

# Control Flow & Function

# Control Flow

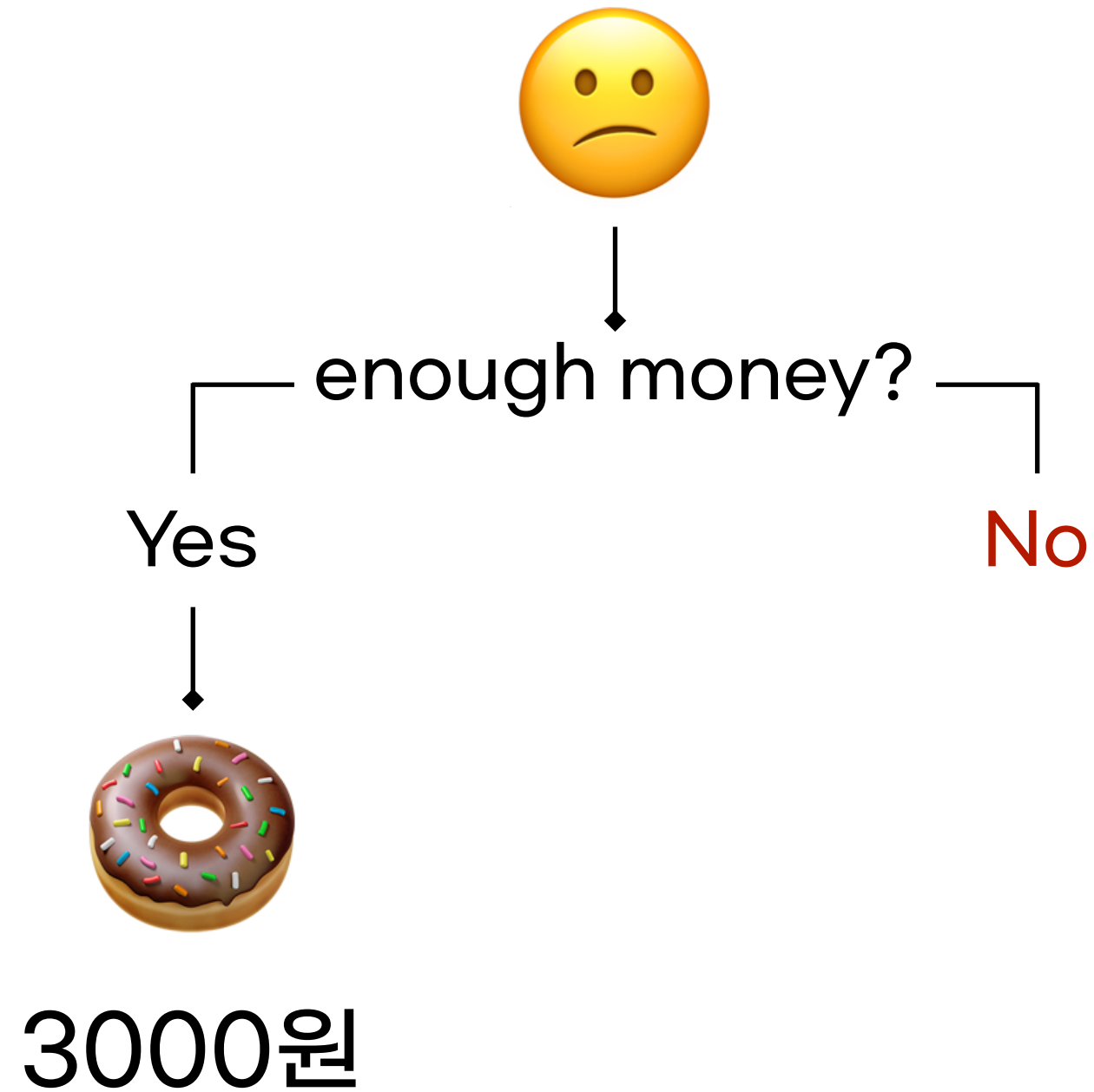


# Control Flow

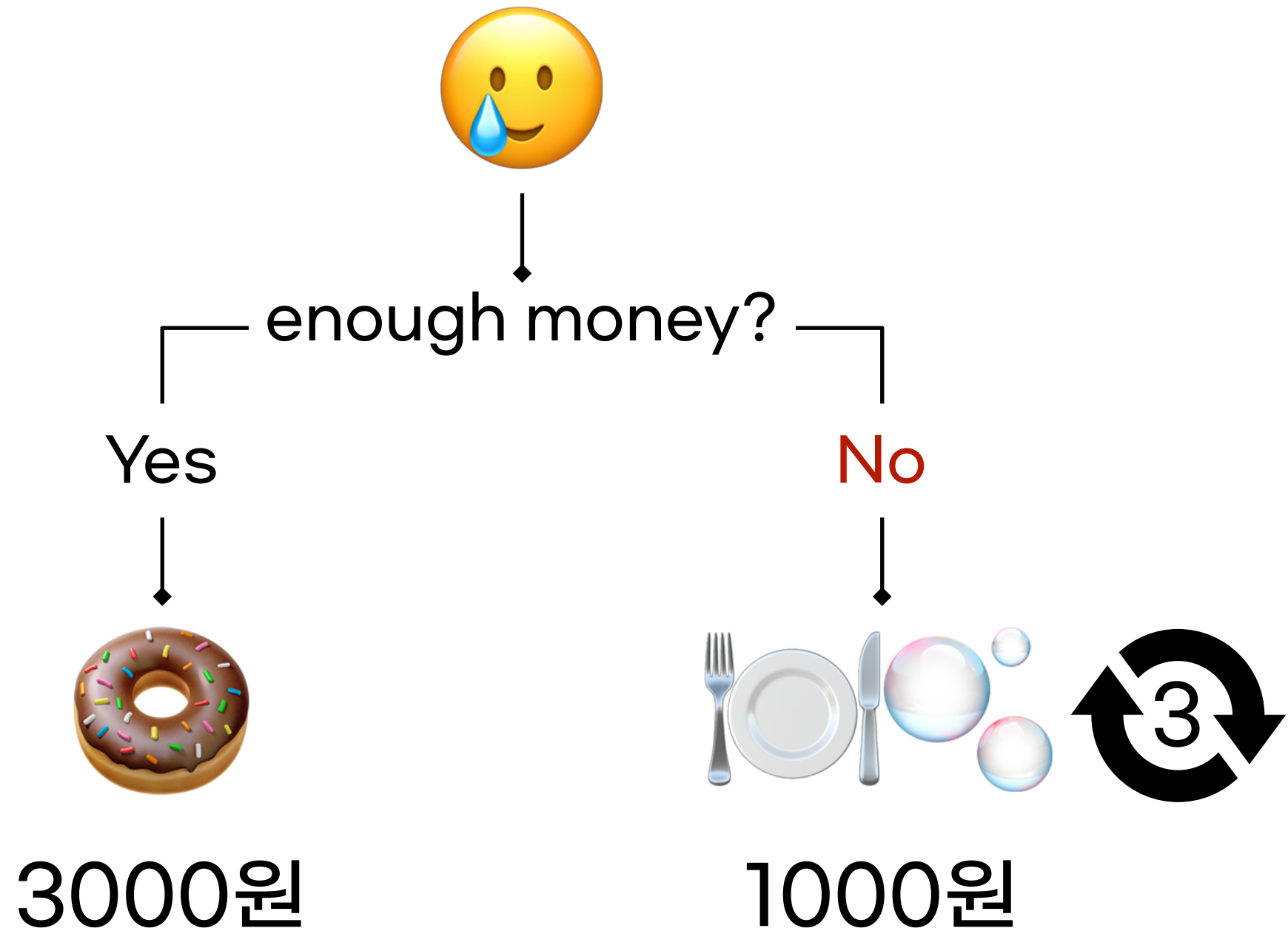


