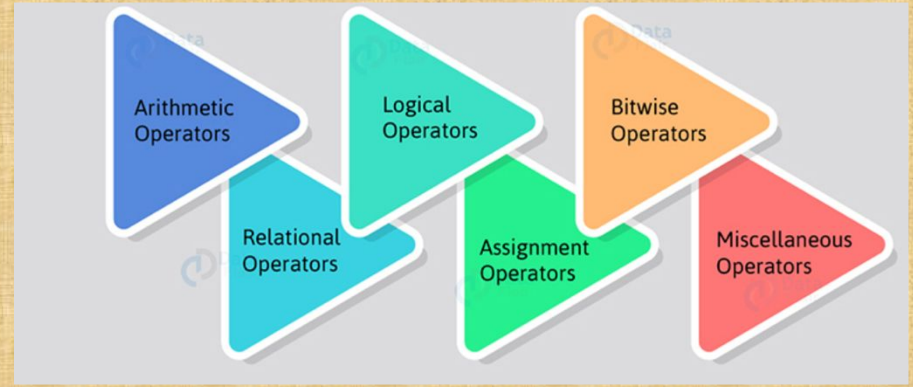
iOS Programming

Lecture 3

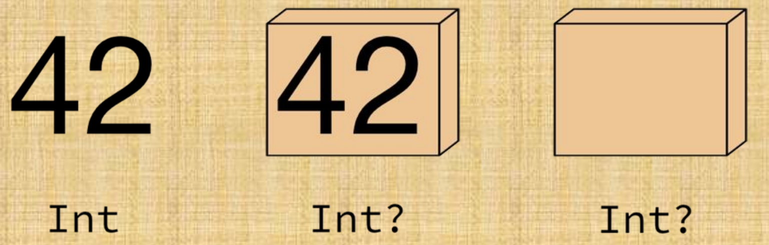
Recap

Variables and Constants

Operators

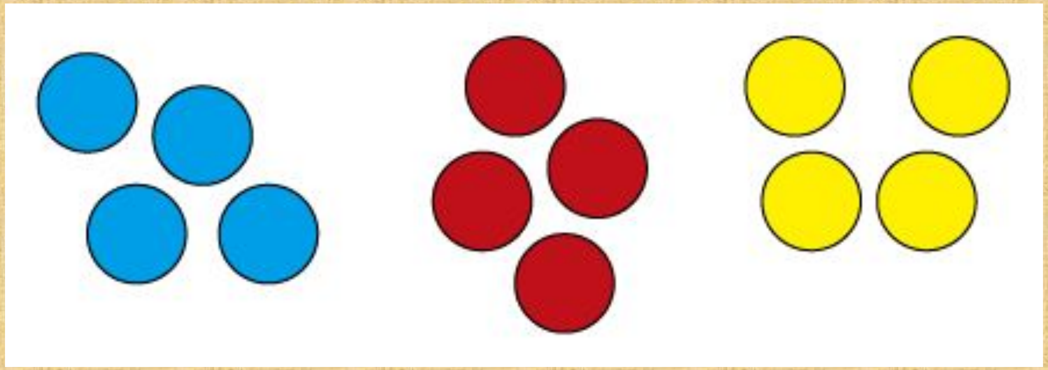


Optionals



Today

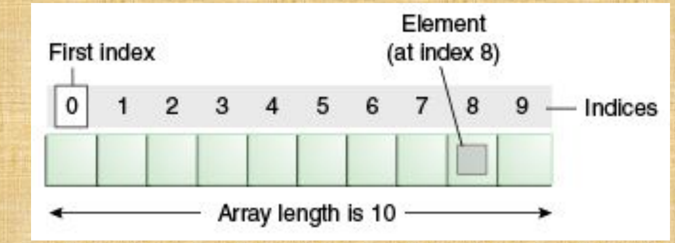
Collections

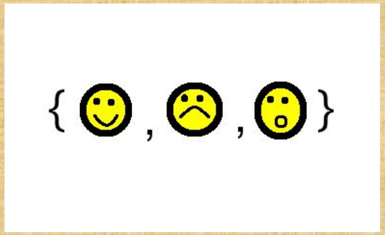


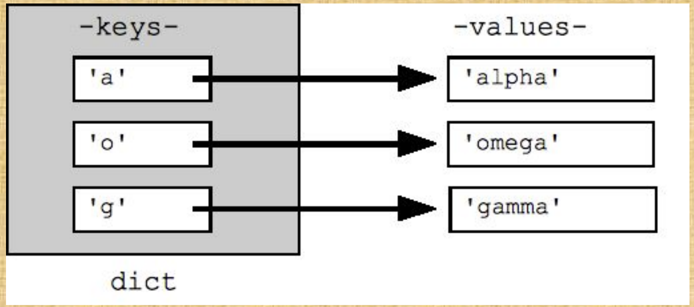
Array

Set

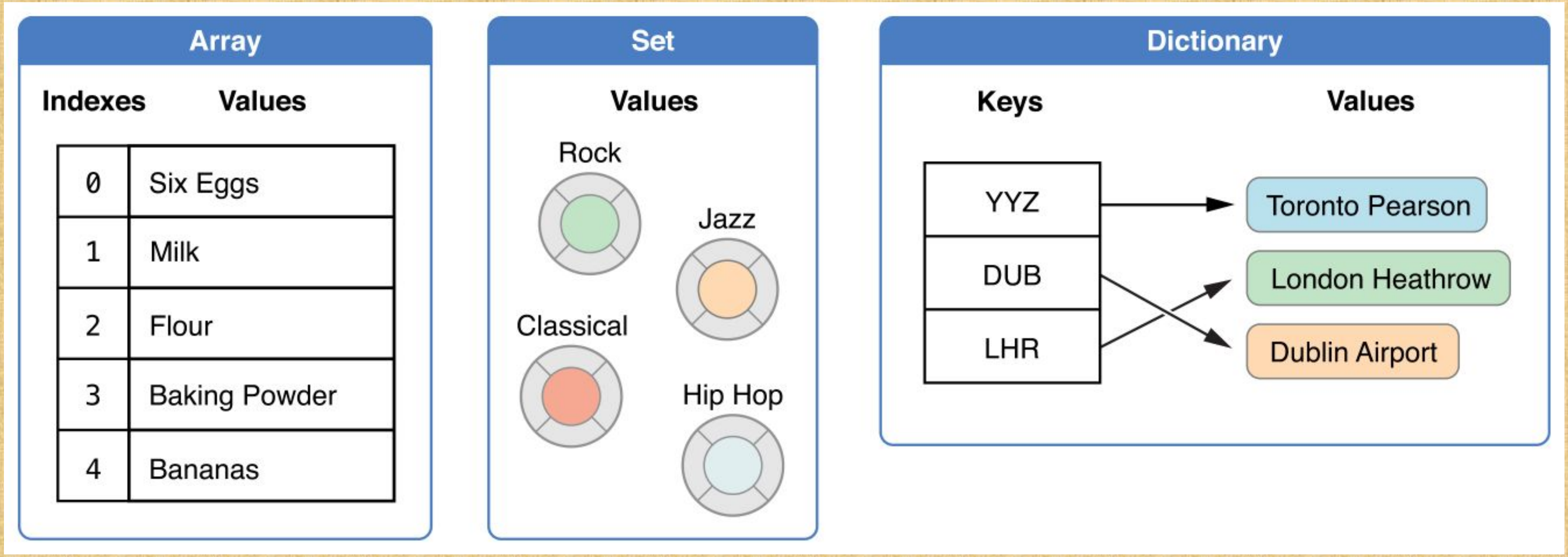
Dictionary







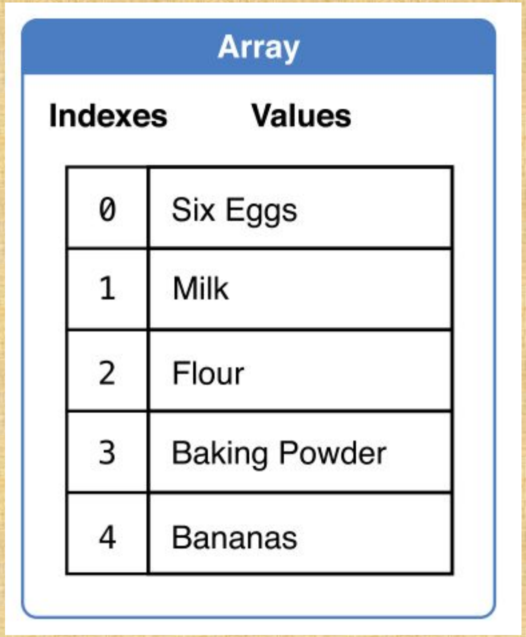
Collections



Ordered

Unordered

Arrays



Array

Indexes	Values
0	Six Eggs
