List

```
# Create a list
numbers = [1, 2, 3, 4, 5]

# Modify the first element of the list
numbers[0] = 10

print(numbers)  # Output: [10, 2, 3, 4, 5]

# Create a tuple
numbers = (1, 2, 3, 4, 5)

# Try to modify the first element of the tuple
numbers[0] = 10

print(numbers)  # Output: TypeError: 'tuple' object does not support item assignment


# Create a list of numbers
numbers = [1, 2, 3, 4, 5]

# Create a list of strings
names = ["Alice", "Bob", "Charlie"]

# Create an empty list
empty_list = []


# Access the first element of the list
first_element = numbers[0]  # first_element = 1

# Access the last element of the list
last_element = numbers[-1]  # last_element = 5

# Access the element at index 2
third_element = numbers[2]  # third_element = 3


# Modify the first element of the list
numbers[0] = 10

# Modify the element at index 2
numbers[2] = 20


# Add the number 6 to the end of the list
numbers.append(6)

# Remove the number 4 from the list
numbers.remove(4)
```

```
# Insert the number 15 at index 1
numbers.insert(1, 15)


# Reverse the list
reversed_numbers = numbers[::-1]

print(reversed_numbers)  # Output: [6, 20, 15, 10]


# Sort the list
sorted_numbers = sorted(numbers)

print(sorted_numbers)  # Output: [10, 15, 20]


# Iterate over the list and print each element
for number in numbers:
    print(number)


# Check if the number 5 is in the list
if 5 in numbers:
    print("5 is in the list")


# with heterogeneous items
my_list3 = [1.0, 'Jessa', 3]
print(my_list3)
# Output [1.0, 'Jessa', 3]


# empty list using list()
my_list4 = list()
print(my_list4)
# Output []


my_list = [10, 20, 'Jessa', 12.50, 'Emma']
# accessing 2nd element of the list
print(my_list[1])  # 20
# accessing 5th element of the list
print(my_list[4])  # 'Emma'


my_list = [10, 20, 'Jessa', 12.50, 'Emma']
# accessing last element of the list
print(my_list[-1])
# output 'Emma'

# accessing second last element of the list
print(my_list[-2])
```

```
# output 12.5

# accessing 4th element from last
print(my_list[-4])
# output 20


my_list = [10, 20, 'Jessa', 12.50, 'Emma', 25, 50]
# Extracting a portion of the list from 2nd till 5th element
print(my_list[2:5])
# Output ['Jessa', 12.5, 'Emma']


my_list = [5, 8, 'Tom', 7.50, 'Emma']

# slice first four items
print(my_list[:4])
# Output [5, 8, 'Tom', 7.5]

# print every second element
# with a skip count 2
print(my_list[::2])
# Output [5, 'Tom', 'Emma']

# reversing the list
print(my_list[::-1])
# Output ['Emma', 7.5, 'Tom', 8, 5]

# Without end_value
# Stating from 3nd item to last item
print(my_list[3:])
# Output [7.5, 'Emma']


my_list = list([5, 8, 'Tom', 7.50])

# Using append()
my_list.append('Emma')
print(my_list)
# Output [5, 8, 'Tom', 7.5, 'Emma']

# append the nested list at the end
my_list.append([25, 50, 75])
print(my_list)
# Output [5, 8, 'Tom', 7.5, 'Emma', [25, 50, 75]]


my_list = list([5, 8, 'Tom', 7.50])

# Using insert()
# insert 25 at position 2
my_list.insert(2, 25)
print(my_list)
# Output [5, 8, 25, 'Tom', 7.5]

# insert nested list at at position 3
my_list.insert(3, [25, 50, 75])
```

```
print(my_list)
# Output [5, 8, 25, [25, 50, 75], 'Tom', 7.5]


my_list = list([5, 8, 'Tom', 7.50])

# Using extend()
my_list.extend([25, 75, 100])
print(my_list)
# Output [5, 8, 'Tom', 7.5, 25, 75, 100]


my_list = list([2, 4, 6, 8, 10, 12])

# modify single item
my_list[0] = 20
print(my_list)
# Output [20, 4, 6, 8, 10, 12]

# modify range of items
# modify from 1st index to 4th
my_list[1:4] = [40, 60, 80]
print(my_list)
# Output [20, 40, 60, 80, 10, 12]

# modify from 3rd index to end
my_list[3:] = [80, 100, 120]
print(my_list)
# Output [20, 40, 60, 80, 100, 120]


my_list = list([2, 4, 6, 8, 10, 12])

# remove item 6
my_list.remove(6)
# remove item 8
my_list.remove(8)

print(my_list)
# Output [2, 4, 10, 12]


my_list = list([2, 4, 6, 8, 10, 12])

# remove item present at index 2
my_list.pop(2)
print(my_list)
# Output [2, 4, 8, 10, 12]

# remove item without passing index number
my_list.pop()
print(my_list)
# Output [2, 4, 8, 10]
```

