Lecture 2

기초 문법과 데이터 구조

Jung-Il Choi

School of Mathematics and Computing (Computational Science and Engineering)

YONSEI UNIVERSITY

Material based on https://post.naver.com/hjchang0314

1. 변수와 자료형

- **변수**
  - 컴퓨터에서 자료가 저장되는 메모리공간의 위치를 의미한다.
  - 변수를 생성한다 = 메모리 공간을 확보한다
  - 변수를 할당한다 = 확보된 메모리 공간에 데이터를 저장한다
  - 변수의 이름은 영문, 밑줄(_), 숫자의 조합으로 설정한다.

```
variable_1 = 1 # 옳은 변수명 #

variable 1 = 1 # 공백 x #
1_variable = 1 # 시작은 숫자 x  #
variable1! = 1 # 특수문자 x#
```

- **자료형**
  - 자료의 기능과 역할에 따라 구분된 종류
  - 기본 자료형(숫자, 문자열, 불)
  - 복합 자료형(리스트, 튜플, 딕셔너리)
  - 사용자 자료형(클래스)

숫자	-30, 0, 23, 0.5, -0.352
문자열	'a', 'hello'
불	True, False
리스트	[10,20,30]
튜플	(10,20,30)
딕셔너리	[1:'키', 2:'몸무게']

2. 기본 자료형

- **숫자**
  - Python은 정수, 실수, 2진수, 8진수, 16진수, 복소수 등의 다양한 종류의 숫자들을 지원한다.

1) 정수형(int)

➔ 정수를 의미하며 다양한 산술연산이 가능하다.

➔ 파이썬의 경우 무한 자릿수 정수를 지원한다. 무한 자릿수 정수는 과학 기술 분야 등에서 유용하다.

```
In [1]: 2**1024

Out[1]: 179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622
492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245
938479716304835356329624224137216
```

2) 8진수

```
In [2]: a = 0o177   # 8진수는 0o로 시작한다.
        a

Out[2]: 127
```

2. 기본 자료형

- **숫자**

3) 16진수

```
In [3]: a = 0x10   # 16진수는 0x로 시작한다.
        a
Out[3]: 16

In [4]: b = 0xABC
        b
Out[4]: 2748
```

4) 2진수

➔ 2진수는 0b로 시작하는 1또는 0의 나열이다. 내장 함수 bin(x)를 사용하면 정수를 2진수 문자열로 바꿀 수 있다.

➔ 파이썬에서 음의 2진수는 –를 붙여 표현한다.

2. 기본 자료형

- 숫자

```
In [5]: c = 0b10   # 2진수는 0b로 시작한다.
        c
Out[5]: 2

In [6]: d = 0b100000000
        d
Out[6]: 256

In [7]: bin(2) # 정수 2를 2진수 문자열로 반환
Out[7]: '0b10'

In [8]: e = -0b10  # -2의 2진수 표현
        e
Out[8]: -2

In [9]: bin(0o77)  #  8진수 77의 2진수 문자열 반환
Out[9]: '0b111111'

In [10]: bin(0xff)  # 16진수 ff의 2진수 문자열 반환
Out[10]: '0b11111111'
```

2. 기본 자료형

- **숫자**

5) 실수(float)

➔ 소수점이 포함된 숫자를 말하며 절대값이 $(10^{-308}, 10^{308})$ 범위의 실수를 표현 가능하다.

➔ 내장 함수 float()을 이용하여 실수를 반환할 수 있다.

```
In [13]: a = 1.2
         b = -3.45
         c = 3.45e10
         d = 3.45e-10

         float('12.3')   # 문자열 12.3을 실수로 반환
Out[13]: 12.3

In [14]: float(12)       # 정수 12를 실수로 반환
Out[14]: 12.0
```

➔ 양의 무한대와 음의 무한대는 각각 float('inf')와 float('-inf')로, 숫자가 아닌 값(Not a Number)는 float('NaN')으로 표현된다.

2. 기본 자료형

- **숫자**

6) 복소수(complex)

➔ 허수를 j로 표현하여 정의하거나 complex함수를 이용해 정의할 수 있다.

```
In [20]: x = complex(1,3)
         x
Out[20]: (1+3j)

In [21]: y = 2 + 4j
         y
Out[21]: (2+4j)

In [22]: x + y
Out[22]: (3+7j)

In [23]: x.real      # 복소수 x의 실수부 #
Out[23]: 1.0

In [24]: x.imag      # 복소수 x의 허수부 #
Out[24]: 3.0

In [25]: abs(x)      # 복소수 x의 크기   #
Out[25]: 3.1622776601683795

In [26]: x.conjugate() # 복소수 x의 켤레복소수#
Out[26]: (1-3j)
```

2. 기본 자료형

- **숫자**
  - 숫자의 산술계산에는 다음과 같은 산술연산자를 사용할 수 있다.

