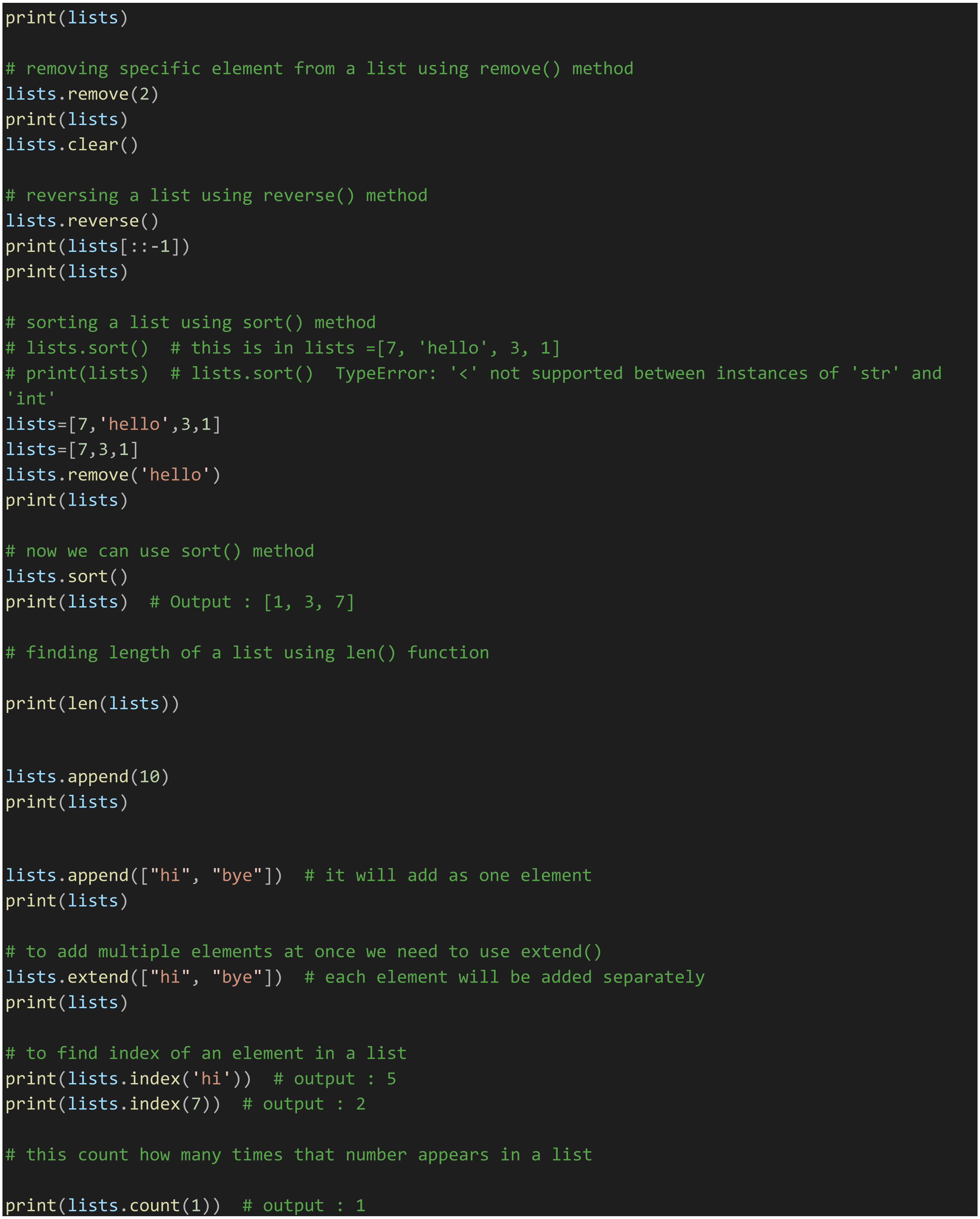
List Methods

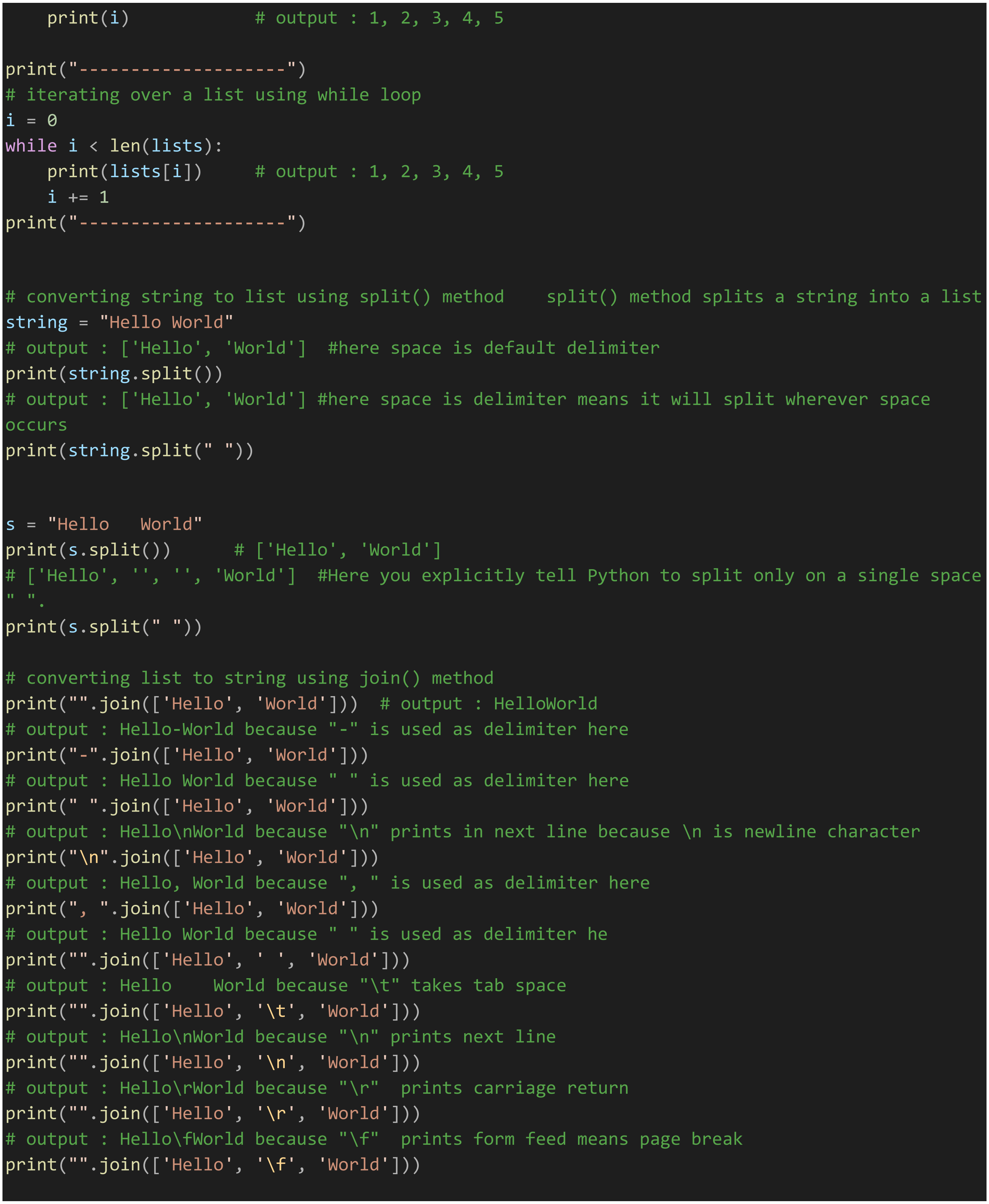
```python
# list is first in first out data structure
# creating List

import copy
lists = [1, 2, 3]
print(lists)

range_method = range(5)
range2 = list(range(lists))
print(range_method)
print(range2)

for i in lists:
    print(i)

# creating list using iterables

list_range = range(5)  # range(0,5)
print(list_range)

list_range = list(range(5))  # [0,1,2,3,4]
print(list_range)


# accessing elements from the list
lists = [1, 2, 'hello', 3]
element = lists[0], lists[1], lists[2]  # but this creates tuples
single_element = lists[0]      # single element is int type
print(element)  # output : (1, 2, 3)
print(type(element))  # output : tuple
print(single_element)  # output : 1
print(type(single_element))  # output : int

# adding elements to a list using append method and extend method

lists.append("hello")
print(lists)

lists.append(2)
lists.extend([7, 'hello'])
print(lists)

# removing last element from a list using pop() method

lists.pop()
```



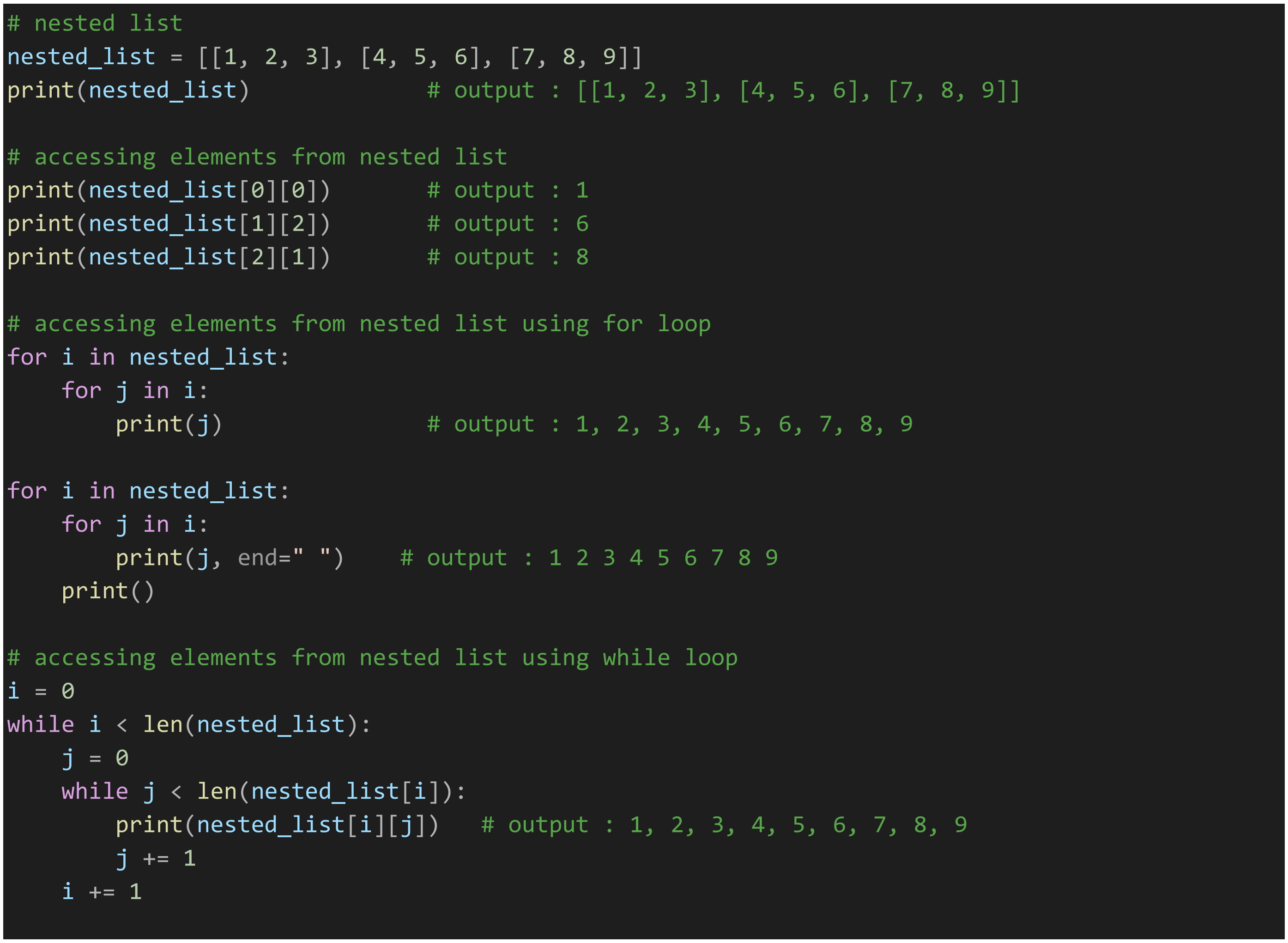
```
print(lists)

# removing specific element from a list using remove() method
lists.remove(2)
print(lists)
lists.clear()

# reversing a list using reverse() method
lists.reverse()
print(lists[::-1])
print(lists)

# sorting a list using sort() method
# lists.sort()  # this is in lists =[7, 'hello', 3, 1]
# print(lists)  # lists.sort()  TypeError: '<' not supported between instances of 'str' and
'int'
lists=[7,'hello',3,1]
lists=[7,3,1]
lists.remove('hello')
print(lists)

# now we can use sort() method
lists.sort()
print(lists)  # Output : [1, 3, 7]

# finding length of a list using len() function

print(len(lists))


lists.append(10)
print(lists)


lists.append(["hi", "bye"])  # it will add as one element
print(lists)

# to add multiple elements at once we need to use extend()
lists.extend(["hi", "bye"])  # each element will be added separately
print(lists)

# to find index of an element in a list
print(lists.index('hi'))  # output : 5
print(lists.index(7))  # output : 2

# this count how many times that number appears in a list

print(lists.count(1))  # output : 1
```

```python
print(lists.count(10))  # output : 1

# copy a list into another list using copy() method
new_lists = lists.copy()
print(new_lists)
reverse = lists.reverse()

