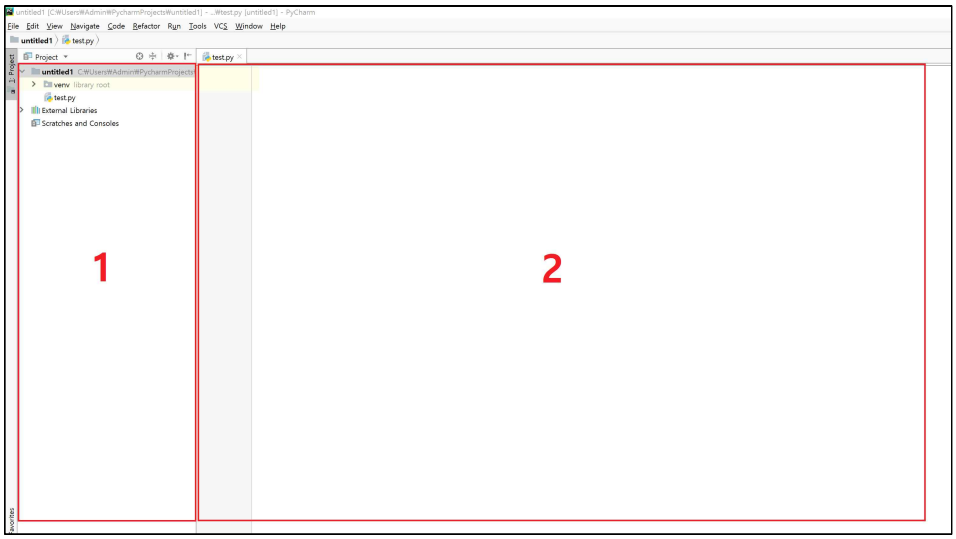
Python 0주차

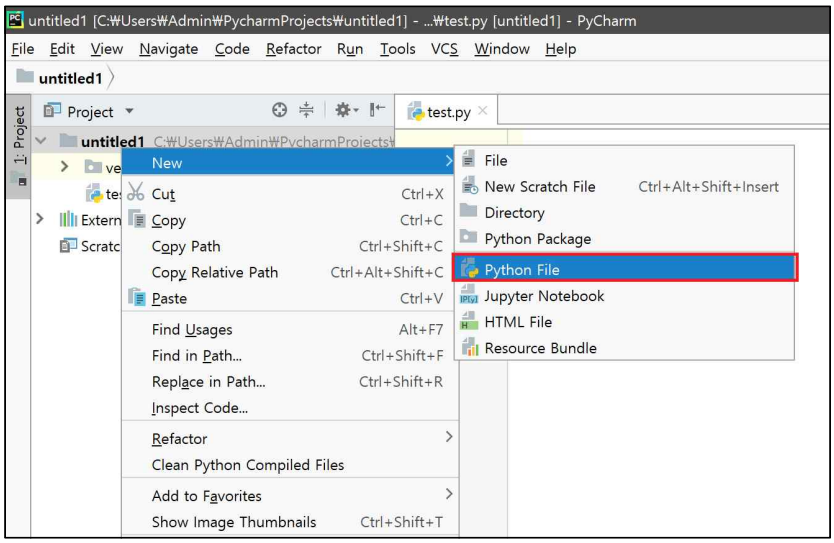
<기본 입출력 함수와 자료형>

[1. Pycharm 화면 구성]

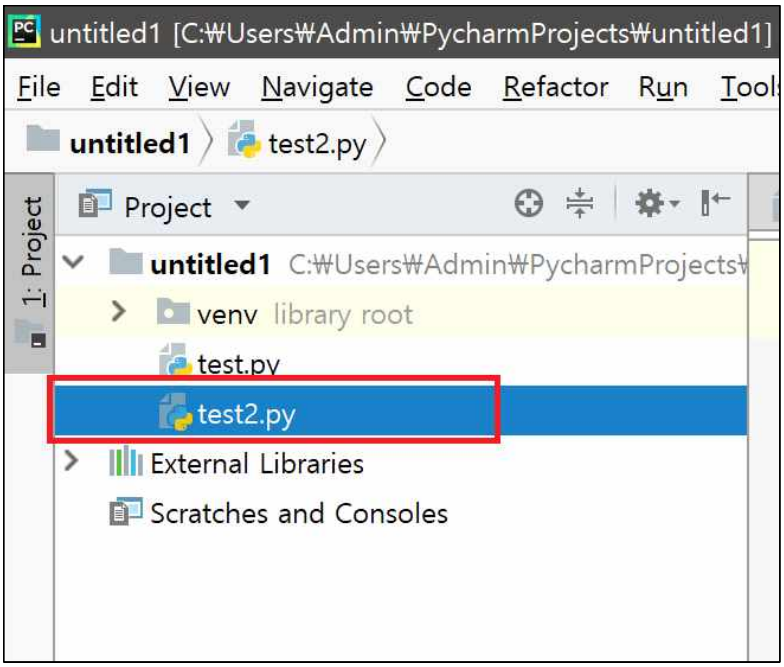
OT 자료를 끝까지 잘 따라하셨다면, 현재 새로운 프로젝트가 만들어져 있는 상태이고 이런 화면이 떠있을거에요.



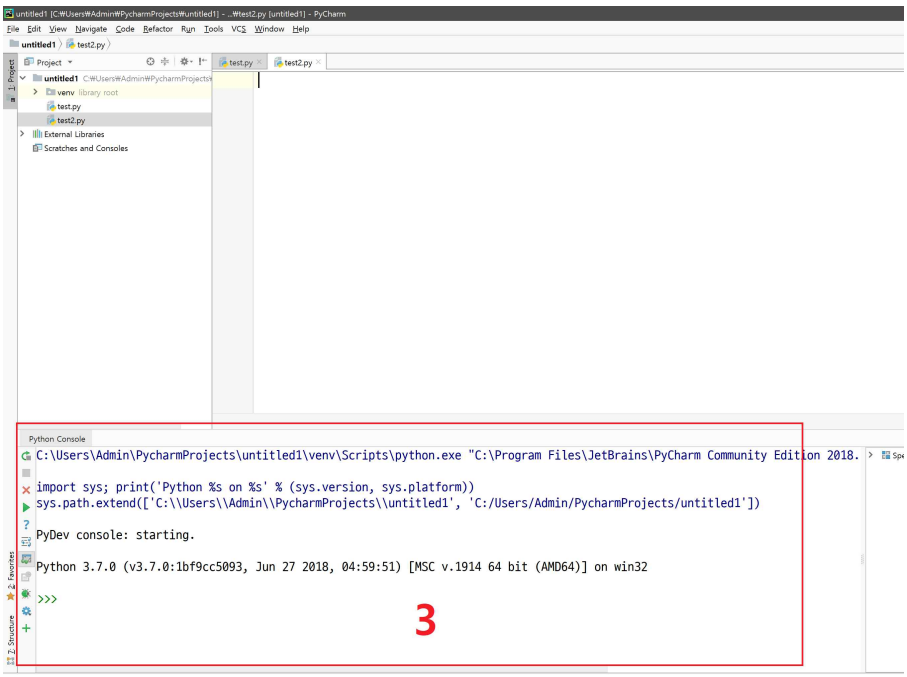
1번 화면은 파일 탐색기라고 생각하시면 됩니다. 우리가 제일 처음으로 만들었던 프로젝트 폴더(저 사진에서는 프로젝트 폴더 이름이 'untitled1'이라고 되어 있습니다.)가 있고, 그 아래에는 venv 라는 폴더와 test.py가 생성되어 있습니다.



