



## CHAPTER 3 & 4





# 파이썬 문법 미리 알아두기

## Preview of Fundamental Python Syntax



박진수 교수  
서울대학교·경영대학  
jinsoo@snu.ac.kr



# 학습 목차

☒ 확장자 이름

☒ 인코딩

☒ 변수

☒ 변수 이름

☒ 변수 연산

☒ 들여쓰기

☒ 출력

☒ 입력

☒ 자료형 변환

☒ 주석 달기

# 확장자 이름

- 파이썬 프로그램 파일의 확장자 이름 : **.py**
  - Linux나 macOS 같은 UNIX기반 시스템에서는 파이썬 파일의 확장자가 없을 수도 있다
- 파이썬 GUI 프로그램의 확장자 이름 : **.pyw**

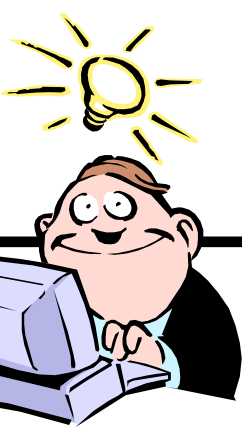


- **UTF-8** 문자 인코딩(encoding)
  - 기본적으로 파이썬 파일은 **UTF-8** 문자 인코딩 방식을 사용
  - UTF-8 은 ASCII 보다 상위개념
  - 지구상에 존재하는 대부분 언어의 문자를 지원
  - 파이썬 2로부터 분리된 가장 큰 이유





# 파이썬을 계산기로 사용



- 파이썬 대화형 모드에서 더하기, 빼기 곱하기 등의 연산을 하면 결과를 바로 알려준다
  - Q1. 1부터 10까지 더하면 얼마인가?
  - Q2. 1부터 10까지 더한 값에 10 이하의 짝수를 모두 더한 값을 빼면 얼마인가?
- 만약 계산된 결과를 반복해서 사용해야 한다면?
  - Q3. 1부터 10까지 더한 값에 10 이하의 홀수를 모두 더한 값을 빼면 얼마인가?
  - Q4. 1부터 10까지 더한 값에 10 이하의 3의 배수를 모두 더한 값을 빼면 얼마인가?



계산을 할 때 띄어쓰기를 하지 않아도 되지만, 가독성을 높이기 위해 일반적으로 숫자와 연산자 사이에 빈칸을 하나 넣어 띄어쓰기를 한다

$1+2-3$

# 띄어쓰기를 하지 않았다.

$1 + 2 - 3$  # 세 칸 띄어쓰기를 했다.

$1 +2- 3$  # 띄어쓰기에 일관성이 없다.

$1 + 2 - 3$  # (권장) 빈칸은 항상 하나만 넣는다.

## ● 변수(variable)란?

- 컴퓨터 메모리 어딘가에 위치해(저장되어) 있는 객체를 참조하기 위해 사용하는 이름
  - 객체(object) : 숫자, 문자, 클래스 등 값을 가지고 있는 모든 것
  - 파이썬은 객체지향 언어이며 숫자와 문자를 포함해 모든 것이 '객체'(object)로 구현되어 있다
- 동일 변수에 다른 객체를 언제든지 할당할 수 있기 때문에, 즉 참조하는 객체가 바뀔 수 있기 때문에 '변수'라고 부른다

## ● 변수 사용

- 일반적으로 **변수이름 = 객체(값)** 형식으로 사용
  - 객체란 숫자나 문자 등 값을 가지고 있는 데이터를 의미

```
x = 1  
y = 1  
z = 1
```

하지만 다음과 같이 약간 다른 형식으로도 사용할 수 있다.

```
x = y = z = 1  
print(x, y, z)
```

```
1 1 1
```



변수를 사용하면 컴퓨터와 프로그래머 모두가 데이터를 쉽게 관리할 수 있다

```
x = 1  
y = 'Hello'  
z = 'Good Bye~~~!'
```

```
print(x)  
print(y)  
print(z)
```

```
1  
Hello  
Good Bye~~~!
```

## 실습

- 내 마음대로 변수의 이름을 지어서
- 다양한 데이터를 저장하는 변수를 선언해보자
- 오류가 발생한다면 조교에게 알리자!



어떤 데이터에는 따옴표가 있고 어떤 데이터에는 따옴표가 없나요?

```
x = 1
y = 'Hello'
z = 'Good Bye~~~!'
```

```
x = 1
y = '1'

print(x)
print(y)
```

```
1
1
```

## 코드 설명

- 변수  $x$  : 따옴표가 없다
- 변수  $y$  : 따옴표가 있다

위의 코드를 보면 두 변수  $x$  와  $y$  는 같은 값을 갖고 있는 것처럼 보인다

하지만 변수의 종류를 알려주는 `type` 함수를 이용하면 두 변수는 다른 종류인 것을 알 수 있다

```
print(type(x))  
print(type(y))
```

```
<class 'int'>  
<class 'str'>
```

## 코드 설명

- 변수 `x` : 숫자형인 `int` (정수)
- 변수 `y` : 문자열인 `str` (문자열)
- 변수에도 **자료형**(data type)이 있다

```
x = hello
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'hello' is not defined
```

## 이유

- 숫자는 따옴표로 묶지 않기로 약속
- 따옴표로 묶지 않은 문자(문장)는 컴퓨터가 **변수의 이름**으로 인식

```
hello = 'Good Bye~~~!'
```

```
print(hello)  
print('hello')
```

```
Good Bye~~~!  
hello
```

## 코드 설명

- `print(hello)` : 변수 *hello*가 담고 있는 'Good Bye'라는 문장을 출력
- `print('hello')` : 'hello'라는 문장을 출력



**변수 이름은 마음대로 아무거나 사용해도 되나요?**

```
한국어 = '영어'
숫자 = 5
```

```
print(한국어)
print(숫자)
```

```
영어
5
```

## 코드 설명

- 변수 **한국어** : '영어'라는 문장을 저장
- 변수 **숫자** : 숫자 5를 저장
- 심지어 한국어로 변수 이름을 사용하는 것도 가능
- 하지만 ...



