codingOn X posco

K-Digital Training 신재생에너지 활용 IoT 과정

codingOn

Python 함수

오늘 수업은?

- 지난시간에는 조건문과 반복문에 대해서 공부하였습니다.
- 오늘은 지금껏 이름만 들어봤던 함수에 대해서 학습하면서 직접 함수를 만들어보는 시간을 가지겠습니다.
- 함수는 프로그래밍의 핵심으로 매우 중요하니 이번시간의 내용을 꼭 이해하고 잘 활용할 수 있도록 해야합니다.

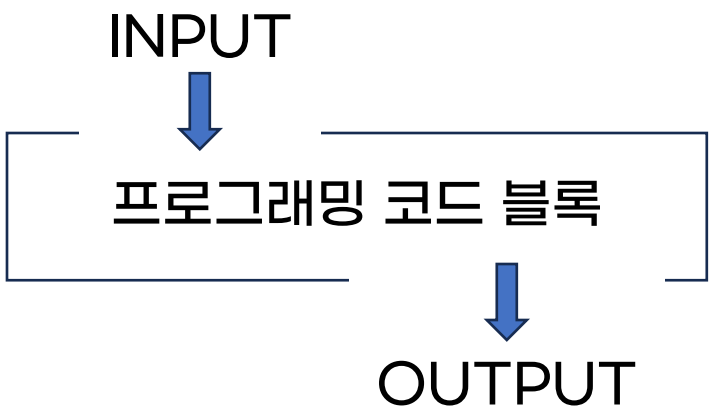
학습목표

- 함수를 만들고 사용할 수 있다

함수

함수란?

- 함수는 특정 작업을 수행하도록 만든 코드
- 코드를 미리 정의해 두면, 나중에 반복적으로 그 작업을 하고 싶을 때 함수 이름만 불러주면 실행이 됨
- 무엇인가를 넣으면 그에 대한 결과값이 나오는것




```
print("Hello, World")
# Hello, World
```

Input값

Output값

함수와 메서드

codingOn

- 식별자에 괄호가 붙은 것
- 식별자. 으로 연결된 것은 메서드라고 불림
- 파이썬에 기본으로 포함되어 있는 함수(내장함수)도 있고 프로그래머가 직접 만들 수도 있음

```
# 내장 함수
a = "Strawberry Moon"
print(len(a)) # 15
print(a.count("r")) # 3
```

함수 관련 용어

- 수학에서의 함수로 설명

  다음과 같은 함수를 정의함

  $$f(x) = x^2 + 2x + 1$$

- x = 3을 대입한 f(3) -> 함수 호출,
  3 -> 매개 변수(함수에 3이라는 값 전달)
- f(3) = 16 -> 리턴 값! (함수를 호출해서 나오는 결과)

함수 관련 용어

- 호출하다 : 함수를 사용하다
- 매개변수 : 함수 괄호 내부에 데이터를 넣는 것(개수에 주의)
  - 예) 매개변수가 2개이면 호출하는 함수도 2개 값을 입력
- 리턴 값 : 함수를 호출해서 나오는 결과

```
# 함수의 기본적인 사용법

def 함수 이름 (매개변수1, 매개변수2, ...):
    수행문 1
    수행문 2
    return 반환값


# def : 함수 선언을 위한 키워드
# 매개변수 : 함수에 입력할 값
```

함수 기본 구조

- 입력값X, 결과값X → 그저 함수 내부의 일만 수행

```
# 함수 정의
def 함수명():
  문장

# 함수 호출
함수명()
```

```
def say_hello():
    print("hello")

say_hello()  # hello
say_hello()  # hello
```

함수 정의

함수 호출

매개변수를 받는 함수 구조

codingOn

- 입력값O, 결과값X -> 입력 값을 이용해 함수 내부의 일을 수행
  - 매개변수 개수 일치하게

```
# 함수 정의
def 함수명(매개변수1, 매개변수2, ...):
  문장

# 함수 호출
함수명(인수1, 인수2, ...)


# 주의
매개변수: 함수 정의부에 전달하는 값
인수: 함수 호출부에 전달하는 값
```

```
def say_hello(name):
    print("hello!", name)

say_hello("민지")  # hello! 민지
say_hello("혜린")  # hello! 혜린
```

