1. 개요

이 pdf 파일은 폴리매스 코드 챔피언십 문제들을 풀기 위해 필요한 python 문법과, 대회 환경을 사용하는 방법을 다루고 있습니다. 대회의 모든 문제는 이 pdf 파일에 쓰여 있는 python 문법만으로 풀 수 있음이 보장됩니다. **이 pdf 파일을 반드시 읽어 주시기 바랍니다.**

코딩 대회의 특성상, 문제의 풀이를 알아냈다고 하더라도 프로그래밍 언어를 다루지 못하면 점수를 받을 수 없습니다. 따라서 이 pdf 파일에 쓰여 있는 python 문법을 최대한 숙지해 주시기 바랍니다.

2. 출력

기본 중의 기본인 출력부터 시작해 봅시다. (이 대회에서는 **출력만 할 수 있어도 점수를 받을 수 있습니다!**)

python에서 **함수**란 입력을 받아 계산을 한 뒤 결과를 내놓는 것을 말합니다. 때에 따라서는 입력을 받아 특정 행동을 하고, 결과를 내놓지 않기도 합니다.
python에서 출력을 하기 위해서는 **print** 함수를 사용합니다. 사용법은 간단합니다. 아래 코드를 보세요.

코드	실행 결과
`print('Hello!')`	`Hello!`

위 코드에서 'Hello!'는 문자열인데, 뒤에서 다루게 될 것입니다. 위 코드를 실행하면, Hello! 가 출력되는 것을 보실 수 있습니다.

python은 한 번 출력할 때마다 줄을 바꿉니다. 아래 예시를 보시면 이해되실 겁니다.

코드	실행 결과
`print(1)` `print(2)`	`1` `2`

줄이 바뀌는 이유는 python이 기본적으로 값을 출력할 때 뒤에 **개행문자**(줄바꿈 역할을 하는 문자. 사실 줄바꿈도 'a', 'b'와 같은 형식의 **문자**입니다!)를 붙이기 때문입니다. 따라서 줄바꿈을 하지 않으려면, 아래와 같은 코드를 이용하면 됩니다.

코드	실행 결과
`print(1, end=' ')` `print(2, end=' ')`	`1 2`

출력할 때 끝에 붙는 문자를 개행 문자가 아닌, 공백으로 설정해 준 것입니다. 이렇게 하면 한 줄에 여러 번 출력할 수 있습니다.

3. 변수

변수는 자료를 담는 기본 공간입니다. python에서 변수에 자료를 담는 것을 **대입**이라고 합니다. python에서는 변수를 따로 만들 필요 없이 바로 값을 대입할 수 있습니다. 대입 연산자는 = 기호이며, 사용 방법은 아래와 같습니다.

코드	실행 결과
`a = 1` `print(a)`	`1`

위 코드는 a라는 변수에 1이라는 값을 대입해 출력하는는 코드입니다. 1은 정수입니다. 따라서 a는 정수형 변수가 됩니다.

자료형이란 자료의 형식을 말합니다. 예를 들어 위에서 a의 자료형은 정수, **int**입니다.[1] 그러나 a에 문자 'x'를 대입하게 되면, a는 문자형 변수로 바뀝니다. 이처럼 변수의 자료형은 **어떤 값을 대입하느냐에 따라 언제든지 바뀔 수 있습니다.**[2] python에서 알아야 할 중요한 **자료형**은 아래와 같습니다.

- 정수 (int)
- 실수 (float)
- 참 / 거짓 (bool)
- 리스트 (list)
- 튜플 (tuple)
- 문자열 (str)

하나씩 살펴봅시다. **정수**는 말 그대로 정수를 의미합니다. python에서 정수형 변수는 한계 없이 모든 정수를 담을 수 있습니다. 기본적인 정수 0부터 시작해서, 2^{100}과 같은 매우 큰 정수도 담을 수 있습니다.

1) 정수를 뜻하는 integer의 약자입니다.
2) C와의 차이점입니다. C는 int로 선언된 정수형 변수에는 영원히 정수밖에 저장할 수 없습니다.

실수 역시 말 그대로 실수를 의미합니다. 다만 정수형 변수와의 차이점이 있다면, **실수형 변수는 정확한 값을 저장할 수 없습니다.** 어쩌면 당연한 것이, 1/3 같이 나누어떨어지지 않는 실수나, 원주율과 같은 무리수는 정확한 값을 저장하려면 무한한 메모리가 필요하기 때문입니다. 따라서 python은 실수형 변수에 대충 비슷한 값을 저장하게 됩니다.

정수와 실수에서는 여러 가지 연산이 가능합니다. 아래 예시를 봅시다.