In [24]:
```
a = 2.0 + 1.0
b = 2.0 - 1.5
c = 2.0 * 1.5
d = 5.0 / 2.0
e = 5 / 2
f = 10.0 % 3.0
g = 10.0 // 3.0
h = 5.0 ** 2

print('a=',a,'b=',b,'c=',c,'d=',d,'e=',e,'f=',f,'g=',g,'h=',h)
```

```
a= 3.0 b= 0.5 c= 3.0 d= 2.5 e= 2.5 f= 1.0 g= 3.0 h= 25.0
```

+	더하기
-	빼기
*	곱하기
/	나누기
%	나머지 연산
//	소수점 이하 절삭
**	거듭제곱

2. 기본 자료형

- **문자열(str)**
  - 파이썬은 문자열을 표현하는 4가지 방법을 지원한다.
  - 기본적으로는 작은 따옴표를 양쪽으로 둘러싸는 문자열 표현방법을 사용한다.

`'This'`	작은따옴표로 양쪽 둘러싸기
`"This"`	큰따옴표로 양쪽 둘러싸기
`'''This'''`	작은따옴표 3개를 연속으로 써서 양쪽 둘러싸기
`"""This"""`	큰따옴표 3개를 연속으로 써서 양쪽 둘러싸기

2. 기본 자료형

- **문자열(str)**
  - 문자열의 연산자

➔ 문자열의 연산자에는 +,*,[],[:]이 있다.

+	문자열 결합
*	문자열 곱하기
[]	문자열 색인
[:]	문자열 슬라이싱

1) 문자열 결합(+)

```
In [31]: head = 'Hello'
         tail = ' wrold!'
         head + tail

Out[31]: 'Hello wrold!'
```

2) 문자열 곱하기(*)

```
In [32]: a = 'Python'
         a*3

Out[32]: 'PythonPythonPython'
```

2. 기본 자료형

- **문자열(str)**
  - 문자열의 연산자

3) 문자열 색인([])

➔ 문자열의 색인(index)이란 문자열을 구성하는 각 문자의 위치를 의미한다. Python의 색인은 0부터 시작된다.

S	c	i	e	n	c	e		c	o	m	p	u	t	i	n	g		w	i	t	h		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

```
In [35]: a = 'Science computing with Python'
         a[2]
Out[35]: 'i'

In [36]: a[-1]      # 뒤에서 첫번째
Out[36]: 'n'
```

4) 문자열 슬라이싱([:])

➔ 문자열의 부분 집합을 구하는 데 사용된다.

```
In [38]: a[0:7]    # 0~6까지
Out[38]: 'Science'

In [39]: len(a)    # a의 길이를 구할 수 있다.
Out[39]: 29
```

2. 기본 자료형

- **문자열(str)**
  - 파이썬 문자열은 생성된 후에 내용을 수정할 수 없는 수정 불가 객체이다. 변경하려고 하면 오류가 발생한다.

```
In [40]: a = 'pithon'
         a[1] = 'y'

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         ~₩AppData₩Local₩Temp₩ipykernel_25072₩3754840735.py in <module>
               1 a = 'pithon'
         ----> 2 a[1] = 'y'

         TypeError: 'str' object does not support item assignment


In [41]: a = a[:1] + 'y' + a[2:]    # 다음과 같은 방법으로 수정 가능
         a
Out[41]: 'python'
```

2. 복합 자료형

- **리스트(list)**
  - 기본 자료형을 여러 개 나열하여 관리할 수 있는 형태의 자료형
    - 꺾인 괄호([])를 이용하여 표현하며 각 원소는 쉼표(,)로 구분한다.
    - 리스트는 다른 리스트를 원소로 포함할 수 있다.
    - 각 원소는 색인(index)으로 가리킬 수 있으며 문자열과 마찬가지로 0부터 시작한다.

```
In [5]: a = [1,2,3,4,5]                          # 정수로만 이루어진 리스트          #
        b = [3.14, 'Pi', True, [1,2,3]]          # 다양한 자료형을 원소로 갖는 리스트 #
        c = []                                   # 빈 리스트                        #
        print(a[0], a[1], a[2], a[3], a[4])      # 리스트a의 원소를 출력            #
        print(a[-1], a[-2], a[-3], a[-4], a[-5]) # 음수를 이용해 뒤에서부터 지정가능  #

        1 2 3 4 5
        5 4 3 2 1
```

    - 다음은 리스트에서 자주 쓰이는 명령어이다.
    - 리스트는 생성 후 내용의 수정이 가능한 수정 가능 객체이다.

```
In [44]: a+b                    # 두 리스트를 하나로 연결          #
         a*5                    # 리스트를 반복하여 연결           #
         len(a)                 # 리스트의 크기                    #
         a.append(99)           # 리스트의 끝에 한 원소 추가       #
         a.extend([10,11,13])   # 리스트 끝에 여러 원소 추가       #
         a.insert(3,100)        # 리스트의 특정 위치에 한 원소 삽입 #
         a.index(4)             # 특정 원소의 인덱스 출력          #
```

