Lists , Tuples and Dictionaries

Dr. Thyagaraju G S

1.Lists

- The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets.
- Important thing about a list is that items in a list need not be of the same type.
- Creating a list is as simple as putting different comma-separated values between square brackets.
- A list is an ordered collection of objects. The order of the elements is an innate characteristic of the list.
- A list may contain any number of elements (constrained by the computer's memory, of course), of any type. The same object may occur any number of times.

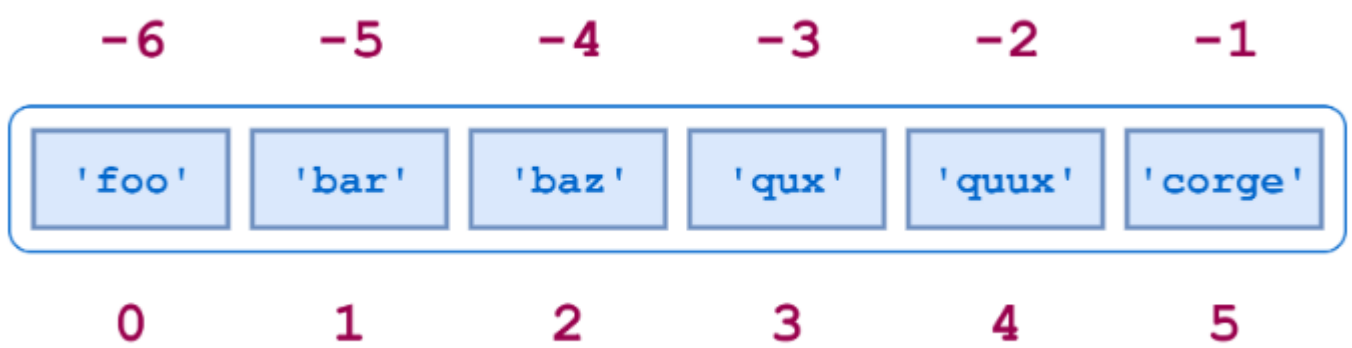
1. Creating a List

```
# Creating List
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

```
print(list1)
print(list2)
print(list3)
```

```
['physics', 'chemistry', 1997, 2000]
[1, 2, 3, 4, 5]
['a', 'b', 'c', 'd']
```

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```



Accessing Values in Lists

- To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.

Example

```
list1 = [22.5 , 'Artificial Intelligence' ,'Python', 'Deep Learning', 1997, 2000];
print("list1[0]: ", list1[0])
print("list1[5]: ", list1[5])
print("list1[0:]: ", list1[0:])
print("list1[:]: ", list1[:])
print("list1[1:6]: ", list1[1:6])
print("list1[-6]: ", list1[-6])
print("list1[0:-5]: ", list1[0:-5])
print("list1[::]: ", list1[::])
print("list1[::-1]: ", list1[::-1])
print("list1[1:4:2]: ", list1[1:4:2])
print("list1[4::-2]: ", list1[4::-2])
```

Adding new Values

```
#Example code for adding new values to lists
list1 = ['AI', 'Deep Learning', 1997, 2000]
print("list1 values: ", list1)
# Adding new value to list
list1.append("MACHINE LEARNING")
print("list1 values post append: ", list1)
```

Updating Existing Values

:

```
#Example code for updating existing values of lists
print("Values of list1: ", list1)
# Updating existing value of list
print("Index 2 value : ", list1[2])
list1[2] = 2020
print("Index 2's new value : ", list1[2])
```

Deleting a List Element

```
1  #Example code for deleting a list element
2  print("list1 values: ", list1)
3  # Deleting list element
4  del list1[3]
5  print("After deleting value at index 2 : ", list1)
```

```
list1 values:  ['AI', 'Deep Learning', 2020, 2000, 'MACHINE LEARNING']
After deleting value at index 2 :  ['AI', 'Deep Learning', 2020, 'MACHINE LEARNING']
```

Basic Operations - 1

