Chapter 58: Logging in Swift

Section 58.1: dump

`dump` prints the contents of an object via reflection (mirroring).

Detailed view of an array:

```
let names = ["Joe", "Jane", "Jim", "Joyce"]
dump(names)
```

Prints:

> ▿ 4 elements
> - [0]: Joe
> - [1]: Jane
> - [2]: Jim
> - [3]: Joyce

For a dictionary:

```
let attributes = ["foo": 10, "bar": 33, "baz": 42]
dump(attributes)
```

Prints:

> ▿ 3 key/value pairs
> ▿ [0]: (2 elements)
> - .0: bar
> - .1: 33
> ▿ [1]: (2 elements)
> - .0: baz
> - .1: 42
> ▿ [2]: (2 elements)
> - .0: foo
> - .1: 10

`dump` is declared as `dump(_:name:indent:maxDepth:maxItems:)`.

The first parameter has no label.

There's other parameters available, like `name` to set a label for the object being inspected:

```
dump(attributes, name: "mirroring")
```

Prints:

> ▿ mirroring: 3 key/value pairs
> ▿ [0]: (2 elements)

- .0: bar
- .1: 33
▿ [1]: (2 elements)
- .0: baz
- .1: 42
▿ [2]: (2 elements)
- .0: foo
- .1: 10

You can also choose to print only a certain number of items with `maxItems:`, to parse the object up to a certain depth with `maxDepth:`, and to change the indentation of printed objects with `indent:`.

Section 58.2: Debug Print

Debug Print shows the instance representation that is most suitable for debugging.

```
print("Hello")
debugPrint("Hello")

let dict = ["foo": 1, "bar": 2]

print(dict)
debugPrint(dict)
```

Yields

```
>>> Hello
>>> "Hello"
>>> [foo: 1, bar: 2]
>>> ["foo": 1, "bar": 2]
```

This extra information can be very important, for example:

```
let wordArray = ["foo", "bar", "food, bars"]

print(wordArray)
debugPrint(wordArray)
```

Yields

```
>>> [foo, bar, food, bars]
>>> ["foo", "bar", "food, bars"]
```

Notice how in the first output it appears that there are 4 elements in the array as opposed to 3. For reasons like this, it is preferable when debugging to use `debugPrint`

Updating a classes debug and print values

```
struct Foo: Printable, DebugPrintable {
    var description: String {return "Clear description of the object"}
    var debugDescription: String {return "Helpful message for debugging"}
}

var foo = Foo()

print(foo)
```

```
debugPrint(foo)

>>> Clear description of the object
>>> Helpful message for debugging
```

Section 58.3: print() vs dump()

Many of us start debugging with simple `print()`. Let's say we have such a class:

```
class Abc {
    let a = "aa"
    let b = "bb"
}
```

and we have an instance of `Abc` as so:

```
let abc = Abc()
```

When we run the `print()` on the variable, the output is

```
App.Abc
```

while `dump()` outputs

```
App.Abc #0
- a: "aa"
- b: "bb"
```

As seen, `dump()` outputs the whole class hierarchy, while `print()` simply outputs the class name.

Therefore, `dump()` is especially useful for UI debugging

```
let view = UIView(frame: CGRect(x: 0, y: 0, width: 100, height: 100))
```

With `dump(view)` we get:

```
 - <UIView: 0x108a0cde0; frame = (0 0; 100 100); layer = <CALayer: 0x159340cb0>> #0
    - super: UIResponder
      - NSObject
```

While `print(view)` we get:

```
<UIView: 0x108a0cde0; frame = (0 0; 100 100); layer = <CALayer: 0x159340cb0>>
```

There is more info on the class with `dump()`, and so it is more useful in debugging the class itself.

Section 58.4: print vs NSLog

In swift we can use both `print()` and `NSLog()` functions to print something on Xcode console.

But there are lot of differences in `print()` and `NSLog()` functions, such as:

1 TimeStamp: `NSLog()` will print timestamp along with the string we passed to it, but `print()` will not print timestamp.
e.g.

```
let array = [1, 2, 3, 4, 5]
print(array)
NSLog(array.description)
```

Output:

> [1, 2, 3, 4, 5]
> 2017-05-31 13:14:38.582 ProjetName[2286:7473287] [1, 2, 3, 4, 5]

It'll also print **ProjectName** along with timestamp.

2 Only String: `NSLog()` only takes String as an input, but `print()` can print any type of input passed to it.
e.g.

```
let array = [1, 2, 3, 4, 5]
print(array) //prints [1, 2, 3, 4, 5]
NSLog(array) //error: Cannot convert value of type [Int] to expected argument type 'String'
```

3 Performance: `NSLog()` function is very **slow** compare to `print()` function.

4 Synchronization: `NSLog()` handles simultaneous usage from multi-threading environment and prints output without overlapping it. But `print()` will not handle such cases and jumbles while prating output.

5 Device Console: `NSLog()` outputs on device console also, we can see this output by connecting our device to Xcode. `print()` will not print output to device's console.