2주차 수업 내용

- 복습
- 입출력 함수 기초
- 자료형의 종류
- 여러 연산자

지난 주 복습

- 식을 주면 계산이 가능함
- 프로그래밍 *

기본 입출력 함수

- 프로그램을 실행해 원하는 결과를 얻으려면 적절한 값을 프로그램에 입력한 후
- 입력값에 따른 계산 결과를 화면에 출력해 확인해야 한다.
- 이 경우 필요한 것이 기본 입출력 함수인 input()과 print() 함수이다.

input() 함수

- 사용자가 키보드를 이용해 숫자 또는 문자를 입력받을 때 사용하는 함수
- 주어진 메시지를 화면에 출력하고 사용자의 입력을 기다린다
- 사용자가 입력하면 input()은 그 입력을 문자열 형태로 반환
- 따라서 숫자를 입력해도 숫자 자체가 아니라 숫자로 이루어 진 문자열을 입력받음
- 숫자를 이용하려면 적절하게 정수 또는 실수로 바꾸어야 한다.
- 이때 사용하는 함수가 각각 int()와 float()이다.

input() 함수 - 사용예 1

- 정수를 키보드에서 입력받는 예

```
>>> n = input('integer number : ')
integer number : 3
>>> n              # 이렇게 입력받은 것은 문자임
'3'
>>> n = int(n)   # 문자 n을 정수로 변환
>>> n
3
```

- 다음과 같이 한번에 할 수도 있다.

```
>>> n = int(input('integer number : '))
integer number : 3
>>> n
3
```

print() 출력함수

- print() 함수는 숫자나 문자열 등의 데이터를 화면에 출력할 때 사용
- 여러 종류 변수(정수, 실수, 문자열)의 출력 예

```
n = 10
>>> print('n=', n) # , 있음에 주의
n= 10
>>> print('n=%d' %n) # ,없음에 주의. %d는 정수 출력 형식지정
n=10
>>> fn = 1.2
>>> print('fn=%f' %fn)     # %f는 실수 출력 형식지정
fn=1.200000
>>> print('n=',n,'fn=',fn,'s=',s)
n= 10 fn= 1.2 s= string
>>> print('n=%d' %n,'fn=%f' %fn,'s=%s' %s)
n=10 fn=1.200000 s=string
```

참고 자료

Built-in Data Types

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

- Text Type: str
- Numeric Types: int, float, complex
- Sequence Types: list, tuple, range
- Mapping Type: dict
- Set Types: set, frozenset
- Boolean Type: bool
- Binary Types: bytes, bytearray, memoryview
- None Type: NoneType

https://www.w3schools.com/python/python_datatypes.asp

자료형

파이썬 언어에는 다음과 같은 자료형이 있다.

- Bool(부울, 불)형 : True, False의 값을 갖는 자료형
- 숫자형 :
  정수형, 실수형, 복소수형,
- 문자열, 리스트, 딕셔너리, 튜플, 세트 등

자료형을 알고싶으면 type() 함수를 사용 (다음을 shell에서 직접 실행해 보자)

```
>>> type(123)
>>> type(1.2)
>>> type('str')
>>> type(3 > 2)
>>> type((1,2))
```

부울(Bool)형

- 부울(Bool)형은 참(True) 또는 거짓(False)값 중 하나를 갖는 데이터형
- 비교의 결과를 True 또는 False로 저장하는 데 사용
- 데이터형을 알고 싶으면 type() 함수를 사용
- bool() 함수를 사용하면 결과값을 True/False로 알 수 있음

```
>>> type (3 > 2)
<class 'bool'>
>>>
>>> bool(5>3)
True
```

자료형의 True, False

- 문자열, 리스트, 튜플, 딕셔너리 등의 값이 비어 있으면(`""`, `[]`, `()`, `{}`) False이고 비어 있지 않으면 True
- 숫자에서는 그 값이 0이면 False
- None은 False

```
>>> bool(0.0)
False
>>> bool(0.0000001)
True
>>> bool(None)
False
>>>
```

숫자형(Number) 자료형

- 숫자 형태로 이루어진 자료형
- 정수형(10진수, 2진수, 8진수, 16진수)
- 실수형
- 복소수형

정수형

- 소수점이 없는 숫자를 표현하는 데이터형

```
>>> intVar = 0
>>> type(intVar)
<class 'int'>
```

- 10진수가 아닌 정수 표현 앞에는 접두어가 붙는다

  (2진수는 `0b` 또는 `0B`, 8진수는 `0o` 또는 `0O`, 16진수는 `0x` 또는 `0X`로 시작)

```
1, 23, 3493 # 10진수
0b010101, 0B110010 # 2진수
0o1, 0o27, 0o6645 # 8진수
0x1, 0x17, 0xDA5 # 16진수
```

