**数据类型**

1. 分类(2大类)  
 **\* 基本(值)类型**  
 \* Number: 任意数值  
 \* String: 任意文本  
 \* Boolean: true/false  
 \* undefined: undefined  
 \* null: null  
 **\* 对象(引用)类型**  
 \* Object: 任意对象  
 \* Array: 特别的对象类型(下标/内部数据有序)  
 \* Function: 特别的对象类型(可执行)  
2. 判断  
  **\* typeof:**  
 \* 可以区别: 数值, 字符串, 布尔值, undefined, function  
 \* 不能区别: null与对象, 一般对象与数组返回值都是object  
  **\* instanceof**  
 \* 专门用来判断对象数据的类型: Object, Array与Function  
  **\* ===**  
 \* 可以判断: undefined和null  
-->

**1. undefined与null的区别?** \* undefined代表没有赋值  
 \* null代表赋值了, 只是值为null  
**2. 什么时候给变量赋值为null呢?** \* var a = null //a将指向一个对象, 但对象此时还没有确定  
 \* a = null //让a指向的对象成为垃圾对象  
**3. 严格区别变量类型与数据类型?** \* js的变量本身是没有类型的, 变量的类型实际上是变量内存 中数据的类型  
  **\* 变量类型:**  
 \* 基本类型: 保存基本类型数据的变量  
 \* 引用类型: 保存对象地址值的变量  
  **\* 数据对象**  
 \* 基本类型  
 \* 对象类型

**数据**

**什么是数据?**  
 \* 存储于内存中代表特定信息的'东东', 本质就是0101二进制  
 \* 具有可读和可传递的基本特性  
 \* 万物(一切)皆数据, 函数也是数据  
 \* 程序中所有操作的目标: 数据  
 \* 算术运算  
 \* 逻辑运算  
 \* 赋值  
 \* 调用函数传参  
 ...

**内存**  
2. 什么是内存?  
 \* 内存条通电后产生的存储空间(临时的)  
 \* 产生和死亡: 内存条(集成电路板)==>通电==>产生一定容量的存储空间==>存储各种数据==>断电==>内存全部消失  
 \* 内存的空间是临时的, 而硬盘的空间是持久的  
 \* 一块内存包含2个数据  
 \* 内部存储的数据(一般数据/地址数据)  
 \* 内存地址值数据  
  **\* 内存分类**  
 \* 栈: 全局变量, 局部变量 (空间较小)  
 \* 堆: 对象 (空间较大)  
**3. 什么是变量?**  
 \* 值可以变化的量, 由**变量名与变量值组成**  
 \* 一个变量对应一块小内存, 变量名用来查找到内存, 变量值就是内存中保存的内容  
**4. 内存,数据, 变量三者之间的关系**  
 \* 内存是一个容器, 用来存储程序运行需要操作的数据  
 \* 变量名是内存的标识, 我们通过变量找到对应的内存, 进而操作(读/写)内存中的数据

**问题: var a = xxx, a内存中到底保存的是什么?**  
\* xxx是一个基本数据  
\* xxx是一个对象  
\* xxx是一个变量

**关于引用变量赋值问题**  
 \* 2个引用变量指向同一个对象, 通过一个引用变量修改对象内部数据, 另一个引用变量也看得见  
 \* 2个引用变量指向同一个对象,让一个引用变量指向另一个对象, 另一个引用变量还是指向原来的对象

问题: 在js调用函数时传递变量参数时, 是值传递还是引用传递  
 \* 只有**值传递**, 没有引用传递, 传递的都是变量的值, 只是这个值可能是基本数据, 也可能是地址(引用)数据  
 \* 如果后一种看成是引用传递, 那就值传递和引用传递都可以有

**JS引擎如何管理内存?**  
 1. 内存生命周期  
 1). 分配需要的内存  
 2). 使用分配到的内存  
 3). 不需要时将其释放/归还  
 2. 释放内存  
 \* 为执行函数分配的栈空间内存: 函数执行完自动释放  
 \* 存储对象的堆空间内存: 当内存没有引用指向时, 对象成为垃圾对象, 垃圾回收器后面就会回收释放此内存

**对象**

1. **什么是对象?**  
 \* 代表现实中的某个事物, 是该事物在编程中的抽象  
 \* 多个数据的集合体(封装体)  
 \* 用于保存多个数据的容器  
**2. 为什么要用对象?**  
 \* 便于对多个数据进行统一管理  
**3. 对象的组成**  
 \* 属性  
 \* 代表现实事物的状态数据  
 \* 由属性名和属性值组成  
 \* 属性名都是字符串类型, 属性值是任意类型  
 \* 方法  
 \* 代表现实事物的行为数据  
 \* 是特别的属性==>属性值是函数

**4. 如何访问对象内部数据**?  
 \* .属性名: 编码简单, 但有时不能用  
 \* ['属性名']: 编码麻烦, 但通用

**问题: 什么时候必须使用['属性名']的方式?**  
 \* 属性名不是合法的标识名  
 \* 属性名不确定

**函数**

**1. 什么是函数?**  
 \* 具有特定功能的n条语句的封装体  
 \* 只有函数是可执行的, 其它类型的数据是不可执行的  
 \* 函数也是对象  
**2. 为什么要用函数?**  
 \* 提高代码复用  
 \* 便于阅读和交流  
**3. 如何定义函数?** \* 函数声明  
 \* 表达式  
**4. 如何调用(执行)函数? 不同调用方法函数中的this不同** \* test(): 直接调用 window  
 \* new test(): 通过new 调用 new的对象  
 \* obj.test(): 通过对象调用 obj  
 \* test.call/apply(obj): 通过函数对象的call/apply()调用 obj

**1. 什么函数才是回调函数?** \* 你定义的  
 \* 你没有直接调用  
 \* 但最终它执行了(在特定条件或时刻)

**2. 常见的回调函数?** \* DOM事件函数  
 \* 定时器函数  
 \* ajax回调函数(后面学)  
 \* 生命周期回调函数(后面学)

**1. 理解**  
 \* 全称: Immediately-Invoked Function Expression 立即调用函数表达式  
 \* 别名: 匿名函数自调用 / 自调用匿名函数  
**2. 作用**  
 \* 隐藏内部实现  
 \* 不污染外部命名空间

**This**

**1. this是什么?**  
 \* 一个关键字, 一个内置的引用变量  
 \* 在函数中都可以直接使用this  
 \* this代表调用函数的当前对象  
 \* 在定义函数时, this还没有确定, 只有在执行时才动态确定(绑定)的  
**2. 如何确定this的值?**  
 \* test() : window  
 \* obj.test(): obj  
 \* new test(): new的对象  
 \* test.call(obj) test.apply(obj) : obj  
前置知识:  
 \* 本质上任何函数在执行时都是通过某个对象调用的