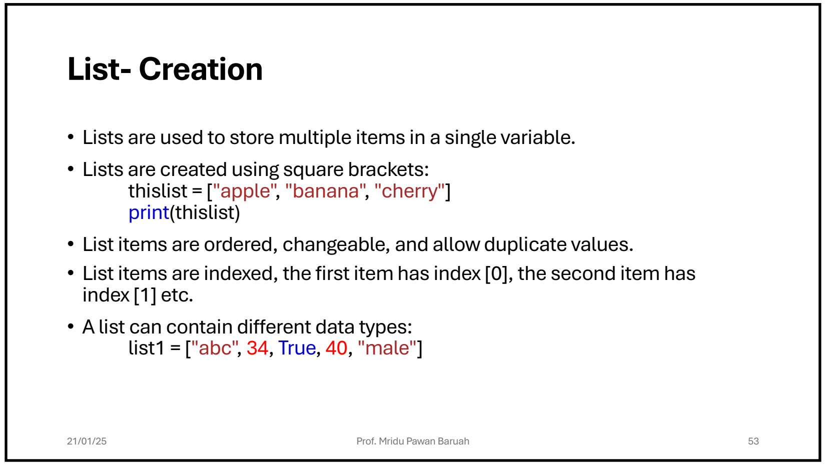


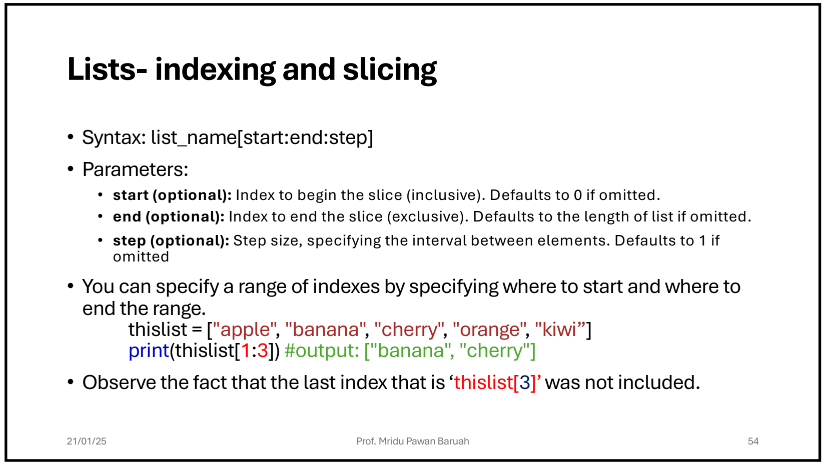
List- Creation

- Lists are used to store multiple items in a single variable.
- Lists are created using square brackets:

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index [0], the second item has index [1] etc.
- A list can contain different data types:

```
list1 = ["abc", 34, True, 40, "male"]
```

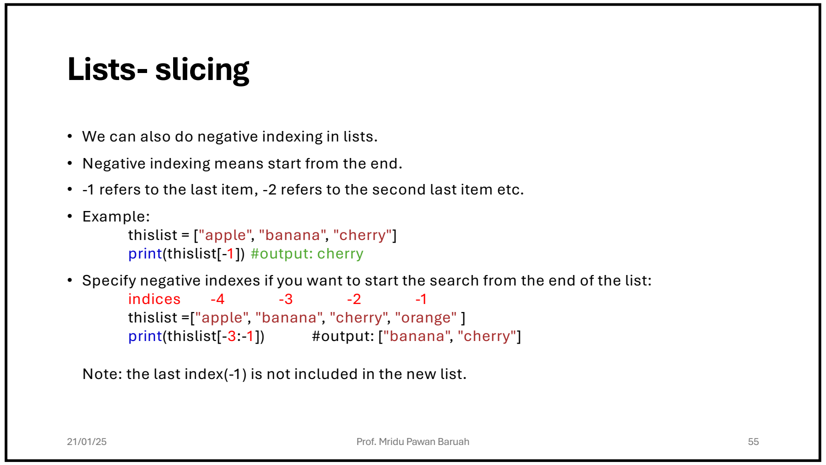


Lists- indexing and slicing

- Syntax: list_name[start:end:step]
- Parameters:
  - **start (optional):** Index to begin the slice (inclusive). Defaults to 0 if omitted.
  - **end (optional):** Index to end the slice (exclusive). Defaults to the length of list if omitted.
  - **step (optional):** Step size, specifying the interval between elements. Defaults to 1 if omitted
- You can specify a range of indexes by specifying where to start and where to end the range.

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi”]
print(thislist[1:3]) #output: ["banana", "cherry"]
```

- Observe the fact that the last index that is ‘thislist[3]’ was not included.



Lists- slicing

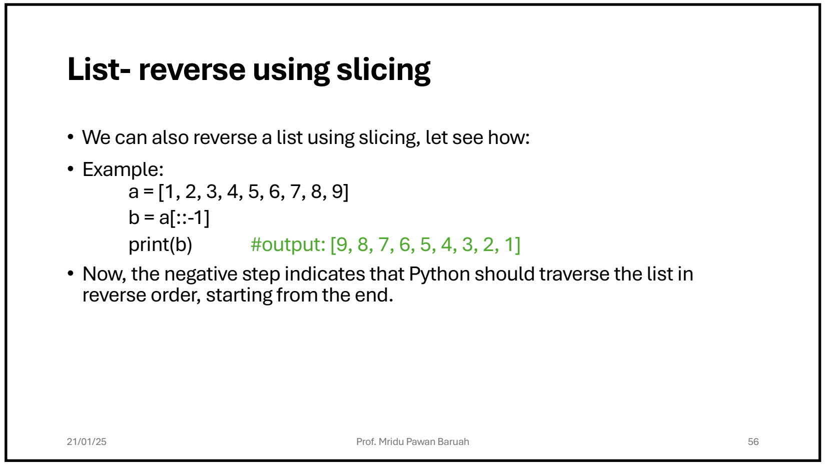
- We can also do negative indexing in lists.
- Negative indexing means start from the end.
- -1 refers to the last item, -2 refers to the second last item etc.
- Example:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1]) #output: cherry
```

- Specify negative indexes if you want to start the search from the end of the list:

```
indices      -4          -3          -2          -1
thislist =["apple", "banana", "cherry", "orange" ]
print(thislist[-3:-1])        #output: ["banana", "cherry"]
```

Note: the last index(-1) is not included in the new list.

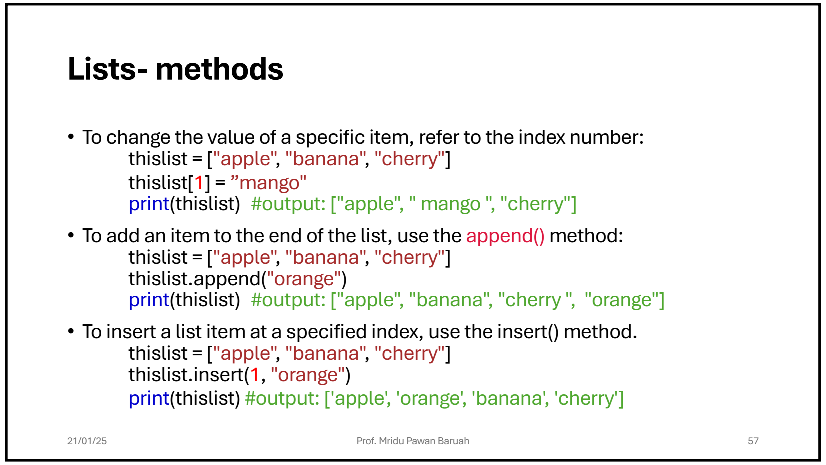


List- reverse using slicing

- We can also reverse a list using slicing, let see how:
- Example:

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

- Now, the negative step indicates that Python should traverse the list in reverse order, starting from the end.



Lists- methods

- To change the value of a specific item, refer to the index number:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = ”mango"
print(thislist)  #output: ["apple", " mango ", "cherry"]
```

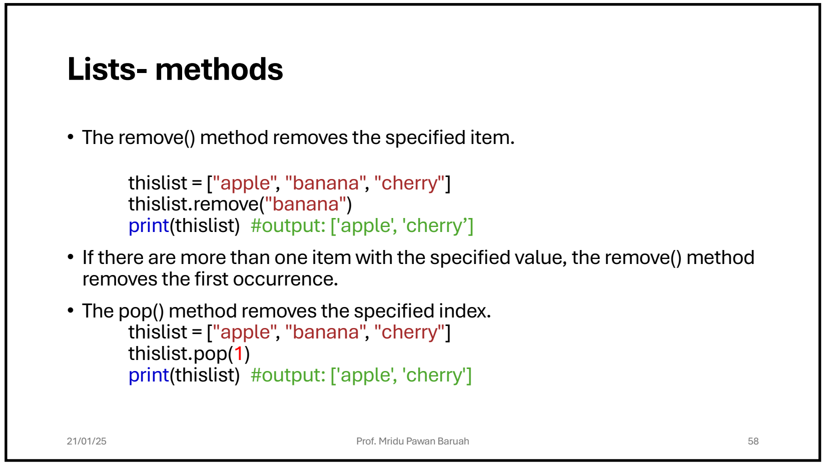
- To add an item to the end of the list, use the append() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)  #output: ["apple", "banana", "cherry ",  "orange"]
```

- To insert a list item at a specified index, use the insert() method.

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist) #output: ['apple', 'orange', 'banana', 'cherry']
```

57



Lists- methods

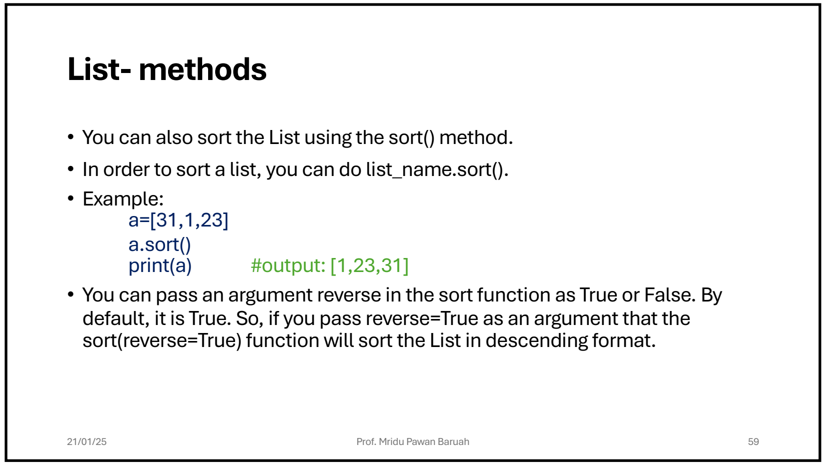
- The remove() method removes the specified item.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)  #output: ['apple', 'cherry’]
```

- If there are more than one item with the specified value, the remove() method removes the first occurrence.
- The pop() method removes the specified index.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)  #output: ['apple', 'cherry']
```

58



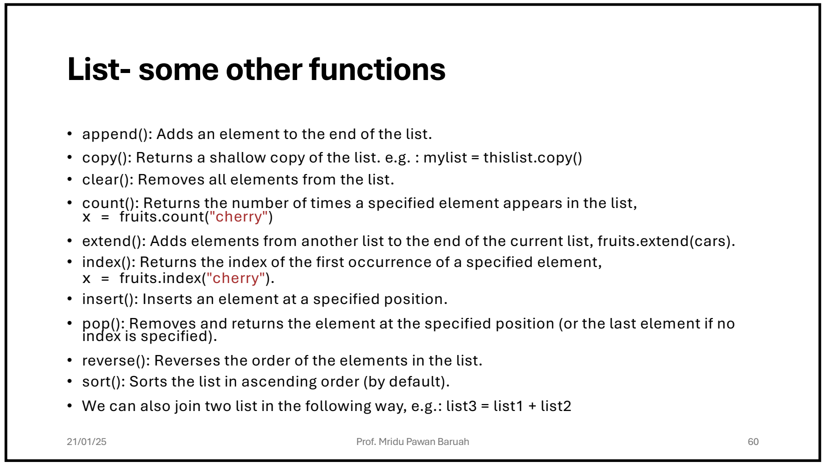
List- methods

- You can also sort the List using the sort() method.
- In order to sort a list, you can do list_name.sort().
- Example:

