Python 기초 이론 및 실습

(주)딥비전

1. Python 소개

Python 소개

- **Python의 역사**
  - 1989년: 네덜란드 국립연구소에서 구현 시작
  - 1991년: 네덜란드 개발자 귀도 반 로섬(Guido van Rossum)에 의해 발표
  - 2000년: python 2.0 배포
  - 2008년: python 3.0 배포 (이전 버전과 호환 되지 않음)

- **Python의 장점**
  1) **고수준 언어**
     - Matlab과 매스플롯 라이브러리 처럼 행렬 수학 처리 가능
     - 높은 생산성: 텍스트 조작 및 데이터 처리에 이상적임
     - 재사용성: Object oriented, Procedural, functional 모두 사용 가능

  2) **쉬운 난이도**
     - 간단하고 간결한 문법, 초보자도 쉽게 이해 가능

  3) **오픈소스**
     - 무료 소프트웨어, 다른 개발자들의 소스코드 활용 가능,
       공동작업 시 편리함

1. Python 소개

Python 소개

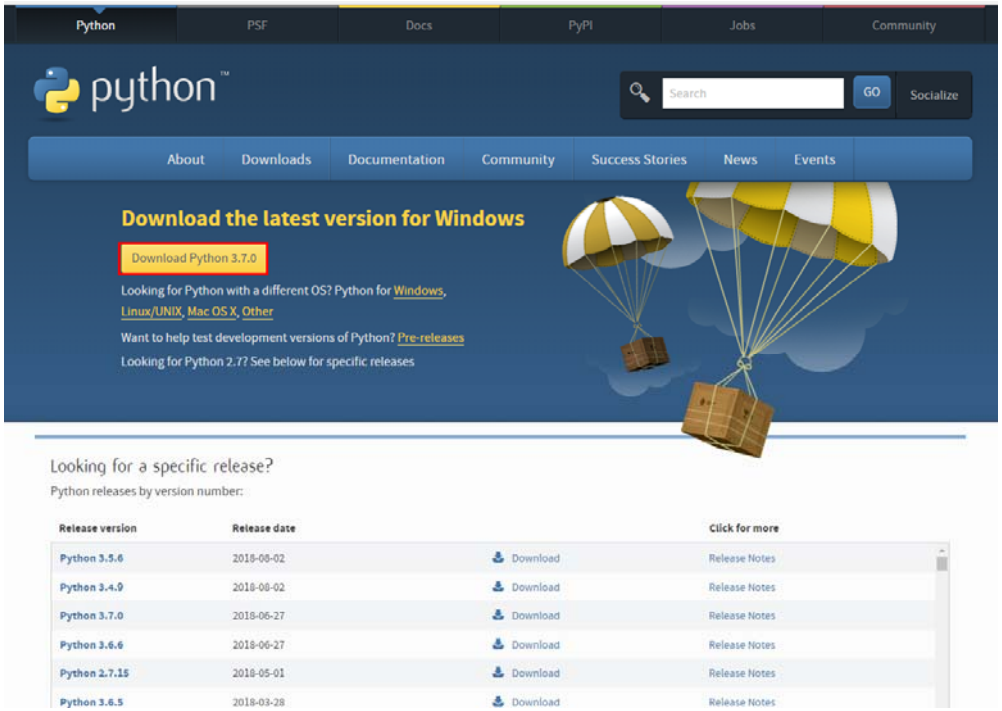
- **Python의 활용 분야**
  1) 웹 프로그래밍
  2) 수치 연산 프로그래밍
  3) C/C++과 결합 가능
  4) GUI 프로그래밍
  5) 시스템 유틸리티

- **Python을 추천하는 이유**
  1) 낮은 진입 장벽
  2) Sub 언어로의 뛰어난 활용성
  3) 온라인에 공개된 풍부한 자료
  4) 고수준언어로 알고리즘 개발 가능
  5) 뛰어난 확장성

2. Python 시작하기

Python 시작하기

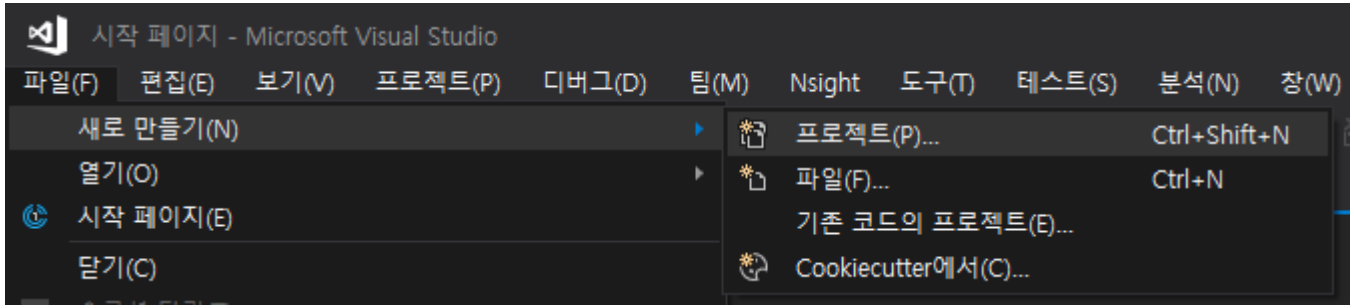
- http://python.org/download

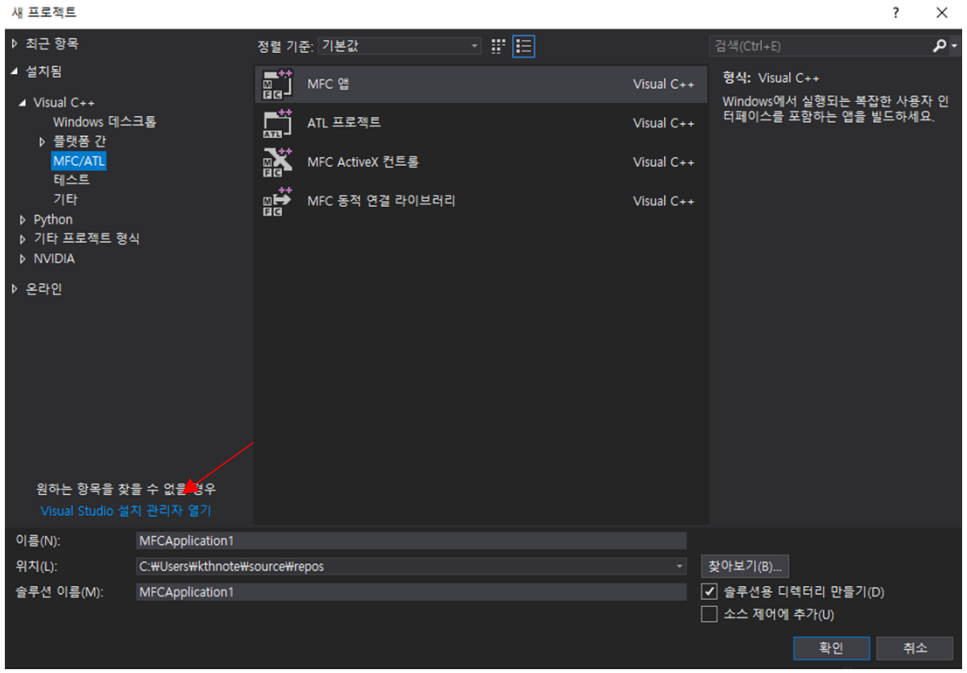


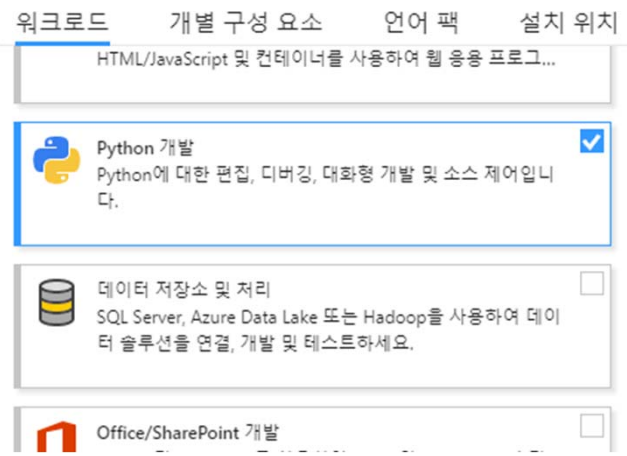
- 다른 라이브러리와 호환성을 고려한 설치 권장
  (2020.02.04. 기준 Python 3.7.6)

2. Python 시작하기

Python 시작하기







3. Python 입문

Python 입문

- **코드블록**
  1) 파이썬은 타 언어와 달리 탭이나 스페이스바를 통해 코드를 구분
  2) if, for, while문과 같이 특정 키워드 다음에 나오는 문장은 콜론(:)으로 구분

- **라인구분**
  1) 파이썬에서 선언한 함수나 클래스는 2개의 공백 라인을 추가하여 구분
  2) 매서드는 한 개의 공백 라인으로 구분

- **명명규칙**
  1) 함수, 변수, Attribute는 소문자로 단어 간은 밑줄을 사용하여 연결
  2) 클래스의 protected instance attribute는 하나의 밑줄로 시작
  3) 클래스의 private instance attribute는 2개의 밑줄로 시작
  4) 인스턴스 메서드는(객체 자신을 가리키기 위해)self를 사용
  5) 클래스 메서드는 (클래스 자신을 가리키기 위해)cls를 사용

- **문장 표현식**
  1) if, for, while 블록 문장은 한 줄에 작성하지 않는다.
  2) import문은 항상 파일의 상단에 위치한다.

