python

For - While
Break - Continue
List Comprehension
Errors
Exceptions Handling

Seaborn Kusto

Official Team

Malka
Rusyd
Abdussalam

Trianto
Haryo
Nugroho

Maulana
Ishaq
Siregar

Fauzi
Wardah
Ali

Rizki
Afrinal

For, While, Break, Continue

For

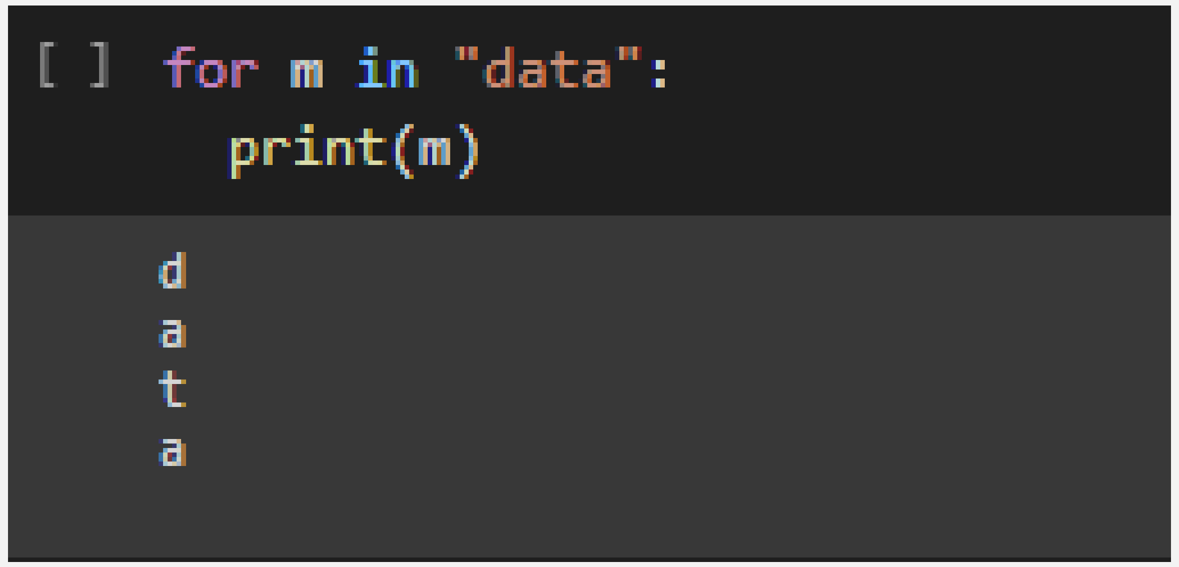
A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

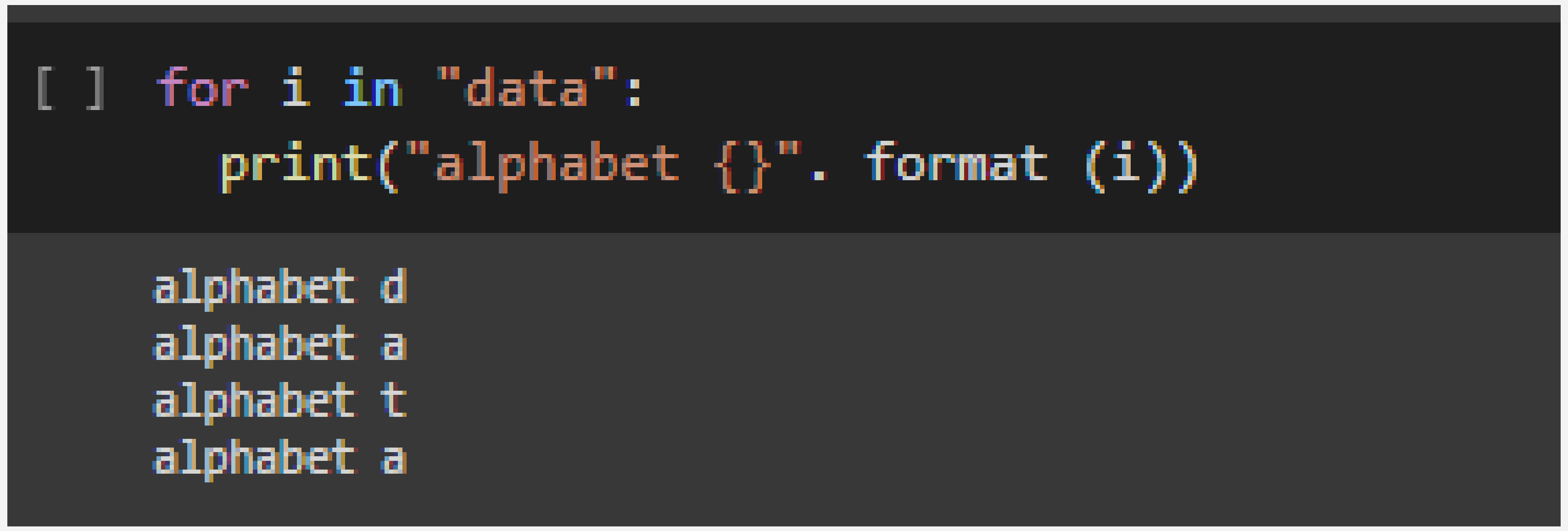
With the for loop we can execute a set of statements, once for each item in a list, string, range, etc.

String

Even strings are iterable objects, they contain a sequence of characters:



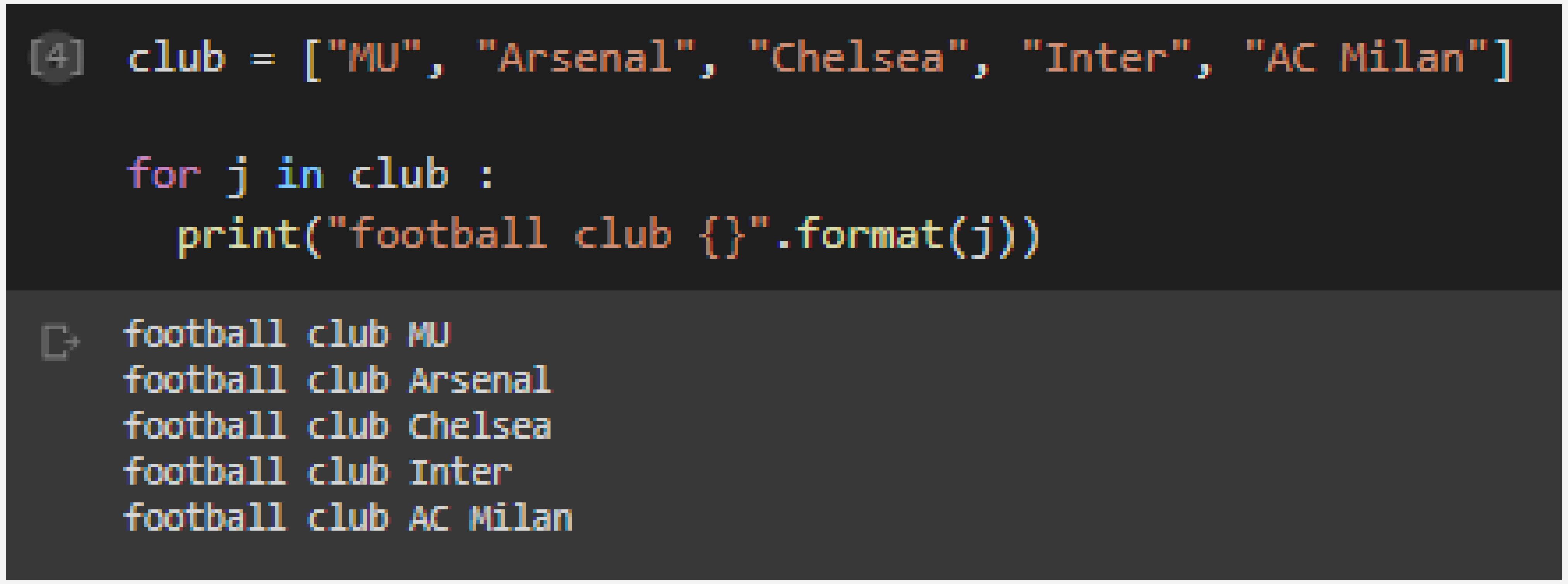
```
[ ]  for m in "data":
       print(m)

     d
     a
     t
     a
```



```
[ ]  for i in "data":
       print("alphabet {}". format (i))

     alphabet d
     alphabet a
     alphabet t
     alphabet a
```

List

Even List are iterable objects, they contain a sequence of characters:

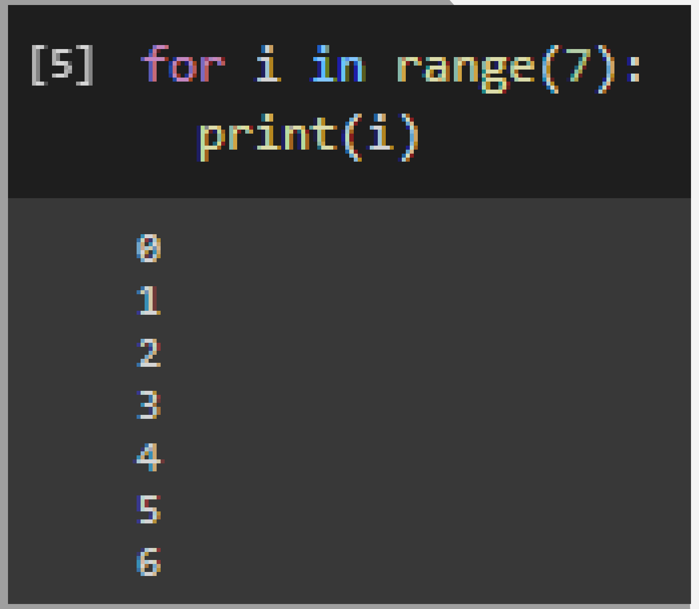


```
club = ["MU", "Arsenal", "Chelsea", "Inter", "AC Milan"]

for j in club :
  print("football club {}".format(j))
```

```
football club MU
football club Arsenal
football club Chelsea
football club Inter
football club AC Milan
```

Range()

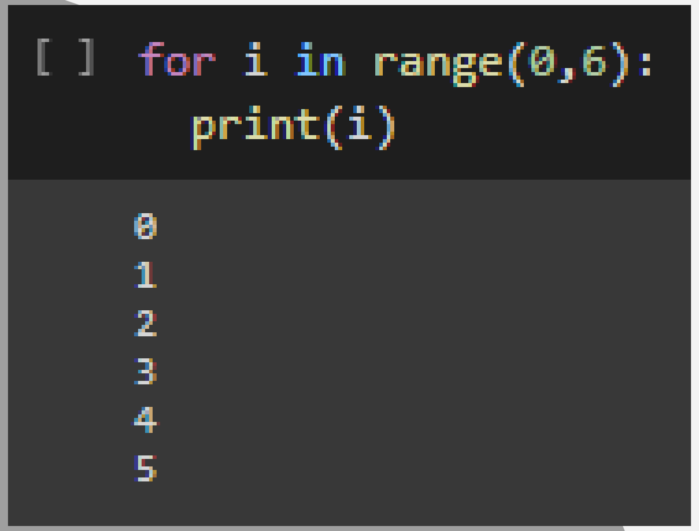
To loop through a set of code a specified number of times, we can use the range() function,
The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.



```
[5]  for i in range(7):
       print(i)

     0
     1
     2
     3
     4
     5
     6
```

Range()

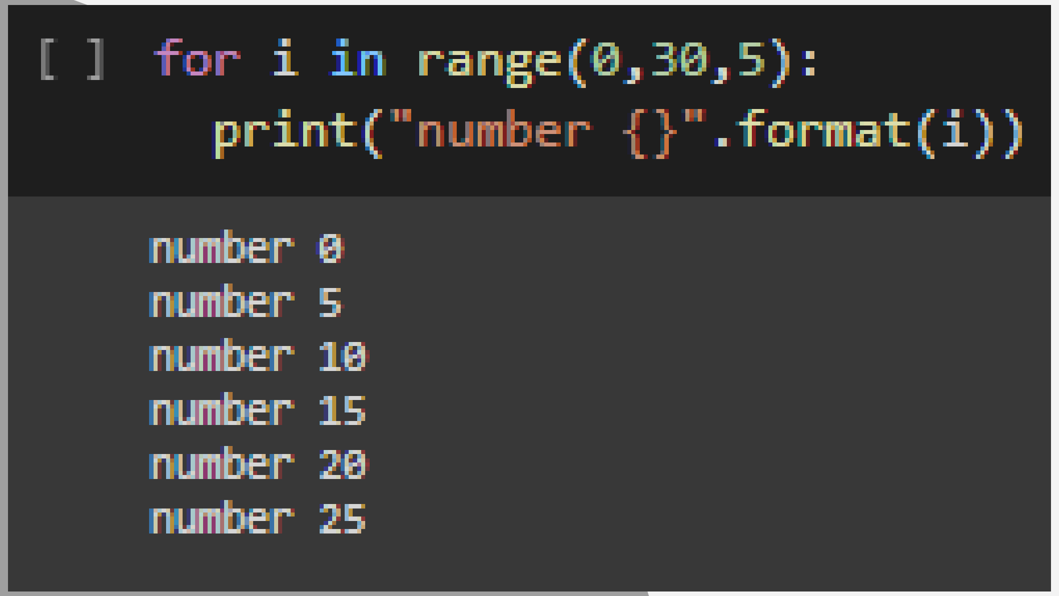
The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(0, 6), which means values from 0 to 6 (but not including 6):



```
[ ]  for i in range(0,6):
       print(i)

     0
     1
     2
     3
     4
     5
```

Range()

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 20, 4):

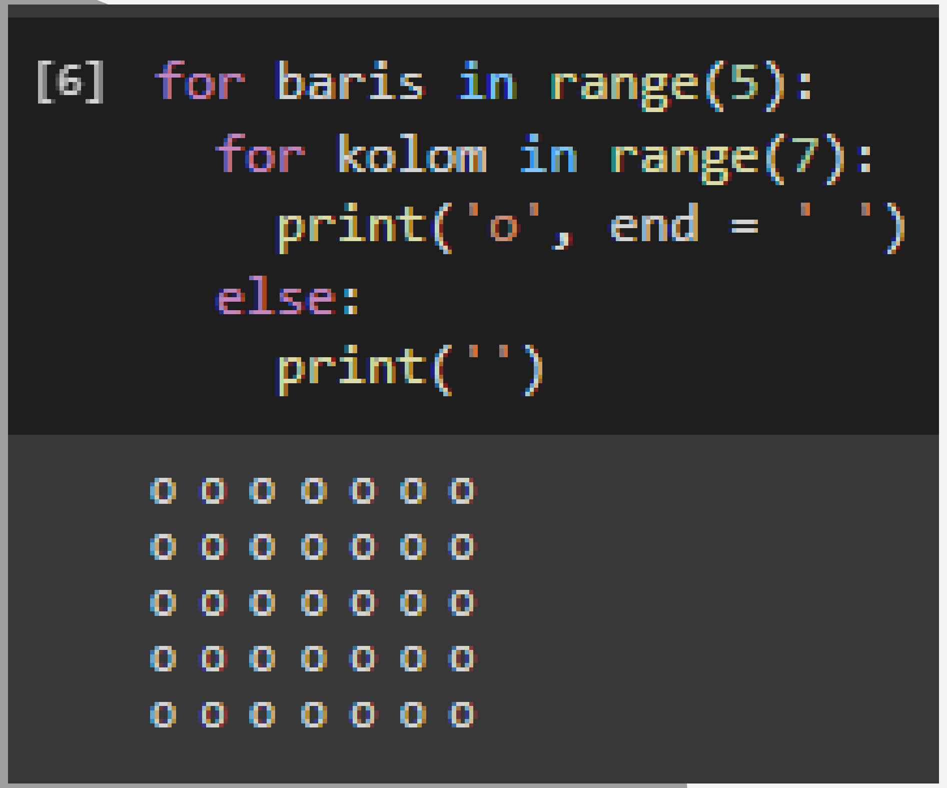


```
[ ]  for i in range(0,30,5):
       print("number {}".format(i))
```

```
number 0
number 5
number 10
number 15
number 20
number 25
```

