Python 기초 **Chap1:** 변수 개념 이해와 연산들

Table of contents

1 변수 개념 이해하기

파이썬에서 변수는 데이터를 저장하는 컨테이너입니다. 변수를 사용하면 코드 내에서 데이터를 더 쉽게 관리할 수 있습니다.

1.1 변수 선언과 할당

파이썬에서 변수를 선언하고 값을 할당하는 것은 매우 간단합니다. 변수 이름을 정하고 값을 할당하려면 = 연산자를 사용합니다.

```
# 변수에 값을 할당
number = 10
greeting = "안녕하세요!"
```

1.2 변수 사용하기

변수에 값을 할당한 후에는, 변수 이름을 통해 해당 값을 언제든지 참조할 수 있습니다.

```
{r}
# 변수 값을 출력
print(number)
print(greeting)
```

이 예제에서 number 변수는 정수 10을, `greeting` 변수는 문자열 “안녕하세요!"를 각각 저장하고 있습니다. `print()` 함수를 사용하여 이들 변수의 내용을 콘솔에 출력합니다.

1.2.a 변수의 동적 타이핑

파이썬은 동적 타입 언어입니다. 이는 변수의 데이터 타입을 미리 선언할 필요가 없다는 의미입니다. 변수의 타입은 할당된 값에 따라 자동으로 결정됩니다.

```
{r}
# 변수 타입 변경
number = "이제 문자열입니다"
print(number)
```

1.3 Python에서 변수명 짓기

파이썬에서 변수를 선언할 때는 몇 가지 중요한 규칙과 관례를 따라야 합니다. 이러한 규칙들은 코드의 가독성과 유지 보수성을 높이는 데 기여합니다.

1.3.a 변수명 규칙

파이썬에서 변수명을 지을 때 따라야 하는 기본적인 규칙들입니다.

1. 변수명은 알파벳(대소문자 구분), 숫자, 밑줄(_)로 구성할 수 있으나, 숫자로 시작할 수는 없습니다.
2. 파이썬은 대소문자를 구분합니다. 예를 들어, `data`와 `Data`는 서로 다른 변수입니다.
3. 파이썬의 예약어는 변수명으로 사용할 수 없습니다. 예: `if`, `for`, `class`, 등.
4. 변수명은 그 용도나 저장된 데이터를 명확히 설명할 수 있도록 의미 있는 이름을 사용하는 것이 좋습니다.

1.3.a.a 좋은 변수명 예제

적절한 변수명은 코드의 이해를 돕고, 나중에 코드를 유지보수할 때 큰 도움이 됩니다.

- `user_age` : 사용자의 나이를 나타냅니다.
- `total_price` : 총 가격을 나타냅니다.
- `num_items` : 항목 수를 나타냅니다.
- `is_active` : 활성 상태 여부를 나타냅니다.

1.3.b 변수명 스타일

파이썬 커뮤니티에서 권장하는 몇 가지 변수명 스타일입니다.

- 스네이크 케이스**(snake_case)** : 모든 글자를 소문자로 하고 단어 사이에 밑줄을 사용합니다. 함수와 변수명에 주로 사용됩니다.
- 카멜 케이스**(CamelCase)** : 각 단어의 첫 글자를 대문자로 시작합니다. 클래스 이름에 주로 사용됩니다.

이러한 규칙을 따르는 것은 코드를 보다 효율적으로 작성하고, 다른 개발자들과의 협업 시에도 이해하기 쉬운 코드를 유지하는 데 도움이 됩니다.

2 기본 계산관련 연산자 학습하기

파이썬에서는 다양한 종류의 연산자를 제공하여 데이터 처리와 계산을 용이하게 합니다. 기본적인 산술 연산자부터 시작해 보겠습니다.

