Table of Contents

软件测试**Python**课程

本阶段课程不仅可以帮助我们进入Python语言世界，同时也是后续UI自动化测试、接口自动化测试等课程阶段的语言基础。

> Life is short, you need Python! -- 人生苦短，我用Python！

课程大纲

序号	章节	知识点
1	Python基础	1. 认识Python 2. Python环境搭建 3. PyCharm 4. 注释、变量、变量类型、输入输出、运算符
2	流程控制结构	1. 判断语句 2. 循环
3	数据序列	1. 字符串 2. 列表 3. 元组 4. 字典
4	函数	1. 函数基础 2. 变量进阶 3. 函数进阶 4. 匿名函数
5	面向对象	1. 面向对象编程介绍 2. 类和对象 3. 面向对象基础语法 4. 封装、继承、多态 5. 类属性和类方法
6	异常、模块、文件操作	1. 异常 2. 模块和包 3. 文件操作
7	UnitTest框架	1. UnitTest基本使用 2. UnitTest断言 3. 参数化 4. 生成HTML测试报告

课程目标

1. 掌握如何搭建Python开发环境；
2. 掌握Python基础语法, 具备基础的编程能力；
3. 建立编程思维以及面向对象程序设计思想；
4. 掌握如何通过UnitTest编写测试脚本，并生成HTML测试报告。

流程控制结构

目标

1. 知道判断语句在开发中的应用场景
2. 掌握`if`判断语句的语法
3. 能够使用`if`判断进行综合应用
4. 知道程序的三大流程
5. 掌握`while`循环的基本使用
6. 掌握`break`和`continue`的用法
7. 掌握`for`循环

判断语句

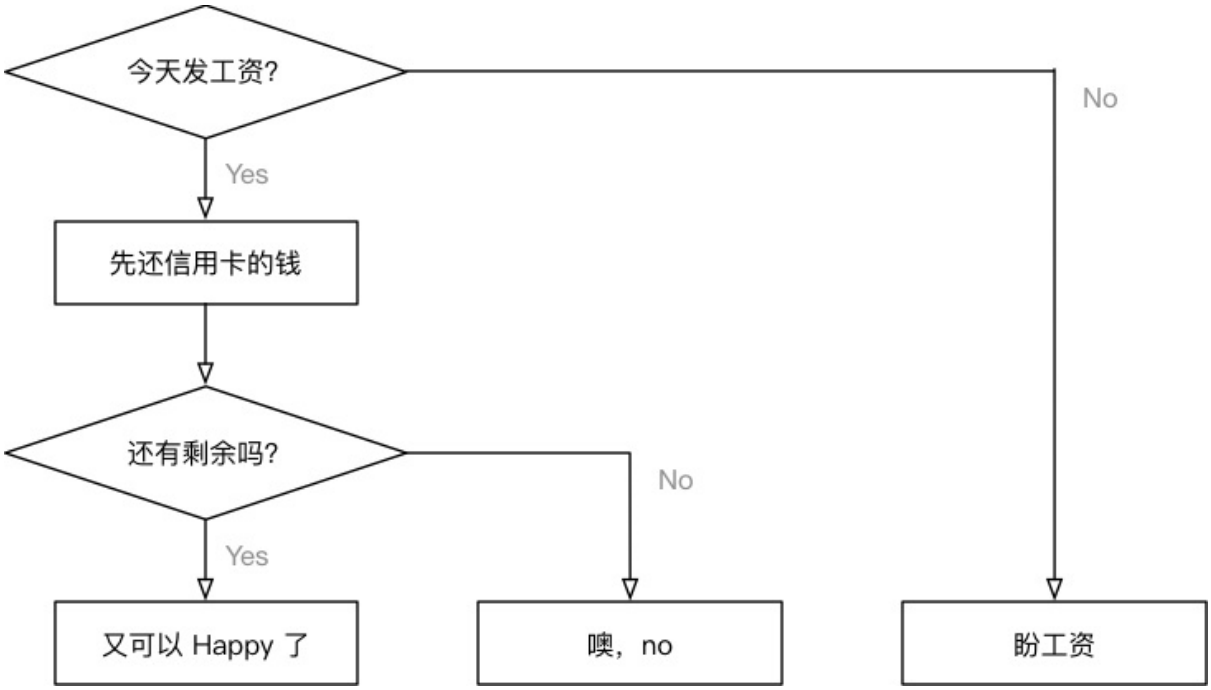
目标

1. 知道判断语句在开发中的应用场景
2. 掌握`if`判断语句的语法
3. 能够使用`if`判断进行综合应用

1. 开发中的应用场景

生活中的判断几乎是无所不在的，我们每天都在做各种各样的选择，如果这样？如果那样？……

1.1 程序中的判断




```
if 今天发工资:
    先还信用卡的钱
    if 有剩余:
        又可以happy了，O(n_n)O哈哈~
    else:
        噢，no。。。还的等30天
else:
    盼着发工资
```