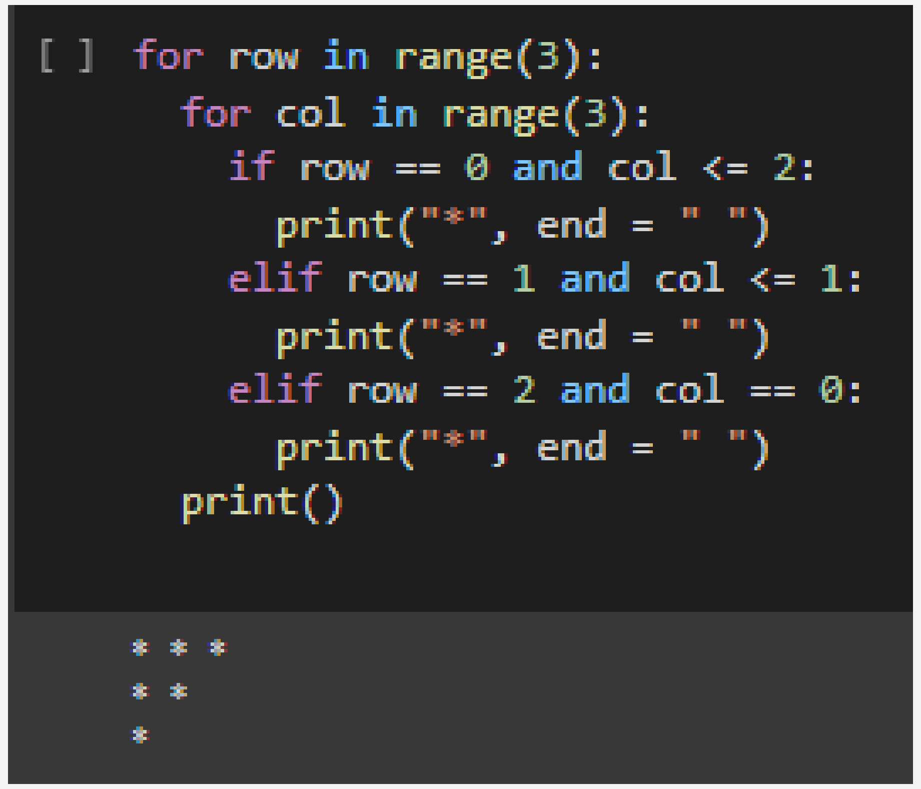
Nested Loop

A nested loop is a loop inside a loop.
The "inner loop" will be executed one time for each iteration of the "outer loop"



```
[6] for baris in range(5):
      for kolom in range(7):
        print('o', end = ' ')
      else:
        print('')
```

```
o o o o o o o
o o o o o o o
o o o o o o o
o o o o o o o
o o o o o o o
```

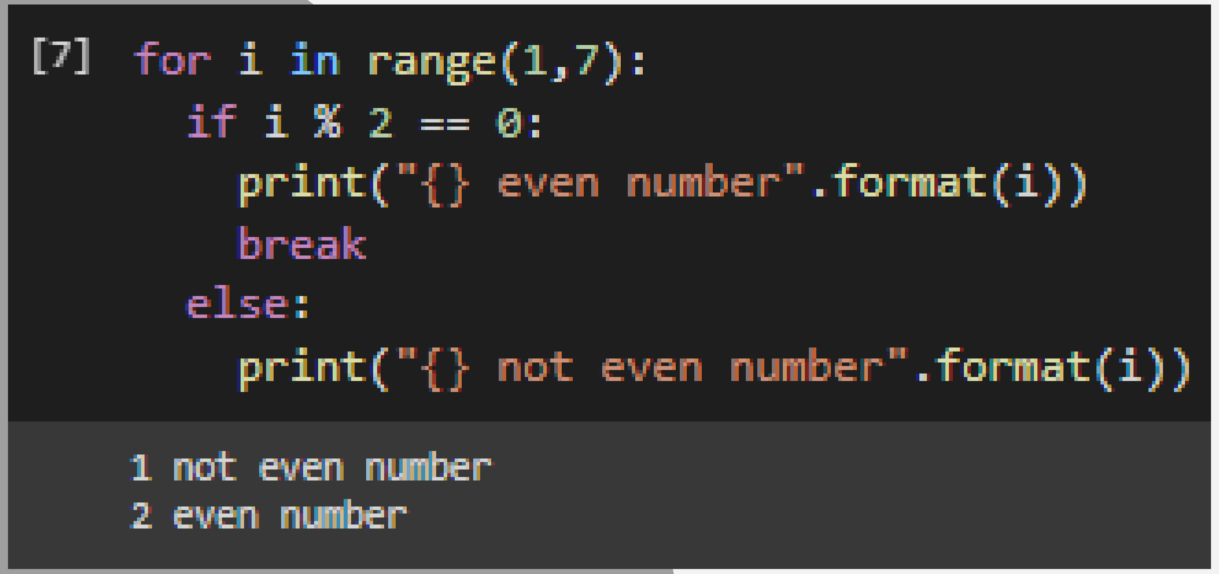


```
[ ] for row in range(3):
      for col in range(3):
        if row == 0 and col <= 2:
          print("*", end = " ")
        elif row == 1 and col <= 1:
          print("*", end = " ")
        elif row == 2 and col == 0:
          print("*", end = " ")
      print()
```

```
* * *
* *
*
```

For Loop With Else

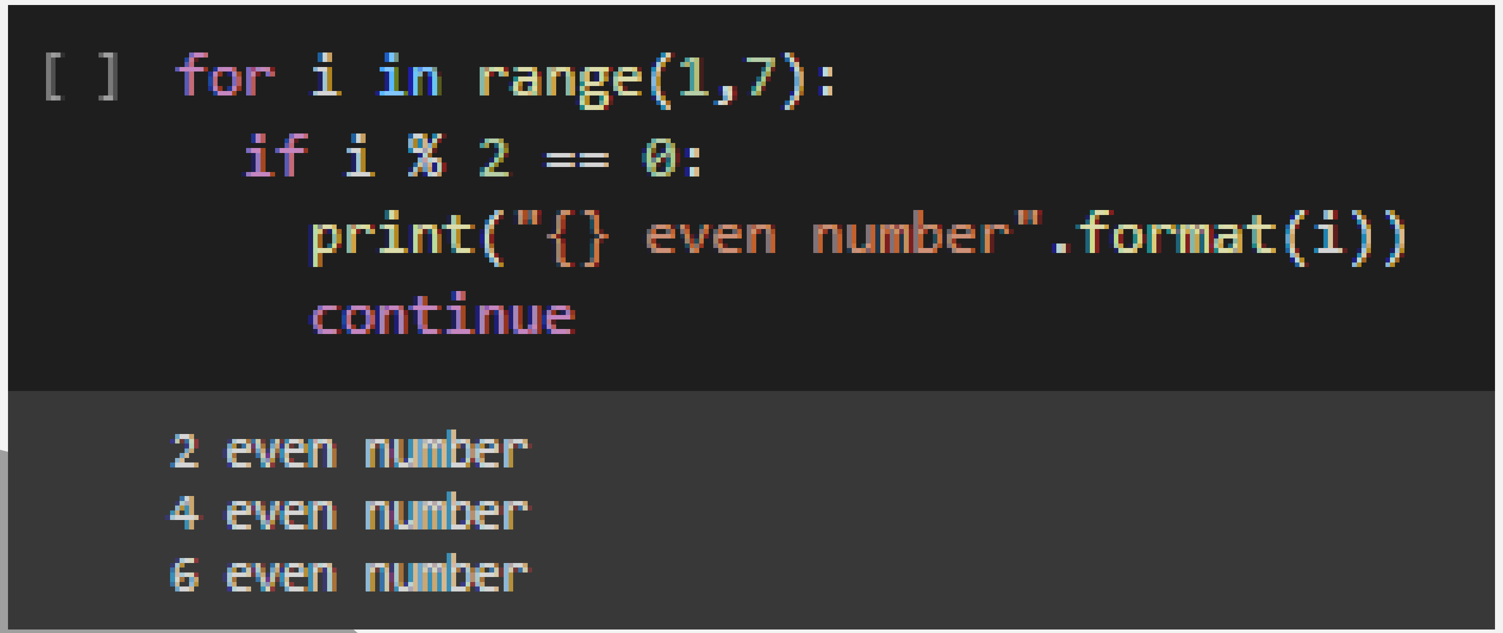
A for loop can have an optional else block as well. The else part is executed if the items in the sequence are used for loop exhausts. The break keyword can be used to stop a for a loop. In such cases, the else part is ignored.



```
[7] for i in range(1,7):
      if i % 2 == 0:
        print("{} even number".format(i))
        break
      else:
        print("{} not even number".format(i))
```

```
1 not even number
2 even number
```

For Loop Without Else



```
[ ]  for i in range(1,7):
       if i % 2 == 0:
         print("{} even number".format(i))
         continue
```

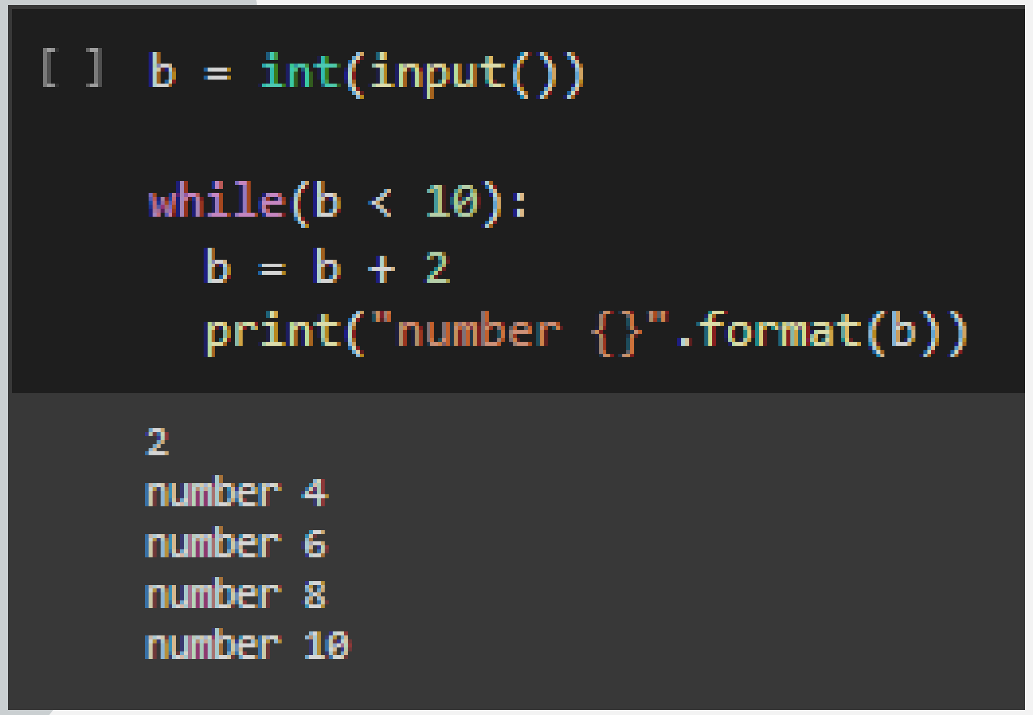
```
2 even number
4 even number
6 even number
```

While

A while loop statement in Python programming language **repeatedly executes a target statement as long as a given condition is true**. The syntax of a while loop in Python programming language is − Here, statement (s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value.

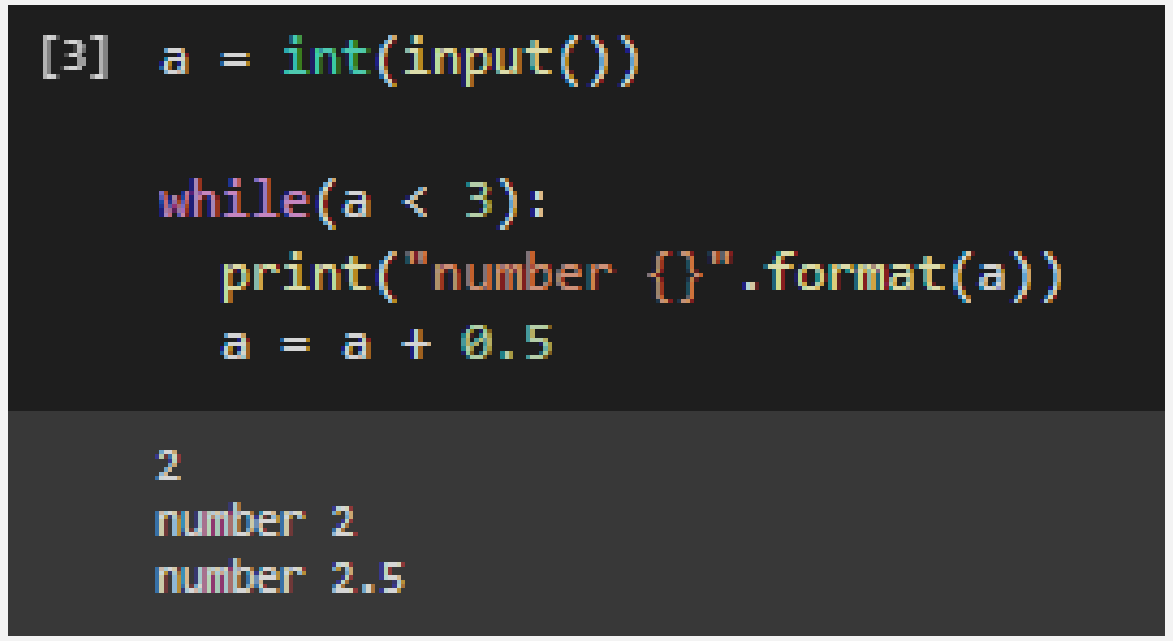
While

With the while loop we can execute a set of statements as long as a condition is true.



```
[ ]  b = int(input())

     while(b < 10):
       b = b + 2
       print("number {}".format(b))
```

```
2
number 4
number 6
number 8
number 10
```

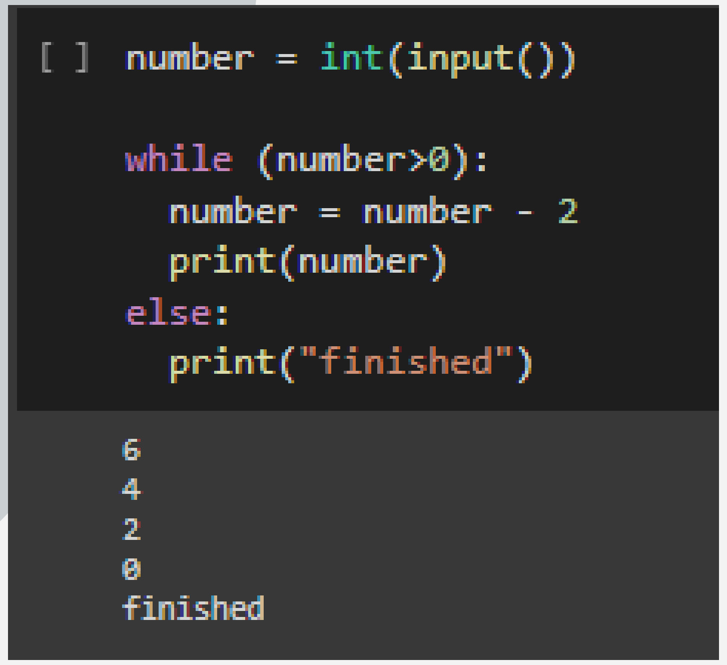


```
[3]  a = int(input())

     while(a < 3):
       print("number {}".format(a))
       a = a + 0.5
```

```
2
number 2
number 2.5
```

