

# PYTHON FUNCTIONS

**Example:**

## Python Functions:

### Examples :

1.

```
main.py  [Run]  Output  [Clear]

1 def square():
2     num = int(input("Enter a number to square: "))
3     """This function computes the square of the number."""
4     return num**2

Enter the example number (1-24): 1
Enter a number to square: 1000000000
1000000000000000000
```

2.

```
main.py [Icons] [Share] [Run] Output [Clear]
1 - def square():
2   num = int(input("Enter a number to square: "))
3   """This function computes the square of the number."""
4   return num**2
Enter the example number (1-24): 2
Enter a string to find its length: HAPPYMANINHAPPYWORLD^_^
23
```

3.

main.py

```
--  
11- def square_list():  
12     item_list = list(map(int, input("Enter numbers separated  
    by space to find squares: ").split()))  
13     '''This function will find the square of items in the  
        list'''  
14     squares = []  
15     for l in item_list:  
16         squares.append(l**2)  
17     return squares
```

Output

Clear

```
Enter the example number (1-24): 3  
Enter numbers separated by space to find squares: 6 1 5  
[36, 1, 25]  
  
=== Code Execution Successful ===
```

4.

```
main.py [ ] [ ] [ ] Share Run Output Clear
19 - def function():
20     n1 = int(input("Enter number 1: "))
21     n2 = int(input("Enter number 2 (default 20): ") or 20)
22     print("number 1 is:", n1)
23     print("number 2 is:", n2)
24
```

Enter the example number (1-24): 4  
Enter number 1: 60000000  
Enter number 2 (default 20): 1000000000000  
number 1 is: 60000000  
number 2 is: 1000000000000

5.

```
main.py [ ] [ ] Share Run Output Clear
25 def keyword_function():
26     n1 = int(input("Enter number 1: "))
27     n2 = int(input("Enter number 2: "))
28     print("number 1 is:", n1)
29     print("number 2 is:", n2)
```

Enter the example number (1-24): 5  
Enter number 1: 23613265  
Enter number 2: 9941392955  
number 1 is: 23613265  
number 2 is: 9941392955

```
main.py  [Full Screen] [Settings] [Share] [Run] [Clear]

31 def function_with_args():
32     args_list = input("Enter multiple words separated by space
        : ").split()
33     ans = []
34     for l in args_list:
35         ans.append(l.upper())
36     return ans
37
```

Enter the example number (1-24): 6  
Enter multiple words separated by space: Happy man in happy world  
['HAPPY', 'MAN', 'IN', 'HAPPY', 'WORLD']

=== Code Execution Successful ===

```
main.py  [Icons] Share Run Output Clear
38 def square_with_return():
39     num = int(input("Enter a number to square: "))
40     return num**2

Enter the example number (1-24): 7
Enter a number to square: 77
5929
```

```
main.py  [Run] [Share] [Output] [Clear]
46 lambda_ = lambda: int(input("Enter number 1: ")) + int(input
    ("Enter number 2: "))
47
48
49
50
```

Enter the example number (1-24): 9  
Enter number 1: 20  
Enter number 2: 28  
48  
=== Code Execution Successful ===

```
main.py [ ] [ ] Share Run Output Clear
52 def scope_example():
53     num = 50
54     print("Value of num inside the function:", num)
55
56
57
```

Enter the example number (1-24): 10  
Value of num inside the function: 50  
=== Code Execution Successful ===

```
main.py  [ ] [ ] [ ] Share Run Output Clear
58- def nested_function():
59-     string = 'Python functions tutorial'
60-     x = 5
61-     def inner_function():
62-         print(string)
63-         print(x)
64-     inner_function()
65-
```

Enter the example number (1-24): 11  
Python functions tutorial  
5

=== Code Execution Successful ===

Programiz Python Online Compiler

Programiz PRO >

main.py

Share

Run

Output

Clear

```
66
67 def abs_example():
68     integer = int(input("Enter a number to find its absolute
69                         value: "))
70     print('Absolute value is:', abs(integer))
```

```
Enter the example number (1-24): 12
Enter a number to find its absolute value: -22
Absolute value is: 22

=== Code Execution Successful ===
```

Programiz Python Online Compiler

Programiz PRO >

main.py

Share

Run

Output

Clear

```
--
70 def bin_example():
71     x = int(input("Enter a number to convert to binary: "))
72     y = bin(x)
73     print('Binary value is: ', y)
```

Enter the example number (1-24): 13  
Enter a number to convert to binary: 123  
Binary value is: 0b1111011

13.

```
Programiz Python Online Compiler Programiz PRO >
```

```
main.py 🔍 🌞 🔗 Share Run Output Clear
```

```
75 def callable_example():
76     x = 8
77     print(callable(x))
78
```

```
Enter the example number (1-24): 14
False
=== Code Execution Successful ===
```

14.

```
Programiz Python Online Compiler Programiz PRO >
```

```
main.py 🔍 🌞 🔗 Share Run Output Clear
```

```
79 def sum_example():
80     s = sum([int(i) for i in input("Enter numbers separated by
81     space to sum: ").split())]
82     print("Sum of numbers is:", s)
83
```

```
Enter the example number (1-24): 15
Enter numbers separated by space to sum: 3 7 2 1 2 3 1
Sum of numbers is: 19
=== Code Execution Successful ===
```

15.

```
Programiz Python Online Compiler Programiz PRO >
```

```
main.py 🔍 🌞 🔗 Share Run Output Clear
```

```
79 def sum_example():
80     s = sum([int(i) for i in input("Enter numbers separated by
81     space to sum: ").split())]
82     print("Sum of numbers is:", s)
83
```

```
Enter the example number (1-24): 16
Enter numbers separated by space to check 'any': 2 3 4 1 2
True
=== Code Execution Successful ===
```

16.

```
main.py 🔍 🌞 🔗 Share Run Output Clear
```

```
97 def frozenset_example():
98     letters = input("Enter letters separated by space to
99     create a frozen set: ").split()
100     fSet = frozenset(letters)
101     print('Frozen set is:', fSet)
102
```

```
Enter the example number (1-24): 20
Enter letters separated by space to create a frozen set: 2 8 9 7 6
Frozen set is: frozenset({'2', '6', '7', '9', '8'})
=== Code Execution Successful ===
```

17.

```
main.py 🔍 🌞 🔗 Share Run Output Clear
```

```
1 # Example 1: tuple() function
2 def tuple_example():
3     print("\nExample 1: Creating a Tuple")
4     t1 = tuple()
5     print('t1=', t1)
6
7     t2 = tuple([1, 6, 9])
8     print('t2=', t2)
9
10    t1 = tuple('Java')
11    print('t1=', t1)
12
13    t1 = tuple({'4': 'four', 5: 'five'})
14    print('t1=', t1)
15
```

```
Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 1

Example 1: Creating a Tuple
t1= ()
t2= (1, 6, 9)
t1= ('J', 'a', 'v', 'a')
t1= (4, 5)

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 
```

18.

Programiz Python Online Compiler

Programiz PRO >

main.py

Share

Run

```
16 # Example 2: type() function
17 def type_example():
18     print("\nExample 2: Python type() Function")
19     List = [4, 5]
20     print(type(List))
21
22     Dict = {4: 'four', 5: 'five'}
23     print(type(Dict))
24
25     class Python:
26         a = 0
27     InstanceOfPython = Python()
28     print(type(InstanceOfPython))
29
```

Output

Clear

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 2

Example 2: Python type() Function
<class 'list'>
<class 'dict'>
<class '\_\_main\_\_.type\_example.<locals>.Python'>

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit:

19.

Programiz Python Online Compiler

Programiz PRO >

main.py

Share

Run

```
30 # Example 3: vars() function
31 def vars_example():
32     print("\nExample 3: Python vars() Function")
33     class Python:
34         def __init__(self, x=7, y=9):
35             self.x = x
36             self.y = y
37     InstanceOfPython = Python()
38     print(vars(InstanceOfPython))
39
```

Output

Clear

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 3

Example 3: Python vars() Function
{'x': 7, 'y': 9}

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit:

20.

Programiz Python Online Compiler

Programiz PRO >

main.py

Share

Run

```
40 # Example 4: zip() function
41 def zip_example():
42     print("\nExample 4: Python zip() Function")
43     numList = [4, 5, 6]
44     strList = ['four', 'five', 'six']
45     result = zip(numList, strList)
46     resultSet = set(result)
47     print(resultSet)
```

Output

Clear

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 4

Example 4: Python zip() Function
{(6, 'six'), (5, 'five'), (4, 'four')}

Choose an example to run (1-10):

21.