```
a=[31,1,23]
a.sort()
print(a)          #output: [1,23,31]
```

- You can pass an argument reverse in the sort function as True or False. By default, it is True. So, if you pass reverse=True as an argument that the sort(reverse=True) function will sort the List in descending format.

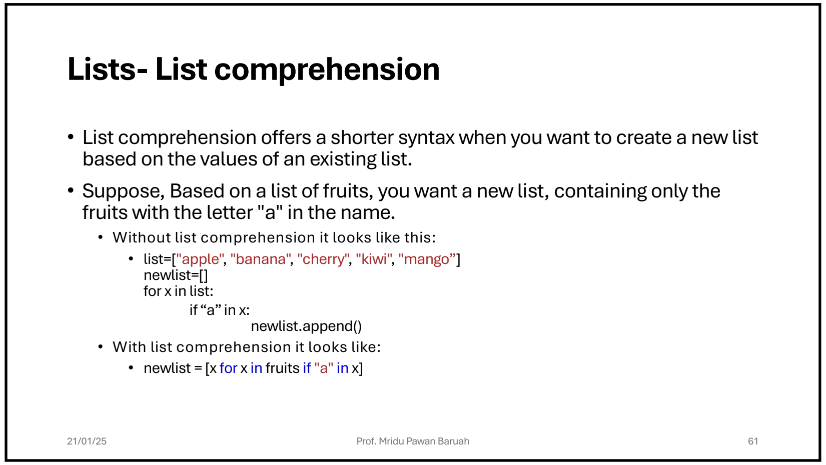
59



List- some other functions

- append(): Adds an element to the end of the list.
- copy(): Returns a shallow copy of the list. e.g. : mylist = thislist.copy()
- clear(): Removes all elements from the list.
- count(): Returns the number of times a specified element appears in the list, x = fruits.count("cherry")
- extend(): Adds elements from another list to the end of the current list, fruits.extend(cars).
- index(): Returns the index of the first occurrence of a specified element, x = fruits.index("cherry").
- insert(): Inserts an element at a specified position.
- pop(): Removes and returns the element at the specified position (or the last element if no index is specified).
- reverse(): Reverses the order of the elements in the list.
- sort(): Sorts the list in ascending order (by default).
- We can also join two list in the following way, e.g.: list3 = list1 + list2

60



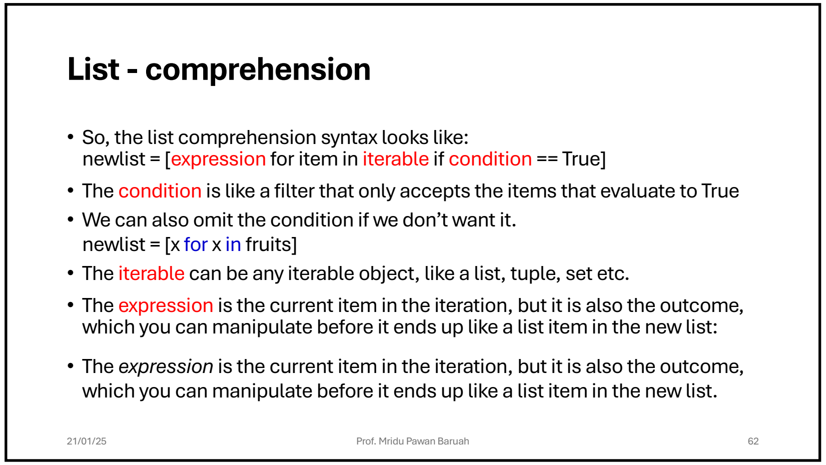
Lists- List comprehension

- List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.
- Suppose, Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.
  - Without list comprehension it looks like this:

```
list=["apple", "banana", "cherry", "kiwi", "mango”]
newlist=[]
for x in list:
        if “a” in x:
                newlist.append()
```

  - With list comprehension it looks like:
    - newlist = [x for x in fruits if "a" in x]

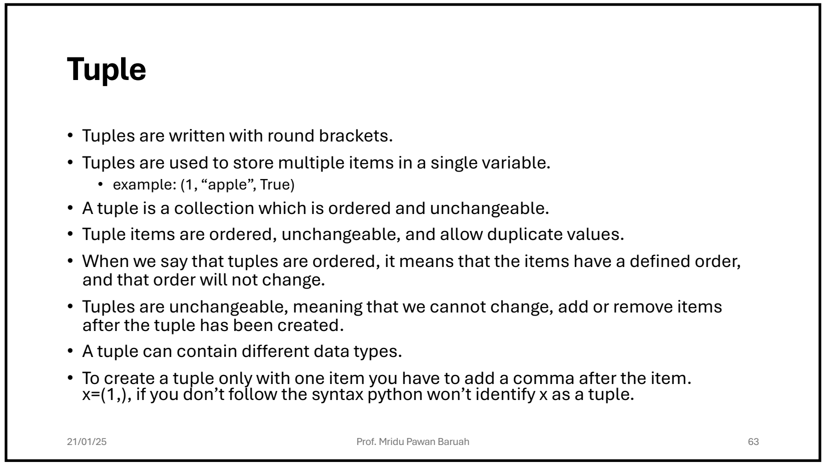
61



List - comprehension

- So, the list comprehension syntax looks like:
  newlist = [expression for item in iterable if condition == True]
- The condition is like a filter that only accepts the items that evaluate to True
- We can also omit the condition if we don’t want it.
  newlist = [x for x in fruits]
- The iterable can be any iterable object, like a list, tuple, set etc.
- The expression is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:
- The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list.

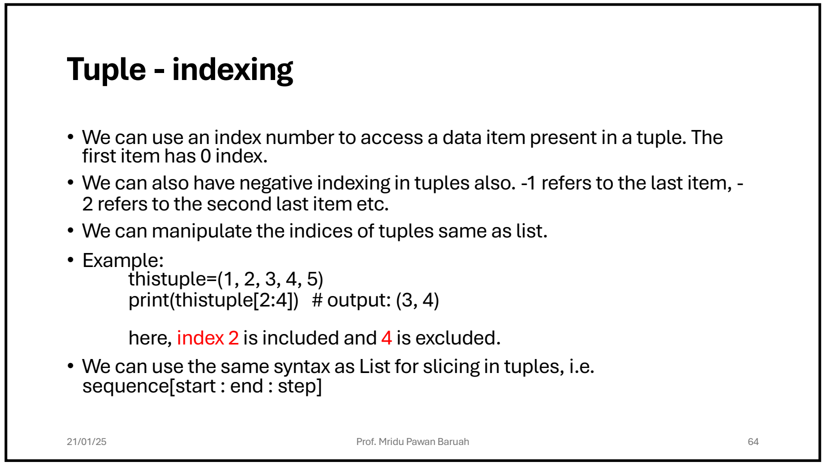
62



Tuple

- Tuples are written with round brackets.
- Tuples are used to store multiple items in a single variable.
  - example: (1, “apple”, True)
- A tuple is a collection which is ordered and unchangeable.
- Tuple items are ordered, unchangeable, and allow duplicate values.
- When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.
- Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.
- A tuple can contain different data types.
- To create a tuple only with one item you have to add a comma after the item. x=(1,), if you don’t follow the syntax python won’t identify x as a tuple.

63



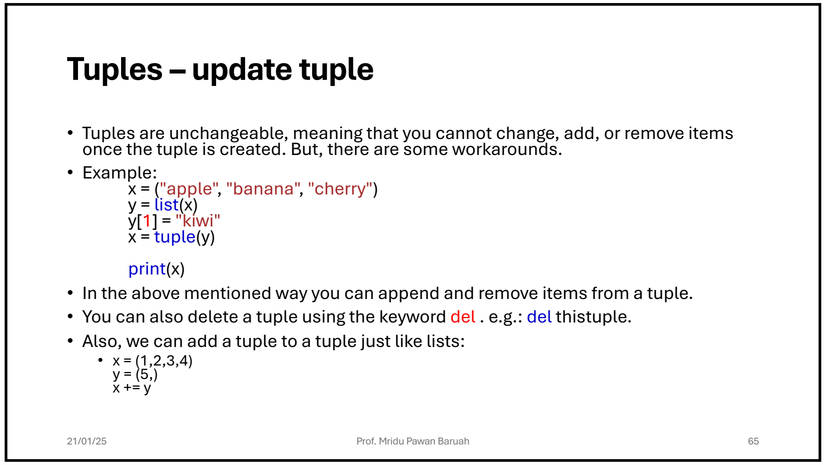
Tuple - indexing

- We can use an index number to access a data item present in a tuple. The first item has 0 index.
- We can also have negative indexing in tuples also. -1 refers to the last item, -2 refers to the second last item etc.
- We can manipulate the indices of tuples same as list.
- Example:

```
thistuple=(1, 2, 3, 4, 5)
print(thistuple[2:4])  # output: (3, 4)
```

  here, index 2 is included and 4 is excluded.
- We can use the same syntax as List for slicing in tuples, i.e. sequence[start : end : step]

64



Tuples – update tuple

- Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created. But, there are some workarounds.
- Example:

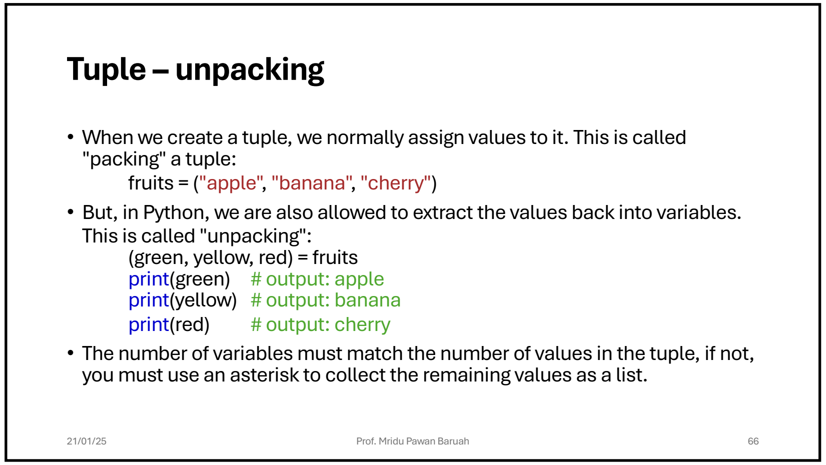
```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

- In the above mentioned way you can append and remove items from a tuple.
- You can also delete a tuple using the keyword del . e.g.: del thistuple.
- Also, we can add a tuple to a tuple just like lists:

  -
    ```
    x = (1,2,3,4)
    y = (5,)
    x += y
    ```

21/01/25 Prof. Mridu Pawan Baruah 65

65



Tuple – unpacking

- When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

```
fruits = ("apple", "banana", "cherry")
```

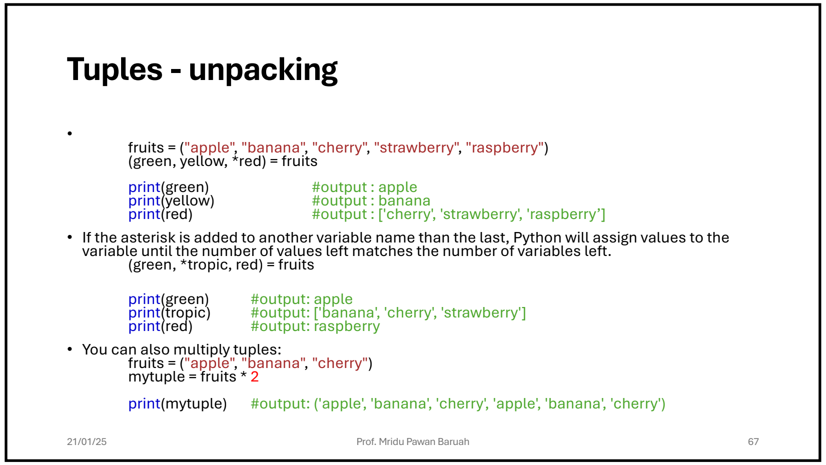
- But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

