Swift 学习笔记

1. 基础部分
2. swift 使用let作为关键字来声明常亮，用var作为关键字来声明变量
3. 有一种更简单的把值转换成字符串的方法:把值写到括号中，并且在括号之前写一个反斜杠。
4. switch 支持任意类型的数据以及各种比较操作——不仅仅是整数以及测试相等 运行 switch 中匹配到的子句之后，程序会退出 switch 语句，并不会继续向下运行，所以不 需要在每个子句结尾写 break。
5. 你可以在循环中使用..来表示范围，也可以使用传统的写法，两者是等价的
6. 类的每个属性都需要赋值不管是构造的时候还是声明的时候，也就是说构造的时候可以通过构造器（init方法）或者声明属性的时候都需要来初始化属性的值
7. 子类的构造器 引用父类的构造器的时候一定要先把自己子类中的新的属性先复制 然后再调用父类的构造器方法 不然的话会报错的
8. 你可以用任何你喜欢的字符作为常量和变量名，包括 Unicode 字符，但是不能包含数学符号、箭头或者非法的Unicode符号，并且不能以数字开头
9. 常量一旦被确定了就不能被更改了
10. print(\_:separator:terminator:)是一个用来输出一个或多个值到适当输出区的全局函数。如果你用 Xcode，print(\_:separator:terminator:)将会输出内容到“console”面板上。separator和terminator参数具有默认值，因此你调用这个函数的时候可以忽略它们。默认情况下，该函数通过添加换行符来结束当前行。如果不想换行，可以传递一个空字符串给terminator参数--例如，print(someValue, terminator:"")。关于参数默认值的更多信息，请参考[默认参数值](http://wiki.jikexueyuan.com/project/swift/chapter2/06_Functions.html#default_parameter_values)。
11. Swift 用字符串插值（string interpolation）的方式把常量名或者变量名当做占位符加入到长字符串中，Swift 会用当前常量或变量的值替换这些占位符。将常量或变量名放入圆括号中，并在开括号前使用反斜杠将其转义
12. 与其他大部分编程语言不同，Swift 并不强制要求你在每条语句的结尾处使用分号（;），当然，你也可以按照你自己的习惯添加分号。有一种情况下必须要用分号，即你打算在同一行内写多条独立的语句：
13. swift也不需要指定int类型的所占空间的位数，可以用通用的int 关键字来表示 这样系统就会在相应 的编译环境中选择相应的大小的空间来存储，但是int是一种有符号的整形变量，无符号的整形变量用Uint，但是一般情况下最好不要用Uint
14. Swift 是一个类型安全（type safe）的语言。类型安全的语言可以让你清楚地知道代码要处理的值的类型。如果你的代码需要一个String，你绝对不可能不小心传进去一个Int。
15. 当你要处理不同类型的值时，类型检查可以帮你避免错误。然而，这并不是说你每次声明常量和变量的时候都需要显式指定类型。如果你没有显式指定类型，Swift 会使用*类型推断*（type inference）来选择合适的类型。有了类型推断，编译器可以在编译代码的时候自动推断出表达式的类型。原理很简单，只要检查你赋的值即可。
16. 当你在声明常量或者变量的时候赋给它们一个字面量（literal value 或 literal）即可触发类型推断。（字面量就是会直接出现在你代码中的值，比如42和3.14159。）
17. 浮点字面量可以是十进制（没有前缀）或者是十六进制（前缀是0x）
18. 小数点两边必须有至少一个十进制数字（或者是十六进制的数字）。浮点字面量还有一个可选的指数（exponent，在十进制浮点数中通过大写或者小写的e来指定，在十六进制浮点数中通过大写或者小写的p来指定。
19. 数值类字面量可以包括额外的格式来增强可读性。整数和浮点数都可以添加额外的零并且包含下划线，并不会影响字面量
20. 用变量类型（变量名称）来强转变量类型，但是也是不能随意强转，当将int类型强转成double类型的时候，就直接就能转换成相同的数值，但是如果将double类型强转成int类型的化就会直接截取整形部分。
21. 你可以使用typealias关键字来定义类型别名
22. swift的BOOL类型只有true和false这两个值 ，1和0是不行的
23. 元组（tuples）把多个值组合成一个复合值。元组内的值可以是任意类型，并不要求是相同类型。
24. 定义元组：let http404Error = (404, "Not Found")
25. 将一个元组分解成几部分：let (statusCode, statusMessage) = http404Error
26. 只需要一个元组的一部分数据let (justTheStatusCode, \_) = http404Error
27. 此外，你还可以通过下标来访问元组中的单个元素，下标从零开始：http404Error.0
