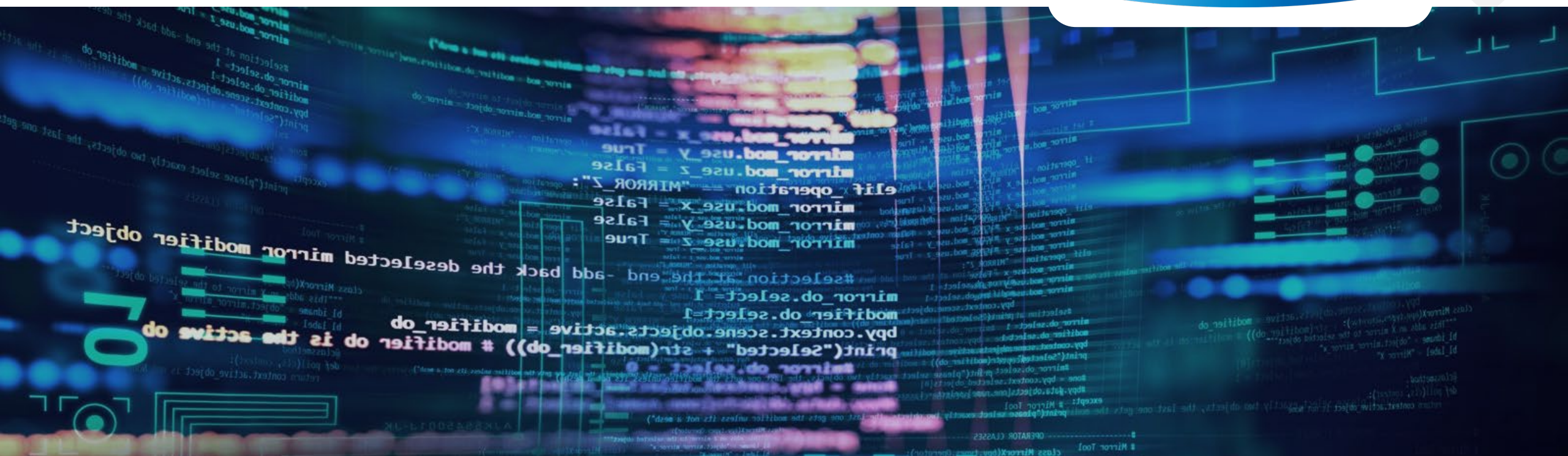


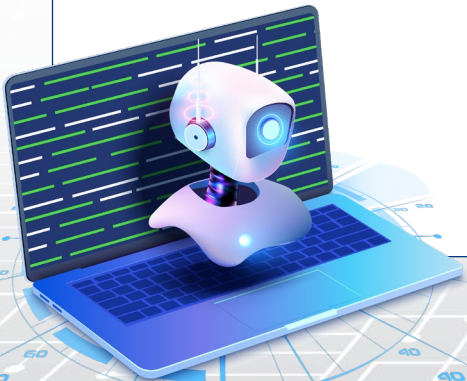
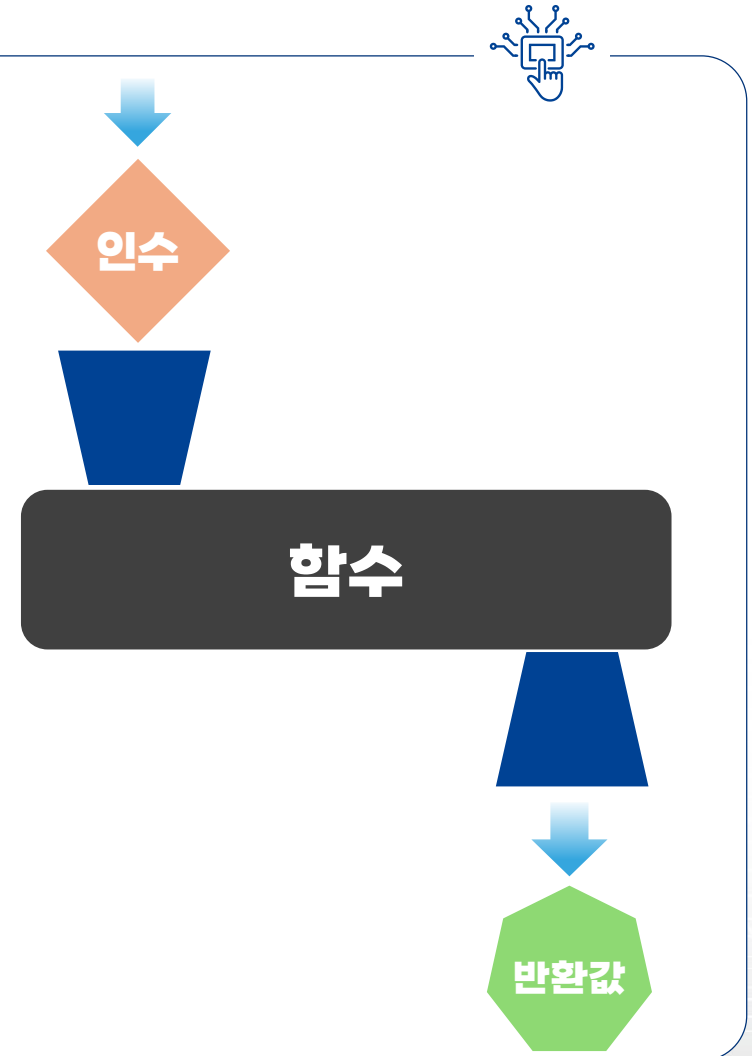
Lecture 5