3000원

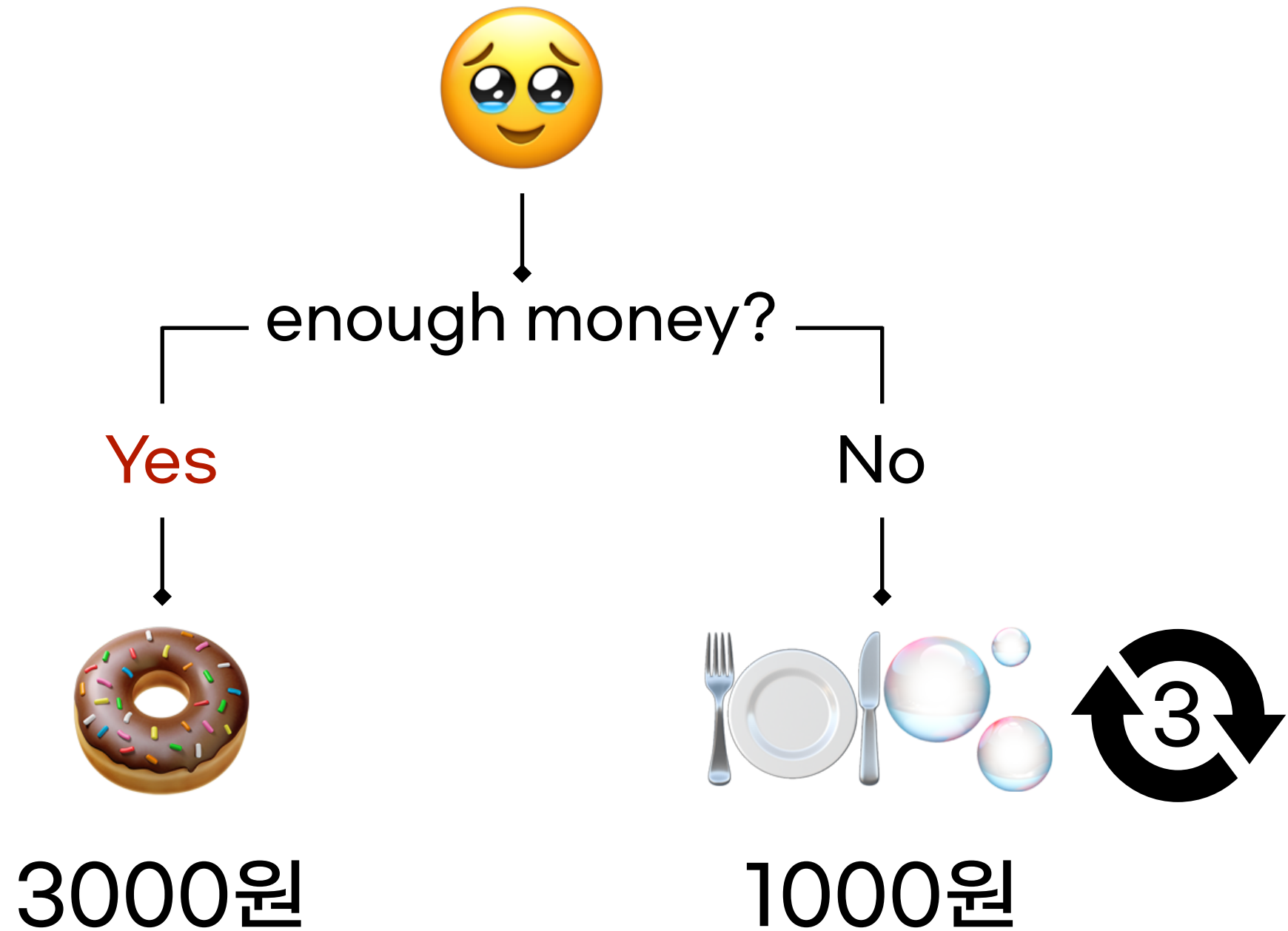
# Control Flow



# Control Flow



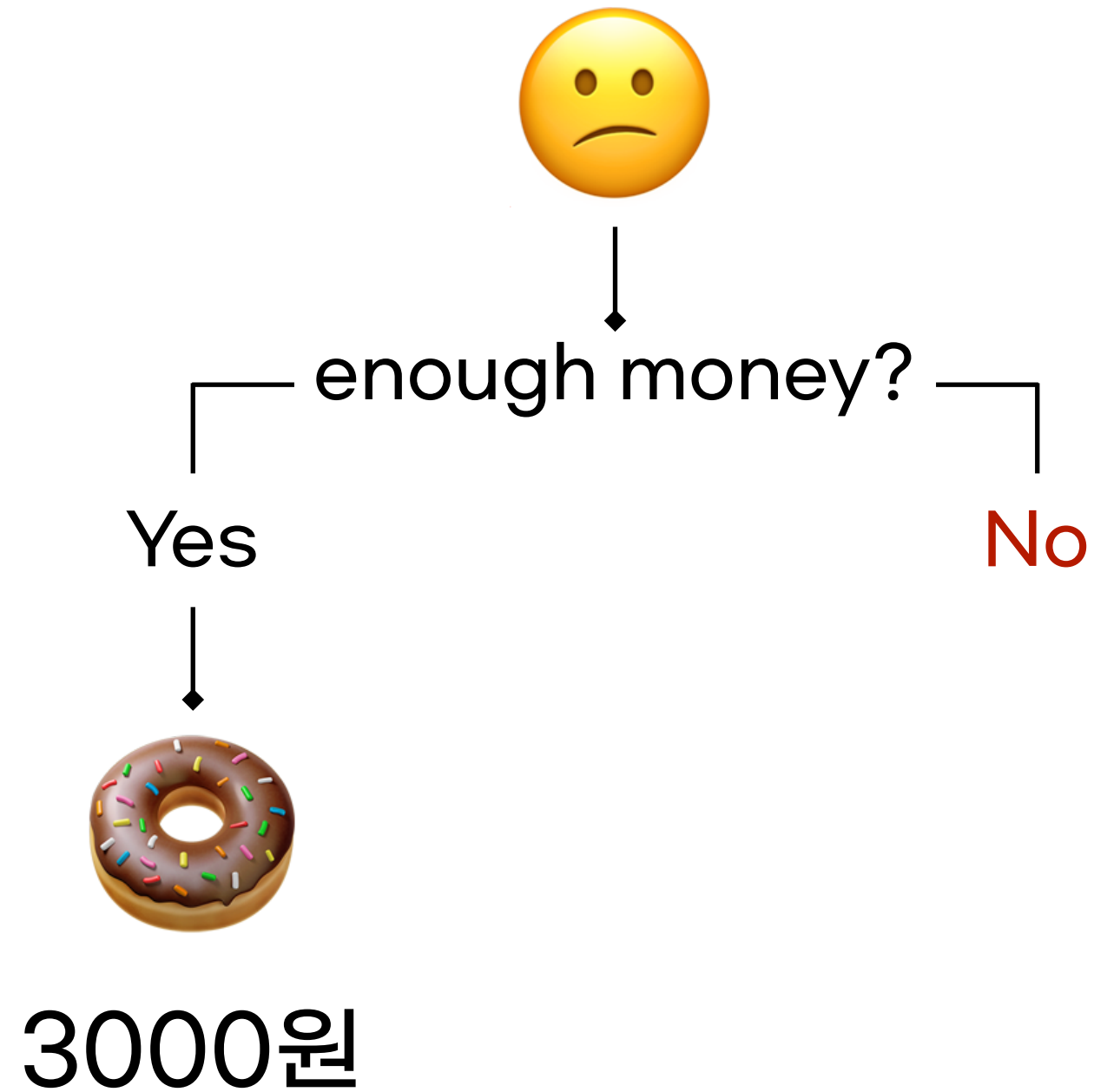
# Control Flow



# Control