MODULE-2

CHAPTER 4-LISTS

CHAPTER 5-DICTIONARIES AND STRUCTURING DATA

The List Data Type

- A ***list*** is a value that contains multiple values in an ordered sequence.
- ***List value****:* Refers to the list itself, not the values inside the list value.

  Its a value that can be stored in a variable or passed to a function like any other value.
- A ***list value*** looks like: ['cat', 'bat', 'rat', 'elephant']
- List begins with an opening square bracket and ends with a closing square bracket, [].
- ***Items****:* values inside the list are called items. Items are separated with commas.
- *Example:*

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ['cat', 'bat', 'rat', 'elephant']
['cat', 'bat', 'rat', 'elephant']
>>> ['hello', 3.1415, True, None, 42]
['hello', 3.1415, True, None, 42]
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam
['cat', 'bat', 'rat', 'elephant']
```

- The spam variable is assigned only one value: the list value. But the list value itself contains other values.
- The value [] is an empty list that contains no values, similar to '', the empty string.

Data Types

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`

`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict

Getting individual values in a list with indexes

- ***Index:*** The integer inside the square brackets that follows the list is called an index.
- Example:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0]
'cat'
>>> spam[3]
'elephant'
>>> ['cat', 'bat', 'rat', 'elephant'][3]
'elephant'
>>> 'Hello ' + spam[0]
'Hello cat'
>>> 'The ' + spam[1] + ' ate the ' + spam[0] + '.'
'The bat ate the cat.'
```

IndexError

- ***IndexError:*** Python gives an *IndexError* error message if an index is used that exceeds the number of values in the list value.
- Example

```
>>> spam = ['cat', 'bat', 'rat', 'elephant’]
>>> spam[10000]
IndexError: list index out of range
```

TypeError

- *TypeError*: Indexes can be only integer values, not floats
- Example: >>> spam[1.0]

```
Traceback (most recent call last):
File "<pyshell#13>", line 1, in <module> spam[1.0]
TypeError: list indices must be integers, not float
>>> spam[int(1.0)]
'bat'
```

- Lists can also contain other list values. The values in these lists of lists can be accessed using multiple indexes
- If only one index is used, the program prints the full list value at that index.
- The first index dictates which list value to use, and the second indicates the value within the list value
- Example:

```
>>> spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
>>> spam[0]
['cat', 'bat’]
>>> spam[0][1]
'bat’
>>> spam[1][4]
50
```

Negative Indexes:

- While indexes start at 0 and go up, one can also use negative integers for the index.
- The integer value -1 refers to the last index in a list, the value -2 refers to the second-to-last index in a list, and so on
- Example:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant’]
>>> spam[-1]
'elephant’
>>> spam[-3]
'bat’
>>> 'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + '.’
The elephant is afraid of the bat.’
```

- ***Getting sublists with slices:*** Just as an index can get a single value from a list, a slice can get several values from a list, in the form of a new list.
- A slice is typed between square brackets, like an index, but it has two integers separated by a colon.

- Difference between indexes and slices:

- spam[2] is a list with an index (one integer).
- spam[1:4] is a list with a slice (two integers).

- In a slice, the first integer is the index where the slice starts. The second integer is the index where the slice ends
- A slice goes up to, but will not include, the value at the second index. A slice evaluates to a new list value
- Example:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
>>> spam[1:3]
['bat', 'rat']
>>> spam[0:-1]
['cat', 'bat', 'rat']
```

- One can leave out one or both of the indexes on either side of the colon in the slice.
- Leaving out the first index is the same as using 0, or the beginning of the list.
- Leaving out the second index is the same as using the length of the list, which will slice to the end of the list
- Example:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant’]
>>> spam[:2]
['cat', 'bat’]
>>> spam[1:]
['bat', 'rat', 'elephant’]
```

Getting a list's length with len()

The len() function will return the number of values that are in a list value passed to it, just like it can count the number of characters in a string value.

- Example:

```
>>> spam = ['cat', 'dog', 'moose']
>>> len(spam)
3
```

Changing values in a list with indexes

- One use an index of a list to change the value at that index.

Example:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1] = 'aardvark'
>>> spam
['cat', 'aardvark', 'rat', 'elephant']
>>> spam[2] = spam[1]
>>> spam
['cat', 'aardvark', 'aardvark', 'elephant']
>>> spam[-1] = 12345
>>> spam
['cat', 'aardvark', 'aardvark', 12345]
```

