문자열

문자열 (String)

- 문자, 단어 등으로 구성된 문자들의 집합
  - 따옴표로 둘러싸여 있으면 무조건 문자열이다.
  - “hello”, ‘반갑습니다! 잘 부탁드립니다.’, “12345”, …

```
1 "hello"
```

```
'hello'
```

```
1 '반갑습니다! 잘 부탁드립니다.'
```

```
'반갑습니다! 잘 부탁드립니다.'
```

```
1 "12345"
```

```
'12345'
```

문자열 (String)

- 문자열
  - 튜플처럼, 문자열 내의 문자(항목)를 수정하거나 삭제할 수 없고 새로운 항목을 추가할 수도 없다.
  - 문자열의 내용을 수정하려면 인덱싱, 슬라이싱 및 덧셈 등을 이용하여 문자열을 새로 만들어 할당한다.

```
1  s = "Python!"
2
3  s[0] = "Q"
4
5  print(s)
```

```
TypeError                                 Tr
<ipython-input-6-de5fa920ee33> in <module>
      1 s = "Python!"
      2
----> 3 s[0] = "Q"
      4
      5 print(s)
```

```
1  s = "Python!"
2
3  s = "Q" + s[1:]
4
5  print(s)
```

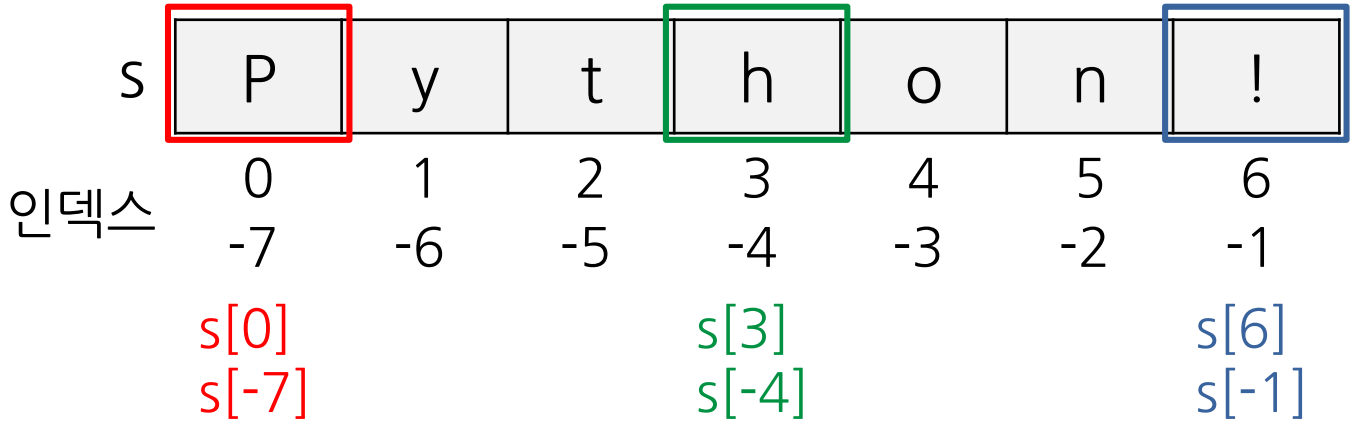
```
Qython!
```

문자열 (String)

- 문자열 내의 1개의 항목 접근하기

  문자열의 이름[항목의 인덱스]

  – 리스트 또는 튜플과 완전히 동일한 방식으로 1개의 문자(항목)을 인덱싱한다.

