分支结构

迄今为止，我们写的 Python 程序都是一条一条语句按顺序向下执行的，这种代码结构叫做顺序结构。然而仅有顺序结构并不能解决所有的问题，比如我们设计一个游戏，游戏第一关的过关条件是玩家获得 1000 分，那么在第一关完成后，我们要根据玩家得到的分数来决定是进入第二关，还是告诉玩家“Game Over”（游戏结束）。在这种场景下，我们的代码就会产生两个分支，而且只有一个会被执行。类似的场景还有很多，我们将这种结构称之为“分支结构”或“选择结构”。给大家一分钟的时间，你应该可以想到至少 5 个以上类似的例子，赶紧试一试吧！

使用if和else构造分支结构

在 Python 中，构造分支结构最常用的是 `if` 、 `elif` 和 `else` 三个关键字。所谓**关键字**就是编程语言中有特殊含义的单词，很显然你不能够使用它作为变量名。当然，我们并不是每次构造分支结构都会把三个关键字全部用上，我们通过例子加以说明。例如我们要写一个身体质量指数（BMI）的计算器。身体质量质数也叫体质指数，是国际上常用的衡量人体胖瘦程度以及是否健康的一个指标，计算公式如下所示。通常认为 $18.5 \leq BMI < 24$ 是正常范围， $BMI < 18.5$ 说明体重过轻， $BMI \geq 24$ 说明体重过重， $BMI \geq 27$ 就属于肥胖的范畴了。

$$BMI = \frac{体重}{身高^2}$$

> **说明**：上面公式中的体重以千克（kg）为单位，身高以米（m）为单位。

```
"""
BMI计算器

Version: 1.0
Author: 骆昊
"""
height = float(input('身高(cm): '))
weight = float(input('体重(kg): '))
bmi = weight / (height / 100) ** 2
print(f'{bmi = :.1f}')
if 18.5 <= bmi < 24:
    print('你的身材很棒！')
```

> **提示**： `if` 语句的最后面有一个 `:` ，它是用英文输入法输入的冒号；程序中输入的 `'` 、 `"` 、 `=` 、 `(` 、 `)` 等特殊字符，都是在英文输入法状态下输入的，这一点之前已经提醒过大家了。很多初学者经常会忽略这一点，等到执行代码时，就会看到一大堆错误提示。当然，认真读一下错误提示还是很容易发现哪里出了问题，但是**强烈建议**大家在写代码的时候**切换到英文输入法**，这样可以避免很多不必要的麻烦。

上面的代码中，我们在计算和输出 BMI 之后，加上了一段分支结构，如果满足 $18.5 \leq BMI < 24$ ，程序会输出“你的身材很棒！”，但是如果不满足条件，这段输出就没有了。这就是刚才提到的，代码可以有不同的执行路径，有些代码不一定会执行到。我们在 `if` 关键字的后面给出了一个表达式 `18.5 <= bmi < 24` ，之前我们说过，关系运算会产生布尔值，如果 `if` 后面的布尔值为 `True` ，那么 `if` 语句下方，有四个空格缩进的 `print('你的身材很棒！')` 就会被执行。我们先输入几组数据运行上面的代码，如下所示。

第一组输入：

```
身高(cm): 175
体重(kg): 68
bmi = 22.2
你的身材很棒！
```

第二组输入：

```
身高(cm): 175
体重(kg): 95
bmi = 31.0
```

第三组输入：

```
身高(cm)：175
体重(kg)：50
bmi = 16.3
```

只有第一组输入的身高和体重计算出的 BMI 在 18.5 到 24 这个范围值内，所以触发了 `if` 条件，输出了“你的身材很棒”。需要说明的是，不同于 C、C++、Java 等编程语言，Python 中没有用花括号来构造代码块而是**使用缩进的方式来表示代码的层次结构**，如果 `if` 条件成立的情况下需要执行多条语句，只要保持多条语句具有相同的缩进就可以了。换句话说，若干行连续的语句如果保持了相同的缩进，那么它们就属于同一个**代码块**，相当于是一个执行的整体。缩进可以使用任意数量的空格，但**通常使用4个空格**，强烈建议大家**不要使用制表键（Tab键）来缩进代码**，如果你已经习惯了这么做，可以设置你的代码编辑器自动将 1 个制表键变成 4 个空格，很多代码编辑器都支持这项功能，PyCharm 中默认也是这样设定的。还有一点，在 C、C++、Java 等编程语言中， `18.5 <= bmi < 24` 要写成两个条件 `bmi >= 18.5` 和 `bmi < 24` ，然后把两个条件用与运算符连接起来，Python 中也可以这么做，例如刚才的 `if` 语句也可以写成 `if bmi >= 18.5 and bmi < 24:` ，但是没有必要，难道 `if 18.5 <= bmi < 24:` 这个写法它不香吗？下面用 Java 代码做了同样的事情，看不懂 Java 代码没关系，感受一下它和 Python 语法的区别就可以了。

```Java
import java.util.Scanner;

class Test {

    public static void main(String[] args) {
        try (Scanner sc = new Scanner(System.in)) {
            System.out.print("身高(cm): ");
            double height = sc.nextDouble();
            System.out.print("体重(kg): ");
            double weight = sc.nextDouble();
            double bmi = weight / Math.pow(height / 100, 2);
            System.out.printf("bmi = %.1f\n", bmi);
            if (bmi >= 18.5 && bmi < 24) {
                System.out.println("你的身材很棒！");
            }
        }
    }
}
```

