[1-3. 기본 문법 및 데이터 타입 소개1]

1. 파이썬의 기본 문법

1-1. 들여쓰기

파이썬에서 들여쓰기는 코드의 블록을 구분하는 매우 중요한 역할을 합니다. 많은 프로그래밍 언어들이 코드 블록을 나타내기 위해 중괄호({})를 사용하는 반면, 파이썬은 들여쓰기를 사용하여 코드의 구조를 나타냅니다. 이러한 특징은 파이썬 코드의 가독성을 높이는 데 크게 기여합니다.

1-1-1. 들여쓰기의 기본 규칙:

- 같은 블록의 코드는 동일한 수준으로 들여써야 합니다. 예를 들어, if문의 조건 다음에 오는 코드 블록은 모두 같은 수준으로 들여써야 합니다.
- 들여쓰기 수준이 변경되면 새로운 블록이 시작된 것으로 간주됩니다. 이는 함수 정의, 클래스 정의, 조건문(if), 반복문(for, while) 등 사양한 구조에서 블록을 정의하는 데 사용됩니다.
- 하나의 블록 내에서는 들여쓰기 방식을 일관되게 유지해야 합니다. 즉, 공백과 탭을 섞어 사용하지 않아야 합니다. (PEP 8, 파이썬 코드 스타일 가이드에 따르면 공백 4칸을 표준으로 권장합니다.) ##### 1-1-2. 들여쓰기의 중요성:
- 구문 오류 방지: 잘못된 들여쓰기는 구문 오류를 발생시키고, 프로그램이 실행되지 않게 합니다.
- 코드의 구조화 : 들여쓰기를 통해 코드의 구조가 명확해지며, 다른 개발자가 코드를 이해하기 쉬워집니다.

1-1-3. 들여쓰기 예제:

```
if True:
    print("이 줄은 들여쓰기 되었습니다.")
    if False:
        print("이 줄은 두 번 들여쓰기 되었습니다.")
    print("이 줄도 첫 번째 if 문의 일부입니다.")
```

- 위 예제에서 볼 수 있듯이, 'if' 조건문 블록 내의 'print'문은 들여쓰기 되어 있으며, 이를 통해 어떤 'print'문이 어떤 조건문에 속하는지 명확하게 구분됩니다. 들여쓰기는 파이썬 코드 작성 시 반드시 지켜야 하는 규칙 중 하나입니다.

1-2. 변수

1-2-1. 변수 개요

- 변수는 데이터를 저장하는 메모리의 위치를 가리키는 이름입니다. 파이썬에서 변수를 선언하고 변수에 값을 할당할 때는 "=" 연산자를 사용합니다. #### 1-2-2. 변수명의 규칙

- **문자, 숫자, 밑줄**: 변수명은 문자(a-z, A-Z), 밑줄(_), 또는 숫자(0-9)로 구성될 수 있습니다. 단, 변수명의 첫 글자로 숫자가 올 수는 없습니다.
- **대소문자 구분**: 파이썬은 대소문자를 구분합니다. 예를 들어, 'variable'과 'Variable'은 서로 다른 변수로 인식됩니다.
- **예약어 사용 금지**: 파이썬에는 특정 기능을 위해 예약된 키워드들이 있으며, 이들을 변수명으로 사용할 수 없습니다. 예를 들어, 'False', 'True', 'if', 'else', 'for', 'while', 'and', 'or', 'not' 등이 있습니다.
- 밑줄 시작: 변수명이 밑줄로 시작하는 경우가 있는데, 일반적으로 밑줄(_) 하나로 시작하는 변수는 "내부적으로 사용되는" 또는 "비공개(private)" 변수를 의미합니다. "__"(밑줄 두개)로 시작하는 이름은 클래스의 특별한 메서드나 속성에 사용됩니다. #### 1-2-3. 변수명의 관례
- **소문자 사용**: 일반 변수는 소문자로 시작하는 것이 일반적입니다.
- **가독성**: 변수명은 그 목적이나 사용하는 데이터를 명확하게 설명할 수 있도록 선택해야 합니다. 예를 들어, 'student_name' 또는 'total_price'와 같이 의미가 명확한 이름을 사용하는 것이 좋습니다.
- **밑줄로 단어 구분**: 파이썬에서는 변수명 내에서 단어를 구분할 때 밑줄을 사용하는 스네이크 케이스(snake_case)를 권장합니다. 예: number_of_students
- **클래스 이름**: 클래스 이름을 지을 때는 각 단어의 첫 글자를 대문자로 시작하는 카멜 케이스(CamelCase)를 사용하는 것이 관례입니다. 예: MyClass

