

# 자료형과 문자열

# 자료형과 기본 자료형

- **자료형** (data type)
  - 자료의 형식
  - **type()** 함수로 확인

```
>>> print(type("안녕하세요"))  
<class 'str'>  
>>> print(type(273))  
<class 'int'>
```

- **str**: 문자열
- **int**: 정수

# 문자열 만들기

- 문자열 (string)
  - 따옴표로 둘러싸 입력하는, 글자가 나열된 것

"Hello"    'String'    '안녕하세요'    "Hello Python Programming"

```
# 하나만 출력합니다.
print("# 하나만 출력합니다.")
print("Hello Python Programming...!")
print()

# 여러 개를 출력합니다.
print("# 여러 개를 출력합니다.")
print(10, 20, 30, 40, 50)
print("안녕하세요", "저의", "이름은", "윤인성입니다!")
...
```

문자열

문자열

문자열

# 문자열 만들기

- 큰따옴표로 문자열 만들기

```
>>> print("안녕하세요")  
안녕하세요
```

- 작은따옴표로 문자열 만들기

```
>>> print('안녕하세요')  
안녕하세요
```

# 문자열 만들기

- 이스케이프 문자 (escape character)
  - 역슬래시 기호와 함께 조합해서 사용하는 특수한 문자
  - \ “ : 큰따옴표를 의미
  - \ ‘ : 작은따옴표를 의미

```
>>> print("\"안녕하세요\"라고 말했습니다")
"안녕하세요"라고 말했습니다
>>> print("'배가 고픈다'라고 생각했습니다')
'배가 고픈다'라고 생각했습니다
```

- \n : 줄바꿈 의미
- \t : 탭 의미

# 문자열 만들기

- \\: 역슬래시를 의미

```
>>> print("\\ \\ \\ \\")  
\\ \\ \\
```

## • 여러 줄 문자열 만들기

- \n 사용

```
>>> print("동해물과 백두산이 마르고 닳도록\n하느님이 보우하사 우리나라 만세\n무궁화 삼천리 화려강\n산 대한사람\n대한으로 길이 보전하세")  
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세
```

# 문자열 만들기

- 여러 줄 문자열 기능 활용: 큰따옴표 혹은 작은따옴표를 세 번 반복

```
>>> print("""동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세""")  
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세
```

# 문자열 만들기

- 줄바꿈 없이 문자열 만들기
  - \ 기호 사용

```
>>> print("""  
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세  
""")
```

```
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세
```

→ 위 아래로 의도하지 않은 줄바꿈이 들어갑니다.



# 문자열 만들기

- 줄 뒤에 \ 붙여서 코드 쉽게 보기 위한 줄바꿈이며, 실질적 줄바꿈 아님을 나타냄

```
>>> print("""\
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산 대한사람
대한으로 길이 보전하세\
""")
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산 대한사람
대한으로 길이 보전하세
```

→ 줄을 바꿔 출력하지 않겠다고 선언합니다.


# 문자열 연산자

- 두 문자열 연결하여 새로운 문자열 만들어냄

```
>>> print("안녕" + "하세요")
안녕하세요
>>> print("안녕하세요" + "!")
안녕하세요!
```

- 문자열과 숫자 사이에는 사용할 수 없음

```
>>> print("안녕하세요" + 1)
```

 오류

TypeError: can only concatenate str (not "int") to str

- 문자열은 문자끼리, 숫자는 숫자끼리 연결
- 문자열과 숫자 연결하여 연산하려면 큰따옴표 붙여  
문자열로 인식하게 함

# 문자열 연산자

- 문자열 반복 연산자 : \*
- 문자열을 숫자와 \* 연산자로 연결

```
>>> print("안녕하세요" * 3)  
안녕하세요안녕하세요안녕하세요
```

```
>>> print(3 * "안녕하세요")  
안녕하세요안녕하세요안녕하세요
```

