SWIFT PROGRAMMING

RESOURCES

- The basic concept is shown on the next slides.
- The official documentation can be found here: https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conce

VARIABLES AND TYPES I

```
//This is a variable of type Int
var x : Int;

//set a value. If we only have one command in our
//line we can omit the ending semicolon
x = 99

//here, we need a semicolon
x = 88; x = 99

print (x); //print "99" to the console
//The line above won't compile if you try to use a
//variable that is not initialized
```

VARIABLES AND TYPES II

```
//There are a number of other types available
var a : UInt = 1; //unsigned Integer
var b : Double = 0.5; //64 bit floating point
var c : Float = 1.5; //32 bit floating point
var d : Character = "a"; //1 character
var e : String = "abc"; //a string
var f : Bool = true; // a boolean value
//conversion
var j : Double = Double(a)
var k : String = String(c)
```

CONSTANTS

```
let π : Double = 3.14159; //let = const declaration
var pi : Double = 3.2 // var = variable declaration

//this works
pi = π;

//this does not work (compile error)
π = pi;
```

OPTIONALS I

```
//Optionals: Optionals can have a nil value. Optionals are defined by adding a
//question mark to the type name.
var a : Int?;

//Now a is nil as it is not defined

a = 9

//forced unpacking. This will crash if a is nil. (EXC_BAD_INSTRUCTION)
//Therefore, only use it if you are sure that the variable is not nil
print a!

//Only works if we force unpack (put the exclamation mark at the end)
var b : Int = a!
```

OPTIONALS II

```
//Conversion from optional to non-optional is always tricky
var a : Int?;
var b : Int;

//this won't work as a is optional
b = a

//this will work but crashes if a is nil
b = a!

//you can check for nil
if a != nil {b = a!}

//but the coalescing operator (??) is king as you can define a default.
//b will be a if a has a value or 0 if a is nil
var a = b ?? 0
```

EXERCISE

```
//wich lines won't compile?
a = 10
var b : Int = 10
let c : Int? = 10
var d : Double? = nil
let e : Double =

a = b, e = d
b = 20
b = b + c - 10 + 20 / 10
c = a
d = a
d = Double(c)
e = Double(b) ?? 0
```

ARRAYS I

```
//create an empty array
var integerArray = [Int]();

//add an element with append
integerArray.append(12345);

//get the length of an array
print (integerArray.count) //will print "1"

//add or retrieve a value by index
integerArray[0] = 12345

//when adding, make sure that there is space. This will crash
integerArray[1] = 12345

//this won't compile as it will always return an optional
```

ARRAYS II

```
//use auto initialization (3 values, all with 99.0)
var threeDoubles = [Double](count: 3, repeatedValue: 99.0)
threeDoubles
print (threeDoubles[2]) //prints 99.0

//create an array with explicit values
var integerArray : [Int] = [99, 8, 12, 102]

//append values
integerArray += [5, 232, 232]

//remove value
integerArray.removeAtIndex(1)

//the integerArray is now:
//[99, 12, 102, 5, 232, 232]
```

DICTIONARY I

```
//stores associations. Creates a dictionary
//with keys of type string and values of type int
var simple = [Int : String]()

//add a value
simple[1] = "a value";

//init with multiple values
var ratingOfArtists : [String : Int] =
        ["Elvis Presley" : 9, "Bob Marley" : 8, "Neil Diamond" : -1];

//count returns the number of items in the dictionary
print (ratingOfArtists.count) // prints 3
```

DICTIONARY II

```
//check if the dictionary is empty
print (ratingOfArtists.isEmpty()) //prints false

//delete an item
ratingOfArtists.removeValueForKey("Elvis Presley");
```

EXERCISE

```
//wich lines won't compile and which lines will yield a
//runtime exception (after all compile errors are solved)?
var ratingOfMovies : [String : Int] =
        ["Godfather I" : 9, "Mulholland Dr" : 10, "A Beautiful Mind" : 1]
var ratingArray = [Int]();

ratingArray.append(ratingOfMovies["Mulholland Dr"]);
ratingArray.append(ratingOfMovies[""])
ratingArray[2] = 4
var c = ratingArray[1]
ratingOfMovies["Drive"] = 8
ratingOfMovies["Drive"] = 7
print (ratingArray.count)
ratingOfMovies.removeAtIndex(4)
ratingArray.removeAtIndex(10)
```

