

# 학습 내용

1부. 프로그래밍 언어 기본



1장. 파이썬 개요 및 개발환경 구성



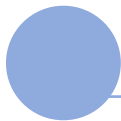
2장. 자료형과 연산자



3장. 데이터 구조



4장. 제어문



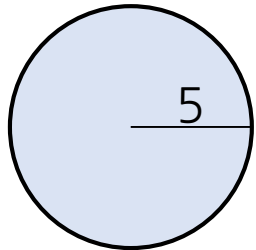
5장. 함수

- 1. 변수
- 2. 화면 입출력
- 3. 기본 자료형
- 4. 포매팅
- 5. 연산자
- 6. 문자열 다루기
- 7. 날짜 다루기

# 1절. 변수

## 2장. 자료형과 연산자

- 프로그램이 실행되는 동안 상황에 따라 **변하는 값을 저장**
- 코드 내에서 매번 값을 지정해 주면 코드를 작성하거나 수정하기 어려움
- 변수는 코드 내에서 **자료를 일관성 있게 사용하고 관리**하기 위해서 이름(identifier)을 부여해 다른 변수 또는 자료와 구분해서 사용할 수 있도록 함.
- 이름을 부여하고 값을 저장해 놓으면 **연산에 변수 이름을 사용할 수** 있음.



반지름이 5(cm)인 원의 넓이를 구하려면?  
 $5 \times 5 \times 3.14 = 78.5(\text{cm}^2)$

# 1.1. 변수

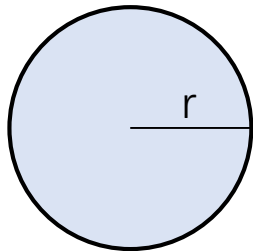
1절. 변수

## ● 변수 선언(Declaration)

- 값을 저장할 수 있는 변수를 만드는 것
- 변수 선언을 위해서는 변수에 어떤 종류의 값이 저장되어야 하는지 알리기 위한 타입이 있어야 함
- 파이썬은 변수를 선언하기 위한 타입이 없으므로 변수 선언 과정이 없음

## ● 변수 할당(Assignment)

- 어떤 값이 이름을 갖도록 하는 과정
- = 연산자를 이용
- 변수의\_이름 = 연산\_결과\_값



반지름(radius)  $r$ 이 5(cm)인 원의 넓이를 구하려면?

$$r = 5$$

$$\text{area} = r \times r \times 3.14$$

## 1.2. 변수 이름 규칙

1절. 변수

- 변수의 이름은 **문자, 숫자, 밑줄(\_ (underscore))**문자를 포함하며 수에 사용하는 문자는 **대소문자를 구분**
- 변수는 **숫자로 시작할 수 없음**
- **공백, 문장부호, 특수문자(밑줄 문자만 유일하게 가능)** 등은 **사용 불가**
- 파이썬의 **예약어**(예: class, def)는 **사용할 수 없음**
- **사용 중인 내장 함수나 모듈 이름**(예: id, list, print)등은 **사용 지양**

| 바른 예        |                  | 잘 못된 예                     |                        |
|-------------|------------------|----------------------------|------------------------|
| year2000    | 숫자 포함 가능         | 2000year                   | 숫자로 시작 못함              |
| Class       | 대/소문자를 구분함       | class                      | 예약어 임                  |
| member_name | 두 단어이면 _로 연결     | member name<br>member.name | .과 공백 등 특수문자를 포함할 수 없음 |
| print_      | _를 붙여 사용할 것을 권장함 | print                      | 문법 오류는 없지만 권장하지 않음     |

# 1.3. 변수에 값 할당

1절. 변수

- 할당은 **변수에 값을 저장**하는 것을 의미
- 파이썬에서 변수에 값을 저장하기 위해서는 **할당연산자**를 사용
- 가장 많이 사용하는 할당 연산자는 “=”
- 변수 a에 정수 값 10을 할당하고 출력하려면...

```
1 a = 10
2 print(a)
```

10

```
1 10 = a
```

File "<ipython-input-3-a1ebfc217b9f>", line 1

```
10 = a
    ^
```

**SyntaxError:** can't assign to literal

```
1 int a = 10
```

File "<ipython-input-4-392d75864529>", line 1

```
int a = 10
    ^
```

**SyntaxError:** invalid syntax

## 1.4. id()

1절. 변수

- 변수는 파이썬에서 가장 많이 사용되는 객체
- id()는 객체의 주소 값을 출력

```
1 x = 100  
2 print(x)
```

100

```
1 print(id(x))
```

1643870368

# 1.5. 변수 삭제

1절. 변수

- `del` 변수
- `del`은 현재 커널의 변수를 삭제
- 여러 개 변수를 동시에 삭제하려면 변수 목록을 콤마(,)로 구분해 나열
- `whos` 명령을 이용하면 현재 커널에 정의되어 있는 변수 목록을 확인

|          |      |           |
|----------|------|-----------|
| 1        | whos |           |
| Variable | Type | Data/Info |
| <hr/>    |      |           |
| a        | int  | 10        |
| b        | int  | 20        |

|                                 |                       |
|---------------------------------|-----------------------|
| 1                               | <code>del a, b</code> |
| 1                               | whos                  |
| Interactive namespace is empty. |                       |