Flow

if      for      while

# If Statement





# If Statement



# If Statement

if condition:

code to run

# If Statement

if condition:

└ code to run

# If Statement

if condition 1:

code to run 1

else:

code to run 2

# If Statement

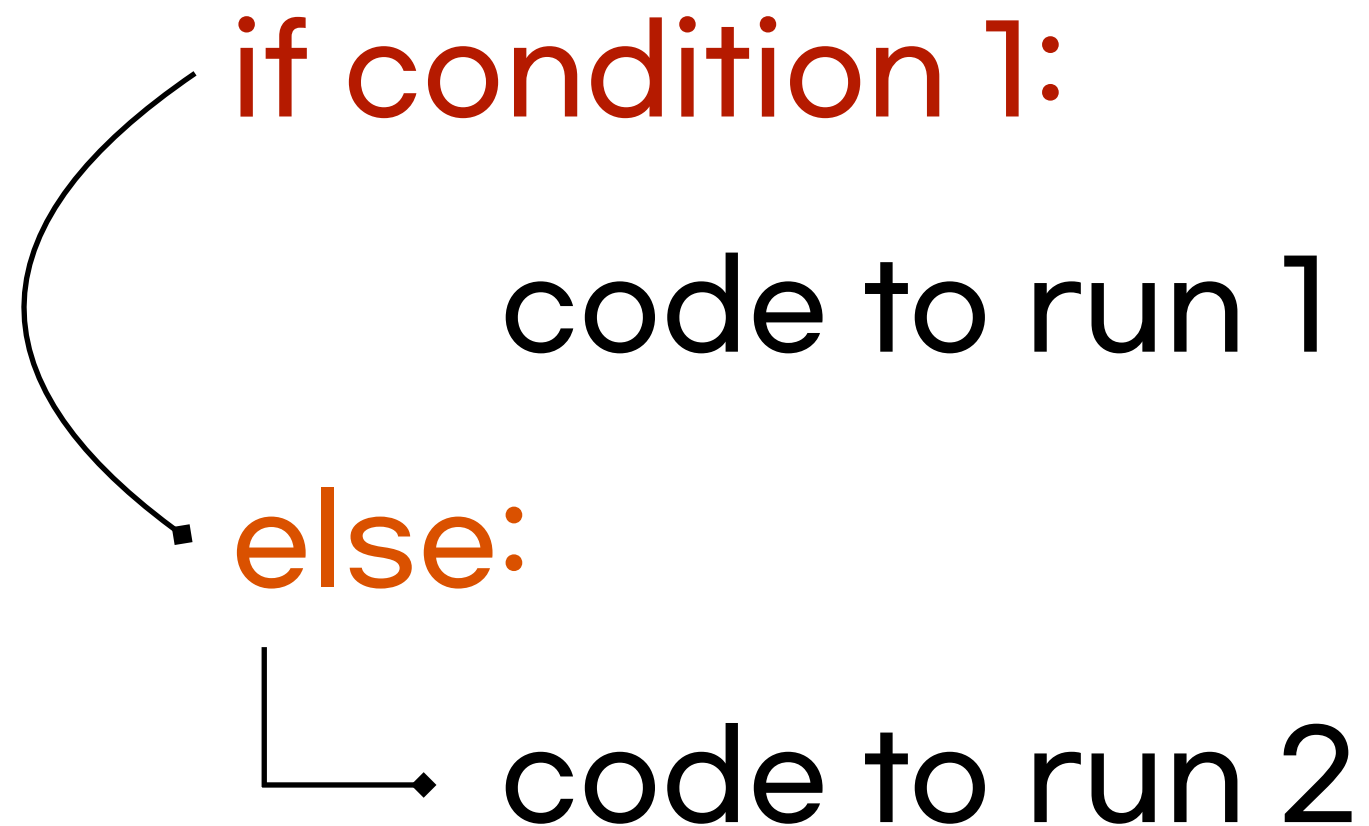
if condition 1:

