함수와 흐름 제어 1

함수 / 조건문 / 내장 함수

GOAL

- 함수 정의와 return 이해
- 매개변수 있는 함수와 없는 함수 구분
- 람다 함수 이해하기
- 조건문 (if, elif, else, 삼항연산자, pass) 익히기
- 내장 함수(map, filter, zip 등) 활용하기
- 데이터 타입 관점에서 함수 동작 파악하기

목차

1. 함수(Function)란?

1. 함수(Function)란?

- 자주 쓰는 코드를 **묶어두는 도구**
- 필요할 때마다 꺼내서 실행 가능
- 돈(입력)을 넣으면 → 음료수(출력)이 나온다!

함수 정의 문법

```
def 함수이름(매개변수):
    실행할 코드
    return 결과값
```

- `def` : 함수 정의 키워드
- `return` : 결과값 돌려줌 (없으면 None)

매개변수(Parameter) vs 인자(Argument)

- 매개변수(Parameter)
    - 함수를 정의할 때, 함수가 받을 변수 이름
      (자판기에 있는 '콜라', '사이다', 라벨에 있는 금액)
- 인자(Argument)
    - 함수를 실제로 호출할 때 넣는 값
      (실제로 투입한 동전)

매개변수(Parameter) vs 인자(Argument)

```
def add(a, b):   # a, b → 매개변수 (설계도에 적힌 이름)
    return a + b

print(add(3, 5)) # 3, 5 → 인자 (실제 넣은 값)
```

매개변수 없는 함수

```
def hello():
    return "안녕하세요"

print(hello())  # '안녕하세요'
```

- 입력: 없음
- 출력: 문자열(str)

매개변수가 있는 함수

```
def add(a, b):
    return a + b

print(add(3,5))  # 8
```

- 입력: int 2개
- 출력: int

여러 매개변수

```
def introduce(name, age):
    return f"저는 {name}, {age}살입니다."

print(introduce("철수", 20))
# "저는 철수, 20살입니다."
```

- 입력: str, int
- 출력: str

함수 반환값 (return)

```
def square(x):
    return x * x

num = 4
print(square(num))  # 16
```

- 함수는 항상 **입력 → 출력** 흐름을 가진다

Default 매개변수 (기본값)

```
def greet(name="손님"):
    return f"{name}님, 어서오세요!"

print(greet())        # "손님님, 어서오세요!"
print(greet("철수")) # "철수님, 어서오세요!"
```

- 값을 안 넣으면 기본값 사용
- 편리하고 오류 방지에 좋음

키워드 인자 (Keyword Arguments)

```
def introduce(name, age):
    return f"{name}, {age}살"

print(introduce(age=20, name="영희"))
```

- 순서를 헷갈려도 안전
- 가독성이 좋아짐

*args (여러 개 위치 인자 받기)

```
def add_all(*numbers):
    return sum(numbers)

print(add_all(1,2,3,4))  # 10
```

- 여러 개 인자를 **튜플**로 묶어서 받음

**kwargs (여러 개 키워드 인자 받기)

```
def show_info(**info):
    for k, v in info.items():
        print(k, ":", v)

show_info(name="민수", city="Seoul", job="Dev")
```

- 여러 개 키워드 인자를 **dict**로 묶어서 받음

Docstring (함수 설명 달기)

```
def square(x):
    """숫자를 제곱해서 반환하는 함수"""
    return x * x

help(square)
```

- 함수에 **설명 문서**를 달 수 있음
- 협업, 유지보수에 매우 유용

함수와 변수 범위 (Scope)

```
def make_num():
    x = 10
    return x

print(make_num())  # 10
print(x)           # ❌ 오류! (x는 함수 안에서만 존재)
```

- 함수 안에서 만든 변수는 밖에서 쓸 수 없음

함수 정리

- def, return 기본 문법
- default 값으로 편리하게
- 키워드 인자로 안전하게
- *args, **kwargs로 유연하게
- Docstring으로 설명 달기
- 함수 안 변수는 밖에서 못 씀

함수 예제 (할인 쿠폰 적용)

```
def apply_discount(*prices, **options):
    """
    환자 결제 금액에 할인 적용
    - *prices: 여러 가격 합산
    - **options:
        - rate: 할인율 (예: 0.1 → 10%)
        - member: 회원 여부 (True면 추가 1000원 할인)
    """
    total = sum(prices)
    if "rate" in options:
        total *= (1 - options["rate"])
    if options.get("member"):
        total -= 1000
    return int(total)

print(apply_discount(10000, 20000, rate=0.1))
print(apply_discount(5000, 5000, member=True))
print(apply_discount(10000, 20000, rate=0.1, member=True))
```

2. 람다 함수

- 이름 없는 짧은 함수
- `lambda 매개변수: 표현식`

```
square = lambda x: x * x
print(square(5))  # 25 (int)
```

