

AI Programming

Lecture 3

Review

- `print(a, '+', b, '=', result)`
- `input('첫 번째 숫자를 입력하세요 : ')`
- `int("100") = ?`

Preview

- Ch. 3 변수와 데이터형
 - 3.3 변수의 선언과 사용
 - 3.5 기본 데이터형
 - 3.2 `print()` 함수를 사용한 다양한 출력

3.3 변수의 선언과 사용

Variables

• 변수의 선언

- 변수는 값을 저장하는 메모리 공간 (그릇)
- 변수 선언은 그릇을 준비하는 것
- Python은 변수 선언이 필요 없음

• 변수의 이름

- 대소문자 구분: `myVar` \neq `MyVar`
- 숫자나 언더바 포함 가능 (숫자로 시작 X): `var2`, `_var`, `var_2`, `2var`
- 예약어 사용 금지: `True`, `False`, `None`, `and`, `or`, `not`, `break`, `continue`, `return`, `if`, `elif`, `for`, `while`, `except`, `finally`, `global`, `import`, `try`,...

```
>>> 2var = 3  
SyntaxError: invalid syntax
```

Variables

- **Types**

- Integer, float, string
- Bool: True or False

```
boolVar = True  
intVar = 0  
floatVar = 0.0  
strVar = ""
```



그림 3-7 변수의 종류

Variables

- `type()` **function**
 - 변수의 type 확인

```
type(boolVar), type(intVar), type(floatVar), type(strVar)
```

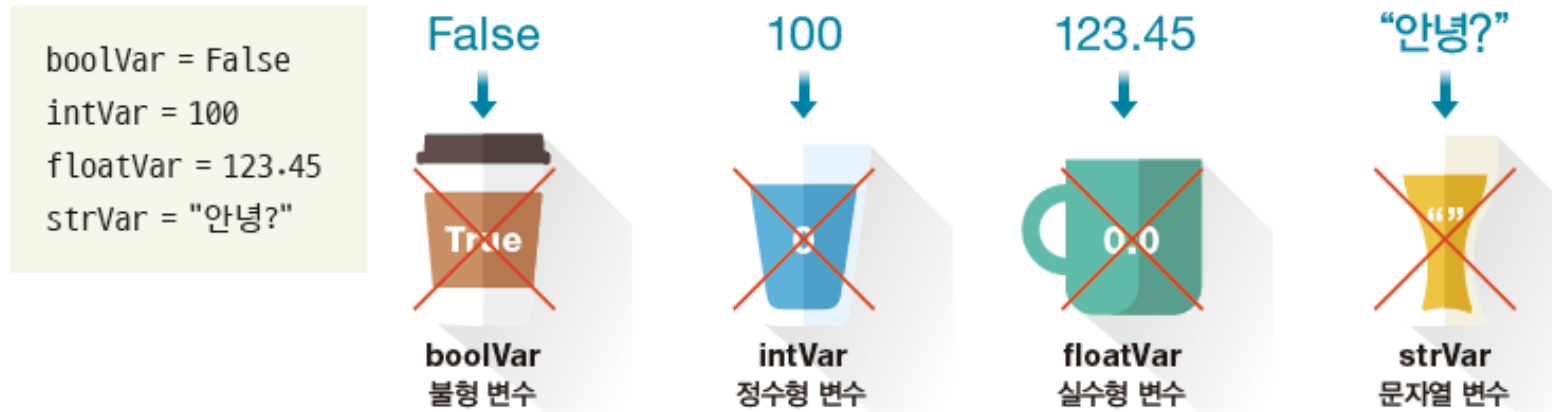
출력 결과

```
(<class 'bool'>, <class 'int'>, <class 'float'>, <class 'str'>)
```

