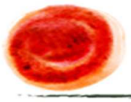
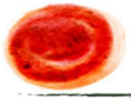


02. 함수, 바이트 변환, 파일 입출력



함수, FUNCTION

함수, function

- 특정 기능 또는 처리를 위한 명령문들로 구성
- 종류
  - 기본 제공 함수, built-in function
  - 사용자 정의 함수, user-defined function

함수 호출

- 정의된 함수의 기능 사용
- 함수 이름과 파라미터를 이용하여 호출

func_name(*[parameters]*)

variable = **func_name**(*[parameters]*)

- 함수 호출의 예:

```
year = int(input('input year : '))
print('Total = ', total, 'Average = ', average)
```

함수 이름

파라미터

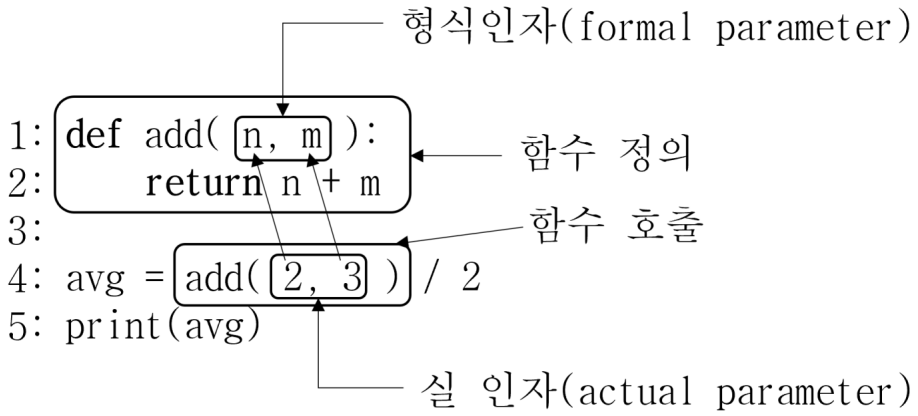
함수 정의

- 함수의 이름, 파라미터 및 기능 정의
- 방법 :

```
def func_name( [parameters]) :
    function_body
    return value
```

  - 반환 값이 없으면 return 생략

함수 정의 및 호출



리스트의 합과 평균을 계산하는 함수의 예

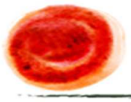
```
1  def calcTotal(lst):
2      total = 0
3      for value in lst:
4          total = total + value
5      return total
6  def calcAverage(lst):
7      return calcTotal(lst)/len(lst)
8  data = [1, 2, 3, 4, 5]
9  print('average=', calcAverage(data))
```

형식 인자의 기본 값(default value)

- 실 인자가 전달되지 않을 경우에 형식 인자에 저장되는 기본 값
- 기본값 설정 방법 :

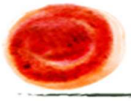
```
def func_name( [parameter=default_value] ) :
    function_body
    return value
```

- 기본 값을 갖지 않는 형식 인자는 기본 값을 갖는 형식 인자 다음에 올 수 없음.



기본 값을 갖는 형식 인자 사용의 예:

```
1  def foo(a, b, c=0, d=0):
2      print(a, b, c, d)
3  foo(1, 2)
4  foo(1, 2, 3)
5  foo(1, 2, 3, 4)
```



잘못된 기본 값의 형식 인자 사용 예:

```
def foo(a, b, c=0, d):
    print(a, b, c, d)
foo(1, 2, 3)
```

실 인자와 반환 값의 종류

- 정수, 실수, 문자열, 함수 등…
- 다양한 실 인자 사용의 예:

```
1  def gettotal(lst):
2      total = 0
3      for v in lst:
4          total = total + v
5      return total
6  def calc(fn, data):
7      return fn(data)
8  lst = [1, 2, 3, 4]
9  print(calc(gettotal, lst))
```

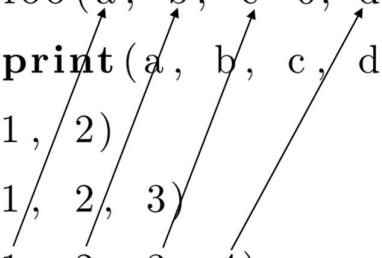
파라미터의 종류

- 위치 기반 인자, positional parameter
- 키워드 기반 인자, kerword parameter
- 위치 기반만의 인자, positional-only parameter
- 키워드 기반만의 인자, keyword-only parameter

위치 기반 인자, positional parameter

- 실 인자가 나열된 순서대로 형식 인자로 전달
- 기본적인 인자 전달 방법

```
1  def foo(a, b, c=0, d=0):
2      print(a, b, c, d)
3  foo(1, 2)
4  foo(1, 2, 3)
5  foo(1, 2, 3, 4)
```