# 문자열 연산자

- 문자 선택 연산자 (인덱싱) : []

- 문자열 내부의 문자 하나를 선택
- 대괄호 안에 선택할 문자의 위치를 지정

- 인덱스 (index)

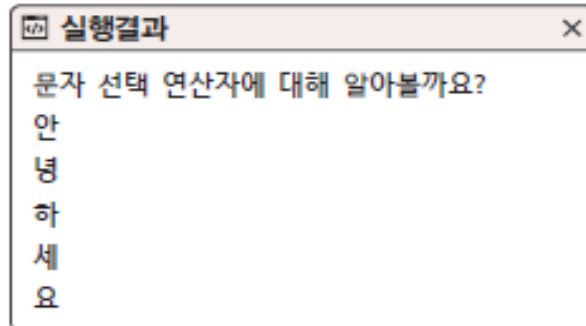
- 제로 인덱스 (zero index) : 숫자를 0부터 셈
- 원 인덱스 (one index) : 숫자를 1부터 셈
- 파이썬은 제로 인덱스 유형 사용

안	녕	하	세	요
[0]	[1]	[2]	[3]	[4]

# 문자열 연산자

- 예시

```
01 print("문자 선택 연산자에 대해 알아보까요?")
02 print("안녕하세요"[0])
03 print("안녕하세요"[1])
04 print("안녕하세요"[2])
05 print("안녕하세요"[3])
06 print("안녕하세요"[4])
```



# 문자열 연산자

- 문자를 거꾸로 출력하려는 경우
  - 대괄호 안 숫자를 음수로 입력

안	녕	하	세	요
[-5]	[-4]	[-3]	[-2]	[-1]

```
01 print("문자를 뒤에서부터 선택해 볼까요?")
02 print("안녕하세요"[-1])
03 print("안녕하세요"[-2])
04 print("안녕하세요"[-3])
05 print("안녕하세요"[-4])
06 print("안녕하세요"[-5])
```

실행결과

문자를 뒤에서부터 선택해 볼까요?  
요  
세  
하  
녕  
안

# 문자열 연산자

- **문자열 범위 선택 연산자** (슬라이싱) : [:]

- 문자열의 특정 범위를 선택
- 대괄호 안에 범위 구분 위치를 콜론으로 구분

```
>>> print("안녕하세요"[1:4])  
녕하세
```

- 마지막 숫자 포함
- 마지막 숫자 포함하지 않음
  - 파이썬에서 적용

안	녕	하	세	요
[0]	[1]	[2]	[3]	[4]

# 문자열 연산자

- 예시

```
>>> print("안녕하세요"[0:2])
안녕
>>> print("안녕하세요"[1:3])
녕하
>>> print("안녕하세요"[2:4])
하세
```

- 대괄호 안에 넣는 숫자 둘 중 하나를 생략하는 경우
  - 뒤의 값 생략 : n번째부터 끝의 문자까지
  - 앞의 값 생략 : 0번째부터 뒤의 숫자 n번째 앞의 문자까지

[1:]  
[:3]

```
>>> print("안녕하세요"[1:])
녕하세요
>>> print("안녕하세요"[:3])
안녕하
```



# 문자열 연산자

- **인덱싱** (indexing)
  - [] 기호 이용해 문자열의 특정 위치에 있는 문자 참조하는 것
- **슬라이싱** (slicing)
  - [:] 기호 이용해 문자열 일부를 추출하는 것
  - 문자열 선택 연산자로 슬라이스해도 원본은 변하지 않음에 주의

```
>>> hello = "안녕하세요" → ❶  
>>> print(hello[0:2]) → ❷  
안녕  
>>> hello → ❸  
'안녕하세요'
```

# 문자열 연산자

- **IndexError** (index out of range) 예외
  - 리스트/문자열 수를 넘는 요소/글자 선택할 경우 발생

```
>>> print("안녕하세요"[10])
```

❗ 오류

→ 파이썬 IDLE 에디터에서 실행했을 때 나타나는 내용으로 에디터마다 다르게 나타납니다.