함수:정의와 호출

KAIST



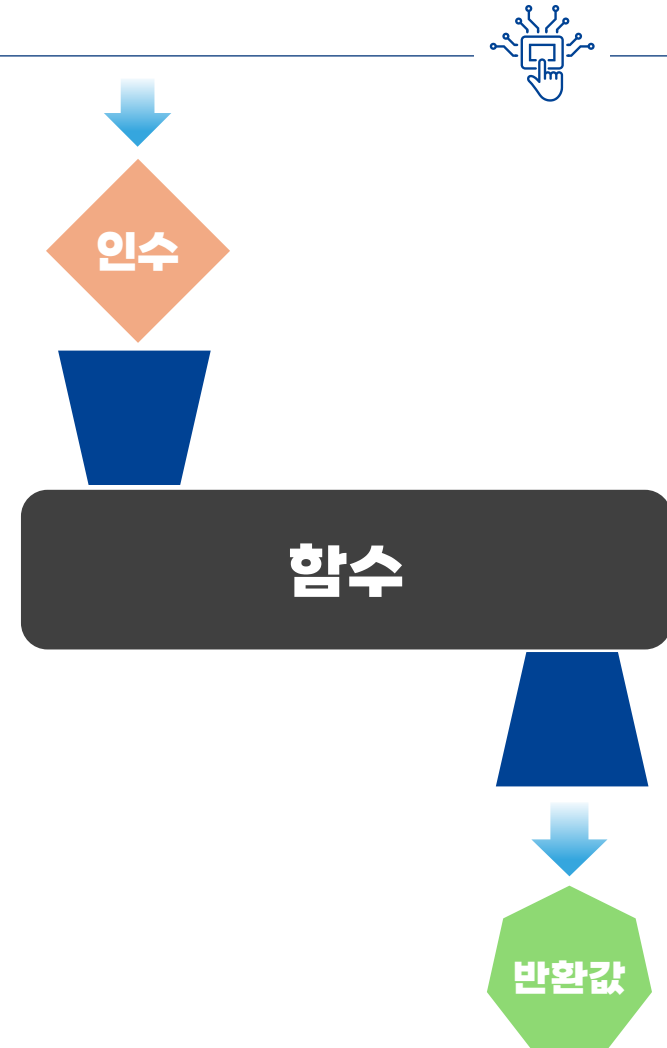
- 지금까지 print(), input() 등 다양한 함수를 써봤다
- 함수를 호출할 때 인수를 주고, 객체를 반환 받는다





함수를 왜 쓸까?

