第三讲
控制语句与函数
Lecture 3
Control Statements and Functions

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

if语句
if Statement

- If语句主要表示当条件测试通过时的逻辑处理。

  If statement preambles the logic processing when conditional tests pass.
- 对于条件测试，有下面几种类型：

  There are various conditional tests listed as below:
  - 相等测试（Equality Checking）： ==
  - 不等测试（Inequality Checking）：!=
  - 数值比较（Numeric Comparison）：>，<，>=，<=
- 多个条件联立的情况：

  Multiple conditions joint:
  - 全部成立（all hold）： and
  - 任一成立（any holds）：or

if语句
if Statement

- 用于条件测试的表达式，经计算其结果为布尔值True或者False。因此，尽管不常见，确实可以直接以True或False作为条件测试。

  The expressions for conditional tests are evaluated into Boolean values True or False. Therefore, although it is very rare, True or False can be directed used for conditional tests.

- 除此之外，还可以用in来测试一个元素是否位于列表或元组中。

  Besides that, the keyword in can be used to check whether a list or tuple contains a specific element.

- 注意，与其它语言不同的是，条件测试语句不需要放在小括号中，但需要在最后加一个冒号，去导引一个代码段。

  Distinguished from other languages, conditional test expressions needn’t to be put into parentheses, but a colon must be presented to lead a code block.

if语句
if Statement

- if 语句的最通常的格式为：

  The general form of if structure is as follows:

```
if condiontional_expression:
    statements
elif conditional_expression:
    statements
elif conditional_expression:
    statements
…
else:
    statements
```

if语句
if Statement

```
>>> def score2band(score):
...     if score >= 90:
...         band = "Excellent"
...     elif score >= 80 and score < 90:
...         band = "Very Good"
...     elif score >= 70 and score < 80:
...         band = "Good"
...     elif score >= 60 and score < 70:
...         band = "Pass"
...     else:
...         band = "Fail"
...     return band
...
>>> print("you got {}, belonging to band '{}'".format(90,
... score2band(90)))
you got 90, belonging to band 'Excellent'
>>> print("you got {}, belonging to band '{}'".format(69,
... score2band(69)))
you got 69, belonging to band 'Pass'
```

循环语句
Loop Statement

- 首先介绍while语句，其语法形式是while后跟条件测试语句，当测试语句为真时，执行代码块中的代码。

  We first introduce the while statement. For the syntax, the while keyword leads a conditional test expression, upon evaluating to True to execute the code block:

```
while condiontional_expression:
    statements
    break
    statements
    continue
```

循环语句
Loop Statement

- 在某些情况下，循环语句没有明确结束条件，因此需要额外的机制能保证循环在适当的时候结束。break用来从无限循环中退出，当有多层循环时，break从最内层循环退出。在某些情况下，当前循环的代码段因某些情况不适合处理时，需要进行下次循环继续处理。Continue语句用于需要中止当前循环进行新的循环的情况。

  In some cases, there is no explicit condition for ending the loop, which means extra mechanisms are needed to ensure its termination appropriately. break is used to have the code return from infinite loop. For nested loop, break only takes effect from the most inner loop enclosing the break. In instances that the current code block is invalidate due to some reason and recommences the next loop, continue statement is used for this case.

- 注意，空的串，列表，元组均作为False来对待。

  Notably, the empty string, empty list and tuple are all treated as False.