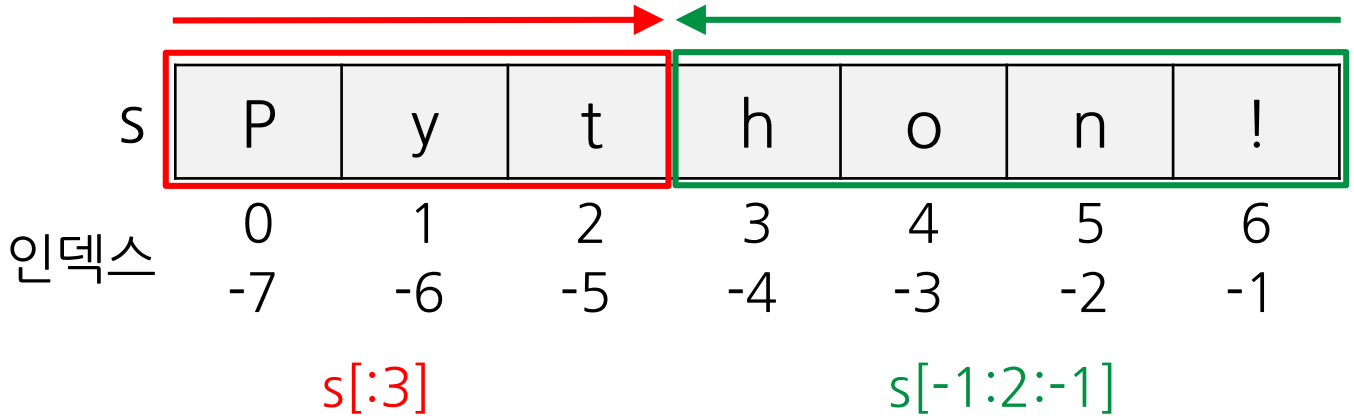


문자열 (String)

- 문자열 내의 여러 개의 항목 접근하기

  문자열의 이름[시작 인덱스:끝 인덱스:간격]

  – 리스트 또는 튜플과 완전히 동일한 방식으로 여러 개의 문자(항목)들을 슬라이싱한다.



※ 슬라이싱한 항목들은 1개의 문자열이다.

문자열 (String)

- 문자열에 특정 값이 존재하는지 확인하기
  찾으려는 값 in 문자열의 이름
  찾으려는 값 not in 문자열의 이름

- 문자열의 길이 구하기
  len(문자열의 이름)

- 문자열에 있는 항목들 중 최대값, 최소값 구하기
  max(문자열의 이름)
  min(문자열의 이름)

  ※ 문자열에 대해서 sum은 동작하지 않는다.

문자열 (String)

```
1  s1 = "aBcDeFg"
2
3  if "a" in s1:
4      print("s1의 길이 =", len(s1))
5
6  print("s1에서 가장 작은 문자 =", min(s1))
7  print("s1에서 가장 큰 문자 =", max(s1))
8  print()
9
10 s2 = s1[::2] * 2
11
12 if "B" not in s2:
13     print("s2의 길이 =", len(s2))
14
15 print("s2에서 가장 작은 문자 =", min(s2))
16 print("s2에서 가장 큰 문자 =", max(s2))
```

```
s1의 길이 = 7
s1에서 가장 작은 문자 = B
s1에서 가장 큰 문자 = g

s2의 길이 = 8
s2에서 가장 작은 문자 = a
s2에서 가장 큰 문자 = g
```

문자열 (String)

- 문자열 포맷 지정하기

  서식이 포함된 문자열.format(서식에 넣을 내용(들))

  – 대상 문자열 내에서 중괄호 '{}' 기호로 서식을 지정하고 그 서식에 맞도록 다른 내용을 대체하여 추가한다.

```
1  s = '''문자열 안에 {} 기호가 있으면 format 명령어가 담고 있는 내용이
2  {} 안으로 순서대로 대체되어 들어갑니다.'''.format(123, "python")
3
4  print(s)
```

```
문자열 안에 123 기호가 있으면 format 명령어가 담고 있는 내용이
python 안으로 순서대로 대체되어 들어갑니다.
```

```
1  s = '''문자열 내의 {0} 기호가 여러 개일 때 괄호 안에
2  인덱스 번호 {1}{0}를 기재하면 그 번호에 맞게 대체되어 들어갑니다.'''.format(123, "python")
3
4  print(s)
```

```
문자열 내의 123 기호가 여러 개일 때 괄호 안에
인덱스 번호 python123를 기재하면 그 번호에 맞게 대체되어 들어갑니다.
```

문자열 (String)

