1. 파이썬 자료형

(1) 숫자형 자료

· 정수형 자료

```
num=2                                   #num 변수에 숫자 2를 저장
print(num)                              #num 변수의 값을 출력, 실행 결과=2
print(type(num))                        #num 변수의 형식(int)을 출력, 실행 결과=<class 'int'>
a=1
b=1
sum=a*b
print(sum)                              #1*2를 더한 값을 출력, 실행 결과=2
```

· 실수형 자료

```
print(17/3)          #17/3의 결과값에 대한 형식(float)을 출력, 실행 결과=5.666666667
print(type(17/3))                       #형식(float)을 출력, 실행 결과=<class 'float'>
print(17//3)                            #나눗셈 결과의 몫을 출력, 실행 결과=5
print(17%3)                             #나눗셈 결과의 나머지 출력, 실행 결과=2
```

· 복소수형 자료

```
a=1+3j                                  #실수부: 1, 허수부: 3
print(a.real, a.imag)                   #실수, 허수를 출력, 실행 결과=1.0 3.0
print(type(a))          #a 변수의 형식(complex)을 출력, 실행 결과=<class 'complex'>
```

(2) 문자형 자료

• 문자형 자료의 표현 방식

```
print(‘“Isn\`t,” they said.‘)                    #큰따옴표가 문자열의 일부로 구성
실행 결과=“Isn`t,” they said.
print(“’Isn\`t,‘ they said.”)                    #작은따옴표가 문자열의 일부로 구성
실행 결과=’Isn`t,‘ they said.
```

• 문자형 자료의 줄바꿈

```
print(‘C:\some\name’)                            #\n이 줄바꿈으로 해석
실행 결과=C:\some
          ame
print(r‘C:\some\name’)                           #문자열 앞에 r을 붙여 \n이 문자로 취급됨
실행 결과=C:\some\name
```

• 문자형 자료 연결 및 반복

```
print(“Py”+“thon”)          # +연산자로 문자열을 연결, 실행 결과=Python
print(3*“Py”+“thon”)        # *연산자로 문자열 반복, 실행 결과=PyPyPython
```

• 문자형 자료 서브스크립트

```
str=’Python’ print(str[0])          #인덱스는 0부터 시작함, 실행 결과=P
print(str[2])                                    #실행 결과=t
print(str[-1])          # -로 인덱스하면 문자열의 끝부터 리턴, 실행 결과=n
print(str[-2])                                   #실행 결과=o
```

• 문자형 자료 슬라이싱

```
str=’Python’
print(str[0:2])          #인덱스 0번부터 2번 이전까지 출력, 실행 결과=Py
print(str[2:5])          #인덱스 2번부터 5번 이전까지 출력, 실행 결과=tho
```

• 문자형 자료의 길이

```
str=’Python’
print(len(str)) #문자열의 길이를 리턴, 실행 결과=6
```

(3) 리스트

· 리스트 자료의 표현 방식

```
L=["Python", 1, 2, 3, 4, "Last"]                    #리스트에 숫자, 문자열 동시 존재
print(L[0])                    #문자열 인덱싱과 동일하게 사용, 실행 결과=Python
print(L[2:6])                  #슬라이싱 가능, 실행 결과=[2, 3, 4, 'Last']
print(L[5][1])          #리스트의 5번째 자료의 두 번째 문자열 리턴, 실행 결과=a
```

· 리스트에 요소 추가하기

① append( ) 메소드

```
L=[1, 2, 3]
L.append(4)                                    #리스트 자료형 L에 요소 4 추가
L.append(5)                                    #리스트 자료형 L에 요소 5 추가
print(len(L))                                  #리스트의 크기를 출력, 실행 결과=5
print(L)                    #리스트 자료형 L 전체 출력, 실행 결과=[1, 2, 3, 4, 5]
```

② extend( ) 메소드

```
L=[1, 2, 3]
L.extend([4, 5]) #요소 [4, 5]를 한 번에 추가
print(len(L)) #실행 결과=5
print(L) #실행 결과=[1, 2, 3, 4, 5]
```