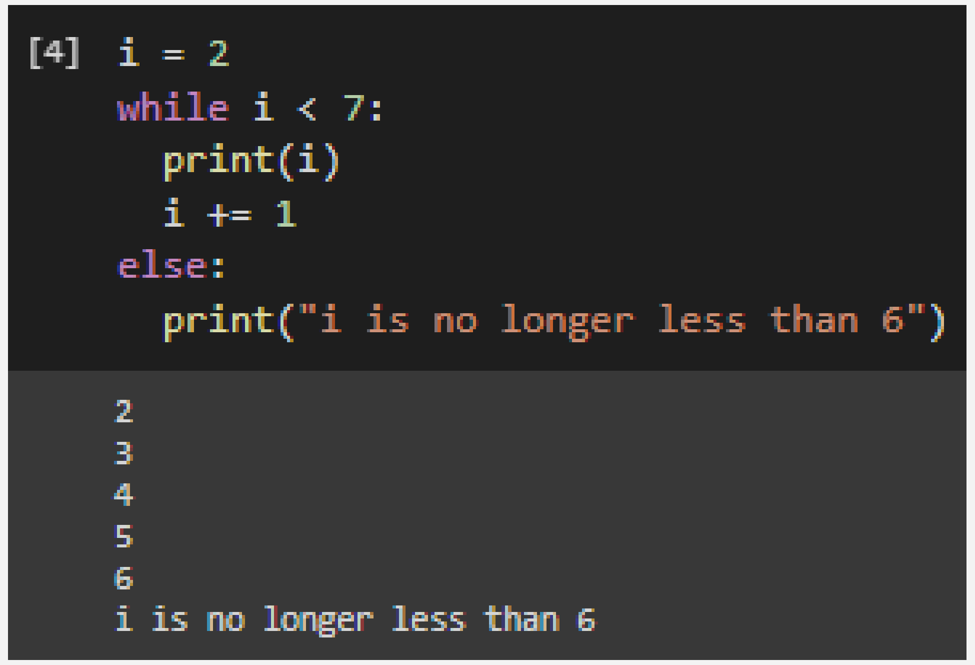
While Loop With Else

The else part is executed if the condition in the while loop evaluates to False. The while loop can be terminated with a break statement. In such cases, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.



```
[ ]  number = int(input())

     while (number>0):
       number = number - 2
       print(number)
     else:
       print("finished")
```

```
     6
     4
     2
     0
     finished
```



```
[4]  i = 2
     while i < 7:
       print(i)
       i += 1
     else:
       print("i is no longer less than 6")
```

```
     2
     3
     4
     5
     6
     i is no longer less than 6
```

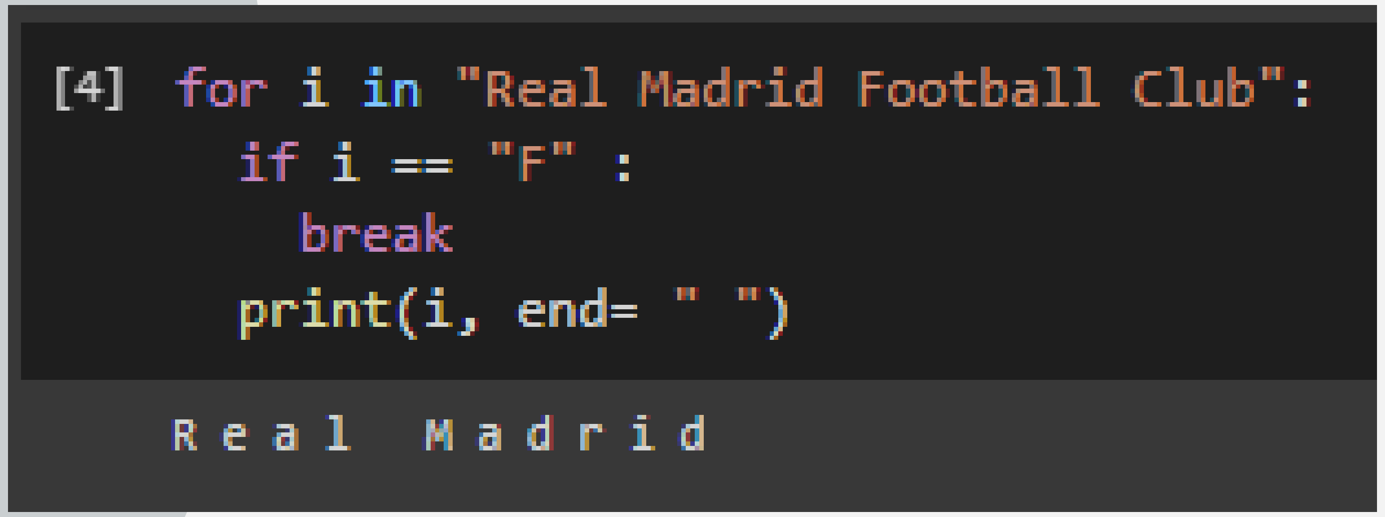
Break

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

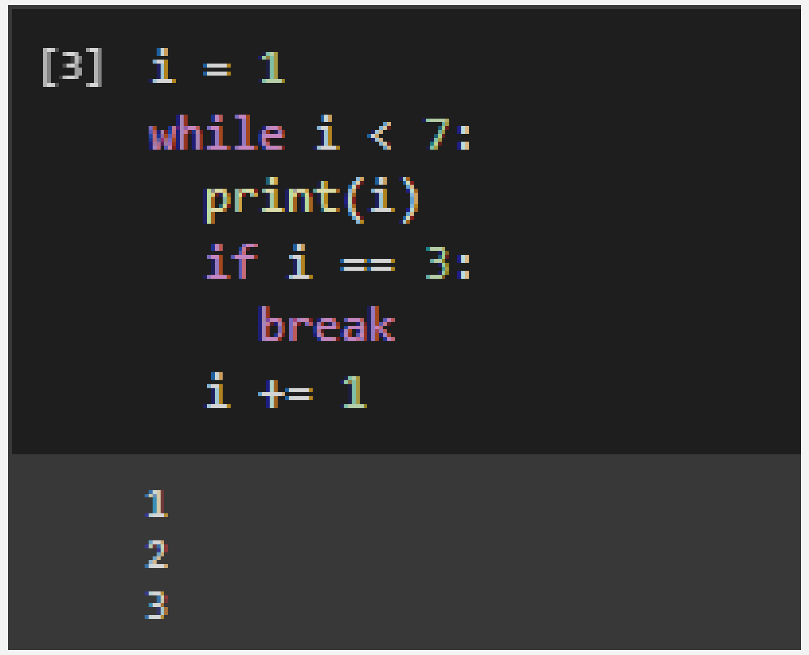
If the break statement is inside a nested loop (loop inside another loop), the break statement will terminate the innermost loop.

Break

With the break statement we can stop the loop even if the while condition is true



```
[4] for i in "Real Madrid Football Club":
      if i == "F" :
        break
      print(i, end= " ")

    R e a l   M a d r i d
```



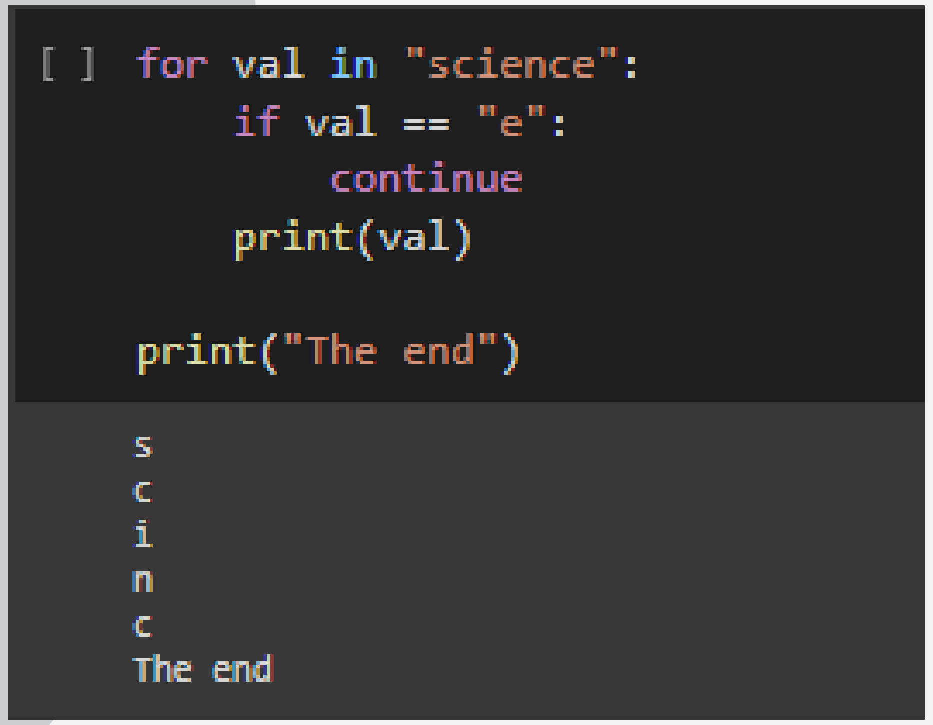
```
[3] i = 1
    while i < 7:
      print(i)
      if i == 3:
        break
      i += 1

    1
    2
    3
```

Continue

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

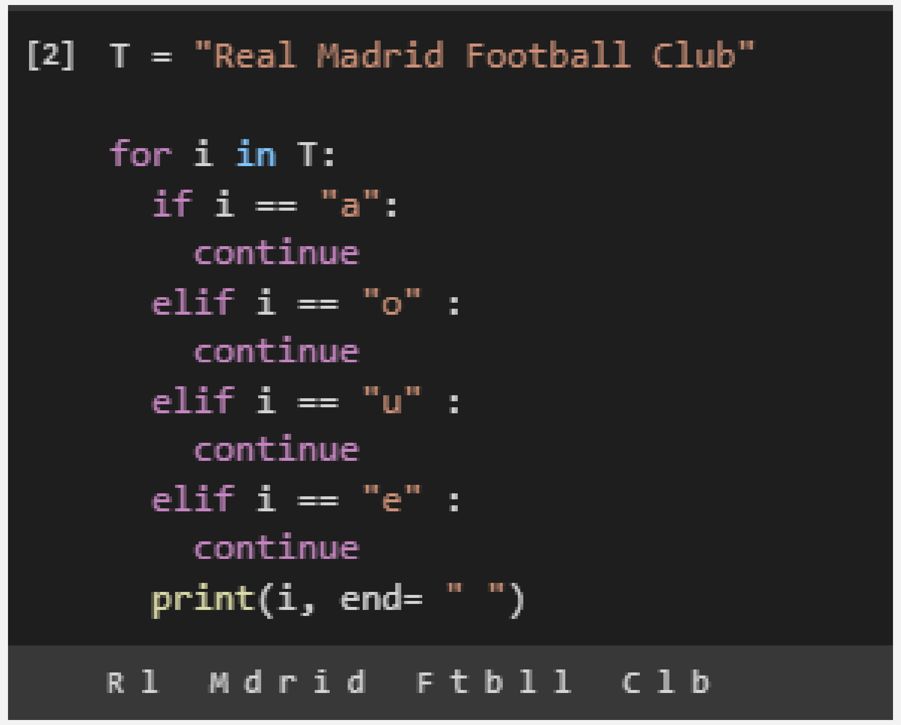
Continue

With the continue statement we can stop the current iteration, and continue with the next



```
[ ] for val in "science":
        if val == "e":
            continue
        print(val)

    print("The end")
```

```
s
c
i
n
c
The end
```

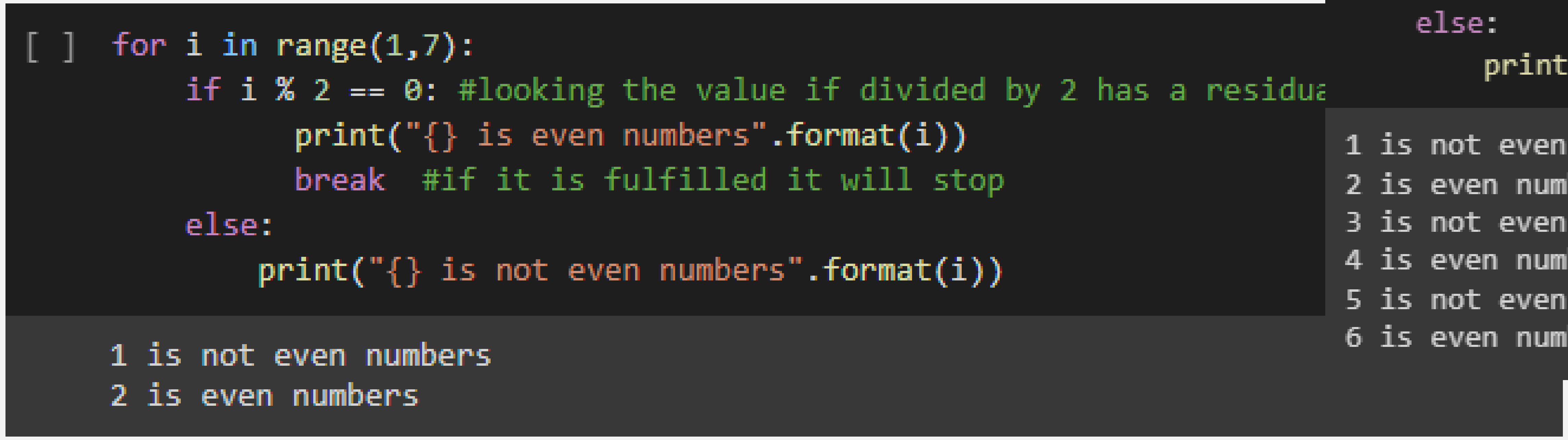


```
[2] T = "Real Madrid Football Club"

    for i in T:
      if i == "a":
        continue
      elif i == "o" :
        continue
      elif i == "u" :
        continue
      elif i == "e" :
        continue
      print(i, end= " ")
```

```
R l  M d r i d  F t b l l  C l b
```

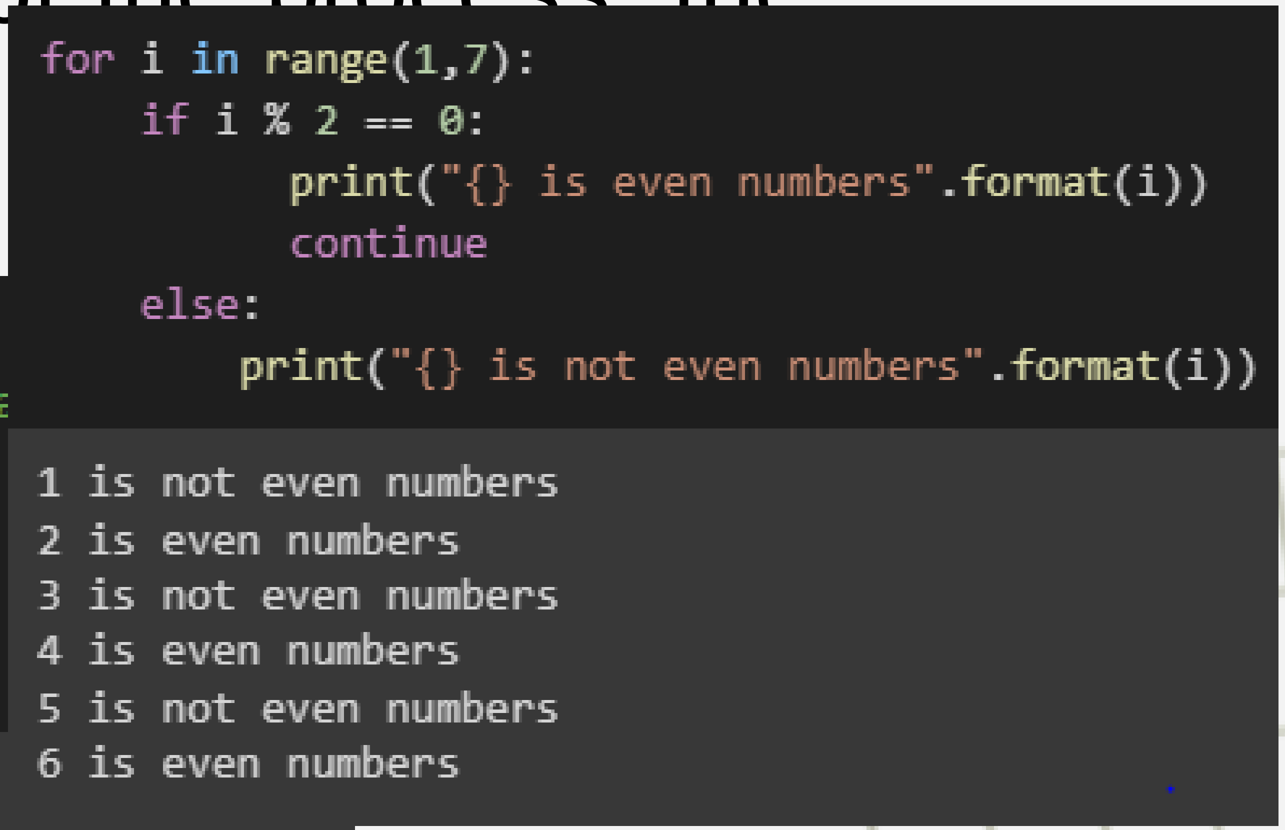
Else After For

For is used to loop or interact up to the range limit that we have specified and for is also commonly used to loop through code that has known many iterations. else function to print the final result of the loop that we created, so if the result of the loop that we created has reached the end of the process, the else command will be executed.



```
[ ]  for i in range(1,7):
         if i % 2 == 0: #looking the value if divided by 2 has a residua
               print("{} is even numbers".format(i))
               break  #if it is fulfilled it will stop
         else:
             print("{} is not even numbers".format(i))
```

