**3. break 문법**

break 문은 현재 실행 중인 반복문을 즉시 종료하는 데 사용됩니다. break 문은 반복문 내부의 어느 위치에서든 사용할 수 있으며, 루프의 잔여 코드 블록은 실행되지 않습니다.

Python

for i in range(10):

if i == 5:

break

print(i)

위 코드는 0부터 4까지의 숫자를 출력합니다.

**break 문의 활용 사례**

* 특정 조건을 만족할 때 반복문을 종료하는 경우
* 무한 루프에서 탈출하는 경우
* 반복문 내에서 오류가 발생했을 때 루프를 종료하는 경우

**주의 사항**

* break 문은 반복문 내부에서만 사용할 수 있습니다.
* break 문은 중첩된 반복문에서 벗어나는 데 사용할 수 없습니다. 중첩된 반복문에서 벗어나려면 해당 반복문 내에서 break 문을 사용해야 합니다.

## **파이썬 반복문 구조 및 문법: continue**

파이썬에서 반복문은 반복적으로 코드 블록을 실행하는 데 사용됩니다.

**1. 반복문 종류:**

* **for 문:** 특정 범위의 값을 반복적으로 처리하는 데 사용됩니다.
  + for 변수 in 반복 가능 객체:
    - 변수: 반복 가능 객체의 각 값을 차례대로 할당받는 변수
    - 반복 가능 객체: 리스트, 튜플, 문자열, 범위 등 반복 가능한 데이터 구조

예시:

Python  
for i in range(5):

* + print(i)
* 출력: 0 1 2 3 4

**while 문:** 조건이 참인 동안 반복적으로 코드 블록을 실행하는 데 사용됩니다.

* while 조건:
  + 조건: 반복 여부를 결정하는 논리식
  + 조건이 거짓이 될 때까지 반복 블록을 실행

예시:

Python  
i = 0

while i < 5:

print(i)

* i += 1
* 출력: 0 1 2 3 4

**2. continue 문법:**

* continue는 **현재 반복**을 건너뛰고 **다음 반복**으로 이동하도록 합니다.
* **for 문**과 **while 문** 모두에서 사용 가능합니다.
* continue 뒤에 코드는 실행되지 않습니다.

**3. continue 활용 예시:**

* **짝수만 출력하는 for 문:**Python  
  for num in range(10):

if num % 2 == 0:

print(num)

else:

* continue

출력: 0 2 4 6 8

* **특정 값을 만나면 무시하고 반복하는 while 문:**

Python  
i = 0

while i < 10:

i += 1

if i == 5:

continue

print(i)

* 출력: 1 2 3 4 6 7 8 9 10

**4. continue 주의 사항:**

* continue를 잘못 사용하면 예상치 못한 결과를 초래할 수 있습니다.
* 반복 루프에서 특정 조건을 건너뛰는 경우 신중하게 사용해야 합니다.
* 루프 종료 조건을 명확하게 설정하는 것이 중요합니다.

## **파이썬 콘솔 입출력: print 함수**

파이썬에서 가장 기본적인 입출력 함수는 print 함수입니다. print 함수는 원하는 데이터를 콘솔 창에 출력하는 데 사용됩니다.

### **기본 사용법**

Python

print("안녕하세요!")

print(10)

print(3.14)

print(True)

print("이 문장은 여러 개의 값을 출력합니다: ", 1, 2, 3, 4, 5)

위 코드는 다음과 같은 결과를 출력합니다.

안녕하세요!

10

3.14

True

이 문장은 여러 개의 값을 출력합니다: 1 2 3 4 5

**여러 값 출력:**

print 함수는 쉼표(,)를 사용하여 여러 값을 동시에 출력할 수 있습니다. 각 값 사이에는 공백이 자동으로 삽입됩니다.

**문자열 출력:**

문자열을 출력할 때는 따옴표(') 또는 쌍따옴표(")로 묶어야 합니다.

**숫자 출력:**

정수와 실수를 직접 출력할 수 있습니다.

**논리값 출력:**

True와 False 논리값도 출력할 수 있습니다.

**서식 지정:**

print 함수는 더 많은 기능을 제공합니다. 예를 들어, 출력 형식을 지정하거나, 변수 값을 문자열에 포함하는 방법 등을 제어할 수 있습니다.

## **파이썬 콘솔 입출력: input 함수**

파이썬에서 콘솔 입출력은 두 가지 기본 함수인 input과 print를 사용하여 수행됩니다.

