Python One-Liners

Stephen CUI

May 4, 2023

Contents

Chapter 1

Python Refresher

1.1 Basic Data Structures

1.1.1 Numerical Data Types and Structures

1.1.2 Booleans

Boolean Operator Precedence

Boolean operators are ordered by priority—the operator not has the highest priority, followed by the operator and, followed by the operator or. The following values are automatically evaluated to False: the keyword None, the integer value 0, the float value 0.0, empty strings, or empty container types.

1.1.3 strings

Often, you'll explicitly want to use whitespace characters in strings. The most frequently used whitespace characters are the newline character $\backslash n$, the space character $\backslash s$, and the tab character $\backslash t$.

1.1.4 The Keyword None

The keyword None is a Python constant and it means the absence of a value. Other programming languages such as Java use the value null instead. However, the term null often confuses beginners, who assume it's equal to the integer value 0. Instead, **Python uses the keyword None to indicate that it's different from any numerical value for zero, an empty list, or an empty string**. An interesting fact is that the value None is the only value in the NoneType data type.

1.2 Container Data Structures

1.2.1 Lists

Keyword: is

The keyword is simply checks whether both variables refer to the same object in memory.

1.2.2 stacks

The stack data structure works intuitively as a last-in, first-out (LIFO) structure. Python lists can be used intuitively as stacks with the list operations append() to add to the stack and pop() to remove the most recently added item.

1.2.3 Sets

A set is an unordered collection of unique elements.

collection

The collection consists of either primitive elements (integers, floats, strings), or complex elements (objects, tuples). However, **all data types in a set must be *hashable*, meaning that they have an associated hash value.** A hash value of an object never changes and is used to compare the object to other objects.

You can create a set of strings because strings are hashable. But you cannot create a set of lists, because lists are unhashable. The reason is that **the hash value depends on the content of the item, and lists are mutable; if you change the list data type, the hash value must change too.** Because mutable data types are not hashable, you cannot use them in sets.

Unordered

Unique

No matter how often you put the same value into the same set, the set stores only one instance of this value. An extension of the normal set data structure is the multiset data structure, which can store multiple instances of the same value.

1.2.4 Dictionaries

The dictionary is a useful data structure for storing (key, value) pairs.

1.2.5 Membership

Use the keyword in to check whether the set, list, or dictionary contains an element. We say x is a member of y if element x appears in the collection y.

Checking set membership is faster than checking list membership: to check whether element x appears in list y, you need to traverse the whole list until you find x or have checked all elements. However, sets are implemented much like dictionaries: to check whether element x appears in set y, Python internally performs one operation y[hash(x)] and checks whether the return value is not None.

1.2.6 List and Set Comprehension

List comprehension is a popular Python feature that helps you quickly create and modify lists. The simple formula is [expression + context]:

- **Expression** Tells Python what to do with each element in the list.
- **Context** Tells Python which list elements to select. The context consists of an arbitrary number of for and if statements.

1.3 Control Flow

1.3.1 if, else, and elif

1.3.2 Loops

To allow for repeated execution of code snippets, Python uses two types of loops: for loops and while loops. There are two fundamental ways of terminating a loop: you can define a loop condition that eventually evaluates to False, or use the keyword break at the exact position in the loop body.

It is also possible to force the Python interpreter to skip certain areas in the loop without ending it prematurely. You can achieve this by using the continue statement, which finishes the current loop iteration and brings the execution flow back to the loop condition. Code that never executes is known as *dead code*.

1.4 Functions

Functions help you to reuse code snippets at your leisure: write them once but use them often. You define a function with the keyword def, a function name, and a set of arguments to customize the execution of the function body.

The keyword return terminates the function and passes the flow of execution to the caller of the function. You can also provide an optional value after the return keyword to specify the function result.