매개변수가 한개이기 때문에 입력값인 인수도 한개

매개변수를 받는 함수 구조

codingOn

- 예) 구구단을 출력하는 함수

```python
def gugudan(dan, end):
    print(f'{dan}단')
    for i in range(1, end):
        print(f'{dan} x {i} = {dan * i}')

gugudan(7, 15)
```

```
7단
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
7 x 11 = 77
7 x 12 = 84
7 x 13 = 91
7 x 14 = 98
```

결과값이 있는 함수 (return)

codingOn

- 함수 내부의 수행 결과를 return에 담아서 내보냄
- 입력값이 없을 수도 있고 있을 수도 있음
- 예) 입력값이 없는 함수

```
def 함수명():
    문장
    ...
    return 결과값

# 결과값을 변수에 할당하여 사용
식별자 = 함수명()
```

```
def calculate_sum():
    total = 0
    for i in range(1, 11):
        total += i
    return total

result = calculate_sum()
print(result)  # 55
```

결과값이 있는 함수 (return)

codingOn

- 예) 입력값이 있는 함수

```
def 함수명(매개변수1, 매개변수2, ...):
    문장
    ...
    return 결과값

# 결과값을 변수에 할당하여 사용
식별자 = 함수명()
```

```
def calculate_sum(num1, num2):
    total = 0
    for i in range(num1, num2):
        total += i
    return total

result = calculate_sum(1, 11)
print(result)  # 55
```

결과값이 있는 함수 (return)

codingOn

- return은 상황에 따라 리스트, 튜플, 딕셔너리, 셋(Set) 등 다양한 컬렉션으로 반환이 가능

```python
# 리스트 반환
def get_fruits():
    return ["apple", "banana", "cherry", "orange"]

fruits = get_fruits()
print(fruits)  # ["apple", "banana", "cherry", "orange"]

# 딕셔너리 반환
def get_student_info():
    return {
        "name": "홍길동",
        "age": 20,
        "major": "컴퓨터공학"
    }

student_info = get_student_info()
print(student_info)
# {'name': '홍길동', 'age': 20, 'major': '컴퓨터공학'}
```

실습1.

두 수(2, 2)를 매개변수 전달하여 서로 같으면 곱하고, 서로 다르면 더하는 함수를 정의하고 호출하는 프로그램을 작성하세요.

실행 결과 ☞

```
결과(곱): 4
결과(합): 5
```

실습2.

주문 상품 가격이 20,000원 미만이면 배송비 (2,500원) 포함하고, 아니면 배송비를 포함하지 않는 프로그램을 작성하세요

실행 결과 ☞

```
상품1 가격: 30,000원
상품2 가격: 17,500원
```

매개변수로 리스트 전달

- 리스트를 매개변수로 새로운 리스트 만들기
- 리스트 내포 사용

```python
def times(num):
    return [i * 3 for i in num]

arr = [1, 2, 3, 4, 5]
newArr = times(arr)
print(newArr) # [3, 6, 9, 12, 15]
```

실습3. 자판기 프로그램 함수화

codingOn

- 지난 시간에 한 자판기 프로그램을 함수화 해보세요.

```
1. check_machine : 남은 음료수를 확인할 수 있는 함수
2. is_drink : 음료수가 있는지 확인하는 함수
3. add_drink : 음료수를 추가하는 함수
4. remove_drink : 음료수를 제거하는 함수
```

codingOn

변수

변수 범위

- 전역 변수: 프로그램 전체에서 사용되는 변수
- 지역 변수: 한정된 지역(ex. 함수)에서만 사용되는 변수

변수의 유효 범위

codingOn

- **전역 변수**
- 함수, 클래스 외부에 선언하여 사용하고 영향 범위가 파일 전체
- 프로그램이 종료되면 메모리에서 소멸
- 코드의 어디서는 접근할 수 있지만 코드의 다른부분이 전역 변수에 접근해 값을 변경할 수 있어 복잡성이 증가

```
quantity = 2  # 전역변수

def get_price():
    price = 4000 * quantity
    return price

unit_price = get_price()
print(f'{quantity}개에 {unit_price}원입니다.')
# 2개에 8000원입니다.
```

