Python Control Statement/Function

ESM2014-41

객체지향프로그래밍 및 실습

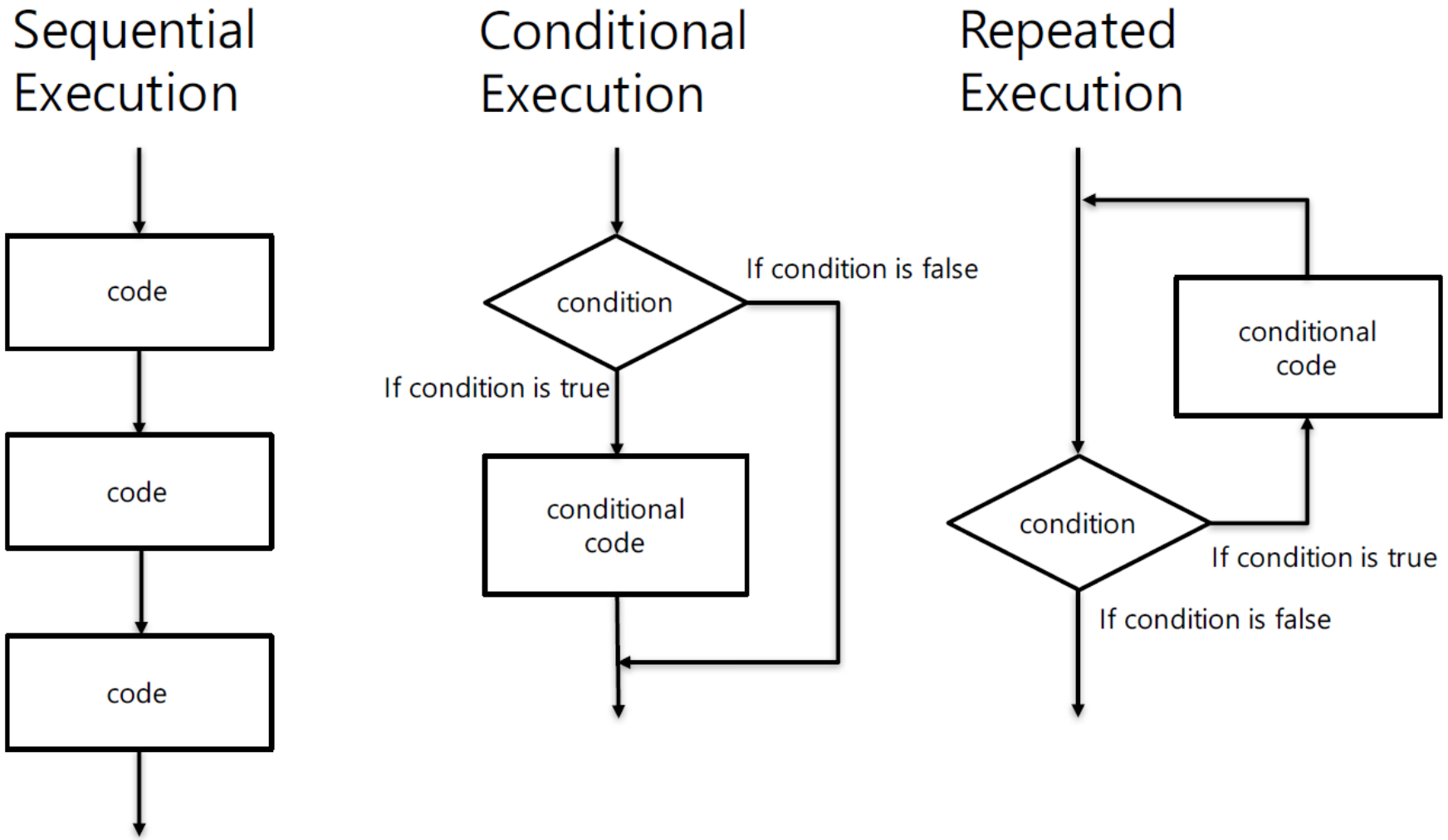
SKKU 시스템경영공학과

조영일

Python Control Statement

- 일반적으로 모든 프로그램은 작성된 코드의 위쪽에서 아래쪽으로 순차적으로 모든 코드를 실행한다.
- Control Statement(제어문)은 특정 조건에 따라 코드의 실행 여부를 결정하거나 실행 되는 순서를 변경하기 위해 사용되는 구문을 의미한다.

Python Control Statement



Python Control Statement

- Python Control Statement
  - Conditional Execution : if (elif / else)
  - Repeated Execution : while loop / for loop

Python Control Statement – Conditional Execution

- Relational Operator : 2개의 항(변수)를 비교하여 참인지 거짓인지 판단하는 연산자(결과값은 항상 Boolean)
  - == : 두 피연산자가 같으면 True
  - != : 두 피연산자가 같지않으면 True
  - < : 왼쪽 연산자가 작으면 True
  - > : 왼쪽 연산자가 크면 True
  - <= : 왼쪽 연산자가 작거나 같으면 True
  - >= : 왼쪽 연산자가 크거나 같으면 True

Python Control Statement – Conditional Execution

- Logical Operator : 여러 개의 Boolean 을 결합하여 하나의 Boolean 을 반환하는 연산자

** a = True, b = False 이라 가정한다.

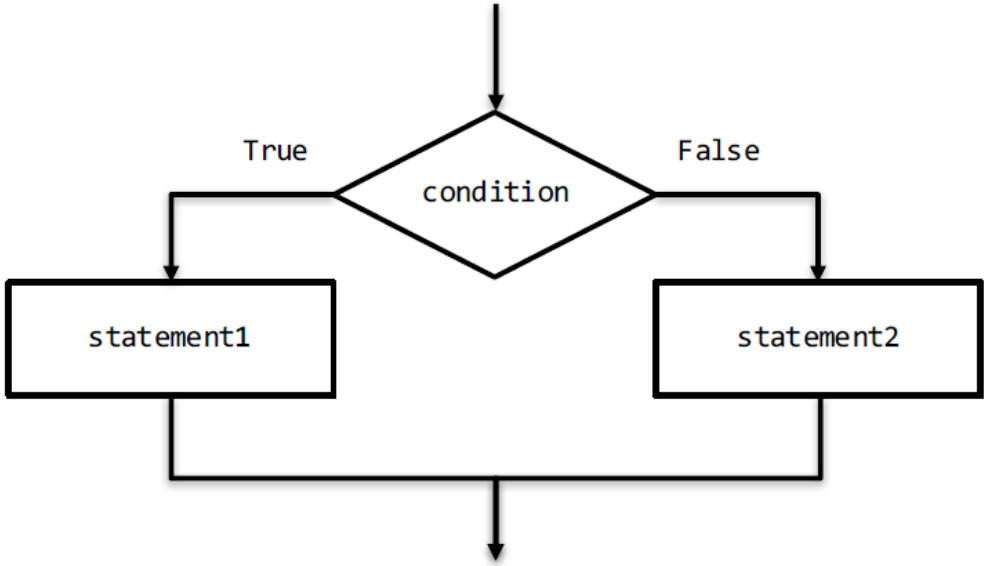
Operator	Description	Example
and	논리 AND 연산. 둘다 참일때만 참	(a and b) = False
or	논리 OR 연산. 둘 중 하나만 참이여도 참	(a or b) = True
not	논리 NOT 연산. 논리 상태를 반전	not(a and b) = True

