

단원 02

자료형과 변수

인공지능소프트웨어학과

강환수 교수



Section 1. 파이썬 자료형과 리터럴

-



파이썬 주요 자료형

저장되는 자료 종류의 범주로, 크게 숫자형, 논리, 시퀀스, 매핑, 집합형으로 분류

• 숫자형

- int: 정수형
 - 예를 들어, 20, 16, -7
- float: 실수형
 - 예를 들어, 1.28, -3.67
- complex: 복소수형
 - 예를 들어, 3 + 4j

• 논리형

- bool
 - 참(True)
 - 거짓(False)을 표현

• 시퀀스형:

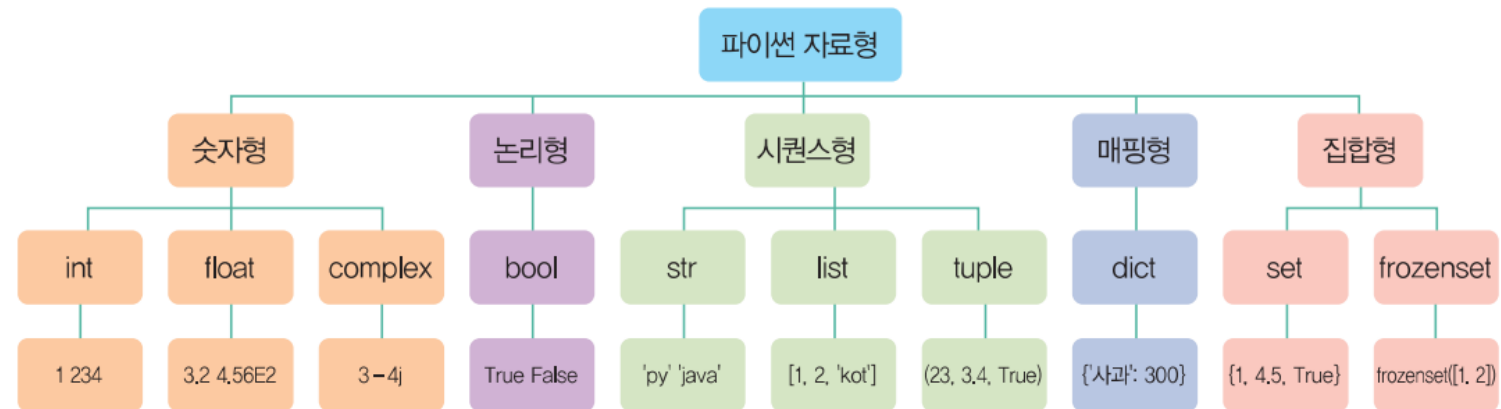
- str
 - 문자열형
- list, tuple
 - 여러 문자의 나열로 구성

• 매핑형

- dict
 - 키-값 쌍으로 구성

• 집합형

- set, frozenset
 - 중복을 허용하지 않는 항목의 모임



숫자형 (Numeric Data Types)

- **int:**
 - 정수
 - 예: 128, -367
- **float:**
 - 소수점을 포함하는 수
 - 예: 128.367, 3.141592
- **complex:**
 - 복소수
 - 예: $3 + 4j$

문자열 (String Data Types)

- 문자열

- str

- 문자열을 나타냄

- 작은따옴표(')나 큰따옴표(")로 감싸진 문자열

- "Python is powerful", 'python'

- 앞뒤가 다른 따옴표를 사용하면 오류가 발생

- 여러 줄 문자열:

- 작은 따옴표나 큰 따옴표를 3개 연속으로 사용하여 표현

내장 함수(built-in functions)

• 함수(functions)

- 함수 이름 뒤 괄호 사이의 인자로 입력 값을 전달해 특정한 기능을 수행하는 일을 처리하는 프로그램 단위



• 파이썬의 내장 함수(built-in functions)

- 특정한 기능을 수행하기 위해 바로 사용할 수 있는 함수
- 내장(built-in)이란 표준 파이썬 설치 후 바로 사용할 수 있다는 의미

• 내장 함수 예

- type(data)
 - 괄호 사이의 자료 data의 자료형을 알려 줌
- print(exp1, exp2, exp3, ...)
 - 하나 exp1 또는 여러 값 exp2, exp3 등을 하나의 공백문자로 구분하여 콘솔에 출력

