# 条件语句

## 格式一

if 判断条件:
    执行语句...

```
gender = '男'
if gender == '男':
    print("男士请进")
    print("这是男士专用洗手间...")


gender = "女"
if gender == '女': print("女士请进"); print("这是女士专用洗手间...")
```

## 格式二

if 判断条件:
    执行语句...
else:
    执行语句...

```
gender = '男'

if gender == '男':
    print('男士请进')

else:
    print('女士请进')
```

## 格式三

if 判断条件1:
    执行语句1...
elif 判断条件2:
    执行语句2...
elif 判断条件3:
    执行语句3...
else:
    执行语句4...

```
gender = input("请输入你的性别 (男/女): ")

if gender == "男":
    print("男士请进")

elif gender == "女":
    print("女士请进")

else:
    print("输入错误!")

print("程序结束")
```

## 三元表达式

- 它是一个表达式，而不是一个结构化的代码块

```
score = float(input("请在下方输入你的考试分数:\n"))

if score >= 90:
    print("牛逼")

elif score >= 80:
    print("优秀")

elif score >= 60:
    print("可以")

# elif score < 60:
#     print("同志仍需努力")

else:
    print("同志仍需努力")


# 三元表达式
result = print("牛逼") if score >= 90 else print("优秀")
if score >= 80 else print("可以") if score >= 60 else
print("同志仍需努力")
print(result)  # None



if score >= 90:
    result = "牛逼"
```

```
elif score >= 80:
    result = "优秀"

elif score >= 60:
    result = "可以"

else:
    result = "同志仍需努力"

print(result)

# 三元表达式
result = "牛逼" if score >= 90 else "优秀" if score >= 80
else "可以" if score >= 60 else "同志仍需努力"
print(result)


print("老师说我" + ("牛逼" if score >= 90 else "优秀" if
score >= 80 else "可以" if score >= 60 else "同志仍需努
力"))
```

## if 语句嵌套

```
print("敲门声: 咣咣咣~")
gender = input("请输入你的性别 (男/女): ")
if gender == "男":
    print("滚犊子...")
else:
    answer = input("请问是不是包租婆? (是/不是): ")
    if answer == '是':
        print("屋里没人...")
    else:
```

```
        answer2 = input("请问是不是小姐姐? (是/不是): ")
        if answer2 == '是':
            print("请进, 你是找我的吗?")
        else:
            print("不好意思, 您找错人了...")


# 三元表达式嵌套（了解一下）
print("不好意思, 我今天不能见男的, 您请回吧") if input("请问你是男是女 (男/女): ") == "男" else (print("不好意思, 今天我不在家, 房租下次给你") if input("请问是包租婆吗? (是/不是): ") == "是" else (print("小姐姐, 您好, 请进...") if input("请问你是小姐姐吗? (是/不是): ") == "是" else print("不好意思, 我今天不宜面圣, 您请回吧")))
```

## 条件语句特点

- 每个条件语句只会满足一次结果

```
num = 5

if num > 0:  # 满足条件, 输出a, 后面就不会执行了
    print("a")

elif num > 1:
    print("b")

elif num > 2:
    print("c")

elif num > 3:
    print("d")
```

```
else:
    print("e")

if num > 4:  # 然后又跳到这个条件语句中执行
    print("f")

else:
    print("g")
```

- 当判断条件是一个值时，会对该值进行 bool 判断

```
if 123456:
    print("123456为True")


if []:
    print("[]为True")
else:
    print("[]为False")


num = None

if num == None:  # num == None -> True
    print("123456")

if None:
    print("123456")
```

## 实现：石头剪刀布游戏

```
import random


INFO = {0: "石头", 1: "剪刀", 2: "布"}  # ["石头", "剪刀",
"布"]

computer = random.randint(0, 2)
player = int(input("请出拳！ 输入0代表石头、1代表剪刀、2代表
布： "))

# 展示电脑和玩家的出拳结果
print(f"电脑出拳： {INFO[computer]}\n玩家出拳：
{INFO[player]}")

# 判断谁输谁赢
if computer == player:
    print("平局!")
elif player-computer in (-1, 2):
    print("玩家胜利!")
else:
    print("电脑胜利!")
```

random 模块常用函数

- random.random()　返回 [0.0, 1.0) 范围内的随机浮点数
- random.randint(a, b)　返回 [a, b] 范围内的随机整数
- random.uniform(a, b)　返回 [a, b] / [b, a] 范围内的随机浮点数
- random.choice(seq)　从非空序列 seq 返回一个随机元素。 如果 seq 为空，则引发 IndexError
- random.sample(population, k)　从序列或者集合中随机获取k个元素，以列表形式返回
  （Python3.9 版本, 集合中采样已弃用）
- random.shuffle(x)　将可变序列 x 随机打乱位置