2. 복합 자료형

- **리스트(list)**
  - 다음과 같이 쉽게 리스트를 생성할 수도 있다.

```
In [36]: a = range(10)          # 0부터 1씩 증가하는 원소의 갯수가 10인 리스트 생성 #
         a = range(-6,3)        # -6부터 2까지 1씩 증가하는 리스트 생성            #
         a = range(-6,3,2)      # -6부터 2까지 2씩 증가하는 리스트 생성            #
         a = range(3,-6,-2)     # 3부터 -6까지 2씩 감소하는 리스트 생성            #
```

  - 리스트의 경우 원소간 사칙연산이 불가능하고 읽고 쓰는데 시간이 오래 걸려 **과학계산에는 적합하지 않다**.
  - 과학계산에는 주로 **배열**을 사용한다.
- **튜플(tuple)**
  - 튜플은 괄호()를 이용하여 표현한다.
  - 리스트와 동일한 기능을 하지만, 나열된 원소를 수정할 수 없는 **불변 객체**이다.

```
t = ()                  # 빈 튜플                          #
t = (4,'숫자',[1,2,3])  # 다양한 원소로 이루어진 튜플 #
t = (1,)                # 원소가 1개인 튜플은 원소 뒤에 쉼표를 사용해 수식과 구분한다. #
```

```
t[0] = 1                # 튜플은 원소 수정 불가능    #
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~₩AppData₩Local₩Temp₩ipykernel_25132₩1253691622.py in <module>
----> 1 t[0] = 1

TypeError: 'tuple' object does not support item assignment
```

2. 복합 자료형

- **집합(set)**
  - 집합은 다음과 같은 특징을 가진 자료형이다.
    - ➔ 중복을 허용하지 않는다.
    - ➔ 순서가 중요하지 않다.

```
In [1]: s1 = {1,2,3}
        s1

Out[1]: {1, 2, 3}

In [2]: s2 = {1,2,2,3}        # 동일한 원소를 넣어도 동일한 집합이다.
        s2

Out[2]: {1, 2, 3}

In [3]: {1,2,3} == {3,2,1}    # 원소의 순서를 바꾸어도 동일한 집합이다.

Out[3]: True
```

2. 복합 자료형

- **집합(set)**
  - 집합자료형은 교집합, 합집합, 차집합 연산을 지원한다.

  - 교집합연산

```
In [4]: s1 = {1,2,3,4,5,6}
        s2 = {4,5,6,7,8,9}
        s1 & s2
Out[4]: {4, 5, 6}

In [6]: s1.intersection(s2)
Out[6]: {4, 5, 6}
```

  - 합집합연산

```
In [12]: s1 = {1, 2, 3, 4, 5, 6, 7, 8, 9}
         s2 = {4, 5, 6, 7, 8, 9}

         s1 | s2
Out[12]: {1, 2, 3, 4, 5, 6, 7, 8, 9}

In [13]: s1.union(s2)
Out[13]: {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

2. 복합 자료형

- **집합(set)**
  - 차집합연산

```
In [14]: s2 - s1
Out[14]: set()

In [15]: s1.difference(s2)
Out[15]: {1, 2, 3}
```

  - 원소의 추가와 삭제

```
In [16]: s1 = {1,2,3,4}
         s1.add(10)
         s1
Out[16]: {1, 2, 3, 4, 10}

In [17]: s1.remove(1)
         s1
Out[17]: {2, 3, 4, 10}
```

2. 복합 자료형

- **딕셔너리(dict)**
  - 키(Key)와 값(value)로 이루어진 자료형으로 중괄호({})를 이용해 표현한다.
  - 딕셔너리의 원소들은 서로 다른 키를 가져야 한다.
  - 딕셔너리의 키는 수정불가의 객체를 사용해야한다.
    ➔ 튜플은 키로 사용가능하지만 리스트는 사용할 수 없다.
  - 색인(index)로 접근하지 않고 키를 이용해 접근한다.

```
In [17]: # key (Newton, Einstein, Neumann)에 각각 값을 넣은 딕셔너리 생성 #
         dictionary = {'Newton': 1234, 'Einstein': 5678, 'Neumann':1011}
         dictionary = dict(Newton=1234, Einstein=5678, Neumann=1011)

         dictionary['Einstein']          # 딕셔너리에에서 키를 통해 저장된 값에 접근 #
         dictionary['Hubble'] = 1889     # 딕셔너리에서 새로운 키와 값을 저장       #
```

  - 딕셔너리는 원소의 수정이 가능한 수정 가능 객체이다.