하지만... 파이썬이 이미 사용하고 있는 이름을 변수 이름으로 사용하면 복잡한 문제가 발생할 수 있다

```
print = 3  
print('Hello, Python!')
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'int' object is not callable
```

## 코드 설명

- 출력 함수 `print` 를 변수 이름으로 사용했기 때문에
- `print` 는 이제 숫자 3을 참조하는 변수다
- 즉, `print` 는 더 이상 출력을 할 수 없게 되어버렸다

따라서... 파이썬이 이미 사용하고 있는 이름은 건드리지 말자!!!

이처럼 변수 이름으로 사용할 수 있는 것들에는 **제약**이 생긴다.



## ● 식별자(identifier)란?

- 파이썬 ‘객체’(데이터)를 식별하기 위해 붙이는 이름(즉, 컴퓨터의 메모리 어딘가에 저장(위치)한 객체(데이터)를 컴퓨터와 프로그래머 모두가 쉽게 관리할 수 있도록 **변수, 함수, 클래스** 등에 붙이는 이름)

## ● 식별자 구성 규칙

- 하나 또는 그 이상의 공백이 없는 문자와 숫자의 조합으로 길이는 제한이 없지만 숫자로 시작할 수는 없음

시작 문자 + 0개 또는 그 이상의 문자

📌 **문자** : 유니코드 문자(Unicode characters) 중 영어 알파벳(a ... z, A ... Z), 밑줄('\_',) , 한글

- 변수 이름이 길거나 두 개 이상의 단어 조합인 경우 단어 사이에 밑줄('\_')을 사용하면 가독성이 높아짐(e.g., *firstname*, *lastname*보다는 *first\_name*, *last\_name*이 좋음)
- 한글도 사용 가능하지만 플랫폼에 따라 인코딩(encoding) 문제가 발생할 수 있기 때문에 가급적이면 영어를 사용하는 것이 좋음

📌 **숫자** : 0, 1,..., 9

- 첫 글자로 숫자가 올 수 없지만, 첫 글자 이 외에는 사용이 가능(예, *x1*(○), *1x* (X))

## ● 대소문자 구분(case sensitive)

📌 '*jinsoopark*', '*Jinsoopark*', '*JinSooPark*', '*JinsooPark*', '*JINSOOPARK*'은 모두 다른 식별자

- 파이썬 예약어(키워드)와 동일한 이름은 사용 불가

- Legal... BUT NOT Recommended

- 내장 자료형(built-in data types) 이름
  - `int`, `float`, `list`, `str`, `tuple` 등
- 내장(built-in) 함수 이름이나 예외(exceptions) 클래스 이름과 같이 파이썬이 지정한 식별자
  - 대문자로 시작하는 이름은 주로 예외 클래스 이름
- 이름의 시작과 끝에 밑줄 2개(e.g., `__package__`)

**NOT RECOMMENDED**

# 퀴즈 : 식별자

---

- tax4
- 4tax
- jinsoo-park
- jinsoo\_park
- park's
- str
- int
- 이름

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



# 연산

숫자 연산  
사칙 연산

5 + 7

12

5 - 7

-2

7 \* 2

14

7 / 2

3.5

문자열 연산  
덧셈과 곱셈만 가능

'안녕' + '파이썬'

'안녕파이썬'

'파이썬' \* 3

'파이썬파이썬파이썬'

# 변수 연산

*i*, *j*, *s*를 '변수'라고 한다

```
i = 5  
j = 7  
s = '파이썬'
```

```
i + j
```

12

```
i = i - j  
i
```

-2

```
j = -5  
i * j
```

10

```
i * j / 4
```

2.5

```
'안녕' + s
```

'안녕파이썬'

```
s = s * 3  
s
```

'파이썬파이썬파이썬'

```
s = 'Python'  
s + ' is fun'
```

'Python is fun'

```
s
```

'Python'


## 한 줄이 넘는 코드 작성하기

```
>>> if 7 > 3:  
...     들여쓰기 print(7 - 3 - 1)  
...     <Enter> 키  
3
```



# 들여쓰기 규칙

## Space sensitive

- 명령문을 구분할 때 중괄호('{', '}') 대신 <들여쓰기(indentation)>를 사용
- 들여쓰기를 할 때는 4칸(space키 4번) 또는 1탭(Tab키 1번)을 입력
-  한 코드 안에서는 반드시 한 종류의 들여쓰기를 사용 -> 혼용하면 안된다
  - 탭(tab)으로 들여쓰면 계속 탭으로 들여써야 한다
  - 원칙으로는 공백(빈칸, space) 사용을 권장(PEP 8 권장 사항)

```
jinsoo — Python — 35x7
[>>> x = 1
[>>> y = 3
      File "<stdin>", line 1
        y = 3
        ^
IndentationError: unexpected indent
>>>
```


```
jinsoo — python — 61x8
>>> if True:
...     print('4 spaces') # space 4칸 사용
...     print('1 tab')    # tab 1개 사용
      File "<stdin>", line 3
        print('1 tab')    # tab 1개 사용
        ^
TabError: inconsistent use of tabs and spaces in indentation
>>>
```

```
jinsoo — python — 61x9
[>>> if True:
[...     print('4 spaces') # space 4칸 사용
[...     print('3 spaces') # space 3칸 사용
      File "<stdin>", line 3
        print('3 spaces') # space 3칸 사용
        ^
IndentationError: unindent does not match any outer indentation level
>>>
```