# 1.6. 다중 변수 선언

1절. 변수

- 한 라인에 여러 개 변수를 선언해 사용

|   |               |
|---|---------------|
| 1 | a, b = 10, 20 |
| 2 | print(a+b)    |

30

- 변수에 값의 할당 작업은 순차적으로 발생하는 것은 아님

|   |               |
|---|---------------|
| 1 | a, b = 10, 20 |
| 2 | a, b = b, a+b |

|   |      |
|---|------|
| 1 | a, b |
|---|------|

(20, 30)

|   |               |
|---|---------------|
| 1 | a, b = 10, 20 |
| 2 | a = b         |
| 3 | b = a+b       |

|   |      |
|---|------|
| 1 | a, b |
|---|------|

(20, 40)



## 1.7. 도움말

1절. 변수

- `help(/x/)`
- `x` : 도움말을 얻을 함수 이름입니다. 대괄호([ ])는 선택사항을 의미

```
1 help(print)
```

Help on built-in function print in module builtins:

```
print(...)
```

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

Prints the values to a stream, or to sys.stdout by default.

Optional keyword arguments:

file: a file-like object (stream); defaults to the current sys.stdout.

sep: string inserted between values, default a space.

end: string appended after the last value, default a newline.

flush: whether to forcibly flush the stream.

## 2.1. 사용자 입력

2절. 화면 입출력

- *read\_message = input("prompt\_message")*
- 사용자로부터 값을 입력 받기 위한 함수
- 입력한 값은 항상 **문자열** 임

```
1 first_number = int(input('첫 번째 숫자:'))
```

첫 번째 숫자:20

```
1 second_number = int(input('두 번째 숫자:'))
```

두 번째 숫자:30

```
1 print(first_number + second_number)
```

50

## 2.2. 화면 출력

2절. 화면 입출력

- `print()`

```
1 print('welcome to', 'python')
```

welcome to python

2.x 버전 스타일

```
1 print 'welcome to', 'python'
```

File "<ipython-input-26-dbab04424bcc>", line 1

print 'welcome to', 'python'

^

**SyntaxError:** Missing parentheses in call to 'print'. Did you mean print('welcome to', 'python')?

## 2.2. 화면 출력(sep 속성)

2절. 화면 입출력

- `print('message', sep=' ', end='\n', file=sys.stdout)`

```
1 print('Hello', 'World')
```

Hello World

```
1 print('Hello', 'World', sep=',')
```

Hello,World

```
1 print('Hello', 'World', sep='Wt')
```

Hello World

## 2.2. 화면 출력(end 속성)

2절. 화면 입출력

- `print('message', sep=' ', end='\n', file=sys.stdout)`

```
1 print('Hello')  
2 print('World')
```

Hello  
World

```
1 print('Hello', end='Wt')  
2 print('World')
```

Hello WtWorld

# 3절. 기본 자료형

## 2장. 자료형과 연산자

- 자료형(데이터 타입, data type)
  - 프로그래밍 언어들은 변수가 가져야 할 값의 크기, 형식, 범위 등에 따라서 값의 유형을 정해 놓고 사용
  - 프로그래밍 언어에서 정수, 실수, 논리 등 여러 종류의 자료(데이터)가 어떤 값을 가질 수 있는지에 대해 알려주는 속성
  - 변수에 저장할 수 있는 값의 범위(또는 크기)와 값을 저장하는 방식이 달라짐
  - 해당 자료형을 이용해서 수행할 수 있는 명령들이 달라질 수 있음
- 자료형을 지정하는 키워드
  - 정수(integer)를 지정하는 키워드는 int
  - 부동소수점(실수, floating-point)를 지정하는 float 또는 double
  - 논리(boolean)을 지정하는 boolean
  - 문자 한 개(character)를 지정하는 char
  - 문자와 숫자로 이루어진 문자열(string)을 지정하는 string
- 파이썬은 자료형의 개념이 있지만 변수를 선언할 때 자료형을 지정하지 않음

## 3.1. 숫자형

3절. 기본 자료형

- 정수(int), 실수(float), 복소수(complex) 형
- 소수점이 없는 정수는 int 형
  - 파이썬 3.x 버전에서는 2.x 버전에 있었던 long 형이 없어지고 모든 정수를 int 형으로 인식
  - 모든 정수는 int로 표현됨. 시스템의 최대수(sys.maxsize)가 있으나, 그보다 큰 숫자도 정수로 처리됨
- 소수점이 있는 숫자는 float 형
- 허수부를 포함하는 복소수는 complex 형
  - 허수부를 표현하는 문자는 i가 아니고 j