2.1 변수 할당 및 기본 산술 연산자

먼저, 두 변수 `a`와 `b`에 각각 숫자를 할당하고, 이를 사용하여 기본적인 산술 연산을 수행해 보겠습니다.

```
# 변수 할당
a = 10
b = 3.3
```

```
# 기본 산술 연산자
print("a + b =", a + b)  # 덧셈
print("a - b =", a - b)  # 뺄셈
print("a * b =", a * b)  # 곱셈
print("a / b =", a / b)  # 나눗셈
print("a % b =", a % b)  # 나머지
print("a // b =", a // b)  # 몫
print("a ** b =", a ** b)  # 거듭제곱
```

```
a + b = 13.3
a - b = 6.7
a * b = 33.0
a / b = 3.0303030303030303
a % b = 0.10000000000000053
a // b = 3.0
a ** b = 1995.2623149688789
```

2.1.a 연산자 설명

1. 덧셈 **(+)**: 두 숫자를 더합니다.
2. 뺄셈 **(-)**: 첫 번째 숫자에서 두 번째 숫자를 뺍니다.
3. 곱셈 **(*)**: 두 숫자를 곱합니다.
4. 나눗셈 **(/)**: 첫 번째 숫자를 두 번째 숫자로 나누고, 결과는 부동 소수점 수입니다.
5. 나머지 **(%)**: 첫 번째 숫자를 두 번째 숫자로 나눈 후의 나머지를 반환합니다.
6. 거듭제곱 **(**)**: 첫 번째 숫자를 두 번째 숫자만큼 거듭제곱합니다.
7. 정수 나눗셈 **(//)**: 첫 번째 숫자를 두 번째 숫자로 나눈 후의 몫을 정수로 반환합니다.

이러한 연산자들은 일반적인 수학 연산을 수행할 때 매우 유용하며, 파이썬 프로그래밍에서 자주 사용됩니다.

3 비교관련 연산자 학습하기

비교 연산자는 두 값의 관계를 평가하는 데 사용됩니다. 이 결과는 조건문과 반복문에서 중요한 역할을 합니다. 파이썬에서 사용되는 비교 연산자들과 그 의미를 아래 표에서 확인할 수 있습니다.

연산자	설명
==	동등. 두 값이 같음
!=	부등. 두 값이 다름
<	미만. 왼쪽 값이 오른쪽 값보다 작음
>	초과. 왼쪽 값이 오른쪽 값보다 큼
<=	이하. 왼쪽 값이 오른쪽 값보다 작거나 같음
>=	이상. 왼쪽 값이 오른쪽 값보다 크거나 같음

이 연산자들은 수학적 비교를 기반으로 하며, 논리적 조건 평가에 사용됩니다. 이어지는 섹션에서는 이 연산자들을 실제 코드 예제와 함께 더 자세히 살펴보겠습니다.

3.1 비교 연산자의 종류

파이썬에서 사용되는 주요 비교 연산자는 다음과 같습니다:

```
# 동등 비교
a = 10
b = 20
print("a == b:", a == b)  # a와 b가 같은지 비교

# 부등 비교
print("a != b:", a != b)  # a와 b가 다른지 비교

# 크기 비교
print("a < b:", a < b)   # a가 b보다 작은지 비교
print("a > b:", a > b)   # a가 b보다 큰지 비교

# 크거나 같음, 작거나 같음
print("a <= b:", a <= b)  # a가 b보다 작거나 같은지 비교
print("a >= b:", a >= b)  # a가 b보다 크거나 같은지 비교
```

```
a == b: False
a != b: True
a < b: True
a > b: False
a <= b: True
a >= b: False
```

3.2 비교 연산자의 사용 예

비교 연산자는 변수 또는 표현식의 값을 평가하여 조건문에서 매우 유용하게 사용됩니다. 예를 들어, 사용자 입력값의 검증, 데이터의 필터링, 특정 조건에 따른 처리 등에 사용됩니다.

```python
# 사용자 나이 검증 예제
user_age = 25
is_adult = user_age >= 18
print("성인입니까?", is_adult)
```

```
성인입니까? True
```

이 예제에서는 사용자의 나이가 성인 기준인 18세 이상인지를 검사합니다. 이를 통해 `True` 또는 `False` 결과를 얻어 프로그램에서 다양한 로직을 구현할 수 있습니다.

비교 연산자는 프로그램에서 조건을 설정하고, 데이터를 분류하며, 특정 조건에 따라 다른 작업을 수행하게 하는 등의 중요한 역할을 합니다. 이들 연산자의 정확한 이해와 사용은 프로그램의 효율성과 정확성을 높이는 데 기여합니다.

4 논리 관련 연산자 학습하기

논리 연산자는 주로 불리언(참 또는 거짓) 값을 조작하는 데 사용됩니다. 이 연산자들은 복잡한 조건문을 구성할 때 매우 유용하게 사용됩니다.