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>

print("안녕하세요"[10])

**IndexError: string index out of range** → IndexError 예외가 발생했어요.

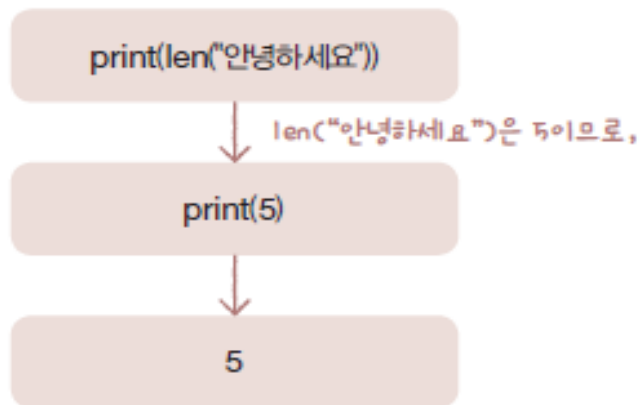
# 문자열의 길이 구하기

- `len()` 함수

- 문자열 길이 구할 때 사용
- 괄호 내부에 문자열 넣으면 문자열의 문자 개수 세어 줌

```
>>> print(len("안녕하세요"))  
5
```

- 중첩된 구조의 함수는 괄호 안쪽부터 먼저 실행



# 숫자형

- 정수형

- 소수점이 없는 숫자
- 0, 1, 273, -52
- 정수 (integer)

- 실수형

- 소수점이 있는 숫자
- 0.0, 52.273, -1.2
- 실수 (floating point, 부동 소수점)

# 숫자의 종류

- 숫자를 만들기 위해서는 단순히 숫자 입력하면 됨

```
>>> print(273)
273
>>> print(52.273)
52.273
```

- type() 함수로 소수점 없는 숫자와 있는 숫자를 출력

```
>>> print(type(52))
<class 'int'>
>>> print(type(52.273))
<class 'float'>
```

- Int : 정수
- Float : 부동 소수점 (실수)

# 숫자 연산자

- 사칙 연산자 : +, -, \*, /
  - 덧셈, 뺄셈, 곱셈, 나눗셈

연산자	설명	구문	연산자	설명	구문
+	덧셈 연산자	숫자+숫자	*	곱셈 연산자	숫자*숫자
-	뺄셈 연산자	숫자-숫자	/	나눗셈 연산자	숫자/숫자

```
>>> print("5 + 7 =", 5 + 7)
5 + 7 = 12
>>> print("5 - 7 =", 5 - 7)
5 - 7 = -2
>>> print("5 * 7 =", 5 * 7)
5 * 7 = 35
>>> print("5 / 7 =", 5 / 7)
5 / 7 = 0.7142857142857143
```

# 숫자 연산자

- 정수 나누기 연산자: //

- 숫자를 나누고 소수점 이하 자릿수 삭제한 후 정수 부분만 남김

```
>>> print("3 / 2 =", 3 / 2)
3 / 2 = 1.5
>>> print("3 // 2 =", 3 // 2)
3 // 2 = 1
```

- 나머지 연산자 : %

- A를 B로 나누었을 때의 나머지를 구함

```
>>> print("5 % 2 =", 5 % 2)
5 % 2 = 1
```



# 숫자 연산자


- **제곱 연산자 : \*\***
  - 숫자를 제곱함

```
>>> print("2 ** 1 =", 2 ** 1)
2 ** 1 = 2
>>> print("2 ** 2 =", 2 ** 2)
2 ** 2 = 4
>>> print("2 ** 3 =", 2 ** 3)
2 ** 3 = 8
>>> print("2 ** 4 =", 2 ** 4)
2 ** 4 = 16
```

