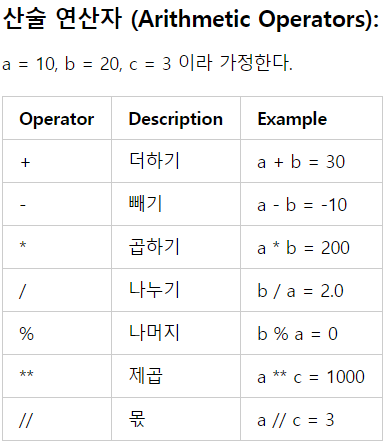
9주차. 연산자(산술, 관계, 논리)

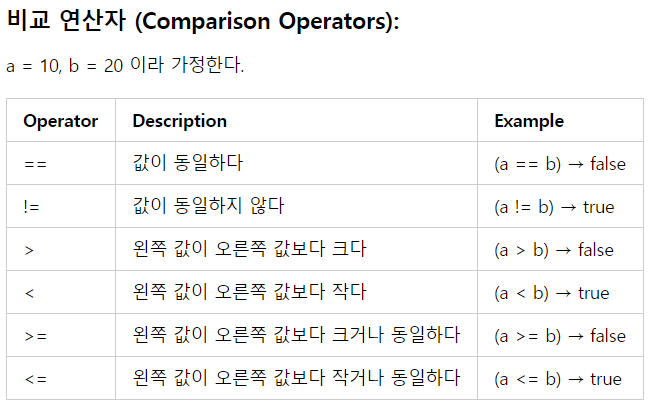
1. 산술 연산자



**%(나머지)**: 17 % 4 = 1, 30 % 7 = 2 등 나누면 생기는 나머지 값을 계산함.

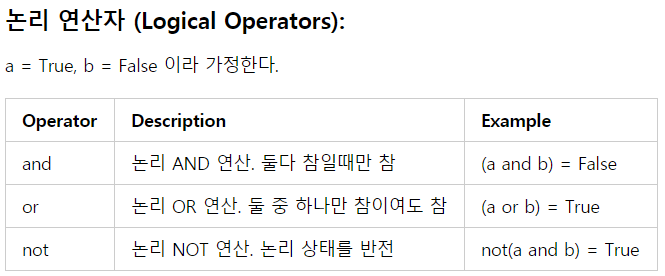
**//(몫)**: 17 // 4 = 4, 30 // 7 = 4, 1000 // 10 = 100 등 나누면 생기는 몫을 계산함.

2. 관계 연산자



|  |  |  |
| --- | --- | --- |
| 파이썬에서 | 현실에서 | 설명 |
| **==** | **=** | 값이 동일하면 True. **대입을 의미하는 =과 헷갈리면 안 됨!!!** |
| **!=** | **≠** | 값이 동일하지 않으면 True. Ex) 10 != 3 → True |
| **>** | **>** | Ex) 8 > 7 → True / 9 > 10 → False |
| **<** | **<** | Ex) 10 < 39 → True / 493 < 428 → False |
| **>=** | **≥** | Ex) 10 >= 10 → True / 10 >= 9 → True / 20 >= 21 → False |
| **<=** | **≤** | Ex) 15 <= 15 → True / 19 <= 20 → True / 9 <= 8 → False |

3. 논리 연산자



|  |  |
| --- | --- |
| 파이썬 코드 | 예시 |
| and | 10 > 9 and 20 < 22 → True / 1 < 2 and 21 < 20 → False |
| or | 1 < 0 or 1 < 2 → True / 10 < 9 or 10 < 8 → False |
| not | not 10 > 9 → False / not 20 < 18 → True |

4. 할당 연산자



수업 시간에 추가로 할당 연산자에 대해서도 다루었는데, 말 그대로 표의 Example 수준 정도의 내용만 충분히 이해 할 수 있으면 상관 없어서, 설명은 생략할게. 모르겠으면 따로 질문해줘.

일단 수업 때 언급된 내용은 이 정도지만, 나중에 충분히 다룰 수 있는 다른 연산자들도 파이썬에 몇 개 더 있어. 출처에 그런 다른 연산자도 있으니 한 번 들어가서 다른 것도 봐봐.

출처: <https://wikidocs.net/1043>

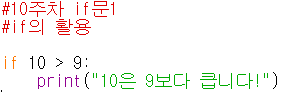
10주차. 선택문(if, if~else, if~elif~else, 중첩 if문)

**if**는 **조건문이 참일 경우, if문 블록 안에 있는 코드를 실행**시키는 명령어야. if는 쉽게 생각해서 ‘만약 (조건문)이 참이라면~’ 의 의미를 지니고 있는 거지. if (조건문): 이 그 형식이고, 다음 줄부터 if문 위치에서 한 칸 들여쓰고(Tab 키 혹은 스페이스바 4번) 코드를 작성하면 그 코드는 ‘if문 블록 안의 코드’가 돼. (참고로 우리가 쓰는 IDLE Editor에서는 :를 쓰면 다음 줄은 들여쓰기가 자동으로 적용되니 걱정 안 해도 돼.)

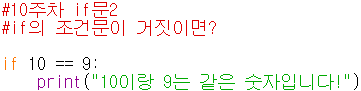
**else**는 **if문이 만약 거짓일 경우, else문 블록 안에 있는 코드를 실행**시키는 명령어야. 쉽게 생각하서 if문과 연계해서 ‘아니면~(if문의 조건문이 거짓이라면~)’의 의미를 지니고 있는 거지. else: 가 그 형식이고, 반드시 if문과 연계 되어야해. 밑도 끝도 없이 ‘아니면~’ 하면 컴퓨터는 “뭐가 아닌건데??” 하면서 이해를 못 하겠지?

**elif**는 **else와 if가 결합한 형태**야. ‘if문이 거짓이야? 그렇다면 이 elif문이 참이면 이 elif 블록을 실행해 줘~’의 의미로 쓰여. if문이 거짓일 경우, 다음에 실행되는 새로운 if문이라고 생각하면 돼. 이 역시 처음에 if로 시작하면서 연계가 되야겠지? elif (조건문): 이 그 형식이야.

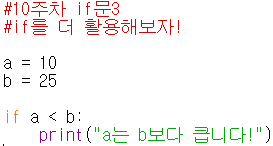
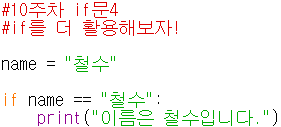
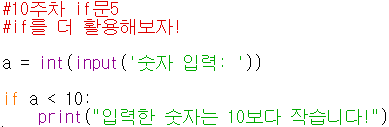
위에 설명만 듣고는 감이 안 오지? 여러 예시를 보면서 저 선택문이란 걸 이해를 해보자.



다음 예시를 보면, (10 > 9) 는 맞는 말이니까 True겠지? 그러면 이제 저 파일을 실행하면 Shell창에는 ‘10은 9보다 큽니다!’ 라는 메시지가 뜰거야. 그런데 만약에 이러면?

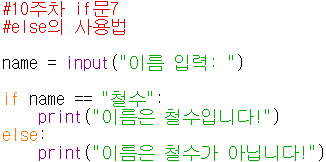
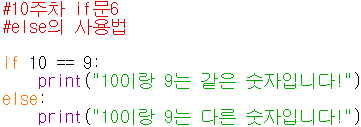


10이랑 9는 같은 숫자가 아니지? 저 파일을 실행하면 if문의 조건문은 거짓이니까, 이후 print문은 실행시키지 않겠지? 실제로 저거 실행하면 Shell창에서는 그냥 아무 것도 출력하지 않을거야.