- '불리언(Boolean)'은 수학자 조지 불(George Boole)의 이름에서 유래되었으며, 컴퓨터 과학에서는 참(True) 또는 거짓(False)의 두 가지 값만을 가질 수 있는 데이터 타입을 지칭합니다.

4.1 논리 연산자의 종류

아래 표는 파이썬에서 사용되는 주요 논리 연산자들과 그 의미를 설명합니다.

연산자	설명
`and`	두 조건이 모두 참일 때 참
`or`	두 조건 중 하나라도 참일 때 참
`not`	조건의 불리언 값을 반전 (참이면 거짓, 거짓이면 참)

4.1.a 논리 연산자 사용 예제

이제 이 연산자들을 어떻게 실제 코드에서 사용하는지 살펴보겠습니다.

```python
# 논리 연산자 예제
a = True
b = False

# and 연산자
print("a and b:", a and b)  # False, a와 b 둘 다 참이어야 참 반환
```

```
# or 연산자
print("a or b:", a or b)    # True, a와 b 중 하나라도 참이면 참 반환

# not 연산자
print("not a:", not a)      # False, a의 반대 불리언 값 반환
```

```
a and b: False
a or b: True
not a: False
```

위 예제에서 a는 True, b는 False로 정의되었습니다. and 연산자는 두 조건이 모두 참일 때만 참을 반환하며, or 연산자는 두 조건 중 하나라도 참이면 참을 반환합니다. not 연산자는 조건의 불리언 값을 반전시킵니다.

논리 연산자를 사용하는 것은 프로그램의 로직을 제어하는 데 매우 중요합니다. 복잡한 조건을 평가하거나, 특정 조건에 따라 다른 동작을 수행하도록 프로그램을 설계할 때 필수적으로 사용됩니다.

5 복합 대입 연산자 학습하기

복합 대입 연산자는 값을 연산하고 그 결과를 같은 변수에 할당하는 축약된 방법을 제공합니다. 이 연산자들은 코드를 더 간결하게 만들고, 자주 사용되는 연산을 더 효율적으로 처리할 수 있게 돕습니다.

5.1 복합 대입 연산자의 종류

아래 표는 파이썬에서 사용되는 주요 복합 대입 연산자들과 그 의미, 사용 예를 설명합니다.

- 연사자= 형식임을 기억하면 좋습니다.

연산자	설명	사용 예	풀어서 쓰기
+=	왼쪽 변수에 오른쪽 값을 더하고 결과를 할당	a += 10	a = a + 10
-=	왼쪽 변수에서 오른쪽 값을 빼고 결과를 할당	a -= 10	a = a - 10
*=	왼쪽 변수에 오른쪽 값을 곱하고 결과를 할당	a *= 10	a = a * 10
/=	왼쪽 변수를 오른쪽 값으로 나누고 결과를 할당	a /= 10	a = a / 10
%=	왼쪽 변수를 오른쪽 값으로 나눈 나머지를 할당	a %= 10	a = a % 10
**=	왼쪽 변수를 오른쪽 값의 거듭제곱 후 결과를 할당	a **= 10	a = a ** 10
//=	왼쪽 변수를 오른쪽 값으로 나눈 몫을 할당	a //= 10	a = a // 10

5.2 복합 대입 연산자 사용 예제

이제 이 연산자들을 어떻게 실제 코드에서 사용하는지 살펴보겠습니다.

```
# 복합 대입 연산자 예제
a = 100

a += 10
print("a += 10:", a)  # a = a + 10

a -= 20
print("a -= 20:", a)  # a = a - 20

a *= 2
print("a *= 2:", a)  # a = a * 2

a /= 2
print("a /= 2:", a)  # a = a / 2

a %= 14
print("a %= 14:", a)  # a = a % 3

a **= 2
print("a **= 2:", a)  # a = a ** 2

a //= 2
print("a //= 2:", a)  # a = a // 2
```

```
a += 10: 110
a -= 20: 90
a *= 2: 180
a /= 2: 90.0
a %= 14: 6.0
a **= 2: 36.0
a //= 2: 18.0
```

위 예제에서 각 연산자는 변수 a의 값을 변경하고, 해당 연산을 수행한 결과를 다시 a에 할당합니다. 이 방식은 특히 반복되는 계산에서 변수를 업데이트할 때 매우 유용합니다.