- 문자열 포맷 지정하기

  서식이 포함된 문자열.format(서식에 넣을 내용(들))

```
1  a = 123
2  b = "python"
3
4  s = '''format 명령어의 값은 {1}과 같이 값 자체도 가능하고
5  {2} 또는 {0}과 같이 변수에서 대체되는 것도 가능합니다.'''.format(a, 3.14 ,b)
6
7  print(s)
```

```
format 명령어의 값은 3.14과 같이 값 자체도 가능하고
python 또는 123과 같이 변수에서 대체되는 것도 가능합니다.
```

```
1  a = 123
2  b = "python"
3
4  s = '''format 명령어에 인덱스 대신 {a}와 같이
5  '변수명=값' 형태로 대체할 수도 {0} {1} 있습니다.'''.format(a, b, a=456)
6
7  print(s)
```

```
format 명령어에 인덱스 대신 456와 같이
'변수명=값' 형태로 대체할 수도 123 python 있습니다.
```

문자열 (String)

- 문자열 내에서 전체 자리수 지정하기
  {변수명 또는 인덱스:전체자리수}

```
1  s = "내용 {0:10}을 전체 10자리로 맞춥니다.".format("python")
2
3  print(s)
```

```
내용 python     을 전체 10자리로 맞춥니다.
```

```
1  s = "내용 {a:5}를 전체 5자리로 맞춥니다.".format(a=123)
2
3  print(s)
```

```
내용   123를 전체 5자리로 맞춥니다.
```

```
1  a = "hello"
2
3  s = "내용 {:3}와 {:10}의 자리수를 각각 맞춥니다.".format(a, 3.14)
4
5  print(s)
```

```
내용 hello와       3.14의 자리수를 각각 맞춥니다.
```

문자열 (String)

- 문자열 정렬하기 (Alignment)

  {변수명 또는 인덱스:정렬방향 전체자리수}

```
1  s = "오른쪽으로 정렬하고 {0:>10}, 전체 10자리로 맞춥니다.".format("python")
2
3  print(s)
```

```
오른쪽으로 정렬하고       python, 전체 10자리로 맞춥니다.
```

```
1  s = "왼쪽으로 정렬하고 {a:<5}, 전체 5자리로 맞춥니다.".format(a=123)
2
3  print(s)
```

```
왼쪽으로 정렬하고 123  , 전체 5자리로 맞춥니다.
```

```
1  a = "hello"
2
3  s = "가운데로 정렬하고 {:^3}, {:^10}, 각각 자리수를 맞춥니다.".format(a, 3.14)
4
5  print(s)
```

```
가운데로 정렬하고 hello,    3.14    , 각각 자리수를 맞춥니다.
```

문자열 (String)

- 문자열 공백 채우기

  {변수명 또는 인덱스:공백을채울문자 정렬방향 전체자리수}

```
1  s = "왼쪽으로 정렬하고 {0:!<10}, 빈공간을 !로 채웁니다.".format("python")
2
3  print(s)
```

왼쪽으로 정렬하고 python!!!!, 빈공간을 !로 채웁니다.

```
1  s = "오른쪽으로 정렬하고 {a:a>5}, 빈공간을 a로 채웁니다.".format(a=123)
2
3  print(s)
```

오른쪽으로 정렬하고 aa123, 빈공간을 a로 채웁니다.

```
1  a = "hello"
2
3  s = "가운데로 정렬하고 {:=^8}, 빈공간을 =로 채웁니다..".format(a)
4
5  print(s)
```

가운데로 정렬하고 =hello==, 빈공간을 =로 채웁니다..