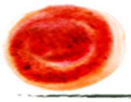


키워드 기반 인자, keyword parameter

- 실 인자가 전달될 형식 인자를 직접 지정

$$paramter\_name=value$$

- 실 인자와 형식 인자의 순서 무관



위치 기반 인자와 키워드 기반 인자 전달의 예

```
1  def foo(a, b, c, d):
2      print(a, b, c, d)
3  foo(1, 2, 3, 4)
4  foo(d=1, c=2, b=3, a=4)
5  foo(1, 2, d=3, c=4)
```

위치 기반만의 인자 전달, positional-only parameter

- 실 인자의 순서대로만 형식 인자로 전달
- 키워드 기반 인자 전달 방법 사용 X
- 형식 인자로 심볼 /을 사용
  - 앞쪽의 형식 인자는 무조건 위치 기반

키워드 기반만의 인자 전달, keyword-only parameter

- 키워드 기반 인자 전달 방법만 사용
- 위치 기반 인자 사용 X
- 형식 인자로 심볼 *을 사용
  - 뒤쪽의 형식 인자는 무조건 키워드 기반

위치 기반만의 인자와 키워드 기반만의 인자 전달 예

```
1  def foo(a, b, /, c, d):
2      print(a, b, c, d)
3  def foo2(a, b, *, c, d):
4      print(a, b, c, d)
5  foo(1, 2, 3, 4)
6  foo(1, 2, d=3, c=4)
7  # foo(a=1, 2, 3, 4)
8  foo2(1, 2, d=3, c=4)
9  foo2(b=1, a=2, d=3, c=4)
10 # foo2(1, 2, 3, 4)
```

10진번 이하 n-진법의 숫자 유효성 검사 예

```
1 def isValidDigit(n, /, *, radix=10):
2     return radix >= 2 and radix <= 10 and n >= 0 and n < radix
3 print(isValidDigit(2))
4 print(isValidDigit(2, radix=8))
5 print(isValidDigit(2, radix=2))
```

가변 개수의 파라미터 전달

- 0 또는 그 이상의 실 인자를 한 개의 형식 인자로 전달
- 형식 인자 이름 앞에 심볼 * 사용
- var-positional argument
- 실 인자들을 원소

```
def func_name(*parameter):
    function_body
    return value
```

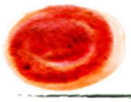
가변 개수의 파라미터 전달의 예

```
1  def foo(a, b=0, *c):
2      print(a, b, c)
3  foo(1, 2)
4  foo(1, 2, 3, 4, 5)
5  foo(1)
```

가변 수의 키워드 기반 인자

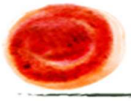
- 0 또는 그 이상의 키워드 기반 인자를 한 개의 형식 인자로 전달
- 키워드 기반 인자를 원소로 갖는 딕셔너리로 전달
- 형식 인자 이름 앞에 심볼 ** 사용
- var-keyword argument

```
def func_name(**parameter) :
    function_body
    return value
```



가변 수의 키워드 기반 인자 사용의 예:

```
1  def foo(a, b=0, **c):
2      print(a, b, c)
3  foo(1)
4  foo(1, 2, n1=2, n2=3)
```



가변 개수의 실 인자 사용 예 : 가변 개수의 데이터 합 계산

```
1  def calcTotal(*nums):
2      total = 0
3      for v in nums:
4          total = total + v
5      return total
6  print(calcTotal())
7  print(calcTotal(1, 2, 3, 4, 5))
```

데이터의 바이트열 변환

데이터와 바이트 열의 변환

- struct 모듈 이용
- pack() : 데이터 → 바이트열
  - 변환된 바이트열에는 원본 데이터 형에 대한 정보 X
  - 그냥 바이트 데이터들…
- unpack() : 바이트열 → 데이터

struct.pack() 사용

- 데이터를 바이트열로 변환
- 형식 : struct.pack(fmt, v1, v2, v3, …)