List concatenation and List Replication

- The + operator can combine two lists to create a new list value in the same way it combines two strings into a new string value
- The * operator can also be used with a list and an integer value to replicate the list.
- Example:

```
>>> [1, 2, 3] + ['A', 'B', 'C’]
[1, 2, 3, 'A', 'B', 'C’]
>>> ['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z’]
>>> spam = [1, 2, 3]
>>> spam = spam + ['A', 'B', 'C’]
>>> spam
[1, 2, 3, 'A', 'B', 'C’]
```

Removing values from lists with del statement

- *del* statement delete values at an index in a list.
- *All of the values in the list after the deleted value will be moved up one index*
- Example: >>> spam = ['cat', 'bat', 'rat','elephant’]

```
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant’]
```

Working with lists

- Create many individual variables to store a group of similar values as below

```
catName1 = 'Zophie’
catName2 = 'Pooka’
catName3 = 'Simon’
```

-
```
print('Enter the name of cat 1:’)
catName1 = input()
print('Enter the name of cat 2:’)
catName2 = input()
….
….
print('The cat names are:’)
print(catName1 + ' ' + catName2 + ' ' + catName3 + ' ' + catName4 + ' '
+ catName5 + ' ‘ +      catName6)
```

Using List

- Instead of using multiple, repetitive variables, one can use a single variable that contains a list value
- Eg:

```
catNames = []
while True:
    print('Enter the name of cat ' + str(len(catNames) + 1) + ' (Or enter nothing to stop.):’)
    name = input()
    if name == '': break
        catNames = catNames + [name] # list concatenation print('The cat names are:’)
    for name in catNames:
        print(' ' + name)
```

- ***Using for loops with lists***
- Technically, a for loop repeats the code block once for each value in a list or list-like value.
- Say for example, the return value from range(4) is a list-like value that Python considers similar to [0, 1, 2, 3]. Or for i in [0, 1, 2, 3]: print(i). Both returns the same output.
- A common Python technique is to use range(len(someList)) with a for loop to iterate over the indexes of a list
- Example:

```
>>> supplies = ['pens', 'staplers', 'flame-throwers', 'binders']
>>> for i in range(len(supplies)):
        print('Index ' + str(i) + ' in supplies is: ' + supplies[i])
```

O/P: Index 0 in supplies is: pens

Index 1 in supplies is: staplers

Index 2 in supplies is: flame-throwers

Index 3 in supplies is: binders

- 2/11/2022

The in and not in operators

- *in* and *not in* operators : To determine whether a value is in or not in the list used.
- *in* and *not in* are used in expressions and connect two values: a value to look for in a list the list where it may be found.
- These expressions will evaluate to a Boolean value.
- Example:

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas’]
True
>>> spam = ['hello', 'hi', 'howdy', 'heyas’]
>>> 'cat' in spam
False
>>> 'howdy' not in spam
False
>>> 'cat' not in spam
True
```

The Multiple Assignment Trick

- The multiple assignment trick is a shortcut that lets you assign multiple variables with the values in a list in one line of code.

- Example:

```
>>> cat = ['fat', 'black', 'loud’]
>>> size = cat[0]
>>> color = cat[1]
>>> disposition = cat[2]
```

- Instead one can use:
- Example: >>> cat = ['fat', 'black', 'loud’]
  >>> size, color, disposition = cat
- The number of variables and the length of the list must be exactly equal, or Python gives an ValueError

- Example: >>> cat = ['fat', 'black', 'loud’]
  >>> size, color, disposition, name = cat
  Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module> size, color, disposition, name = cat
  ValueError: need more than 3 values to unpack

Augmented Assignment Operators

- Example:
```
>>> spam = 42
>>> spam = spam + 1
>>> spam
43
```
- As shortcut, Augmented Operators can be used
- Example:
```
>>> spam = 42
>>> spam += 1
>>> spam
43
```

- There are augmented assignment operators for the +, -, *, /, and % operators
- **Augmented assignment statement Equivalent assignment statement**

spam += 1	spam = spam + 1
spam -= 1	spam = spam - 1
spam *= 1	spam = spam * 1
spam /= 1	spam = spam / 1
spam %= 1	spam = spam%1

- The += operator can also do string and list concatenation, and the *=operator can do string and list replication

- Example:

```
>>> spam = 'Hello’
>>> spam += ' world!’
>>> spam
'Hello world!’
>>> bacon = ['Zophie’]
>>> bacon *= 3
>>> bacon
['Zophie', 'Zophie', 'Zophie']
```

Methods

- A method is the same thing as a function, except it is “called on” a value.
- Example: spam=[‘hello’,2.3,43]

  spam.Index(‘hello’)
- The method part comes after the value, separated by a period.

Finding a Value in a List with the index() Method

- index() method: method that can be passed a value.

  if that value exists in the list, the index of the value is returned.

  If the value isn't in the list, then Python produces a ValueError error

Example:

```
>>> spam = ['hello', 'hi', 'methods', 'functions']
>>> spam.index('hello')
0
>>> spam.index('how how how')
ValueError: 'howdy how how' is not in list
```

- When there are duplicates of the value in the list, the index of its first appearance is returned
- Example:

```
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
>>> spam.index('Pooka')
1
```

Append() and insert() methods

- Append() and insert() methods: Used to add new values to a list.

Append(): Adds argument to the end of the list.

- Example:

```
>>> spam = ['cat', 'dog', 'bat’]
>>> spam.append('moose’)
>>> spam
['cat', 'dog', 'bat', 'moose']
```

- The insert() method can insert a value at any index in the list.
- The first argument to insert() is the index for the new value, and the second argument is the new value to be inserted
- Example:

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken')
>>> spam
['cat', 'chicken', 'dog', 'bat']
```

- Methods belong to a single data type.
- The append() and insert() methods are list methods and can be called only on list values, not on other values such as strings or integers
- Example:

```
>>> eggs = 'hello'
>>> eggs.append('world')
AttributeError: 'str' object has no attribute 'append'
```

Removing Values from Lists with remove()

- The remove() method : passed the value to be removed from the list it is called on
- Example:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']
```

- Attempt to delete value in the list that doesn't exist results in ValueError
- Example:

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('chicken')
ValueError: list.remove(x): x not in list
```

- If the value appears multiple times in the list, only the first instance of the value will be removed.
- Example:

```
>>> spam = ['cat', 'bat', 'rat', 'cat', 'hat', 'cat']
>>> spam.remove('cat')
>>> spam
['bat', 'rat', 'cat', 'hat', 'cat']
```

- The del statement is good to use when you know the index of the value you want to remove from the list.

```
>>> a=[1,2,3,4]
>>> a
[1, 2, 3, 4]
>>> del[3]
SyntaxError: incomplete input
>>> del a[3]
>>> a
[1, 2, 3]
>>>
```

- The remove() method is good when you know the value you want to remove from the list.

- ***Sorting the Values in a List with the sort() Method***
- Lists of number values or lists of strings can be sorted with the sort() method
- Example:

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants’]
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants’]
```

- To have sort(),sort the values in reverse order: pass True for the reverse keyword argument
- Example:

```
>>> spam.sort(reverse=True)
>>> spam
['elephants', 'dogs', 'cats', 'badgers', 'ants’]
```

Things to note about the sort() method:

1. the sort() method sorts the list in place

spam=spam.sort() will not return any thing.

2. cannot sort lists that have both number values and string values in them, since Python doesn't know how to compare these values.

Example: >>> spam = [1, 3, 2, 4, 'Alice', 'Bob']
>>> spam.sort()
TypeError: unorderable types: str() < int()

3. sort() uses “ASCIIbetical order” rather than actual alphabetical order for sorting strings.

- This means that uppercase letters come before lowercase letters
- Example: >>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats’]

```
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats’]
```

- To sort the values in regular alphabetical order, pass str.lower for the key keyword argument in the sort() method call.
- Example: >>> spam = ['a', 'z', 'A', 'Z’]

```
>>> spam.sort(key=str.lower)
#To treat all the items as if in lowercase
>>> spam
['a', 'A', 'z', 'Z']
```

```
>>> a=['z','A','w','a','Z']
>>> a.sort()
>>> a
['A', 'Z', 'a', 'w', 'z']
>>> a.sort(key=str.lower)
>>> a
['A', 'a', 'w', 'Z', 'z']
```

- ***Example Program: Magic 8 Ball with a List***
- import random

```
messages = ['It is certain’,
             'It is decidedly so’,
            'Yes definitely’,
            'Reply hazy try again’,
           'Ask again later’,
            'Concentrate and ask again’,
            'My reply is no’,
            'Outlook not so good’,
            'Very doubtful’]
           print(messages[random.randint(0, len(messages) - 1)])
```

List-like Types: Strings and Tuples

- Like lists, strings are also a data type that represent ordered sequences of values.

- Many things that can be done with lists can be also done with strings like

  : indexing; slicing; and using them with for loops, with len(), and with

  the in and not in operators

- Example:

```
>>> name = 'Zophie’
>>> name[0]
'Z’
>>> name[-2]
'i’
>>> name[0:4]
'Zoph’
```

```
>>> 'Zo' in name
 True
 >>> 'z' in name
 False
 >>> 'p' not in name
 False
 >>> for i in name:
          print('* * * ' + i + ' * * *')