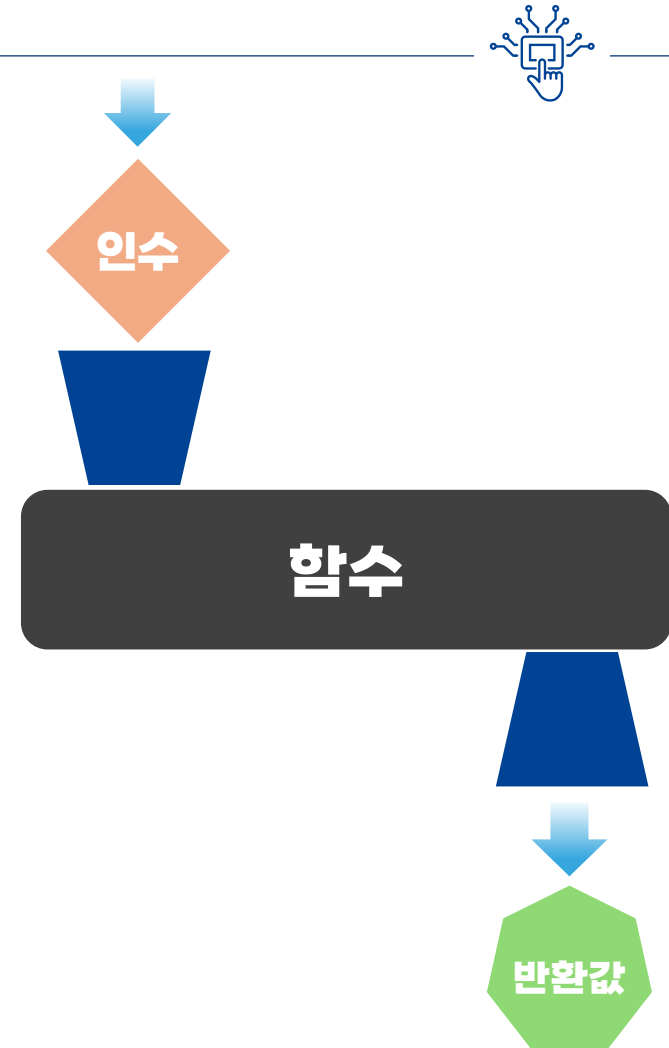
- 함수를 사용해서 함수 안에서 실행되는 코드를 숨길 수 있다
- 사용자는 함수의 코드에 무엇이 쓰이는지 하나도 몰라도 함수를 쉽게 활용할 수 있다
= 함수의 기능을 **추상화 (abstraction)**한 것
- 따라 함수를 블랙박스(blackbox)와 비유할 수 있다





함수를 왜 쓸까?

