Assignment No. 4

Submitted By: Sachin Dodake

1. What exactly is []?

Ans-1

- It is `Index bracket/square bracket` ([]) and used to represent the LIST in python.
- `Index bracket/square bracket` ([]) have many uses in Python. First, they are used to define `list literals`, allowing you to `declare a list` and its contents in your program.
- `Index bracket/square bracket` are also used to write expressions that evaluate to a single item within a list, or a single character in a string.
- For `lists` and other mutable sequences (but not strings), you can overwrite a value at a particular index using the assignment operator (=).
- `Negative numbers` inside of index brackets cause Python to start counting from the end of the sequence, instead of from the beginning. For example, the expression x[-1] evaluates to the last item of list x, x[-2] evaluates to the second-to-last item of list x, and so forth.
- `Index bracket/square bracket` are used to retrieve or set the value for a given key in a dictionary. For example, the expression x[a] evaluates to whatever the value for key a is in dictionary x. The statement x[a] = b will set the value for key a in dictionary x to a new value b (overwriting any existing value).
- Specifying an index beyond the bounds of the sequence raises an `IndexError exception`. Attempting to retrieve the value for a key that does not exist in a dictionary raises a `KeyError exception`.

2. In a list of values stored in a variable called spam, how would you assign the value 'hello' as the third value? (Assume [2, 4, 6, 8, 10] are in spam.)

Ans-2

In [17]:
```
# given list is spam
spam = [2,4,6,8,10]       # indeces (0,1,2,3,4) , third postion mean 2nd index value

spam[2] = "Hello"         # assigning 'Hello' value as a third value, i.e. 2nd index value (0,1,2,3,4)

print(f"The all elements of list are:   {spam}")                  # printing newly created list with replac
print(f"The 3rd elements of list is:    {spam[2]}")             # printing 3rd value if list
```
```
The all elements of list are:   [2, 4, 'Hello', 8, 10]
The 3rd elements of list is:    Hello
```

Let's pretend the spam includes the list ['a', 'b', 'c', 'd'] for the next three queries i.e. Que3, Que4, Que5

3. What is the value of spam[int(int('3' * 2) / 11)]?

Ans-3

- The value of `spam[int(int('3' * 2) / 11)]` will be `d` as shown below,

In [28]:
```
# the given list is as below

spam = ['a', 'b', 'c', 'd']

result = spam[int(int('3' * 2) / 11)]                # saving output in 'result'  variable.

print(f"The expected output is: {result}")       # printing the result value.
```
```
The expected output is: d
```

4. What is the value of spam[-1]?

Ans-4

- The value of `spam[-1]` will be `d` as shown below,

In [29]:
```
# the given list is as below

spam = ['a', 'b', 'c', 'd']

result = spam[-1]                  # saving output in 'result'  variable.

print(f"The expected output is: {result}")       # printing the result value.
```
```
The expected output is: d
```

5. What is the value of spam[:2]?

Ans-5

- The value of `spam[:2]` will be `['a', 'b']` as shown below,

In [42]:
```
# the given list is as below

spam = ['a', 'b', 'c', 'd']              # given list-bacon

result = spam[:2]                  # saving output in 'result'  variable.

print(f"The expected output is: {result}")       # printing the result value.
```
```
The expected output is: ['a', 'b']
```

Let's pretend bacon has the list [3.14, 'cat,' 11, 'cat,' True] for the next three questions i.e. Que6, Que7, Que8

6. What is the value of bacon.index('cat')?

Ans-6

- The value of `bacon.index('cat')` will be `1` as shown below,

In [49]:
```
# the given list is as below

bacon = [3.14, 'cat', 11, 'cat', True]  # given list-bacon

output = bacon.index('cat')                  # saving index of 'cat' in output variable

print(f"The expected output is: {output}")       # printing the result value.
```
```
The expected output is: 1
```

7. How does bacon.append(99) change the look of the list value in bacon?

Ans-7

- The value of `bacon.append(99)` will be `[3.14, 'cat', 11, 'cat', True, 99]` as shown below,