1	Milk
2	Flour
3	Baking Powder
4	Bananas

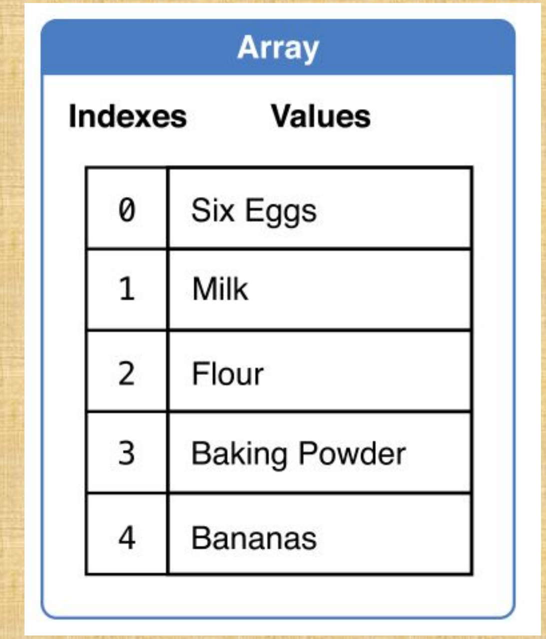
Zero Based Index

Ordered

Type Safe

Arrays

```
let arraySample = ["Six Eggs", "Milk",
                   "Flour", "Baking Powder", "Bananas"]


print(arraySample.count) //Count - 5
print(arraySample[0])    //Note - We start at
zero for Six Eggs
print(arraySample[1])    //Milk is index 1
```



Array

Indexes	Values
0	Six Eggs
1	Milk
2	Flour
3	Baking Powder
4	Bananas

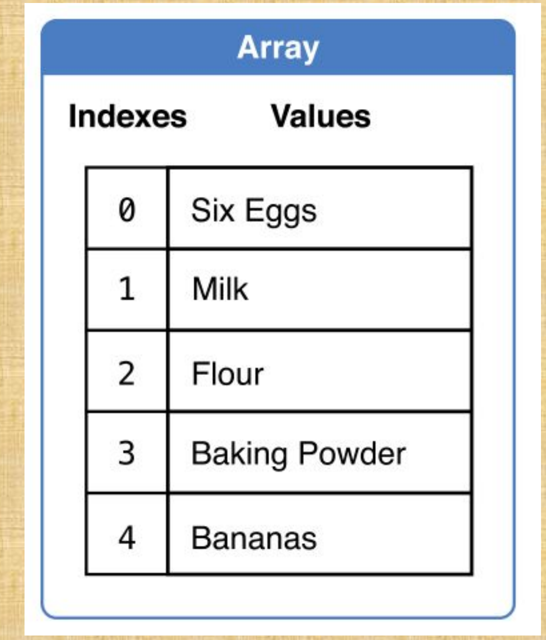
Arrays – Type Inference

```
let arraySample = ["Six Eggs", "Milk",