내장 함수 호출(사용)

• 콘솔 출력 함수 print() 호출

```
>>> print(20, type(-7))
```

```
20 <class 'int'>
```

```
>>> print(-3.67, type(-3.18))
```

```
-3.67 <class 'float'>
```

```
>>>
```

```
>>> print(0.34e1, 0.34E2)
```

```
3.4 34.0
```

```
>>> print(64.6784e-1, 123.34E-2)
```

```
6.46784 1.2334
```

출력문 print()에서 여러 자료를 콤마로 구분해 나열하면 공백 문자 하나로 구분하여 각각의 값을 한 줄에 순서대로 출력한다.

3.1 + 4.5j와 같은 복소수는 complex형이다.

```
>>> print(3 + 4j, type(5 - 4.3j))
```

```
(3+4j) <class 'complex'>
```

문자열 자료형(string data types) str

• 자료형은 str

- 여러 문자의 나열인 문자열(string)
- 작은 따옴표(single quote)나 큰 따옴표(double quote)로 문자열 앞 뒤에 같은 따옴표를 둘러 싸 표현
- 여러 줄의 문자열
 - 작은 따옴표나 큰 따옴표를 연속해서 3개를 동일하게 앞 뒤에 둘러 싸서 표현
- Indexing
 - 'py'[0]처럼 정수인 첨자(index)를 대괄호([]) 내부에 사용해 각각의 문자를 참조

```
>>> print('Python is powerful.')
```

```
Python is powerful.
```

```
>>> print("파이썬은 쉬워요.")
```

```
파이썬은 쉬워요.
```

```
>>> print('"python"')
```

```
"python"
```

```
>>> print("'java'")
```

```
'java'
```

```
>>> print('파이썬은 강력해요.")
```

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

```
print('파이썬은 강력해요.")
```

```
^
```

```
SyntaxError: unterminated string literal (detected at line 1)
```

```
>>> print("파이썬은 확장성이 좋아요.')
```

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

```
print("파이썬은 확장성이 좋아요.')
```

```
^
```

```
SyntaxError: unterminated string literal (detected at line 1)
```

```
>>> print('py'[0], 'py'[1])
```

```
p y
```

첨자는 순서대로 0부터 시작하므로 'py'[0]은 p, 'py'[1]은 y이다.

논리 자료형(Boolean Data Type) bool

- 불, 불리언 (Boolean Data Type)

- bool:

- 참과 거짓을 나타내는 자료형
 - True 또는 False 두 가지 값

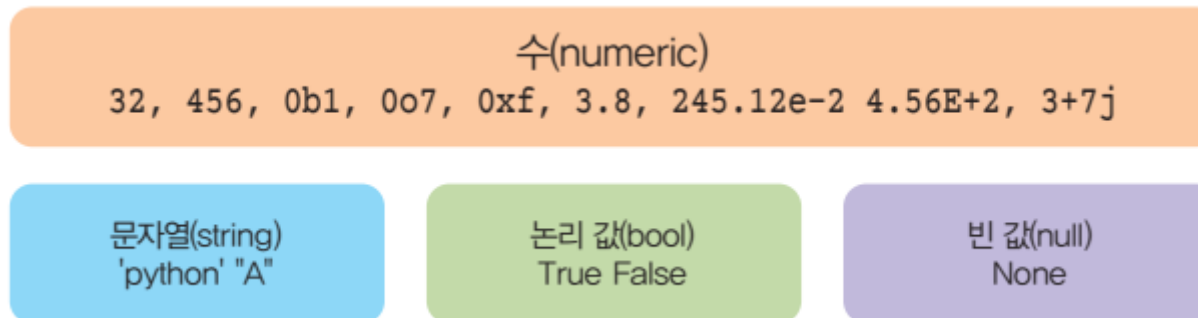
- 비교 연산자 (>, <, ==) 등을 사용하여 불 자료형을 생성

```
>>> print(3 > 4, type(3 < 4))  
False <class 'bool'>
```

리터럴 (Literals)