1-2-4. 변수명 예시

```
# 좋은 변수명 예시
user_age = 30
total_price = 199.99
is_valid = True

# 나쁜 변수명 예시 (가독성이 떨어짐)
a = 30          # 'a'가 무엇을 의미하는지 명확하지 않음
tp = 199.99     # 'tp'가 무엇을 의미하는지 명확하지 않음
v = True        # 'v'가 무엇을 의미하는지 명확하지 않음
```

1-2-5. 동적 타이핑

- 파이썬은 동적 타이핑 언어로, 변수에 특정 타입을 사전에 선언할 필요가 없습니다. 변수에 값을 할당할 때 해당 값의 타입에 따라 변수의 타입이 결정됩니다. 또한, 같은 변수에 다른 타입의 값을 재할당하는 것도 가능합니다. ``` # 동적 타이핑 예시 x = 10 # 정수 y = 20.5 # 실수 name = "Alice" # 문자열 is_valid = True # 불리언

x = "Sally" # 기존에는 정수(int)였으나, 이제 x는 string 타입입니다.

```

## 1-3. 자료형
```

- **정수(int)**: 정수형 데이터를 나타냅니다. 예를 들어, '5', '-3', '42' 등이 있습니다.
- **실수(float)**: 부동소수점을 포함한 숫자를 나타냅니다. 예를 들어, '3.14', '-0.001', '2.0' 등이 있습니다.
- **문자열(str)**: 문자의 시퀀스를 나타냅니다. 문자열은 작은따옴표('')나 큰따옴표("")로 묶어서 표현합니다. 예를 들어, 'hello', "Python" 등이 있습니다.
- **불리언(bool)**: 참(True) 또는 거짓(False)의 두가지 값을 가지는 타입입니다.
- **리스트(list)**: 여러 값을 순서대로 저장하는 값의 나열입니다. 대괄호([]) 안에 값을 나열하여 생성합니다. 예: [1, 2, 3], ['a', 'b', 'c']
- **튜플(tuple)**: 리스트와 유사하지만, 한 번 생성되면 변경할 수 없는 값의 나열입니다. 소괄호(())를 사용합니다. 예: (1, 2, 3), ('a', 'b', 'c')
- **딕셔너리(dict)**: 키와 값의 쌍으로 데이터를 저장하는 데이터 구조입니다. 중괄호({}) 안에 'key: value' 형태로 나열합니다. 예: {'name': 'Alice', 'age': 25}
- **집합(set)**: 중복되지 않는 유일한 값들의 모임입니다. 중괄호({})를 사용하며, 순서는 없습니다. 예: {1, 2, 3}, {'apple', 'banana', 'cherry'}

1-4. 연산자

- **산술 연산자**: +, -, *, /, // (정수 나누기), % (나머지), ** (거듭제곱) 등이 있습니다.
- **비교 연산자**: ==, !=, >, <, >=, <= 등이 있습니다.
- **논리 연산자**: and, or, not 등이 있습니다.

---

2. 정수와 실수 및 연산자

2-1. 정수와 실수 정의하기

- **정수(Integer)**: 정수는 소수점이 없는 숫자입니다. 파이썬에서는 'int' 타입으로 표현됩니다. 예를 들어, '5', '-3', '42'등이 정수입니다.
- **실수(Floating-point number)**: 실수는 소수점을 포함하는 숫자입니다. 파이썬에서는 'float' 타입으로 표현됩니다. 예를 들어, '3.14', '-0.001', '2.0' 등이 실수입니다.