- bin(), oct(), hex(), int() 함수는 각각 2진수, 8진수, 16진수, 10진수로 변환하는 함수

실수형(floating-point)

- 소수점이 포함된 숫자

```
>>> a = 1.2
>>> type(a)
<class 'float'>
```

- 지수 표현 방식 :

  `4.1e10` 또는 `4.1E10` 처럼 표현 : $4.1 * 10^{10}$

- 식별을 쉽게 할 수 있도록 _를 사용한 숫자의 표현도 가능(파이썬 3.6부터 적용)

```
>>> 100_000.000_0001, 0xFF_FF, 0o7_777, 0b_1010_1010
(100000.0000001, 65535, 4095, 170)
>>> 0.00_10_27
0.001027
```

복소수형

복소수를 사용하려면 수의 허수부에 `j` 또는 `J`를 사용한다. 다음의 사용 예를 보자.

```
>>> a=1+1j
>>> b=1-1J
>>> a+b
(2+0j)
>>> a-b
2j
>>> a*b
(2+0j)
>>> a/b
1j
```

복소수형 관련 연산

- 복소수의 실수부분과 허수부분

```
>>> a = 1+1j
>>> a.real
1.0
>>> a.imag
1.0
```

- 복소수 데이터형을 알고 싶으면

```
>>> type(a)
<class 'complex'>
```

문자열(string)

- 큰따옴표("...") 또는 작은따옴표('...')로 둘러싸인 글자 또는 숫자 데이터

  `"String" 'string123' '123'`

- 연속되는 큰따옴표(""") 3개 또는 작은따옴표(''') 3개로 둘러싸인 여러 라인의 데이터 (주석처리에 사용되기도 함)

```
"""
  multi
  line
  data
"""
```

- 문자열 데이터형의 확인

```
>>> type('python')
<class 'str'>
```

<교재 02-2 ~ 6의 문자열, 리스트, 튜플, 딕셔너리, 집합 자료형은 추후 배움>

연산자의 종류

- 계산 또는 필요한 일을 수행하는 데 연산자가 연산항과 같이 사용됨
- 연산자의 종류
  - 지정연산자
  - 산술연산자
  - 비교(관계)연산자
  - 논리연산자
  - 비트연산자
  - 복합지정연산자
  - 멤버연산자
  - 식별연산자

지정연산자(assignment operator)

- 대입연산자, 할당연산자 라고도 함
- 변수나 표현에 어떤 값을 지정할 때 사용하는 연산자

```
i = 3
i = i + 2
f = 1.2
```

- 위의 `=` 기호가 지정연산자인데 오른쪽(우변)의 값을 먼저 계산한 후
  그 값을 왼쪽(좌변)의 변수 값으로 지정한다는 의미
- 수학의 등호 의미(좌변과 우변이 같다)가 아님
- 따라서 좌변, 우변을 바꾸어서 사용 불가
- 오른쪽의 i는 현재값(변경전 값), 왼쪽의 i는 새로운 값(변경후 값)으로 생각

산술연산자(arithmetic operator)

- 덧셈, 뺄셈, 곱셈, 나눗셈을 위한 사칙연산자(`+ - * /`), 정수나눗셈(`//`), 나머지연산자(`%`), 거듭제곱연산자(`**`)가 있다
- 정수나눗셈(//) : 정수끼리 나눗셈을 할 때는 소수점이하의 계산을 하지 않고 버리기 때문에 주의(예: 7//3의 결과는 2)
- 나머지 연산자(%)는 정수끼리의 나눗셈에서 나머지를 구할 때 사용되며, 실수형 데이터에는 적용불가. 다음 참조

```
>>> c = 10 % 3
>>> c
1
```

- 연산자의 우선순위는 `** * / %`가 `+ -` 보다 높다.