# 들여쓰기 규칙

## Space sensitive

- 명령문을 구분할 때 중괄호('{', '}') 대신 <들여쓰기(indentation)>를 사용
- 들여쓰기를 할 때는 4칸(space키 4번) 또는 1탭(Tab키 1번)을 입력
-  한 코드 안에서는 반드시 한 종류의 들여쓰기를 사용 -> 혼용하면 안된다
  - 탭(tab)으로 들여쓰면 계속 탭으로 들여써야 한다
  - 원칙으로는 공백(빈칸, space) 사용을 권장(PEP 8 권장 사항)

```
jinsoo — Python — 35x7
[>>> x = 1
[>>> y = 3
      File "<stdin>", line 1
        y = 3
        ^
IndentationError: unexpected indent
>>>
```

```
jinsoo — python — 61x8
>>> if True:
...     print('4 spaces') # space 4칸 사 용
...     print('1 tab')    # tab 1개 사 용
      File "<stdin>", line 3
        print('1 tab')    # tab 1개 사 용
        ^
TabError: inconsistent use of tabs and spaces in indentation
>>>
```

```
jinsoo — python — 61x9
[>>> if True:
[...     print('4 spaces') # space 4칸 사 용
[...     print('3 spaces') # space 3칸 사 용
      File "<stdin>", line 3
        print('3 spaces') # space 3칸 사 용
        ^
IndentationError: unindent does not match any outer indentati
on level
>>>
```

가독성을 높이는 수단이지만 잘못 사용해서 문법적 오류 대신 논리적 오류로 빠지게 되면 디버깅하기가 매우 힘들어 질 수도 있다



출력은 특히 초보 프로그래머들에게 필수인 기능(함수)

파이썬에서는 `print` 함수를 이용해 다양한 것들을 출력할 수 있다

```
print('Hello, Python!')
```

따옴표 안의 내용을 수정해 다양한 문장을 출력해보자



**한글도 출력할 수 있나요?**

# 출력 방식

## ● 대표 형식

- 대화형 모드에서만 가능
- `print` 함수를 사용하지 않고 출력
- 객체의 자료형이 나타나는 형식으로 출력
- 특수 문자(줄바꿈, 탭 등)가 적용되지 않은 상태로 출력

```
'안녕 파이썬!!!'
```

```
'안녕 파이썬!!!'
```

```
11 + 22
```

```
33
```

## ● 텍스트 형식

- 대화형 모드와 인터프리터 모드 둘 다에서 사용 가능
- `print` 함수를 사용하여 출력
- 객체를 화면에서 보기 편한 형식으로 출력
- 특수 문자(줄바꿈, 탭 등)가 적용된 상태로 출력

```
print('안녕 파이썬!!!')
```

```
안녕 파이썬!!!
```

```
print(11 + 22)
```

```
33
```

둘의 차이는



# 예시 : 출력 방식

```
x = 27          # 변수 x에 숫자를 할당한다.
```

```
x              # 숫자를 대표 형식으로 출력한다.
```

```
27
```

```
print(x)        # 숫자를 텍스트 형식으로 출력한다.
```

```
27
```

```
y = '홍길동'    # 변수 y에 문자를 할당한다.
```

```
y              # 문자를 대표 형식으로 출력한다.
```

```
'홍길동'
```

```
print(y)        # 문자를 텍스트 형식으로 출력한다.
```

```
홍길동
```

```
z = '첫째 줄\n둘째 줄'  # 변수 z에 특수문자가 있는 문자를 할당한다.
```

```
z              # 특수문자가 있는 문자를 대표 형식으로 출력한다.
```

```
'첫째 줄\n둘째 줄'
```

```
print(z)        # 특수문자가 있는 문자를 텍스트 형식으로 출력한다.
```

```
첫째 줄  
둘째 줄
```

```
'Python', 1, 2, 3      # 여러 개의 객체를 대표 형식으로 출력한다.
```

```
('Python', 1, 2, 3)
```

```
print('Python', 1, 2, 3)  # 여러 개의 객체를 텍스트 형식으로 출력한다.
```

```
Python 1 2 3
```



## 작성 방법

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- 다수의 위치 전달인자(\**objects*)를 받을 수 있고 4개의 키워드 전달인자를 받을 수 있음
  - 각 키워드 전달인자는 기본값이 있음
- *sep* 매개변수의 기본값은 공백(' ')
  - 두 개 이상의 위치 전달인자가 주어질 경우 각각의 전달인자는 *sep* 값으로 나뉘어 출력됨
  - 위치 전달인자가 하나만 주어졌을 경우 *sep* 매개변수는 아무런 역할도 하지 않음
- *end* 매개변수의 기본값은 새줄바꿈(newline)
  - 위치 전달인자들이 출력되고 난 후 마지막에 *end* 매개변수의 값이 출력됨
- *file* 매개변수의 기본값은 표준 출력 스트림(standard output stream)
  - 표준 출력 스트림은 주로 콘솔(console)을 사용
- *flush* 매개변수는 파이썬 3.3부터 사용 가능
  - 값이 *True*이면 *file* 스트림이 강제적으로 내보내짐(flush)

# print 함수 잘 활용하기

```
print( 'Python' , 3 )
```

```
print( 'Python' , 3 , sep=' ' )
```

```
print( 'Python' , 3 , sep=' version ' )
```

```
print( 'Python is fun' )  
print( 'Python is easy' )
```

```
print( 'Python is fun' , end=' and ' )  
print( 'Python is easy' )
```