Python Control Statement – Conditional Execution

- Conditional Execution – if : 조건이 참(True)일 경우에만, 블록 내의 코드 실행

```
if battery < 20: # Condition - Relational Operator
    print("Low Battery!")  # If "battery < 90" is true, this code will be executed.
```

Python Control Statement – Conditional Execution



- Conditional Execution – if/else : 조건이 참(True)일 경우와 거짓(True)일 경우 서로 다른 코드 실행

```
if battery < 20: # Condition
    print("Low Battery!") # If condition is true, It will be executed.
else:
    print("Normal Battery!") # If condition is false, It will be executed.
```

Python Control Statement – Conditional Execution

- Conditional Execution – if/elif/else : 복수개의 조건을 차례로 검사하고 모든 조건에 해당하지 않을 경우 마지막 else 코드 블록의 내용을 실행함

```
if score > 90:
    print("Grade : A")
elif score > 80:
    print("Grade : B")
elif score > 60:
    print("Grade : C")
elif score > 40:
    print("Grade : D")
else:
    print("Grade : F")
```

Python Control Statement – Conditional Execution

- Conditional Execution – Ternary Operator : 삼항 연산자 – if 구문의 단축 방법

```
if x < y:
    small = x
else:
    small = y
```

```
small = x if x < y else y
```

조건이 참일 경우 값

조건

조건이 거짓일 경우 값

Python Control Statement – Repeated Execution(while)

- While Loop : 특정 조건이 만족될 경우 블록내의 코드 반복 실행

```
i = 1
sum = 0
while i <= 10: # 조건
    sum += i # 실행할 구문
    i += 1
print("1부터 10까지의 합 : " + str(sum))
```

Python Control Statement – Repeated Execution(for)

- For Loop : Iterable에 대하여 Iterable의 값을 가지고 차례 차례 코드를 실행

```
list_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] # list => iterable

sum = 0

for i in list_1: # list_1의 값을 차례차례 가져옴
    sum += i # 실행할 구문

print("1부터 10까지의 합 : " + str(sum))
```

Python Control Statement – Repeated Execution

- range() 함수: 연속된 숫자의 list를 반환하는 함수
- 일반적으로 for 문과 함께 사용됨

- range(n)
  - [0, 1, 2, …, n – 1]
  - n번 반복
- range(n1, n2)
  - [n1, n1 + 1, n1 + 2, …, n2 – n1]
  - n2 – n1번 반복
- range(n1, n2, n3)
  - [n1, n1 + n3, n1 + 2 * n3, …, n1 + k * n3]
  - n1 + k * n3 < n2 일 때까지 (n3 > 0인 경우)
  - n1 + k * n3 > n2 일 때까지 (n3 < 0인 경우)

Python Control Statement – Repeated Execution(for)

- For Loop + range() example

```
sum = 0

for i in range(1, 11): # range()
    sum += i # 실행할 구문

print("1부터 10까지의 합 : " + str(sum))
```

Python Control Statement – Repeated Execution(control keyword)

- break : loop 실행을 중단하고 loop 밖으로 빠져나감

```
while True: # while True : => infinite loop
    key = input("Enter q to quit: ")
    if key == 'q':
        break  # 입력된 값이 'q'일 경우 while loop를 중단하고 빠져나감
    else:
        print("Loop Again")
print("Out of loop")


for i in ['apple', 'strawberry', 'blueberry', 'pineapple']:
    if i == 'blueberry':  # 'blueberry' 에서 for loop를 중단하고 빠져나감
        break
    print(i)
print("Out of loop")
```

Python Control Statement – Repeated Execution(control keyword)

- continue : 현재 loop의 남은 부분 실행을 중단하고 그 다음 loop로 진행함

```
i = 0
while i <= 10:
    if i % 3 == 0: # i 가 3의 배수일 경우
        i += 1
        continue # i를 1 증가 시키고 다음 코드 실행은 중단, 다음 loop 실행
    print(i)
    i += 1
```

Python Control Statement – Repeated Execution(control keyword)

- else : if 구문의 else와는 다른 기능. for/while loop 정상 종료 될 경우 블록 내의 코드 실행 됨. (break으로 종료되는 경우 실행하지 않음)

```
count = 0
while count < 10:
    key = input("Enter q to quit: ")
    if key == 'q':
        print("Break before finish")
        break # 입력된 값이 'q'일 경우 while loop를 빠져나감
    else:
        count += 1
        print("Loop Again")
else:
    print("Finishing 10 times loop") # q 입력 없이 loop 실행을 마쳤을 경우 실행됨
```

Python Control Statement – Repeated Execution(control keyword)

- pass : 아무것도 하지 않음을 명시적으로 표현하기 위한 keyword. 문법적으로 구문이 필요하지만, 특별히 수행 할 일이 없을 때 사용 가능.

```
for i in ['apple', 'strawberry', 'blueberry', 'pineapple']:
    if i == 'blueberry':
        pass # 'blueberry' 일때 아무것도 하고 수행하지 않고 싶지만 if block을 비워 두면 에러가 난다.
    else:
        print(i)
```

Python Syntax – Function

- Function(함수) : 값을 입력 받아 특정한 작업을 수행 한 이후에 결과물을 반환하는 구문들의 집합(Named sequence of statements)

- 현대 프로그래밍에서 함수는 필수적인 요소

Python Syntax – Function

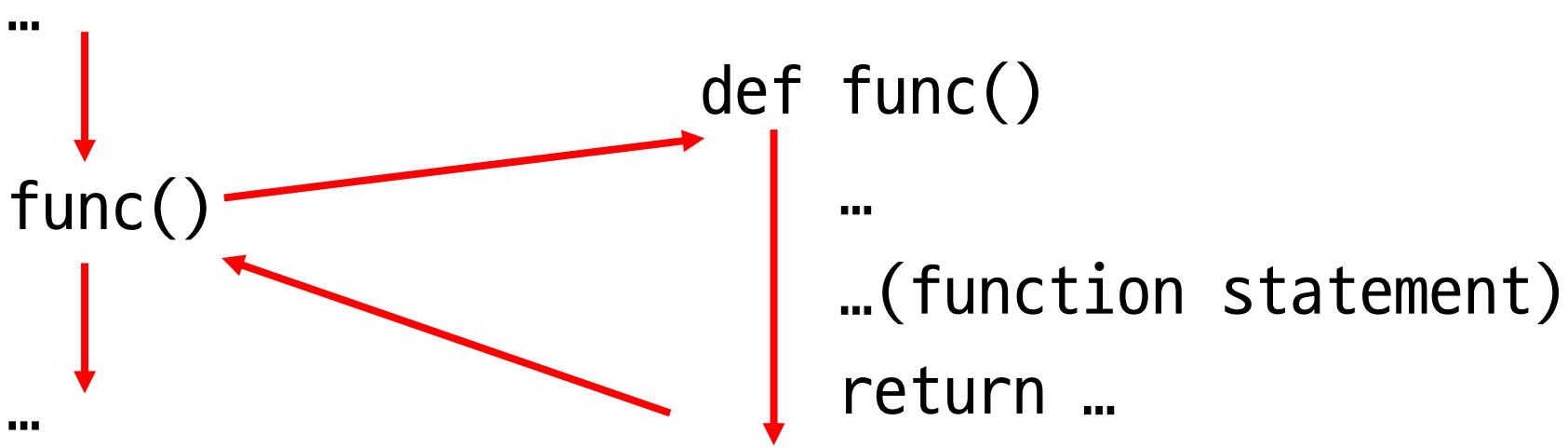
- 왜 함수를 사용하는가?
  - 프로그램의 여러 곳에서 공통으로 사용되는 기능을 하나의 함수를 통해 처리함으로써 코드의 중복을 줄인다.
  - 복잡한 기능을 작은 함수 단위로 쪼개서 해결한다.(Divide & Conquer)
  - 코드의 수정과 유지보수를 쉽게 한다.