1.2 判断的定义

- 如果 条件满足，才能做某件事情，
- 如果 条件不满足，就做另外一件事情，或者什么也不做

> 正是因为有了判断，才使得程序世界丰富多彩，充满变化！
>
> 判断语句 又被称为“分支语句”，正是因为有了判断，才让程序有了很多的分支

2. if语句体验

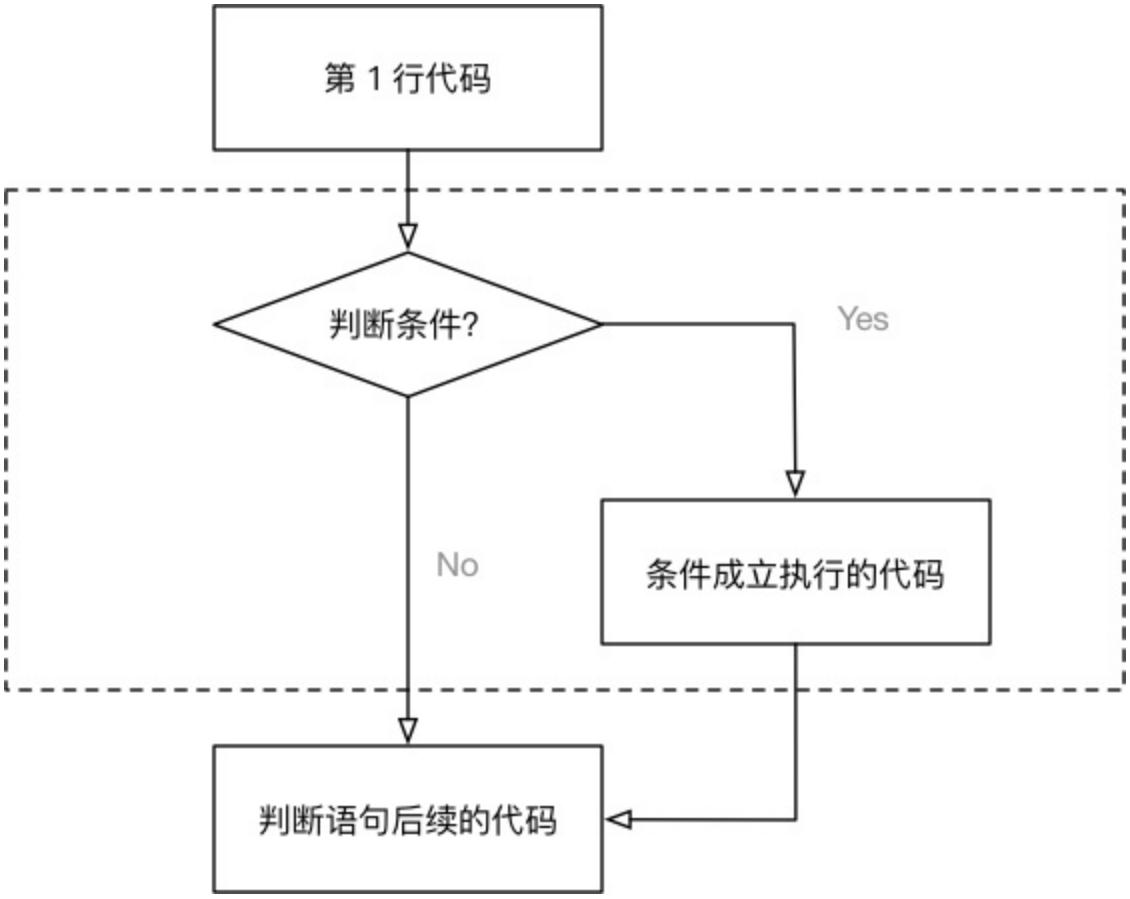
2.1 if判断语句基本语法

在Python中，**if** 语句 就是用来进行判断的，格式如下：

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

> 注意：代码的缩进为一个 `tab` 键，或者 **4** 个空格 —— 建议使用空格
>
> - 在 Python 开发中，Tab 和空格不要混用！

我们可以把整个 **if** 语句看成一个完整的代码块



2.2 判断语句演练 —— 判断年龄

需求

1. 定义一个整数变量记录年龄
2. 判断是否满 18 岁 （**>=**）
3. 如果满 18 岁，允许进网吧嗨皮

```
# 1. 定义年龄变量
age = 18

# 2. 判断是否满 18 岁
# if 语句以及缩进部分的代码是一个完整的代码块
if age >= 18:
    print("可以进网吧嗨皮……")

# 3. 思考! - 无论条件是否满足都会执行
print("这句代码什么时候执行?")
```