위 사진처럼 프로젝트 폴더 위에서 '마우스 우클릭-New-Python File'을 클릭해주시면 새로운 파이썬 파일이 아래 사진처럼 생성됩니다.



2번 화면은 우리가 코딩을 하게 될 '도화지'입니다.
이 화면에 대해 설명하기 전에 또 보여드릴 화면이 있습니다.
상단 메뉴에서 "View-Tool Windows-Python Consol'을 클릭해보세요. 그러면 아래와 같이 창이 하나 생성됩니다.



이 3번 화면을 '콘솔 창'이라고 합니다. 이 3번 화면도 2번 화면과 마찬가

지로 코딩을 하는 창입니다. 2번 화면과의 차이점이 뭔지 한번 확인해볼게요. 2번 화면에 아래와 같이 쳐보겠습니다.

```
print(“Hello World”)
print(10+20)
```

그리고서 Ctrl+Shift+F10을 눌러주시면 아래와 같은 실행결과를 얻을 수 있습니다.

```
Hello World
30

Process finished with exit code 0
```

똑같은 코드를 3번 화면에 타이핑 해볼게요

```
>>> print("Hello World")
Hello World
>>> print(10+20)
30
```

뭔가 차이점을 느끼셨나요.

3번 화면은 한 줄 단위로 실행결과를 바로바로 알려줍니다. 그래서 이 콘솔창을 보고 ‘대화형 창’이라고도 얘기를 합니다. 파이썬을 기계어로 번역해주는 인터프리터라는 번역가는 이렇게 “한 줄 단위”로 코드를 번역하는 특징이 있습니다.

“엥 근데 2번 화면은 한 줄 단위로 번역한게 아니잖아요??”

2번 화면은 마치 도화지를 통째로 번역해서 실행결과를 한번에 보여준 것처럼 보이긴 합니다. 하지만 인터프리터는 한 줄 단위로 번역하는 놈이기 때문에 한 줄 단위로 번역해놓고서 실행결과를 저렇게 나중에 뿌려주는 것뿐입니다.

우리는 앞으로 2번 화면과 3번 화면을 골고루 쓰게 될 겁니다. 어떤 하나의 프로그램을 만들게 된다면 2번 화면에 코딩을 해서 그 결과를 한 번에 확인하게 될 것이고, 어떤 하나의 문장만을 테스트 해보고 싶을 땐 인터프리터와 대화하듯이 그 결과를 바로바로 알아야하기 때문에 3번 화면을 활용하게 될 것입니다.

[2. 기본 입출력 함수]

[2-1. 출력 함수 – print()]

기본 출력 함수인 print() 함수에 대해 느낌 아셨겠지만 조금만 더 알려드릴게요. 뭐 직감적으로 알고 계셨듯이 뭔가를 화면에 출력해서 우리 눈으로 확인하고자 할 때 쓰는 함수입니다.
print() 함수는 특징이 몇 가지 있습니다. 이 특징들을 외울 필요는 없습니다! 절대 외우지 말아주세요ㅠㅠ 코딩하시다보면 분명히 자연스럽게 외워집니다.

(특징 1) 괄호 안에 있는 내용을 출력하고 나면 엔터가 자동으로 쳐진다.

---> 설명:
아래와 같이 코딩을 해볼게요.

```
print("Hello")
print("World")
```

코드를 실행하고 싶을 땐 <Ctrl+Shift+F10>!
결과는 이렇게 두 줄로 출력이 됩니다.

```
Hello
World

Process finished with exit code 0
```

그 말은 print() 함수는 괄호 안의 내용을 출력한 뒤에 엔터 (더 정확히 말하면 "개행")을 자동으로 해줍니다.

(특징 2) 여러 개의 값들을 콤마(,)를 활용해 한 줄에 출력할 수 있다.

---> 설명:
"Hello" 라는 문자열과 "World" 라는 문자열을 한 줄에 출력하고 싶을 땐, 아래와 같이 콤마를 활용합니다.

```
print("Hello", "World")
```

결과는 아래와 같습니다.

```
Hello World

Process finished with exit code 0
```

한 줄에 출력하고 싶은 것들을 콤마(,) 로 구분하여 넣어줍니다. 콤마(,) 로 구분하여 출력하면 두 문자열이 스페이스바(공백)으로 구분되어 출력됩니다.

예제)

1. 아래 문자열을 print() 함수를 이용하여 출력해보세요.

```
Life is too short, You need Python.
```

2. 아래 문자열을 print() 함수를 이용하여 출력해보세요.(정확히 두 줄을 한 번에 출력해야 합니다.)

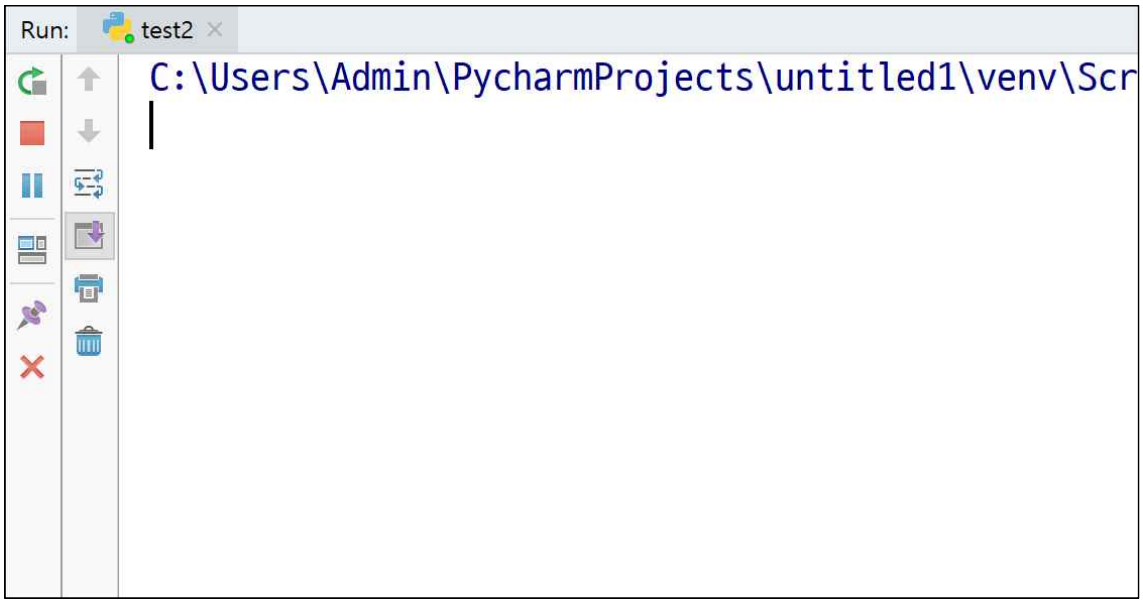
```
돈 많은
백수가 되고 싶다...
```

[2-2. 입력 함수 – input()]

출력 함수가 어떤 “값”을 “모니터”에 출력하는 것이였다면
입력 함수는 어떤 “값”을 “키보드”로부터 입력받는 것입니다.
입력함수는 이렇게 생겼습니다.

```
input()
```