변수의 유효 범위

codingOn

- **지역변수**
- 지역변수는 함수나 명령문(조건, 반복)의 블록 안에서 생성되며 블록{ }을 벗어나면 메모리에서 소멸
- 파이썬에서는 : 와 들여쓰기가 블록{ } 처럼 사용

```
def oneUp():
    x = 0      # 지역변수
    x = x + 1
    return x

print(oneUp()) # 1
print(oneUp()) # 1
# print(x)
```

변수의 유효범위 비교

```python
a = 11

def calculate_sum():
    a = 7
    total = 0
    for i in range(1, a):
        total += i
    return total

print(calculate_sum())  # 21
print(a)  # 11
```

동일한 식별자의 지역변수가 존재할 경우

```python
a = 11

def calculate_sum():
    # a = 7
    total = 0
    for i in range(1, a):
        total += i
    return total

print(calculate_sum())  # 55
print(a)  # 11
```

동일한 식별자의 지역변수가 없을 경우

global 키워드

- 전역 변수의 유효 범위 - global 키워드 사용
- 전역 변수의 값을 함수 내부에서 변경하려면 global 키워드를 사용
- 전역 변수를 함수 내에서 읽기만 할 때는 global 키워드 없이 접근 가능

```
x = 0

def oneUp():
    global x
    x = x + 1
    return x

print(oneUp())  # 1
print(oneUp())  # 2
print(oneUp())  # 3
```

매개변수

함수의 기본 매개변수

- 기본 매개변수란?
- 매개변수를 초기화하여 선언하고 함수 호출시 인수를 생략하면 기본 값으로 출력된다.

```
def 함수 이름(변수1, 변수2=1):
    코드블럭
```

```python
def pr_str(txt, count=1):
    for __ in range(count):
        print(txt)

pr_str("Hello", 3)
pr_str("Hi")
# Hello
# Hello
# Hello
# Hi
```

함수 호출 키워드

codingOn

- 함수 호출시 매개변수에 값을 전달할때 매개변수의 이름을 지정하여 전달하는 방식
- 순서에 상관없이 값을 전달 할 수 있음
- 위치 매개변수와 같이 사용할때는 위치 매개변수가 항상 앞에있어야 함

```python
def introduce(name, age, city):
    print(f"이름: {name}") # 이름: 홍길동
    print(f"나이: {age}") # 나이: 25
    print(f"사는곳: {city}") # 사는곳: 서울

# 키워드를 사용하여 호출
introduce(city="서울", name="홍길동", age=25)
```

```python
# 올바른 예
introduce("홍길동", age=25, city="서울")

# 잘못된 예
introduce(age=25, "홍길동", city="서울")
```

함수의 가변 매개변수

codingOn

- 가변 위치 매개변수
- 매개변수의 개수가 가변적일때 사용
- 식별자앞에 * 을 표시하며 관례상 *args 식별자를 사용
- 모두 위치 매개변수를 튜플 형태로 묶어 사용

```
def calc_avg(*args):
    sums = 0
    for i in args:
        sums += i
    average = sums / len(args)
    return average

print(calc_avg(1, 2)) # 1.5
print(calc_avg(1, 2, 3, 4, 5)) # 3.0
```

```
def text_def(a, *args):
    print("a:", a) # a: 1
    print("args:", args) # args: (2, 3, 4, 5)

text_def(1, 2, 3, 4, 5)
```

주의) 고정 위치 매개변수 뒤에 있어야함

함수의 가변 매개변수

codingOn

- 가변 키워드 매개변수
- 키워드 개수가 가변적일때 사용
- 식별자앞에 ** 을 표시하며 관례상 **kwargs 식별자를 사용
- 모든 키워드 매개변수를 딕셔너리 형태로 묶어서 사용

```
def introduce(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

introduce(city="서울", name="홍길동", age=25)
# city: 서울
# name: 홍길동
# age: 25
```