  인자 :
  fmt : 변환할 형식, 정수 – i, 실수 – d
  v1, v2, v3 : 변환할 값

  반환 :
  바이트 스트림

struct.unpack() 사용

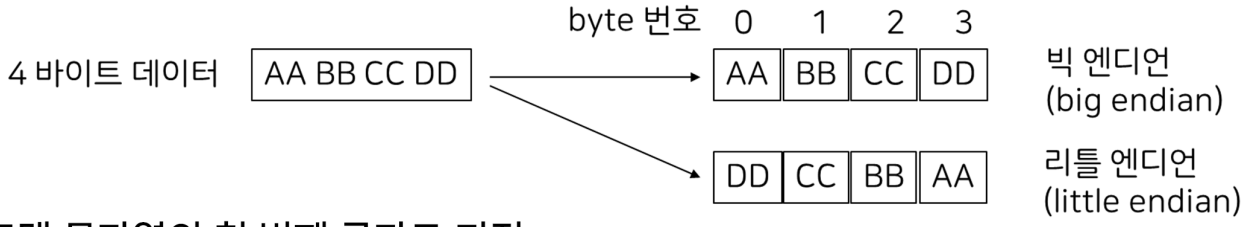
- 바이트열을 변환된 데이터를 원소로 갖는 튜플
- 형식 : struct.unpack(fmt, bytes)

  인자 :
  fmt : 변환할 형식, 정수 – i, 실수 – d
  data : 데이터로 복원할 바이트 스트림

  반환 :
  복원된 데이터 튜플

포맷 문자열

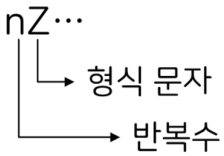
- 데이터-바이트 간 변환 방법
- 바이트 순서 지정
  - 여러 바이트(정수 등)로 표현되는 데이터를 위한 변환된 바이트 순서



  - 포맷 문자열의 첫 번째 글자로 지정
    - < : 리틀 엔디언
    - > : 빅 엔디언 # : 네트워크(빅 엔디언)
    - @ : 프로세서에 종속(생략시 기본, x86, AMD)

포맷 문자열

- 형식 :



형식문자	의미	바이트 수
c	문자	1
b	부호있는 정수(-128~+127)	1
B	부호없는 정수(0~255)	1
?	True/False	1
h	부호있는 정수	2
H	부호없는 정수	2
i	부호있는 정수	4
I	부호없는 정수	4
l	부호있는 정수	4
L	부호없는 정수	4
f	실수	4
d	실수	8
s	문자열의 1 byte	1

포맷 문자열의 예 - 1

```
bytes1 = struct.pack('>ih', 1, -1)
bytes2 = struct.pack('<ih', 1, -1)
bytes3 = struct.pack('ih', 1, -1)
print(bytes1)
print(bytes2)
print(bytes3)

tup1 = struct.unpack('>ih', bytes1)
tup2 = struct.unpack('<ih', bytes2)
print(tup1, tup2)
```

```
b'\x00\x00\x00\x01\xff\xff'
b'\x01\x00\x00\x00\xff\xff'
b'\x01\x00\x00\x00\xff\xff'
(1, -1) (1, -1)
```

포맷 문자열의 예 - 2

```
import struct

d1 = b'₩xbf₩xc0₩x00₩x00₩xff₩xff₩x80₩x80'
tp = struct.unpack('>fhbB', d1)

print(tp[0], tp[1], tp[2], tp[3])
```

```
-1.5 -1 -128 128
```

문자열의 encode, decode

- 문자열 → 바이트열

  string. encode()
  string. encode('utf-8')

  반환 :
  바이트열

- 바이트열 → 문자열

  byte_stream. decode()
  byte_stream. decode('utf-8')

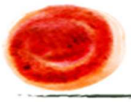
  반환 :
  문자열

유니코드를 이용한 문자열 출력, encode, decode

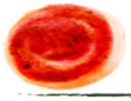
- 유니코드 형식 : '₩u????'

```
recycle_unicode = '₩u267b안녕'
print(type(recycle_unicode), len(recycle_unicode), recycle_unicode)

recycle_encode = recycle_unicode.encode()
print(type(recycle_encode), len(recycle_encode), recycle_encode )
```

```
<class 'str'> 3 ♻안녕
<class 'bytes'> 9 b'\xe2\x99\xbb\xec\x95\x88\xeb\x85\x95'
```



파일 입출력



파일, file

- 데이터 저장의 기본 단위

- 종류
  - 텍스트 파일
  - 이진 파일

파일, file

- 파일 사용 절차
  1. 파일 열기
  2. 파일 읽기/쓰기
  3. 파일 닫기

파일 열기

- 어떤 파일을 어떤 목적으로?
- 형식 :

file_variable = *open(파일명, 모드)*

- 모드
  - r(read), w(write), r+(read/write)
  - a(append)
  - t(text file), b(binary file)