28. 可以在定义元组的时候直接给元素定义名字：let http200Status = (statusCode: 200, description: "OK") ，然后可以直接使用点语法来去的相应 的值
29. 使用可选类型（optionals）来处理值可能缺失的情况。可选类型表示：有值，等于X或者没有值
30. 可选类型（optional）Int，而不是一个Int。一个可选的Int被写作Int?而不是Int。问号暗示包含的值是可选类型，也就是说可能包含Int值也可能不包含值。（不能包含其他任何值比如Bool值或者String值。只能是Int或者什么都没有。）
31. C 和 Objective-C 中并没有可选类型这个概念。最接近的是 Objective-C 中的一个特性，一个方法要不返回一个对象要不返回nil，nil表示“缺少一个合法的对象”。然而，这只对对象起作用——对于结构体，基本的 C 类型或者枚举类型不起作用。对于这些类型，Objective-C 方法一般会返回一个特殊值（比如NSNotFound）来暗示值缺失。这种方法假设方法的调用者知道并记得对特殊值进行判断。然而，Swift 的可选类型可以让你暗示\_任意类型\_的值缺失，并不需要一个特殊值。一般情况下可选类型一般都是用在可能会失败的操作中，用来标识这个操作可能会失败，但并不是设计或者语法的错误造成的，这种错误是在允许范围内的
32. nil不能用于非可选的常量和变量。如果你的代码中有常量或者变量需要处理值缺失的情况，请把它们声明成对应的可选类型。
33. 如果你申明一个常亮或者变量但是并没有给他赋值，那么这个变量或者常量可能会被赋值成nil，但是Swift 的nil和 Objective-C 中的nil并不一样。在 Objective-C 中，nil是一个指向不存在对象的指针。在 Swift 中，nil不是指针——它是一个确定的值，用来表示值缺失。任何类型的可选状态都可以被设置为nil，不只是对象类型。
34. 当你确定可选类型确实包含值之后，你可以在可选的名字后面加一个感叹号（!）来获取值。这个惊叹号表示“我知道这个可选有值，请使用它。”这被称为可选值的强制解析（forced unwrapping）：print("convertedNumber has an integer value of \(convertedNumber!).")
35. 使用!来获取一个不存在的可选值会导致运行时错误。使用!来强制解析值之前，一定要确定可选包含一个非nil的值。
36. 当你确定可选类型确实包含值之后，你可以在可选的名字后面加一个感叹号（!）来获取值。这个惊叹号表示“我知道这个可选有值，请使用它。”这被称为可选值的强制解析（forced unwrapping）：
37. 有时候在程序架构中，第一次被赋值之后，可以确定一个可选类型\_总会\_有值。在这种情况下，每次都要判断和解析可选值是非常低效的，因为可以确定它总会有值。这种类型的可选状态被定义为隐式解析可选类型（implicitly unwrapped optionals）
38. 把想要用作可选的类型的后面的问号（String?）改成感叹号（String!）来声明一个隐式解析可选类型
39. **let** possibleString: String? = "An optional string."
40. **let** forcedString: String = possibleString! *// 需要惊叹号来获取值*
41. **let** assumedString: String! = "An implicitly unwrapped optional string."
42. **let** implicitString: String = assumedString *// 不需要感叹号*
43. 如果你在隐式解析可选类型没有值的时候尝试取值，会触发运行时错误。和你在没有值的普通可选类型后面加一个惊叹号一样。
44. 如果一个变量之后可能变成nil的话请不要使用隐式解析可选类型。如果你需要在变量的生命周期中判断是否是nil的话，请使用普通可选类型。
45. 加法运算符也可用于String的拼接：
46. "hello, " + "world" // 等于 "hello, world"
47. 在对负数b求余时，b的符号会被忽略。这意味着 a % b 和 a % -b的结果是相同的。不同于 C 语言和 Objective-C，Swift 中是可以对浮点数进行求余的。
48. Swift 也提供恒等===和不恒等!==这两个比较符来判断两个对象是否引用同一个对象实例
49. 空合运算符(a ?? b)将对可选类型a进行空判断，如果a包含一个值就进行解封，否则就返回一个默认值b.这个运算符有两个条件:表达式a必须是Optional类型 默认值b的类型必须要和a存储值的类型保持一致 如果a为非空值(non-nil),那么值b将不会被估值。这也就是所谓的短路求值。
50. 短路运算就是比如在逻辑与运算的时候，如果直接判断第一个数就是false的话就直接不计算第二个值就能知道整个运算的值是false