# print 함수 잘 활용하기

```
print( 'Python' , 3)
```

Python 3

```
print( 'Python' , 3, sep=' ')
```

```
print( 'Python' , 3, sep=' version ')
```

```
print( 'Python is fun' )  
print( 'Python is easy' )
```

```
print( 'Python is fun' , end=' and ' )  
print( 'Python is easy' )
```



# print 함수 잘 활용하기

```
print( 'Python' , 3 )
```

Python 3

```
print( 'Python' , 3 , sep=' ' )
```

Python3

```
print( 'Python' , 3 , sep=' version ' )
```

```
print( 'Python is fun' )  
print( 'Python is easy' )
```

```
print( 'Python is fun' , end=' and ' )  
print( 'Python is easy' )
```

# print 함수 잘 활용하기

```
print('Python', 3)
```

Python 3

```
print('Python', 3, sep='')
```

Python3

```
print('Python', 3, sep=' version ')
```

Python version 3

```
print('Python is fun')  
print('Python is easy')
```

```
print('Python is fun', end=' and ')  
print('Python is easy')
```

# print 함수 잘 활용하기

```
print('Python', 3)
```

Python 3

```
print('Python', 3, sep='')
```

Python3

```
print('Python', 3, sep=' version ')
```

Python version 3

```
print('Python is fun')  
print('Python is easy')
```

Python is fun  
Python is easy

```
print('Python is fun', end=' and ')  
print('Python is easy')
```

# print 함수 잘 활용하기

```
print('Python', 3)
```

Python 3

```
print('Python', 3, sep='')
```

Python3

```
print('Python', 3, sep=' version ')
```

Python version 3

```
print('Python is fun')  
print('Python is easy')
```

Python is fun  
Python is easy

```
print('Python is fun', end=' and ')  
print('Python is easy')
```

Python is fun and Python is easy

# print 함수와 문자열 결합

```
name = '파이썬'
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('Hi!', name)
```

```
print('Hi! ' + name) # 문자열 결합
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('The length of', name, 'is', len(name))
```

```
print('The length of ' + name + ' is ' + str(len(name))) # 문자열 결합
```

```
age = 19
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('You will be', age + 1, 'years old in the next year.')
```

```
print('당신의 나이는 내년에 ' + str(age + 1) + '살이 됩니다.') # 문자열 결합
```



# print 함수와 문자열 결합

```
name = '파이썬'
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('Hi!', name)
```

```
print('Hi! ' + name) # 문자열 결합
```

Hi! 파이썬

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('The length of', name, 'is', len(name))
```

```
print('The length of ' + name + ' is ' + str(len(name))) # 문자열 결합
```

```
age = 19
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('You will be', age + 1, 'years old in the next year.')
```

```
print('당신의 나이는 내년에 ' + str(age + 1) + '살이 됩니다.') # 문자열 결합
```

# print 함수와 문자열 결합

```
name = '파이썬'
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('Hi!', name)
```

```
print('Hi! ' + name) # 문자열 결합
```

Hi! 파이썬

Hi! 파이썬

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('The length of', name, 'is', len(name))
```

```
print('The length of ' + name + ' is ' + str(len(name))) # 문자열 결합
```

```
age = 19
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('You will be', age + 1, 'years old in the next year.')
```

```
print('당신의 나이는 내년에 ' + str(age + 1) + '살이 됩니다.') # 문자열 결합
```

# print 함수와 문자열 결합

```
name = '파이썬'
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('Hi!', name)
```

```
print('Hi! ' + name) # 문자열 결합
```

Hi! 파이썬

Hi! 파이썬

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('The length of', name, 'is', len(name))
```

```
print('The length of ' + name + ' is ' + str(len(name))) # 문자열 결합
```

The length of 파이썬 is 3

```
age = 19
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('You will be', age + 1, 'years old in the next year.')
```

```
print('당신의 나이는 내년엔 ' + str(age + 1) + '살이 됩니다.') # 문자열 결합
```



# print 함수와 문자열 결합

```
name = '파이썬'
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('Hi!', name)
```

```
print('Hi! ' + name) # 문자열 결합
```

Hi! 파이썬

Hi! 파이썬

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('The length of', name, 'is', len(name))
```

```
print('The length of ' + name + ' is ' + str(len(name))) # 문자열 결합
```

The length of 파이썬 is 3

The length of 파이썬 is 3

```
age = 19
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('You will be', age + 1, 'years old in the next year.')
```

```
print('당신의 나이는 내년에 ' + str(age + 1) + '살이 됩니다.') # 문자열 결합
```

# print 함수와 문자열 결합

```
name = '파이썬'
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('Hi!', name)
```

```
print('Hi! ' + name) # 문자열 결합
```

Hi! 파이썬

Hi! 파이썬

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('The length of', name, 'is', len(name))
```

```
print('The length of ' + name + ' is ' + str(len(name))) # 문자열 결합
```

The length of 파이썬 is 3

The length of 파이썬 is 3

```
age = 19
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('You will be', age + 1, 'years old in the next year.')
```

```
print('당신의 나이는 내년엔 ' + str(age + 1) + '살이 됩니다.') # 문자열 결합
```

You will be 20 years old in the next year.

# print 함수와 문자열 결합

```
name = '파이썬'
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('Hi!', name)
```

```
print('Hi! ' + name) # 문자열 결합
```

Hi! 파이썬

Hi! 파이썬

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('The length of', name, 'is', len(name))
```

```
print('The length of ' + name + ' is ' + str(len(name))) # 문자열 결합
```

The length of 파이썬 is 3

The length of 파이썬 is 3

```
age = 19
```

```
# 아래 코드의 차이가 뭔지 확인하세요.
```

```
print('You will be', age + 1, 'years old in the next year.')
```

```
print('당신의 나이는 내년에 ' + str(age + 1) + '살이 됩니다.') # 문자열 결합
```

You will be 20 years old in the next year.