In [ ]:
```
# 파이썬 내장 함수 'type()'은 객체의 타입을 반환합니다.
print("type(5): ", type(5))
print("type(-3): ", type(-3))
print("type(42): ", type(42))
print("type(3.14): ", type(3.14))
print("type(-0.001): ", type(2.0))
```

```
type(5):  <class 'int'>
type(-3):  <class 'int'>
type(42):  <class 'int'>
type(3.14):  <class 'float'>
type(-0.001):  <class 'float'>
```

2-2. 정수와 실수 변환하기

- **정수로 변환**: 'int()' 함수를 사용하여 실수나 다른 타입의 값을 정수로 변환할 수 있습니다. 실수를 정수로 변환하면 소수점 이하의 부분을 버려집니다.

In [ ]:
```
print("result of int(3.14): ", int(3.14))
print("result of int(10.9): ", int(10.9))
print("result of int('100'): ", int('100'))
```

```
result of int(3.14):  3
result of int(10.9):  10
result of int('100'):  100
```

- **실수로 변환**: 'float()' 함수를 사용하여 정수나 다른 타입의 값을 실수로 변환할 수 있습니다.

In [ ]:
```
print("result of float(15): ", float(15))
print("result of float('3.14'): ", float('3.14'))
```

```
result of float(15):  15.0
result of float('3.14'):  3.14
```

2-3. 정수와 실수 연산하기

파이썬에서 정수와 실수를 포함한 기본적인 수학 연산은 다음과 같습니다.

- 덧셈: '+'
- 뺄셈: '-'
- 곱셈: '*'
- 나눗셈: '/' (결과는 항상 실수입니다)
- 정수 나눗셈: '//' (나눗셈의 결과를 정수로 내림하여 반환)
- 나머지: '%',
- 거듭제곱: '**'

In [ ]:
```python
# 덧셈
result = 5 + 2.0  # 7.0, 정수와 실수의 연산 결과는 실수
print(result)

# 뺄셈
result = 5 - 3  # 2, 두 정수의 연산 결과는 정수
print(result)

# 곱셈
result = 3 * 3.5  # 10.5
print(result)

# 나눗셈
result = 7 / 2  # 3.5
print(result)

# 정수 나눗셈
result = 7 // 2  # 3
print(result)

# 나머지
result = 7 % 2  # 1
print(result)

# 거듭제곱
result = 2 ** 3  # 8
print(result)
```

```
7.0
2
10.5
3.5
3
1
8
```

파이썬에서 정수와 실수를 연산할 때, 하나라도 실수가 포함되면 결과는 실수로 반환됩니다. 이는 정밀도를 유지하기 위한 파이썬의 설계입니다. 또한, 나눗셈 연산자('/')는 항상 실수 결과를 반환하는 것에 주의해야 합니다. 정수 결과를 원한다면 정수 나눗셈 연산자('//')를 사용해야 합니다.

3. 문자열

파이선의 문자열(String) 자료형은 텍스트 데이터를 처리할 때 사용됩니다. 문자열은 작은따옴표('), 큰따옴표(")를 사용하여 표현합니다. 또는 삼중 따옴표(''', """)를 사용하여 표현할 수 있으며, 이는 여러 줄에 걸친 문자열을 포함하여 문자 데이터를 표현하는데 사용됩니다.

3-1. 문자열의 기본적인 사용

- **문자열 생성**: 'hello', "world", '''hello world''', """hello world"""와 같은 방법으로 문자열을 생성할 수 있습니다.
- **이스케이프 문자**: 문자열 내에서 특수 문자를 표현하기 위해 백슬래시('\')를 사용합니다. 예를 들어, 줄바꿈은 '\n', 탭은 '\t'로 표현됩니다.
- **연결과 반복**: '+' 연산자로 문자열을 연결하고, '*' 연산자로 문자열을 반복합니다.