# 연산자의 우선순위

- **TypeError 예외**
  - 서로 다른 자료를 연산할 경우

```
>>> string = "문자열"  
>>> number = 273  
>>> string + number
```

 오류

```
Traceback (most recent call last):  
  File "<pyshell#3>", line 1, in <module>  
    string+number
```

`TypeError: can only concatenate str (not "int") to str` → **TypeError 예외가 발생했어요.**

# 변수와 입력

# 변수 만들기/사용하기

- 앞 예시에서 입력한 pi는 숫자 자료에 이름 붙인 것이기 때문에 숫자 연산 모두 수행할 수 있음

```
>>> pi = 3.14159265
>>> pi + 2
5.14159265
>>> pi - 2
1.1415926500000002
>>> pi * 2
6.2831853
>>> pi / 2
1.570796325
>>> pi % 2
1.1415926500000002
>>> pi * pi
9.869604378534024
```

# 변수 만들기/사용하기

- pi는 숫자 자료이므로 숫자와 문자열 연산은 불가능

```
pi + "문자열"
```

- 예시 – 원의 둘레와 넓이 구하기

```
01  # 변수 선언과 할당
02  pi = 3.14159265
03  r = 10
04
05  # 변수 참조
06  print("원주율 =", pi)
07  print("반지름 =", r)
08  print("원의 둘레 =", 2*pi*r)      # 원의 둘레
09  print("원의 넓이 =", pi*r*r)     # 원의 넓이
```

실행결과

```
원주율 = 3.14159265
반지름 = 10
원의 둘레 = 62.831853
원의 넓이 = 314.159265
```

# 복합 대입 연산자

- 복합 대입 연산자

- 기본 연산자와 = 연산자 함께 사용해 구성

```
a += 10
```

연산자 이름	설명
+=	숫자 덧셈 후 대입
-=	숫자 뺄셈 후 대입
*=	숫자 곱셈 후 대입
/=	숫자 나눗셈 후 대입
%=	숫자의 나머지를 구한 후 대입
**=	숫자 제곱 후 대입

# 사용자 입력 : input()

- input() 함수

- 명령 프롬프트에서 사용자로부터 데이터 입력받을 때 사용

- input() 함수로 사용자 입력받기

- 프롬프트 함수 : input 함수 괄호 안에 입력한 내용

```
>>> input("인사말을 입력하세요> ")
```

- 블록 (block) : 프로그램이 실행 중 잠시 멈추는 것

인사말을 입력하세요> | → 입력 대기를 알려주는 커서입니다. 커서는 프로그램에 따라 모양이 다를 수 있습니다.

- 명령 프롬프트에서 글자 입력 후 [Enter] 클릭

```
인사말을 입력하세요> 안녕하세요 [Enter]
```

```
'안녕하세요'
```

# 사용자 입력 : input()

- input() 함수의 입력 자료형
  - type() 함수로 자료형 알아보기

```
>>> print(type(string))  
<class 'str'>
```

```
>>> number = input("숫자를 입력하세요> ")  
숫자를 입력하세요> 12345   
>>> print(number)  
12345
```

```
>>> print(type(number))  
<class 'str'>
```

- input() 함수는 사용자가 무엇을 입력해도 결과는 무조건 문자열 자료형



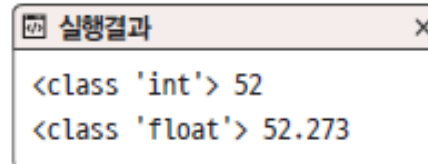
# 문자열을 숫자로 바꾸기

- **캐스트** (cast)
  - input() 함수의 입력 자료형은 항상 문자열이므로 입력받은 문자열을 숫자 연산에 활용하기 위해 숫자로 변환
- **int()** 함수
  - 문자열을 int 자료형으로 변환.
- **float()** 함수
  - 문자열을 float 자료형으로 변환

