Lists, Tuples, Sets, Dictionaries

Final Revision

Python Lists

```
mylist = ["apple", "banana", "cherry"]
```

Lists are used to **store multiple items in a single variable.**

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are **Tuple, Set, and Dictionary**, all with different qualities and usage.

Lists are created using square brackets: **[]**

Example

```
thislist = ["apple", "banana", "cherry"]

print(thislist)
```

List Items

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]`

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

Changeable

Allow Duplicates

Example

A list can contain different data types:

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

Tuples

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

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

Example

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

print(thistuple)
```

Ordered

```
Unchangeable

Allow Duplicates
```

Example

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

print(thistuple)
```

Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
thistuple = ("apple",)

print(type(thistuple))


#NOT a tuple

thistuple = ("apple")

print(type(thistuple))
```

Tuple Items - Data Types

Tuple items can be of any data type:

```
tuple1 = ("apple", "banana", "cherry")

tuple2 = (1, 5, 7, 9, 3)

tuple3 = (True, False, False)

tuple4 = ("abc", 34, True, 40, "male")
```

Access Tuple Items

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

print(thistuple[1])
```

Negative/ Range Indexing

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

```
print(thistuple[-1])
```

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
```

```
print(thistuple[2:5])
```

Update Tuples

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.

But there are some workarounds.

You can convert the tuple into a list, change the list, and convert the list back into a tuple.

ExampleGet

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

y = list(x)

y[1] = "kiwi"

x = tuple(y)



print(x)
```

Add Items

Since tuples are immutable, they do not have a built-in `append()` method, but there are other ways to add items to a tuple.

Convert into a list: Just like the workaround for *changing* a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

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

y = list(thistuple)

y.append("orange")

thistuple = tuple(y)
```

Add Item 2

Add tuple to a tuple. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

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

y = ("orange",)

thistuple += y


print(thistuple)
```

Unpack Tuples

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"**:

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

(green, yellow, red) = fruits

print(green)

print(yellow)

print(red)
```

Using Asterisk*

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

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

```
print(green)

print(yellow)

print(red)
```

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.

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, *tropic, red) = fruits

print(green)

print(tropic)

print(red)
```

Loop Through a Tuple

You can loop through the tuple items by using a for loop.

\

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

for x in thistuple:

  print(x)
```

Join Two Tuples

```
tuple1 = ("a", "b" , "c")

tuple2 = (1, 2, 3)



tuple3 = tuple1 + tuple2

print(tuple3)
```

Multiply Tuples

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

mytuple = fruits * 2



print(mytuple)
```

Sets

```
myset = {"apple", "banana", "cherry"}
```

A set is a collection which is unordered, unchangeable*, and unindexed.

ExampleGet

```
thisset = {"apple", "banana", "cherry"}

print(thisset)
```

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Example

```
thisset = {"apple", "banana", "cherry", "apple"}
```

```
print(thisset)
```

```python
thisset = {"apple", "banana", "cherry", True, 1, 2}

print(thisset)
```

```python
thisset = {"apple", "banana", "cherry", False, True, 0}

print(thisset)
```

Set Items - Data Types

```
set1 = {"apple", "banana", "cherry"}

set2 = {1, 5, 7, 9, 3}

set3 = {True, False, False}

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

Which one of these is a set?

1. myset = ('apple', 'banana', 'cherry')
2. myset = ['apple', 'banana', 'cherry']
3. myset = {'apple', 'banana', 'cherry'}

Access Items

You cannot access items in a set by referring to an **index or a key.**

But you 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**.

Example

```python
thisset = {"apple", "banana", "cherry"}

for x in thisset:

  print(x)
```

```python
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

```python
thisset = {"apple", "banana", "cherry"}

print("banana" not in thisset)
```

Add Set Items

Once a set is created, you cannot change its items, but you can add new items.

To add one item to a set use the `add()` method.

```
thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)
```

Add Sets

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)
```

Add Any Iterable

The object in the update() method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

```
thisset = {"apple", "banana", "cherry"}

mylist = ["kiwi", "orange"]


thisset.update(mylist)


print(thisset)
```

Remove Set Items

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

```
thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

Note: If the item to remove does not exist, `discard()` will NOT raise an error.

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

pop()

You 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

```
thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)
```

Note: 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)
```

The `del` keyword will delete the set completely:

```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)
```

Loop Items

```
thisset = {"apple", "banana", "cherry"}


for x in thisset:

  print(x)
```

Join Sets

There are several ways to join two or more sets in Python.

The `union()` and `update()` methods joins all items from both sets.

The `intersection()` method keeps ONLY the duplicates.

The `difference()` method keeps the items from the first set that are not in the other set(s).

The `symmetric_difference()` method keeps all items EXCEPT the duplicates.

https://www.w3schools.com/python/python_sets_join.asp

Dictionaries

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

Example

```
thisdict = {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

print(thisdict)
```

Dictionary Items

Dictionary items are ordered, changeable, and do not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

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

Duplicates Not Allowed

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```

Dictionary Items - Data Types

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

Exercise

Which one of these is a dictionary?

```
x = ('apple', 'banana', 'cherry')
x = {'type' : 'fruit', 'name' : 'banana'}
x = ['apple', 'banana', 'cherry']
```

Accessing Item

You can access the items of a dictionary by referring to its key name, inside square brackets:

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

get()

There is also a method called get() that will give you the same result:

```
x = thisdict.get("model")
```

Get Keys

The keys() method will return a list of all the keys in the dictionary.

Get a list of the keys:

```
x = thisdict.keys()
```

```
car = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}


x = car.keys()


print(x) #before the change

car["color"] = "white"

print(x) #after the change
```

Get Values

The values() method will return a list of all the values in the dictionary.

```
x = thisdict.values()
```

```
The list of the values is a view of the dictionary, meaning that any changes done to the
dictionary will be reflected in the values list.
```

```python
car = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}


x = car.values()

print(x) #before the change


car["year"] = 2020

print(x) #after the change
```