일상 생활에서 사용되며 코드에 직접 사용할 수 있는 값을 의미

- 숫자 리터럴
 - 32, 456, 0b1, 0o7, 0xf, 3.8, 345.12e-2, 3+7j
- 문자열 리터럴
 - "python", "A"
- 불(bool) 리터럴
 - True, False
- None
 - 값이 없음을 나타내는 리터럴



다양한 진수 표현

- 다음처럼 수 앞에 0b 또는 0B를 선행하면 2진수 리터럴
 - 맨 앞은 숫자 0
 - 0b10, 0B11 # 0b 0B 2진수
- 앞에 0o(소문자 o) 또는 0O(대문자 O)를 선행
 - 8진수
- 0x 또는 0X를 선행
 - 16진수
- 수에서 중간 밑줄(underscore) _은 삽입할 수 있어 수의 자릿수 쉽표처럼 사용 가능
 - 1_000, 1_000_000, 1_00

주석

- 프로그램 실행 도중 인터프리터가 무시하는 부분
 - 코드의 가독성을 높이고 코드를 이해할 수 있도록 도와주는 문장으로 구성
- 한 줄에서 # 이후 그 줄 끝까지는 '주석(comments)'을 의미
 - 프로그래밍 언어의 문법과 상관없이 코드의 이해를 돕는 문장을 삽입 가능
 - # 주석 comments
 - 여러 줄 주석
 - """ 또는 '''로 감싸줌

주석 comments

프로그램 실행 도중 인터프리터가 무시하는 부분으로, 코드의 가독성을 높이고 코드를 이해할 수 있도록 도와주는 문장으로 구성

Section 2. 변수와 모임 자료형

-



변수(Variables)

프로그래밍 언어에서 (실행 중) 프로자료 값을 임시로 저장하는 공간

• 변수의 필요성

- 변수를 사용하여 반복적으로 변하는 값을 저장 가능

```
>>> prog_lang = '파이썬은'
>>> devp_year = 1990
>>> print(prog_lang, devp_year, '년에 개발되었다.')
파이썬은 1990 년에 개발되었다.
>>>
>>> prog_lang = '자바는'
>>> devp_year = 1995
>>> print(prog_lang, devp_year, '년에 개발되었다.')
자바는 1995 년에 개발되었다.
```

• 변수 (Variables)

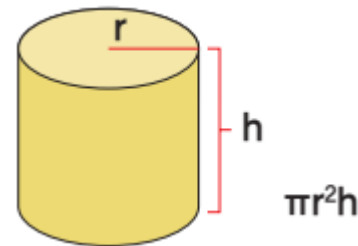
- 변수는 데이터를 저장하는 공간
- 일반적으로 변수를 선언할 때는 변수 이름과 데이터 타입을 지정
 - 동적 타이핑 (dynamic typing) 언어이므로 변수 선언 시 데이터 타입을 명시하지 않아도 됨
- 변수명 = 값 형식으로 할당
 - `name = "홍길동"` # 자료형 지정도 가능(type hint)
 - `age = 20` `name: str = " 홍길동 "`
 - `pi = 3.14`

변수와 대입 연산자

• 변수

- 프로그램 실행 중 변화할 수 있는 자료 값을 저장하는 공간
- 변수의 이름을 의미 있게 부여하면 코드 이해가 쉬워 짐
 - 원통의 체적을 구할 때 원주율(pi) 등을 변수에 저장하여 사용

```
>>> print(5 * 5 * 3.14 * 4.5)
353.25
>>> pi = 3.14
```



- 변수 이름, 저장 값, 자료형으로 구성

• 대입 연산자(assignment operator) =

- 변수명 = 값
 - 오른쪽의 값을 왼쪽의 변수에 저장하는 역할
 - '수학에서 같다' 과는 다르게 작용, 프로그래밍 언어에서는 같다 ==
 - 여러 변수에 한 대입 연산으로 각각의 값을 저장 가능
- 대입 연산 하나로 여러 변수에 각각의 값 저장 가능

```
>>> radius, height = 5, 4.5
>>> print(radius * radius * pi * height)
353.25
```

대입 연산자 하나와 콤마를 사용하면 여러 변수에 각각의 값을 대입할 수 있다.