```
# Basic Operations on List
import math
import string
import operator
#Example code for basic operations on lists

print("Length: ", len(list1))

print("Concatenation: ", [1,2,3] + [4, 5, 6])

print("Repetition :", ['Hello'] * 4)

print("Membership :", 3 in [1,2,3])

print("Iteration :")
for x in [1,2,3]:
    print(x)
```

Basic Operations 2

```
list2 = ['PSYCHOLOGY','STATISTICS','MATHS','DOMAIN']
print("Max of list: ", max([1,2,3,4,5]))

print("Min of list: ", min([1,2,3,4,5]))

print("Count number of 1 in list: ", [1,1,2,3,4,5,].count(1))

list1.extend(list2)

print("Extended :", list1)
```

Basic Operations 3

```
print("pop last item in list: ", list1.pop())
print("List after POP",list1)
print("pop the item with index 2: ", list1.pop(2))
print("List after POP 2",list1)
list1.remove('MATHS')
print("removed b from list: ", list1)
list1.reverse()
print("Reverse: ", list1)
list1 = ['a','c','b']
list1.sort()
print("Sort ascending: ", list1)
list1.sort(reverse = True)
print("Sort descending: ", list1)
```

Q1

Which of the following are true of Python lists?

- [ ] A list may contain any type of object except another list
- [ ] There is no conceptual limit to the size of a list
- [ ] All elements in a list must be of the same type
- [ ] A given object may appear in a list more than once
- [ ] These represent the same list:

```
['a', 'b', 'c']
```

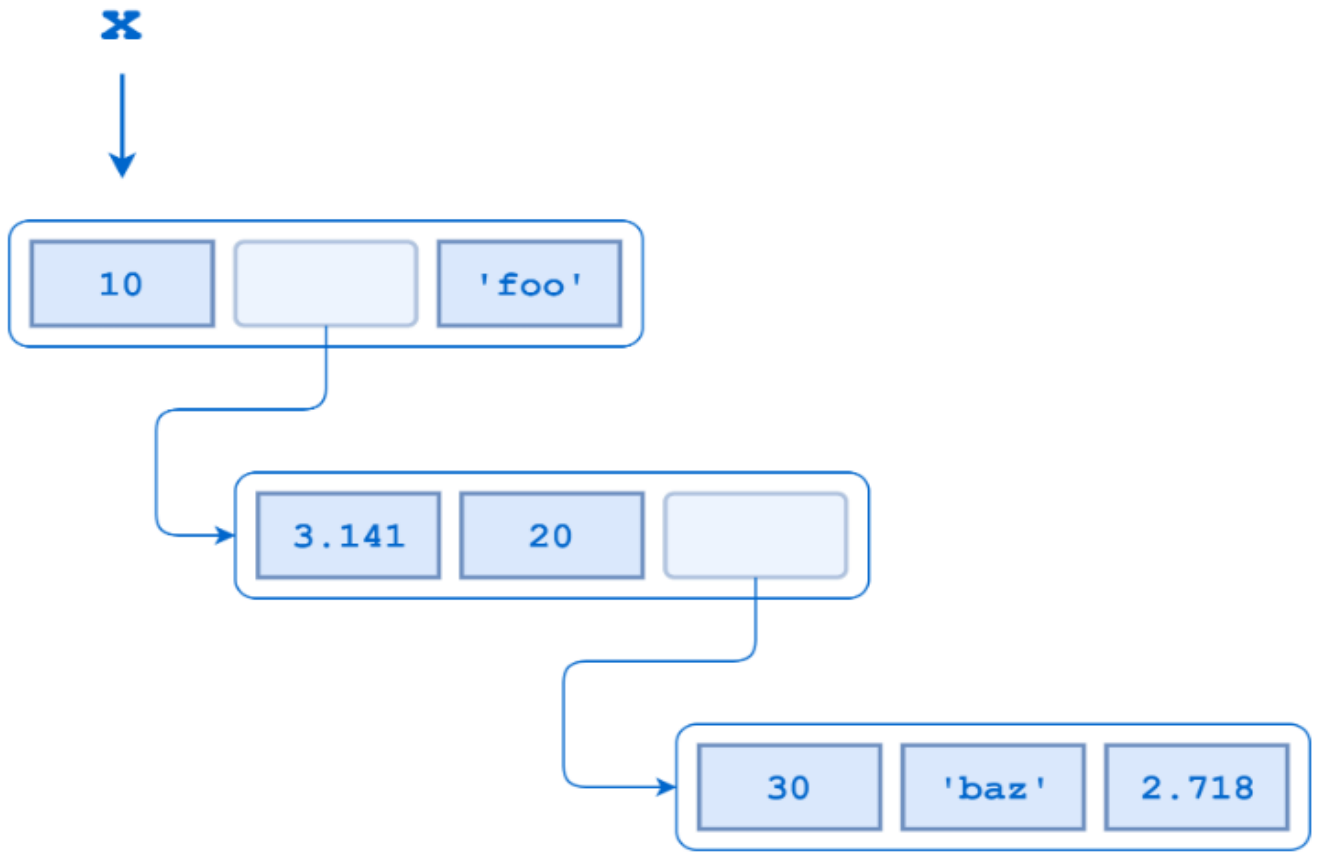
```
['c', 'a', 'b']
```

2. Assume the following list definition:

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```

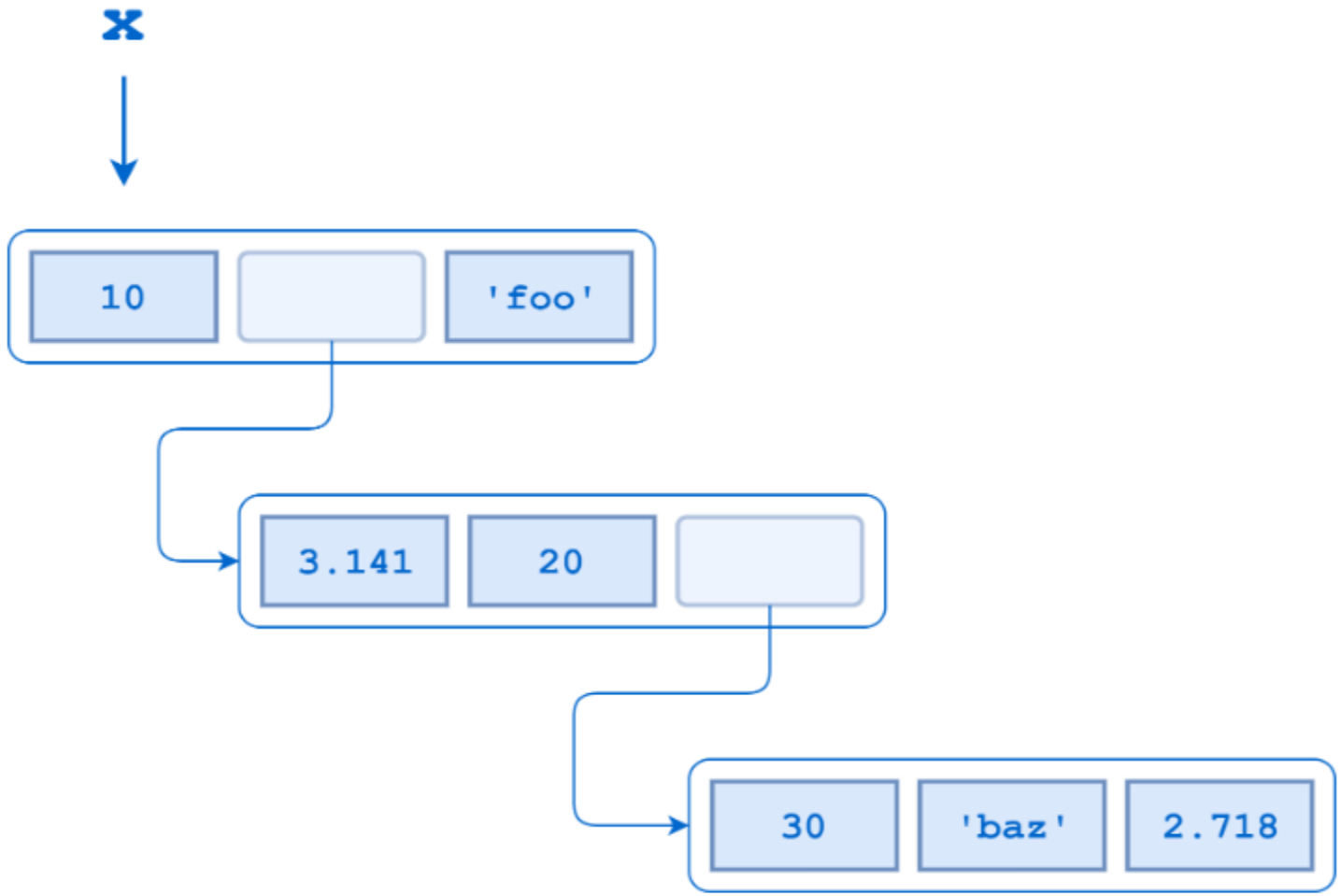
```
x = [10, [3.141, 20, [30, 'baz', 2.718]], 'foo']
```

A schematic for this list is shown below:



What is the expression that returns the `'z'` in `'baz'`?

```
x = [10, [3.141, 20, [30, 'baz', 2.718]], 'foo']
```



What expression returns the list `['baz', 2.718]`?

What is the output of the following Snippet?

```
1  a = [1, 2, 3, 4, 5]
2  print(a[2:2] )
3  a = [1, 2, 3, 4, 5]
4  print(a.remove(3) )
5  a = [1, 2, 3, 4, 5]
6  del a[2]
7  print(a)
8  a = [1, 2, 3, 4, 5]
9  print(a[2:3] )
```

What is the output of the following Snippet?

```
1  a = ['a', 'b', 'c']
2  a += ['d', 'e']
3  print(a)
4  a = ['a', 'b', 'c']
5  a[-1:] = ['d', 'e']
6  print(a)
7  a = ['a', 'b', 'c']
8  a[len(a):] = ['d', 'e']
9  print(a)
10 a = ['a', 'b', 'c']
11 a += 'de'
12 print(a)
13 a = ['a', 'b', 'c']
14 a.extend(['d', 'e'])
15 print(a)
```

What is the output of the following Snippet?

```
1  a = [1, 2, 7, 8]
2  a = a[0:2] + [3,4,5,6] + a[2:]
3  print(a)
4  a = [1, 2, 7, 8]
5  a[2:2] = [3, 4, 5, 6]
6  print(a)
```

2. Tuples

- A Python tuple is a sequences or series of immutable Python objects very much similar to the lists. However there exist some essential differences between lists and tuples, which are the following.
  - Unlike list, the objects of tuples cannot be changed.
  - Tuples are defined by using parentheses, but lists are defined by square brackets

Creating Tuples

```
:
1  # Example code for creating tuple
2  # Creating a tuple
3  Tuple = ()
4  print("Empty Tuple: ", Tuple)
5  Tuple = (1,)
6  print("Tuple with single item: ", Tuple)
7  Tuple = ('a','b','c','d',1,2,3)
8  print("Sample Tuple :", Tuple)
```

```
Empty Tuple:  ()
Tuple with single item:  (1,)
Sample Tuple : ('a', 'b', 'c', 'd', 1, 2, 3)
```

Accessing elemenst of Tuples

:

```
#Example code for accessing tuple
# Accessing items in tuple
Tuple = ('a', 'b', 'c', 'd', 1, 2, 3)
print("3rd item of Tuple:", Tuple[2])
print("First 3 items of Tuple", Tuple[0:2])
```

```
3rd item of Tuple: c
First 3 items of Tuple ('a', 'b')
```

Deleting Elements of Tuples

:

```
1 #Example code for deleting tuple
2 # Deleting tuple
3 print("Sample Tuple: ", Tuple)
4 del Tuple
5 print(Tuple) # Will throw an error message as the tuple does not exist
```

```
Sample Tuple:  ('a', 'b', 'c', 'd', 1, 2, 3)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-67-efdc3134feeb> in <module>
      3 print("Sample Tuple: ", Tuple)
      4 del Tuple
----> 5 print(Tuple) # Will throw an error message as the tuple does not exist

NameError: name 'Tuple' is not defined
```

Basic Tuple Operations -1

```
1  # Example code for basic operations on tupe (not exhaustive)
2  # Basic Tuple operations
3  Tuple = ('a','b','c','d',1,2,3)
4  print("Length of Tuple: ", len(Tuple))
5  Tuple_Concat = Tuple + (7,8,9)
6  print("Concatinated Tuple: ", Tuple_Concat)
7
8  print("Repetition: ", (1,'a',2, 'b') * 3)
9  print("Membership check: ", 3 in (1,2,3))
```

```
Length of Tuple:  7
Concatinated Tuple:  ('a', 'b', 'c', 'd', 1, 2, 3, 7, 8, 9)
Repetition:  (1, 'a', 2, 'b', 1, 'a', 2, 'b', 1, 'a', 2, 'b')
Membership check:  True
```

Basic Tuple Operations -2

```
# Iteration
for x in (1, 2, 3): print(x)
print("Negative sign will retrieve item from right: ", Tuple_Concat[-2])
print("Sliced Tuple [2:] ", Tuple_Concat[2:])
# Find max
print("Max of the Tuple (1,2,3,4,5,6,7,8,9,10): ",
max((1,2,3,4,5,6,7,8,9,10)))
print("Min of the Tuple (1,2,3,4,5,6,7,8,9,10): ",
min((1,2,3,4,5,6,7,8,9,10)))
print("List [1,2,3,4] converted to tuple: ", type(tuple([1,2,3,4])))
```

```
1
2
3
Negative sign will retrieve item from right:  8
Sliced Tuple [2:]  ('c', 'd', 1, 2, 3, 7, 8, 9)
Max of the Tuple (1,2,3,4,5,6,7,8,9,10):  10
Min of the Tuple (1,2,3,4,5,6,7,8,9,10):  1
List [1,2,3,4] converted to tuple:  <class 'tuple'>
```

What is the Output of the following ?

```
1  t = ('foo', 'bar', 'baz')
2  t[1:1] = 'qux'
3  print(t)
4  t = ('foo', 'bar', 'baz')
5  t[1] = 'qux'
6  print(t)
```

What is the output of the following and Why ?

```
1  a, b, c = (1, 2, 3, 4, 5, 6, 7, 8, 9)[1::3]
2  print(a)
3  print(b)
4  print(c)
```

What is the output of the following and Why ?

```
1  t= (1,2,3,4,5)
2  print(t[::-1])
3  x = 5
4  y = -5
5  x, y = (y, x)[::-1]
6  print(x)
7  print(y)
```

3. Dictionaries

- The Python dictionary will have a key and value pair for each item that is part of it.
- The key and value should be enclosed in curly braces. Each key and value is separated using a colon (:), and further each item is separated by commas (,).
- Note that the keys are unique within a specific dictionary and must be immutable data types such as strings, numbers, or tuples, whereas values can take duplicate data of any type.

Points to remember

- [x] Dictionaries are accessed by key.
- [x] Dictionaries are mutable.
- [x] Dictionaries can be nested to any depth.

Creating Dictionaries

```
1  # Example code for creating dictionary
2  # Creating dictionary
3  dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
4  print("Sample dictionary: ", dict)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
```

```
1  dict0 = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
2  print("Sample dictionary: ", dict0)
3  k=1
4  for i in dict0:
5      print(k,i,dict0[i])
6      k=k+1
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
1 Name Jivin
2 Age 6
3 Class First
```

Accessing Dictionary Element

```
# Example code for accessing dictionary
# Accessing items in dictionary
print("Value of key Name, from sample dictionary:", dict['Name'])
```

```
Value of key Name, from sample dictionary: Jivin
```

Deleting Dictionary Elements