```
In [21]: dic = {'name' : 'A', 'age' : 20}
         dic['country'] = 'Korea'      # 원소 추가
         dic
Out[21]: {'name': 'A', 'age': 20, 'country': 'Korea'}

In [22]: del dic['age']    # 원소 삭제
         dic
Out[22]: {'name': 'A', 'country': 'Korea'}
```

3. 참과 거짓

- **Python 키워드 True와 False는 각각 참과 거짓을 표현한다.**
- **이외에 다음에 해당하는 것들을 거짓으로 취급한다.**

  ➔ False, None, 숫자 0에 해당하는 것들, '', (), {}, set(), range(0)

- **논리 연산자**
  - 조건이 참인지 거짓인지 판단해주는 연산자이다.
- **비교 연산자**
  - 두 변수가 같은지 다른지 논리적으로 비교해주는 연산자이다.

논리연산자	
조건 1 and 조건 2	조건 1,2 모두 참일 때 참이 된다.
조건 1 or 조건 2	조건 1 또는 조건 2 중 하나만 참이어도 참이 된다.
not 조건	조건이 참일 때 결과는 거짓, 조건이 거짓일 때 결과는 참이 된다.

비교연산자	
a>b	a는 b보다 크다.
a<b	a는 b보다 작다.
a==b	a는 b와 같다.
a!=b	a는 b와 같지 않다.
a<=b	a는 b보다 작나 같다.
a>=b	a는 b보다 크거나 같다.

3. 참과 거짓

- 논리연산자 예시

```
In [39]: print('== 연산자: ', 1==1)          # True
         print('!= 연산자: ', 2!=1)          # True
         print('> 연산자 : ', 1>2)           # False
         print('< 연산자: ', 1<2)           # True
         print('>= 연산자: ', 10>=9)        # True
         print('<= 연산자 : ', 10<=10)      # True

         == 연산자:  True
         != 연산자:  True
         > 연산자 :  False
         < 연산자:  True
         >= 연산자:  True
         <= 연산자 :  True
```

3. 참과 거짓

- 비교연산자 예시

In [38]:
```
# 논리 연산자
print('AND 연산자')      # 둘 다 True 여야 함...
print(True and True)   # True
print(True and False)  # False

print('₩n', 'OR 연산자', sep='') # 둘 중 하나만 True여도 True
print(True or False)           # True
print(True or True)            # True
print(False or False)          # False

print('₩n', 'NOT 연산자', sep='') # 반대로 뒤집기
print(not True)                  # False
print(not 1)                     # False
```

```
AND 연산자
True
False

OR
True
True
False

NOT
False
False
```

4. 파이썬 기본 문법

- **식별자(identifier)**
  - 프로그램에서 사용되는 모든 종류의 이름을 의미한다.
  - 변수, 함수, 클래스, 모듈, 객체의 이름 등이 모두 해당한다.
  - 파이썬에서 식별자는 다음과 같은 형태를 가져야한다.
    - ➔ 식별자는 A~Z, a~z, 또는 _로 시작해야 한다. Python 3에서는 한글 글자로 식별자를 시작할 수도 있다.
    - ➔ 이어서 A~Z, a~z, _, 0~9 가 0개 이상 나타날 수 있다. Python 3에서는 이어서 한글 글자들이 0개 이상 나타날 수도 있다.
    - ➔ Python 식별자는 영문 알파벳 대소문자를 구분한다. 예를 들어 식별자 xpos, Xpos, 그리고 xPos는 모두 다른 식별자이다.
  - 키워드는 식별자로 사용할 수 없다.
- **키워드**
  - 미리 의미와 사용법이 정해져있어 다른 의미나 용도로 사용할 수 없는 단어이다.
  - Python 3.7의 경우 다음과 같은 35개의 키워드가 존재한다.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

4. 파이썬 기본 문법

- **할당문**
  - 파이썬은 변수에 처음으로 값이 할당 될 때에 그 변수가 생성된다는 특징이 있다.
  - 할당 연산자 '='을 이용하여 변수에 값을 할당한다.

```
In [1]: x
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_21520\32546335.py in <module>
----> 1 x

NameError: name 'x' is not defined

In [2]: x = 100

In [3]: x
Out[3]: 100

In [4]: x = 200
```

➔ [1]에서는 변수 x가 아직 생성되지 않았으므로 변수 x를 참조하면 오류가 난다.

➔ [2]에서는 변수 x에 처음으로 값이 할당되므로 변수 x가 생성되고 그 값은 100이 된다.

➔ [3]에서는 이미 생성된 변수 x가 참조되므로 100이 출력된다.

➔ [4]에서는 이미 생성되어 있는 변수 x에 값 200이 할당된다.

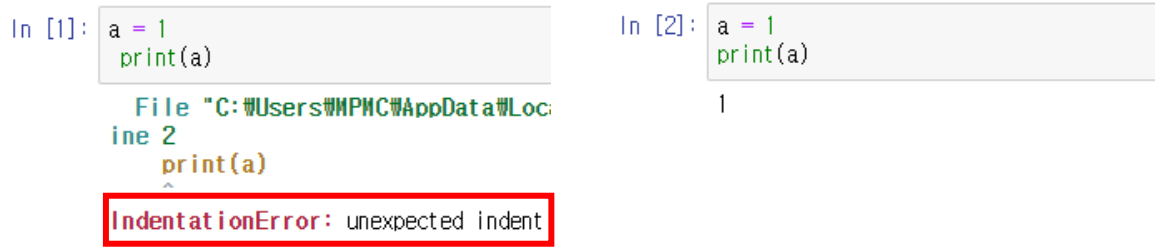
  - 파이썬은 다중 할당문을 지원한다.