Variables

• 변수의 사용

- 변수는 값을 대입하면 사용 가능
- 기존 값이 사라지고 새로 입력한 값으로 변경됨



Variables

• 변수의 사용

- 변수에 다른 변수의 값을 대입할 수 있음
- 계산 결과를 대입하는 것도 가능

```
var2 = 200  
var1 = var2
```



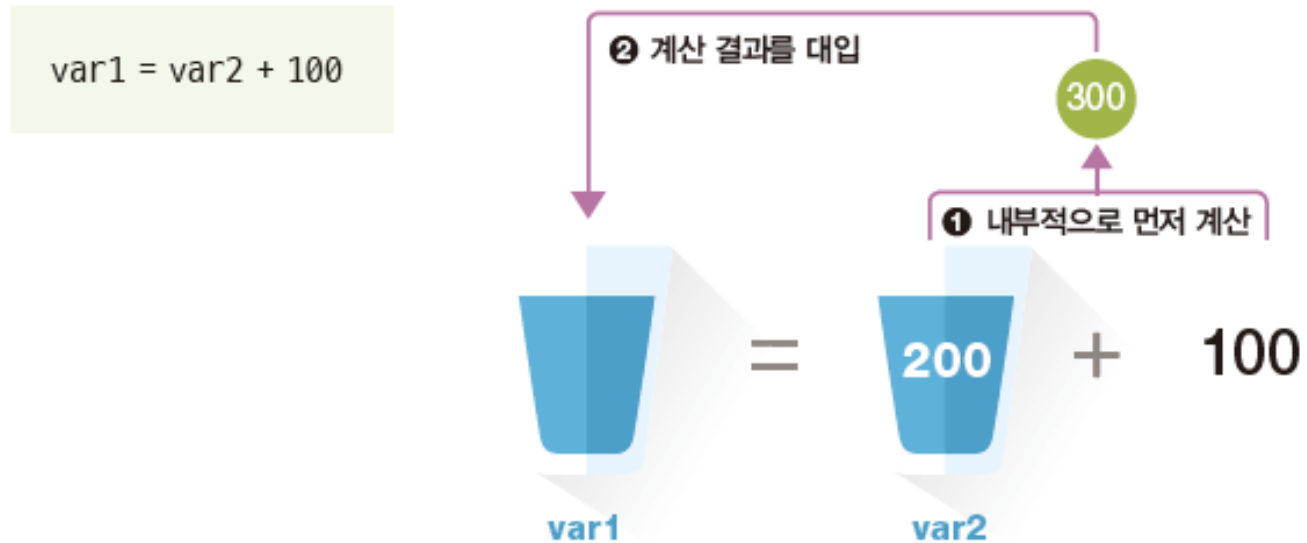
```
var1 = 100 + 100
```



Variables

- 변수의 사용

- 변수에 숫자와 변수의 연산을 대입할 수 있음



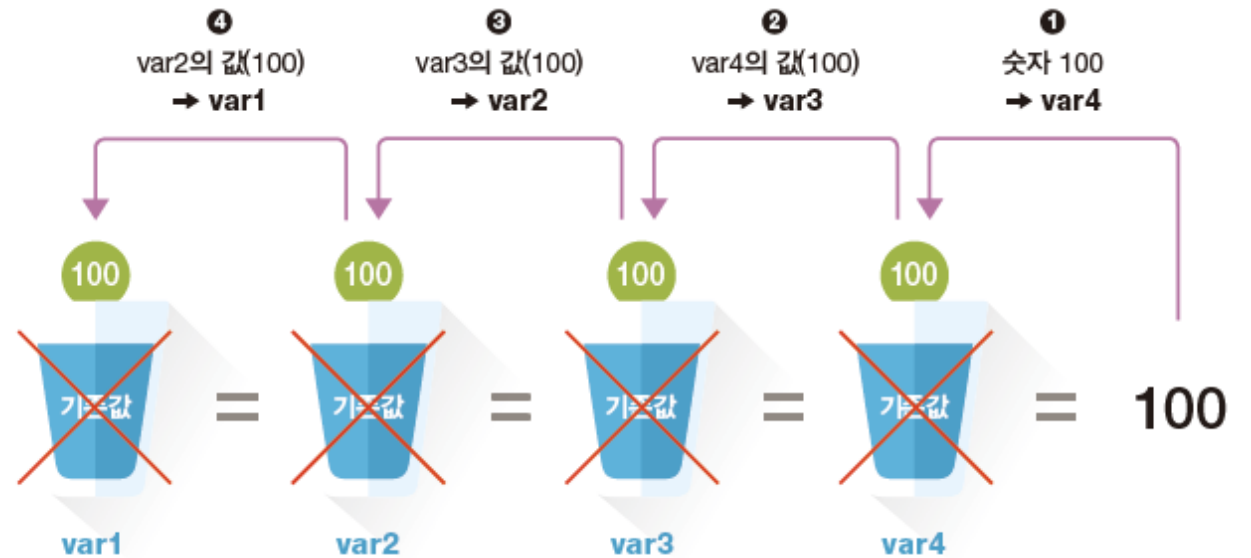
Variables

• 변수의 사용

- 변수에 연속된 값을 대입하는 방식

```
var1 = var2 = var3 = var4 = 100
```

```
var4 = 100  
var3 = var4  
var2 = var3  
var1 = var2
```