윗 줄에 저 뻘건 글씨를 고대로 도화지에 코딩을 하고서 실행시킨 후(실행은 <Ctrl+Shift+F10> !!!) 결과 창을 봐보면



이렇게 커서가 계속 깜빡거립니다.(깜빡거리지 않는다면 마우스 커서를 결과 창 위에 두고서 한번 클릭!해주세요)
파이썬이 키보드의 입력을 기다리고 있는 겁니다. 어쩌고 저쩌고 키보드로 치고싶은대로 다 치고 엔터를 딱! 쳐보세요. 그 결과..
네, 아무 것도 나타나지 않습니다..
파이썬은 제가 키보드로 입력한 문자열을 읽어들입니다. 그런데 그냥

input() 함수만 쓰게 되면, 이 문자열을 저장할 박스가 없기 때문에 제가 쳤던 문자열을 그냥 공중에서 증발합니다. 따라서 도화지에 쳤던 코드를 이렇게 수정할게요.

```
var = input()
```

var이란 제가 그냥 맘대로 지은 변수 이름입니다.
이렇게 input() 함수 왼편에다가 변수를 써주고 등호(“=”)를 붙여주면 제가 키보드로 입력한 문자열들이 공중분해 되지 않고, var 이라는 변수로 고스란히 들어가게 됩니다.
과연 문자열들이 잘 저장이 되었는지 확인을 해볼까요
눈으로(모니터로) 직접 어떤 결과를 출력해보고 싶을 땐 print() 함수를 쓴다는 거 잊지 마세요
코드를 아래처럼 수정하고 한번 결과를 확인해볼게요.

```
var = input()
print(var)
```

<실행 결과>

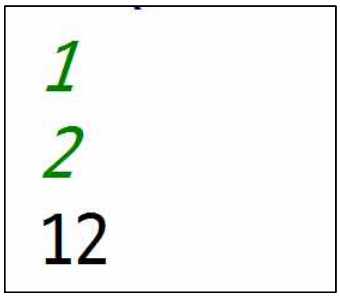
```
어쩌고저쩌고헤헤헤헤
어쩌고저쩌고헤헤헤헤

Process finished with exit code 0
```

제가 아무 문자열이나 키보드로 입력하고(초록색 문자열) 엔터를 치니 고대로 출력이 되었습니다(검은색 문자열). 즉, input() 함수가 제 문자열들을 잘 읽어들인 것을 확인했습니다.
자, 그러면 간단하게.. 두 수를 입력받아 이를 덧셈 해주는 프로그램을 만들어볼까요?

```
a = input()
b = input()
print(a+b)
```

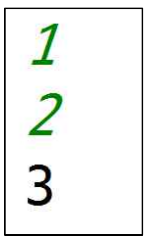
어떤 결과가 나올지 예측 되시나요?
프로그램 실행 후 숫자하나를 입력하고 엔터를 치면 그게 a에 저장되고
또 다른 숫자 하나를 입력하고 엔터를 치면 그게 b에 저장되고
a와 b를 더한 값이 출력이 되겠죠? 한번 실행해볼게요



엥.. 저는 분명 1하고 2를 입력해서 3이나오길 기대했는데 12가 나왔죠
자자자자, 당황하면 안돼요..! 우선 이것에 대한 이야기는 조금만 있다가 할게요! 여기서 설명이 길어지면 우리 너무 피곤해지니까.....
우선 이를 해결하기 위해선 아래처럼 코드를 수정하면 됩니다.

```
a = int(input())
b = int(input())
print(a+b)
```

수정하시고 다시 실행해보시면 정상적인 덧셈 결과를 출력할 겁니다.
<실행 결과>



딴건 아직 몰라도 되는데 이것만 기억하세요! 숫자를 숫자답게 쓸려면(숫자답게 연산 같은걸 할려면) input() 함수 바깥에 int() 라는 함수를 씌워준다.

예제)

1. 입력받은 두 개의 문장을 한 줄에 나란히 출력하는 프로그램을 작성하세요.
   (두 개의 문장 사이에는 띄어쓰기 하나가 껴있도록!)

예시 입력	예시 출력
강한친구 대한육군	강한친구 대한육군

2. 두 개의 숫자를 입력받으면 그 두 숫자의 덧셈, 뺄셈, 곱셈, 나눗셈을 출력하는 프로그램을 작성하세요.

예시 입력	예시 출력
20 10	30 10 200 2

[3. 자료형]

자료형만 알아도 그 언어의 절반을 배웠다라는 말이 있을 정도로 자료형은 정말 중요합니다. 따라서 이번 내용이 조금 지루하게 느껴지더라도 꼭 집중해서 익히셔야합니다.

이 쯤에서 쬐만한 팁하나 드려야겠다. 파이참 코딩하실 때 글자 크기가 너무 작아서 답답하진 않으신가요
상단 메뉴 중 "File - Settings" 클릭하신 담에
"Editor - Font" 들어가시면 글자 크기를 조절할 수 있습니다.

[3-1. 숫자형]

숫자형의 가장 큰 특징!! 연산이 가능하다!
여기서 말하는 연산이란 덧셈, 뺄셈, 곱셈, 나눗셈과 같은 사칙연산과
크거나 같다, 작거나 같다, 완전히 같다 등의 비교연산 을 이야기합니다.

프로그래밍에서는 숫자를 정수와 실수로 구분합니다. 예를들어 0과 0.0 은 확연히 다른 수 입니다. 0은 정수형, 0.0은 실수형이라고 합니다. 즉, 소수점이 없으면 정수! 소수점이 있으면 실수라고 할 수 있겠습니다. 정수형의 예를 더 들자면 -1 -2 1 +1 0 123 -123 ... / 실수형의 예를 더 들자면 1.0 1.23 123.32 -120.45 0.0 .... 파이썬에서 정수형은 int(integer의 준말입

니다.) , 실수형은 float(float point의 준말입니다. 한국말로 '부동 소수점'이라고 하는데 이 용어에 대해서는 몰라도 상관 없습니다. 그냥 float은 실수형! 이라고만 알고계셔도 당신은 지식인) 이라고 부릅니다.
숫자형의 연산은 "숫자형"끼리만 가능합니다. 당연하겠죠.. 숫자와 문자를 더할 수는 없으니..(10 + "책" = ???) 그치만 이 사실은 좀 중요합니다. 나중에 이런 실수 해놓고서는 뭐가 잘못됐는지 몰라서 헤맬 때가 있습니다.. 나만 그런건가..ㅠㅠ
숫자형은 예를 들어 1 + 1 , 2 – 1 , 3 * 3 , 12 / 2.3 ... 이런식으로 사칙연산을 할 수 있습니다. (정수형과 실수형끼리의 연산도 당연히 가능합니다. ==> 파이썬에서는 당연히 "가능" 이지만, 안타깝게도 몇몇개의 언어에서는 이게 당연히 "불가능"입니다.. 파이썬 짱)
하나 특이한 것은 사칙연산자 이외에 파이썬에서는 '%' '//' '**' 요런 연산자들이 추가로 더 있습니다.

- % : 두 수를 나눈 후 나머지를 알려줍니다.

  (ex: 5%3 의 결과는 2, 15%7의 결과는 1)

- // : 두 수를 나눈 후 몫을 알려줍니다.

  (ex: 5//3 의 결과는 1, 15//7의 결과는 2)

- ** : 제곱을 말합니다.

