TeamLab/lab_for_gachon_cs50

Lab #4 - 간단한 연산 (basic_operations)

Introduction

축하한다. 2주차임에도 불구하고 이 글을 읽고 있다는 것은 여러분들이 포기하지 않고 수업을 들을 의지를 가지고 있다는 것이다. 그 의지를 16주까지 유지해 보자. 우리는 할 수 있다... 아니 있을 것이다... 아니 있었으면 좋겠다. 이번 Lab은 python의 간단한 연산 문제들을 수행하는 것이다. 연산이라고 해서 간단한 산수만 생각할 수 있으나, 문자열을 포함하여 파이썬에서 기초 데이터형을 어떻게 다루는지 연습하게 된다.
이번 차시부터 Lab Assignment의 설명이 상당히 불친절 해 질 것이다. 이것은 이 수업의 전반적인 추세로 시간이 지나감에 따라 상세한 설명과 가이드는 없어지고 스스로 하는 부분들이 많아진다. 해보면 별거 없다. 해보자.

숙제 template 파일 다운로드

먼저 숙제 template 파일을 cs50 서버로부터 다운로드 받는다. 이미 해보았기 때문에 어렵지 않을 것이다. 로그인 후 나타나는 bash shell에서 다음과 같은 명령을 입력하자.

```
python3.4 submit_assignment.py -get basic_operations
```

위 명령을 실행시키고 로그인 정보를 정확히 입력하여 파일의 다운로드가 끝났다면 다음과 같은 메세지가 나올 것이다.

```
basic_operations.py file is created for your basic_operations assignment
Thank you for using the program. Enjoy Your Assignment - From TeamLab
```

실제 현재 directory에 파일이 제대로 다운로드 되었는지 확인하기 위해서는 현재 bash shell에서 `ls basic_operations.py`을 입력하면 다음 줄에 `basic_operations.py`에 출력이 될 것이다. `ls`는 `list`의 준말로 directroy에 있는 파일을 표시해주는 명령어로 `ls` 뒤에 `파일명`을 입력하면 해당 파일이 있을 경우, 그 파일명을 다시한번 출력해준다. `ls`만 입력하면 현재 디렉토리에 있는 모든 명령어`(#여기 이상합니다. 모든 명령어..? 모든 파일명..아닐까요)`를 표시해준다. 자세한 내용은 구글로 가서 `linux ls 명령어` 라고 치면 다양한 페이지에서 설명해 줄 것이다.

참고로 이제는 작업환경에 대한 용어에 익숙해져야 한다. 아래는 우리가 주로 사용하는 작업 환경에 대한 설명이다.

분류	설명
bash shell	putty 프로그램으로 CS50 서버에 로그인 하자마자 바로 나오는 작업환경을 의미하며, linux라는 OS에 명령어를 입력할 수 있게 해주는 환경. `ls`, `rm` 등 리눅스 OS를 사용하기 위한 명령어를 사용

	할 수 있음
python shell	bash shell에서 `python3.4`를 입력했을 나오는 환경으로 파이썬의 다양한 명령어를 실행 시킬 수 있음
vim editor	bash shell 에서 `vi 파일명` 을 입력해주면 나오는 환경으로 윈도우의 메모장과 같은 기능을 제공함. 본 강의의 파이썬 코드 파일은 vim editor에서 수정되는 것을 기본으로 함

수정 해야할 함수 종류들

숙제 파일을 다운로드 후 `vi basic_operations.py` 명령어를 입력하여 숙제 파일을 살펴보자. 코드가 좀 길긴 하지만 기본적인 형태는 lab 2의 `arithmetic_function.py`와 다르지 않다. 본 lab에서 수정해야 할 함수들은 아래와 같다.

함수	설명
str_to_int	문자열 값을 입력받아 정수형으로 바꾸는 함수
str_to_float	문자열 값을 입력받아 실수형으로 바꾸는 함수
number_to_str	정수형 또는 실수형의 값을 입력받아 문자열 값으로 바꾸는 함수
add_string_number	문자열 값과 숫자형 값을 입력받아, 두 값을 문자열 값으로 연결하는 함수
add_string_string	문자열 값과 문자열 값을 입력받아, 두 값을 문자열 값으로 연결하는 함수
associative_law_add	(a + b) + c 와 같이 숫자형 값을 입력받아 결합 법칙을 이용한 덧셈 결과 값을 반환해주는 함수
associative_law_mutiple	(a * b) * c 와 같이 숫자형 값을 입력받아 결합 법칙을 이용한 곱셈 결과 값을 반환해주는 함수
distributive_law	a * (b + c) 와 같이 숫자형 값을 입력받아 분배 법칙을 이용한 결과 값을 반환해주는 함수
exponent	밑과 승수 값을 입력 받아 지수 계산 결과 값을 반환해주는 함수