In [50]:
```
# the given list is as below

bacon = [3.14, 'cat', 11, 'cat', True]      # given list-bacon

bacon.append(99)                 # addding new element 99 in existing list-bacon

print(f"The updated list is: {bacon}")       # printing the new updated list-bacon.
```
```
The updated list is: [3.14, 'cat', 11, 'cat', True, 99]
```

8. How does bacon.remove('cat') change the look of the list in bacon?

Ans-8

- The value of `bacon.remove('cat')` will be `[3.14, 11, 'cat', True]` as shown below,

In [51]:
```
# the given list is as below

bacon = [3.14, 'cat', 11, 'cat', True]               # given list-bacon

bacon.remove('cat')                         # removing  element 'cat' from existing list-bacon

print(f"The updated list is: {bacon}")       # printing the new updated list-bacon.
```
```
The updated list is: [3.14, 11, 'cat', True]
```

9. What are the list concatenation and list replication operators?

Ans-9

`List Concatenation:`

- **List Concatenation** is an operation where the elements of one list are added at the end of another list.
- The list concatenation operator is `(+)`
- When `(+)` appears between two lists, the expression will be evaluated as a new list that contains the elements from both lists. The elements in the list on the left of + will appear first, and the elements on the right will appear last.
- `Example:`

In [64]:
```
letter_list = ['A', 'B', 'C']
number_list = [1, 2, 3]

new_list = (letter_list + number_list)

print(f"The new concatenated list is: {new_list}")
```
```
The new concatenated list is: ['A', 'B', 'C', 1, 2, 3]
```

`List Replication:`

- **List Replication** is an operation where the list repeat a specific number of times..
- The list concatenation operator is `(*)`
- When `(*)` appears between a list and an integer, the expression will be evaluated as a new list that consists of several copies of the original list concatenated together. The number of copies is set by the integer.
- `Example:`

In [67]:
```
letter_list = ['A', 'B']
number   = 3

new_list = (letter_list * number)

print(f"The new replicated list is: {new_list}")
```
```
The new replicated list is: ['A', 'B', 'A', 'B', 'A', 'B']
```

10. What is difference between the list methods append() and insert()?

Ans-10

`Difference Between append() and insert():`

- The difference between the two methods is that .append() adds an item to the end of a list, whereas .insert() inserts and item in a specified position in the list.
- **Example:**

`append():`

- It adds an element at the end of the list. The argument passed in the append function is added as a single element at end of the list and the length of the list is increased by 1.
- **Syntax:**

```
list_name.append(element)
```

- **Example:**

In [71]:
```
a = ["apple", "banana", "cherry"]   # defining first list
b = ["Ford", "BMW", "Volvo"]
a.append(b)

print(a)
```
```
['apple', 'banana', 'cherry', ['Ford', 'BMW', 'Volvo']]
```

`insert():`

- Python List insert() method inserts a given element at a given index in a list using Python.
- **Syntax:**

```
list.insert(position, element)
            # position-position where we wants to insert new element
            # element-element which we wants to insert in old list.
```

- **Example:**

In [72]:
```
fruits = ['apple', 'banana', 'cherry']

fruits.insert(1, "orange")

print(fruits)
```
```
['apple', 'orange', 'banana', 'cherry']
```

11. What are the two methods for removing items from a list?

Ans-11

- Python `List data-type` helps you to store items of different data types in an `ordered sequence`. The data is written inside `square brackets ([])`, and the values are separated by `comma(,)`.
- In Python, there are many methods available to `remove` an element from a given list.

```
i) remove(),
ii) pop(),
iii) clear() .
```

- Besides the list methods, you can also use a del keyword to remove items from a list.

