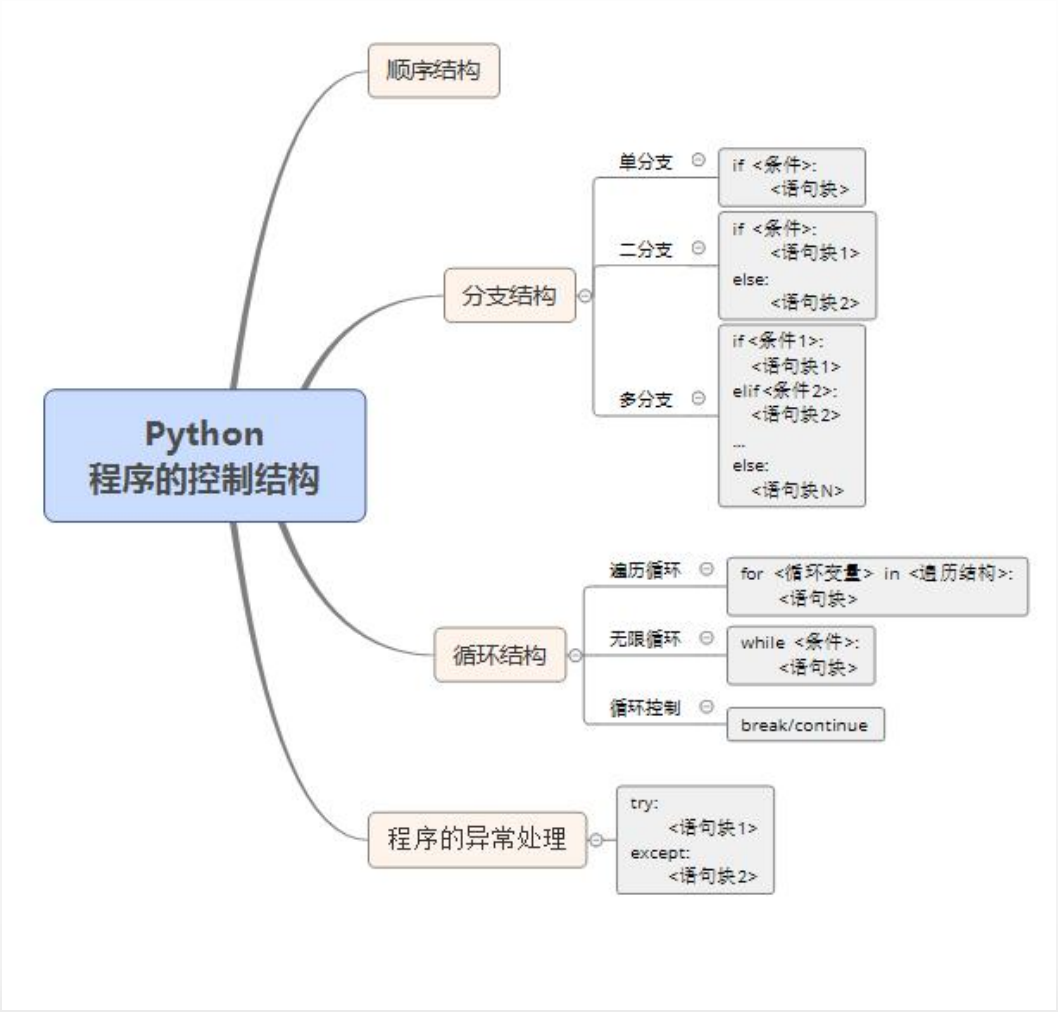
全国计算机等级考试二级教程
Python语言程序设计
(2018年版)

【第5章】
程序的控制结构

考纲考点

- 程序的三种控制结构
- 程序的分支结构: 单分支结构、二分支结构、多分支结构
- 程序的循环结构: 遍历循环、无限循环、break和continue循环控制
- 程序的异常处理: try-except