In [ ]:
```
print('hello')
```

```
hello
```

In [ ]:
```
print("""
hello
world
""")
```

```
hello
world

```

In [ ]:
```
print("hello\nworld")
```

```
hello
world
```

In [ ]:
```
print("This is 'string'.")
```

```
This is 'string'.
```

In [ ]:
```
print(type("This is 'string'"))
```

```
<class 'str'>
```

In [ ]:
```
print('py' + 'thon')
```

```
python
```

In [ ]:
```
print('py' * 3)
```

```
pypypy
```

3-2. 문자열의 고급 기능

3-2-1. 문자열 포맷팅

[.format() 메서드]

- 파이썬의 ".format()" 메서드는 문자열 내에 특정 값을 삽입하거나 형식을 지정할 때 사용됩니다. 이 메서드는 중괄호 "{}"를 사용한 플레이스홀더를 문자열 내에 포함시키고, ".format()" 메서드에 전달된 인자를 이 위치에 삽입하여 새로운 문자열을 생성합니다.
- 기본 사용법: ".format()" 메서드는 중괄호 "{}"를 사용하여 문자열 내에서 변수의 위치를 지정합니다. 각 중괄호는 ".format()" 메서드에 전달된 인자의 순서대로 대체됩니다.

In [ ]:
```
greeting = "Hello"
name = "Alice"
sentence = "{}. My name is {}.".format(greeting, name)
print(sentence)  # 출력: Hello. My name is Alice.
```

```
Hello. My name is Alice.
```

- 위치와 순서를 지정: ".format()" 메서드에서는 중괄호 내에 숫자를 사용하여 인자의 위치를 명시적으로 지정할 수 있습니다. 이는 같은 변수를 여러 번 사용하거나 순서를 변경하고 싶을 때 유용합니다.

In [ ]:
```
sentence = "{1} is a {0}.".format("programmer", "Alice")
print(sentence)  # 출력: Alice is a programmer.
```

```
Alice is a programmer.
```

- 키워드 인자 사용 : ".format()" 메서드에는 키워드 인자를 사용하여 플레이스홀더에 명시적인 이름을 할당할 수도 있습니다. 이 방법은 코드의 가독성을 높이는 데 도움이 됩니다.

In [ ]:
```
sentence = "{name} is a {job}.".format(name="Alice", job="programmer")
print(sentence)  # 출력: Alice is a programmer.
```

```
Alice is a programmer.
```

- 형식 지정: ".format()" 메서드는 숫자, 날짜 등 다양한 데이터 타입에 대한 형식 지정도 지원합니다. 예를 들어, 소숫점 이하의 자릿수를 지정하거나, 3자리마다 콤마를 찍어서 출력하는 것이 가능합니다.

In [ ]:
```
# 소수점 둘째 자리까지 표시
price = 49.99
sentence = "The price is {:.1f} dollars.".format(price)
print(sentence)  # 출력: The price is 50.0 dollars.

# 실수에 대해 3자리마다 콤마를 찍고, 소수점 이하 두 자리까지 표시하기
num = 1234567.8910
formatted_num = "{:,.2f}".format(num)
print(formatted_num)  # 출력: 1,234,567.89

# 정수에 대해 3자리마다 콤마를 찍기
num_int = 1234567
formatted_num_int = "{:,}".format(num_int)
print(formatted_num_int)  # 출력: 1,234,567
```

```
The price is 50.0 dollars.
1,234,567.89
1,234,567
```

[f-string]

- "f-string"은 파이썬 3.6 버전부터 도입된 문자열 포매팅 기법입니다. f-string은 문자열 앞에 "f"나 "F"를 붙여서 사용하며, 중괄호 "{}" 안에 직접 변수 이름이나 표현식을 넣어 동적으로 문자열을 생성할 수 있습니다.
- 이 방법은 코드의 가독성을 높이고, ".format()" 메서드를 사용하는 방법에 비해 훨씬 간결하고 직관적입니다.
- 기본 사용방법: f-string을 사용할 때는 문자열 앞에 "f"를 붙이고, 중괄호 "{}" 안에 변수명이나 표현식을 직접 넣어줍니다. f-string 내에서는 단순한 변수 뿐만 아니라, 수학 연산, 함수 호출, 객체 메서드 호출 등이 가능합니다.