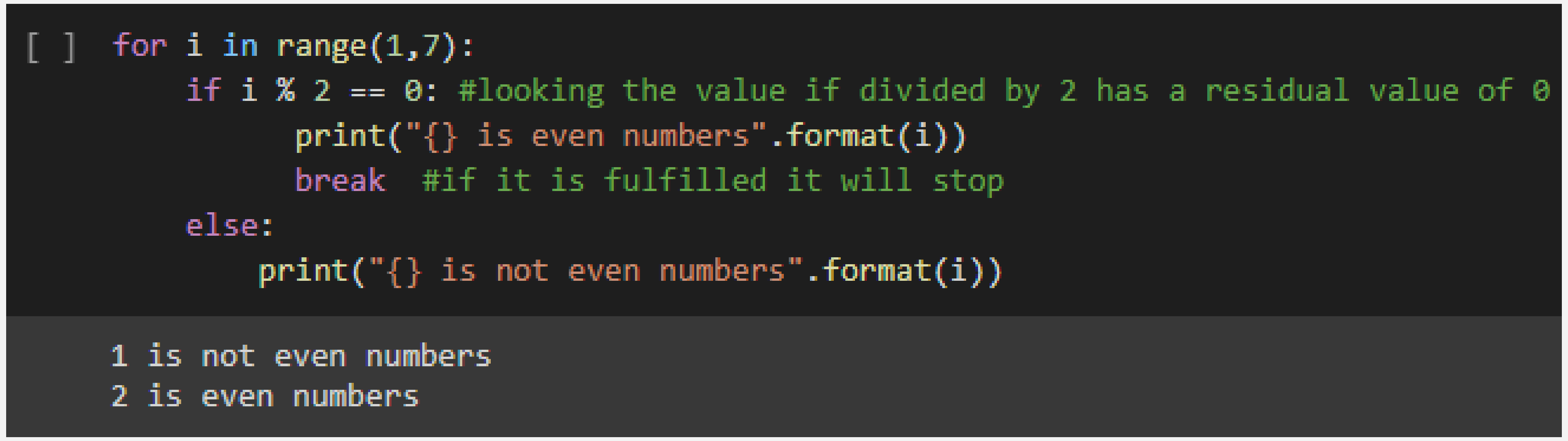
```
1 is not even numbers
2 is even numbers
```



```
for i in range(1,7):
    if i % 2 == 0:
          print("{} is even numbers".format(i))
          continue
    else:
        print("{} is not even numbers".format(i))
```

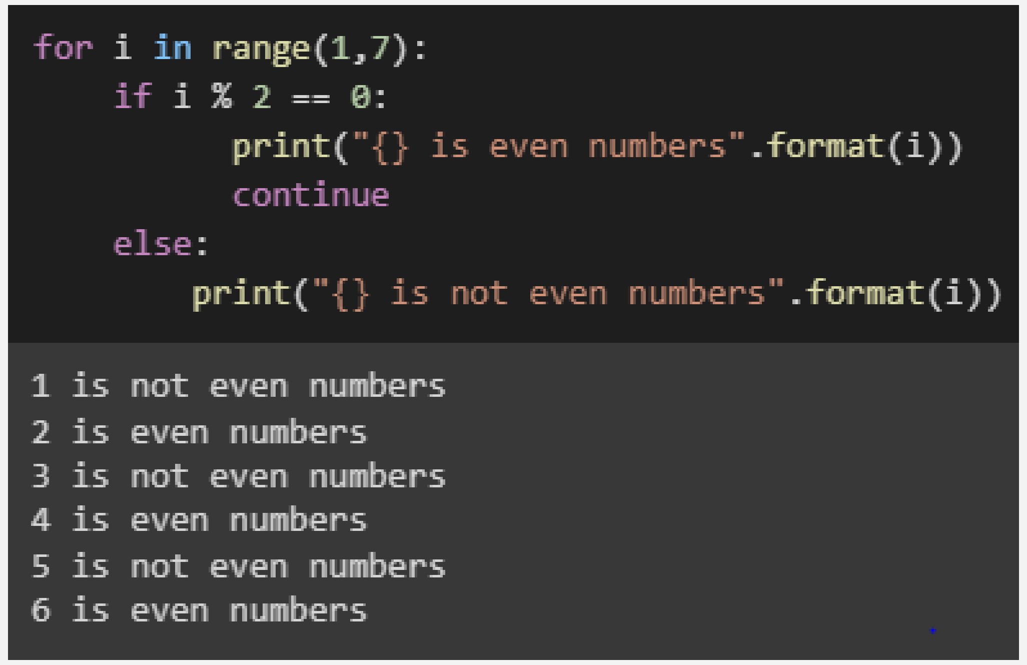
```
1 is not even numbers
2 is even numbers
3 is not even numbers
4 is even numbers
5 is not even numbers
6 is even numbers
```

Else After For



```
[ ]  for i in range(1,7):
         if i % 2 == 0: #looking the value if divided by 2 has a residual value of 0
               print("{} is even numbers".format(i))
               break  #if it is fulfilled it will stop
         else:
             print("{} is not even numbers".format(i))
```

```
1 is not even numbers
2 is even numbers
```

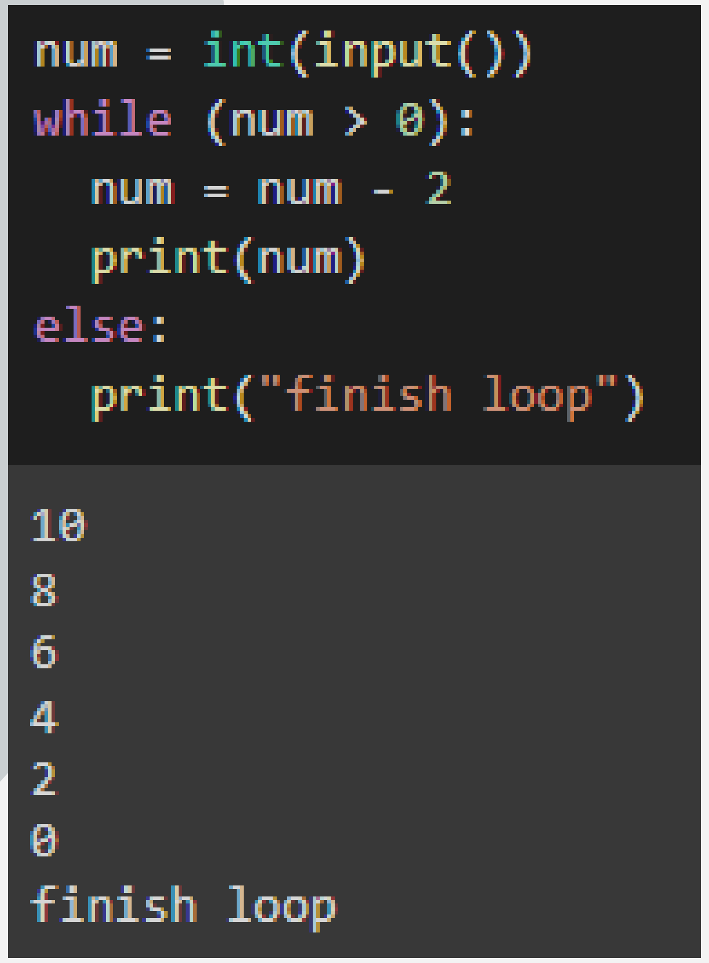


```
for i in range(1,7):
    if i % 2 == 0:
          print("{} is even numbers".format(i))
          continue
    else:
        print("{} is not even numbers".format(i))
```

```
1 is not even numbers
2 is even numbers
3 is not even numbers
4 is even numbers
5 is not even numbers
6 is even numbers
```

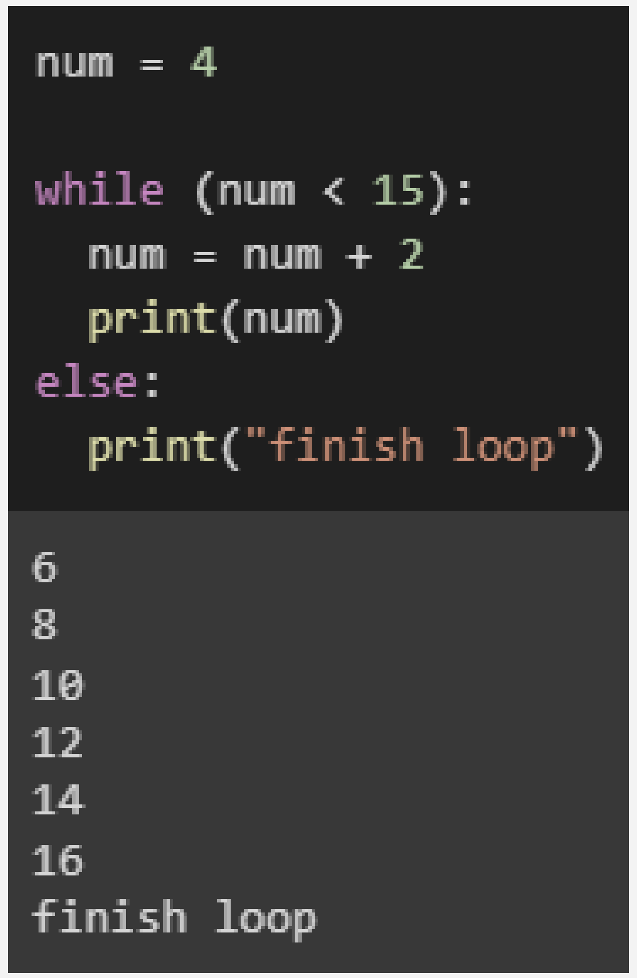
Else After While

to define a task to be executed when the loop has finished naturally without being forced to stop.



```
num = int(input())
while (num > 0):
  num = num - 2
  print(num)
else:
  print("finish loop")
```

```
10
8
6
4
2
0
finish loop
```



```
num = 4

while (num < 15):
  num = num + 2
  print(num)
else:
  print("finish loop")
```

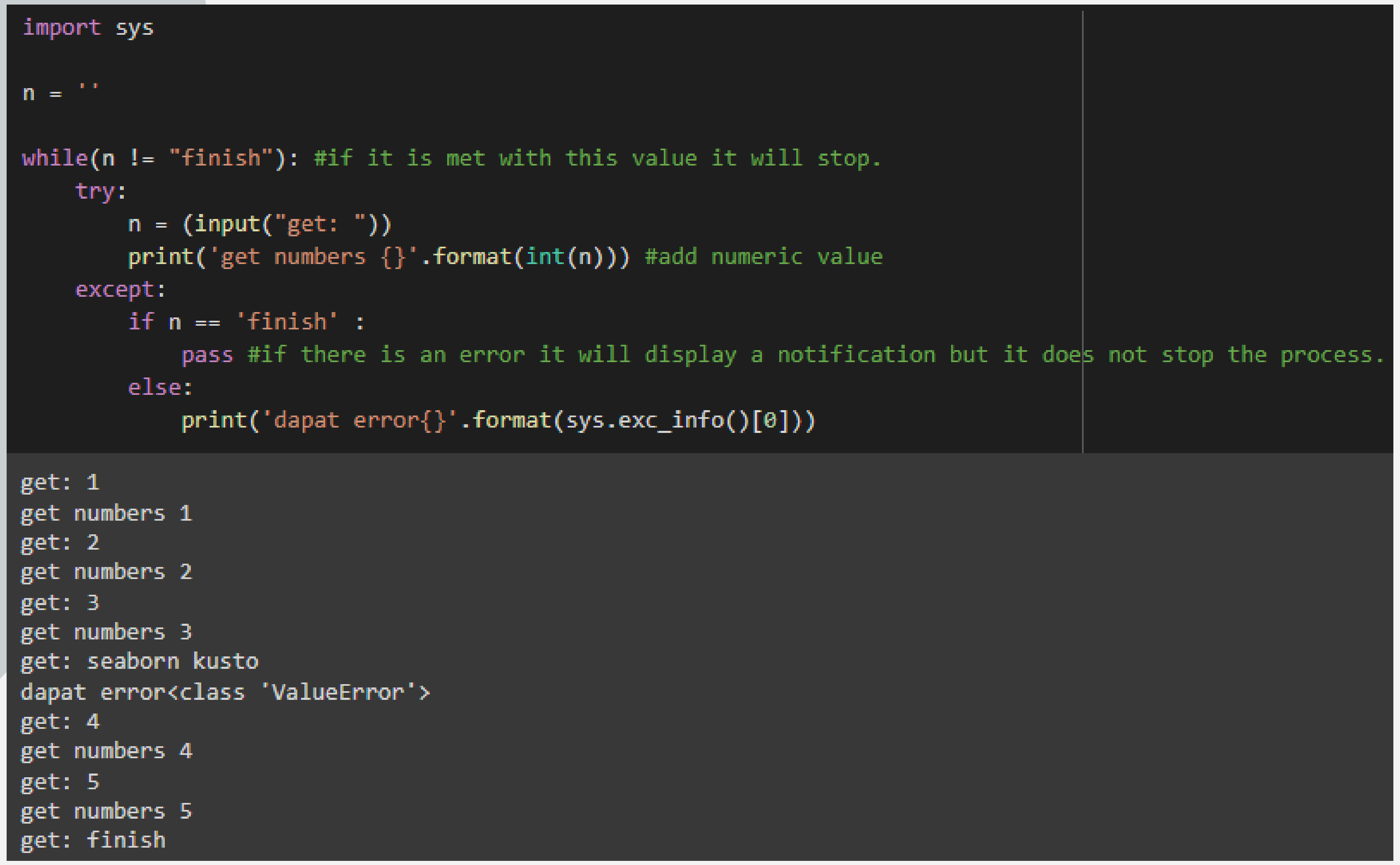
```
6
8
10
12
14
16
finish loop
```

Pass

Pass is an instruction to python that no code is executed, so python continues executing the program below it. Lots of code blocks in python that we can't just leave empty. If we leave it blank the python interpreter will give an error. To work around this, we can put the pass statement as the body of the code block. That way the code block still doesn't do anything, but it doesn't cause an error either.