└→ code to run 1

else:

code to run 2

# If Statement



# If Statement

if condition 1:

└→ code to run 1

elif condition 2:

code to run 2

else:

code to run 3

# If Statement

if condition 1:

code to run 1

elif condition 2:

└→ code to run 2

else:

code to run 3



# If Statement

if condition 1:

code to run 1

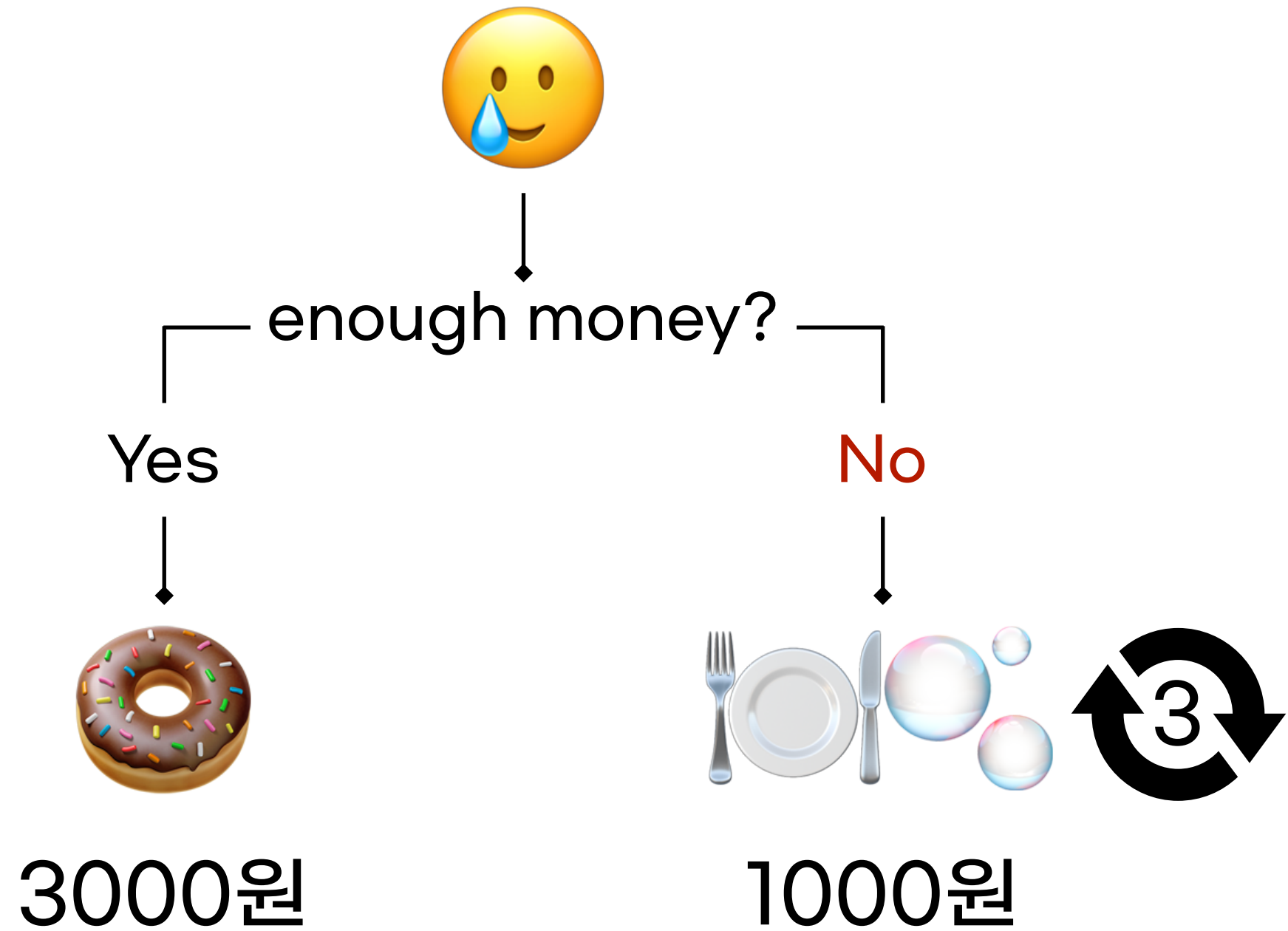
elif condition 2:

code to run 2

else:

└─ code to run 3

# For Loop



# For Loop

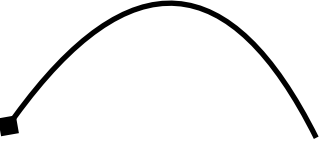
for element in sequence:

code to run

# For Loop

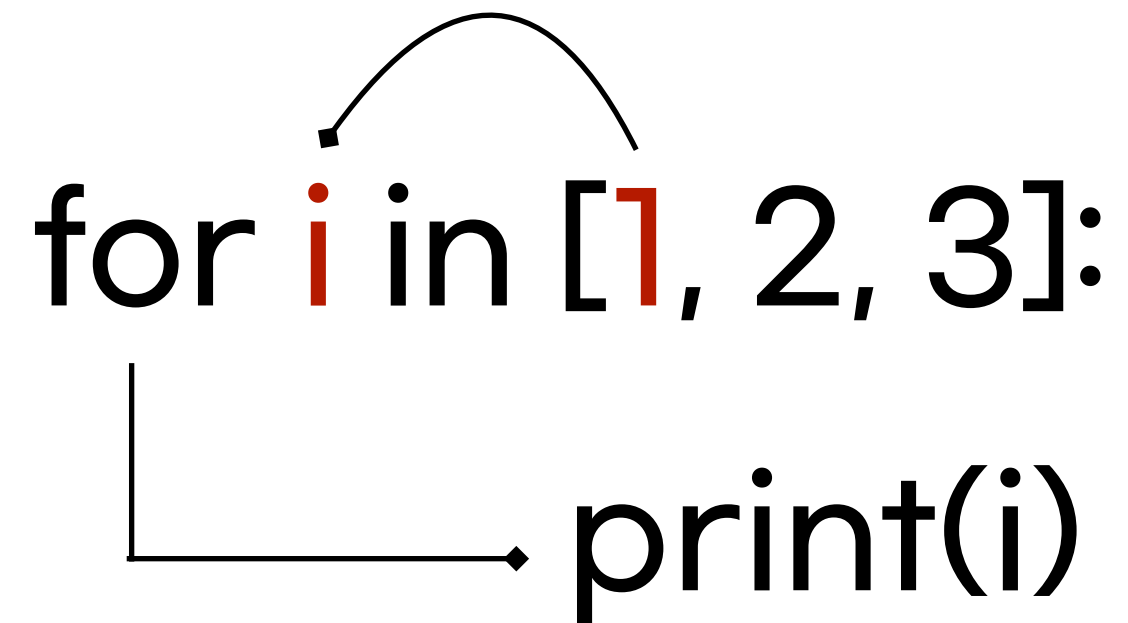
```
for i in [1, 2, 3]:  
    print(i)
```

