# Learning Journal Unit 6

Godfrey Ouma

University of the People

CS 1101: Programming Fundamentals

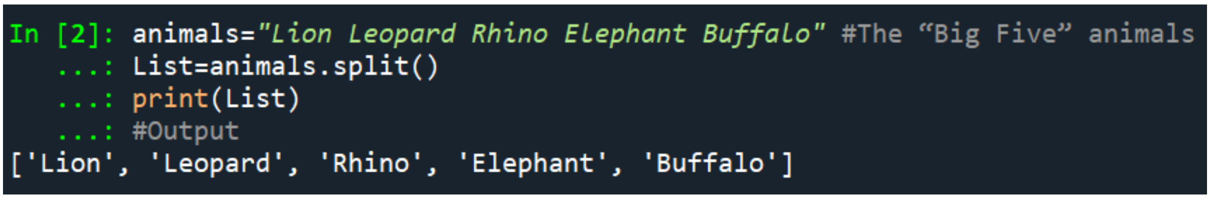
Janice Block

July 28, 2023

# Learning Journal Unit 6

## Part 1: Creating a string

### a) Turn the string into a list of words using split



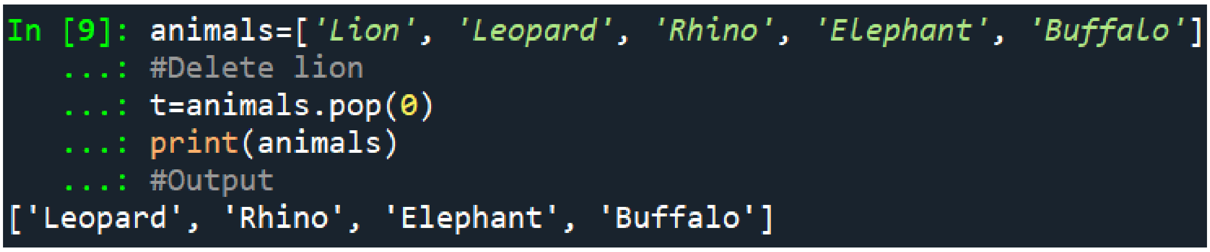
```
In [2]: animals="Lion Leopard Rhino Elephant Buffalo" #The "Big Five" animals
   ...: List=animals.split()
   ...: print(List)
   ...: #Output
['Lion', 'Leopard', 'Rhino', 'Elephant', 'Buffalo']
```

**Explanation:**

The code above shows how to split a string into a list using the `.split()` method. Downey (2015) notes that `.split()` method is often used to break a string into words, instead of letters. The `split()` method separates the `animals` string into a list of substrings, using spaces as the default delimiter. The resulting list displays each animal name as a separate element, allowing access to the "Big Five" animals individually.

b) Delete three words from the list

**Method 1: Using the `.pop()` method**

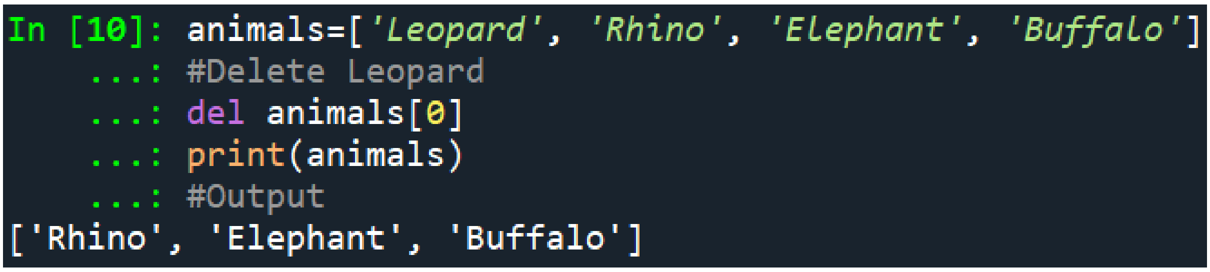


```
In [9]: animals=['Lion', 'Leopard', 'Rhino', 'Elephant', 'Buffalo']
   ...: #Delete lion
   ...: t=animals.pop(0)
   ...: print(animals)
   ...: #Output
['Leopard', 'Rhino', 'Elephant', 'Buffalo']
```

**Explanation:**

The code above shows how to delete an element from a list using the `.pop()` method, which usually modifies the list and returns the element that was removed (Downey, 2015). The list `animals` contains five elements: 'Lion', 'Leopard', 'Rhino', 'Elephant', and 'Buffalo'. The removed element 'Lion' is assigned to variable `t` and outputs the modified list `animals` with the remaining four elements, including 'Leopard', 'Rhino', 'Elephant', and 'Buffalo'.