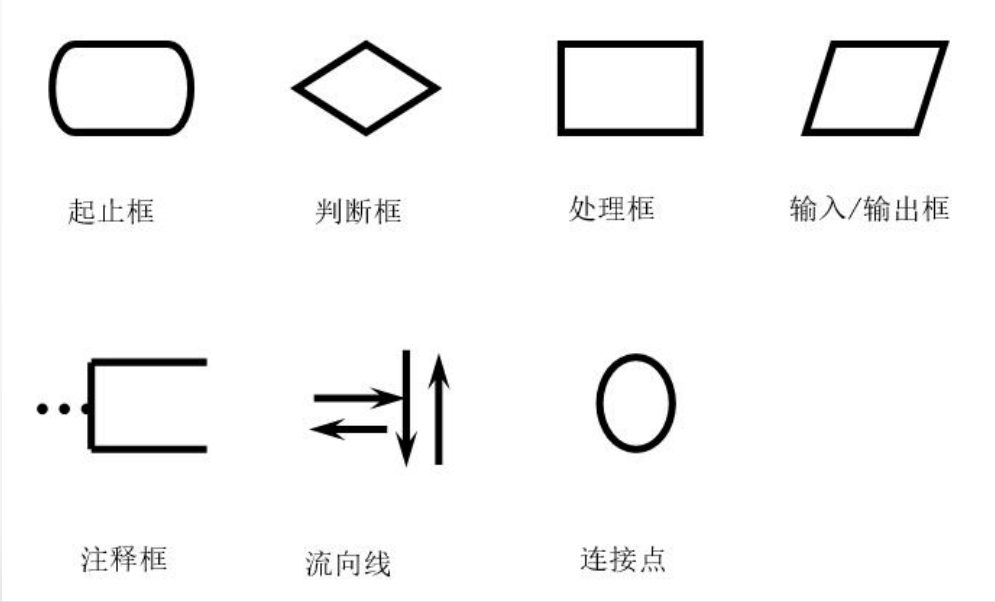
知识导图


Python
程序的控制结构
顺序结构
分支结构
单分支
if <条件>:
<语句块>
二分支
if <条件>:
<语句块1>
else:
<语句块2>
多分支
if<条件1>:
<语句块1>
elif<条件2>:
<语句块2>
...
else:
<语句块N>
循环结构
遍历循环
for <循环变量> in <遍历结构>:
<语句块>
无限循环
while <条件>:
<语句块>
循环控制
break/continue
程序的异常处理
try:
<语句块1>
except:
<语句块2>

程序的基本结构

程序的流程图

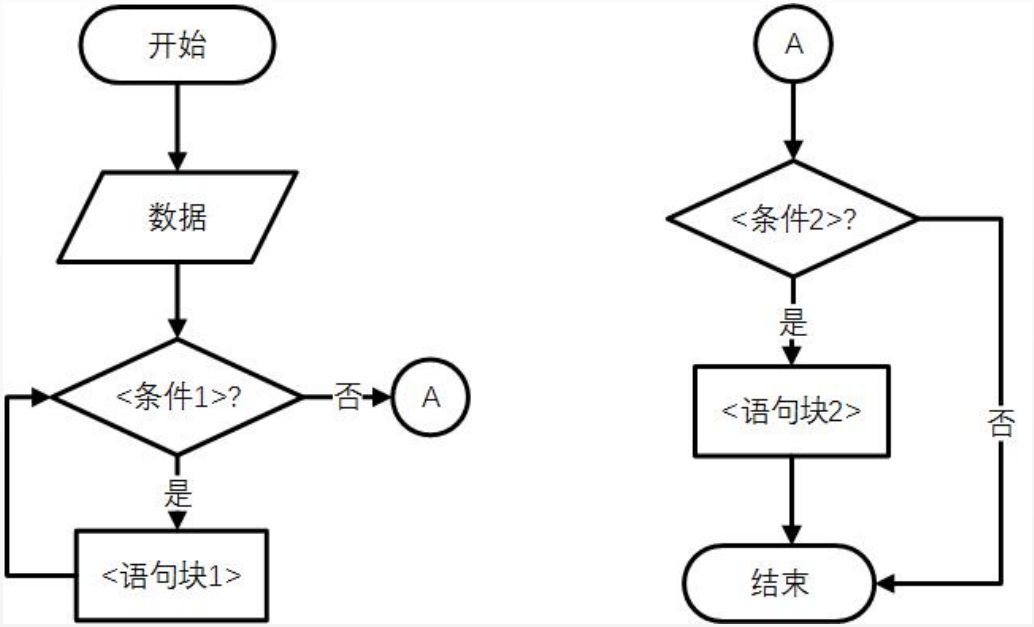
- 程序流程图用一系列图形、流程线和文字说明描述程序的基本操作和控制流程，它是程序分析和过程描述的最基本方式。
- 流程图的基本元素包括7种