### **1. print 함수**

* **기능**: 콘솔에 데이터를 출력합니다.
* **사용법**:  
  Python

print(출력할 내용)

**예시:**

Python

print("안녕하세요!") # "안녕하세요!" 문자열 출력

print(10) # 숫자 10 출력

print("10" + 20) # 문자열 연결 오류 발생 (숫자와 문자열 연산 불가능)

print(int("10") + 20) # "10"을 숫자 10으로 변환 후 20과 더하여 출력 (30)

**2.input 함수**

**기능:** 사용자로부터 입력값 받기

**사용법:**Python  
변수 = input(프롬프트 문자열)

예시:

Python

name = input("이름을 입력하세요: ")

print(f"안녕하세요, {name}씨!") # 입력받은 이름으로 문자열 출력

age = int(input("나이를 입력하세요: "))

print(f"{name}씨는 올해 {age}살입니다.") # 입력받은 이름과 나이를 사용하여

**input() 함수의 활용:**

* 사용자 입력값을 기반으로 프로그램 동작 조정
* 사용자와의 상호작용 구현
* 게임, 시뮬레이션 등 다양한 프로그램 개발에 활용

python

## **파이썬 콘솔 입출력(print, input) 활용 코드 예시**

**1. 사용자 이름 출력 및 반갑습니다 메시지**

Python

# 사용자 이름 입력

name = input("이름을 입력하세요: ")

# 반갑습니다 메시지 출력

print(f"안녕하세요, {name}님! 반갑습니다!")

**2. 두 수의 합 계산 및 결과 출력**

Python

# 첫 번째 수 입력

num1 = float(input("첫 번째 수를 입력하세요: "))

# 두 번째 수 입력

num2 = float(input("두 번째 수를 입력하세요: "))

# 두 수의 합 계산

sum = num1 + num2

# 합 계산 결과 출력

print(f"{num1} + {num2} = {sum}")

**3. 나이 입력 및 만 나이 계산 후 출력**

Python

# 생년월일 입력 (YYYYMMDD 형식)

birth\_date = input("생년월일을 YYYYMMDD 형식으로 입력하세요: ")

# 현재 연도 입력

current\_year = int(input("현재 연도를 입력하세요: "))

# 만 나이 계산

age = current\_year - int(birth\_date[:4])

# 만 나이 출력

print(f"만 나이는 {age}세입니다.")

**4. 문자열 입력 및 대문자/소문자 변환 후 출력**

Python

# 문자열 입력

sentence = input("문자열을 입력하세요: ")

# 대문자 변환

upper\_case = sentence.upper()

# 소문자 변환

lower\_case = sentence.lower()

# 변환 결과 출력

print(f"대문자: {upper\_case}")

print(f"소문자: {lower\_case}")

**5. 가위바위보 게임**

Python

import random

# 사용자 선택 입력

user\_choice = input("가위(r), 바위(p), 보(s) 중 하나를 선택하세요: ")

# 컴퓨터 선택

computer\_choice = random.choice(["r", "p", "s"])

# 승패 판별 및 결과 출력

if user\_choice == computer\_choice:

print("비겼습니다!")

elif (user\_choice == "r" and computer\_choice == "s") or \

(user\_choice == "p" and computer\_choice == "r") or \

(user\_choice == "s" and computer\_choice == "p"):

print("이겼습니다!")

else:

print("졌습니다!")

## **파이썬 파일 입출력: open() 함수**

파이썬에서 파일을 열고 읽고 쓰는 작업은 open() 함수를 사용합니다.

**1. 파일 열기**

Python

file = open('파일경로', '모드')

* 파일경로: 열려는 파일의 경로를 문자열로 지정합니다. 절대 경로 또는 상대 경로를 사용할 수 있습니다.
* 모드: 파일을 어떤 방식으로 열지 지정하는 문자열입니다. 주요 모드는 다음과 같습니다.
  + r: 읽기 모드 (기본값)
  + w: 쓰기 모드 (기존 파일 내용 삭제)
  + a: 추가 모드 (기존 파일 내용 끝에 추가)
  + b: 바이너리 모드 (텍스트 파일이 아닌 바이너리 데이터 처리)

**예시:**

Python

# 현재 디렉토리에 있는 'data.txt' 파일을 읽기 모드로 열기

file = open('data.txt', 'r')

# '/home/user/mydata/image.png' 파일을 바이너리 쓰기 모드로 열기

file = open('/home/user/mydata/image.png', 'wb')

**2. 파일 읽기**

파일 객체를 사용하여 다양한 방법으로 파일 내용을 읽을 수 있습니다.

* read(): 파일 전체 내용을 문자열로 읽습니다.
* readline(): 파일의 한 줄을 문자열로 읽습니다.
* readlines(): 파일의 모든 줄을 리스트로 읽습니다.

**예시:**

Python

# 'data.txt' 파일의 모든 내용을 읽어 변수에 저장

with open('data.txt', 'r') as f:

data = f.read()

# 'data.txt' 파일의 각 줄을 반복 처리

with open('data.txt', 'r') as f:

for line in f:

print(line)

**3. 파일 쓰기**

파일 객체를 사용하여 파일에 문자열을 쓰거나 바이너리 데이터를 쓸 수 있습니다.

* write(): 문자열을 파일에 씁니다.
* writelines(): 문자열 리스트를 파일에 씁니다.

**예시:**

Python

# 'data.txt' 파일에 "Hello, world!" 문자열 쓰기

with open('data.txt', 'w') as f:

f.write('Hello, world!')