파일 닫기

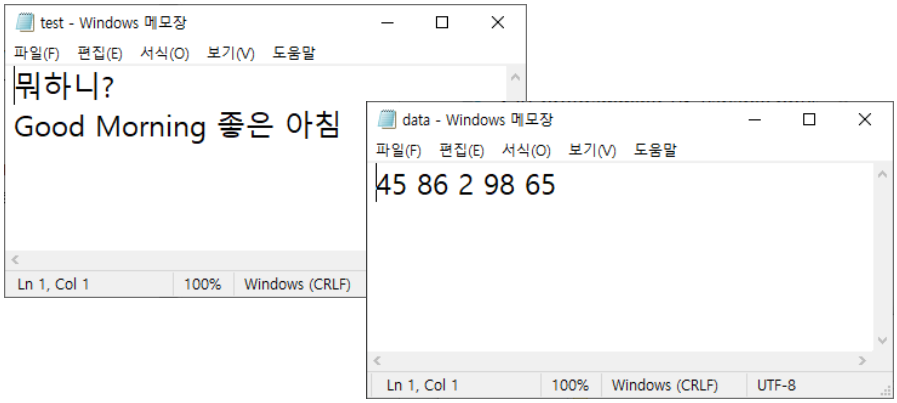
- 형식 :

file_variable.close()

텍스트 파일 입출력

텍스트 파일, text file

- 문자 형태로 파일 저장 또는 읽기
  - 윈도10의 메모장 프로그램으로 내용 확인 가능
- 예
  - .txt 파일



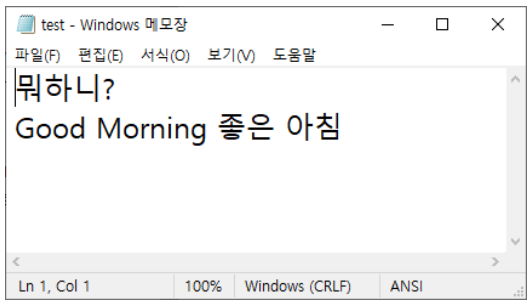
출력

- 화면 출력
  - print()

- 파일 출력
  - file_variable.*write(문자열)*
  - file_variable.*writelines(문자열_리스트)*

파일 출력의 예:

```
lst = ['Good ', 'Morning ', '좋은 아침₩n']

myFile = open('test.txt', 'w')

myFile.write('뭐하니?₩n')
myFile.writelines( lst )

myFile.close()

print('File Write End...')
```



수 데이터의 출력

- 문자열로 변환 후 파일 저장
- str() 함수 이용

```
import random

tfile = open('data.txt', 'w')
for i in range(0, 5):
   tfile.write( random.randint(0,100) + ' ')

tfile.close()

print('End of Program')
```

```
import random

tfile = open('data.txt', 'w')
for i in range(0, 5):
   tfile.write( str( random.randint(0,100) ) + ' ' )
   # tfile.write( '%d ' % random.randint(0,100) )

tfile.close()

print('End of Program')
```

```
Tr                                    last):
  File "D:/temp/00-동영상강의/Python/sample2.py", line 5, in <module>
    tfile.write( ' ' + random.randint(0,100))
TypeError: must be str, not int
```

입력

- 키보드 입력
  - input() : 문자열 반환
- 파일 입력
  - file_variable.*read()* ; 파일의 모든 문자 읽음
  - file_variable.*read(n)* ; 파일에서 n개 문자 읽음
  - file_variable.*readline()* ; 파일에서 한 줄 읽음
  - file_variable.*readlines()* ; 파일에서 모든 줄을 읽어 리스트 반환

파일 입력의 예:

```
rfile = open( 'test.txt', 'r')
done = False

while not done :
    linedata = rfile.readline()
    if linedata == '':
        done = True
    else :
        print( linedata, end='' )

rfile.close()
```

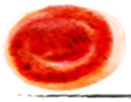
수 데이터 파일 입력

- 파일에서 문자열로 읽은 후 수로 변환
- int(), float() 이용

```
file = open( 'data.txt', 'r' )
nums = file.read().split()

total = 0

for n in nums:
    print( n )
    total += int( n )

print( 'Total = %d' % total )
```

```
45
86
2
98
65
Total = 296
```

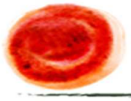
이진 파일(BINARY FILE)



바이너리 파일, binary file

- 문자 형태로 파일 저장 또는 읽기 X
- 바이트 단위로 파일 저장 또는 읽기

- 예
  - MP3 파일, 엑셀 파일, 실행 파일 등등…



이진 파일 입출력

- 저장할 데이터는 바이트열(byte stream)로 변환
  - 수(정수, 실수) 데이터는 struct 모듈의 pack() 이용
  - 문자열은 encode() 이용
- 이진 파일에서 읽은 바이트열을 원래 데이터로 복원
  - 수(정수, 실수) 데이터는 struct 모듈의 unpack() 이용
  - 문자열은 decode() 이용