③ insert(인덱스, 요소) 메소드

```
L=["a", "b", "c"]
L.insert(0, -1)                                #인덱스 [0], "a" 앞에 추가
L.insert(2, 0)                                 #인덱스 [2], "b" 앞에 추가
print(len(L))                                  #리스트의 크기 출력, 실행 결과=5
print(L)                  #리스트 자료형 L 전체 출력, 실행 결과=[-1, 'a', 0, 'b', 'c']
```

· 리스트에서 요소 삭제하기

① pop(인덱스) 메소드

```
L=["a", "b", "c", "d", "e"]
print(L.pop(1))                #인덱스 [1] 요소 삭제하고 값 반환, 실행 결과=b
print(len(L))                                                    #실행 결과=4
print(L)                                    #실행 결과['a', 'c', 'd', 'e']
```

② remove(값) 메소드

```
L=["a", "b", "c", "d", "e"]
L.remove("c")                                                  #"c" 값의 요소 삭제
print(len(L))                                                    #실행 결과=4
print(L)                                  #실행 결과=['a', 'b', 'd', 'e']
```

③ clear( ) 메소드

```
L=["a", "b", "c", "d", "e"]
L.clear()                          #리스트의 모든 요소 삭제
print(len(L))                      #실행 결과=0
```

· 리스트에 요소 정렬하기

① sort( ) 메소드

```
L=["3", "1", "2", "5", "4"]
L.sort()                           #리스트 자료형을 오름차순으로 정렬
print(L)                           #실행 결과=['1', '2', '3', '4', '5']
```

```
L=["3", "1", "2", "5", "4"]
L.sort(reserve=True)               #리스트 자료형을 내림차순으로 정렬
print(L)                           #실행 결과=['5', '4', '3', '2', '1']
```

② reserve( ) 메소드

```
L=["3", "1", "2", "5", "4"]
L.reverse()                        #리스트의 모든 요소의 순서를 반대로
print(L)                           #실행 결과['4', '5', '2', '1', '3']
```

(4) 튜플

· **튜플 자료의 표현 방식**

```
T=("Python", 1, 2, 3, 4, "Last")   #괄호 안 모든 요소들을 쉼표를 이용해 구분해 나열
print(T[0])                        #문자열 인덱싱과 동일하게 사용 가능, 실행 결과=Python
print(T[2:6])                      #슬라이싱 가능, 실행 결과=(2, 3, 4, 'Last')
print(T[5][1])                     #6번째 자료의 두 번째 문자열 출력, 실행 결과=a
```

(5) 딕셔너리

· 딕셔너리 자료의 표현 방식

```
SCORE={"Math":80, "Eng":70, "Pi":100}   #Key:Value를 쉼표로 구분하여 나열
print(SCORE["Pi"])                      #실행 결과=100
```

· 딕셔너리 항목 추가

```
SCORE={"Math":80, "Eng":70, "Pi":100}   #Key:Value를 쉼표로 구분하여 나열
SCORE["Kor"]=90                         #key="Kor", value=90 추가
print(SCORE["Kor"]                      #실행 결과=90
```

· 딕셔너리 항목 삭제

```
SCORE={"Math":80, "Eng":70, "Pi":100}   #Key:Value를 쉼표로 구분하여 나열
del SCORE["Math"]                       #"Math" 요소 삭제
print(SCORE)                            #실행 결과={'Eng':70, 'Pi':100}
```

· 딕셔너리 값 변경

```
SCORE={"Math":80, "Eng":70, "Pi":100}          #Key:Value를 쉼표로 구분하여 나열
SCORE["Math"]=100                              #"Math" 요소의 값 변경
print(SCORE)                                   #실행 결과={'Math':100, 'Eng':70, 'Pi':100}
```

2. 파이썬 연산자

(1) 사칙연산

```
add=4+2
sub=4-2
mul=4*2
div=4/2
print("add=", add)                             #실행 결과= add=6
print("sub=", sub)                             #실행 결과= sub=2
print("mul=", mul)                             #실행 결과= mul=8
print("div=", div)                             #실행 결과= div=2.0
```

(2) 정수만 구하는 나눗셈 연산자

```
div=5//2
print(div, "=5//2")                            #실행 결과= 2=5//2
div=5/2
print(div, "=5/2")                             #실행 결과= 2.5=5/2
```