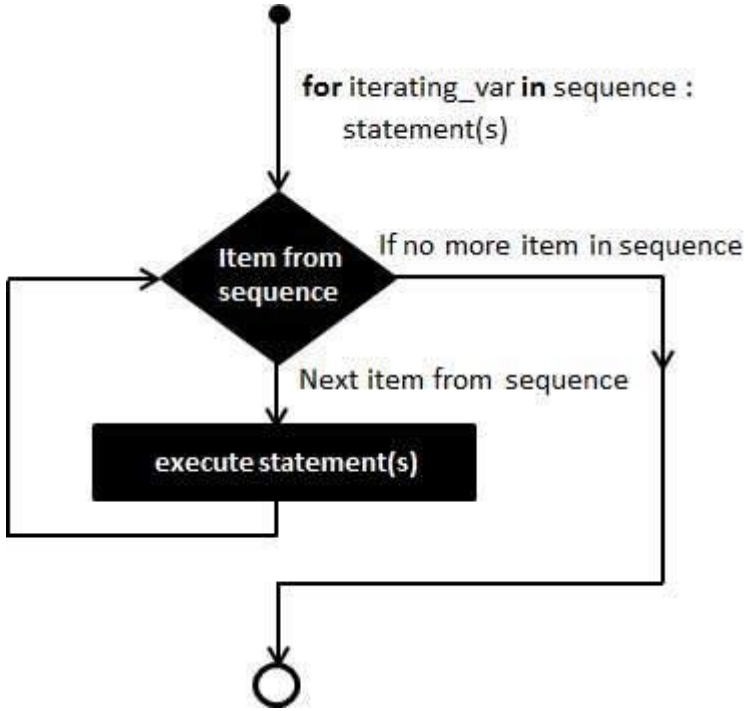
循环语句
Loop Statement

```
>>> import collections
>>> StudentInfo = collections.namedtuple("Student", ["Name", "Score"])
>>> students = []
>>> while input("[C]ontinue?").lower() == "c":
...     name = input("Student name:")
...     score = input("Score: ")
...     student = StudentInfo(name, score)
...     students.append(student)
...
[C]ontinue?c
Student name:Andy
Score: 91
[C]ontinue?c
Student name:Billy
Score: 77
[C]ontinue?c
Student name:David
Score: 82
[C]ontinue?n
>>> while students:
...     print(students.pop())
...
Student(Name='David', Score='82')
Student(Name='Billy', Score='77')
Student(Name='Andy', Score='91')
```

循环语句
Loop Statement

- 第二种循环结构为for循环语句，break与continue语句在for循环中有同样的语义。

  The second loop structure in Python is the for statement. break and continue retains the same syntax as in the while loop.



循环语句
Loop Statement

- for循环可能未必执行到循环结束（考虑break的情况）。当for循环需要当结束时显式执行代码块时，可用 else 关键字标识。

  The for loop might terminate before exhausting the test (considering the break instance). The else keyword is used to specify a block of code to be executed when the for loop is really finished.

- for 循环不能为空，但如果由于某种原因需要一个没有内容的 for 循环，可以利用 pass 语句以避免出错。

  for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

```
>>> for i in range(2):
...     print(i)
... else:
...     print("for ends exhausting tests")
...
0
1
for ends exhausting tests
>>> for i in range(2):
...     if i == 1:
...         break
...     print(i)
... else:
...     print("for ends exhausting tests")
...
0
>>> for i in range(2):
...     pass
...
>>>
```

函数
Function

- 函数是为完成特定功能而编写的命名代码块。当执行在函数中定义的特定功能时，调用该函数即可。

  Functions are named blocks of code that are designed to do one specific job. To perform a particular task defined in that function, just call the function responsible for it.

- 调用函数时，有时需要通过参数来提供额外的信息。某些语言可能会区分函数与过程：调用时需要参数的称为函数，不需要参数的称为过程。Python里统一称为函数。

  Sometimes, extra information needs to be provided via parameters when invoking a function. Some languages explicitly distinguishes between functions and procedures: functions are with parameters while procedures are not. However, in Python they are just called functions.

- Python提供了def来定义一个函数。

  Python provides the keyword def to define a function.