# 1) 정수

3절. 기본 자료형 > 3.1. 숫자형

- 숫자의 크기가 얼마가 되든지 모든 정수는 int 자료형으로 처리

```
1 import sys
2 sys.maxsize
```

9223372036854775807

```
1 a = 10
2 type(a)
```

int

```
1 type(sys.maxsize)
```

int

```
1 import sys
2 b = sys.maxsize
3 print(b)
```

9223372036854775807

```
1 type(sys.maxsize+1)
```

int

```
1 type(b)
```

int



## 2) 실수

3절. 기본 자료형 > 3.1. 숫자형

- 부동소수점
- float 형으로 처리
- 정수형과 정수형의 나눗셈 연산은 실수형(float)

```
1 c = 3.5  
2 type(c)
```

float

```
1 3/2
```

1.5

```
1 type(2/2)
```

float

### 3) 복소수

3절. 기본 자료형 > 3.1. 숫자형

- 복소수(complex number)는 실수(實數, (real number)와 허수(虛數, imaginary number)의 합으로 나타내는 수 체계
- 허수는 j문자를 붙여 표현

```
1 c = 3 + 2j
2 type(c)
```

complex

```
1 d = 1j
2 print(d**2)
```

(-1+0j)

```
1 type(d**2)
```

complex

```
1 c = 3 + j

-----
NameError                                Traceback (most
recent call last)
<ipython-input-62-ae86b6a564e7> in <module>()
----> 1 c = 3 + j

NameError: name 'j' is not defined
```

```
1 d = 3 + 1j
```

```
1 d
```

(3+1j)

# 1) 문자형의 표현

3절. 기본 자료형 > 3.2. 문자형

- 단일문자와 문자열을 구분하지 않음
- 겹따옴표("와 ") 또는 홑따옴표('와 ') 로 묶어서 사용

```
1 name = "Hong GilDong"  
2 address = '서울시 마포구'  
3 print(name, address)
```

Hong GilDong 서울시 마포구

## 2) 여러 줄 문자 표현

3절. 기본 자료형 > 3.2. 문자형

- 여러 줄 문자열은 겹따옴표 3개(''')와 """" 또는 홑따옴표 3개(''')를 사용

```
1 text = '''이렇게 작성하면  
2 줄 바꿈도 그대로 적용해서  
3 여러 줄의 문자를 작성할 수 있습니다.'''
```

```
1 print(text)
```

이렇게 작성하면  
줄 바꿈도 그대로 적용해서  
여러 줄의 문자를 작성할 수 있습니다.

### 3) 소스코드 줄 바꿈

3절. 기본 자료형 > 3.2. 문자형

- 라인의 맨 마지막에 있는 역슬래시(\ 또는 \n)는 소스코드 줄 바꿈

```

1  text = '''\
2  이렇게 작성하면
3  줄 바꿈도 그대로 적용해서
4  여러 줄의 문자를 작성할 수 있습니다.\
5  '''

```

```

1  print(text)

```

이렇게 작성하면  
 줄 바꿈도 그대로 적용해서  
 여러 줄의 문자를 작성할 수 있습니다.

## 4) 탈출 문자

3절. 기본 자료형 > 3.2. 문자형

- 문자열 내에서 특별한 의미를 갖는 문자들은 역슬래시(\)를 이용하여 이스케이프(escape) 문자를 사용

| 문자 | 의미                 |
|----|--------------------|
| \n | 줄 바꿈               |
| \t | 탭                  |
| \r | 리턴(행의 첫 번째 열로 돌아옴) |
| \0 | 널(null)            |
| \\ | \ 문자 표시            |
| \' | '(홀따옴표) 문자 표시      |
| \" | "(겹따옴표) 문자 표시      |

## 4) 탈출 문자

3절. 기본 자료형 > 3.2. 문자형

```
1 print("Hello\nWorld")
```

Hello  
World

```
1 print("Hello\tWorld")
```

Hello      World

```
1 print("Hello\rWorld")
```

World

```
1 print("Hello\0World")
```

HelloWorld

```
1 print("Hello\WWorld")
```

HelloWWorld

```
1 print("Hello\'World")
```

Hello'World

```
1 print("Hello\"World")
```

Hello"World

## 5) 문자열 연결하기

3절. 기본 자료형 > 3.2. 문자형

- 문자열과 문자열을 **+**(덧셈)하면 **문자열을 연결**
- 문자열을 **공백**으로 연결해도 **문자열을 연결**
- 문자열과 숫자를 **\***(곱셈) 연산 하면 문자열을 곱셈함 숫자만큼 **반복**

```
1 "Hello"+"World"
```

```
'HelloWorld'
```

```
1 "Hello" "World"
```

```
'HelloWorld'
```

```
1 2 * "Hello"
```

```
'HelloHello'
```



## 6) 문자열 인덱싱

3절. 기본 자료형 > 3.2. 문자형

- `문자열[index]` 형식으로 문자열에서 **지정한 위치(index)의 문자**를 뽑아낼 수 있음
- 첫 문자의 인덱스가 0
- 음수는 맨 뒤의 문자부터 의미

```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
 0   1   2   3   4   5
-6  -5  -4  -3  -2  -1
```

|   |             |
|---|-------------|
| 1 | "Python"[0] |
|   | 'p'         |
| 1 | "Python"[3] |
|   | 'h'         |