注意：

- `if` 语句以及缩进部分是一个 完整的代码块

2.3 else 处理条件不满足的情况

思考

在使用 `if` 判断时，只能做到满足条件时要做的事情。那如果需要在 不满足条件的时候，做某些事情，该如何做呢？

答案

`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
    ……
```

注意：

- `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

2.4 判断语句演练 —— 判断年龄改进

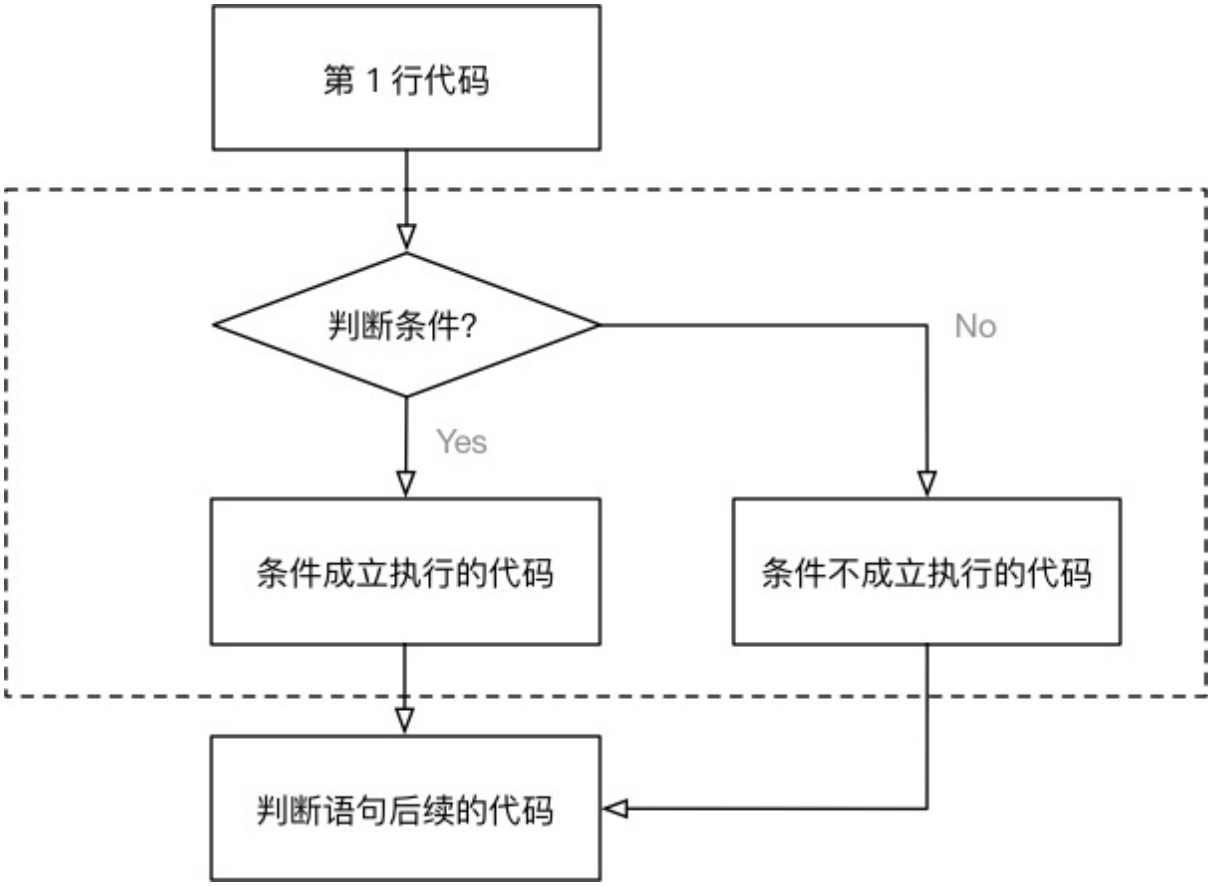
需求

1. 输入用户年龄
2. 判断是否满 18 岁 （**>=**）
3. 如果满 18 岁，允许进网吧嗨皮
4. 如果未满 18 岁，提示回家写作业