Pass

Pass function to continue the process even though there is an error.



```
import sys

n = ''

while(n != "finish"): #if it is met with this value it will stop.
    try:
        n = (input("get: "))
        print('get numbers {}'.format(int(n))) #add numeric value
    except:
        if n == 'finish' :
            pass #if there is an error it will display a notification but it does not stop the process.
        else:
            print('dapat error{}'.format(sys.exc_info()[0]))
```

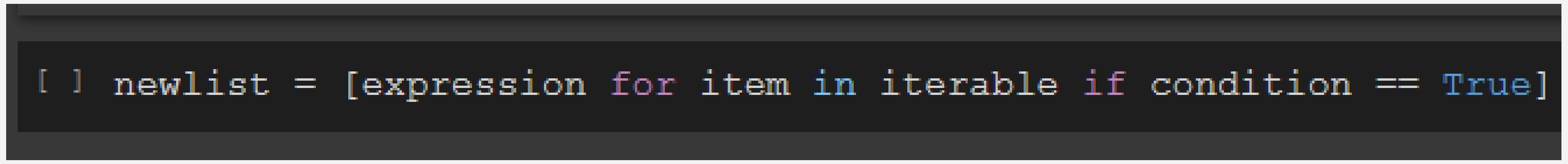
```
get: 1
get numbers 1
get: 2
get numbers 2
get: 3
get numbers 3
get: seaborn kusto
dapat error<class 'ValueError'>
get: 4
get numbers 4
get: 5
get numbers 5
get: finish
```

List Comprehension

List Comprehension

List comprehension is how to create a new list in a short way based on an existing list. Without list comprehension you will have to write a *for* statement with a conditional test inside

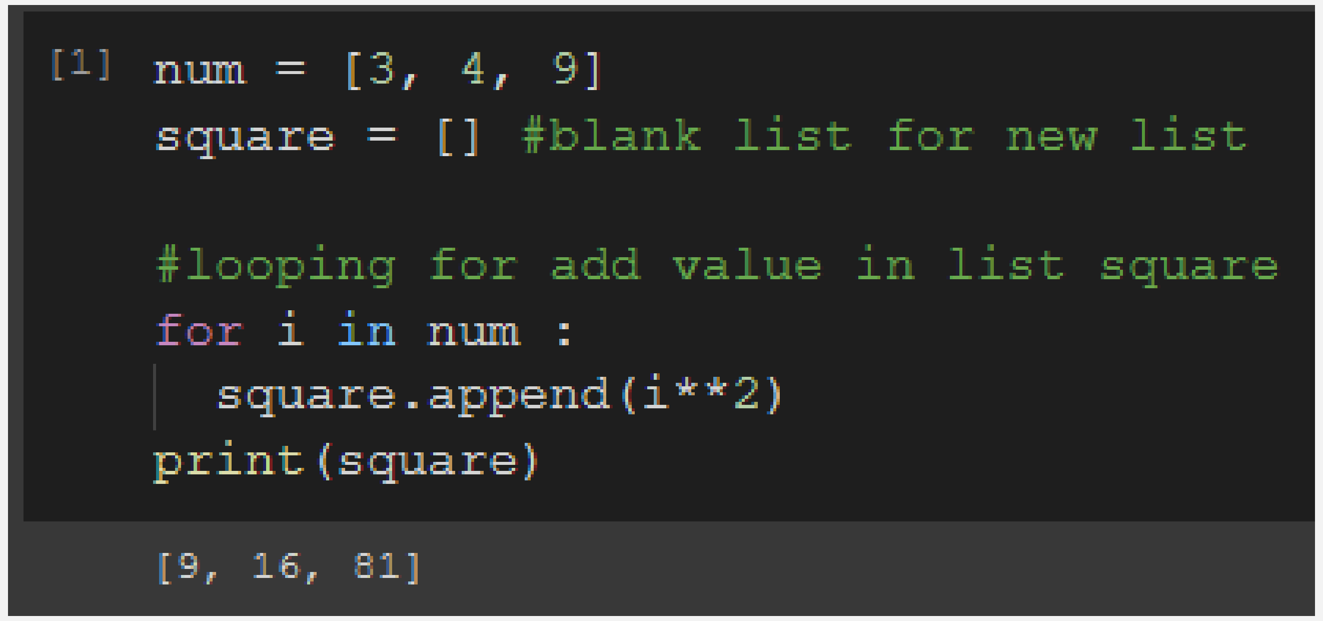
Syntax List Comprehension



```
[ ] newlist = [expression for item in iterable if condition == True]
```

Create New List Without List Comprehension

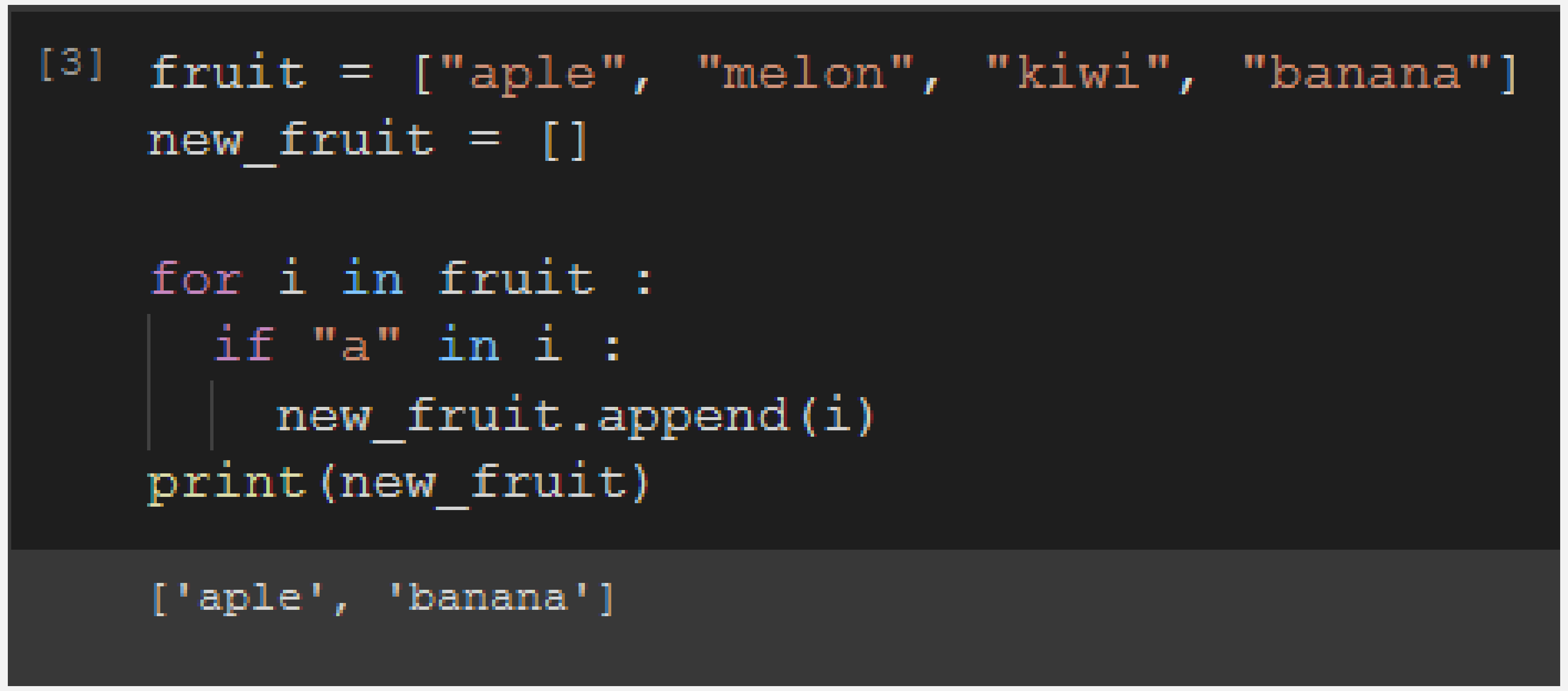
Without *if* condition



```
[1] num = [3, 4, 9]
    square = [] #blank list for new list

    #looping for add value in list square
    for i in num :
      square.append(i**2)
    print(square)

    [9, 16, 81]
```

Create New List Without List Comprehension

With *if* condition

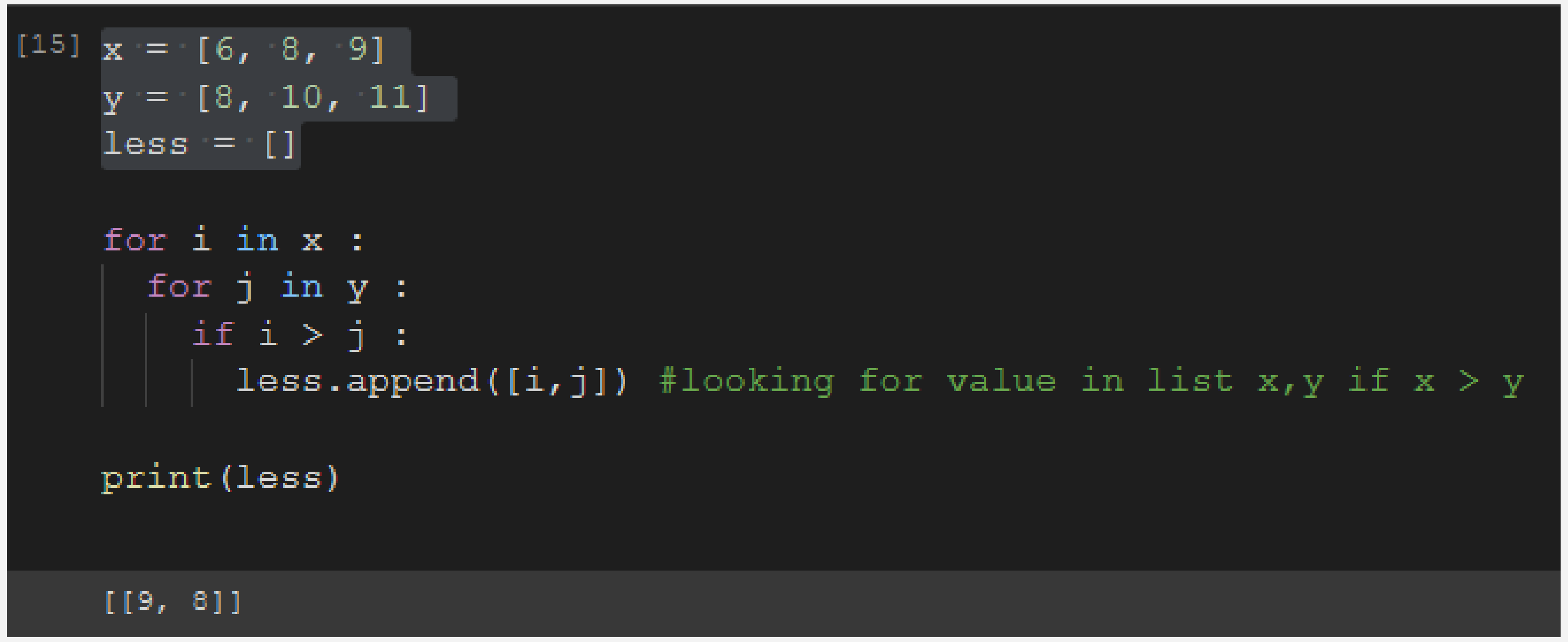


```
[3] fruit = ["aple", "melon", "kiwi", "banana"]
    new_fruit = []

    for i in fruit :
      if "a" in i :
        new_fruit.append(i)
    print(new_fruit)
```

```
['aple', 'banana']
```

Create New List Without List Comprehension

With Nested Loops

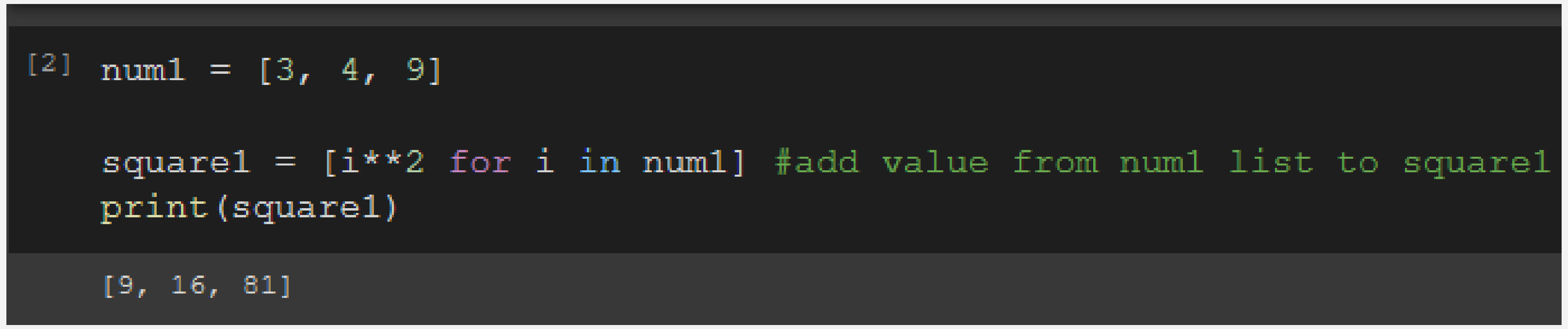


```
[15] x = [6, 8, 9]
     y = [8, 10, 11]
     less = []

     for i in x :
       for j in y :
         if i > j :
           less.append([i,j]) #looking for value in list x,y if x > y

     print(less)
```

```
[[9, 8]]
```

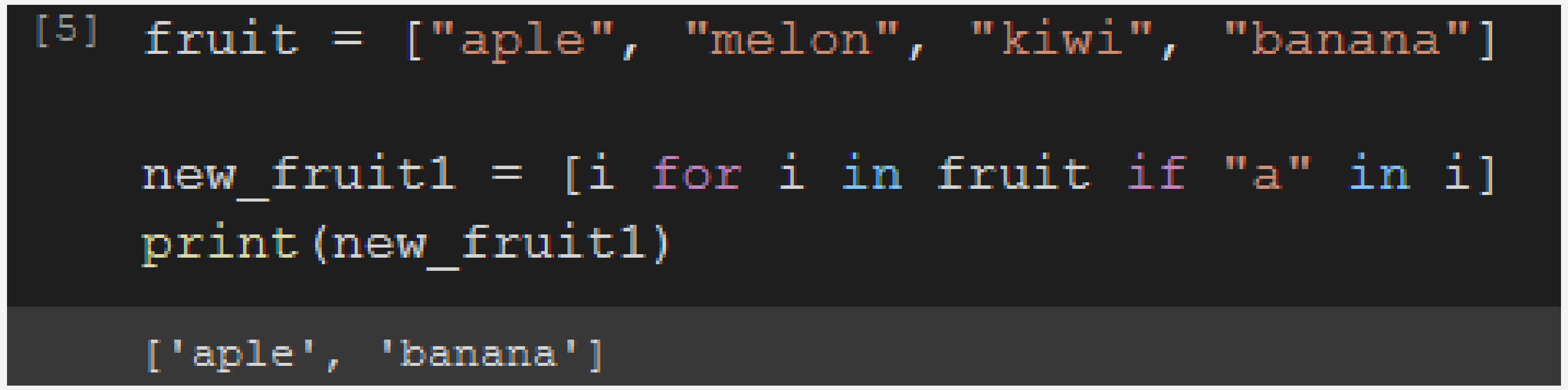
Create New List With List Comprehension

Without *if* condition



```
[2] num1 = [3, 4, 9]

    square1 = [i**2 for i in num1] #add value from num1 list to square1
    print(square1)

    [9, 16, 81]
```

