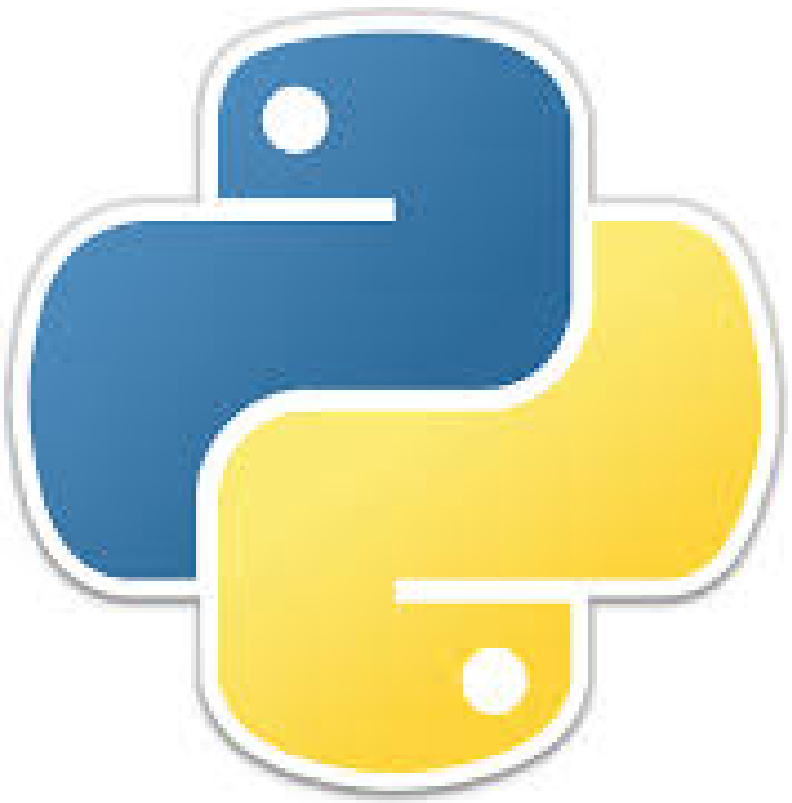

Python Collections

21.09.2019

Python Notes

INTRO

- There are four collection data types in the Python programming language:
  - **List** is a collection which is `ordered` and `changeable`. Allows duplicate members.
  - **Tuple** is a collection which is `ordered` and `unchangeable`. Allows duplicate members.
  - **Set** is a collection which is `unordered` and `unindexed`. No duplicate members.
  - **Dictionary** is a collection which is `unordered`, `changeable` and `indexed`. No duplicate members.
- When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

LISTS

Intro

- A list is a collection which is **ordered** and **changeable**. In Python lists are written with square brackets.

Example: Create a List

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

Access Items

- You access the list items by referring to the index number:

Example: Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1]) #prints banana
```

Negative Indexing

- Negative indexing means beginning from the end, `-1` refers to the last item, `-2` refers to the second last item etc.

Example: Print the last item of the list:

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

Range of Indexes

- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new list with the specified items.

Example: Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5]) # prints index 2, index 3, index 4 elements
```

Change the Item Value

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

Example: Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
```

Loop Through a List

- You can loop through the list items by using a `for` loop:

Example: Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Check If Item Exists

- To determine if a specified item is present in a list use the `in` keyword:

Example: Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

List Length

- To determine how many items a list has, use the `len()` method:

Example: Print the number of items in the list:

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

Add Items

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

Example : Using the append() method to append an item:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
```

- To add an item at the specified index, use the `insert()` method:

Example : Insert an item as the second position:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange") #indexes 2 and 3 are shifted.
```

Remove Items

- There are several methods to remove items from a list:

Example : The remove() method removes the specified item:

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

Example : The pop() method removes the specified index, (or the last item if index is not specified):

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

Example : The del keyword removes the specified index:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
```

Example : The del keyword can also delete the list completely:

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

Example : The clear() method empties the list:

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

Copy a List

- You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.
- There are ways to make a copy, one way is to use the built-in List method `copy()`.

- Another way to make a copy is to use the built-in method `list()`.

Example : Make a copy of a list with the copy() method or with the list() method:

```
thislist = ["apple", "banana", "cherry"]
copyList1 = thislist.copy()
copyList2 = list(thislist)
```

The list() Constructor

- It is also possible to use the list() constructor to make a new list.

Example: Using the list() constructor to make a List:

```
thislist = list(("apple", "banana", "cherry"))
```

TUPLES

Intro

- A tuple is a collection which is **ordered** and **unchangeable**. In Python tuples are written with round brackets.