```
# 1. 输入用户年龄
age = int(input("今年多大了？"))

# 2. 判断是否满 18 岁
# if 语句以及缩进部分的代码是一个完整的语法块
if age >= 18:
    print("可以进网吧嗨皮……")
else:
    print("你还没长大，应该回家写作业！")

# 3. 思考！- 无论条件是否满足都会执行
print("这句代码什么时候执行?")
```



3. 逻辑运算

- 在程序开发中，通常 在判断条件时，会需要同时判断多个条件
- 只有多个条件都满足，才能够执行后续代码，这个时候需要使用到 逻辑运算符
- 逻辑运算符 可以把 多个条件 按照 逻辑 进行 连接，变成 更复杂的条件
- Python 中的 逻辑运算符 包括：与 **and** / 或 **or** / 非 **not** 三种

3.1 and

```
条件1 and 条件2
```

- 与 / 并且
- 两个条件同时满足，返回 `True`
- 只要有一个不满足，就返回 `False`

条件 **1**	条件 **2**	结果
成立	成立	成立
成立	不成立	不成立
不成立	成立	不成立
不成立	不成立	不成立

3.2 or

```
条件1 or 条件2
```

- 或 / 或者
- 两个条件只要有一个满足，返回 `True`
- 两个条件都不满足，返回 `False`

条件 **1**	条件 **2**	结果
成立	成立	成立
成立	不成立	成立
不成立	成立	成立
不成立	不成立	不成立

3.3 not

```
not 条件
```

- 非 / 不是

条件	结果
成立	不成立
不成立	成立

逻辑运算演练

1. 练习1: 定义一个整数变量 `age`，编写代码判断年龄是否正确
   - 要求人的年龄在 0-120 之间
2. 练习2: 定义两个整数变量 `python_score`、`c_score`，编写代码判断成绩
   - 要求只要有一门成绩 > 60 分就算合格
3. 练习3: 定义一个布尔型变量 `is_employee`，编写代码判断是否是本公司员工
   - 如果不是提示不允许入内

答案 1：

```
# 练习1: 定义一个整数变量 age，编写代码判断年龄是否正确
age = 100

# 要求人的年龄在 0-120 之间
if age >= 0 and age <= 120:
    print("年龄正确")
else:
    print("年龄不正确")
```

答案 2：

```
# 练习2: 定义两个整数变量 python_score、c_score，编写代码判断成绩
python_score = 50
```

```
c_score = 50

# 要求只要有一门成绩 > 60 分就算合格
if python_score > 60 or c_score > 60:
    print("考试通过")
else:
    print("再接再厉！")
```

答案 3：

```
# 练习3: 定义一个布尔型变量 `is_employee`，编写代码判断是否是本公司员工
is_employee = True

# 如果不是提示不允许入内
if not is_employee:
    print("非公勿内")
```

4. if 语句进阶

4.1 `elif` 多重判断

- 在开发中，使用 `if` 可以 判断条件
- 使用 `else` 可以处理 条件不成立 的情况
- 但是，如果希望 再增加一些条件，条件不同，需要执行的代码也不同 时，就可以使用 `elif`
- 语法格式如下：

```
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
    ……
```

- 对比逻辑运算符的代码

```
if 条件1 and 条件2:
    条件1满足 并且 条件2满足 执行的代码
    ……
```

注意

1. `elif` 和 `else` 都必须和 `if` 联合使用，而不能单独使用
2. 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

elif 演练 —— 根据考试成绩,进行分级

需求

1. 定义 `score` 变量记录考试分数

2. 如果分数是 大于等于 90分 应该显示 优
3. 如果分数是 大于等于 80分 并且 小于 90分 应该显示 良
4. 如果分数是 大于等于 70分 并且 小于 80分 应该显示 中
5. 如果分数是 大于等于 60分 并且 小于 70分 应该显示 差
6. 其它分数显示 不及格

```
score = 49

if score >= 90:
    print("优")
elif score >= 80 and score < 90:
    print("良")
elif score >= 70 and score < 80:
    print("中")
elif score >= 60 and score < 70:
    print("差")
else:
    print("不及格")
```

> 拓展: score >= 80 and score < 90 可以简化为 80<= score < 90

4.2 `if` 的嵌套

> **elif** 的应用场景是：同时 判断 多个条件，所有的条件是 平级 的

- 在开发中，使用 `if` 进行条件判断，如果希望 在条件成立的执行语句中 再 增加条件判断，就可以使用 **if** 的嵌套
- **if** 的嵌套 的应用场景就是：在之前条件满足的前提下，再增加额外的判断
- **if** 的嵌套 的语法格式，除了缩进之外 和之前的没有区别
- 语法格式如下：