변수 이름 규칙

- 영어 대문자와 소문자(A-Z, a-z), 한글도 가능
- 숫자 0에서 9와 밑줄 문자 _
- 다만 숫자는 맨 앞에 올 수 없음
- 한 프로그램에서 변수는 유일해야 함
- 이미 정해진 키워드는 사용할 수 없음

```
>>> 2026_worldcup = '미국 캐나다 멕시코'
      File "<stdin>", line 1
        2026_worldcup = '미국 캐나다 멕시코'
            ^
SyntaxError: invalid decimal literal
>>>
>>> worldcup_2026 = '미국 캐나다 멕시코'
>>> print('2026년 월드컵 개최국:', worldcup_2026)
2026년 월드컵 개최국: 미국 캐나다 멕시코
```


키워드

프로그래밍 언어에서 키워드는 특정 용도로 사용하기 위해 미리 정의된 단어, 예약어(reserved words)라고도 부름

- 키워드는 파이썬에서 이미 특정 의미를 갖는 예약어

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

```
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',
'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

내장 함수 help()

- 도움말 함수 help()

- 내장 함수, 모듈 등과 같은 다양한 정보를 출력
- 파이썬 셸에서 도움말 help('keywords') 로 키워드 35개를 확인

```
>>> help('keywords')
```

```
Here is a list of the Python keywords. Enter any keyword to get more help.
```

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

리스트(list)

리스트는 목록이라고도 부를 예정

- 시퀀스형

- 순서가 있는 항목의 나열
 - 리스트(list), 튜플(tuple)
- 항목(items)은 원소(elements)라고도 부름

```
>>> lst = [10, 20, 30]
>>> print(lst, type(lst))
[10, 20, 30] <class 'list'>
```

- 여러 데이터를 순서대로 저장하는 가변형 자료형, 대괄호 []를 사용하여 생성

- 리스트는 변경 가능 (mutable)
- append() 함수를 사용하여 리스트에 새로운 값을 추가

- 인덱스(index)

- 리스트 내부의 데이터에 접근하기 위해 사용

튜플(tuple)

- 리스트와 유사하지만

- 항목을 콤마와 소괄호 ()를 사용하여 생성

```
>>> tpl = (10, 20, 30)
>>> print(tpl, type(tpl))
(10, 20, 30) <class 'tuple'>
```

- 튜플은 변경 불가능

- 튜플은 수정할 수 없는 제한

- 그 처리 속도가 리스트보다 빠름

- 리스트와 비슷한 항목의 나열인 튜플을 따로 제공하는 이유

- 튜플 활용

- 변경되지 않아야 하는 데이터들을 저장할 때 유용

- 요일, 월, 설정값 등

매핑형 딕셔너리(dict)

사전이라고도 부름

- 키(key)와 값(value) 쌍으로 데이터를 저장하는 자료형, 중괄호 {}를 사용하여 생성
 - 각각의 항목에서 키(key)로 값(value)을 지정하므로 매핑형
 - `mart = {'야채': 1200, '음료수': 1500, '과자': 800}`
 - `print(mart, type(mart))`
- 딕셔너리는 변경 가능(mutable)
 - 값을 변경하거나 추가 가능
 - `mart['음료수'] = 1700`
 - `print(mart)`
- 딕셔너리 메소드
 - `keys()`, `values()`를 사용하여 키와 값들을 각각 확인
 - 딕셔너리 값 변경:
 - `mart['과자'] = mart['과자'] + 100`
 - `print(mart)`
- 주의
 - 키는 튜플, 문자열, 수 등과 같이 불변(mutable)이어야 함
 - 리스트와 같은 변경 가능한 객체를 키로 사용 불가능

집합형, 셋(set)과 frozenset

• 셋(set)

- 중복을 허용하지 않으며 순서가 없는 항목의 모임,
- 중괄호 {}로 여러 항목을 쉼표로 구분해 저장