  (ex: 4**2 의 결과는 16, 2**3의 결과는 8)

이 숫자들을 어디 저장하고 싶다?? 그러면 왼쪽에다가 변수 써주시고(이때, 변수 이름은 아~~무거나 지어서 그냥 써주시면 됩니다.) 변수와 숫자 사이에 등호("=")를 써주시면 됩니다.

```
a = 10
number = 10 + 13
asdf = 3 ** 2
```

파이썬에서의 등호(“ = ”)

파이썬에서 “=”는 수학에서의 “=”과 조금 다릅니다. 파이썬에서는 이 기호가 하나의 ‘연산자’입니다. 오른쪽에 있는 아이를 왼쪽에 있는 아이에게 저장을 시킨다는 뜻을 갖습니다. 즉,

a = 10 => “10을 a 라는 변수에 저장시키겠다!”

number = 10 + 13 => “10+13의 결과를 number 라는 변수에 저장시키겠다!”

asdf = 3 ** 2 => “3**2의 결과를 asdf 라는 변수에 저장시키겠다!”

예제)

1. 숫자 하나를 입력받아서 2로 나눈 나머지를 출력하는 프로그램을 작성하세요.

예시 입력	예시 출력
35	1

[3-2. 문자열]

[3-2-1. 문자열 표현방법]

예를 들어 a 라는 변수에 Hello 라는 문자열을 저장하고 싶을 때에는

```
a = “Hello”
```

라고 타이핑하면 됩니다.

```
a = Hello
```

위와 같이 큰따옴표로 둘러 쌓여 있지 않으면 에러가 납니다.

에러 나는 이유! 설명해드릴게요.

따옴표로 둘러쌓여 있지 않은 문자들은 모두 ‘변수 이름’으로 생각을 합니다.

그럼,

```
a = Hello
```

라고 코딩하게 되면 파이썬의 인터프리터는 이렇게 생각하게 돼요.

“아, Hello 라는 변수에 들어있는 값을 고대로 a라는 변수에 저장을 해야겠다.”

“엥? 근데 Hello 라는 변수가 위에서 쓰이질 않아서 들어있는 값이 없는데?”

“에이..퉤.. Error!!!”

즉, 문자열을 나타내기 위해서는 큰따옴표(“”) 로 둘러싸야만 합니다.

근데 파이썬은 문자열을 나타내기 위한 방법이 큰따옴표(“”) 뿐만 아니라 네가지 방법이나 있습니다.

1. 큰 따옴표(“”)
2. 작은 따옴표(‘’)
3. 큰 따옴표 3개(“““ ”””)
4. 작은 따옴표 3개(‘‘‘ ’’’)

어떤 방법을 쓰던 상관 없습니다. 그렇다면 왜 파이썬에서는 문자열을 표현하는 방법을 4가지 방법이나 제공할까요.

예를 들어 여러분들이 아래 문자열을 변수에 저장하고 싶습니다.

그는 “안녕” 이라고 말했다.

따라서 배운대로 아래와 같이 타이핑을 해봅니다.

a = “그는 ”안녕“ 이라고 말했다.”

이렇게 해보면 에러가 납니다. 왜냐면 “그는 ” 이라는 문자열과 “ 이라고 말했다.” 이라는 문자열로 나뉘어 버리고 안녕 이라는 문자열은 큰따옴표에 둘러싸이지 못하고 소외되는 현상이 발생하기 때문입니다. 큰따옴표에 둘러싸이지 못하면 당연히 에러가 나겠죠?

따라서 이럴 때는 다음과 같이 문자열을 작은 따옴표로 둘러싸서 해결합니다.

a = ‘그는 ”안녕“ 이라고 말했다.’

이렇게 표현하고 싶은 문자열에 큰따옴표가 들어있으면 작은 따옴표로 문자열을 둘러싸고 반대의 경우에도 똑같겠죠?

그렇다면 따옴표 3개짜리는 왜 있을까요?

여러분이 표현하고 싶은 문자열이 다음과 같다고 생각해볼게요.

그는
안녕이라고
말했다.

이렇게 세 줄짜리 문장을 온전히 표현하고 싶어요. 즉 엔터 자체를 문자열에 저장을 시키고 싶다하면 요렇게 하면됩니다.

a = ‘‘‘그는
안녕이라고
말했다.‘‘‘

이렇게요! 이 이후에 print(a) 를 해보시면 세줄이 모두 온전히 출력됩니다. 혹시나 저 문자열 안에 작은따옴표가 포함되어 있다면 그 때는 큰따옴표 세개를 쓰면 되겠죠?

근데 사실 위의 방법을 쓰지 않아도 가능한 방법이 있습니다.

```
a = '그는\n안녕이라고\n말했다.'
```

이렇게 하고서

```
print(a)
```

를 해보시면 같은 결과를 얻을 수 있어요.
"\n" 이란, 엔터를 나타내는 특수문자에요.
이렇게 앞에 역슬래쉬를 하고서 어떤 문자를 쓰게 되면 특수문자로서 작동을 하는데, 이를 이스케이프 코드 라고 합니다. 이 이스케이프 코드는 꼭! 문자열 안에 (즉, 큰따옴표 또는 작은따옴표 안에) 써줘야지만 작동을 합니다. 역슬래시는 키보드의 "₩"를 누르면 나옵니다.
이스케이프 코드는 아래와 같은 종류들이 있습니다.

코드	설명
`\n`	개행 (줄바꿈)
`\t`	수평 탭
`\\`	문자 " \ "
`\'`	단일 인용부호(')
`\"`	이중 인용부호(")

하나씩 실험 한 번 해보세요!
예를 들어

```
print("a\tb\tc\nd\te\tf")
print("\"안녕하세요\"")
```

[3-2-2. 문자열 연산]

파이썬에서는 특이하게도 문자열끼리 연산도 가능합니다. 문자열끼리 더하기를 한다는게 사실 낯설긴한데.. 우선 한번 해보시면 바로 이해됩니다.

```
>>> print("Hello"+"world")
Helloworld
```

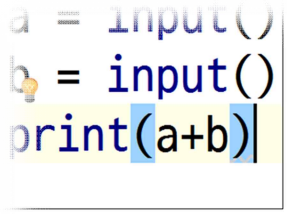
문자열끼리의 덧셈은 그냥 두 문자열을 이어 붙혀줍니다. 띄어쓰기 없이 붙여준다는 점을 주의하세요!
문자열은 심지어 곱셈도 됩니다.

```
>>> print("Hello"*3)
HelloHelloHello
```

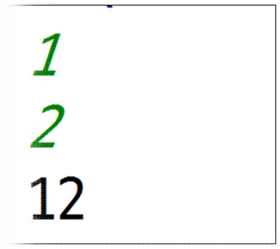
요런 점에 있어서 파이썬이 직관적이라고 할 수 있는 것 같아요.
다음과 같이 문자열과 숫자끼리 덧셈, 또는 문자열과 문자열끼리 곱셈은 불가능합니다.