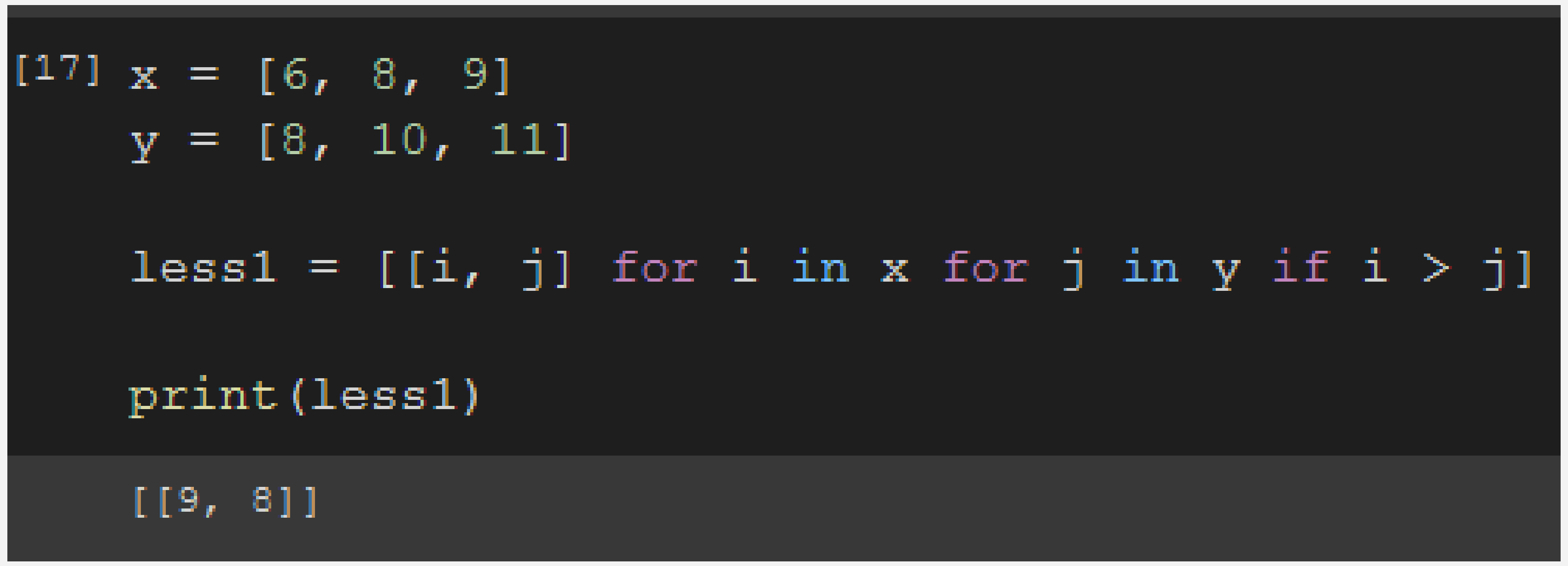
With *if* condition



```
[5] fruit = ["aple", "melon", "kiwi", "banana"]

    new_fruit1 = [i for i in fruit if "a" in i]
    print(new_fruit1)

    ['aple', 'banana']
```

Create New List With List Comprehension

With *Nested Loops*



```
[17] x = [6, 8, 9]
     y = [8, 10, 11]

     less1 = [[i, j] for i in x for j in y if i > j]

     print(less1)

     [[9, 8]]
```

Syntax Errors & Exceptions

Syntax Errors and Exceptions

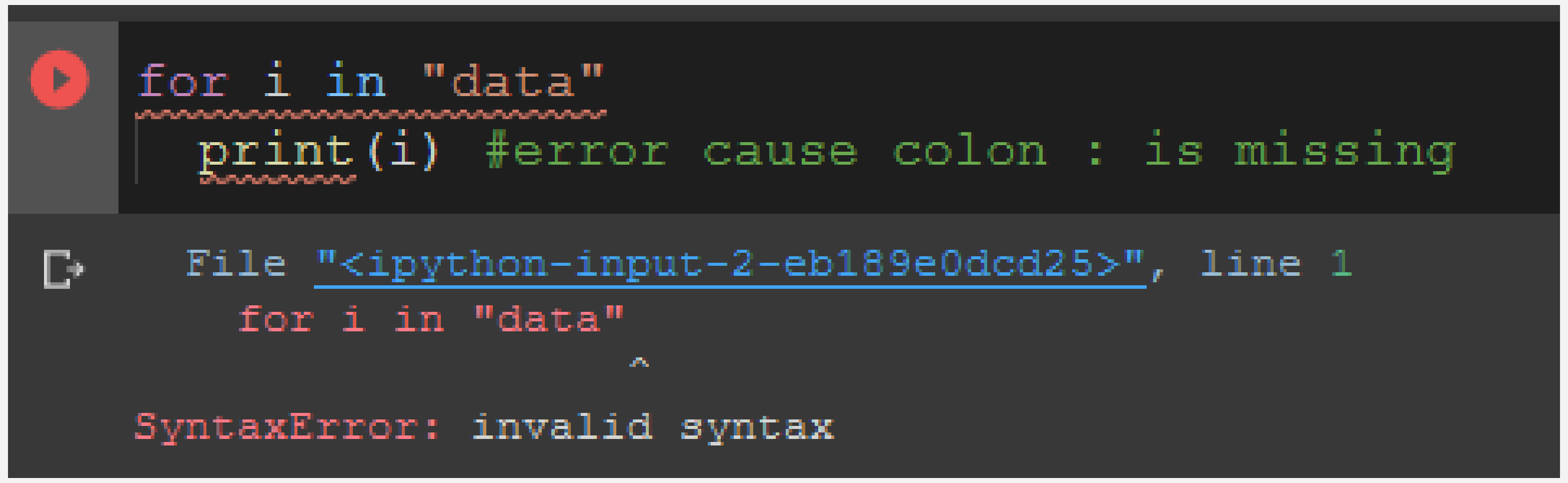
Python has two types of eror based on their occurrence

1. **Syntax Errors**
2. **Logical Errors (exceptions)**

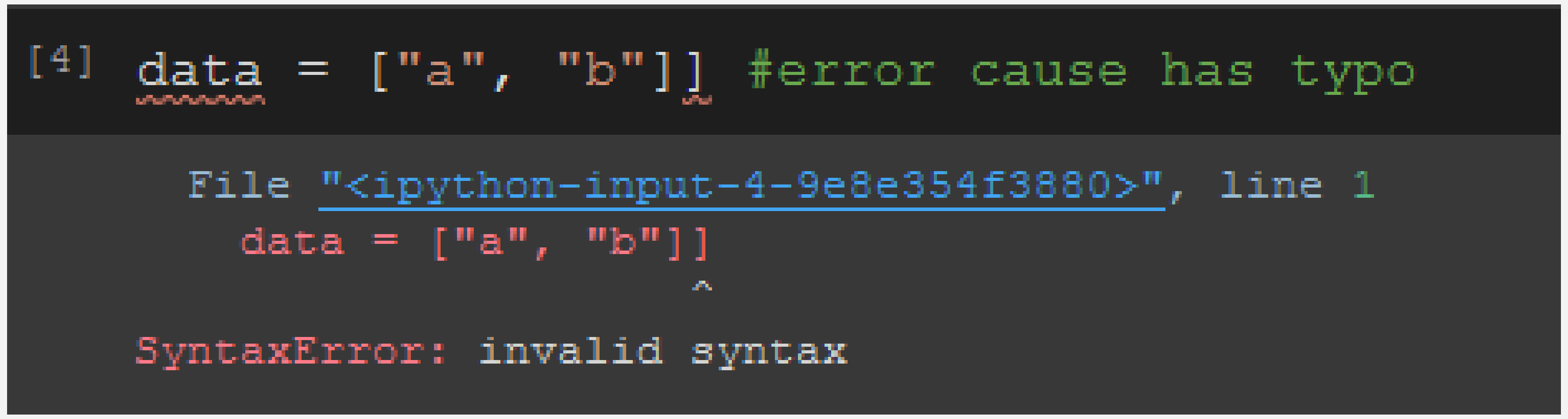
Syntax errors happens when Python don't understand what you ordered.

Logical errors (Exceptions) happens When in the runtime an error that occurs after passing the syntax test is called exception or logical type.

Syntax Error

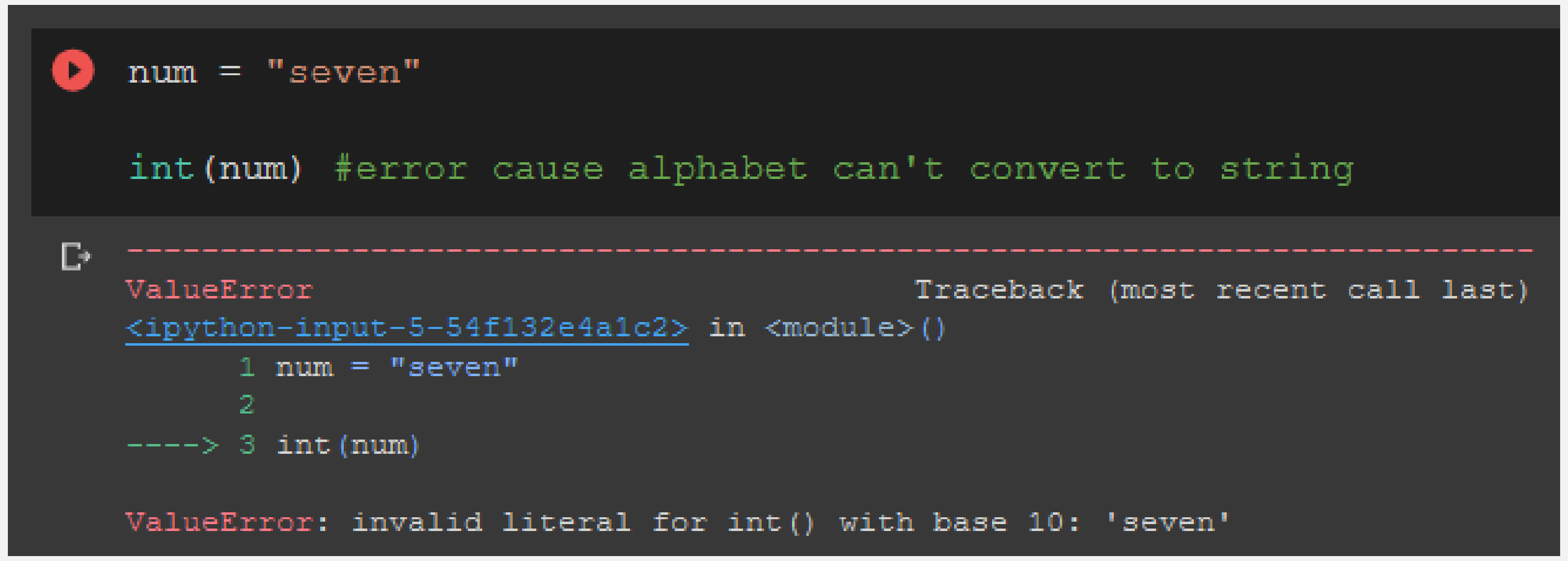


```
for i in "data"
  print(i) #error cause colon : is missing

  File "<ipython-input-2-eb189e0dcd25>", line 1
    for i in "data"
                   ^
SyntaxError: invalid syntax
```



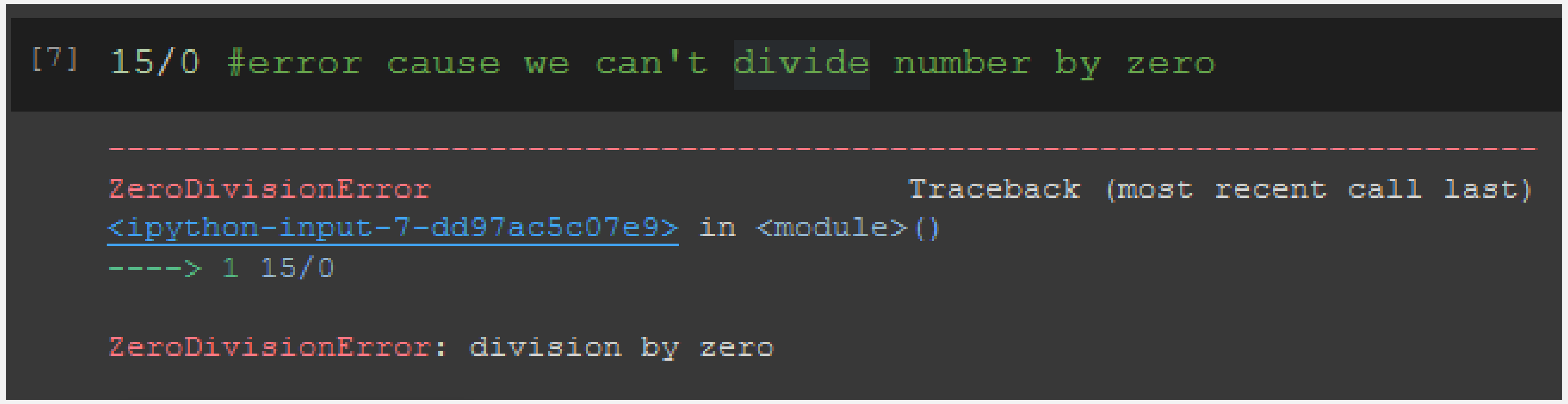
```
[4] data = ["a", "b"]] #error cause has typo

  File "<ipython-input-4-9e8e354f3880>", line 1
    data = ["a", "b"]]
                     ^
SyntaxError: invalid syntax
```

Logical Errors (Exceptions)



```
num = "seven"

int(num) #error cause alphabet can't convert to string
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-5-54f132e4a1c2> in <module>()
      1 num = "seven"
      2
----> 3 int(num)

ValueError: invalid literal for int() with base 10: 'seven'
```



```
[7] 15/0 #error cause we can't divide number by zero
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-7-dd97ac5c07e9> in <module>()
----> 1 15/0

ZeroDivisionError: division by zero
```

Exceptions Handling

Exceptions Handling

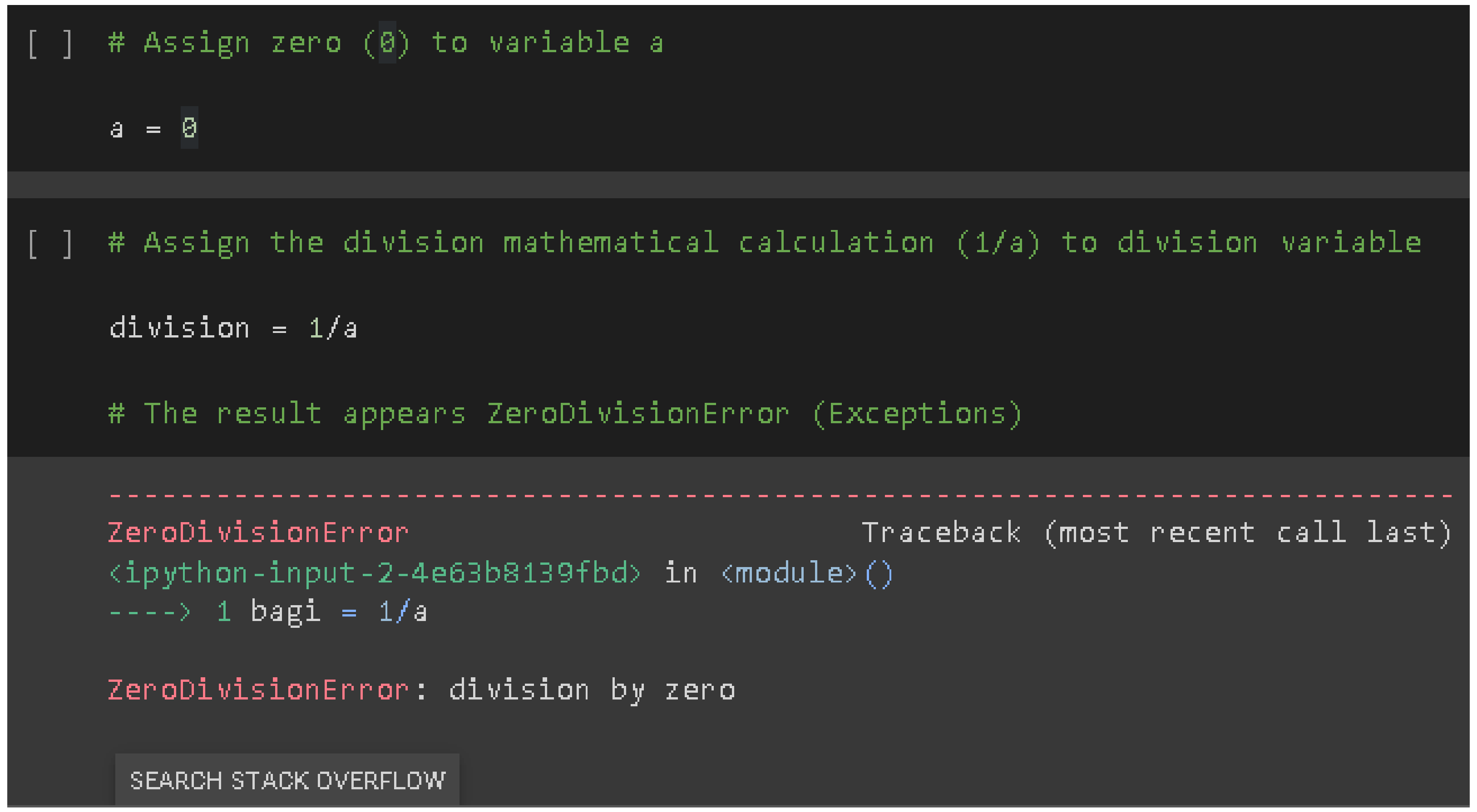
The exceptions handling process uses a try statement paired with except.