1.5 Lambdas

You use the keyword lambda to define lambda functions in Python. *Lambda functions* are anonymous functions that are not defined in the namespace. Roughly speaking, they are functions without names, intended for single use.

```
lambda <arguments> : <return expression>
```

A lambda function can have one or multiple arguments, separated by commas. After the colon (:), you define the return expression that may (or may not) use the defined argument. The return expression can be any expression or even another function.

```
assert (lambda x: x + 3)(3) == 6
```

First, you create a lambda function that takes a value x and returns the result of the expression x + 3. The result is a function object that can be called like any other function. Because of its semantics, you denote this function as an **incrementor function**(增量函数). When calling this incrementor function with the argument x=3—the suffix (3) within the print statement—the result is the integer value 6.

Chapter 2

Python Tricks

2.1 Using List Comprehension to Find Top Earners

2.2 Reading a File

To access files on your computer, you need to know how to open and close files. You can access a file's data only after you've opened it. After closing the file, you can be sure that the data was written into the file. Python may create a buffer and wait for a while before it writes the whole buffer into the file (Figure 2-1). The reason for this is simple: file access is slow.

For efficiency reasons, Python avoids writing every single bit independently. Instead, it waits until the buffer has filled with enough bytes and then flushes the whole buffer at once into the file.

That's why it's good practice to close the file after reading it with the command f.close(), to ensure all the data is properly written into the file instead of residing in temporary memory. However, in a few exceptions, Python closes the file automatically: one of these exceptions occurs when the reference count drops to zero.

2.3 Using Lambda and Map Functions

You can use lambda functions to define a simple function with a return value (the return value can be any object, including tuples, lists, and sets). In other words, every lambda function returns an object value to

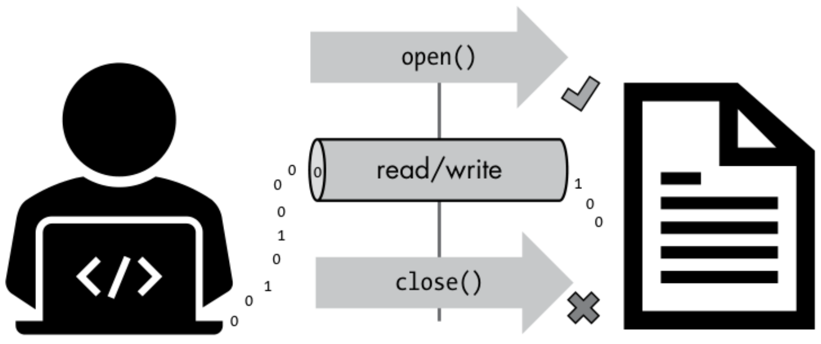

Figure 2.1: Opening and closing a file in Python

its calling environment. **Note that this poses a practical restriction to lambda functions, because unlike standard functions, they are not designed to execute code without returning an object value to the calling environment.**

```
lambda arguments : return expression
```

You start the function definition with the keyword lambda, followed by a sequence of function arguments. When calling the function, the caller must provide these arguments. You then include a colon (:) and the return expression, which calculates the return value based on the arguments of the lambda function. The return expression calculates the function output and can be any Python expression.

One common example is using lambda with the map() function that takes as input arguments a function object f and a sequence s. The map() function then applies the function f on each element in the sequence s.

2.4 Using Slicing to Extract Matching Substring Environments

```
x[start:stop:step]
```

These variants of the basic [start:stop:step] pattern of Python slicing highlight the technique’ s many interesting properties:

- If start $>=$ stop with a positive step size, the slice is empty.
- If the stop argument is larger than the sequence length, Python will slice all the way to and including the rightmost element.
- If the step size is positive, the default start is the leftmost element, and the default stop is the rightmost element (included).
- If the step size is negative (step < 0), the slice traverses the sequence in reverse order. With empty start and stop arguments, you slice from the rightmost element (included) to the leftmost element (included). Note that if the stop argument is given, the respective position is excluded from the slice.

2.5 Combining List Comprehension and Slicing