**Method 2: Using the del operator**

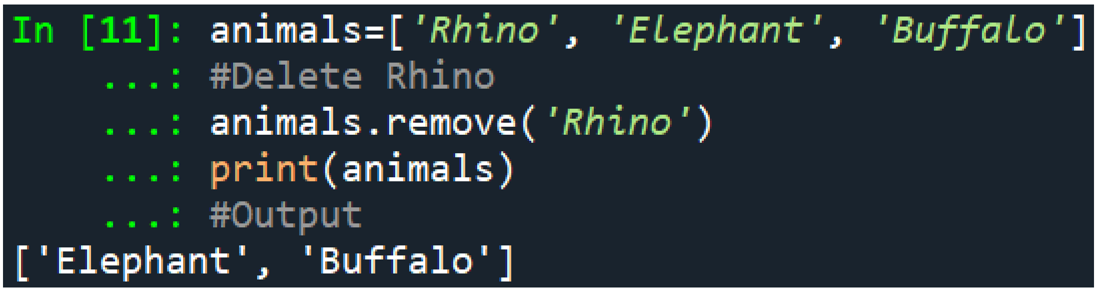


```
In [10]: animals=['Leopard', 'Rhino', 'Elephant', 'Buffalo']
    ...: #Delete Leopard
    ...: del animals[0]
    ...: print(animals)
    ...: #Output
['Rhino', 'Elephant', 'Buffalo']
```

**Explanation:**

The above shows how to delete an element from a list using the `del` statement, which is often used when removed value is not needed (Downey, 2015). The list `animals` contains five elements: Lion, Lion, Lion, Rhino, Elephant, and Buffalo. The `del animals[0]` statement removes the 'Leopard' element at index 0 from the list. The modified list `animals` contains only the remaining four elements, and 'Leopard' is successfully removed using the `del` statement.

**Method 3: Using remove statement**

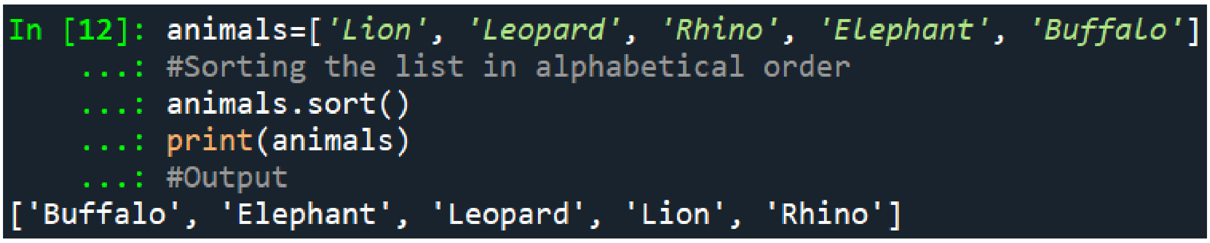


```
In [11]: animals=['Rhino', 'Elephant', 'Buffalo']
    ...: #Delete Rhino
    ...: animals.remove('Rhino')
    ...: print(animals)
    ...: #Output
['Elephant', 'Buffalo']
```

**Explanation:**

The code above shows how to delete an element from a list using the `remove()` method, which Downey (2015) opines that is often used to delete an element that is known. The `remove()` method is used to remove the specified element 'Rhino' from the list `animals`. The modified list `animals` contains only 'Elephant' and 'Buffalo', with 'Rhino' successfully removed.