程序的流程图

- 起止框：表示程序逻辑的开始或结束；
- 判断框：表示一个判断条件，并根据判断结果选择不同的执行路径；
- 处理框：表示一组处理过程，对应于顺序执行的程序逻辑；
- 输入输出框：表示程序中的数据输入或结果输出；
- 注释框：表示程序的注释；
- 流向线：表示程序的控制流，以带箭头直线或曲线表达程序的执行路径；
- 连接点：表示多个流程图的连接方式，常用于将多个较小流程图组织成较大流程图。

程序的流程图



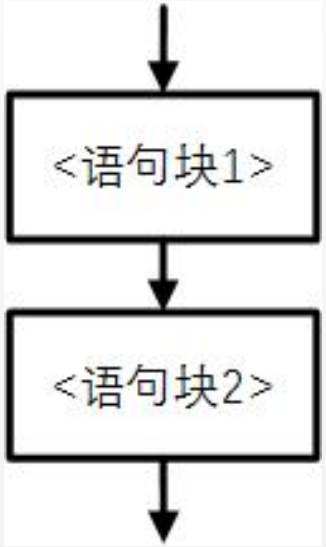
- 程序流程图示例：由连接点A连接的一个程序

程序的基本结构

- 程序由三种基本结构组成：**顺序结构、分支结构和循环结构**。
- 任何程序都由这三种基本结构组合而成
- 这些基本结构都有一个入口和一个出口。任何程序都由这三种基本结构组合而成