```
>>> basket = {'apple', 'orange', 'apple', 'pear'}
>>> print(basket)
{'pear', 'orange', 'apple'}
```

항목 apple이 두 번 있지만, 중복을 불허하므로 결과적으로 하나만 존재한다.

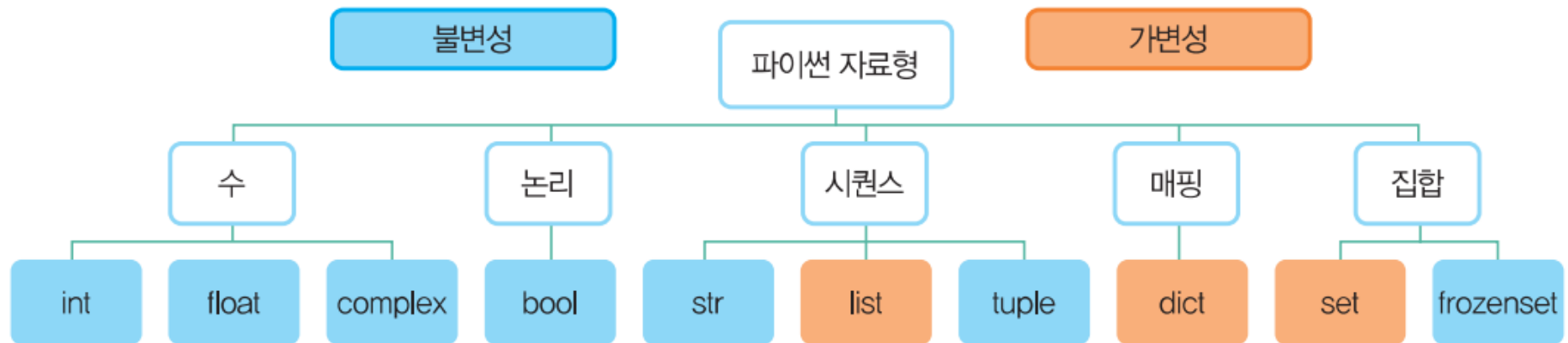
- 변경 가능 (mutable)
 - add() 메소드를 사용하여 요소를 추가할 수 있지만
- 수정 불가능한 항목만 사용 가능
 - 수, str, tuple

• frozenset

- 셋과 유사하지만 변경 불가능(immutable)한 자료형
 - add() 메소드 사용 불가
 - pl = frozenset({'python', 'kotlin'})
 - print(pl, type(pl))

가변성과 불변성

- 가변형(mutable) 객체
 - 값을 변경할 수 있는 객체
 - list, dict, set
- 불변형(immutable) 객체
 - 값을 변경할 수 없는 객체
 - int, float, str, bool, tuple, frozenset.



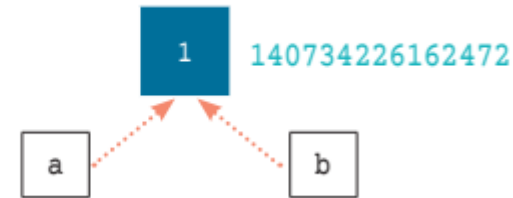
가변성과 불변성 비교

구분	가변성	불변성
용어	mutable	immutable hashable
기본 자료 종류	-	int float complex bool
모임 종류	list dict set bytearray	str tuple frozenset bytes
구분 방식	생성 후 수정 가능	생성 후 수정 불가능
모임 객체	항목을 수정하거나 추가 또는 삭제하는 메소드 제공	항목을 수정하거나 추가 또는 삭제하는 메소드 미제공
참조 속도	느린 참조	빠른 참조
수정 비용	쉽고 비용이 적으며 효율적	수정이 불가능하거나 새로 생성해야 하므로 비용이 많이 소요
활용	자료 수정이 자주 필요하면 활용	자료 수정이 적거나 없으면 활용

내장함수 id(obj)

- 객체 obj의 고유 주소값(address)을 반환하는 함수
- 변수 a, b는 모두 1이 저장
 - 함수 id()로 두 변수 모두 주소가 같다는 것을 확인
 - 객체 1이 저장된 공간을 변수 a, b가 모두 참조

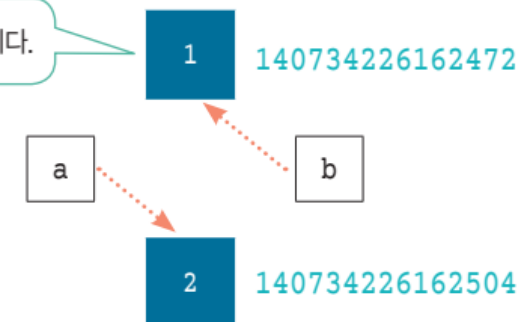
```
>>> a = b = 1
>>> print(id(a), id(b))
140734226162472 140734226162472
>>> print(id(a) == id(b))
True
```



- 변수 a를 1 증가시키면 변수 a와 b의 주소가 달라진 것 확인
 - a의 값을 수정한 것이 아니라 새로운 값 2를 생성해 저장하고 그 주소를 변수 a가 참조
 - 변수 b는 여전히 1이 저장된 주소를 참조