3. Python 입문

Python 입문

- **변수명**
  1) 영문자(대, 소문자 구분), 숫자, 언더바(_) 사용 가능
  2) 첫 자리에는 숫자 사용 불가능
  3) 파이썬 키워드는 변수 명으로 사용 할 수 없음
  (ex: False, None, True, and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield 등)

- **변수에 값 대입:** 변수에 값을 대입할 때에는 =을 사용
  ex: A = 1 # A에 1을 저장
  B = 2 # B에 2를 저장

- **숫자:** 파이썬에서 숫자는 정수, 실수, 복소수등 다양하게 표현할 수 있음
  ex: pi = 3.14 # 실수를 저장
  complex = 1+2j # 복소수를 저장
  hex_i = 0xFF # 10진수 255를 16진수로 저장
  oct_j = 0o10 # 10진수 8을 8진수로 저장
  bin_k = 0b1000 # 10진수 8을 2진수로 저장

- **문자열:** 문자열은 연속된 문자를 표현한 것으로 따옴표(“”) 혹은 싱글 따옴표(‘’)로 묶어서 표현 가능
  ex: Myname = “Eric”

Python 입문

- **연산자**

1) 산술 연산자

a) 사칙 연산자: +, -, *, /
b) 제곱 연산자: **
c) 나머지 산출: %(Modulus)
d) 나누기에 소수점 이하를 버리는 연산자: //(Floor Division)

2) 비교 연산자: 등호(==), 같지 않음(!=), 부등호(<, >, <=, >=)등

3) 할당 연산자: 변수에 값을 할당하기 위하여 사용

ex: a = a * 10
a *= 10 # 위와 동일한 표현

4) 논리 연산자:

a) and: 양쪽의 값이 모두 참인 경우 참
b) or: 어느 한쪽만 참이면 참
c) not: 참이면 거짓으로 거짓이면 참으로

Ex3-1.

```
a = [1, 2, 3, 4]
b = 3 in a          # True
print(b)
```

5) 멤버쉽 연산자: 멤버쉽 연산자에는 in, not in이 있는데 이는 Operand가 우측 컬렉션에 속해 있는지 아닌지를 체크

Python 입문

- **연산자 적용 우선순위(Order of Operations)**
  : 1개 이상의 연산자가 표현식에 사용되면 연산자 평가 순서는 우선순위 규칙(rules of precedence)에 따른다.

  1) 괄호(Parentheses)는 가장 높은 순위를 가지고 원하는 순위에 맞춰 실행 할 때 사용된다. 괄호 내의 식이 먼저 실행되기 때문에 2*(3-1)은 4가 정답이고, (1+1)**(5-2)는 8이 정답이다. 또한, 괄호는 표현식의 가독성을 높이기 위하여 사용되기도 한다. (minute*100)/60은 실행 순서가 결과 값에 영향을 주지 않지만 가독성이 상대적으로 더 좋다.

  2) 지수승(Exponentiation)이 다음으로 높은 우선순위를 가진다. 그래서 2**1+1은 4가 아니라 3이고, 3*1**3은 27이 아니고 3이다.

  3) 곱셈(Multiplication)과 나눗셈(Division)은 동일한 우선순위를 가지지만, 덧셈(Addition), 뺄셈(Substraction)보다 높은 우선 순위를 가진다. 덧셈과 뺄셈은 같은 실행 우선순위를 갖는다.

  4) 같은 실행 순위를 갖는 연산자는 왼쪽에서부터 오른쪽으로 실행된다.

Python 입문

- 기타 연산자

Operator	Description
**	Exponentiation
~ + -	Complement, unary plus and minus
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise ‘AND’
^|	Bitwise exclusive ‘OR’ and regular ‘OR’
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
Not or and	Logical operators

Python 입문

- 기타 연산자 예시

1) 산술 연산자 (+, -, *, /, %, **, //)

```
#산술 연산자
a=12.34
b=2.1

print("a:{0}, b:{1}".format(a,b))
print("a+b=",a+b)
print("a-b=",a-b)
print("a*b=",a*b)
print("a/b= {0:.3f}".format(a/b))

print("{0}/{1}의 나머지:{2}".format(14,2,14%2))
print("{0}의 {1}승={2}".format(3,7,3**7))

print("===Floor Division===")
print("{0}//{1}=>{2}".format(10,3,10//3))
print("{0}//{1}=>{2}".format(10.8123,3.1,10.8123//3.1))
```

```
a:12.34, b:2.1
a+b= 14.44
a-b= 10.24
a*b= 25.914
a/b= 5.876
14/2의 나머지:0
3의 7승=2187
===Floor Division===
10//3=>3
10.8123//3.1=>3.0
Press any key to continue . . .
```

2) 비교 연산자 (==, !=, >, <, >=, <=)

```
#비교 연산자
a=10
b=7
c=10
print("{0}=={1}=>".format(a,b),a==b)
print("{0}=={1}=>".format(a,c),a==c)
print("{0}!={1}=>".format(a,b),a!=b)
print("{0}!={1}=>".format(a,c),a!=c)
print("{0}>{1}=>".format(a,b),a>b)
print("{0}>{1}=>".format(a,c),a>c)
print("{0}>={1}=>".format(a,b),a>=b)
print("{0}>={1}=>".format(a,c),a>=c)
print("{0}<{1}=>".format(a,b),a<b)
print("{0}<{1}=>".format(a,c),a<c)
print("{0}<={1}=>".format(a,b),a<=b)
print("{0}<={1}=>".format(a,c),a<=c)
```

```
10==7=> False
10==10=> True
10!=7=> True
10!=10=> False
10>7=> True
10>10=> False
10>=7=> True
10>=10=> True
10<7=> False
10<10=> False
10<=7=> False
10<=10=> True
Press any key to continue . . .
```

Python 입문

- 기타 연산자 예시

3) 대입 연산자 (=, +=, -=, *=, /=, %=, **=, //=)

```
#대입연산자

a=2
print("a:",a)
a+=3
print("a+=3 , a:",a)
a-=1
print("a-=1 , a:",a)
a*=3
print("a*=3 , a:",a)
a/=2
print("a/=2 , a:",a)
a%=4
print("a%=4 , a:",a)
a**=3
print("a**=3 , a:",a)
a//=2
print("a//=2 , a:",a)
```

```
a: 2
a+=3 , a: 5
a-=1 , a: 4
a*=3 , a: 12
a/=2 , a: 6.0
a%=4 , a: 2.0
a**=3 , a: 8.0
a//=2 , a: 4.0
Press any key to continue . . .
```

4) 비트 연산자 (&, |, ^, ~, <<, >>)

```
#비트 연산자
a=6
b=3
print("{0}&{1} => {2}".format(a,b,a&b))
print("{0}|{1} => {2}".format(a,b,a|b))
print("{0}^{1} => {2}".format(a,b,a^b))
print("~{0} => {1}".format(a,~a))
print("{0}<<{1} ==> {2}".format(a,3,a<<3))
print("{0}>>{1} ==> {2}".format(a,1,a>>1))
```

```
6&3 => 2
6|3 => 7
6^3 => 5
~6 => -7
6<<3 ==> 48
6>>1 ==> 3
Press any key to continue . . .
```

Python 입문

- 기타 연산자 예시

5) 논리 연산자 (and, or, not)

```
#논리 연산자
print("True and True:",True and True)
print("True and False:",True and False)
print("False and True:",False and True)
print("False and False:",False and False)

print("True or True:",True or True)
print("True or False:",True or False)
print("False or True:",False or True)
print("False or False:",False or False)

print("not True :", not True)
print("not False :", not False)
```

```
True and True: True
True and False: False
False and True: False
False and False: False
True or True: True
True or False: True
False or True: True
False or False: False
not True : False
not False : True
Press any key to continue . . .
```

6) 멤버쉽 연산자 (in, not in)

```
#멤버쉽 연산자

fruits =["사과","배","복숭아"]
print("사과"in fruits)
print("딸기"in fruits)
print("사과"not in fruits)
print("딸기"not in fruits)
```

```
True
False
False
True
Press any key to continue . . .
```