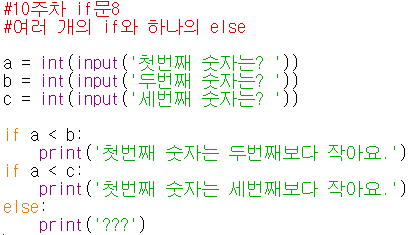
  

우리는 if문3과 같이, 저렇게 변수를 활용해서 if문을 작성할 수도 있고, if문4와 같이 숫자뿐만 아니라 다른 자료형도 if문의 조건문으로 활용할 수 있다는 걸 알 수 있어. 그리고 더 나아가 if문5처럼 우리가 input문을 직접 받은 내용을 if문의 조건문에 활용할 수도 있다는 것을 알 수 있어.

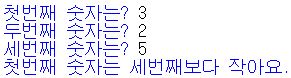
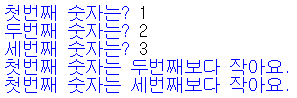
이 예시들은 극히 단순한 내용에 지나지 않고, if문은 훨씬 무궁무진하게 활용할 수 있는 명령어니까, 기말 실습을 배우고, 개인적으로도 파이썬 코딩을 해보면서 그 용례를 익혀나가자.

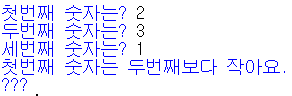


**else**문은 아까 말했듯이, **if문과 연계해서 if문이 거짓일 경우, 실행한다는 명령어**야. if랑 else는 사진처럼 같은 들여쓰기(같은 가로위치)에 있어야 서로 상호작동 해. 그리고 중간에 같은 들여쓰기 위치에 if문이 더 있다면, else문은 나중에 쓰인 if문과 상호작동 해.

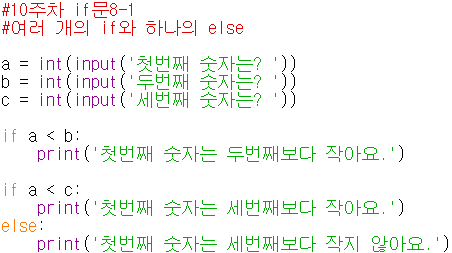


만약 else가 나중의 if문과 상호작용을 한다는 것을 모른다면, if문8처럼 적어놓고, 이 파일을 실행해서 일일히 상황을 대입해봐야 어떻게 작동되는지 알겠지?

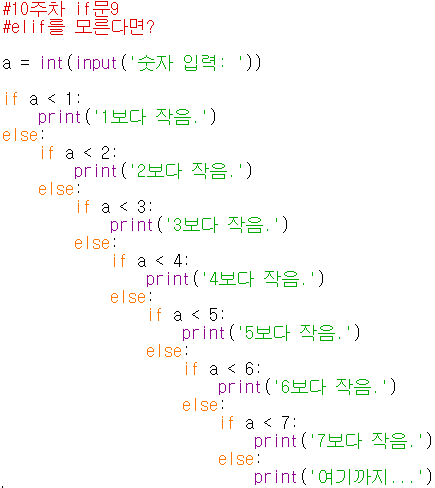




위 사례들을 보면, a < b 일 경우에는 ‘~두번째보다 작아요’ 프린트문을 실행하는데 반해, a < b가 아닐 경우에는 아무 것도 출력하지 않는 것을 알 수 있지?(첫번째 사진과 두번째 사진 비교해 봐.) 그리고 a < c 일 경우에는 ‘~세번째보다 작아요’ 프린트문을 실행시키고, a < c가 아니면 ‘???’를 출력하는 것을 알 수 있지?(첫번째 사진과 세번째, 네번째 사진 비교해 봐.) 이걸 보면 else문은 처음 if문이 아닌, 마지막에 쓰인 두번째 if문과 상호작용한다는 것을 알 수 있겠지? 우리는 이제 else문의 이런 특징을 알았으니, if문8을 더 보기 쉽게 if문8-1처럼 바꿀 수 있을거야.



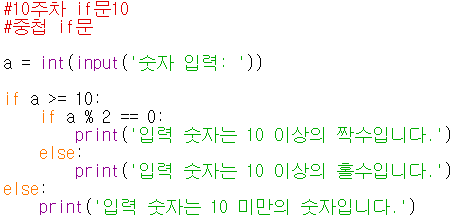
우리는 이제 if문과 else까지 배웠어. 하지만 이 두개만 알고 있다면, if문9 같이 깊숙히 파고 들어가는 복잡한 코드를 만들 수 밖에 없는 상황이 생길 수 있어.



if문9를 보면, if문이 거짓이면, else문을 실행하고, 그 else문 안의 if문을 실행하고, 그게 거짓이면 else문 안의 else문을 실행하고…. 를 반복하고 있지? 그런데 else문을 이용하려면, 실행코드를 들여쓰기를 해야하는데, 그게 계속 반복되니까 코드가 안쪽으로 파고 들어가지? 저러면 나중에 코드를 보기 매우 불편할거야.

다행히 파이썬에서는 else(아니면~)와 if(만약~)을 합친 **elif**(아니면 만약~)라는 명령어를 지원하고 있어. elif를 활용하면 if문9의 코드를 if문9-1 처럼 만들수 있어. else: 와 다음줄의 들여쓰기+if(조건문)를, elif (조건문): 형태로 쉽게 바꾸는 걸 알 수 있지? 더 쉽게 생각하면, 원래 코드에서 el 이후로 if 전까지 다 지우면 만들 수 있는거야.

이제 마지막으로 **중첩 if문**이야. 말은 어려워 보이지만, 그냥 if문 안에 if문 몇 개 더 있는거야.



a가 만약에 10 이상이라면, a >= 10은 참이니까 그 안의 블록을 실행하겠지? 그러면 이제 저 a % 2 == 0 이 참인지를 판별하고, 그것에 따라 출력문이 달라지겠지? 반면에 a가 10 미만이라면, a >= 10은 거짓이니까 if문의 블록은 실행하지 않겠지? 그러면 a % 2 == 0 이런 거 참거짓 판별할 필요 없이 컴퓨터는 바로 맨 아래 else문을 실행하게 될거야.

if문은 매우 활용성이 높으니, 자신이 직접 if문을 활용해서 코드를 한 번 만들어 보는걸 추천할게.

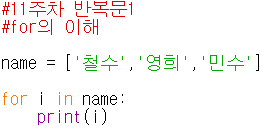
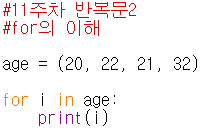
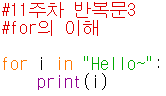
11주차. 반복문(for, while, break, continue)

**for**은 반복문 종류 중 하나이고, for (변수) in (리스트, 튜플, 문자열, range함수 등..): 의 형태로 쓰여. in 뒤에 오는 자료형이나 함수의 각각의 원소는 (변수)가 되고, 이 각각의 원소를 처음부터 끝까지 (변수)에다 대입해서 반복문의 블록을 각각 실행하게 하는 반복문이야. 무슨 소리인지 모르겠지? 이걸 쓰는 나도 이해하기 힘든 설명 같다고 느껴. 이건 특히 예시를 보면서 많이 익혀가자.