Example: Create a Tuple:

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

Access Items

- Very similar with lists.
- You can access tuple items by referring to the index number, inside square brackets:

Example: Print the second item of the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Change Tuples Values

- Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.
- But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example: Convert the tuple into a list to be able to change it:

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

print(x)
```

Loop Through a Tuple

- Very similar with lists.

Check If Item Exists

- Very similar with lists.

Tuple Length

- Very similar with lists.

Add Items

- `Once a tuple is created, you cannot add items to it. Tuples are unchangeable`

Create Tuple With one Item

- To create a tuple with only one item, you have add a comma after the item, unless Python will not recognize the variable as a tuple.

Example: One item tuple, remember the commma:

```
thistuple = ("apple",)
print(type(thistuple)) #prints tuple

thistuple = ("apple")
print(type(thistuple)) #prints str
```

Remove Items

- `Tuples are unchangeable, so you cannot remove items from it, but you can delete the tuple completely`

Example: The del keyword can delete the tuple completely:

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

The tuple() Constructor

- Similar to lists, It is also possible to use the tuple() constructor to make a tuple.

Example: Using the tuple() method to make a tuple:

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

SETS

Intro

- A set is a collection which is **unordered** and **unindexed**. In Python sets are written with curly brackets.

Example: Create a Set:

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

Access Items

- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.
- 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.

Change Items

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

Loop Through a Set

- Very similar with lists and tuples

Example: Loop through the set, and print the values:

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

for x in thisset:
  print(x)
```

Check If Item Exists

- Very similar with lists and tuples

Tuple Length

- Very similar with lists and tuples.

Add Items

- To add one item to a set use the **`add()`** method.
- To add more than one item to a set use the **`update()`** method.

Example: Add an item to a set, using the add() method or Add multiple items to a set, using the update() method:

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

thisset.add("orange") # one item is added.
thisset.update(["orange", "mango", "grapes"]) # multiple items are added.
```

Remove Items

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

Example: Remove "banana" by using the remove() method:

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

thisset.remove("banana")
```

- The **`del`** keyword will delete the set completely.

The set() Constructor

- Similar to lists, It is also possible to use the set() constructor to make a set.

Example: Using the set() method to make a set:

```
thisset = set(("apple", "banana", "cherry"))
```

DICTIONARIES

Intro

- A dictionary is a collection which is **unordered**, **changeable** and **indexed**. In Python dictionaries are written with curly brackets, and they have **keys** and **values**.

Example: Create and print a dictionary:

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

Access Items

- You can access the items of a dictionary by referring to its key name, inside square brackets:
- There is also a method called `get()` that will give you the same result:

Example: Get the value of the "model" key:

```
x = thisdict["model"]
#or
x = thisdict.get("model")
```

Change Values

- You can change the value of a specific item by referring to its key name:

Example: Change the "year" to 2018:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
```

Loop Through a Dictionary

- Very similar with lists, tuples and sets. But notice that there are keys and values for dictionaries.

Example: Print all key names in the dictionary, one by one:

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

```
for x in thisdict: # x represents keys.
  print(x)
```

Example: Print all values in the dictionary, one by one:

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

Example: Loop through both keys and values, by using the items() function:

```
thisdict =    {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

Check If Key Exists

- To determine if a specified key is present in a dictionary use the `in` keyword:

Example: Check if "model" is present in the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Dictionary Length

- Very similar with lists tuples and sets.

Add Items

- Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example:

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

Remove Items

- There are several methods to remove items from a dictionary:

Example: The pop() method removes the item with the specified key name:

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

Example: The popitem() method removes the last inserted item

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

Example: The del keyword removes the item with the specified key name:

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

Example: The clear() keyword empties the dictionary:

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

Copy a Dictionary

- You cannot copy a dictionary simply by typing `dict2 = dict1`, because: `dict2` will only be a ***reference*** to `dict1`, and changes made in `dict1` will automatically also be made in `dict2`.
- There are ways to make a copy, one way is to use the built-in Dictionary method `copy()`.

Example: Make a copy of a dictionary with the copy() method:

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

```
mydict = thisdict.copy()
print(mydict
```

Example: Make a copy of a dictionary with the dict() method:

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

Nested Dictionaries

- A dictionary can also contain many dictionaries, this is called nested dictionaries.

Example: Create a dictionary that contain three dictionaries:

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

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}

print(myfamily)
```

The dict() Constructor

- It is also possible to use the dict() constructor to make a new dictionary.

Example:

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)
```

More blogs