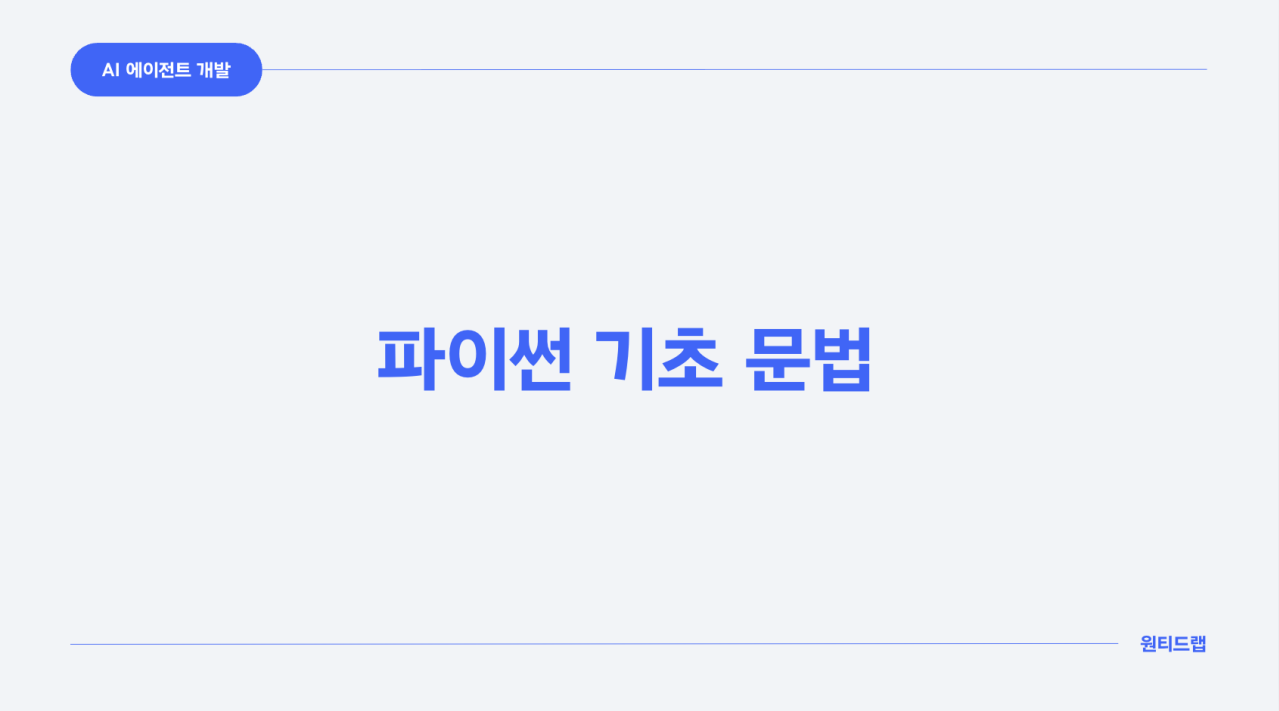
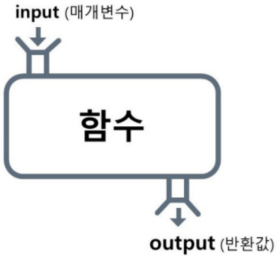
01 파이썬 기초 문법 III (함수, 클래스)



1. 함수



- 입력을 받아 무엇인가를 처리한 후 그 결과를 반환하는 것
- 특정 기능에 대해 함수로 분리해서 코드를 작성하면 기능이나 전체 코드의 흐름 파악에 용이
- 반복해서 사용할 코드를 미리 함수로 정의하면 코드의 재사용이 가능
- 함수는 정의한 후 **호출**을 해야 실행이 가능

1) 표현

```
def 함수이름(매개변수1, 매개변수2, ...):
    실행할 코드 작성

    return 반환값
```

- 함수 이름은 주로 동사 또는 동사구 형태로 작성한다.
- 매개변수: 실행할 코드에 필요한 변수
- `return` : 함수를 호출할 때 변수에 값을 대입할 할 수 있음

2) 변수

지역변수(Local Variable)

- 함수 내부에서 정의된 변수로 함수가 종료되면 소멸됨
- 함수 외부에서는 지역변수를 불러올 수 없음

```
def myfunc():
    local_var = "지역변수"
    print(local_var)

myfunc()
print(local_var)

# 실행 결과(오류)
```

전역변수(Global Variable)

- 함수 외부에서 정의된 변수로 프로그램 전체에서 접근 가능
- 전역 변수를 함수 내부에서 수정하려면 `global` 키워드를 사용

```
global_var = "전역변수"

def myfunc():
```

```
    global global_var
    global_var = "전역변수입니다"

myfunc()
print(global_var)
```

3) 매개변수

- 매개변수를 설정하면 함수 호출 시 반드시 매개변수를 입력해야 함
- `매개변수=값` 형태로 매개변수를 입력하지 않았을 때 자동으로 기본값이 입력되도록 설정할 수 있음
- 매개변수(Parameter): 함수에 입력으로 전달받는 변수
- 인수(Arguments): 함수를 호출할 때 전달하는 입력값
- 매개변수 이름을 지정해서 함수를 호출하면 순서를 바꿔도 상관 없음
- 매개변수 순서에 맞게 값을 넣으면 매개변수를 생략하고 인수만 넣어도 가능

```
def make_message(name, num):
    print(f"{name}님 {num} 진료실로 들어오세요")

make_message(name="홍길동", num=1) # 가능
make_message(num=1, name="홍길동") # 가능
make_message("홍길동", num=1)      # 가능
make_message(name="홍길동", 1)     # 불가능
```

매개변수의 수가 정해져 있지 않을 때

`*args`, `**kwargs` 로 표현할 수 있다.

```
def get_average_score(*args):
    length = len(args)
    return sum(args) / length

avg_score= get_average_score(70, 70, 40)
print(avg_score)
```

```
def make_info(name, **kwargs):
    print(f"Name: {name}")
    for key, value in kwargs.items():
    print(f"{key.capitalize()}: {value}")

make_info("홍길동")
make_info("홍길동", email="gildong@gmail.com", message="안녕하세요")
```

4) Lambda 함수

- `lambda` 키워드로 한줄 코드 작성
- 간단한 함수일 때 사용

```
calc_add = lambda x, y: x + y
result = calc_add(2, 3)
print(result)

# 실행 결과
5
```

2. 파일 읽기/쓰기

1) 파일 경로

2) 파일 생성/읽기/추가

생성

```
content = "파일에 넣을 내용"
with open("파일경로", mode="w", encoding="utf-8") as f:
    f.read(content)
```

읽기

```
with open("파일경로", mode="r", encoding="utf-8") as f:
    content = f.read()
```

추가

```
with open("파일경로", mode="a", encoding="utf-8") as f:
    f.write()
```

4. 예외 처리

- 프로그램 실행 중 오류가 발생할 수 있는 상황을 **안전하게 처리**하는 방법

1) 표현

```
try:
    실행할 코드
except:
    오류가 발생했을 때 실행할 코드
```

2) 활용

특정 오류에 대한 예외 처리

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
```

모든 오류에 대한 예외 처리

```
try:
    x = 10 / 0
except Exception as e:
    print(f"오류 종류: {type(e).__name__}")
    print(f"오류 메세지: {e}")
```

예외 처리 후 무조건 실행(`finally`)

```
try:
    print("나눗셈 시작")
    x = 10 / 0
```

```
except Exception as e:
    print(f"오류 발생: {e}")
finally:
    print("나눗셈 종료")
```