하나의 함수를 예로 들면 모든 함수는 아래와 같은 구조로 쓰여져 있다.

```
# 데이터 형변환=============================================
def str_to_int(string_number):
    # """
    #   Input:
    #   -string_number: 숫자형태의 문자열
    # Output:
    #   -integer: 정수 값
    # Examples:
    #   >>> str_to_int("3")
    #   3
```

```
    #   >>> str_to_int("135")
    #   135
    # """
    # ===Modify codes below================
    result = None

    # =====================================

    return result
```

함수 제일 상단에 `def str_to_int(string_number)`에 `def`는 함수를 선언하는 예약어이고, `str_to_int`는 본 함수의 이름이다. `string_number`는 본 함수에 입력되는 변수의 이름이다. 마지막에 붙어 있는 `:`는 파이썬의 기본 문법으로 `:` 이하로 들여쓰기가 있는 줄까지 본 함수의 영역이다. 그 아래부터 `# ===Modify codes below================` 전 까지는 본 함수에 대한 설명으로 Input 변수, Output 변수, 결과 확인을 위한 예시들로 구성되어 있다. 예시의 경우 간단히 표시하기 위해 함수명과 Input 값만 적었지만 실제로 `python shell`에서 실행 시키기 위해서는 아래와 같이 입력해야 한다. 혹시 헷갈릴까봐 하는 말이지만 `python shell`로 들어가기 위해서는 `bash shell`에서 `python3.4`라는 명령어를 입력해 주어야 한다. 실제 현재 코드를 본 Lab에 목적에 맞게 수정한 후 `python shell`에서 아래와 같이 입력해보자.

```
>>> import basic_operations as bo
>>> bo.str_to_int("3")
3
```

위 코드에서 첫 번째 줄 `import basic_operations as bo` 은 `basic_operations` 이라는 모듈을 부르는 명령어로 `import basic_operations` 모듈을 `bo`라는 이름으로 사용하겠다는 뜻이다. 모듈에 대한 설명은 이후에 한다. 지금은 단지 모듈의 이름은 우리가 작성한 파일의 이름과 동일하다고만 이해하고 있으면 된다. 아래줄에 `bo.str_to_int("3")`는 우리가 작성한 프로그램의 함수를 실제로 실행 시키는 명령이다. 우리가 `basic_operations`의 이름을 축약하여 `bo`로 바꾸었기 때문에 `bo` 라는 이름으로 함수를 실행할 수 있다. 만약 첫번째 줄의 명령어에 `as bo`가 빠져 있다면 `basic_operations.str_to_int("3")`로 실행할 수 있다. 이미 알고 있겠지만 `str_to_int`는 함수 이름이고, 함수에 정의한대로 입력 값을 넣어야 한다. 우리가 함수를 선언 할 때, `str_to_int(string_number)` 에서 보듯이 `string_number`라고 하는 하나의 값만 입력할 수 있도록 지정했기 때문에 테스트 예제에서도 `("3")` 으로 하나의 값만 입력하였다. 만약 `("3","4")`와 같이 다른 값들을 입력하게 되면 에러 메세지를 보게 될 것이다. 각 함수에 목적에 맞게 9개의 함수를 수정해보자.

수정후 테스트 하기

본 lab 숙제를 맞게 작성했는지 확인해보도록 하자. 참고로 모든 함수를 다 수정한 후 테스트 할 필요는 없다. 함수 하나 수정 할때마다 테스트는 가능하다. 테스트를 위한 기본코드는 `main`함수에 아래와 같이 들어가 있다. 상당히 많은 내용이기 때문에 상단에 테스트 코드만 보도록 한다.

```
def main():
    print("Str_to_int Test")
    print(str_to_int("56"))  # Expected Result: 56
    print("====> ", str_to_int("27") == 27)  # Expected Result: True
    print("Str_to_int Test Closed Wn")
```

```
    print("str_to_float Test")
    print(str_to_float("8.4501"))  # Expected Result: 8.4501
    print("====> ", str_to_float("3.4") == 3.4)  # Expected Result: True
    print(str_to_float("6.74") == 9.8)  # Expected Result: False
    print("Str_to_float Test Closed ₩n")
    # 이하 생략
```