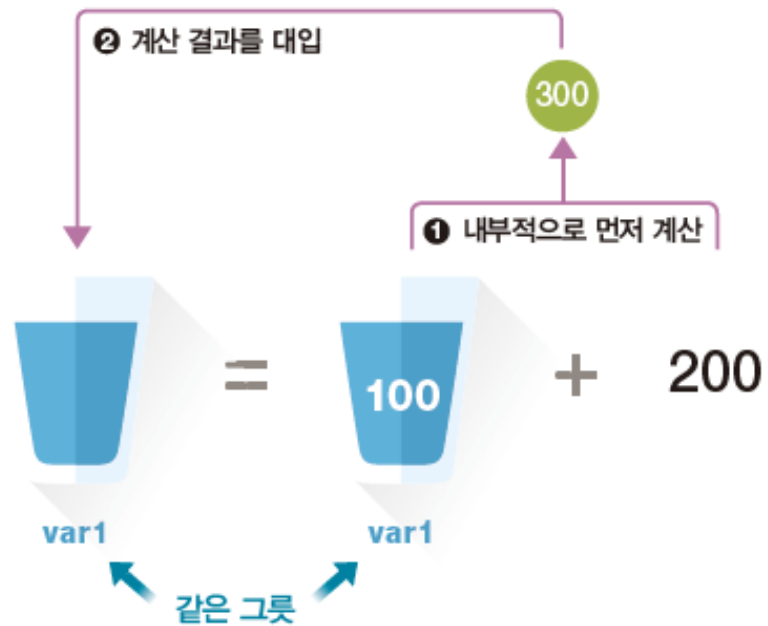


Variables

• 변수의 사용

- 변수에 연산 결과를 자신의 값으로 다시 대입하는 방식

`var1 = var1 + 200`



Variables

- 고급 활용법

- 하나의 값을 다수의 변수에 할당

- `x = y = z = 0`

- 동시에 다수의 변수에 값을 할당

- `x, y, z = 10, 20, 30`

- `x = 10; y = 20; z = 30`

- 동시 할당: 두 변수의 값 교환

- `x, y = y, x`

```
tmp = x
```

```
x = y
```

```
y = tmp
```

3.5 기본 데이터형

Data Type

- Data Type

- 변수의 데이터형은 값을 넣는 순간마다 변경

```
>>> myVar = 100
>>> type(myVar)
<class 'int'>
>>> myVar = 100.0
>>> type(myVar)
<class 'float'>
>>> myVar = 100
>>> type(myVar)
<class 'int'>
>>> myVar = 'myVar'
>>> type(myVar)
<class 'str'>
```

Data Type

- Float

```
a = 3.14  
b = 3.14e5    3.14e5 = 3.14 × 105  
print(a, b)
```

출력 결과

```
3.14 314000.0
```


Data Types

- **String**

- 그대로 출력하면 작은따옴표와 함께 출력
- `print()`로 출력하면 그대로 출력

```
a = "파이썬 만세"  
a  
print(a)  
type(a)
```

출력 결과

```
'파이썬 만세'  
파이썬 만세  
<class 'str'>
```

Data Types

- Bool

- True, False

```
a = True  
type(a)
```

출력 결과

```
<class 'bool'>
```

Data Types

- Bool & relational operators (관계 연산자, Ch. 4)
 - Logics: 비교 결과를 참이나 거짓으로 저장

```
a = (100 == 100)
b = (10 > 100)
print(a, b)
```

출력 결과

True False

```
if a:
    print("100 == 100은 참 입니다")
if b:
    print("10 > 100은 참 입니다")
```

Data Types

- Arithmetic operators (Ch. 4)

- `+`, `-`, `*`, `/`
- `**`: 제곱
- `%`: 나머지
- `//`: 몫 (정수 나눗셈)

```
a, b = 9, 2  
print(a ** b, a % b, a // b)
```

출력 결과

81 1 4

3.2 `print()` 함수를 사용한 다양한 출력

Print Function

- String formatting

```
print(num1, "의 제곱근은 ", sqrt, " 입니다.")
```

- 문자열과 변수를 print 함수로 출력하는 방법
 - % formatting
 - format function
 - f-string

Print Function

- % formatting

```
print("안녕하세요?")
```

```
>>> print("안녕하세요?")  
안녕하세요?
```

❶ `print("100")`

❷ `print("%d" % 100)`

Placeholder

```
>>> print("100")  
100
```

```
>>> print("%d" % 100)  
100
```

Print Function

- % formatting

❸ `print("100 + 100")`

❹ `print("%d" % (100 + 100))`

```
>>> print("100+100")
100+100
```

```
>>> print("%d" % (100+100))
200
```