>>> “Hello” + 3 (x)
>>> “Hello” * “World” (x)

어?!?!?!? 잠깐 여기서 뭔가가 생각나지 않나요???? 잠시 몇페이지 전에 읽었던걸 회상 해볼게요.
<↓ 회상 중.....>



떤 결과가 나올지 예측 되시나요?
로그램 실행 후 숫자하나를 입력하고 엔터를 치면 그게 a에 저장되고
다른 숫자 하나를 입력하고 엔터를 치면 그게 b에 저장되고
와 b를 더한 값이 출력이 되겠죠? 한번 실행해볼게요



우리 input() 배울 때, 1과 2를 입력했더니 결과가 3이 안나오고 12가 나왔잖아요. 이 이유를 이제야 이해할 수 있게 됐어요. input() 함수는 내가 문자를 치던 숫자를 치던 모든 것을 “문자열 형”으로 받아들여요. 즉, 내가 1과 2를 키보드로 입력했지만 input() 함수는 숫자형으로 읽어들인게 아닌 “문자열 형”으로 읽어들인거죠.
그럼 변수 a에는 문자열 형 “1”이 들어가 있고,
변수 b에는 문자열 형 “2”가 들어가 있는 상태입니다.
그 상태로 a와 b를 더한다? 이는 방금 배운 문자열끼리의 덧셈이 됩니다.
즉, “1”과 “2”를 띄어쓰기 없이 그냥 붙여쓴 “12”가 되는 겁니다.
따라서, 정상적인 사칙 연산을 위해선 숫자형으로 강제 형 변환을 해줘야 합니다. 숫자형으로 강제 형 변환을 시켜주는 함수가 바로 int()라는 함수였고, input() 함수 바깥에 아래와 같이 int() 함수를 씌워주시면 드디어 a와 b 변수에는 문자열 형이 아닌 레알 숫자!(연산이 가능한 진짜 숫자)를 넣을 수 있게 됩니다.

```
a = int(input())
b = int(input())
```

[3-2-3. 문자열 인덱싱, 슬라이스]

문자열 인덱싱, 슬라이스를 더 쉬운 말로 바꿔 표현하자면
인덱싱 => 선택연산자
슬라이스 => 범위 선택연산자
우선 한번 보여드릴게요.

```
>>> a="PYTHON"
>>> print(a[0])
P
```

바로 이해 되시나요? a 변수에 "PYTHON" 이라는 문자열을 저장시키고 그중 P라는 글자 하나만 가져온 것입니다. 인덱싱은 형태가 다음과 같습니다.

```
<변수이름>[<인덱스 번호>]
```

문자열이 저장된 변수이름 뒤에 대괄호([]) 를 붙히고 그 안에 원하는 번호를 넣으면 됩니다. 문자의 번호는 문자열에 가장 앞에서 부터 0번, 그다음이 1번, 그다음이 2번,.. 이런식으로 자동으로 매겨집니다.
즉 PYTHON 에서 P 는 0번, Y는 1번, T는 2번... 이런 식이죠.
그렇다면 N이라는 글자를 가져올 때는,

```
print(a[5])
```

라고 하면 되겠죠. 그런데 이것도 됩니다.

```
print(a[-1])
```

즉 문자열의 가장 끝 글자에는 -1이라는 번호가 또 붙습니다. N 이 -1 이면, 그 앞의 O는 -2 , 그 앞의 H 는 -3 이런 식입니다. 즉, 한 글자당 인덱스 번호는 각자 두가지씩 가지고 있는 셈입니다.

슬라이스는 한글자가 아닌 여러글자를 가져오는 기능을 합니다.
"korea university" 라는 문자열에서 korea 를 가져와볼게요.

```
>>> name="korea university"
>>> print(name[0:5])
korea
```

이런 식으로 콜론(“:”)을 이용하면 몇 번부터 몇 번까지! 범위를 지정해서 해당 범위에 있는 문자들을 가져올 수 있습니다. 근데 여기서 하나 주의할 것은
제가 name 문자열의 0번부터 5번까지라고 범위를 정했습니다. 하지만 korea의 k는 0번이 맞는데, a는 4번이죠. 즉 제가 name[0:5] 라고 적은 것은 0번부터 5번 미만이라는 뜻입니다. 주의해야돼요!
슬라이스를 사용할때는 이런 방법도 가능합니다.

```
name[:5]     # 첫 글자부터 5번 미만까지
name[6:]     # 6번부터 끝 글자까지
name[-4:]    # -4번부터 끝 글자까지
```

[3-2-4. 문자열 포매팅]

문자열에서 또 하나 알아야 할 것으로는 문자열 포매팅(Formatting)이 있습니다. 매우 중요합니다!!! 꼭 문자열 포매팅에 대해 완벽하게 익히고 넘어가세요!~
문자열 포매팅은 별거 없습니다.
아래와 같이 문자열 안에 중괄호를 넣어주고서
“{}”
그 문자열 뒤에 .format()을 붙여주면 됩니다. format() 의 소괄호 안에는 내가 넣고 싶은 것을 써넣으면 됩니다. 문자열을 넣어도 되고, 숫자형을 넣어도 됩니다.
예를 들어 이렇게요!

```
“{}”.format(“Hello World”)
```

이렇게 하면 format의 소괄호 안에 있는 것들이 중괄호 안으로 쏙 들어갑니다.

근데 이게 왜 필요할까요.. 그냥 “Hello World” 라고 하면 될 것을..?

자, 다음과 같은 문자열을 출력하는 프로그램을 작성했다고 가정해볼게요.
"현재 비트코인 가격은 10원입니다."

시간이 지나서 비트코인 가격이 오르면 아래와 같은 문장을 출력합니다.

"현재 비트코인 가격은 2000원입니다."

위의 두 문자열은 모두 같은데 10이라는 숫자와 2000이라는 숫자만 다릅니다. 나머지 부분은 모두 같죠. 이렇게 서식을 유지하면서 문자열 내의 특정 값만을 바꿔야 할 경우가 있는데, 이것을 가능하게 해주는 것이 바로 문자열 포매팅 기법입니다.

“현재 비트코인 가격은 {}원입니다.”.format(10)
“현재 비트코인 가격은 {}원입니다.”.format(200)

요렇게 사용하면 됩니다. 변수들을 활용해서 코드를 더 이쁘게 만들어볼게요.

```
alert = "현재 비트코인 가격은 {}원 입니다."
bitcoin = 10
print(alert.format(bitcoin))
bitcoin = 2000
print(alert.format(bitcoin))
bitcoin = 50000
print(alert.format(bitcoin))
```

<실행 결과>

```
현재 비트코인 가격은 10원 입니다.
현재 비트코인 가격은 2000원 입니다.
현재 비트코인 가격은 50000원 입니다.

Process finished with exit code 0
```

위 코드가 조금 헷갈리시면 파이썬의 인터프리터 입장에서 찬찬히 생각해보면 쉽습니다.
alert 라는 변수에는 문자열이 저장되어 있고, bitcoin 이라는 변수에는 숫자가 저장되어 있습니다.

```
print(alert.format(bitcoin))
```

위 문장을 만나게 되면, 첫 번째로 인터프리터는 alert 변수를 해당 변수에 저장되어 있는 문자열로 치환해주고, bitcoin 변수를 해당 변수에 저장되어 있는 숫자로 치환해줍니다. 이렇게요.

```
print("현재 비트코인 가격은 {}원입니다".format(10))
```

두 번째로 중괄호("{}")에 format() 함수의 소괄호 안에 들어 있는 10 이라는 숫자를 넣어줍니다.

```
print("현재 비트코인 가격은 10원입니다")
```

이제 이걸 고대로 출력하게 되는 겁니다.

[3-2-5. 문자열 관련 함수들]

함수관련된 내용은 외부 사이트 내용을 복사 했습니다. (출처 : 위키독스)앞으로 여러 함수들을 보시게 될 건데 이 함수들을 모두 외우려 하지 마세요! 현직 개발자들도 모든 함수들을 외우지 않습니다. 구글링이라는 좋은게 있으니까요..! 실제로 현직 개발자들이 구글링을 잘 활용하라고 많이 얘기합니다.
그래도 여기 기본적인 함수들 한번씩 실습해보시는 것을 추천드리고, 앞으로 많이 쓰이게 될 함수는 *중요* 표시를 해둘테니 그것만큼은 꼭 익히셔야 합니다.

앞으로 나올 함수들의 사용법이 조금 특이합니다. 함수의 이름이 예를 들어 func() 이고, 그 함수를 적용시킬 변수이름이 var 이라 했을 때,
어떤 함수는 func(var) 이라 하는 반면에
어떤 함수는 var.func() 이라고 합니다.
이를 이해하기 위해선 조금 더 깊은 지식이 필요합니다. 우선은 함수마다 그 사용법을 '암기'를 해주세요. '이해'는 나중에 해도 괜찮습니다.
공부법에는 '선 이해, 후 암기' 공부법이 있고, '선 암기, 후 이해' 공부법이 있습니다. '선 이해, 후 암기' 공부법이 더 좋다고 대부분 생각하지만, 때에 따라선 '선 암기, 후 이해' 공부법이 더 효과적일 때가 있습니다!

문자 개수 세기(count)

```
>>> a = "hobby"
>>> a.count('b')
2
```

- 문자열 중 문자 b의 개수를 반환합니다.

위치 알려주기(find)

```
>>> a = "Python is best choice"
>>> a.find('b')
10
>>> a.find('k')
-1
```

- 문자열 중 문자 b가 처음으로 나온 위치를 반환합니다. 만약 찾는 문자나 문자열이 존재하지 않는다면 -1을 반환합니다.

소문자를 대문자로 바꾸기(upper)

```
>>> a = "hi"
>>> a.upper()
'HI’
```

- upper() 함수는 소문자를 대문자로 바꾸어 줍니다. 만약 문자열이 이미 대문자라면 아무런 변화도 일어나지 않습니다.

대문자를 소문자로 바꾸기(lower)

```
>>> a = "HI"
>>> a.lower()
'hi'
```

- lower() 함수는 대문자를 소문자로 바꾸어 줍니다.

* 중요 *

양쪽 문자열 또는 공백 지우기(strip)

```
>>> a = “abcHELLOabc”
>>> a.strip(“abc”)
‘HELLO’
>>> a = "   hi    "
>>> a.strip()
'hi'
```

- a.strip(“abc”)처럼 괄호 안에 특정 문자열이 들어가 있는 경우, a 변수에 저장되어 있는 문자열의 양 끝에 있는 “abc”를 삭제한다.
- a.strip()처럼 괄호 안에 아무것도 없다면, a 변수에 저장되어 있는 문자열의 양 끝에 있는 공백을 삭제한다.

* 중요 *
문자열 바꾸기(replace)

```
>>> a = "Life is too short"
>>> a.replace("Life", "Your leg")
'Your leg is too short'
```

- replace(바뀌게 될 문자열, 바꿀 문자열)처럼 사용해서 문자열 내의 특정한 값을 다른 값으로 치환해 줍니다.

* 중요 *
문자열 나누기(split)

```
>>> a = "Life is too short"
>>> a.split()
['Life', 'is', 'too', 'short']
>>> a = "a:b:c:d"
>>> a.split(':')
['a', 'b', 'c', 'd']
```

- a.split()처럼 괄호 안에 아무런 값도 넣어 주지 않으면 공백(스페이스, 탭, 엔터등)을 기준으로 문자열을 나누어 줍니다.
- 만약 a.split(':')처럼 괄호 안에 특정한 값이 있을 경우에는 괄호 안의 값을 구분자로 해서 문자열을 나누어 줍니다. 이렇게 나눈 값은 리스트에 하나씩 들어가게 됩니다. (리스트에 대해서는 다음 내용에 나옵니다.)

예제)

1. 다음과 같은 문자열을 출력해보세요

엑셀보다 쉬운“파이썬”

2. 문자열을 하나 입력받고 그 문자열의 양쪽에 “=” 기호를 30개 추가하여 출력해보세요.

예시 입력	예시 출력
THE END	==========THE END==========

3. 문자열을 입력받고 그 문자열의 가장 끝 글자만 출력하는 프로그램을 작성해보세요.

예시 입력	예시 출력
안녕	녕
Gold	d
Korea	a

4. 문자열 포매팅을 사용하여 현재 날짜를 출력해보세요.

(문자열 포매팅을 사용해야 하므로 코드에 “{}년 {}월 {}일 {}요일” 이라는 부분이 꼭 들어가야 합니다.)

5. 문자열을 입력받고 그 문자열의 띄어쓰기를 모두 개행(엔터)으로 바꿔주세요.

예시 입력	예시 출력
I am Son	I am Son
You need Python	You need Python
Python is very very easy	Python is very very easy

[3-3. 리스트]

예를 들어, 사람들의 이름을 name 이란 변수에 저장한다고 할게요.

```
name1="아이언맨"
name2="스파이더맨"
name3="앤트맨"
name4="타노스"
name5="헐크"
...
```

뭐 사람의 수가 적다고 하면 이런식으로 저장할 수 있지만 사람의 수가 100명, 1000명이라면? 그러면 굉장히 번거롭죠ㅠㅠ 이럴 때 쓰는게 리스트 자료형입니다. 즉, 여러개의 데이터들을 한 공간에 저장하고, 그 데이터들을 관리하기 쉽게 만들어줘요. 따라서 위와 같은 예시를 다음과 같이 나타낼 수 있습니다.

```
name = ["아이언맨", "스파이더맨", "앤트맨", "타노스", "헐크", ...]
```

"아이언맨"이라는 문자열을 출력하고 싶다면

```
print(name[0])
```

"앤트맨"이라는 문자열을 출력하고 싶다면

```
print(name[2])
```

인덱스 번호를 사용하여 값을 선택하는 것이 마치 문자열과 비슷하죠?
리스트는 문자열과 마찬가지로 인덱싱과 슬라이스를 할 수 있고, 또한 리스트끼리 연산도 가능하죠.

```
>>> name = ["아이언맨", "스파이더맨", "앤트맨", "타노스", "헐크"]
>>> name[0]
'아이언맨'
>>> name[1]
'스파이더맨'
>>> name[1:4]
['스파이더맨', '앤트맨', '타노스']
```

```
>>> another = ["배트맨", "슈퍼맨"]
>>> name + another
['아이언맨', '스파이더맨', '앤트맨', '타노스', '헐크', '배트맨', '슈퍼맨']
>>> another * 3
['배트맨', '슈퍼맨', '배트맨', '슈퍼맨', '배트맨', '슈퍼맨']
```

단, 리스트가 문자열과 다른 것은 값을 수정할 수 있다는 것!

문자열은 한번 저장되면 그 중간에 있는 문자를 아래처럼 수정할 수 없습니다.

```
>>> a = "PITHON"
>>> a[1] = "Y"
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

인덱스 번호 1번에 있는 “I” 라는 글자를 “Y”로 수정하려 하니 에러가 납니다.(굳이 수정할려면 방법이 몇 가지가 있겠는데, a 변수에 그냥 처음부터 값을 다시 집어넣는거죠. 그냥 a = “PYTHON” 이렇게요. 또는 replace()함수를 사용해서, a.replace(“I”, “Y”) 라고 하면 수정할 수 있긴합니다. 다만 인덱스 번호를 사용하여 값을 수정하는 게 불가능한 것이죠.) \

하지만 리스트는 인덱스 번호를 사용하여 값을 쉽게 수정할 수 있습니다.

위 예제에서 인덱스 번호 3번에 있는 “타노스”를 “나쁜놈”으로 수정해보겠습니다.

```
>>> name[3] = "나쁜놈"
>>> name
['아이언맨', '스파이더맨', '앤트맨', '나쁜놈', '헐크']
```

리스트는 이렇게 여러 개의 값들을 한 번에 저장, 수정, 삭제하기가 쉽습니다.(삭제에 대해서는 아래 [3-3-1. 리스트 관련 함수]를 참고해주세요.)

지금은 리스트 값에 문자열만 저장했지만 정수형, 실수형, 그리고 “리스트”(리스트 안에 또 다른 리스트가 들어갈 수 있습니다.)까지 뭐든 넣을 수 있습니다.

data = [1, 2, “가나다”, 2.3, [5,6,7,8,“홍길동”]]

여기서 문제 하나 살짝!
위 data 변수에는 리스트 자료형이 저장되어 있습니다. 이 리스트의 인덱스 번호 4번 공간에 또 다른 리스트가 들어가 있는데요. 저 “홍길동”이라는 문자열을 갖고 오려면 어떻게 해야할까요??
정답은....
직접 고민해보시고 찾아보세요..! 정 모르시겠다면 저에게 카톡으로 언제든 물어보세요!

[3-3-1. 리스트 관련 함수]

* 중요 *
리스트에 요소 추가(append)
append를 사전에서 검색해 보면 "덧붙이다, 첨부하다"라는 뜻이 있다. 이 뜻을 안다면 아래의 예가 금방 이해가 될 것이다. append(x)는 리스트의

맨 마지막에 x를 추가시키는 함수이다.

```
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
```

리스트 안에는 어떤 자료형도 추가할 수 있다.
아래의 예는 리스트에 다시 리스트를 추가한 결과이다.

```
>>> a.append([5,6])
>>> a
[1, 2, 3, 4, [5, 6]]
```

* 중요 *

리스트에 요소 삭제(del)

del 함수는 그 함수 사용법이 조금 특이하다. 함수 사용법에 주의해야 한다. “인덱싱”을 하여 삭제 하고자 하는 값을 알려주면 된다.

```
>>> a = [1, 2, 3]
>>> del a[0]
>>> a
[2, 3]

>>> a = [1, 2, 3, 4, [5, 6]]
>>> del a[4]
>>> a
[1, 2, 3, 4]
```

리스트 정렬(sort)

sort 함수는 리스트의 요소를 순서대로 정렬해 준다.

```
>>> a = [1, 4, 3, 2]
>>> a.sort()
>>> a
[1, 2, 3, 4]
```

문자 역시 알파벳 순서로 정렬할 수 있다.

```
>>> a = ['a', 'c', 'b']
>>> a.sort()
>>> a
['a', 'b', 'c']
```

리스트 뒤집기(reverse)

reverse 함수는 리스트를 역순으로 뒤집어 준다. 이때 리스트 요소들을 순서대로 정렬한 다음 다시 역순으로 정렬하는 것이 아니라 그저 현재의 리스트를 그대로 거꾸로 뒤집을 뿐이다.

```
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']
```

위치 반환(index)

index(x) 함수는 리스트에 x라는 값이 있으면 x의 위치값을 리턴한다.

```
>>> a = [1,2,3]
>>> a.index(3)
2
>>> a.index(1)
0
```

위의 예에서 리스트 a에 있는 3이라는 숫자의 위치는 a[2]이므로 2를 리턴하고, 1이라는 숫자의 위치는 a[0]이므로 0을 리턴한다.
아래의 예에서 0이라는 값은 a 리스트에 존재하지 않기 때문에 값 오류(ValueError)가 발생한다.

```
>>> a.index(0)
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
ValueError: 0 is not in list
```

리스트에 요소 삽입(insert)

insert(a, b)는 리스트의 a번째 위치에 b를 삽입하는 함수이다. 파이썬에서는 숫자를 0부터 센다는 것을 반드시 기억하자.

```
>>> a = [1, 2, 3]
>>> a.insert(0, 4)
[4, 1, 2, 3]
```

위의 예는 0번째 자리, 즉 첫 번째 요소(a[0]) 위치에 4라는 값을 삽입하라는 뜻이다.

```
>>> a.insert(3, 5)
[4, 1, 2, 5, 3]
```

위의 예는 리스트 a의 a[3], 즉 네 번째 요소 위치에 5라는 값을 삽입하라는 뜻이다.

리스트 요소 제거(remove)

remove(x)는 리스트에서 첫 번째로 나오는 x를 삭제하는 함수이다.

```
>>> a = [1, 2, 3, 1, 2, 3]
>>> a.remove(3)
[1, 2, 1, 2, 3]
```

a가 3이라는 값을 2개 가지고 있을 경우 첫 번째 3만 제거되는 것을 알 수 있다.

```
>>> a.remove(3)
[1, 2, 1, 2]
```

remove(3)을 한 번 더 실행하면 다시 3이 삭제된다.

리스트 요소 끄집어내기(pop)

pop()은 리스트의 맨 마지막 요소를 돌려 주고 그 요소는 삭제하는 함수이다.

```
>>> a = [1,2,3]
>>> a.pop()
3
>>> a
[1, 2]
```

a 리스트 [1,2,3]에서 3을 끄집어내고 최종적으로 [1, 2]만 남는 것을 볼 수 있다.

pop(x)는 리스트의 x번째 요소를 돌려 주고 그 요소는 삭제한다.

```
>>> a = [1,2,3]
>>> a.pop(1)
2
>>> a
[1, 3]
```

a.pop(1)은 a[1]의 값을 끄집어낸다. 다시 a를 출력해 보면 끄집어낸 값이 삭제된 것을 확인할 수 있다.

리스트에 포함된 요소 x의 개수 세기(count)

count(x)는 리스트 내에 x가 몇 개 있는지 조사하여 그 개수를 돌려주는 함수이다.

```
>>> a = [1,2,3,1]
>>> a.count(1)
2
```

1이라는 값이 리스트 a에 2개 들어 있으므로 2를 돌려준다.

리스트 확장(extend)

extend(x)에서 x에는 리스트만 올 수 있으며 원래의 a 리스트에 x 리스트를 더하게 된다.

```
>>> a = [1,2,3]
>>> a.extend([4,5])
>>> a
[1, 2, 3, 4, 5]
>>> b = [6, 7]
>>> a.extend(b)
>>> a
[1, 2, 3, 4, 5, 6, 7]
```

a.extend([4,5])는 a += [4,5]와 동일하다.

예제)

1. 문자열을 입력하면, 각 단어들을 리스트에 저장하고 그 리스트를 출력하는 프로그램을 작성해보세요.

예시 입력	예시 출력
Life is too short, You need Python	['Life', 'is', 'too', 'short,', 'You', 'need', 'Python']

2. 아래에 리스트가 주어져 있습니다. 입력한 값이 주어진 리스트의 가장 마지막 원소로 추가 되고, 그 리스트를 출력하는 프로그램을 작성해보세요.

a = [‘유튜브’, ‘페이스북’, ‘카카오톡’]

예시 입력	예시 출력
인스타그램	[‘유튜브’, ‘페이스북’, ‘카카오톡’, ‘인스타그램’]

3. 아래에 리스트가 주어져 있습니다. 뭔가 크게 잘못되어 있는 부분을 찾아 적절하게 수정하여 리스트를 통째로 출력해주세요.

Korea = [‘독도는’, ‘일본’, ‘땅’]

4. 아래에 리스트가 주어져 있습니다. 1의 개수가 몇 개 인지 출력해보세요.(정확히 1이요! 11, 111, 1111 등은 제외) 리스트는 여기서 복사해서 사용하세요(직접 세지 말구요ㅠㅠ)

number = [1,1,1,1,1,1,1,111,1111,1,11,111,11111,1111,11111111,111,11,1,1]

[3-4. 튜플]

튜플은 다음과 같이 생겼습니다.

```
name = ("아이언맨", "스파이더맨", "앤트맨", "타노스", "헐크")
```

리스트와 똑같이 생겼는데 다른 점은 대괄호([]) 가 아닌 소괄호 (()) 로 둘러쌓여있죠? 소괄호로 둘러싸인 데이터들을 튜플이라고 합니다.
리스트와 사용법은 똑같아요!

```
>>> name = ("아이언맨", "스파이더맨", "앤트맨", "타노스", "헐크")
>>> name[0]
'아이언맨'
>>> name[0:3]
('아이언맨', '스파이더맨', '앤트맨')
```

그치만 리스트와 가장 큰 차이점은 값의 수정이 안된다는 것!

```
>>> name[3] = "나쁜놈"
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

다음과 같이 값을 수정하려하니 에러가 뜹니다.
수정뿐만 아니라 삭제도 불가능합니다.

```
>>> del name[3]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

그럼 값을 수정도 못하는데 이런 자료형이 왜 존재하냐..
프로그램을 만들다보면 절대로 손상이 되거나 수정이 되서는 안될 데이터들이 있어요. 예를들어 주민번호를 관리하는 프로그램이라 하면, 주민번호와 이름은 절대로 수정이 되면 안되잖아요. 그럴 때 튜플을 사용합니다. 리스트를 사용하면 개발자 실수나 누군가의 해킹으로 바뀔 수도 있으니까요.

[3-5. 딕셔너리]

우선 리스트 하나를 만들어 볼게요.
순서대로 사과, 배, 포도의 가격을 저장해보겠습니다.

```
fruits = [300, 500, 700]
```

리스트를 만들면 자동으로 각 원소마다 인덱스 번호가 매겨집니다. 앞에서부터 0, 1, 2 라고 매겨지죠. 자 근데 이렇게 놓고 보면 어떤 게 무슨 과일의 가격인지 모르겠잖아요. 이 리스트를 만든 저는 당연히 0번이 사과 가격이라는 것을 알고, 1번이 배 가격이라는 것을 알지만 다른 사람은 모르는게 당연하죠. 시간이 지나서 제가 다시 이 리스트를 봤을 때 제가 까먹을 수도 있구요. 따라서 이를 해결하기 위해 딕셔너리 라는 것이 존재합니다.

```
fruits = {“사과” : 300, “배” : 500, “포도” : 700}
```

즉, 0번, 1번, 2번 이라고 자동으로 매겨졌던 번호를 내가 원하는 대로 이름을 만들어 줄 수가 있어요. 각 과일의 가격을 출력하고 싶다하면 아래와 같이 쓰면 됩니다.

```
print(fruits[“사과”])
print(fruits[“배”])
print(fruits[“포도”])
```

인덱싱을 할 때, 인덱스 번호가 아닌 내가 만든 인덱스 이름을 사용할 수 있게 되었습니다.

딕셔너리란 개념이 조금 헷갈릴 수도 있는데, 딱 이렇게만 생각해주세요!

- 자동으로 매겨지는 인덱스 번호가 아닌, 내가 인덱스 이름을 만들어 줄 수 있다.
- 인덱싱을 사용할 때, 내가 만든 인덱스 이름을 사용할 수 있다.

리스트는 대괄호([]), 튜플은 소괄호(()) 로 둘러쌓여 있었다면, 딕셔너리는 중괄호({})로 둘러쌓여 있습니다.

딕셔너리란 말 그대로 사전의 형태에요. 한영사전을 예로 들면 다음과 같은 형태로 단어들이 나열되어 있습니다.

```
야구 : baseball
축구 : football
농구 : basketball
```

이렇게 콜론(:) 을 기준으로 앞에 있는 것을 key, 뒤에 있는 것을 value 라고 합니다.

자 그럼 다시 과일로 돌아와서.. 한번 value 값들만 뽑아 볼게요.

values() 함수를 사용하면 됩니다. 함수의 사용법을 항상 주의하세요.

변수이름 뒤에 마침표(.) 를 찍고 그다음 함수 이름을 씁니다.

```
>>> fruits = {"사과" : 300, "배" : 500, "포도" : 700}
>>> fruits.values()
dict_values([300, 500, 700])
```

반대로 key 값들만 뽑을 수도 있어요.
keys() 함수를 사용하면 됩니다.

```
>>> fruits = {"사과" : 300, "배" : 500, "포도" : 700}
>>> fruits.keys()
dict_keys(['사과', '배', '포도'])
```

리스트에서는 값을 추가하고 싶을 땐 append() 함수를 사용했습니다. 아래처럼요!

```
>>> price = [300, 500, 700]
>>> price.append(900)
>>> price
[300, 500, 700, 900]
```

딕셔너리에서는 값을 추가하고 싶을 땐 따로 함수를 사용하지 않고, 그냥 이렇게 간단하게 할 수 있습니다.

```
>>> fruits = {"사과" : 300, "배" : 500, "포도" : 700}
>>> fruits["수박"] = 900
>>> fruits
{'사과': 300, '배': 500, '포도': 700, '수박': 900}
```

예제)

1. 아래 표를 딕셔너리 형태로 만들어 그 딕셔너리를 출력해보세요.

항목	값
name	앤트맨
age	35
height	10mm
weight	50g

2. 앤트맨의 키가 12m로, weight가 10ton으로 바뀌었습니다. 딕셔너리에 값을 변경하여 그 딕셔너리를 출력해보세요.

[3-6. 불]

파이썬에서는

`True`

`False`

를 불 자료형이라고 합니다.

원래 큰따옴표(또는 작은 따옴표) 로 둘러쌓여 있지 않은 문자들은 이를 변수 이름으로 인식하는 거 이제는 알고 계시죠~? 그런데 `True/False` 만큼은 변수 이름이 아닌 “불 자료형”이라고 인식합니다.

0주차 요약은 요기까지입니다.

0주차 내용은 파이썬을 다루는데 있어 매우매우매우매우매우매우매우매우매우 중요합니다.

영어로 치면 알파벳 같은? 이걸 알아야 진도를 나갈 수 있는 그런 내용입니다. 꼭 완벽히 익히세요!

혹시 눈으로만 보셨다면 지금 얼른 컴퓨터를 켜시고 손가락을 움직이세요..!