당신의 나이는 내년에 20살이 됩니다.



파이썬에서 입력을 받는 대표적인 함수로는 `input`이 있다

```
input('무엇이든 입력해 주세요: ')
```

```
무엇이든 입력해 주세요: 5  
'5'
```

## 코드 설명

- `input` 함수 안에 적은 내용이 화면으로 출력되며
- 입력을 기다리는 창이 나오는 것을 알 수 있다

## 실습

- '무엇이든 입력해 주세요:' 대신에 화면에
- '좋아하는 숫자를 입력해 주세요:'라고 표시되게 코드를 수정해보자
- 오류가 발생한다면 조교에게 알리자!





사용자가 값을 입력한 경우

```
input() # no prompt
```

```
파이썬  
'파이썬'
```

```
input('이름을 입력하세요: ') # using prompt
```

```
이름을 입력하세요: 파이썬  
'파이썬'
```

사용자가 값을 입력하지 않고 <Enter> 또는 <return> 키를 눌렀을 경우

```
input('이름을 입력하세요: ') # using prompt
```

```
이름을 입력하세요:  
' '
```



**그런데, 우리가 입력한 값은 어디로 가는 것일까요?**



프로그램에서 입력을 받았다면 그 값을 사용할 수 있어야 한다

```
x = input('좋아하는 숫자를 입력해 주세요: ')\nprint('당신이 좋아하는 숫자는', x)
```

```
좋아하는 숫자를 입력해 주세요: 11\n당신이 좋아하는 숫자는 11
```

좋아하는 숫자  $x$  에다 5를 더해보자

```
x + 5
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: can only concatenate str (not "int") to str
```

어찌된 영문일까?



입력받은 값의 자료형을 확인해보자

```
print(type(x))
```

```
<class 'str'>
```

오류가 난 이유

- ❶  $x$ 의 자료형이 문자열(str)이다
- ❷ 따라서 숫자 5와 문자열은 덧셈 연산을 할 수 없다

# 입력 값의 자료형을 육안으로 확인하기

문자열을 입력한 경우

```
name = input( '이름을 입력하세요: ' ) # using prompt
```

이름을 입력하세요: 홍길동

name

'홍길동'

문자열 형태로 반환한다

숫자를 입력한 경우

```
age = input( '나이를 입력하세요: ' ) # using prompt
```

나이를 입력하세요: 19

age

'19'

# 정리 : 사용자 키보드 입력

## ● `input([prompt])` 함수

- 사용자로부터 값을 입력 받을 수 있는 내장함수
  - 실행하면 즉시 커서가 나타나 입력을 기다림

- 대괄호(" [ ]") 부분에 문자열 값을 넣으면 사용자로부터 입력받을 때 해당 문자열을 출력할 수 있음

- e.g., `input()` => \_

- 아무 것도 출력되지 않은 상태에서 사용자의 입력을 기다림

- e.g., `input('당신의 이름은 무엇입니까?')` => 당신의 이름은 무엇입니까?\_

- 해당 문자열을 출력하고 사용자의 입력을 기다림

- 사용자가 값을 입력을 한 후 <Enter> 또는 <return> 키를 누르면 입력이 종료

## ● 반환 값

- 사용자가 입력한 값은 항상 문자열(str) 형태로 반환

- 사용자가 아무런 값도 입력하지 않고 <Enter> 또는 <return> 키를 눌렀을 경우 빈 문자열을 반환

자료형 변환(casting)이란? 어떤 자료형(data type)을 다른 자료형으로 변환시키는 과정

형변환 전

```
age = input('나이를 입력하세요: ') # using prompt
```

나이를 입력하세요: 19

```
age
```

```
'19'
```

```
age + 1
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: must be str, not int
```

형변환 후

```
int(age) + 1
```

```
20
```

파이썬은 변수를 선언할 때 자료형을 지정할 필요가 없다

```
x = 1  
y = 2  
print(x + y)
```

3

## 코드 설명

- 변수  $x$ 와  $y$ 를 숫자라고 선언해 주지 않았음에도 불구하고, 파이썬은 두 변수가 정수(integer)임을 알아채고  $x + y$ 의 결과로 3을 반환
- 명시적으로 변수의 자료형을 지정하지 않았음에도 불구하고 변수의 자료형이 존재한다는 것을 알 수 있음(viz. 동적 언어)

이번에는 숫자 대신에 문자열을 사용해보자

```
x = '1'
y = '2'
print(x + y)

12
```

## 코드 설명

- 이번에는 변수  $x$  와  $y$  를 선언할 때 숫자를 작은 따옴표(' ')로 둘러싸서 문자열(str) 변수라고 선언해 주자 파이썬은 이를 알아채고 결과로 '12'를 반환



그렇다면...

만약 호환될 수 없는 서로 다른 자료형 변수를 가지고  
연산을 한다면 어떤 결과가 나올까?