산술연산자 요약

연산자	설명	사용예
x + y	x와 y를 더함	c = a + b
x - y	x에서 y를 뺌	s = a - b
x * y	x와 y를 곱함	m = a * b
x / y	x를 y로 나눔	d = a / b
x // y	x를 y로 나눔(정수 나눗셈)	d = a // b
x % y	x를 y로 나눈 나머지	r = a % b
x ** n	x의 n 제곱	r = 2 ** 5

비교연산자(comparison operator)

- 연산항의 대소관계 또는 동등관계를 판정하고자 할 때 사용하는 연산자
- 관계연산자라고 부르기도 하며 연산의 결과는 부울형 True 또는 False이다
- 주로 뒤에 나오는 조건문이나 반복문에서 많이 사용
- 두 연산항의 크고 작음(`> < >= <=`)을 비교하거나 같거나 다름(`== !=`)을 비교하는데 사용

```
>>> a = 2; b = 1
>>> a > b
True
>>> a <= b
False
>>> a == b
False
>>> a != b
True
```

참고자료

eight comparison operations in Python

Operation	Meaning
<	strictly less than
<=	less than or equal
>	strictly greater than
>=	greater than or equal
==	equal
!=	not equal
is	object identity
is not	negated object identity

https://docs.python.org/3.13/library/stdtypes.html#comparisons

식별연산자(identity operators)

- 식별연산자 is와 is not은 메모리 주소가 같은 지를 시험하는 연산자

```
>>> a = 2
>>> b = 2
>>> a is b
True
>>> a is not b
False
```

- a와 b의 메모리 주소가 같은 지 확인하기 위해 다음과 같이 실행해 보자(물론 각자의 실행 결과는 다름)

```
>>> id(a)
140726593053552
>>> id(b)
140726593053552
```

참고:

- 파이썬은 -5부터 256의 정수값을 미리 지정된 특정메모리 주소에 저장해 놓고 사용
- 다음과 같이 확인

```
>>> a=-5
>>> b=-5
>>> a is b
True
>>> a == b
True
>>> a=-6
>>> b=-6
>>> a is b
False
>>> a == b
True
```

논리연산자(logical operator)

- 복잡한 조건은 비교(관계)연산자만 이용해 표현하기 어려워 논리연산자를 같이 사용
- 예: 성적 A를 받으려면 exam 점수가 90점 이상이거나 또는 report점수가 90점 이상이어야 한다는 조건을 `exam >= 90 or report >= 90`로 표현. `or`가 논리연산자
- 논리연산자는 아래의 3종류 있음
  - 논리곱(and)연산자
  - 논리합(or)연산자
  - 논리부정(not)연산자

논리곱 연산자(and)

- 논리 AND 연산자라고도 함
- 두 개의 연산항 모두가 참일때만 연산결과가 참

```
>>> n = 4
>>> m = 5
>>> n > 0 and m > 0
True
```

- 다음은 정수 i, j가 주어진 조건을 모두 만족할 때에만 값을 출력하는 코드 부분이다.
  따라서 두 조건 중 하나라도 만족되지 않으면 print 문은 수행되지 않는다.

```
if i > 0 and j > 0 :
    print("both i, j are positive")
```

논리합 연산자(or)

- 논리 OR 연산자라고도 함
- 두 개의 연산항 중 하나만 참이어도 연산결과는 참

```
>>> n = 4
>>> m = -5
>>> n > 0 or m > 0
True
```

- 다음은 정수 i, j가 주어진 두 조건 중 하나라도 만족하면 print 문은 수행됨

```
if i > 0 or j > 0 :
    print("at least one of i, j is positive")
```

논리부정 연산자(not)

- 논리 NOT 연산자라고도 함
- 연산항이 True면 연산결과는 False, 연산항이 False이면 연산결과는 True

```
>>> n = True
>>> not n
False
>>> m = False
>>> not m
True
```

논리연산자 요약

연산자	설명	사용 예
x and y	x와 y가 모두 True일때 결과가 True	a=5; print(a>0 and a<10)
x or y	x 또는 y가 True일때 결과가 True	a=5; print(a<-3 or a>3)
not x	x의 부정. x가 True이면 결과는 False	a=True; print(not a)

논리연산자의 우선순위

- not and or의 순서
- and가 or보다는 우선 순위가 높음

```
>>> T = True
>>> F = False
>>> T or F and F
```

- or의 연산순위가 더 높다면 위 결과는 False 이어야 하나, 실제 결과는 True임!!

복합지정연산자

- 산술연산자 또는 비트연산자를 지정연산자를 합쳐놓은 것
- 다음 첫 식을 두번째 식처럼 압축해 사용 가능

```
a = a+1
a += 1
```

- 위의 연산자 +처럼 연산자의 좌우에 두 개의 연산항을 갖는 연산자를 이항연산자라 하는 데 여러 이항연산자가 위에서 처럼 압축된 형태로 사용 가능
- 압축하여 사용할 수 있는 연산자에는 다음과 같은 것들이 있다.
- `+ - * / % // ** << >> & ^ |`
- 복합 연산자를 사용하면 코드를 더욱 간결하고 가독성 있게 작성 가능