Programiz Python Online Compiler

Programiz PRO >

main.py

Share

Run

```
49 # Example 5: Lambda Function Example 1 (Add 4 to a number)
50 def lambda_example_1():
51     print("\nExample 5: Lambda Function to Add 4")
52     add = lambda num: num + 4
53     num = int(input("Enter a number to add 4: "))
54     print("Result:", add(num))
55
```

Output

Clear

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 5

Example 5: Lambda Function to Add 4
Enter a number to add 4: 11
Result: 15

22,23,24:

main.py

Share

Run

Output

Clear

```
56 # Example 6: Lambda with filter() to find odd numbers
57 def lambda_filter_example():
58     print("\nExample 6: Lambda with filter()")
59     list_ = [35, 12, 69, 55, 75, 14, 73]
60     odd_list = list(filter(lambda num: (num % 2 != 0), list_))
61     print('The list of odd numbers is:', odd_list)
62
63 # Example 7: Lambda with map() to square numbers
64 def lambda_map_example():
65     print("\nExample 7: Lambda with map() to square numbers")
66     numbers_list = [2, 4, 5, 1, 3, 7, 8, 9, 10]
67     squared_list = list(map(lambda num: num ** 2, numbers_list))
68     print('Square of each number in the given list:', squared_list
69         )
70
71 # Example 8: Lambda with List Comprehension
72 def lambda_comprehension_example():
73     print("\nExample 8: Lambda with List Comprehension")
74     squares = [lambda num=num: num ** 2 for num in range(0, 11)]
75     print('The square value of all numbers from 0 to 10:', end=" "
```

Choose an example to run (1-10):  
Enter the example number (1-10) or 0 to exit: 6  
  
Example 6: Lambda with filter()  
The list of odd numbers is: [35, 69, 55, 75, 73]  
  
Choose an example to run (1-10):  
Enter the example number (1-10) or 0 to exit: 7  
  
Example 7: Lambda with map() to square numbers  
Square of each number in the given list: [4, 16, 25, 1, 9, 49, 64, 81, 100]  
  
Choose an example to run (1-10):  
Enter the example number (1-10) or 0 to exit: 8  
  
Example 8: Lambda with List Comprehension  
The square value of all numbers from 0 to 10: 0 1 4 9 16 25 36 49 64 81 100  
Choose an example to run (1-10):

25,26:

Programiz Python Online Compiler

Programiz PRO >

main.py

Share

Run

Output

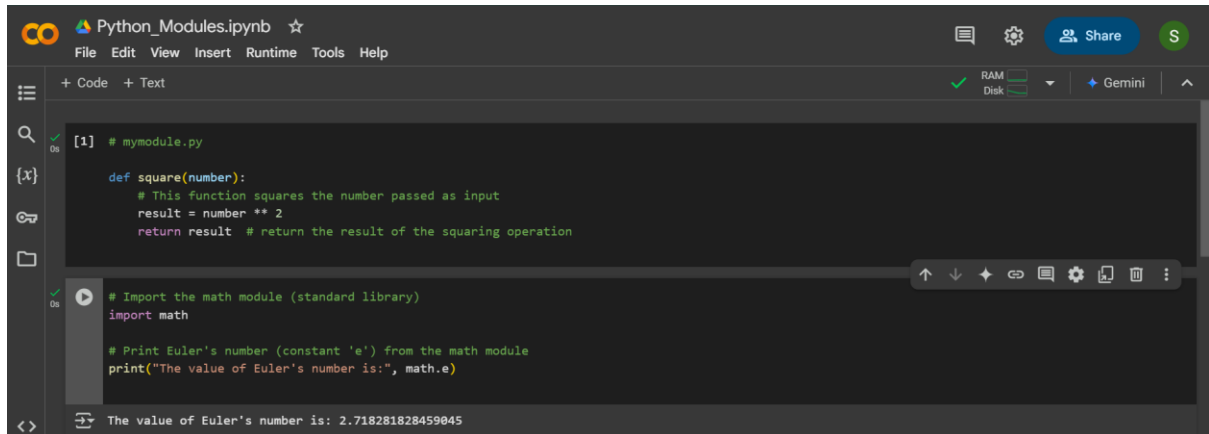
Clear

```
77
78 # Example 9: Lambda with if-else
79 def lambda_if_else_example():
80     print("\nExample 9: Lambda Function with if-else")
81     Minimum = lambda x, y: x if (x < y) else y
82     x = int(input("Enter the first number: "))
83     y = int(input("Enter the second number: "))
84     print(f'The smaller number is: {Minimum(x, y)}')
85
86 # Example 10: Lambda with Multiple Statements (Find third largest)
87 def lambda_multiple_statements_example():
88     print("\nExample 10: Lambda with Multiple Statements")
89     my_list = [[3, 5, 8, 6], [23, 54, 12, 87], [1, 2, 4, 12, 5]]
90     sort_list = lambda num: (sorted(n) for n in num)
91     third_largest = lambda num, func: [l[len(l) - 2] for l in func
92         (num)]
93     result = third_largest(my_list, sort_list)
94     print('The third largest number from every sub-list is:',
95         result)
```

Choose an example to run (1-10):  
Enter the example number (1-10) or 0 to exit: 9  
  
Example 9: Lambda Function with if-else  
Enter the first number: 11  
Enter the second number: 22  
The smaller number is: 11  
  
Choose an example to run (1-10):  
Enter the example number (1-10) or 0 to exit: 10  
  
Example 10: Lambda with Multiple Statements  
The third largest number from every sub-list is: [6, 54, 5]  
  
Choose an example to run (1-10):  
Enter the example number (1-10) or 0 to exit:

# Python Modules

## Python import Statement



The image shows a Jupyter Notebook interface with a dark theme. The top bar includes the Colab logo, the filename 'Python\_Modules.ipynb', and a star icon. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The left sidebar contains icons for file explorer, search, and other notebook functions. The main area displays two code cells. The first cell, labeled '[1]', contains a Python function definition: 

```
def square(number):  
    # This function squares the number passed as input  
    result = number ** 2  
    return result # return the result of the squaring operation
```

. The second cell contains an import statement and a print statement: 

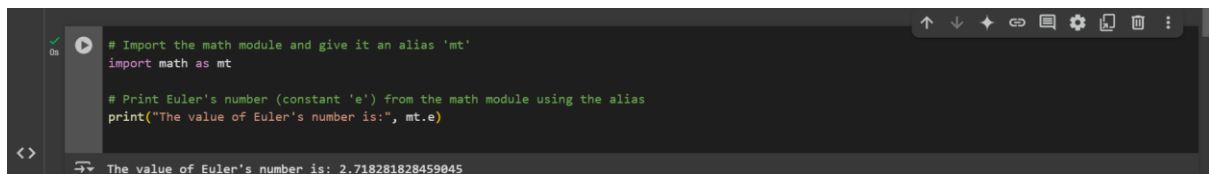
```
# Import the math module (standard library)  
import math  
  
# Print Euler's number (constant 'e') from the math module  
print("The value of Euler's number is:", math.e)
```

. The output of the second cell is displayed at the bottom: 'The value of Euler's number is: 2.718281828459045'.

```
[1] # mymodule.py  
  
def square(number):  
    # This function squares the number passed as input  
    result = number ** 2  
    return result # return the result of the squaring operation  
  
# Import the math module (standard library)  
import math  
  
# Print Euler's number (constant 'e') from the math module  
print("The value of Euler's number is:", math.e)
```

The value of Euler's number is: 2.718281828459045

## Importing and also Renaming



The image shows a Jupyter Notebook interface with a dark theme. The main area displays a code cell with the following code: 

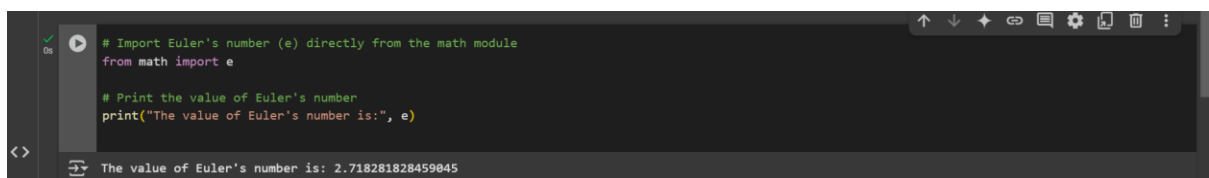
```
# Import the math module and give it an alias 'mt'  
import math as mt  
  
# Print Euler's number (constant 'e') from the math module using the alias  
print("The value of Euler's number is:", mt.e)
```

. The output of the cell is displayed at the bottom: 'The value of Euler's number is: 2.718281828459045'.

```
# Import the math module and give it an alias 'mt'  
import math as mt  
  
# Print Euler's number (constant 'e') from the math module using the alias  
print("The value of Euler's number is:", mt.e)
```

The value of Euler's number is: 2.718281828459045

## Python from...import Statement



The image shows a Jupyter Notebook interface with a dark theme. The main area displays a code cell with the following code: 

```
# Import Euler's number (e) directly from the math module  
from math import e  
  
# Print the value of Euler's number  
print("The value of Euler's number is:", e)
```

. The output of the cell is displayed at the bottom: 'The value of Euler's number is: 2.718281828459045'.