문자열 (String)

- 문자열에서 소수점 자리수 지정하기
  {**변수명 또는 인덱스**: . . . **전체자리수**.**소수점자리수**}

```
1  s = '''숫자 {0:.4f}와 {1:0.4f}의 소수점 이하 자리수를
2  4자리까지만 표현합니다.'''.format(3.1415926, 98765.4321012345)
3
4  print(s)
```

```
숫자 3.1416와 98765.4321의 소수점 이하 자리수를
4자리까지만 표현합니다.
```

```
1  pi = 3.14159265358897
2
3  s = "숫자 {:=<10.3f}를 전체 10자리, 소수점 이하 3자리까지만 표현합니다.".format(pi)
4
5  print(s)
```

```
숫자 3.142=====를 전체 10자리, 소수점 이하 3자리까지만 표현합니다.
```

문자열 (String)

- 문자열에서 특정 항목의 개수 세기

  문자열의 이름.count(개수를 세려는 문자(열))

  – 대상 문자열에서 특정 항목이 몇 개 존재하는지 확인하여 그 개수를 되돌려 준다.

```
1  a = "12321"
2
3  b = a.count("1")
4  print(b)
5
6  c = a.count(" ")
7  print(c)
```

```
2
0
```

```
1  a = "Hello, my name is Hong Kil-Dong."
2
3  b = a.count("H")
4  print(b)
5
6  c = a.count(" ")
7  print(c)
```

```
2
5
```

문자열 (String)

- 문자열에서 특정 항목 찾기
  - ① 문자열의 이름.find(찾으려는 문자(열))
  - 대상 문자열에서 찾으려는 문자(열)이 처음으로 나오는 위치를 결과로 되돌려 준다.
  - 찾으려는 문자(열)이 존재하지 않는 경우, -1을 결과로 되돌려 준다.

```
1  a = "12321"
2
3  b = a.find("1")
4  print(b)
5
6  c = a.find("3")
7  print(c)
```

```
0
2
```

```
1  a = "Hello, my name is Hong Kil-Dong."
2
3  b = a.find("h")
4  print(b)
5
6  c = a.find(" ")
7  print(c)
```

```
-1
6
```

문자열 (String)

- 문자열에서 특정 항목 찾기
  - ② 문자열의 이름.index(찾으려는 문자(열))
  - 대상 문자열에서 찾으려는 문자(열)이 처음으로 나오는 위치를 결과로 되돌려 준다.
  - 찾으려는 문자(열)이 존재하지 않는 경우, 오류가 발생 한다.

```
1  a = "12321"
2
3  b = a.index("1")
4  print(b)
5
6  c = a.index("3")
7  print(c)
```

```
0
2
```

```
1  a = "Hello, my name is Hong Kil-Dong."
2
3  b = a.index(" ")
4  print(b)
5
6  c = a.index("h")
7  print(c)
```

```
6

---------------------------------------------------------------------------
ValueError
```

문자열 (String)

- 문자열의 일부 내용을 바꾸기

  문자열의 이름.replace(원래문자(열), 바꿀문자(열))

  – 대상 문자열에서 특정 문자(열)을 다른 문자(열)로 변경한 새로운 문자열을 결과로 되돌려 준다.

```
1  s1 = "Hello, my name is Hong Kil-Dong."
2
3  s2 = s1.replace("Hong", "Kim")
4
5  print(s1)
6  print(s2)
```

```
Hello, my name is Hong Kil-Dong.
Hello, my name is Kim Kil-Dong.
```

문자열 (String)

- 문자열에서 대/소문자 바꾸기

  문자열의 이름.upper( )

  – 대상 문자열 안에 존재하는 모든 소문자들을 대문자로 바꾼 새로운 문자열을 결과로 되돌려 준다.

  문자열의 이름.lower( )

  – 대상 문자열 안에 존재하는 모든 대문자들을 소문자로 바꾼 새로운 문자열을 결과로 되돌려 준다.

```
1  s1 = "Hello, my name is Hong Kil-Dong."
2
3  s2 = s1.upper()
4  s3 = s1.lower()
```

```
1  print(s1)
2  print(s2)
3  print(s3)
```

```
Hello, my name is Hong Kil-Dong.
HELLO, MY NAME IS HONG KIL-DONG.
hello, my name is hong kil-dong.
```

문자열 (String)

- 문자열에서 왼쪽 끝 부분의 문자(열) 지우기

  문자열의 이름.lstrip(지우려는 문자(열))

  – 대상 문자열의 맨 왼쪽에 있는 특정 문자(열)을 지운 새로운 문자열을 결과로 되돌려 준다.

