6장. 파이썬 프로그래밍 어떻게 시작해야 할까?

I. 구구단 프로그램 - 2단

프로그램을 만들려면 먼저 '입력'과 '출력'을 생각하라.

❖ **구구단 프로그램 중 2단을 만든다면, 2를 입력값으로 주었을 때 어떻게 출력되어야 할까?**

1. **함수의 이름은?** GuGu
2. **입력 받는 값은?** 2
3. **출력하는 값은?** 2단(2, 4, 6, 8, …, 18)
4. **결과는 어떤 형태로 저장하지?** 연속된 자료형이니까 리스트

I. 구구단 프로그램 - 2단

1. GuGu라는 함수에 2를 입력값으로 주고, result라는 변수에 결과값 넣기

```
result = GuGu(2)
```

2. 결과값 형태 고민하기

✓ 리스트 자료형. 따라서 result = [2, 4, 6, 8, 10, 12, 14, 16, 18]

3. 이름을 GuGu로 지은 함수 만들기

```
def GuGu(n):
    print(n)
```

✓ GuGu(2)처럼 실행하면 2를 출력하게 됨

✓ 즉, 입력값으로 2가 들어오는지 확인함

4. 결과값을 담을 리스트 생성하기

```
def GuGu(n):
    result = [ ]
```

✓ print(n)은 입력이 되는지 확인하기 위한 것이므로 지워도 좋음

5. result 리스트에 요소 추가하기 - append 내장함수 사용

```
def GuGu(n):
    result = []
    result.append(n*1)
    result.append(n*2)
    result.append(n*3)
    result.append(n*4)
    result.append(n*5)
    result.append(n*6)
    result.append(n*7)
    result.append(n*8)
    result.append(n*9)

    return result

print(GuGu(2))
```

result 리스트에 요소 추가

결과값 : [2, 4, 6, 8, 10, 12, 14, 16, 18]

6. 반복문 만들기

- ✓ result.append(n*□)의 □위치에 1부터 9까지 숫자만 다르게 들어가 있음
- ✓ 똑같은 일을 반복할 때는 '반복문'을 사용함

```
>>> i = 1
>>> while i < 10:      ←—— i가 10보다 작은 동안 반복
...         print(i)    ←—— i 값 출력
...         i = i + 1   ←—— i 값 1 증가
```

```
결과값 :
1
2
3
4
5
6
7
8
9
```

7. GuGu 함수 완성하기

```
#  C:/doit/ggd01.py
def GuGu(n):
    result = [ ]
    i = 1
    while i < 10:
        result.append(n * i)
        i = i + 1

    return result

print(GuGu(2))
```

결과값 : [2, 4, 6, 8, 10, 12, 14, 16, 18]

II. 3과 5의 배수 합하기

❖ **1000미만의 자연수에서 3과 5의 배수는 어떻게 구할 수 있을까?**

- 10 미만의 자연수에서 3과 5의 배수를 구하면 3, 5, 6, 9이고, 총합은 23이다.
- 1000 미만의 자연수에서 3의 배수와 5의 배수의 총합을 구하라

1. **입력 받는 값은?** 1부터 999까지(1000 미만의 자연수)
2. **출력하는 값은?** 3의 배수와 5의 배수의 총합
3. **생각해 볼 것은?**
   하나. 3의 배수와 5의 배수는 어떻게 찾지?
   둘. 3의 배수와 5의 배수가 겹칠 때는 어떻게 하지?

II. 3과 5의 배수 합하기

1. 1000 미만의 자연수 구하기

① 변수에 초기값 1을 준 후 루프를 돌리며 1씩 증가시켜서 999까지 진행

```
n = 1
while n < 1000:
    print(n)
    n += 1
```

② range함수를 사용

```
for n in range(1, 1000):
    print(n)
```

2. 3과 5의 배수 구하기

- ✓ 1부터 1000까지의 수 중 3으로 나누었을 때 나누어 떨어지는 경우, 즉 3으로 나누었을 때 나머지가 0인 경우가 3의 배수임
- ✓ 따라서 %연산자를 이용하면 3의 배수를 찾을 수 있음

```
for n in range(1, 1000):
    if n % 3 == 0:
        print(n)
```

- ✓ 5의 배수는 n % 5가 0이 되는 수로 구할 수 있음

II. 3과 5의 배수 합하기

3. 3과 5의 배수에 해당하는 수를 result 변수에 계속해서 더해 주기

```
# C:/doit/multi01.py
result = 0
for n in range(1, 1000):
    if n % 3 == 0 or n % 5 == 0:
        result += n
print(result)
```