TUPLES

```
//create the tuple
var errorNoSession = (-22, "User has no session")
var errorNoRegistration = (-1, "User not registered")

//retrieve the values of the tuple
let (errorCode, errorMessage) = errorNoSession


print (errorMessage) //prints "User has no session"
```

LOOPS I

```
//The for loop is straightforward
//(as in other program languages)
var counter = 0;
for (var i : Int = 0; i < 5;i++) {
        counter += i;
}

//you can use ranges (called for-in loop)
//note that the round brackets are optional
for index in 0...4 {
        counter += index;
}
```

LOOPS II

```
//for-in loops can also be applied to arrays or
//dictionaries
let myDictionary : [String : Int] = ["Hello" : 1, "World" : 2]
//for iterating over the dictionary, a tuple is returned
for (key,value) in myDictionary {
        print (key + " " + String(value)); //prints "Hello 1" /newline/ "World 2"
}

//you can iterate only over the keys or values
for key in myDictionary.keys ...
for value in myDictionary.values ...
```

LOOPS III

```
//There is also a while loop
var counter = 0;
var i = 10;
while i-- > 0 {
        counter += i;
}

//and a do-while loop
do {
        counter++;
} while (i-- > 0);
```

IF THEN ELSE

```
//straightforward (as in other program languages)
//the curly brackets are always required
if (a==0) {
        print ("a is null")
}

//the round brackets are optional
if b == 6 && c == 5 {
        print ("b is six and c is five");
}

//valid logical operators are
//NOT (!a)
//AND (a && b)
//OR (a || b)
```

EXERCISE

The program has 3 errors. Try to find them. What is printed at the end?

```
let message = "setec astronomy "

let permutation = [9, 10, 0, 13, 11, 3, 5, 15, 14, 12, 1, 6, 2, 4, 7, 8];

let result = [Character](count: message.characters.count, repeatedValue: "AA");

var count = 0;
for c in message.characters {
    result[permutation[count++]] = c;
}

for c in result
    print(c)
```

SWITCH

```
//you can use nearly all types (strings, tuples, )
switch (film) {
        case 1: print("Flew over the Cuckoo's nest");
        case 3: print("Couleurs");
        case 5: print ("Elements");
        case 6: print("Sense");
        case 7: print("Magnificent");
        case 8: print ("And a half");
        case 9: print("District");
        case 11...13: print ("Ocean's");
        case 123: print ("Taking of Pelham");
        case 300: print("This is sparta");
        //the default must always be set!
        default: print("I don't know");
}
```

EXERCISE

Create a switch statement similar to the example with three popular songs.

FUNCTIONS I

```
//functions are defined by the keyword func, the name of the function,
//the parameter list and a return value. Parameters are const by default
func printGoodMorningFor(name : String) {
        print("Good Morning " + name);
        //the following won't compile as name is const:
        name = "Hallo Welt"
}
func sum(a: Int, b: Int) -> Int {
        return a + b;
}
//call the function
printGoodMorningFor("Peter");
//if you have more than 1 parameter, you need to add their name when calling (b:)
var result = sum(5, b: 2);
```

FUNCTIONS II

```
//you can add default values
func printGoodMorningFor(name : String = "Mr X") {
        print("Good Morning " + name);
}
printGoodMorningFor();
//you can call a function with variable parameters (call by reference).
//These are called inout parameters
func changeName(inout name : String) {
        name = name + "!";
}
var name = "Peter"
//passing an inout parameter requires an ampersand
changeName(&name);
print (name); //prints "Peter!"
```

FUNCTIONS III

```
//each function has its own function type. The function sum has
//function type (Int, Int) -> Int
func plus(a: Int, b: Int) -> Int {
    return a + b;
}

//you can use functions as parameters via their function type.
//we expect a function with this type and call it with the two
//parameters a and b. Then we print out the result.
func printRes(f : ((Int, Int) -> Int), a : Int, b : Int) {
    print(f(a, b));
}

printRes(plus, a : 30, b: 20);
```