## 7) 문자열 슬라이싱

3절. 기본 자료형 > 3.2. 문자형

- `[start:stop]`를 이용하면 **부분 문자열**을 빼 낼 수 있음
  - `start` 위치의 문자는 포함하지만 `stop` 위치의 문자는 포함하지 않음
  - `[:]` 형식에서도 인덱스를 음수로 지정할 수 있음
- `[start:stop:step]` 형식은 **매 `step`번째 아이템을 추출해 줌**
  - `start`, `stop`, `step`은 생략될 수 있음
  - 만일 `start`와 `stop`이 생략되면 `[::step]` 형식이 됨
  - 이 형식은 문자열 뿐만 아니라 리스트, 튜플 등에서도 사용할 수 있음

## 7) 문자열 슬라이싱 - [*start*:*stop*]

3절. 기본 자료형 > 3.2. 문자형

- [*start*:*stop*]를 이용하면 부분 문자열을 빼 낼 수 있음
  - *start* 위치의 문자는 포함하지만 *stop* 위치의 문자는 포함하지 않음
  - [:]형식에서도 인덱스를 음수로 지정할 수 있음

```
1 str_ = 'Python'
```

```
1 str_[0:6]
```

'Python'

```
1 str_[:]
```

'Python'

```
1 str_[0:]
```

'Python'

```
1 str_[ :6]
```

'Python'

```
1 str_[-6:-1]
```

'Pytho'

```
1 str_[-6:]
```

'Python'

```
1 str_[:-1]
```

'Pytho'

```
1 str_[-1:-5]
```

''

## 8) 문자열 슬라이싱 - $[start:stop:step]$

3절. 기본 자료형 > 3.2. 문자형

- $[start:stop:step]$  형식은 매  $step$ 번째 아이템을 추출해 줌
  - $start, stop, step$ 은 생략될 수 있음
  - 만일  $start$ 와  $stop$ 이 생략되면  $[::step]$  형식이 됨
  - 이 형식은 문자열 뿐만 아니라 리스트, 튜플 등에서도 사용할 수 있음

|   |                    |
|---|--------------------|
| 1 | '0123456789' [::2] |
|---|--------------------|

'02468'

|   |                    |
|---|--------------------|
| 1 | '0123456789' [::3] |
|---|--------------------|

'0369'

|   |                     |
|---|---------------------|
| 1 | '0123456789' [::-2] |
|---|---------------------|

'97531'

|   |                      |
|---|----------------------|
| 1 | '0123456789' [2:8:2] |
|---|----------------------|

'246'

|   |                       |
|---|-----------------------|
| 1 | '0123456789' [-1::-2] |
|---|-----------------------|

'97531'

|   |                      |
|---|----------------------|
| 1 | '0123456789' [9::-2] |
|---|----------------------|

'97531'

|   |                         |
|---|-------------------------|
| 1 | '0123456789' [-1:-7:-2] |
|---|-------------------------|

'975'

## 9) raw 문자열

3절. 기본 자료형 > 3.2. 문자형

- 문자열 앞에 r을 붙여 raw 문자열을 선언해 사용
- r을 붙이면 역슬래시 문자를 해석하지 않고 남겨둠
- 탈출 문자를 사용하지 않고 역슬래시 등의 문자를 그대로 표현
- 정규표현식(Regular Expression)또는 디렉토리 경로 표현에 사용

```
1 print("Wn Wt Wr WW W' W")
```

W ' "

```
1 print(r"Wn Wt Wr WW W' W")
```

Wn Wt Wr WW W' W"

```
1 import re
2 data = "이름:홍길동, 주소:서울시,W
3 전화번호:010-2345-6789, 특징:동해번쩍서해번쩍"
4 phone_pattern = r'[Wd]{3,4}-[Wd]{4}-[Wd]{4}'
5 phone = re.findall(phone_pattern, data)
6 print(phone)
```

['010-2345-6789']

## 3.3. 논리형

3절. 기본 자료형

- 논리형(Bool)은 True 또는 False 값
- true 또는 TRUE를 논리형 값으로 사용할 수 없음
- 다음은 False로 판단되는 값들
  - None
  - False
  - 숫자 타입 0에 해당하는 것(예: 0, 0L, 0.0, 0j)
  - 빈 문자(예: "", "")
  - 빈 튜플 또는 리스트(예: (), [])
  - 빈 딕셔너리(예: {})
- True로 판별되는 경우는 False로 판별되는 경우를 제외하고 모든 경우

# 1) False로 판별되는 경우

3절. 기본 자료형 > 3.3 논리형

```
1 ▼ if 0:
2     print(True)
3 ▼ else:
4     print(False)
```

False

```
1 ▼ if 0+0j:
2     print(True)
3 ▼ else:
4     print(False)
```

False

```
1 ▼ if 0.0:
2     print(True)
3 ▼ else:
4     print(False)
```

False

```
1 ▼ if '':
2     print(True)
3 ▼ else:
4     print(False)
```

False

## 2) True로 판별되는 경우

3절. 기본 자료형 > 3.3 논리형

- False로 판별되는 경우를 제외하고 모든 경우
- 널 문자(' ')와 공백 문자(' ')는 True로 판별

```
1 ▼ if ' ':  
2     print(True)  
3 ▼ else :  
4     print(False)
```