Python Syntax – Function

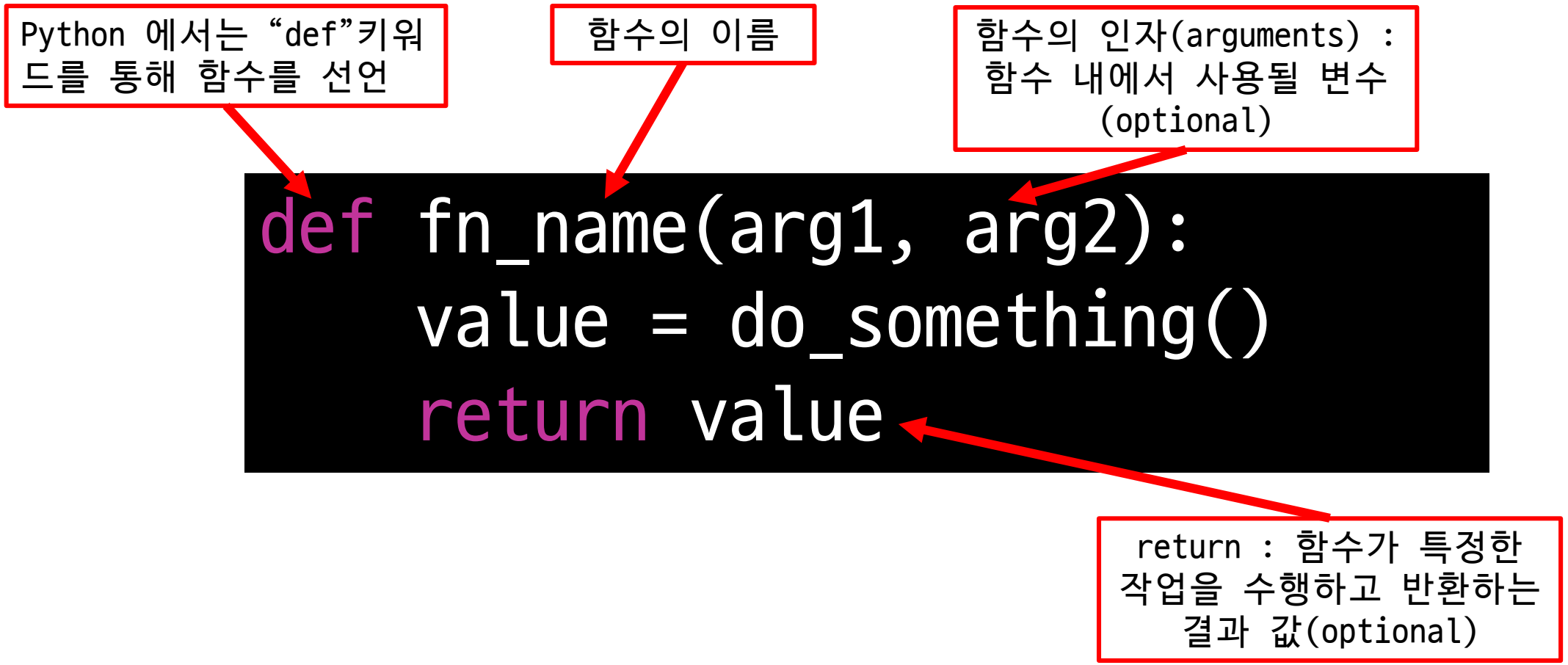
- 함수의 종류
  - 내장 함수(Built-in Function) : 일반적인 프로그래밍에서 자주 사용되는 기능들을 Python Interpreter에 기본적으로 탑재(print(), iter(), dict(), set() 등) -> 함수의 내용을 바꿀 수 없으며, 미리 정의된 사용법을 지켜서 사용하여야 함
  - 사용자 정의 함수(User-defined Function) : 사용자가 직접 정의한 함수