코드	실행 결과

`print(1+3, 2-7)`
`print(3.6*2.5, 6.3/3.0)`
`print((1+3+1)//2, (1+3+1)%2)`
`print(2**20)`
	`4 -5`
	`9.0 2.1`
	`2 1`
	`1048576`

위 일곱 가지 연산이 정수와 실수에서 가능한 기본 연산입니다. +, -, *, /는 각각 사칙연산을 나타내며, //는 몫을, %는 나머지를, **는 거듭제곱을 나타냅니다. 괄호도 존재합니다. **이제 폴리매스 코드 챔피언십 연습 대회 1번 문제를 풀어보세요.**

코드	실행 결과

`a = 1`
`b = 1.0`
`print(a, b)`
	`1 1.0`

위 코드에서 변수 a에는 정수 1이 대입되지만, 변수 b에는 실수 1.0이 들어갑니다. 정수 뒤에 '.0'을 붙이면 실수로 바꿀 수 있는 것입니다. 또 위에 보면 print문 안에서 쉼표(,)를 활용하고 있는데, 쉼표를 활용하면 같은 줄에 여러 개의 값을 띄어쓰기를 사이에 두고 저장할 수 있습니다.

불리안(참/거짓, bool)은 참 또는 거짓을 저장하는 변수입니다. python에서 참은 True, 거짓은 False라고 합니다.

리스트와 **튜플**은 모두 여러 개의 변수를 한 데 묶어주는 역할을 합니다. 리스트와 튜플은 아래와 같이 생성합니다. (변수를 만드는 것이 아닌, 리스트를 만들어 변수에 대입하는 것입니다.)

코드	실행 결과

`a = [1, 2, 3, 'a']`
`b = (1, 2, 3, 'a')`
`print(type(a), type(b))`
	`<class 'list'> <class 'tuple'>`

위 코드는 리스트와 튜플을 만들고 각각의 자료형을 출력하는 코드입니다. **type** 함수는 값의 자료형을 알려주는 함수입니다. 위 코드에서 a는 리스트, b는 튜플이 됩니다. 리스트와 튜플의 차이점은, 리스트는 값을 바꿀 수 있지만 튜플은 바꿀 수 없다는 것입니다.

리스트와 **튜플**의 원소를 알고 싶을 때에는 아래와 같은 코드를 사용합니다. 조심할 점이 많으니 주의해주세요.

코드	실행 결과

`a = [1, 2, 3, 'a']`
`b = (1, 2, 3, 'a')`
`print(a[0], b[3])`
	`1 a`

맞습니다. 리스트 내의 값에 접근하기 위해서는 **대괄호 []**를 이용합니다. 이때 [] 안에 들어가는 값은 찾고 싶은 값이 리스트의 **몇 번째** 원소인지를 말합니다. **순서는 0부터 시작한다는 점을 조심하세요!**

리스트에는 몇 개의 원소가 있을까요? 또, 리스트에 값을 추가하거나 제거하려면 어떻게 해야 할까요? 간단합니다.

코드	실행 결과

`a = [1, 2]`
`print(len(a))`
`a.append(3)`
`print(len(a))`
`del a[1]`
`print(a)`
	`2`
	`3`
	`[1, 3]`

리스트의 길이를 알기 위해서는 len 함수를 사용합니다. 위 코드의 2, 4번째 줄에서 사용 예시를 볼 수 있습니다. 리스트 맨 끝에 값을 추가하려면 append 함수를 사용합니다. 또, 리스트의 값을 제거하려면 **del** 함수를 사용하는데요, 위 코드에서 사용법을 충분히 파악하실 수 있을 거라고 생각합니다.

튜플은 len 함수는 사용할 수 있지만, 값을 바꿀 수 없기 때문에 append나 del은 사용할 수 없습니다.

리스트끼리, 또는 튜플끼리 이어붙이는 것도 가능한데, +를 사용하면 됩니다.

코드	실행 결과
print([1] + [2]) print((1,) + (2,))	[1, 2] (1, 2)

문자열은 tuple과 비슷합니다. 원소를 바꿀 수 없는 점도 같습니다. 단, 문자열의 모든 원소는 문자여야 합니다. 즉, 문자열은 **문자만 담고 있는 튜플** 정도로 생각하면 됩니다.

문자열을 나타낼 때는 **큰따옴표, 작은따옴표**를 사용합니다.

코드	실행 결과
a = 'string' print(a) print(type(a))	string <class 'str'>

가끔씩 특이한 문자들이 필요할 때가 있습니다. 그 예시로 앞에서 설명한 개행문자, 탭, 따옴표 등이 필요할 때가 있습니다. 이때 필요한 것이 이스케이프 코드인데요, 아래와 같습니다.