True

```
1 ▼ if ' ':  
2     print(True)  
3 ▼ else :  
4     print(False)
```

True



# 1) 자료형 확인 - type()

3절. 기본 자료형 > 3.4 자료형 확인 및 변환

1

```
type(120)
```

int

1

```
type('hello')
```

str

1

```
type(2147483648)
```

int

1

```
type("HelloWorld")
```

str

1

```
type(3.141592)
```

float

1

```
type(True)
```

bool

1

```
type(1.23456e3)
```

float

1

```
type(False)
```

bool

## 2) 자료형 변환

3절. 기본 자료형 > 3.4 자료형 확인 및 변환

### ● int(), float(), str(), bool(), isinstance(변수명, 타입명)

|   |                           |       |   |                                |          |   |                          |           |   |                        |       |
|---|---------------------------|-------|---|--------------------------------|----------|---|--------------------------|-----------|---|------------------------|-------|
| 1 | <code>int(3.14)</code>    | 3     | 1 | <code>float(100)</code>        | 100.0    | 1 | <code>str(100)</code>    | '100'     | 1 | <code>bool(0)</code>   | False |
| 1 | <code>int(3.6)</code>     | 3     | 1 | <code>float(True)</code>       | 1.0      | 1 | <code>str(3.14)</code>   | '3.14'    | 1 | <code>bool(0.0)</code> | False |
| 1 | <code>int(True)</code>    | 1     | 1 | <code>float(False)</code>      | 0.0      | 1 | <code>str(True)</code>   | 'True'    | 1 | <code>bool('')</code>  | False |
| 1 | <code>int(False)</code>   | 0     | 1 | <code>float('3.141592')</code> | 3.141592 | 1 | <code>str(1.23e4)</code> | '12300.0' | 1 | <code>bool(10)</code>  | True  |
| 1 | <code>int('12345')</code> | 12345 | 1 | <code>float(314)</code>        | 314.0    |   |                          |           | 1 | <code>bool(3.6)</code> | True  |

## 4절. 포매팅

2장 자료형과 연산자

- 문자, 숫자, 날짜 데이터에 형식을 지정하는 것
- 이전 스타일 포매팅
  - `'%s %s' % ('one', 'two')`
- 새로운 스타일 포매팅
  - `'{} {}'.format('one', 'two')`
- 결과
  - one two

## 4.1. 문자열에 형식 지정

4절. 포매팅

- 포매팅(formatting)

- 문자, 숫자, 날짜 데이터에 형식을 지정하는 것

```
1 name = "홍길동"  
2 age = 20
```

```
1 print(name, '님의 나이는 ', age, '세입니다.', sep='')
```

홍길동님의 나이는 20세입니다.

변수와 텍스트를 번갈아 사용해서  
형식을 지정하는 것은 가독성이 떨어지고  
코드를 작성하기 불편함

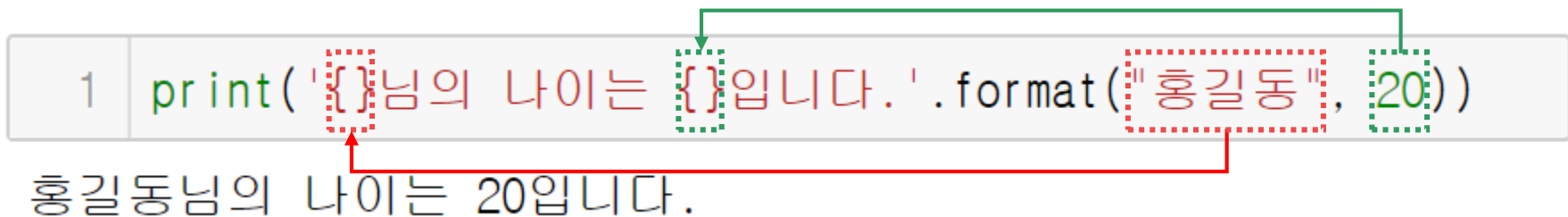
## 4.2. format()

4절. 포매팅

'{[인덱스]:[공백대체문자][정렬방법][자릿수][타입]}' .format(변수명)

### ● 구문에서...

- **인덱스**: format() 함수의 인수 중에서 해당 자리에 출력할 인수의 인덱스
- **정렬방법**: 정렬할 방법을 지정. < 기호는 왼쪽 정렬, > 기호는 오른쪽 정렬 그리고 ^ 기호는 가운데 정렬해서 출력
- **자릿수**: 변수의 값을 출력할 최대 자릿수를 지정.
- **타입**: 출력 형식을 지정. 'd'는 10진 정수, 'f'는 실수(부동소수점), 's'는 문자열을 의미. 숫자 형식 'b'는 2진수, 'o'는 8진수, 'x'는 16진수로 출력.



```
1 print('{ }님의 나이는 { }입니다.' .format("홍길동", 20))
```

홍길동님의 나이는 20입니다.