← 1부터 999까지 n에 대입하며 반복

← n을 3으로 나눈 나머지가 0이거나 n을 5로 나눈 나머지가 0이라면

✓ 15와 같이 3의 배수도 되고 5의 배수도 되는 값이 이중으로 더해지지 않기 위해 or 연산자를 사용함

III. 게시판 페이징하기

❖ A씨는 게시판 프로그램을 작성하고 있는데, 게시물의 총 건수와 한 페이지에 보여줄 게시물 수를 입력으로 주었을 때 총 페이지 수를 출력하는 프로그램이 필요함

1. **함수의 이름은?** getTotalPage
2. **입력 받는 값은?** 게시물의 총 건수(m), 한 페이지에 보여줄 게시물 수(n)
3. **출력하는 값은?** 총 페이지수

- 게시물의 총 건수가 5이고 한 페이지에서 보여 줄 게시물 수가 10이면, 총 페이지수는 1
- 게시물의 총 건수가 15이고 한 페이지에서 보여 줄 게시물 수가 10라면, 총 페이지수는 2

게시물의 총 건수(m)	페이지당 보여줄 게시물 수(n)	총 페이지 수
5	10	1
15	10	2
25	10	3
30	10	3

1. 총 페이지수 구하기

✓ 총 건수(m)를 한 페이지에 보여줄 게시물 수(n)로 나누고 1을 더하면 총 페이지 수를 얻을 수 있음

```
총 페이지수 = 총 건수 / 한 페이지당 보여줄 건수 + 1
```

2. 공식 적용하고 확인하기

✓ m을 n으로 나눌 때 소수점 아래 자리를 버리기 위해 / 대신 // 연산자를 사용함

```
def getTotalPage(m, n):
    return m // n + 1

print(getTotalPage(5,10))    ←—— 첫 번째 케이스, 1이 출력됨
print(getTotalPage(15,10))   ←—— 두 번째 케이스, 2이 출력됨
print(getTotalPage(25,10))   ←—— 세 번째 케이스, 3이 출력됨
print(getTotalPage(30,10))   ←—— 네 번째 케이스, 3이 출력되어야 하는데 4가 출력되어 실패함
```

✓ 첫 번째, 두 번째, 세 번째 케이스는 공식에 맞게 결과가 출력됨

✓ 네 번째 케이스는 총 건수가 30이고 한 페이지에 보여줄 건수가 10이라면, 총 페이지 수는 3되어야 함

3. 실패 케이스 해결하기

✓ 실패 케이스는 총 건수를 한 페이지당 보여 줄 건수로 나눈 나머지 값이 0이 될 때 발생함을 유출할 수 있으며, 이를 해결하려면 소스 코드를 변경해야 함

```
# C:/doit/getpage.py
def getTotalPage(m, n):
    if m % n == 0:
        return m // n
    else:
        return m // n + 1

print(getTotalPage(5,10))
print(getTotalPage(15,10))
print(getTotalPage(25,10))
print(getTotalPage(30,10))
```

← 네 번째 케이스, 3이 출력됨

✓ 나누었을 때 나머지가 0인 경우는 나누기의 몫만 돌려주고 그 외의 경우에는 1을 더하여 돌려주도록 변경함

IV. 간단한 메모장 만들기

❖ **원하는 메모를 파일에 저장하고 추가 및 조회가 가능한 간단한 메모장을 만듦**

1. **필요한 기능은?** 메모 추가하기, 메모 조회하기
2. **입력 받는 값은?** 메모의 내용, 프로그램 실행 옵션
3. **출력하는 값은?** memo.txt

▪ 다음 명령을 실행했을 때 메모를 추가할 수 있도록 만들어 봄

```
c:₩doit>python memo.py –a “Life is too short”
```

✓ memo.py는 우리가 작성할 프로그램 이름임
✓ -a는 이 프로그램의 실행 옵션임
✓ “Life is too short”는 추가할 메모 내용임

1. 입력으로 받은 옵션과 메모를 출력하는 소스 코드를 작성하기

```
# C:/doit/memo.py
import sys

option = sys.argv[1]
memo = sys.argv[2]

print(option)
print(memo)
```