  – 특정 문자(열)을 지정하지 않으면 연속된 공백을 지운다.

```
1  s1 = "Hello, my name is Hong Kil-Dong."
2
3  s2 = s1.lstrip("Hell")
4
5  print(s1)
6  print(s2)
```

```
Hello, my name is Hong Kil-Dong.
o, my name is Hong Kil-Dong.
```

```
1  s1 = "          python"
2
3  s2 = s1.lstrip()
4
5  print(s1)
6  print(s2)
```

```
          python
python
```

문자열 (String)

- 문자열에서 오른쪽 끝 부분의 문자(열) 지우기

  문자열의 이름.rstrip(지우려는 문자(열))

  – 대상 문자열의 맨 오른쪽에 있는 특정 문자(열)을 지운 새로운 문자열을 결과로 되돌려 준다.

  – 특정 문자(열)을 지정하지 않으면 연속된 공백을 지운다.

```
1  s1 = "Hello, my name is Hong Kil-Dong."
2
3  s2 = s1.rstrip(".")
4
5  print(s1)
6  print(s2)
```

```
Hello, my name is Hong Kil-Dong.
Hello, my name is Hong Kil-Dong
```

```
1  s1 = "python        "
2
3  s2 = s1.rstrip()
4
5  print(s1)
6  print(s2)
```

```
python
python
```

문자열 (String)

- 문자열에서 양쪽 끝 부분의 문자(열) 지우기

  문자열의 이름.strip(지우려는 문자(열))

  – 대상 문자열의 양쪽 끝에 있는 특정 문자(열)을 지운 새로운 문자열을 결과로 되돌려 준다.

  – 특정 문자(열)을 지정하지 않으면 연속된 공백을 지운다.

```
1  s1 = "!!!WELCOME! Nice to see you!!!"
2
3  s2 = s1.strip("!")
4
5  print(s1)
6  print(s2)
```

```
!!!WELCOME! Nice to see you!!!
WELCOME! Nice to see you
```

```
1  s1 = "    python      "
2
3  s2 = s1.strip()
4
5  print(s1)
6  print(s2)
```

```
    python
python
```

문자열 (String)

- 문자열을 여러 개의 문자열들로 나누기

  문자열의 이름.split(나누는 기준이 되는 문자(열))

  – 대상 문자열을 특정 문자(열)을 기준으로 분리하여 리스트로 만들어 그 리스트를 결과로 되돌려 준다.

  – 특정 문자(열)을 지정하지 않으면 공백을 기준으로 나눈다.

```
1  s = "TAG:#홍길동#조선셀럽#호부호형"
2
3  t = s.split("#")
4
5  print(s)
6  print(t)
```

```
TAG:#홍길동#조선셀럽#호부호형
['TAG:', '홍길동', '조선셀럽', '호부호형']
```

```
1  s = "Hello, my name is Hong Kil-Dong."
2
3  t = s.split()
4
5  print(s)
6  print(t)
```

```
Hello, my name is Hong Kil-Dong.
['Hello,', 'my', 'name', 'is', 'Hong', 'Kil-Dong.']
```

문자열 (String)

- 여러 개의 자료를 하나의 문자열로 결합하기

  결합할 기준이 되는 문자(열).join(결합할 문자열 자료들)

  – 특정 문자(열)을 기준으로 여러 개의 문자열을 결합하여 하나의 문자열로 되돌려 준다.