문자형 변수와 숫자형 변수의 합 연산 결과는 어떻게 될까?

```
x = 1
y = '2'
print(x + y)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

## 코드 설명

- 위와 같이 정수와 문자열 변수 간에는 + 연산을 할 수 없다면서 **TypeError**가 발생
- 즉, 두 변수의 자료형이 맞지 않아 에러가 난 것으로, 이런 경우에 연산을 수행하고 싶다면 **자료형 변환** (type casting)을 통해 같은 자료형(또는 연산이 가능한 유사 자료형)으로 바꿔 줘야 한다

## 자료형 변환 활용 예시

```
x = 1
y = '2'
print(x + int(y))
```

3

### 코드 설명

- 변수 *y* 는 문자형으로 선언되었지만 *int* 클래스를 통해 정수로 변환했기 때문에 1과 합 연산이 가능

## 자주 사용하는 형변환 클래스

- *str*(객체) : 객체를 문자형으로 변환
- *int*(객체) : 객체를 정수형으로 변환
- *float*(객체) : 객체를 실수형으로 변환

# 잠깐...

이번 실습에서 변수  $x$ 와  $y$ 를 여러 번 사용하였다.

변수  $x$ 에 새로운 값이 할당되면 이전의 값은 사라져버린다.

따라서 변수를 여러 개 사용해야 하는 프로그램을 작성하는 경우 적절한 변수 이름을 생각하는 것도 굉장한 노동이다.

# 상호 형변환이 가능한 자료형

자료형 변환은 데이터 분석시 매우 유용하고 강력한 기능이지만

변수의 자료형이 서로 변환 가능한 형태인 경우에만 자료형 변환을 할 수 있다

```
# 정수(integer)를 실수(float)로, 실수(float)를 into 정수(integer)로 변환
i = 5
print(type(i))

f = float(i)
print(type(f))

j = int(f)
print(type(j))
```

# 상호 형변환이 가능한 자료형

자료형 변환은 데이터 분석시 매우 유용하고 강력한 기능이지만

변수의 자료형이 서로 변환 가능한 형태인 경우에만 자료형 변환을 할 수 있다

```
# 정수(integer)를 실수(float)로, 실수(float)를 into 정수(integer)로 변환
i = 5
print(type(i))

f = float(i)
print(type(f))

j = int(f)
print(type(j))

<class 'int'>
```

# 상호 형변환이 가능한 자료형

자료형 변환은 데이터 분석시 매우 유용하고 강력한 기능이지만

변수의 자료형이 서로 변환 가능한 형태인 경우에만 자료형 변환을 할 수 있다

```
# 정수(integer)를 실수(float)로, 실수(float)를 into 정수(integer)로 변환
i = 5
print(type(i))

f = float(i)
print(type(f))

j = int(f)
print(type(j))

<class 'int'>
<class 'float'>
```

# 상호 형변환이 가능한 자료형

자료형 변환은 데이터 분석시 매우 유용하고 강력한 기능이지만

변수의 자료형이 서로 변환 가능한 형태인 경우에만 자료형 변환을 할 수 있다

```
# 정수(integer)를 실수(float)로, 실수(float)를 into 정수(integer)로 변환
i = 5
print(type(i))

f = float(i)
print(type(f))

j = int(f)
print(type(j))

<class 'int'>
<class 'float'>
<class 'int'>
```



# 상호 형변환이 가능한 자료형

```
# 숫자를 문자열(string)로 문자열을 숫자로 변환
i = 5
print(type(i))

s = str(i)
print(type(s))

f = float(s)
print(type(f))
```

```
# 만약 자료형이 서로 변환 가능한 형태가 아니면 오류가 발생
s = 'drum'
print(type(s))

f = float(s)           # error
```

# 상호 형변환이 가능한 자료형

```
# 숫자를 문자열(string)로 문자열을 숫자로 변환
i = 5
print(type(i))

s = str(i)
print(type(s))

f = float(s)
print(type(f))

<class 'int'>
```

```
# 만약 자료형이 서로 변환 가능한 형태가 아니면 오류가 발생
s = 'drum'
print(type(s))

f = float(s)           # error
```

# 상호 형변환이 가능한 자료형

```
# 숫자를 문자열(string)로 문자열을 숫자로 변환
```

```
i = 5
```

```
print(type(i))
```

```
s = str(i)
```

```
print(type(s))
```

```
f = float(s)
```

```
print(type(f))
```

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

```
<class 'str'>
```

```
# 만약 자료형이 서로 변환 가능한 형태가 아니면 오류가 발생
```

```
s = 'drum'
```

```
print(type(s))
```

```
f = float(s)           # error
```

# 상호 형변환이 가능한 자료형

```
# 숫자를 문자열(string)로 문자열을 숫자로 변환
```

```
i = 5
```

```
print(type(i))
```

```
s = str(i)
```

```
print(type(s))
```

```
f = float(s)
```

```
print(type(f))
```

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

```
<class 'str'>
```

```
<class 'float'>
```

```
# 만약 자료형이 서로 변환 가능한 형태가 아니면 오류가 발생
```

```
s = 'drum'
```

```
print(type(s))
```

```
f = float(s)           # error
```

# 상호 형변환이 가능한 자료형

```
# 숫자를 문자열(string)로 문자열을 숫자로 변환
```

```
i = 5
```

```
print(type(i))
```

```
s = str(i)
```

```
print(type(s))
```

```
f = float(s)
```

```
print(type(f))
```

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

```
<class 'str'>
```

```
<class 'float'>
```

```
# 만약 자료형이 서로 변환 가능한 형태가 아니면 오류가 발생
```

```
s = 'drum'
```

```
print(type(s))
```

```
f = float(s)          # error
```

```
<class 'str'>
```



# 상호 형변환이 가능한 자료형

```
# 숫자를 문자열(string)로 문자열을 숫자로 변환
i = 5
print(type(i))

s = str(i)
print(type(s))

f = float(s)
print(type(f))