주의) *args와 **kwargs를 함께 사용할때 반드시 *args가 앞에 오고 **kwargs가 뒤에 와야 함

내장함수

파이썬 내장함수

- 내장 함수
  - abs()
  - aiter()
  - all()
  - anext()
  - any()
  - ascii()
  - bin()
  - bool
  - breakpoint()
  - callable()
  - chr()
  - classmethod()
  - compile()
  - complex
  - delattr()
  - dir()
  - divmod()
  - enumerate()
  - eval()
  - exec()
  - filter()

더 많은 내장함수는 아래 링크로 접속

https://docs.python.org/ko/3/library/index.html

내장함수 예제

```
# abs(정수) - 절대값을 구하는 내장함수
print(abs(-7))  # 7


def myabs(x):
    if x < 0:
        return -x
    else:
        return x

print(myabs(-5))  # 5
print(myabs(9))   # 9
```

내장함수 예제 - 거듭제곱

codingOn

- 거듭제곱 함수 만들고 비교하기

```python
def my_pow(x, y):
    num = 1
    for i in range(0, y):
        print(f'i = {i}, {num*x} = {num} * {x}')
        num = num * x
    return num

print(my_pow(2, 3))
print(my_pow(3, 3))
print(pow(2, 3))
print(pow(3, 3))
```

```
i = 0, 2 = 1 * 2
i = 1, 4 = 2 * 2
i = 2, 8 = 4 * 2
8
i = 0, 3 = 1 * 3
i = 1, 9 = 3 * 3
i = 2, 27 = 9 * 3
27
8
27
```

내장함수 예제 - map()

codingOn

- map() : 리스트의 각 요소에 주어진 함수를 적용하여 새로운 리스트를 반환
- map(함수, 반복 가능한 객체)
- 결과는 list()로 변환

```
def square(x):
    return x**2

numbers = [1, 2, 3, 4]

squared = map(square, numbers)
print(list(squared))  # 출력: [1, 4, 9, 16]
```

내장함수 예제 - filter()

codingOn

- filter() : 리스트에서 주어진 조건을 만족하는 요소만 필터링하여 반환
- filter(함수, 반복 가능한 객체)
- 결과는 list()로 변환

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

def even_number(x):
    return x % 2 == 0

even_numbers = filter(even_number, numbers)
print(list(even_numbers))  # 출력: [2, 4]
```

실습4. 함수만들기

1~30까지의 자연수 중 배수와 배수의 개수를 계산하는 함수를 정의하시오.

실행 결과 ☞

```
3 6 9 12 15 18 21 24 27 30
3의 배수의 개수: 10
```

재귀함수

재귀함수

- 재귀함수는 **반복적인 문제를 깔끔하게 풀고 싶을 때** 씀
- 복잡한 문제를 작은 하위 문제로 나누어 직관적으로 해결할 수 있음
  - 예: 팩토리얼, 피보나치, 하노이의 탑 등.
- 반복적인 구조의 간결한 표현
  - 트리 구조 탐색, 그래프 탐색(DFS) 등에서 반복문보다 코드가 간결하고 이해하기 쉬움

재귀함수

- 어떤 함수 안에서 자기 자신을 부르는 것을 말한다.
- 재귀호출은 무한 반복하므로 종료 조건이 필요함

```
def func(입력 값):
    if 입력값이 충분히 작으면: #종료 조건
        return 결과값
    else:    # 더 작은 값으로 호출
        return 결과값
```

```python
def sos(i):
    print("Help me!")
    if i == 1:  # 종료 조건
        return ""
    else:
        return sos(i-1)

sos(10)
```

팩토리얼 구하기

codingOn

- 팩토리얼 : 그 수보다 작거나 같은 모든 양의 정수의 곱
- Ex) 5! = 5 x 4 x 3 x 2 x 1 = 120

