python_기초반_1주차

1. 개발 환경 설치

→ 개발 환경은 '텍스트 에디터 + 코드 실행기'로 구성되어 있다.
- 텍스트 에디터: 프로그래밍 언어로 이루어진 코드를 작성하는 곳
- 코드 실행기: 직접 코드를 실행시키는 프로그램

2. 파이썬 개발 환경은?

→ '텍스트 에디터(Visual Studio Code) + 코드 실행기(Python 인터프리터(Interpreter))'로 구성
- 텍스트 에디터는 다양한 종류가 있지만, 수업 시간에는 Visual Studio Code를 사용
- '텍스트 에디터 + 코드 실행기'를 합쳐 '통합개발환경(IDE)'라고 부른다.

3. Naming Rules

* 개발자들 사이에서 일종의 암묵적인 약속으로 지은 규칙. 무조건 naming rule을 따라 지어야 할 필요는 없지만, 소통의 편의를 위해 구분하여 이름 짓는 걸 선호함.

→ Snake Case – 단어와 단어 사이에 밑줄 언더바(_)를 넣고 모두 소문자로 구성 (ex: this_is_python)
- 주로 함수와 변수 이름 지을 때 사용
- 함수는 만들 때 def 키워드를 사용하고, 함수를 사용할 때는 함수이름을 적고 뒤에 ()가 붙는다.
- 변수는 만들 때 변수 이름을 적고 대입연산자(=)를 사용하고, 변수를 사용할 때는 변수 이름만 적는다.

→ Camel Case – 단어와 단어 사이에 공백을 제거하고 각 단어 모두 대문자로 시작 (ex: ThisIsPython)
- 주로 클래스 이름 지을 때 사용

(+) keyword – 이미 예약되어 있는 문자열로 다른 용도로 사용이 불가능한 문자열
(ex: def, if, print, in, else 등등)
- 코드에서 색으로 구별

→ **실습** – 아래 코드보고 naming rule을 따라 class(클래스), variable(변수), function(함수), 키워드 구분하기 (class와 function은 추후 자세히 배울 예정 – 일단 naming rule로만 구분해 보기)

```
class DynamicConverter:
    _dynamic_to_python = None
    _dynamic_to_url = None

    @property
    def regex(self):
        return r"[0-9a-zA-Z]+"

    @regex.setter
    def regex(self):
        raise Exception("You can't modify the regular expression.")

    def to_python(self, value):
        return type(self)._dynamic_to_python(value)

    def to_url(self, value):
        return type(self)._dynamic_to_url(value)

    @classmethod
    def register_to_python(cls, value):
        cls._dynamic_to_python = value

    @classmethod
    def register_to_url(cls, value):
        cls._dynamic_to_url = value
```

4. 주석(Comments)

* 내가 구현한 코드를 다른 사람에게 보여주거나, 다른 사람의 코드를 내가 이해해야 할 때 부가설명의 용도로 만든 장치

→ 한 줄) # 넣기
→ 두 줄 이상) ''' 나 """ 넣기

→ **실습** - 주석 두 종류 찾아보기

```
        if self.disabled:
            return False
        try:
            data = self.to_python(data)
            if hasattr(self, "_coerce"):
                return self._coerce(data) != self._coerce(initial)
        except ValidationError:
            return True
        # For purposes of seeing whether something has changed, None is
        # the same as an empty string, if the data or initial value we get
        # is None, replace it with ''.
        initial_value = initial if initial is not None else ""
        data_value = data if data is not None else ""
        return initial_value != data_value

    def get_bound_field(self, form, field_name):
        """
        Return a BoundField instance that will be used when accessing the form
        field in a template.
        """
        return BoundField(form, self, field_name)
```

5. **컴퓨터로 표현할 수 있는 자료 타입**

→ 크게 4가지로 나눌 수 있다 – 문자형(string), 숫자(float, integer), 불(bool)

- 문자형(string)은 '' 또는 "" 사이에 원하는 내용을 넣으면 완성

- 숫자는 크게 소수점을 표현하는 'float'와 정수형인 'integer' 타입으로 나눌 수 있다.

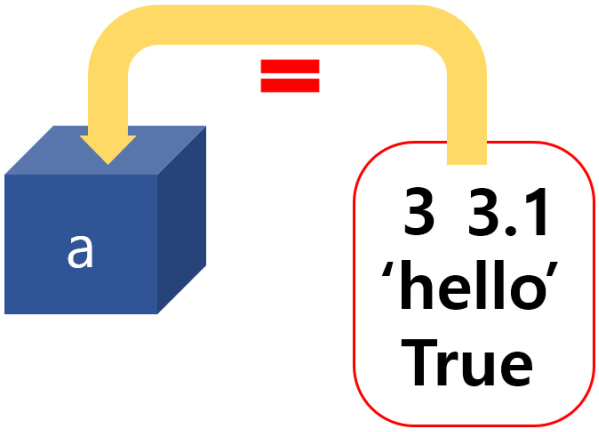
- 불은 True와 False 두 가지로 나뉜다.