# list.copy() → creates a new list and returns it. It does not modify the existing list.
# list.reverse() → reverses the list in place and returns None.
print(reverse)  # returns None because there is no return value for reverse()
print(lists)  # returns reversed list because it changes original list also


# clear all elements from a list using clear() method
lists.clear()
print(lists)

# create a list with same values repeated n times using * operator
a = [1]*5
print(a)  # output : [1, 1, 1, 1, 1]


b = ['hi']*5
print(b)  # output : ['hi', 'hi', 'hi', 'hi', 'hi']

c = [[1]]*5
print(c)  # output : [[1], [1], [1], [1], [1]]
# this is wrong way to create a list with same values repeated n times because it will create
a reference to the same object
# creating like this  [[1]]*5 means [[1],[1],[1],[1],[1]] which is wrong
# because element inside list like this [1]] is mutable so if you change any element then
other elements will also get changed
d = [[1]*5]*5


# output : [[1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1,
1]]
print(d)
e = [[i]*5 for i in range(5)]
# output : [[0, 0, 0, 0, 0], [1 , 1, 1, 1, 1], [2, 2, 2, 2, 2], [3, 3, 3, 3, 3], [4, 4, 4, 4,
4]]
print(e)


# checks whether 3 is present in the list or not
is_present = 3 in [1, 2, 3, 4, 5]
print("Check if 3 in list:", is_present)
```

```
max_val = max([1, 2, 3, 4, 5])   # prints maximum value from the list
min_val = min([1, 2, 3, 4, 5])


print("Max value:", max_val, "Min value:", min_val)

total = sum([1, 2, 3, 4, 5])
print("Sum of list:", total)

# slicing a list
print(lists[:])     # output : [] because lists is empty
print(lists[::-1])   # output : []

lists = list(range(1, 6))
print(lists)         # output : [1, 2, 3, 4, 5]
# output : [1, 3, 5]           because it starts from 0 and goes till end by skipping every
second element
print(lists[::2])
# output : [5, 4, 3, 2, 1]   because it starts from -1 and goes till start
print(lists[-1::-1])
# output : []                 it is empty  because it starts from start and goes to last
directly not from right side but from left side
print(lists[:-1:-1])
print(lists[::-1])   # output : [5, 4, 3, 2, 1]     printing reversed list
# output : [5, 3, 1]              printing reversed list by skipping every second element
print(lists[::-2])


# concatenating two lists using + operator
lists = list(range(1, 6))
print(lists)          # output : [1, 2, 3, 4, 5]
print(lists+lists)    # output : [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
print(lists*2)        # output : [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

# checking whether an element exists in a list or not using in keyword
print(1 in lists)       # output : True
print(10 in lists)      # output : False

my_list = ["apple", "banana", "cherry"]
print("Original list:", my_list)
shallow_copy = copy.copy(my_list)
deep_copy = copy.deepcopy(my_list)
print("Shallow copy:", shallow_copy)
print("Deep copy:", deep_copy)

print("--------------------")
# iterating over a list using for loop
for i in lists:
```



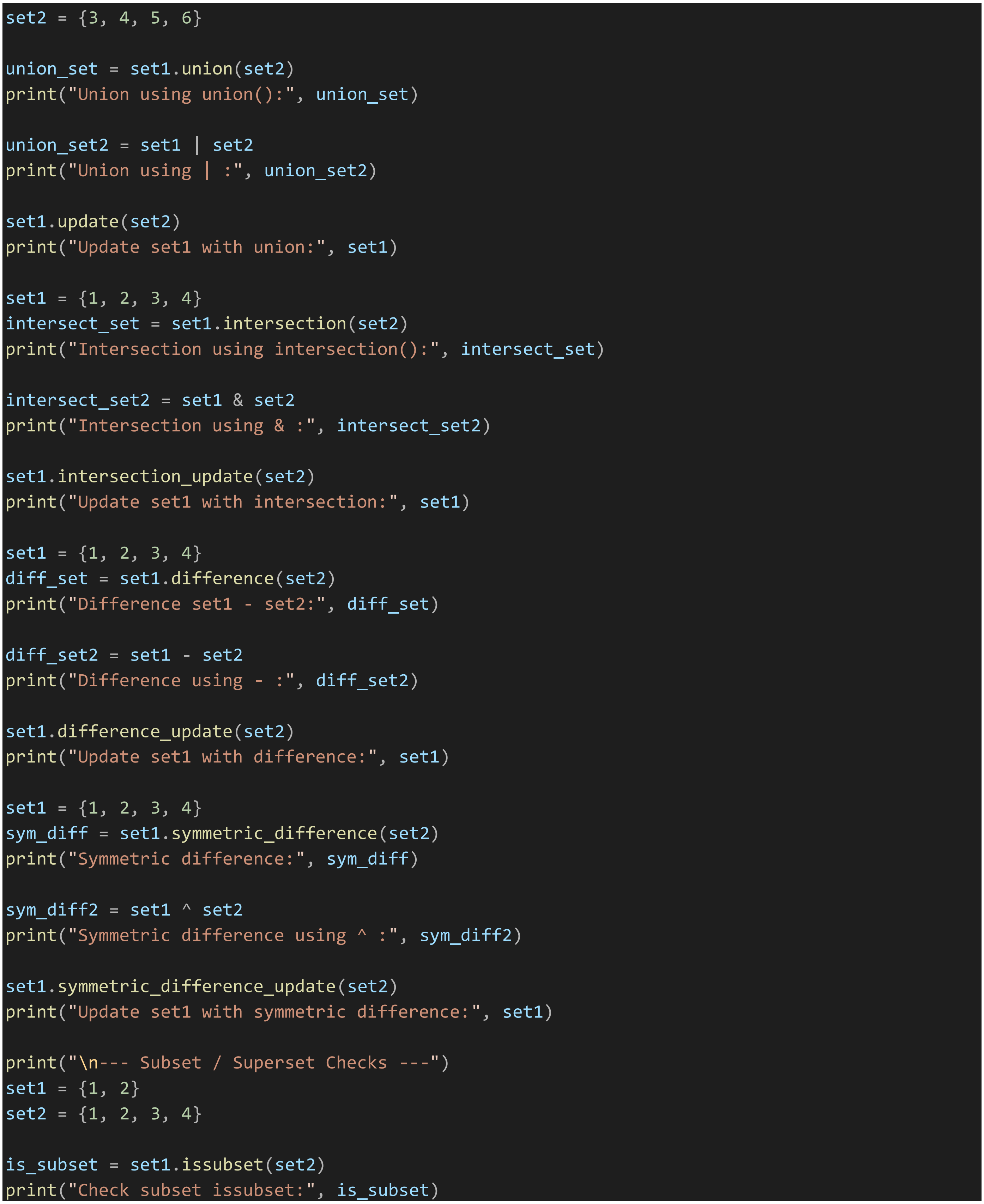
```
    print(i)            # output : 1, 2, 3, 4, 5

print("--------------------")