# 문자열을 숫자로 바꾸기

- 예시 – int() 함수와 float() 함수 활용하기

```
01 output_a = int("52")
02 output_b = float("52.273")
03
04 print(type(output_a), output_a)
05 print(type(output_b), output_b)
```



실행결과

```
<class 'int'> 52
<class 'float'> 52.273
```

- 예시 – int() 함수와 float 함수 조합하기

```
01 input_a = float(input("첫 번째 숫자> "))
02 input_b = float(input("두 번째 숫자> "))
03
04 print("덧셈 결과:", input_a + input_b)
05 print("뺄셈 결과:", input_a - input_b)
06 print("곱셈 결과:", input_a * input_b)
07 print("나눗셈 결과:", input_a / input_b)
```



실행결과


```
첫 번째 숫자> 273 Enter
두 번째 숫자> 52 Enter
덧셈 결과: 325.0
뺄셈 결과: 221.0
곱셈 결과: 14196.0
나눗셈 결과: 5.25
```

# 문자열을 숫자로 바꾸기

- ValueError 예외

- 변환할 수 없는 것을 변환하려 할 경우
- 숫자가 아닌 것을 숫자로 변환하려 할 경우

```
int("안녕하세요")  
float("안녕하세요")
```


 오류

```
Traceback (most recent call last):  
  File "intconvert.py", line 2, in <module>  
    int_a = int(string_a)  
ValueError: invalid literal for int() with base 10: '안녕하세요'
```

# 문자열을 숫자로 바꾸기

- 소수점이 있는 숫자 형식의 문자열을 int() 함수로 변환하려 할 때

```
int("52.273")
```

 오류

```
Traceback (most recent call last):
```

```
  File "intconvert.py", line 2, in <module>
```

```
    int_a = int(string_a)
```

```
ValueError: invalid literal for int() with base 10: '52.273'
```

# 숫자를 문자열로 바꾸기

- `str()` 함수
  - 숫자를 문자열로 변환

`str(다른 자료형)`

```
01 output_a = str(52)
02 output_b = str(52.273)
03 print(type(output_a), output_a)
04 print(type(output_b), output_b)
```

실행결과

```
<class 'str'> 52
<class 'str'> 52.273
```

# 숫자와 문자열을 다루는 함수

# 문자열의 format() 함수

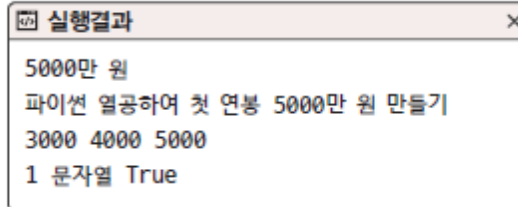
- **format() 함수로 숫자를 문자열로 변환**
  - 중괄호 포함한 문자열 뒤에 마침표 찍고 format() 함수 사용하되, 중괄호 개수와 format 함수 안 매개변수의 개수는 반드시 같아야 함
  - 문자열의 중괄호 기호가 format() 함수 괄호 안의 매개변수로 차례로 대치되면서 숫자가 문자열이 됨

```
"{}".format(10)  
 "{} {}".format(10, 20)  
 "{} {} {} {} {}".format(101, 202, 303, 404, 505)
```

# 문자열의 format() 함수

- 예시 – format() 함수의 다양한 형태

```
01  # format() 함수로 숫자를 문자열로 변환하기
02  format_a = "{}만 원".format(5000)
03  format_b = "파이썬 열공하여 첫 연봉 {}만 원 만들기 ".format(5000)
04  format_c = "{} {} {}".format(3000, 4000, 5000)
05  format_d = "{} {} {}".format(1, "문자열", True)
06
07  # 출력하기
08  print(format_a)
09  print(format_b)
10  print(format_c)
11  print(format_d)
```