```
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
    ……
```

if 的嵌套 演练 —— 火车站安检

需求

1. 定义布尔型变量 `has_ticket` 表示是否有车票
2. 定义整型变量 `knife_length` 表示刀的长度，单位：厘米
3. 首先检查是否有车票，如果有，才允许进行 安检
4. 安检时，需要检查刀的长度，判断是否超过 20 厘米
    - 如果超过 20 厘米，提示刀的长度，不允许上车
    - 如果不超过 20 厘米，安检通过
5. 如果没有车票，不允许进门

```
# 定义布尔型变量 has_ticket 表示是否有车票
has_ticket = True

# 定义整数型变量 knife_length 表示刀的长度，单位：厘米
knife_length = 20

# 首先检查是否有车票，如果有，才允许进行 安检
if has_ticket:
    print("有车票，可以开始安检...")

    # 安检时，需要检查刀的长度，判断是否超过 20 厘米
    # 如果超过 20 厘米，提示刀的长度，不允许上车
    if knife_length >= 20:
        print("不允许携带 %d 厘米长的刀上车" % knife_length)
    # 如果不超过 20 厘米，安检通过
    else:
        print("安检通过，祝您旅途愉快……")

# 如果没有车票，不允许进门
else:
    print("大哥，您要先买票啊")
```

5. 综合应用 —— 石头剪刀布

目标

1. 强化 多个条件 的 逻辑运算
2. 体会 `import` 导入模块（“工具包”）的使用

需求

1. 从控制台输入要出的拳 —— 石头（1） / 剪刀（2） / 布（3）
2. 电脑 随机 出拳 —— 先假定电脑只会出石头，完成整体代码功能
3. 比较胜负

序号	规则
1	石头 胜 剪刀
2	剪刀 胜 布
3	布 胜 石头

5.1 基础代码实现

- 先 假定电脑就只会出石头，完成整体代码功能

```
# 从控制台输入要出的拳 —— 石头（1） / 剪刀（2） / 布（3）
player = int(input("请出拳 石头（1） / 剪刀（2） / 布（3）： "))

# 电脑 随机 出拳 - 假定电脑永远出石头
computer = 1

# 比较胜负
# 如果条件判断的内容太长，可以在最外侧的条件增加一对大括号
# 再在每一个条件之间，使用回车，PyCharm 可以自动增加 8 个空格
if ((player == 1 and computer == 2) or
        (player == 2 and computer == 3) or
        (player == 3 and computer == 1)):

    print("噢耶！！！电脑弱爆了！！！")
elif player == computer:
    print("心有灵犀，再来一盘！")
else:
    print("不行，我要和你决战到天亮！")
```

5.2 随机数的处理

- 在 `Python` 中，要使用随机数，首先需要导入 随机数 的 模块 ——“工具包”

```
import random
```

- 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
- `random.randint(a, b)` ，返回 `[a, b]` 之间的整数，包含 `a` 和 `b`
- 例如：