(3) 나머지 연산자

```
div=5/2
print(div, "=5/2")                             #실행 결과= 2.5=5/2
div=5%2
print(div, "=5%2")                             #실행 결과= 1=5%2
```

(4) 거듭제곱 연산자

```
print(2**10, "=2**10")                         #실행 결과= 1024=2**10
```

(5) 논리 연산자

연산식(A)	연산자	연산식(B)	결과
True	and	True	True
True	and	False	False
True	or	False	True
False	or	False	False
	not	False	True
	not	True	False

(6) 비교 연산자

연산식(A)	연산자	연산식(B)	결과
4	==	4	True
4	!=	4	False
4	>	4	False
4	>=	4	True
4	<=	4	True
4	<	4	False

· 숫자 비교하기

```
print(4==4)                    #실행 결과=True
print(4!=4)                    #실행 결과=False
print(4>4)                     #실행 결과=False
print(4>=4)                    #실행 결과=True
print(4<=4)                    #실행 결과=True
print(4<4)                     #실행 결과=False
```

· 문자열 비교하기

```
print('Python'=='Python')      #실행 결과=True
print('Python'!='Python')      #실행 결과=False
print('Python'=='PYTHON')      #실행 결과=False
```

3. 파이썬 제어문

(1) if문

• if <조건식>

```
x=3
if x==3:                                    #x가 3이면 if문 이하 실행
    print("x==3")                           #실행 결과= x==3
```

• if <조건식>, else

```
x=3
if x!=3:                                    #x가 3이 아니면 if문 이하 실행
    print("x!=3")
else:                                       #x가 3이면 else문 이하 실행
    print("x==3") #실행 결과= x==3
```

• if <조건식>, elif <조건식>, else

```
x=1
if x==3:                                    #x가 3이면 if문 이하 실행
    print("x==3")
elif x==4:                                  #x가 4이면 elif문 이하 실행
    print("x==4")
else:                                       #x가 3이 아니고, 4도 아니면 else문 이하 실행
    print("x!=3 and x!=4")                  #실행 결과= x!=3 and x!=4
```

(2) for 반복문

・for 변수 in range(반복 횟수)

```
sum=0
for i in range(11):                          #0~10까지 11번 반복
    sum=sum+i

print("i=", i)                               #실행 결과= i=0, i=1, …, i=10
print("sum=", sum)                           #실행 결과= sum=55
```

・for 변수 in range(시작, 끝, 증가)

```
sum=0
for i in range(1, 11, 2):                    #1~10까지 2씩 증가하며 5회 반복
    sum=sum+i
    print("i=", i)                           #실행 결과= i=1, i=3, …, i=9

print("sum=", sum)                           #실행 결과= sum=25
```

```
sum=0
for i in range(10, 1, -2):                   #10~2까지 2씩 감소하며 5회 반복
    sum=sum+i
    print("i=", i)                           #실행 결과= i=10, i=8, …, i=2

print("sum=", sum)                           #실행 결과= sum=30
```

・for 변수 in 시퀀스 객체

```
score=[80, 90, 100]
sum=0
for i in score:
    sum=sum+i
    print("i=", i)                           #실행 결과= i=80, i=90, i=100

print("sum=", sum)                           #실행 결과= sum=270
```

(3) while <조건식> 반복문

```
sum=0
i=1
while i<11:                          #i 변수를 1부터 시작해서 10까지 10번 반복
    sum=sum+i
    print("i=", i)                   #실행 결과= i=1, i=2, …, i=10
    i=i+1

print("sum=", sum)                   #실행 결과= sum=55
```

・break로 반복문 끝내기

```
sum=0
i=1
while i<11:                          #i 변수를 1부터 시작해서 10까지 10번 반복
    sum=sum+i
    print("i=", i)                   #실행 결과= i=1, i=2, …, i=5
    if(i==5): break                  #반복 중에 i가 5가 되면 while 루프를 빠져나감
    i=i+1

print("sum=", sum)                   #실행 결과= sum=15
```

・continue로 반복문 처음으로 이동하기

```
sum=0
i=0
while i<11:                          #i 변수를 1부터 시작해서 10까지 10번 반복
    i=i+1
    if i%2 == 0:                     #나머지 연산자 %, 2로 나누어서 나머지가 0이면 짝수
        print("i=", i)               #실행 결과= i=2, i=4, …, i=10
        sum=sum+i
    else:
        continue

print("sum=", sum)                   #실행 결과= sum=30
```