문자	이스케이프 코드
줄바꿈	\n
탭	\t
큰따옴표	\“
작은따옴표	\‘

여기서 \으로 보이는 것은 **역슬래시**인데, 대부분의 한국 키보드에는 ₩로 표시되어 있으니 참고하시기 바랍니다.

코드	실행 결과
print('1+2\n3')	1+2 3

리스트, 튜플, 문자열에는 **슬라이싱**이라고 하는 좋은 기능이 있습니다. 슬라이싱은 리스트, 튜플, 문자열의 일부분을 자를 수 있게 해 주는 기능입니다.

코드	실행 결과
a=[0, 1, 2, 3, 4, 5, 6, 7] print(a[2:5]) print(a[:4]) print(a[6:])	[2, 3, 4] [0, 1, 2, 3] [6, 7]

감이 오셨나요? a[x:y]는 a의 x번부터 (y-1)번까지의 원소들을 뽑아 새로운 리스트로 만드는 것입니다. : 앞을 생략하면 처음부터 시작하고, : 뒤를 생략하면 맨 마지막에서 끝나게 됩니다.

a[x:y:z] 와 같은 방식으로 간격을 조정할 수도 있습니다. 아래와 같습니다.

코드	실행 결과
`a = [1, 2, 3, 5, 8, 13, 21, 34]` `print(a[0:6:2])`	`[1, 3, 8]`

x번 원소에서 시작해서, y번 원소로 도달할 때까지 z칸씩 건너뛰며 뽑아오네요.

3. 변수 자료형 변환

가끔씩 int를 float로, list를 tuple로 바꿔야 할 때가 있습니다. 이처럼 한 자료형의 원소를 다른 자료형으로 바꿔야 할 때가 많은데요, 이때 사용하는 함수들이 있습니다.

사실 함수의 이름은, 우리가 도달하고자 하는 자료형의 이름과 같습니다. 예시를 들어 봅시다.

코드	실행 결과

`a = 3`
`print(a, type(a))`
`a = float(a)`
`print(a, type(a))`
`a = str(a)`
`print(a, type(a))`
	`3 <class 'int'>`
	`3.0 <class 'float'>`
	`3.0 <class 'str'>`

위 코드가 상당히 직관적이라, 자세한 설명은 생략하겠습니다. 마지막 줄에서 출력되는 3.0은 실수 3.0이 아니라, '3.0'이라는 **문자열**임을 알아 두시면 좋겠습니다. 예외적으로 문자의 유니코드 값은 **ord** 라는 함수로 구할 수 있습니다.

4. 입력

이제 입력을 받을 차례입니다. 입력을 받는 함수는 input입니다.

코드	입력	실행 결과
`a = input()` `print(a, type(a))`	`3a6b`	`3a6b <class 'str'>`

input은 입력에서 **문자열 형태**로 한 줄을 읽어서 반환합니다. 만약 입력으로 들어온 값을 수로 사용하고 싶다면, 문자열을 수로 바꿔줘야 합니다. 위에서 다룬 변수 자료형 변환을 활용하면 됩니다.

코드	입력	실행 결과
`a = int(input())` `print(a*a)`	25	625

위 코드는 정수를 입력받아 제곱을 출력하는 코드입니다.

가끔씩 한 줄에 여러 개의 수를 입력으로 받아야 할 때도 있습니다. 이럴 때는, 공백을 기준으로 문자열을 쪼개야 합니다. 이러한 역할을 해 주는 것이 **split** 함수입니다.

코드	입력	실행 결과
`a = input().split()` `print(a)`	3 9	['3', '9']

위 예시와 같이 split 함수는 문자열 '3 9'를 공백을 기준으로 '3'과 '9'로 나눴다는 사실을 알 수 있습니다. split 함수는 문자열을 공백 기준으로 나눈 **리스트를 반환합니다.**

그런데, 이 두 값을 각각 다른 변수에 집어넣으려면 어떻게 할까요? 방법은 다음과 같습니다.

코드	입력	실행 결과

`a, b = input().split()`
`print(a, type(a))`
`print(b, type(b))`
	3 9	3 <class 'str'>
		9 <class 'str'>

다만 이 방법을 쓰려면 입력받은 줄에 값이 몇 개 있는지 그 개수를 **정확히 알아야만 한다는 사실**을 기억하셔야 합니다.

split을 사용해서 나온 값들 또한 문자열이기 때문에, 수로 활용하려면 int로 바꾸어야 합니다. 그런데 값이 매우 많다면, 일일이 int 함수를 적용하기 힘듭니다. 따라서, int와 같은 함수를 리스트의 각 원소에 **일괄 적용**시키는 함수, map을 사용할 것입니다.