In [ ]:
```
x = 10
y = 20
print(f"{x} times {y} is {x * y}")
```

```
10 times 20 is 200
```

- 형식 지정: 중괄호 안에 콜론 ":"을 이용해 숫자 형식을 지정하거나, 날짜를 포매팅하는 등의 작업을 할 수 있습니다.

In [ ]:
```
price = 49.99
print(f"The price is {price:.1f} dollars.")
```

```
The price is 50.0 dollars.
```

3-2-2. 문자열 메서드

- **.upper()**: 문자열의 모든 문자를 대문자로 변환합니다.
- **.lower()**: 문자열의 모든 문자를 소문자로 변환합니다.
- **.strip()**: 문자열의 앞과 뒤에서 공백문자들을 제거합니다. 별도의 문자를 지정해주는 경우 지정된 문자들을 제거합니다.
- **.split()**: 문자열을 공백문자를 기준으로 분할합니다. 별도의 문자를 지정해주는 경우 지정된 문자를 기준으로 분할합니다.
- **.replace()**: 문자열 내의 특정 문자 또는 문자열을 다른 문자 또는 문자열로 교체합니다.

In [ ]:
```
print(f"'hello'.upper()의 결과는 {'hello'.upper()}입니다.")
print(f"'HELLO'.lower()의 결과는 {'HELLO'.lower()}입니다.")
print(f"' hello '.strip()의 결과는 {' hello '.strip()}입니다.")
print(f"'--hello--'.strip()의 결과는 {'--hello--'.strip('-')}입니다.")
print(f"'one two three'.split()의 결과는 {'one two three'.split()}입니다.")
print(f"'one,two,three'.split()의 결과는 {'one,two,three'.split(',')}입니다."
print(f"'hello world'.replace('world', 'Python')의 결과는"
      f"{'hello world'.replace('world', 'Python')}입니다.")
```

```
'hello'.upper()의 결과는 HELLO입니다.
'HELLO'.lower()의 결과는 hello입니다.
' hello '.strip()의 결과는 hello입니다.
'--hello--'.strip()의 결과는 hello입니다.
'one two three'.split()의 결과는 ['one', 'two', 'three']입니다.
'one,two,three'.split()의 결과는 ['one', 'two', 'three']입니다.
'hello world'.replace('world', 'Python')의 결과는hello Python입니다.
```

3-2-3. 문자열 슬라이싱

- 문자열 슬라이싱은 파이썬에서 문자열의 부분을 선택하거나 추출할 때 사용하는 기능입니다. 이를 통해 문자열의 특정 섹션을 쉽게 접근하고 조작할 수 있습니다.
- 기본 구조: 문자열[시작 인덱스 : 끝 인덱스 : 간격]
  - 시작 인덱스: 슬라이싱을 시작할 위치입니다. 이 인덱스에 해당하는 문자는 결과에 포함됩니다. 인덱스는 0부터 시작합니다.
  - 끝 인덱스: 슬라이싱을 끝낼 위치로, 이 인덱스 바로 앞까지의 문자가 결과에 포함됩니다. 따라서, 끝 인덱스에 해당하는 문자는 결과에 포함되지 않습니다.
  - 간격: 선택적으로 지정할 수 있으며, 간격마다 문자를 선택합니다. 간격이 음수인 경우 문자열을 거꾸로 슬라이싱합니다.

In [ ]:
```python
s = "Hello, Python!"

# 전체 문자열 선택
print(s[:])  # Hello, Python!

# 첫 번째부터 5번째 문자까지 선택 (0부터 인덱스 시작)
print(s[0:5])  # Hello

# 7번째 문자부터 끝까지 선택
print(s[7:])  # Python!

# 시작 인덱스를 생략하고 5번째 문자까지 선택
print(s[:5])  # Hello

# 간격을 2로 하여 문자 선택
print(s[::2])  # Hlo yhn

# 문자열을 거꾸로
print(s[::-1])  # !nohtyP ,olleH
```

```
Hello, Python!
Hello
Python!
Hello
Hlo yhn
!nohtyP ,olleH
```

In [ ]:
```python

```