# iterating over a list using while loop
i = 0
while i < len(lists):
    print(lists[i])     # output : 1, 2, 3, 4, 5
    i += 1
print("--------------------")


# converting string to list using split() method    split() method splits a string into a list
string = "Hello World"
# output : ['Hello', 'World']  #here space is default delimiter
print(string.split())
# output : ['Hello', 'World'] #here space is delimiter means it will split wherever space
occurs
print(string.split(" "))


s = "Hello   World"
print(s.split())      # ['Hello', 'World']
# ['Hello', '', '', 'World']  #Here you explicitly tell Python to split only on a single space
" ".
print(s.split(" "))

# converting list to string using join() method
print("".join(['Hello', 'World']))  # output : HelloWorld
# output : Hello-World because "-" is used as delimiter here
print("-".join(['Hello', 'World']))
# output : Hello World because " " is used as delimiter here
print(" ".join(['Hello', 'World']))
# output : Hello\nWorld because "\n" prints in next line because \n is newline character
print("\n".join(['Hello', 'World']))
# output : Hello, World because ", " is used as delimiter here
print(", ".join(['Hello', 'World']))
# output : Hello World because " " is used as delimiter he
print("".join(['Hello', ' ', 'World']))
# output : Hello    World because "\t" takes tab space
print("".join(['Hello', '\t', 'World']))
# output : Hello\nWorld because "\n" prints next line
print("".join(['Hello', '\n', 'World']))
# output : Hello\rWorld because "\r"  prints carriage return
print("".join(['Hello', '\r', 'World']))
# output : Hello\fWorld because "\f"  prints form feed means page break
print("".join(['Hello', '\f', 'World']))
```



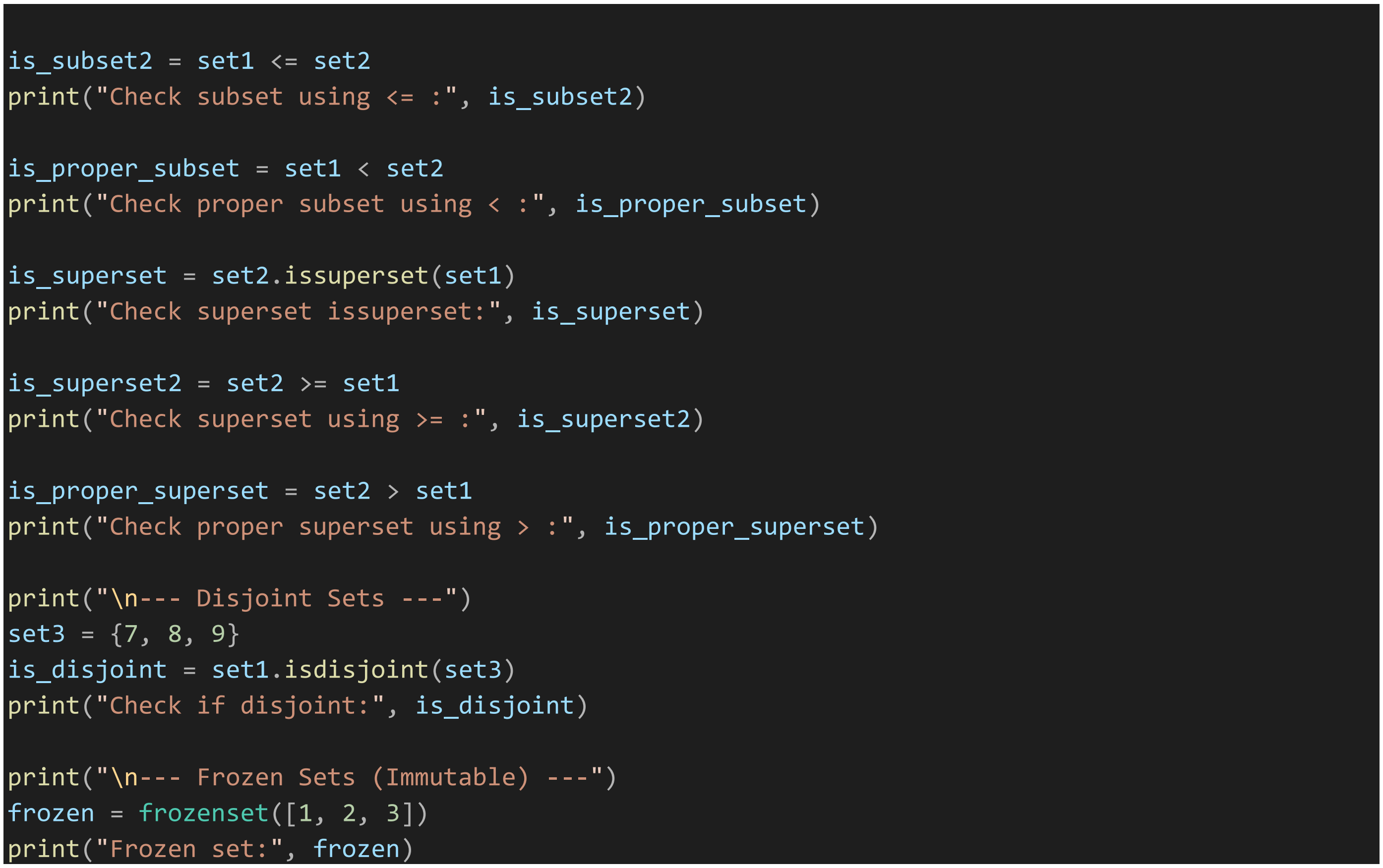
```python
# nested list
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(nested_list)             # output : [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# accessing elements from nested list
print(nested_list[0][0])       # output : 1
print(nested_list[1][2])       # output : 6
print(nested_list[2][1])       # output : 8

# accessing elements from nested list using for loop
for i in nested_list:
    for j in i:
        print(j)               # output : 1, 2, 3, 4, 5, 6, 7, 8, 9

for i in nested_list:
    for j in i:
        print(j, end=" ")    # output : 1 2 3 4 5 6 7 8 9
    print()

# accessing elements from nested list using while loop
i = 0
while i < len(nested_list):
    j = 0
    while j < len(nested_list[i]):
        print(nested_list[i][j])   # output : 1, 2, 3, 4, 5, 6, 7, 8, 9
        j += 1
    i += 1
```

Set Methods

```python
# Python Sets

print("\n--- Creating Sets ---")
my_set = {1, 2, 3, 4, 5}
print("Create a set:", my_set)

my_set2 = set([1, 2, 3, 4, 5])
print("Create set from list:", my_set2)

empty_set = set()
print("Empty set:", empty_set)

print("\n--- Adding / Updating ---")
my_set.add(6)
print("Add element 6:", my_set)

my_set.update([7, 8, 9])
print("Update with multiple elements [7,8,9]:", my_set)

print("\n--- Removing Elements ---")
my_set.remove(3)
print("Remove element 3:", my_set)

my_set.discard(10)  # no error if element not found
print("Discard element 10 (no error if missing):", my_set)