> **说明**：上面就是 BMI 计算器 1.0 版本对应的 Java 代码，很多人喜欢 Python 语言不是没有道理的，通常它都能用更少的代码解决同样的问题。

接下来，我们对上面的代码稍作修改，在 BMI 不满足 $18.5 \leq BMI < 24$ 的情况下，也给出相信的提示信息。我们可以在 `if` 代码块的后面增加一个 `else` 代码块，它会在 `if` 语句给出的条件没有达成时执行，如下所示。很显然， `if` 下面的 `print('你的身材很棒！')` 和 `else` 下面的 `print('你的身材不够标准哟！')` 只有一个会被执行到。

```Python
"""
BMI计算器

Version: 1.1
Author: 骆昊
"""
height = float(input('身高(cm)：'))
weight = float(input('体重(kg)：'))
bmi = weight / (height / 100) ** 2
print(f'{bmi = :.1f}')
if 18.5 <= bmi < 24:
    print('你的身材很棒！')
else:
    print('你的身材不够标准哟！')
```

如果要给出更为准确的提示信息，我们可以再次修改上面的代码，通过 `elif` 关键字为上面的分支结构增加更多的分支，如下所示。

```
"""
BMI计算器

Version: 1.2
Author: 骆昊
"""
height = float(input('身高(cm): '))
weight = float(input('体重(kg): '))
bmi = weight / (height / 100) ** 2
print(f'{bmi = :.1f}')
if bmi < 18.5:
    print('你的体重过轻! ')
elif bmi < 24:
    print('你的身材很棒! ')
elif bmi < 27:
    print('你的体重过重! ')
elif bmi < 30:
    print('你已轻度肥胖! ')
elif bmi < 35:
    print('你已中度肥胖! ')
else:
    print('你已重度肥胖! ')
```

我们再用刚才的三组数据来测试下上面的代码，看看会得到怎样的结果。

第一组输入：

```
身高(cm): 175
体重(kg): 68
bmi = 22.2
你的身材很棒!
```

第二组输入：

```
身高(cm): 175
体重(kg): 95
bmi = 31.0
你已中度肥胖!
```

第三组输入：

```
身高(cm): 175
体重(kg): 50
bmi = 16.3
你的体重过轻!
```

使用match和case构造分支结构

Python 3.10 中增加了一种新的构造分支结构的方式，通过使用 `match` 和 `case` 关键字，我们可以轻松的构造出多分支结构。Python 的官方文档在介绍这个新语法时，举了一个 HTTP 响应状态码识别的例子（根据 HTTP 响应状态输出对应的描述），非常有意思。如果不知道什么是 HTTP 响应状态吗，可以看看 MDN 上面的文档。下面我们对官方文档上的示例稍作修改，为大家讲解这个语法，先看看下面用 `if-else` 结构实现的代码。

```
status_code = int(input('响应状态码: '))
if status_code == 400:
    description = 'Bad Request'
elif status_code == 401:
    description = 'Unauthorized'
elif status_code == 403:
    description = 'Forbidden'
elif status_code == 404:
    description = 'Not Found'
elif status_code == 405:
    description = 'Method Not Allowed'
elif status_code == 418:
    description = 'I am a teapot'
elif status_code == 429:
    description = 'Too many requests'
else:
    description = 'Unknown status Code'
print('状态码描述:', description)
```

运行结果：

```
响应状态码: 403
状态码描述: Forbidden
```

下面是使用 match-case 语法实现的代码，虽然作用完全相同，但是代码显得更加简单优雅。

```
status_code = int(input('响应状态码: '))
match status_code:
    case 400: description = 'Bad Request'
    case 401: description = 'Unauthorized'
    case 403: description = 'Forbidden'
    case 404: description = 'Not Found'
    case 405: description = 'Method Not Allowed'
    case 418: description = 'I am a teapot'
    case 429: description = 'Too many requests'
    case _: description = 'Unknown Status Code'
print('状态码描述:', description)
```

> **说明**：带有 _ 的 case 语句在代码中起到通配符的作用，如果前面的分支都没有匹配上，代码就会来到 case _ 。 case _ 的是可选的，并非每种分支结构都要给出通配符选项。如果分支中出现了 case _ ，它只能放在分支结构的最后面，如果它的后面还有其他的分支，那么这些分支将是不可达的。

