Day-2

- Video-045-算数运算符
  - // 取整除
  - % 取余数
  - ** 幂
  - * 可以用于字符串，计算结果就是字符串重复指定次数的结果
  - 优先级
- Video-046-052程序执行原理
  - 三大件
    - 1.CPU：中央处理器，超大规模的集成电路，**负责处理数据/计算**
    - 2.内存：临时 存储数据，**速度快**，空间小
    - 3.硬盘：永久 存储数据，**速度慢**，空间大
  - 程序安装在硬盘中
  - 执行原理
    - 运行程序时，操作系统会**首先让CPU把程序复制到内存**中，CPU执行内存中的程序代码

    - Python程序执行：CPU把Python解释器的程序复制到内存中，Python解释器从上向下让CPU翻译Python程序中的代码，CPU负责执行翻译完成的代码

    - 查看Python解释器的大小

```
# 1. 确认解释器所在位置
$ which python

# 2. 查看 python 文件大小(只是一个软链接)
$ ls -lh /usr/bin/python

# 3. 查看具体文件大小
$ ls -lh /usr/bin/python2.7
```

> 提示：建立 **软链接** 的目的，是为了方便使用者不用记住使用的解释器是 **哪一个具体版本**

    - 程序用来处理数据，变量用来存储数据
      - 变量：程序内部，在内部中分配的空间

- Video-053-080变量的使用
  - 1.定义：变量赋值后才会被创建
    - 变量名只有第一次出现才是定义变量，后面出现是直接使用定义过的变量
  - 2.类型
    - 创建的变量：1.变量的名称；2.变量保存的数据；3.变量存储数据的类型；4.变量的地址（标示）
    - 定义变量时不需要指定类型
    - type函数可以查看一个变量的类型
    - 数字型
      - int：整型，long：长整型
        - Python2.x区分int和long
      - float：浮点型
      - bool：真假（非0即真）
      - complex：复数型
    - 非数字型
      - str：字符串
      - 列表
      - 元组
      - 字典
  - 3.不同类型变量之间的计算
    - 1.数字型变量之间可以直接计算
      - bool型，True -> 1 False -> 0
    - 2.字符串变量之间使用 + 拼接字符
    - 3.使用 字符 * 次数
  - 4.变量的输入：用代码获取用户通过键盘输入的信息
    - input函数实现键盘输入
  - 5.类型转换函数
    - int(x)：将变量转换成整数
    - float(x)：将变量转换成浮点数
  - 6.变量的格式化输出
    - 包含%的字符串，被称为格式化字符串
    - %和不同的字符连用，不同类型的数据需要使用不同的格式化字符

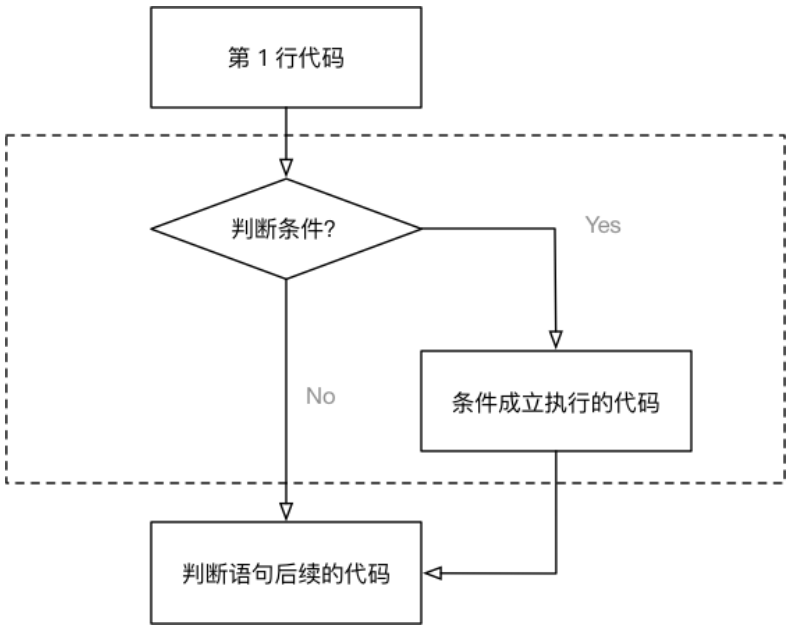
格式化字符	含义
%s	字符串
%d	有符号十进制整数，`%06d` 表示输出的整数显示位数，不足的地方使用 `0` 补全
%f	浮点数，`%.2f` 表示小数点后只显示两位
%%	输出 `%`