코드	입력	실행 결과

`a, b = map(int, input().split())`
`print(a, type(a))`
`print(b, type(b))`
	3 9	3 <class 'int'>
		9 <class 'int'>

실행 결과에서 변수의 형식이 int인 것에 주목하세요! 이제 이 상태로 다양한 문제를 풀 수 있게 됩니다.

코드	입력	실행 결과
`a, b = map(int, input().split())` `print(a+b)`	`3 9`	`12`

위 코드는 두 수를 입력받아 더하는 코드입니다. **이제 폴리매스 코드 챔피언십 연습 대회 2번 문제를 풀어보세요.**

몇몇 문제에서는 리스트를 입력으로 받아야 합니다. 예를 들어, 첫 줄에 리스트의 크기를 주고, 둘째 줄에 리스트의 값을 주게 됩니다. 이런 경우도 살펴봅시다.

코드	입력	실행 결과
`a = list(map(int, input().split()))` `print(a)`	`3 9`	`[3, 9]`

위 코드의 첫 번째 줄은 입력을 받아서 수 리스트로 만들어 주는 코드입니다.

5. if, else, elif

이제 입출력과 변수에 대해 알아보았으니, 프로그램의 흐름을 제어해 봅시다. 지금까지 짠 코드는 모두 위에서 아래로 실행되었죠. 다른 방법은 없을까요?

if는 어떤 조건이 만족되는지 확인한 뒤에, 프로그램의 흐름을 제어하는 역할을 합니다. 말로 설명하긴 힘들기 때문에, 예제를 보시죠.

코드	입력	실행 결과

`a = int(input())`
`if a>10:`
`  print('Big')`
	`15`	`Big`

	`10`	(출력 없음)

위 코드를 이해하셨나요? 입력으로 15를 넣게 되면, if문 안의 조건이 만족됩니다. 따라서 if문 안으로 코드가 흘러가게 됩니다. 반면 10을 넣게 되면, if문 안의 조건이 만족하지 못합니다. 따라서 if문 안으로 코드가 들어가지 못합니다. 그래서 출력도 없는 것이죠.

여기서 **>**는 비교 연산자입니다. python에는 >, <, ==, !=, >=, <= 6가지의 비교 연산자가 있습니다. 여기서 >, <, >=, <=는 그 역할이 분명하므로 생략합니다. ==는 두 값이 같은지 비교하는 연산자인데요, 두 값이 같으면 True, 다르면 False를 반환합니다. =를 두 개나 쓰는 이유는 =가 하나만 있으면 대입 연산자와 겹치기 때

문입니다. !=는 두 값이 다른지 비교하고, 다르면 True, 같으면 False를 반환합니다.

또 if문 안쪽은 들여쓰기를 한 것을 볼 수 있는데요, 이렇게 if문 안쪽 영역을 **블록(block)**이라고 하며, 일정하게 들여써야 합니다. 그래야 블록의 시작과 끝을 알 수 있는 것입니다.

else는 if문으로 들어가지 못한 코드가 들어가는 공간입니다. 아래 예시를 보시죠.

코드	입력	실행 결과

`a = int(input())`
`if a>10:`
`  print('Big')`
`else:`
`  print('Not Big')`
	15	Big

	10	Not Big

a가 15일 때는 위쪽 if문의 조건을 만족하여, if문 바로 아래의 블록으로 들어가게 됩니다. 하지만 a가 10일 때는 위쪽 if문의 조건을 만족하지 못해, 아래의 else문 블록으로 들어가게 됩니다.

코드	입력	실행 결과

`a = int(input())`
`if a>10:`
`  print('Big')`
`elif a<10:`
`  print('Small')`
`else:`
`  print('Ten')`
	15	Big

	10	Ten
	5	Small

elif는 else와 if를 합친 것인데요, else문이 있는 자리에서 한 번 더 조건을 검사하기 위해 사용합니다.

in 명령어는 어떤 값이 리스트, 튜플 혹은 문자열 안에 있는지를 판별합니다. 아래 코드를 보세요.

코드	입력	실행 결과

`a = int(input())`
`if a in [1, 2, 3]:`
`  print('Exist')`
`else:`
`  print('Not exist')`
	1	Exist

	2	Exist
	4	Not exist

이렇게 x in A는 원소 x가 배열, 튜플 혹은 문자열 A에 들어 있다면 True를, 들어 있지 않다면 False를 반환합니다.

가끔씩 if문에서 조건 A와 조건 B를 둘 다 만족하는지 검사해야 할 때가 있습니다. 이렇게 조건을 중첩시키려면 **and, or**를 사용합니다. 또, 어떤 조건이 성립하지 않는지 확인할 때는 **not**을 사용합니다. 아래 예시를 보시죠.