```
(green, yellow, red) = fruits
print(green)   # output: apple
print(yellow)  # output: banana
print(red)     # output: cherry
```

- The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

21/01/25 Prof. Mridu Pawan Baruah 66

66



Tuples - unpacking

-
  ```
  fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
  (green, yellow, *red) = fruits

  print(green)                #output : apple
  print(yellow)               #output : banana
  print(red)                  #output : ['cherry', 'strawberry', 'raspberry']
  ```
- If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

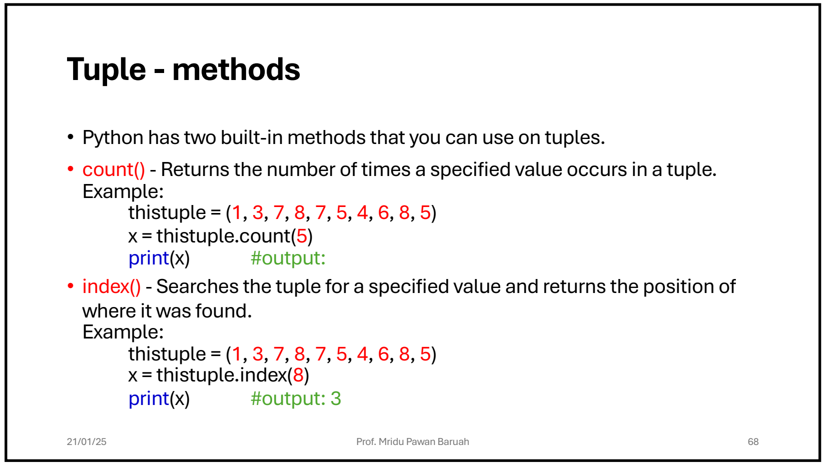
```
(green, *tropic, red) = fruits

print(green)     #output: apple
print(tropic)    #output: ['banana', 'cherry', 'strawberry']
print(red)       #output: raspberry
```

- You can also multiply tuples:

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)   #output: ('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

21/01/25 Prof. Mridu Pawan Baruah 67

67



Tuple - methods

- Python has two built-in methods that you can use on tuples.
- count() - Returns the number of times a specified value occurs in a tuple. Example:

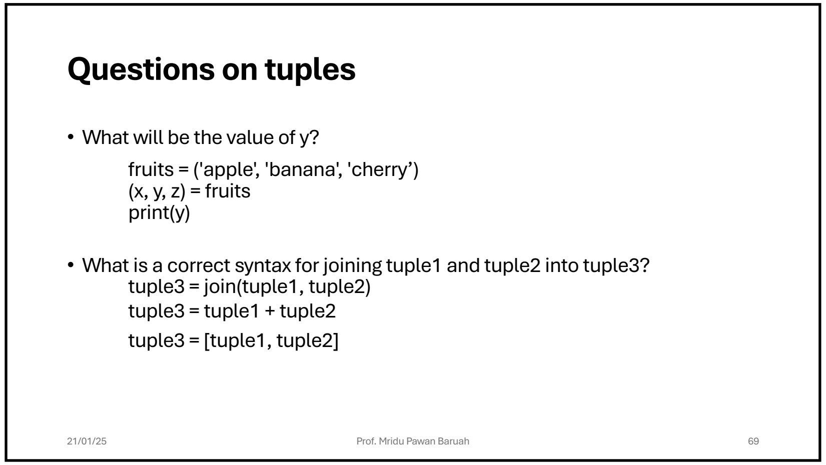
```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.count(5)
print(x)          #output:
```

- index() - Searches the tuple for a specified value and returns the position of where it was found. Example:

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
x = thistuple.index(8)
print(x)          #output: 3
```

21/01/25 Prof. Mridu Pawan Baruah 68

68



Questions on tuples

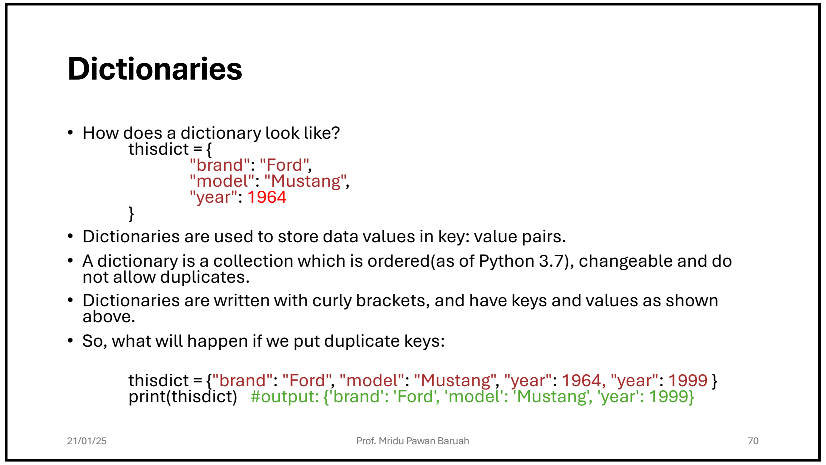
- What will be the value of y?

```
fruits = ('apple', 'banana', 'cherry')
(x, y, z) = fruits
print(y)
```

- What is a correct syntax for joining tuple1 and tuple2 into tuple3?

```
tuple3 = join(tuple1, tuple2)
tuple3 = tuple1 + tuple2
tuple3 = [tuple1, tuple2]
```

69



Dictionaries

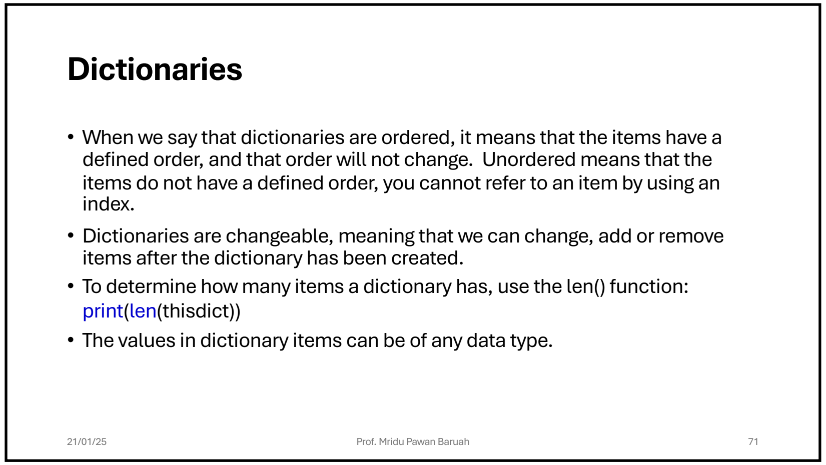
- How does a dictionary look like?

```
thisdict = {
        "brand": "Ford",
        "model": "Mustang",
        "year": 1964
}
```

- Dictionaries are used to store data values in key: value pairs.
- A dictionary is a collection which is ordered(as of Python 3.7), changeable and do not allow duplicates.
- Dictionaries are written with curly brackets, and have keys and values as shown above.
- So, what will happen if we put duplicate keys:

```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964, "year": 1999 }
print(thisdict)   #output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1999}
```

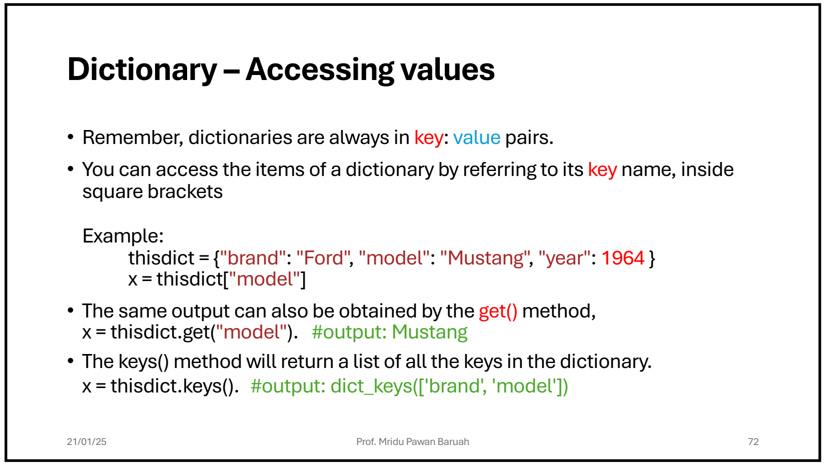
70



Dictionaries

- When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change. Unordered means that the items do not have a defined order, you cannot refer to an item by using an index.
- Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.
- To determine how many items a dictionary has, use the len() function: print(len(thisdict))
- The values in dictionary items can be of any data type.

71



Dictionary – Accessing values

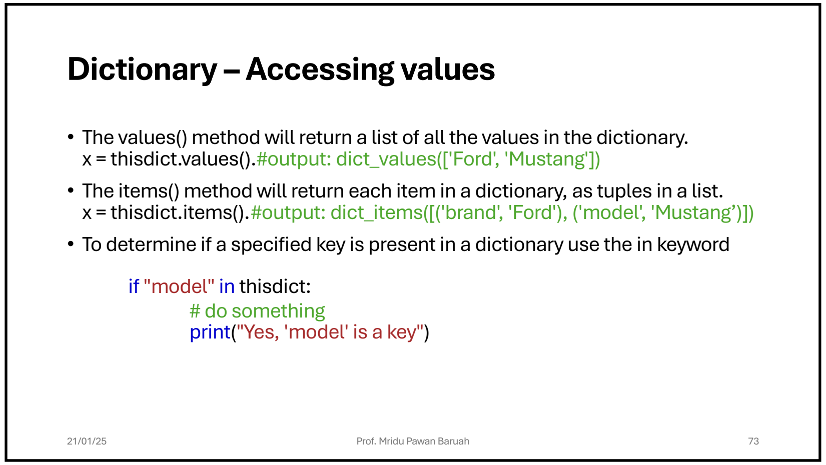
- Remember, dictionaries are always in key: value pairs.
- You can access the items of a dictionary by referring to its key name, inside square brackets

  Example:

```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964 }
x = thisdict["model"]
```

- The same output can also be obtained by the get() method,
  x = thisdict.get("model").   #output: Mustang
- The keys() method will return a list of all the keys in the dictionary.
  x = thisdict.keys().   #output: dict_keys(['brand', 'model'])

72

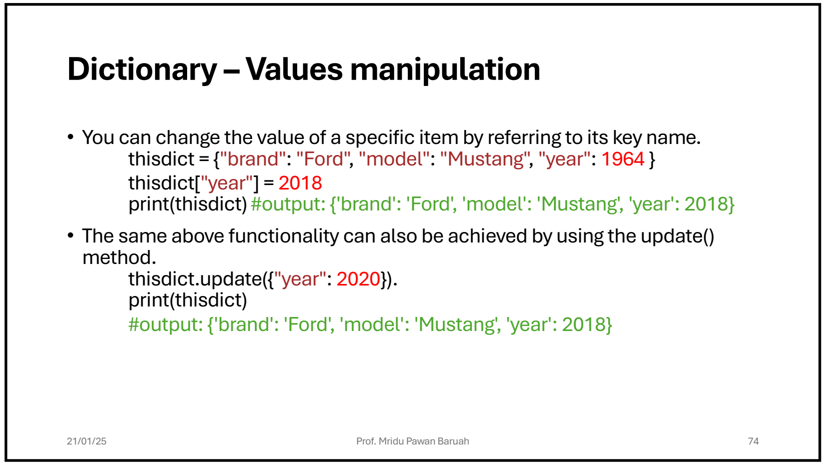


Dictionary – Accessing values

- The values() method will return a list of all the values in the dictionary.
```
x = thisdict.values().#output: dict_values(['Ford', 'Mustang'])
```
- The items() method will return each item in a dictionary, as tuples in a list.
```
x = thisdict.items().#output: dict_items([('brand', 'Ford'), ('model', 'Mustang’)])
```
- To determine if a specified key is present in a dictionary use the in keyword

```
if "model" in thisdict:
        # do something
        print("Yes, 'model' is a key")
```