<class 'int'>
<class 'str'>
<class 'float'>
```

```
# 만약 자료형이 서로 변환 가능한 형태가 아니면 오류가 발생
s = 'drum'
print(type(s))

f = float(s)           # error

<class 'str'>
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: 'drum'
```

# (제일 중요한) 주석 달기

## ● 코드에 대한 설명

- 중요한 점이나 다시 확인해야 하는 부분(reminder)을 표시
- 컴퓨터는 주석을 인식하지 않음, 즉 사용자만을 위한 것

## ● 가장 중요한 습관

- 개발자에게 있어서 주석을 다는 습관은 매우 중요
- 주석을 잘 작성하면 다른 사용자뿐만 아니라 본인도 나중에 작성한 코드를 쉽게 이해할 수 있고 코드의 분석 및 수정이 용이하므로 반드시 코딩하면서 동시에 주석다는 습관을 들이는 것이 중요

## ● 파이썬에서 주석다는 법

- 주석으로 처리할 내용 맨 앞에 '#'를 입력
- 주석은 한 줄을 온전히 차지할 수도 있고 그 줄의 코드 뒷 부분에 작성할 수도 있다
  - e.g. 한 줄 전체를 주석 처리할 경우
    - # 이름을 출력합니다.  
`print('Park')`
  - e.g. 그 줄의 마지막 부분에 작성할 경우
    - `print('Park')` # 이름을 출력합니다.
- 그 줄의 '#' 뒤에 작성되는 모든 내용은 주석으로 처리된다



#####

- ☒ 코드 실행에 영향을 미치지 않을 뿐만 아니라,
- ☒ 프로그램 실행 속도를 느리게 하지도 않고,
- ☒ 실행 프로그램의 용량을 늘리지도 않음

#####

# 매우 중요!!! : 프로그램 과제 머리말



## ● 머리말(header) 주석

- 프로그램 머리말 부분에 주석으로 프로그램에 대한 기본 정보를 입력하는 습관을 가지는 것이 중요
- 프로그램의 내용, 목적에 대한 간단한 설명
- 프로그램의 작성자, 소속, 프로젝트명, 버전, 작성 일자, 수정 일자 등의 정보를 입력

## ● 과제 머리말

- 본 강의에서 배부되는 과제의 머리말은 반드시 다음과 같이 작성해야 한다



```
# Assignment Number...:
# Assignment Title....:
# Student ID.....:
# Student Name.....: 반드시 한글 이름으로 이름 작성
# File Name.....:
# Program Description.:
```



## ● eval (표현식)

- 표현식은 '실행 가능한' 파이썬 표현식으로 파싱(구조 분석)을 하여 계산한 결과 값을 반환
- 실행 가능한 파이썬 표현식이 아니면 **SyntaxError** 예외를 발생 시킨다

```
'5 + 7'
```

```
eval('5 + 7')
```

```
eval(5 + '7')
```

```
'안녕' + '파이썬'
```

```
eval('안녕' + '파이썬')
```

```
eval("'안녕' + '파이썬'")
```

```
eval('print(10 + 5)')
```

```
x = eval(input('숫자를 입력하세요...: '))  
print(x * 10)
```



## eval (표현식)

- 표현식은 '실행 가능한' 파이썬 표현식으로 파싱(구조 분석)을 하여 계산한 결과 값을 반환
- 실행 가능한 파이썬 표현식이 아니면 `SyntaxError` 예외를 발생 시킨다

```
'5 + 7'
```

```
'5 + 7'
```

```
eval('5 + 7')
```

```
eval(5 + '7')
```

```
'안녕' + '파이썬'
```

```
eval('안녕' + '파이썬')
```

```
eval("'안녕' + '파이썬'")
```

```
eval('print(10 + 5)')
```

```
x = eval(input('숫자를 입력하세요...: '))  
print(x * 10)
```



## eval (표현식)

- 표현식은 '실행 가능한' 파이썬 표현식으로 파싱(구조 분석)을 하여 계산한 결과 값을 반환
- 실행 가능한 파이썬 표현식이 아니면 `SyntaxError` 예외를 발생 시킨다

```
'5 + 7'
```

```
'5 + 7'
```

```
eval('5 + 7')
```

```
12
```

```
eval(5 + '7')
```

```
'안녕' + '파이썬'
```

```
eval('안녕' + '파이썬')
```

```
eval("'안녕' + '파이썬'")
```

```
eval('print(10 + 5)')
```

```
x = eval(input('숫자를 입력하세요...: '))  
print(x * 10)
```



## eval (표현식)

- 표현식은 '실행 가능한' 파이썬 표현식으로 파싱(구조 분석)을 하여 계산한 결과 값을 반환
- 실행 가능한 파이썬 표현식이 아니면 **SyntaxError** 예외를 발생 시킨다

```
'5 + 7'
```

```
'5 + 7'
```

```
eval('5 + 7')
```

```
12
```

```
eval(5 + '7')
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unsupported operand type(s) for +:  
'int' and 'str'
```

```
'안녕' + '파이썬'
```

```
eval('안녕' + '파이썬')
```

```
eval("'안녕' + '파이썬'")
```

```
eval('print(10 + 5)')
```

```
x = eval(input('숫자를 입력하세요...: '))  
print(x * 10)
```





## eval (표현식)

- 표현식은 '실행 가능한' 파이썬 표현식으로 파싱(구조 분석)을 하여 계산한 결과 값을 반환
- 실행 가능한 파이썬 표현식이 아니면 **SyntaxError** 예외를 발생 시킨다

```
'5 + 7'
```

```
'5 + 7'
```

```
eval('5 + 7')
```

```
12
```

```
eval(5 + '7')
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unsupported operand type(s) for +:  
'int' and 'str'
```