코드	입력	실행 결과

`a = int(input())`
`if (a <= 7 and a%2 == 1) or (a > 7 and a%2==0):`
`    print(31)`
`elif not (a==2):`
`    print(30)`
`else:`
`    print(28)`
	1	31

	2	28
	11	30

위 코드는 and, or, not을 모두 활용했고, 윤년이 아닌 해에서 달을 입력받은 뒤 이 달이 며칠까지 있는지 출력하는 프로그램입니다. 어떤 원리로 동작하는지 분석해 보세요! **이제 폴리매스 코드 챔피언십 연습 대회 3번 문제를 풀어보세요.**

6. while

우리가 코딩을 하는 이유 중 하나는, 사람이 하면 오래 걸릴 작업을 컴퓨터는 더 빨리 수행할 수 있기 때문입니다. 특히 컴퓨터는 같은 작업을 반복하는 데 걸리는 시간이 상당히 짧습니다. 제어문을 배웠으니, 기초적인 반복문 while에 대해 배워 봅시다.

코드	입력	실행 결과

`a = int(input())`
`ans = 0`
`while a > 0:`
`    ans += a`
`    a -= 1`
`print(ans)`
	4	10

	10	55

while문은 오른쪽의 조건이 만족하지 않을 때까지 계속해서 while 블록 안의 명령어를 수행합니다. 따라서 a가 0보다 클 때까지 ans에 a를 더하고, a에서는 1을 빼는 것입니다. 즉 이 프로그램은 1부터 a까지의 정수의 합을 구하는 프로그램입니다.

여기서 **+=, -=**와 같은 연산이 등장하는데, a += b는 a = a + b를 줄인 것입니다. 비슷한 방식으로 +=, -=, *=, /=, //=, %=, **=가 존재합니다.

만약 while문의 조건이 처음부터 만족하지 않는다면, 프로그램은 while문을 그냥 지나치게 됩니다. while문 안의 조건이 True인 경우, 프로그램은 무한히 반복문을 돌게 됩니다. 이 **무한 루프**는 상당히 자주 겪게 될 현상이니, 알아두시면 좋습니다.

while문에서 쓸 수 있는 명령어로 break와 continue가 있습니다. 예시로 알아봅시다.

코드	입력	실행 결과

`a = int(input())`
`while True:`
`    if a%4==0:`
`        break`
`    a += 1`
`print(a)`
	4	4

	10	12

break는 현재 돌고 있는 while문이나, 추후에 배울 for문에서 빠져나가는 역할을 담당합니다. 따라서 while True로 무한 루프를 만들었음에도 불구하고, a가 4의 배수가 되면 빠져나오는 것입니다. 짐작하셨겠지만, 이 코드는 a 이상의 가장 작은 4의 배수를 찾는 코드입니다.

코드	입력	실행 결과

`a = int(input())`
`while a > 0:`
`    a -= 1`
`    if a%2==0:`
`        continue`
`    print(a, end=' ')`
	4	3 1

	9	7 5 3 1

continue는 현재 돌고 있는 while문이나, 추후에 배울 for문에서 반복문의 맨 처음으로 돌아가는 역할을 합니다. 따라서, 위 코드에서 a가 짝수이면 print를 하지 않고 while의 맨 처음으로 돌아가게 됩니다. 즉 이 코드는 a보다 작은 홀수를 찾아 출력하는 코드입니다.

7. for

for는 반복문이지만, while보다 훨씬 강력한 기능을 가지고 있습니다. for문은 어떤 리스트 안에 있는 모든 원소를 돌면서 반복하는 기능을 가집니다. 예를 들어 봅시다.

아래 코드는 수열을 입력받은 뒤, 수열에 있는 수들의 합을 구하는 코드입니다.

코드	입력	실행 결과
a = list(map(int, input().split())) add = 0 for i in a: add += i print(add)	3 2 1	6
	7 8 9 10	34

작동 원리가 무엇일까요? 그것은 바로 **for** i **in** a에서 리스트 a의 원소들을 돌아가면서 i에 대입하기 때문입니다. 따라서 i에는 a의 각 원소들이 한 번씩 더해지게 되는 것이죠.

for와 같이 나오는 함수로는 **range**가 있습니다. **range**는 매우 편리한 함수로, 특정 범위의 값을 가진 리스트를 만들어 줍니다. 예를 들어,

코드	실행 결과
a = list(range(1, 8)) print(a)	[1, 2, 3, 4, 5, 6, 7]

와 같이 하면 a에는 1 이상 8 미만의 정수가 담기게 됩니다. 간격을 설정하려면

코드	실행 결과
a = list(range(1, 8, 3)) print(a)	[1, 4, 7]

과 같이 하면 3씩 증가하게 됩니다. for문과 range문을 같이 써 볼까요?