                   "Flour", "Baking Powder", "Bananas"]
```

and

```
let arraySample: [String] = ["Six Eggs", "Milk",
                   "Flour", "Baking Powder", "Bananas"]
```

is same. Swift inferred the type as String in the first case



Array

Indexes	Values
0	Six Eggs
1	Milk
2	Flour
3	Baking Powder
4	Bananas

Arrays – Difference from other languages

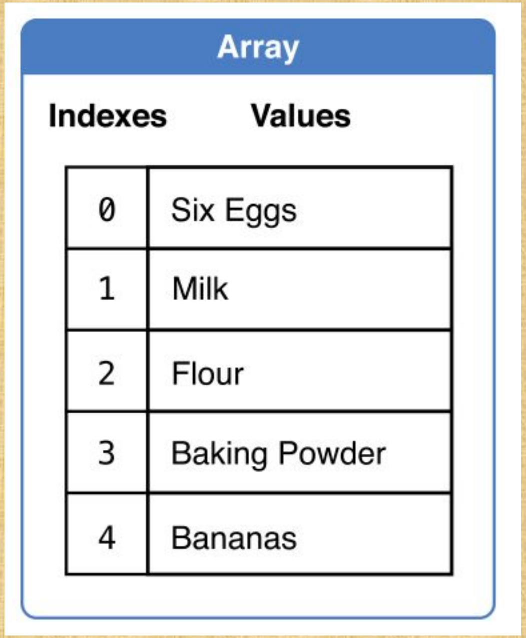
Trailing commas are ok

```
let arraySample = ["Six Eggs", "Milk",
                "Flour", "Baking Powder", "Bananas"]
```