❺ `print("%d" % (100, 200))`

❻ `print("%d %d" % (100))`

```
>>> print("%d" % (100, 200))
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    print("%d" % (100, 200))
TypeError: not all arguments converted during string formatting
```

```
>>> print("%d %d" % (100))
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print("%d %d" % (100))
TypeError: not enough arguments for format string
```


Print Function

- Basics

`print("%d %d" % (100 , 200))`



```
>>> print("%d %d" % (100, 200))  
100 200
```

- `%d`: decimal number (integer)
- # of `%d` = # of decimals formatted

Print Function

- **Numbers**

- `%d`: decimal number
- `%x`: hexadecimal number
- `%o`: octal number
- `%f`: floating number

- **Words**

- `%c`: a single character
- `%s`: string (more than two characters)

Print Function

- Examples

```
print("%d / %d = %d" % (100, 200, 0.5))
```

```
>>> print("%d / %d = %d" % (100, 200, 0.5))  
100 / 200 = 0
```

```
print("%d / %d = %5.1f" % (100, 200, 0.5))
```

```
>>> print("%d / %d = %5.1f" % (100, 200, 0.5))  
100 / 200 = 0.5
```

Print Function

- Practice

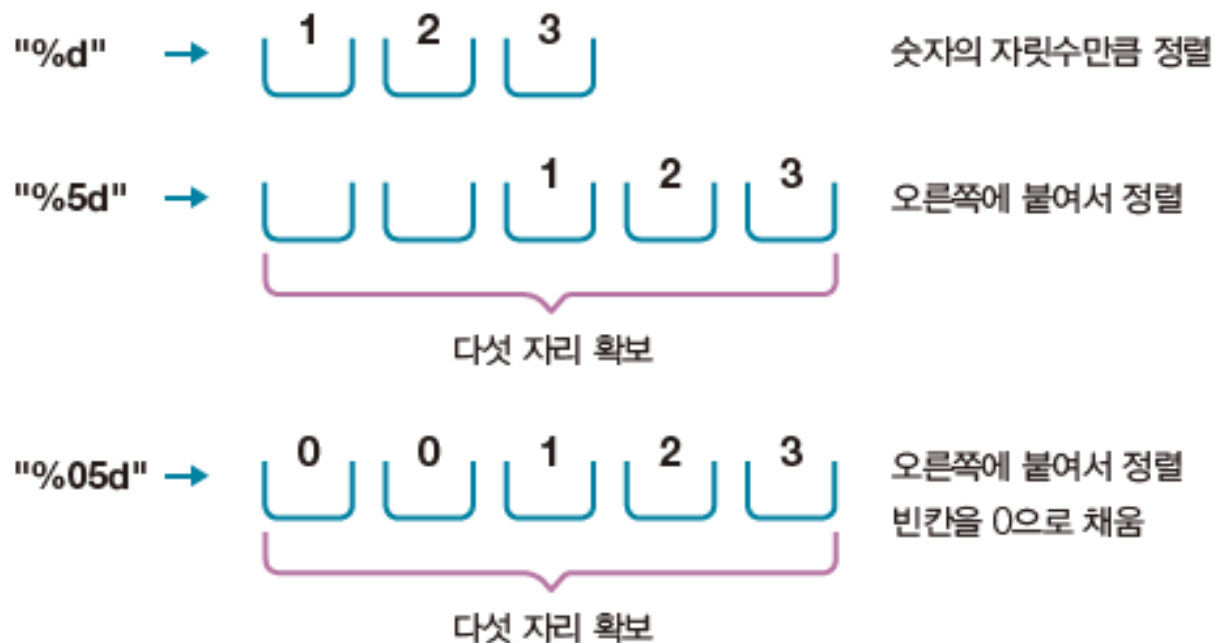
Code03-01.py

```
1 print("%d" % 123)
2 print("%5d" % 123)
3 print("%05d" % 123)
4
5 print("%f" % 123.45)
6 print("%7.1f" % 123.45)
7 print("%7.3f" % 123.45)
8
9 print("%s" % "Python")
10 print("%10s" % "Python")
```

```
123
  123
00123
123.450000
  123.5
123.450
Python
   Python
```