**c) Sorting the list**



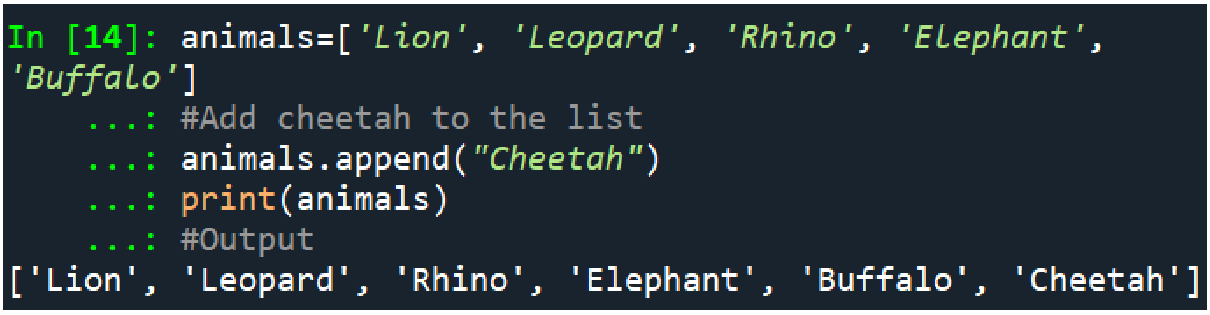
```
In [12]: animals=['Lion', 'Leopard', 'Rhino', 'Elephant', 'Buffalo']
    ...: #Sorting the list in alphabetical order
    ...: animals.sort()
    ...: print(animals)
    ...: #Output
['Buffalo', 'Elephant', 'Leopard', 'Lion', 'Rhino']
```

**Explanation:**

The code above shows how to sort a list of strings in alphabetical order using the `sort()` method. According to Downey (2015), sort arranges the elements of the list from low to high. The sorted list `animals` contains five elements: Lion, Lion, Leopard, Rhino, Elephant, and Buffalo. The `sort()` method sorts the list in alphabetical order, modifying the original list. The sorted list is displayed in the output, ensuring the original list remains unchanged.

d) **Add new words to the list**

**Method 1: Using append() method**

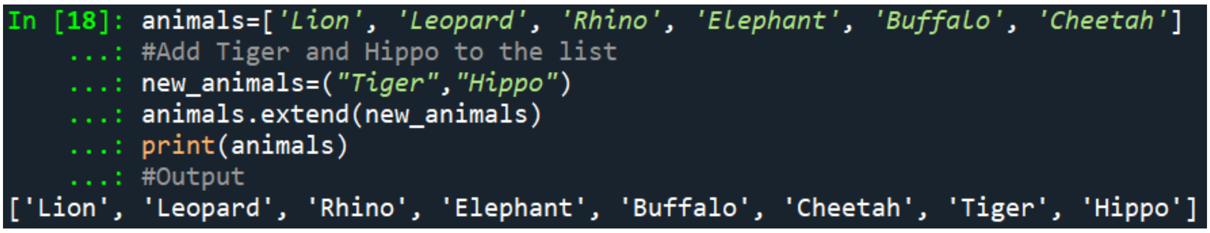


```
In [14]: animals=['Lion', 'Leopard', 'Rhino', 'Elephant',
'Buffalo']
    ...: #Add cheetah to the list
    ...: animals.append("Cheetah")
    ...: print(animals)
    ...: #Output
['Lion', 'Leopard', 'Rhino', 'Elephant', 'Buffalo', 'Cheetah']
```

**Explanation:**

In the code above, the append() method is used to add the string "Cheetah" as a new element to the end of the list. According to Downey (2015), append adds a new element to the end of a list. The output of the code will be ['Lion', 'Leopard', 'Rhino', 'Elephant', 'Buffalo', 'Cheetah'], which is the finals list containing all the original animals plus the newly added "Cheetah" at the end.