函数
Function

- 调用函数时，Python 必须将函数调用中的每个参数与函数定义中的参数匹配。 最简单的方法是基于提供的参数的顺序。以这种方式匹配的值称为位置参数，注意位置参数的位置很重要。

  When a function is invoked, Python must match each argument in the function call with a parameter in the function definition. The simplest way to do this is based on the order of the arguments provided. Values matched up this way are called positional arguments. Order matters in positional arguments

```
>>> def greet_user():
...     print("Hello")
...
>>> greet_user()
Hello
>>>
>>> def greet_user(username):
...     print(f"Hello {username.title()}")
...
>>> greet_user("alice")
Hello Alice
>>>
>>> def describe_pet(animal_type, pet_name):
...     print(f"\nI have a {animal_type}.")
...     print(f"My {animal_type}'s name is {pet_name.title()}.")
...
>>> describe_pet('hamster', 'harry')

I have a hamster.
My hamster's name is Harry.
>>>
```

函数
Function

- 利用位置传递参数，需要严格按照位置传递。同时，可以利用函数定义时的参数名称作为关键字来传递参数，即按键值方式传递给函数。 直接将参数中的名称和值关联传递给函数时，不会造成混淆，即不必担心在函数调用中正确排序参数。

  Position matters when passing parameters using positional arguments. Meantime, The names of the arguments when define the function can be used as keys to pass parameters in a key-value pair way. Directly associate the name and the value within the argument when passing the argument to the function causes no confusion, it means it will free the caller from having to worry about correctly ordering the arguments in the function call.

```
>>> def show_name(given_name, surname):
...     print(f"Hi, {given_name.title()} {surname.upper()}")
...
>>> show_name(surname="ming", given_name="yurui")
Hi, Yurui MING
```

函数
Function

```
>>> def print_address(country="china", city="beijing", district, street, building):
...     print(f"{country.title()}\t{city.title()}\t{street.title()}\t{building.title()}")
...
  File "<stdin>", line 1
SyntaxError: non-default argument follows default argument
```

- Python支持默认参数，即编写函数时，为选定参数提供一个默认值。如果在函数调用时提供了参数的参数，Python将使用参数值。否则，它将使用参数的默认值。这种方式提供了一定的便捷性与灵活性。

  Python support default values for arguments. It means when writing a function, one can define a default value for each parameter. If an argument for a parameter is provided in the function call, Python uses the argument value. If not, it uses the parameter's default value. So, a default value for a parameter provides the convenience and flexibility.

- 使用默认值定义函数时，任何具有默认值的参数，都需要位于所有没有默认值的参数之后。这允许 Python 继续正确地解释位置参数。

  When use default values, any parameter with a default value needs to be listed after all the parameters that don't have default values. This allows Python to continue interpreting positional arguments correctly.

函数
Function

- 函数可能在处理一些数据后返回一个值或一组值。函数返回的值称为返回值。return 语句用来从函数内部获取一个值并将返回调用该函数的行。

  A function might return a value or set of values after processing some data. The value the function returns is called a return value. The return statement takes a value from inside a function and sends it back to the line that called the function.

```
>>> def assemble_person_info(first_name, last_name):
...     person = {"first": first_name, "last": last_name}
...     return person
...
>>> musician = assemble_person_info("Ludwig", "Beethoven")
>>> print(musician)
{'first': 'Ludwig', 'last': 'Beethoven'}
```

函数
Function

```
>>> def make_pizza(*toppings):
...     """Print the list of requested toppings"""
...     print(toppings)
...
>>> make_pizza('pepperoni')
('pepperoni',)
>>> make_pizza('tomato', 'cucumber')
('tomato', 'cucumber')
>>> make_pizza('mushrooms', 'green peppers', 'extra cheese')
('mushrooms', 'green peppers', 'extra cheese')
```

- Python也支持变长参数，允许函数从调用语句中接收任意数量的参数。

  Python also support functions with variance length functions, aka, Python allows a function to collect an arbitrary number of arguments from the calling statement.

- 例如，考虑一个制作披萨的函数。它需要接受若干个浇头，但无法提前知道某个个人想要多少浇头。因此，如下定义的函数中有一个参数 *toppings，但此参数接收调用者实际提供的所有的参数。

  For example, consider a function that builds a pizza. It needs to accept a number of toppings, but it is hard to anticipate how many toppings a person will want. The function in the following example has one parameter, *toppings, but this parameter collects as many arguments as the calling line provides.