Python 입문

- 기타 연산자 예시
  7) 참조 비교 연산자 (is, is not)

```
#참조 비교 연산자
a=10 #값 형식
b=10
c=20
print("a:{0}, b:{1}, c:{2}".format(a,b,c))

print("a == b :",a == b)
print("a is b :",a is b)

print("a == c :",a == c)
print("a is c :",a is c)

d=3+2j #참조 형식
e=3+2j
f=4+2j
print("d == e :",d == e)
print("d is e :",d is e)

print("d == f :",d == f)
print("d is f :",d is f)
```

```
a:10, b:10, c:20
a == b : True
a is b : True
a == c : False
a is c : False
d == e : True
d is e : True
d == f : False
d is f : False
Press any key to continue . . .
```

Python 입문

- **리스트 자료형**
  1) 1, 3, 5, 7, 9라는 숫자 모음 만들기
     ex: odd = [1, 3, 5, 7, 9]

  2) 여러가지 리스트의 생김새
     ex: a = []
     b = [1, 2, 3]
     c = ['Life', 'is', 'too', 'short']
     d = [1, 2, 'Life', 'is']
     e = [1, 2, ['Life', 'is']]

- **리스트 연산자**
  1) 리스트 더하기

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a+b
print (c)
```

```
[1, 2, 3, 4, 5, 6]
계속하려면 아무 키나 누르십시오 . . .
```

  2) 여러가지 리스트의 생김새

```
a = [1, 2, 3]
b = a * 3

print (b)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
계속하려면 아무 키나 누르십시오 . . .
```

Python 입문

- **튜플 (Tuple)**
  1) 튜플의 생김새

     ex: t1 = ()

     t2 = (1,)

     t3 =(1, 2, 3)

     t4 = 1, 2, 3

     t5 = ('a', 'b', ('ab', 'cd'))
  2) 튜플의 인덱싱과 슬라이싱
     a) 인덱싱

```
t1 = (1, 2, 'a', 'b')
print(t1[0])
print(t1[3])
```

```
1
b
계속하려면 아무 키나 누르십시오 . . .
```

     b) 슬라이싱

```
t1 = (1, 2, 'a', 'b')
print(t1[1:])
```

```
(2, 'a', 'b')
계속하려면 아무 키나 누르십시오 . . .
```

     c) 튜플 더하기

```
t1 = (1, 2, 'a', 'b')
t2 = (3, 4)

print(t1+t2)
```

```
(1, 2, 'a', 'b', 3, 4)
계속하려면 아무 키나 누르십시오 . . .
```

     d) 튜플 곱하기

```
t2 = (3, 4)
print(t2*3)
```

```
(3, 4, 3, 4, 3, 4)
계속하려면 아무 키나 누르십시오 . . .
```

Python 입문

- **2차원 리스트**
  1) 2차원 리스트는 리스트 안에 리스트를 만들어 넣을 수 있다.
     Ex: 리스트 = [[값, 값], [값, 값], [값, 값]]

```
a = [[10, 20], [30, 40], [50, 60]]
print(a)
```

```
[[10, 20], [30, 40], [50, 60]]
계속하려면 아무 키나 누르십시오 . . .
```

  2) 2차원 리스트의 요소에 접근하거나 값을 할당할 때는 리스트 뒤에 [ ]를 두번 사용하며 [ ] 안에 세로(row)인덱스와 가로(column) 인덱스를 지정해 주면 된다.

```
a = [[10, 20], [30, 40], [50, 60]]

print(a[0][0]) # 세로 인덱스 0, 가로 인덱스 0인 요소 출력

print(a[1][1]) # 세로 인덱스 1, 가로 인덱스 1인 요소 출력

print(a[2][1]) # 세로 인덱스 2, 가로 인덱스 1인 요소 출력

a[0][1] = 1000 # 세로 인덱스 0, 가로 인덱스 1인 요소에 값 할당

print(a[0][1])
```

```
10
40
60
1000
계속하려면 아무 키나 누르십시오 . . .
```

Python 입문

- **Numpy:** 고성능의 다차원 배열 객체를 다룰 때 사용하는 라이브러리
  1) 파이썬의 리스트로 정의될 수 있으며, 직접 정의도 가능

```
import numpy as np

data1 = [1, 2, 3, 4, 5]
np1_arr = np.array(data1)
np2_arr = np.array([1, 2, 3, 4, 5])

print(np1_arr)
print(np2_arr)
```

```
[1 2 3 4 5]
[1 2 3 4 5]
계속하려면 아무 키나 누르십시오 . . .
```

  2) 다양한 배열 생성 함수 제공

```
import numpy as np

a = np.zeros((2, 2))                # 0을 모두 채운 행렬 생성
print(a)

b = np.ones((1, 2))                 # 1로 모두 채운 행렬 생성
print(b)

c = np.full((2, 2), 7)              # 특정 수로 모두 채운 행렬 생성
print(c)

d = np.eye(3)                       # 단위행렬 생성
print(d)

e = np.random.random((2, 2))        # 임의의 값으로 채운 행렬 생성
print(e)

f = np.arange(5, 9)                 # 시작값부터 끝값까지 증가하는 행렬 생성
print(f)
```

```
[[0. 0.]
 [0. 0.]]
[[1. 1.]]
[[7 7]
 [7 7]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[[0.92419475 0.33162447]
 [0.50288462 0.88037467]]
[5 6 7 8]
계속하려면 아무 키나 누르십시오 . . .
```

Python 입문

- **배열의 연산:** 배열끼리, 혹은 배열과 스칼라 연산은 산술 연산자 모두 지원

```
import numpy as np

arr1 = np.array([[1, 2, 3], [4, 5, 6]], dtype = np.float64)
arr2 = np.array([[7, 8, 9], [10, 11, 12]], dtype = np.float64)

print(arr1+arr2)

print(arr1-arr2)

print(arr1*arr2)

print(arr1/arr2)

print(arr1**arr2)
```

1) Numpy 배열 끼리의 곱과 나눗셈은 행렬의 곱이 아님

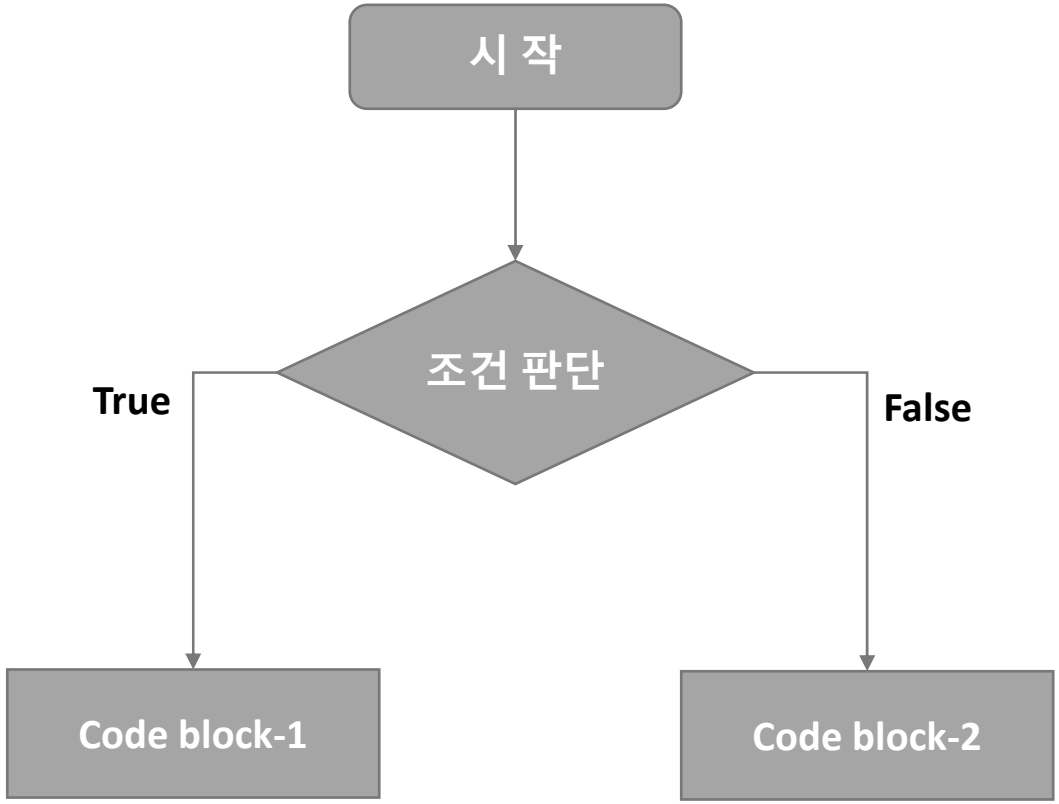
2) 배열의 차원과 크기도 서로 동일 해야만 연산이 가능함