# 1) 순서 지정

4절. 포매팅 > 4.1 문자열에 형식 지정

- {}에 출력할 변수의 인덱스를 지정할 수 있음

```
1 print('출력 : {2}, {1}, {0}'.format(10, 20, 30))
```

출력 : 30, 20, 10

## 2) 숫자 출력

4절. 포매팅 > 4.1 문자열에 형식 지정

```
1 a = 12345
```

```
1 print('출력: [{}], [{}:10], [{}:3]'.format(a, a, a))
```

출력: [12345], [ 12345], [12345]

d: 10진 정수  
f: 실수  
b: 2진수  
o: 8진수  
x: 16진수

```
1 2 print('출력: [{}:d], [{}:f], [{}:b], [{}:o], [{}:x]'.format(a, a, a, a, a))
```

출력: [12345], [12345.000000], [11000000111001], [30071], [3039]

```
1 2 print('출력: [{}], [{}:f], [{}:15f], [{}:10.2f], [{}:20.10f]'.format(a, a, a, a, a))
```

출력: [12345], [12345.000000], [ 12345.000000], [ 12345.00], [12345.0000000000]

### 3) 문자열 출력

4절. 포매팅 > 4.1 문자열에 형식 지정

- 문자열의 포맷 코드는 s
- {width.precision} 형식 : {전체자릿수.출력할문자열의개수}

```
1 b = 'Hello World'
```

```
1 print('출력: [{}], [{:s}], [{:20}], [{:5}]'.format(b, b, b, b))
2
```

출력: [Hello World], [Hello World], [Hello World ], [Hello World]

```
1 print('출력: [{}], [{:.5}], [{:10.5s}]'.format(b, b, b))
```

출력: [Hello World], [Hello], [Hello ]



## 4) 정렬 방법 지정

4절. 포매팅 > 4.1 문자열에 형식 지정

- {}의 콜론(:)뒤에 '<', '>', '^'을 이용하면 정렬 상태를 지정
- '<'는 왼쪽정렬, '>'는 오른쪽 정렬, '^'는 왼쪽 정렬

```
1 a, b, c = 10, 20, 30
```

```
1 print('출력: [{:>5d}], [{:<5d}], [{:^5d}]'.format(a, b, c))
```

출력: [ 10], [20 ], [ 30 ]

```
1 ▼ print('출력: [{:>10d}], [{:<10d}], [{:^10d}]'W
2     .format(a, b, c))
```

출력: [ 10], [20 ], [ 30 ]

## 5) 공백 대체 문자

4절. 포매팅 > 4.1 문자열에 형식 지정

```
1 a, b = 10, 'Hello'
```

```
1 print("출력: {:>10}, {:*<20}, {:_<10.5s}".format(a, b, b))
```

출력: \$\$\$\$\$\$10, Hello\*\*\*\*\* , Hello\_\_\_\_\_

공백 대체 문자 직접 지정  
공백 대체문자 작성시 정렬문자 꼭

```
1 a, b = 123, -123
```

```
1 print("출력: [{:05}], [{:05}]".format(a, b))
```

출력: [00123], [-0123]

숫자 앞에 0으로 채우기

```
1 print("출력: [{:5}], [{:+5}], [{:+05}]".format(a, a, a))
```

출력: [ 123], [ +123], [+0123]

숫자 앞에 부호 붙이기

```
1 print("출력: [{:=10}], [{:=+10}], [{:=+010}]".format(a, a, a))
2 print("출력: [{:=10}], [{:=+10}], [{:=+010}]".format(b, b, b))
```

출력: [ 123], [+ 123], [+000000123]

출력: [- 123], [- 123], [-000000123]

부호를 전체 자릿수 맨 앞에 놓기

```
1 print("출력: [{:=10}], [{:=+010}], [{:=>010}]".format(a, a, a))
2 print("출력: [{:=10}], [{:=+010}], [{:=>010}]".format(b, b, b))
```

출력: [\*\*\*\*\*123], [+\*\*\*\*\*123], [\*\*\*\*\*+123]

출력: [-\*\*\*\*\*123], [-\*\*\*\*\*123], [\*\*\*\*\*-123]

부호와 공백대체문자 사용하기

## 6) 매개변수를 갖는 포맷

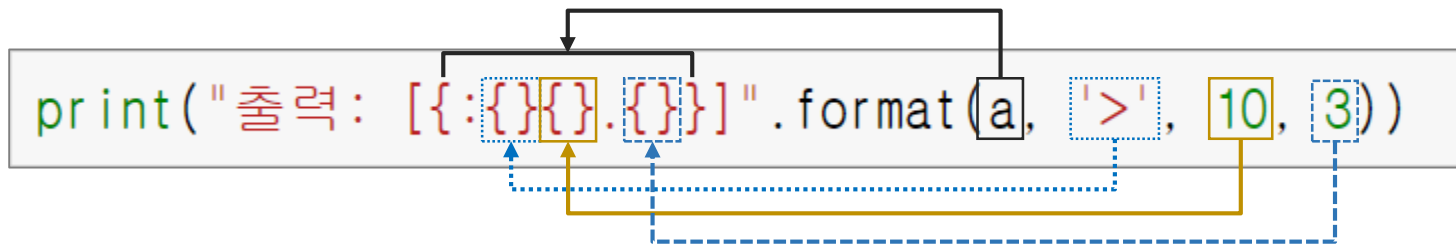
4절. 포매팅 > 4.1 문자열에 형식 지정

- {}안에 {}를 이용하면 format() 함수의 인수를 이용해 포맷 형식 지정
- format 함수의 인수에 이름을 지정할 수 있음
  - 이 경우 이름을 갖는 인수(키워드 인수)는 함수 인수 목록에서 이름이 없는 인수(위치 인수) 뒤에 와야 함

```
1 a = 2.7182818284
```

```
1 print("출력: [{:}{:}.{:}]" .format(a, '>', 10, 3))
```