Python Syntax – Function

- Function Execution Flow

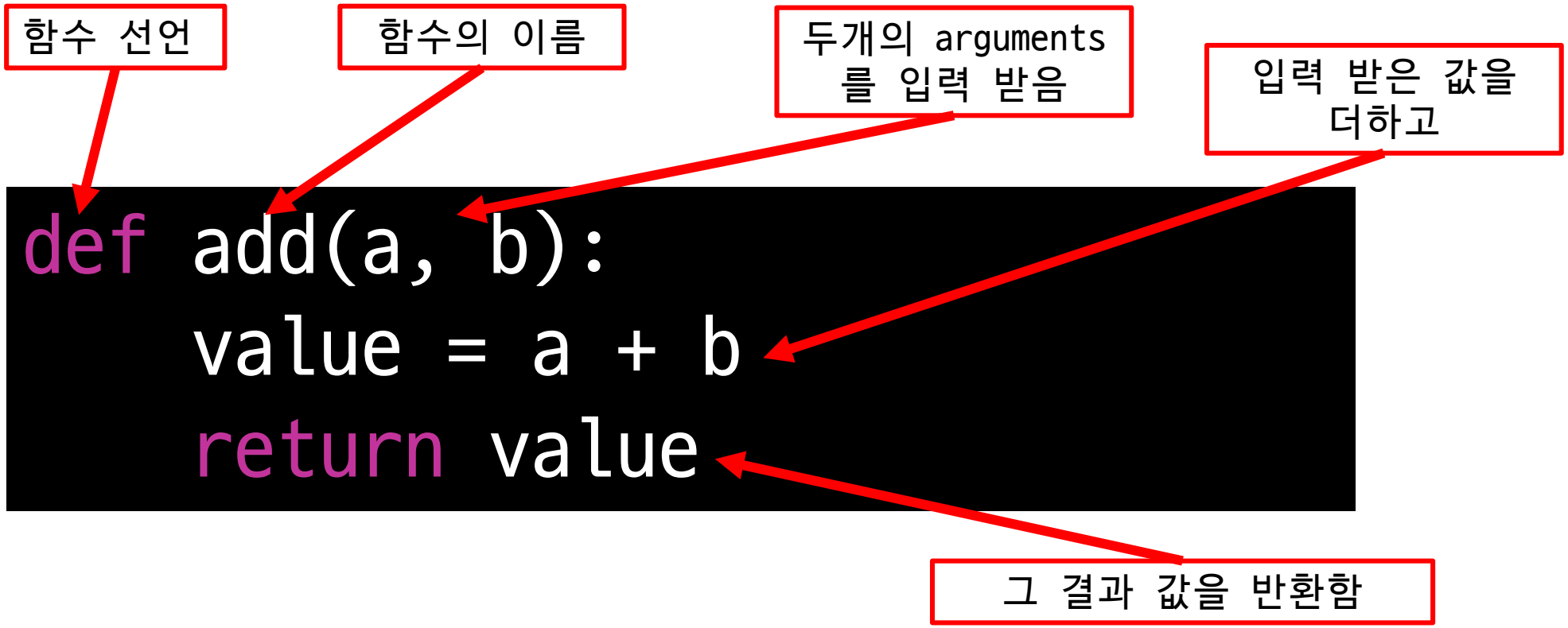


Python Syntax – Function



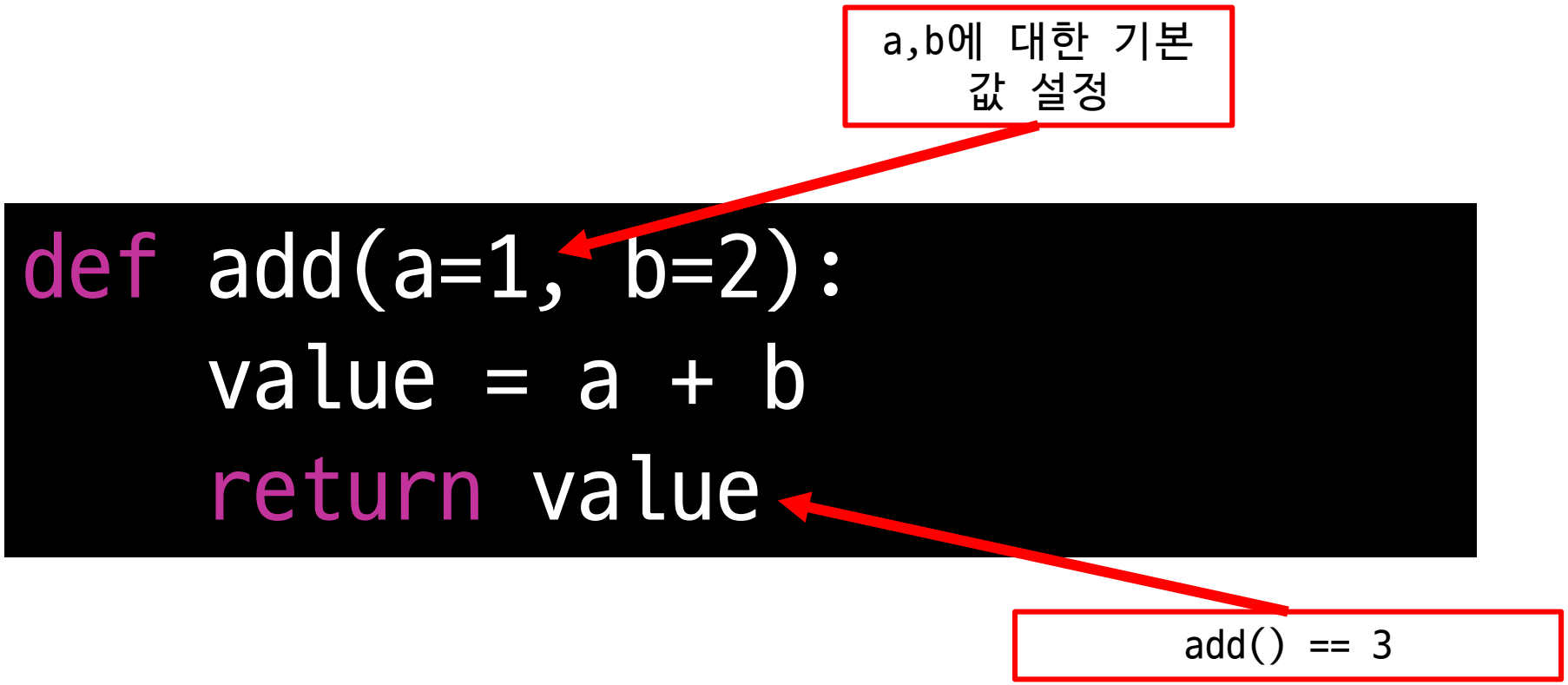
Python Syntax – Function

- 입력된 두 값의 합을 구하는 함수 add(a, b)



Python Syntax – Function

- Default Arguments Values : arguments 값을 지정하지 않고 함수를 호출하는 경우 지정된 기본값을 사용하기



Python Syntax – Function

- Positional Arguments vs Keyword Arguments
  - Positional Arguments : 순서에 따른 arguments 지정
  - Keyword Arguments : arguments의 이름에 따른 지정

```
def minus(a=1, b=2, c=3):
    value = a – b - c
    return value
```

minus(1, 2) == -4 (2 Positional Arguments)

minus(b=1, a=2) == -2 (2 Keyword Arguments)

minus(1, c=6) == -7 (1 Positional, 1 Keywords)

Python Syntax – Function

- Arbitrary Argument Lists
  - 함수의 Arguments 개수가 정해지지 않은 경우
    => asterisk(*) 을 사용하면 개수 미정의 arguments를 list 형태로 전달 가능

```
def add_all(*args):
    value = 0
    for a in args:
        value = value + a
    return value
```

```
add_all(1, 2) == 3
add_all(1, 2, 3) == 6
add_all(1, 2, 3, 4) == 10
```

Python Syntax – Function

- Keyword Dictionary Argument
  - 함수의 Arguments 개수가 정해지지 않았고, Arguments Keyword를 함께 전달하려는 경우
    => double-asterisk(**)를 사용하면 개수 미정의 arguments를 keyword와 함께 dict 형태로 전달 가능

```
def show_my_greetings(**kwargs):
    print ("Hello")
    for key, value in kwargs.items():
        print ("My " + key + " is " + value)



>>> show_my_greetings(school="SKKU", name="Jeffrey")
>>> "Hello"
>>> "My school is SKKU"
>>> "My name is Jeffrey"
```

Python Syntax – Function Variable Scope

- Variable Scope : 정의한 변수가 어디까지 유효한가?
- Local Variable : 변수가 함수 내에 정의 되어 있으며, 변수는 함수 내에서만 접근 가능하다.
- Global Variable : 변수가 함수 외부에 정의 되어 있으며, 어디서든 접근이 가능하다.

Python Syntax – Function Variable Scope

- Global vs Local Variable example

```
seconds_per_minute = 60  # 1분은 60초 => Global Variable

def minutes_to_seconds(minutes):
    """분을 입력받아 같은 시간만큼의 초를 반환한다."""
    seconds = minutes * seconds_per_minute  # Local Variable
    return seconds

print(minutes_to_seconds(3))  # 화면에 180이 출력된다
print(seconds)  # Error : 함수 밖에서 Local Variable을 사용함
```

Python Syntax – Function Variable Scope

- Global vs Local Variable

특징	Global	Local
함수 안에서 읽기	가능	가능
함수 안에서 수정	불가(*)	가능
함수 밖에서 읽기	가능	불가
함수 밖에서 수정	가능	불가

(*) ‘global’ keyword 사용할 경우

Python Syntax – Function Variable Scope

- Edit global variable in function

```
count = 0  # Global Variable

def get_total_count():
    count = count + 1  # Will work?
    print(count)

get_total_count()
>>> UnboundLocalError: local variable 'count' referenced before assignment
```

Python Syntax – Function Variable Scope

- Edit global variable in function – ‘global’ keyword

```
count = 0  # Global Variable

def get_total_count():
    global count # 'global' keyword를 사용하면 함수 내에서 전역변수 수정 가능
    count = count + 1  # Will work?
    print(count)

get_total_count()
>>> 1
get_total_count()
>>> 2
```

Python Syntax – Function Variable Scope

- General Rule about Variables

1. 함수는 복잡한 전체 문제를 작게 쪼개어 해결하기 위한(Divide & Conquer) 수단이다 : 함수 내에서 여러 함수가 공유하는 Global Variable을 수정하기 시작하면 함수들은 Global Variable 로 인해 서로 상호작용을 하게 되며, 이는 예측 할 수 없는 결과를 불러온다.