```
In [5]: a = 1
        b = 1
        c = 1
        #위와 같은 표현의 다중 할당문
        a = b = c = 1
```

```
In [6]: x = 'a'
        y = 'b'
        #위와 같은 표현의 다중 할당문
        x, y = 'a', 'b'
```

4. 파이썬 기본 문법

- **블록과 들여쓰기(Indentation)**
  - 파이썬은 코딩블록을 표시하기 위해 들여쓰기를 사용한다.
  - {}를 사용하는 다른 C, C#, Java등과 다른 특징을 가진다.
  - 동일한 블록에서 공백 수를 다르게 하면 에러가 발생한다.



- **줄과 문장**
  - 파이썬에서 하나의 문장은 하나의 줄에 나타내는 것이 원칙이다.
  - 하지만 \를 이용하여 하나의 문장을 여러줄로 나누어 표현할 수도 있다.
- **주석**
  - 주석은 실제 프로그램에는 영향을 주지 않고 코드에 대한 설명을 할 때 사용하며, (#, """, ''')기호를 사용한다.

4. 파이썬 기본 문법

- **파이썬 코딩 스타일**
  - 들여쓰기로 4개의 스페이스를 사용하고 탭(tab)을 사용하지 말라. 왜냐하면 개발 환경이나 편집기 마다 탭의 설정이 달라서 혼란을 일으킬 수 있다.
  - 한 줄은 79자를 넘지 않도록 하라. 이 조건은 작은 디스플레이 화면을 사용하는 사용자를 돕고, 큰 화면을 사용하는 경우에는 여러 개의 코드들을 나란히 띄워놓고 볼 수 있도록 한다.
  - 함수, 클래스, 함수 내의 큰 코드 블록 사이에 빈 줄을 넣어 분리하라.
  - 가능하다면 주석은 별도의 줄로 넣어라.
  - 주석 문자열을 사용하라.
  - 연산자들 주변과 콤마 뒤에 스페이스를 넣고, 괄호 바로 안쪽에는 스페이스를 넣지 말라. 예를 들어, a = f(1, 1) + g(3, 4) 와 같이 한다.
  - 클래스들과 함수들에게 일관성 있는 이름을 붙여라. 인스턴스 메소드의 첫 번째 인자의 이름은 항상 self로 하라.

5. 제어문

- **If 조건문**
  - 조건에 따라 다른 계산을 수행할 때 사용하는 문장으로 if문의 구조는 다음과 같다.
    1) If 뒤의 조건이 참일 때 (:) 뒤의 문장을 실행한다.
    2) 앞선 조건이 거짓이고 elif 뒤의 조건이 참일 때 콜론(:) 뒤의 문장을 실행한다.
    3) 앞선 조건이 모두 거짓일 때 else 뒤의 문장을 실행한다.

```
if 조건문 :
    실행할 문장 1
    실행할 문장 2
    ...
    실행할 문장 n
elif 조건문 :
    실행할 문장 1
    실행할 문장 2
    ...
    실행할 문장 n
else:
    실행할 문장 1
    실행할 문장 2
    ...
    실행할 문장 n
```

  - Ex)

```
if x<10:
    print("한자리수")
elif x<100:
    print("두자리수")
elif x<1000:
    print("세자리수")
else:
    print("네자리수 이상")

print("들여쓰기를 통해 조건문의 블럭을 구분")
```

5. 제어문

- **반복문**
  - 반복계산을 하기 위한 문장으로 for와 while을 사용한다.
- **For문**
  - For 문은 리스트, 튜플, 집합, 딕셔너리, 문자열 등과 같은 반복 가능 객체의 원소들을 대상으로 하는 반복처리에 사용된다.
  - For 문의 구조는 다음과 같다.

    ➔ For다음의 변수가 리스트의 첫 값부터 끝 값까지 변하며 콜론(☺ 다음부터 들여쓰기가 끝나는 곳까지 반복 수행한다.

```
for 변수 in 리스트(또는 배열,문자열,튜플...):
    실행할 문장 1
    실행할 문장 2
    ...
```

  - Ex)

```
In [7]: score_list = [77, 87, 65, 98]

        for x in score_list:
            print(x)

        77
        87
        65
        98
```

5. 제어문

- **For문**
  - for 문에서 많은 원소들을 갖는 반복 가능 객체를 직접 표현하는 것은 코드의 작성이 번거롭고, 주기억 장치 공간을 소모할 수 있는 방법이다.