**Method 2: Using extend method**

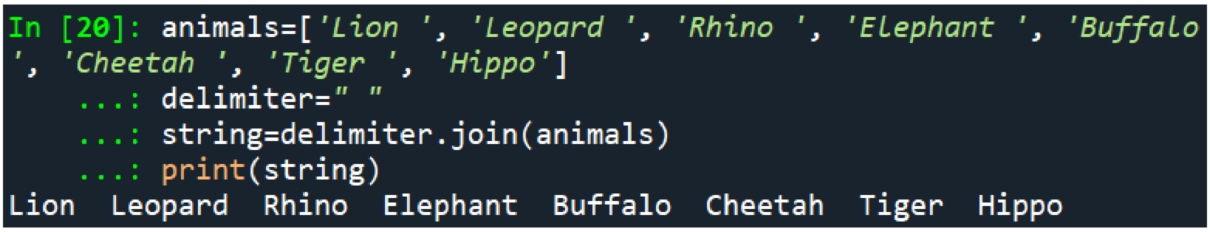


```
In [18]: animals=['Lion', 'Leopard', 'Rhino', 'Elephant', 'Buffalo', 'Cheetah']
    ...: #Add Tiger and Hippo to the list
    ...: new_animals=("Tiger","Hippo")
    ...: animals.extend(new_animals)
    ...: print(animals)
    ...: #Output
['Lion', 'Leopard', 'Rhino', 'Elephant', 'Buffalo', 'Cheetah', 'Tiger', 'Hippo']
```

**Explanation:**

The code above shows how to add multiple elements to an existing list using the `extend()` method. Downey (2015) opines that extend takes a list as an argument and appends all of the elements. The list `animals` contains six original elements, including Lion, Lion, Lion, Rhino, Elephant, Buffalo, and Cheetah. To add new elements 'Tiger' and 'Hippo', a tuple `new_animals` is defined with 'Tiger' and 'Hippo'. The `extend()` method efficiently adds elements from the tuple to the end of the list, maintaining the order of the elements.

e) **Turn the list of words back into a single string using join**



```
In [20]: animals=['Lion ', 'Leopard ', 'Rhino ', 'Elephant ', 'Buffalo
', 'Cheetah ', 'Tiger ', 'Hippo']
    ...: delimiter=" "
    ...: string=delimiter.join(animals)
    ...: print(string)
Lion  Leopard  Rhino  Elephant  Buffalo  Cheetah  Tiger  Hippo
```

**Explanation:**

The code above shows how to join all the elements of the animals list into a single string, removing any delimiter between them. To achieve this, the code uses the join() method of the empty string delimiter. ***join*** takes a list of strings and concatenates the elements. Since ***join*** is a string method, it has to be invoked on the delimiter and pass the list as a parameter (Downey, 2015). The resulting string is then printed to the console with the output having a single string containing all the animal names with spaces in between them.

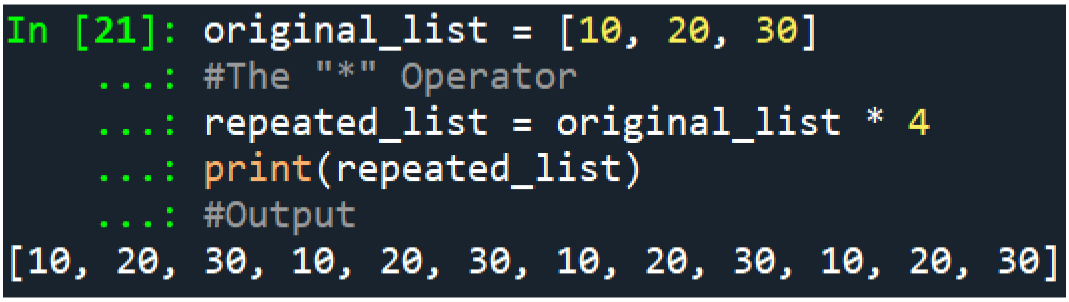
**Part 2: Providing Examples Using Python List**

**a) Nested list**

A nested list is a list that contains other lists as its elements. Although a list can contain another list, the nested list still counts as a single element (Downey, 2015). For example, ['Cow', 4, [‘Boy’, ‘Girl'], [‘USA’, ‘Kenya’, ‘Uganda’]] is a nested list containing four elements, where some elements are themselves lists. It contains a combination of simple elements (strings and integers) and more complex elements (other lists)

**b) The "*" Operator**

The * operator is usually used to repeat a list a given number of times (Downey, 2015). See the example below.