2. 따라서 함수 내에선 가능한 Local Variable만을 수정하며 'global' 키워드를 통한 Global Variable 수정은 피한다.

3. Global Variable은 프로그램 전체적으로 공유되어야 하는 값, 한번 정의되면 잘 변하지 않는 값을 저장 하는 용도로만 사용한다.

4. Global Variable을 함수 내에서 다루려면 : 함수의 arguments를 사용한다.

Python Syntax – Function Variable Scope

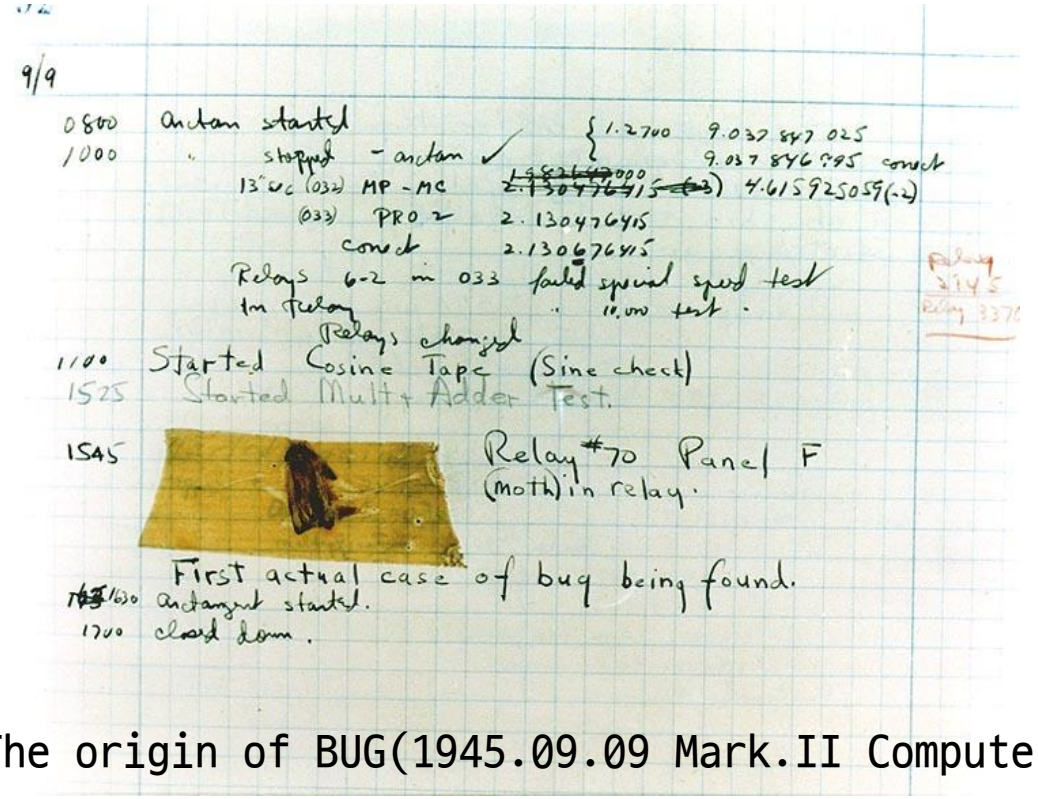
- Using global variable by arguments

```
count = 0  # global variable

def get_total_count(count):  # passing global variable by arguments
    count = count + 1 # arguments로 전달된 count는 함수 내에서 local variable임
    print(count)
    return count

count = get_total_count(count)  # 함수의 return value를 glabal variable에 대입
count = get_total_count(count)
```

Appendix – Debugging Technique



The origin of BUG(1945.09.09 Mark.II Computer)

- What is Bug?

요구사항과 다르게 동작하는 프로그램의 잘못된 동작 혹은 오류

Appendix – Debugging Technique

- (이 세상 거의 모든) 프로그램에는 버그가 있다.
- 사람이 작성하다 보니 모든 상황에 대한 테스트가 불가능 하기 때문
- 따라서, 코드를 작성하는 과정에는 항상 버그를 없애기 위한 상당한 노력이 수반됨

Appendix – Debugging Technique

- What is Debugging?

프로그램상 오류(Bug)를 없애는(De) 과정(-ing) = Debugging

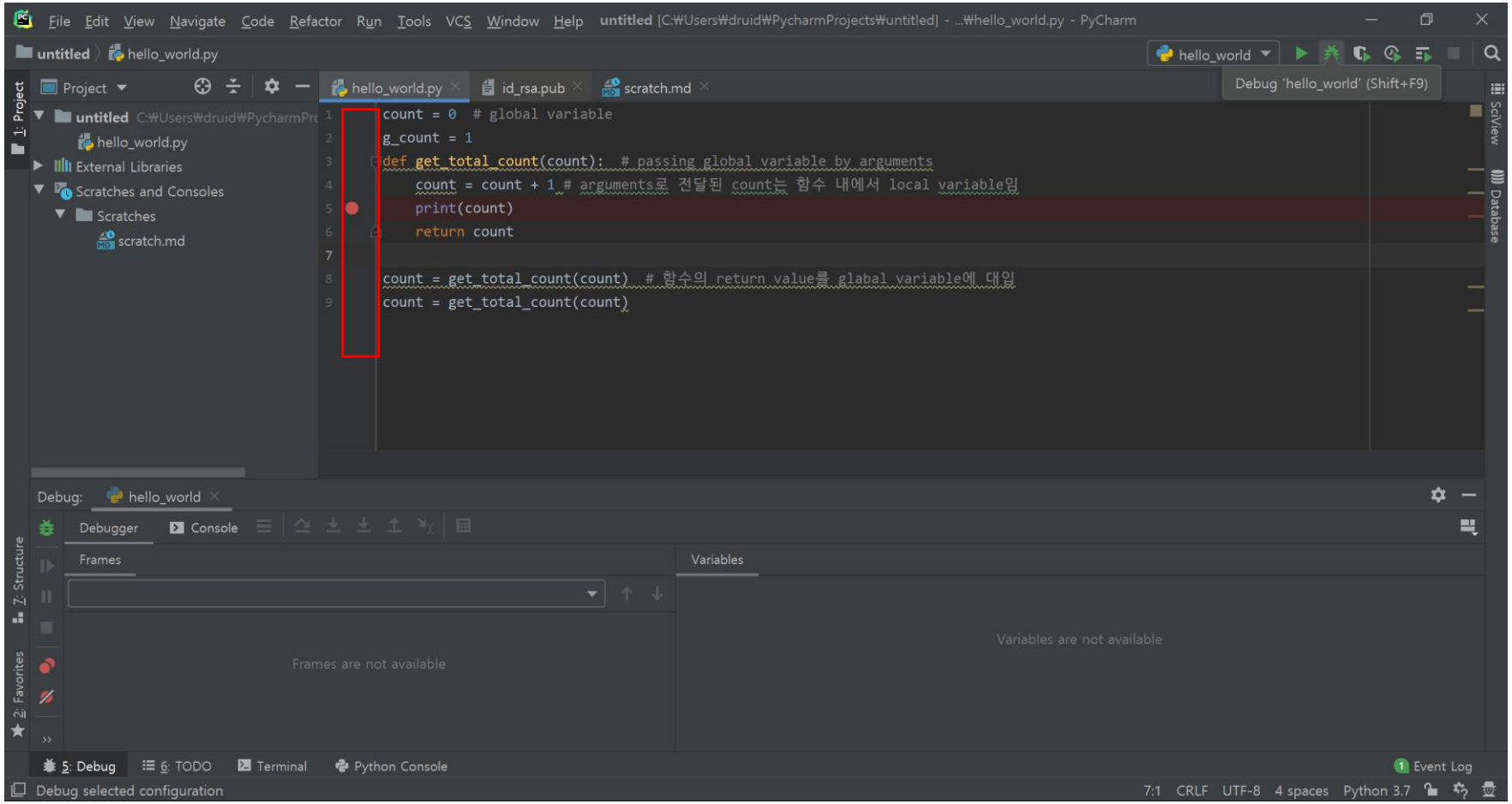
- 대부분의 통합개발환경(IDE)는 사용자의 디버깅 과정을 돕는 디버깅 도구(Debugging Tools)를 탑제하고 있음