출력: [ 2.72]



## 6) 매개변수를 갖는 포맷

4절. 포매팅 > 4.1 문자열에 형식 지정

```
1 print("출력: [{:}{}}{}{}].{}}]".format(a, '>', '+', 10, 3))
```

출력: [ +2.72]

```
1 print("출력: [{:}{dir}{sign}{width}.{precision}] "W
2 .format(a, dir='>', sign='+', width=10, precision=2))
```

출력: [ +2.7]

```
1 print("출력: [{:}{dir}{sign}{}.{}]"W
2 .format(a, 10, 3, dir='>', sign='+'))
```

출력: [ +2.72]

## 7) 날짜 출력

4절. 포매팅 > 4.1 문자열에 형식 지정

- %Y는 연도(Year), %m은 월(Month), %d는 일(Day)
- %H는 시간(Hour), %M(Minute)은 분, %S는 초(Second)를 의미

```
1 from datetime import datetime
```

```
1 print('{:%Y-%m-%d %H:%M}'.format(datetime(2001, 2, 3, 4, 5)))
```

2001-02-03 04:05

## 5.1. 산술 연산자

### 5절. 연산자

- 덧셈, 뺄셈, 곱셈, 나눗셈은 수학에서의 연산과 동일
- 나머지 연산자는 피젯수를 젯수로 나눈 나머지 값을 구하는 연산
- 나머지 연산의 실제 계산은 피젯수에서 젯수를 계속 빼서 피젯수의 값이 젯수의 값보다 작아질 때의 피젯수의 값이 최종적으로 나머지 연산의 값이 됨

| 연산자 | 설명  |
|-----|-----|
| +   | 덧셈  |
| -   | 뺄셈  |
| *   | 곱셈  |
| /   | 나눗셈 |
| //  | 몫   |
| %   | 나머지 |
| **  | 제곱  |

## 5.2. 대입 연산자

5절. 연산자

- 대입연산자 =은 변수에 값을 저장하기 위한 연산자
- 할당연산자라고 부르기도 함
- = 기호의 오른쪽은 값이 오거나 계산되어서 값으로 출력될 수 있는 표현식 또는 함수 호출 구문이 올 수 있음
- 복합대입연산자
  - 연산자와 대입연산자를 함께 사용한 연산자
  - +=, -=, \*=, \*\*=, /=, //=, %= 등

## 5.3. 논리 연산자

### 5절. 연산자

- & 와 and 연산자는 양쪽 항의 값이 모두 True 인 경우에만 True를 반환
- | 와 or 연산자는 양쪽 항 중에서 어느 한쪽만 True 이면 True를 반환
- ‘and’ 연산자는 False로 판별되는 첫 번째 항의 결과가 반환
- ‘or’ 연산자는 True로 판별되는 첫 번째 항의 결과가 반환
- 논리 반전은 not 연산자를 사용
  - 자바 또는 C 언어에서 사용하는 ! 부호를 논리 반전(not) 연산자로 사용 못함

| 연산자 | 설명                 |
|-----|--------------------|
| &   | AND                |
|     | OR                 |
| and | AND(short circuit) |
| or  | OR(short circuit)  |
| not | NOT                |



## 5.4. 비교 연산자

5절. 연산자

- 크기를 비교해 결과를 True 또는 False로 반환
- 문자열 타입도 가능합니다. 소문자가 대문자보다 큰 값
- 논리 타입도 크기 비교가 가능합니다. True가 False 보다 큼

| 연산자        | 설명                       |
|------------|--------------------------|
| $a < b$    | b가 a보다 크면 true           |
| $a \leq b$ | b가 a보다 크거나 같으면 true      |
| $a > b$    | a가 b보다 크면 true           |
| $a \geq b$ | a가 b보다 크거나 같으면 true      |
| $a == b$   | a와 b가 같으면 true(조건문에서 유용) |
| $a != b$   | a와 b가 같지 않으면 true        |

## 5.5. 비트 연산

5절. 연산자

### ● 숫자를 2진수로 변환하여 연산

| 연산자          | 설명  |          |          |          |         |              |
|--------------|---|----------|----------|----------|---------|--------------|
| $a \& b$     | AND 연산, 두 비트가 모두 1이면 1                    |          |          |          |         |              |
| $a   b$      | OR 연산, 두 비트중 하나 이상 1이면 1                  |          |          |          |         |              |
| $a \wedge b$ | XOR 연산, 두 비트가 같으면 0, 다르면 1                |          |          |          |         |              |
| $\sim a$     | NOT 연산, 0을 1로, 1을 0으로 변환                  |          |          |          |         |              |
| $a \gg n$    | Shift 연산, a를 n비트만큼 오른쪽으로 이동, 으로 나눈 결과와 같음 |          |          |          |         |              |
| $a \ll n$    | Shift 연산, a를 n비트만큼 왼쪽으로 이동, 을 곱한 결과와 같음   |          |          |          |         |              |
| X            | Y   | $\sim X$ | $\sim Y$ | $X \& Y$ | $X   Y$ | $X \wedge Y$ |
| 0            | 0   | 1        | 1        | 0        | 0       | 0            |
| 0            | 1   | 1        | 0        | 0        | 1       | 1            |
| 1            | 0   | 0        | 1        | 0        | 1       | 1            |
| 1            | 1   | 0        | 0        | 1        | 1       | 0            |