- 语法
  - print("格式化字符串" % 变量1)
  - print("格式化字符串" % (变量1, 变量2...))
- 7.变量的命名
  - 标识符：程序员定义的变量名、函数名，区分大小写
    - 可以由**字母、下划线和数字**组成
    - 不能以数字开头
    - 不能与关键字重名
  - 关键字：Python内部已经使用的标识符
    - 具有特殊的功能和含义
    - 不允许定义和关键字相同的名字的标识符
    - 查看Python中的关键字：import keyword print(keyword.kwlist)
  - 命名规则
    - 每个单词都是用小写字母
    - 单词与单词之间使用_下划线连接

- Video-081-107判断if语句
  - 基本语法

```
if 要判断的条件：
    条件成立时，要做的事情
    ……
```

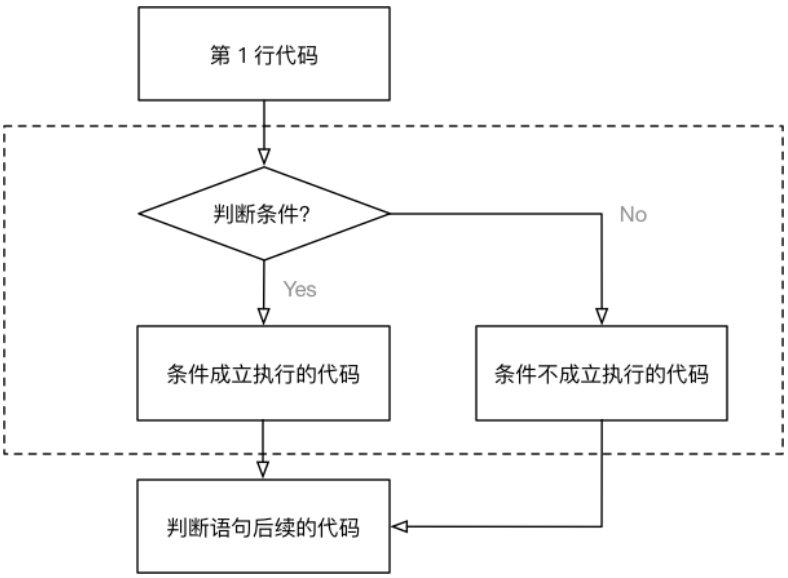
    - if 语句以及缩进部分是一个 **完整的代码块**



    - else处理条件不满足的情况：if 和 else 语句以及各自的缩进部分共同是一个 **完整的代码块**

`else`，格式如下：

```
if 要判断的条件：
    条件成立时，要做的事情
    ……
else:
    条件不成立时，要做的事情
    ……
```



- Python开发中，Tab和空格不要混用
- 比较运算符

运算符	描述
==	检查两个操作数的值是否 **相等**，如果是，则条件成立，返回 True
!=	检查两个操作数的值是否 **不相等**，如果是，则条件成立，返回 True
>	检查左操作数的值是否 **大于** 右操作数的值，如果是，则条件成立，返回 True
<	检查左操作数的值是否 **小于** 右操作数的值，如果是，则条件成立，返回 True
>=	检查左操作数的值是否 **大于或等于** 右操作数的值，如果是，则条件成立，返回 True
<=	检查左操作数的值是否 **小于或等于** 右操作数的值，如果是，则条件成立，返回 True

> Python 2.x 中判断 **不等于** 还可以使用 `<>` 运算符
>
> `!=` 在 Python 2.x 中同样可以用来判断 **不等于**

- 逻辑运算：可以把多个条件按照逻辑进行连接，变成更复杂的条件
  - 1.and：必须两个条件同时成立
  - 2.or：两个条件只要有一个成立
  - 3.not：非，不是，**取非**
    - 在开发中，通常希望某个条件不满足时，执行一些代码，可以使用not
    - 如果需要拼接复杂的逻辑计算条件，同样也有可能使用到not
- if语句进阶
  - elif：再增加一些不同的条件；**条件不同，需要执行的代码也不同；**所有条件是平级的

```
if 条件1:
    条件1满足执行的代码
    ……
elif 条件2:
    条件2满足时，执行的代码
    ……
elif 条件3:
    条件3满足时，执行的代码
    ……
else:
    以上条件都不满足时，执行的代码
    ……
```

- elif 和 else 都必须和 if 联合使用，而不能单独使用
- 可以将 if、elif 和 else 以及各自缩进的代码，看成一个 **完整的代码块**

- if嵌套：在之前条件满足的前提下，再增加额外的判断
  - 语法格式

```
if 条件 1:
    条件 1 满足执行的代码
    ……

    if 条件 1 基础上的条件 2:
        条件 2 满足时，执行的代码
        ……

    # 条件 2 不满足的处理
    else:
        条件 2 不满足时，执行的代码

# 条件 1 不满足的处理
else:
    条件1 不满足时，执行的代码
    ……
```

- random 工具包
  - 导入模块后，可以直接在模块名称后面敲一个 . 然后按Tab键，会提示该模块中包含的所有函数

以上内容整理于 幕布文档