4. 사용자 입력 및 출력

(1) 사용자 입력

```
math=input("math=")    #input 함수를 이용해 사용자에게 데이터 입력을 받음, 입력=20
eng=input("eng=")      #input 함수를 이용해 사용자에게 데이터 입력을 받음, 입력=30
print(math+eng)                                          #실행 결과= 2030
```

```
math=int(input("math="))                                 #입력=20
eng=int(input("eng="))                                   #입력=30
print(math+eng)                                          #실행 결과= 50
```

(2) print( ) 함수 출력

・문자열 출력

```
print('Hello Python!')                                   #실행 결과= Hello Python!
print('Math=', 300)                                      #실행 결과= Math=300
```

・sep 키워드

```
print('math', 'eng', 'kor', sep=',') #","로 출력 항목 분리, 실행 결과= math, eng, kor
print('math', 'eng', 'kor', sep='\t')                    #탭으로 출력 항목 분리
실행 결과= math    eng    kor
print('math', 'eng', 'kor', sep='\n')                    #줄바꿈 문자로 항목 분리
실행 결과= math
           eng
           kor
```

・end 키워드

```
for i in range(0, 5):
    print(i, end=' ')          #줄바꿈 문자 대신 ' ' 출력, 실행 결과= 01234
print(' ')                                               #줄바꿈 문자 출력

for i in range(0, 4):
    print(i, end=',')          #줄바꿈 문자 대신 ',' 문자 출력, 실행 결과= 0,1,2,3,4

print(4)                                                 #숫자 4와 줄바꿈 문자 출력

for i in range(0, 5):
    print(i, end='\n')                                   #출력할 때마다 줄바꿈 문자 출력
실행 결과= 0
           1
           2
           3
           4
```

(3) format 메소드

・다양한 데이터 형식 지정

```
print('math={0}, eng={1}'.format(95, 100))                          #숫자 인덱스 사용
실행 결과 =math=95, eng=100
print('math={}, eng={}'.format(95, 100))                            #인덱스 번호 생략
실행 결과 =math=95, eng=100
print('math={math}, eng={eng}'.format(math=95, eng=100))            #인덱스 대신 이름 지정
실행 결과 =math=95, eng=100
```

・format에 값 대신 변수 사용

```
math=95
eng=100
print('sum {0}+{1}={2}'.format(math, eng, math+eng))                #변수 사용
실행 결과= sum 95+100=195
```

5. 파이썬 함수

· 함수를 사용하면 좋은 점

· 코드를 재사용할 수 있다.

· 반복되는 코드를 1개의 함수로 구현하면 프로그램의 크기를 줄일 수 있다.

· 유사한 기능을 함수로 구현해서 분리함으로써 프로그램의 가독성을 높일 수 있다.

(1) 함수의 정의

```
def sum( ):
    return (10+20)

print("sum=", sum( ))                    #sum( ) 함수 호출, 실행 결과= sum=30
```

(2) 함수의 인자와 리턴값

```
def sum(math, eng):
    return (math+eng)

print("sum=", sum(30, 40))              #sum(30, 40) 함수 호출, 실행 결과= sum=70
```

(3) 언패킹 인자값 전달

```
def sum(math, eng, kor):
    return (math+eng+kor)

score=[90, 80, 100]
print("sum=", sum(*score)) #sum(90, 80, 100)과 같음, 실행 결과= sum=270
```

(4) 가변 인자값 전달

```
def sum(*scores):
    sum=0
    for arg in scores:                   #가변 인자값은 for문 사용
        sum=sum+arg

    return (sum)

print("sum=", sum(30, 40, 80, 90))      #4개의 인자 전달, 실행 결과= sum=240
print("sum=", sum(30, 40, 80))          #3개의 인자 전달, 실행 결과= sum=150
```