**while**은 반복문 종류 중 하나이고, **조건반복문**이라고도 해. while (조건문): 의 형태로 쓰이고, 조건문이 거짓이 될 때까지 반복문의 내용을 실행하는 거야. 중간고사 범위 실습 시간에, while True: ~~ 라는 걸 몇 번 봤지? 조건문에 쓰인 True는 참이라는 의미니까, 이건 그냥 무한정 반복실행하겠다는 의미로 쓰인 함수인거였어. 더 구체적으로는 얘도 예시를 보면서 어떻게 쓰이는 지 익혀가자.

**break**는 반복문을 중단하는 명렁어야. 사용법은 반복문 블록 코드 안에 특정 상황에 실행하도록 넣으면 되는건데, 이 친구도 예시를 보면서 익히자.

**continue**는 이후 반복문 블록을 무시하고, 반복문의 처음으로 돌아가 다음 반복을 실행하라는 의미야. 사용법은 얘도 반복문 블록 코드 안에 특정 상황에 실행하도록 넣으면 되는데, 이 친구 역시 예시를 보면서 익히자.

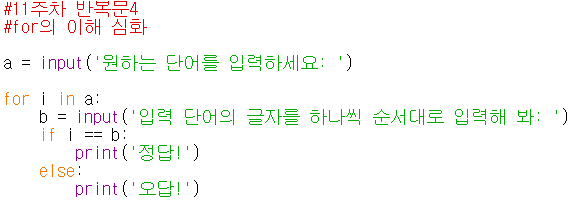
설명하기 앞서, 설명할 때 앞으로 (변수)칸에 i를 많이 넣을거야. for문의 이 변수는 for문에서 처음 선언하는 변수인데, 긴 이름으로 쓰면 보기 불편하니 대부분의 사람들은 i로 표현하고는 해. 실제로 인터넷에 for문에 대해 검색하면 for i in ~ 의 형태를 많이 볼 수 있을거야. 다른 거로 쓰고 싶으면 개인적으로 코드를 만들 땐 다른 변수를 써도 되니까 신경 쓰지 않아도 돼.

반복문1을 먼저 보자. name은 세 명의 이름이 들어간 리스트지? 저걸 이제 for문에 넣으면 어떻게 될까? 저 반복문을 처음 실행할 경우, i는 이제 리스트의 첫 번째 원소, 즉 ‘철수’가 돼. 그러니까 i = ‘철수’ 가 되는거야. 그 상태에서 print(i)를 하면 ‘철수’가 출력되고, 반복문이 끝나지? 그러면 이제 다음 원소에 대한 반복문을 시작해. 그러니까 이제 i = ‘영희’ 가 되는거야. 그리고 print(i)를 하면 ‘영희가’ 출력이 되겠지? 그리고 반복문이 끝나니까, 다음 원소에 대한 반복문을 실행하게 될거야. 그러니까 이제 i = ‘민수’가 되는거야. 그리고 print(i)를 하면 ‘민수’가 출력되고, 반복문은 종료가 될꺼야.

아직 이해가 안 되면 반복문2도 한 번 보자. age는 네 숫자가 들어간 튜플이지? 그리고 저 for문을 실행하게 되면, 처음에 i = 20 이 되고 20을 출력하고 첫번째 반복이 종료 될거야. 그리고 두 번째로 i = 22 가 되고 22를 출력하고 두번째 반복이 종료 될거야. 이렇게 되면 다음에는 21을, 그 다음에는 32를 출력하고 반복문이 종료가 되겠지?

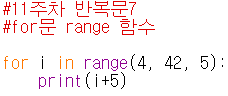
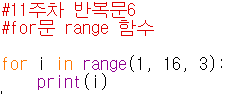
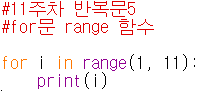
마지막으로 반복문3을 보자. 이번에는 “Hello~”라는 문자열이야. 저 문자열의 첫번째 원소(글자)는 H지? 그러면 반복문을 실행하게 되면 i = ‘H’ 가 되고, ‘H’를 출력하고 첫번째 반복이 종료되겠지? 그리고 다음으로 i = ‘e’가 되고 ‘e’를 출력하고 두번째 반복이 종료될거야. 다음에는 i = ‘l’이 되고, ‘l’을 출력하고 세번째 반복이 종료 될거야. 다음에는 i = ‘l’이 되고, ‘l’을 출력하고 네번째 반복이 종료 될거야. 그리고 다음에는 i = ‘o’가 되고, ‘o’를 출력하고 다섯번째 반복이 종료 될거야. 그리고 마지막으로 i = ‘~’가 되고, ‘~’를 출력하고 마지막 반복이 종료되면서, 반복문 전체가 종료가 될거야. 이걸로 어느 정도 for문의 원리가 이해가 가니? 저 파이썬 코드는 같이 올려줄 테니, 실제로 한번 실행해보고 그 결과물을 보는 걸 추천할게.

for문의 원리가 이해갔으니, 단순한 print(i)에서 벗어나서, 몇 가지 코드를 더 활용해보자.

먼저 a라는 변수에 input함수를 입력 받고, 문자열인 a의 원소 하나하나가 이제 반복문의 i의 값이 되겠지? ‘hello’를 입력했다면 첫번째 반복 때 i = ‘h’가 되겠지? 마찬가지로 글자 순서대로 i의 값이 변하게 될거야. 오른쪽 사진 보면 h, e, l, l, o 순서대로 입력하니 모두 정답이지? 이렇게 print(i) 말고도 이런 프로그램을 만들 때, for문을 활용하면 도움이 될 수 있어.

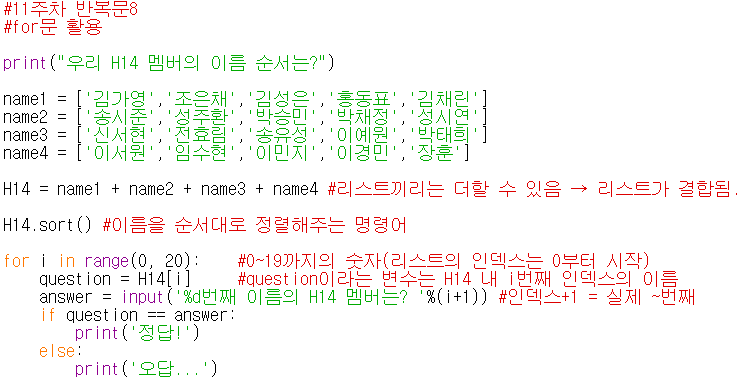
이제 아까 for문을 소개하면서 언급했던 **range 함수**를 설명할게. 일단 range 함수는 for문의 in 뒤에 range(숫자1, 숫자2, 숫자3)의 형태로 쓸 수 있어. 이 range 함수는 (숫자1)이상 (숫자2)미만의 정수들을 불러오는 역할을 한다고 생각하면 돼. 예를 들면, for i in range(1, 11): 이라면 첫번째 반복에서 i = 1, 두번째는 i = 2, … , 아홉번째는 i = 9, 열번째는 i = 10. 이렇게 순서대로 반복문의 변수가 지정된다고 생각하면 돼. (숫자3)이 먼저 생략 된 이후면, (숫자1)이 0일 때 (숫자1)은 생략 가능해. 그리고 생략 가능한 (숫자3)은 숫자의 뛰는 간격이라고 생각하면 돼. (숫자3) 기본값은 1이야. range(1, 11)이라면 1,2,3,…,9,10 순서대로 나오는 것이고, 만약 range(1, 11, 2)라면 1, 3, 5, 7, 9 순서대로 나오게 되는거지.