and

```
let arraySample = ["Six Eggs", "Milk",
                "Flour", "Baking Powder", "Bananas”, ]
```

are one and the same thing



Array

Indexes	Values
0	Six Eggs
1	Milk
2	Flour
3	Baking Powder
4	Bananas

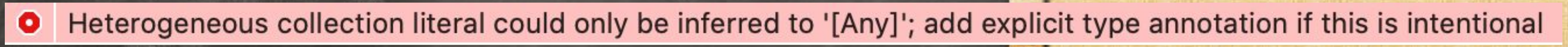
Arrays – Single Data Type

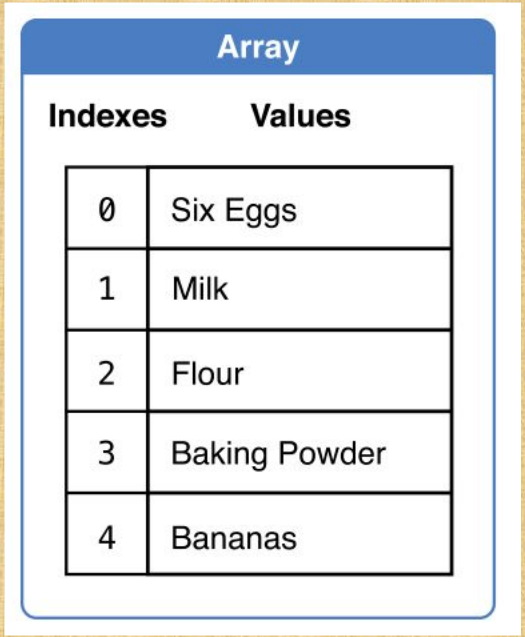
```
let arraySample = ["Six Eggs", "Milk",
              "Flour", "Baking Powder", "Bananas"]
```

is good, but

```
let arraySample = ["Six Eggs", "Milk",
              "Flour", "Baking Powder", 5,
"Bananas" ]
```

is not


Heterogeneous collection literal could only be inferred to '[Any]'; add explicit type annotation if this is intentional


Array

Indexes	Values
0	Six Eggs
1	Milk
2	Flour
3	Baking Powder
4	Bananas

Arrays – Mutable vs Immutable

No distinction like Array vs ArrayList

```
var variableArray = ["Six Eggs", "Milk",
                "Flour", "Baking Powder", "Bananas"]
```

is variable, but

```
let constantArray = ["Six Eggs", "Milk",
                "Flour", "Baking Powder", "Bananas" ]
```

is not

Arrays – Mutable vs Immutable

No distinction like Array vs ArrayList

```
var variableArray = ["Six Eggs", "Milk",
                 "Flour", "Baking Powder", "Bananas"]
print (variableArray.count) //5

//We can append and increase and size of the array
variableArray.append("Apples")
print (variableArray.count) //6

let constantArray = ["Cherries", "Butter", "Whipped Cream"]
//We can append array to an array
variableArray.append(contentsOf: constantArray)
print (variableArray.count) //9
```

Arrays – Insert and Remove

```
var variableArray = ["Six Eggs", "Milk",
                     "Flour", "Baking Powder", "Bananas"]
print (variableArray.count) //5

//We can also add element at a particular index
variableArray.insert("Cherries", at: 1)

//We can also remove element at a particular index
variableArray.remove(at: 0)

print (variableArray)
```

Arrays – Bounds are important

```
var variableArray = ["Six Eggs", "Milk",
                     "Flour", "Baking Powder", "Bananas"]
print (variableArray.count) //5

//Let's try some edge cases
variableArray.insert("Cherries", at: 15)
```

```
error: Execution was interrupted, reason: EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0).
```

```
//We can also remove element at a particular index
variableArray.remove(at: 15)
```

```
error: Execution was interrupted, reason: EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0).
```

```
print (variableArray)
```