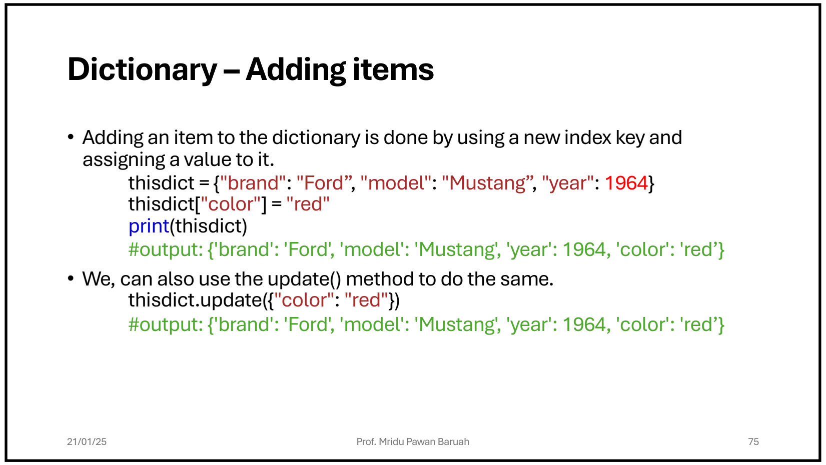
73



Dictionary – Values manipulation

- You can change the value of a specific item by referring to its key name.
```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964 }
thisdict["year"] = 2018
print(thisdict) #output: {'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```
- The same above functionality can also be achieved by using the update() method.
```
thisdict.update({"year": 2020}).
print(thisdict)
#output: {'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

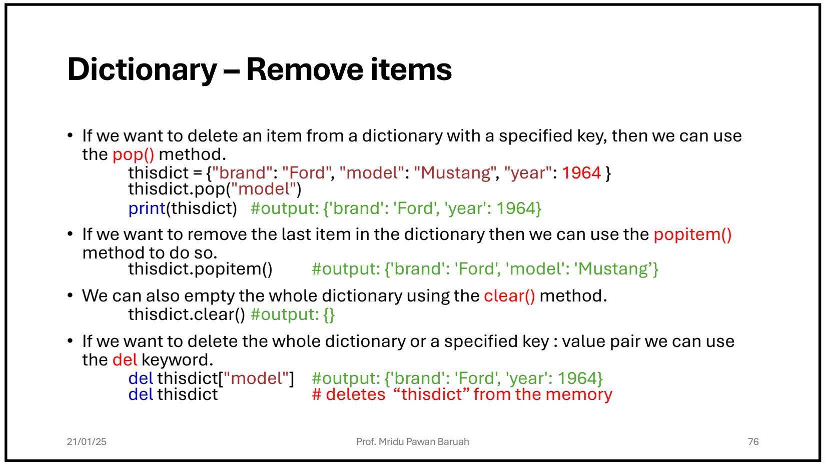
74



Dictionary – Adding items

- Adding an item to the dictionary is done by using a new index key and assigning a value to it.
```
thisdict = {"brand": "Ford”, "model": "Mustang”, "year": 1964}
thisdict["color"] = "red"
print(thisdict)
#output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red’}
```
- We, can also use the update() method to do the same.
```
thisdict.update({"color": "red"})
#output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red’}
```

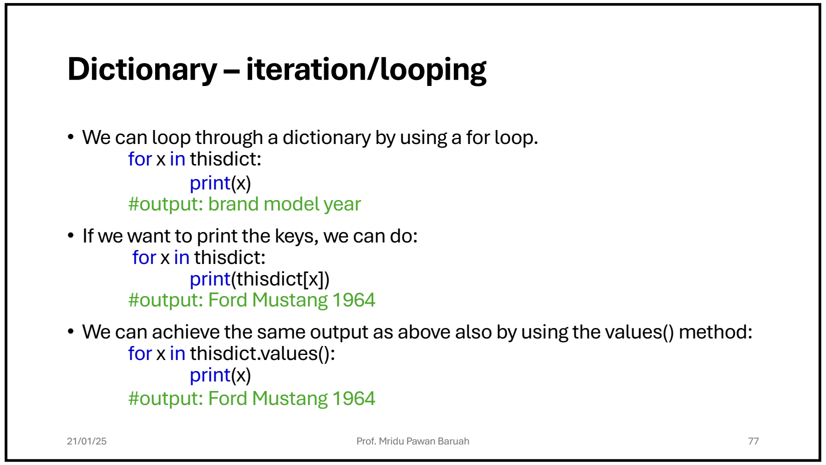
75



Dictionary – Remove items

- If we want to delete an item from a dictionary with a specified key, then we can use the pop() method.
```
thisdict = {"brand": "Ford", "model": "Mustang", "year": 1964 }
thisdict.pop("model")
print(thisdict)   #output: {'brand': 'Ford', 'year': 1964}
```
- If we want to remove the last item in the dictionary then we can use the popitem() method to do so.
```
thisdict.popitem()       #output: {'brand': 'Ford', 'model': 'Mustang’}
```
- We can also empty the whole dictionary using the clear() method.
```
thisdict.clear() #output: {}
```
- If we want to delete the whole dictionary or a specified key : value pair we can use the del keyword.
```
del thisdict["model"]    #output: {'brand': 'Ford', 'year': 1964}
del thisdict             # deletes  “thisdict” from the memory
```

76



Dictionary – iteration/looping

- We can loop through a dictionary by using a for loop.

```
for x in thisdict:
        print(x)
#output: brand model year
```

- If we want to print the keys, we can do:

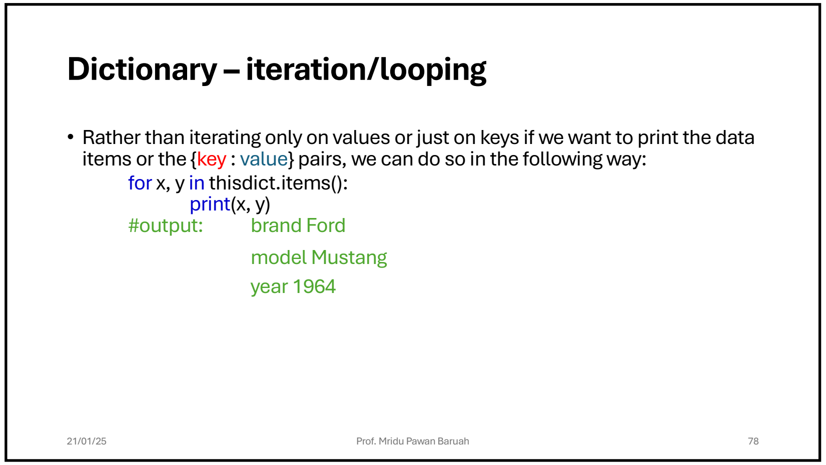
```
 for x in thisdict:
        print(thisdict[x])
#output: Ford Mustang 1964
```

- We can achieve the same output as above also by using the values() method:

```
for x in thisdict.values():
        print(x)
#output: Ford Mustang 1964
```

21/01/25 Prof. Mridu Pawan Baruah 77

77



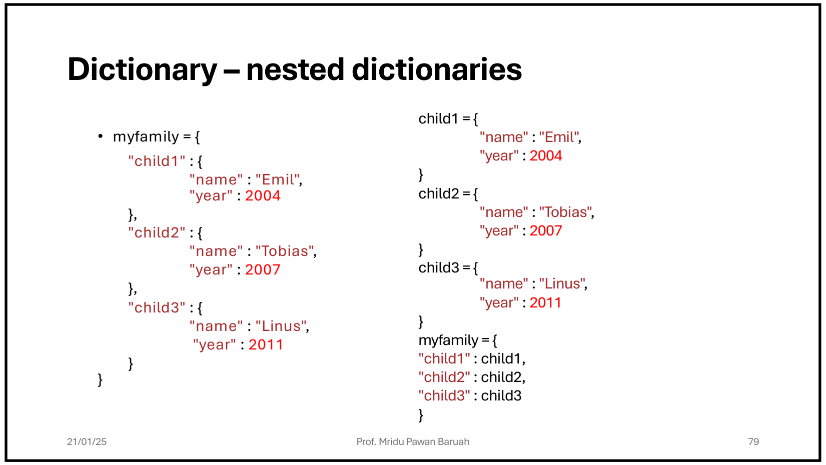
Dictionary – iteration/looping

- Rather than iterating only on values or just on keys if we want to print the data items or the {key : value} pairs, we can do so in the following way:

```
for x, y in thisdict.items():
        print(x, y)
#output:      brand Ford
              model Mustang
              year 1964
```

21/01/25 Prof. Mridu Pawan Baruah 78

78



Dictionary – nested dictionaries

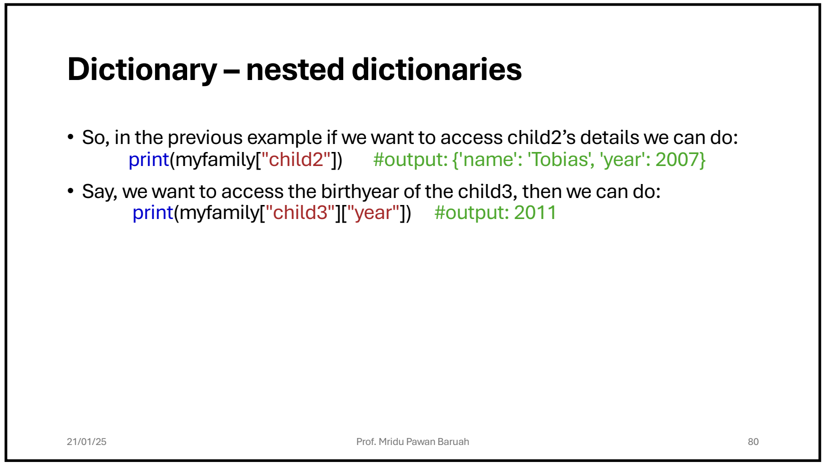
- myfamily = {

```
    "child1" : {
            "name" : "Emil",
            "year" : 2004
    },
    "child2" : {
            "name" : "Tobias",
            "year" : 2007
    },
    "child3" : {
            "name" : "Linus",
             "year" : 2011
    }
}
```

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

21/01/25 Prof. Mridu Pawan Baruah 79

79



Dictionary – nested dictionaries

- So, in the previous example if we want to access child2's details we can do:

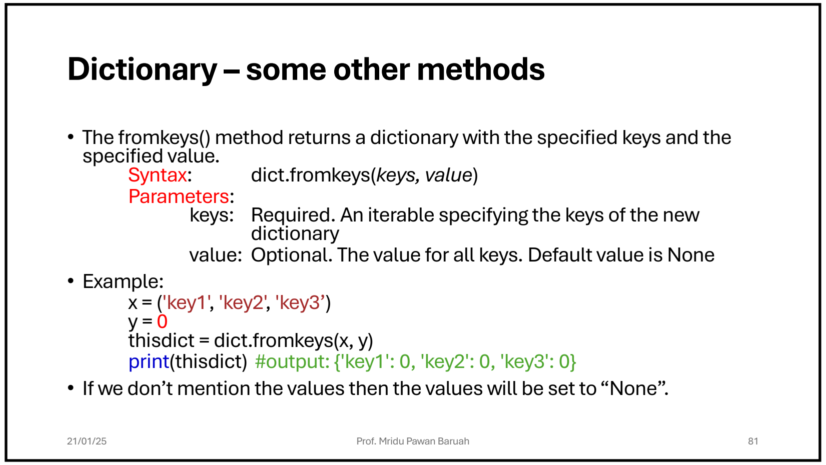
```
print(myfamily["child2"])     #output: {'name': 'Tobias', 'year': 2007}
```

- Say, we want to access the birthyear of the child3, then we can do:

```
 print(myfamily["child3"]["year"])     #output: 2011
```

21/01/25 Prof. Mridu Pawan Baruah 80

80



Dictionary – some other methods

- The fromkeys() method returns a dictionary with the specified keys and the specified value.