* * * Z * * *
* * * o * * *
* * * p * * *
* * * h * * *
* * * i * * *
* * * e * * *
```

Mutable and Immutable Data Types:

- A list value is a mutable data type:

  It can have values added, removed, or changed

- A string is immutable: It cannot be changed

- Example:

```
>>> name = 'Zophie a cat’
>>> name[7] = 'the’
TypeError: 'str' object does not support item assignment
```

- The proper way to “mutate” a string is to use slicing and concatenation to build a new string by copying from parts of the old string.
- Example:
```
>> name = 'Zophie a cat’
>>> newName = name[0:7] + 'the' + name[8:12]
>>> name
'Zophie a cat’
>>> newName
'Zophie the cat’
```
- >>> eggs = [1, 2, 3]
- >>> eggs = [4, 5, 6]
- >>> eggs
- [4, 5, 6] # new list values is entirely overwriting old list values

Comparison

Key	List	Tuple
Type	List is mutable.	Tuple is immutable.
Iteration	List iteration is slower and is time consuming.	Tuple iteration is faster.
Appropriate for	List is useful for insertion and deletion operations.	Tuple is useful for readonly operations like accessing elements.
Memory Consumption	List consumes more memory.	Tuples consumes less memory.
Methods	List provides many in-built methods.	Tuples have less in-built methods.
Error prone	List operations are more error prone.	Tuples operations are safe.

The Tuple Data Type:

- The tuple data type is almost identical to the list data type, except in two ways:

1. tuples are typed with parentheses, (and), instead of square brackets, [and]
   Example: >>> eggs = ('hello', 42, 0.5)
   >>> eggs[0]
   'hello’
   >>> eggs[1:3]
   (42, 0.5)
   >>> len(eggs)
   3

Tuples, like strings, are immutable. Tuples cannot have their values modified, appended, or removed.

Example: >>> eggs = ('hello', 42, 0.5)

>>> eggs[1] = 99

TypeError: 'tuple' object does not support item assignment

- Example: >>> type(('hello',))

<class 'tuple’>

>>> type(('hello’))

<class 'str'>

Converting Types with the list() and tuple() Functions

- The functions list() and tuple() will return list and tuple versions of the values passed to them.
- Example:

```
>>> tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
>>> list(('cat', 'dog', 5))
['cat', 'dog', 5]
>>> list('hello’)
['h', 'e', 'l', 'l', ‘o’]
```

Introduction to Python references

```
>>> a=100
>>> print(id(a))
2589792996688
>>> b=100
>>> print(id(b))
2589792996688
>>> c=101
>>> print(id(c))
2589792996720
>>> d=101
>>> print(id(d))
2589792996720
>>>
```

References

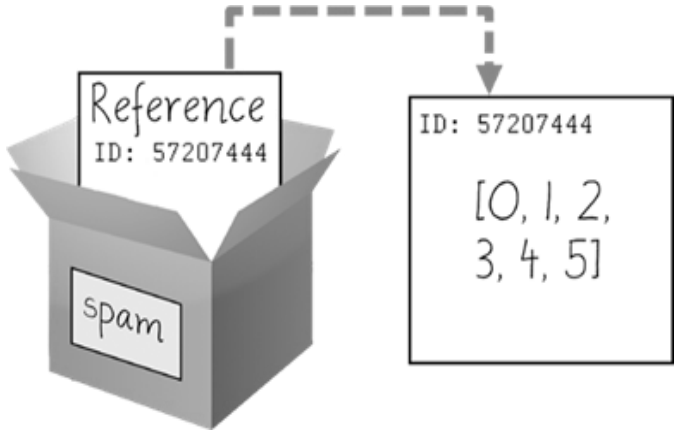
- variables store strings and integer values
- Example:
```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```
- This is because spam and cheese are different variables that store different values.

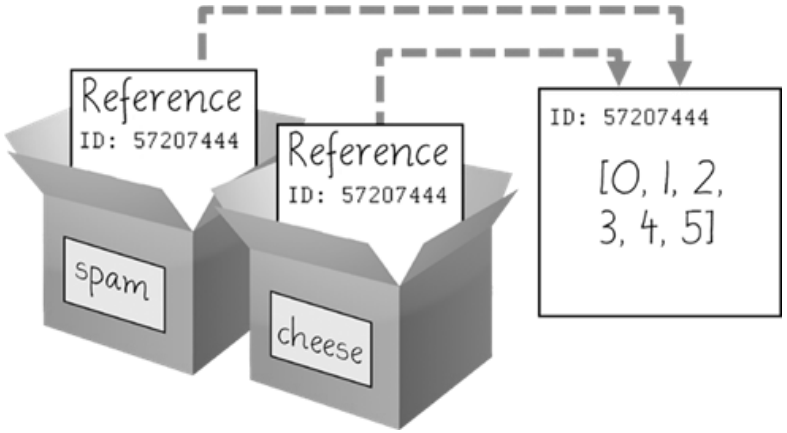
References in list

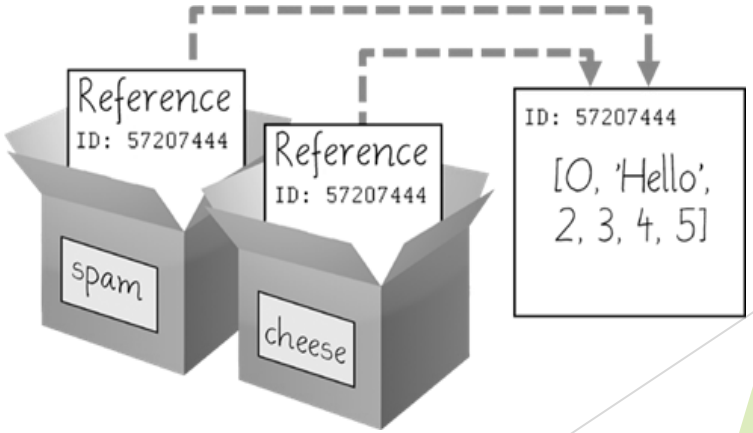
```
>>> t1=[1,2,3,4]
>>> id(t1)
2523790522688
>>> t2=t1
>>> t2
[1, 2, 3, 4]
>>> id(t2)
2523790522688
>>> t2[1]='Hello'
>>> t2
[1, 'Hello', 3, 4]
>>> id(t2)
2523790522688
```

- When you assign a list to a variable, you are actually assigning a list reference to the variable.
- A reference is a value that points to some bit of data, and a list reference is a value that points to a list.
- Example:

```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> cheese = spam
>>> cheese[1] = 'Hello!'
>>> spam
[0, 'Hello!', 2, 3, 4, 5]
>>> cheese
[0, 'Hello!', 2, 3, 4, 5]
```







- Variables will contain references to list values rather than list values themselves.
- But for strings and integer values, variables simply contain the string or integer value.
- Python uses references whenever variables must store values of mutable data types, such as lists or dictionaries.
- For values of immutable data types such as strings, integers, or tuples, Python variables will store the value itself

Passing References

- When a function is called, the values of the arguments are copied to the parameter variables.
- For lists, this means a copy of the reference is used for the parameter.

- Example:

```
def eggs(someParameter):
        someParameter.append('Hello’)
spam = [1, 2, 3]
eggs(spam)
print(spam)
```

O/P: [1, 2, 3, 'Hello']

Passing Reference

how does arguments get passed to functions?

- When a function is called, the values of the arguments are copied to the parameter variables. For lists, this means a copy of the reference is used for the parameter.

- Example:

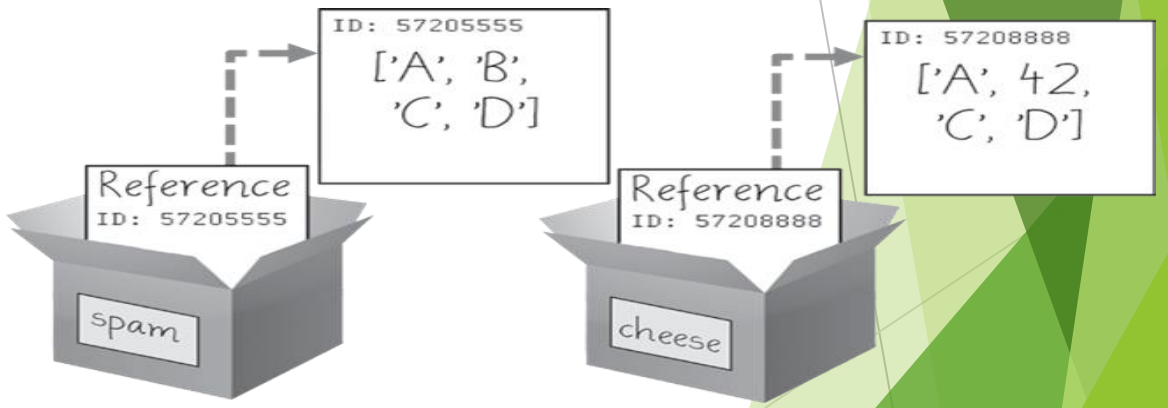
```
def eggs(someParameter):
    someParameter.append('Hello’)

spam = [1, 2, 3]
eggs(spam)
print(spam)
```

The copy Module's copy() and deepcopy() Functions

- if the function modifies the list or dictionary that is passed, one may not want these changes in the original list or dictionary value, then python provides a Module named ***copy*** that provides both copy() and deecopy() functions.

- **copy.copy(), can be used to make a duplicate copy of a mutable value like a list or dictionary, not just a copy of a reference.**

```
>>> import copy
>>> spam=[1,2,3,4]
>>> trial=copy.copy(spam)
>>> id(spam)
1499859425856
>>> id(trial)
1499859423360
>>> trial[1]='hello'
>>> spam
[1, 2, 3, 4]
>>> trial
[1, 'hello', 3, 4]
>>>
```



Chapter-5: Dictionaries and Structuring Data

The Dictionary Data Type

- A dictionary is a collection of many values.
- But unlike indexes for lists, indexes for dictionaries can use many different data types, not just integers.
- Indexes for dictionaries are called keys, and a key with its associated value is called a key-value pair.
- a dictionary is typed with braces, {}.
- Example: >>> myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud’}

myCat is a dictionary variable.

Keys are: size, color, disposition

Values are: fat, gray, loud

Accessing values through keys:

- Example: >>> myCat['size’]

  'fat’

  >>> 'My cat has ' + myCat['color'] + ' fur.’

  'My cat has gray fur.’

- Dictionaries can still use integer values as keys, just like lists use integers for indexes, but they do not have to start at 0 and can be any number.
- Example:

```
>>> spam = {12345: 'Luggage Combination', 42: 'The Answer'}
```

- ***Dictionaries vs. Lists:***
- Items in dictionaries are unordered
- The first item in a list spam would be spam[0]. But there is no first item in a dictionary.
- While the order of items matters in list, it does not matter in what order the key-value pairs are typed in a dictionary.
- Example:

```
>>> spam = ['cats', 'dogs', 'moose']
>>> bacon = ['dogs', 'moose', 'cats']
>>> spam == bacon
False
>>> eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
>>> ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
>>> eggs == ham
True
```

- Trying to access a key that does not exist in a dictionary will result in a KeyError error message.

- Example:
```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> spam['color’]
spam['color’]
KeyError: 'color’
```

-

- Arbitrary values for the keys allows to organize data in powerful ways.
- Example:

```
birthdays = {'Alice': 'Apr 1', 'Bob': 'Dec 12', 'Carol': 'Mar 4’}
while True:
    print('Enter a name: (blank to quit)’)
    name = input()
    if name == ‘’:
        break
    if name in birthdays:
        print(birthdays[name] + ' is the birthday of ' + name)
    else:
        print('I do not have birthday information for ' + name)
        print('What is their birthday?’)
        bday = input()
        birthdays[name] = bday
        print('Birthday database updated.’)
```

- Output:

```
Enter a name: (blank to quit)
Eve
I do not have birthday information for Eve
What is their birthday?
Dec 5
Birthday database updated.
Enter a name: (blank to quit)
```

The keys(), values(), and items() Methods:

- Three dictionary methods that will return list-like values of the dictionary's keys, values, or both keys and values: keys(), values(), and items().
- Values returned by these methods are not true lists.
- They cannot be modified and no append() method.
- dict_keys, dict_values, and dict_items are the datatypes of Keys(), values() and items() respectively.

A for loop iterates over keys, or values or both keys and values(items)

Example:

```
>>> spam = {'color': 'red', 'age': 42}
>>> for v in spam.values():
        print(v)
red
42
>>> for k in spam.keys():
        print(k)
color
age
>>> for i in spam.items():
        print(i)
('color', 'red')        #tuples of the key and value.
('age', 42)
```

- 10/11

- To return true list, pass its list-like return value to the list() function.
- Example:

```
>>> spam = {'color': 'red', 'age': 42}
>>> spam.keys()
dict_keys(['color', 'age’])
>>> list(spam.keys())
['color', 'age’]
```

- Multiple assignment trick in a for loop to assign the key and value to separate variables.
- Example:

```
>>> spam = {'color': 'red', 'age': 42}
>>> for k, v in spam.items():
print('Key: ' + k + ' Value: ' + str(v))
Key: age Value: 42
Key: color Value: red
```

Checking Whether a Key or Value Exists in a Dictionary

- One can use in and not in operator to check whether a value exists in a list.
- Example:

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> 'name' in spam.keys()
True
>>> 'Zophie' in spam.values()
True
>>> 'color' in spam.keys()
False
>>> 'color' not in spam.keys()
True
>>> 'color' in spam
False
```

- ***The get() Method***
- Example:

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
KeyError: 'eggs
```

- To check whether a key exists in a dictionary before accessing that key's value, get() method is used.
- Dictionaries have a get() method that takes two arguments: the key of the value to retrieve and a fallback value to return if that key does not exist.
- Example:

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
'I am bringing 0 eggs.'
```

The setdefault() Method

- One can set a value in a dictionary for a certain key only if that key does not already have a value.
- Example:

```
spam = {'name': 'Pooka', 'age': 5}
if 'color' not in spam:
spam['color'] = 'black'
```

- The setdefault() method offers a way to do this in one line of code.
- The first argument passed to the method is the key to check for, and the second argument is the value to set at that key if the key does not exist.
- If the key does exist, the setdefault() method returns the key's value.
- Example:

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

- 1. **Write a python program to enter a temperature in Celsius into Fahrenheit by using a function.**
- **2. Write a python program to accept the username “Admin” as the default argument and password 123 entered by the user to allow login into the system.**
- **3. Write a python program to demonstrate the concept of variable length argument to calculate the sum and product of the first 10 numbers**
- **4. Write a python program to find maximum and minimum numbers among the given 4 numbers using list**
- **5.** Write a function that receives two string arguments and checks whether they are same-length strings. (returns True in this case otherwise False).

Solutions : 1

```
def tempConvert():
    cels = float(input("Enter temperature in celsius: "))
    fh = (cels * 9/5) + 32
    print('%.2f Celsius is: %0.2f Fahrenheit' %(cels, fh))
    tempConvert()
```

2.

```
def user_pass(password,username="Admin"):
    if password=='123':
        print("You have logged into system")
    else:
        print("Password is incorrect!!!!!!")
password=input("Enter the password:")
user_pass(password)
```

3.

```
def sum10(*n):
    total=0
    for i in n:
        total=total + i
    print("Sum of first 10 Numbers:",total)
sum10(1,2,3,4,5,6,7,8,9,10)

def product10(*n):
    pr=1
    for i in n:
       pr=pr * i
    print("Product of first  10 Numbers:",pr)
product10(1,2,3,4,5,6,7,8,9,10)
```

4.

```
def find_max():
    l=[]
    max1=0
    for i in range(4):
        n=int(input("Enter number into list:"))
        l.append(n)
    print("The list is:",l)
    for i in l:
        if i>max1:
            max1=i
    print("Max:",max1)
```

4-using sort

```
def find_max():
    l=[]
    max1=0
    for i in range(4):
        n=int(input("Enter number into list:"))
        l.append(n)
    l.sort()
    print("Max:",l[-1])
```

5.

```
def samelength(txt1,txt2):
  if len(txt1)==len(txt2):
    return True
  else:
    return False


text1=input("Enter the text1:")
text2=input("Enter the text2:")
print("The string is same length?",samelength(text1,text2))
```

Example:

```
message = 'It was a bright cold day in April.’
count = {}
for character in message:
 count.setdefault(character, 0)
 count[character] = count[character] + 1
 print(count)
```

Nested Dictionaries'

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
print(myfamily)
```

- ***Pretty Printing***
- For a cleaner display of the items in a dictionary than what print() provides, import the pprint module and access to the pprint() and pformat() functions that will “pretty print” a dictionary’s values.
- Example:

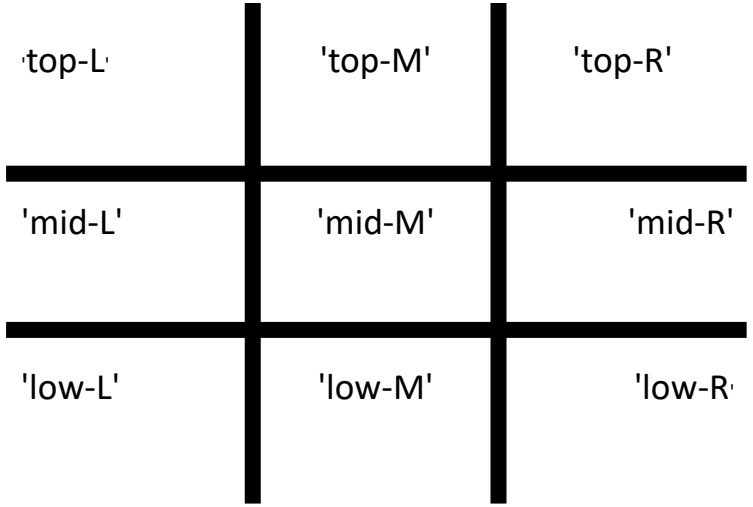
```
import pprint
message = 'It was a bright cold day in April.’
count = {}
for character in message:
    count.setdefault(character, 0)
    count[character] = count[character] + 1
pprint.pprint(count)
```

