파이썬 기본 실전 프로그래밍

01 구구단 프로그램 만들기

In [1]:

```
for i in range(1, 10):
    for j in range(1, 10):
        print("{} x {} = {}".format(i, j, i*j))
    print()
```

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9

2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18

3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27

4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36

5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45

6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
```

```
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54

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

8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72

9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
```

02. 게시물의 페이지 확인 계산 프로그램

한 페이지에 표시할 수 있는 게시물을 입력했을 때, 게시물 페이지는 얼마인가?

In [3]:

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

In [4]:

```
contents = 5  # 현재 게시물 수
one_page = 10 # 한 페이지 표시 가능 건수

print( NumPage(contents, one_page), "페이지에 표시됩니다.")   # 5건, 표시할 건수 10
```

```
1 페이지에 표시됩니다.
```

In [5]:

```
contents = 45  # 현재 게시물 수
one_page = 10  # 한 페이지 표시 가능 건수

print( NumPage(contents, one_page), "페이지에 표시됩니다.")   # 5건, 표시할 건수 10
```

5 페이지에 표시됩니다.

[실습] 추가 기능

- m(건수), n(표시할 건수)를 입력받는다.
- 페이지 수와 마지막 페이지에 표시될 건수를 확인하는 프로그램을 만들어보자.

In [7]:

```
def NumPage(m, n):
    page = m // n+1
    last_page_num = m % n
    return page, last_page_num
```

In [8]:

```
m1 = int(input("총 게시물 수 : "))
n1 = int(input("한 페이지 게시물 건수 : "))

page, last_num = NumPage(m1, n1)
print()
print(f"게시물 전체 페이지 : {page} 마지막 페이지 게시물 건수 : {last_num}" )
```

```
총 게시물 수 : 85
한 페이지 게시물 건수 :10
게시물 전체 페이지 : 9, 마지막 페이지 게시물 건수 : 5
```

03 하위 디렉터리 검색 후, 원하는 파일 형태만 출력

- 내용 : 확장자가 html인 파일만 출력시켜보기

- os.walk(대상경로) : 하위의 폴더들을 반복적으로 탐색
  - 인자로 전달된 경로에 대해 3개의 값의 tuple로 넘겨준다.
    - path : dir과 files가 있는 경로
    - dirs : path아래에 있는 폴더들
    - files : root아래에 있는 파일들

- os.path.splitext(filename) : 파일의 확장자를 알아보기
  - 예제 결과 : ('D:\Github\CLASS_PYTHON_START01_python_start', '.html')
  - 두번째 값이 파일의 확장자

In [18]:

```
import os

# 확인하고자 하는 대상 폴더
test_path = "D:₩GithubWCLASS_PYTHON_START"

for (path, dir, files) in os.walk(test_path):
    for filename in files:
        ext = os.path.splitext(filename)[-1]  # 경로와 확장자로 구분
        if ext=='.html':   # 확장자명
            print("%s%s" % (path, filename))
```

```
D:₩GithubWCLASS_PYTHON_START01_01_python_start.html
D:₩GithubWCLASS_PYTHON_START01_02_python_start_if_for_list.html
D:₩GithubWCLASS_PYTHON_START01_03_python_start_file_220408.html
D:₩GithubWCLASS_PYTHON_START01_04_python_start_classA.html
D:₩GithubWCLASS_PYTHON_START01_04_python_start_classB_2204.html
D:₩GithubWCLASS_PYTHON_START01_04_python_start_fnc_module_220412.html
D:₩GithubWCLASS_PYTHON_START02_01_programming_2204.html
D:₩GithubWCLASS_PYTHON_START03_01_Seaborn_Basic.html
D:₩GithubWCLASS_PYTHON_START03_02_ml_start.html
D:₩GithubWCLASS_PYTHON_START03_03_DB.html
```

04 정보의 암호화 하기 - 일반적 방법

In [19]:

```
# 사전 확인
# 문자열.isdigit() : 숫자인지 확인하는 함수
# 리스트.append() : 숫자인지 확인하는 함수
```

In [20]:

```
data = """
park,10234-1422351
lim ,22342-1422251
"""
```

In [21]:

```
result = []
for line in data.split("₩n"):  # 한줄 단위로 구분
    word_result = []

    one_line = line.split(",")  # 한줄 데이터를 공백으로 나누기

    for word in one_line:
        if len(word) == 13 and word[:5].isdigit() and word[6:].isdigit():
            word = word[:6] + "-" + "*******"
        word_result.append(word)
    result.append(" ".join(word_result))
print("₩n".join(result))
```

```
park 10234--*******
lim  22342--*******
```

04 정보의 암호화 하기 - 정규 표현식

In [12]:

```
import re

data = """
park 80112-1422351
lim  81012-1422251
"""

pat = re.compile("(₩d{5})[-]₩d{7}")  # 숫자 5개, 숫자 7개
print(pat.sub("₩g<1>-*******", data))
```

```
park 80112-*******
lim  81012-*******
```

05 예외처리 exception 알아보기

5-1 문자로 나눌 때, 이에 대한 에러 메시지 출력하기

In [13]:

```
def divide(a, b):
    return a/b

try:
    c = divide(5, 'string')
except ZeroDivisionError:
    print("두번째 인자는 0이어서는 안됩니다.")
except TypeError:
    print("모든 인자는 숫자여야 합니다.")
except:
    print("무슨 에러?")
```

모든 인자는 숫자여야 합니다.

In [14]:

```
def divide(a, b):
    return a/b

try:
    c = divide(5, 'kkk')
except ZeroDivisionError:
    print("두번째 인자는 0이어서는 안됩니다.")
except TypeError:
    print("모든 인자는 숫자여야 합니다.")
except Exception:
    print("무슨 에러?")
```

모든 인자는 숫자여야 합니다.

- (실습해보기) 나누는 값을 0으로 할 때는 어떻게 되는가?

5-2 예외처리 try, except, else, finally

```
try:
    실행할 코드
except:
    예외가 발생했을 때 처리하는 코드
else:
    예외가 발생하지 않았을 때 실행할 코드
finally:
    예외 발생 여부와 상관없이 항상 실행할 코드
```

In [15]:

```
def divide(a, b):
    return a/b

try:
    c = divide(5, 1)
except ZeroDivisionError:
    print("두번째 인자는 0이어서는 안됩니다.")
except TypeError:
    print("모든 인자는 숫자여야 합니다.")
else:
    print("예외 발생 없이 정상 실행")
finally:
    print("모든 프로그램 정상 실행하였음")
```

```
예외 발생 없이 정상 실행
모든 프로그램 정상 실행하였음
```

5-3 예외 발생시키기

기본 구조

```
try:
    ...
except [발생 오류[as 오류 메시지 변수]]:
    ...
```

try 블록 수행 중 오류가 발생하면 except 블록이 수행됩니다. 다만, try 블록에서 오류가 발생하지 않는다면 except 블록은 수행되지 않습니다.

기본 구조

```
try:
    # 실행문
except:
    # 오류 발생시 실행


try:
    # 실행문
except 발생 오류:
    # 오류 발생시 실행

try:
    # 실행문
except 발생 오류 as 오류 메시지 변수:
    # 오류 발생시 실행
```

raise 는 강제로 에러를 발생시킬 때 사용한다.

- raise [에러이름]
  - raise문에 의해 [에러] Error가 발생
- raise Exception
  - raise문에 의해 Exception Error가 발생

In [35]:

```
try:
    x = int(input('5의 배수를 입력하세요: '))
    if x % 5 != 0:                                  # x가 3의 배수가 아니면
        raise Exception('5의 배수가 아닙니다.')     # 예외를 발생시킴
    print(x)
except Exception as e:                               # 예외가 발생했을 때 실행됨
    print('예외가 발생했습니다.', e)
```

```
5의 배수를 입력하세요: 33
예외가 발생했습니다. 5의 배수가 아닙니다.
```

특정 오류 회피시에

In [38]:

```
# 입력시에 문자열을 입력
x = int(input('5의 배수를 입력하세요: '))
print(x)
```

```
5의 배수를 입력하세요: aaaa

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-38-3fd676ef797d> in <module>
      1 # 입력시에 문자열을 입력
----> 2 x = int(input('5의 배수를 입력하세요: '))
      3 print(x)

ValueError: invalid literal for int() with base 10: 'aaaa'
```

In [39]:

```
try:
    x = int(input('5의 배수를 입력하세요: '))
    print(x)
except ValueError:            # 특정 오류 회피할때
    pass
```

```
5의 배수를 입력하세요: aaa
```

어떤 에러 메시지인지 알 수 있을까?

- Exception : 예외 발생 해당되는 에러가 없을 때 실행된다.
- Exception as e : 이 구문을 통해 에러 내용을 e를 통해 출력 가능

In [40]:

```
try:
    x = int(input('5의 배수를 입력하세요: '))
    print("내가 입력한 값 : ", x)
except Exception as e:                              # 예외가 발생했을 때 실행됨
    print('예외가 발생했습니다. ₩n[에러내용]', e)
```

5의 배수를 입력하세요: aaab
예외가 발생했습니다.
[에러내용] invalid literal for int() with base 10: 'aaab'

다중 except

In [2]:

```
try:
    3 / 0
except IndexError as e:
    print("인덱스가 안 맞습니다.", e)
except ZeroDivisionError as e:
    print("0으로 나누면 안됩니다.", e)

print("프로그램의 정상 종료!")
```

0으로 나누면 안됩니다. division by zero
프로그램의 정상 종료!

else절

- 예외가 발생하지 않았을 때, 실행되는 부분

In [4]:

```
try:
    age=int(input('나이를 입력하세요: '))
except:
    print('입력이 정확하지 않습니다.')
else:
    if age <= 18:
        print('미성년자는 출입금지입니다.')
    else:
        print('입장 가능합니다.')
```

예외가 발생 안됨!
프로그램이 정상적으로 종료됩니다!

finally 절은 예외와 상관없이 언제나 실행

```
try:
    실행문1
except 발생 오류1:
    실행문2
except 발생 오류2:
    실행문3
finally:
    실행문
```

In [28]:

```
try:
    3 / 0
except IndexError as e:
    print("인덱스가 안 맞습니다.", e)
except ZeroDivisionError as e:
    print("0으로 나누면 안됩니다.", e)
finally:
    print("예외에 상관없이 언제나 실행!")

print("프로그램이 정상적으로 종료됩니다!")
```

```
0으로 나누면 안됩니다. division by zero
예외에 상관없이 언제나 실행!
프로그램이 정상적으로 종료됩니다!
```

보통 finally절은 사용한 리소스를 close해야 할 때에 많이 사용

In [29]:

```
f = open('test.txt', 'w')
try:
    f.write("ttt")
finally:
    f.close()
```