- ✓ sys.argv는 프로그램 실행 시 입력된 값을 읽어 들일 수 있는 라이브러리임
- ✓ sys.argv[0]는 프로그램 이름인 memo.py이므로 만들려는 기능에는 필요 없음
- ✓ sys.argv[1]은 프로그램 실행 옵션 값임
- ✓ sys.argv[2]는 메모 내용임

2. 옵션과 메모를 출력하는 소스 코드를 확인하기

```
C:₩doit>python memo.py –a “Life is too short”
-a
Life is too short
```

✓ 입력으로 전달한 옵션과 메모 내용이 그대로 출력되는 것을 확인할 수 있음

3. 입력으로 받은 메모를 파일에 쓰도록 소스 코드를 변경하기

```
# C:/doit/memo.py
import sys

option = sys.argv[1]

if option == '-a':
    memo = sys.argv[2]
    f = open('memo.txt', 'a')
    f.write(memo)
    f.write('₩n')
    f.close()
```

- ✓ 옵션 '-a'인 경우에만 memo값을 읽어 memo.txt 파일에 그 값을 쓰도록 함
- ✓ 메모는 항상 새로운 내용이 작성되는 것이 아니라 한 줄씩 추가되어야 하므로 파일 열기 모드를 'a'로 함
- ✓ 메모를 추가할 때마다 다음 줄에 저장되도록 하기 위해서 줄바꿈 문자(₩n)도 추가로 파일에 쓰게 했음

4. 명령을 수행하고 메모 확인하기

```
C:₩doit>python memo.py –a "Life is too short"
C:₩doit>python memo.py –a "You need python"
```

✓ 파일에 메모가 기입되었는지 확인함

```
C:₩doit>type memo.txt
Life is too short
You need python
```

✓ 추가된 메모가 정상적으로 저장된 것을 볼 수 있음

5. 작성한 메모를 출력하는 소스 코드를 추가하기

✓ 메모 출력은 다음과 같이 동작하도록 만들어 봄

```
python memo.py –v
```

✓ 메모 추가는 –a 옵션을 사용함

✓ 메모 출력은 –v 옵션을 사용함

✓ 메모 출력을 위해서 소스 코드를 추가하기

```
# C:/doit/memo.py
import sys

option = sys.argv[1]

if option == '-a':
    memo = sys.argv[2]
    f = open('memo.txt', 'a')
    f.write(memo)
    f.write('₩n')
    f.close()
elif option == '-v':
    f = open('memo.txt')
    memo = f.read()
    f.close()
    print(memo)
```

✓ 옵션으로 –v가 들어온 경우 memo.txt 파일을 읽어서 출력함

6. 소스 코드를 수정한 후 명령을 수행하기

```
C:₩doit>python memo.py –v
Life is too short
You need python
```

✓ 입력했던 메모가 그대로 출력되는 것을 확인할 수 있음

V. 탭을 4개의 공백으로 바꾸기

❖ **문서 파일을 읽어서 그 문서 파일 내에 있는 탭(tab)을 공백(space) 4개로 바꿈**

1. **필요한 기능은?** 문서 파일 읽어 들이기, 문자열 변경하기
2. **입력 받는 값은?** 탭을 포함한 문서 파일
3. **출력하는 값은?** 탭이 공백으로 수정된 문서 파일

- 다음과 같은 형식으로 프로그램이 수정되도록 만들어 봄

```
python tabto4.py src dst
```

- tabto4.py는 우리가 작성해야 할 파이썬 프로그램 이름임
- src는 탭을 포함하고 있는 원본 파일 이름임
- dst는 파일 내의 탭을 공백 4개로 변환한 결과를 저장할 파일 이름임

V. 탭을 4개의 공백으로 바꾸기

1. tabto4.py 파일을 작성하기

```
# C:/doit/tabto4.py
import sys

src = sys.argv[1]
dst = sys.argv[2]

print(src)
print(dst)
```

- ✓ sys.argv를 이용하여 입력값을 확인하도록 만든 코드임

2. 입력값이 정상적으로 출력되는지 확인하기

```
C:₩doit>python tabto4.py a.txt b.txt
a.txt
b.txt
```

✓ 입력으로 전달한 a.txt와 b.txt가 정상적으로 출력되는 것을 확인할 수 있음

3. 테스트를 위한 원본 파일(탭을 포함하는 파일)인 a.txt를 작성하기

```
Life	is	too	short
You	need	python
```