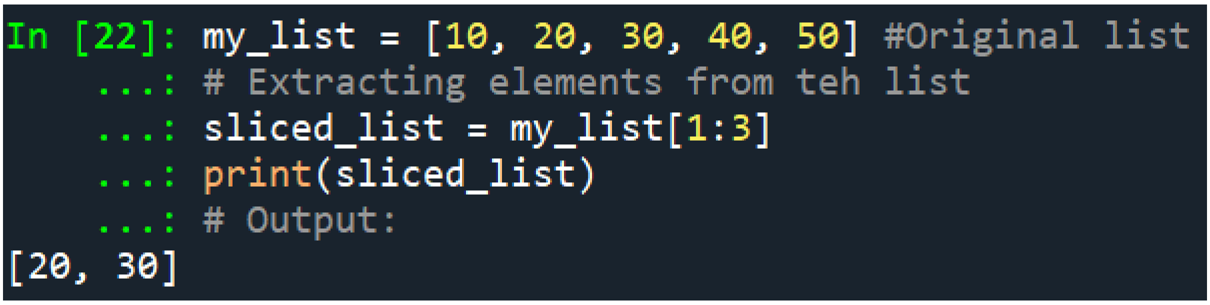
```
In [21]: original_list = [10, 20, 30]
    ...: #The "*" Operator
    ...: repeated_list = original_list * 4
    ...: print(repeated_list)
    ...: #Output
[10, 20, 30, 10, 20, 30, 10, 20, 30, 10, 20, 30]
```

**Explanation:**

The code above shows a list named original_list with elements [10, 20, 30]. The "*" operator has been used to repeat the elements 4 times, as shown in the output.

**c) List Slices**

List slices uses a range of slices to extract a portion of a list. See the example below.



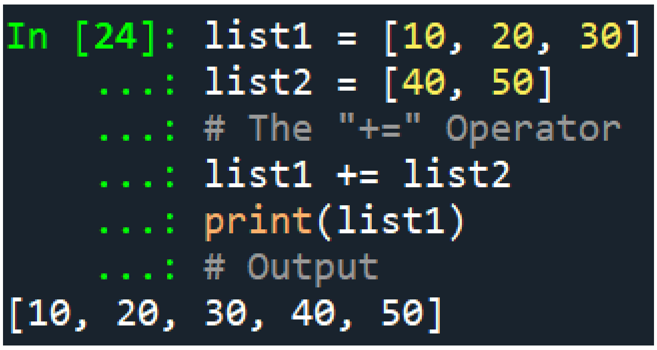
```
In [22]: my_list = [10, 20, 30, 40, 50] #Original list
    ...: # Extracting elements from teh list
    ...: sliced_list = my_list[1:3]
    ...: print(sliced_list)
    ...: # Output:
[20, 30]
```

**Explanation:**

The code above shows how a list slicing with the range 1:3 can be used to extract elements from index 1 (inclusive) to 3 (exclusive) from a list named my_list with elements [10, 20, 30, 40, 50]. The new list named sliced_list in the output contains elements [20, 30].

**d) The "+=" Operator**

The "+=" operator is often used to concatenate two lists together. See example below.



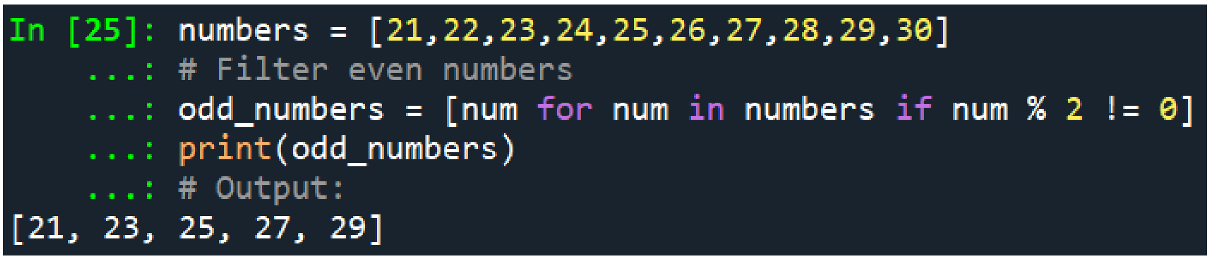
```
In [24]: list1 = [10, 20, 30]
    ...: list2 = [40, 50]
    ...: # The "+=" Operator
    ...: list1 += list2
    ...: print(list1)
    ...: # Output
[10, 20, 30, 40, 50]
```

**Explanation:**

The code above shows how the "+=" operator can be used to concatenate list2 to list1 with elements [10, 20, 30] and [40, 50] respectively. This results in a modified list1 with elements [10, 20, 30, 40, 50], as shown in the output.