# For Loop



```
for i in [1, 2, 3]:  
    print(i)
```

# For Loop



```
for i in [1, 2, 3]:  
    print(i)
```

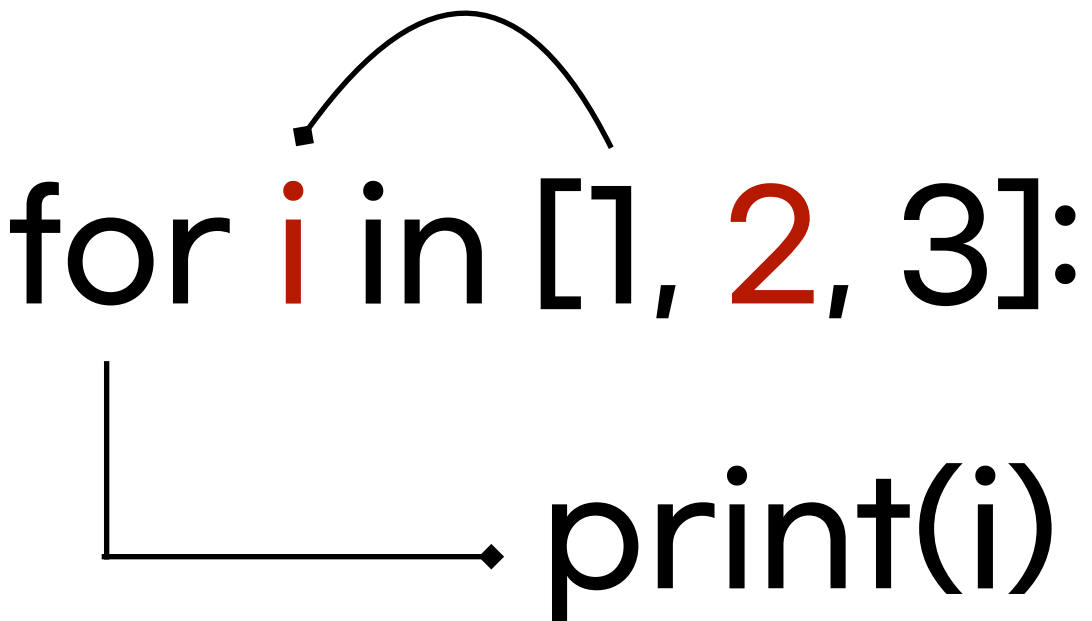
The diagram illustrates a for loop. The variable `i` is highlighted in red. A curved arrow points from the first element of the list, `1`, to the variable `i`. A straight arrow points from the colon at the end of the loop header to the `print(i)` statement, indicating the flow of execution.

# For Loop



```
for i in [1, 2, 3]:  
    print(i)
```

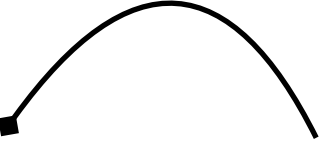
# For Loop



```
for i in [1, 2, 3]:  
    print(i)
```



# For Loop



```
for i in [1, 2, 3]:  
    print(i)
```

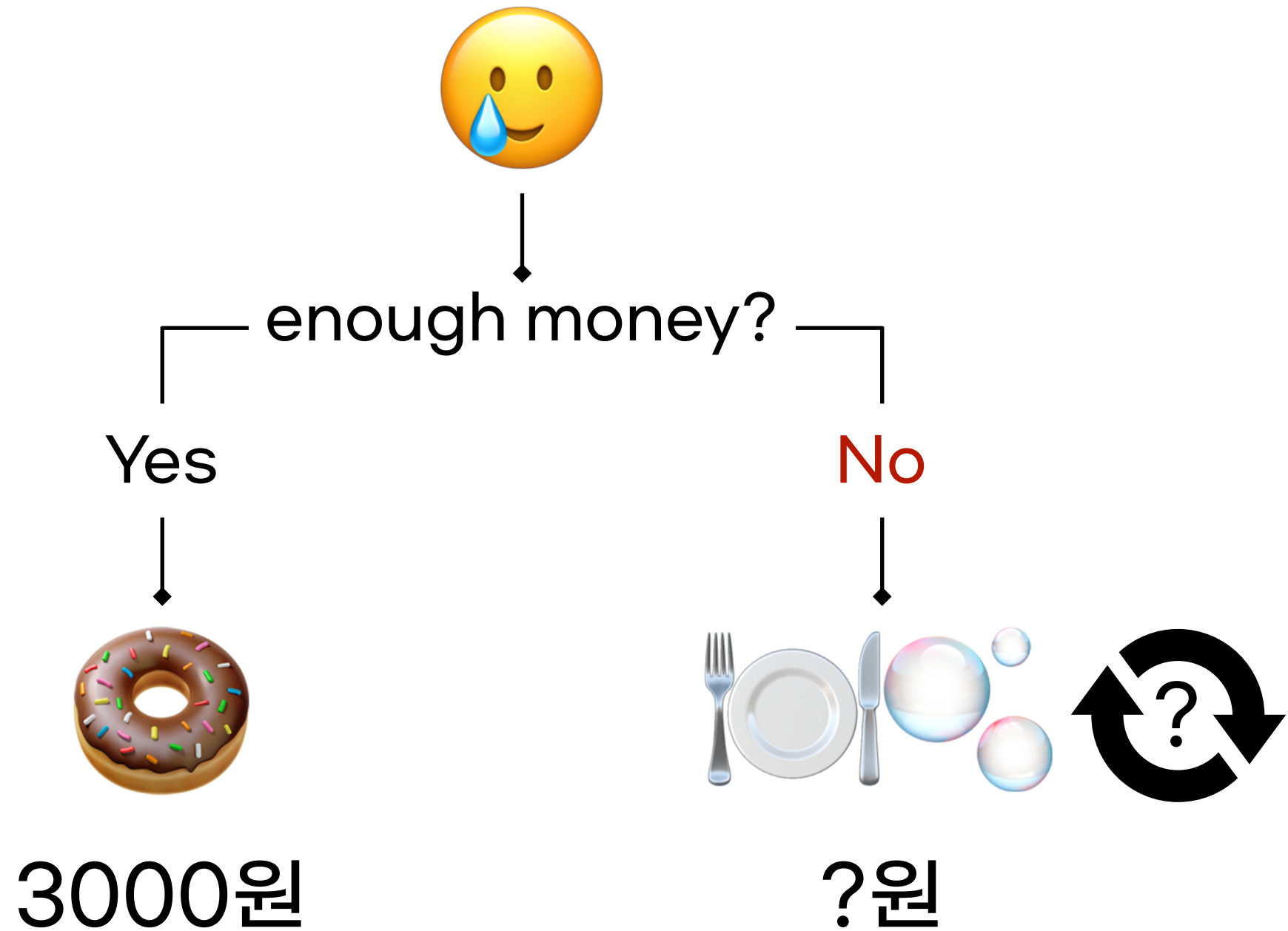
# For Loop



```
for i in [1, 2, 3]:  
    print(i)
```