element = my_set.pop()
print("Pop random element:", element, "->", my_set)

my_set.clear()
print("Clear set:", my_set)

print("\n--- Copying ---")
my_set = {1, 2, 3, 4, 5}
new_set = my_set.copy()
print("Copy set:", new_set)

import copy
shallow_copy = copy.copy(my_set)
deep_copy = copy.deepcopy(my_set)
print("Shallow copy:", shallow_copy)
print("Deep copy:", deep_copy)

print("\n--- Set Operations ---")
set1 = {1, 2, 3, 4}
```



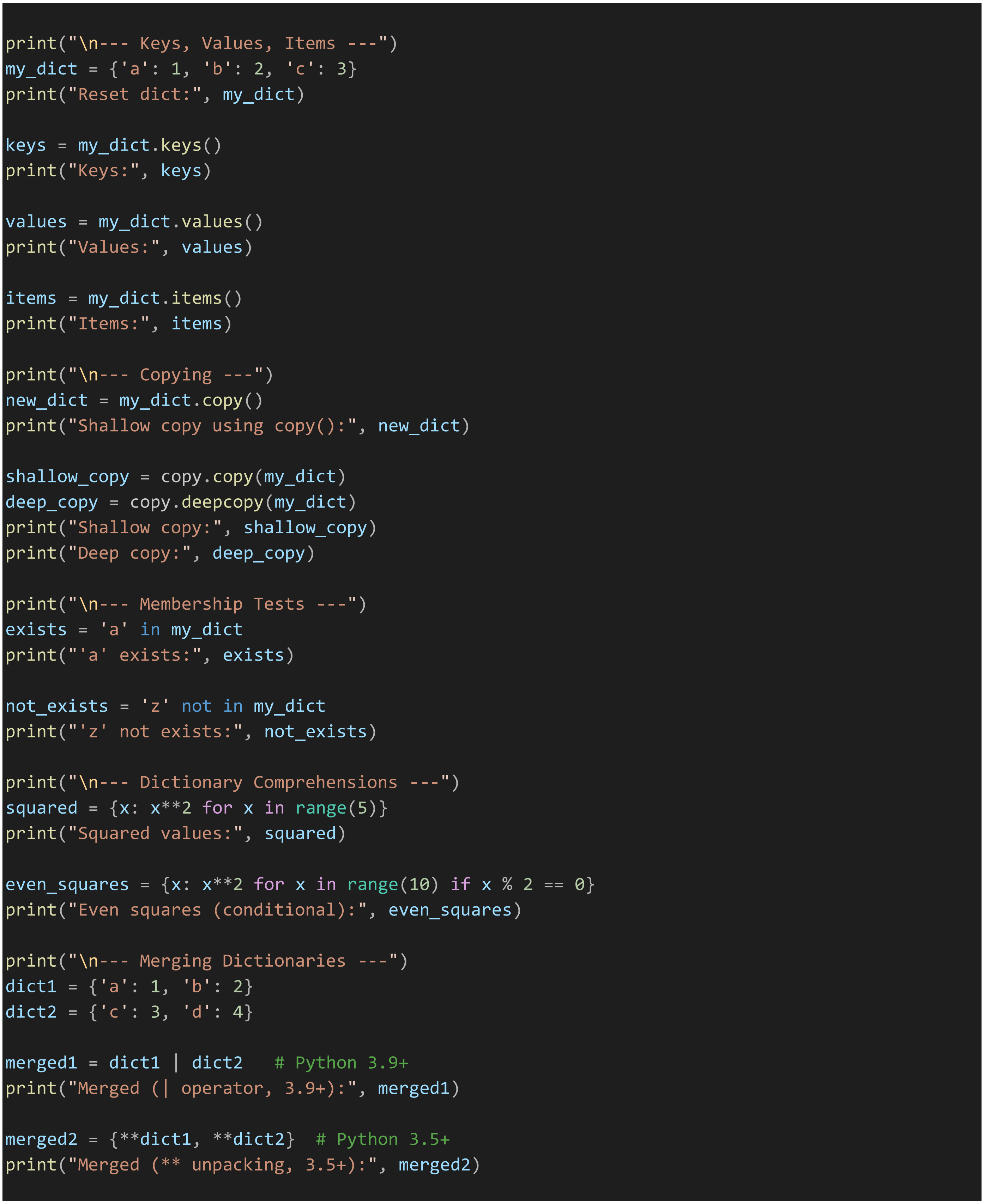
```
set2 = {3, 4, 5, 6}

union_set = set1.union(set2)
print("Union using union():", union_set)

union_set2 = set1 | set2
print("Union using | :", union_set2)

set1.update(set2)
print("Update set1 with union:", set1)

set1 = {1, 2, 3, 4}
intersect_set = set1.intersection(set2)
print("Intersection using intersection():", intersect_set)

intersect_set2 = set1 & set2
print("Intersection using & :", intersect_set2)

set1.intersection_update(set2)
print("Update set1 with intersection:", set1)

set1 = {1, 2, 3, 4}
diff_set = set1.difference(set2)
print("Difference set1 - set2:", diff_set)

diff_set2 = set1 - set2
print("Difference using - :", diff_set2)

set1.difference_update(set2)
print("Update set1 with difference:", set1)

set1 = {1, 2, 3, 4}
sym_diff = set1.symmetric_difference(set2)
print("Symmetric difference:", sym_diff)

sym_diff2 = set1 ^ set2
print("Symmetric difference using ^ :", sym_diff2)

set1.symmetric_difference_update(set2)
print("Update set1 with symmetric difference:", set1)

print("\n--- Subset / Superset Checks ---")
set1 = {1, 2}
set2 = {1, 2, 3, 4}

is_subset = set1.issubset(set2)
print("Check subset issubset:", is_subset)
```



```python
is_subset2 = set1 <= set2
print("Check subset using <= :", is_subset2)

is_proper_subset = set1 < set2
print("Check proper subset using < :", is_proper_subset)

is_superset = set2.issuperset(set1)
print("Check superset issuperset:", is_superset)

is_superset2 = set2 >= set1
print("Check superset using >= :", is_superset2)

is_proper_superset = set2 > set1
print("Check proper superset using > :", is_proper_superset)

print("\n--- Disjoint Sets ---")
set3 = {7, 8, 9}
is_disjoint = set1.isdisjoint(set3)
print("Check if disjoint:", is_disjoint)

print("\n--- Frozen Sets (Immutable) ---")
frozen = frozenset([1, 2, 3])
print("Frozen set:", frozen)
```

Dictionary Methods

```python
# Python Dictionaries Methods


from collections import OrderedDict
import copy
print("\n--- Creating Dictionaries ---")
my_dict = {'a': 1, 'b': 2, 'c': 3}
print("Create dictionary:", my_dict)

my_dict2 = dict(a=1, b=2, c=3)
print("Create with dict() constructor:", my_dict2)

my_dict3 = dict([('a', 1), ('b', 2), ('c', 3)])
print("Create from list of tuples:", my_dict3)

print("\n--- Accessing Values ---")
value = my_dict['a']
print("Access value by key ['a']:", value)

value = my_dict.get('d', 0)
print("Get value with default (key 'd'):", value)

print("\n--- Adding / Updating ---")
my_dict['d'] = 4
print("Add/update item d=4:", my_dict)

my_dict.update({'e': 5, 'f': 6})
print("Update multiple items:", my_dict)

print("\n--- Removing Items ---")
del my_dict['b']
print("Remove item with del ['b']:", my_dict)