函数
Function

- 名称为*toppings的参数中的星号告诉Python，将接收到的任何值打包到这个名为toppings元组中。函数体中的 print() 调用产生的输出，表明 Python 可以处理从具有一个值的函数调用，到具有三个值的调用，即它以类似方式处理不同的调用。请注意，Python 会将参数打包到一个元组中，即使函数只接收一个值。

  The asterisk in the parameter name *toppings tells Python to pack whatever values it receives into this tuple called toppings. The print() call in the function body produces output showing that Python can handle a function call with one value and a call with three values. It treats the different calls similarly. Note that Python packs the arguments into a tuple, even if the function receives only one value.

```
>>> def make_pizza(*toppings):
...     """Summarize the pizza we are about to make."""
...     print("\nMaking a pizza with the following toppings:")
...     for topping in toppings:
...         print(f"- {topping}")
...
>>> make_pizza('mushrooms', 'green peppers', 'extra cheese')

Making a pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
>>>
```

函数
Function

```
>>> def make_pizza(size, *toppings):
...     """Summarize the pizza we are about to make."""
...     print(f"\nMaking a {size}-inch pizza with the following toppings:")
...     for topping in toppings:
...         print(f"- {topping}")
...
>>> make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
>>>
```

- 如果希望函数接受几种不同类型的参数，则接受任意数量参数的参数必须放在函数定义的最后。 Python 首先匹配位置参数和关键字参数，然后在最后一个参数中收集任何剩余的参数。例如，如果函数需要输入披萨的大小，则该参数必须位于参数 *toppings 之前：。

  If a function needs to accept several different kinds of arguments, the parameter that accepts an arbitrary number of arguments must be placed last in the function definition. Python matches positional and keyword arguments first and then collects any remaining arguments in the final parameter. For example, if the function needs to take in a size for the pizza, that parameter must come before the parameter *toppings:

函数
Function

```
>>> def build_profile(first, last, **user_info):
...     """Build a dictionary containing everything we know about a user."""
...     user_info['first_name'] = first
...     user_info['last_name'] = last
...     return user_info
...
>>> user_profile = build_profile('albert', 'einstein', location='princeton', field='physics')
>>> print(user_profile)
{'location': 'princeton', 'field': 'physics', 'first_name': 'albert', 'last_name': 'einstein'}
>>>
```

- 有时函数需要能够接受任意数量的参数，但将传递给函数的信息类型不能提前知晓。在这种情况下，可以编写函数来接受调用语句提供的尽可能多的键值对。 一个例子涉及建立用户档案：由于很难得到关于用户的全部的信息，因此不能确定会收到什么样的信息。以下示例中的函数build_profile()始终接受名字和姓氏，但它也接受任意数量的关键字参数：

  Sometimes a function needs to accept an arbitrary number of arguments, but what kind of information will be passed to the function is unknown ahead of time . In this case, functions can be defined to accept as many key-value pairs as the calling statement provides. One example involves building user profiles: it is hard to get full information about a user, hence it is hard to anticipate what kind of information will be provided when invoking the function. The function build_profile() in the following example always takes in a first and last name, but it accepts an arbitrary number of keyword arguments as well.

函数
Function

- 在编写函数时，可以以多种不同的方式混合位置、关键字和任意值参数。知道所有这些参数类型都存在对熟练掌握Python编程是很有裨益的，因为读者很可能在阅读其他人的代码时会经常看到它们。可以通过练习学习正确使用不同类型并知道何时使用每种类型。

  One can mix positional, keyword, and arbitrary values in many different ways when writing functions. It's useful to know that all these argument types exist because the reader might see them often when reading other people's code. It takes practice to learn to use the different types correctly and to know when to use each type.

- 在泛型编程时，读者会经常看到通用参数名称 *args，其用于收集任意位置参数；也会经常看到参数名称 **kwargs，其用于收集非特定关键字参数。

  In generic programming, readers might notice the generic parameter name *args, which collects arbitrary positional arguments like this. And readers often see the parameter name **kwargs used to collect non-specific keyword arguments.