Print Function

- Integer



Print Function

- Float

"%f" →

1	2	3	.	4	5	0	0	0	0
---	---	---	---	---	---	---	---	---	---

 소수점 아래 여섯 자리까지
무조건 출력

"%7.1f" →

			1	2	3	.	5
--	--	--	---	---	---	---	---

 소수점 아래 첫째 자리만 출력
소수점 아래 둘째 자리에서 반올림

일곱 자리 확보

"%7.3f" →

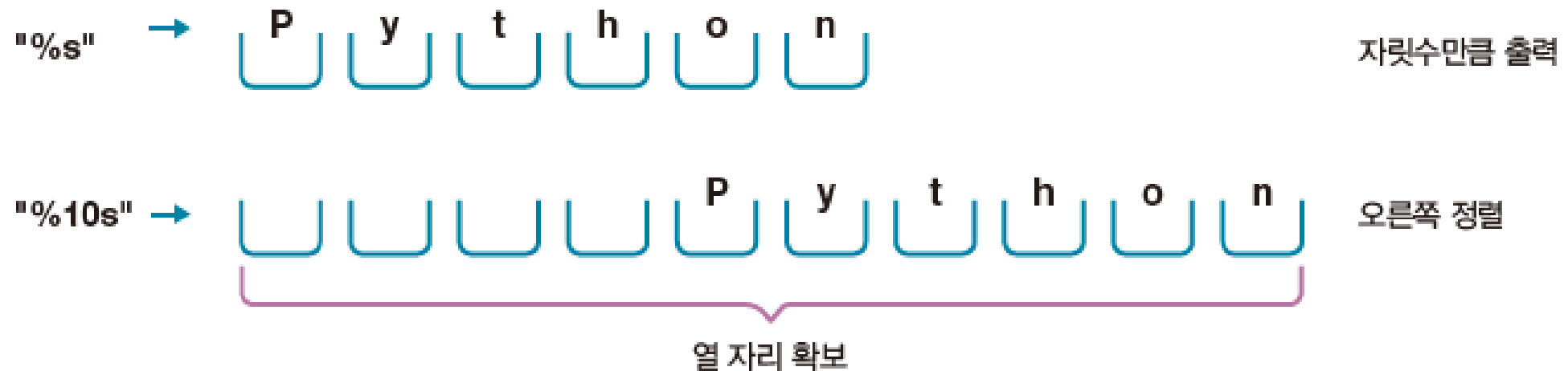
1	2	3	.	4	5	0
---	---	---	---	---	---	---

 소수점 아래 셋째 자리까지 출력
오른쪽 빈칸은 0으로 채움

일곱 자리 확보

Print Function

- String



Format Function

- `format()`
 - `"{index1} {index2}".format(num1, num2)`

```
num1, num2 = 2, 4  
print("구구단 {0:d} X {1:d} = {2:d}".format(num1, num2, num1*num2))
```

```
diameter, pi = 2, 3.14  
print("지름이 {0:d}인 원의 둘레: {0:d} X {1:.2f} = {2:.2f}".format(diameter, pi, diameter*pi))
```

```
print("{2:d} {1:d} {0:d}".format(100, 200, 300))
```


Format Function

- `format()`

```
print("%d %5d %05d" % (123, 123, 123))  
print("{0:d} {1:5d} {2:05d}".format(123, 123, 123))
```

`print("{0:d} {1:5d} {2:05d}".format(123, 123, 123))`

↑ ↑ ↑
0번째 1번째 2번째

123 □□123 00123

f-String

- f-string formatting

- 출력을 원하는 문자열 앞에 "f" 문자 삽입
- 변수는 중괄호 { }로 표현

```
num1 = 1
num2 = 3
print(f"첫 번째 숫자: {num1}")
print(f"두 번째 숫자: {num2}")
print(f"{num1} + {num2}는 {num1 + num2} 입니다.")
```

```
pi = 3.141592
print(f"원주율: {pi}")
print(f"원주율: {pi:.3f}")
print(f"원주율: {pi:1.2f}")
print(f"원주율: {pi:7.3f}")
```

Escape

- Escape

이스케이프 문자	역할	설명
\n	새로운 줄로 이동	<code>Enter</code> 를 누른 효과
\t	다음 탭으로 이동	<code>Tab</code> 을 누른 효과
\b	뒤로 한 칸 이동	<code>Backspace</code> 를 누른 효과
\\	\ 출력	
\'	' 출력	
\"	" 출력	

```
print("한 행입니다. 또 한 행입니다.")
print("한 행입니다. \n또 한 행입니다.")
```

```
>>> print("한 행입니다. 또 한 행입니다.")
한 행입니다. 또 한 행입니다.
>>> print("한 행입니다. \n또 한 행입니다.")
한 행입니다.
또 한 행입니다.
```