  – 결합 대상이 되는 자료는 문자열이거나, 항목이 문자열인 리스트, 튜플, 집합이 될 수 있다.

```
1  a = "python"
2
3  s = ",".join(a)
4
5  print(s)
```

```
p,y,t,h,o,n
```

```
1  b = ["a", "b", "c"]
2  delimiter = "+++"
3
4  s = delimiter.join(b)
5
6  print(s)
```

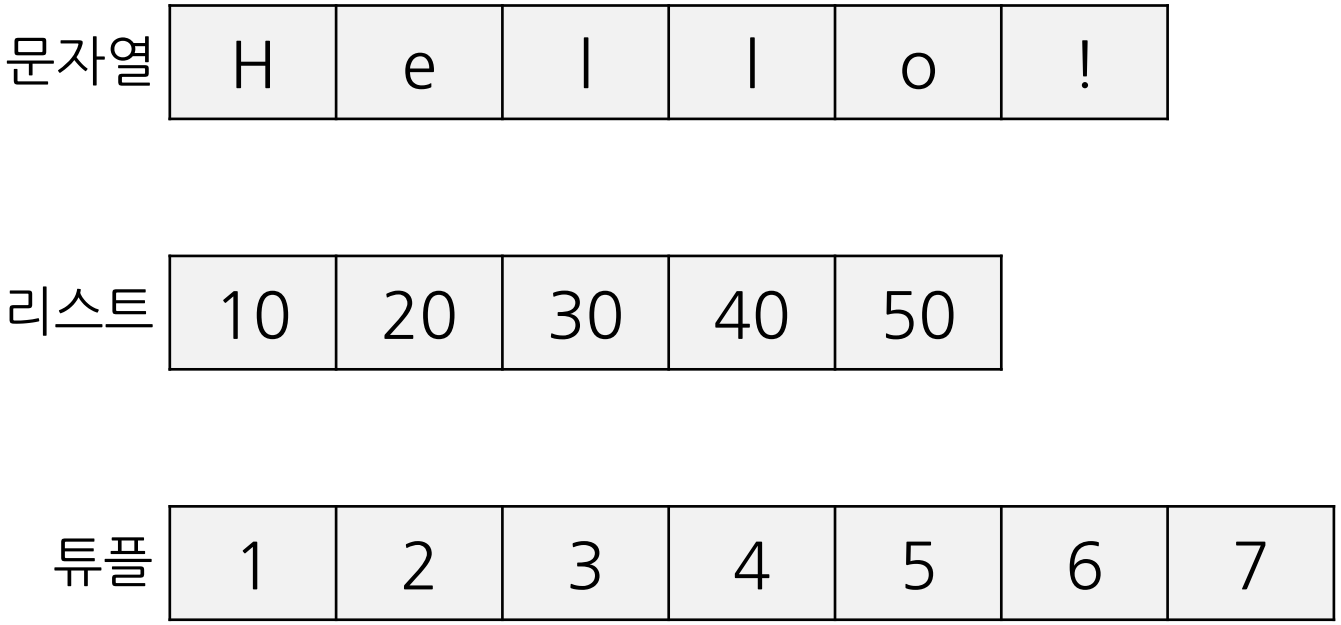
```
a+++b+++c
```

```
1  s = " ".join(("1","2","3","4","5"))
2
3  print(s)
```

```
1 2 3 4 5
```

순서가 있는 자료형들

- 순차 (Sequence) 자료
  - 여러 개의 항목 값들이 순서대로 모여 있는 자료형을 순차 자료라고 한다.
  - 문자열, 리스트, 튜플은 모두 순차 자료에 해당한다.



자료 구조의 할당 및 비교 시 주의점

동일 객체 비교 연산자 (Identity of Object)

- 2개의 자료가 동일한 객체인지 검사하는 연산자

연산자	기능	설명
is	같다	2개의 자료가 동일한 객체인 경우 참이 된다.
is not	같지 않다	2개의 자료가 동일한 객체가 아닌 경우 참이 된다.

※ 기본 논리 연산자 (Equality of Value)

연산자	기능	설명
==	같다	2개의 자료가 동일한 값인 경우 참이 된다.
!=	같지 않다	2개의 자료가 동일한 값이 아닌 경우 참이 된다.