```
[[ 8. 10. 12.]
 [14. 16. 18.]]
[[-6. -6. -6.]
 [-6. -6. -6.]]
[[ 7. 16. 27.]
 [40. 55. 72.]]
[[0.14285714 0.25       0.33333333]
 [0.4        0.45454545 0.5       ]]
[[1.00000000e+00 2.56000000e+02 1.96830000e+04]
 [1.04857600e+06 4.88281250e+07 2.17678234e+09]]
계속하려면 아무 키나 누르십시오 . . .
```

4. 제어문

if문

- **if 문:** 뒤에 존재하는 조건식이 참이냐 거짓이냐에 따라 아래와 같이 두개 이상의 코드 블록 중 하나를 실행



4. 제어문

if문

- **if 문의 기본 구조**

```
if <조건문>:
        <실행할 명령문1>
elif <조건문>:
        <실행할 명령문2>
else:
        <실행할 명령문3>
```

- 조건이 참일 경우 if 문을 실행하고 거짓일 경우 다음 문자 else문을 실행하기 때문에 else는 if 없이 단독으로 사용 불가능

4. 제어문

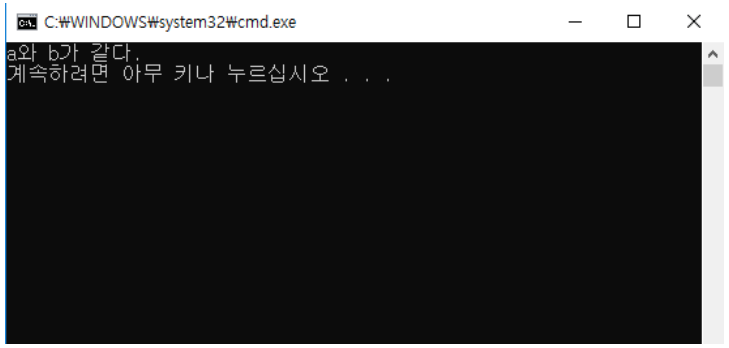
if문

[예제]

EX1) 주어진 a와 b의 크기를 비교하는 조건문

ex1.py

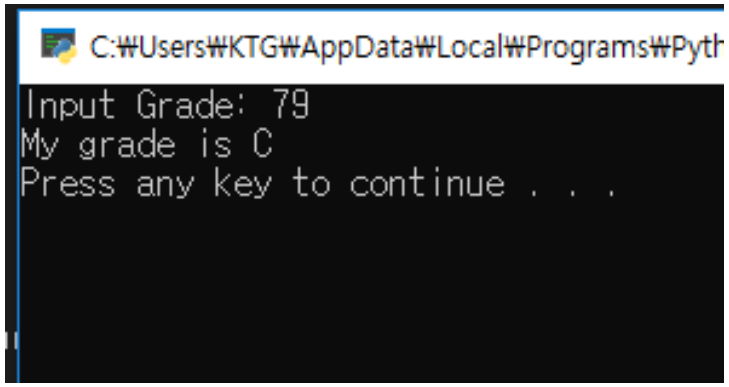
```
a = 100
b = 100

if a>b:
    print ("a가 b보다 크다.")
elif a<b:
    print ("a가 b보다 작다.")
else:
    print ("a와 b가 같다.")
```



EX2) 사용자로 부터 정수값을 입력 받아 점수 분류 90이상 A, 80이상 B, 70이상 C, 60 이상 D, 그 외 F

ex2.py* ex1.py

```
grade = int(input(('Input Grade: ')))

if ( 100 >= grade >= 90):
    grade = "A"
elif (90 > grade >= 80):
    grade = "B"
elif (80 > grade >= 70):
    grade = "C"
elif (70 > grade >= 60):
    grade = "D"
else:
    grade = "F"

print ("My grade is " + grade)
```

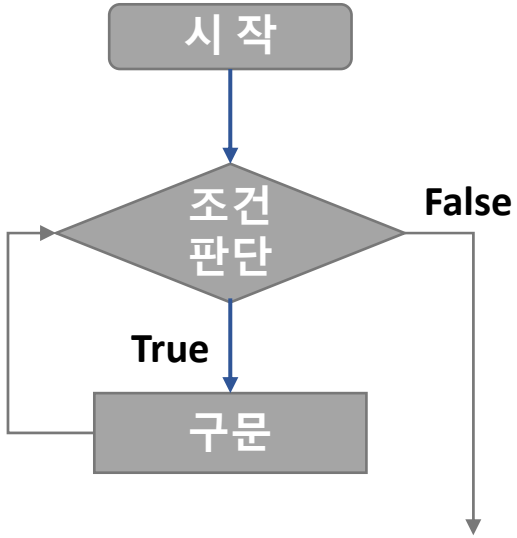


4. 제어문

while문

- **while문 실행 단계:**
  1) 조건을 평가해서 참(True) 혹은 거짓(False)를 산출
  2) 만약 조건이 거짓이면, while 문을 빠져나가 다음 문장을 계속 실행
  3) 만약 조건이 참이면, 몸통 부문의 문장을 실행하고 다시 처음 1번 단계로

  - 3)번째 단계에서 처음으로 다시 돌아가는 반복을 하기 때문에 이런 종류의 흐름을 루프(loop)라고 부른다. 매번 루프 몸통 부문을 실행할 때마다, 이것을 반복(iteration)이라고 한다.



4. 제어문

while문

- **while 문의 기본 구조**

  while <조건문>:

  <구문>

- **while 문:** 조건이 거짓이 될 때까지 수행 문장을 반복하는 구문

```
i = 0

while i < 5:          # 조건문
    print(i)          # 수행문장
    i += 1            # 수행문장
```

- **break 문:** while문 실행 중에 강제로 while문을 탈출할 수 있게 된다.

```
i = 1

while i:
    print (i)
    i += 1
    if i >= 5:
        break
```

- **continue 문:** while문 실행 중에 조건문으로 돌아갈 수 있다.

```
i = 0

while i < 11:         # 조건문
    i += 1            # i를 2로 나눈 값이 1이면 continue 실행

    if i % 2 != 0:
        continue      # 조건문으로 돌아간다.
    print(i)
```

4. 제어문

while문

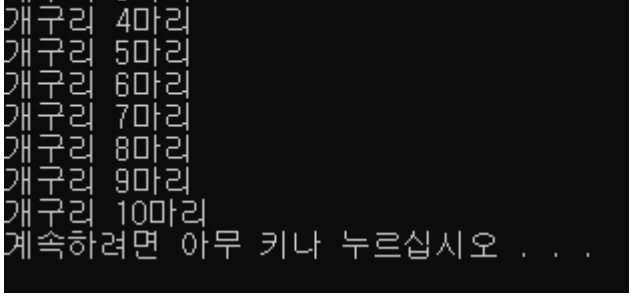
[예제]

EX3) While 반복문을 이용하여 다음과 같이 출력하시오.

1) while 문이 10번 반복되도록 조건을 설정
2) while 문의 결과로 다음과 같이 출력되도록 문자열 포매팅을 이용

출력 결과: 개구리 1마리
개구리 2마리
…
개구리 10마리

```
i = 1

while i <= 10:
    print("개구리 %d마리" % i)
    i += 1
```



4. 제어문

for문

- **for 문:** 순서형 자료를 이용하여 원하는 명령을 반복할 때 사용
- **for 문 기본구조:**

  for <아이템> in <시퀀스형 객체>:

  <구문>

```
family = ['mother', 'father', 'gentleman', 'sexy lady']

for x in family:
    print('%s %s' % (x, len(x)))
```

```
mother 6
father 6
gentleman 9
sexy lady 9
계속하려면 아무 키나 누르십시오 . . .
```

```
for x in [1, 2, 3]:                    # 리스트의 요소들을 하나씩 x에 대입
    print(x)

print('₩n')                            # 개행

for x in (1, 2, 3):                    # 튜플
    print(x)

print('₩n')

dic = {'key1':1, 'key2':2, 'key3':3}   # 딕셔너리
for x in dic:
    print(x)
    print(dic[x])

print('₩n')

for x in 'abcdefg':                    # 문자열
    print(x)
```

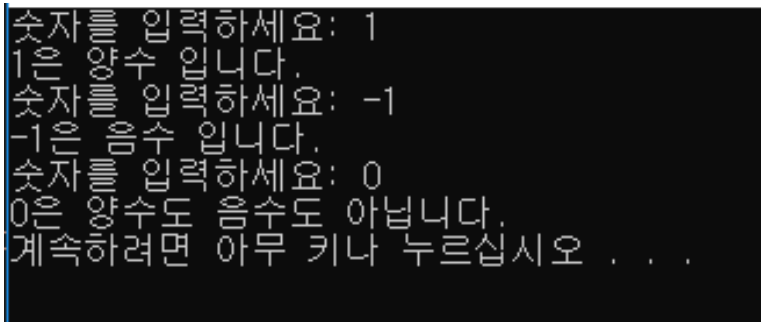
```
1
2
3