  Syntax: dict.fromkeys(*keys, value*)

  Parameters:

  keys: Required. An iterable specifying the keys of the new dictionary

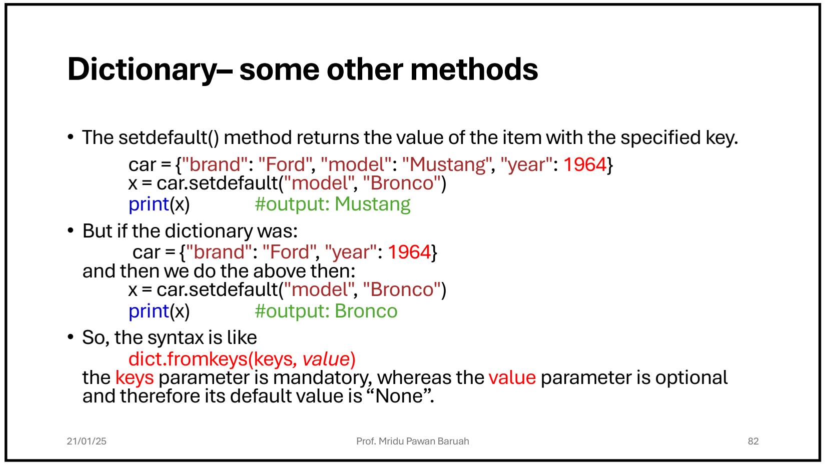
  value: Optional. The value for all keys. Default value is None
- Example:

```
x = ('key1', 'key2', 'key3')
y = 0
thisdict = dict.fromkeys(x, y)
print(thisdict)  #output: {'key1': 0, 'key2': 0, 'key3': 0}
```

- If we don’t mention the values then the values will be set to “None”.

21/01/25 Prof. Mridu Pawan Baruah 81

81



Dictionary– some other methods

- The setdefault() method returns the value of the item with the specified key.

```
car = {"brand": "Ford", "model": "Mustang", "year": 1964}
x = car.setdefault("model", "Bronco")
print(x)          #output: Mustang
```

- But if the dictionary was:

```
car = {"brand": "Ford", "year": 1964}
```

  and then we do the above then:

```
x = car.setdefault("model", "Bronco")
print(x)          #output: Bronco
```

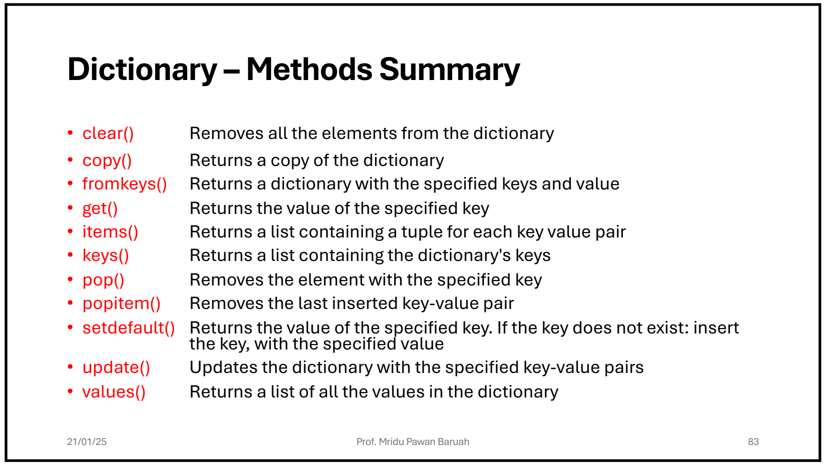
- So, the syntax is like

  dict.fromkeys(keys, *value*)

  the keys parameter is mandatory, whereas the value parameter is optional and therefore its default value is “None”.

21/01/25 Prof. Mridu Pawan Baruah 82

82

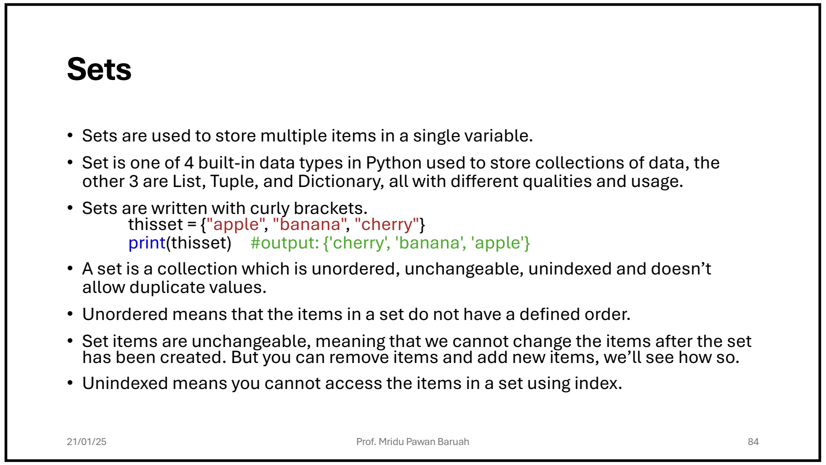


Dictionary – Methods Summary

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

21/01/25 Prof. Mridu Pawan Baruah 83

83



Sets

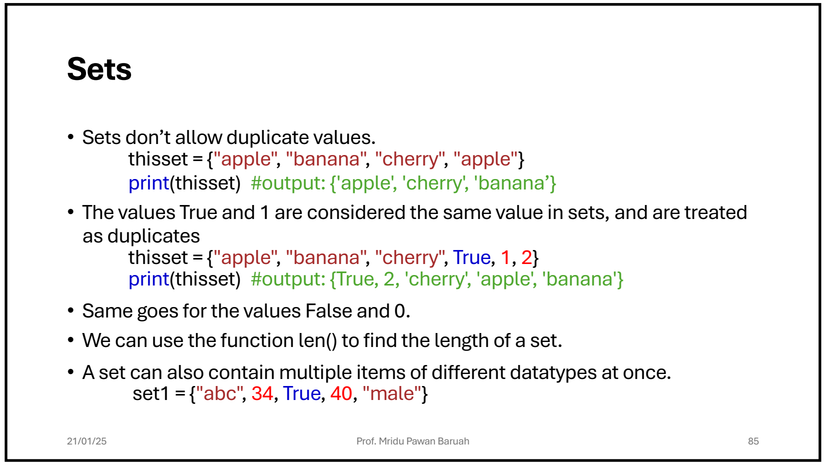
- Sets are used to store multiple items in a single variable.
- Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.
- Sets are written with curly brackets.

```
thisset = {"apple", "banana", "cherry"}
print(thisset)    #output: {'cherry', 'banana', 'apple'}
```

- A set is a collection which is unordered, unchangeable, unindexed and doesn’t allow duplicate values.
- Unordered means that the items in a set do not have a defined order.
- Set items are unchangeable, meaning that we cannot change the items after the set has been created. But you can remove items and add new items, we’ll see how so.
- Unindexed means you cannot access the items in a set using index.

21/01/25 Prof. Mridu Pawan Baruah 84

84



Sets

- Sets don't allow duplicate values.

```
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)  #output: {'apple', 'cherry', 'banana'}
```

- The values True and 1 are considered the same value in sets, and are treated as duplicates

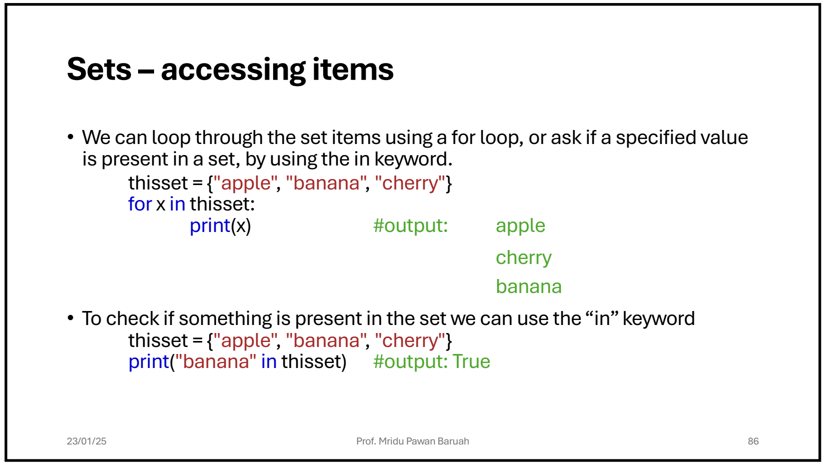
```
thisset = {"apple", "banana", "cherry", True, 1, 2}
print(thisset)  #output: {True, 2, 'cherry', 'apple', 'banana'}
```

- Same goes for the values False and 0.
- We can use the function len() to find the length of a set.
- A set can also contain multiple items of different datatypes at once.

```
set1 = {"abc", 34, True, 40, "male"}
```

21/01/25 Prof. Mridu Pawan Baruah 85

85



Sets – accessing items

- We can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

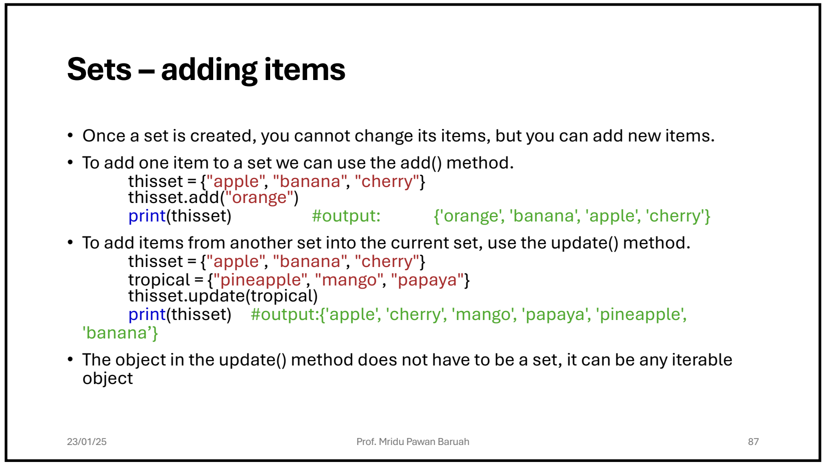
```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
    print(x)                #output:    apple
                                        cherry
                                        banana
```

- To check if something is present in the set we can use the "in" keyword

```
thisset = {"apple", "banana", "cherry"}
print("banana" in thisset)     #output: True
```

23/01/25 Prof. Mridu Pawan Baruah 86

86



Sets – adding items

- Once a set is created, you cannot change its items, but you can add new items.
- To add one item to a set we can use the add() method.

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)          #output:       {'orange', 'banana', 'apple', 'cherry'}
```

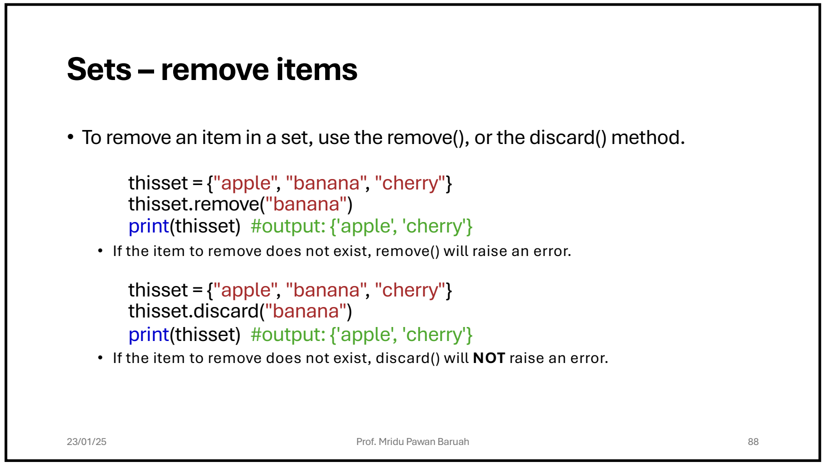
- To add items from another set into the current set, use the update() method.

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)    #output:{'apple', 'cherry', 'mango', 'papaya', 'pineapple',
'banana'}
```

- The object in the update() method does not have to be a set, it can be any iterable object

23/01/25 Prof. Mridu Pawan Baruah 87

87



Sets – remove items

- To remove an item in a set, use the remove(), or the discard() method.