Arrays – Bounds are important

```
var variableArray = ["Six Eggs", "Milk",
                     "Flour", "Baking Powder", "Bananas"]
print (variableArray.count) //5

//Safely remove element
if variableArray.count > 15 {
    variableArray.remove(at: 15)
}

print (variableArray)
```

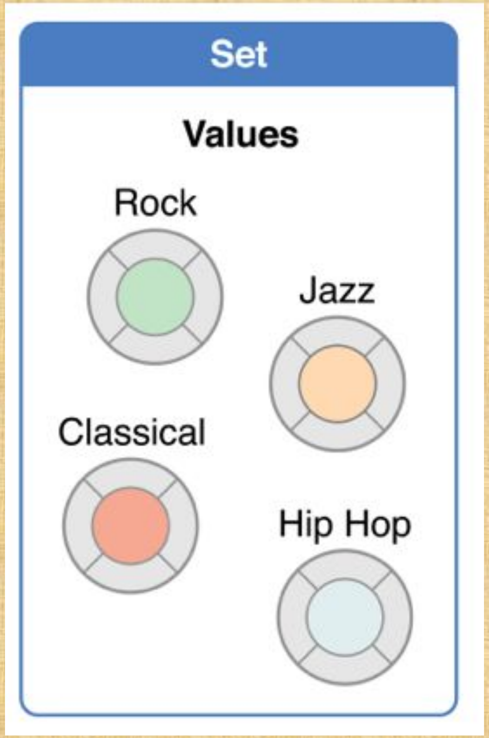
Arrays – Initialization

```
var stringArrayUnInt: [String]
//Let's try and ad something
stringArrayUnInt.append("Hello")
```

Variable 'stringArrayUnInt' passed by reference before being initialized

```
//Create an empty string array
var stringArray: [String] = []
//Let's try and ad something
stringArray.append("Hello")

//Create an empty double array
var doubleArray: [Double] = []
```

Set



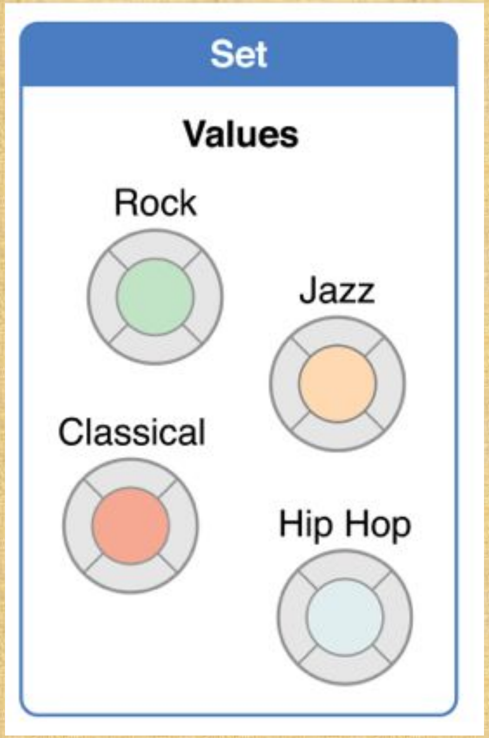
Unique

Unordered

Type Safe

Set

```
let setSample: Set = ["Rock", "Jazz", "Classical",
"Hip Hop"]

for music in setSample {
    print(music, terminator:", ")
}

print()
print ("====================================")

let setSample2: Set = ["Rock", "Jazz", "Classical",
"Hip Hop"]
for music in setSample2 {
    print(music, terminator:", ")
}
```