```
random.randint(12, 20)  # 生成的随机数n: 12 <= n <= 20
random.randint(20, 20)  # 结果永远是 20
random.randint(20, 10)  # 该语句是错误的，下限必须小于上限
```

循环

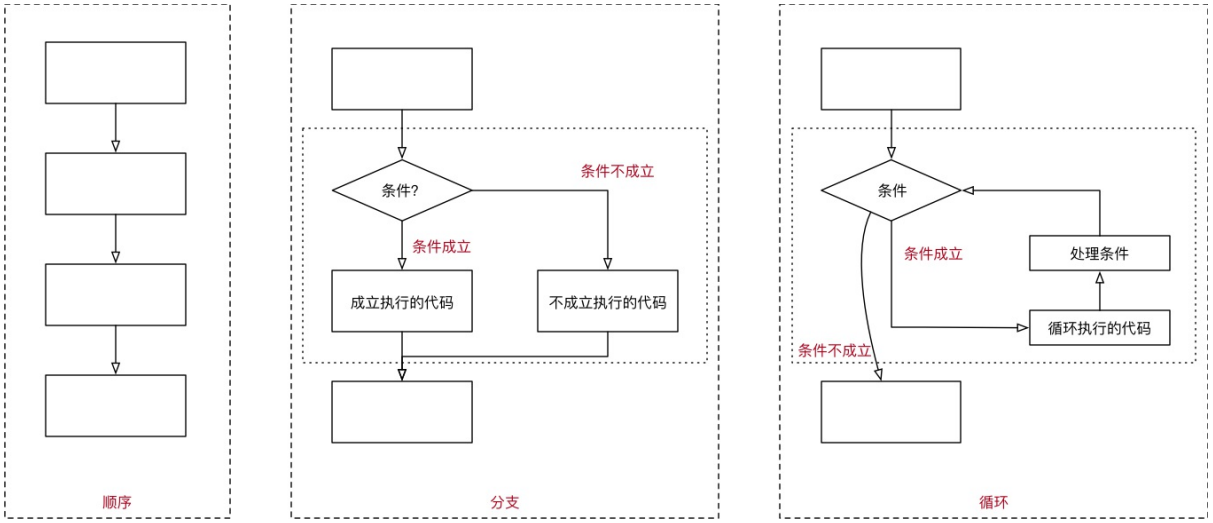
目标

1. 知道程序的三大流程
2. 掌握while循环的基本使用
3. 掌握break和continue的用法
4. 掌握for循环

1. 程序的三大流程

在程序开发中，一共有三种流程方式：

- 顺序 —— 从上向下，顺序执行代码
- 分支 —— 根据条件判断，决定执行代码的 分支
- 循环 —— 让 特定代码 重复 执行



2. while循环基本使用

- 循环的作用就是让 指定的代码 重复的执行
- `while` 循环最常用的应用场景就是 让执行的代码 按照 指定的次数 重复 执行
- 需求: 假如我有个女朋友，有一天我们闹矛盾生气了，女朋友说：道歉，说100遍“媳妇儿，我错 了”。这个时候程序员会怎么做？

> 答：100遍 `print('媳妇儿，我错了')`
> 思考：复制粘贴100次吗？
> 答：重复执行100次一样的代码，程序中循环即可

2.1 while语句基本语法

```
初始条件设置 — 通常是重复执行的 计数器

while 条件(判断计数器是否达到目标次数):
    条件满足时，做的事情1
    条件满足时，做的事情2
    条件满足时，做的事情3
    ...(省略)...

    处理条件(计数器 + 1)
```

注意：`while` 语句以及缩进部分是一个 完整的代码块

第一个 **while** 循环

需求: 打印 100 遍 "媳妇儿,我错了"

```
# 1. 定义重复次数计数器
i = 1

# 2. 使用 while 判断条件
while i <= 100:
    # 要重复执行的代码
    print("媳妇儿,我错了")
    # 处理计数器 i
    i = i + 1

print("循环结束后的 i = %d" % i)
```

> 注意：循环结束后，之前定义的计数器条件的数值是依旧存在的

死循环

> 由于程序员的原因，忘记 在循环内部 修改循环的判断条件，导致循环持续执行，程序无法终止！

2.2 Python 中的计数方法

常见的计数方法有两种，可以分别称为：

- 自然计数法（从 `1` 开始）—— 更符合人类的习惯
- 程序计数法（从 `0` 开始）—— 几乎所有的程序语言都选择从 0 开始计数

因此，大家在编写程序时，应该尽量养成习惯：除非需求的特殊要求，否则 循环 的计数都从 **0** 开始

2.3 循环计算

> 在程序开发中，通常会遇到 利用循环 重复计算 的需求

遇到这种需求，可以：

1. 在 `while` 上方定义一个变量，用于 存放最终计算结果
2. 在循环体内部，每次循环都用 最新的计算结果，更新 之前定义的变量

需求: 计算0~100所有数字的累加和结果

```
# 计算 0 ~ 100 之间所有数字的累计求和结果
# 0. 定义最终结果的变量
```

```python
result = 0

# 1. 定义一个整数的变量记录循环的次数
i = 0

# 2. 开始循环
while i <= 100:
    print(i)

    # 每一次循环，都让 result 这个变量和 i 这个计数器相加
    result += i
    # 处理计数器
    i += 1

print("0~100之间的数字求和结果 = %d" % result)
```

> 注意: 为了验证程序的准确性, 可以先改小数值, 验证结果正确后, 再改成1-100做累加

需求进阶**:** 计算0~100之间所有 偶数 的累计求和结果

开发步骤

1. 编写循环 确认 要计算的数字
2. 添加 结果 变量，在循环内部 处理计算结果