val = my_dict.pop('c')
print("Pop item ['c']:", val, "->", my_dict)

val2 = my_dict.pop('e', None)
print("Pop with default (e):", val2, "->", my_dict)

item = my_dict.popitem()
print("Remove and return last item:", item, "->", my_dict)

my_dict.clear()
print("Clear dictionary:", my_dict)
```



```python
print("\n--- Keys, Values, Items ---")
my_dict = {'a': 1, 'b': 2, 'c': 3}
print("Reset dict:", my_dict)

keys = my_dict.keys()
print("Keys:", keys)

values = my_dict.values()
print("Values:", values)

items = my_dict.items()
print("Items:", items)

print("\n--- Copying ---")
new_dict = my_dict.copy()
print("Shallow copy using copy():", new_dict)

shallow_copy = copy.copy(my_dict)
deep_copy = copy.deepcopy(my_dict)
print("Shallow copy:", shallow_copy)
print("Deep copy:", deep_copy)

print("\n--- Membership Tests ---")
exists = 'a' in my_dict
print("'a' exists:", exists)

not_exists = 'z' not in my_dict
print("'z' not exists:", not_exists)

print("\n--- Dictionary Comprehensions ---")
squared = {x: x**2 for x in range(5)}
print("Squared values:", squared)

even_squares = {x: x**2 for x in range(10) if x % 2 == 0}
print("Even squares (conditional):", even_squares)

print("\n--- Merging Dictionaries ---")
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}

merged1 = dict1 | dict2   # Python 3.9+
print("Merged (| operator, 3.9+):", merged1)

merged2 = {**dict1, **dict2}  # Python 3.5+
print("Merged (** unpacking, 3.5+):", merged2)
print("\n--- Other Useful Methods ---")
length = len(my_dict)
print("Length of dict:", length)

new_dict2 = dict.fromkeys(['a', 'b', 'c'], 0)
print("Create dict with default values:", new_dict2)

my_dict = {'a': 1, 'b': 2}
val = my_dict.setdefault('c', 5)
print("Set default for missing key 'c':", val, "->", my_dict)

val2 = my_dict.setdefault('b', 10)
print("Set default for existing key 'b':", val2, "->", my_dict)

ordered = OrderedDict([('a', 1), ('b', 2), ('c', 3)])
print("OrderedDict:", ordered)
```

Tuple Methods

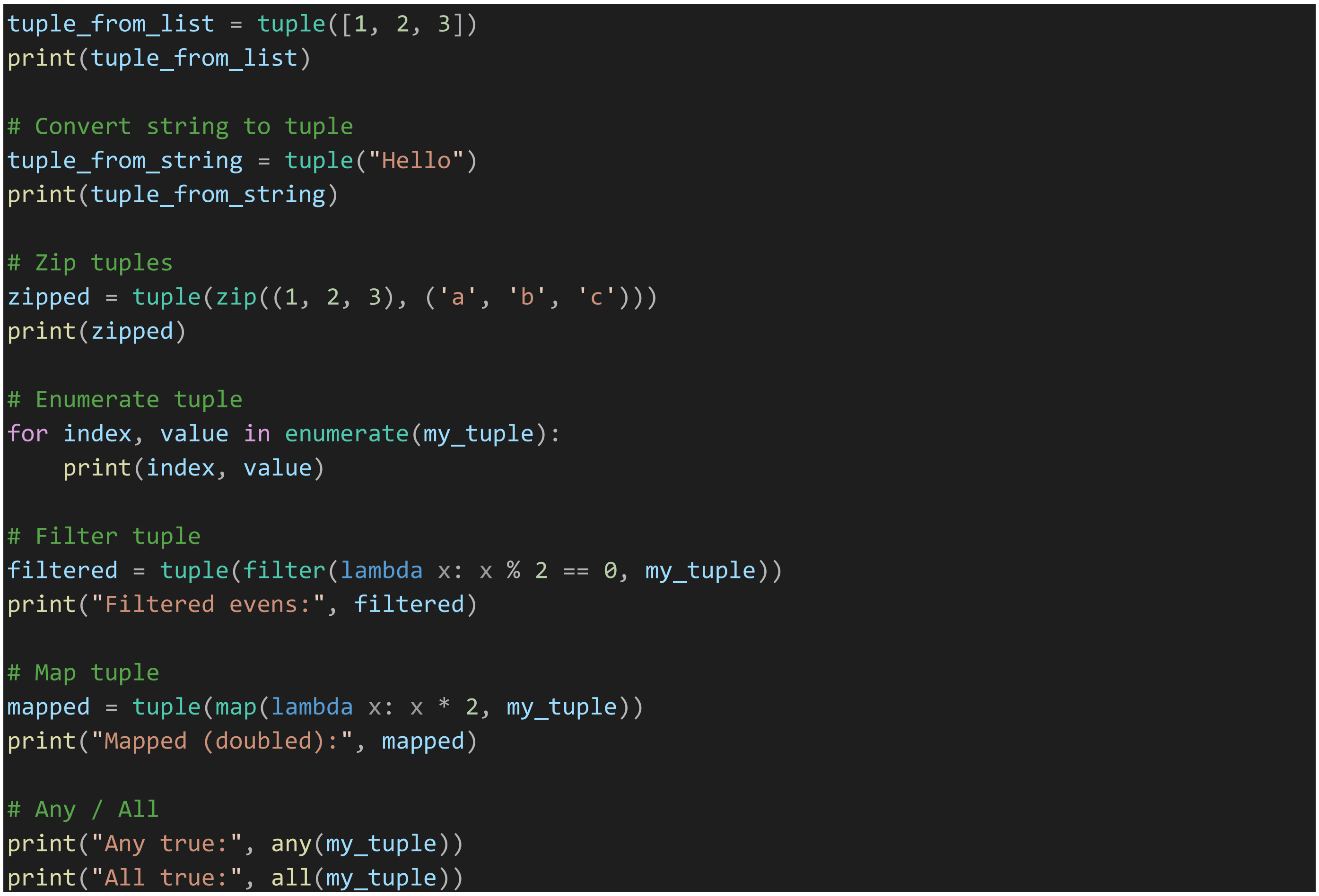
```python
# Python Tuples
print("\n--- Creating Tuples ---")
my_tuple = (1, 2, 3, 4, 5)
print("Create a tuple:", my_tuple)

my_tuple2 = tuple([1, 2, 3, 4, 5])
print("Create tuple from list:", my_tuple2)

single_tuple = (42,)
print("Create tuple with single element:", single_tuple)

empty_tuple = ()
print("Empty tuple:", empty_tuple)

print("\n--- Accessing Elements ---")
element = my_tuple[0]
print("Access element at index 0:", element)

last = my_tuple[-1]
print("Negative indexing (last element):", last)

sub_tuple = my_tuple[1:4]
print("Slicing [1:4]:", sub_tuple)

print("\n--- Tuple Operations ---")
new_tuple = my_tuple + (6, 7, 8)
print("Concatenate tuples:", new_tuple)

repeated = my_tuple * 3
print("Repeat tuple *3:", repeated)

a, b, c = (1, 2, 3)
print("Unpack tuple (a, b, c):", a, b, c)

a, *rest = (1, 2, 3, 4, 5)
print("Unpack with * (a, *rest):", a, rest)

a, b = 1, 2
a, b = b, a
print("Swap values:", a, b)

nested = ((1, 2), (3, 4))
print("Nested tuples:", nested)

print("\n--- Tuple Methods ---")
```

```python
count = my_tuple.count(2)
print("Count occurrences of 2:", count)

index = my_tuple.index(3)
print("Index of element 3:", index)