복합 대입 연산자를 사용하면 코드를 더 간결하고 읽기 쉽게 만들 수 있습니다. 이 연산자들을 적절히 활용하면 프로그램의 효율성을 높이고, 오류 가능성을 줄이는 데 도움이 됩니다.

5.3 연산자 적용 우선순위

파이썬에서 연산자 우선순위는 표현식 내에서 연산자가 평가되는 순서를 결정합니다. 이 우선순위를 이해하는 것은 복잡한 표현식을 올바르게 계산하고 프로그래밍 오류를 방지하는 데 중요합니다.

5.3.a 연산자 우선순위 표

아래 표는 파이썬에서 사용되는 주요 연산자들의 우선순위를 높은 것부터 낮은 것까지 나열하며, 각 연산자의 의미를 추가적인 칼럼으로 설명합니다.

우선순위	연산자	설명
1	(), [], {}	괄호: 그룹화 및 우선순위 지정
2	**	지수 연산: 거듭제곱 계산
3	+x, -x, ~x	단항 연산자: 각각 양수, 음수, 비트 NOT 연산
4	*, /, //, %	곱셈, 나눗셈, 정수 나눗셈, 나머지 연산
5	+, -	이항 연산자: 덧셈과 뺄셈
6	<<, >>	비트 시프트: 비트를 왼쪽 혹은 오른쪽으로 이동
7	&	비트 AND: 비트 단위 AND 연산
8	^	비트 XOR: 비트 단위 XOR 연산
9	|	비트 OR: 비트 단위 OR 연산
10	==, !=, <, <=, >, >=	비교 연산자: 등식 및 부등식 평가
11	not	논리 NOT: 불리언 반전
12	and	논리 AND: 불리언 AND 연산
13	or	논리 OR: 불리언 OR 연산

5.3.b 연산자 우선순위 예제

이제 연산자 우선순위가 어떻게 적용되는지 실제 예제를 통해 살펴보겠습니다.

```
# 연산자 우선순위 예제
result = 10 + 2 * 3 ** 2 / 6 - (4 + 2) ** 2
print("Result of the expression:", result)
```

```
Result of the expression: -23.0
```

위 식에서 연산자 우선순위에 따라 다음과 같은 순서로 계산됩니다:

1. 괄호 내부의 표현식 `(4 + 2)` 계산
2. 지수 연산 `3 ** 2`
3. 곱셈 `2 * 9`
4. 나눗셈 `18 / 6`
5. 괄호 결과의 지수 연산 `(6) ** 2`

6. 덧셈과 뺄셈은 왼쪽에서 오른쪽으로 계산 `10 + 3 - 36`

연산자 우선순위를 이해하는 것은 복잡한 표현식을 올바르게 계산하는 데 필수적입니다. 표현식을 작성할 때 괄호를 사용하여 우선순위를 명시적으로 지정하면 의도하지 않은 오류를 방지할 수 있습니다.

5.4 문자열에 대한 + 연산자 적용

파이썬에서 문자열은 매우 유연하게 다룰 수 있는 데이터 타입 중 하나입니다. 여기서는 문자열 변수를 만들고, 이를 사용하는 + 연산자에 대해 알아보겠습니다.

- 문자열을 다룰 때 + 연산자는 두 문자열을 연결하는 데 사용됩니다. 이를 문자열 연결(concatenation)이라고 합니다.

```
# 문자열 변수 할당
str1 = "Hello, "
str2 = "world!"

# 문자열 연결
result = str1 + str2
print("Concatenated string:", result)
```

```
Concatenated string: Hello, world!
```

5.4.a 문자열과 숫자의 덧셈

파이썬에서는 문자열과 숫자를 직접 더하려고 하면 에러가 발생합니다. 이는 데이터 타입이 서로 다르기 때문에 발생하는 현상입니다.

```
# 문자열과 숫자의 덧셈 시도
number = 123
new_result = str1 + number
```

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

위의 예제에서 `str1`은 문자열이고, `number`는 정수입니다. 이 두 타입을 + 연산자로 더하려고 할 때 파이썬은 `TypeError`를 발생시키고, 연산을 수행할 수 없음을 알려줍니다.