동일 객체 비교 연산자 (Identity of Object)

- 2개의 자료가 동일한 객체인지 검사하는 연산자

```
1  a = 123
2  b = 123
3
4  print(a == b)
5  print(a is b)
```

```
True
True
```

```
1  a = "python"
2  b = "python"
3
4  print(a == b)
5  print(a is b)
```

```
True
True
```

```
1  a = 123
2  b = 3.141592
3
4  print(a == b)
5  print(a is b)
```

```
False
False
```

```
1  a = "python"
2  b = "python!"
3
4  print(a == b)
5  print(a is b)
```

```
False
False
```

동일 객체 비교 연산자 (Identity of Object)

- 2개의 자료가 동일한 객체인지 검사하는 연산자

```
1  a = "Hello, my name is Hong Kil-Dong."
2  b = "Hello, my name is Hong Kil-Dong."
3
4  print(a == b)
5  print(a is b)
```

```
True
False
```

```
1  a = [1, 2, 3]
2  b = [1, 2, 3]
3
4  print(a == b)
5  print(a is b)
```

```
True
False
```

```
1  a = "Hello, my name is Hong Kil-Dong."
2  b = a
3
4  print(a == b)
5  print(a is b)
```

```
True
True
```

```
1  a = [1, 2, 3]
2  b = a
3
4  print(a == b)
5  print(a is b)
```

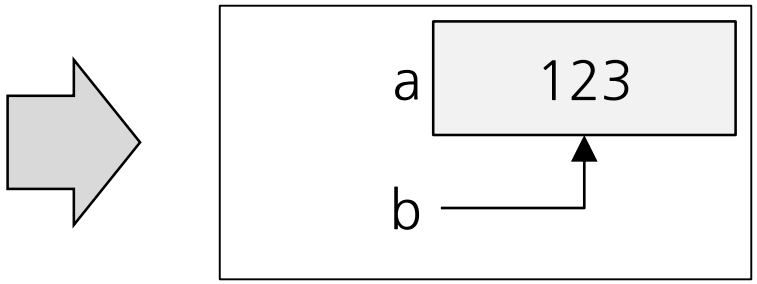
```
True
True
```

동일 객체 비교 연산자 (Identity of Object)

- 변수 할당을 할 때 발생하는 과정
  - 수치형 및 짧은 문자열을 변수에 할당하는 경우에 그와 동일한 자료(변수)가 이미 존재한다면, 이미 존재하는 변수를 가리키기만 한다.

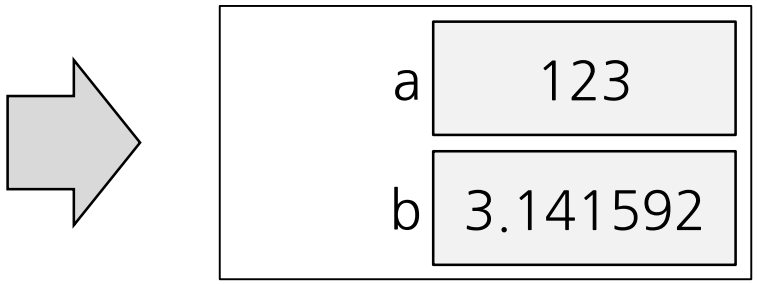
```
1  a = 123
2  b = 123
3
4  print(a == b)
5  print(a is b)
```

```
True
True
```




```
1  a = 123
2  b = 3.141592
3
4  print(a == b)
5  print(a is b)
```

```
False
False
```

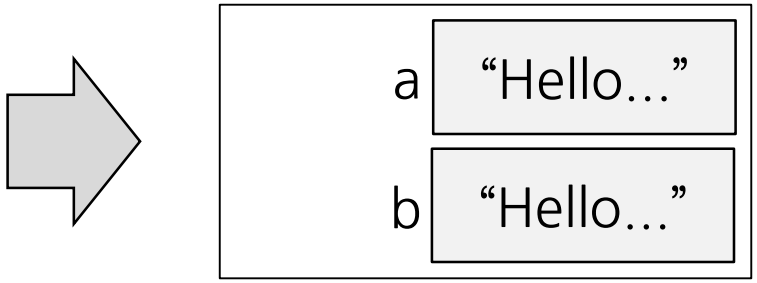


동일 객체 비교 연산자 (Identity of Object)

- 변수 할당을 할 때 발생하는 과정
  - 리스트, 튜플과 같은 복잡한 자료 구조이거나 길이가 긴 문자열을 변수에 할당하는 경우에는 동일한 자료(변수)가 이미 존재하더라도 새로 만든다.

