Swift 4 dictionaries are used to store unordered lists of values of the same type. Swift 4 puts strict checking which does not allow you to enter a wrong type in a dictionary even by mistake.

Swift 4 dictionaries use unique identifier known as a key to store a value which later can be referenced and looked up through the same key. Unlike items in an array, items in a dictionary do not have a specified order. You can use a dictionary when you need to look up values based on their identifiers.

A dictionary key can be either an integer or a string without a restriction, but it should be unique within a dictionary.

If you assign a created dictionary to a variable, then it is always mutable which means you can change it by adding, removing, or changing its items. But if you assign a dictionary to a constant, then that dictionary is immutable, and its size and contents cannot be changed.

Creating Dictionary

You can create an empty dictionary of a certain type using the following initializer syntax –

```
var someDict = [KeyType: ValueType]()
```

You can use the following simple syntax to create an empty dictionary whose key will be of Int type and the associated values will be strings –

```
var someDict = [Int: String]()
```

Here is an example to create a dictionary from a set of given values –

```
var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]
```

Sequence Based Initialization

Swift 4 allows you to create Dictionary from arrays (Key-Value Pairs.)

```
var cities = [“Delhi”,”Bangalore”,”Hyderabad”]
```

You can use the following simple syntax to create an empty dictionary whose key will be of Int type and the associated values will be strings –

```
var Distance = [2000,10, 620]
```

Here is an example to create a dictionary from a set of given values –

```
let cityDistanceDict = Dictionary(uniqueKeysWithValues: zip(cities, Distance))
```

The above lines of code will create a dictionary with Cities as key and Distance as Value –

Filtering

Swift 4 allows you to filter values from a dictionary.

```
var closeCities = cityDistanceDict.filter { $0.value < 1000 }
```

If we run the above code our closeCities Dictionary will be.

```
["Bangalore" : 10 , "Hyderabad" : 620]
```

Dictionary Grouping

Swift 4 allows you to create grouping of Dictionary values.

```
var cities = ["Delhi","Bangalore","Hyderabad","Dehradun","Bihar"]
```

You can use the following simple syntax to group the values of dictionary according to first alphabet.

```
var GroupedCities = Dictionary(grouping: cities ) { $0.first! }
```

The result of above code will be

```
["D" :["Delhi","Dehradun"], "B" : ["Bengaluru","Bihar"], "H" : ["Hyderabad"]]
```

Accessing Dictionaries

You can retrieve a value from a dictionary by using subscript syntax, passing the key of the value you want to retrieve within square brackets immediately after the name of the dictionary as follows –

```
var someVar = someDict[key]
```

Let's check the following example to create, initialize, and access values from a dictionary –

```
var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]
var someVar = someDict[1]

print( "Value of key = 1 is \(someVar)" )
print( "Value of key = 2 is \(someDict[2])" )
print( "Value of key = 3 is \(someDict[3])" )
```

When the above code is compiled and executed, it produces the following result –

```
Value of key = 1 is Optional("One")
Value of key = 2 is Optional("Two")
Value of key = 3 is Optional("Three")
```

Modifying Dictionaries

You can use updateValue(forKey:) method to add an existing value to a given key of the dictionary. This method returns an optional value of the dictionary's value type. Here is a simple example –

```
var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]
var oldVal = someDict.updateValue("New value of one", forKey: 1)
var someVar = someDict[1]

print( "Old value of key = 1 is \(oldVal)" )
print( "Value of key = 1 is \(someVar)" )
print( "Value of key = 2 is \(someDict[2])" )
print( "Value of key = 3 is \(someDict[3])" )
```

When the above code is compiled and executed, it produces the following result –

```
Old value of key = 1 is Optional("One")
Value of key = 1 is Optional("New value of one")
Value of key = 2 is Optional("Two")
Value of key = 3 is Optional("Three")
```

You can modify an existing element of a dictionary by assigning new value at a given key as shown in the following example –

```
var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]
var oldVal = someDict[1]
someDict[1] = "New value of one"
var someVar = someDict[1]

print( "Old value of key = 1 is \(oldVal)" )
print( "Value of key = 1 is \(someVar)" )
```

```
print( "Value of key = 2 is \(someDict[2])" )
print( "Value of key = 3 is \(someDict[3])" )
```