- random.randrange ([start,] stop [,step])　等效于从 range(start, stop, step) 里随机返回一个元素
- random.seed ([x])　起固定随机数的作用，x 可以是任意数字（x 可以理解为种子的名字）

```
import random


# 返回 [0.0, 1.0) 范围内的随机浮点数
a = random.random()
print(a)

# 返回 [2, 4] 范围内的随机整数
a = random.randint(2, 4)
print(a)

# 返回 [2.5, 3.5] 范围内的随机浮点数
a = random.uniform(2.5, 3.5)
a = random.uniform(3.5, 2.5)
print(a)

# 从序列返回一个随机元素
a = random.choice(range(10))
print(a)

# 从序列中随机获取k个元素，以列表形式返回
a = random.sample(("a", 1, 2, 3, "b"), 2)
print(a)

# 从集合中随机获取k个元素，以列表形式返回
a = random.sample({"a", 1, 2, 3, "b"}, 2)
print(a)

# 将可变序列随机打乱位置
a = [1, 2, 3, 4, 5]
```

```
random.shuffle(a)
print(a)

# 从[star, stop)范围按照step步长组成的数据范围里返回一个随机数
a = random.randrange(start=2, stop=100, step=2)
print(a)



# 设定好种子之后，每次执行程序随机数将是固定的
# 使用相同的随机数种子，生成的随机数值是一样的
print(random.random())

random.seed(10)
print("seed10:", random.random())

random.seed(7)
print("seed7:", random.random())

random.seed(10)
print("seed10:", random.random())
```

# 循环语句

# while 循环

格式：　　while 判断条件:
　　执行语句...

```
a = 1
while a < 4:
    print(a)  # 1 2 3
    a += 1
```

## while True

- 无限循环（俗称：死循环），配合控制语句去限制循环次数

```
a = 0

while True:
    print("hello")
    a += 1
    if a == 3:
        break
```

## while ... else

```
count = 0

while count < 5:
   print(count, "小于5")
   count = count + 1
```

```
else:
    print(count, "大于或等于 5")



count = 0

while count < 5:
    print(count, "小于5")
    count = count + 1

print(count, "大于或等于 5")




count = 0

while True:
    print(count, "小于5")
    count = count + 1
    if count >= 5:
        break

else:
    print(count, "大于或等于 5")



count = 0

while True:
    print(count, "小于5")
    count = count + 1
    if count >= 5:
```

```
        break

print(count, "大于或等于 5")
```

**while** 循环嵌套

```
# 实现九九乘法表
right = 1
while right <= 9:
    left = 1
    while left <= right:
        print(f"{left}x{right}={left*right}", end="\t")
        left += 1
    print()
    right += 1
```

## for 循环

for 变量 in 可迭代对象:
    执行语句...

```
str1 = "123ab"
lis1 = [1, 2, 3, "a", "b"]
tup1 = (1, 2, 3, "a", "b")
dic1 = {"one": 1, "two": 2}
set1 = {3, 2, 4, "a"}

for i in str1:
   print(i)
```

```
for i in lis1:
   print(i)

for i in tup1:
   print(i)

for i in dic1:
   print(i)

for i in set1:
   print(i)
```

### for ... in ... else

```
for i in [1, 2, 3, 4]:
    print(i)
    if i > 2:
        break

else:
    print(5)



for i in [1, 2, 3, 4]:
    print(i)
    if i > 2:
        break

print(5)
```

**for** 循环嵌套

```
# 实现九九乘法表
for right in range(1, 10):
    for left in range(1, right+1):
        print(f"{left}x{right}={left*right}", end="\t")
    print()
```

range([start], stop[, step])

- 返回一个按步骤生成的从start(包括)到stop(不包括)的整数序列，它是不可变的序列
- start：计数从 start 开始，默认是从 0 开始
- stop：计数到 stop 结束，但不包括 stop
- step：步长，默认为1
- range 类型相比常规 list 或 tuple 的优势在于一个 range 对象总是占用固定数量的（较小）内存，不论其所表示的范围有多大（因为它只保存了 start, stop 和 step 值）

```
print(list(range(4)))  # [0, 1, 2, 3]
print(list(range(1, 5)))  # [1, 2, 3, 4]
print(list(range(1, 8, 2)))  # [1, 3, 5, 7]
print(list(range(8, 1, -2)))  # [8, 6, 4, 2]
```

enumerate(iterable, start=0)