```
>>> a = a + 1
>>> print(id(a), id(b))
140734226162504 140734226162472
>>> print(id(a) == id(b))
False
```

이 자체가 수정될 수 없으므로 불변성이다.



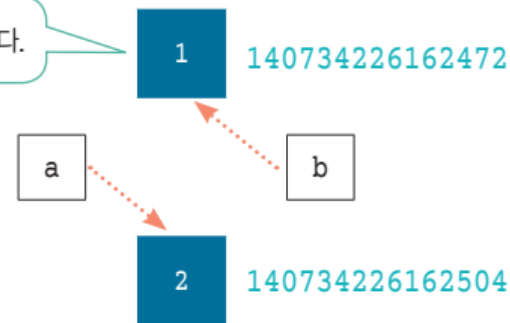
수의 불변성

- 변수 a를 1 증가시키면 변수 a와 b의 주소가 달라진 것 확인

- a의 값을 수정한 것이 아니라 새로운 값 2를 생성해 저장하고 그 주소를 변수 a가 참조
 - 변수 b는 여전히 1이 저장된 주소를 참조

```
>>> a = a + 1
>>> print(id(a), id(b))
140734226162504 140734226162472
>>> print(id(a) == id(b))
False
```

이 자체가 수정될 수 없으므로 불변성이다.



- 직접 정수 2의 id(2)의 결과도 위의 주소 값과 일치

- 즉, 위와 다음 코드는 정수 int 객체는 수정할 수 없는 불변성 객체라는 것을 보여 줌

```
>>> print(id(2), id(1+1), id(3-1))
140734226162504 140734226162504 140734226162504
```

리스트의 가변성

- 리스트 [1, 2, 3]의 id()

```
>>> lst = [1, 2, 3]
>>> print(id(lst))
2075366178240
```

2075366178240



lst

- 리스트 [1, 2, 3]에 항목 4를 추가

- 여전히 id() 결과는 동일

- 리스트는 한번 생성한 객체를 수정할 수 있는 가변성 객체

- 즉, 객체의 id() 값을 변경하지 않고 해당 객체의 값을 변경할 수 있으면 변경 가능한 객체

2075366178240

이 자체가 수정되므로 가변성이다.



lst

Section 3. 표준 입력과 표준 출력

-



자료 입력

프로그램 내부에서 사용자로부터 키보드 입력을 통해 데이터를 얻어야 하는 경우

• input() 함수 사용

- 프로그램 실행을 일시 중지하고 사용자의 입력을 기다림
- 입력 방식
 - 사용자가 입력한 후 Enter 키를 누르면, 입력된 내용이 문자열로 반환
- 입력 안내
 - input("사용자에게 입력을 요청하는 메시지를 여기에 입력") 형태로 안내 메시지를 출력
- 변수 저장
 - 입력된 데이터는 변수에 저장하여 이후에 사용
 - age = input('나이 입력 >> ')
 - 나이 입력 후 Enter키 누르면 이후 코드 진행

```
>>> age = input('나이 입력 >> ')
나이 입력 >> 20
>>> print(age)
20
```

```
>>> a = input('정수 입력 >> ')
정수 입력 >> 10
>>> print(a, type(a))
10 <class 'str'>
>>> print(a + 5) # 오류 발생
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

내장 함수 int(), float()

- input() 함수

- 항상 문자열 형태로 값을 반환하기 때문에
 - 숫자 계산을 위해서는 int() 또는 float() 함수를 사용하여 정수 또는 실수 형태로 변환