람다와 일반 함수 비교

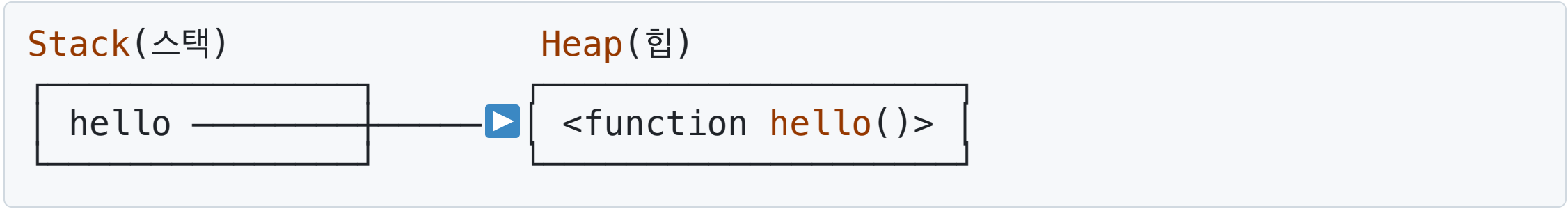
```
def add(a, b):
    return a + b

add_lambda = lambda a, b: a + b

print(add(3,4))        # 7
print(add_lambda(3,4)) # 7
```

- 기능은 같지만, 람다는 간단하게 표현할 때 사용

함수 역시 Reference 타입!



- hello 라는 이름은 단지 주소표일 뿐, 실제 함수의 내용은 Heap에 저장됨.

3. 조건문 (if, elif, else)

```
x = 10

if x > 0:
    print("양수")
elif x == 0:
    print("0")
else:
    print("음수")
```

- bool 결과(True/False)에 따라 실행 분기

범위를 다 쓰는 조건문

```
x = 88

if x > 95:
    print("A")
elif x <= 95 and x > 85:
    print("B")
elif x <= 85 and x > 75:
    print("C")
else:
    print("D")
```

범위를 다 안 쓰는 조건문(권장)

```
x = 88

if x > 95:
    print("A")
elif x > 85:   # 여기 들어올 때 이미 x <= 95
    print("B")
elif x > 75:   # 여기 들어올 때 이미 x <= 85
    print("C")
else:
    print("D")
```

pass 키워드

```
x = 5
if x > 0:
    pass  # 아무 일도 안 함
else:
    print("음수")
```

- 아직 코드를 안 짰을 때 자리 채우기용

삼항 연산자

```
age = 20
status = "성인" if age >= 18 else "미성년자"
print(status)  # 성인
```

- 조건을 한 줄로 표현 가능

삼항 연산자 응용

```
score = 88

grade = "A" if score > 95 else "B" if score > 85 else "C" if score > 75 else "D"

print("등급:", grade)
```

4. 내장 함수란?

- 파이썬이 기본으로 제공하는 함수들
- 별도 `import` 없이 바로 사용 가능
- **반드시 알아야 할 기초 도구들**

map()

- 리스트(이터러블)의 모든 값을 **변환(가공)** 할 때 사용

```
nums = [1,2,3]
squared = list(map(lambda x: x**2, nums))
print(squared)  # [1,4,9]
```

- 입력: 리스트(int)
- 출력: 리스트(int)

filter()

- 조건에 맞는 값만 **선별**

```
nums = [1,2,3,4,5]
evens = list(filter(lambda x: x%2==0, nums))
print(evens)  # [2,4]
```

- 입력: 리스트(int)
- 출력: 조건 만족하는 리스트(int)

zip()

- 여러 리스트를 **쌍으로 묶음**

```
names = ["Alice", "Bob"]
ages = [25, 30]

paired = list(zip(names, ages))
print(paired)  # [('Alice',25), ('Bob',30)]
```

- 입력: 리스트 2개
- 출력: 튜플 리스트

len()

- 길이(원소 개수)를 구함

```
nums = [1, 2, 3]
print(len(nums))  # 3
```

- 문자열, 리스트, 튜플 등 모든 시퀀스 자료형에 사용 가능

sum(), max(), min()

- 합계, 최댓값, 최솟값을 구함

```
nums = [1, 2, 3]
print(sum(nums))  # 6
print(max(nums))  # 3
print(min(nums))  # 1
```

- 숫자 리스트 다룰 때 가장 자주 씀

sorted()

- 정렬된 새로운 리스트 반환
- `key` 매개변수로 함수 전달 가능

```
words = ["apple", "banana", "kiwi"]
print(sorted(words))              # ['apple','banana','kiwi']
print(sorted(words, key=len))     # ['kiwi','apple','banana']
```

any(), all()

- 불리언 조건을 한 번에 평가
- any: True가 하나라도 있으면 True 반환.
- all: 모든 값이 True일 때만 True 반환.

```
values = [False, False, True]
print(any(values))  # True

values = [True, True, True]
print(all(values))  # True
```

내장함수 요약

- map, filter → 함수를 인자로 전달해 데이터 가공
- zip → 여러 리스트를 묶어주기
- len, sum, max, min → 기본 연산 필수 도구
- sorted(key=) → 함수와 함께 강력한 정렬
- any, all → 조건 검사 간단히 표현