CLOSURES I

```
//A closure is a self contained block of code that you can pass around.
//A closure is wrapped in curly brackets. There is a function type
//for calling the closure first.
//{ (params) -> returnType in
//  statements
//}

//define a string array
let names = ["John", "Paul", "George", "Ringo"]

//The method "sort" accepts a closure as comparison function.
let sortedNames = names.sort(
    { (s1: String, s2: String) -> Bool in
        return s1 < s2
    }
)
```

CLOSURES II

```
//Closures can be shortened. You can omit the function type and
//use $i for accessing parameter i.
let sortedNames = names.sort(
    {
        return $0 < $1
    }
)
```

ENUMERATIONS

```
//Definition
enum Planet {
    case Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto
}

let p : Planet;
if (p == Planet.Pluto) {
        print("Sorry, not a planet anymore");
}
```

CLASSES

```
class Animal {
    var canFly: Bool // member variable
    var sound : String //another member variable

    //you can write multiple constructors with different parameters
    init(canFly : Bool, sound : String){
        self.canFly = canFly
        self.sound = sound
    }
    //the swift constructor is called init.
    //just make sure that you fully initialize all member variables
    init(){
        self.canFly = false
        self.sound = "";
    }
    func animalText() -> String {
```

CLASS INSTANTIATION

```
var sparrow = Animal(canFly: true, sound: "tweet tweet tweet");
let duck = Animal (canFly: true, sound: "quack")
let foo = Animal()
foo.sound = "foo fooo foooo"; //automatic setter

//class instances are passed by reference
sparrow = duck
sparrow.sound = "chirp chirp chirp"
print (duck.sound) //prints "chirp chirp chirp"
```

CLASS INHERITANCE

```
//of course, classes can use inheritance
class ServiceAnimal : Animal {
        var usedAs : String;
        init(usedAs: String, canFly : Bool, sound : String){
                self.usedAs = usedAs;
                super.init(canFly: canFly, sound: sound)
        }
        //the term "override" is mandatory
        override func animalText() -> String {
                return super.animalText() + " used as: " + self.usedAs;
        }
}

let dog = ServiceAnimal(usedAs: "guide", canFly: false, sound: "bark");
let monkey = ServiceAnimal(usedAs: "wrench", canFly: false, sound: "aaargh mooh");
```

EXERCISE

Try to add a member variable "numberOfLegs" to the Animal class.

CLASS UP- AND DOWNCASTING

```
//of course, classes can use inheritance
let duck = Animal (canFly: true, sound: "quack")
let dog = ServiceAnimal(usedAs: "guide", canFly: false, sound: "bark");

//upcast a ServiceAnimal to an animal
let animal : Animal = dog as Animal

//downcast an animal to a service animal
//downcasting either requires an exclamation point (forced downcast)
let serviceAnimalForced : ServiceAnimal = animal as! ServiceAnimal

//or on optional class that will be nil if the downcasting fails
let serviceAnimalOptional : ServiceAnimal? = animal as? ServiceAnimal
```

IS CLASS

```
//sometimes, you want to know the class of an object
//you can do this in swift with "is"
if dog is ServiceAnimal {} //true
if dog is Animal {} //true
if duck is ServiceAnimal {} //false
```

EXCEPTION HANDLING I

```
//Exception handling is similar to other program languages
//define errors
enum FileLineErrors : ErrorType {
        case NoFileFound
        case LineTooLong(lineLength: Int)
}
//mark a function that can throw an exception
func readFirstLineOfFile() throws -> String {

//and throw
if (file == nil) {
        throw FileLineErrors.NoFileFound
}
else if (lineLength > 80) {
        throw FileLineErrors.LineTooLong(lineLength: lineLength);
}}
```

EXCEPTION HANDLING II

```
//Now you can catch the exceptions
do {
        try readFirstLineOfFile();
}
catch (FileLineErrors.NoFileFound) {
        NSLog("No file found");
}
catch (FileLineErrors.LineTooLong(let lineLength)) {
        NSLog("Your line has " + String(lineLength) +
                " characters, but should only have 80.");
}
```

EXCEPTION HANDLING III

The protocol ErrorType is somewhat special. See:
https://realm.io/news/testing-swift-error-type/

EXERCISE

Write a function "gausseanSumForN" that accepts a positive integer n and returns the sum 1+2+...+n. If n<=0, an exception should be thrown.