✓ 각 단어들은 탭(₩t)문자로 분리되도록 입력해야 함

4. a.txt 파일을 읽어서 탭을 공백 4개로 변환할 수 있도록 소스 코드를 변경하기

```
# C:/doit/tabto4.py
import sys
src = sys.argv[1]
dst = sys.argv[2]

f = open(src)
tab_content = f.read()
f.close()

space_content = tab_content.replace("₩t", " "*4)
print(space_content)
```

- ✓ src에 해당되는 입력 파일을 읽어서 그 내용을 tab_content라는 변수에 저장함
- ✓ 문자열의 replace 함수를 이용하여 탭(₩t)을 4개의 공백으로 변경함

5. tabto2.py 변경한 후 명령을 수행하기

```
C:₩doit>python tabto4.py a.txt b.txt
Life    is    too    short
You    need    python
```

- ✓ 탭 문자가 공백 4개로 변경되어 출력될 것임
- ✓ 하지만 탭과 공백의 차이점을 눈으로 확인할 수 없으므로 탭이 정상적으로 공백으로 변경되었는지 확인하기 어려움

6. 변경된 내용을 b.txt 파일에 저장할 수 있도록 프로그램을 변경하기

```
# C:/doit/tabto4.py
import sys

src = sys.argv[1]
dst = sys.argv[2]

f = open(src)
tab_content = f.read()
f.close()

space_content = tab_content.replace("₩t", " "*4)

f = open(dst, 'w')
f.write(space_content)
f.close()
```

✓ 탭이 공백으로 변경된 space_content를 출력 파일인 dst에 쓰도록 코드를 수정함

7. 프로그램 실행하기

```
C:₩doit>python tabto4.py a.txt b.txt
```

- ✓ b.txt 파일이 C:₩doit 디렉터리에 생성됨
- ✓ 에디터로 b.txt 파일을 열어 탭이 4개의 공백 문자로 변경되었는지 확인함

VI. 하위 디렉터리 검색하기

❖ **특정 디렉터리부터 시작해서 그 하위 모든 파일 중 파이썬 파일(*.py)만 출력해 주는 프로그램을 만들려면 어떻게 해야 할까?**

1. **함수의 이름은?** search
2. **입력 받는 값은?** 디렉터리("C:/")
3. **출력하는 값은?** 파이썬 파일(*.py)

1. sub_dir_search.py 파일을 작성하기

```
# C:/doit/sub_dir_search.py

def search(dirname):
    print(dirname)

search("C:/")
```

✓ search라는 함수를 만들고 시작 디렉터리를 입력 받도록 코드를 작성함

2. 디렉터리에 있는 파일을 검색할 수 있도록 소스를 변경하기

```
# C:/doit/sub_dir_search.py
import os

def search(dirname):
        filenames = os.listdir(dirname)
        for filename in filenames:
            full_filename = os.path.join(dirname, filename)
            print(full_filename)

search("C:/")
```

- ✓ **os.listdir**을 사용하면 해당 디렉터리에 있는 파일들의 리스트를 구할 수 있음
- ✓ 파일 리스트는 파일 이름만 포함되어 있으므로 경로를 포함한 파일 이름을 구하기 위해서는 입력으로 받은 dirname을 앞에 덧붙여 주어야 함
- ✓ **os.path.join** 함수는 디렉터리와 파일 이름을 이어주는 전체 경로를 구해 줌

✓ 위 코드를 수행하면 "C:/" 디렉터리에 있는 파일이 출력될 것임

```
[디렉터리 출력 예]
C:₩$Recycle.Bin
C:₩$WINDOWS.~BT
C:₩$Windows.~WS
C:₩adb
C:₩AMD
… 생략 …
```

3. "C:/" 디렉터리에 있는 확장자가 .py인 파일만을 출력하도록 소스 코드를 변경하기

```
# C:/doit/sub_dir_search.py
import os

def search(dirname):
    filenames = os.listdir(dirname)
    for filename in filenames:
        full_filename = os.path.join(dirname, filename)
        ext = os.path.splitext(full_filename)[-1]
        if ext == '.py':
            print(full_filename)

search("C:/")
```

- ✓ 파일에서 확장자를 추출하기 위해 os 모듈의 os.path.splitext 함수를 사용하였음
- ✓ **os.path.splitext**는 파일 이름을 확장자를 기준으로 두 부분으로 나눔
- ✓ 따라서 os.path.splitext(full_filename)[-1]은 해당 파일의 확장 이름이 됨
- ✓ "C:/" 디렉터리에 파이썬 파일이 없다면 아무것도 출력되지 않음

