**Python Collection Variables**

With JavaScript we learned to use arrays and objects which can be classified as collection data types. These are variables that contain multiple items. Python has numerous versions of arrays and object like variables. They differ based on whether they can:

1. contain **duplicate** items e.g. **[ 1, 2, 3, 1 ]**
2. be **changed** after being created
3. have their items **ordered** e.g. the first element added is in the first position
4. are **indexed**, we can access their items with indexed position numbers

There are 4 main types of collection variables in Python: **Lists**, **Tuples**, **Sets** and **Dictionaries**.

**List** <https://techvidvan.com/tutorials/python-lists/>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Duplicates** | **Changeable** | **Ordered** | **Indexed** | **[ ]** |

Lists are the most like JavaScript arrays. Even the syntax of using square brackets is the same **[]**.

**myList = [ "A", "B", "C", "D" ]**

**myListCasted = list( ("A", "B", "C", "D") )**

**myList[ 0 ] #** *"A"*

**myList[ -1 ] #** *"D" gets last element in list*

**Slicing :**

A very handy thing to use with Python lists is slicing. Using a colon **:** when accessing elements let’s you grab specific elements only and generates a shallow copy of the list – one that doesn’t mutate the original list.

**myList [ start:stop ]** ***#*** *items start through stop-1*

**myList [ start: ]** ***#*** *items start through the rest of the array*

**myList [ :stop ]**  ***#*** *items from the beginning through stop-1*

**myList [ : ]** ***#*** *a copy of the whole array*

Stop is the first item that is not selected in the slice.

**myList[ 2: ] # [** *"C", "D"* **]**

**myList[ :2 ] # [** *"A", "B"* **]**

**myList[ 1:-1 ] # [** *"B", "C"* **]**

You can also add a second colon : and another number to represent the steps in which you’d like to slice items. If you want to slice every second item in the list you would use the number 2 in the steps position.

**myList[ start:stop:steps ]** ***#*** *items start through stop-1*

**myList[ 0:3:2 ] # [** *"A", "C"* **]**

**myList[ ::2 ] # [** *"A", "C"* **]**

**len( myList ) #** *4*

**myList.append( "D" ) #** *Adds "D" to end of list*

**myList.extend( ["E", "F"] ) #** *Adds list of elements to end of list*

**myList + [ "G", "H" ] # *+*** *also adds collections together*

**myList.pop() #** *Removes & returns last element*

**"C" in myList #** *True*

**Tuple**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Duplicates** | **Changeable** | **Ordered** | **Indexed** | **( )** |

Tuples are very similar to Lists however they are unchangeable. They are created using rounded brackets **()**.

**myTuple = ( "A", "B", "C", "C" )**

**myTupleCasted = tuple( ("A", "B", "C", "C") )**

**myTuple.append( "D" ) #** *ERROR can’t change a tuple*

**Sets**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Duplicates** | **Changeable** | **Ordered** | **Indexed** | **{ }** |

Sets are unchangeable collections that contain only unique values which are not ordered and not indexed. They are created using curly brackets **{ }**.

**mySet = { "A", "B", "C", "D", "D" } #** *Only 1 “D” will be added to this set*

**mySetCasted = set( ("A", "B", "C", "D", "D") )**

Casting for sets is very useful. You can cast a **list** to a **set** and it will return only the unique elements (remove all the duplicates).

**mySetCasted = set( myList ) #** *All duplicates are now gone*

**mySet[ 0 ] #** *ERROR sets are not indexed/ordered*

**"X" in mySet #** *False*

**Dictionary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Duplicates** | **Changeable** | **Ordered\*** | **Indexed** | **{ }** |

Dictionaries, shorten to **dict**, are collection variables that store values with corresponding keys. You access the values using the correct key much like objects in JavaScript. Dictionaries have become ordered since Python V3.7. They are created using curly brackets **{ }**. Take note that the keys in the dictionary must be “strings”

**myDict = { "A": 1, "B": 2, "C": 3, "D": 4 }**

**myDict[ "C" ] #** *3*

**list( myDict.keys() ) #** *[“A”, “B”, “C”, “D”]*

**list( myDict.values() ) #** *[1, 2, 3, 4]*

**Useful Collection Methods and Codes**

To demonstrate some of the useful in-built Python methods on collection variables we’re going to use the name **col** to represent an arbitrary collection variable. Because some of these methods don’t apply to certain types, the table below will highlight which ones work and don’t work.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **List** | **Tuple** | **Set** | **Dictionary** |
| **len( col )** | Returns number of items |  |  |  |  |
| **col.append( “a” )** | Adds single item to end |  |  |  |  |
| **col.extend([“b”, “c”])** | Adds multiple items to end |  |  |  |  |
| **col.pop( 1 )** | Removes & returns item at position |  |  |  | **col.pop(“key”)** |
| **col.remove( “a” )** | Removes specific item |  | co |  |  |
| **col[ 0 ]** | Returns item at index |  |  |  | **col[“key”]** |
| **“a” in col** | Returns True if item is present |  |  |  | “a” is key |
| **col.copy()** | Creates shallow copy of collection |  |  |  |  |
| **col.index( “a” )** | Returns first position of item and ERROR if isn’t present |  |  |  |  |
| **col.count( “a” )** | Returns number of occurrences |  |  |  |  |