(5) 여러 개의 값 리턴

```
def sum_avg(*scores):                                   #가변 인자로 넘어온 점수 합계 변수
    sum=0                                               #과목 점수 평균
    avg=0                                               #가변 인자(과목점수) 개수 카운트
    count=0
    for arg in scores:
        sum=sum+arg                                     #전체 과목 점수 합계
        count=count+1                                   #평균을 구하기 위해서 과목 수 카운트

    avg=sum/count                                       #평균값 구하기
    return (sum, arg)                                   #합계, 평균 리턴

ret_sum=0
ret_avg=0

ret_sum, ret_avg=sum_avg(30, 40, 80, 90)                #4개 과목의 합계와 평균
print("sum=", ret_sum, ", avg=", ret_avg)               #실행 결과= sum=240, avg=60.0

ret_sum, ret_avg=sum_avg(30, 40, 80)                    #3개 과목의 합계와 평균
print("sum=", ret_sum, ", avg=", ret_avg)               #실행 결과= sum=150, avg=50.0
```

(6) 변수의 통용 범위

```
sum=0                                                   #전역 변수
avg=0                                                   #전역 변수

def sum_avg(*scores):
    count=0
    global sum                                          #변수 앞에 global 키워드 사용
    for arg in scores:
        sum=sum+arg
        count=count+1

    global avg                                          #변수 앞에 global 키워드 사용
    avg=sum/count


sum=0                                                   #sum 변수 초기화
sum_avg(30, 40, 80, 90)
print("sum=", sum, ", avg=", avg)                       #실행 결과= sum=240, avg=60.0

sum=0                                                   #sum 변수 초기화
sum_avg(30, 40, 80)
print("sum=", sum, ", avg=", avg)                       #실행 결과= sum=150, avg=50.0
```

6. 파이썬 클래스

(1) 클래스 정의

```
class 클래스 이름:
    클래스 변수-1(멤버변수)
    클래스 변수-2(멤버변수)
    …
    def 메소드( )
        사용자 코드
        …
```

(2) 클래스 사용

```
class terran:
    hp=30                                          #멤버변수: 체력
    power=10                                       #멤버변수: 공격력
    atk=“”                                         #멤버변수: 공격 표시

    def attack(self):
        print(self.atk)



u1=terran( )                                       #클래스 인스턴스 생성
u2=terran( )                                       #클래스 인스턴스 생성

u1.atk=“---->”                                     #u1 공격 동작
u2.atk=“<----”                                     #u2 공격 동작

while u1.hp!=0 and u2.hp!=0:                       #u1, u2의hp가 0이 될 때까지 반복
    at=input(“attack(1 or 2):”)                    #“1”을 입력하면 u1이 공격
    if at==“1”:
        u1.attack( )
        u2.hp=u2.hp-u1.power                       #공격 방향에 따라서 hp 감소
    elif at==“2”:                                  #“2”을 입력하면 u2가 공격
        u2.attack( )
        u1.hp=u1.hp-u2.power                       #공격 방향에 따라서 hp 감소

    print(“u1.hp=”, u1/hp, “, u2.hp=”, u2.hp)      #공격이 끝나면 hp 표시

if u1.hp==0:                                       #u1의 hp가 0이면 u2 승리
    print(“u2 win!”)
elif u2.hp==0:                                     #u2의 hp가 0이면 u1 승리
    print(“u1 win!”)
```

7. 파이썬 모듈

(1) 파이썬 모듈화

```
class class_sum:
    math=0
    eng=0

    def set_score(self, math, eng):
        self.math=math
        self.eng=eng
        return

    def do_sum(self):
        return (self.math+self.eng)
```

```
import module_sum                                   #import <모듈이름>

sum=module_sum.class_sum( )                         #모듈 초기화

sum.set_score(30, 80)

print("sum=", sum.do_sum())                         #실행 결과= sum=110
```

(2) 파이썬 내장 모듈

```
import math as m                                    #수학 함수 가져오기

print(m.sqrt(2.0))
help('modules')                                     #설치된 모든 모듈의 리스트 출력
```

<실행 결과>

1.4142135623730951

Please wait a moment while I gather a list of all available modules...

1_hello_world	asynchat	help_about	runscript
__future__	asyncio	history	sched
__main__	asyncore	hamc	scrolledlist
_abc	atexit	html	search
_ast	audioop	htttp	searchbase
_asyncio	autocomplete	hyperparser	searchengine
_bisect	autocomplete_w	idle	secrets