Python 条件判断

条件判断

计算机之所以能做很多自动化的任务，因为它可以自己做条件判断。 If 是 Python 中的主要选择工具 代表 Python 程序所拥有的大多数逻辑

Python 的 if 语句的格式为 if 测试 后面跟着一个或多个可选的 elif 以及一个 最终可选的 else，在 if 执行时 Python 会测试第一个计算结果为真的代码块 如果测试都为假的时候 则执行 else

基本例子:

if expression:

    argument1

    argument2

if …. Else….

If expression:

    Argument1

Else:

    Sdfsfsdf

多路分支

If – elif – else

If expression:

Elif expression:

Elif expression:

Else:

Python 没有 switch

Switch express

Case))

Case))

Default))

真：所有的非 0 所有的非空

假: 0 [] () set() {} None

比如，输入用户年龄，根据年龄打印不同的内容，在 Python 程序中，用 `if` 语句实现：

```
age = 20

if age >= 18:

    print('your age is', age)

    print('adult')
```

根据 Python 的缩进规则，如果 `if` 语句判断是 `True`，就把缩进的两行 print 语句执行了，否则，什么也不做。

也可以给 `if` 添加一个 `else` 语句，意思是，如果 `if` 判断是 `False`，不要执行 `if` 的内容，去把 `else` 执行了：

```
age = 3

if age >= 18:

    print('your age is', age)

    print('adult')

else:
```

```
    print('your age is', age)

    print('teenager')
```

注意不要少写了冒号`:`。

当然上面的判断是很粗略的，完全可以用`elif`做更细致的判断：

```
age = 3


多路分支


if age >= 18:

    print('adult')

elif age >= 6:

    print('teenager')

else:

    print('kid')
```

`elif`是`else if`的缩写，完全可以有多个`elif`，所以`if`语句的完整形式就是：

```
if <条件判断 1>:

    <执行 1>

elif <条件判断 2>:

    <执行 2>

elif <条件判断 3>:

    <执行 3>

else:
```

```
    <执行 4>
```

`if`语句执行有个特点，它是从上往下判断，如果在某个判断上是`True`，把该判断对应的语句执行后，就忽略掉剩下的`elif`和`else`，所以，请测试并解释为什么下面的程序打印的是`teenager`：

```
age = 20

if age >= 6:

    print('teenager')

elif age >= 18:

    print('adult')

else:

    print('kid')
```

`if`判断条件还可以简写，比如写：

```
if x:

    print('True')
```

只要`x`是非零数值、非空字符串、非空 list 等，就判断为`True`，否则为`False`。

再议 input()　raw_input()

最后看一个有问题的条件判断。很多同学会用`input()`读取用户的输入，这样可以自己输入，程序运行得更有意思：

```
birth = input('birth: ')

if birth < 2000:

    print('00 前')

else:

    print('00 后')
```

输入`1982`，结果报错：

```
Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

TypeError: unorderable types: str() > int()
```

这是因为 `input()` 返回的数据类型是 `str`，`str` 不能直接和整数比较，必须先把 `str` 转换成整数。Python 提供了 `int()` 函数来完成这件事情：

```
s = input('birth: ')

birth = int(s)

if birth < 2000:

    print('00 前')

else:

    print('00 后')
```

再次运行，就可以得到正确地结果。但是，如果输入 `abc` 呢？又会得到一个错误信息：

```
Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

ValueError: invalid literal for int() with base 10: 'abc'
```

原来 `int()` 函数发现一个字符串并不是合法的数字时就会报错，程序就退出了。

如何检查并捕获程序运行期的错误呢？后面的错误和调试会讲到。

Python 的语法规则

1. 语句是逐个运行的 Python 会按照次序从头到尾执行文件中嵌套块当中的语句 像 if 以及 循环 for 这样的语句会让解释器在程序内跳跃 Python 经过一个程序的路径叫做控制流程 像 if 这种会对控制流程产生影响的语句叫做流程控制语句

2. 块和语句的边界会自动检测

   python 中没有 begin 或 end 这样的分隔符 也没用大括号对包住一个区块的语句 python 使用的是首行缩进的形式 区分代码的逻辑 语句结束也不用以分号结尾

3. 复合语句 = 首行 + : + 缩进语句 Python 中所有复合语句都遵循相同的格式 首行会以 : 终止 再接一个或多个嵌套的语句

真值测试

1. 任何非 0 的数字和非空的对象都是真
2. 数字 0 空对象 None 会被认为是假
3. 比较和相等测试会返回 真或假
4. and 和 or 运算会返回真假

x and y

如果 x y 都为真 则返回真

x or y

如果 x 或 y 有一个是真则返回真

not x

如果 x 是假 则返回真

In [2]: bool(1) and bool(2)

Out[2]: True

In [3]: bool(1) or bool(2)

Out[3]: True

In [5]: bool(1) or bool(0)

Out[5]: True

In [6]: bool(1) and bool(0)

Out[6]: False

In [7]: not bool(0)

Out[7]: True

In [8]: not bool(1)

Out[8]: False

If / else 三元表达式

一个简单的 if 语句根据 x 的真假把 A 设置成 y 或 z

```
if x:
    a = y
```

else:

    a = z

如上面的例子所展示 有时这类语句设计的元素相当简单 但是我们用 4 行代码来实现似乎代码不够优雅

可以用三元表达式 ['good' if j > 5 else 'bad' for j in range(10)]

In [9]: 'good' if 3 else 'bad'

Out[9]: 'good'

In [10]: 'good' if 0 else 'bad'

Out[10]: 'bad'

~