- int() 함수

- 문자열을 정수로 변환

- float() 함수

- 문자열을 실수로 변환

내장 함수 int(x)

인자 x가 문자열이면 문자열을 정수로 반환하고, 인자 x가 실수이면 소수 부분이 없는 정수를 반환하며, 변환할 수 없으면 오류 발생

내장 함수 float(x)

인자 x가 문자열이면 문자열을 실수로 반환하고, 인자 x가 정수이면 정수를 실수로 반환하며, 변환할 수 없으면 오류 발생

숫자 입력을 위한 형 변환: int(), float()

- 표준 입력으로 받은 문자열을 변환 함수 int()나 float()를 사용
 - 표준 입력을 바로 정수나 실수로 저장
- 표준 입력 10을 정수 10으로 변수 a에 저장

```
>>> a = int(input('정수 입력 >> '))
정수 입력 >> 10
>>> print(a, type(a))
10 <class 'int'>
>>> print(a + 5)
15
```

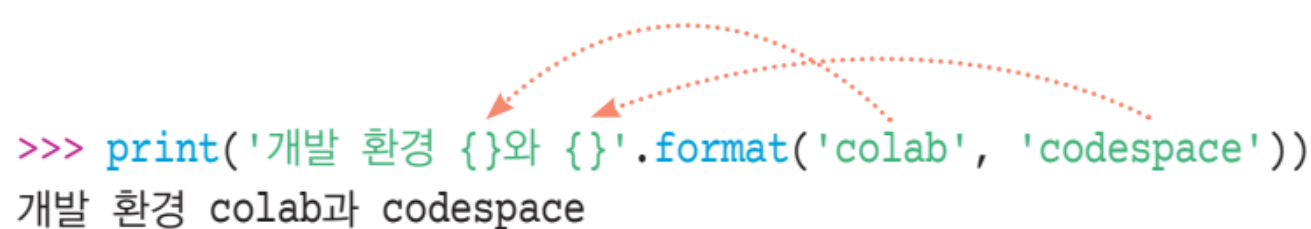
- 표준 입력 3.72를 실수 3.72로 변환해 변수 x에 저장

```
>>> x = float(input('실수 입력 >> '))
실수 입력 >> 3.72
>>> print(x, type(x))
3.72 <class 'float'>
>>> print(a + x)
13.72
```

str.format() 메서드

- 문자열 내에 특정 형식(format)으로 데이터를 삽입하는 데 사용

- 중괄호 {} 를 사용하여 삽입할 위치를 지정하고, .format() 메서드에 삽입할 값(인자)을 전달



```
>>> print('개발 환경 {}와 {}'.format('colab', 'codespace'))
```

개발 환경 colab과 codespace

- 인덱스를 사용한 삽입

- str.format() 메서드에 전달된 값들은 인덱스를 사용하여 순서대로 {} 에 삽입

```
>>> print('개발 환경 {0}와 {1}'.format('colab', 'codespace'))
```

개발 환경 colab과 codespace

```
>>> print('개발 환경 {1}와 {0}'.format('colab', 'codespace'))
```

개발 환경 codespace와 colab

이므로 두 번째 인자인 문자열 'codespace'

- 키워드 인자를 사용한 삽입

- str.format() 메서드에 키워드 인자를 사용하여 특정 {} 에 값을 지정하여 삽입
- print("개발 환경: {ide1} {ide2}".format(ide1="colab", ide2="codespace"))

포맷 문자열 리터럴 f-string

- Python 3.6 버전부터 f-string이라는 새로운 문자열 포맷 방식이 도입

- 변수를 문자열에 직접 삽입할 수 있는 간편하고, 효율적인 방법
- 문자열 앞에 f 접두어를 붙이고, {} 안에 변수를 넣으면 해당 변수의 값이 문자열에 삽입

```
>>> print(f'{3 * 3 * 3.14}')
```

```
28.26
```

```
>>> print(f'{3 ** 4}')
```

```
81
```

```
>>> width, height = 3, 6
```

```
>>> print(f'가로: {width}, 세로: {height}, 사각형 면적: {width*height}')
```

```
가로: 3, 세로: 6, 사각형 면적: 18
```

```
>>> print(f'가로: {width}, 세로: {height}, 삼각형 면적: {width*height / 2}')
```

```
가로: 3, 세로: 6, 삼각형 면적: 9.0
```