`i) remove():`

- Python removes () method is a built-in method available with the list. It helps to remove the given very first element matching from the list.
- **Syntax:**

```
list.remove(element)
```

- **ReturnValue:**

```
There is no return value for this method.
```

`ii) pop():`

- The pop() method removes an element from the list based on the index given.
- **Syntax:**

```
list.pop(index)
```

- **ReturnValue:**

```
    The pop() method will return the element removed based on the index given. The
final list is also updated and will not have the element.
```

`iii) clear():`

- The clear() method will remove all the elements present in the list.
- **Syntax:**

```
list.clear()
```

- **ReturnValue:**

```
Ther is no return value. The list() is emptied using clear() method.
```

In [76]:
```
# combine  example on remove(), pop(), clear() methods to remove elements from list

list_name = ['Siya', 'Tiya',  'Riya', 'Shyam', 'INDIA']                          # defining name list
list_num = [0,1,2,3,4,5,6,7,8,9,10]                                                     # defining number
list_city = [['Pune', 'Delhi',  'Jaipur', 'Nagpur', 'Nashik'] ]              # defining city list

list_name.remove('INDIA')        # applying remove ()method to remove element "INDIA"
list_num.pop(5)                            # applying pop() method to remove 5th element  ie 5
list_city.clear()                                # applying clear() method to remove all elements

print(f"The updated name list is: {list_name}")              # it will print list_name without element 'IND
print(f"\nThe updated number list is: {list_num}")      # it will print list_num without 5th index element '
print(f"\nThe updated city list is: {list_city}")             # it will print list_city without elements
```
```
The updated name list is: ['Siya', 'Tiya', 'Riya', 'Shyam']

The updated number list is: [0, 1, 2, 3, 4, 6, 7, 8, 9, 10]

The updated city list is: []
```

12. Describe how `list values` and `string values` are identical.

Ans-12

- The similarity between `Lists Values` and `Strings Values` in Python is that both are sequences. The differences between them are that firstly, Lists are mutable but Strings are immutable. Secondly, elements of a list can be of different types whereas a String only contains characters that are all of String type.
- A `string` is a `sequence of characters` between single or double quotes. A `list` is a `sequence of items`, where each item could be anything (an integer, a float, a string, etc).
- Both `strings` and `lists` have lengths: a string's length is the number of characters in the string; a list's length is the number of items in the list.
- Each character in a `string` as well as each item in a `list` has a `position`, also called an **index**. In python, positions start at 0, so the "H" in the string = 'Hello' is at position 0, and the "o" is at position 4.
- Any sequence in python can be used in a for loop. For `strings`, we can either loop over characters in the string or indices (0 to len(S)-1). For `lists`, we can either loop over items in the list or indices.
- We can use the `accumulator pattern` (increament/decreament) to grow/create both a `string` and a `list`.

13. What's the difference between `tuples and lists`?

Ans-13

`Lists:`

- List are mutable
- Iterations are time-consuming.
- Inserting and deleting items is easier with a list.
- Lists consume more memory.
- Lists have several built-in methods.
- A unexpected change or error is more likely to occur in a list.

`Tuples:`

- Tuples are immutable.
- Iterations are comparatively Faster
- Accessing the elements is best accomplished with a tuple data type.
- Tuple consumes less than the list.
- A tuple does not have many built-in methods because of immutability.
- In a tuple, changes and errors don't usually occur because of immutability.

14. How do you type a `tuple value` that only contains the integer 42?

Ans-14

- The tuple value that contains only integer 42 can be written as below,

In [80]:
```
tuple = (42,)

print(f"The tuple value is : {tuple}")
```
```
The tuple value is : (42,)
```

15. How do you get a `list value's` `tuple form`? How do you get a `tuple value's` `list form`?

Ans-15

`List-to-Tuple Conversion:`

- The three most common methods to convert `list-values` into `tuple-form` are listed below:

```
i) Using tuple() builtin function,
ii) Using loop inside the tuple,
iii) Unpack list inside the parenthesis.
```

`i) Using tuple() builtin function:`

- Python's built-in functions `tuple()` is one of the way to convert `list-values` to `tuple-values`.
- This `tuple ()` function can take any iterable as an argument and convert it into a tuple object.
- If we wants to convert a python `list to a tuple`, we can pass the `entire list` as a parameter within the tuple() function, and it will return the tuple data type as an output.

`ii) Using loop inside the tuple:`

- This method is a small variation of the above-given approach.
- You can use a `loop` inside the built-in function `tuple()` to convert a python list into a tuple object.
- However, it is the least used method for type conversion in comparison to others.