4. "C:/" 하위 디렉터리(sub directory)도 검색이 가능하도록 소스 코드를 변경하기

```
# C:/doit/sub_dir_search.py
import os

def search(dirname):
    try:
        filenames = os.listdir(dirname)
        for filename in filenames:
            full_filename = os.path.join(dirname, filename)
            if os.path.isdir(full_filename):
                search(full_filename)
            else:
                ext = os.path.splitext(full_filename)[-1]
                if ext == '.py':
                    print(full_filename)
    except PermissionError:
        pass

search("C:/")
```

VI. 하위 디렉터리 검색하기

- ✓ try … exception PermissionError로 함수 전체를 감싼 이유는 os.listdir를 수행할 때 권한이 없는 디렉터리에 접근하더라도 프로그램이 오류로 종료되지 않고 수행되도록 하기 위해서임
- ✓ **os.path.isdir** 함수는 full_filename이 디렉터리인지 파일인지 구분하기 위해 사용함
- ✓ 디렉터리일 경우 해당 경로를 입력받아 다시 search함수를 호출함 (재귀 호출)
- ✓ 해당 디렉터리의 파일이 디렉터리일 경우, 다시 search함수를 호출하면 해당 디렉터리의 하위 파일을 다시 검색하기 시작하므로 결국 모든 파일들을 검색할 수 있게 됨

VII. 행운권 추첨 프로그램

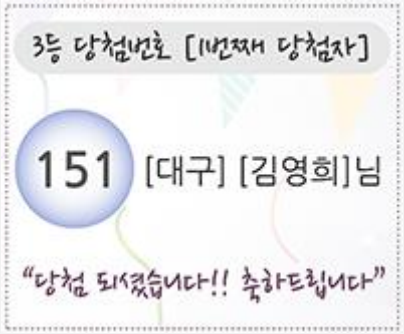
❖ **1부터 200 사이의 번호 중에서 난수 값을 발생시켜 행운번호를 출력함**

1. **함수의 이름은?** getNumber
2. **입력 받는 값은?**

   행운권 추첨을 시작합니다(y/n)
3. **출력하는 값은?**

   당첨번호 : 151

   당첨되셨습니다!! 축하합니다



1. 행운권 추첨 프로그램(1/2)

```
#lucky01.py
import random

## 함수 선언 부분 ##
def getNumber() :
    return random.randint(1, 200)

## 메인 프로그램 부분 ##
print("** 행운권 추첨을 시작합니다 ** ") ;
num = getNumber()
print("당첨번호 : %d" % num)
print("당첨되셨습니다!! 축하합니다₩n")
```

1. 행운권 추첨 프로그램(2/2)

```
#lucky02.py
import random

## 함수 선언 부분 ##
def getNumber() :
    return random.randint(1, 200)

## 변수 선언 부분 ##
lotto = [ ]
num = 0

## 메인 프로그램 부분 ##
while True :
    start = input("** 행운권 추첨을 시작합니다(y/n) **") ;
    if start == "n" :
        break

    num = getNumber()

    if lotto.count(num) == 0 :
        lotto.append(num)

    print("          당첨번호 : %d" % num)
    print("    당첨되셨습니다!! 축하합니다₩n")
```

VII. 행운권 추첨 프로그램

❖ **1부터 46 사이의 번호 중에서 난수 값을 발생시켜 6개의 로또번호를 출력함**

1. **함수의 이름은?** getNumber
2. **입력 받는 값은?**

   로또 추첨을 시작합니다
3. **출력하는 값은?**

   당첨된 로또 번호 → 13 14 19 26 40 43

1. 로또 추첨 프로그램

```
# C:/doit/lotto.py
import random

## 함수 선언 부분 ##
def getNumber() :
    return random.randrange(1, 46)

## 변수 선언 부분 ##
lotto = [ ]
num = 0

## 메인 프로그램 부분 ##
print("** 로또 추첨을 시작합니다 ** ₩n") ;

while True :
    num = getNumber()

    if lotto.count(num) == 0 :
        lotto.append(num)

    if len(lotto) >= 6 :
        break

print("당첨된 로또 번호 :", end = ' ')
lotto.sort()
for i in range(0, 6) :
    print("%d" % lotto[i], end = ' ')
```