문자열 % 포맷 방식

- 문자열 내부에 %문자를 사용하여 변수를 삽입
 - name = "영철"
 - print('안녕, %s!' % name)
 - '안녕, 영철!'을 출력
 - 다양한 데이터 타입
 - 정수는 %d, 실수는 %f를 사용하여 출력
 - 자릿수 지정
 - % 포맷 방식은 세부적인 자릿수를 지정할 수 있어, 예를 들어 %5d는 5자리에서 오른쪽 정렬로 정수를 표현

사전 자료형 출력

- 사전 자료형
 - key: value 쌍으로 데이터를 저장
- `format()` 메서드를 사용하여 사전 자료형을 출력할 때, `{key}` 를 사용하여 해당 값을 출력
 - 키워드 인자 사용: `format(**st)`를 사용하여 사전의 필드를 간단히 삽입

```
>>> st = {'name': '김현수', 'age': 20, 'dept': '인공지능학과'}
```

```
>>> '이름: {name} 나이: {age} 학과: {dept}'.format(**st)  
'이름: 이희소 나이: 19 학과: 빅데이터학과'
```

f-string에서 사전 필드 문자열 사용

- 버전 3.11 이하

- 중괄호 내부 사전의 키에 작은따옴표를 사용하려는 경우, 키가 포함된 f-문자열에 큰따옴표를 사용
- 사전의 키에 큰따옴표를 사용한다면 f-문자열에 작은따옴표를 사용

- 버전 3.12 이상

- 같아도 상관없음

3.11 이하에서는 동일 따옴표 사용 불가능

```
na = {'nation': '대한민국', 'capital': '서울'}
```

```
f'국가: {na["nation"]} 수도: {na["capital"]}'
```

```
f"국가: {na['nation']} 수도: {na['capital']}"
```

3.12 이상에서는 동일 따옴표 사용 가능

```
f'국가: {na['nation']} 수도: {na['capital']}'
```

```
f"국가: {na["nation"]} 수도: {na["capital"]}"
```

f-string에서 사전 필드 문자열 사용

(교재의 내용 수정)

- P85 (수정 전)

- 문자열 f-string을 사용해 사전 필드를 문자열로 구성할 경우에는 따옴표에 주의해야 한다.

- P85 (수정 후)

- 파이썬 버전 3.11 이하에서는 다음처럼 문자열 f-string을 사용해 사전 필드를 문자열로 구성할 경우에는 따옴표에 주의해야 한다. 그러나 파이썬 버전 3.12 이상에서는 동일한 따옴표를 사용해도 문제없다.

상세 형식 지정자

- **format() 메서드를 사용하여 문자열을 다양한 형식으로 출력**

- 자료형, 정렬, 공백, 소수점 자릿수 등을 지정

```
>>> won, ex_rate = 150000, 1228.4
>>> '{} 원은 ${}'.format(won, won/ex_rate)
'150000 원은 $122.110061869098'
```

- {:10f} 전체 자릿수가 10자리로 줄여 표현
- {:.2f} 소수점 이하 두 자리를 표현

```
>>> '{:d} 원은 ${:10f}'.format(won, won/ex_rate)
'150000 원은 $122.110062'
>>> '{:10d} 원은 ${:.2f}'.format(won, won/ex_rate)
'      150000 원은 $122.11'
```

math 모듈

- 모듈의 정의

- 파이썬에서 모듈은 여러 사람들이 사용할 수 있도록 만들어 놓은 파이썬 소스

- math 모듈의 기능

- 수학에서 자주 사용되는 계산 함수와 상수를 정의해 놓은 모듈

- 사용 방법

- import 모듈명

```
>>> import math
```

- 함수와 상수

- 대표적인 함수: `pow()`, `fsum()`, `sqrt()`
- 상수: 원주율 `pi`와 자연수 `e`

- `dir(math)`

- 사용할 수 있는 함수나 상수를 참조

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt', 'ceil', 'comb',
```