```python
# 0. 最终结果
result = 0
# 1. 计数器
i = 0

# 2. 开始循环
while i <= 100:
    # 判断偶数
    if i % 2 == 0:
        print(i)
        result += i
    # 处理计数器
    i += 1

print("0~100之间偶数求和结果 = %d" % result)
```

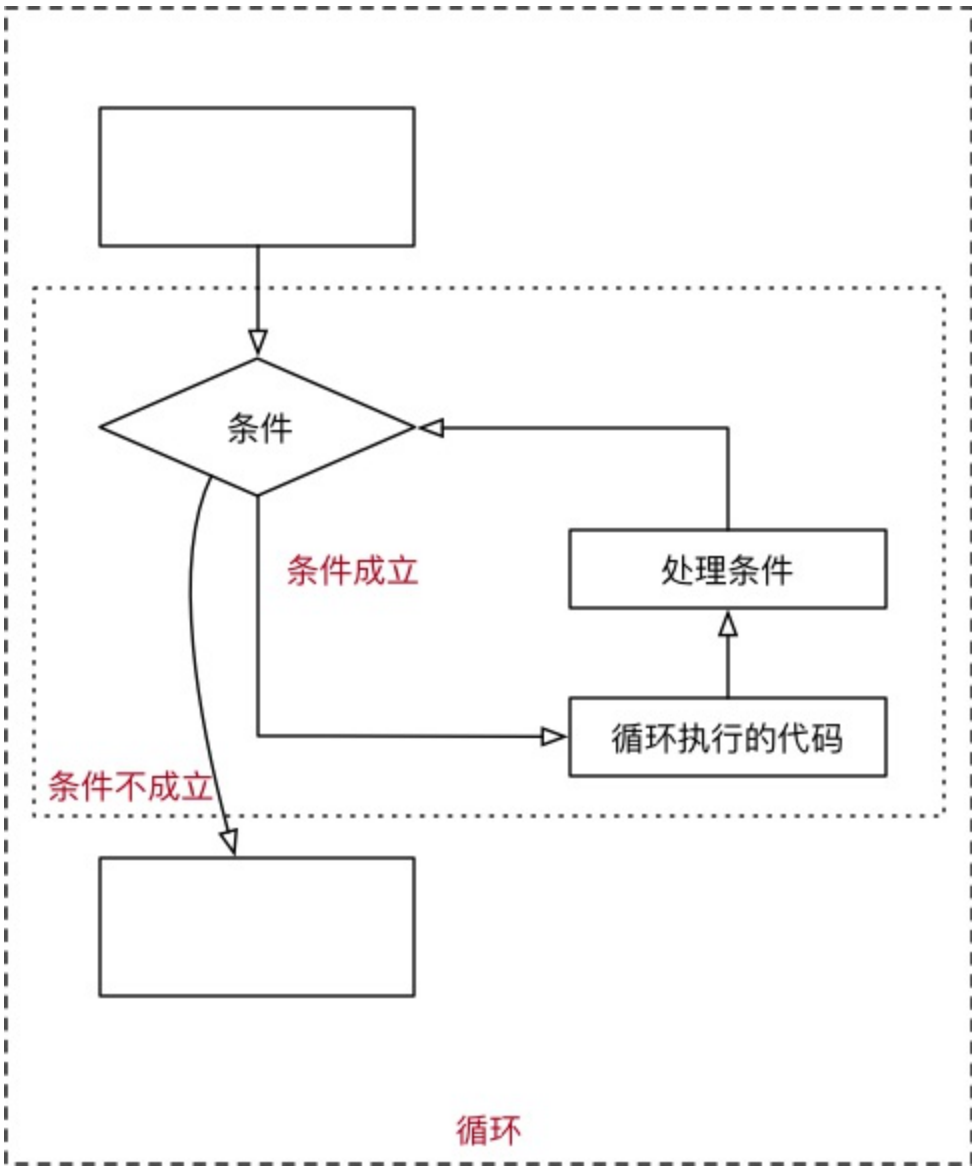
---

3. break 和 continue

> 注意: `break` 和 `continue` 是专门在循环中使用的关键字

- `break` 某一条件满足时，退出单层循环
- `continue` 某一条件满足时，结束本次循环(不执行continue后的循环体语句),执行下次循环

> `break` 和 `continue` 只针对 当前所在循环 有效



3.1 break

- 在循环过程中，如果 某一个条件满足后，不 再希望 循环继续执行，可以使用 `break` 退出循环
- 举例：一共吃5个苹果，吃完第一个，吃第二个...，这里"吃苹果"的动作是不是重复执行？

  情况一：如果吃的过程中，吃完第三个吃饱了，则不需要再吃第4个和第五个苹果，即是吃苹果的动作 停止，这里就是break控制循环流程，即终止此循环。