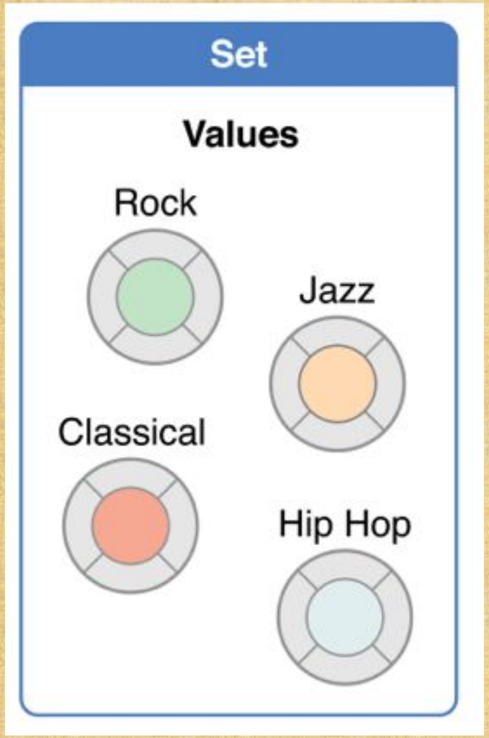
Set

Jazz, Classical, Hip Hop, Rock,
=====================================
Hip Hop, Rock, Jazz, Classical,

Why is the ordering different?

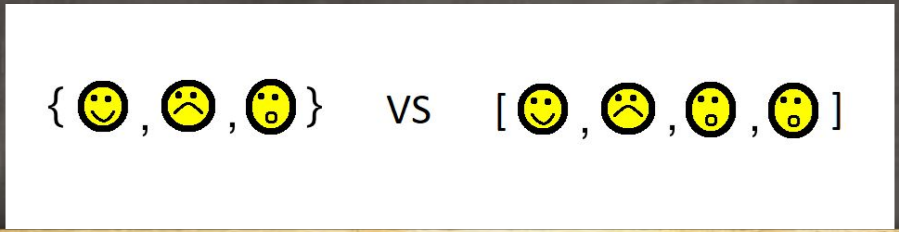
Remember:

Unordered



Set

```
var setSample: Set = ["Rock", "Jazz", "Classical", "Hip Hop"]

for music in setSample {
    print(music, terminator:", ")
}

print ()

var (inserted, memberAfterInsert) = setSample.insert("Jazz")
print (inserted)

========================================

false - Why??
```



Unique

Set

```swift
var setSample: Set = ["Rock", "Jazz", "Classical", 5, "Hip Hop"]
```

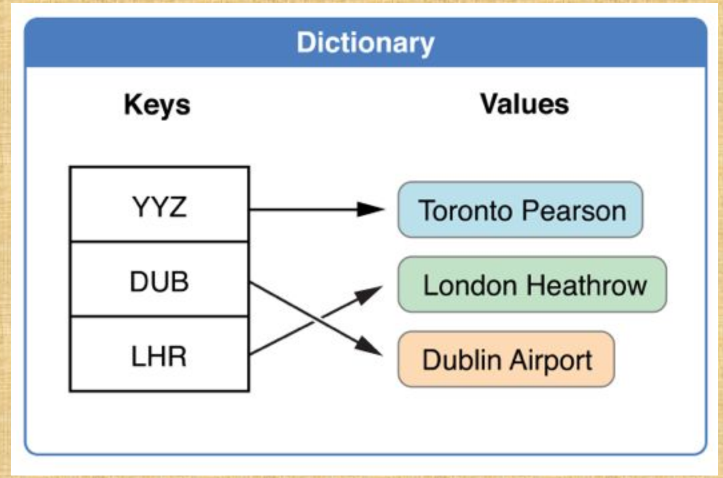
```
Cannot convert value of type 'Int' to expected element type 'String'
```

Type Safe

Collection – Default Type Inference

```
var setSample: Set = ["Rock", "Jazz", "Classical", "Hip Hop"]
print (type(of: setSample))
//Set<String>
for music in setSample {
    print(music, terminator: ", ")
}

print ()
var collectionSample = ["Rock", "Jazz", "Classical", "Hip Hop"]
print (type(of: collectionSample))
//Array<String>
for music in collectionSample {
    print(music, terminator: ", ")
}
```

Dictionary



Key-Value pairs

Unordered

Type Safe

Dictionary

Key-Value pairs

```
var antonyms = [
    "Hot": "Cold",
    "Sunny": "Cloudy",
    "Light": "Dark",
    "Clear": "Murky"
]
print (type(of: antonyms)) //Dictionary<String, String>

antonyms["true"] = "false" //["Sunny", "Hot", "Clear", "true", "Light"]
var whatsUnderCup = [
    0: "Empty",
    1: "Empty",
    2: "Gold Coin",
    3: "Empty"
]
print (type(of: whatsUnderCup)) //Dictionary<Int, String>
```