# 'data.txt' 파일에 여러 줄의 문자열 쓰기

with open('data.txt', 'a') as f:

f.writelines(['안녕하세요', '파이썬입니다!'])

**4. 파일 닫기**

파일 객체를 사용한 작업이 끝나면 반드시 close() 함수를 사용하여 파일을 닫아야 합니다.

Python

with open('data.txt', 'r') as f:

data = f.read()

# 자동으로 파일을 닫아줍니다.

**5. with 구문 사용**

with 구문을 사용하면 파일 열고 닫는 작업을 간략하게 처리할 수 있습니다.

Python

with open('data.txt', 'r') as f:

data = f.read()

# 위 코드와 동일한 의미이지만, 파일을 명시적으로 닫지 않아도 됩니다.

## **파이썬 파일 쓰기: write() 함수 활용**

파이썬에서 write() 함수를 사용하면 파일에 데이터를 쉽게 저장할 수 있습니다. 텍스트 파일뿐만 아니라 바이너리 파일까지 다양한 형식의 데이터 쓰기에 활용됩니다.

**1. 기본 사용법:**

Python

# 파일 열기 (쓰기 모드)

file = open('output.txt', 'w')

# 데이터 쓰기

file.write('파일에 저장할 내용입니다.\n')

file.write('다음 줄에 추가할 내용입니다.')

# 파일 닫기

file.close()

**2. 주의사항:**

* 'w' 모드는 기존 파일에 존재하는 내용을 **덮어쓰게** 됩니다. 새로운 내용만 저장하고 싶다면 'a' 모드를 사용하세요.
* write() 함수는 **바이트 단위**로 데이터를 저장합니다. 텍스트 파일을 다룰 때는 인코딩 방식을 지정해야 합니다. 예를 들어, 한국어를 저장하려면 'utf-8' 인코딩을 사용합니다.

**3. 실제 활용 예시:**

**1) 여러 줄 쓰기:**

**Python**

**file = open('output.txt', 'w', encoding='utf-8')**

**for i in range(10):**

**file.write(f'현재 라인: {i}\n')**

**file.close()**

**2) 리스트 내용 쓰기:**

**Python**

**data = ['사과', '바나나', '오렌지']**

**file = open('fruit.txt', 'w', encoding='utf-8')**

**for item in data:**

**file.write(f'{item}\n')**

**file.close()**

**3) CSV 파일 쓰기:**

**Python**

**import csv**

**with open('data.csv', 'w', newline='') as csvfile:**

**writer = csv.writer(csvfile)**

**writer.writerow(['이름', '나이', '직업'])**

**writer.writerow(['김철수', 30, '개발자'])**

**writer.writerow(['이영희', 25, '디자이너'])**

**4. 이진 데이터 쓰기:**

**Python**

**import struct**

**data = struct.pack('i', 1024) # 32비트 정수 1024를 바이너리 데이터로 변환**

**with open('binary.data', 'wb') as file:**

**file.write(data)**

## **파이썬 파일 입출력: read() 함수 사용법**

**파이썬에서 open() 함수와 함께 read() 함수를 사용하면 파일의 내용을 문자열로 읽어올 수 있습니다.**

**1. 기본 사용법:**

**Python**

**with open('파일경로', '모드') as f:**

**데이터 = f.read()**

* **파일경로: 읽을 파일의 경로를 문자열 형식으로 지정합니다.**
* **모드: 파일 열기 모드를 지정합니다. 기본값은 'r' (읽기 모드)이며, 'w' (쓰기 모드), 'a' (추가 모드) 등 다양한 모드가 있습니다.**
* **데이터: read() 함수는 파일의 모든 내용을 읽어 문자열로 반환합니다.**

**2. 주의 사항:**

* **read() 함수는 파일의 모든 내용을 한 번에 읽어 메모리에 로드합니다. 파일이 매우 크면 메모리 부족 문제가 발생할 수 있으므로 주의해야 합니다.**
* **파일을 읽고 나면 반드시 close() 함수를 사용하여 파일을 닫아야 합니다. with 구문을 사용하면 자동으로 닫히므로 편리합니다.**

**3. 예시:**

**Python**

**# 파일 내용을 읽어 변수에 저장**

**with open('data.txt', 'r') as f:**

**data = f.read()**

**# 읽은 내용을 출력**

**print(data)**

**4. 더 많은 정보:**

* **read() 함수는 파일의 전체 내용을 읽어 문자열로 반환합니다. 반면, readline() 함수는 한 줄씩 읽어 문자열로 반환하고, readlines() 함수는 모든 줄을 리스트 형식으로 반환합니다.**
* **encoding 인수를 사용하여 파일의 인코딩 방식을 지정할 수 있습니다. 예를 들어, 한국어 파일을 읽을 때는 encoding='utf-8' 을 사용해야 합니다.**
* **open() 함수의 errors 인수를 사용하여 오류 처리 방식을 지정할 수 있습니다.**