```python
# 반복문
n = 5
factorial = 1

for i in range(1, n+1):
    factorial *= i

print(factorial)  # 120
```

```python
# 재귀함수
def factorial(n):
    if n == 1:  # 종료 조건
        return 1
    else:
        return n * factorial(n - 1)  # 재귀 단계

print(factorial(5))  # 120
```

실습5. 피보나치 수열 만들기

- 피보나치 수열은 다음과 같이 정의된다

$$F(0) = 0, F(1) = 1$$

$$F(n) = F(n-1) + F(n-2) \quad (n \geq 2)$$

1	1	2	3	5	8

- 숫자 N에 대해, 피보나치 수열의 N번째 숫자를 구하는 프로그램을 작성
- 재귀함수를 사용하여 코드 작성
- 출력 예시

```
print(fibonacci(6))  # 8
```

람다식

Lambda Expressions(람다식)

- 람다 함수(Lambda Function)는 짧은 익명 함수를 작성할 때 사용
- 일반적인 def 키워드를 사용하는 함수와 달리, 이름 없이 한 줄로 간단히 정의할 수 있음

Lambda Expressions(람다식)

codingOn

- Lambda Expressions(람다식)

  lambda 키워드로 익명 함수를 만들수 있다.

  return 키워드 생략함

  lambda 매개변수 : 표현식

```
# 일반 함수
def add(x, y):
    return x + y

print(add(1, 2))
```

```
# 람다식
add = lambda x, y: x + y
print(add(1, 2))
```

Lambda Expressions(람다식)

codingOn

- 매개변수가 1개인 람다 함수

```
# 1 더하기
oneup = lambda x: x + 1
print(oneup(1))
print((lambda x: x + 1)(1))

# 3의 배수
times = lambda x: x * 3
print(times(2))
print((lambda x: x * 3)(2))

# 제곱수
square = lambda x: x * x * x
print(square(4))
print((lambda x: x * x * x)(4))
```

Lambda Expressions(람다식)

codingOn

- 매개변수가 2개인 람다 함수

```
# 사칙연산
add = lambda x, y: x + y
print(add(3, 4))
print((lambda x, y: x + y)(3, 4))

sub = lambda x, y: x - y
print(sub(3, 4))
print((lambda x, y: x - y)(3, 4))

mul = lambda x, y: x * y
print(mul(3, 4))
print((lambda x, y: x * y)(3, 4))

div = lambda x, y: x / y
print(div(3, 4))
print((lambda x, y: x / y)(3, 4))
```

Lambda Expressions(람다식)

codingOn

- 람다 함수를 매개 변수로 전달하기

```
# 함수를 매개 변수로 전달
print("콜백(callback) 함수")

def call_10(func):
    for _ in range(10):
        func()

def hello():
    print("안녕하세요")

# 호출
call_10(hello)

# lambda 함수로 정의 - 매개변수가 없는 경우
# 단순한 한 줄 출력에는 람다 함수가 더 적합
hello2 = lambda: print("안녕하세요")
call_10(hello2)
```

Lambda Expressions(람다식)

codingOn

- lambda - map() 함수와 함께 사용

```
def square(x):
    return x**2

numbers = [1, 2, 3, 4]

squared = map(square, numbers)
print(list(squared))  # 출력: [1, 4, 9, 16]

# 람다 함수 사용
squared = map(lambda x: x**2, numbers)
print(list(squared))  # 출력: [1, 4, 9, 16]
```

Lambda Expressions(람다식)

codingOn

- lambda - filter() 함수와 함께 사용

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

def even_number(x):
    return x % 2 == 0

even_numbers = filter(even_number, numbers)
print(list(even_numbers))  # 출력: [2, 4]

# 람다 함수 사용
even_numbers = filter(lambda x: x % 2 == 0, numbers)
print(list(even_numbers))  # 출력: [2, 4]
```

실습6. 함수 종합 프로그래밍

codingOn

- 날씨 데이터 분석 프로그래밍 만들기
- 날씨 데이터는 아래와 같이 2차원리스트 형태로 작성되어있음
- 안쪽 리스트는 날짜, 지역, 온도, 강수량 순으로 표기되어있음

```
weather_data = [
    ["2024-11-20", "서울", 15.2, 0.0],
    ["2024-11-20", "부산", 18.4, 0.0],
    ["2024-11-21", "서울", 10.5, 2.3],
    ["2024-11-21", "부산", 14.6, 1.2],
    ["2024-11-22", "서울", 8.3, 0.0],
    ["2024-11-22", "부산", 12.0, 0.0]
]
```

실습6. 함수 종합 프로그래밍

codingOn

- 아래와 같이 날씨 데이터를 분석할 수 있는 함수들을 생성하여 제작
  - 도시별 평균기온 계산
  - 도시별 최고/최소 기온 찾기
  - 도시별 강수량 분석
  - 데이터 추가
  - 전체 데이터 출력

실습6. 함수 종합 프로그래밍

- 출력예시

```
[날씨 데이터 분석 프로그램]
1. 평균 기온 계산
2. 최고/최저 기온 찾기
3. 강수량 분석
4. 날씨 데이터 추가
5. 전체 데이터 출력
6. 종료
원하는 기능의 번호를 입력하세요: 1
도시 이름을 입력하세요: 서울
서울의 평균 기온: 11.33℃
```

```
[날씨 데이터 분석 프로그램]
1. 평균 기온 계산
2. 최고/최저 기온 찾기
3. 강수량 분석
4. 날씨 데이터 추가
5. 전체 데이터 출력
6. 종료
원하는 기능의 번호를 입력하세요: 2
도시 이름을 입력하세요: 부산
부산의 최고 기온: 18.4℃, 최저 기온: 12.0℃
```

```
[날씨 데이터 분석 프로그램]
1. 평균 기온 계산
2. 최고/최저 기온 찾기
3. 강수량 분석
4. 날씨 데이터 추가
5. 전체 데이터 출력
6. 종료
원하는 기능의 번호를 입력하세요: 3
도시 이름을 입력하세요: 서울
서울의 총 강수량: 2.3mm
서울의 강수량이 있었던 날: 1일
```

실습6. 함수 종합 프로그래밍

codingOn

- 출력예시

```
[날씨 데이터 분석 프로그램]
1. 평균 기온 계산
2. 최고/최저 기온 찾기
3. 강수량 분석
4. 날씨 데이터 추가
5. 전체 데이터 출력
6. 종료
원하는 기능의 번호를 입력하세요: 4
날짜를 입력하세요 (YYYY-MM-DD): 2024-11-23
도시를 입력하세요: 서울
평균 기온을 입력하세요 (℃): 14.4
강수량을 입력하세요 (mm): 0
서울의 날씨 데이터가 추가되었습니다.
```

```
[날씨 데이터 분석 프로그램]
1. 평균 기온 계산
2. 최고/최저 기온 찾기
3. 강수량 분석
4. 날씨 데이터 추가
5. 전체 데이터 출력
6. 종료
원하는 기능의 번호를 입력하세요: 5

현재 저장된 날씨 데이터:
날짜: 2024-11-20, 도시: 서울, 평균 기온: 15.2℃, 강수량: 0.0mm
날짜: 2024-11-20, 도시: 부산, 평균 기온: 18.4℃, 강수량: 0.0mm
날짜: 2024-11-21, 도시: 서울, 평균 기온: 10.5℃, 강수량: 2.3mm
날짜: 2024-11-21, 도시: 부산, 평균 기온: 14.6℃, 강수량: 1.2mm
날짜: 2024-11-22, 도시: 서울, 평균 기온: 8.3℃, 강수량: 0.0mm
날짜: 2024-11-22, 도시: 부산, 평균 기온: 12.0℃, 강수량: 0.0mm
날짜: 2024-11-23, 도시: 서울, 평균 기온: 14.4℃, 강수량: 0.0mm
```

학습정리

- def 를 이용하여 함수를 정의할 수 있다.
- 함수는 매개변수가 있는 함수, 결과값이 있는 함수가 있으며 주로 매개변수와 결과값이 있는 함수로 사용한다.
- 람다식은 lambda라는 키워드를 활용하여 함수를 한줄로 표현하는 방법이다.
- 함수를 이용해서 코드의 중복을 최소화 한다.

다음 수업은?

- 이번수업은 앞으로 파이썬의 함수에 대해서 알아보았습니다.
- 다음시간에는 클래스의 개념에 대해서 설명하겠습니다.

복.습.철.저

수고하셨습니다