```
In [8]: sum = 0
        for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, ₩
                  17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, ₩
                  31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, ₩
                  45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, ₩
                  59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, ₩
                  73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, ₩
                  87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]:
            sum = sum + i
        print(sum)

        5050
```

  - Python 표준 함수 range()를 이용하면 for 문에서 원소의 수가 많은 리스트를 편리하고 효율적으로 사용할 수 있다.

```
In [9]: sum = 0
        for i in range(1, 101):
            sum = sum + i
        print(sum)

        5050
```

5. 제어문

- **While문**
  - 일정 조건을 만족시키는 경우에 반복 계산을 수행할 때 사용
  - While문의 구조는 다음과 같다.

```
while 조건문:
    실행할 문장 1
    실행할 문장 2
    ...
```

  - 조건이 참이면 블록 안(콜론(:)부터 들여쓰기가 종료되는 부분)의 문장을 실행하고 조건을 다시 판단한다.
  - Ex)

```
n = 0       # while문의 조건을 판단하기 전에 선언해주어야 오류가 생기지 않는다. #
x = 0
while n<11:
    x = x + n
    n = n + 1
print(x)
```

```
55
```

5. 제어문

- **Break 문**
  - For문 또는 while문을 실행하다가 중간에 계산을 중지할 때 사용한다.

  - Ex)

```
n = 0
while True:  # 무한루프 #
    n += 1
    if n==5:
        break # 반복문에서 벗어난다.
    print(n)
```

```
1
2
3
4
```

```
for x in range(10):
    if x==5:
        break
    print(x)
```

```
0
1
2
3
4
```

6. 함수

- **함수란?**
  - 특별한 기능을 수행하기 위해 독립적으로 설계된 코드블록을 함수라고 한다.
  - 프로그램을 함수로 나누어 작성하면 가독성을 높일 수 있고 유지보수가 쉬워진다.
  - 함수에는 내장함수와 사용자정의함수, 람다함수가 있다.
- **내장함수**
  - 파이썬 내부에 이미 만들어져 있는 함수
  - Print(), type(), len() 등으로 적절히 호출하여 사용할 수 있다.
  - Ex)

```
print('값을 출력해주는 기본 내장함수')
type(123)  # 변수의 자료형을 출력해주는 내장함수 #
```

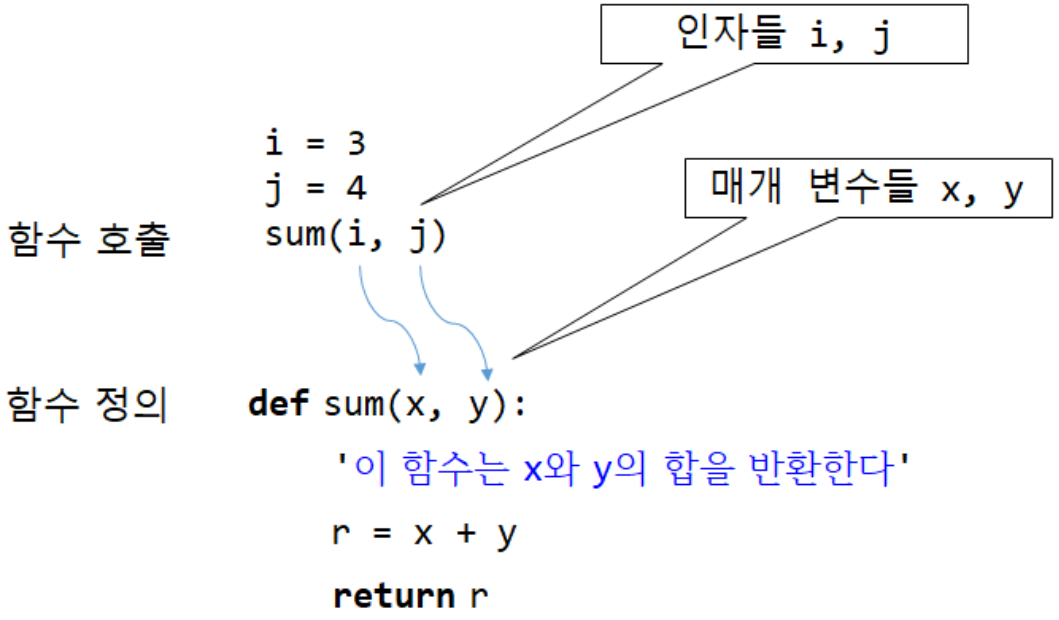
```
값을 출력해주는 기본 내장함수

int
```

- **사용자정의함수**
  - 사용자가 프로그램에서 반복되는 부분을 직접 함수로 정의한 것
  - 함수는 def를 이용하여 다음과 같은 구조로 정의한다.

