Python

Topics: Conditionals

Conditionals

Tutorial

Python Control Structures - Loops and Conditionals

You can control the flow of logic in your code through various methods.

Basic control flows

- Selection (if statements)
- Iteration (for loops)

More advanced control flows

- Procedural Abstraction (functions)
- Recursion
- Concurrency
- Exception Handling and Speculation
- Nondeterminacy

In this tutorial you will come to know:

How to have sequential, selective and iterative flows in your code. This can be achieved using the for loop. How to achieve procedural abstraction. This can be done by the use of functions.

Other topics like Recursion, Exception Handling, Concurrency will be discussed in later tutorials.

Loops

Working on items of the iterable

If you want to run an operation on a collection of items, then you can do it using for loops. The skeleton for using the for loop is shown below.Note that the for statement line will end with a colon `:` and the rest of the code block must be indented with a spacing of 4 spaces. An iterable is any object that can be looped on such as list, tuple, string etc.

```
for item in iterable: # you can place any list or tuple or string in place of iterable
    # write your code here.
    pass
```

If you want to print an element of a list of fruits, you can write the following code to achieve that.

```
>>> fruits = ["apples", "oranges", "mangoes"]
>>> for fruit in fruits:
...     print(fruit)
...
apples
oranges
mangoes
```

In the example above, note that items in the iterable (i.e fruits) will be assigned to the for loop variable (i.e fruit) during the iteration process. So, we can access the item directly.

```
>>> fruits = ["apples", "oranges", "mangoes"]
>>> for fruit in fruits:
...     string_size = 0
...     for alphabet in fruit:
...         string_size += 1
...     print("name of fruit: %s is has length %s" % (fruit, string_size))
...
name of fruit: apples is has length 6
name of fruit: oranges is has length 7
name of fruit: mangoes is has length 7
```

Looping on both indexes and items

In the previous section, index or the place value of the item in the iterable was not considered. However, if you are interested in working with the index, then you can call the `enumerate` function

?

which returns a tuple of the index and the item. Taking the example above, you can print the name of the fruit and the index of the list of fruits.

```
>>> fruits = ["apples", "oranges", "mangoes"]
>>> for index, fruit in enumerate(fruits):
...      print("index is %s" % index)
...      print("fruit is %s" % fruit)
...      print("############################")
...
index is 0
fruit is apples
############################
index is 1
fruit is oranges
############################
index is 2
fruit is mangoes
############################
```

While statement

The while statement will execute a block of code as long as the condition is true. The skeleton of a while block is shown below.

```
while condition:
    code_block
```

Note that similar to the for loop, the `while` statement ends with a colon `:` and the remaining code block is indented by 4 spaces. We can implement the fruit example in the while block as well, although the logic becomes a bit complicated than the for block.

```
>>> fruits = ["apples", "oranges", "mangoes"] # get the list
>>> length = len(fruits) # get the length that will be needed for the while
condition
>>> i = 0 # initialise a counter
>>> while i < length: # give the condition
...          print(fruits[i]) # the code block
...          i += 1 # increment the counter
...
apples
oranges
mangoes
```

Nested for loops

You can have one or more nested for loops. For example, look at the following example where you can print the multiplication table. The table is shown only for 1 and 2 to save space. You can try for the remaining digits.

```
>>> for i in range(1,3):
...         for j in range(1,3):
...                 print('%d x %d = %d' % (i, j, i*j))
...
1 x 1 = 1
1 x 2 = 2
2 x 1 = 2
2 x 2 = 4
```

Selection and Python If statements

Creating if blocks

As a programmer, you will continually feel the need to control the flow of your program and let it make runtime decisions based on some condition. The is done using the if syntax. To implement this you can look at the if .. elif .. else syntax.

```
if condition1:
        code_block1
elif condition2:
        code_block2
else:
        code_block3
```

You can try the following example to understand better .

```
>>> num = 42
>>> if num == 42: # condition
...     print("number is 42") # direction 1
...
number is 42
```

Adding an else block:

```
>>> num = 43
>>> if num == 42:
...          print("number is 42")
...      else:
...          print("number if not 42")
...
number if not 42
```

Now, let us add an elif block to it as well and see what happens:

```
>>> num = 44
>>> if num == 42:
...          print("number is 42")
...      elif num == 44:
...          print("num is 44")
...      else:
...          print("num is neither 42 nor 44")
...
num is 44
```

Nested if statements

You can have one or more nested if blocks inside if statements.

```
>>> num = 42
>>> if num > 20:
...          if num < 50:
...                  print("num between 20 and 50")
...
num between 20 and 50
```

Contributed by: Joydeep Bhattacharjee

+1-650-461-4192

contact@hackerearth.com

For Developers

Practice programming

Complete reference to competitive programming

Competitive coding challenges

Code Monk

Start a programming club

Developer Resources

Developers blog

Learn to code by competitive programming

Developers wiki

How to conduct a hackathon

For Business

Assess developers

Conduct remote interviews

Assess university talent

Organize hackathon

Company

About us

Press

Careers

Contact us

Technical support

?