## 6.1. 문자열 다루기

### 6절. 문자열 다루기

| 방법   | 설명   |
|--|--|
| “ ” 또는 ‘ ’                                     | 문자열을 만듭니다.   |
| +  | 문자열을 연결합니다.  |
| len(“문자열”)                                     | 문자열의 길이를 반환합니다.  |
| [start: stop]                                  | 문자열을 start 위치부터 stop 위치까지 자릅니다. stop은 포함 안 됩니다.            |
| [start: stop: step]                            | 문자열을 start 위치부터 stop 위치까지 매 step 마다 반환합니다. stop은 포함 안 됩니다. |
| split( <i>delimiter</i> )                      | 문자열을 delimiter로 잘라 리스트로 반환합니다.                             |
| <i>delimiter</i> .join(["str1", ...])          | 문자열 리스트를 delimiter로 연결합니다.                                 |
| capitalize()                                   | 첫 문자를 대문자로, 나머지 문자를 소문자로 바꿔줍니다.                            |
| upper, lower()                                 | 문자열을 모두 대문자(upper) 또는 소문자(lower)로 바꿉니다.                    |
| startswith(), endswith()                       | 특정 문자로 시작하는지와 끝나는지를 식별해서 논리(True/False)값을 반환합니다.           |
| find(), index()                                | 특정 문자의 인덱스를 반환합니다.   |
| isalnum(), isalpha(), isnumeric(), isdecimal() | 이 문자열이 숫자인지, 문자인지 판별해 줍니다.                                 |
| replace( <i>old</i> , <i>new</i> )             | old 문자를 new 문자로 치환합니다.                                     |

## 7.1. 날짜 및 시간

7절. 날짜 다루기

```
datetime.date(year, month, day)
```

```
datetime.time(hour, minute, second, microsecond, tzinfo)
```

```
datetime.datetime(year, month, day, hour, minute, second)
```

구문에서...

- *year, month, day, hour, minute, second, microsecond* : 년, 월, 일, 시( $0 \leq \text{hour} < 24$ ), 분( $0 \leq \text{minute} < 60$ ), 초( $0 \leq \text{second} < 60$ ), 마이크로초( $0 \leq \text{microsecond} < 1000000$ )를 의미합니다.
- *tzinfo* : tzinfo 추상클래스를 상속받아 구현한 타임존 객체를 지정합니다.

## 8. 연습문제

문제

1. 이름과 나이 변수를 다음 형식으로 출력하도록 format() 함수를 이용해 형식화하세요  
[출력형식 : 홍길동님의 나이는 23살입니다]
2. 두 정수를 입력받아 두 수의 덧셈, 뺄셈, 곱셈, 나눗셈, 몫, 나머지를 출력하세요
3. 문자열의 분리하기와 합치기 기능을 이용하여 'Hello World'가 할당된 변수를 이용하여 'World Hello'로 출력하세요
4. x = 'abcdef ' 를 이용하여 'bcdefa'로 출력하세요(문자 슬라이싱이용).
5. x = 'abcdef'를 이용하여 'fedcba'로 출력하세요
6. 오늘의 온도를 섭씨온도로 입력받아 화씨 온도로 변환하는 프로그램을 작성하세요. 화씨 온도는 소수점이 있을 경우만 두번째 자리까지 출력되고, 소수점이하 값이 없는 경우 소수점을 출력하지 않도록 합니다.(다음은 섭씨와 화씨의 변환 공식입니다. C는 섭씨, F는 화씨)  
$$C = (F - 32) / 1.8$$
$$F = (C * 1.8) + 32$$

## 8. 연습문제

문제

7. 다음 중 변수 선언으로 잘못된 것을 모두 고르시오?

- ① for
- ② 10th
- ③ Student.name
- ④ \_1234

8. 다음의 코드의 실행결과는?

```
1 text = "Seoul A001 - programming with python"
2 print(text[:4]+text[-1]+text.split()[0])
3 print(text[:4], text[-1], text.split()[0])
```

9. 다음 중 파이썬 3.x 버전에서 연산식과 그 결과의 출력이 잘못된 것은?

- ① 수식 8//2 결과 4
- ② 수식 8/2 결과 4
- ③ 수식 8 \*\*2 결과 64

## 8. 연습문제

문제

10. 아래 프로그램의 결과는?

```
1 x = 5
2 y = "John"
3 print(x + y)
```

11. split()함수를 이용하여 str\_의 값 중 " 나라 " 를 출력하는 프로그램을 완성하시오

```
1 str_ = "우리 나라 대한 민국"
```