```
def 함수이름(매개 변수):
    함수 내용
    return 반환값
```

6. 함수

- **사용자정의함수**
  - 함수가 정의되고 나면 함수를 호출하여 사용할 수 있다. 함수 호출은 다음과 같은 형태를 갖는다.
    ➔ 함수 이름(인자)
- **함수의 매개변수와 인자**
  - 함수 정의에서 () 안에 정의된 변수들을 매개 변수(parameter)라고 한다. 그리고 함수 호출에서 () 안에 사용된 값 또는 값을 담은 수식을 인자(argument)라고 한다.



6. 함수

- **함수의 매개변수와 인자**
  - 매개 변수는 기본 값을 가질 수 있다. 다음 calc() 함수 정의는 4개의 매개 변수들을 가지며 그들 중 c와 d는 기본 값을 갖는 매개 변수이다.

```
# 4개의 매개 변수를 갖는 함수. c와 d는 기본 값을 갖는 매개 변수
def calc(a, b, c=0, d=0):
    return (a*b + c - d)
```

  - 기본 값을 갖는 매개 변수(default parameter) 뒤에는 기본 값을 갖지 않은 매개 변수(non-default parameter)가 올 수 없다.

  - 위치 인자와 키워드 인자
    - ➔ 위치 인자 : 위치인자는 대부분의 프로그래밍 언어에서 사용되는 기본적인 방식이다.
      함수 호출 시 위치 인자들과 매개 변수들의 대응은 그들의 위치에 의해 정해진다.

```
a = 3
b = 4
c = 5
sum3(a, b, c) # 위치 인자들 사용

                인자들의 값이 매개 변수들에게 위치에 따라 차례대로 전달된다.
def sum3(x, y, z):
    '이 함수는 x, y, z의 합을 반환한다'
    r = x + y + z
    return r
```

6. 함수

- **함수의 매개변수와 인자**
  - 위치 인자와 키워드 인자
    - ➔ 키워드 인자: 위치 인자는 값 또는 수식의 형태를 갖지만 키워드 인자는 매개 변수에 대한 할당문의 형태를 갖는다. 키워드 인자가 사용되는 경우, 매개 변수의 이름에 대응하여 인자들의 값이 전해진다.
    - ➔ 키워드 인자를 사용하는 경우 매개 변수의 이름들이 사용되므로 함수 호출에 사용되는 키워드 인자들의 순서와 매개 변수들의 순서가 달라도 된다.
    - ➔ 함수를 호출할 때 위치 인자와 키워드 인자를 함께 사용할 수도 있다. 다만 키워드 인자 다음에는 위치 인자가 나올 수 없다.

```
a = 3
b = 4
c = 5
sum3(x=a, z=b, y=c) # 키워드 인자들 사용
```

매개 변수 이름에 대응하여 인자들이 전달된다.

```
def sum3(x, y, z):
    '이 함수는 x, y, z의 합을 반환한다'
    r = x + y + z
    return r
```

```
def sum(x, y):
    '이 함수는 x와 y의 합을 반환한다.'
    r = x + y
    return r

r1 = sum(x=33, y=44)  # 키워드 인자들만 사용
r2 = sum(33, y=44)    # 위치 인자와 키워드 인자 함께 사용
sum(33, 44)           # 위치 인자들만 사용
r4 = sum(y=44, x=33) # 키워드 인자들의 위치는 바뀔 수 있음
```

6. 함수

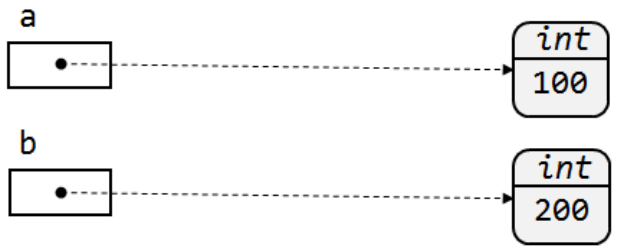
- **함수의 매개변수와 인자**
  - 가변길이 인자
  - Python에서는 매개 변수 앞에 * 또는 **를 붙여 인자들의 개수가 가변인 함수를 정의할 수 있다. 가변 길이 위치 인자들에 대응하는 매개 변수의 이름 앞에는 *가 붙는다.

```
def multiply(*args):
    # 입력인자들의 곱을 구하는 함수 #
    # 호출할 때마다 인자의 수가 변하는 경우 *args를 이용한다. #
    mul = 1
    for num in args:

        mul=mul*num
    return mul

test1 = multiply(3,4)
test2 = multiply(3,4,5)
test3 = multiply(3,4,5,6)
print(test1)
print(test2)
print(test3)
```

```
12
60
360
```

6. 함수

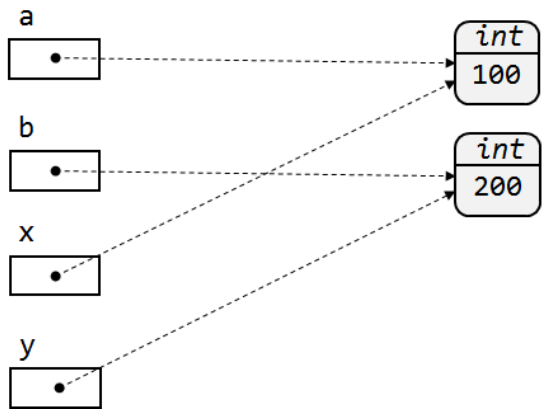
- **함수의 매개변수와 인자**
  - 함수의 인자 전달 방식
    - ➔ Python의 함수 인자 전달 방식은 값에 의한 전달(pass by value)이다.
    - ➔ Ex)
      - ✓ 문장 6과 7이 수행되면 변수 a와 b는 각각 100과 200이 된다.




```
In [12]: def swap(x, y):
             temp = x
             x = y
             y = temp

         a = 100
         b = 200
         print(a, b)
         swap(a, b)
         print(a, b)
```

```
100 200
100 200
```

      - ✓ 문장 8이 호출되어 문장 1이 수행되고 나면 매개 변수 x와 y의 값은 각각 100과 200이 된다.

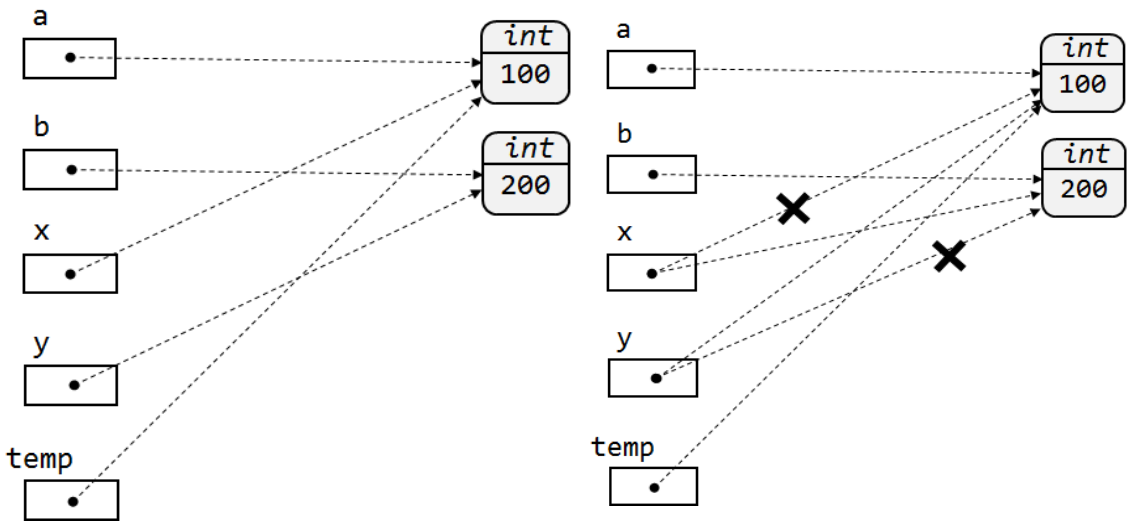


6. 함수

- **함수의 매개변수와 인자**
  - 함수의 인자 전달 방식

➔ Ex)

✓ 문장 2가 수행되고 나면 다음 좌측 그림과 같은 상태가 된다.




```
In [12]: def swap(x, y):
             temp = x
             x = y
             y = temp

         a = 100
         b = 200
         print(a, b)
         swap(a, b)
         print(a, b)
```

```
100 200
100 200
```

✓ 문장 3, 4가 수행되고 나면 변수들의 값은 위 우측 그림과 같은 상태가 된다.

✓ 따라서 10번 문장을 수행하면 결과로 100, 200이 출력된다.

6. 함수

- **Return문**
  - return은 Python 키워드이다. 함수 몸체에는 0개 이상의 return 문이 나타날 수도 있다. return 문의 형태는 다음 중 하나이다.
  - return <expression>: expression을 평가하여 그 결과 값을 반환하고 함수의 수행을 종료한다.
  - return: 함수의 수행을 종료하고 None을 반환한다. return 문이 없는 함수는 return None 문장이 함수 몸체의 마지막에 있는 것으로 취급된다.

```
In [16]: def swap(x, y):
             temp = x
             x = y
             y = temp
             return x, y

         a = 100
         b = 200
         print(a, b)
         a, b = swap(a, b)
         print(a, b)
```

```
100 200
200 100
```

6. 함수

- **람다함수**
  - 한 줄로 사용하는 매우 간단한 함수로, 이름이 없어서 익명함수라고도 불린다.
  - 람다함수의 구조는 다음과 같다.

```
lambda 인자1, 인자2, 인자3, ... : 계산식
```

  - Ex)

```
f = lambda x : (x+2)**2
print(f(1))
```

```
9
```

```
minimum = lambda x,y : x if x<y else y
print(minimum(5,3))
print(minimum(13,15))
```

```
3
13
```

Q&A

Thanks for listening