코드	실행 결과
mult = 1 for i in range(3, 6): mult *= i print(mult)	60

네, 맞습니다. 이 코드는 mult에 3부터 5까지의 자연수를 곱하는 코드입니다. 따라서 이 코드는 3*4*5=60을 출력하게 됩니다.

8. 제어문과 반복문의 중첩

당연하게도, if, for, while 여러 개를 같이 쓰는 것도 가능합니다. 이미 위에서 몇 가지 예시를 든 것 같기 때문에, 예제는 생략하도록 하겠습니다. **들여쓰기 조심!**

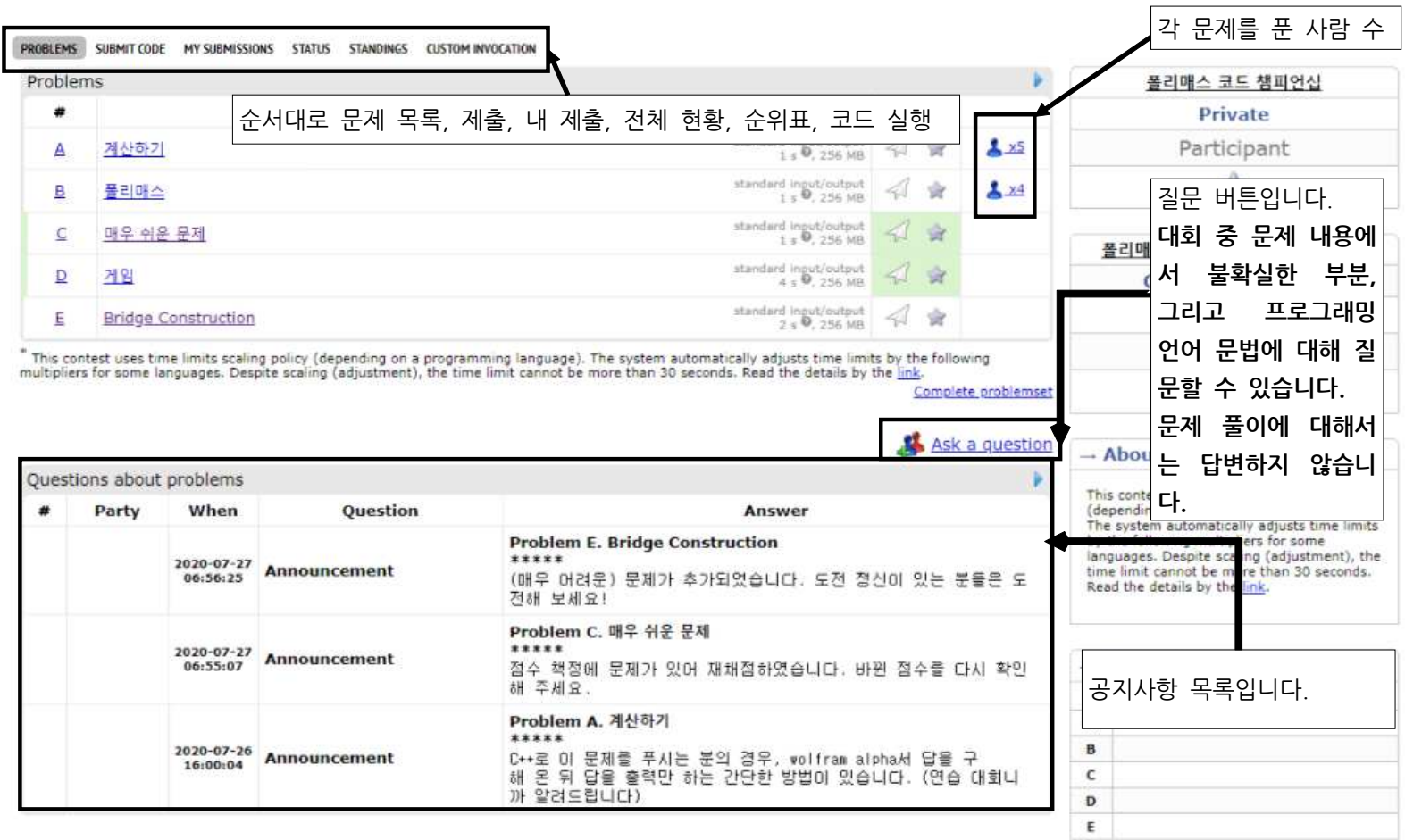
9. 함수

함수는 정말 다루고 싶었지만, 여기에 함수를 쓰게 되면 분량이 두 배가 되고, 작성 시간이 몇 시간 이상 걸리는 등 많은 부작용이 있습니다. 그래서 함수 관련 내용은 가장 잘 설명된 '점프 투 파이썬'의 함수 설명 링크로 대체하겠습니다. 죄송합니다.