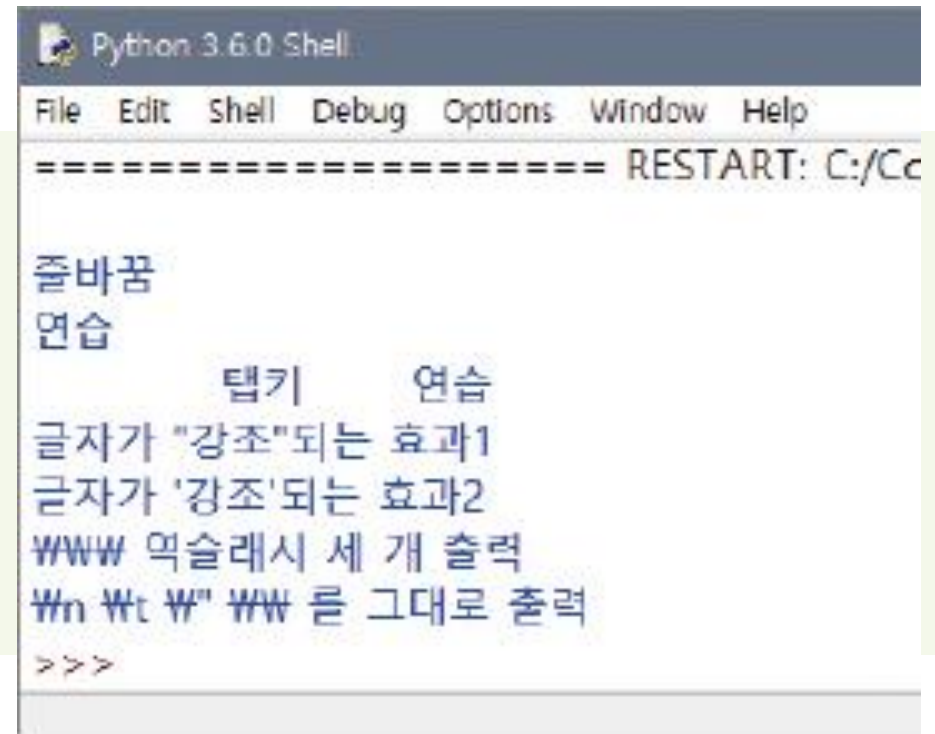
Escape

• Exercise

Code03-02.py

```
1 print("\n줄바꿈\n연습 ")
2 print("\t탭키\t연습")
3 print("글자가 \"강조\"되는 효과1")
4 print("글자가 \'강조\'되는 효과2")
5 print("\\\\\\\\ 역슬래시 세 개 출력")
6 print(r"\n \t \" \"를 그대로 출력")
```

Escape 문자를 그대로 출력



```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Cc

줄바꿈
연습

        탭키      연습
글자가 "강조"되는 효과1
글자가 '강조'되는 효과2
\\\\\\ 역슬래시 세 개 출력
\n \t \" \"를 그대로 출력
>>>
```

Summary

- Print function

```
print("%d %5d %05d" % (123, 123, 123))  
print("{0:d} {1:5d} {2:05d}".format(123, 123, 123))
```

```
x = 1  
y = 1  
print(f"첫 번째 숫자: {x}")  
print(f"두 번째 숫자: {y}")  
print(f"{x} + {y}는 {x + y} 입니다.")
```

- Variables

```
type(boolVar), type(intVar), type(floatVar), type(strVar)
```

출력 결과

```
(<class 'bool'>, <class 'int'>, <class 'float'>, <class 'str'>)
```

Assignment 2

- 챗봇 프로그램

- % formatting
- format function
- f-string
- Escape (\n)

- 마지막 두 문장은 한 개의 print 함수로 구현

이름이 어떻게 되시나요? 이훈
만나서 반갑습니다 이훈 씨.

취미가 무엇인가요? 영화보기
저도 영화보기 좋아합니다.

나이가 어떻게 되시나요? 34
내년이면 35살이 되시는군요.

키가 어떻게 되시나요? 181.57
반올림하면 181.6 cm 이군요.

만나서 반가웠습니다 이훈 씨.
다음에는 영화보기를 같이 합시다.