멤버 연산자(membership operators)

- 멤버 연산자는 특정값이 문자열, 리스트, 튜플 등에 속해 있는지 알려주는 연산자
- 연산자 `in` 은 두 연산항 중 첫번째 항이 두번째 항에 있으면 True를, 없으면 False를 반환
- 연산자 `not in` 은 두 연산항 중 첫번째 항이 두번째 항에 있으면 False를, 없으면 True를 반환
- 사용 예

```
>>> 'py' in 'python'
True
>>> 1 not in [1,2,3]
False
>>> 1 in (1,2,3)
True
```

비트연산자

- 비트연산자는 정수형 데이터에만 사용. 실수형 데이터에는 사용 불가.
- 비트단위로 연산을 진행하는 연산자
- `&` `|` `^` `>>` `<<` `~`
- `~` 는 1개의 연산항을 필요한 단항연산자
- 나머지는 모두 2개의 연산항을 필요로 하는 이항 연산자

비트 AND 연산자

- 비트 AND 연산자(&)는 비교하는 두 비트가 모두 1일 경우에만 결과가 1이 되는 연산자
- 다음의 예를 검토

```
  1101 0111 0xD7
& 0011 0011 0x33
---------------
  0001 0011 0x13
```

- shell에서 위 결과를 확인

```
>>> 0xD7 & 0x33
19
>>> hex(19)
'0x13'
>>> bin(19)
'0b10011'
```

비트 OR 연산자

- 비트 OR 연산자(|)는 비교하는 두 비트중 하나라도 1이면 결과가 1이 되는 연산자
- 다음의 예를 검토

```
  1101 0111 0xD7
| 0011 0011 0x33
----------------
  1111 0111 0xF7
```

- shell에서 위 결과를 간단히 확인

```
>>> hex(0xD7 | 0x33)
'0xf7'
>>> bin(0xD7 | 0x33)
'0b11110111'
```

비트 Exclusive OR 연산자

- 비트 Exclusive OR 연산자(^)는 두 비트가 서로 다를 때에만 결과가 1이 되는 연산자
- 다음의 예를 검토

```
  1101 0111 0xD7
^ 0011 0011 0x33
---------------
  1110 0101 0xE4
```

- shell에서 위 결과를 간단히 확인

```
>>> hex(0xD7 ^ 0x33)
'0xe4'
>>> bin(0xD7 ^ 0x33)
'0b11100100'
```

왼쪽이동 연산자

- 왼쪽이동 연산자(<<)는 연산항의 비트열을 정해진 양수만큼 왼쪽으로 이동시키는 연산자
- 빈 부분은 0으로 채움
- 연산항은 모두 정수이어야 함.
- 다음의 예를 검토해 보자. b의 값이 얼마일까? shell에서 확인

```
>>> a = 1
>>> b = a << 1
```

- 비트의 열을 한 칸씩 왼쪽으로 이동시키면 값이 두 배가 됨
- 곱셈연산자를 대신하여 정수(2의 거듭제곱수)의 곱셈 가능

오른쪽이동 연산자

- 오른쪽이동 연산자(>>)는 연산항의 비트열을 정해진 양수만큼 오른쪽으로 이동시키는 연산자
- 연산항은 모두 정수이어야 함.
- 다음의 예를 검토해 보자. b의 값이 얼마일까? shell에서 확인

```
>>> a = 8
>>> b = a >> 1
```

- 비트의 열을 한 칸씩 오른쪽으로 이동시키면 값의 반으로 됨
- 나눗셈연산자를 대신하여 정수(2의 거듭제곱수)의 나눗셈 가능

1의 보수 연산자

- 1의 보수 연산자(`~`)는 주어진 수의 1의 보수를 계산함
- 다음의 예를 검토

```
>>> n = 8
>>> ~n
-9
>>> ~n + 1
-8
```

1의 보수

- 주어진 수의 각 비트를 뒤집는 연산(즉 비트1은 0으로, 0은 1로 바꾸는)결과 로 나오는 수

2의 보수

- 1의 보수에 1를 더한 값

연산자 우선 순위

- If two operators have the same precedence, the expression is evaluated from left to right.

Operator	Description
()	Parentheses
**	Exponentiation
+x -x ~x	Unary plus, unary minus, and bitwise NOT
* / // %	Multiplication, division, floor division, and modulus
+ -	Addition and subtraction
<< >>	Bitwise left and right shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
== != > >= < <= is is not in not in	Comparisons, identity, and membership operators
not	Logical NOT
and	AND
or	OR

https://www.w3schools.com/python/python_operators.asp