```
1  dict0 = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
2  print("Sample dictionary: ", dict0)
3  del (dict0['Name']) # Delete specific item
4
5  print("Sample dictionary post deletion of item Name:", dict0)
6
7  dict0 = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
8  dict0.clear() # Clear all the contents of dictionary
9  print("dict post dict.clear():", dict0)
10
11 dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
12 del (dict0) # Delete the dictionary
13 print(dict0)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
Sample dictionary post deletion of item Name: {'Age': 6, 'Class': 'First'}
dict post dict.clear(): {}
```

Updating Dictionary Element

```
# Updating Dictionary Element
#Example code for updating dictionary
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Sample dictionary: ", dict)
dict['Age'] = 6.5
print("Dictionary post age value update: ", dict)
```

```
Sample dictionary:  {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
Dictionary post age value update:  {'Name': 'Jivin', 'Age': 6.5, 'Class': 'First'}
```

Basic Dictionary Operations -1

```
#Example code for basic operations on dictionary
# Basic operations
dict = {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
print("Length of dict: ", len(dict))

# Copy the dict
dict1 = dict.copy()
print("Copy:\n",dict1)


```

```
Length of dict:  3
Copy:
 {'Name': 'Jivin', 'Age': 6, 'Class': 'First'}
```

Basic Dictionary Operations -2

```
:
  1 # Retrieve value for a given key
  2 print("Value for Age: ", dict.get('Age'))
  3
  4 # Return items of dictionary
  5 print("dict items: ", dict.items())
  6
  7 # Return items of keys
  8 print("dict keys: ", dict.keys())
  9
 10 # return values of dict
 11 print("Value of dict: ", dict.values())
 12
```

```
Value for Age:  6
dict items:  dict_items([('Name', 'Jivin'), ('Age', 6), ('Class', 'First')])
dict keys:  dict_keys(['Name', 'Age', 'Class'])
Value of dict:  dict_values(['Jivin', 6, 'First'])
```

Basic Dictionary Operations -3

```
1  # Concatenate dicts
2  dict1 = {'Name': 'Jivin', 'Age': 6}
3  dict2 = {'Sex': 'male' }
4  dict1.update(dict2)
5  print("dict1.update(dict2) = ", dict1)
```

```
dict1.update(dict2) =  {'Name': 'Jivin', 'Age': 6, 'Sex': 'male'}
```

What is the output of the following and Why?

```
1  d = {'foo': 100, 'bar': 200, 'baz': 300}
2  print(d)
3  d = {}
4  d['foo'] = 100
5  d['bar'] = 200
6  d['baz'] = 300
7  print(d)
```

What is the output of the following and Why?

```
1  d = {'foo': 100, 'bar': 200, 'baz': 300}
2  d['bar':'baz']
```

```
1  d = {'foo': 100, 'bar': 200, 'baz': 300}
2  d.pop('bar')
3  print(d)
```

Suppose x is defined as follows :

```
x = [
    'a',
    'b',
    {
        'foo': 1,
        'bar':
        {
            'x' : 10,
            'y' : 20,
            'z' : 30
        },
        'baz': 3
    },
    'c',
    'd'
]
```

What is the expression involving x that accesses the value 30?

What is the output ?

```
'z' in x[2]['bar']
```

```
17 print(x)
```

```
['a', 'b', {'foo': 1, 'bar': {'x': 10, 'y': 20, 'z': 30}, 'baz': 3}, 'c', 'd']
```

```
1 x[2]['bar']['z']
```

```
30
```

Which of the following could be a valid dictionary key:

- [ ] `['foo', 'bar']`
- [ ] `(3+2j)`
- [ ] `'foo'`
- [ ] `len`
- [ ] `dict(foo=1, bar=2)`
- [ ] `('foo', 'bar')`

```python
d1 = {'foo': 100, 'bar': 200, 'baz': 300}
d2 = {}
d2.update(d1)
print(d2)
```

```python
d1 = {'foo': 100, 'bar': 200, 'baz': 300}
d2 = {}
d2.update(d1)
print(d2)
d2 = d1
print(d2)
```

Practice , Practice and Practice

Makes Perfect