Introduction to Python

Oscar Dalmau

Centro de Investigación en Matemáticas

August 26, 2020

Outline

if...else statements

An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a false value. The else statement is optional

```
if expression:
    statement(s)
else:
    statement(s)
```

elif Statement

The elif statement allows you to check multiple expressions for truth value and execute a block of code as soon as one of the conditions evaluates to true. else and elif statements are optional.

```
if  expression1:
    statement(s)
elif  expression2:
    statement(s)
elif  expression3:
    statement(s)
else :
    statement(s)
```

while loop

```
while  expression:
    statement(s)
```

Example:

```
count = 0
while (count < 9):
    print ('The count is:', count)
    count = count + 1
```

The Infinite Loop

Example:

```
num = 1
while num == 1 :  # Infinite loop
    num = input("Enter a number  :")
    print("You enter a number  :")
    print("You entered: ", num)
print( "Good bye!")
```

for loop

```
for iterating_var in  sequence:
        statement(s)
```

Example:

```
for i in  range(10):
  if i%2==0:
    print('Number: ', i)
```

Loop Control Statements

Loop control statements change execution from its normal sequence.

Control statement	Description
break	Terminates the loop and transfers execution to the statement immediately following the loop.
continue	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

See example1statements.py

Functions

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).
- Arguments (parameters) should be placed within these parenthesis
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.

Syntax:

```
def functionname( parameters ):
 # ayuda que se obtiene usando help(functionname)
  "function_docstring"
  function_suite
  return [expression]
```

Functions

- The code block should be indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax:

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return  [expression]
```

Functions

- Calling a Function: You can execute it by calling it from another function or directly from the Python prompt
- Pass by reference vs value: All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.
- Inmutable objects act like "by-value": integer, float, long, strings, tuples
- Mutable objects act like "by-pointer": lists and dictionaries

Avoiding mutable argument changes

- To avoid mutable argument changes create a copy

Examples:

```
x = [1,2]; y = {0:'a', 1: 'b'}
foo(x[:], y.copy()  )
```

You can also create a copy inside the function

```
def foo(x,y):
      xc = x[:]  # for lists
      yc = y.copy()   # for dictionaries
```

Avoiding mutable argument changes

```
def mifunc(a,b):
  print id(a),id(b)# The same address
                   as the input parameter.
  a=100# Crete a new variable with a new address.
  b[0]=10# Modify the first entry.
  print id(a),id(b)# b has the same address
                   as the input parameter.
  b=[0,1]# Crete a new variable with a new address.
  print id(a),id(b)# The addresses have changed.
x,y=(1,2),(3,4);mifunc(x,y);Error!!
x,y=(1,2),[3,4];mifunc(x,y);Ok!!
```

Function Arguments

- Required arguments: Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition
- Keyword arguments: When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name: This allows us to place arguments out of order.
- Default arguments: It is an argument that assumes a default value if a value is not provided in the function call for that argument

Function Arguments

- Variable-length arguments: *def func(*targs)*. args is a tuple.

```
>>>def func1(*args):
        s = 0
        for i in range(len(args)): s += args[i]
        return s

>>>  func1(1,2,3) or  func1(*(1,2,3))
```

Function Arguments

- Variable-length arguments *def func(**kwargs)*. kwargs is a dictionary. It is similar to the previous *targs, but only works for keywords arguments

```
>>> def func(**kwargs):
        print (kwargs)
>>> func(x = 3, y = 4, z = 5)   # or
>>> func(**{'x':3, 'y':4, 'z': 5})
```

Function Arguments

```
def mifuncion(**kwargs):
    if 'Iteraciones' in kwargs :
        Iteraciones=kwargs['Iteraciones']
    else:
        Iteraciones=100
    print ('Numero de iteraciones=
        {0:4d}'.format(Iteraciones))
mifuncion()
mifuncion(Iteraciones=10)
```

Function Arguments

- Example