Dictionary

Unordered

```
var antonyms = [
   "Hot": "Cold",
   "Sunny": "Cloudy",
   "Light": "Dark",
   "Clear": "Murky"
]

print (type(of: antonyms)) //Dictionary<String, String>

antonyms["true"] = "false"

//Un-ordered
print (antonyms.keys) //["Sunny", "Hot", "Clear", "true", "Light"]
```

Dictionary

Type Safe

```
var antonyms = [
   "Hot": "Cold",
   "Sunny": "Cloudy",
   "Light": "Dark",
   "Clear": "Murky"
]

print (type(of: antonyms)) //Dictionary<String, String>

antonyms["Smart"] = 9
```

```
Cannot assign value of type 'Int' to subscript of type 'String'
```

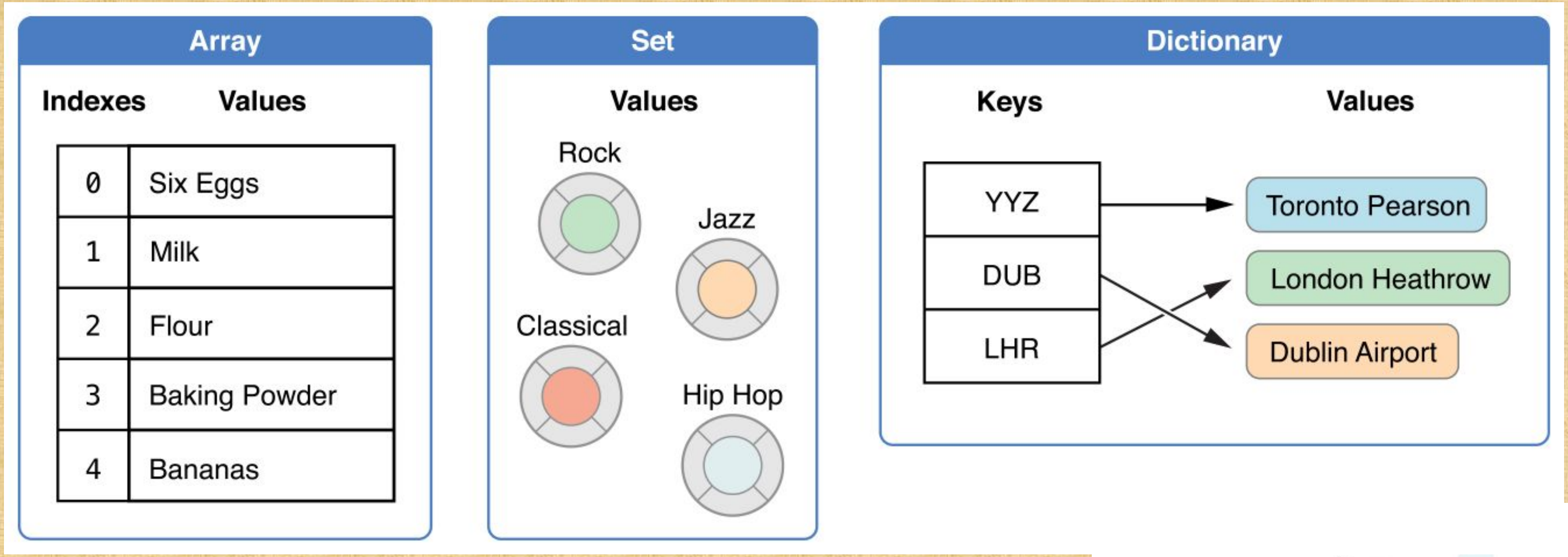
```
//Un-ordered
print (antonyms.keys) //["Sunny", "Hot", "Clear", "true", "Light"]
```

Dictionary – Complex Dictionaries

```
var contactChannels: [String: Array<String>] = [
    "Email": ["something@random.com", "kingkong@movie.com",
"jurassic@park.com"],
    "Phone": ["9801234567", "9804561290"]
]

//Get Value of a Key
if let emails = contactChannels["Email"] {
    print(emails) //["something@random.com", "kingkong@movie.com",
"jurassic@park.com"]
} else{
    print("No email available on file")
}
```

Parting Notes

Today, we explored collections



Do Practice – Write Code

- Arrays – CRUD (Create Read Update Delete)
- Set – CRUD
- Dictionary – CRUD
- Think and document Use Cases