Classes, Structures, Functions, Instantitatiion Guide for xcode

Creating variables:

—-

“let” creates a constant that cannot be changed later on.

let a=10

—-

“var” creates a variable that can be changed with code later on.

var b=5

—-

If you want to terminate multiple statements in a line, use a “;”.

let a=10 **;** var b=5

—-

Creating variables with specific types and a default string

var greeting: String = “unknown”

^ *This statement declares greeting as a variable, specifies the variable as a string type, and then gives it a default value of unknown.*

—-

Creating variables that can have a nil value

```
var greeting2: String?
print(greeting2 ?? "default")
```

Because the variable greeting 2 does not have a value and has a “?” after String, it can have a nil value. Because there is “??” inside the print statement, it is saying that if the variable greeting 2 has a nil value, it outputs the string “default”

—-

Private variables are not accessible outside of their class or method. This ensures that data is kept secret from unauthorized programmers. This is an example:

```
class BankAccount{

        private var value: Double

        init(value: Double) {
                self.value = value
        }
}
```

The value variable is listed as private so that no one can access the data outside of the BankAccount class.
—-

If statements:
—-
A basic example of an if statement with a predefined variable

```
var greeting: String = “unknown”
var lang = “french”

if lang == “french”
{greeting = “bonjour”}-

else if lang == “english”
{greeting = “hello”}

print(greeting)
```

This if statement changes the variable greeting based on what string the variable lang is assigned to, and then it prints greeting.
—-

Lists and Tuples:
—-
How to create a list:

```
var purchasedCars: [Car] = []
```

This creates a variable that is a list named purchasedCars that can hold an array of objects, named “Car” to create a tuple, do the same but use parenthesis instead of brackets
—
How to append onto a list with a function:

```
Func addCar(car:Car){
        purchasedCars.append
addCar(car: car1)
```

This piece of code has a function with a variable that is called Car and is instantiated using “car:” this allows the variable, car1, which holds information like the make, model, and year of the car to be appended to the end of the purchasedCars list by reverencing the addCar function.
—-

Functions:

—-
Lines used to create/close a function:

```
func greet(lang: String, name: String) -> String {
        (where you would put your  statements)
}
```

This line creates a function called greet that has two parameters called lang and name, which are strings. The "->" part of the function is used to say that you want to return a value as a string.
—-
Lines within the function that create return statements:

```
        if lang == "English" {
                return "Hello, \(name)"
        } else if lang == "Spanish"
                return "Hola, \(name)
```

The standard style of formatting the brackets is to have the "{" on the same line of the function definition and a "}" on a new line that is properly indented. "\()" allows you to insert variables into return/print strings.
—-
Calling a function and then printing it:

```
let message = greet(lang : "English", name : "Alice")
print(message)
```

This statement calls the greet function with the language "English" and the name "Alice". It then prints the message (which is a constant because it used "let" instead of "var").
—-

<u>Class vs Structures</u>
—
A class is a reference type that can be changed (is mutable) and allows for subclassing and overriding. A structure is an immutable reference type and all of the things inside of it cannot be changed. A class can be used for holding a person's contact number (because it changes) while a structure can be used for holding the make, model, and year of a car (because it does not change). This is an example of a class.

```
class Person{
        var name: String
        var contactNumber: String

        init(name: String, contactNumber: String) {
```

```
        self.name = name
        self.contactNumber = contactNumber
    }

    func contactInfo() -> String {
        return "Name: \(name), Contact Number: \(contactNumber)"
    }
}
```

The variables in this class are not locally defined. Because of this, we have to initialize them so that we can call on them outside of the class and define them. The initialization of variables is separated from the functions within the class.

—-

The line of code that is used to initialize an instance of a class:

```
let person1= Person(name: "Alice", contactNumber: "2545418035")
```

This line assigns the variable person1 with a name and contact number that can be referenced in the Person class.

—-

The line of code that is used to print from a specific function in a class:

```
print(person1.contactinfo)
```

Because we previously defined person1 with attributes in the Person class (using "let") we can now reference that class without stating it because it is already connected to the variable person1.

—-

What import Foundation and @propertyWrapper does:

—-

Always start a program with "import Foundation" *it will give you access to more tools.*

—-

A property wrapper is something that has all the logic used to manage a specific property. It can validate data, modify values before storing it, or add observers. This is an example of a property wrapper.

```
@propertyWrapper
struct ValidatePrice{
    var value: Double

    var wrappedValue: Double {
        get {value}
```

```
		set{ value = max(wrappedValue, 0)}
	}
	init(wrappedValue: Double) {
		self.value= max(wrappedValue,0)
	}
}
```