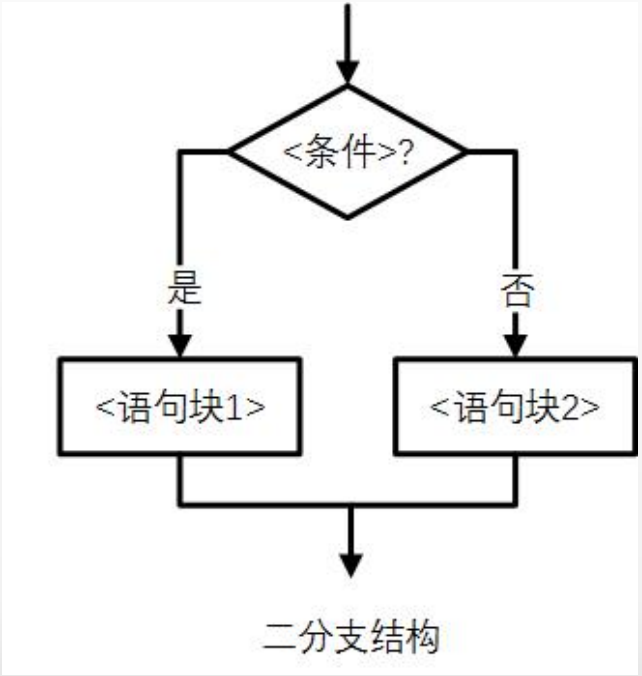
程序的基本结构

- 顺序结构是程序按照线性顺序依次执行的一种运行方式，其中语句块1S1和语句块S2表示一个或一组顺序执行的语句



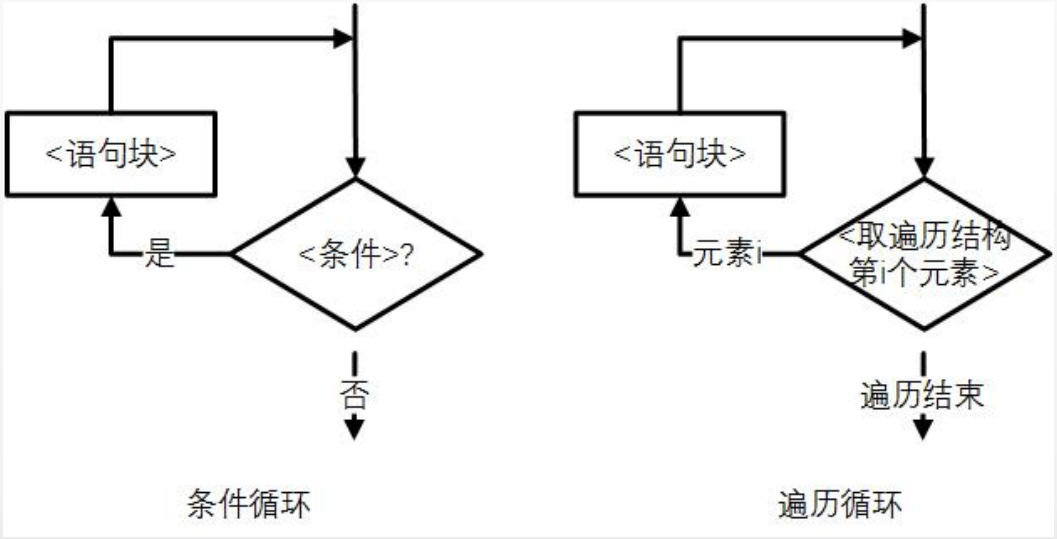
程序的基本结构

- 分支结构是程序根据条件判断结果而选择不同向前执行路径的一种运行方式，基础的分支结构是**二分支结构**。由二分支结构会组合形成多分支结构



程序的基本结构

- 循环结构是程序根据条件判断结果向后反复执行的一种运行方式，根据循环体触发条件不同，包括条件循环和遍历循环结构



程序的基本结构

- 在三种基本控制逻辑基础上，Python语言进行了必要且适当的扩展。
- 在分支结构原理的基础上，Python增加了异常处理，使用**try-except**保留字
- 异常处理以程序异常为判断条件，根据一段代码执行的正确性进行程序逻辑选择。异常处理是分支结构的一种扩展。

程序的基本结构

- 在循环结构原理的基础上，Python提供两个循环控制符**break和continue**，对循环的执行过程进行控制。break控制符用来结束当前循环，continue控制符用来结束当前循环的当次循环过程，

程序的分支结构

单分支结构: if语句

- Python的单分支结构使用if保留字对条件进行判断，使用方式如下

```
if <条件>:
    语句块
```

- 其中，if、:和<语句块>前的缩进都是语法的一部分。<语句块>是if条件满足后执行的一个或多个语句序列，缩进表达<语句块>与if的包含关系。<条件>是一个产生True或False结果的语句，当结果为True时，执行<语句块>，否则跳过<语句块>。

单分支结构: if语句

```
1  # 判断用户输入数字的奇偶性
2  s = eval(input("请输出一个整数："))
3  if s % 2 == 0:
4      print("这是个偶数")
5  print("输入数字是:", s)
```

- <条件>是一个或多个条件，多个条件间采用and或or进行逻辑组合。and表示多个条件“与”的关系，or表示多个条件“或”的关系

```
1  # 判断用户输入数字的特定
2  s = eval(input("请输出一个整数："))
3  if s % 3 == 0 and s % 5 == 0:
4      print("这个数字既能被3整除，又能被5整除")
5  print("输入数字是:", s)
```

二分支结构: if-else语句

- Python的二分支结构使用if-else保留字对条件进行判断，语法格式如下：

```
if <条件>:
    <语句块1>
else:
    <语句块2>
```

- 其中，if、:和语句块前的缩进都是语法的一部分。

二分支结构: if-else语句

- <语句块1>在if中<条件>满足即为True时执行，<语句块2>在if中<条件>不满足即为False时执行。简单说，二分支结构根据条件的True或False结果产生两条路径。

```
# 判断用户输入数字的某个属性
s = eval(input("请输出一个整数："))
if s % 3 == 0 and s % 5 == 0:
  print("这个数字能够同时被3和5整除")
else:
  print("这个数字不能够同时被3和5整除")
```