```
i = 1

while i <= 5:
    print("吃了第%d个苹果" % i)
    # break 某一条件满足时，退出循环，不再执行后续重复的代码
    if i == 3:
        print("吃饱了不吃了")
        break
    i += 1

print("over")
```

> `break` 只针对当前所在循环有效

3.2 continue

- 在循环过程中，如果 某一个条件满足后，不 希望 执行循环代码，但是又不希望退出循环，可以使用 `continue`
- 情况二：如果吃的过程中，吃到第三个吃出一个大虫子...,是不是这个苹果就不吃了，开始吃第四个苹 果，这里就是continue控制循环流程，即退出当前一次循环继而执行下一次循环代码。

```
i = 1
while i <= 5:
    if i == 3:
        print(f'大虫子，第{i}个苹果不吃了')
        # 在continue之前一定要修改计数器，否则会陷入死循环
        i += 1
        continue

    print(f'吃了第{i}个苹果')
    i += 1
```

- 注意：使用 `continue` 时，条件处理部分的代码，需要特别注意, 在**continue**之前一定要修改计数器，否则会陷入死循环

> `continue` 只针对当前所在循环有效

4. for循环

4.1 作用

- for循环也可以让指定的代码重复执行
- for循环可以遍历容器中的数据(遍历: 从容器中把数据一个一个取出)

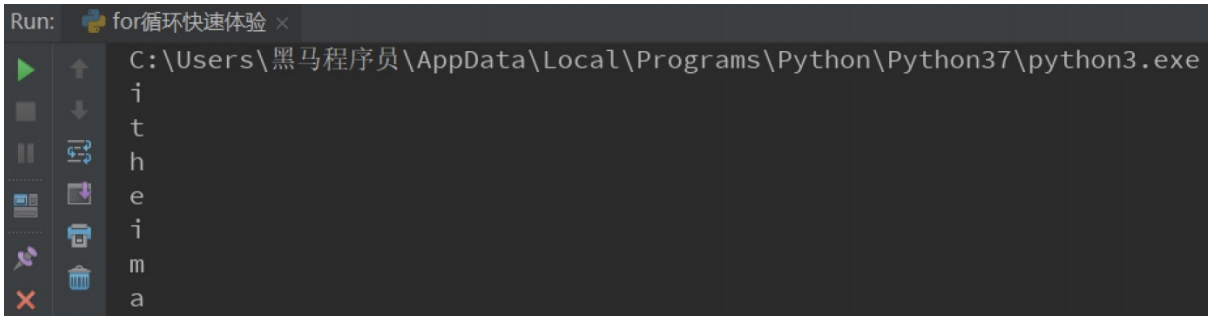
4.2 语法

```
for 临时变量  in  容器:
    重复执行的代码1
    重复执行的代码2
    ...
```

4.3 快速体验

```
str1 = 'itheima'
for i in str1:
    print(i)
```

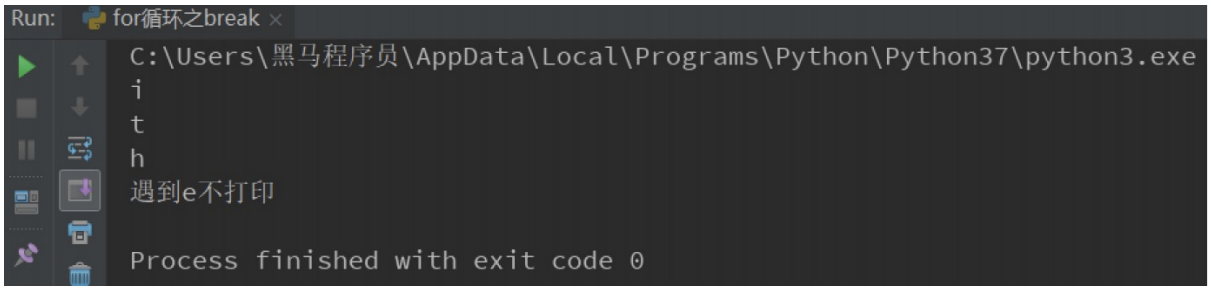
执行结果:



4.4 break作用于for循环

```
str1 = 'itheima'
for i in str1:
    if i == 'e':
        print('遇到e不打印')
        break
    print(i)
```

执行结果:



4.5 continue作用于for循环

```
str1 = 'itheima'
for i in str1:
    if i == 'e':
        print('遇到e不打印')
        continue
    print(i)
```

执行结果: