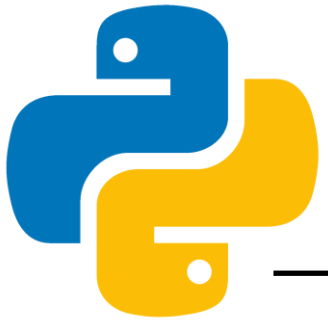

창직 IoT 종합설계입문

파이썬 (5)

알고리즘 (1)

알고리즘 (Algorithm)

- 수학과 컴퓨터 과학, 언어학 또는 관련 분야에서 어떠한 문제를 해결하기 위해 정해진 일련의 절차나 방법을 공식화한 형태로 표현한 것.

- 알고리즘의 복잡도는 점근 표기법 O을 사용하여 나타낼 수 있음.
  $Ex: O(1),\ O(\log n),\ O(n^2),\ \cdots$

알고리즘 (2)

자연수 N 이하의 소수를 구하는 알고리즘

- 1. 소수 (prime number)는 자신보다 작은 두 개의 자연수를 곱하여 만들 수 없는 1보다 큰 자연수.

- 2. 자연수 N보다 작은 임의의 자연수 k에 대하여 k가 소수인지 판별하는 방법.

- 3. 2번의 방법에 대하여 1부터 N까지 N번 반복하여 자연수 N이하의 소수들을 구함.

알고리즘 (3)

```
while True :
  try :
    # 변수 integer에 자연수 N 입력.
    integer = int(input("1 이상의 정수를 입력하세요 : "))
    while integer < 1 :
        print("잘못 입력하셨습니다.")
        integer = int(input("1 이상의 정수를 입력하세요 : "))
    print("입력된 정수 : ", integer)

    break
  except :
    print("잘못 입력하셨습니다.")
    print("정수를 입력하세요.")
```

```
1 이상의 정수를 입력하세요 : -5
잘못 입력하셨습니다.
1 이상의 정수를 입력하세요 : a
잘못 입력하셨습니다.
정수를 입력하세요.
1 이상의 정수를 입력하세요 : 1.0
잘못 입력하셨습니다.
정수를 입력하세요.
1 이상의 정수를 입력하세요 : 7
입력된 정수 :  7
```

알고리즘 (4) – while 반복문

- input() 함수를 통해 integer 변수에 자연수 N에 대한 정보 입력.
- While 반복문을 통해 특정 조건에 부합할 때까지 반복.

```
# while 문의 구조.

condition = True

while condition :
  print("조건은 '참'입니다.")
  break # 무한 루프를 방지하기 위한 break
```

```
조건은 '참'입니다.
```

- condition은 비교, 상등 등 참과 거짓을 판별할 수 있는 구문으로 표현.

알고리즘 (5) – for 문

```
# 소수를 판별하기 위한 알고리즘
if integer != 1 :
  for i in range(2, integer + 1) :
    nCount = 0
    for j in range(2, i) :
      if i % j == 0 :
        nCount += 1
        break
    if nCount == 0 :
      prime_number.append(i)
```

- for 반복문을 통해 정해진 반복 구간만큼 반복.

```
# for 문의 구조.

for i in range(0, 11) :
  print(i, end = " ")
print()

for j in ["대한민국", "중국", "미국"] :
  print(j, end = " ")
print()
```

```
0 1 2 3 4 5 6 7 8 9 10
대한민국 중국 미국
```

알고리즘 (6) – if 문

- range() 함수를 통해 반복 범위 설정. 반복 범위는 컴파운드 자료형으로 설정 가능.

- 조건에 따라 알고리즘의 흐름을 분기하기 위해서 If 조건문을 사용.

```
# if 문의 구조.

nAbs = 3 # nAbs = 결석 횟수

if nAbs >= 5 :
  print("F 학점 입니다.")
else :
  print("결석 횟수가 5번 미만 입니다.")
```

```
결석 횟수가 5번 미만 입니다.
```

알고리즘 (7) – if 문

- 조건이 3개 이상일 때는 if ~ elif ~ else 구문을 사용.
- 조건문은 중첩하여 사용이 가능.

```
# if 문의 구조.

nAbs = 3 # nAbs = 결석 횟수

if nAbs >= 5 :
  print("F 학점 입니다.")
else :
  nScore = 87
  if nScore >= 90 :
    print("A 학점 입니다.")
  elif nScore >= 80 :
    print("B 학점 입니다.")
  elif nScore >= 70 :
    print("C 학점 입니다.")
  elif nScore >= 60 :
    print("D 학점 입니다.")
  else :
    print("F 학점 입니다.")
```

```
B 학점 입니다.
```

알고리즘 (8)

```
while True :
  try :
    # 주어진 정수 N 입력.
    integer = int(input("1 이상의 정수를 입력하세요 : "))
    while integer < 1 :
        print("잘못 입력하셨습니다.")
        integer = int(input("1 이상의 정수를 입력하세요 : "))
    print("입력된 정수 : ", integer)

    # 자연수 N 이하의 소수를 포함하기 위한 빈 리스트 생성.
    prime_number = []

    # 소수를 판별하기 위한 알고리즘
    if integer != 1 :
      for i in range(2, integer + 1) :
        nCount = 0
        for j in range(2, i) :
          if i % j == 0 :
            nCount += 1
            break
        if nCount == 0 :
          prime_number.append(i)

    # 결과 값 출력.
    print("자연수", integer, "이하의 소수는", len(prime_number),"개이며 다음과 같습니다 : ")
    print(prime_number)
    break
  except :
    print("잘못 입력하셨습니다.")
    print("정수를 입력하세요.")
```

```
1 이상의 정수를 입력하세요 : 100
입력된 정수 :  100
자연수 100 이하의 소수는 25 개이며 다음과 같습니다 :
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

루프 제어

Break 문

- **특정 조건을 만족하였을 때, 반복문을 종료.**

```
# Break 문

for i in range(0, 101) :
  if i == 50 :
    print(i)
    break
```

```
50
```

Continue 문

- **특정 조건을 만족하였을 때, 그 반복 요소를 건너뜀.**

```
# Continue 문

for i in range(0, 11) :
  if i % 2 != 0 :
    continue
  print(i, end = " ")
```

```
0 2 4 6 8 10
```

For 문 예제

테일러 급수

- $e^x = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \cdots$

```
# 테일러 급수를 이용하여 자연상수 e를 구하는 방법.
import math

# 근사화 정도를 위한 count 변수 선언.
# 근사화 정도를 변화시키며 비교
count = 20

x = float(input("x의 값을 입력하세요."))
result = 0

for i in range(0, count) :
  result += 1 / math.factorial(i) * (x ** i)

print("exp(", x, ") = ", result)
print(math.exp(x))
```

```
x의 값을 입력하세요.4.5
exp( 4.5 ) =  90.01712525453128
90.01713130052181
```

While 문 예제

Guessing Game

- 임의의 숫자를 맞추는 알고리즘.

```
# Guessing Game
from random import * # 난수 발생을 위한 random 모듈.

target_integer = randint(1, 100)

while True :
  answer = int(input("숫자를 입력하세요 : "))

  if answer == target_integer :
    print("정답입니다.")
    break # 루프를 탈출.
  elif answer > target_integer :
    print(answer, "보다 작은 수를 입력하세요 :")
  else :
    print(answer, "보다 큰 수를 입력하세요 :")
```

```
숫자를 입력하세요 : 50
50 보다 작은 수를 입력하세요 :
숫자를 입력하세요 : 30
30 보다 큰 수를 입력하세요 :
숫자를 입력하세요 : 40
40 보다 큰 수를 입력하세요 :
숫자를 입력하세요 : 45
정답입니다.
```

함수 (1)

함수

- 일련의 기능을 구현한 코드의 묶음.
- 되도록 한 가지의 기능만을 포함하도록 작성.
- def 예약어를 통해 작성하며 함수의 이름, 매개변수, 반환 값의 유무 등으로 구분.

함수 (2)

함수

- Built-in 함수 : 파이썬 프로그램에 이미 내장된 함수.
- User-defined 함수 : 사용자가 특정 기능을 수행하도록 작성한 코드 묶음.

함수 (3)

함수의 구조

- 함수는 매개변수에 따라 출력 값이 결정되는 코드 묶음.

```
'''

def [함수 이름](매개변수 1, 매개변수 2, ...) :
  [함수의 실행 내용 1]
  [함수의 실행 내용 2]
  [함수의 실행 내용 3]
  return 함수의 출력 값

'''
```

함수 (4)

매개변수가 없고, 출력 값 또한 없는 함수

```
def my_function() :
  print("Hi, Python!")

my_function()
```

```
Hi, Python!
```

매개변수가 없지만 출력 값이 있는 함수

```
def my_function() :
  return "my_function()"

print(my_function())
type(my_function())
```

```
my_function()
str
```

함수 (5)

매개변수가 있고, 출력 값이 없는 함수

```
def my_function(name) :
  print("Hello,", name)

my_function("Python")
```

```
Hello, Python
```

매개변수가 있고 출력 값 또한 있는 함수

```
def my_function(base, weight) :
  return base * weight

print(my_function(2, 3))
```

```
6
```

재귀함수 (1)

재귀함수 (Recursive Function)

- 함수 안에서 함수 자기자신을 호출하는 방식.
- 함수 내 탈출 조건이 존재해야 함.

```
def recursive_factorial(integer) :
  if integer == 1 :
    return 1
  elif integer < 1 :
    return False
  else :
    return integer * recursive_factorial(integer - 1)

print(recursive_factorial(6))
```

```
720
```

재귀함수 (2)

피보나치 수열

- 첫째 및 둘째 항이 1이며 그 뒤의 모든 항은 바로 앞 두 항의 합인 수열.

```
# N번째 피보나치 수열 계산.
def fibonacci_numbers(N) :
  if N <= 1 :
    return N
  else :
    return fibonacci_numbers(N - 1) \
    + fibonacci_numbers(N - 2)

N = int(input("자연수 N을 입력하세요 : "))

print(fibonacci_numbers(N))
```

```
자연수 N을 입력하세요 : 4
3
```

람다함수 (1)

람다함수 (Lambda Function)

- 기존의 함수와 다르게 정의할 수 있는 함수.

```
nArea = lambda base, height : base * height

print(nArea(3, 5))

nList = [1, 2, 3, 4]
nResult = list(map(lambda x : x ** 2, nList))
print(nResult)
```

```
15
[1, 4, 9, 16]
```

람다함수 (2)

피보나치 수열

- 첫째 및 둘째 항이 1이며 그 뒤의 모든 항은 바로 앞 두 항의 합인 수열.

```
# N번째 피보나치 수열 계산.

fibonacci_numbers = lambda n : 1 \
if n <= 2 else fibonacci_numbers(n - 1) + fibonacci_numbers(n - 2)

fibonacci_numbers(6)
```

```
8
```