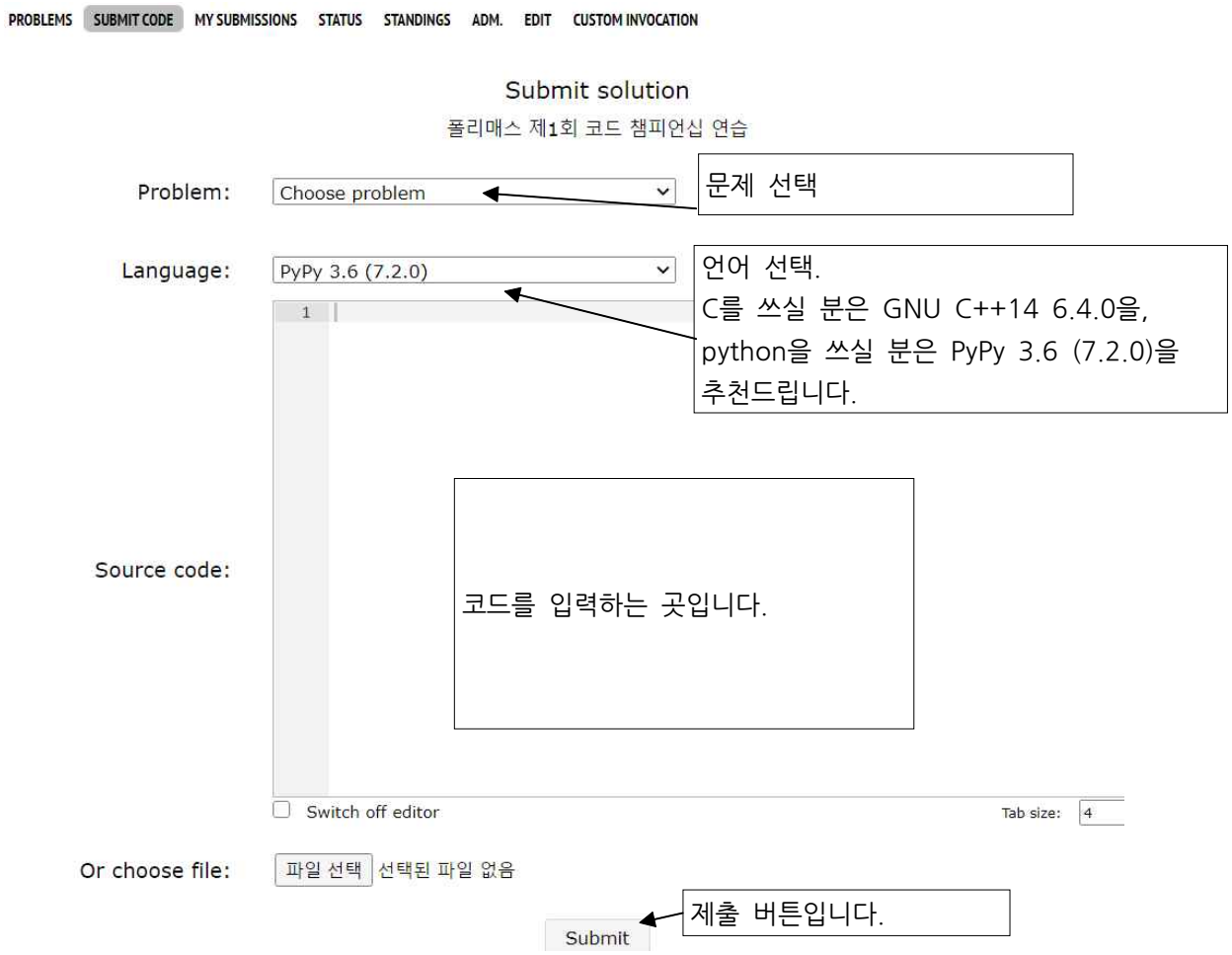
https://wikidocs.net/24

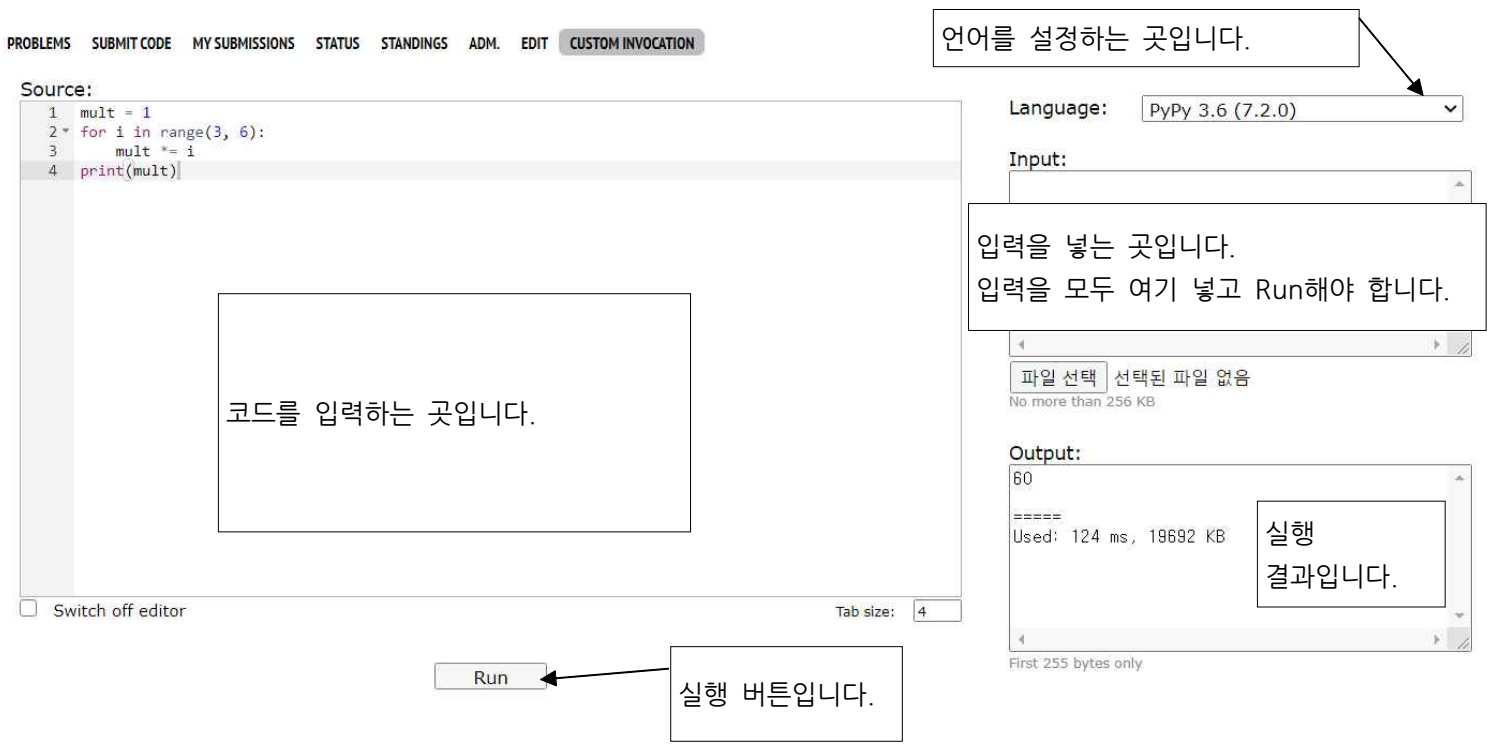
10. 대회 환경 다루기

가장 중요한 부분입니다. 아무리 코드를 잘 짜도, 대회 환경을 다루는 법을 모르면 안 되겠죠? 설명 들어갑니다.



위 사진은 폴리매스 코드 챔피언십 연습 대회를 캡처한 것입니다. 대회 메인 화면에서 중요한 부분은 이 정도입니다.


PROBLEMS
SUBMIT CODE
MY SUBMISSIONS
STATUS
STANDINGS
ADM.
EDIT
CUSTOM INVOCATION
Submit solution
폴리매스 제1회 코드 챔피언십 연습
Problem:
Choose problem
문제 선택
Language:
PyPy 3.6 (7.2.0)
언어 선택.
C를 쓰실 분은 GNU C++14 6.4.0을,
python을 쓰실 분은 PyPy 3.6 (7.2.0)을
추천드립니다.
Source code:
코드를 입력하는 곳입니다.
Switch off editor
Tab size:
4
Or choose file:
파일 선택
선택된 파일 없음
Submit
제출 버튼입니다.


PROBLEMS
SUBMIT CODE
MY SUBMISSIONS
STATUS
STANDINGS
ADM.
EDIT
CUSTOM INVOCATION
언어를 설정하는 곳입니다.
Source:
mult = 1
for i in range(3, 6):
mult *= i
print(mult)
코드를 입력하는 곳입니다.
Language:
PyPy 3.6 (7.2.0)
Input:
입력을 넣는 곳입니다.
입력을 모두 여기 넣고 Run해야 합니다.
파일 선택
선택된 파일 없음
No more than 256 KB
Output:
60
=====
Used: 124 ms, 19692 KB
실행
결과입니다.
First 255 bytes only
Switch off editor
Tab size:
4
Run
실행 버튼입니다.