(+) type()을 이용하면 자료 타입을 알 수 있다.

```
print(type(2.3)) #<class 'float'> 출력
print(type(2.00)) #<class 'float'> 출력
print(type(2)) #<class 'int'> 출력
```

6. 변수(variable)

* 변수(variable)란 '내용을 담을 수 있는 상자'를 뜻한다. '='이라는 대입 연산자를 통해 우리가 원하는 내용을 '변수'라는 상자에 담을 수 있다.



→ ex) a = 3이라면 3이라는 숫자를 a라는 이름을 가진 상자에 넣겠다는 뜻이다.

**** 주의) 수학의 = 부등호와 의미가 다르다는 점** 꼭! 기억

→ 변수라는 상자 안에는 앞서 5번에서 언급한 네 개의 자료형 모두 들어갈 수 있다.

7. print()문

* 원하는 내용을 출력하고 싶을 때 사용하는 함수

→ 변수를 넣거나, 직접 원하는 내용을 ()안에 집어넣어 출력할 수 있다

```
print(a, b, c, d) #3 hello True 2.1 출력

print(3, 'hello', True, 2.1) #3 hello True 2.1 출력
```

→ print('') 또는 print()문 안에 직접 ₩n을 넣으면 한 줄을 띄어쓸 수 있다.

→ print('')안에 따옴표나 역슬래쉬(₩)를 출력하고 싶으면, 따옴표나 역슬래쉬 앞에 역슬래쉬(₩)를 쓰면 된다. (예시 참조)

```
print('this is\npython')

print('he said, \'I love it!\'')
```

```
this is
python
he said, 'I love it!'
```

7. 문자열 심화

1) 문자열 연산자 +와 *

→ 문자열 연산자 +를 통해 두 개 이상의 문자열을 이어 붙일 수 있다.

→ 문자열 연산자 *를 통해 **동일 문자열**을 원하는 횟수만큼 이어 붙일 수 있다.

* 주의) 문자열 연산자 +는 print()내의 ,(콤마)와 달리 자동 띄어쓰기가 지원되지 않는다. 따라서 문자열을 만들 때 의도적으로 띄어쓰기를 한 채로 만들어주는게 좋다. (예시 참조)

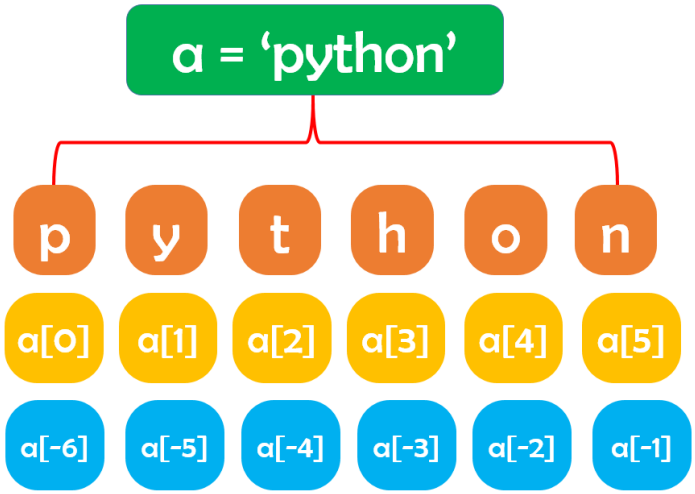
```
a = 'hello'
b = ' nice to meet you.' #의도적 띄어쓰기

c = ' hi' #의도적 띄어쓰기

print(a + b + c*3)
```

```
hello nice to meet you. hi hi hi
```

2) 인덱싱 & 슬라이싱

→ 문자열의 구성

- 문자열은 각 글자마다 나누어 대괄호 [ ]를 이용해 한 문자씩 표현할 수 있다.



→ 인덱싱(indexing)을 통해 문자열 내에 내가 원하는 문자를 찾고, 출력도 할 수 있다.

- 옆 예시) 내가 'p'를 찾고 싶다면 a[0], 또는 a[-6]을 통해 찾을 수 있다.

→ 슬라이싱(slicing)을 통해 문자열 내가 원하는 두 개 이상의 문자들 모음을 찾고, 출력도 할 수 있다.
- :를 이용해 범위를 지정한다. [시작위치 : 끝나는 위치 + 1]

- 여 예시) 내가 'py'를 찾고 싶다면 a[0:2], 또는 a[-6:-4]를 통해 찾을 수 있다.

* 주의) '콜론(:) 기준 오른쪽 숫자 -1' 위치가 찾고자 하는 문자열 모음의 끝 위치

- 콜론 기준 왼쪽 숫자만 적었다면 문자열 끝까지 가져옴을 뜻하고, 오른쪽 숫자만 적었다면 문자열 첫부분까지 가져온다.

- 콜론 기준 양쪽 숫자 모두 적지 않으면 문자열 전체를 가져옴을 뜻한다.

```
a = 'python'

#모두 python 출력
print(a[-6:])
print(a[:])
print(a[0:])
```

3) len() 함수

→ 문자열의 길이를 출력해주는 함수이다. (문자열 내의 띄어쓰기도 포함)