실행결과

```
5000만 원
파이썬 열공하여 첫 연봉 5000만 원 만들기
3000 4000 5000
1 문자열 True
```

- format\_a : 중괄호 옆에 다른 문자열 넣음
- format\_b : 중괄호 앞뒤로 다른 문자열 넣음
- format\_c : 매개변수 여러 개 넣음



# 문자열의 format() 함수

- IndexError 예외

- 중괄호 기호의 개수가 format() 함수의 매개변수 개수보다 많은 경우

```
>>> "{} {}".format(1, 2, 3, 4, 5)
'1 2'
>>> "{} {} {}".format(1, 2)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    "{} {} {}".format(1, 2)
IndexError: tuple index out of range
```

# 대소문자 바꾸기 : upper()와 lower()

- upper() 함수
  - 문자의 알파벳을 대문자로 바꿈
- lower() 함수
  - 문자의 알파벳을 소문자로 바꿈

```
>>> a = "Hello Python Programming...!"  
>>> a.upper()  
'HELLO PYTHON PROGRAMMING...!'
```

```
>>> a.lower()  
'hello python programming...!'
```

# 문자열 양옆의 공백 제거하기: strip()

- strip() 함수

- 문자열 양옆의 공백을 제거

- lstrip() 함수

- 왼쪽의 공백을 제거

- rstrip() 함수

- 오른쪽의 공백을 제거

## 문자열의 구성 파악하기 : isOO()

- 문자열이 소문자로만, 알파벳으로만, 혹은 숫자로만 구성되어 있는지 확인
  - `isalnum()`: 문자열이 알파벳 또는 숫자로만 구성되어 있는지 확인합니다.
  - `isalpha()`: 문자열이 알파벳으로만 구성되어 있는지 확인합니다.
  - `isidentifier()`: 문자열이 식별자로 사용할 수 있는 것인지 확인합니다.
  - `isdecimal()`: 문자열이 정수 형태인지 확인합니다.
  - `isdigit()`: 문자열이 숫자로 인식될 수 있는 것인지 확인합니다.
  - `isspace()`: 문자열이 공백으로만 구성되어 있는지 확인합니다.
  - `islower()`: 문자열이 소문자로만 구성되어 있는지 확인합니다.
  - `isupper()`: 문자열이 대문자로만 구성되어 있는지 확인합니다.

# 문자열의 구성 파악하기 : isOO()

- **불** (boolean)
  - 출력이 **T** rue 혹은 **F** alse로 나오는 것

```
>>> print("TrainA10".isalnum())
True
>>> print("10".isdigit())
True
```

# 문자열 찾기: find()

- find()
  - 왼쪽부터 찾아서 처음 등장하는 위치 찾기

```
>>> output_a = "안녕안녕하세요".find("안녕")  
>>> print(output_a)  
0
```

# 문자열과 in 연산자

- In 연산자

- 문자열 내부에 어떤 문자열이 있는지 확인할 때 사용
- 결과는 True(맞다), False(아니다)로 출력

```
>>> print("안녕" in "안녕하세요")  
True
```

```
>>> print("잘자" in "안녕하세요")  
False
```

# 문자열 자르기 : split()

- split() 함수
  - 문자열을 특정한 문자로 자름

```
>>> a = "10 20 30 40 50".split(" ")
>>> print(a)
['10', '20', '30', '40', '50']
```

- 실행 결과는 리스트 (list)로 출력



- **format() 함수** : 숫자와 문자열을 다양한 형태로 출력
- **upper() 및 lower() 함수** : 문자열의 알파벳을 대문자 혹은 소문자로 변경
- **strip() 함수** : 문자열 양옆의 공백 제거
- **find() 함수** : 문자열 내부에 특정 문자가 어디에 위치하는지 찾을 때 사용
- **in 연산자** : 문자열 내부에 어떤 문자열이 있는지 확인할 때 사용
- **split() 함수** : 문자열을 특정한 문자로 자를 때 사용