二分支结构: if-else语句

- 二分支结构还有一种更简洁的表达方式，适合<语句块1>和<语句块2>都只包含简单表达式的情况，语法格式如下：

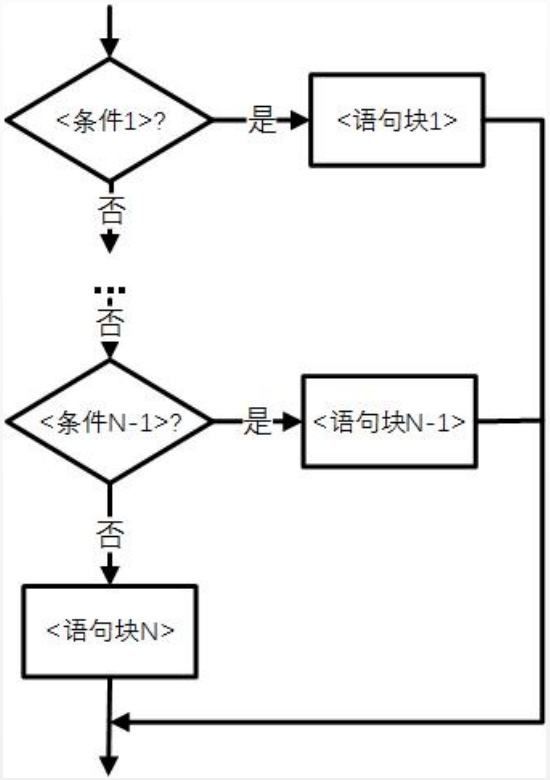
<表达式1> if <条件> else <表达式2>

```
# 判断用户输入数字的某个属性
s = eval(input("请输出一个整数："))
token = "" if s % 3 == 0 and s % 5 == 0  else "不"
print("这个数字{}能够同时被3和5整除".format(token))
```

多分支结构: if-elif-else语句

- Python的if-elif-else描述多分支结构，语句格式如下：

```
if <条件1>:
    <语句块1>
elif <条件2>:
    <语句块2>
...
else:
    <语句块N>
```



多分支结构: if-elif-else语句

- 多分支结构通常用于判断同一个条件或一类条件的多个执行路径。要注意，Python会按照多分支结构的代码顺序依次评估判断条件，寻找并执行第一个结果为True条件对应的语句块，当前语句块执行后跳过整个if-elif-else结构。
- 利用多分支结构编写代码时要注意多个逻辑条件的先后关系。

多分支结构: if-elif-else语句

- 获取用户输入的一个百分制成绩，转换成五分制，给出对应的A、B、C、D、E等级。

```
# 将百分制成绩转换为五分制成绩
score = eval(input("请输出一个百分制成绩："))
if score >= 60.0:
        grade = "D"
elif score >= 70.0:
        grade = "C"
elif score >= 80.0:
        grade = "B"
elif score >= 90.0:
        grade = "A"
else:
   grade = "E"
print("对应的五分制成绩是：{}".format(grade))
```

```
>>>
请输出一个百分制成绩：80
对应的五分制成绩是：D
```

多分支结构: if-elif-else语句

- 显然，百分制80分不应该是等级D，上述代码运行正确但逻辑存在错误，在于弄错了多个逻辑条件的先后关系，修改后代码如下。

```
# 将百分制成绩转换为五分制成绩
score = eval(input("请输出一个百分制成绩："))
if score >= 90.0:
        grade = "A"
elif score >= 80.0:
        grade = "B"
elif score >= 70.0:
        grade = "C"
elif score >= 60.0:
        grade = "D"
else:
   grade = "E"
print("对应的五分制成绩是：{}".format(grade))
```

判断条件及组合

- 分支结构中的判断条件可以使用任何能够产生True或False的语句或函数。形成判断条件最常见的方式是采用关系操作符

操作符	数学符号	操作符含义
<	<	小于
<=	≤	小于等于
>=	≥	大于等于
>	>	大于
==	=	等于
!=	≠	不等于

判断条件及组合

```
>>>4 < 5
True
>>>"Python" > "python"
False
```

- Python语言中，任何非零的数值、非空的数据类型都等价于True，0或空类型等价于False，可以直接用作判断条件。

```
>>>0 == False
True
>>>"" == True
False
```

判断条件及组合

- Python语言使用保留字**not、and和or**对条件进行逻辑运算或组着。
- 保留字not表示单个条件的“否”关系，and表示多个条件之间的“与”关系，保留字or表示多个条件之间的“或”关系。

```
>>>not True
False
>>>a = 80
>>>( a > 100) or ( a > 50 and a < 90)
True
```