print("\n--- Membership Tests ---")
exists = 3 in my_tuple
print("Check if 3 exists:", exists)

not_exists = 10 not in my_tuple
print("Check if 10 not exists:", not_exists)

print("\n--- Tuple Properties ---")
length = len(my_tuple)
print("Length of tuple:", length)

max_val = max(my_tuple)
min_val = min(my_tuple)
print("Max:", max_val, "Min:", min_val)

total = sum(my_tuple)
print("Sum of tuple:", total)

print("\n--- Sorting ---")
sorted_tuple = sorted(my_tuple)
print("Sorted tuple (returns list):", sorted_tuple)

sorted_desc = sorted(my_tuple, reverse=True)
print("Sorted descending:", sorted_desc)

print("\n--- Conversions ---")
tuple_from_list = tuple([1, 2, 3])
print("Convert list to tuple:", tuple_from_list)

tuple_from_string = tuple("Hello")
print("Convert string to tuple:", tuple_from_string)

print("\n--- Advanced Operations ---")
zipped = tuple(zip((1, 2, 3), ('a', 'b', 'c')))
print("Zip tuples:", zipped)

for index, value in enumerate(my_tuple):
    print(f"Enumerate: index={index}, value={value}")

filtered = tuple(filter(lambda x: x % 2 == 0, my_tuple))
print("Filter even numbers:", filtered)
```

```python
mapped = tuple(map(lambda x: x * 2, my_tuple))
print("Map function (double each):", mapped)

print("\n--- Boolean Checks ---")
any_true = any(my_tuple)
print("Any true in tuple:", any_true)

all_true = all(my_tuple)
print("All true in tuple:", all_true)
```

```python
# Create a tuple
my_tuple = (1, 2, 3, 4, 5)
print(my_tuple)

# Create tuple from list
my_tuple = tuple([1, 2, 3, 4, 5])
print(my_tuple)

# Create tuple with single element
single_tuple = (42,)
print(single_tuple)

# Create empty tuple
empty_tuple = ()
print(empty_tuple)

# Access element
element = my_tuple[0]
print("First element:", element)

# Negative indexing
last = my_tuple[-1]
print("Last element:", last)

# Slicing
sub_tuple = my_tuple[1:4]
print("Sliced tuple:", sub_tuple)

# Concatenate tuples
new_tuple = my_tuple + (6, 7, 8)
print("Concatenated:", new_tuple)
```

```
# Repeat tuple
repeated = my_tuple * 3
print("Repeated:", repeated)

# Unpack tuple
a, b, c = (1, 2, 3)
print(a, b, c)

# Unpack with *
a, *rest = (1, 2, 3, 4, 5)
print("a:", a, "rest:", rest)

# Swap values
a, b = 1, 2
a, b = b, a
print("Swapped:", a, b)

# Nested tuple
nested = ((1, 2), (3, 4))
print(nested)

# Count occurrences
count = my_tuple.count(2)
print("Count of 2:", count)

# Index of element
index = my_tuple.index(3)
print("Index of 3:", index)

# Membership check
exists = 3 in my_tuple
not_exists = 10 not in my_tuple
print("Exists 3:", exists, "Not exists 10:", not_exists)

# Tuple length, min, max, sum
print("Length:", len(my_tuple))
print("Max:", max(my_tuple))
print("Min:", min(my_tuple))
print("Sum:", sum(my_tuple))

# Sorting
sorted_tuple = sorted(my_tuple)
sorted_desc = sorted(my_tuple, reverse=True)
print("Sorted:", sorted_tuple)
print("Sorted desc:", sorted_desc)

# Convert list to tuple
```



```python
tuple_from_list = tuple([1, 2, 3])
print(tuple_from_list)

# Convert string to tuple
tuple_from_string = tuple("Hello")
print(tuple_from_string)

# Zip tuples
zipped = tuple(zip((1, 2, 3), ('a', 'b', 'c')))
print(zipped)

# Enumerate tuple
for index, value in enumerate(my_tuple):