The diagram illustrates a for loop. A curved arrow above the list [1, 2, 3] indicates the iteration process. A straight arrow points from the variable `i` to the `print(i)` statement, showing its use within the loop body.

# While Loop



# While Loop

**while condition:**

code to run

# While Loop

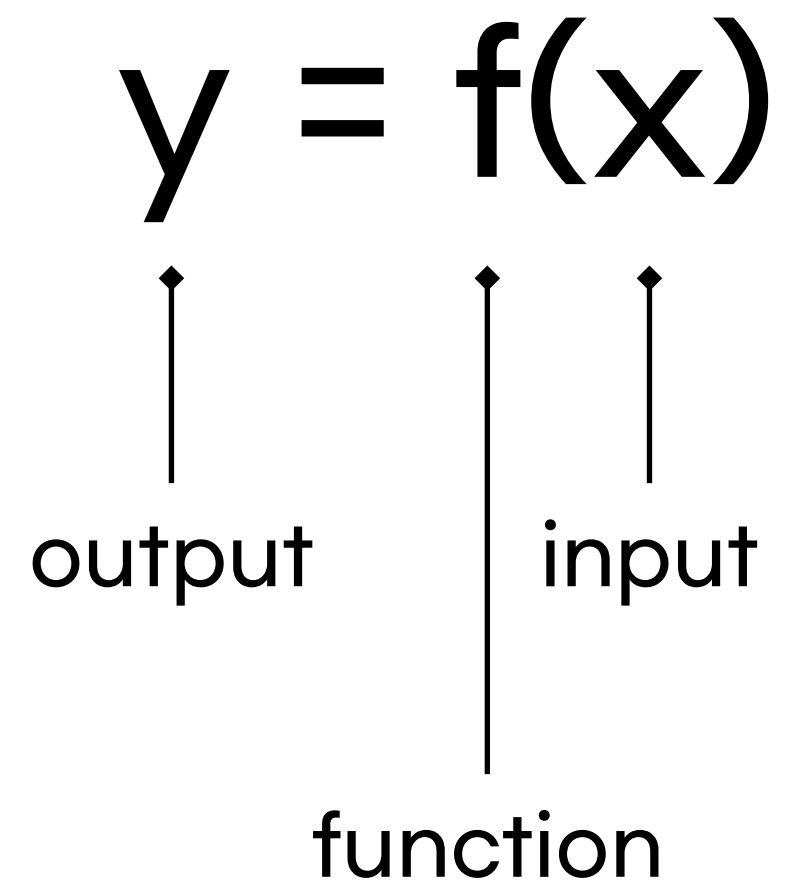
while condition:

└→ code to run

# Function

$$y = f(x)$$

# Function



# Function

```
def function_name(input):  
    code_to_run  
    return output
```



# Function

```
def function_name():  
    code_to_run  
    return output
```

# Function

```
def function_name(input):  
    code_to_run
```

# Function

```
def function_name():  
    code_to_run
```

# Local Scope

```
def function_name(input):  
    code_to_run  
    return output
```

# Global Scope

```
a = "global variable"
```

```
def function_name(input):  
    code_to_run  
    return output
```