이진 파일의 데이터 입출력

- 파일 열기 – open() 에서 'rb' 또는 'wb' 사용
- 파일 읽기/쓰기 – read()/write()
- 파일 닫기 – close()

수 데이터의 이진 파일 저장 및 읽기 예 :

```
import struct

age = 20
height = 178.5

# binary file write
file = open( 'data.bin', 'wb')
byteData = struct.pack( 'id', age, height )
file.write( byteData )
file.close()

# binary file read
file = open( 'data.bin', 'rb')
byteData = file.read()
data = struct.unpack( 'id', byteData )
for d in data:
    print( d )
file.close()
```

문자열의 이진 파일 저장 및 읽기 예 :

```
sampleString = 'Good Monring'

# binary file write for string
bfile = open( 'str.bin', 'wb' )
bfile.write( sampleString.encode() )
bfile.close()

# binary file read for string
bfile = open( 'str.bin', 'rb')
data = bfile.read().decode()
print( data )
bfile.close()
```

문자열과 수 데이터의 이진 파일 저장 및 읽기

- 파일에 저장되어 있는 바이트들은 '그냥' 바이트들임
  - 저장한 데이터의 길이, 타입에 대한 정보 X
- 문자열의 길이, 수 데이터이 타입 및 수 정보는 프로그램에서 관리

문자열과 수 데이터의 이진 파일 저장 예:

```
import struct

str = 'Good Morning'
lst = [ 1, 2, 3, 4, 5, 6, 7]

byteData = struct.pack( 'i',len( str ) )
byteData += str.encode()
byteData += struct.pack( 'i', len( lst ) )
for d in lst :
    byteData += struct.pack( 'i', d )

file = open( 'bdata.bin', 'wb')
file.write( byteData )
file.close()
```

문자열과 수 데이터의 이진 파일 읽기 예:

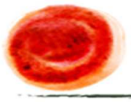
```
import struct

file = open( 'bdata.bin', 'rb' )
byteData = file.read()
file.close()

strlen = struct.unpack( 'i', byteData[:4] )
str = byteData[4:4+strlen[0]].decode()
posIntData = 4 + strlen[0]
nofInt = struct.unpack( 'i', byteData[posIntData:posIntData+4] )
lst = struct.unpack( 'i'*nofInt[0], byteData[posIntData+4:] )

print( str )
for d in lst:
    print( d )
```

문자열, 수를 저장하고, 읽어오는 예:

```
import struct

age = 20
name = '홍길동'
height = 178.5

en_name = name.encode('utf-8')

fd =open('data.dat','wb')
fd.write(struct.pack('if%ds' % len(en_name), age, height, en_name))
fd.close()

fd = open('data.dat','rb')
readdata = fd.read()
fd.close()
strlen = len(readdata) - struct.calcsize('if')
data = struct.unpack('if%ds' % strlen, readdata)
print(data[0], data[1], data[2].decode('utf-8'))
```

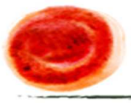
실습 #1

- 4 바이트 정수 -1이 0xFFFFFFFF임을 확인하라.
- 4 바이트 실수 값 1.5와 -1.5가 각각 0x3FC00000, 0xBFC00000임을 확인하고, 왜 그 값이 나왔는지를 간단히 설명하라.(Hint: IEEE 754)



실습 #2

- 텍스트 파일의 이름을 인자로 받아 그 파일에 저장되어 있는 정수를 읽어 그것을 원소로 하는 리스트를 반환하는 함수를 정의하라. 단, 이 파일은 각 정수 사이에 공백 문자가 한 개 삽입되어 있으며, 저장된 정수 데이터의 수는 사전에 알려져 있지 않다.
- 리스트를 인자로 받아 그 리스트에 저장되어 있는 값의 합을 계산하고 반환하는 함수를 정의하라.
- 앞서 정의한 함수를 이용하여 제공된 data1000.dat에 저장된 정수 데이터의 합을 출력하라.



실습 #3

- 이진 파일의 이름을 인자로 받아 그 파일에 저장되어 있는 4 바이트 길이의 정수를 읽어 그것을 원소로 하는 리스트를 반환하는 함수를 정의하라. 단, 이 파일에 저장된 정수 데이터의 수는 알려져 있지 않다.
- 리스트를 인자로 받아 그 리스트에 저장되어 있는 값의 합을 계산하고 반환하는 함수를 정의하라.
- 앞서 정의한 함수를 이용하여 제공된 data1000.bin에 저장된 정수 데이터의 합을 출력하라.

Q&A