앞에서 배운 내용을 바탕으로, 다음 코드를 실행하면 어떤 내용이 출력될 지 상상해보고, 그 생각과 실제 출력 결과가 일치한 지 비교해보자.

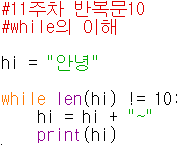
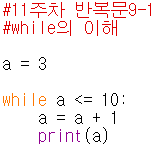
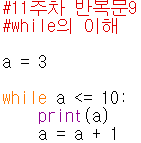
(검토하니까 **range(len(~))**을 설명하는 걸 빠뜨려서, while함수 이후에 p.9에 추가로 설명할게.)

마지막으로, 앞에서 배웠던 for문 내용들을 바탕으로, 이전까지 배웠던 내용까지 활용해서 for문으로 코드를 만들어보자. **모든 실습 내용, 특히 기말고사 파트 내용은 활용성이 풍부한 명령어, 함수 등을 배우니까, 꼭 자신이 학습내용을 활용해서 무언가를 만들어보는 것이 중요할 거 같아.** 나는 이런 코드를 만들어 봤어.



꼭 자기만의 코드를 만들어보자~. 뭘 만들지 모르겠으면 기초적인 반복문1,2,3,5,6,7 처럼 기본적인 거 부터라도 만들어보고 결과가 어떻게 나오는 지 비교해보자. 그리고 만들고 싶은 게 있는데 ‘이런 상황을 명령해주는 명령어는 없나?’ 싶으면 나한테 물어보면 아는 선에선 다 알려줄게. 일단은 모르고 어렵더라도, 무언가 뿌듯한 결과물이 나오면 재밌어지면서 계속 하고 싶거든 ㅎㅎ

다음은 while이야. **while**은 아까 말했듯이 조건반복문이라고도 불리고, while (조건문): 의 형태에서, 조건문이 거짓이 될 때 까지 반복문을 계속 실행하는 거야. 바로 예시를 보자.

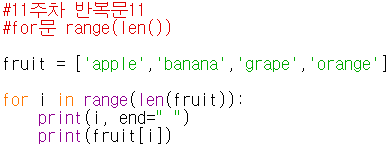


먼저 반복문9를 보자. 먼저 a=3은 a라는 변수는 3이라는 정수라고 정한거지? 그리고 이제 while문으로 들어가면 조건문은 a<=10, 즉 ‘a가 10 이하면~ (a가 10을 초과하기 전까지~)’라는 의미겠지? 처음에 a=3이라고 했으니까 a≤10은 만족했으니, 저 조건반복문을 실행하겠지? 그러면 처음 반기는 코드는 print(a)니까 3을 출력하겠고, a는 이제 앞으로 a+1(=4)가 되고 반복문이 종료되겠지? 그러면 이제 a=4니까 다시 조건문(a≤10)을 만족하니, 다시 실행하겠지? 그러면 4를 출력하고, a는 이제 5가 되고, 아직 a≤10를 만족하고…. 이렇게 반복하다 보면 a=10일 때 10을 출력하고, a=11이 되면서 반복문이 종료되겠지? 그런데 보니까 이제 a≤10이 아니니까 조건문을 만족하지 않지? 그러면 더 이상 조건을 만족하지 않으니까 while 반복문을 종료하게 돼.

반복문9-1을 보자. 아까 반복문9와 거의 비슷한데, a = a+1이 print(a)의 위에 있지? 그러면 결과가 9-1 옆처럼 바뀌는 건 이해가 되지? 출력하고 +1 하는게 아니라 +1하고 출력을 하는 거니까. 그런데 a = 10일 때 반복문을 실행하면 a = a+1 에서 a = 11이 되서 조건문을 만족하지 않게 되는데 왜 11을 출력할까? 그 이유는 저 조건반복문(while)은 시작할 때만 조건을 만족하면, 그러니까 여기선 시작할 때 a≤10이기만 하면, 조건문 중간에서 그 조건을 만족하지 않게 되더라도 반복문을 계속 진행하게 돼. 그래서 a = 11인데도 print(a)를 해서 11을 출력한거지. 어떻게 보면 당연하다고 볼 수 있는 거긴 한데, 약간 헷갈릴 수도 있는 내용 같아서 이야기 해봤어.

반복문10을 보자. **len()**은 리스트나 문자열 등의 길이를 구하는 함수야. **for문 range함수에서도 중요하게 쓰이는데**, 생각해보니 까먹고 설명을 안해서 while 다 설명하고 한 번 알아보자. 일단 여기선 그냥 len()의 의미만 이해하면 되고, 그러면 처음 반복문을 시작할 때 hi = “안녕”이니까 len(hi)는 2겠지? 그러면 조건문 len(hi) ≠ 10을 만족하니 반복문을 실행하겠지. 변수 hi는 문자열이고 ‘~’도 문자열이니, 문자열끼리 +면 붙여쓴다는 거 기억나지? 이제 hi = “안녕~”이 되고 print(hi)로 “안녕~”이 출력되고 반복문 블록 실행이 끝날거야. 그리고 결과처럼 쭉 실행되다가 hi = “안녕~~~~~~~~”이 되고 출력한 다음에 종료된 다음에, 조건반복문을 다시 실행시키려고 보니 len(hi) == 10이니까(=문자열 길이가 10이니까) 조건문을 만족하지 않지? 그래서 조건반복문을 종료하게 될 거야.

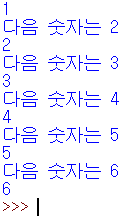
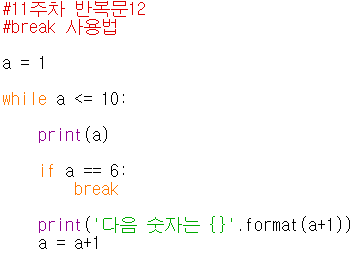
**len()**은 아까 말했듯이 길이를 구하는 명령어야. 여기서 len()의 괄호 안에는 리스트나 문자열 등을 넣을 수 있어. 이 명령어는 많은 경우에 쓰이는데, 특히 **for문 range함수에서도 많이 쓰이는 명령어**야. range(숫자1, 숫자2(, 숫자3))의 형태로 range를 활용한다고 했었지? 숫자1 이상 숫자2 미만의 정수를(그리고 숫자3을 추가로 입력하면 숫자3의 간격만큼 띄엄띄엄) 불러온다고 했던 게 기억날거야. 그리고 (숫자1)이 0일 경우에는 생략이 가능하다고도 언급했을거야. range 함수에서 len()을 활용하는 형식은 **range(len())**인데, 여기서 그 생략이 가능한 걸 활용했다는 것을 알 수 있어. 생략을 안하면 range(0, len())으로도 쓸 수 있다는 거지. 이런 range(len())은 **for문의 range(len())는 리스트나 문자열의 그 길이만큼 for문을 실행하게 해**. 예시를 보면서 이해해보자.