Tuple

```
# Create a tuple of numbers
numbers = (1, 2, 3, 4, 5)

# Access the first element of the tuple
first_element = numbers[0]  # first_element = 1

# Access the last element of the tuple
last_element = numbers[-1]  # last_element = 5

# Access the element at index 2
third_element = numbers[2]  # third_element = 3

# Check if the number 5 is in the tuple
if 5 in numbers:
    print("5 is in the tuple")


# Find the index of the number 20 in the tuple
index_of_20 = numbers.index(20)

print(index_of_20)  # Output: 3


# create a tuple using ()
# number tuple
number_tuple = (10, 20, 25.75)
print(number_tuple)
# Output (10, 20, 25.75)

# string tuple
string_tuple = ('Jessa', 'Emma', 'Kelly')
print(string_tuple)
# Output ('Jessa', 'Emma', 'Kelly')

# mixed type tuple
sample_tuple = ('Jessa', 30, 45.75, [25, 78])
print(sample_tuple)
# Output ('Jessa', 30, 45.75, [25, 78])

# create a tuple using tuple() constructor
sample_tuple2 = tuple(('Jessa', 30, 45.75, [23, 78]))
print(sample_tuple2)
# Output ('Jessa', 30, 45.75, [23, 78])


# without comma
single_tuple = ('Hello')
print(type(single_tuple))
# Output class 'str'
print(single_tuple)
# Output Hello

# with comma
```

```
single_tuple1 = ('Hello',)
# output class 'tuple'
print(type(single_tuple1))
# Output ('Hello',)
print(single_tuple1)


# packing variables into tuple
tuple1 = 1, 2, "Hello"
# display tuple
print(tuple1)
# Output (1, 2, 'Hello')

print(type(tuple1))
# Output class 'tuple'

# unpacking tuple into variable
i, j, k = tuple1
# printing the variables
print(i, j, k)
# Output 1 2 Hello



tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')
for i in range(4):
    print(tuple1[i])


P
Y
T
H


tuple1 = ('P', 'Y', 'T', 'H', 'O', 'N')
# Negative indexing
# print last item of a tuple
print(tuple1[-1])  # N
# print second last
print(tuple1[-2])  # O

# iterate a tuple using negative indexing
for i in range(-6, 0):
    print(tuple1[i], end=", ")
# Output P, Y, T, H, O, N,


tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# slice a tuple with start and end index number
print(tuple1[1:5])
# Output (1, 2, 3, 4)


tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
# slice a tuple without start index
print(tuple1[:5])
# Output (0, 1, 2, 3, 4)

# slice a tuple without end index
print(tuple1[6:])
# Output (6, 7, 8, 9, 10)


tuple1 = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# slice a tuple using negative indexing
print(tuple1[-5:-1])
# Output (6, 7, 8, 9)


tuple1 = (10, 20, 30, 40, 50, 60, 70, 80)
# Limit the search locations using start and end
# search only from location 4 to 6
# start = 4 and end = 6
# get index of item 60
position = tuple1.index(60, 4, 6)
print(position)
# Output 5


tuple1 = (0, 1, 2, 3, 4, 5)

# converting tuple into a list
sample_list = list(tuple1)
# add item to list
sample_list.append(6)

# converting list back into a tuple
tuple1 = tuple(sample_list)
print(tuple1)
# Output (0, 1, 2, 3, 4, 5, 6)
```

Dictionary

What is the output of the following code

```
dict1 = {"key1":1, "key2":2}
dict2 = {"key2":2, "key1":1}
print(dict1 == dict2)
```

True
False

What is the output of the following dictionary operation

```
dict1 = {"name": "Mike", "salary": 8000}
temp = dict1.get("age")
print(temp)
```

KeyError: 'age'
None

Select correct ways to create an empty dictionary

```
sampleDict = {}
sampleDict = dict()
sampleDict = dict{}
```

What is the output of the following dictionary operation

```
dict1 = {"name": "Mike", "salary": 8000}
temp = dict1.pop("age")
print(temp)
```

KeyError: 'age'
None

Select all correct ways to copy a dictionary in Python

dict1 = {"name": "Mike", "salary": 8000}

```
dict2 = dict1.copy()
dict2 = dict(dict1)
dict2 = dict1
```

Select the correct way to print Emma's age.

```
student = {1: {'name': 'Emma', 'age': '27', 'sex': 'Female'},
           2: {'name': 'Mike', 'age': '22', 'sex': 'Male'}}
```

```
student[0][1]
```

```
student[1]["age"]
```

```
student[0]["age"]
```

student[1]["name"]