Appendix – Debugging Technique

- Key feature of Debugging Tools
  - Break Point(중단점) : 코드의 실행을 사용자가 지정한 위치에서 중단시키는 역할
  - Frames(Call Stack) : 중단점 까지의 코드가 실행되기 위해 호출된 함수의 목록
  - Variables : 중단점에서의 Local Variable 들의 값
  - Watches : 중단점에서 Local Variable 이외의 값들을 알아내고자 할 경우

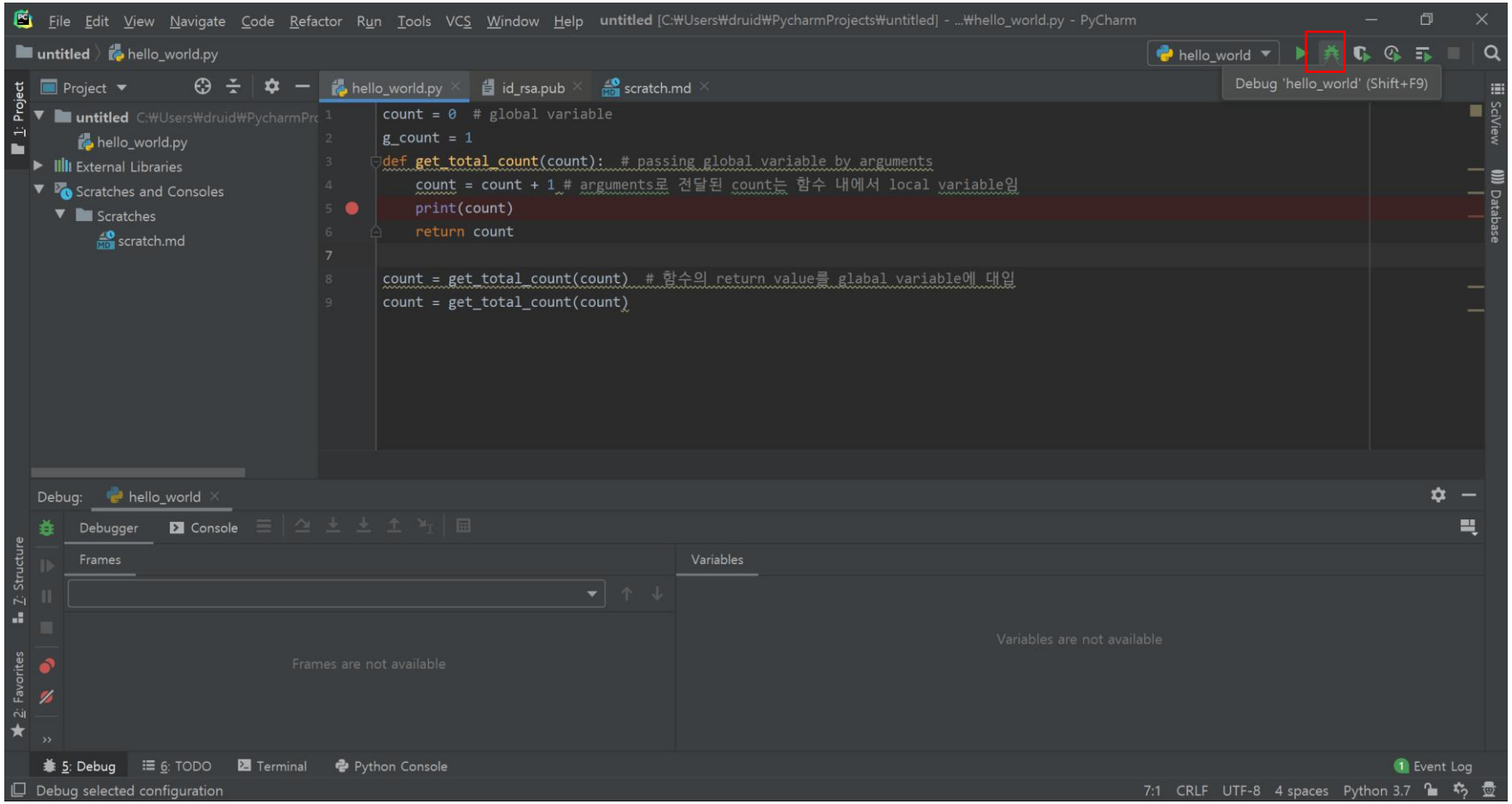
Appendix – Debugging Technique

- Break Point



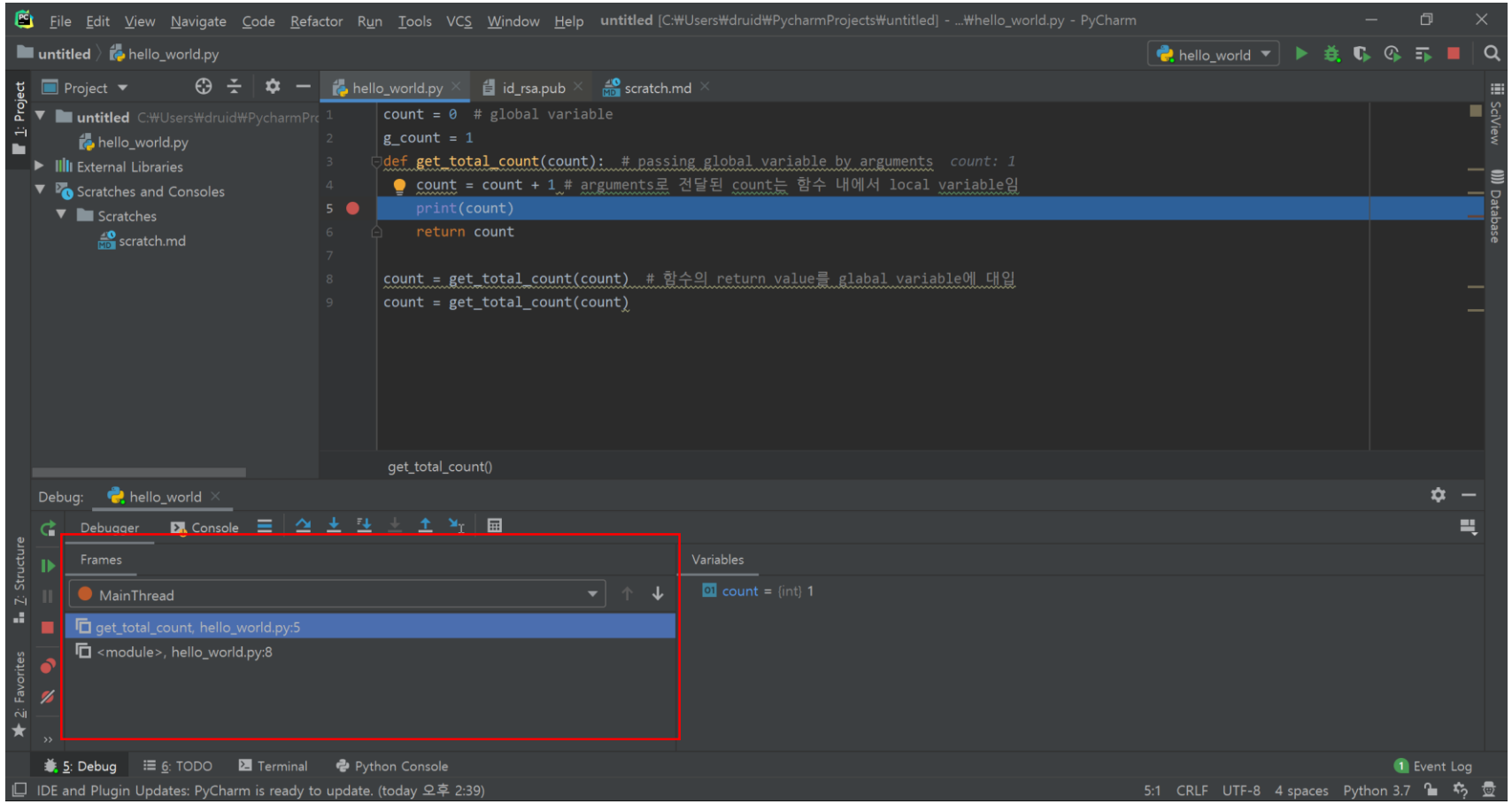
Appendix – Debugging Technique

- Entering “Debug Mode”



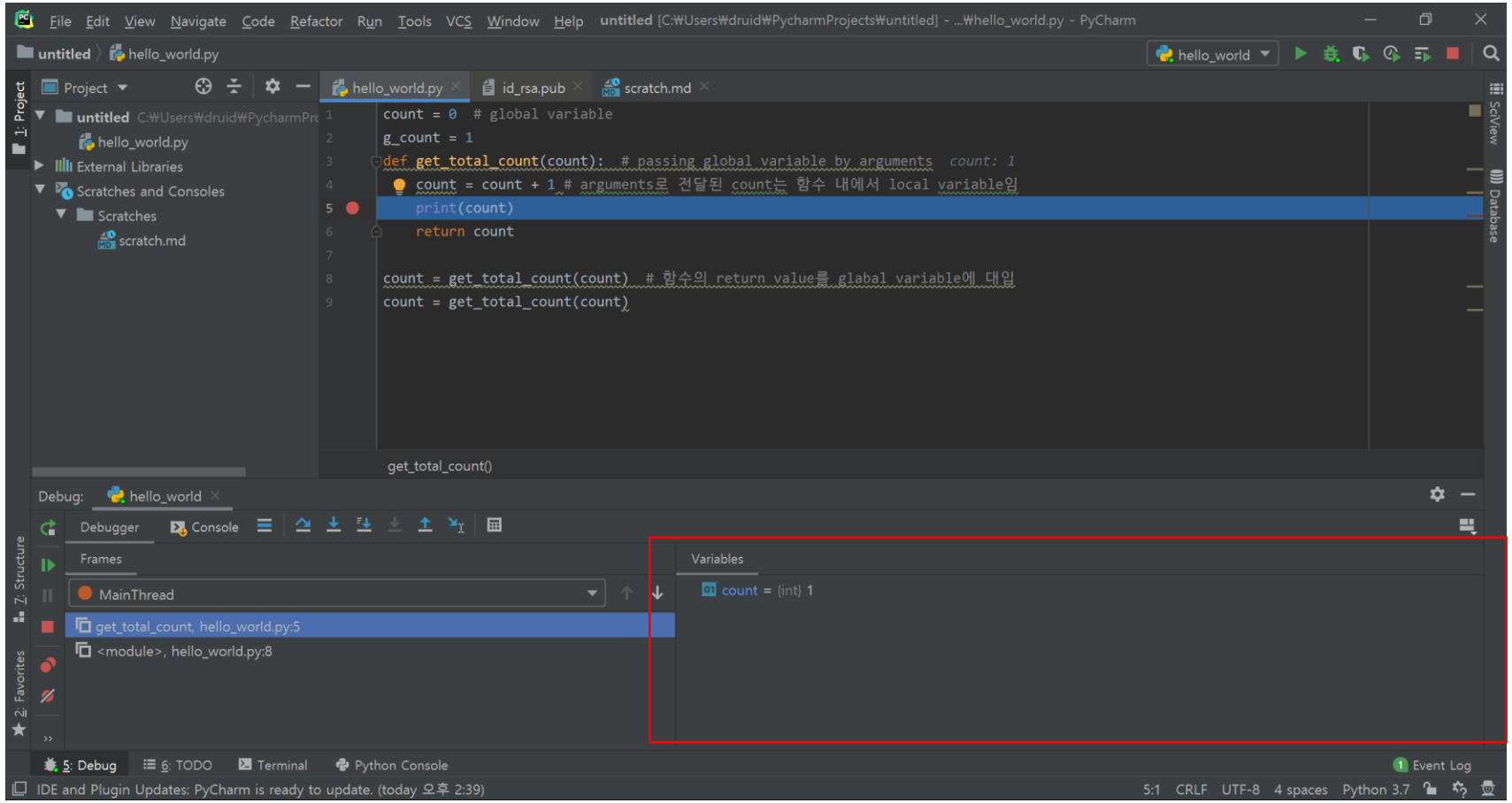
Appendix – Debugging Technique

- Frames(Call Stack)