When the above code is compiled and executed, it produces the following result –

```
Old value of key = 1 is Optional("One")
Value of key = 1 is Optional("New value of one")
Value of key = 2 is Optional("Two")
Value of key = 3 is Optional("Three")
```

Remove Key-Value Pairs

You can use removeValueForKey() method to remove a key-value pair from a dictionary. This method removes the key-value pair if it exists and returns the removed value, or returns nil if no value existed. Here is a simple example –

```
var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]
var removedValue = someDict.removeValue(forKey: 2)

print( "Value of key = 1 is \(someDict[1])" )
print( "Value of key = 2 is \(someDict[2])" )
print( "Value of key = 3 is \(someDict[3])" )
```

When the above code is compiled and executed, it produces the following result –

```
Value of key = 1 is Optional("One")
Value of key = 2 is nil
Value of key = 3 is Optional("Three")
```

You can also use subscript syntax to remove a key-value pair from a dictionary by assigning a value of nil for that key. Here is a simple example –

```
var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]

someDict[2] = nil

print( "Value of key = 1 is \(someDict[1])" )
print( "Value of key = 2 is \(someDict[2])" )
print( "Value of key = 3 is \(someDict[3])" )
```

When the above code is compiled and executed, it produces the following result –

```
Value of key = 1 is Optional("One")
Value of key = 2 is nil
Value of key = 3 is Optional("Three")
```

Iterating Over a Dictionary

You can use a for-in loop to iterate over the entire set of key-value pairs in a Dictionary as shown in the following example –

```
var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]

for (index, keyValue) in someDict.enumerated() {
print("Dictionary key \(index) - Dictionary value \(keyValue)")
}
```

When the above code is compiled and executed, it produces the following result –

```
Dictionary key 2 - Dictionary value Two
Dictionary key 3 - Dictionary value Three
Dictionary key 1 - Dictionary value One
```

You can use enumerate() function which returns the index of the item along with its (key, value) pair as shown below in the example –

```
var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]
for (key, value) in someDict.enumerated() {
print("Dictionary key \(key) - Dictionary value \(value)")
}
```

When the above code is compiled and executed, it produces the following result –

```
Dictionary key 0 - Dictionary value (key: 2, value: "Two")
Dictionary key 1 - Dictionary value (key: 3, value: "Three")
Dictionary key 2 - Dictionary value (key: 1, value: "One")
```

Convert to Arrays

You can extract a list of key-value pairs from a given dictionary to build separate arrays for both keys and values. Here is an example –

```
var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]

let dictKeys = [Int](someDict.keys)
let dictValues = [String](someDict.values)

print("Print Dictionary Keys")

for (key) in dictKeys {
print("\(key)")
```

```
}
print("Print Dictionary Values")

for (value) in dictValues {
print("\(value)")
}
```

When the above code is compiled and executed, it produces the following result –

```
Print Dictionary Keys
2
3
1
Print Dictionary Values
Two
Three
One
```

The count Property

You can use the read-only count property of a dictionary to find out the number of items in a dictionary as shown below –

```
var someDict1:[Int:String] = [1:"One", 2:"Two", 3:"Three"]
var someDict2:[Int:String] = [4:"Four", 5:"Five"]

print("Total items in someDict1 = \(someDict1.count)")
print("Total items in someDict2 = \(someDict2.count)")
```

When the above code is compiled and executed, it produces the following result –

```
Total items in someDict1 = 3
Total items in someDict2 = 2
```

The empty Property

You can use read-only empty property of a dictionary to find out whether a dictionary is empty or not, as shown below –

```
var someDict1:[Int:String] = [1:"One", 2:"Two", 3:"Three"]
var someDict2:[Int:String] = [4:"Four", 5:"Five"]
var someDict3:[Int:String] = [Int:String]()

print ("someDict1 = \(someDict1.isEmpty)")
```

```
print("someDict2 = \(someDict2.isEmpty)")
print("someDict3 = \(someDict3.isEmpty)")
```

When the above code is compiled and executed, it produces the following result –

```
someDict1 = false
someDict2 = false
someDict3 = true
```