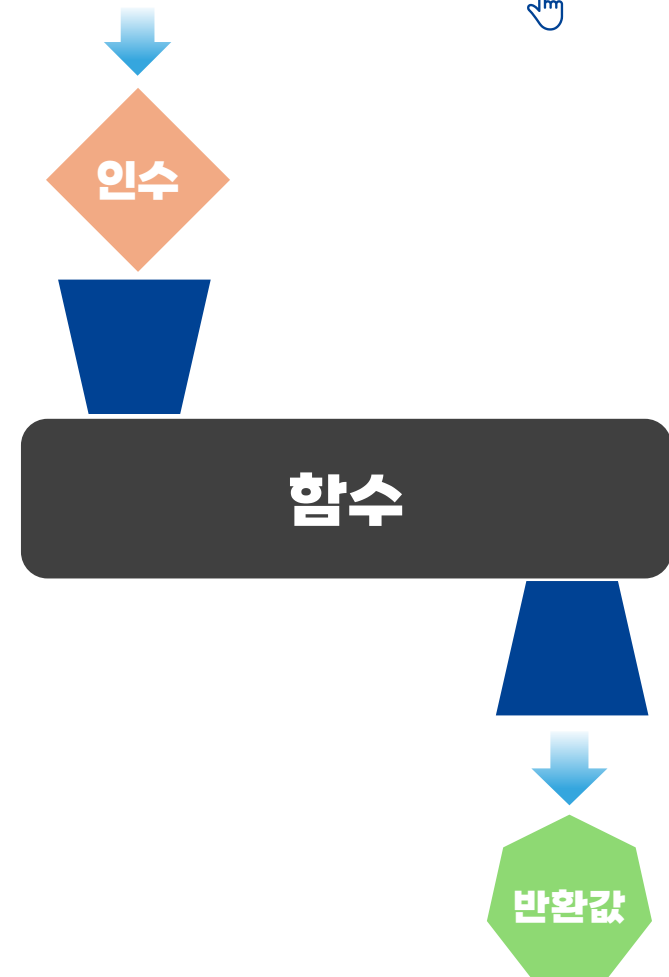
- 여러 줄의 작업을 함수로 묶으면 작업을 쉽게 반복할 수 있다
- 다른 파일에 있는 함수도 갖고 올 수 있어 자주 쓸 기능을 함수로 만들어 계속 활용할 수 있다 (이건 나중에 배울 것!)





- 함수를 def 키워드로 직접 정의할 수도 있다
- 함수 안에 실행할 코드를 들여쓰기 해야 된다

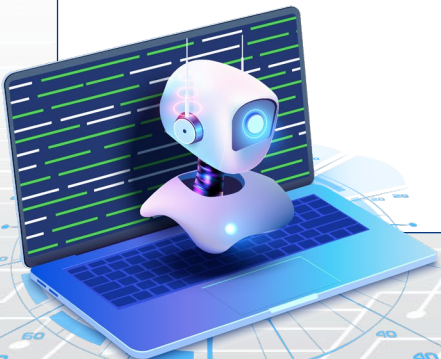
```
1  def function_name():  
2      # first line of code  
3      # .  
4      # .  
5      # .
```





- 함수를 구현한 후 활용할 수 있다
- 함수 호출은 함수의 정의 후에 나와야 한다

```
1  function_name()  ✗
2
3  def function_name():
4      # first line of code
5      # .
6      # .
7      # .
8
9  function_name()  ○
```





- 함수 안에 변수를 대입문으로 선언할 때 파이썬은 그 변수가 **지역변수라는 사실을 기억**한다
- **지역변수**란 **유효범위(scope)**가 **함수 안**인 변수 → 그 함수 안에서만 쓸 수 있다

```
1  def func1():  
2      a = 3  
3  
4  func1()  
5  print(a) # error
```





➤ 함수 외에 선언된 변수는 유효범위가 코드의 전체라 **전역변수**라 부른다

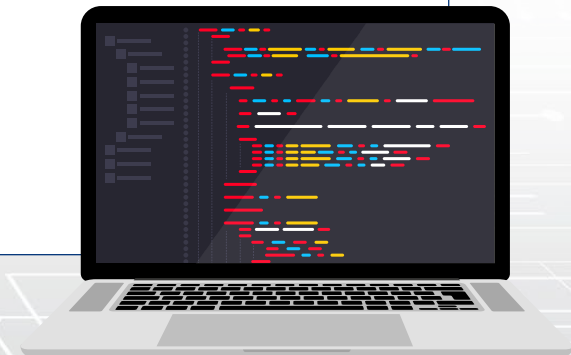
```
1     b = 3
2
3     def func1():
4         a = 3
5
6     print(b) # 3
```





➤ 함수 안에 변수가 선언이 되지 않았는데 언급이 되면 파이썬이 **전역변수 중에 찾는다**

```
1     a = 10
2
3     def func1():
4         print(a)
5
6     func1() # 10
```





- 이름이 같은 지역변수와 전역변수가 동시에 있을 수 있다: 파이썬이 알아서 잘 관리해준다
- 둘 다 있으면 함수 안에서는 지역변수가 쓰인다

```
1  a = 10
2
3  def func1():
4      a = 3
5      print(a + 1)
6
7  func1() # 4
8  print(a) # 10
```

a (global) 10

a (local) 3



➤ 함수 안에 전역변수 쓰고 싶으면 조심해야 한다

```
1     a = 10
2
3     def func1():
4         a += 1
5         print(a)
6
7     func1() # ?
```





➤ 함수 안에 전역변수 쓰고 싶으면 조심해야 한다

```
1  a = 10
2
3  def func1():
4      a += 1  # a = a + 1
5      print(a)
6
7  func1() # error
```

a (global)

10

a (local)

?



➤ global 예약어로 함수 안에 전역변수만 쓰게끔 할 수 있다

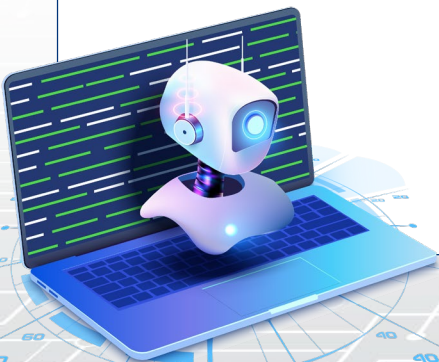
```
1     a = 10
2
3     def func1():
4         global a
5         a += 1
6         print(a - 5)
7
8     func1() # 6
9     print(a) # 11
```





- 파이썬은 함수 호출을 **호출 스택**으로 관리한다
- 스택은 **Last In, First Out (LIFO)**으로 데이터를 관리하는 데이터구조(data structure)다
- **LIFO 예시**
설거지하면서 접시를 쌓을 때 제일 최근에 쌓은 접시부터 처리를 한다

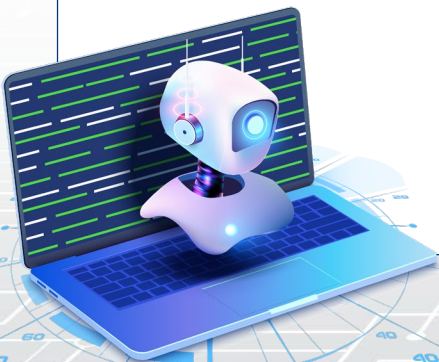
호출 스택





- 함수가 호출이 될 때 스택으로 넣는다
(어느 줄에서 호출이 되었는지도 저장이 된다)
- 파이썬은 스택 위에 있는 함수부터 처리를 하려고 한다
- 함수가 종료될 때 스택에서 뺀다

호출 스택





```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
→ 12 f1()
```



f1, 12

호출 스택



```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```



f2, 2

f1, 12

호출 스택



```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```



f3, 6

f2, 2

f1, 12

호출 스택



```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```



print, 10

f3, 6

f2, 2

f1, 12

호출 스택



```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```

Hello in f3!

f3, 6

f2, 2

f1, 12

호출 스택



```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```

Hello in f3!

f2, 2

f1, 12

호출 스택



```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```

Hello in f3!

print, 7

f2, 2

f1, 12

호출 스택



```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```

```
Hello in f3!  
Hello in f2!
```

f2, 2

f1, 12

호출 스택



```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```

```
Hello in f3!  
Hello in f2!
```

f1, 12

호출 스택



→

```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```

```
Hello in f3!  
Hello in f2!
```

print, 3

f1, 12

호출 스택



```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```

```
Hello in f3!  
Hello in f2!  
Hello in f1!
```

f1, 12

호출 스택



```
1  def f1():  
2      f2()  
3      print("Hello in f1!")  
4  
5  def f2():  
6      f3()  
7      print("Hello in f2!")  
8  
9  def f3():  
10     print("Hello in f3!")  
11  
12 f1()
```

```
Hello in f3!  
Hello in f2!  
Hello in f1!
```

호출 스택