(첫번째 print문에서 end=” ”는 print문을 출력하고 자동으로 줄바꿈하는 걸 띄어쓰기로 바꾸는 명령어야. 기본 값(=아무것도 안쓰면)은 \n, 그러니까 줄바꿈이야. 중간 때 실습시간에 이미 다루었던 내용이니 모르겠으면 개인적으로 질문해 줘.)

fruit라는 변수의 리스트의 길이는 4지? 그러면 len(fruit)는 4일거야. 그러므로 range(len(fruit))는 range(4), 혹은 range(0, 4)와 같다는 걸 알 수 있겠지? 그러면 i=0, i=1, i=2, i=3 이 순서대로 총 4번 for문을 실행하게 될건데, **i의 개수가 fruit의 길이와 동일**하지? 그러니까 range(len())은 길이만큼 실행한다는 걸 알 수 있을거야. 그리고 반복문11을 보면 i의 값은, 리스트 내 값을 순서대로 나열했을 때, 그 값들의 인덱스하고도 일치하는 걸 볼 수 있어. 그래서 반복문11 같이 fruit[i]를 출력하라고 해서 리스트 내 모든 값을 차례대로 출력하는 방법으로도 활용할 수 있겠지?

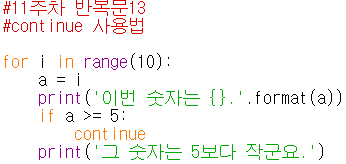
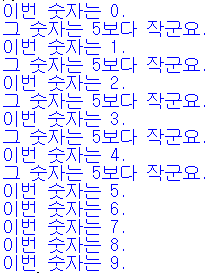
다음은 break에 대해서 알아보자. **break**는 for, while 등의 반복문을 다 실행하지 않고 중간에 완전히 종료(중단)하는 명령어야. 활용방식은 예시를 보면서 익히자.



일단 while문 중간에 a가 6이라면 만족하는 if문 안에 break가 있지? 보통 while문 안에 break문을 바로 넣으면 반복문을 키자마자 바로 반복문이 끊기게 되니까 사용하지 않고, 저렇게 while문 안에 조건문을 넣는 방식으로 해서, 특정 조건에서만 break가 실행되도록 많이들 활용하고는 해.

반복문12의 while문을 더 제대로 분석해보자. 처음에 a=1이니 a≤10은 만족하니 조건반복문은 실행될 거고, print(a)니까 1이 출력되겠지? 그리고 a는 6이 아니니 if문은 작동 안하고, ‘다음 숫자는 2’라고 출력 되고, a는 이제 2가 되겠지? 그런 방식으로 반복문이 반복되다가, a=6일 때, print(a)가 실행된 이후 if문을 만족해서 while문에 break가 걸렸지? 그래서 반복문이 종료되면서 그 이후의 ‘다음 숫자는 7’이라는 문자열은 출력되지 않았고, a≤10인데도 불구하고 조건반복문이 작동하지 않은 것을 볼 수 있을거야. 이렇게 break는 반복문을 완전히 종료시키는 명령어라는 걸 알 수 있겠지?크게 설명할 내용은 없고, 자신이 직접 break를 활용한 반복문을 만들어보면서 익히는 걸 추천할게.

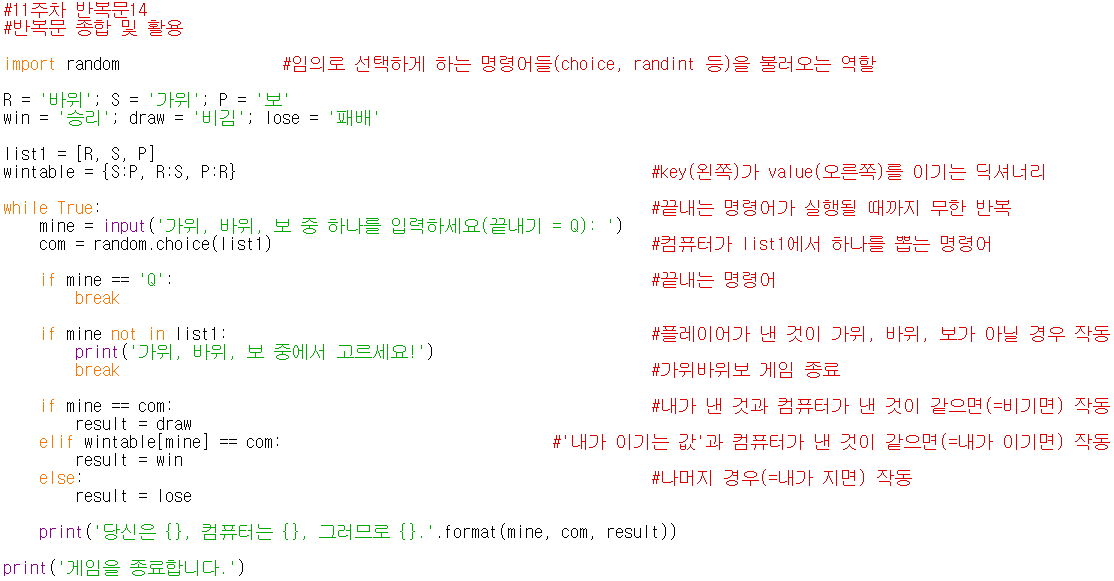
다음은 continue에 대해서 알아보자. **continue**는 for, while 등의 반복문의 실행 블록에서, continue 명령어 이후 남은 코드들을 더 실행시키지 않고, 다음 반복을 진행하게 하는 명령어야.

a가 5 이상일 경우 continue를 진행하는 if문을 넣은 for문이야. 실행 결과문을 보면 if문을 만족하지 않을 때는 continue 이후의 반복문, 그러니까 print(‘그 숫자는 5보다 작군요.’)를 출력하지만, 그 이후로 if문을 만족할 때는 이후의 print문을 무시하는 걸 볼 수 있지? 이렇게 continue는 반복문 내에서, continue 이후의 코드를 무시하고 다음 반복을 진행하는 명령어인 것을 알 수 있었어.

마지막으로 11주차의 모든 내용을 종합해 본 코드를 만들어 보면서 끝내보자. 항상 강조하지만 자신이 직접 만들어 보는 게 정말 중요한 거 같아. 나도 이렇게 내용 정리하면서 직접 코딩을 쳐보고, 중간에 오류가 생기면 오류의 원인을 찾고, 설명을 빠뜨리지 않기 위해 개념을 찾아보고 하니 얻어가는 게 많은 거 같다고 생각했거든. 꼭! 쉬운 거라도 평소에 만들어 보는 연습을 하기!

나는 이런 코드를 만들어 볼 수 있다고 생각했어. 반복문뿐만 아니라 지금까지 배웠던 여러 개념, 어떤 건 안 배운 개념도 쓰였는데, 이 코드 중에서 배운 내용인데 이해가 안 되는 부분이 있다면 언제든지 나한테 물어 봐. 안 배운 내용도 궁금하면 언제든지 물어보면 설명해줄게.

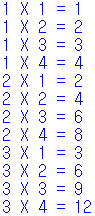
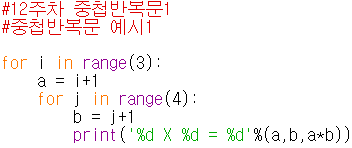


random 외장함수 빼고는 우리가 지금까지 모두 배웠던 내용이야. 컴퓨터의 랜덤 선택 없이 1P, 2P가 하나씩 선택하는 방식으로 게임을 만들면 random 외장함수도 필요 없겠지? 그리고 if문이 조금 많아지겠지만 딕셔너리를 모르는 상태에서도 충분히 가위바위보 프로그램을 만들 수 있을 거 같아. 한 번 시간 나면 저 반복문14를 참고해서 1P, 2P를 차례대로 정해서 가위바위보 승패를 가르는 프로그램, 그리고 딕셔너리가 없는 가위바위보 게임을 만들어 보는 건 어때? 11주차는 여기에서 마무리를 지을게.

12주차. 중첩 반복문(nested for)

우리는 11주차에 반복문에 대해서 배웠는데, 만약 반복문 안에 반복문이 있으면 어떻게 작동할까? 일단 우리는 앞으로 반복문 안에 반복문이 있는 것을 **중첩 반복문**이라고 부르자.

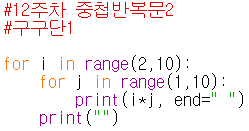
중첩 반복문은 큰 반복문이 한 번 실행될 때, 내부의 작은 반복문은 모두 실행되고 종료해. 그러니까 큰 반복문이 총 10번 실행되는 반복문이고, 작은 반복문 자체는 총 5번 실행되는 반복문이라고 한다면, 실제로 이 코딩을 실행할 때 작은 반복문은 5X10 = 50번 실행되게 되는거야. 이거 역시 백문이불여일견, 예시를 보자.



i가 변수인 for문, 그러니까 외부(큰) 반복문은 i=0,1,2 총 3번 시행되는 반복문이지? 그리고 그 때마다 각각 a=1,2,3이고. 그리고 내부(작은) 반복문은 j가 for문의 변수이고, j=0,1,2,3 총 4번 시행되는 반복문인 걸 알 수 있을거야. 물론 b는 그때마다 각각 1,2,3,4일 거고.

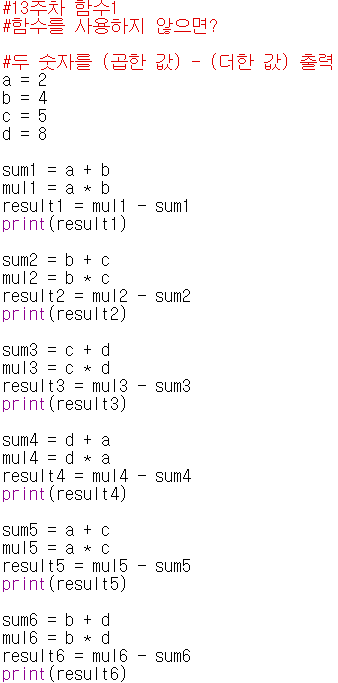
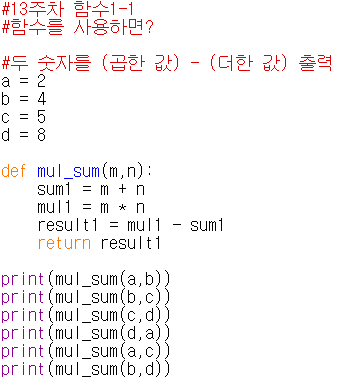
이제 저 코드를 실행했다고 해보자. 그러면 먼저 i for문에서 i=0, a=1인걸로 시작하겠지? 그러고 보니 내부에 j for문이 있어서 그 for문을 시행해야할거야. 그러면 이제 j=0, b=1이겠고, print문을 출력해서 ‘1 X 1 =1’를 출력했을거야. 그리고 j for문의 첫번째 실행이 종료되고 j=1, b=2가 되겠지? 하지만 i=0, a=1인건 여전히 변함이 없어서 ‘1 X 2 = 2’가 출력될거고… 그렇게 반복하다보면 j=3, b=4 일때 j for문이 완전히 종료될거야. 그러면 그제서야 i for문의 다음 반복이 시작되는 거야. 이제 i=1, a=2가 되겠지? 그리고 j for문을 다시 작동해서 j=0~3을 한번 씩 또 시행하고 종료되고, 또 i=2, a=3일 때 j=0~3을 진행하면서, j for문만 놓고보면 3(i for문)X4(j for문) 총 12번 반복을 진행하게 되는거야.

중첩반복문1을 보니, 저걸 더 확장하면 ‘구구단’을 만들 수 있을 것 같다는 생각이 들지? 나도 처음 중첩반복문을 접할 때 구구단 코드를 접했었는데, 구구단이 중첩반복문의 정말 좋은 예시라고 생각해서 아래에 사진을 첨부해놓을게. 12주차 설명은 일단 여기까지고, 실습 때 추가로 어려운 내용이 나오면 추가로 설명을 준비해놓거나 할게.



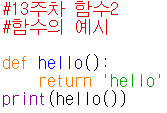
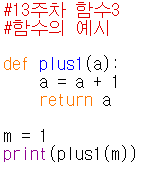
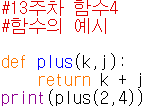
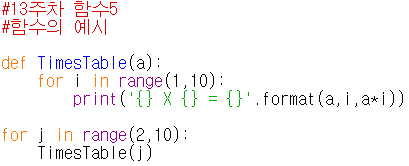
13주차. 함수(def), 14주차. 함수 활용

만약 우리가 어떤 일정 코드를 여러 번 사용하고 싶으면, 그걸 일일히 매번 복사해서 써야할까? 도입이지만 하나의 예시를 보면서 그게 얼마나 불편한지 보자.

딱 봐도 코드 줄 수부터 다르지? 간단한 식에서는 몰라도, 점점 복잡한 코드를 반복적으로 많이 실행해야 할 경우 빛을 발하는 파이썬의 기능이 바로 함수야.

**함수**는 일정 코드를 한 덩어리로 만들고, 사용할 경우 그 덩어리 코드를 한 번에 실행하게 해주는 기능이야. 형식은 def 함수명(입력인수): 이고, 그 아래로 일제히 실행하고 싶은 코드를 함수 안에 작성하면 돼. 바로 예시를 보자.

함수2, 3, 4를 보면 return이라는 것이 보이지? **return**은 def에서 많이 쓰이는 명령어인데, return 옆에 적은 값을 함수의 결과로 돌려주는 거야. 예를 들면 함수4를 보면 a, b를 입력 인수로 입력하면, 그 두 값을 더한 것을 함수의 결과로서 돌려주겠지. 그리고 그 print문에 함수를 넣으면 함수의 결과를 출력하겠지? 주의해야할 것은 return은 단순히 값을 돌려주는 것이지, print 마냥 값을 출력까지는 안 해준다는거야. 그래서 함수2~4에서는 마지막에 출력까지 하려고 print문을 추가로 쓴 거지. 함수 하나로 출력까지 하고 싶으면 함수5처럼 함수 안에 return 말고 print문을 넣으면 돼.

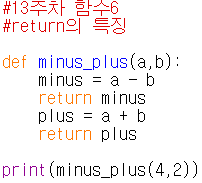
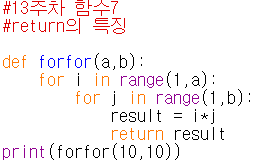
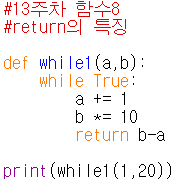
그러면 본격적으로 함수2~5가 어떻게 쓰인 함수인지 알아보자. 먼저 **함수2**를 보면 괄호 안에 아무 것도 없지? 저런 함수는 입력 인자, 그러니까 함수에서 받고 싶은 입력값이 없다는 거야. ‘hello’ 하나 출력하는 함수인데, 굳이 숫자 같은 다른 자료가 필요하진 않겠지? 그런 경우는 저렇게 입력 인자를 비워놔도 돼. hello()를 실행하면 그 함수는 ‘hello’라는 문자열을 결과로 갖고 있는 거고, hello()를 출력한다고 하면 그 함수의 결과값인 ‘hello’가 출력되는 거지.

**함수3**을 보자. plus1이라는 함수는 입력 인자를 하나 받는다는 것은 이해되지? 저기서 쓰인 a는 함수의 입력인자를 의미하는 거야. 실제로 함수를 쓰고 싶을 때는 a가 쓰인 자리에 다른 변수나 숫자 등을 입력해서 사용할 수 있어. 그러니까 plus1(m)은 m에다가 1을 더해서 그 m값을 결과값으로 돌려준다는 거지. 그러니까 print(plus1(m))은 2가 되겠지?

**함수4**를 보자. plus라는 함수는 입력 인자를 두 개 받는 함수야. 입력인자는 아무거나 편한 거 쓸 수 있다는 점도 알아두고. 첫 번째 입력한 인자와 두 번째 입력한 인자를 더해서 그 값을 돌려주는 함수니까, plus(2,4)는 6이라는 결과값을 가지겠지? 출력하면 6이라고 나올 거고.

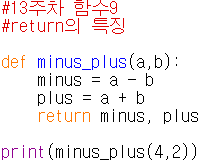
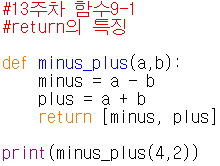
**함수5**를 보자. 아까 말했듯이, 함수 안에 print가 있으면 굳이 print문을 다시 사용하지 안해도 출력값이 나온다는 건 이해했을거야. 그리고 저 값이 어떻게 나올지는 예상이 가지? 중첩 for문도 배웠으니까 어떻게 나올 지 값을 상상해보고 한 번 직접 실행해보는 걸 추천할게.

이제 return의 한 가지 특징에 대해서 알아보자. return은 마치 반복문의 break 마냥 뒤에 남은 코드들을 무시하고, 결과값을 돌려주고 함수를 끝내는 특징이 있어.

자세한 설명은 생략하고, 함수6에서 return값은 minus만 나오겠지? 뒤의 코드들이 뭐든간에 값을 돌려준다고 했으니까. 그리고 함수7에서는 return값으로 1과 1을 곱한 1만 한 번 출력하고 끝내겠지? return은 break마냥 값 돌려주면 for문이고 뭐고 땡이라고 했으니까. 함수8도 동일하게 200-2인 198 하나 출력하고 끝내겠지?

근데 함수6마냥 **만약 두 개 이상의 결과값을 받고 싶은 함수를 만들고 싶으면 어떻게 할까?** 저렇게 return을 두 개 쓰는 것은 안 될거야. return은 한 번 작동하면 함수를 끝내니까 말이지. 하지만 return 한 번에 두 개 이상의 값을, 정확히는 두 개 이상의 값을 담은 하나의 자료(튜플, 리스트 등)를 결과값으로 돌려주면 두 개 이상의 값을 동시에 결과값으로 돌려줄 수 있겠지?

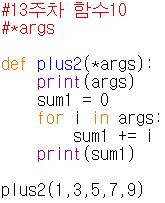
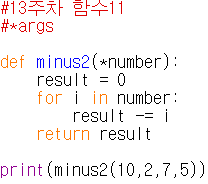
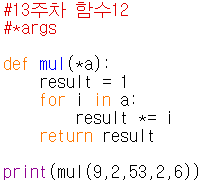
 

**함수9에서 함수 minus\_plus의 결과값은 (2, 6)이라는 튜플로 돌려주겠지?** 여기서 잠시 복습하자면 튜플은 소괄호를 사용하거나, 괄호 없이 반점만으로도 표현할 수 있었어. 보통 return으로 결과값을 여러 개 돌려주고 싶다고 하면, 사람들은 리스트보다도 이걸 훨씬 더 많이 사용하고는 해. 나도 이유는 정확하게 모르겠지만, 일단 [] 이렇게 대괄호를 쓸 필요도 없고, 내용의 값이 바뀌지 않는 튜플이 더 안정적이어서 그런 것 같아. 내가 지금까지 본 모든 설명들에도 함수 9-1마냥 return을 리스트로 쓰는 경우는 한 번도 못 봤어. 물론 리스트를 return에 사용할 수 있으니까 굳이 너무 연연하진 않아도 될 거 같아.

**함수9-1에서 함수 minus\_plus의 결과값은 [2, 6]이라는 리스트로 돌려주겠지?** 이렇게 여러 값을 return으로 돌려주려면, 하나의 자료에 담아서 return을 하고, 그 결과가 담긴 자료를 슬라이스([0:3] 등) 같은 걸 활용해서 자유자재로 사용하면 될 거 같아.

(이 아래로는 14주차에 나올 것 같은 내용을 정리했는데, 나도 정확히 뭐가 나올지는 잘 감이 안 잡혀서 일단 \*args하고 함수 내 변수의 유효 범위 등을 정리해 봤어. \*args는 아마 안 나올 것 같은데 내용이 크게 어렵지 않아서 일단 설명해놨어. 그리고 함수 내 변수의 유효 범위, 입력 인자의 유효 범위 같은 내용은, 설명을 적는 나도 아직까지도 많이 헷갈리는 내용이니까, 꼭 위의 내용들을 완벽하게 숙지하고 설명을 읽는 걸 추천할게.)

그런데 만약 함수 인자의 수를 딱 정해놓지 않고, 입력한 개수 만큼 알아서 실행하고 싶은 함수를 만들고 싶으면 어떻게 할까? 지금까지는 함수(a, b) 이렇게 인자의 개수를 특정했었지만, 만약 입력 인자의 개수가 특정하고 싶지 않을 경우, 우리는 **\*args**를 사용할수 있어. 저기서 중요한 건은 **\***이고, args는 변수 이름 아무거나 자신이 쓰고 싶은 것을 쓰면 되지만, 보통 arguments의 줄임말인 args를 많이 사용한다고 해. 마치 for문의 변수를 i로 설정하듯이 말이야. 예시를 보자.

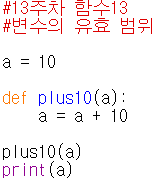
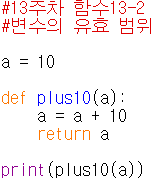
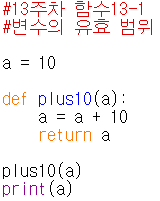
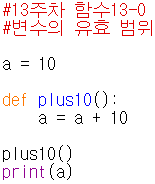
  

함수10을 실행해보자. print(args)를 보면 알겠지만, 함수에서 입력 받은 인자를 그대로 args라는 이름의 튜플에 값을 저장했다는 것을 볼 수 있을거야. 그러니까 \*args를 사용하면 입력 받은 입력 인자들을 해당 이름을 가진 튜플에 저장한다는 것을 알 수 있어. 나머지도 설명하자면, sum1이라는 변수를 먼저 만들어 놓고, args 튜플에 있는 값, 그러니까 우리가 함수의 입력 인자로 입력했던 값들 하나하나를 sum1에 더하는 거니까, 마지막에 sum1은 1+3+5+7+9의 값을 가지겠지? 그리고 함수 자체가 print문을 가지고 있으니까 print를 따로 쓸 필요도 없고.

함수11, 12를 보자. 아까 말했듯이 args를 꼭 안 써도 되고, 저렇게 자신이 사용하고 싶은 변수명을 사용해도 상관 없어. 결과값은 설명 안 할테니까 예상해보고 한 번 직접 실행해 봐

드디어 기말고사 마지막 범위야. 강의계획서에는 ‘변수의 유효 범위’라고 적힌 내용인데, 아마 함수 안팎으로 쓰인 변수가 함수를 기준으로 어디까지 영향이 유효한지를 알아보는 것 같아. 그런데 여긴 나도 헷갈리는 파트여서, 완벽하진 않겠지만 최대한 이해하기 쉽게 설명해보도록 노력해볼게.

우리는 함수를 작성할 때, 괄호 안에 입력인자를 입력하고는 했어. 그런데 만약 이런 경우가 있다면 어떨까?

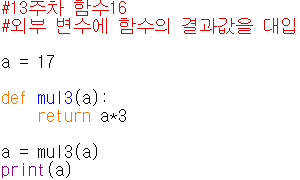
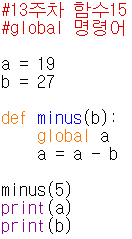
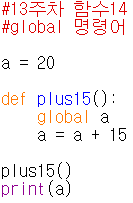
 

함수13을 보자. a=10 이고, plus10이라는 함수는 입력인자 a를 가져. 그리고 plus10 함수에서 입력인자 a에 a+10을 대입하고, 결과를 반환하지 않았어. 그렇다면 함수 밖의 a는 이제 20이 됐을까? 함수에서 a에 대해 10을 더했으니까 말이야. 결론부터 말하자면 함수 밖의 a는 여전히 10, 그러니까 print(a)의 결과는 10이 나올거야. 여기서 헷갈리면 안 되는 것은, **함수 밖의 a, 즉 10이라는 정수를 가진 변수 a와, 함수의 입력 인자 a는 같은 a가 아니야**. 함수의 입력 인자 a는 변수 a와는 아무 관련도 없다고 생각하면 돼. 그러니까 **plus10(a)에서의 a는 단순히 함수 입력 인자를 받으면 어떻게 그 인자를 처리할 지 코드를 짜게 해주는 입력 인자의 대역**에 불과해. 함수 밖의 a와는 아무 관련도 없는 친구라는 거야. 헷갈리면 안 되는 중요한 내용이야.

함수13-0을 보자. 아까 함수13에서 plus10(a)의 a는 함수 입력 인자의 대역이라고 했지? 그러니까 13-0에서 plus10()은 함수 인자의 대역 역할을 하는 함수 인자 a를 설정하지 않았으니, 당연히 오류가 발생하겠지? 함수 바깥에 있는 변수 a에 10을 더하는 함수가 아니라는 것을 다시 유의해두고.

그렇다면 함수13-1은 어떨까? 변수 a와 함수 입력 인자 a가 다르다는 것을 인지하고, 저 변수 a에 10을 더한다는 의미에서 마지막에서 두번째 줄에 plus10(a)를 입력했어. 함수에서도 a에 10을 더한 값을 a로 return하니까, 그렇다면 plus10(a)를 실행함과 동시에 변수 a는 20이 됐을까? 결론은 아니야. **특별한 명령어나 외부에서의 입력 없이, 함수 내부의 코드만으로는 절대로 외부의 변수에 간섭할 수 없어**. 13-1에서 plus10(a)가 20이라는 것은 맞지만, 그것은 단순히 a=10이라는 변수를 함수 입력인자로 해서 함수를 처리한 결과 20이 나왔다는 것이지, 앞으로 변수 a도 20이라고 바꾸는 것은 아니라는 거지. 실제로 13-2처럼 plus10(a) 자체를 프린트 할 경우, 출력값은 20인 걸 알 수 있어. 그러니까 함수는 단순히 입력 인자를 처리한 값만 돌려주는 역할인 것이지, 함수 외부의 변수에 아무 명령어 없이 간섭할 수는 없다는 거지.

그렇다면 함수 자체나 함수의 결과값을 통해 외부 변수를 바꾸고 싶다면 어떻게 해야할까? 두 가지 방법이 있어. 첫 번째는 **global 명령어** 사용, 두 번째는 추후에 **외부 변수에 함숫값을 대입하는 코드를 따로 작성**하는 것이야. 예시를 보자.



함수14를 보자. **함수에서 global (변수)를 선언하면,** **함수 안의 (변수)는 이제 앞으로 함수 인자가 아닌 외부의 변수를 의미한다고 말하는 것이고, 동시에 외부의 변수는 해당 함수의 영향을 직접 받게 돼.** 그래서 plus15() 라는 함수를 실행하면(코드에 plus15() 입력) 이후 a라는 변수는 35의 값을 가지게 될 거야. 실제로 저 파일 실행해서 결과를 봐봐.

함수15를 보자. 이제 앞의 내용들을 완벽하게 숙지했다면, a가 몇으로 출력될 지, b가 몇으로 출력될 지 이해가 될거야. 힌트를 주자면 minus(b)에서의 b와 변수 b는 다른 것, 그리고 minus 함수에서의 a는 변수 a와 동일하다는 것. 한 번 결과를 예상하고 직접 결과를 출력해서 비교 해 봐.

global 명령어는 함수와 외부의 결합력을 높이므로 코딩에서 지양해야할 명령어라고 해. 모듈 간의 결합력이 높다는 건 중간고사 이론 시험범위였으니 무슨 말인지 이해할 거야. 모르겠더라도 하여튼 global 명령어는 실전 코딩에서는 조금 지양하는 명령어라는 것만 알아두면 돼.

함수16을 보자. global 명령어하고 다르게 큰 설명 없이도 이해가 갈 거야. a를 대입한 함수의 결과값을, 변수 a에 새로 대입한다는 거니까. 그러니까 저렇게 함수의 결과값을 외부의 변수에 대입하면, 함숫값을 실제로 변수에 반영할 수 있다는 거지.

이렇게 기말고사 파트의 파이썬 기본개념 설명은 끝이 났어. 설명이 어렵지? 내 설명력이 많이떨어져서 그런 것도 있고, 기말파트가 진짜 어려워서 그런 걸 수도 있어. 중간 때와 다르게 이번에는 시험기간에 제때제때 답변을 못 해 줄수도 있을 거 같으니까, 설명 읽고 모르는 부분 있으면 평소에 언제든지 나한테 질문하고! 중간고사 때처럼 벼락치기로 하려고 하려면 많이 힘들거야. 외워서는 해결되지 않을 내용들이 많고, 중간 이론 때와 다르게 기말 이론 때는 이론 시간에도 실습 내용을 설명하기 때문에 교안 달달 외워서 점수 잘 맞는게 어려울 수 있기 때문이야. 평소에 열심히 하고, 평소에 실습 복습하고, 질문하고, 내용을 이해하면 컴사 다 잘 볼 수 있을거야. 여기까지 다 이해하고 왔다면 정말 수고 많았고, 아직 이해 다 못 했더라면 열심히 공부하고! **기말고사 컴사는 H14가 상위권 싹쓸어가자. 파이팅!!!!!**