**e) A List Filter**

A list filter provides means for developing a new list containing elements that satisfy a specific condition. See example below.



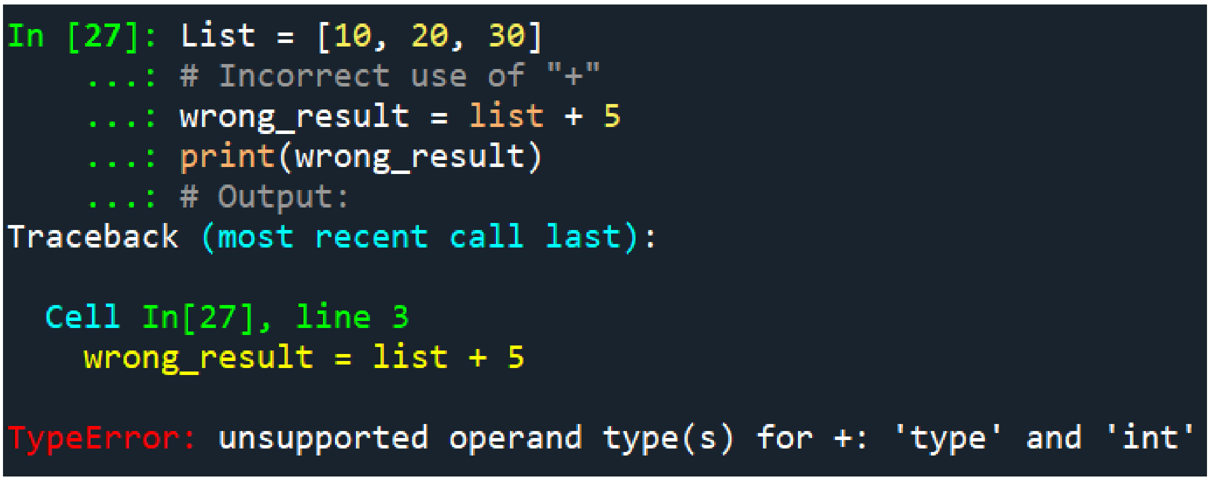
```
In [25]: numbers = [21,22,23,24,25,26,27,28,29,30]
    ...: # Filter even numbers
    ...: odd_numbers = [num for num in numbers if num % 2 != 0]
    ...: print(odd_numbers)
    ...: # Output:
[21, 23, 25, 27, 29]
```

**Explanation:**

The code above shows use a list comprehension can be used to filter even numbers from the list. The condition num % 2!= 0 checks if the number is odd, and only the odd numbers are included in the new list named odd_numbers, which results in [21, ,23, 25, 27, 29], as shown in the output.

### f) A list operation that is legal but does the "wrong" thing

A common mistake is using the "+" operator with a list and an integer, expecting the integer to be added to each element in the list. See example below.



```
In [27]: List = [10, 20, 30]
    ...: # Incorrect use of "+"
    ...: wrong_result = list + 5
    ...: print(wrong_result)
    ...: # Output:
Traceback (most recent call last):

  Cell In[27], line 3
    wrong_result = list + 5

TypeError: unsupported operand type(s) for +: 'type' and 'int'
```

Explanation:

In the example above, the "+" operator was mistakenly used to add 5 to each element in the list, which Python could not execute the operation because the "+" operator is not defined between a list and an integer. As a result, the code raises a TypeError, indicating that a list cannot be concatenated to an integer.

# Reference

Downey, A. (2015). *Think Python: How to think like a computer scientist.* Green Tree Press.