

# PYTHON PROGRAMMING

## LECTURE 2: VARIABLE & OPERATOR

goorm

**KAIST AI**  
Graduate School of AI

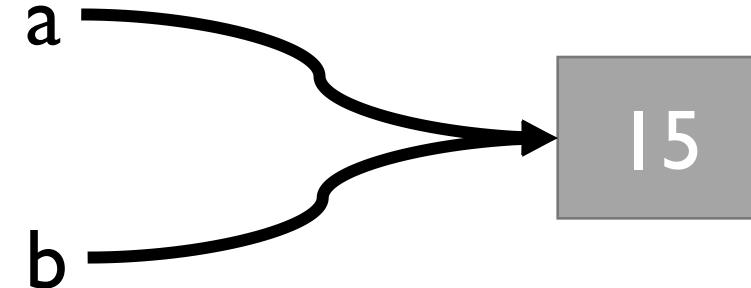
 DAVIAN

# Introduction to Variable

- 변수: 값을 저장하는 공간
  - = 연산자로 대입 연산
- 파이썬 변수의 특징
  - 모든 변수는 메모리 주소를 가르친다. (즉, 모든 것은 포인터다)
  - 변수 명 ← 일종의 이름표
    - 선언한 변수에 특정 공간이 생기는 개념이 아님
    - 필요하면 공간을 만들고 변수명을 붙이는 격

```
>>> a = 15  
>>> b = 4  
>>> a + b  
19
```

```
>>> a = 15  
>>> b = a  
>>> b  
15
```



# How to Name Variables

- 알파벳, 숫자, 언더스코어(\_)로 선언
  - 숫자로 변수명을 시작할 수는 없음
  - Ex) data = 0, \_al2 = 2, \_gg = 'afdf'
- 변수명은 그 변수의 특징이 잘 살아 있게 하자 (**가독성은 중요하다!**)
  - Ex) name = 'Hojun Cho'
- 변수명은 대소문자가 구분
  - Ex) abc와 Abc는 다른
- 변수명으로 쓸 수 없는 예약어가 존재
  - Ex) for, if, else, True 등

# Assigning Variables

- C와 달리 대입 연산이 딱히 반환 값을 가지는 것은 아님

```
>>> (a = 2) == 2
      File "<stdin>", line 1
          (a = 2) == 2
                  ^
SyntaxError: invalid syntax
```

- 연속해서 대입 가능 (C와 같이 뒤에서부터 대입)

```
>>> a = b = 2
>>> a
2
>>> b
2
```

- (Python 3.9 이상) := 연산으로 대입과 동시에 반환 가능

```
>>> (a := 2) == 2
True
```

# Primitive Data Types

원시 자료형: 가장 기본이 되는 자료형

유형			비고	예시
수치자료	정수	int	Overflow가 발생하지 않음 명시적인 Short type 등이 없음	1 2 3 100 -50 1342512515234451234
	실수	float	부동소수점, Double precision	1.7 -5.7 3e5 .08 9. 4.67e-3
	복소수	complex	실수부와 허수부로 표현 허수부는 j 또는 J로 표현	1+8j 1.6+.8J
문자열		string	따옴표로 표현, Unicode 큰 따옴표, 작은 따옴표 차이 없음	'text' "한글도 되요" "a"
논리		bool	참/거짓을 표현 True, False 값 밖에 없음	True False
None		None	None 타입, 일종의 null None 값 밖에 없음	None

## Example: Data Types

```
>>> a = 15      # 정수
>>> a
15
>>> b = 1.5     # 실수
>>> b
1.5
>>> c = 1+5j    # 복소수
>>> c
(1+5j)
>>> d = "text"  # 문자열
>>> d
'text'
>>> e = True    # Boolean, 대소문자에 주의
>>> e
True
>>> f = None    # None, 대소문자에 주의
>>> f           # 아무것도 출력하지 않음
>>>
```

# Arithmetic Operator

- 산술 연산을 위해서 산술 연산자를 활용

연산자	비고
+	덧셈
-	뺄셈
*	곱셈
**	거듭 제곱
/	나누기
//	나누기의 몫
%	나누기의 나머지

```
>>> 50 + 12  
62  
>>> 30 - 46  
-16  
>>> 2 * 6  
12  
>>> 2 ** 3  
8  
>>> 1.7 / 0.8  
2.125  
>>> 13 // 5  
2  
>>> 13 % 5  
3
```

# Bit Operator

- 비트 연산을 위해서 비트 연산자를 활용

연산자	비고
<code>~</code>	비트 부정
<code> </code>	비트합
<code>&amp;</code>	비트곱
<code>^</code>	배타적 비트합
<code>&gt;&gt; &lt;&lt;</code>	비트 시프트

Ex)  $5 \rightarrow 0101_{(2)}$ ,  $\sim 5 \rightarrow 1010_{(2)} \rightarrow -6$

```
>>> ~5  
-6  
  
>>> 2 | 3  
3  
  
>>> 2 & 3  
2  
  
>>> 2 ^ 3  
1
```

## Features of Arithmetic & Bit Operator



- 산술 연산자와 비트 연산자는 대입 연산자와 함께 축약 가능
  - 단,  $a += 1$ 는 In-place 연산이고  $a = a + 1$ 은 Out-place 연산
  - Out-place 명시적으로 새로운 객체 생성
  - In-place 연산은 기존 객체를 수정 시도하고, 불가능할 시 새로운 객체 생성
    - 추후 자세히 설명

```
>>> a = 5  
  
>>> a = a + 1  
>>> a  
6  
  
>>> a += 1  
>>> a  
7
```

```
>>> a = 7  
  
>>> a = a ^ 4  
>>> a  
3  
  
>>> a ^= 4  
>>> a  
7
```

- $a++$  연산은 파이썬에 존재하지 않음

# Condition Operators

- 객체 간의 비교를 위해서 비교 연산자를 활용

비교연산자	비교상태	설명
$x < y$	~보다 작음	$x$ 가 $y$ 보다 작다
$x > y$	~보다 큼	$x$ 가 $y$ 보다 크다
$x == y$	같음	$x$ 와 $y$ 가 값이 같다
$x \text{ is } y$		$x$ 와 $y$ 가 주소가 같다
$x != y$	같지 않음	$x$ 가 $y$ 가 값이 다르다
$x \text{ is not } y$		$x$ 가 $y$ 가 주소가 다르다
$x >= y$	크거나 같음	$x$ 가 $y$ 이상이다
$x <= y$	작거나 같음	$x$ 가 $y$ 이하이다
$x \text{ in } X$	포함	$x$ 가 $X$ 에 포함된다, 1 in [1, 2, 3]
$x \text{ not in } X$	포함하지 않음	$x$ 가 $X$ 에 포함되지 않는다, 1 not in [2, 4, 5]

# Condition Operator

- bool끼리의 연산을 위해서 논리 연산자를 활용
  - 비트 연산자와는 다르다!
  - Python에서의 비교연산은 한번에 평가된다
    - $(1 < 2 < 3) == (1 < 2 \text{ and } 2 < 3)$ ,  $a == 4 > 3$

연산자	비고
not	부정
or	논리합
and	논리곱

```
>>> a = 3
>>> a < 4
True

>>> a > 2
True

>>> 2 < a and a < 4
True

>>> 2 < a < 4
True

>>> not a > 3
True
```

# Operator Priority

연산자	설명
(...) [...] {...}	괄호, 리스트, 딕셔너리 생성
x[...] x(...) x.attr	인덱싱, 함수 호출, 속성 참조
**	거듭제곱
+x -x ~x	단항 연산자
* / // &	곱셈, 나눗셈 등
+	덧셈, 뺄셈
<< >>	비트 시프트
&	비트곱
^	배타적 비트합
	비트합
in not in is is not < <= > >= == !=	포함 & 비교 연산자
not x	논리 부정
and	논리곱
or	논리합
[x] if [condition] else [y]	삼항 연산자 (추후 설명)

# Features of Primitive Data Types

- 문자열 (str) 타입은 따옴표 또는 큰 따옴표로 정의
  - Ex) sentence = “hello”, sentence = ‘python’
  - 기능상 차이는 없으나 한 쪽을 쓰면 다른 쪽을 Text에 넣을 수 있음

```
>>> sentence = 'hello "python"'  
>>> sentence  
'hello "python'"
```

- 원시자료형(Primitive Data Type)들은 불변 타입 (Immutable Type)이다.
  - 파이썬의 모든 것은 객체이기 때문에 원시자료형들 역시 객체
  - 그러나 불변 타입들은 저장된 값이 변하지 않는다!
  - 모든 타입은 물리적 메모리 주소를 가르침 (C에서의 pointer)
  - 원시자료형과 Tuple을 제외한 다른 모든 파이썬 객체는 가변 타입 (Mutable Type)

# Immutable Types & Mutable Types

- 파이썬에서 대입은 원칙적으로 메모리 주소 복사 (즉, 값을 복사하지 않고 같은 주소를 공유)
  - 불변형의 경우 수정이 필요할 경우에 새로운 객체를 생성

```
>>> a = 10                                # int는 불변 타입
>>> b = a                                  # a와 b는 같은 메모리 주소를 가르침
>>> a += 1                                 # a는 불변 타입 -> 객체를 새로 생성해서 할당 (a = a + 1)
>>> a, b, a is b                          # a가 새로 할당되었기 때문에 a is not b
(11, 10, False)

>>> a = [1, 2, 3]                           # List는 가변 타입
>>> b = a                                  # a와 b는 같은 메모리 주소를 가르침
>>> a += [4]                               # a는 가변 타입 -> 원 객체를 수정
>>> a, b, a is b                          # a의 메모리 주소는 변함 없으므로 a is b
([1, 2, 3, 4], [1, 2, 3, 4], True)

>>> a = a + [5]                            # a에 [5]를 추가한 객체를 생성해서 a에 할당
>>> a, b, a is b                          # a가 새로 할당되었기 때문에 a is not b
([1, 2, 3, 4, 5], [1, 2, 3, 4], False)
```

# Features of Data Assignment

- 파이썬에서 적당한 크기의 원시 자료형 대입은 **기존 객체를 할당** (불변 타입이라 상관 없음)

```
>>> a = 1                      # int는 Primitive Type  
>>> b = 1                      # b에 새로 할당한 것 처럼 보이지만 기존 객체를 가져옴  
>>> a is b                     # a와 b는 같은 메모리 주소를 가지는지 확인  
True  
  
>>> a = 13453436                # int는 Primitive Type이지만 복잡한 값을 가짐  
>>> b = 13453436                # b에 값을 새로 할당  
>>> a is b                     # a와 b는 같은 메모리 주소를 가지는지 확인  
False  
  
>>> a = 'text'                  # bool는 Primitive Type, 짧은 텍스트  
>>> b = 'long-long-text'        # bool는 Primitive Type, 긴 텍스트  
>>> a is 'text', a == 'text', b is 'long-long-text', b == 'long-long-text'  
(True, True, False, True)  
  
>>> a = True                    # bool는 Primitive Type, True/False 모두 간단  
>>> a is True  
True  
  
>>> a = None                    # None type은 Primitive Type & None 값만 존재  
>>> a is None  
True
```

# Dynamic Typing

코드 실행 지점에서 데이터의 타입을 결정함

```
int first = 10  
int second = 20  
  
printf("%d", first + second);
```

```
first = 10  
second = 20  
  
print(first + second)
```

C

Python

# Implicit Type Conversion

```
>>> a = True
```

```
>>> a = a + 2
```

```
>>> a
```

```
3
```

```
>>> a = a + 1.5
```

```
>>> a
```

```
4.5
```

```
>>> a = a + 7.1j
```

```
>>> a
```

```
(4.5+7.1j)
```

```
>>> 4 / 3
```

```
1.3333333333333333
```

```
>>> 4 // 3
```

```
1
```

- bool → int → float → complex
- None 타입과, str 타입은 별개
- int간의 나누기 → float (정수 나누기는 // )

```
>>> a = 1
```

```
>>> a + None
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'
```

```
>>> a + 'text'
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

# Explicit Type Conversion

```
>>> a = 12345  
>>> a  
12345  
  
>>> float(a)  
12345.0  
  
>>> complex(a)  
(12345+0j)  
  
>>> str(a)  
'12345'  
  
>>> bool(a)  
True
```

```
>>> int(75.75)  
75  
  
>>> str(75.75)  
'75.75'  

```

```
>>> float('75.75')  
75.75  
  
>>> bool('False')  
True  
  
>>> bool('')  
False
```

- [Type](value)로 명시적 형 변환 가능
  - int(a), float(text), str(value)
  - 적절한 text는 적절한 값으로 변형
  - 실수 → 정수: 소수점 내림
    - 반올림의 경우 round 내장함수 사용
  - 빈문자열, 0, None 은 False로 변환

# Type Checking

```
>>> a = 123  
>>> b = 12.3  
>>> c = '12.3'  
  
>>> type(a)  
<class 'int'>  
>>> type(b)  
<class 'float'>  
>>> type(c)  
<class 'str'>
```

- `type` 함수로 변수의 타입 확인 가능
- `isinstance` 함수로 변수가 지정 타입인지 확인
  - `isinstance([variable], [type])`

```
>>> isinstance(a, float)  
False  
>>> isinstance(b, float)  
True  
>>> isinstance(c, str)  
True
```