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)  #output: {'apple', 'cherry'}
```

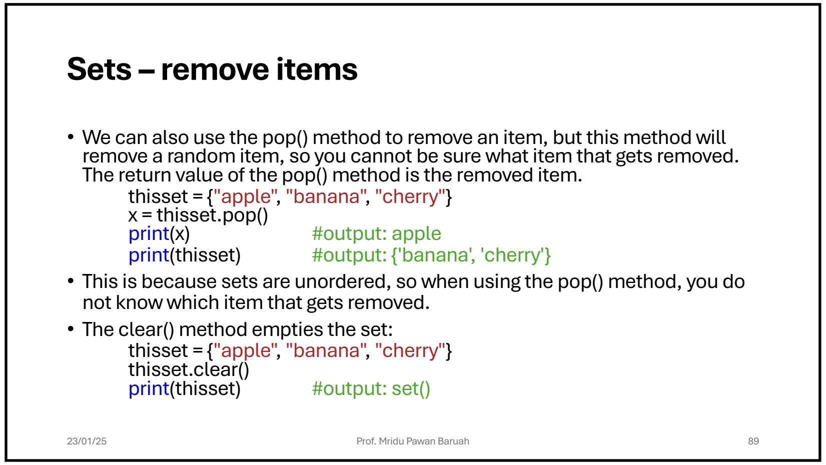
- If the item to remove does not exist, remove() will raise an error.

```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)  #output: {'apple', 'cherry'}
```

- If the item to remove does not exist, discard() will **NOT** raise an error.

23/01/25 Prof. Mridu Pawan Baruah 88

88



Sets – remove items

- We can also use the pop() method to remove an item, but this method will remove a random item, so you cannot be sure what item that gets removed. The return value of the pop() method is the removed item.

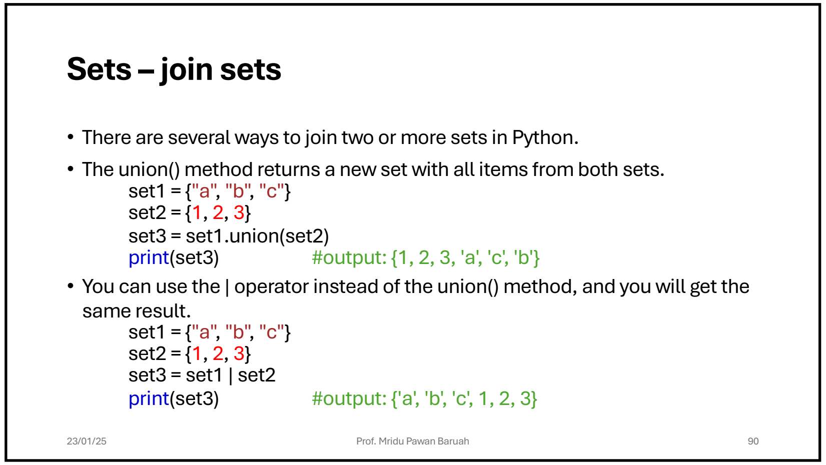
```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)                #output: apple
print(thisset)          #output: {'banana', 'cherry'}
```

- This is because sets are unordered, so when using the pop() method, you do not know which item that gets removed.
- The clear() method empties the set:

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)          #output: set()
```

23/01/25 Prof. Mridu Pawan Baruah 89

89



Sets – join sets

- There are several ways to join two or more sets in Python.
- The union() method returns a new set with all items from both sets.

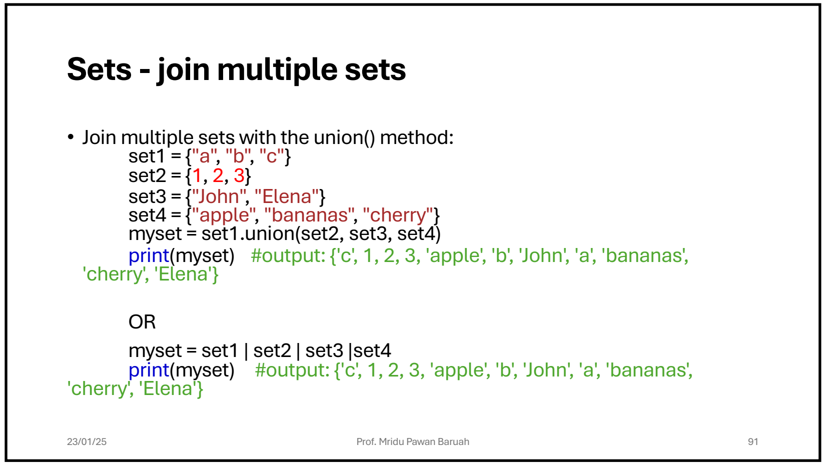
```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)             #output: {1, 2, 3, 'a', 'c', 'b'}
```

- You can use the | operator instead of the union() method, and you will get the same result.

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = set1 | set2
print(set3)             #output: {'a', 'b', 'c', 1, 2, 3}
```

23/01/25 Prof. Mridu Pawan Baruah 90

90



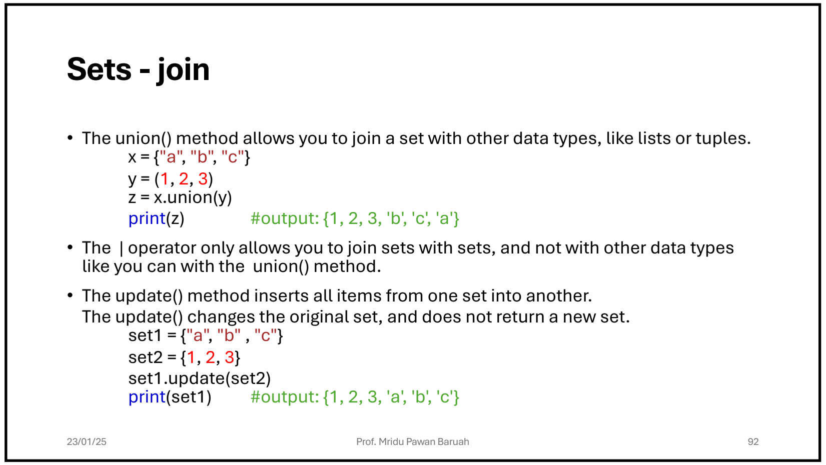
Sets - join multiple sets

- Join multiple sets with the union() method:

```
    set1 = {"a", "b", "c"}
    set2 = {1, 2, 3}
    set3 = {"John", "Elena"}
    set4 = {"apple", "bananas", "cherry"}
    myset = set1.union(set2, set3, set4)
    print(myset)   #output: {'c', 1, 2, 3, 'apple', 'b', 'John', 'a', 'bananas',
'cherry', 'Elena'}

    OR

    myset = set1 | set2 | set3 |set4
    print(myset)    #output: {'c', 1, 2, 3, 'apple', 'b', 'John', 'a', 'bananas',
'cherry', 'Elena'}
```

23/01/25 Prof. Mridu Pawan Baruah 91

91



Sets - join

- The union() method allows you to join a set with other data types, like lists or tuples.

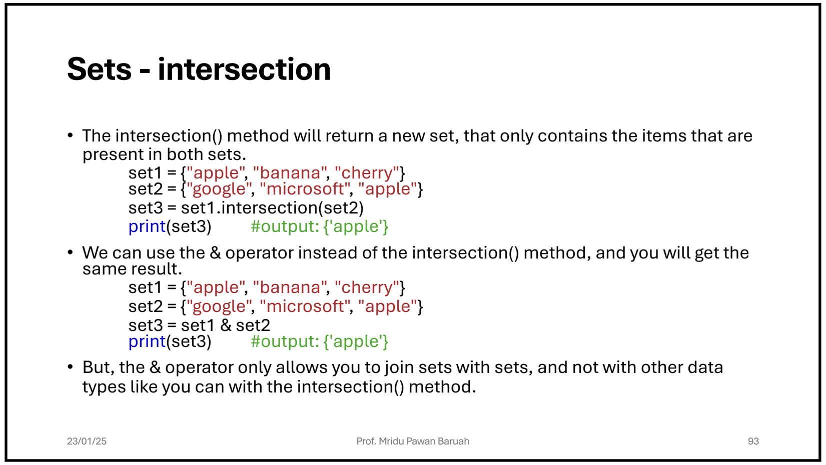
```
x = {"a", "b", "c"}
y = (1, 2, 3)
z = x.union(y)
print(z)          #output: {1, 2, 3, 'b', 'c', 'a'}
```

- The | operator only allows you to join sets with sets, and not with other data types like you can with the union() method.
- The update() method inserts all items from one set into another. The update() changes the original set, and does not return a new set.

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)       #output: {1, 2, 3, 'a', 'b', 'c'}
```

23/01/25 Prof. Mridu Pawan Baruah 92

92



Sets - intersection

- The intersection() method will return a new set, that only contains the items that are present in both sets.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1.intersection(set2)
print(set3)        #output: {'apple'}
```

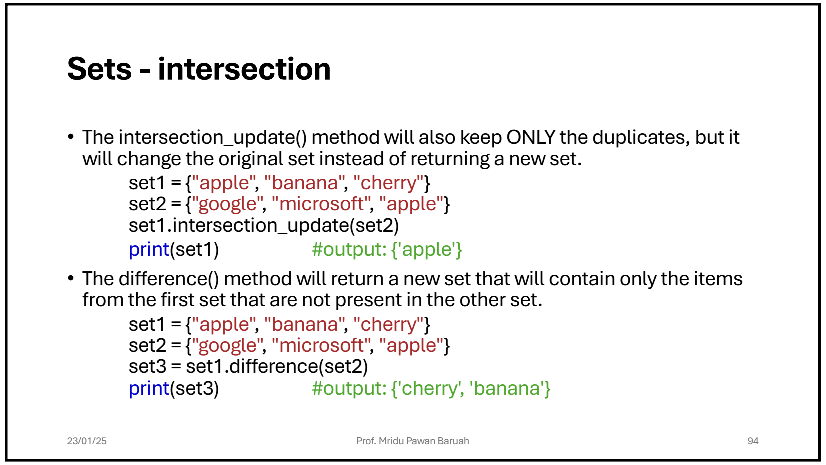
- We can use the & operator instead of the intersection() method, and you will get the same result.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1 & set2
print(set3)        #output: {'apple'}
```

- But, the & operator only allows you to join sets with sets, and not with other data types like you can with the intersection() method.

23/01/25 Prof. Mridu Pawan Baruah 93

93



Sets - intersection

- The intersection_update() method will also keep ONLY the duplicates, but it will change the original set instead of returning a new set.

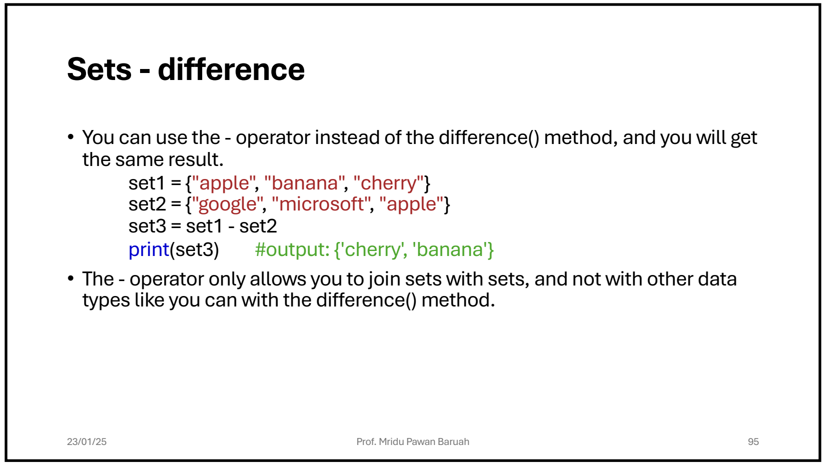
```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set1.intersection_update(set2)
print(set1)              #output: {'apple'}
```

- The difference() method will return a new set that will contain only the items from the first set that are not present in the other set.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1.difference(set2)
print(set3)              #output: {'cherry', 'banana'}
```

23/01/25 Prof. Mridu Pawan Baruah 94

94



Sets - difference

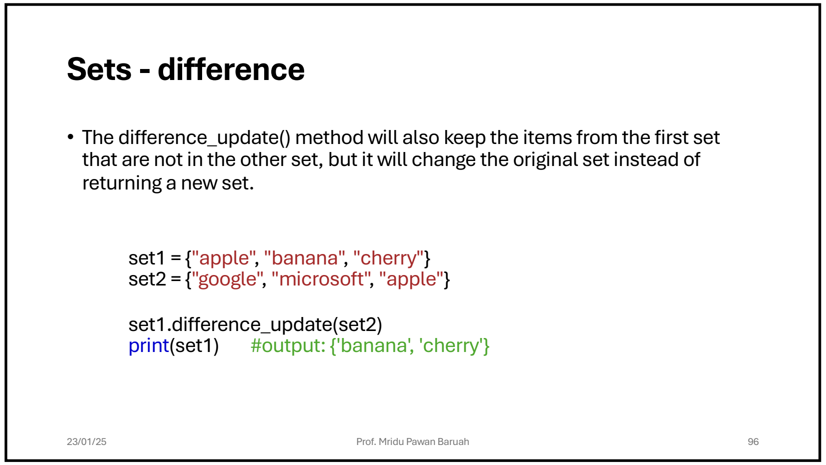
- You can use the - operator instead of the difference() method, and you will get the same result.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1 - set2
print(set3)       #output: {'cherry', 'banana'}
```

- The - operator only allows you to join sets with sets, and not with other data types like you can with the difference() method.