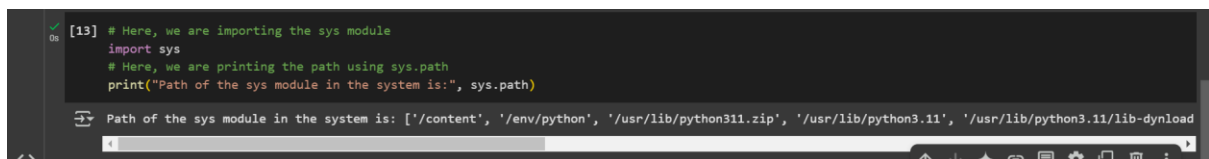
```
# Import Euler's number (e) directly from the math module  
from math import e  
  
# Print the value of Euler's number  
print("The value of Euler's number is:", e)
```

The value of Euler's number is: 2.718281828459045

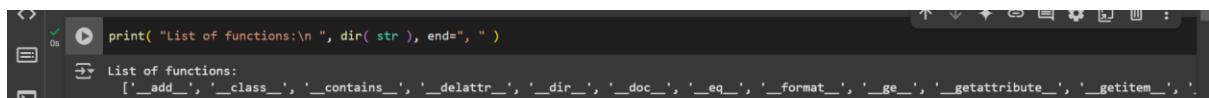
## Import all Names - From import \* Statement

```
[x] ✓ On  #from name_of_module import *  
# Import everything from the math module using *  
from math import *  
  
# Access functions directly without using the dot operator  
  
# Calculate the square root of 25  
print("Calculating square root:", sqrt(25))  
  
# Calculate the tangent of an angle (pi/6 radians)  
print("Calculating tangent of an angle:", tan(pi/6))  
  
Calculating square root: 5.0  
Calculating tangent of an angle: 0.5773502691896257
```

## Locating Path of Modules

```
[13] ✓ On  # Here, we are importing the sys module  
import sys  
# Here, we are printing the path using sys.path  
print("Path of the sys module in the system is:", sys.path)  
  
Path of the sys module in the system is: ['./content', '/env/python', '/usr/lib/python311.zip', '/usr/lib/python3.11', '/usr/lib/python3.11/lib-dynload']
```

## The dir() Built-in Function

```
<> ✓ On  print("List of functions:\n", dir( str ), end=" ", )  
  
List of functions:  
['_add_', '_class_', '_contains_', '_delattr_', '_dir_', '_doc_', '_eq_', '_format_', '_ge_', '_getattr_', '_getitem_', '.']
```

## Namespaces and Scoping

```
 # Global variable  
Number = 204  
  
# Define the function AddNumber  
def AddNumber():  
    # Accessing the global variable 'Number'  
    global Number  
    # Modify the global 'Number' variable by adding 200  
    Number = Number + 200  
    # Print the number after addition within the function  
    print("The number inside the function is:", Number)  
  
# Call the AddNumber function  
AddNumber()  
  
# Print the global 'Number' after calling the function  
print("The number outside the function is:", Number)  
  
The number inside the function is: 404  
The number outside the function is: 404
```

## Python Exceptions

```
string = "Python Exceptions"

# Loop through each character in the string
for s in string:
    # Check if the character is not 'o'
    if s != 'o':
        # Print the character if it's not 'o'
        print(s)

P
y
t
h
o
n

E
x
c
e
p
t
i
o
n
s

0s completed at 4:58 PM
```

## Try and Except Statement - Catching Exceptions

```
# Define the list
a = ["Python", "Exceptions", "try and except"]

try:
    # Looping through the elements of the list, going beyond the length of the list
    for i in range(4):
        # Print the index and element from the array
        print("The index and element from the array is", i, a[i])

# Catch any exception that occurs and print a message
except:
    print("Index out of range")

The index and element from the array is 0 Python
The index and element from the array is 1 Exceptions
The index and element from the array is 2 try and except
Index out of range
```

## How to Raise an Exception

```
try:
    num = [3, 4, 5, 7]
    if len(num) > 3:
        raise Exception(f"Length of the given list must be less than or equal to 3 but is {len(num)}")
except Exception as e:
    print(f"Caught an error: {e}")

Caught an error: Length of the given list must be less than or equal to 3 but is 4
```

## Assertions in Python

```
def square_root(Number):
    assert (Number >= 0), "Give a non-negative integer" # Assert non-negative number
    return Number ** (1/2)

# Calling function with valid and invalid inputs
print(square_root(36)) # Valid input, should return 6.0
print(square_root(-36)) # Invalid input, should raise AssertionError

[ ] # Calling function and passing the values
----> 8 print( square_root( 36 ) )
9 print( square_root( -36 ) )
Input In [23], in square_root(Number)
3 def square_root( Number ):
----> 4 assert ( Number <= 0 ), "Give a positive integer"
5 return Number**(1/2)
AssertionError: Give a positive integer
```



## Try with Else Clause

```
{x}
def reciprocal(num1):
    try:
        # Attempting to calculate the reciprocal
        reci = 1 / num1
    except ZeroDivisionError:
        # Catching division by zero error
        print("We cannot divide by zero")
    else:
        # Executed if no exception occurs
        print(reci)

# Calling the function with valid and invalid inputs
reciprocal(4) # Valid input, should print the reciprocal
reciprocal(0) # Invalid input, should print the error message
```

0.25  
We cannot divide by zero

## Finally Keyword in Python

```
# Raising an exception in try block
try:
    div = 4 // 0 # This will raise a ZeroDivisionError
    print(div)
except ZeroDivisionError:
    # This block will handle the exception raised
    print("Attempting to divide by zero")
finally:
    # This block will always be executed, no matter if an exception was raised or not
    print('This is code of finally clause')
```

Attempting to divide by zero  
This is code of finally clause

## User-Defined Exceptions

```
# Defining a custom exception class
class EmptyError(RuntimeError):
    def __init__(self, argument):
        self.arguments = argument

# Code that raises the exception
var = " " # Variable that will be checked
try:
    if not var.strip(): # Check if the variable is empty or contains only whitespace
        raise EmptyError("The variable is empty")
except EmptyError as e: # Catching the custom exception
    print(e.arguments) # Output the exception message
```

The variable is empty

## try, except, else, and finally clauses

```
try:
    # Code block
    # These statements are those which can probably have some error
    num1 = 10
    num2 = 0
    result = num1 / num2 # This will raise a ZeroDivisionError
    print(result)
except ZeroDivisionError:
    # This block is optional.
    # If the try block encounters an exception, this block will handle it.
    print("You can't divide by zero!")
else:
    # If there is no exception, this code block will be executed by the Python interpreter
    print("Division was successful")
finally:
    # Python interpreter will always execute this code.
    print("This is the finally block. It always runs, regardless of exceptions.")
```

You can't divide by zero!  
This is the finally block. It always runs, regardless of exceptions.

## Python Arrays

### Accessing array element

```
import array as arr

# Creating an array of integers
a = arr.array('i', [2, 4, 5, 6])

# Printing elements by positive index
print("First element is:", a[0])
print("Second element is:", a[1])
print("Third element is:", a[2])
print("Forth element is:", a[3])

# Printing elements by negative index
print("Last element is:", a[-1])
print("Second last element is:", a[-2])
print("Third last element is:", a[-3])
print("Forth last element is:", a[-4])

# Printing all elements using both positive and negative indices
print(a[0], a[1], a[2], a[3], a[-1], a[-2], a[-3], a[-4])
```

First element is: 2  
Second element is: 4  
Third element is: 5  
Forth element is: 6  
Last element is: 6  
Second last element is: 5  
Third last element is: 4  
Forth last element is: 2  
2 4 5 6 6 5 4 2

Arrays are mutable, and their elements can be changed similarly to lists.

```
import array as arr

# Creating an array of integers
numbers = arr.array('i', [1, 2, 3, 5, 7, 10])

# Changing the first element (index 0) from 1 to 0
numbers[0] = 0
print(numbers) # Expected Output: array('i', [0, 2, 3, 5, 7, 10])

# Changing the last element (index 5) from 10 to 8
numbers[5] = 8
print(numbers) # Expected Output: array('i', [0, 2, 3, 5, 7, 8])

# Replacing elements from index 2 to 4 with new values [4, 6, 8]
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers) # Expected Output: array('i', [0, 2, 4, 6, 8, 8])
```

array('i', [0, 2, 3, 5, 7, 10])  
array('i', [0, 2, 3, 5, 7, 8])  
array('i', [0, 2, 4, 6, 8, 8])

The elements can be deleted from an array using Python's del statement. If we want to delete any value from the Array, we can use the indices of a particular element.

```
import array as arr # Importing the array module

# Creating an array of integers
number = arr.array('i', [1, 2, 3, 3, 4])

# Using del to remove the third element (index 2)
del number[2]

# Printing the array after the element removal
print(number) # Expected Output: array('i', [1, 2, 3, 4])
```

array('i', [1, 2, 3, 4])

## Array Concatenation

We can easily concatenate any two arrays using the + symbol.

### Example 1:

```
import array as arr # Import the array module

# Creating two arrays of type 'd' (floating point numbers)
a = arr.array('d', [1.1, 2.1, 3.1, 2.6, 7.8]) # Array a
b = arr.array('d', [3.7, 8.6]) # Array b

# Creating an empty array c
c = arr.array('d')

# Concatenating arrays a and b
c = a + b

# Printing the resulting array c
print("Array c = ", c)
```

Array c = array('d', [1.1, 2.1, 3.1, 2.6, 7.8, 3.7, 8.6])

### Example 2:

```
import array as arr # Importing the array module

# Initialize the array with integer values
x = arr.array('i', [4, 7, 19, 22])

# Accessing and printing the first element
print("First element:", x[0]) # Output: 4

# Accessing and printing the second element
print("Second element:", x[1]) # Output: 7

# Accessing and printing the second last element using negative indexing
print("Second last element:", x[-2]) # Output: 19
```

First element: 4  
Second element: 7  
Second last element: 19

# Python Decorator

## Example

```
def func1(msg): # Function definition with a parameter 'msg'
    print(msg) # Print the message passed as an argument

func1("Hii, welcome to function ") # Call func1 and pass a string as the argument

func2 = func1 # Assign func1 to func2, making func2 another reference to func1

func2("Hii, welcome to function ") # Call func2 (which references func1) and pass the same message
```

Hii, welcome to function  
Hii, welcome to function

## Inner Function

```
def func(): # Creating the outer function 'func'
    print("We are in first function") # Print message for func

    def func1(): # Creating the first inner function 'func1'
        print("This is first child function") # Print message for func1

    def func2(): # Creating the second inner function 'func2'
        print("This is second child function") # Print message for func2

    func1() # Call the first inner function 'func1'
    func2() # Call the second inner function 'func2'

    func() # Call the outer function 'func'
```

We are in first function  
This is first child function  
This is second child function

```
def add(x): # Define a function 'add' that adds 1 to the input 'x'
    return x + 1 # Return the value of 'x + 1'

def sub(x): # Define a function 'sub' that subtracts 1 from the input 'x'
    return x - 1 # Return the value of 'x - 1'

def operator(func, x): # Define a function 'operator' that takes a function and a value as parameters
    temp = func(x) # Call the passed function (add or sub) with 'x' as the argument
    return temp # Return the result of the function call

print(operator(sub, 10)) # Call 'operator' with the 'sub' function and 10, expected to return 9
print(operator(add, 20)) # Call 'operator' with the 'add' function and 20, expected to return 21
```

9  
21

**A function can return another function. Consider the below example:**

```
def hello(): # Define the outer function 'hello'
    def hi(): # Define the inner function 'hi'
        print("Hello") # The 'hi' function prints "Hello"

    return hi # Return the 'hi' function itself, not the result of calling it

new = hello() # Call 'hello', which returns the 'hi' function and store it in 'new'
new() # Call the function stored in 'new', which is actually 'hi', so it prints "Hello"
```

↔ Hello

## Decorating functions with parameters

```
def divide(x, y): # Define the function 'divide' that takes two parameters
    print(x / y) # Print the result of dividing x by y

def outer_div(func): # Define a function 'outer_div' that takes a function as a parameter
    def inner(x, y): # Define the inner function that will modify the behavior of 'func'
        if x < y: # If the first number is less than the second number, swap them
            x, y = y, x # Swap x and y
        return func(x, y) # Call the original 'func' with the modified parameters
    return inner # Return the inner function, which is a closure that wraps 'func'

# Create a new function 'divide1' by applying the 'outer_div' decorator to 'divide'
divide1 = outer_div(divide)

# Call 'divide1', which will internally call 'inner', and 'inner' will call 'divide'
divide1(2, 4)
```

↔ 2.0

## Syntactic Decorator

```
def outer_div(func): # Define a decorator 'outer_div' that takes a function 'func' as argument
    def inner(x, y): # Define the inner function that will modify the behavior of 'func'
        if x < y: # If the first number is smaller than the second, swap them
            x, y = y, x # Swap the values of x and y
        return func(x, y) # Call the original 'func' with the swapped values
    return inner # Return the 'inner' function which is a modified version of 'func'

@outer_div # Apply the 'outer_div' decorator to 'divide'
def divide(x, y): # Define the 'divide' function that takes two numbers as input
    print(x / y) # Print the result of dividing x by y
```

## Reusing Decorator

```
0s # 1. Define a decorator function
def do_twice(func): # Here, 'func' is the function that we will decorate
    # 2. Define a wrapper function to call 'func' twice
    def wrapper_do_twice():
        func() # Call the function once
        func() # Call the function again
    # 3. Return the wrapper function
    return wrapper_do_twice

# 4. Now, using the decorator:
# We can import 'do_twice' in another file, but here it's defined in the same script
# from decorator import do_twice # Assuming the decorator is in a file named 'decorator.py'

# 5. Apply the decorator to a function
@do_twice # This is the decorator syntax
def say_hello():
    print("Hello There")

# 6. Calling 'say_hello' will execute the wrapped version that calls the function twice
say_hello() # This will print "Hello There" twice
```

⌕ Hello There  
Hello There

## Fancy Decorators

### Example: 1

```
0s class Student: # here, we are creating a class with the name Student
    def __init__(self, name, grade): # Constructor to initialize the attributes
        self.name = name
        self.grade = grade

    @property # Using the property decorator for the display method
    def display(self): # Property method to get the student's name and grade
        return self.name + " got grade " + self.grade

# Create an instance of the Student class
stu = Student("John", "B")

# Accessing attributes
print("Name of the student: ", stu.name) # Prints the name of the student
print("Grade of the student: ", stu.grade) # Prints the grade of the student

# Using the property display to print the name and grade formatted
print(stu.display) # This prints the formatted string from the @property method
```

⌕ Name of the student: John  
Grade of the student: B  
John got grade B

## Example: 2-

```
✓ 0s ▶ class Person: # here, we are creating a class with the name Person
    @staticmethod
    def hello(): # here, we are defining a static method hello
        print("Hello Peter")

# Creating an instance of the Person class
per = Person()

# Calling the hello method on the instance
per.hello()

# Calling the hello method directly on the class
Person.hello()
```

⇒ Hello Peter  
Hello Peter

## Decorator with Arguments

```
✓ 0s ▶ import functools # here, we are importing the functools into our program

def repeat(num): # here, we are defining a function repeat and passing parameter num
    # Here, we are creating and returning a wrapper function
    def decorator_repeat(func):
        @functools.wraps(func) # This preserves the original function's metadata (name, docstring)
        def wrapper(*args, **kwargs):
            for _ in range(num): # here, we are initializing a for loop and iterating till num
                value = func(*args, **kwargs) # Calling the original function
            return value # here, we are returning the value
        return wrapper # here, we are returning the wrapper function
    return decorator_repeat # Return the decorator function

# Here we are passing num as an argument, which repeats the print function 5 times
@repeat(num=5) # The decorator repeats the function call 5 times
def function1(name):
    print(f"{name}") # This function prints the name

# Calling the decorated function
function1("Hello")
```

⇒ Hello  
Hello  
Hello  
Hello  
Hello

## Stateful Decorators

```
import functools # here, we are importing the functools into our program

def count_function(func):
    # here, we are defining a function and passing the parameter func
    @functools.wraps(func)
    def wrapper_count_calls(*args, **kwargs):
        wrapper_count_calls.num_calls += 1
        print(f"Call {wrapper_count_calls.num_calls} of {func.__name__!r}")
        return func(*args, **kwargs)

    # Initialize the num_calls attribute
    wrapper_count_calls.num_calls = 0
    return wrapper_count_calls # here, we are returning the wrapper count calls

@count_function # Decorator is applied here
def say_hello():
    # here, we are defining a function that prints a message
    print("Say Hello")

# Calling the decorated function
say_hello()
say_hello()
```

Call 1 of 'say\_hello'  
Say Hello  
Call 2 of 'say\_hello'  
Say Hello

## Classes as Decorators

```
import functools # here, we are importing the functools into our program

class Count_Calls:
    # here, we are creating a class for getting the call count
    def __init__(self, func):
        functools.update_wrapper(self, func)
        self.func = func
        self.num_calls = 0

    def __call__(self, *args, **kwargs):
        self.num_calls += 1
        print(f"Call {self.num_calls} of {self.func.__name__!r}")
        return self.func(*args, **kwargs)

@Count_Calls # Decorator applied here
def say_hello():
    # here, we are defining a function and passing the parameter
    print("Say Hello")

# Calling the decorated function
say_hello()
say_hello()
say_hello()
```

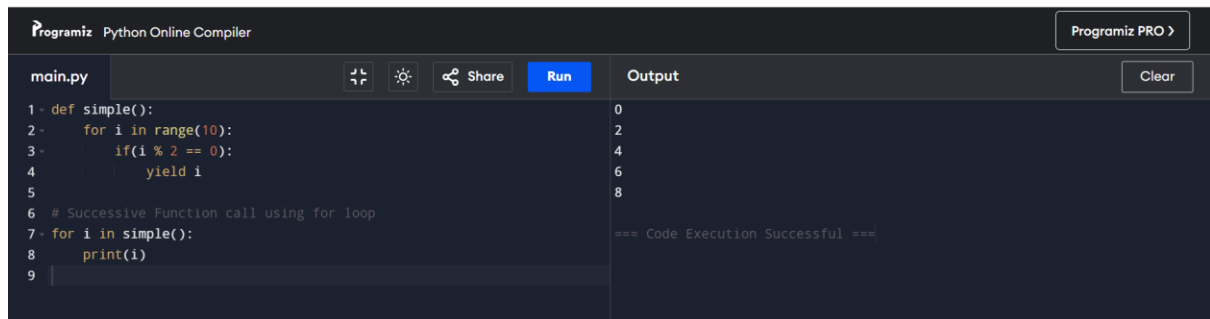
Call 1 of 'say\_hello'  
Say Hello  
Call 2 of 'say\_hello'  
Say Hello  
Call 3 of 'say\_hello'  
Say Hello



# Generator function in Python:

Examples:

## 1. Simple Generator Function



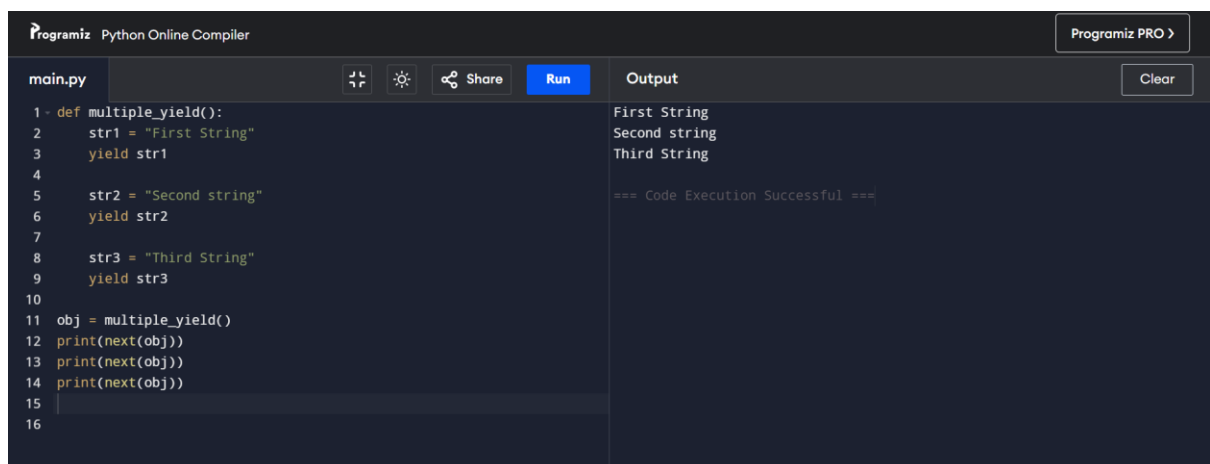
The screenshot shows the Programiz Python Online Compiler interface. The code editor on the left contains the following Python code:

```
1- def simple():
2-     for i in range(10):
3-         if(i % 2 == 0):
4-             yield i
5-
6- # Successive Function call using for loop
7- for i in simple():
8-     print(i)
9-
```

The output panel on the right displays the results of the code execution:

```
0
2
4
6
8
=== Code Execution Successful ===
```

## 2. Using Multiple yield Statements in a Generator



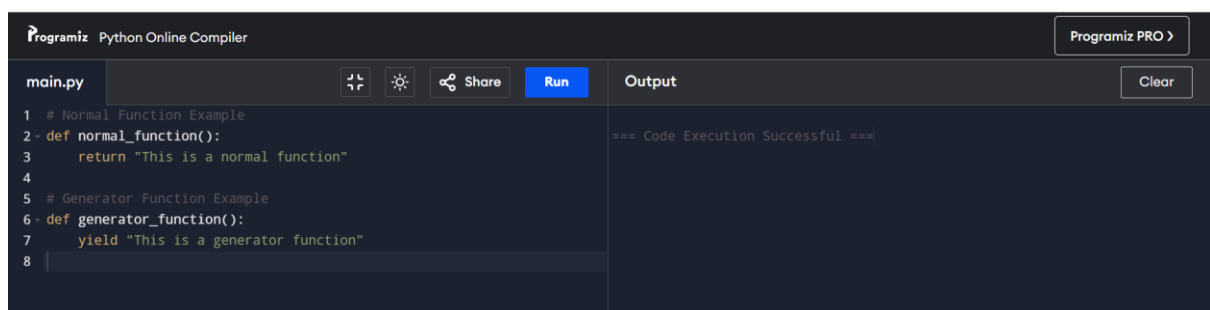
The screenshot shows the Programiz Python Online Compiler interface. The code editor on the left contains the following Python code:

```
1- def multiple_yield():
2-     str1 = "First String"
3-     yield str1
4-
5-     str2 = "Second string"
6-     yield str2
7-
8-     str3 = "Third String"
9-     yield str3
10-
11- obj = multiple_yield()
12- print(next(obj))
13- print(next(obj))
14- print(next(obj))
15-
16-
```

The output panel on the right displays the results of the code execution:

```
First String
Second string
Third String
=== Code Execution Successful ===
```

## 3. Generator vs Normal Function:



The screenshot shows the Programiz Python Online Compiler interface. The code editor on the left contains the following Python code:




```
1 # Normal Function Example
2- def normal_function():
3-     return "This is a normal function"
4-
5 # Generator Function Example
6- def generator_function():
7-     yield "This is a generator function"
8-
```

The output panel on the right displays the results of the code execution:

```
=== Code Execution Successful ===
```

## 4. Generator Expression:

Programiz Python Online Compiler Programiz PRO >

main.py    Share Run

```
1 list = [1, 2, 3, 4, 5, 6, 7]
2
3 # List Comprehension
4 z = [x**3 for x in list]
5
6 # Generator Expression
7 a = (x**3 for x in list)
8
9 print(a)
10 print(z)
11
```




Output Clear

```
<generator object <genexpr> at 0x7b02a0771d80>
[1, 8, 27, 64, 125, 216, 343]

=== Code Execution Successful ===
```

## 5. Using next() with Generator Expression

Programiz Python Online Compiler Programiz PRO >

main.py    Share Run

```
1 list = [1, 2, 3, 4, 5, 6]
2
3 z = (x**3 for x in list)
4
5 print(next(z))
6 print(next(z))
7 print(next(z))
8 print(next(z))
9
```




Output Clear

```
1
8
27
64

=== Code Execution Successful ===
```

## 6. Generating a Table Using Generator

Programiz Python Online Compiler Programiz PRO >

main.py    Share Run

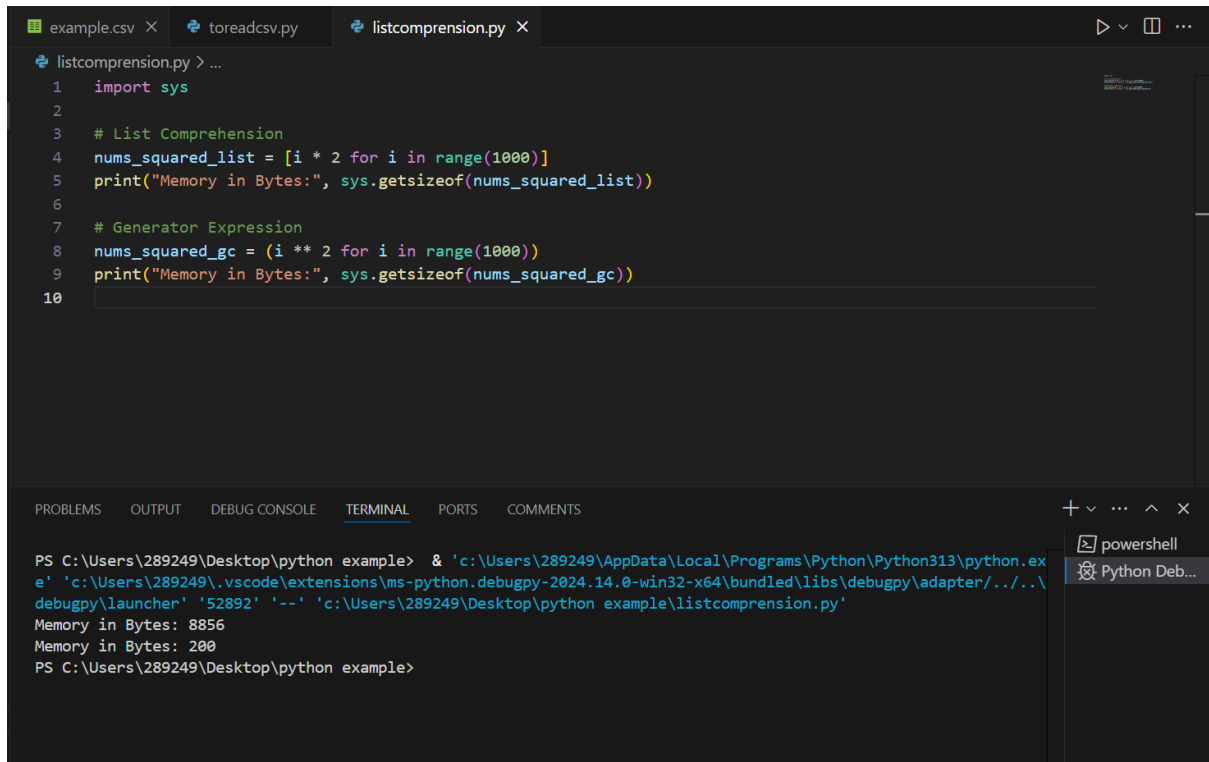
```
1 def table(n):
2     for i in range(1, 11):
3         yield n * i
4
5 for i in table(15):
6     print(i)
7
```

Output Clear

```
15
30
45
60
75
90
105
120
135
150

=== Code Execution Successful ===
```

## 7. Memory Efficiency Comparison Between List Comprehension and Generator Expression

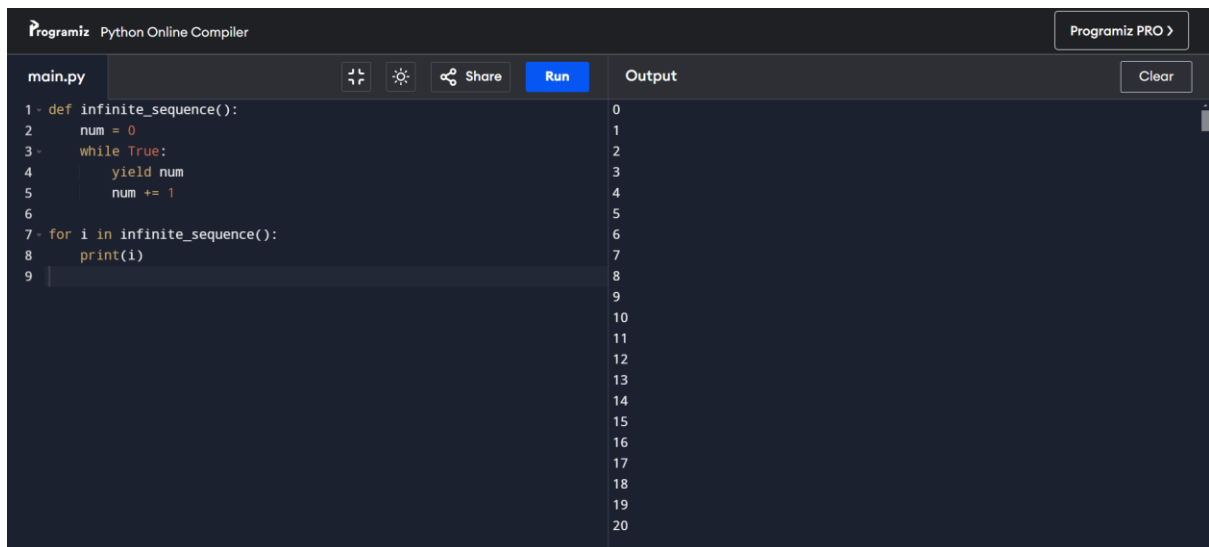


```
example.csv x toreadcsv.py listcomprehension.py x
listcomprehension.py > ...
1 import sys
2
3 # List Comprehension
4 nums_squared_list = [i * 2 for i in range(1000)]
5 print("Memory in Bytes:", sys.getsizeof(nums_squared_list))
6
7 # Generator Expression
8 nums_squared_gc = (i ** 2 for i in range(1000))
9 print("Memory in Bytes:", sys.getsizeof(nums_squared_gc))
10
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS COMMENTS

```
PS C:\Users\289249\Desktop\python example> & 'c:\Users\289249\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\289249\.vscode\extensions\ms-python.debugpy-2024.14.0-win32-x64\bundle\libs\debugpy\adapter\..\..\debugpy\launcher' '52892' '--' 'c:\Users\289249\Desktop\python example\listcomprehension.py'
Memory in Bytes: 8856
Memory in Bytes: 200
PS C:\Users\289249\Desktop\python example>
```

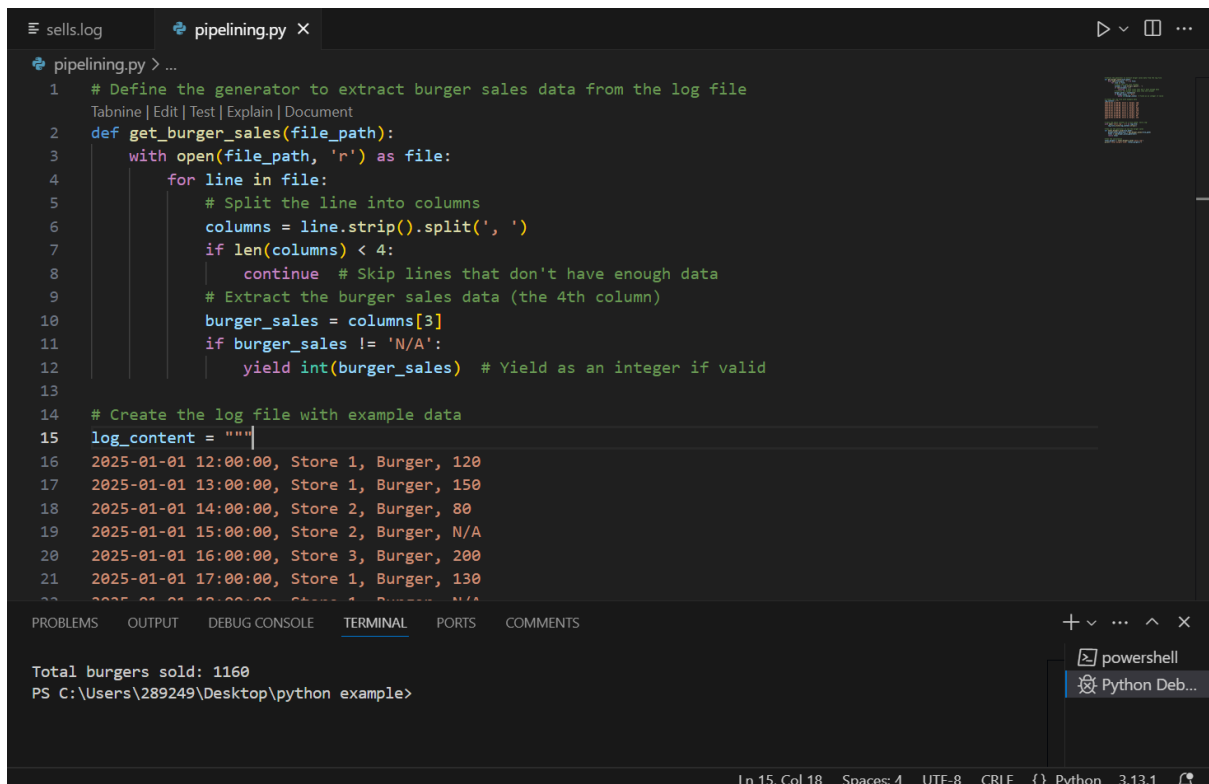
## 8. Generating Infinite Sequence Using Generator:



```
Programiz Python Online Compiler
main.py Run Output Clear
1 def infinite_sequence():
2     num = 0
3     while True:
4         yield num
5         num += 1
6
7 for i in infinite_sequence():
8     print(i)
9
```

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20

## 9. Using Generator for Pipelining with Data



The screenshot shows a Visual Studio Code editor with a file named `pipelining.py` open. The script defines a generator function `get_burger_sales` that reads a log file and yields burger sales data. It also creates a sample log file with 10 lines of data. The terminal output shows the total burgers sold as 1160.

```
1 # Define the generator to extract burger sales data from the log file
2 def get_burger_sales(file_path):
3     with open(file_path, 'r') as file:
4         for line in file:
5             # Split the line into columns
6             columns = line.strip().split(',')
7             if len(columns) < 4:
8                 continue # Skip lines that don't have enough data
9             # Extract the burger sales data (the 4th column)
10            burger_sales = columns[3]
11            if burger_sales != 'N/A':
12                yield int(burger_sales) # Yield as an integer if valid
13
14 # Create the log file with example data
15 log_content = """
16 2025-01-01 12:00:00, Store 1, Burger, 120
17 2025-01-01 13:00:00, Store 1, Burger, 150
18 2025-01-01 14:00:00, Store 2, Burger, 80
19 2025-01-01 15:00:00, Store 2, Burger, N/A
20 2025-01-01 16:00:00, Store 3, Burger, 200
21 2025-01-01 17:00:00, Store 1, Burger, 130
22 2025-01-01 18:00:00, Store 1, Burger, N/A
23 2025-01-01 19:00:00, Store 2, Burger, 100
24 2025-01-01 20:00:00, Store 3, Burger, 180
25 2025-01-01 21:00:00, Store 1, Burger, 110
26 2025-01-01 22:00:00, Store 2, Burger, 90
27 2025-01-01 23:00:00, Store 3, Burger, 160
28 2025-01-02 00:00:00, Store 1, Burger, 140
29 2025-01-02 01:00:00, Store 2, Burger, 70
30 2025-01-02 02:00:00, Store 3, Burger, 190
31 2025-01-02 03:00:00, Store 1, Burger, 120
32 2025-01-02 04:00:00, Store 2, Burger, 100
33 2025-01-02 05:00:00, Store 3, Burger, 170
34 2025-01-02 06:00:00, Store 1, Burger, 150
35 2025-01-02 07:00:00, Store 2, Burger, 80
36 2025-01-02 08:00:00, Store 3, Burger, 210
37 2025-01-02 09:00:00, Store 1, Burger, 130
38 2025-01-02 10:00:00, Store 2, Burger, 90
39 2025-01-02 11:00:00, Store 3, Burger, 180
40 2025-01-02 12:00:00, Store 1, Burger, 110
41 2025-01-02 13:00:00, Store 2, Burger, 100
42 2025-01-02 14:00:00, Store 3, Burger, 160
43 2025-01-02 15:00:00, Store 1, Burger, 140
44 2025-01-02 16:00:00, Store 2, Burger, 70
45 2025-01-02 17:00:00, Store 3, Burger, 190
46 2025-01-02 18:00:00, Store 1, Burger, 120
47 2025-01-02 19:00:00, Store 2, Burger, 100
48 2025-01-02 20:00:00, Store 3, Burger, 170
49 2025-01-02 21:00:00, Store 1, Burger, 150
50 2025-01-02 22:00:00, Store 2, Burger, 80
51 2025-01-02 23:00:00, Store 3, Burger, 210
52 2025-01-03 00:00:00, Store 1, Burger, 130
53 2025-01-03 01:00:00, Store 2, Burger, 90
54 2025-01-03 02:00:00, Store 3, Burger, 180
55 2025-01-03 03:00:00, Store 1, Burger, 110
56 2025-01-03 04:00:00, Store 2, Burger, 100
57 2025-01-03 05:00:00, Store 3, Burger, 160
58 2025-01-03 06:00:00, Store 1, Burger, 140
59 2025-01-03 07:00:00, Store 2, Burger, 70
60 2025-01-03 08:00:00, Store 3, Burger, 190
61 2025-01-03 09:00:00, Store 1, Burger, 120
62 2025-01-03 10:00:00, Store 2, Burger, 100
63 2025-01-03 11:00:00, Store 3, Burger, 170
64 2025-01-03 12:00:00, Store 1, Burger, 150
65 2025-01-03 13:00:00, Store 2, Burger, 80
66 2025-01-03 14:00:00, Store 3, Burger, 210
67 2025-01-03 15:00:00, Store 1, Burger, 130
68 2025-01-03 16:00:00, Store 2, Burger, 90
69 2025-01-03 17:00:00, Store 3, Burger, 180
70 2025-01-03 18:00:00, Store 1, Burger, 110
71 2025-01-03 19:00:00, Store 2, Burger, 100
72 2025-01-03 20:00:00, Store 3, Burger, 160
73 2025-01-03 21:00:00, Store 1, Burger, 140
74 2025-01-03 22:00:00, Store 2, Burger, 70
75 2025-01-03 23:00:00, Store 3, Burger, 190
76 2025-01-04 00:00:00, Store 1, Burger, 120
77 2025-01-04 01:00:00, Store 2, Burger, 100
78 2025-01-04 02:00:00, Store 3, Burger, 170
79 2025-01-04 03:00:00, Store 1, Burger, 150
80 2025-01-04 04:00:00, Store 2, Burger, 80
81 2025-01-04 05:00:00, Store 3, Burger, 210
82 2025-01-04 06:00:00, Store 1, Burger, 130
83 2025-01-04 07:00:00, Store 2, Burger, 90
84 2025-01-04 08:00:00, Store 3, Burger, 180
85 2025-01-04 09:00:00, Store 1, Burger, 110
86 2025-01-04 10:00:00, Store 2, Burger, 100
87 2025-01-04 11:00:00, Store 3, Burger, 160
88 2025-01-04 12:00:00, Store 1, Burger, 140
89 2025-01-04 13:00:00, Store 2, Burger, 70
90 2025-01-04 14:00:00, Store 3, Burger, 190
91 2025-01-04 15:00:00, Store 1, Burger, 120
92 2025-01-04 16:00:00, Store 2, Burger, 100
93 2025-01-04 17:00:00, Store 3, Burger, 170
94 2025-01-04 18:00:00, Store 1, Burger, 150
95 2025-01-04 19:00:00, Store 2, Burger, 80
96 2025-01-04 20:00:00, Store 3, Burger, 210
97 2025-01-04 21:00:00, Store 1, Burger, 130
98 2025-01-04 22:00:00, Store 2, Burger, 90
99 2025-01-04 23:00:00, Store 3, Burger, 180
100 2025-01-05 00:00:00, Store 1, Burger, 110
101 2025-01-05 01:00:00, Store 2, Burger, 100
102 2025-01-05 02:00:00, Store 3, Burger, 160
103 2025-01-05 03:00:00, Store 1, Burger, 140
104 2025-01-05 04:00:00, Store 2, Burger, 70
105 2025-01-05 05:00:00, Store 3, Burger, 190
106 2025-01-05 06:00:00, Store 1, Burger, 120
107 2025-01-05 07:00:00, Store 2, Burger, 100
108 2025-01-05 08:00:00, Store 3, Burger, 170
109 2025-01-05 09:00:00, Store 1, Burger, 150
110 2025-01-05 10:00:00, Store 2, Burger, 80
111 2025-01-05 11:00:00, Store 3, Burger, 210
112 2025-01-05 12:00:00, Store 1, Burger, 130
113 2025-01-05 13:00:00, Store 2, Burger, 90
114 2025-01-05 14:00:00, Store 3, Burger, 180
115 2025-01-05 15:00:00, Store 1, Burger, 110
116 2025-01-05 16:00:00, Store 2, Burger, 100
117 2025-01-05 17:00:00, Store 3, Burger, 160
118 2025-01-05 18:00:00, Store 1, Burger, 140
119 2025-01-05 19:00:00, Store 2, Burger, 70
120 2025-01-05 20:00:00, Store 3, Burger, 190
121 2025-01-05 21:00:00, Store 1, Burger, 120
122 2025-01-05 22:00:00, Store 2, Burger, 100
123 2025-01-05 23:00:00, Store 3, Burger, 170
124 2025-01-06 00:00:00, Store 1, Burger, 150
125 2025-01-06 01:00:00, Store 2, Burger, 80
126 2025-01-06 02:00:00, Store 3, Burger, 210
127 2025-01-06 03:00:00, Store 1, Burger, 130
128 2025-01-06 04:00:00, Store 2, Burger, 90
129 2025-01-06 05:00:00, Store 3, Burger, 180
130 2025-01-06 06:00:00, Store 1, Burger, 110
131 2025-01-06 07:00:00, Store 2, Burger, 100
132 2025-01-06 08:00:00, Store 3, Burger, 160
133 2025-01-06 09:00:00, Store 1, Burger, 140
134 2025-01-06 10:00:00, Store 2, Burger, 70
135 2025-01-06 11:00:00, Store 3, Burger, 190
136 2025-01-06 12:00:00, Store 1, Burger, 120
137 2025-01-06 13:00:00, Store 2, Burger, 100
138 2025-01-06 14:00:00, Store 3, Burger, 170
139 2025-01-06 15:00:00, Store 1, Burger, 150
140 2025-01-06 16:00:00, Store 2, Burger, 80
141 2025-01-06 17:00:00, Store 3, Burger, 210
142 2025-01-06 18:00:00, Store 1, Burger, 130
143 2025-01-06 19:00:00, Store 2, Burger, 90
144 2025-01-06 20:00:00, Store 3, Burger, 180
145 2025-01-06 21:00:00, Store 1, Burger, 110
146 2025-01-06 22:00:00, Store 2, Burger, 100
147 2025-01-06 23:00:00, Store 3, Burger, 160
148 2025-01-07 00:00:00, Store 1, Burger, 140
149 2025-01-07 01:00:00, Store 2, Burger, 70
150 2025-01-07 02:00:00, Store 3, Burger, 190
151 2025-01-07 03:00:00, Store 1, Burger, 120
152 2025-01-07 04:00:00, Store 2, Burger, 100
153 2025-01-07 05:00:00, Store 3, Burger, 170
154 2025-01-07 06:00:00, Store 1, Burger, 150
155 2025-01-07 07:00:00, Store 2, Burger, 80
156 2025-01-07 08:00:00, Store 3, Burger, 210
157 2025-01-07 09:00:00, Store 1, Burger, 130
158 2025-01-07 10:00:00, Store 2, Burger, 90
159 2025-01-07 11:00:00, Store 3, Burger, 180
160 2025-01-07 12:00:00, Store 1, Burger, 110
161 2025-01-07 13:00:00, Store 2, Burger, 100
162 2025-01-07 14:00:00, Store 3, Burger, 160
163 2025-01-07 15:00:00, Store 1, Burger, 140
164 2025-01-07 16:00:00, Store 2, Burger, 70
165 2025-01-07 17:00:00, Store 3, Burger, 190
166 2025-01-07 18:00:00, Store 1, Burger, 120
167 2025-01-07 19:00:00, Store 2, Burger, 100
168 2025-01-07 20:00:00, Store 3, Burger, 170
169 2025-01-07 21:00:00, Store 1, Burger, 150
170 2025-01-07 22:00:00, Store 2, Burger, 80
171 2025-01-07 23:00:00, Store 3, Burger, 210
172 2025-01-08 00:00:00, Store 1, Burger, 130
173 2025-01-08 01:00:00, Store 2, Burger, 90
174 2025-01-08 02:00:00, Store 3, Burger, 180
175 2025-01-08 03:00:00, Store 1, Burger, 110
176 2025-01-08 04:00:00, Store 2, Burger, 100
177 2025-01-08 05:00:00, Store 3, Burger, 160
178 2025-01-08 06:00:00, Store 1, Burger, 140
179 2025-01-08 07:00:00, Store 2, Burger, 70
180 2025-01-08 08:00:00, Store 3, Burger, 190
181 2025-01-08 09:00:00, Store 1, Burger, 120
182 2025-01-08 10:00:00, Store 2, Burger, 100
183 2025-01-08 11:00:00, Store 3, Burger, 170
184 2025-01-08 12:00:00, Store 1, Burger, 150
185 2025-01-08 13:00:00, Store 2, Burger, 80
186 2025-01-08 14:00:00, Store 3, Burger, 210
187 2025-01-08 15:00:00, Store 1, Burger, 130
188 2025-01-08 16:00:00, Store 2, Burger, 90
189 2025-01-08 17:00:00, Store 3, Burger, 180
190 2025-01-08 18:00:00, Store 1, Burger, 110
191 2025-01-08 19:00:00, Store 2, Burger, 100
192 2025-01-08 20:00:00, Store 3, Burger, 160
193 2025-01-08 21:00:00, Store 1, Burger, 140
194 2025-01-08 22:00:00, Store 2, Burger, 70
195 2025-01-08 23:00:00, Store 3, Burger, 190
196 2025-01-09 00:00:00, Store 1, Burger, 120
197 2025-01-09 01:00:00, Store 2, Burger, 100
198 2025-01-09 02:00:00, Store 3, Burger, 170
199 2025-01-09 03:00:00, Store 1, Burger, 150
200 2025-01-09 04:00:00, Store 2, Burger, 80
201 2025-01-09 05:00:00, Store 3, Burger, 210
202 2025-01-09 06:00:00, Store 1, Burger, 130
203 2025-01-09 07:00:00, Store 2, Burger, 90
204 2025-01-09 08:00:00, Store 3, Burger, 180
205 2025-01-09 09:00:00, Store 1, Burger, 110
206 2025-01-09 10:00:00, Store 2, Burger, 100
207 2025-01-09 11:00:00, Store 3, Burger, 160
208 2025-01-09 12:00:00, Store 1, Burger, 140
209 2025-01-09 13:00:00, Store 2, Burger, 70
210 2025-01-09 14:00:00, Store 3, Burger, 190
211 2025-01-09 15:00:00, Store 1, Burger, 120
212 2025-01-09 16:00:00, Store 2, Burger, 100
213 2025-01-09 17:00:00, Store 3, Burger, 170
214 2025-01-09 18:00:00, Store 1, Burger, 150
215 2025-01-09 19:00:00, Store 2, Burger, 80
216 2025-01-09 20:00:00, Store 3, Burger, 210
217 2025-01-09 21:00:00, Store 1, Burger, 130
218 2025-01-09 22:00:00, Store 2, Burger, 90
219 2025-01-09 23:00:00, Store 3, Burger, 180
220 2025-01-10 00:00:00, Store 1, Burger, 110
221 2025-01-10 01:00:00, Store 2, Burger, 100
222 2025-01-10 02:00:00, Store 3, Burger, 160
223 2025-01-10 03:00:00, Store 1, Burger, 140
224 2025-01-10 04:00:00, Store 2, Burger, 70
225 2025-01-10 05:00:00, Store 3, Burger, 190
226 2025-01-10 06:00:00, Store 1, Burger, 120
227 2025-01-10 07:00:00, Store 2, Burger, 100
228 2025-01-10 08:00:00, Store 3, Burger, 170
229 2025-01-10 09:00:00, Store 1, Burger, 150
230 2025-01-10 10:00:00, Store 2, Burger, 80
231 2025-01-10 11:00:00, Store 3, Burger, 210
232 2025-01-10 12:00:00, Store 1, Burger, 130
233 2025-01-10 13:00:00, Store 2, Burger, 90
234 2025-01-10 14:00:00, Store 3, Burger, 180
235 2025-01-10 15:00:00, Store 1, Burger, 110
236 2025-01-10 16:00:00, Store 2, Burger, 100
237 2025-01-10 17:00:00, Store 3, Burger, 160
238 2025-01-10 18:00:00, Store 1, Burger, 140
239 2025-01-10 19:00:00, Store 2, Burger, 70
240 2025-01-10 20:00:00, Store 3, Burger, 190
241 2025-01-10 21:00:00, Store 1, Burger, 120
242 2025-01-10 22:00:00, Store 2, Burger, 100
243 2025-01-10 23:00:00, Store 3, Burger, 170
244 2025-01-11 00:00:00, Store 1, Burger, 150
245 2025-01-11 01:00:00, Store 2, Burger, 80
246 2025-01-11 02:00:00, Store 3, Burger, 210
247 2025-01-11 03:00:00, Store 1, Burger, 130
248 2025-01-11 04:00:00, Store 2, Burger, 90
249 2025-01-11 05:00:00, Store 3, Burger, 180
250 2025-01-11 06:00:00, Store 1, Burger, 110
251 2025-01-11 07:00:00, Store 2, Burger, 100
252 2025-01-11 08:00:00, Store 3, Burger, 160
253 2025-01-11 09:00:00, Store 1, Burger, 140
254 2025-01-11 10:00:00, Store 2, Burger, 70
255 2025-01-11 11:00:00, Store 3, Burger, 190
256 2025-01-11 12:00:00, Store 1, Burger, 120
257 2025-01-11 13:00:00, Store 2, Burger, 100
258 2025-01-11 14:00:00, Store 3, Burger, 170
259 2025-01-11 15:00:00, Store 1, Burger, 150
260 2025-01-11 16:00:00, Store 2, Burger, 80
261 2025-01-11 17:00:00, Store 3, Burger, 210
262 2025-01-11 18:00:00, Store 1, Burger, 130
263 2025-01-11 19:00:00, Store 2, Burger, 90
264 2025-01-11 20:00:00, Store 3, Burger, 180
265 2025-01-11 21:00:00, Store 1, Burger, 110
266 2025-01-11 22:00:00, Store 2, Burger, 100
267 2025-01-11 23:00:00, Store 3, Burger, 160
268 2025-01-12 00:00:00, Store 1, Burger, 140
269 2025-01-12 01:00:00, Store 2, Burger, 70
270 2025-01-12 02:00:00, Store 3, Burger, 190
271 2025-01-12 03:00:00, Store 1, Burger, 120
272 2025-01-12 04:00:00, Store 2, Burger, 100
273 2025-01-12 05:00:00, Store 3, Burger, 170
274 2025-01-12 06:00:00, Store 1, Burger, 150
275 2025-01-12 07:00:00, Store 2, Burger, 80
276 2025-01-12 08:00:00, Store 3, Burger, 210
277 2025-01-12 09:00:00, Store 1, Burger, 130
278 2025-01-12 10:00:00, Store 2, Burger, 90
279 2025-01-12 11:00:00, Store 3, Burger, 180
280 2025-01-12 12:00:00, Store 1, Burger, 110
281 2025-01-12 13:00:00, Store 2, Burger, 100
282 2025-01-12 14:00:00, Store 3, Burger, 160
283 2025-01-12 15:00:00, Store 1, Burger, 140
284 2025-01-12 16:00:00, Store 2, Burger, 70
285 2025-01-12 17:00:00, Store 3, Burger, 190
286 2025-01-12 18:00:00, Store 1, Burger, 120
287 2025-01-12 19:00:00, Store 2, Burger, 100
288 2025-01-12 20:00:00, Store 3, Burger, 170
289 2025-01-12 21:00:00, Store 1, Burger, 150
290 2025-01-12 22:00:00, Store 2, Burger, 80
291 2025-01-12 23:00:00, Store 3, Burger, 210
292 2025-01-13 00:00:00, Store 1, Burger, 130
293 2025-01-13 01:00:00, Store 2, Burger, 90
294 2025-01-13 02:00:00, Store 3, Burger, 180
295 2025-01-13 03:00:00, Store 1, Burger, 110
296 2025-01-13 04:00:00, Store 2, Burger, 100
297 2025-01-13 05:00:00, Store 3, Burger, 160
298 2025-01-13 06:00:00, Store 1, Burger, 140
299 2025-01-13 07:00:00, Store 2, Burger, 70
300 2025-01-13 08:00:00, Store 3, Burger, 190
301 2025-01-13 09:00:00, Store 1, Burger, 120
302 2025-01-13 10:00:00, Store 2, Burger, 100
303 2025-01-13 11:00:00, Store 3, Burger, 170
304 2025-01-13 12:00:00, Store 1, Burger, 150
305 2025-01-13 13:00:00, Store 2, Burger, 80
306 2025-01-13 14:00:00, Store 3, Burger, 210
307 2025-01-13 15:00:00, Store 1, Burger, 130
308 2025-01-13 16:00:00, Store 2, Burger, 90
309 2025-01-13 17:00:00, Store 3, Burger, 180
310 2025-01-13 18:00:00, Store 1, Burger, 110
311 2025-01-13 19:00:00, Store 2, Burger, 100
312 2025-01-13 20:00:00, Store 3, Burger, 160
313 2025-01-13 21:00:00, Store 1, Burger, 140
314 2025-01-13 22:00:00, Store 2, Burger, 70
315 2025-01-13 23:00:00, Store 3, Burger, 190
316 2025-01-14 00:00:00, Store 1, Burger, 120
317 2025-01-14 01:00:00, Store 2, Burger, 100
318 2025-01-14 02:00:00, Store 3, Burger, 170
319 2025-01-14 03:00:00, Store 1, Burger, 150
320 2025-01-14 04:00:00, Store 2, Burger, 80
321 2025-01-14 05:00:00, Store 3, Burger, 210
322 2025-01-14 06:00:00, Store 1, Burger, 130
323 2025-01-14 07:00:00, Store 2, Burger, 90
324 2025-01-14 08:00:00, Store 3, Burger, 180
325 2025-01-14 09:00:00, Store 1, Burger, 110
326 2025-01-14 10:00:00, Store 2, Burger, 100
327 2025-01-14 11:00:00, Store 3, Burger, 160
328 2025-01-14 12:00:00, Store 1, Burger, 140
329 2025-01-14 13:00:00, Store 2, Burger, 70
330 2025-01-14 14:00:00, Store 3, Burger, 190
331 2025-01-14 15:00:00, Store 1, Burger, 120
332 2025-01-14 16:00:00, Store 2, Burger, 100
333 2025-01-14 17:00:00, Store 3, Burger, 170
334 2025-01-14 18:00:00, Store 1, Burger, 150
335 2025-01-14 19:00:00, Store 2, Burger, 80
336 2025-01-14 20:00:00, Store 3, Burger, 210
337 2025-01-14 21:00:00, Store 1, Burger, 130
338 2025-01-14 22:00:00, Store 2, Burger, 90
339 2025-01-14 23:00:00, Store 3, Burger, 180
340 2025-01-15 00:00:00, Store 1, Burger, 110
341 2025-01-15 01:00:00, Store 2, Burger, 100
342 2025-01-15 02:00:00, Store 3, Burger, 160
343 2025-01-15 03:00:00, Store 1, Burger, 140
344 2025-01-15 04:00:00, Store 2, Burger, 70
345 2025-01-15 05:00:00, Store 3, Burger, 190
346 2025-01-15 06:00:00, Store 1, Burger, 120
347 2025-01-15 07:00:00, Store 2, Burger, 100
348 2025-01-15 08:00:00, Store 3, Burger, 170
349 2025-01-15 09:00:00, Store 1, Burger, 150
350 2025-01-15 10:00:00, Store 2, Burger, 80
351 2025-01-15 11:00:00, Store 3, Burger, 210
352 2025-01-15 12:00:00, Store 1, Burger, 130
353 2025-01-15 13:00:00, Store 2, Burger, 90
354 2025-01-15 14:00:00, Store 3, Burger, 180
355 2025-01-15 15:00:00, Store 1, Burger, 110
356 2025-01-15 16:00:00, Store 2, Burger, 100
357 2025-01-15 17:00:00, Store 3, Burger, 160
358 2025-01-15 18:00:00, Store 1, Burger, 140
359 2025-01-15 19:00:00, Store 2, Burger, 70
360 2025-01-15 20:00:00, Store 3, Burger, 190
361 2025-01-15 21:00:00, Store 1, Burger, 120
362 2025-01-15 22:00:00, Store 2, Burger, 100
363 2025-01-15 23:0
```