程序的循环结构

程序的循环结构

- Python语言的循环结构包括两种：遍历循环和无限循环。
- **遍历循环**使用保留字for依次提取遍历结构各元素进行处理；
- **无限循环**使用保留字while根据判断条件执行程序。

遍历循环: for

- 遍历循环可以理解为从遍历结构中逐一提取元素，放在循环变量中，对于每个所提取的元素执行一次语句块。for语句的循环执行次数是根据遍历结构中元素个数确定的。

for <循环变量> in <遍历结构>:

<语句块>

- 遍历结构可以是字符串、文件、range()函数或组合数据类型等。

遍历循环: for

- 对于字符串，可以逐一遍历字符串的每个字符，基本使用方式如下：

  for <循环变量> in **<字符串变量>:**

  <语句块>

```
>>>for c in "Python":
print(c)
P
y
t
h
o
n
```

遍历循环: for

- 使用range()函数，可以指定语句块的循环次数，基本使用方式如下：

  for <循环变量> in **range(<循环次数>):**

  <语句块>

```
>>>for i in range(5):
print(i)
0
1
2
3
4
```

遍历循环: for

- 遍历循环还有一种扩展模式，使用方法如下：

```
for <循环变量> in <遍历结构>:
        <语句块1>
else:
        <语句块2>
```

- 当for循环正常执行之后，程序会继续执行else语句中内容。else语句只在循环正常执行之后才执行并结束，因此，可以在<语句块2>中放置判断循环执行情况的语句。

遍历循环: for

```
1  for s in "PY":
2      print("循环执行中: " + s)
3  else:
4      s = "循环正常结束"
5  print(s)
```

```
>>>
循环执行中: P
循环执行中: Y
循环正常结束
```

无限循环: while

- Python通过保留字while实现无限循环

while <条件>:

<语句块>

- 当程序执行到while语句时，判断条件如果为True，执行循环体语句，语句结束后返回再次判断while语句的条件；当条件为False时，循环终止，执行与while同级别缩进的后续语句。

无限循环: while

```
>>>n = 0
>>>while  n < 10:
        print(n)
        n = n + 3
0
3
6
9
```

无限循环: while

- 无限循环也有一种使用保留字else的扩展模式，使用方法如下：

```
while <条件>:
        <语句块1>
else:
        <语句块2>
```

- 在这种扩展模式中，当while循环正常执行之后，程序会继续执行else语句中内容。else语句只在循环正常执行后才执行，因此，可以在语句块2中放置判断循环执行情况的语句。

无限循环: while

```
1  s, idx = "PY", 0
2  while idx < len(s):
3      print("循环执行中: " + s[idx])
4      idx += 1
5  else:
6      s = "循环正常结束"
7  print(s)
```

```
>>>
循环执行中: P
循环执行中: Y
循环正常结束
```

循环控制: break和continue

- 循环结构有两个辅助循环控制的保留字：**break和continue**。break用来跳出最内层for或while循环，脱离该循环后程序从循环后代码继续执行。

```
1  while True:
2      s = input("请输入一个名字(按Q退出): ")
3      if s == "Q":
4          break
5      print("输入的名字是:", s)
6  print("程序退出")
```

```
>>>
请输入一个名字(按Q退出): 毛泽东
输入的名字是: 毛泽东
请输入一个名字(按Q退出): 邓小平
输入的名字是: 邓小平
请输入一个名字(按Q退出): Q
程序退出
```

循环控制: break和continue

- 如果有2层或多层循环，break退出最内层循环。
- continue用来结束当前当次循环，即跳出循环体中下面尚未执行的语句，但不跳出当前循环。

```
1  for s in "PYTHON":
2      if s == "Y" or s == "y":
3          continue
4      print(s, end="")
```

```
>>>
PTHON
```

- continue语句和break语句的区别是：continue语句只结束本次循环，不终止整个循环的执行，而break具备结束循环的能力。

程序的异常处理

程序的异常处理

- Python程序一般对输入有一定要求，但当实际输入不满足程序要求时，可能会产生程序的运行错误。

```
>>>n = eval(input("请输入一个数字: "))
请输入一个整数: python


Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    n = eval(input("请输入一个数字: "))
  File "<string>", line 1, in <module>
NameError: name 'python' is not defined
```