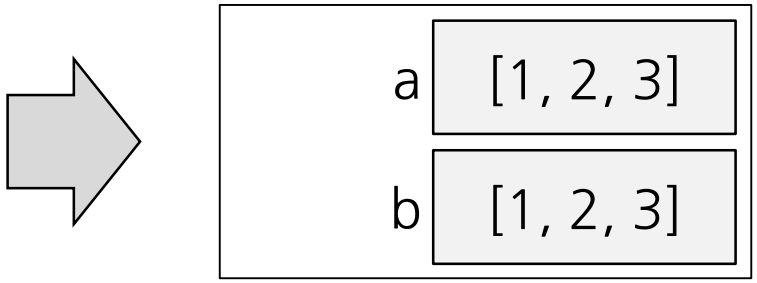
```
1  a = "Hello, my
2  b = "Hello, my
3
4  print(a == b)
5  print(a is b)
```

```
True
False
```




```
1  a = [1, 2, 3]
2  b = [1, 2, 3]
3
4  print(a == b)
5  print(a is b)
```

```
True
False
```

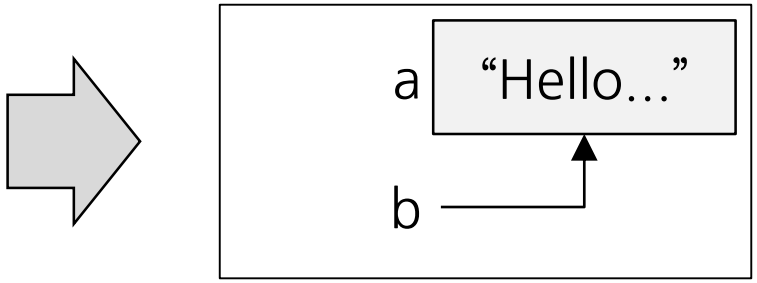


동일 객체 비교 연산자 (Identity of Object)

- 변수 할당을 할 때 발생하는 과정
  - 리스트, 튜플과 같은 복잡한 자료 구조이거나 길이가 긴 문자열이 들어 있는 변수를 명시적으로 직접 할당하게 되면 이미 존재하는 변수를 가리키기만 한다.

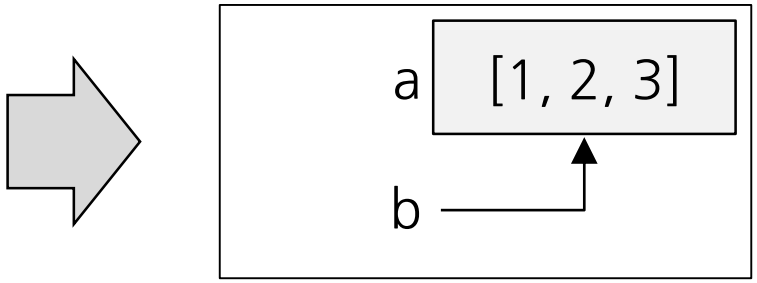
```
1  a = "Hello, my
2  b = a
3
4  print(a == b)
5  print(a is b)
```

```
True
True
```




```
1  a = [1, 2, 3]
2  b = a
3
4  print(a == b)
5  print(a is b)
```

```
True
True
```



동일 객체 비교 연산자 (Identity of Object)

- 변수 할당 및 비교를 할 때 주의할 점
  - 리스트, 튜플과 같은 자료 구조가 들어 있는 변수를 직접 다른 변수에 할당하지 않는 것이 바람직하다.
  - 의도한 것이 아니라면, 변수들끼리 객체 비교 연산자를 이용하여 비교하지 않는 것이 바람직하다.

```
1  a = [1, 2, 3]
2  b = a
3
4  a.pop()
5
6  print(a)
7  print(b)
```

```
[1, 2]
[1, 2]
```

```
1  t = []
2
3  if t is not None:
4      print("t는 존재합니다.")
```

```
t는 존재합니다.
```