23/01/25 Prof. Mridu Pawan Baruah 95

95



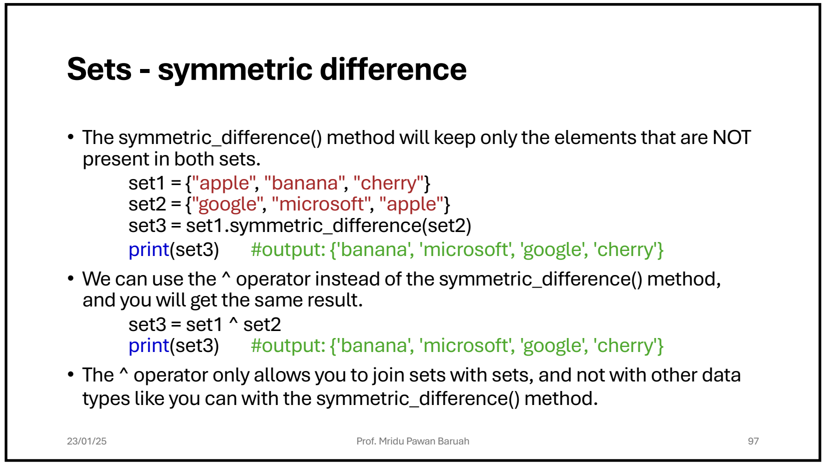
Sets - difference

- The difference_update() method will also keep the items from the first set that are not in the other set, but it will change the original set instead of returning a new set.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.difference_update(set2)
print(set1)      #output: {'banana', 'cherry'}
```

23/01/25 Prof. Mridu Pawan Baruah 96

96



Sets - symmetric difference

- The symmetric_difference() method will keep only the elements that are NOT present in both sets.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1.symmetric_difference(set2)
print(set3)      #output: {'banana', 'microsoft', 'google', 'cherry'}
```

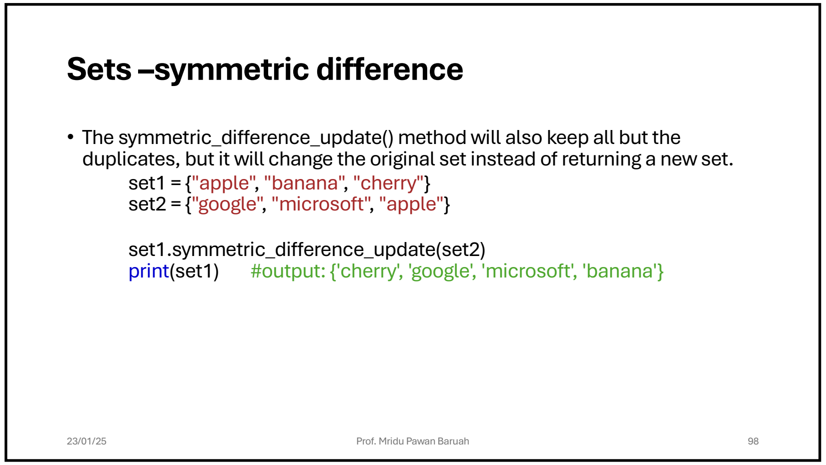
- We can use the ^ operator instead of the symmetric_difference() method, and you will get the same result.

```
set3 = set1 ^ set2
print(set3)      #output: {'banana', 'microsoft', 'google', 'cherry'}
```

- The ^ operator only allows you to join sets with sets, and not with other data types like you can with the symmetric_difference() method.

23/01/25 Prof. Mridu Pawan Baruah 97

97



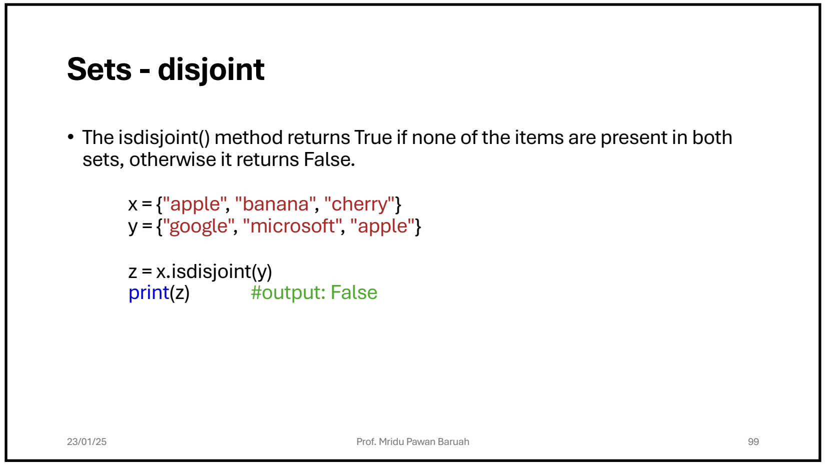
Sets –symmetric difference

- The symmetric_difference_update() method will also keep all but the duplicates, but it will change the original set instead of returning a new set.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}

set1.symmetric_difference_update(set2)
print(set1)      #output: {'cherry', 'google', 'microsoft', 'banana'}
```

23/01/25 Prof. Mridu Pawan Baruah 98

98



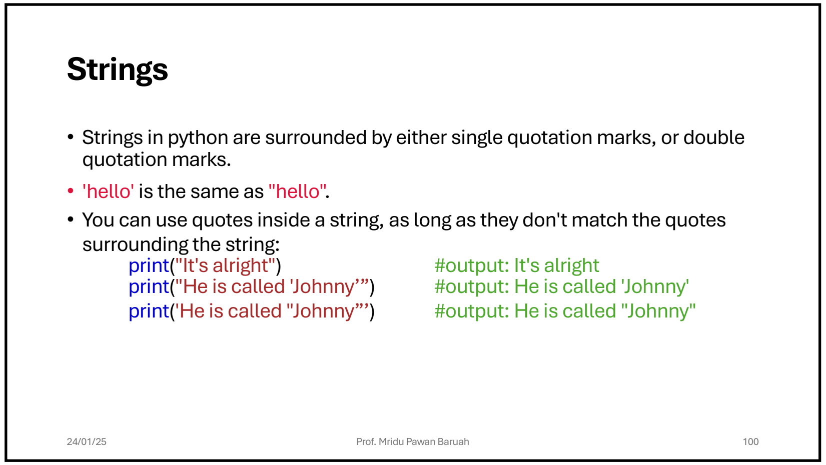
Sets - disjoint

- The isdisjoint() method returns True if none of the items are present in both sets, otherwise it returns False.

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.isdisjoint(y)
print(z)          #output: False
```

23/01/25 Prof. Mridu Pawan Baruah 99

99



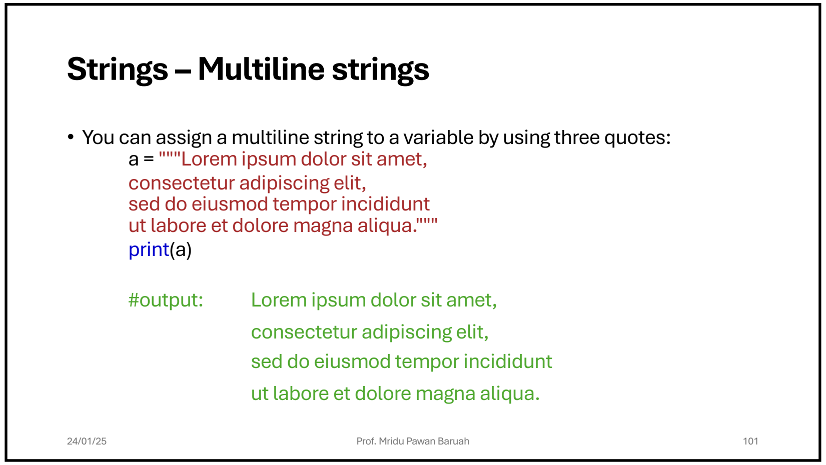
Strings

- Strings in python are surrounded by either single quotation marks, or double quotation marks.
- 'hello' is the same as "hello".
- You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
print("It's alright")                #output: It's alright
print("He is called 'Johnny’”)       #output: He is called 'Johnny'
print('He is called "Johnny”’)       #output: He is called "Johnny"
```

24/01/25 Prof. Mridu Pawan Baruah 100

100



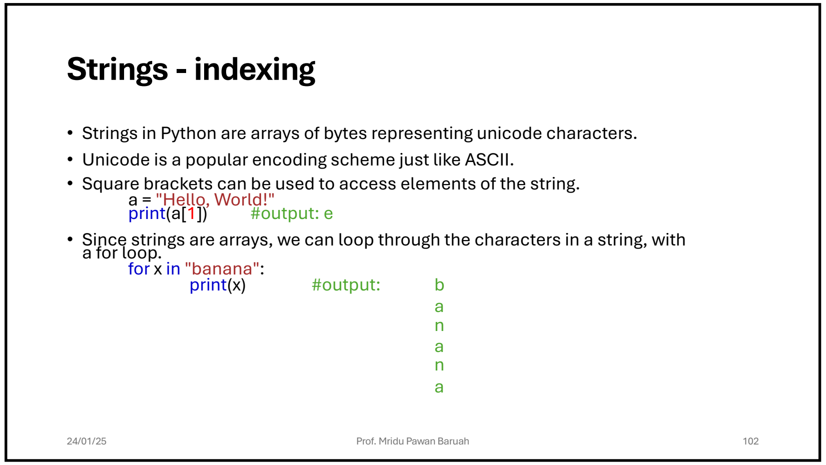
Strings – Multiline strings

- You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)

#output:     Lorem ipsum dolor sit amet,
             consectetur adipiscing elit,
             sed do eiusmod tempor incididunt
             ut labore et dolore magna aliqua.
```

24/01/25 Prof. Mridu Pawan Baruah 101

101



Strings - indexing

- Strings in Python are arrays of bytes representing unicode characters.
- Unicode is a popular encoding scheme just like ASCII.
- Square brackets can be used to access elements of the string.

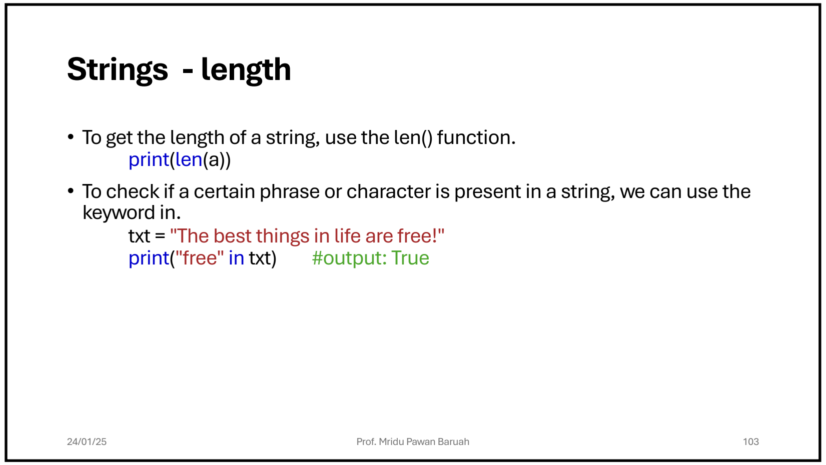
```
a = "Hello, World!"
print(a[1])        #output: e
```

- Since strings are arrays, we can loop through the characters in a string, with a for loop.

```
for x in "banana":
        print(x)          #output:      b
                                        a
                                        n
                                        a
                                        n
                                        a
```

24/01/25 Prof. Mridu Pawan Baruah 102

102



Strings - length

- To get the length of a string, use the len() function.