ZeroDivisionError

ZeroDivisionError is an exception that occurs when program execution results in a mathematical calculation of division by zero.

Number Divided by Zero

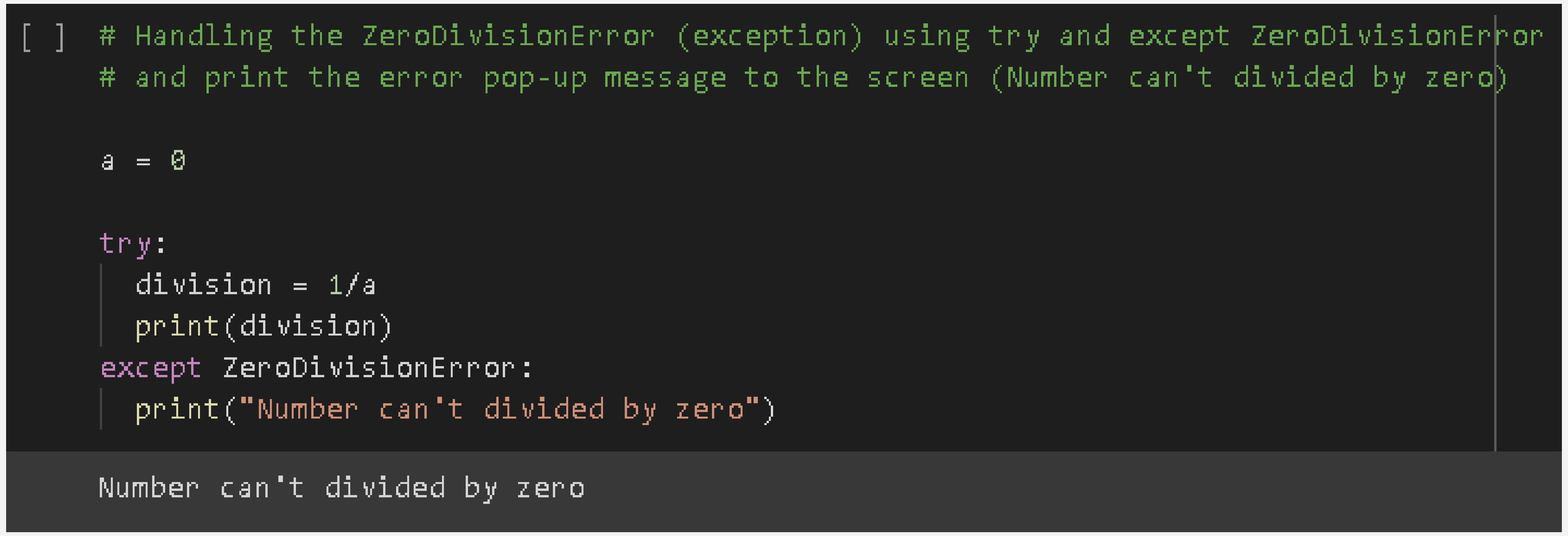
For example, we want to handle an exception that occurs when a number is divided by zero (0).



```
[ ]  # Assign zero (0) to variable a

     a = 0

[ ]  # Assign the division mathematical calculation (1/a) to division variable

     division = 1/a

     # The result appears ZeroDivisionError (Exceptions)

     ---------------------------------------------------------------------------
     ZeroDivisionError                          Traceback (most recent call last)
     <ipython-input-2-4e63b8139fbd> in <module>()
     ----> 1 bagi = 1/a

     ZeroDivisionError: division by zero
```

Handling ZeroDivisionError

Exception handling for ZeroDivisionError is using try and except ZeroDivisionError and print the error pop-up message to the screen.

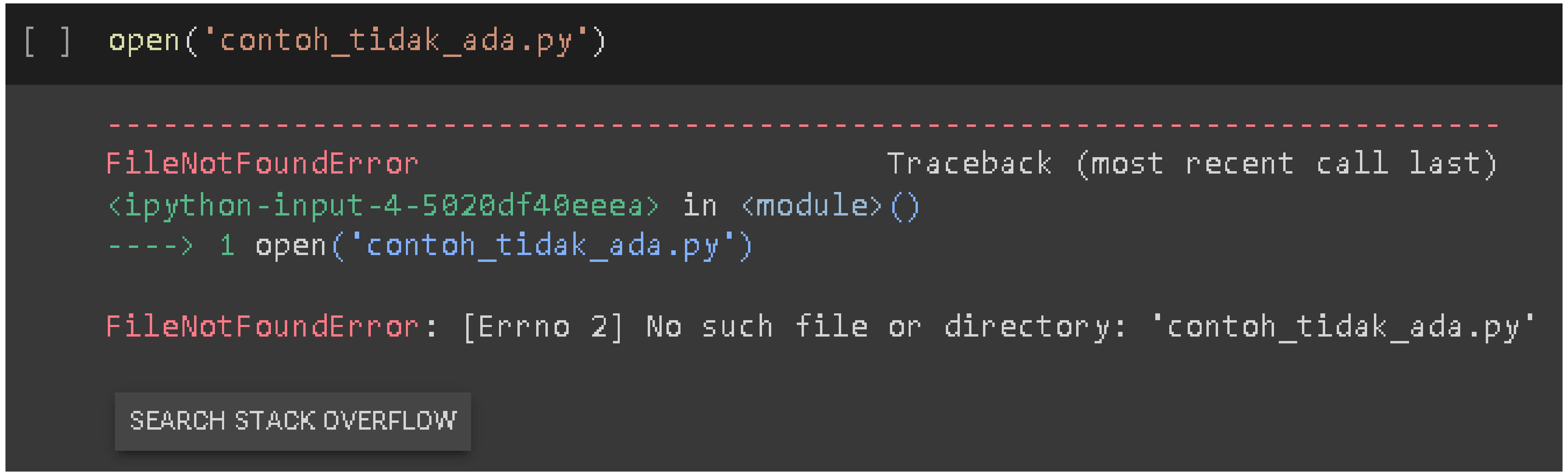


```
[ ]  # Handling the ZeroDivisionError (exception) using try and except ZeroDivisionError
     # and print the error pop-up message to the screen (Number can't divided by zero)

     a = 0

     try:
       division = 1/a
       print(division)
     except ZeroDivisionError:
       print("Number can't divided by zero")
```

```
     Number can't divided by zero
```

FileNotFoundError

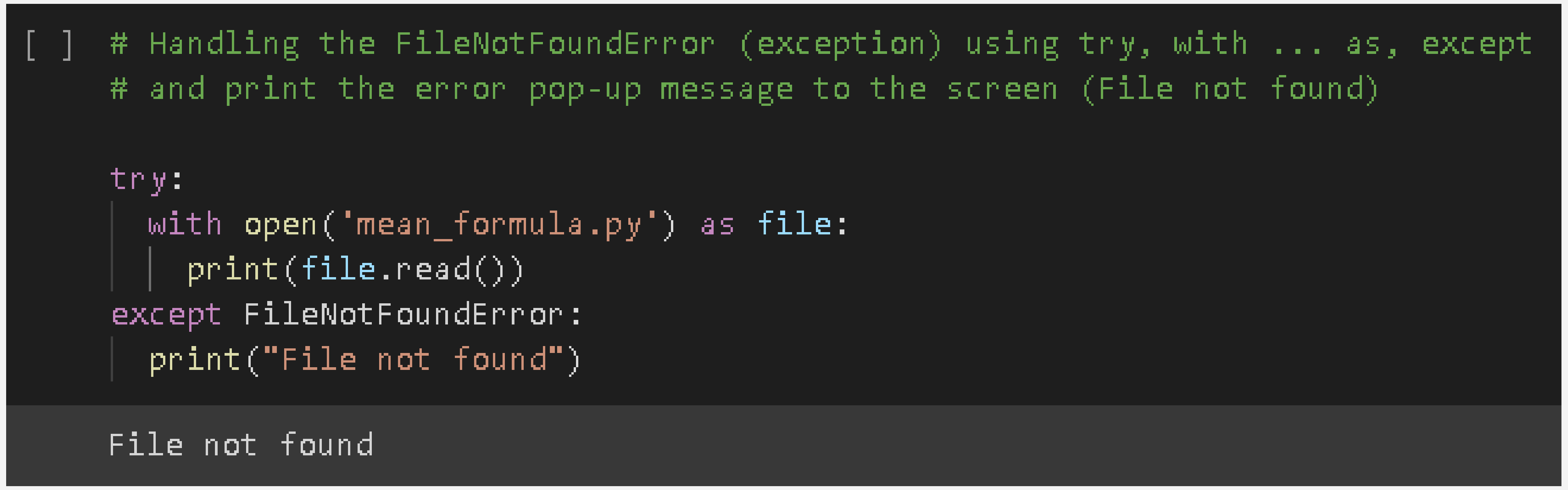
FileNotFoundError is an exception that occurs when the file to be open does not exist.



```
[ ]  open('contoh_tidak_ada.py')

     ---------------------------------------------------------------------------
     FileNotFoundError                         Traceback (most recent call last)
     <ipython-input-4-5020df40eeea> in <module>()
     ----> 1 open('contoh_tidak_ada.py')

     FileNotFoundError: [Errno 2] No such file or directory: 'contoh_tidak_ada.py'
```

Handling FileNotFoundError (try, with ... as, except)

Exception handling for FileNotFoundError is using try, with ... as, except and print the pop-up error message to the screen.

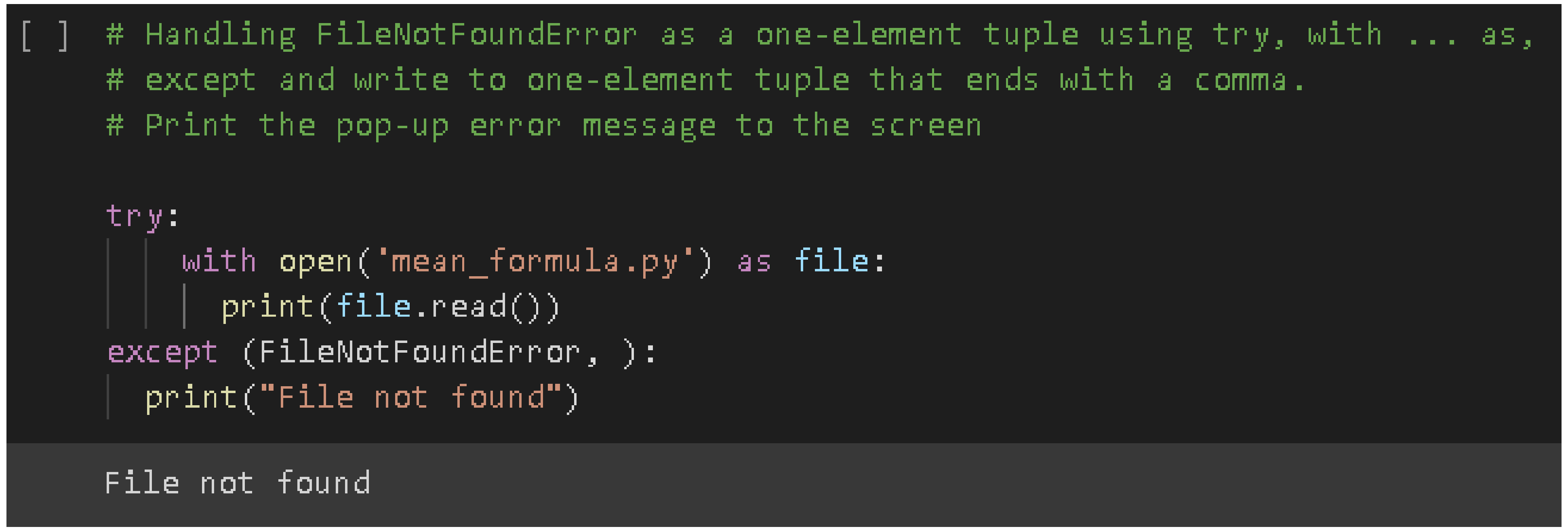


```
[ ]  # Handling the FileNotFoundError (exception) using try, with ... as, except
     # and print the error pop-up message to the screen (File not found)

     try:
       with open('mean_formula.py') as file:
         print(file.read())
     except FileNotFoundError:
       print("File not found")
```

```
     File not found
```

Handling FileNotFoundError (try, with ... as, except tuple)

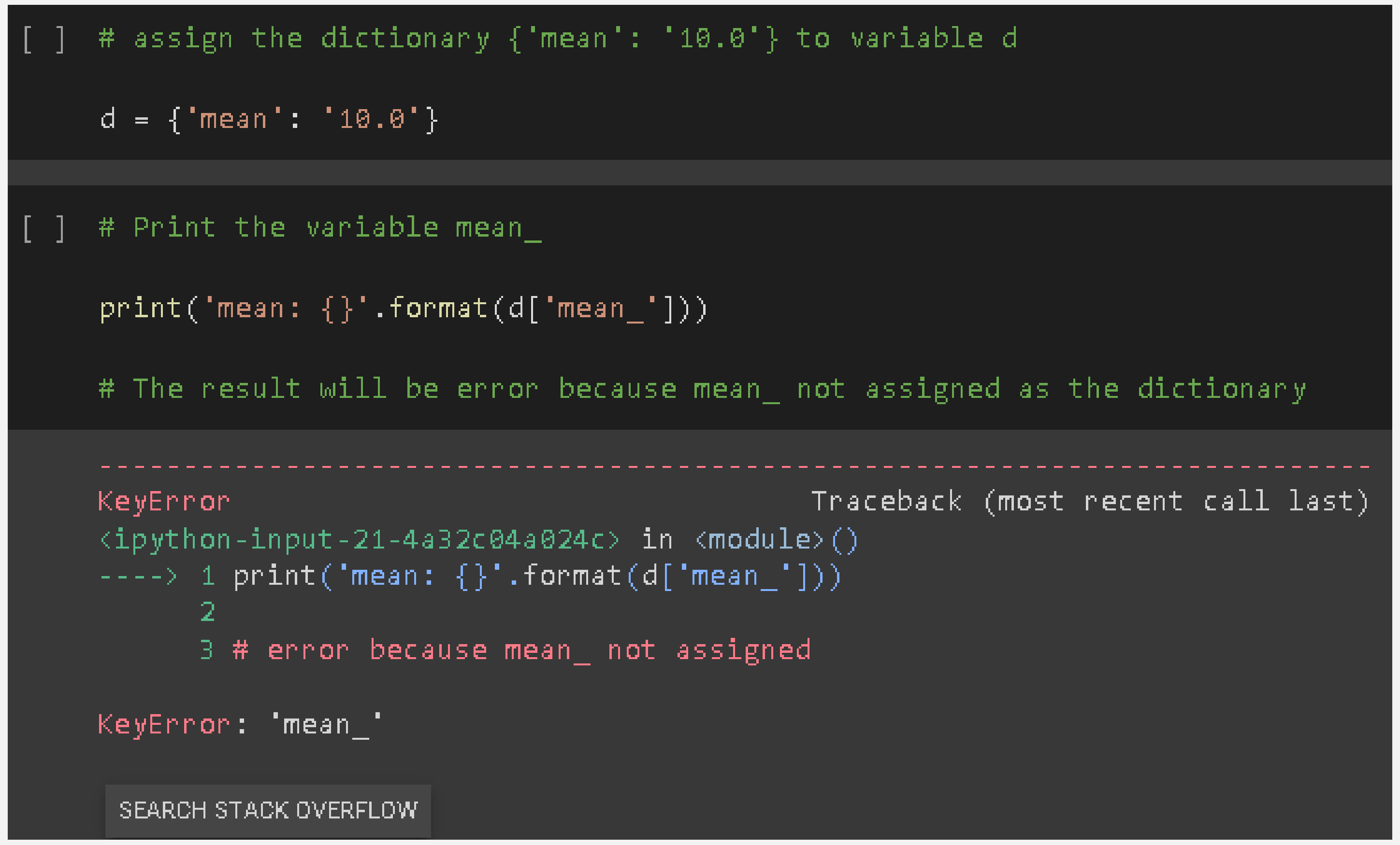
For example, if you handle FileNotFoundError as a one-element tuple, don't forget to write a one-element tuple that ends with a comma.



```
[ ]  # Handling FileNotFoundError as a one-element tuple using try, with ... as,
     # except and write to one-element tuple that ends with a comma.
     # Print the pop-up error message to the screen

     try:
         with open('mean_formula.py') as file:
           print(file.read())
     except (FileNotFoundError, ):
       print("File not found")

     File not found
```

KeyError

With a pair of try and except statements, the application does not stop but prints that the file was not found on the screen.