Packing OF TUPLE

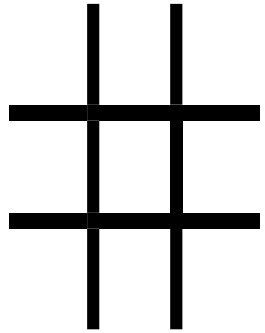
- A tuple can also be created without using parentheses. This is known as tuple packing.

```
my_tuple = 3, 4.6, "dog"
print(my_tuple)

# tuple unpacking is also possible
a, b, c = my_tuple

print(a)      # 3
print(b)      # 4.6
print(c)      # dog
```

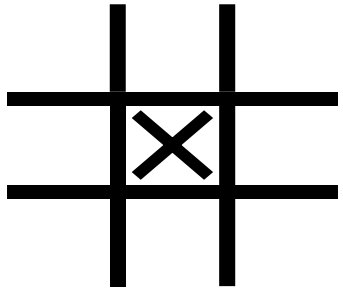
- ***Using Data Structures to Model Real-World Things***
- Real world things can be modelled using lists and dictionaries.
- In this regard, the first example of ***A Tic-Tac-Toe Board***
- Board can be represented with a dictionary
- Nine slots in a board.
- Each slot is represented by string-value keys
- Values to use in each slot are 'X', 'O', or ' '
- Dictionary is a data structure that represents a tic-tac-toe board



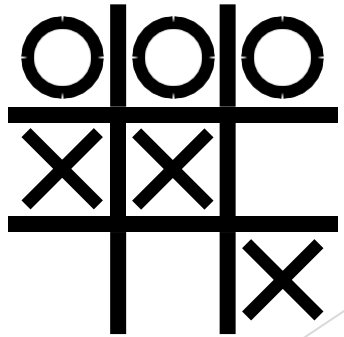
- theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ‘,
  'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ‘,
  'low-L': ' ', 'low-M': ' ', 'low-R': ' ‘}



- theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ‘,
  ‘mid-L': ' ', 'mid-M': 'X', 'mid-R': ' ‘,
  'low-L': ' ', 'low-M': ' ', 'low-R': ' ‘}



- theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O’,
  'mid-L': 'X', 'mid-M': 'X', 'mid-R': ' ‘,
  'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}



- theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ‘,
  'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ‘,
  'low-L': ' ', 'low-M': ' ', 'low-R': ' ‘}

```
def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R’])
    print('-+-+-’)
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R’])
    print('-+-+-’)
    print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R’])
    printBoard(theBoard)
```

```
  |  |
-+-+-
  |  |
-+-+-
  |  |
```

- theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O', 'mid-L': 'X', 'mid-M’:
  'X', 'mid-R': ' ', 'low-L': ' ', 'low-M': ' ', 'low-R': 'X’}

```
def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R’])
    print('-+-+-’)
    print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R’])
    print('-+-+-’)
    print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R’])
    printBoard(theBoard)
```

```
O|O|O
-+-+-
X|X|
-+-+-
  |  |X
```

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ', 'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ', 'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
  def printBoard(board):
   print(board['top-L'] + '|' + board['top-M'] + '|' + board['top-R'])
   print('-+-+-')
   print(board['mid-L'] + '|' + board['mid-M'] + '|' + board['mid-R'])
   print('-+-+-')
  print(board['low-L'] + '|' + board['low-M'] + '|' + board['low-R'])
  turn = 'X'
  for i in range(9):
  printBoard(theBoard)
  print('Turn for ' + turn + '. Move on which space?')
  move = input()
  theBoard[move] = turn
   if turn == 'X':
  turn = 'O'
  else:
  turn = 'X'
  printBoard(theBoard)
```

- ***Nested Dictionaries and Lists***
- For modelling more complicated things, one needs dictionaries and lists that contain other dictionaries and lists.
- Lists are useful to contain an ordered series of values, and dictionaries are useful for associating keys with values.

- Example:

```
allGuests = {'Alice': {'apples': 5, 'pretzels': 12},
             'Bob': {'ham sandwiches': 3, 'apples': 2},
             'Carol': {'cups': 3, 'apple pies': 1}}
def totalBrought(guests, item):
    numBrought = 0
    for k, v in guests.items():
    numBrought = numBrought + v.get(item, 0)
    return numBrought
print('Number of things being brought:’)
print(' - Apples ' + str(totalBrought(allGuests, 'apples')))
print(' - Cups ' + str(totalBrought(allGuests, 'cups')))
print(' - Cakes ' + str(totalBrought(allGuests, 'cakes')))
print(' - Ham Sandwiches ' + str(totalBrought(allGuests, 'ham sandwiches')))
print(' - Apple Pies ' + str(totalBrought(allGuests, 'apple pies')))
```