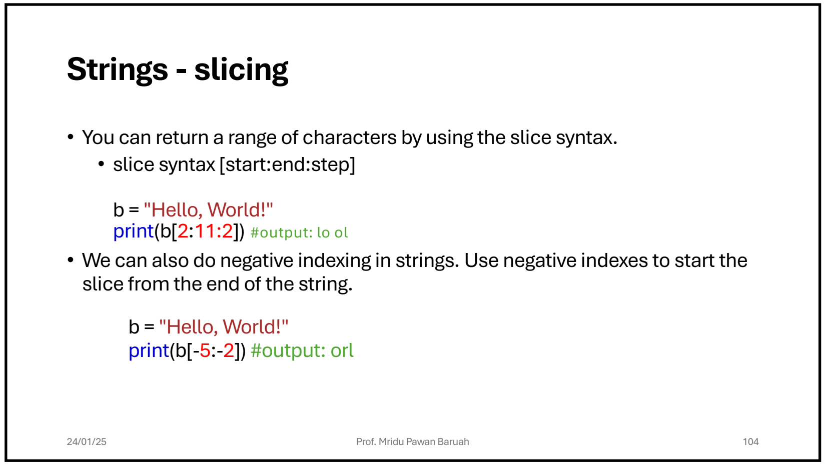
```
print(len(a))
```

- To check if a certain phrase or character is present in a string, we can use the keyword in.

```
txt = "The best things in life are free!"
print("free" in txt)      #output: True
```

24/01/25 Prof. Mridu Pawan Baruah 103

103



Strings - slicing

- You can return a range of characters by using the slice syntax.
  - slice syntax [start:end:step]

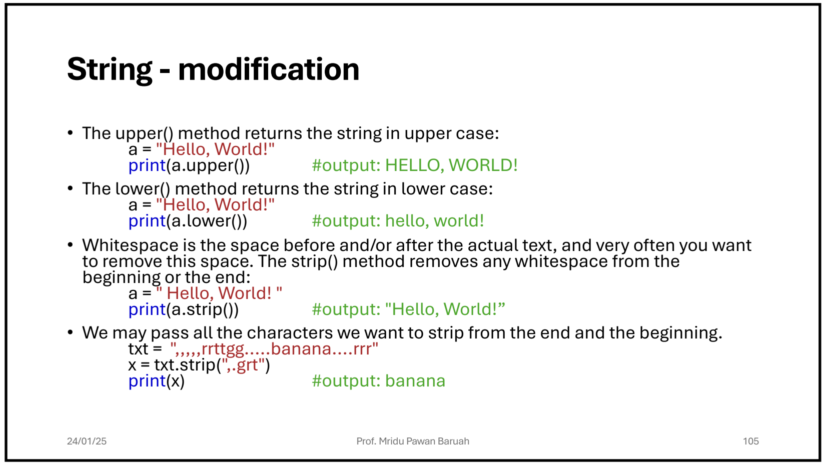
```
b = "Hello, World!"
print(b[2:11:2]) #output: lo ol
```

- We can also do negative indexing in strings. Use negative indexes to start the slice from the end of the string.

```
b = "Hello, World!"
print(b[-5:-2]) #output: orl
```

24/01/25 Prof. Mridu Pawan Baruah 104

104



String - modification

- The upper() method returns the string in upper case:

```
a = "Hello, World!"
print(a.upper())          #output: HELLO, WORLD!
```

- The lower() method returns the string in lower case:

```
a = "Hello, World!"
print(a.lower())          #output: hello, world!
```

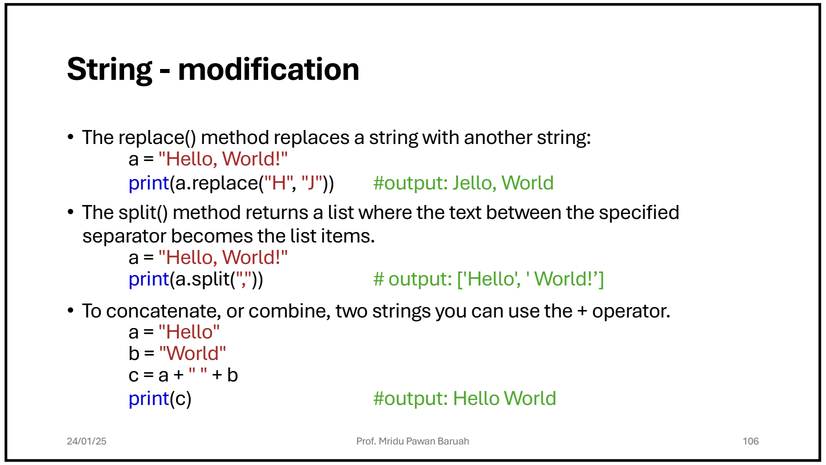
- Whitespace is the space before and/or after the actual text, and very often you want to remove this space. The strip() method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip())          #output: "Hello, World!”
```

- We may pass all the characters we want to strip from the end and the beginning.

```
txt =  ",,,,,rrttgg.....banana....rrr"
x = txt.strip(",.grt")
print(x)                  #output: banana
```

105



String - modification

- The replace() method replaces a string with another string:

```
a = "Hello, World!"
print(a.replace("H", "J"))     #output: Jello, World
```

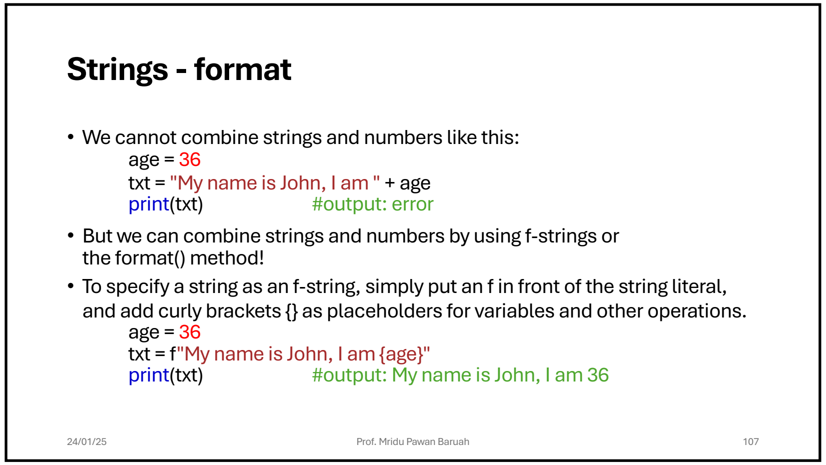
- The split() method returns a list where the text between the specified separator becomes the list items.

```
a = "Hello, World!"
print(a.split(","))            # output: ['Hello', ' World!’]
```

- To concatenate, or combine, two strings you can use the + operator.

```
a = "Hello"
b = "World"
c = a + " " + b
print(c)                       #output: Hello World
```

106



Strings - format

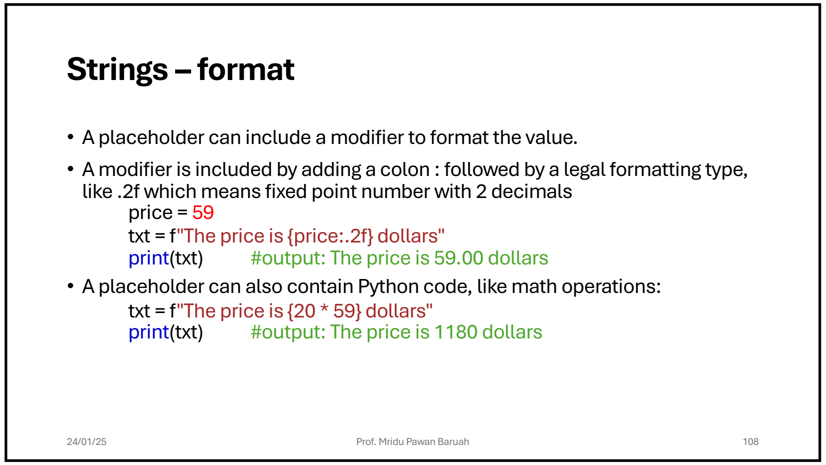
- We cannot combine strings and numbers like this:

```
age = 36
txt = "My name is John, I am " + age
print(txt)                #output: error
```

- But we can combine strings and numbers by using f-strings or the format() method!
- To specify a string as an f-string, simply put an f in front of the string literal, and add curly brackets {} as placeholders for variables and other operations.

```
age = 36
txt = f"My name is John, I am {age}"
print(txt)                #output: My name is John, I am 36
```

107



Strings – format

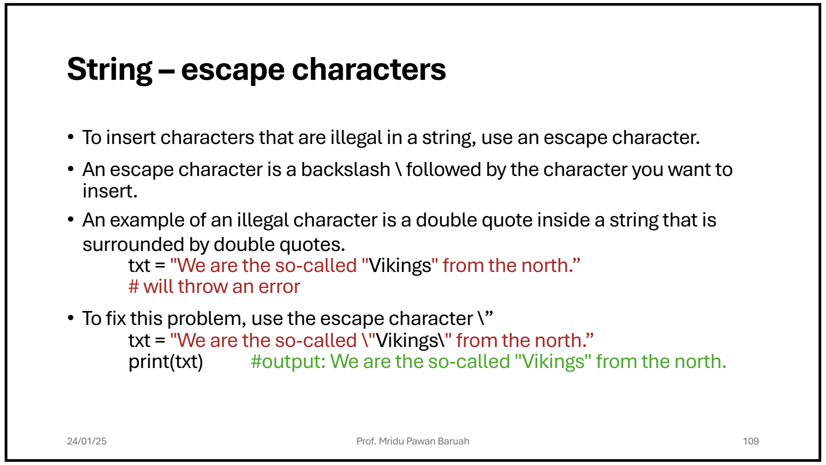
- A placeholder can include a modifier to format the value.
- A modifier is included by adding a colon : followed by a legal formatting type, like .2f which means fixed point number with 2 decimals

```
price = 59
txt = f"The price is {price:.2f} dollars"
print(txt)        #output: The price is 59.00 dollars
```

- A placeholder can also contain Python code, like math operations:

```
txt = f"The price is {20 * 59} dollars"
print(txt)        #output: The price is 1180 dollars
```

108



String – escape characters

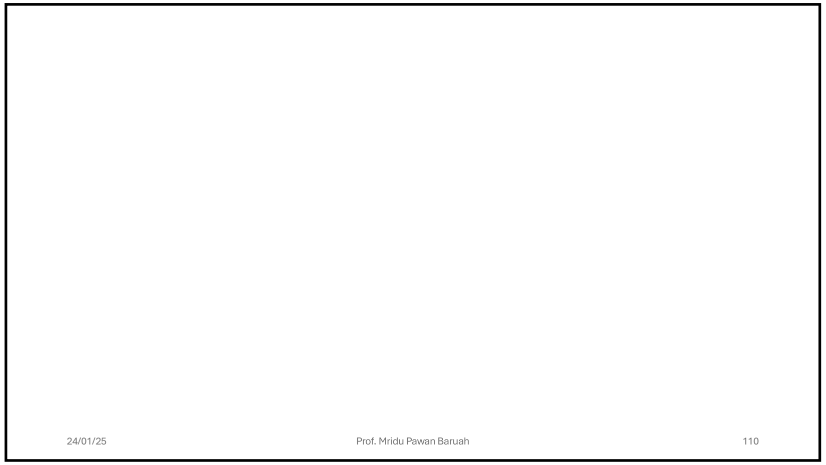
- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash \ followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes.

```
txt = "We are the so-called "Vikings" from the north.”
# will throw an error
```

- To fix this problem, use the escape character \”

```
txt = "We are the so-called \"Vikings\" from the north.”
print(txt)        #output: We are the so-called "Vikings" from the north.
```

109



110