5.5 문자열에 대한 * 연산자 적용

이전 섹션에서는 문자열의 + 연산자를 사용한 연결에 대해 배웠습니다. 이번에는 문자열에서 곱셈을 나타내는 * 연산자의 사용을 알아보겠습니다.

파이썬에서 * 연산자는 문자열을 반복하는 데 사용됩니다. 이 연산자를 사용하면 지정된 횟수만큼 문자열을 반복하여 새로운 문자열을 생성할 수 있습니다.

```python
# 문자열 변수 할당
str1 = "Hello! "

# 문자열 반복
repeated_str = str1 * 3
print("Repeated string:", repeated_str)
```

```
Repeated string: Hello! Hello! Hello!
```

위의 예제에서 `str1` 문자열이 3번 반복되어 `Hello! Hello! Hello!`라는 새로운 문자열을 생성합니다. 이 방식은 특정 패턴이나 메시지를 여러 번 출력하고자 할 때 유용합니다.

5.5.a 문자열과 숫자의 곱셈

문자열과 숫자의 곱셈은 오직 정수와의 곱셈만 가능합니다. 실수나 다른 타입의 데이터와의 곱셈은 타입 에러를 발생시키게 됩니다.

```python
float_str = str1 * 2.5
print("Error:", e)
```

```
TypeError: can't multiply sequence by non-int of type 'float'
```

위 코드에서 `str1 * 2.5`는 `TypeError`를 발생시킵니다. 문자열은 오직 정수와 곱해질 수 있다는 것을 기억해주세요!

문자열 연산은 파이썬에서 매우 자주 사용되는 기능입니다. 문자열을 다룰 때는 데이터 타입에 주의하여 적절한 연산을 선택하는 것이 중요합니다. 문자열과 숫자를 결합하려면 숫자를 문자열로 변환하는 등의 추가 작업이 필요합니다.

6 심화 학습 (Optional)

6.1 단항 연산자 학습하기

파이썬에서는 여러 단항 연산자를 사용하여 특정 연산을 수행할 수 있습니다. 이들 연산자는 각각 피연산자에 대해 특정 작업을 수행하며, 이 문서에서는 +x, -x, ~x에 대해 자세히 설명합니다.

6.1.a 단항 연산자의 종류 및 기능

1. **+x (단항 덧셈 연산자)**:
   - 이 연산자는 변수의 값을 변경하지 않고, 원래의 값(양수)을 유지합니다. 코드 내에서 명시적으로 양의 값을 강조할 때 사용될 수 있습니다.
2. **-x (단항 부정 연산자)**:

- 피연산자의 부호를 반전시킵니다. 양수는 음수로, 음수는 양수로 만듭니다. 이는 수학적인 부정을 수행하는 데 사용됩니다.

3. **~x (비트 NOT 연산자)**:
   - 피연산자의 모든 비트를 반전시킵니다(0은 1로, 1은 0으로). 이 연산자는 주로 저수준 프로그래밍에서 비트 레벨의 데이터 조작에 사용됩니다.

6.1.b 단항 연산자 사용 예제

아래의 코드는 각 단항 연산자의 사용법을 보여줍니다.

```
x = 5
print("Original x:", x)
print("+x:", +x)  # 출력: 5
print("-x:", -x)  # 출력: -5
print("~x:", ~x)  # 출력: -6 (5의 모든 비트를 반전시킨 값)
```

```
Original x: 5
+x: 5
-x: -5
~x: -6
```

위 예제에서 +x는 변수 x의 원래 값을 그대로 출력합니다. -x는 x의 부호를 반전시켜 출력하며, ~x는 x의 비트를 반전시킨 결과를 출력합니다.

단항 연산자는 파이썬에서 간단한 연산을 수행하는 데 유용합니다. 특히, 비트 연산자 ~는 저수준 조작이 필요할 때 사용될 수 있으며, 이러한 연산자의 올바른 이해와 사용은 프로그램의 정확성과 효율성을 높이는 데 기여할 수 있습니다.

6.2 비트 연산자 학습하기

비트 연산자는 정수형 데이터의 비트를 직접 조작하는데 사용됩니다. 이 연산자들은 특히 하드웨어 수준의 작업, 암호화, 퍼포먼스 최적화 등에 유용하게 사용됩니다.