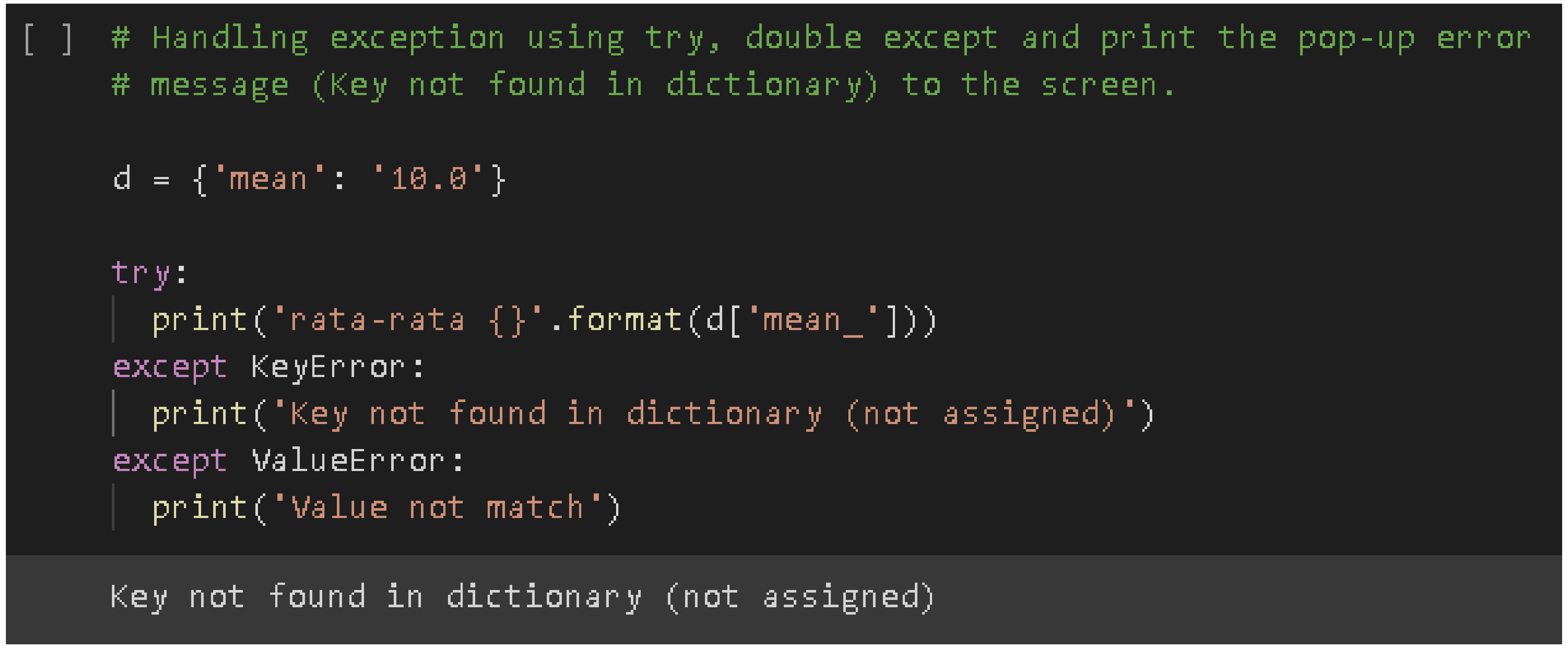
```
[ ]  # assign the dictionary {'mean': '10.0'} to variable d

     d = {'mean': '10.0'}
```

```
[ ]  # Print the variable mean_

     print('mean: {}'.format(d['mean_']))

     # The result will be error because mean_ not assigned as the dictionary
```

```
     ---------------------------------------------------------------------------
     KeyError                                  Traceback (most recent call last)
     <ipython-input-21-4a32c04a024c> in <module>()
     ----> 1 print('mean: {}'.format(d['mean_']))
           2
           3 # error because mean_ not assigned

     KeyError: 'mean_'
```

Handling KeyError (try, double except's statement)

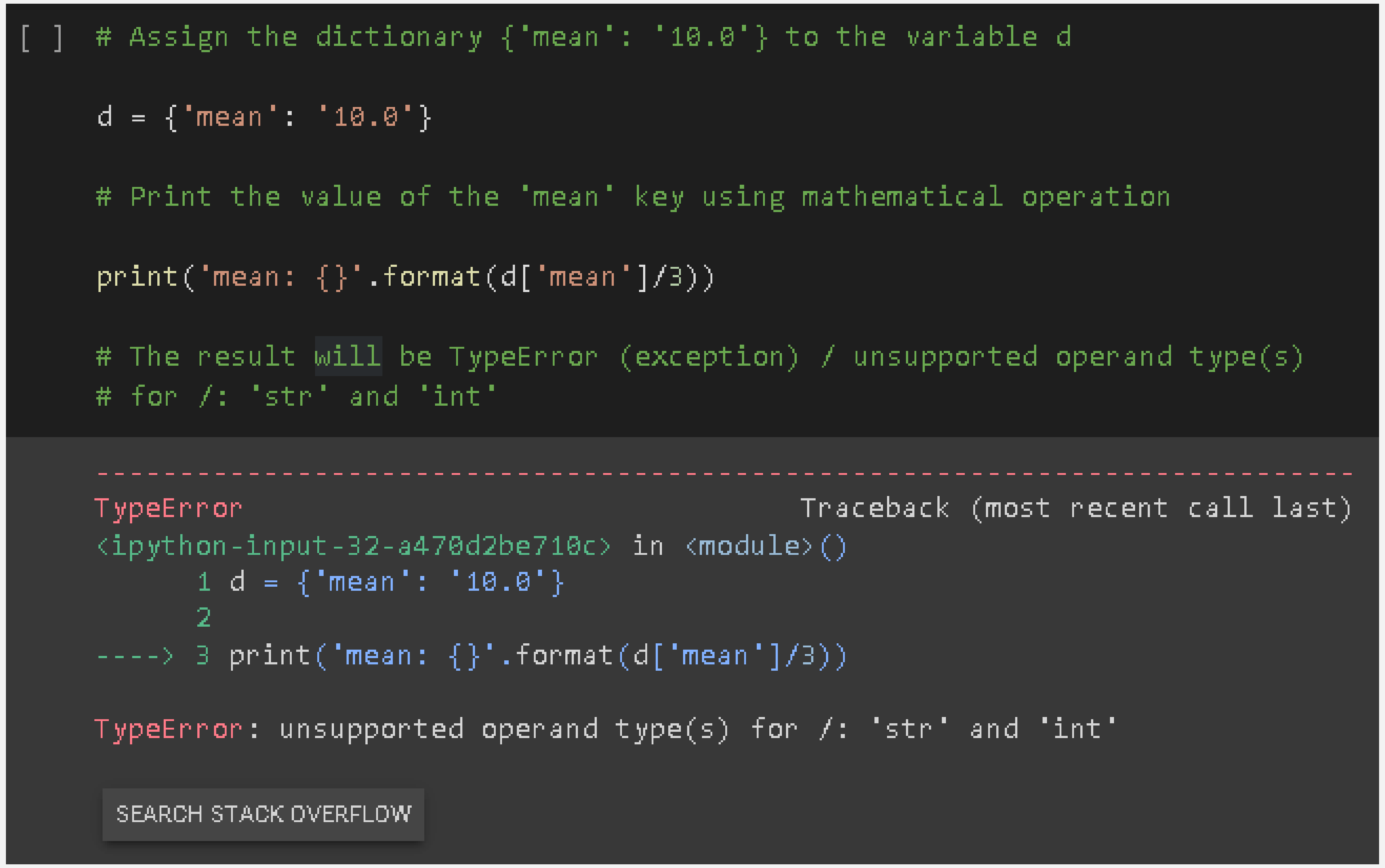
To handle this exception we can using try and double except.



```
[ ]  # Handling exception using try, double except and print the pop-up error
     # message (Key not found in dictionary) to the screen.

     d = {'mean': '10.0'}

     try:
       print('rata-rata {}'.format(d['mean_']))
     except KeyError:
       print('Key not found in dictionary (not assigned)')
     except ValueError:
       print('Value not match')
```

```
     Key not found in dictionary (not assigned)
```

TypeError

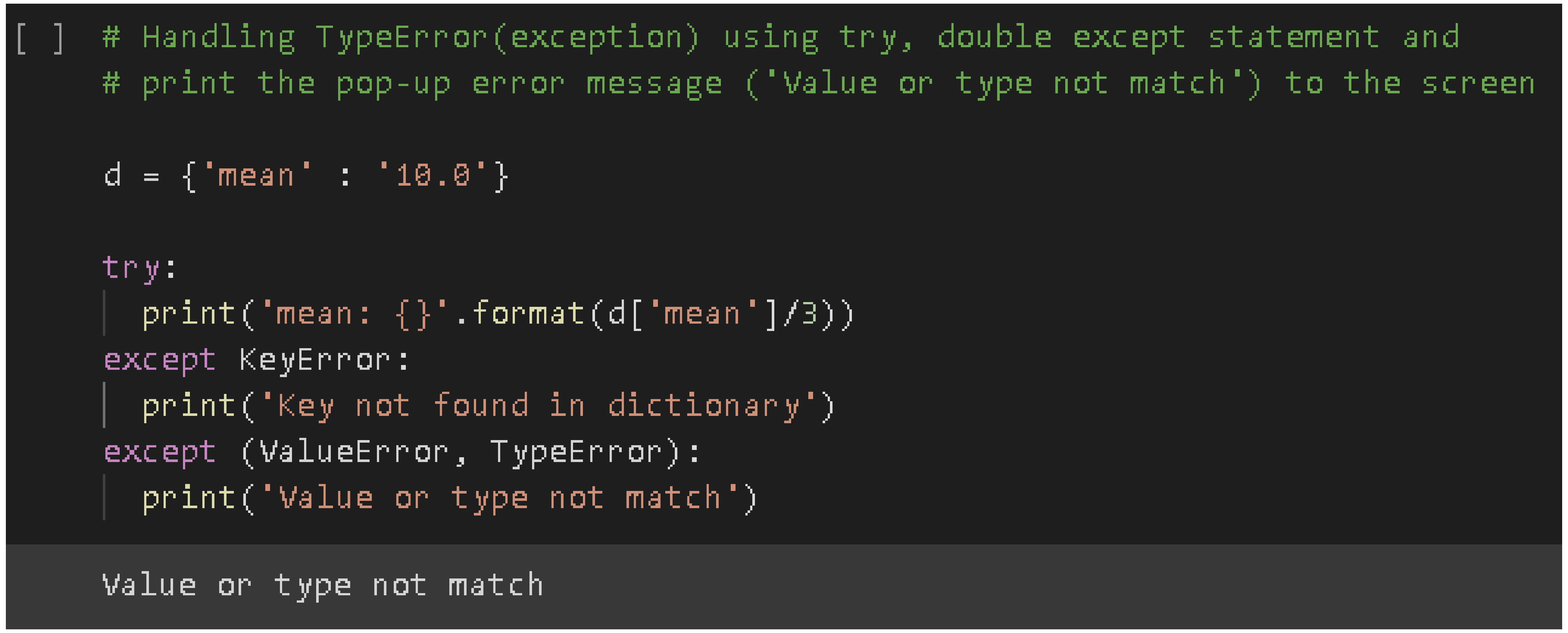


```
[ ] # Assign the dictionary {'mean': '10.0'} to the variable d

    d = {'mean': '10.0'}

    # Print the value of the 'mean' key using mathematical operation

    print('mean: {}'.format(d['mean']/3))

    # The result will be TypeError (exception) / unsupported operand type(s)
    # for /: 'str' and 'int'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-a470d2be710c> in <module>()
      1 d = {'mean': '10.0'}
      2
----> 3 print('mean: {}'.format(d['mean']/3))

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

Handling TypeError (try, double except's statement)

To handle TypeError (exception) use try, double except's statement and print the pop-up error message to the screen.

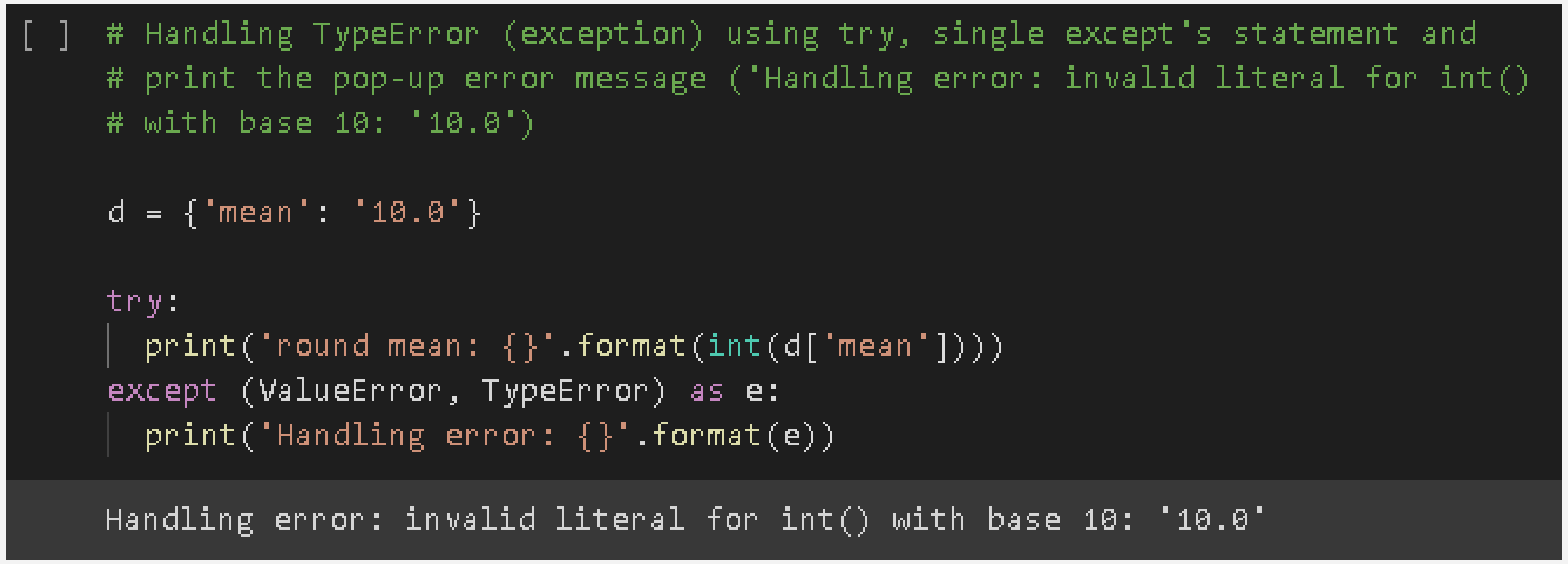


```
[ ] # Handling TypeError(exception) using try, double except statement and
    # print the pop-up error message ('Value or type not match') to the screen

    d = {'mean' : '10.0'}

    try:
      print('mean: {}'.format(d['mean']/3))
    except KeyError:
      print('Key not found in dictionary')
    except (ValueError, TypeError):
      print('Value or type not match')
```

```
    Value or type not match
```

Handling TypeError (try, single except's statement)

To handle TypeError (exception) use try, single except's statement and print the pop-up error message to the screen.



```
[ ]  # Handling TypeError (exception) using try, single except's statement and
     # print the pop-up error message ('Handling error: invalid literal for int()
     # with base 10: '10.0')

     d = {'mean': '10.0'}

     try:
       print('round mean: {}'.format(int(d['mean'])))
     except (ValueError, TypeError) as e:
       print('Handling error: {}'.format(e))

     Handling error: invalid literal for int() with base 10: '10.0'
```

Thank You!