```
'안녕' + '파이썬'
```

```
'안녕파이썬'
```

```
eval('안녕' + '파이썬')
```

```
eval("'안녕' + '파이썬'")
```

```
eval('print(10 + 5)')
```

```
x = eval(input('숫자를 입력하세요...: '))  
print(x * 10)
```





## eval (표현식)

- 표현식은 '실행 가능한' 파이썬 표현식으로 파싱(구조 분석)을 하여 계산한 결과 값을 반환
- 실행 가능한 파이썬 표현식이 아니면 **SyntaxError** 예외를 발생 시킨다

```
'5 + 7'
```

```
'5 + 7'
```

```
eval('5 + 7')
```

```
12
```

```
eval(5 + '7')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +:
'int' and 'str'
```

```
'안녕' + '파이썬'
```

```
'안녕파이썬'
```

```
eval('안녕' + '파이썬')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name '안녕파이썬' is not defined
```

```
eval("'안녕' + '파이썬'")
```

```
eval('print(10 + 5)')
```

```
x = eval(input('숫자를 입력하세요...: '))
print(x * 10)
```



## eval (표현식)

- 표현식은 '실행 가능한' 파이썬 표현식으로 파싱(구조 분석)을 하여 계산한 결과 값을 반환
- 실행 가능한 파이썬 표현식이 아니면 **SyntaxError** 예외를 발생 시킨다

```
'5 + 7'
```

```
'5 + 7'
```

```
eval('5 + 7')
```

```
12
```

```
eval(5 + '7')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +:
'int' and 'str'
```

```
'안녕' + '파이썬'
```

```
'안녕파이썬'
```

```
eval('안녕' + '파이썬')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name '안녕파이썬' is not defined
```

```
eval("'안녕' + '파이썬'")
```

```
'안녕파이썬'
```

```
eval('print(10 + 5)')
```

```
x = eval(input('숫자를 입력하세요...: '))
print(x * 10)
```



## eval (표현식)

- 표현식은 '실행 가능한' 파이썬 표현식으로 파싱(구조 분석)을 하여 계산한 결과 값을 반환
- 실행 가능한 파이썬 표현식이 아니면 **SyntaxError** 예외를 발생 시킨다

```
'5 + 7'
```

```
'5 + 7'
```

```
eval('5 + 7')
```

```
12
```

```
eval(5 + '7')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +:
'int' and 'str'
```

```
'안녕' + '파이썬'
```

```
'안녕파이썬'
```

```
eval('안녕' + '파이썬')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name '안녕파이썬' is not defined
```

```
eval("'안녕' + '파이썬'")
```

```
'안녕파이썬'
```

```
eval('print(10 + 5)')
```

```
15
```

```
x = eval(input('숫자를 입력하세요...: '))
print(x * 10)
```



## eval (표현식)

- 표현식은 '실행 가능한' 파이썬 표현식으로 파싱(구조 분석)을 하여 계산한 결과 값을 반환
- 실행 가능한 파이썬 표현식이 아니면 **SyntaxError** 예외를 발생 시킨다

```
'5 + 7'
```

```
'5 + 7'
```

```
eval('5 + 7')
```

```
12
```

```
eval(5 + '7')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +:
'int' and 'str'
```

```
'안녕' + '파이썬'
```

```
'안녕파이썬'
```

```
eval('안녕' + '파이썬')
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name '안녕파이썬' is not defined
```

```
eval("'안녕' + '파이썬'")
```

```
'안녕파이썬'
```

```
eval('print(10 + 5)')
```

```
15
```

```
x = eval(input('숫자를 입력하세요...: '))
print(x * 10)
```

```
숫자를 입력하세요...: 5
50
```

## CHAPTER 3 & 4 실습

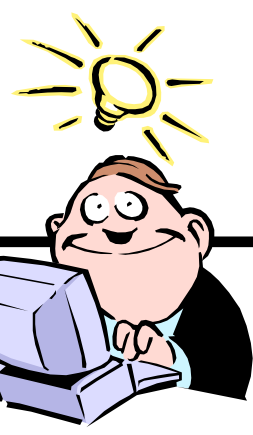


# 파이썬 문법 미리 알아두기

---

## Lab Exercises

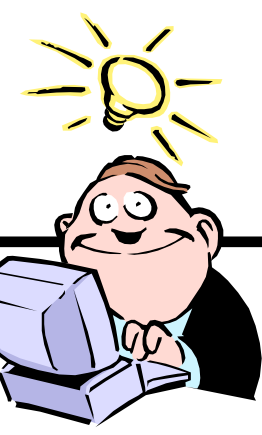




## ● 자료형 변환하기

- *i*와 *s* 변수를 만들어 *i*에 숫자 1234를 *s*에 문자열 '1234'를 할당
- 두 개의 변수가 같은 값을 가졌는지 확인
  - 참(True) 또는 거짓(False)으로 확인
- 만약 두 개의 변수 값이 다르다면 두 변수의 값이 같을 수 있도록 한다
  - 참(True) 또는 거짓(False)으로 확인





## ● 입력, 연산, 출력하기

- 사용자로부터 이름을 입력받는다.
- 사용자의 이름과 함께 환영 메시지를 출력한다.
- 사용자로부터 나이를 입력받아서 내년엔 사용자가 몇 살이 되는지를 계산해서 영어와 한국어로 각각 출력한다.

## ● 실행 결과 예시

```
이름을 입력하세요....: 파이썬
안녕 파이썬
몇 살이세요? 19
You will be 20 years old in the next year.
내년에 20살이 되시는군요.
Bye~~~!
```