6.2.a 비트 연산자의 종류 및 기능

1. 비트 시프트 연산자 **(<<, >>)**:
   - << (비트 왼쪽 시프트): 모든 비트를 왼쪽으로 지정된 수만큼 이동시키며, 새로운 비트는 0으로 채워집니다. 왼쪽 시프트는 값에 2의 지정된 거듭제곱을 곱하는 효과를 가집니다.
   - >> (비트 오른쪽 시프트): 모든 비트를 오른쪽으로 지정된 수만큼 이동시키며, 새로운 비트는 최상위 비트와 같게 채워집니다(부호를 유지). 오른쪽 시프트는 값에 2의 지정된 거듭제곱으로 나누는 효과를 가집니다.
2. 비트 **AND** 연산자 **(&)**:

- 두 피연산자의 비트를 비교하여, 둘 다 1이면 결과의 해당 위치에 1을 설정합니다. 이 연산은 비트 마스킹에 유용하게 사용됩니다.

3. 비트 **XOR** 연산자 **(^)**:
   - 두 피연산자의 비트를 비교하여, 서로 다를 경우(하나만 1일 경우) 결과의 해당 위치에 1을 설정합니다. 이 연산은 값의 토글이 필요할 때 사용될 수 있습니다.
4. 비트 **OR** 연산자 **(|)**:
   - 두 피연산자의 비트 중 하나라도 1이면 결과의 해당 위치에 1을 설정합니다. 이 연산은 비트 설정에 사용됩니다.

6.3 비트 표현 방식 이해하기

아래의 코드는 각 숫자를 비트 표현(이진수)로 표현하는 함수입니다.

```
# 비트 연산자 예제
x = 4  # 4는 이진수로 0100
y = 5  # 5는 이진수로 0101

# 비트 표현 확인
bin(x)
bin(y)
```

```
'0b101'
```

파이썬에서 `bin()` 함수를 사용하면 정수의 이진수 표현을 반환할 수 있습니다. 이 함수는 이진수 표현을 문자열 형식으로 제공하며, 모든 이진수 문자열은 `0b` 접두사로 시작합니다.

또한, `bin()` 함수로 생성된 이진수 표현은 앞쪽의 불필요한 0들을 포함하지 않습니다. 이는 값에 영향을 미치지 않으며, 표현을 간결하게 하기 위한 것입니다.

6.3.a 이진수 포맷팅

이진수를 특정 길이로 표현하려면, 문자열 포매팅을 사용하여 이진수 앞에 0을 추가할 수 있습니다. 예를 들어, 숫자를 4비트 이진수로 포맷팅하려면 다음과 같이 할 수 있습니다:

```
binary_4bit = f'{x:04b}'  # 4비트 길이의 이진수로 포맷
print("이진수 표현:", binary_4bit)  # 출력: 0100
```

```
이진수 표현: 0100
```

이 방법을 통해 필요한 비트 수를 정확하게 조절하여 표현할 수 있습니다. 이는 데이터의 표현을 일관되게 유지하고, 비트 연산을 수행할 때 명확성을 제공합니다.

6.3.b 비트 연산자 사용법 실습

아래의 코드는 각 비트 연산자의 사용법을 보여줍니다.

```
# 비트 시프트
left_shift = x << 2  # 010000, 이진수로 16
right_shift = y >> 1  # 0010, 이진수로 2

# 비트 AND
bit_and = x & y  # 0100 & 0101 = 0100, 이진수로 4

# 비트 XOR
bit_xor = x ^ y  # 0100 ^ 0101 = 0001, 이진수로 1

# 비트 OR
bit_or = x | y  # 0100 | 0101 = 0101, 이진수로 5

print("Left shift:", left_shift)
print("Right shift:", right_shift)
print("Bit AND:", bit_and)
print("Bit XOR:", bit_xor)
print("Bit OR:", bit_or)
```

```
Left shift: 16
Right shift: 2
Bit AND: 4
Bit XOR: 1
Bit OR: 5
```

비트 연산자들은 직접 비트 레벨에서 데이터를 조작할 수 있게 해주어, 높은 수준의 데이터 처리가 필요할 때 매우 유용합니다. 이들을 통해 데이터 압축, 암호화, 하드웨어 인터페이스 제어 등 다양한 저수준 프로그래밍 작업을 수행할 수 있습니다.