```
>>> def func(a,b=4,*targs,**kwargs):
        print(a,b,targs,kwargs)
>>>func(3)#a=3,b=4,targs=(),kwargs={}
>>>func(3,10)#a=3,b=10,targs=(),kwargs={}
>>>func(3,10,5)#a=3,b=10,targs=(5,),kwargs={}
>>>func(b=10,a=4)#a=4,b=10,targs=(),kwargs={}
>>>func(b=1,a=4,x=4)
     #a=4,b=1,targs=(),kwargs={'x':4}
```

Anonymous function: lambda

```
>>> f1 = lambda x: x**3
>>> f1(2)
>>> f2 = lambda x: [x,x**2,x**3]
>>> f2(2)
>>> f1(range(4)) --> Error!!
```

Functions: Map, Filter and Reduce

Map, Filter and Reduce are **functions** which facilitate a functional approach to programming

Function: Map

Map: applies a function to all the items in an input-list

```
map(function-to-apply, list-of-inputs)
```

```
>>> cuadrado = lambda x:   x**2
>>> x = range(10)
>>> cuadradox = map(cuadrado,x)
>>> list(cuadradox)
```

```
>>> fx = map(lambda x:   x**2, range(10))
>>> list(fx)
```

Function: Map

```
>>> f = lambda x:  [x, x**2, x**3]
>>> x = range(10)
>>> fx = map(f,x)
>>> list(fx)
```

Function: filter

Filter: creates a list of elements for which a function returns true **Wrong!!** .
Actually, it is an iterator of elements for which a function returns true. (Ok)

```
>>> result = filter(lambda x: x%2, range(10))
>>> fx = map(lambda x:   x**2, result)
>>> list(fx)

>>> result = filter(lambda x: x<0, range(-4,2))
>>> fx = map(lambda x:   x**2, result)
>>> list(fx)
```

Function: Reduce

Reduce: applies a rolling computation to sequential pairs of values in a list.

```
>>> import functools
>>> functools.reduce(lambda x,y: 2*x+3*y, [2,1,4,3]
# calculates 2*((2*(2*2+3*1)+3*4)+3*3)
```

Step 1 x = 2, y = 1

Step 2 x = 2*x + 3*y=2*2+3*1=7

Step 3 y = 4; x = 2*x + 3*y=2*7+3*4=26

Step 4 y = 3; x = 2*x + 3*y=2*26+3*3=61

Step 5 Return: 61

Function: Reduce

```
>from functools import reduce
# sum: 47 + 11 + 42 + 13
>reduce(lambda x,y: x+y, [47,11,42,13])
# 4! factorial
>reduce(lambda x,y: x*y, range(1,5))
>fact=lambda n:reduce(lambda x,y: x*y,range(1,n+1))
>fact(3)
```

Modules

- A module allows us to logically organize your Python code.
- A module makes the code easier to understand and use.
- A module is simply a file consisting of Python code, where we can define functions, classes and variable

Importing a module

- Import Statement: One can use any Python source file as a module by executing an import statement in some other Python source file. The import has the following syntax:

```
import module1[, module2[,... moduleN]
```

- from ... import Statement: We can import specific attributes from a module

```
from modulename import name1[, ... nameN]]
```

- from ... import * Statement: We can import all names from a module into the current namespace

```
from modulename import *
```

Importing a module. Example

- Import Statement

```
>>> import math
>>> math.cos(math.pi)
```

- from ... import Statement:

```
>>> from math import cos, pi
>>> cos(pi)
```

- from ... import * Statement:

```
>>> from math import *
>>> sin(pi), cos(pi)
```

Locating Modules

- Import Statement

```
>>> import math
>>> math.cos(math.pi)
```

- from ... import Statement:

```
>>> from math import cos, pi
>>> cos(pi)
```

- from ... import * Statement:

```
>>> from math import *
>>> sin(pi), cos(pi)
```

The dir( ) Function

The dir built-in function returns a sorted list of strings containing the names defined by a module

```
>>> import math
>>> content = dir(math)
>>> print(content)
```