当然， match-case 语法还有很多高级玩法，其中有一个合并模式可以先教给大家。例如，我们要将响应状态码 401 、 403 和 404 归入一个分支， 400 和 405 归入到一个分支，其他保持不变，代码还可以这么写。

```
status_code = int(input('响应状态码: '))
match status_code:
    case 400 | 405: description = 'Invalid Request'
    case 401 | 403 | 404: description = 'Not Allowed'
    case 418: description = 'I am a teapot'
    case 429: description = 'Too many requests'
    case _: description = 'Unknown Status Code'
print('状态码描述:', description)
```

运行结果：

```
响应状态码：403
状态码描述：Not Allowed
```

分支结构的应用

例子1：分段函数求值

有如下所示的分段函数，要求输入 x ，计算出 y 。

$$y = \begin{cases} 3x - 5, & (x > 1) \\ x + 2, & (-1 \leq x \leq 1) \\ 5x + 3, & (x < -1) \end{cases}$$

```
"""
分段函数求值

Version: 1.0
Author: 骆昊
"""
x = float(input('x = '))
if x > 1:
    y = 3 * x - 5
elif x >= -1:
    y = x + 2
else:
    y = 5 * x + 3
print(f'{y = }')
```

根据实际开发的需要，分支结构是可以嵌套的，也就是说在分支结构的 `if` 、 `elif` 或 `else` 代码块中还可以再次引入分支结构。例如 `if` 条件成立表示玩家过关，但过关以后还要根据你获得宝物或者道具的数量对你的表现给出评价（比如点亮一颗、两颗或三颗星星），那么我们就需要在 `if` 的内部再构造一个新的分支结构。同理，我们在 `elif` 和 `else` 中也可以构造新的分支，我们称之为嵌套的分支结构。按照这样的思路，上面的分段函数求值也可以用下面的代码来实现。

```
"""
分段函数求值

Version: 1.1
Author: 骆昊
"""
x = float(input('x = '))
if x > 1:
    y = 3 * x - 5
else:
    if x >= -1:
        y = x + 2
    else:
        y = 5 * x + 3
print(f'{y = }')
```

> **说明**：大家可以自己感受和评判一下上面两种写法哪一种更好。在“**Python 之禅**”中有这么一句话：“**Flat is better than nested**”。之所以认为“扁平化”的代码更好，是因为代码嵌套的层次如果很多，会严重的影响代码的可读性。所以，我个人更推荐大家使用第一种写法。

例子2：百分制成绩转换成等级

要求：如果输入的成绩在90分以上（含90分），则输出 `A` ；输入的成绩在80分到90分之间（不含90分），则输出 `B` ；输入的成绩在70分到80分之间（不含80分），则输出 `C` ；输入的成绩在60分到70分之间（不含70分），则输出 `D` ；输入的成绩在60分以下，则输出 `E` 。

```
"""
百分制成绩转换为等级制成绩

Version: 1.0
Author: 骆昊
"""
score = float(input('请输入成绩: '))
if score >= 90:
    grade = 'A'
elif score >= 80:
    grade = 'B'
elif score >= 70:
    grade = 'C'
elif score >= 60:
    grade = 'D'
else:
    grade = 'E'
print(f'{grade = }')
```

例子3：计算三角形的周长和面积。

要求：输入三条边的长度，如果能构成三角形就计算周长和面积；否则给出“不能构成三角形”的提示。

```
"""
计算三角形的周长和面积

Version: 1.0
Author: 骆昊
"""
a = float(input('a = '))
b = float(input('b = '))
c = float(input('c = '))
if a + b > c and a + c > b and b + c > a:
    perimeter = a + b + c
    print(f'周长: {perimeter}')
    s = perimeter / 2
    area = (s * (s - a) * (s - b) * (s - c)) ** 0.5
    print(f'面积: {area}')
else:
    print('不能构成三角形')
```

> **说明：** 上面的 `if` 条件表示任意两边之和大于第三边，这是构成三角形的必要条件。当这个条件成立时，我们要计算并输出周长和面积，所以 `if` 下方有五条语句都保持了相同的缩进，它们是一个整体，只要 `if` 条件成立，它们都会被执行，这就是我们之前提到的代码块的概念。另外，上面计算三角形面积的公式叫做海伦公式，假设有一个三角形，边长分别为 a 、 b 、 c ，那么三角的面积 A 可以由公式 $A = \sqrt{s(s-a)(s-b)(s-c)}$ 得到，其中， $s = \frac{a+b+c}{2}$ 表示半周长。

总结

学会了 Python 中的分支结构和循环结构，我们就可以解决很多实际的问题了。这一节课相信已经帮助大家掌握了构造分支结构的方法，下一节课我们为大家介绍循环结构，学完这两次课你一定会发现，你能写出很多很有意思的代码，继续加油吧！