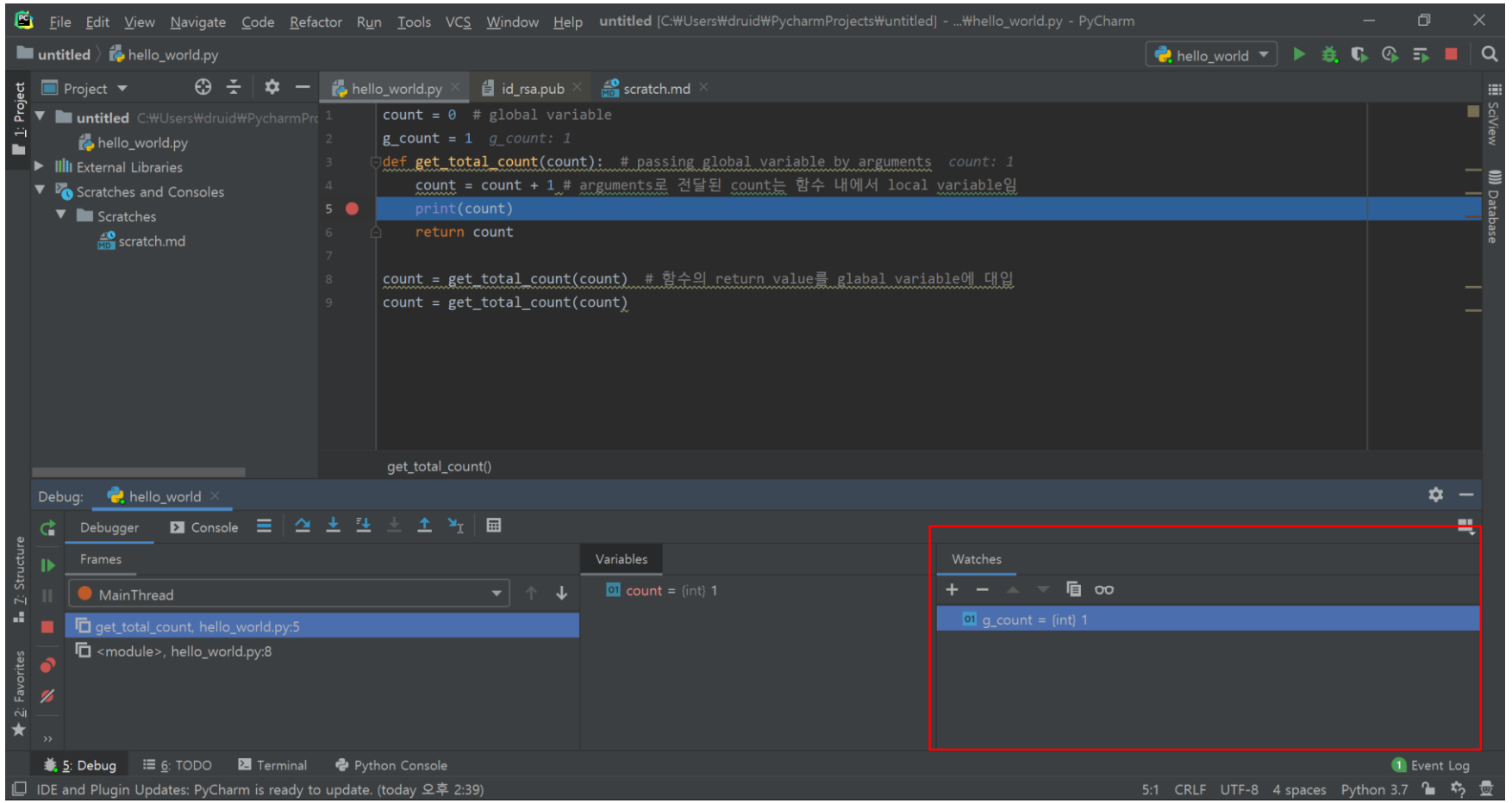
Appendix – Debugging Technique

- Variables



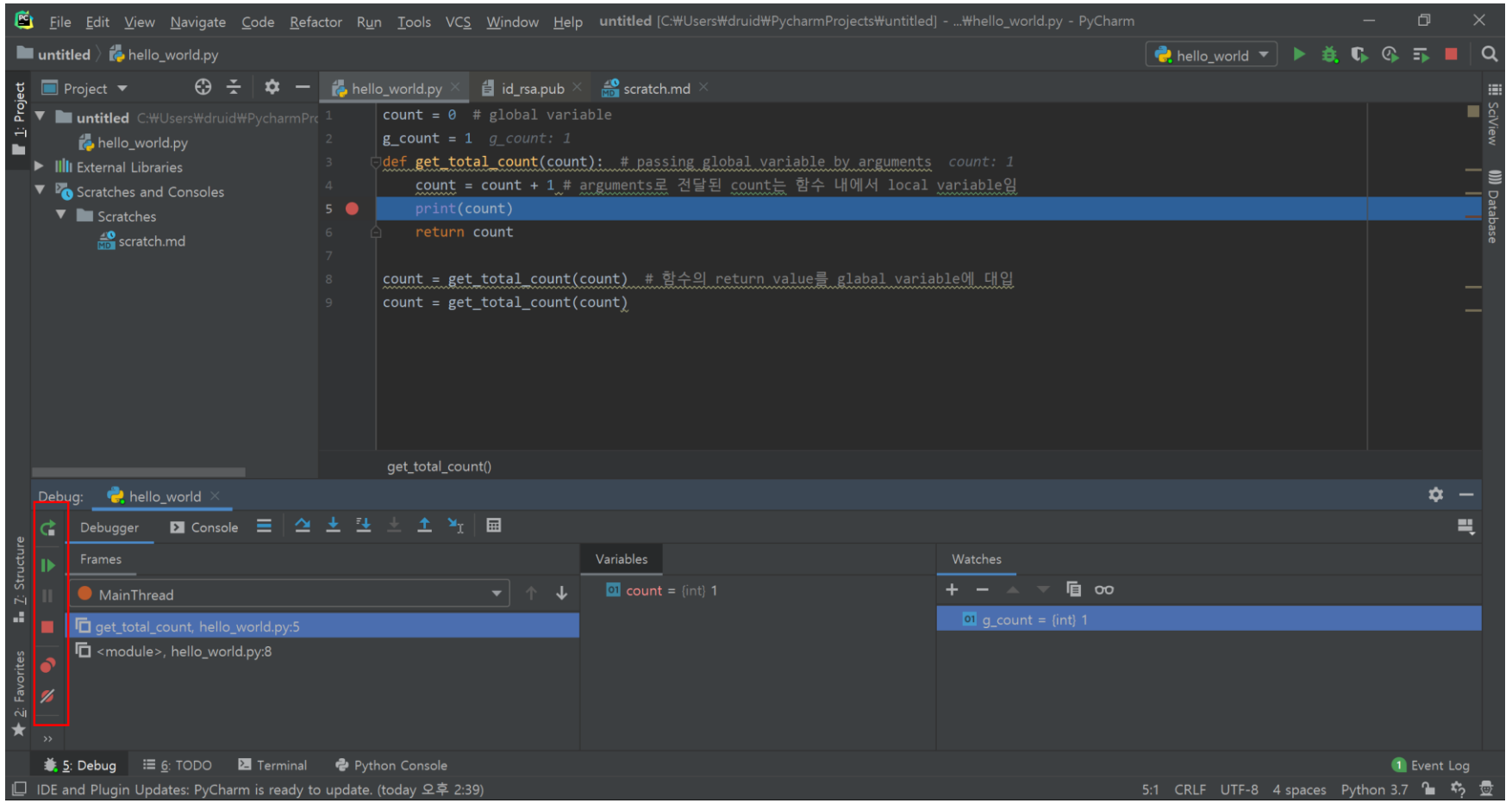
Appendix – Debugging Technique

- Watches



Appendix – Debugging Technique

- Execution Control : Rerun, Resume, Pause, Stop



Summary

- Control Statement – Conditional Execution(if/elif/else)
- Control Statement – Repeated Execution(while/for loop)
- Function Definition
- Function Arguments
- Global vs Local Variable

- Debugging Technique