What is the output of the following

```
sampleDict = dict([
    ('first', 1),
    ('second', 2),
    ('third', 3)
])
print(sampleDict)
```

[(‘first’, 100), (‘second’, 200), (‘third’, 300)]
Options: SyntaxError: invalid syntax
{‘first’: 1, ‘second’: 2, ‘third’: 3}

Select the correct ways to get the value of marks key.

```
student = {
  "name": "Emma",
  "class": 9,
  "marks": 75
}
```

```
m = student.get(2)
```

```
m = student.get('marks')
```

```
m = student[2])
```

```
m = student['marks'])
```

Select the correct way to access the value of a history subject

```
sampleDict = {
   "class":{
      "student":{
         "name":"Mike",
         "marks":{
            "physics":70,
            "history":80
         }
      }
   }
}
```

```
sampleDict['class']['student']['marks']['history']
```

```
sampleDict['class']['student']['marks'][1]
```

```
sampleDict['class'][0]['marks']['history']
```

Select the correct way to **remove** the key **marks** from a dictionary

```
student = {
  "name": "Emma",
  "class": 9,
  "marks": 75
}
```

```
student.pop("marks")
```

```
del student["marks"]
```

```
student.remove("marks")
```

```
student.popitem("marks")
```

Graphs

```
import pprint
from collections import defaultdict


class Graph(object):
    """ Graph data structure, undirected by default. """

    def __init__(self, connections, directed=False):
        self._graph = defaultdict(set)
        self._directed = directed
        self.add_connections(connections)

    def add_connections(self, connections):
        """ Add connections (list of tuple pairs) to graph """

        for node1, node2 in connections:
            self.add(node1, node2)

    def add(self, node1, node2):
        """ Add connection between node1 and node2 """

        self._graph[node1].add(node2)
        if not self._directed:
            self._graph[node2].add(node1)

    def remove(self, node):
        """ Remove all references to node """

        for n, cxns in self._graph.items():  # python3: items(); python2:
iteritems()
            try:
                cxns.remove(node)
            except KeyError:
                pass
        try:
            del self._graph[node]
        except KeyError:
            pass

    def is_connected(self, node1, node2):
        """ Is node1 directly connected to node2 """

        return node1 in self._graph and node2 in self._graph[node1]

    def find_path(self, node1, node2, path=[]):
        """ Find any path between node1 and node2 (may not be shortest) """

        path = path + [node1]
        if node1 == node2:
            return path
        if node1 not in self._graph:
            return None
        for node in self._graph[node1]:
            if node not in path:
                new_path = self.find_path(node, node2, path)
```

```
                if new_path:
                    return new_path
        return None

    def __str__(self):
        return '{}({})'.format(self.__class__.__name__, dict(self._graph))


>>> connections = [('A', 'B'), ('B', 'C'), ('B', 'D'),
                   ('C', 'D'), ('E', 'F'), ('F', 'C')]
>>> g = Graph(connections, directed=True)
>>> pretty_print = pprint.PrettyPrinter()
>>> pretty_print.pprint(g._graph)
{'A': {'B'},
 'B': {'D', 'C'},
 'C': {'D'},
 'E': {'F'},
 'F': {'C'}}

>>> g = Graph(connections)  # undirected
>>> pretty_print = pprint.PrettyPrinter()
>>> pretty_print.pprint(g._graph)
{'A': {'B'},
 'B': {'D', 'A', 'C'},
 'C': {'D', 'F', 'B'},
 'D': {'C', 'B'},
 'E': {'F'},
 'F': {'E', 'C'}}

>>> g.add('E', 'D')
>>> pretty_print.pprint(g._graph)
{'A': {'B'},
 'B': {'D', 'A', 'C'},
 'C': {'D', 'F', 'B'},
 'D': {'C', 'E', 'B'},
 'E': {'D', 'F'},
 'F': {'E', 'C'}}

>>> g.remove('A')
>>> pretty_print.pprint(g._graph)
{'B': {'D', 'C'},
 'C': {'D', 'F', 'B'},
 'D': {'C', 'E', 'B'},
 'E': {'D', 'F'},
 'F': {'E', 'C'}}

>>> g.add('G', 'B')
>>> pretty_print.pprint(g._graph)
{'B': {'D', 'G', 'C'},
 'C': {'D', 'F', 'B'},
 'D': {'C', 'E', 'B'},
 'E': {'D', 'F'},
 'F': {'E', 'C'},
 'G': {'B'}}

>>> g.find_path('G', 'E')
['G', 'B', 'D', 'C', 'F', 'E']
```