`iii) Unpack list inside the parenthesis:`

- To convert a list to a tuple in python programming, you can unpack the list elements inside the parenthesis.
- Here, the list essentially unpacks the elements inside the tuple literal, which is created by the presence of a single comma(,).
- However, this method is faster in comparison to others, but it suffers from readability which is not efficient enough.

In [3]:
```
list_mobile = ['iPHONE', 'NOKIA', 'SAMSUNG']

#convert list into tuple
tuple1 = tuple(list_mobile)
print(f"The tuple form using 1st method: {tuple1}")
print(f"The tuple type using 1st method: {type(tuple1)}")

# Using loop inside the tuple
tuple2 = tuple(i for i in list_mobile)
print(f"\nThe tuple form using 2nd method: {tuple2}")
print(f"The tuple type using 2nd method: {type(tuple2)}")

#unpack list items and form tuple
tuple3 = (*list_mobile,)
print(f"\nThe tuple form using 3rd method: {tuple3}")
print(f"The tuple type using 3rd method: {type(tuple3)}")
```
```
The tuple form using 1st method: ('iPHONE', 'NOKIA', 'SAMSUNG')
The tuple type using 1st method: <class 'tuple'>

The tuple form using 2nd method: ('iPHONE', 'NOKIA', 'SAMSUNG')
The tuple type using 2nd method: <class 'tuple'>

The tuple form using 3rd method: ('iPHONE', 'NOKIA', 'SAMSUNG')
The tuple type using 3rd method: <class 'tuple'>
```

`Tuple-to-List Conversion:`

- The three most common methods to convert `tuple-values` into `list-form` are listed below:

```
i) Using list() builtin function,
ii) Using loop inside the tuple
iii) Using * Unpacking Method,
```

`i) Using list() builtin function:`

- You can convert a tuple to a list using the list() function.
- The list() function is a built-in function in Python that takes any iterable object as an argument and returns a new list object containing the same elements as the iterable object.

`ii) Using loop inside the tuple:`

- This method is a small variation of the above-given approach.
- You can use a `loop` inside the built-in function `tuple()` to convert a python list into a tuple object.
- However, it is the least used method for type conversion in comparison to others.

`iii) Using * Unpacking Method:`

- You can use unpack a tuple and assign its elements to a list by enclosing the tuple in square brackets [] and separating the elements with commas.

In [6]:
```
tuple_laptop = ['Dell', 'HP', 'Lenovo']

#convert list into tuple
list_1 = list(tuple_laptop)
print(f"The list form using 1st method: {list_1}")
print(f"The list type using 1st method: {type(list_1)}")

# Using loop inside the tuple
list_2 = list(i for i in tuple_laptop)
print(f"\nThe list form using 2nd method: {list_2}")
print(f"The list type using 2nd method: {type(list_2)}")

#unpack list items and form tuple
list_3 = (*tuple_laptop,)
print(f"\nThe list form using 3rd method: {list_3}")
print(f"The list type using 3rd method: {type(list_3)}")
```
```
The list form using 1st method: ['Dell', 'HP', 'Lenovo']
The list type using 1st method: <class 'list'>

The list form using 2nd method: ['Dell', 'HP', 'Lenovo']
The list type using 2nd method: <class 'list'>

The list form using 3rd method: ('Dell', 'HP', 'Lenovo')
The list type using 3rd method: <class 'tuple'>
```

16. Variables that "contain" list values are not necessarily lists themselves. Instead, what do they contain?

Ans-16

- 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.
- Although Python variables technically contain references to list or dictionary values, people often casually say that the variable contains the list or dictionary.

17. How do you distinguish between copy.copy() and copy.deepcopy()?

Ans: 17

`copy.copy()`/shallow copy:

- It is the copy of the collection structure, not the elements.
- Affects the initial dataframe.
- Shallow copy doesn't replicate child objects.
- Creating a shallow copy is fast as compared to deep copy.
- The copy is dependent on the original

`copy.deepcopy()`/deep copy:

- It is the copy of the collections with all the elements in the original collection duplicated.
- Does not affect the initial dataframe.
- Deep copy replicates child objects recursively.
- Creating a deep copy is slow as compare to shallow copy.
- The copy is not fully dependent on the original.

```
In [ ]:
```