程序的异常处理

- 由于使用了eval()函数，如果用户输入不是一个数字则可能报错。这类由于输入与预期不匹配造成的错误有很多种可能，不能逐一列出可能性进行判断。为了保证程序运行的稳定性，这类运行错误应该被程序捕获并合理控制。

程序的异常处理

- Python语言使用保留字try和except进行异常处理，基本的语法格式如下：。

```
try:
        <语句块1>
except:
        <语句块2>
```

- 语句块1是正常执行的程序内容，当执行这个语句块发生异常时，则执行except保留字后面的语句块2。

程序的异常处理

```
try:
    n = eval(input("请输入一个数字: "))
    print("输入数字的3次方值为: ", n**3)
except:
    print("输入错误，请输入一个数字!")
```

```
>>>
请输入一个数字: 1010
输入数字的3次方值为:  1030301000
>>>
请输入一个数字: python
输入错误，请输入一个数字!
```

程序的异常处理

- 除了输入之外，异常处理还可以处理程序执行中的运行异常。

```
>>>for i in range(5):
print(10/i, end=" ")
Traceback (most recent call last):
  File "<pyshell#12>", line 2, in <module>
    print(10/i, end=" ")
ZeroDivisionError: division by zero
```

```
1  try:
2      for i in range(5):
3      print(10/i, end=" ")
4  except:
5      print("某种原因，出错了！")
```

```
>>>
某种原因，出错了！
```

实例解析：猜数字游戏

实例解析：猜数字游戏

- 编写一个“猜数字游戏”的程序，在1到1000之间随机产生一个数，然后请用户循环猜测这个数字，对于每个答案只回答“猜大了”或“猜小了”，直到猜测准确为止，输出用户的猜测次数。

实例解析：猜数字游戏

- 为了产生随机数，需要使用Python语言的随机数标准库random

```
1 import random
2 target = random.randint(1,1000)
```

- 根据程序需求，需要考虑不断地让用户循环输入猜测值，并根据猜测值和目标值之间的比较决定程序逻辑。

实例解析：猜数字游戏

```
1  import random
2  target = random.randint(1,1000)
3  count = 0
4  while True:
5      guess = eval(input('请输入一个猜测的整数(1至1000)：'))
6      count = count + 1
7      if guess > target:
8          print('猜大了')
9      elif guess < target:
10         print('猜小了')
11     else:
12         print('猜对了')
13         break
print("此轮的猜测次数是:", count)
```

实例解析：猜数字游戏

- 由于使用了eval(input())方式获得用户输入，如果用户输入非数字产生运行错误，程序将会退出。为了增加程序鲁棒性，增加异常处理机制。

```
import random
target = random.randint(1,1000)
count = 0
while True:
    try:
        guess = eval(input('请输入一个猜测的整数(1至1000): '))
    except:
        print('输入有误，请重试，不计入猜测次数哦！')
        continue
    count = count + 1
    if guess > target:
        print('猜大了')
    elif guess < target:
        print('猜小了')
    else:
        print('猜对了')
        break
print("此轮的猜测次数是:", count)
```

实例解析：猜数字游戏

- 该程序执行效果如下

```
>>>
请输入一个猜测的整数(1至1000)：500
猜大了
请输入一个猜测的整数(1至1000)：Python
输入有误，请重试，不计入猜测次数哦！
请输入一个猜测的整数(1至1000)：260
猜小了
请输入一个猜测的整数(1至1000)：380
猜大了
请输入一个猜测的整数(1至1000)：300
猜小了
请输入一个猜测的整数(1至1000)：340
猜小了
请输入一个猜测的整数(1至1000)：360
猜小了
请输入一个猜测的整数(1至1000)：370
猜对了
此轮的猜测次数是：7
```

本章小结

本章讲解了程序的三种控制结构，具体讲解了分支结构的三种类型：单分支结构、二分支结构合多分支结构，以及判断条件的组合。进一步具体讲解了循环结构的两种类型：遍历循环和无限循环，以及循环控制符break和continue。最后，讲解了程序的基本异常处理方法。通过猜数字游戏的实例帮助读者理解程序结构和异常处理的运用。

猜数字就是人生历程，运气重要？方法重要？或许，快乐的经历才最重要。