```
def plus(i, j):
    return i+j
```

파이썬으로
'두 정수를 받아(Parameter = 매개변수)'
'합을 출력하는(ReturnType = Int)' 함수

```
int plus(int i, int j){
    return i+j;
}
```

C, Java로 '두 정수를 받아(Parameter)'
'합을 출력하는(ReturnType = Int)' 함수
Return값이 Int이므로 함수 이름 앞에 Int를 써준다.

```
public static int plus(int i, int j){
   return i+j;
}
```

반환 값은 없을 수도 있으며, 새로운 함수를 호출하여 나온 값일수도 있다.
C와 Java의 경우에는 결국 쓴 'ReturnType'만 유지하면 오류가 나지 않는다.
반환 값이 없는 경우에는 ReturnType에 void를 사용한다.

```
def plus(i, j):
    sum = i + j
    return sum
print(plus(3, 5))
```
초기

```
def plus(i, j):
    sum = i + j
    return sum
print(plus(3, 5))
```
1번과정

```
def plus(i, j):
    sum = i + j
    return sum

print(plus(3, 5))
```
2번과정

```
def plus(i, j):
    sum = i + j
    return sum

print(plus(3, 5))
```
3번과정

```
def plus(i, j):
    sum = i + j
    return sum

print(8)
```
변수를 반환하는게 아니라,
변수가 갖고있는 값을 반환한다.

```
def plus(i, j):
    sum = i + j
    return sum
print(plus(3, 5))
# res : 8
```

함수의 사용
파일의 어디서든지 해당 함수 이름과 집어넣을 매개변수를 넣어 호출한다.
당연히 함수 안에서도 함수를 호출할 수 있다.
특수하게 어떤 함수가 자기 자신을 호출할 경우 '재귀함수'라고 한다.

함수의 진행
1. print(plus(3, 5))에서, print문보다 안쪽에 있는 plus(3, 5)가 호출된다.
2. plus함수는 적절한 매개변수와 함께 호출되었으므로 함수의 내용을 실행한다.
3. 함수는 return을 만나면 함수는 return 우측의 변수를 반환하고 종료한다.

예시는 파이썬으로 들었으나, C와 Java 모두 같은 방식이다.

```
public static void iLoveInt(){
    System.out.println("So I will return Int!");
    return 3;
}
```

컴파일 에러
iLoveInt의 반환타입은 void인데 Int를 반환합니다.
※ 반환타입은 반드시 지정했던 것과 같아야한다.

```
public static int iHateInt(String k){
    System.out.println("I hate Int!");
    int k = 0; return k;
}
public static void main(){
    System.out.println(iHateInt(3));
}
```

실행 에러
iHateInt는 매개변수로 String을 받지만
주어진 매개변수는 Int형입니다.

```
public static int sadILoveInt(){
    System.out.println("OK I will stay..");
    return;
}
```

컴파일 에러
sadILoveInt의 반환타입은 int인데 void를 반환합니다.

```
public static int plusOne(int i){
    return 0;
    i++; return i;
}
public static void main(){
    System.out.println(plusOne(3));
} // result : 0
```

오류는 나지 않지만 원하는 값이 안 나옴
함수는 실행 중 return을 만나면
바로 반환하고 종료하기때문에,
i++; return i;에 도달할 수 없습니다.

return의 생략
아무것도 반환할 필요가 없는 함수의 경우에는 return을 쓸 필요가 없다.
좋은 예시로, Java의 main함수는 return이 생략되어있다.
생략 가능한 선제 조건으로는 반환타입이 void여야 한다.

```
def plus(i, j):
    print(i+j, end=', ')
print(plus(3, 5))
# result : 8, None
```

파이썬은 return 우측에 value가 없거나,
Return이 없이 함수가 종료된 경우
None을 반환한다.
참고로 파이썬은 main함수가 없다.

```
public static void main(String args[]){
    System.out.println("hi");
} // 자바의 메인함수
```

자바의 main함수는 returnType이 void인 대표적 예시이다.
모든 언어에서, main함수의 return은 프로그램 종료를 뜻한다.

```
void plus(int i, int j){
    printf("%d", i+j);
}
Int main(){
    int k = plus(3, 5);
}
```

plus는 void를 반환하지만, k는 int로 선언되어있어
k에 plus값을 대입할 수 없다.
사실상 반환형이 void인 함수는 어떤 변수에도 대입할 수 없다.

함수의 순서
함수는 마지막에 호출된 것을 가장 먼저 종료하는 특성을 갖고있다.

```
public static int plus(int i, int j, int k, int l){
    int tmp1 = plus(i, j);
    int tmp2 = plus(k, l);
    return tmp1+tmp2;
}

public static int plus(int i, int j){
    int sum = i+j;
    return sum;
}

public static void main(String args[]){
    System.out.println(plus(1, 2, 3, 4));
}
```

1. Main함수에서 plus(1, 2, 3, 4)가 호출된다.
2. Plus함수에서 plus(1, 2)가 호출된다.
3. Plus(1, 2)는 sum에 3을 대입, sum을 반환하고 종료한다.
4. Plus함수에서 plus(3, 4)가 호출된다.
5. Plus(3, 4)는 sum에 7을 대입, sum을 반환하고 종료한다.
6. Plus(1, 2, 3, 4)는 각 두 값을 더한 뒤 반환하고 종료한다.
7. Main함수는 들어온 값으로 println를 실행한 뒤 종료한다.

매개변수의 기본값
함수의 매개변수에, 입력 받지 않아도 자동적용 되는 기본값을 설정할 수 있다.
그냥 변수에 = 쓰고 넣을 값을 쓰면 된다.
모든 언어에 적용되는 규칙이 있는데, 다음과 같다.
1. 반드시 입력받아야 하는 매개변수를 앞에 쓰고, 기본값이 있는 매개변수를 뒤에 쓴다.
2. 함수를 호출 할 때, 이 기본값을 이번에만 바꾸려면 (변수 이름) = (바꿀 값)으로 하거나
   해당 변수 자리에 값을 입력하면 된다.

```
def plus(i=1, j=2):
    sum = i + j
    return sum
print(plus(), plus(3), plus(j=3)
#result : 3 5 4
```

재귀함수
함수는 파일의 어느곳에서나 부를 수 있다고 하였다.
자기 자신을 정의 하는 곳도 예외는 아닌데,
이런 경우는 특별히 재귀함수라는 이름을 붙인다.

```
def plus(n): # 1~n을 더하는 함수
    if n <= 0: return 0
    return n + plus(n-1)
```

```
int plus(int n){
    if (n <= 0) return 0;
    return n + plus(n-1);
```

```
public static int plus(int n){
    if (n <= 0) return 0;
    return n + plus(n-1);
}
```

```
public static int plus(int i, int j, int k, int l){
    int tmp1 = plus(i, j);
    int tmp2 = plus(k, l);
    return tmp1+tmp2;
}

public static int plus(int i, int j){
    int sum = i+j;
    return sum;
}

public static void main(String args[]){
    System.out.println(plus(1, 2, 3, 4));
}
```

이 경우는 같은 함수를 부르는게 아니다.
이름만 같고 매개변수가 다른 함수를 부르는거다.
이름이 같고 기능이 다른 함수를 작성할 수 있는 언어도 있다.

재귀함수
재귀함수를 만들때는 두가지를 생각해야한다.
첫번째로, 재귀를 어디서 부를 것인가.
두번째로, 중단은 어디서 할 것인가.

```
def plus(n): # 1~n을 더하는 함수
    if n <= 0: return 0
    return n + plus(n-1)
```

빨간색은 재귀를 부르는 곳이고,
파란색은 중단조건이다.

재귀를 부르는 위치는 위 예시처럼 간단한 경우에는 상관없지만,
코드가 복잡해지면 재귀를 어디서 불러야하는지 고려할게 많아진다.

```
def plus_odd(n): # 1~n까지의 홀수를 더하는 함수
    if n <= 0: return 0
    if n%2: return plus_odd(n-1)
    return n+plus_odd(n-2)
```

중단조건을 제대로 설정하지 않은 경우 원하는 값을 얻지 못하고,
설정했어도 중단조건에 닿지 않도록 인수를 계속 넘겨주고 있다면
최악의 경우 재귀를 탈출할 수 없게 된다.

중단조건에 도달했을 시
반드시 재귀를 이용하지 않은 return값을 주어야한다.

```
def plus(n): # 1~n을 더하는 함수
    if n <= -10: return 0 # ERROR!
    return n + plus(n+1) # !!ERROR!!
```

	재귀함수	For / while
속도	느림, 파이썬에서는 더 느림	빠름
사용빈도	쓰는 날을 세는게 빠름	안 쓰는 날을 세는게 빠름
코드 짜는 난이도	익숙해지기 전까지 답이 없음	11번 겹쳐도 쟤보단 쉬움
디버깅 난이도	익숙해져도 답이 없음 나같은 경우 재귀는 디버깅 자체를 안하고 머리로 시뮬만 돌림	1234321번 겹쳐도 쟤보단 쉬움
반드시 배워야하는가	백트래킹, DFS때문에라도 해야함. 안하면 안됨.	프로그래밍 관두기 vs for/while 배우기 택1