- 返回一个enumerate对象（迭代器）。迭代它会得到一个个的元组，每个元组是索引（从start开始，默认为 0）和索引对应iterable的值组成的

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
object1 = enumerate(seasons)  # <enumerate object at
0x000002A89E67E6C0>
print(list(object1))  # [(0, 'Spring'), (1, 'Summer'),
(2, 'Fall'), (3, 'Winter')]

object2 = enumerate(seasons, start=1) # 设置开始迭代的索引
print(list(object2))  # [(1, 'Spring'), (2, 'Summer'),
(3, 'Fall'), (4, 'Winter')]
```

## 循环控制语句

### break

- 终止所在的循环

```
for _ in range(3):
    for _ in range(4):
        print("hello")
        break
```

### continue

- 跳过当前，继续到循环位置

```
a = 0
while a <= 5:

    if a == 3:
        a += 1
        continue

    print(a)
    a += 1
```

## 优化：石头剪刀布游戏

```
import random

INFO = {0: '石头', 1: '剪刀', 2: '布'}

while True:
    count_p = 0
    count_c = 0
    for i in range(1, 4):
        computer = random.choice(range(3))

        # 判定玩家的输入是否合理，如果不合理重新输入
        while (player := int(input('请出拳(0代表石头，1代表剪刀，2代表布)：'))) not in INFO:
            print('您的输入有误，请重新输入...')

        # 出拳展示
        print(f'电脑出拳：{INFO[computer]}')
        print(f'玩家出拳：{INFO[player]}')

        # 判断本局胜方
```

```
        if computer == player:
            print(f'第{i}把平局!')
        elif computer - player in (1, -2):
            print(f'第{i}把玩家胜!')
            count_p += 1
        else:
            print(f'第{i}把电脑胜!')
            count_c += 1

        # 如果有一方先赢两把，就不需要再比了
        if 2 in (count_c, count_p):
            break

    # 判断本轮胜方
    if count_p == count_c:
        print('本轮三局两胜平局!')
    elif count_p > count_c:
        print('本轮三局两胜玩家胜!')
    else:
        print('本轮三局两胜电脑胜!')

    # 判定玩家的输入是否合理，如果不合理重新输入
    while (ans := input('请问您是否需要继续游戏(Y/N): '))
not in ('Y', 'y', 'N', 'n'):
        print('您的输入有误，请重新输入...')

    # 在玩家输入合理的前提下，继续判定输入是否为N或者n
    if ans in ('N', 'n'):
        print('游戏结束，欢迎下次光临...')
        break
```

# 推导式

## 列表推导式

由一对方括号里面包含一个表达式，后面跟一个 for 子句，然后是零个或多个 for 或 if 子句组成，其结果将是一个新列表，由表达式依据后面的 for 和 if 子句的内容进行求值计算而得出

```
squares = []
for x in range(10):
    squares.append(x**2)


squares = [x**2 for x in range(10)]
```

```
result = [(x, y) for x in [1,2,3] for y in [3,1,4] if x
!= y]


result = []
for x in [1,2,3]:
    for y in [3,1,4]:
        if x != y:
            result.append((x, y))
```

嵌套的列表推导式

- 列表推导式中的初始表达式可以是任何表达式，包括另一个列表推导式

```
```

```python
matrix = [[1, 2, 3, 4],
          [5, 6, 7, 8],
          [9, 10, 11, 12]]


result = [[row[i] for row in matrix] for i in range(4)]


result = []
for i in range(4):
    result.append([row[i] for row in matrix])


result = []
for i in range(4):
    temp = []
    for row in matrix:
        temp.append(row[i])
    result.append(temp)
```

## 字典推导式

```python
dic = {x: x**2 for x in range(4)}
print(dic)


dic = {x: x**2 for x in range(6) if x % 2 == 0}
print(dic)


dic = {k: v for k, v in zip((1, 2, 3), (4, 5, 6))}
print(dic)
```

## 集合推导式

```
set1 = {x**2 for x in range(4)}
print(set1)


set1 = {x for x in 'abracadabra' if x not in 'abc'}
print(set1)
```

# pass 语句

---

pass 是一个空操作, 当它被执行时, 什么都不发生。它适合当语法上需要一条语句但并不需要执行任何代码时用来临时占位

```
num = int(input("请输入一个数字: "))
if num < 100:
    pass
else:
    pass
```