1
2
3


key1
1
key2
2
key3
3


a
b
c
d
e
f
g
계속하려면 아무 키나 누르십시오 . . .
```

4. 제어문

[연습문제]

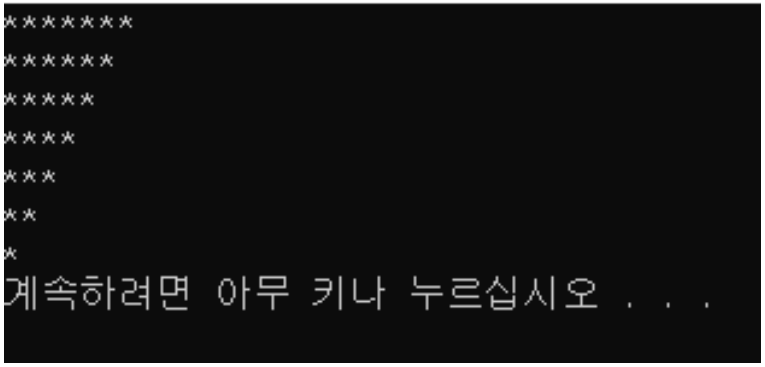
1. if 조건문을 사용하여 양수와 음수 및 0을 판별하는 프로그램을 만드세요.

   출력 결과:



2. while 문을 사용하여 (*)를 표시하는 프로그램을 만드세요.

   출력 결과:

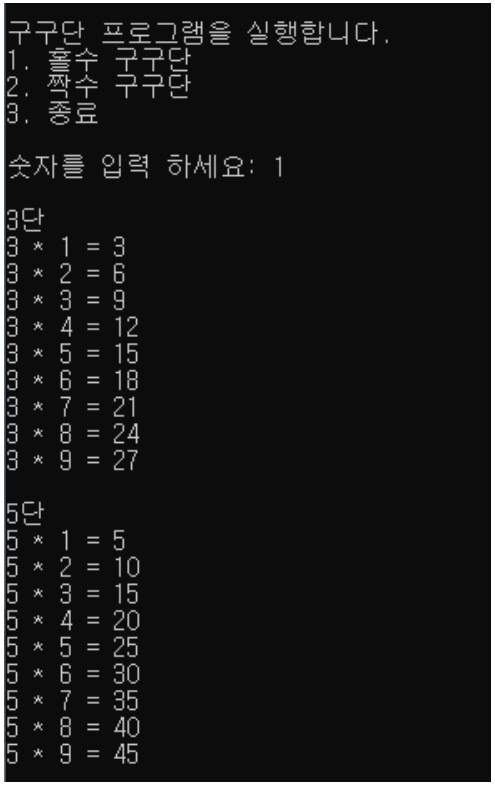


4. 제어문

[연습문제]

3. while 문과 for문 중첩을 상용하여 구구단 프로그램을 만드세요. 전체를 출력하는 구구단이 아니라 1을 입력하면 홀수 단(3, 5, 7, 9) 2번을 입력하면 짝수 단 (2, 4, 6, 8) 3번을 입력하면 종료 그 외의 숫자를 입력하면 처음부터 다시 입력되게 하는 프로그램을 만드세요.

출력 결과:



4. 제어문

[연습문제]

4. 중첩 루프를 이용하여 신문 배달하는 프로그램을 작성하세요. 단, 아래에서 arrears 리스트는 신문 구독료가 미납된 세대에 대한 정보를 포함하고 있는데, 해당 세대에는 신문을 배달하지 않아야 합니다.

```
apart = [[101, 102, 103, 104],[201, 202, 203, 204],[301, 302, 302, 303], [401, 402, 403, 404]]
arrears = [101, 203, 301, 404]
```

출력 결과:

```
Newspaper delivery:  102
Newspaper delivery:  103
Newspaper delivery:  104
Newspaper delivery:  201
Newspaper delivery:  202
Newspaper delivery:  204
Newspaper delivery:  302
Newspaper delivery:  302
Newspaper delivery:  303
Newspaper delivery:  401
Newspaper delivery:  402
Newspaper delivery:  403
Press any key to continue . . .
```

5. A학급에 총 10명의 학생이 있다. 이 학생들의 중간고사 점수는 다음과 같다.
   score=[70,60,55,75,95,90,80,85,100] for 문을 이용하여 A 학급의 평균 점수를 구해보자.
   출력 결과:

```
78.88888888888889
Press any key to continue . . .
```

5. 함수 정의

함수

- **함수**: 자주 사용되는 코드를 함수로 정의해 두면, 그 뒤로는 동일한 코드를 함수 이름만으로 실행 가능

- **함수를 정의하는 def문 기본구조:**

  def 함수이름(): # 헤더 행

  함수의 내용 # 내용 행(여러행)

```
def order():                              # 끝에 콜론(:)을 빠뜨리지 않도록 주의
    print('주문하실 음식을 알려주세요') # 이 블록은 들여쓰기
    food = input()
    print(food, '을 주문 하셨습니다.')

order()                                   # 들여 쓰기 하지 않는다.
```

```
주문하실 음식을 알려주세요
짜장면
짜장면 을 주문 하셨습니다.
계속하려면 아무 키나 누르십시오 . . .
```

- 반복되는 동작을 함수로 정의하면 중복을 제거하여 프로그램을 간결하게 작성 가능

```
a = 23
b = -23

def absolute_value(n):      # 반복되는 동작을 함수로 정의
    if n < 0:
        n = -n
    return n

if absolute_value(a) == absolute_value(b):
    print(a, "과", b, "의 절대값은 같다.")
else:
    print(a, "과", b, "의 절대 값은 다르다.")
```

```
23 과 -23 의 절대값은 같다.
계속하려면 아무 키나 누르십시오 . . .
```

5. 함수 정의

함수

- **매개변수**: 전달된 데이터를 함수 속에서 사용하려면, 그 데이터를 함수 속에서 부를 이름(변수)을 정해 둬야 한다. 함수에 전달된 데이터를 대입하기 위한 변수를 매개변수(Parameter)라고 부른다. 함수에 전달하는 데이터 자체를 인자(argument)라고 부른다. 즉, 함수를 호출하면 함수에 전달한 인자(데이터)가 함수 속의 매개변수에 대입된다.

```
def print_price(num_drink):                          # ❶ 매개변수(num_drink) 정의
    """음료의 잔 수(num_drink)를 전달받아,
    가격을 화면에 출력한다."""
    price_per_drink = 2500                           # 한 잔 당 가격
    total_price = num_drink * price_per_drink        # ❷ num_drink에 전달된 값 사용
    print('음료', num_drink, '잔:', total_price)      # ❸

print_price(3)                                       # ❹ 인자(3)를 전달하여 호출
```

```
음료 3 잔: 7500
Press any key to continue . . .
```

1) def문의 헤더 행에서 함수 이름(print_price)뒤의 괄호 속에 매개변수의 이름(num_drink)을 정의
2) 이렇게 정의한 매개변수는 함수 본문에서 부를 수 있으며, 함수를 호출할 때 전달하는 값으로 바뀐다.

5. 함수 정의

[연습문제]

1. 함수의 인자로 리스트(1 ~5)를 받은 후 리스트 내에 있는 모든 정수 값에 대한 최댓값과 최솟값을 반환하는 함수를 작성하세요.

   출력 결과:

```
5
1
Press any key to continue . . .
```

2. 함수의 인자로 시작과 끝을 나타내는 숫자를 받아 시작부터 끝까지의 모든 정수값의 합을 반환하는 함수를 작성하세요.(시작값과 끝값을 포함)

   출력 결과:

```
55
계속하려면 아무 키나 누르십시오 . . .
```

6. 클래스

클래스 & 인스턴스

- **클래스(Class):** 객체의 상태를 나타내는 필드(field)와 객체의 행동을 나타내는 메소드(method)로 구성
  1) 필드: 클래스에 포함된 변수(variable)를 의미
  2) 메소드: 어떠한 특정 작업을 수행하기 위한 명령문의 집합

- **인스턴스(Instance):** 클래스를 사용하기 위하여 클래스 타입의 객체(object)를 선언 필드

```
class robot:                          # robot 클래스 생성
    name = "robot"
    def info(self):
        print('저의 이름은', self.name, '입니다.')

r = robot()
type(r)
r.info()

print('₩n')
print(r.name)

print('₩n')
print(isinstance(r, int))

print('₩n')
print(isinstance(r, robot))
```

```
저의 이름은 robot 입니다.

robot

False

True
계속하려면 아무 키나 누르십시오 . . .
```

생성자 & 소멸자

- **생성자와 소멸자**
  1) 생성자: 객체가 만들어질 때 호출되는 함수(__init__), 객체 초기화에 사용
  2) 소멸자: 객체가 사라질때 호출되는 함수(__del__)

```
class robot:
    name = "robot"
    age = 0
    def __init__(self, name, age):
        print('생성자 호출')
        self.name = name
        self.age = age
    def __del__(self):
        print('소멸자 호출')
    def info(self):
        print('나의 이름은', self.name, '입니다.')
        print('나이는', self.age, '입니다.')

r = robot('eric', 30)
r.info()
del r
```

```
생성자 호출
나의 이름은 eric 입니다.
나이는 30 입니다.
소멸자 호출
계속하려면 아무 키나 누르십시오 . . .
```

클래스 상속

- 클래스 상속

```
class robot:
    name = "robot"
    age = 0
    def __init__(self, name, age):
        print('생성자 호출')
        self.name = name
        self.age = age
    def __del__(self):
        print('소멸자 호출')
    def info(self):
        print('나의 이름은', self.name, '입니다.')
        print('나이는', self.age, '입니다.')

class strong_robot(robot):
    weapon = 'gun'
    def __init__(self, name, age, weapon):
        print('strong_robot 생성자 호출')
        super().__init__(name, age)
        self.weapon = weapon
    def info(self):
        super().info()
        print(self.weapon, '로 싸웁니다.')


s = strong_robot('eric', 30, 'sword')

s.info()
```

```
strong_robot 생성자 호출
생성자 호출
나의 이름은 eric 입니다.
나이는 30 입니다.
sword 로 싸웁니다.
소멸자 호출
계속하려면 아무 키나 누르십시오 . . .
```

1) 상속 받고 싶은 부모 클래스를 괄호안에 입력
2) 상속을 받으면 부모 클래스의 모든 내용이 자식에게 전달
3) 자식 클래스에서 부모 클래스 메소드를 사용하기 위해 super() 사용
4) 부모 클래스가 제공하는 메서드를 자식 클래스가 재정의 가능(오버라이딩)

6. 클래스

[연습문제]

1. 다음의 조건을 만족하는 Point라는 클래스를 작성하세요.
   1) Point 클래스는 생성자를 통해 (x, y)좌표를 입력 받는다.
   2) setx(x), sety(y) 메서드를 통해 x좌표와 y좌표를 따로 입력 받을 수 있다.
   3) get() 메서드를 호출하면 튜플로 구성된 (x, y) 좌표를 반환한다.
   4) move(dx, dy)메서드는 현재 좌표를 dx, dy만큼 이동시킨다.
   5) 모든 메서드는 인스턴스 메서드이다.

2. 문제 1에서 생성한 Point 클래스에 대한 인스턴스를 생성한 후 4개의 메서드를 사용하여 코드를 작성하세요.

   출력 결과:

```
(3, 3)
(4, 2)
(0, 0)
Press any key to continue . . .
```

7. 모듈 가져오기(import)

import

- **모듈 가져오기:** 다른 프로그램으로부터 데이터를 가져올 수 있다.
- **모듈 사용 이유:** 코드의 재 사용성, 복잡하고 어려운 기능을 포함하는 프로그램을 간단하게 만들 수 있다.

- **Import 기본 문법**
  1) import 모듈 ➔ 모듈 전체를 가져오는 방법
  2) from 모듈 import 변수나 함수 ➔ 모듈 내에서 필요한 것만 가져오는 방법

```
import math
print(math.pow(2, 10))

print(math.pi)
```

```
1024.0
3.141592653589793
계속하려면 아무 키나 누르십시오 . . .
```

- Math 모듈은 수학과 관련된 기능이 들어 있는 내장 모듈임
- dir() 명령을 이용하여 모듈에 어떠한 함수들이 들어있는지 확인 가능

```
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'ata
nh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fm
od', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'ta
u', 'trunc']
계속하려면 아무 키나 누르십시오 . . .
```

7. 모듈 가져오기(import)

import

- **모듈 만들기**
  1) 계산기 모듈 만들기: 다음 코드를 calculator.py로 저장

```
# file 이름: calculator.py


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

def sub(a, b):
    return a - b

def mul(a, b):
    return a * b

def div(a, b):
    return a / b

def mod(a, b):
    return a % b
```

  2) 메인코드 작성: 다음 코드를 main.py로 작정하여 calculator.py와 같은 경로에 저장

```
import calculator

add_result = calculator.add(10, 2)
sub_result = calculator.sub(10, 2)
mul_result = calculator.mul(10, 2)
div_result = calculator.div(10, 2)
mod_result = calculator.mod(10, 2)


print (add_result)
print (sub_result)
print (mul_result)
print (div_result)
print (mod_result)
```

```
12
8
20
5.0
0
Press any key to continue . . .
```

7. 모듈 가져오기(import)

import

- **모듈의 경로가 다를 경우 가져오기**
  1) 계산기 모듈 만들기: 다음 코드를 module_test_2.py로 저장

```
# file 이름: module_test_2.py


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

def sub(a, b):
    return a - b

def mul(a, b):
    return a * b

def div(a, b):
    return a / b

def mod(a, b):
    return a % b
```

  2) 새로운 프로젝트 만들기: 새로운 경로에 module_test 프로젝트 만들기

```
import sys
sys.path.insert(0, 'C:₩python_example₩example')
import module_test_2

add_result = module_test_2.add(10, 2)
sub_result = module_test_2.sub(10, 2)
mul_result = module_test_2.mul(10, 2)
div_result = module_test_2.div(10, 2)
mod_result = module_test_2.mod(10, 2)


print (add_result)
print (sub_result)
print (mul_result)
print (div_result)
print (mod_result)
```

import sys
sys.path.insert(0, 'py파일이 있는 경로')
import import_file

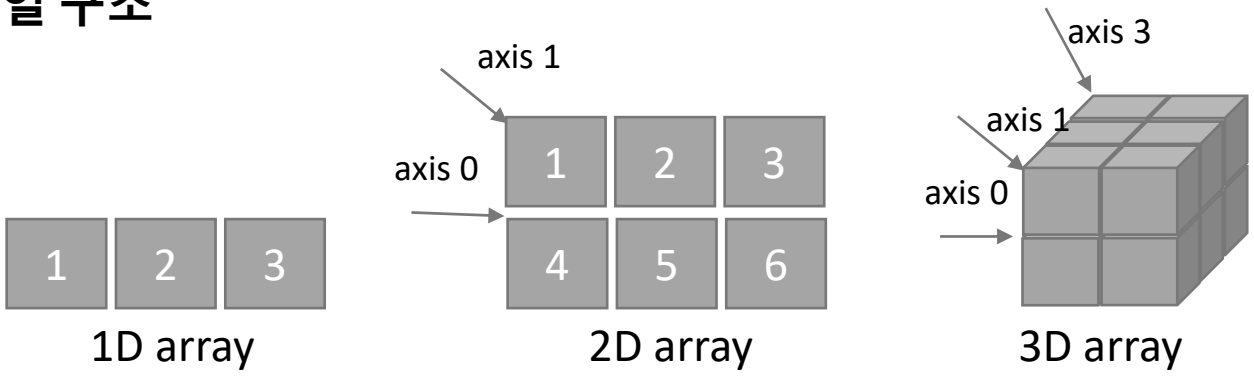
```
12
8
20
5.0
0
Press any key to continue . . .
```

Numpy

- **Numpy:**
  1) 대규모 다차원 배열과 행렬 연산에 필요한 다양한 함수를 제공
  2) 메모리 버퍼에 배열 데이터를 저장하고 처리하는 효율적인 인터페이스 제공

- **Numpy 특징:**
  1) 강력한 N차원 배열 객체
  2) 정교한 브로드케스팅(broadcast) 가능
  3) C/C++ 및 포트란 코드 통합 도구
  4) 유용한 선형 대수학, 푸리에 변환 및 난수 기능
  5) 범용적 데이터 처리에 사용 가능한 다차원 컨테이너

- **Numpy 배열 구조**



8. Numerical Python

Numpy

- **Numpy 배열 구조:** Numpy 배열 구조는 “Shape”으로 표현됩니다. Shape은 배열의 구조를 파이썬 튜플 자료형을 이용하여 정의합니다. 예를 들어 28*28 컬러 사진은 높이가 28, 폭이 28, 각 픽셀은 3개 채널(RGB)로 구성된 데이터 구조를 갖습니다. 즉, 컬러 사진 데이터는 Shape 이 (28, 28, 3)인 3차월 배열입니다. 다차원 배열은 입체적인 데이터 구조를 가지며, 데이터의 차원은 여러 갈래의 데이터 방향을 갖습니다. 다차원 배열의 데이터 방향을 axis로 표현할 수 있습니다. 행방향(높이), 열방향(폭), 채널 방향은 각각 axis=0, axis=1, 그리고 axis=2로 지정됩니다.

- **Numpy 배열 실습 함수 작성:** Numpy 객체 정보를 출력하기 위한 pprint 함수 선언

```
def pprint(arr):
    print("type:{}".format(type(arr)))
    print("shape: {}, dimension: {}, dtype:{}".format(arr.shape, arr.ndim, arr.dtype))
    print("Array's Data:₩n", arr)
```

Numpy

- Python 배열로 Numpy 배열 생성

```
import numpy as np

def pprint(arr):
    print("type:{}".format(type(arr)))
    print("shape: {}, dimension: {}, dtype:{}".format(arr.shape, arr.ndim, arr.dtype))
    print("Array's Data:₩n", arr)



# 1차원 배열(List)로 Numpy 배열 생성

arr = [1, 2, 3]
a = np.array([1, 2, 3])
pprint(a)

# 2차원 배열로 Numpy 배열 생성, 원소 데이터 타입 지정

arr = [(1,2,3), (4,5,6)]
a= np.array(arr, dtype = float)
pprint(a)

# 3차원 배열로 Numpy 배열 생성, 원소 데이터 타입 지정

arr = np.array([[[1,2,3], [4,5,6]], [[3,2,1], [4,5,6]]], dtype = float)
a= np.array(arr, dtype = float)
pprint(a)
```

```
type:<class 'numpy.ndarray'>
shape: (3,), dimension: 1, dtype:int32
Array's Data:
 [1 2 3]
type:<class 'numpy.ndarray'>
shape: (2, 3), dimension: 2, dtype:float64
Array's Data:
 [[1. 2. 3.]
 [4. 5. 6.]]
type:<class 'numpy.ndarray'>
shape: (2, 2, 3), dimension: 3, dtype:float64
Array's Data:
 [[[1. 2. 3.]
  [4. 5. 6.]]

 [[3. 2. 1.]
  [4. 5. 6.]]]
Press any key to continue . . .
```

8. Numerical Python

Numpy

- **배열 생성 및 초기화:** Numpy는 원하는 shape으로 배열을 설정하고, 각 요소를 특정 값으로 초기화하는 zeros, ones, full, eye 함수를 제공, 파라미터로 입력한 배열과 같은 shape의 배열을 만드는 zeros_like, ones_like, full_like 함수도 제공

```
import numpy as np
import pprint

# np.zeros 함수
a = np.zeros((3, 4))
pprint.pprint(a)
print('₩n')

# np.ones 함수
a = np.ones((2,3,4),dtype=np.int16)
pprint.pprint(a)
print('₩n')

# np.full 함수
a = np.full((2,2),7)
pprint.pprint(a)
print('₩n')

# np.eye 함수
np.eye(4)
print('₩n')

# np.empty 함수
a = np.empty((4,2))
pprint.pprint(a)
print('₩n')

# like 함수
a = np.array([[1,2,3], [4,5,6]])
b = np.ones_like(a)
pprint.pprint(b)
```

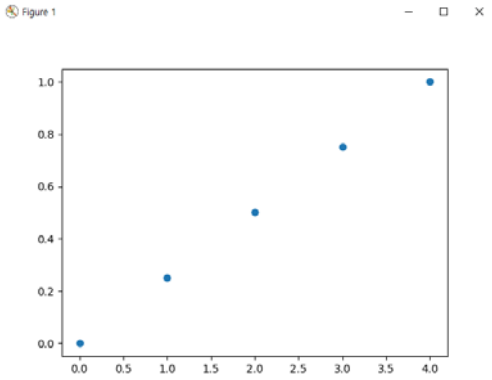
```
type:<class 'numpy.ndarray'>
shape: (3, 4), dimension: 2, dtype:float64
Array's Data:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]


type:<class 'numpy.ndarray'>
shape: (2, 3, 4), dimension: 3, dtype:int16
Array's Data:
 [[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]

 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]


type:<class 'numpy.ndarray'>
shape: (2, 2), dimension: 2, dtype:int32
Array's Data:
 [[7 7]
 [7 7]]



type:<class 'numpy.ndarray'>
shape: (4, 2), dimension: 2, dtype:float64
Array's Data:
 [[6.23042090e-307 1.42417221e-306]
 [1.37961641e-306 2.13620807e-306]
 [1.24611741e-306 2.13620807e-306]
 [1.24611741e-306 4.79026085e-291]]


type:<class 'numpy.ndarray'>
shape: (2, 3), dimension: 2, dtype:int32
Array's Data:
 [[1 1 1]
 [1 1 1]]
Press any key to continue . . .
```

Numpy

- **데이터 생성 함수:** Numpy는 주어진 조건으로 데이터를 생성한 후, 배열을 만드는 데이터 생성 함수를 제공
  1) np.linspace 함수
     a) numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)
     b) start 부터 stop의 범위에서 num개를 균일한 간격으로 데이터를 생성하고 배열을 만드는 함수
     c) 요소 개수를 기준으로 균등 간격의 배열을 생성

```
import numpy as np
import pprint

a = np.linspace(0, 1, 5)
pprint.pprint(a)

# linspace의 데이터 추출 시각화

import matplotlib.pyplot as plt
plt.plot(a, 'o')
plt.show()
```

```
type:<class 'numpy.ndarray'>
shape: (5,), dimension: 1, dtype:float64
Array's Data:
 [0.   0.25 0.5  0.75 1.  ]
```

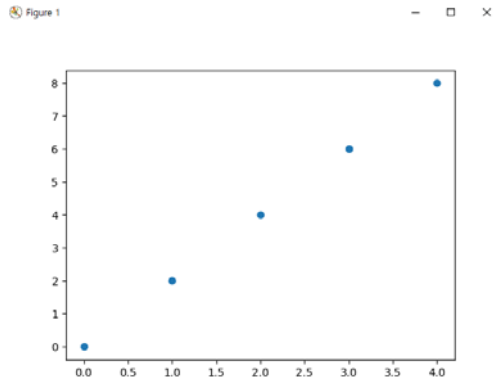


Numpy

2) np.arrange 함수
   a) numpy.arrange([start], stop[,step,], dtype=None)
   b) start 부터 stop 미만까지 step 간격으로 데이터 생성한 후 배열을 만듬
   c) 범위내에서 간격을 기준 균등 간격의 배열
   d) 요소의 개수가 아닌 데이터의 간격을 기준으로 배열 생성

```
import numpy as np
import pprint

a = np.arange(0, 10, 2, np.float)
pprint.pprint(a)

# arange의 데이터 추출 시각화
import matplotlib.pyplot as plt
plt.plot(a, 'o')
plt.show()
```

```
type:<class 'numpy.ndarray'>
shape: (5,), dimension: 1, dtype:float64
Array's Data:
 [0. 2. 4. 6. 8.]
```

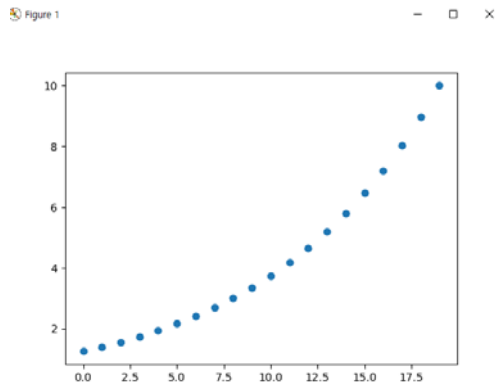


Numpy

3) np.logspace 함수
   a) numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)
   b) 로그 스케일의 linspace 함수
   c) 로그 스케일로 지정된 범위에서 num 개수 만큼 균등 간격으로 데이터 생성한 후 배열 만듬

```
import numpy as np
import pprint


a = np.logspace(0.1, 1, 20, endpoint=True)
pprint.pprint(a)

# logspace의 데이터 추출 시각화
import matplotlib.pyplot as plt
plt.plot(a, 'o')
plt.show()
```

```
type:<class 'numpy.ndarray'>
shape: (20,), dimension: 1, dtype:float64
Array's Data:
 [ 1.25892541  1.40400425  1.565802    1.74624535  1.94748304  2.1719114
  2.42220294  2.70133812  3.0126409   3.35981829  3.74700446  4.17881006
  4.66037703  5.19743987  5.79639395  6.46437163  7.2093272   8.04013161
  8.9666781  10.        ]
```

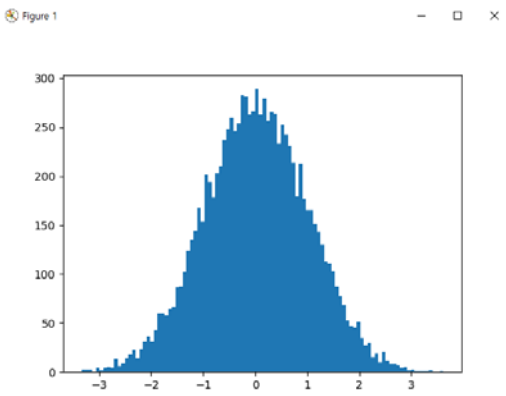


Numpy

- **난수 기반 배열 생성**
  1) np.random.normal 함수
     a) normal(loc=0.0, scale=1.0, size=None)
     b) 정규 분포 확률 밀도에서 표본 추출
     c) loc: 정규 분포의 평균, scale: 표준편차

```
import numpy as np
import pprint

mean = 0
std = 1
a = np.random.normal(mean, std, (2, 3))
pprint.pprint(a)

# 정규 분포로 10000개 표본을 뽑은 결과를 히스토그램으로 표현한 예

data = np.random.normal(0, 1, 10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=100)
plt.show()
```

```
type:<class 'numpy.ndarray'>
shape: (2, 3), dimension: 2, dtype:float64
Array's Data:
 [[-0.3200948   1.24156975  0.43915511]
 [-0.49430362  1.66772401  0.11864014]]
```

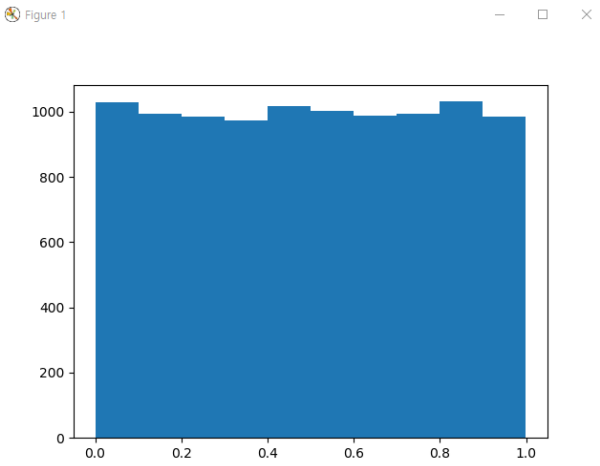


Numpy

2) np.random.rand 함수
   a) numpy.random.rand(d0, d1, ..., dn)
   b) shape이 (d0, d1, ..., dn)인 배열 생성 후 난수로 초기화
   c) 난수: [0, 1]의 균등 분포 형상으로 표본 추출
   d) Gaussian normal

```
import numpy as np
import pprint

a = np.random.rand(3,2)
pprint.pprint(a)

# 균등 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현

data = np.random.rand(10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```

```
type:<class 'numpy.ndarray'>
shape: (3, 2), dimension: 2, dtype:float64
Array's Data:
 [[0.22528626 0.24890774]
 [0.44318187 0.58624623]
 [0.83688736 0.43029005]]
```

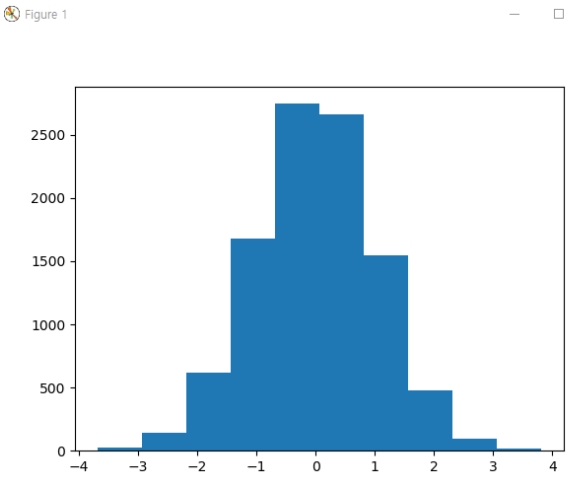


Numpy

3) np.random.randn 함수
   a) numpy.random.randn(d0, d1, …, dn)
   b) (d0, d1, …, dn) shape 배열 생성 후 난수로 초기화
   c) 난수: 표준 정규 분포에서 표본 추출

```
import numpy as np
import pprint

a = np.random.randn(2,4)
pprint.pprint(a)

# 균등 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현

data = np.random.randn(10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```

```
type:<class 'numpy.ndarray'>
shape: (2, 4), dimension: 2, dtype:float64
Array's Data:
 [[ 0.47554548  1.21232352  1.07883202 -0.84407333]
 [-1.15242775  0.45869663  0.42069643  0.66634519]]
```

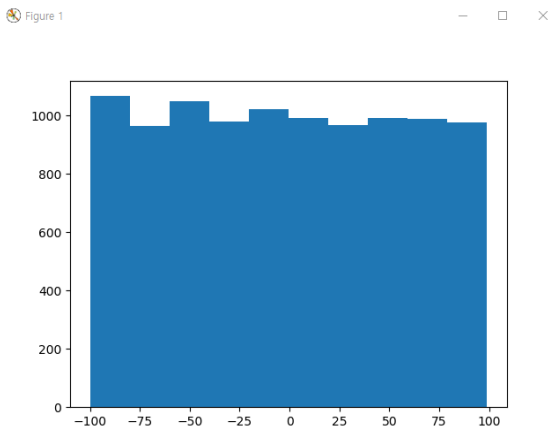


Numpy

4) np.random.randint 함수
   a) numpy.random.randint(low, high=None, size=None, dtype='l')
   b) 지정된 shape으로 배열을 만들고 low 부터 high 미만의 범위에서 정수 표본 추출

```
import numpy as np
import pprint

a = np.random.randint(5, 10, size=(2, 4))
pprint.pprint(a)

a = np.random.randint(1, size=10)
pprint.pprint(a)

# 균등 분포로 10000개를 표본 추출한 결과를 히스토그램으로 표현

data = np.random.randint(-100, 100, 10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=10)
plt.show()
```

```
type:<class 'numpy.ndarray'>
shape: (2, 4), dimension: 2, dtype:int32
Array's Data:
 [[7 8 8 8]
 [5 9 9 8]]
type:<class 'numpy.ndarray'>
shape: (10,), dimension: 1, dtype:int32
Array's Data:
 [0 0 0 0 0 0 0 0 0 0]
```



8. Numerical Python

Numpy

- **배열 연산 :** Numpy는 기본 연산자를 연산자 재정의하여 배열(행렬) 연산에 대한 직관적으로 표현을 강화하였습니다. 모든 산술 연산 함수는 np모듈에 포함되어 있습니다.

```
import numpy as np
import pprint
# arange로 1부터 10 미만의 범위에서 1씩 증가하는 배열 생성
# 배열의 shape을 (3, 3)으로 지정
a = np.arange(1, 10).reshape(3, 3)
pprint.pprint(a)
# arange로 9부터 0까지 범위에서 1씩 감소하는 배열 생성
# 배열의 shape을 (3, 3)으로 지정
b = np.arange(9, 0, -1).reshape(3, 3)
pprint.pprint(b)
print('₩n')
print(a - b)                # 뺄셈
print('₩n')
print(np.subtract(a, b))    # 뺄셈
print('₩n')
print(a+b)                  # 덧셈
print('₩n')
print(np.add(a, b))         # 덧셈
print('₩n')
print(a/b)                  # 나눗셈
print('₩n')
print(np.divide(a, b))      # 나눗셈
print('₩n')
print(a+b)                  # 곱셈
print('₩n')
print(np.multiply(a, b))    # 곱셈
print('₩n')
print(np.exp(b))            # 지수
print('₩n')
print(np.sqrt(a))           # 제곱근
print('₩n')
print(np.sin(a))            # sin
print('₩n')
print(np.cos(a))            # cos
print('₩n')
print(np.tan(a))            # tan
print('₩n')
print(np.log(a))            # log
print('₩n')
print(np.dot(a, b))         # dot product, 내적
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
array([[9, 8, 7],
       [6, 5, 4],
       [3, 2, 1]])


[[-8 -6 -4]
 [-2  0  2]
 [ 4  6  8]]


[[-8 -6 -4]
 [-2  0  2]
 [ 4  6  8]]


[[10 10 10]
 [10 10 10]
 [10 10 10]]


[[10 10 10]
 [10 10 10]
 [10 10 10]]


[[0.11111111 0.25       0.42857143]
 [0.66666667 1.         1.5       ]
 [2.33333333 4.         9.        ]]


[[0.11111111 0.25       0.42857143]
 [0.66666667 1.         1.5       ]
 [2.33333333 4.         9.        ]]


[[ 9 16 21]
 [24 25 24]
 [21 16  9]]


[[ 9 16 21]
 [24 25 24]
 [21 16  9]]


[[8.10308393e+03 2.98095799e+03 1.09683316e+03]
 [4.03428793e+02 1.48413159e+02 5.45981500e+01]
 [2.00855369e+01 7.38906610e+00 2.71828183e+00]]
```

```
[[1.         1.41421356 1.73205081]
 [2.         2.23606798 2.44948974]
 [2.64575131 2.82842712 3.        ]]


[[ 0.84147098  0.90929743  0.14112001]
 [-0.7568025  -0.95892427 -0.2794155 ]
 [ 0.6569866   0.98935825  0.41211849]]


[[ 0.54030231 -0.41614684 -0.9899925 ]
 [-0.65364362  0.28366219  0.96017029]
 [ 0.75390225 -0.14550003 -0.91113026]]


[[ 1.55740772 -2.18503986 -0.14254654]
 [ 1.15782128 -3.38051501 -0.29100619]
 [ 0.87144798 -6.79971146 -0.45231566]]


[[0.         0.69314718 1.09861229]
 [1.38629436 1.60943791 1.79175947]
 [1.94591015 2.07944154 2.19722458]]


[[ 30  24  18]
 [ 84  69  54]
 [138 114  90]]
Press any key to continue . . .
```

감사합니다.

Thank you.