    print(index, value)

# Filter tuple
filtered = tuple(filter(lambda x: x % 2 == 0, my_tuple))
print("Filtered evens:", filtered)

# Map tuple
mapped = tuple(map(lambda x: x * 2, my_tuple))
print("Mapped (doubled):", mapped)

# Any / All
print("Any true:", any(my_tuple))
print("All true:", all(my_tuple))
```

String Methods

```python
# Create string
my_string = "Hello, World!"
print(my_string)

# Multi-line string
multi_line = """This is
a multi-line
string"""
print(multi_line)

# Raw string
raw_string = r"C:\Users\John"
print(raw_string)

# f-string
name = "John"
greeting = f"Hello, {name}!"
print(greeting)

# Access char
char = my_string[1]
print(char)

# String slicing
slice_ = my_string[7:12]
print(slice_)

# Reverse string
reversed_str = my_string[::-1]
print(reversed_str)

# Concatenate
new_string = "Hello" + " " + "World"
print(new_string)

# Repeat
repeated = "Ha" * 3
print(repeated)

# Length
print(len(my_string))

# Case conversions
print(my_string.upper())
print(my_string.lower())
print(my_string.capitalize())
print(my_string.title())
print(my_string.swapcase())

# Strip whitespace/characters
print("   hello   ".strip())
print("   hello   ".lstrip())
print("   hello   ".rstrip())
print("...Hello!!!".strip('!.'))

# Replace
print(my_string.replace("Hello", "Hi"))

# Split and join
print(my_string.split(","))
print("a-b-c".split("-"))
print("-".join(["a", "b", "c"]))

# Alignments
print(my_string.center(20, '*'))
print(my_string.ljust(20, '*'))
print(my_string.rjust(20, '*'))

# Tabs
print("Hello\tWorld".expandtabs(8))

# Startswith, endswith
print(my_string.startswith("Hello"))
print(my_string.endswith("!"))

# Find, rfind, index
print(my_string.find("World"))
print(my_string.rfind("o"))
print(my_string.index("World"))

# Count
print(my_string.count("l"))

# String checks
print("abc123".isalnum())
print("abc".isalpha())
print("123".isdigit())
print("Ⅻ".isnumeric())
print("42".isdecimal())
print("hello".islower())
print("HELLO".isupper())
print("Hello".istitle())
print("   ".isspace())
print("hello".isprintable())
print("var_1".isidentifier())

# Encode / Decode
encoded = my_string.encode('utf-8')
print(encoded)
decoded = encoded.decode('utf-8')
print(decoded)

# Remove prefix/suffix
print("Hello, World!".removeprefix("Hello, "))
print("Hello, World!".removesuffix("!"))

# Partition / rpartition
print("a,b,c".partition(","))
print("a,b,c".rpartition(","))

# Translate
trans_table = str.maketrans('aeiou', '12345')
print("hello".translate(trans_table))

# Format strings
print("Hello, {}!".format("World"))
print("Hello, {name}!".format(name="John"))
print("{0} {1} {0}".format("Hello", "World"))
print("{first} {last}".format({"first": "John", "last": "Doe"}))

# Formatting width / alignment
print("{:<10}".format("left"))
print("{:^10}".format("center"))
print("{:>10}".format("right"))

# Padding, signs, numbers
print("{:05}".format(42))
print("{:+}".format(42))
print("{:.2%}".format(0.12345))
print("{:,}".format(1234567))

# Binary, Octal, Hex, Scientific
print("{:b}".format(42))
print("{:o}".format(42))
print("{:x}".format(42))
print("{:e}".format(1234.5678))

# Zfill
print("42".zfill(5))
```