각 함수의 테스트 코드 들은 Test 시작과 끝을 알리는 `print` 문 사이에 작성되어 있다. 각 테스트에는 `print`문을 사용하여 함수의 결과를 화면에 출력해보게 하고 `====>`뒤에 `True`의 결과값이 나오는지 확인하게 한다. 아무런 수정없이 코드를 `python3.4 basic_operations.py` 명령어로 실행 시키면 아래와 같은 결과 값이 출력될 것이다. 참고로 실행은 당연히 `bash shell`상에서 해야 한다.

```
Str_to_int Test
None
====>  False
Str_to_int Test Closed

str_to_float Test
None
====>  False
False
Str_to_float Test Closed
# 이하 생략
```

각 함수를 수정후 `python3.4 basic_operations.py` 명령어로 다시 실행 시키면 아래와 같은 결과를 볼 수 있을 것이다.

```
Str_to_int Test
56
====>  True
Str_to_int Test Closed

str_to_float Test
8.4501
====>  True
False
Str_to_float Test Closed
# 이하 생략
```

`main` 함수 내에 코드들은 사용자가 마음대로 수정이 가능하다. 테스트하고 싶은 숫자를 새로 넣거나 필요없는 테스트는 지워도 숙제 제출에는 영향을 주지 않으니 참고하기 바란다. 참고로 해당 코드를 실행 시키지 않기 위해서는 아래와 같이 코드 앞에 `#`을 붙여주면 된다.

```
def main():
    #print("Str_to_int Test")
```

```
    #print(str_to_int("56"))  # Expected Result: 56
    #print("====> ", str_to_int("27") == 27)  # Expected Result: True
    #print("Str_to_int Test Closed ₩n")
```

당연히 #을 제거 하면 다시 코드는 정상적으로 실행된다. #은 해당 코드를 실행 시키지 않고 주석(Comment)를 달아 설명해줄 수 있게 하는 예약어이다. 원래는 코드에 대한 설명은 개발자가 달아줄 때 사용되나 테스트 할 경우, 실행하지 않을 코드에 달아서 테스트를 조정할 수 있다. 모든 코드를 다 수정한 후 `python3.4 basic_operations.py` 을 입력하면, 총 9번의 `====> True` 가 표시될 것이다. 하나라도 `====> False`가 있다면 제대로 수정되지 않은 것이니 다시 확인하기 바란다.

숙제 제출하기

아마 첫 번째 차시에서 고생을 한 학생이라면 이번 숙제는 매우 쉽게 수행하였을 것이다. 숙제 제출을 위해 아래 코드를 입력하자. 그 전에 숙제 제출까지 상세히 설명해주는 단계는 이번이 마지막이다. 숙제 제출을 위한 코드는 `python3.4 submit_assignment.py -submit 파일명`으로 구성되어 있으니 반드시 기억하자.

```
python3.4 submit_assignment.py -submit basic_operations.py
# 아이디와 패쓰워드 입력
-------------------- | ---------- | --------------------
       Function Name |    Passed? |             Feedback
-------------------- | ---------- | --------------------
   add_string_number |       PASS |             Good Job
   add_string_string |       PASS |             Good Job
associative_law_mutiple |       PASS |             Good Job
       number_to_str |       PASS |             Good Job
          str_to_int |       PASS |             Good Job
            exponent |       PASS |             Good Job
         str_to_float |       PASS |             Good Job
 associative_law_add |       PASS |             Good Job
    distributive_law |       PASS |             Good Job
-------------------- | ---------- | --------------------
```

왠지 모를 쾌감에 뿌듯할 것이다. 그러나 오늘의 시작일 뿐이니 다음 숙제로 넘어가자.

Next Work

사람에 따라서는 이번 숙제를 정말 쉽게 한 사람도 있을 것이다. 사실 lab 2와 기본적인 숙제하는 방식은 다르지 않다. 그러나 앞으로 진행될 lab들은 상당히 어려울 것이다. 걱정하지말자 우리에겐 slack과 (마음만) 미남미녀인 TA들이있다.

> **Human knowledge belongs to the world** - from movie 'Password' -

Footnotes